

---

# Learning Deep Representations of Data Distributions

**Sam Buchanan\***, Toyota Technological Institute at Chicago

**Druv Pai\***, University of California, Berkeley

**Peng Wang**, University of Michigan & University of Macau

**Yi Ma**, University of California, Berkeley & University of Hong Kong

This manuscript is prepared for a course on the principles of deep learning and intelligence in general, to be taught by the authors at respective institutions. This pre-publication version is free to view and download for personal use only, and is not for redistribution, re-sale or use in derivative works.

Copyright © 2025 Sam Buchanan, Druv Pai, Peng Wang, Yi Ma

August 18, 2025

\* These authors contributed equally.



# Preface

*“All roads lead to Rome.”*

This book reveals and studies a common and fundamental problem behind almost all modern practices of (artificial) intelligence. That is, *how to effectively and efficiently learn a low-dimensional distribution of data in a high-dimensional space and then transform the distribution to a compact and structured representation?* For any intelligent system, natural or man-made, such a representation can be generally regarded as a *memory* or *knowledge* learned from data sensed from the external world.

This textbook aims to provide a systematic introduction to the mathematical and computational principles for learning (deep) representations of such data distributions to *senior undergraduate students and starting graduate students*. The main prerequisite for this book is undergraduate linear algebra, probability/statistics, and optimization. Some familiarity with basic concepts from signal processing (sparse representation and compressed sensing in particular), information theory, and feedback control would enhance your appreciation.

The main motivation for writing this book is because there have been tremendous developments in the past several years, by the authors and many colleagues, that aim to establish a principled and rigorous approach to understand deep neural networks and, more generally, intelligence itself. The deductive methodology advocated by this new approach is in direct contrast, and highly complementary, to the dominant methodology behind current practices of artificial intelligence, which is largely inductive and trial-and-error. The lack of understanding about so-developed powerful AI models and systems has led to increasing hypes and fears in the society. We believe that a serious attempt to establish a principled approach to understand intelligence is more needed than ever. An overarching goal of this book is to provide solid theoretical evidence and experimental evidence showing that it is now possible to study intelligence as a scientific and mathematical subject. Hence, one may view this book as a first attempt to develop a *Mathematical Theory of Intelligence*.

At the technical levels, the theoretical framework presented in this book helps reconcile a long-standing gap between the classical approach to model data structures that are mainly based on analytical geometric, algebraic and probabilistic models (e.g., subspaces, Gaussians, and equations) and the “modern” approach that are based on empirically designed non-parametric models (e.g., deep networks). As it turns out, a unification of the two seemingly separate methodologies becomes possible and even natural if one realizes that they all try to model and learn *low-dimensional* structures in the data distribution of interest. They are merely different ways to pursue, represent, and exploit the low-dimensional structures. From this perspective, even many seemingly unrelated computational techniques, developed independently in separate fields at different times, can now be better understood under a common computational framework and probably could be studied together from now on. As we will see in this book, these techniques include but are not limited to lossy compressive encoding-decoding developed in information theory and coding theory, diffusion and denoising in signal processing and machine learning, and continuation techniques such as augmented Lagrangian methods for constrained optimization.

We believe that the unified conceptual and computational framework presented in this book will be of great value to readers who truly want to clarify mysteries and misunderstandings about deep neural networks and (artificial) intelligence. Furthermore, the framework is meant to provide the readers guiding principles for developing significantly better and truly intelligent systems in the future. More specifically, beside a general introduction (chapter), the main technical content of the book will be organized as six closely related topics (chapters):

1. We will start with the classical and most basic models of Principal Component Analysis (PCA), Independent Component Analysis (ICA), and Dictionary Learning (DL), which assume that the low-dimensional distributions of interest have linear and independent structures. From these simple idealistic models that are well studied and understood in signal processing and compressed sensing, we will introduce the most basic and important ideas for how to learn low-dimensional distributions.
2. To generalize these classical models and their solutions to general low-dimensional distributions, we introduce a universal computational principle for learning such distributions: *compression*. As we will see, data compression provides a unifying view on all seemingly different classic and modern approaches to distribution or representation learning, including dimensionality reduction, entropy minimization, score matching for denoising, and lossy compression with rate distortion etc.
3. Within this unifying framework, modern Deep Neural Networks (DNNs), such as ResNet, CNN, and Transformer, can all be mathematically interpreted as (unrolled) optimization algorithms to iteratively achieve better compression and better representations by reducing coding length/rate or gaining information. Not only does this framework help explain empirically designed deep network architectures thus far, it also leads to new architecture designs that can be significantly simpler and more efficient.
4. Furthermore, to ensure that the learned representation for a data distribution is correct and consistent, the *auto-encoding* architectures that consist of both encoding and decoding become necessary. In order for a learning system to be fully automatic and continuous, we will introduce a powerful *closed-loop transcription framework* that enables an auto-encoding system to self-correct and thus self-improve via a minimax game between the encoder and decoder.
5. We will also study how the so learned data distribution and representation can be utilized as a powerful prior or constraint to conduct Bayesian inference to facilitate almost all types of tasks and settings that are popular in the practice of modern artificial intelligence, including conditional estimation, completion, and generation of real-word high-dimensional data such as images and texts.
6. Last but not the least, to connect theory to practice, we will demonstrate step-by-step how to effectively and efficiently learn deep representations of low-dimensional data distributions with large-scale datasets, including both images and texts, and use them in many practical applications such as image classification, image completion, image segmentation, image generation, and similar tasks for text data too.

To summarize, the technical content presented in this book is to establish strong conceptual and technical connections between the classical analytical approach and the modern computational approach, between simple parametric models and deep non-parametric models, between diverse inductive practices and a unified deductive framework from the first principle. We will reveal that many seemingly unrelated or even competing approaches, though developed in separate fields at different times, all strive to achieve a common objective:

*pursuing and exploiting intrinsic low-dimensional distributions of high-dimensional data.*

To this end, the book will take us through a complete journey from theoretical formulation, to mathematical verification, to computational realization, and to practical applications.

# Declaration of Open Source

This book is meant to be an “open-source project” which is to be made publicly available for all people who are interested in the principles of intelligence. The book will be updated continuously and periodically by all who are willing to contribute to this project, including its translated versions, supporting materials and tools to be developed for studying, teaching, and researching this topic.



# Acknowledgment

This book is primarily based on research results that have been developed within the past eight years. Thanks to the generous startup funds from UC Berkeley (2018) and the University of Hong Kong (2023), Yi Ma was able to embark and focus on this new exciting research direction in the past eight years. Through these years, related to this research direction, Yi Ma and his research team at Berkeley have been supported by the following research grants:

- The multi-university *THEORINET* project for the Foundations of Deep Learning, jointly funded by the Simons Foundation and the National Science Foundation (DMS grant #2031899);
- The *Closed-Loop Data Transcription via Minimaxing Rate Reduction* project funded by the Office of Naval Research (grant N00014-22-1-2102);
- The *Principled Approaches to Deep Learning for Low-dimensional Structures* project funded by the National Science Foundation (CISE grant #2402951).

This book would have not been possible without the financial support for these research projects. The authors have drawn tremendous inspiration from research results by colleagues and students who have been involved in these projects.



# Contents

<b>Preface</b>	<b>iii</b>
<b>Declaration of Open Source</b>	<b>v</b>
<b>Acknowledgment</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Intelligence, Cybernetics, and Artificial Intelligence . . . . .	1
1.2 What to Learn? . . . . .	6
1.2.1 Predictability . . . . .	6
1.2.2 Low Dimensionality . . . . .	9
1.3 How to Learn? . . . . .	12
1.3.1 Analytical Approaches . . . . .	12
1.3.2 Empirical Approaches . . . . .	18
1.4 A Unifying Approach . . . . .	27
1.4.1 Learning Parsimonious Representations . . . . .	27
1.4.2 Learning Informative Representations . . . . .	30
1.4.3 Learning Consistent Representations . . . . .	31
1.4.4 Learning Self-Consistent Representations . . . . .	32
1.5 Bridging Theory and Practice for Machine Intelligence . . . . .	33
<b>2 Learning Linear and Independent Structures</b>	<b>35</b>
2.1 A Low-Dimensional Subspace . . . . .	36
2.1.1 Principal Components Analysis (PCA) . . . . .	36
2.1.2 Pursuing Low-rank Structure via Power Iteration . . . . .	39
2.1.3 Probabilistic PCA . . . . .	40
2.1.4 Matrix Completion . . . . .	42
2.2 A Mixture of Complete Low-Dimensional Subspaces . . . . .	43
2.2.1 Mixtures of Subspaces and Sparsely-Used Dictionaries . . . . .	44
2.2.2 Complete Dictionary Learning . . . . .	46
2.2.3 Connection to ICA and Kurtosis . . . . .	48
2.3 A Mixture of Overcomplete Low-Dimensional Subspaces . . . . .	50
2.3.1 Sparse Coding with an Overcomplete Dictionary . . . . .	50
2.3.2 Overcomplete Dictionary Learning . . . . .	52
2.3.3 Learned Deep Sparse Coding . . . . .	54
2.4 Summary and Notes . . . . .	56
2.5 Exercises and Extensions . . . . .	57
<b>3 Pursuing Low-Dimensional Distributions via Lossy Compression</b>	<b>61</b>
3.1 Entropy Minimization and Compression . . . . .	62
3.1.1 Entropy and Coding Rate . . . . .	62
3.1.2 Differential Entropy . . . . .	62
3.1.3 Minimizing Coding Rate . . . . .	63

3.2	Compression via Denoising . . . . .	65
3.2.1	Diffusion and Denoising Processes . . . . .	65
3.2.2	Learning and Sampling a Distribution via Iterative Denoising . . . . .	73
3.3	Compression via Lossy Coding . . . . .	79
3.3.1	Necessity of Lossy Coding . . . . .	80
3.3.2	Rate Distortion and Data Geometry . . . . .	81
3.3.3	Lossy Coding Rate for a Low-Dimensional Gaussian . . . . .	84
3.3.4	Clustering a Mixture of Low-Dimensional Gaussians . . . . .	86
3.4	Maximizing Information Gain . . . . .	91
3.4.1	Linear Discriminative Representations . . . . .	92
3.4.2	The Principle of Maximal Coding Rate Reduction . . . . .	96
3.4.3	Optimization Properties of Coding Rate Reduction . . . . .	98
3.5	Summary and Notes . . . . .	102
3.6	Exercises and Extensions . . . . .	102
<b>4</b>	<b>Deep Representations from Unrolled Optimization</b>	<b>105</b>
4.1	White-Box Deep Networks via Unrolled Optimization . . . . .	106
4.1.1	Deep Networks from Unrolled Gradient Descent . . . . .	107
4.1.2	Convolutional Networks from Invariant Rate Reduction . . . . .	112
4.2	White-Box Transformers from Unrolled Optimization . . . . .	119
4.2.1	Unrolled Optimization for Sparse Rate Reduction . . . . .	120
4.2.2	Overall White-Box Transformer Architecture: CRATE . . . . .	124
4.3	Variants of Deep Architectures by Design . . . . .	126
4.3.1	Attention-Only Transformer Architecture . . . . .	126
4.3.2	Linear-Time Attention: Token Statistics Transformer . . . . .	129
4.4	Summary and Notes . . . . .	132
4.5	Exercises and Extensions . . . . .	133
<b>5</b>	<b>Consistent and Self-Consistent Representations</b>	<b>135</b>
5.1	Learning Consistent Representations . . . . .	136
5.1.1	Linear Autoencoding via PCA . . . . .	137
5.1.2	Nonlinear PCA and Autoencoding . . . . .	138
5.1.3	Sparse Autoencoding . . . . .	140
5.1.4	Variational Autoencoding . . . . .	141
5.2	Learning Self-Consistent Representations . . . . .	144
5.2.1	Closed-Loop Transcription via Stackelberg Games . . . . .	146
5.2.2	A Mixture of Low-Dimensional Gaussians . . . . .	149
5.3	Continuous Learning Self-Consistent Representations . . . . .	151
5.3.1	Class-wise Incremental Learning . . . . .	151
5.3.2	Sample-wise Continuous Unsupervised Learning . . . . .	156
5.4	Summary and Notes . . . . .	159
5.5	Exercises and Extensions . . . . .	160
<b>6</b>	<b>Inference with Low-Dimensional Distributions</b>	<b>161</b>
6.1	Bayesian Inference and Constrained Optimization . . . . .	161
6.2	Conditional Inference with a Known Data Distribution . . . . .	165
6.3	Conditional Inference with a Learned Data Representation . . . . .	167
6.3.1	Image Completion with Masked Auto-Encoding . . . . .	167
6.3.2	Conditional Sampling with Measurement Matching . . . . .	168
6.3.3	Body Pose Generation Conditioned on Head and Hands . . . . .	176
6.4	Conditional Inference with Paired Data and Measurements . . . . .	176
6.4.1	Class Conditioned Image Generation . . . . .	177
6.4.2	Caption Conditioned Image Generation . . . . .	184
6.5	Conditional Inference with Measurement Self-Consistency . . . . .	186

6.5.1	Linear Measurement Models . . . . .	186
6.5.2	3D Visual Model from Calibrated Images . . . . .	187
6.6	Summary and Notes . . . . .	190
6.7	Exercises and Extensions . . . . .	191
<b>7</b>	<b>Learning Representations for Real-World Data</b>	<b>193</b>
7.1	Technical Setup and Outline of the Chapter . . . . .	193
7.2	Simplified Contrastive Learning . . . . .	195
7.2.1	Data . . . . .	195
7.2.2	Task and Objective Function . . . . .	195
7.2.3	Architecture: Vision Transformer . . . . .	198
7.2.4	Optimization Strategy . . . . .	201
7.2.5	Evaluation Methodology . . . . .	204
7.2.6	Experimental Setup and Results . . . . .	205
7.3	Image Classification . . . . .	207
7.3.1	Task and Objective . . . . .	207
7.3.2	The CRATE Architecture . . . . .	207
7.3.3	Optimization . . . . .	208
7.3.4	Evaluation Methodology . . . . .	208
7.3.5	Experimental Setup and Results . . . . .	209
7.4	Causal Language Modeling . . . . .	210
7.4.1	Data . . . . .	210
7.4.2	Task and Objective . . . . .	211
7.4.3	Architecture: Causal CRATE . . . . .	213
7.4.4	Optimization Strategy . . . . .	215
7.4.5	Evaluation Methodology . . . . .	215
7.4.6	Experimental Setup and Results . . . . .	215
7.5	Scaling White-Box Transformers . . . . .	216
7.5.1	Increasing Network Width: CRATE- $\alpha$ . . . . .	216
7.5.2	Linear Time Complexity Transformers . . . . .	219
7.5.3	Attention-Only Transformers . . . . .	220
7.6	Masked Autoencoding for Imagery Data . . . . .	221
7.6.1	Task and Objective . . . . .	221
7.6.2	Architecture . . . . .	222
7.6.3	Optimization . . . . .	223
7.6.4	Evaluation . . . . .	223
7.6.5	Experiments . . . . .	224
7.7	Summary and Notes . . . . .	224
7.8	Exercises and Extensions . . . . .	225
<b>8</b>	<b>Future Study of Intelligence</b>	<b>227</b>
8.1	Towards Autonomous Intelligence: Close the Loop? . . . . .	227
8.2	Towards Intelligence of Nature: Beyond Back Propagation? . . . . .	228
8.3	Towards Artificial Intelligence of Human: Beyond the Turing Test? . . . . .	230
<b>A</b>	<b>Appendices</b>	<b>233</b>
<b>A</b>	<b>Optimization Methods</b>	<b>233</b>
A.1	Steepest Descent . . . . .	233
A.1.1	Vanilla Gradient Descent for Smooth Problems . . . . .	233
A.1.2	Preconditioned Gradient Descent for Badly-Conditioned Problems . . . . .	236
A.1.3	Proximal Gradient Descent for Non-Smooth Problems . . . . .	238
A.1.4	Stochastic Gradient Descent for Large-Scale Problems . . . . .	239
A.1.5	Putting Everything Together: Adam . . . . .	241
A.2	Computing Gradients via Automatic Differentiation . . . . .	242

A.2.1	Differentials . . . . .	242
A.2.2	Automatic Differentiation . . . . .	245
A.2.3	Back Propagation . . . . .	246
A.3	Game Theory and Minimax Optimization . . . . .	249
A.3.1	Learning Stackelberg Equilibria . . . . .	251
A.3.2	Practical Considerations when Learning Stackelberg Equilibria . . . . .	253
A.4	Exercises . . . . .	254
<b>B</b>	<b>Entropy, Diffusion, Denoising, and Lossy Coding</b>	<b>255</b>
B.1	Differential Entropy of Low-Dimensional Distributions . . . . .	255
B.2	Diffusion and Denoising Processes . . . . .	256
B.2.1	Diffusion Process Increases Entropy Over Time . . . . .	257
B.2.2	Denoising Process Reduces Entropy Over Time . . . . .	258
B.2.3	Technical Lemmas and Intermediate Results . . . . .	261
B.3	Lossy Coding and Sphere Packing . . . . .	269
B.3.1	Proof of Relationship Between Rate Distortion and Covering . . . . .	269
B.3.2	Proof of Lemma B.6 . . . . .	271
	<b>Bibliography</b>	<b>275</b>

# Chapter 1

## Introduction

*“Just as the constant increase of entropy is the basic law of the universe, so it is the basic law of life to be ever more highly structured and to struggle against entropy.”*

– Václav Havel

### 1.1 Intelligence, Cybernetics, and Artificial Intelligence

The world in which we are living is neither fully random nor completely unpredictable.<sup>1</sup> Instead, it follows certain orders, patterns, and laws that make it largely predictable.<sup>2</sup> The very emergence and existence of life depend on a predictable living environment. Only by learning and memorizing what is predictable in the environment can life survive and thrive since good decisions and actions depend on reliable predictions. Because there seem to be unlimited things that are predictable about the world, intelligent beings, such as animals and humans, have continued to improve through evolution their capability to explore and exploit such predictability for a better and better life. To this end, they have developed increasingly more acute senses, including vision, audio, touch, taste, and smell, to perceive what is predictable in the external environment from these high-throughput sensory data. Hence a fundamental task for all intelligent beings is to be able to:

*learn and memorize predictable information from massive sensed data.*

Before we may begin to understand how this is done, we need to consider a few related questions:

- How to model and represent such predictable information in the data mathematically?
- How can such information be learned effectively and efficiently from the data computationally?
- How should such information be best organized for future prediction and inference?

This book aims to provide some answers to these questions. These answers will help us better understand intelligence, especially the computational principles and mechanisms that enable it. There is reason to believe that all forms of intelligence, from low-level intelligence seen in early primitive life to the highest form of intelligence, the practice of modern science, follow the same set of principles and mechanisms. We will elaborate more below.

---

<sup>1</sup>Note there is no need for an intelligent being to learn or memorize anything if the world is fully random.

<sup>2</sup>Some deterministic and some probabilistic.

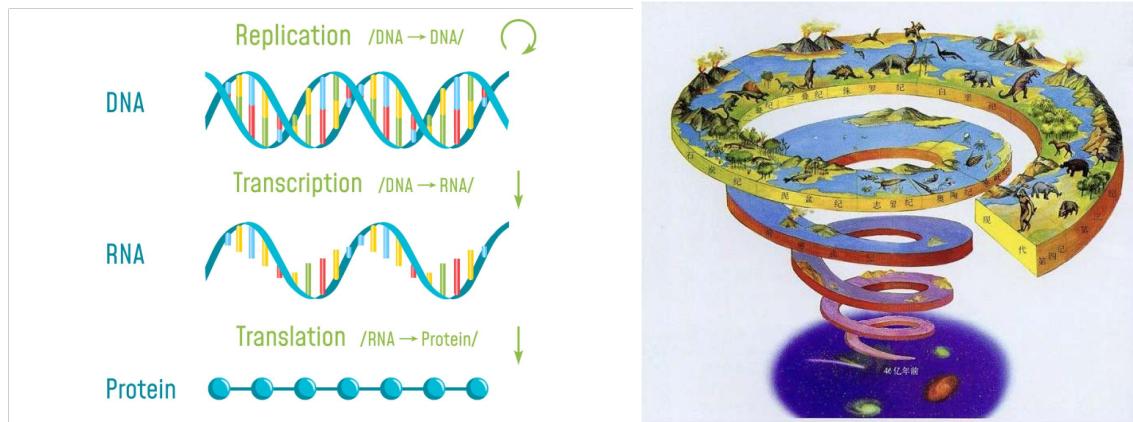


Figure 1.1: Evolution of phylogenetic intelligence: Knowledge of the external world is encoded and passed on via DNAs (left), and it is decoded from DNA to RNA and to Protein etc. In the early stage of life evolution (right), intelligence develops knowledge at the species level via (random) gene mutation and natural selection – “may the fittest survive,” which can be viewed as a primitive form of reinforcement learning.



Figure 1.2: Evolution of life, from the ancestor of all life today (named LUCA — last universal common ancestor), a single-cell-like organism which lived from 3.5-4.3 billion years ago, to the emergence of the first nervous system in worm-like species (middle), about 550 million years ago, to the explosion of life forms in the Cambrian period (right), about 530 million years ago.

**Emergence and evolution of intelligence.** A necessary condition for the emergence of life on earth about 4 billion years ago is that the earth’s environment is largely predictable. In the environment, life has developed mechanisms that allow it to learn what is predictable about the environment, encode the information in a certain way, and use it for survival. Generally speaking, we call the ability to learn *intelligence*. To a large extent, the evolution of life is the mechanism of intelligence at work. In nature, intelligence is mainly developed through two types of learning mechanisms: *phylogenetic* and *ontogenetic* [Wie61].

The *phylogenetic intelligence* refers to learning through the evolution of species. Species inherit and survive mainly based on knowledge inherited from DNAs or genes of their parents. To a large extent, we may call DNAs nature’s pre-trained large models because they play a very similar role. The main characteristic of phylogenetic intelligence is that individuals do not have much learning capacity. Learning is carried out with a “trial-and-error” mechanism based on random mutation of the genes, and then species evolve based on the process of natural selection – survival of the fittest, as shown in Figure 1.1. This can be viewed as nature’s way of implementing what is now known as “reinforcement learning.” However, such a “trial-and-error” learning process can be extremely slow, costly, and unpredictable: It is known that from the emergence of the first life forms, from about 4.4 to 3.8 billion years ago, life has relied on this form of evolution.<sup>3</sup>

The *ontogenetic intelligence* refers to the learning mechanisms that allow an individual to learn through its own senses, memories, and predictions within its specific living environment and to improve and adapt its

<sup>3</sup> Astute readers may have noticed an uncanny similarity between how early life evolves and how large language models evolve today.



Figure 1.3: The development of verbal communication and spoken languages (between 10000-5000 BC), written languages (about 3000 BC), and mathematics (around 500-300 BC) mark three key milestones in the evolution of human intelligence.

behaviors. The ontogenetic learning became possible after the emergence of the nervous system about 550 to 600 million years ago (in worm-like organisms), shown in Figure 1.2 middle. That is, with a sensory and nervous system, an individual is capable of continuously forming and improving its own knowledge about the world, also known as a memory, in addition to what is inherited from its DNAs or genes. This capability has significantly enhanced the survival of the individual and attributed to the explosion of life forms in the Cambrian period about 530 million years ago. Compared to phylogenetic learning, ontogenetic learning is significantly more efficient and predictable, which can be realized with the resource limit of an individual in its lifespan.

Notice that both types of learning mechanisms rely on some form of feedback (from the external environment), in terms of a penalty (death) or reward (food), of a species or individuals' actions<sup>4</sup> to learn. As a result, all intelligent beings, as species or as individuals, rely on a closed-loop feedback mechanism to learn and improve their knowledge about the world. We also notice that from plants, to fish, to birds, and to mammals, more advanced species rely more and more on their ontogenetic learning capabilities. They stay with and learn from their parents longer and longer after birth, because individuals of the same species need to survive in very diverse environments.

**Evolution of human intelligence.** Since the emergence of homo sapiens about 315 thousand years ago, a new and higher form of intelligence emerged which evolves more efficiently and economically. Languages, first spoken<sup>5</sup> and then written<sup>6</sup>, were developed a few thousand years ago. See Figure 1.3. This allows individuals to communicate and share useful information with others. Therefore, a human community or society can behave like a single intelligent organism that can learn much faster and hold more knowledge than any individual. In a way, written languages, or texts, play a role similar to DNAs and genes as they allow human societies to accumulate and pass knowledge of the world onto next generations. We may refer to this type of intelligence as *societal intelligence*, to distinguish it from the phylogenetic intelligence of species and the ontogenetic intelligence of individuals. This type of knowledge accumulation serves as the foundation of ancient civilizations.

Quite miraculously, about a few thousand years ago, another quantum leap in human intelligence occurred, which allowed philosophers and mathematicians to develop knowledge that seem to go way beyond developing empirical knowledge. The development of abstract mathematical concepts and symbols, such as numbers, space and time, as well as mathematical logic, serve as a new precise language for modern science. In addition, the development of the ability to generate new hypotheses and verify their correctness based on logic deduction or scientific experimentation. This, for the first time, has enabled humans to proactively develop new knowledge beyond passive empirical means. The ability to conduct these high-level forms of knowledge development is believed to be unique to humans. This advanced form of intelligence is referred to as “artificial intelligence” (AI), coined by John McCarthy at the Dartmouth summer workshop in 1956.

<sup>4</sup>Gene mutation of the species or actions made by the individual.

<sup>5</sup>It was believed that Sanskrit was the first spoken language, dated as back as 5000 BC.

<sup>6</sup>Sumerian language is believed to be one of the oldest written language in existence, first attested about 3100 BC in southern Mesopotamia.

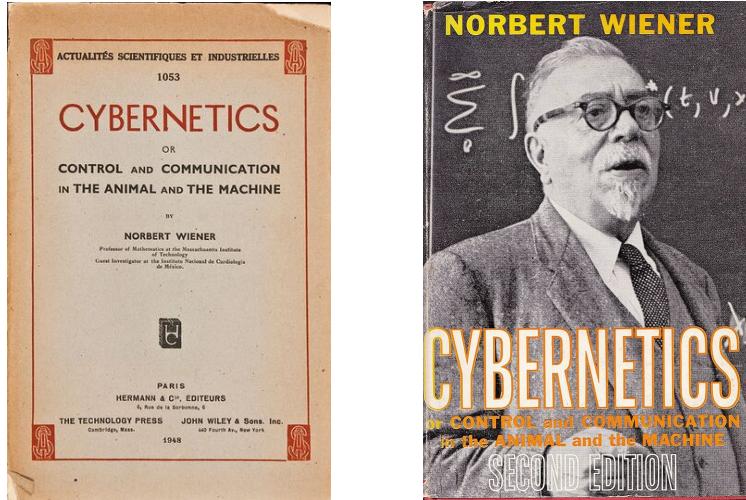


Figure 1.4: The book “Cybernetics” by Norbert Wiener published in 1948 [Wie48] (left) and the second edition in 1961 [Wie61] (right).

Hence, from what we can learn from the nature, from now on, whenever we use the word “intelligence,” we need to be very specific about which level/form of intelligence we mean:

$$\text{phylogenetic} \Rightarrow \text{ontogenetic} \Rightarrow \text{societal} \Rightarrow \text{artificial intelligence}. \quad (1.1.1)$$

A clear characterization and distinction are necessary and important because we want to study intelligence as a scientific and mathematical subject. It is highly likely that, even if they all may share the common objective of learning useful knowledge of the world, the specific computational mechanisms and physical implementations behind each level/form of intelligence could be different. We believe that the reader would better understand and appreciate these differences after having finished study this book. Therefore, we will leave more discussions about general intelligence to the last Chapter 8.

**Origin of machine intelligence – Cybernetics.** In 1940s, partly due to the war effort, intelligence in nature had inspired scientists in the 1940s to emulate animal intelligence by machines, which led to the “Cybernetics” movement advocated by Norbert Wiener. Wiener studied zoology at Harvard as an undergraduate but later became a mathematician and control theorist. Wiener had a life long passion in understanding and developing autonomous systems that could emulate intelligent behaviors of animals. Today, the Cybernetics program is often narrowly interpreted by people as mainly about feedback control systems for which Wiener indeed made his most significant technical contributions. But the Cybernetics program was much broader and deeper than that. It is more about understanding intelligence as a whole<sup>7</sup> and had actually influenced the work of a whole generation of renowned scientists, including Warren McCulloch, Walter Pitts, Claude Shannon, John von Neumann, and Alan Turing.

Wiener was arguably the first person who studied intelligence as a *system*, instead of paying attention to only one component or aspect of it. His comprehensive views on intelligence were elaborated in his famous 1948 book “*Cybernetics: or Control and Communication in the Animal and the Machine*” [Wie48]. In this book and its second edition published in 1961 [Wie61] (see Figure 1.4), he tried to identify several necessary characteristics and mechanisms of intelligent systems, which include (but are not limited to):

- How to *measure and store* information (in the brain) and how to communicate with others.<sup>8</sup> This led to the formulation of information theory and coding theory by Claude Shannon in 1948.
- How to *correct errors* in prediction and estimation based on existing information. Norbert Wiener himself helped formalize the theory for control systems based on closed-loop feedback in the 1940s.

<sup>7</sup> At least at the level of animals.

<sup>8</sup> Norbert Wiener was the first to point out “information” is not matter nor energy, but an independent quantity for study.

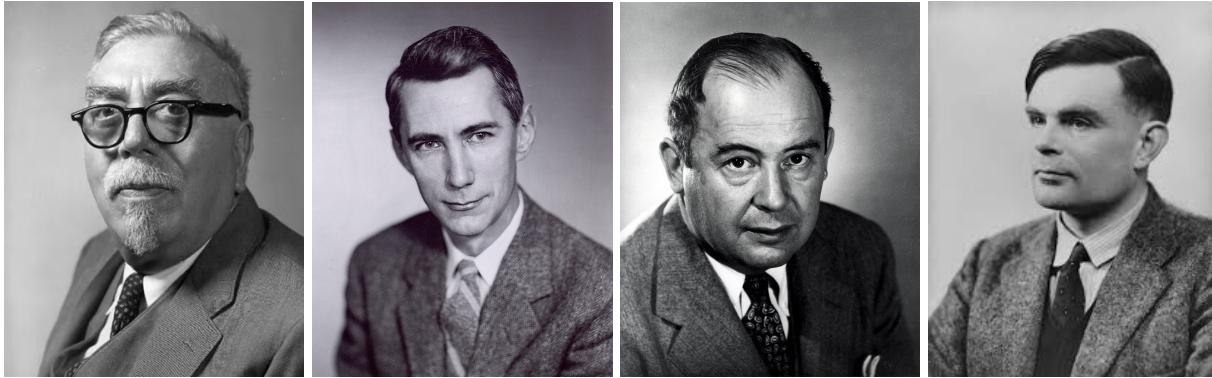


Figure 1.5: Pioneers of theoretical and computational foundations for intelligence: Norbert Wiener (cybernetics and control theory), Claude Shannon (information theory), John von Neumann (game theory), and Alan Turing (computing theory).

- How to learn to *make better decisions* from interacting with a potentially non-cooperative opponent or adversarial environment. This was formalized by John von Neumann as game theory in 1944.

In 1943, very much motivated by Wiener’s Cybernetics program, the psychiatrist Warren McCulloch and the logician Walter Pitts together formalized the first computational model for a neuron [MP43], called *an artificial neuron*, as illustrated later in Figure 1.13. Based on this model, in 1950s, Frank Rosenblatt built a physical machine, named the *Mark I Perceptron*, with a network of hundreds of such artificial neurons. Perceptron was the first artificial neural network physically realized, see Figure 1.15. Notably, John von Neumann’s universal computer architecture, proposed in 1945, was also designed to facilitate the goal of building *computing machines* that can physically realize the mechanisms suggested by the Cybernetics program.

Acute readers probably have noticed that the 1940s was truly a magical era: So many fundamental ideas were invented and influential theories formalized in that era, including the mathematical model of neurons, artificial neural networks, information theory, control theory, game theory, and computing machines. Figure 1.5 shows some of the pioneers of these theories. As we now know, each work above had grown to become the foundation of a scientific or engineering field for the following many decades and has tremendous impact on us. All these fundamental theories were inspired and motivated by the goal of trying to develop machines that can emulate intelligence in nature. Based on historical notes, Wiener’s Cybernetics movement had influenced almost all of these people and work. To a large extent, the Cybernetics program laid out by Wiener can be viewed as the true predecessor to the currently very popular “embodied” intelligence program. In fact, Wiener had described the program in much more concrete terms [Wie61].

Although Wiener had identified in his work many key characteristics and mechanisms of (embodied) intelligence, there was no indication that he knew how to properly integrate all these mechanisms together to build a complete autonomous intelligent system. Judging from today’s knowledge, some of his views on these mechanisms were not entirely accurate or complete. In particular, in the last chapter of the second edition of Cybernetics [Wie61], he pointed out that it is crucial to *deal with nonlinearity* if a machine learning system is designed to emulate typical learning mechanisms in nature. But he did not provide any concrete and effective solutions to this difficult issue. To his defense though, at the time, few people knew how, since even the theory for dealing with linear models and systems was still in its infancy.

Nevertheless, we could not help but marvel at Wiener’s foresight about the importance of nonlinearity. As we will see in this book, the answer was found only recently: nonlinearity can be effectively dealt with through progressive linearization and transformation realized by deep neural networks (see Chapter 4). In addition, we will attempt to show in this book how all these mechanisms listed above can be naturally integrated into a complete system which would exhibit characteristics of an autonomous intelligent system (see Chapter 5).

**Origin of Artificial Intelligence.** From the subtitle of Wiener’s Cybernetics book: “*Control and Communication in the Animal and the Machine*”, one can tell that the studies in the 1940s mainly aimed to emulate intelligence at the level of animals. As we mentioned before, the research agendas about intelligence around the 1940s were very much dominated by Norbert Wiener’s Cybernetics movement.

Alan Turing was one of the first to notice this limitation. In his famous 1950 paper “*Computing Machinery and Intelligence*” [Tur50], Turing formally posted the question whether machines can imitate intelligence even at the human level, to the point of being indistinguishable from the intelligent capabilities of humans. This is now known as the *Turing test*.

Around 1955, a group of young and ambitious scientists tried to break away from the then dominating Cybernetics movement and research agendas so that they would have a chance to create their own legacy. They decided to take on Turing’s challenge of imitating human intelligence and proposed a workshop to be held at Dartmouth College in the summer of 1956. They made their intention clear with a statement in their proposal:

*“The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.”*

In essence, they wanted to formalize and study higher-level intelligence that differentiates humans from animals. The topics they considered ranged from abstraction, symbolic methods, natural languages, and deductive methods (including causal inference, logic deduction, etc.) The organizer of the workshop, John McCarthy, then a young assistant professor of Mathematics of the Dartmouth College, coined the now famous term “Artificial Intelligence” (AI) to encapsulate the set of characteristics or mechanisms that are believed to be *unique to human intelligence*.

**The renaissance of “Artificial Intelligence” or “Cybernetics”?** As the readers may have known, in the past decade or so, machine intelligence has undergone explosive development, powered mainly by the practice of deep artificial neural networks, triggered by the work of Geoffrey Hinton and students in 2012 [KSH12]. This era is also hailed as the “Renaissance” of Artificial Intelligence (AI). However, in terms of tasks that people have actually tried to tackle (recognition, generation, and prediction) and techniques that people have developed and implemented so far (reinforcing, imitating, encoding, decoding, denoising, and compression), we are very much just emulating the mechanisms that are common to the intelligence of early life and animals. Even in that regard, as we will try to clarify in this book, current “AI” models and systems have not correctly implemented all necessary mechanisms for intelligence at these levels, which were already known to the Cybernetics movement in the 1940s.

Hence, strictly speaking, the advancement of machine intelligence in the past decade does not align well with the “Artificial Intelligence” program laid out in the 1956 Dartmouth workshop. Instead, what has been predominantly accomplished so far is more closely related to the objectives of the classic “Cybernetics” program laid out by Norbert Wiener in the 1940s. It is probably more appropriate to call the current era the “Renaissance of Cybernetics”.<sup>9</sup> Only after we have fully understood what we have truly done from the scientific and mathematical perspective, can we truly know what remains to be done and which direction to go to pursue the true nature of intelligence. This is one of the main purposes of this book.

## 1.2 What to Learn?

### 1.2.1 Predictability

Data that carry useful information manifest in many different forms. In the most natural form, they can be modeled as sequences that are predictable and computable. The notion and properties of a predictable and computable sequence were at the heart of the theory of computing and very much led to the invention of

---

<sup>9</sup>The recent rise of the so-called “Embodied AI” for autonomous intelligent robots share even more similarity with the goals of the Cybernetics program.

computers [Tur36]. The role of predictable sequences in (inductive) inference was studied by Ray Solomonoff, Andrey Kolmogorov, and many others in the 1960s [Kol98] as a generalization to Claude Shannon's classic Information Theory [Sha48]. To understand the concept of predictable sequences, let us first start with some concrete simple examples.

**Scalar Case.** The simplest predictable discrete sequence is arguably the sequence of natural numbers:

$$S = 1, 2, 3, 4, 5, 6, \dots, n, n + 1, \dots \quad (1.2.1)$$

in which the next number  $x_{n+1}$  is defined to be its previous number  $x_n$  plus 1:

$$x_{n+1} = x_n + 1. \quad (1.2.2)$$

One may generalize the notion of predictability to any sequence  $\{x_n\}_{n=1}^{\infty}$  with  $x_n \in \mathbb{R}$  if the next number  $x_{n+1}$  can always be computed from its previous one  $x_n$ :

$$x_{n+1} = f(x_n), \quad x_n \in \mathbb{R}, \quad n = 1, 2, 3, \dots \quad (1.2.3)$$

where  $f(\cdot)$  is a *computable* (scalar) function.<sup>10</sup> Note that here we emphasize that the function  $f(\cdot)$  must be computable. There are many functions that can be defined but are not computable. Alan Turing's seminal work in 1936 [Tur36] gives a rigorous definition of computability. In practice, we often further assume that  $f$  is efficiently computable and has nice properties such as being continuous and differentiable, etc. The necessity of these properties will become clear later once we understand more about more refined notions of computability, and their roles in machine learning and intelligence.

**Multi-Variable Case.** Of course, the value of the next number can also depend on two of its predecessors. For example, the famous *Fibonacci sequence* is defined to be:

$$S = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots \quad (1.2.4)$$

where one can easily see:

$$x_{n+2} = x_{n+1} + x_n, \quad x_n \in \mathbb{R}, \quad n = 1, 2, 3, \dots \quad (1.2.5)$$

Similarly, we may generalize this recursion to

$$x_{n+2} = f(x_{n+1}, x_n), \quad x_n \in \mathbb{R}, \quad n = 1, 2, 3, \dots \quad (1.2.6)$$

where  $f(\cdot, \cdot)$  is any computable function that takes two variables as input. We can further generalize the notion of predictability to a sequence whose next value depends on say  $d$  of its predecessors:

$$x_{n+d} = f(x_{n+d-1}, \dots, x_n), \quad x_n \in \mathbb{R}, \quad n = 1, 2, 3, \dots \quad (1.2.7)$$

The number of predecessors  $d$  needed for the prediction is called the *degree* of the recursive prediction. The above expression (1.2.7) is also called an (*auto*) *regression*. Such a sequence is also called a *auto-regressive* sequence. If the function  $f$  is a linear function, we call it a linear (*auto*) regression.

**Vector Case.** To simplify the notation, we may define a vector  $\mathbf{x} \in \mathbb{R}^d$  that collects  $d$  consecutive values in the sequence

$$\mathbf{x}_n \doteq [x_{n+d-1}, \dots, x_n]^{\top}, \quad \mathbf{x}_n \in \mathbb{R}^d, \quad n = 1, 2, 3, \dots \quad (1.2.8)$$

With this notation, the recursive relation (1.2.7) can be conveniently written as

$$\mathbf{x}_{n+1} = g(\mathbf{x}_n) \in \mathbb{R}^d, \quad n = 1, 2, 3, \dots \quad (1.2.9)$$

where the function  $g(\cdot)$  is uniquely defined by the function  $f$  in (1.2.7) and it takes a  $d$ -dimensional vector as input. In different contexts, such a vector is sometimes referred to as a “state” or a “token”. Note that the equation in (1.2.7) denotes a mapping  $\mathbb{R}^d \rightarrow \mathbb{R}$ , but the equation here is  $g : \mathbb{R}^d \rightarrow \mathbb{R}^d$ .

---

<sup>10</sup>Here we emphasize that the function  $f(\cdot)$  itself is computable, say it can be implemented as a program on a computer.

**Controlled Prediction.** We may also define a predictable sequence that depends on another predictable sequence as input:

$$\mathbf{x}_{n+1} = f(\mathbf{x}_n, \mathbf{u}_n) \in \mathbb{R}^d, \quad n = 1, 2, 3, \dots, \quad (1.2.10)$$

where  $\{\mathbf{u}_n\}$  with  $\mathbf{u}_n \in \mathbb{R}^k$  is a (computable) predictable sequence. In other words, the next vector  $\mathbf{x}_{n+1} \in \mathbb{R}^d$  depends on both  $\mathbf{x}_n \in \mathbb{R}^d$  and  $\mathbf{u}_n \in \mathbb{R}^k$ . In the context of control theory, the sequence  $\{\mathbf{u}_n\}$  is often referred to as the “control input” and  $\mathbf{x}_n$  as the “state” or “output” of the system (1.2.10). One classic example is a linear dynamical system:

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n, \quad \mathbf{A} \in \mathbb{R}^{d \times d}, \mathbf{B} \in \mathbb{R}^{d \times k}, \quad (1.2.11)$$

which is widely studied in control theory [CD91].

Very often the control input is given by a computable function of the state  $\mathbf{x}_n$  itself, say:

$$\mathbf{u}_n = h(\mathbf{x}_n), \quad n = 1, 2, 3, \dots \quad (1.2.12)$$

As a result, the sequence  $\{\mathbf{x}_n\}$  is given by composing the two computable functions  $f$  and  $h$  as:

$$\mathbf{x}_{n+1} = f(\mathbf{x}_n, h(\mathbf{x}_n)), \quad n = 1, 2, 3, \dots \quad (1.2.13)$$

In this way, the sequence  $\{\mathbf{x}_n\}$  again becomes an auto-regressive predictable sequence. When the input  $\mathbf{u}_n$  depends on the output  $\mathbf{x}_n$ , we say the resulting sequence is produced by a “closed-loop” system (1.2.13). As the closed-loop system no longer depends on any external input, we say such a system has become *autonomous*. It can be viewed as a special case of auto-regression. For instance, if we choose in the above linear system (1.2.11),  $\mathbf{u}_n = \mathbf{F}\mathbf{x}_n$ , the closed-loop system becomes

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n = \mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{F}\mathbf{x}_n = (\mathbf{A} + \mathbf{B}\mathbf{F})\mathbf{x}_n, \quad (1.2.14)$$

which is a linear auto-regression.

**Continuous Processes.** Predictable sequences have their natural counterparts in the continuous case. We may refer to them as predictable processes. Similar to the sequence of natural numbers, the simplest predictable continuous process is time itself  $x = t$ .

More generally, we say a process, denoted by  $\mathbf{x}(t)$ , is predictable if at any time  $t$ , the value of the process at  $t + \delta t$ , where  $\delta t$  is an infinitesimal increment, is determined by its value at  $t$ . Typically, the change in value  $\delta\mathbf{x}(t)$  is continuous and smooth. So  $\delta\mathbf{x}(t) = \mathbf{x}(t + \delta t) - \mathbf{x}(t)$  is infinitesimally small. Predictable processes are typically described by (multivariate) differential equations:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)), \quad \mathbf{x} \in \mathbb{R}^d. \quad (1.2.15)$$

In the context of systems theory [CD91; Sas99], the above equation is also known as a state-space model. Similar to the discrete case, a controlled process can be given by:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x} \in \mathbb{R}^d, \mathbf{u} \in \mathbb{R}^k, \quad (1.2.16)$$

where  $\mathbf{u}(t)$  is a computable input process.

*Example 1.1.* For example in physics, Newton’s second law of motion describes how to predict the trajectory  $\mathbf{x}(t) \in \mathbb{R}^3$  of a moving object under a force input  $\mathbf{F}(t) \in \mathbb{R}^3$ :

$$m\ddot{\mathbf{x}}(t) = \mathbf{F}(t). \quad (1.2.17)$$

When there is no force  $\mathbf{F}(t) \equiv 0$ , the above law reduces to a special case, known as Newton’s first law: the object maintains a constant speed in a straight line:

$$\ddot{\mathbf{x}}(t) = \mathbf{0} \Leftrightarrow \dot{\mathbf{x}}(t) = \mathbf{v} \quad (1.2.18)$$

for some constant velocity vector  $\mathbf{v} \in \mathbb{R}^3$ . ■

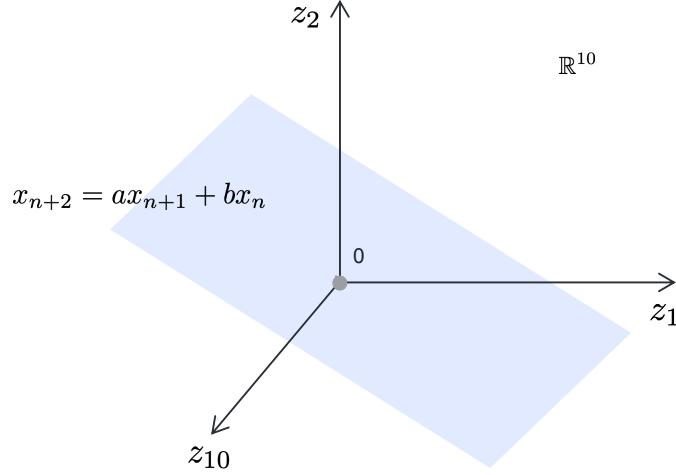


Figure 1.6: A two-dimensional subspace in a ten-dimensional ambient space.

### 1.2.2 Low Dimensionality

**Learn to Predict.** Now suppose you have observed or are given many sequence segments:

$$\{S_1, S_2, \dots, S_i, \dots, S_N\} \quad (1.2.19)$$

all from some predictable sequence  $\{x_n\}_{n=1}^{\infty}$ . Without loss of generality, we may assume the length of each segment is  $D \gg d$ . So each segment is of the form:

$$S_i = [x_{j(i)}, x_{j(i)+1}, \dots, x_{j(i)+D-1}]^\top \in \mathbb{R}^D \quad (1.2.20)$$

for some  $j \in \mathbb{N}$ . Then you are given a new segment  $S_t$  and are asked to predict its future values.

One difficulty here is that you normally do not know the function  $f$  and the degree  $d$  from which the sequence is generated:

$$x_{n+d} = f(x_{n+d-1}, \dots, x_n). \quad (1.2.21)$$

So the hope is somehow “to learn”  $f$  and  $d$  from the given sample segments  $S_1, S_2, \dots, S_N$ . Hence the central task of learning to predict is:

*Given many sampled segments of a predictable sequence, how to effectively and efficiently identify the function  $f$ .*

**Predictability and Low-Dimensionality.** To identify the predictive function  $f$ , we may notice a common characteristic of segments of any predictable sequence, say given by (1.2.21). If we take a long segment, say with a length  $D \gg d$ , of the sequence and view it as a vector:

$$\mathbf{x}_i = [x_i, x_{i+1}, \dots, x_{i+D-1}]^\top \in \mathbb{R}^D. \quad (1.2.22)$$

Then the set of all such vectors  $\{\mathbf{x}_i\}$  are far from random and hence cannot possibly occupy the entire space of  $\mathbb{R}^D$ . Instead, they essentially have at most  $d$  degrees of freedom – given the first  $d$  entries of any  $\mathbf{x}_i$ , values of the rest of the entries are uniquely determined. In other words, all  $\{\mathbf{x}_i\}$  lie on a  $d$ -dimensional surface. In mathematics, such a surface is often called a submanifold, denoted as  $\mathcal{S} \subset \mathbb{R}^D$ .

In practice, if we choose the segment length  $D$  to be large enough, then all segments sampled from the same predicting function lie on a surface with an intrinsic dimension  $d$ , significantly lower than that of the ambient space  $D$ . For example, if the sequence is given by the following linear autoregression:

$$x_{n+2} = a \cdot x_{n+1} + b \cdot x_n, \quad (1.2.23)$$

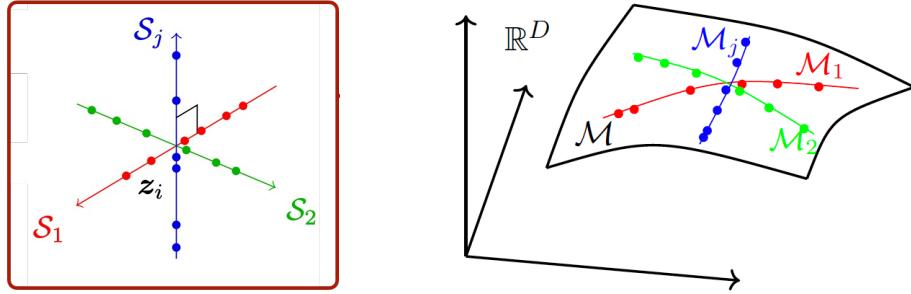


Figure 1.7: Data distributed on a mixture of (orthogonal) subspaces (left) or submanifolds (right).

for some constants  $a, b \in \mathbb{R}$ . If we sample segments of length  $D = 10$  from such a sequence, then all samples lie on a two-dimensional plane or subspace in  $\mathbb{R}^{10}$ , as illustrated in Figure 1.6. If we can identify this two-dimensional subspace, the constants  $a$  and  $b$  in (1.2.23) can be fully determined.

It is easy to see that if the predicting function  $f$  is linear, such as the case with the linear systems given in (1.2.11) and (1.2.14), the long segments always lie on certain low-dimensional linear subspace. Identifying the predicting function is largely equivalent to identifying this low-dimensional subspace, a problem generally known as principal component analysis. We will discuss such classic models and methods in Chapter 2.

As it turns out, this is largely true for general predictable sequences too: if one can identify the low-dimensional surface on which all the segment samples lie, then one can identify the associated predictive function  $f$ .<sup>11</sup> We cannot over-emphasize the importance of this property of segments from a predictable sequence: *All samples of long segments of a predictable sequence lie on a low-dimensional submanifold*. As we will see in this book, all modern learning methods essentially exploit this property, implicitly or explicitly.

In real-world scenarios, the data we observe often do not come from a single predictable sequence. Typically they contain observations of multiple predictable sequences. For example, a video sequence can contain multiple moving objects. It is easy to see that in such scenarios, the data lie on a mixture of multiple low-dimensional linear subspaces or nonlinear submanifolds, as illustrated in Figure 1.7.

**Properties of Low-Dimensionality.** Of course, temporal correlation in predictable sequences is not the only reason why data are low-dimensional. For example, the space of all images is humongous but most of the space consists of images that resemble structureless random images as shown in Figure 1.8 left. Natural images and videos however are highly redundant because there is a strong spatial and temporal correlation among all pixel values. This is the reason why we can easily recognize whether an image is noisy or clean, as shown in Figure 1.8 middle and right. Hence the distribution of natural images has a very low intrinsic dimension (relative to the total number of pixels of an image).

Due to the importance and ubiquity of the task of learning low-dimensional structures, the book “*High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*” [WM22] has stated in the beginning with a statement: “*The problem of identifying the low-dimensional structure of signals or data in high-dimensional spaces is one of the most fundamental problems that, through a long history, interweaves many engineering and mathematical fields such as system theory, signal processing, pattern recognition, machine learning, and statistics.*”

Note that by enforcing the observed data point  $\mathbf{x}$  to be on a low-dimensional surface, we essentially have made the entries of  $\mathbf{x}$  very dependent on each other and in some sense have made the entries very “predictable” from values of other entries. For example, if we know that the data are constrained on a  $d$ -dimensional surface in  $\mathbb{R}^d$ , then it allows us to do many useful things with the data besides prediction:

- **completion:** in general, given more than  $d$  entries of a typical sample  $\mathbf{x}$ , the rest of its entries usually can be *uniquely determined*.<sup>12</sup>

<sup>11</sup>Under mild conditions, there is a one-to-one mapping between the low-dimensional surface and the function  $f$ . This fact has been exploited in problems such as system identification which we will discuss later.

<sup>12</sup>Prediction becomes a special case of this property.

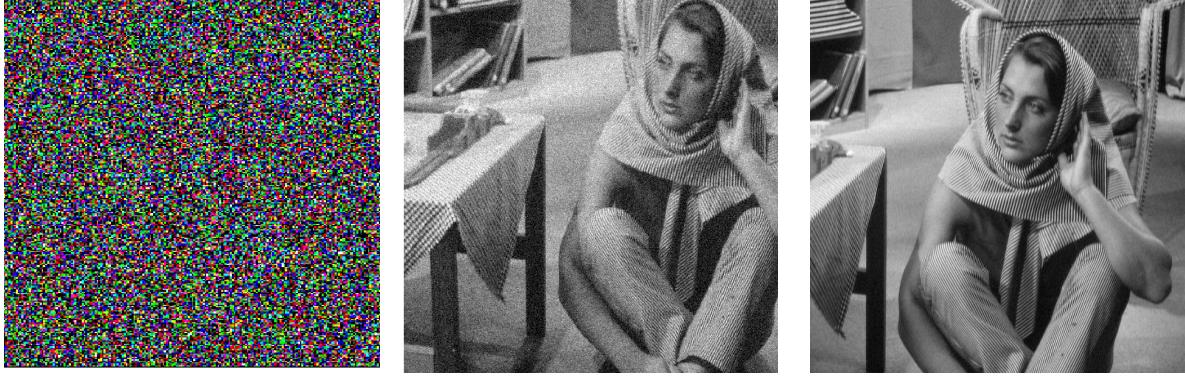


Figure 1.8: An image of random noise versus a noisy image and the original clean image.

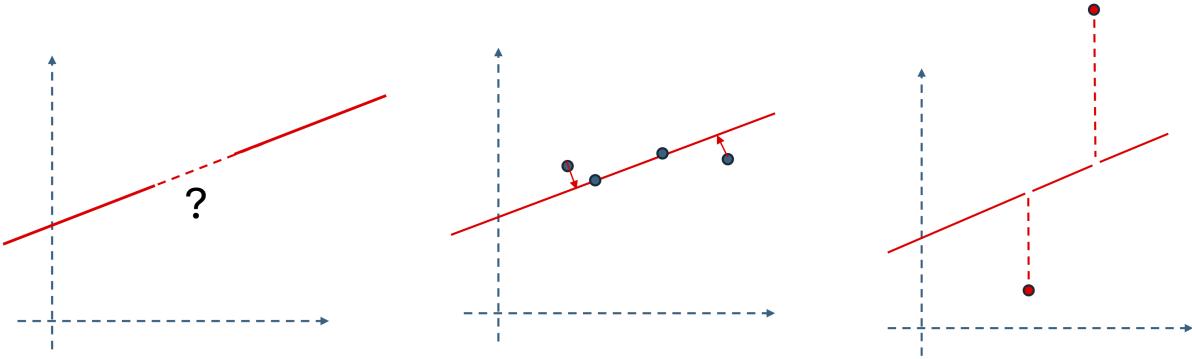


Figure 1.9: Illustration of properties of a low-dimensional (linear) structure: it enables completion (left), denoising (middle), and error correction (right).

- **denoising:** suppose entries of a sample  $\mathbf{x}$  are perturbed by *small* noises, they can be effectively removed by projecting  $\mathbf{x}$  back onto the surface.
- **error correction:** suppose a small number of *unknown* entries of  $\mathbf{x}$  are arbitrarily corrupted, they can be effectively and efficiently corrected.

Figure 1.9 illustrates these properties with a low-dimensional linear structure: a one-dimensional line in a two-dimensional plane.

In fact, under mild conditions, the above properties are generalizable to many other low-dimensional structures in high-dimensional spaces [WM22]. Interestingly, as we will see in this book, these useful properties such as completion and denoising will inspire effective methods to learn such low-dimensional structures.

In the above, for simplicity, we have only used the deterministic case to introduce the important notion of predictability and low-dimensionality. Hence, the data (or sampled segments) precisely lie on some geometric structures such as subspaces or surfaces. In practice, as we have alluded to before, there is always a certain level of uncertainty or randomness in the data. In this case, we may assume that the data have a probability distribution, with a probability density function  $p(\mathbf{x})$ . We say that a distribution is “low-dimensional” if its density concentrates around a geometric structure that is rather low-dimensional, say a subspace, a surface or a mixture of them, as shown in Figure 1.7. Notice that, from a practical point of view, such a density function  $p(\mathbf{x})$ , once learned, can serve as a very powerful prior for estimating  $\mathbf{x}$  based on partial, noising, or corrupted observation, say:

$$\mathbf{y} = f(\mathbf{x}) + \mathbf{n}, \quad (1.2.24)$$

by computing the conditional estimation  $\hat{\mathbf{x}}(\mathbf{y}) = \mathbb{E}(\mathbf{x} \mid \mathbf{y})$  or through sampling the conditional distribution

$$\hat{x}(\mathbf{y}) \sim p(\mathbf{x} \mid \mathbf{y}).^{13}$$

Our discussions above have led to the main and only assumption on which this book will make for a deductive approach to understand intelligence and deep networks in particular:

**The Main Assumption:** *Any intelligent systems or learning methods should and could rely on is that the world is predictable hence the distribution of the observed high-dimensional data samples have low-dimensional supports.*

The remaining question is how to learn such low-dimensional structures correctly from the high-dimensional data, via effective and efficient computable means. As we will see in this book, parametric models that were well studied in classic analytical approaches and non-parametric models such as deep networks that are popular in modern practice are merely different means to achieve the same goal.

## 1.3 How to Learn?

### 1.3.1 Analytical Approaches

Note even if a predictive function is tractable to compute, it does not imply it is tractable or scalable to learn this function from a number of sampled segments. Of course, one classical approach to ensure the problems are tractable or amenable to efficient solutions is to make explicit assumptions about the family of low-dimensional structures we are dealing with. Historically, due to limited computation and data, simple and idealistic analytical models were always the first to be studied as they often offer efficient closed-form or numerical solutions. In addition, they can provide insights to the more general problems and they often already provide useful solutions to important though limited cases. In old days when computational resource was scarce, analytical models that permitted efficient closed-form or numerical solutions were the only cases that could be implemented. *Linear structures* became the first classes of models to be thoroughly studied.

For example, arguably the simplest case is to assume the data is distributed around a single low-dimensional subspace in a high-dimensional space. Or somewhat equivalently, one may assume the data is distributed according to an almost degenerate low-dimensional Gaussian. Identifying such a subspace or Gaussian from a finite number of (noisy) samples is then the classical problem of principal component analysis (PCA) and effective algorithms have been developed for this class of models [Jol02]. One can make the family of models increasingly more complex and expressive. For instance, one may assume the data are distributed around a certain mixture of low-dimensional components (subspaces or low-dimensional Gaussians), as in the case of independent component analysis (ICA) [BJC85], dictionary learning (DL), generalized principal component analysis (GPCA) [VMS05], or the even more general class of sparse low-dimensional models that have been studied extensively in recent years in fields such as compressive sensing [WM22].

Around all these analytical model families, the central problem of study is always how to identify *the most compact* model within each family that best fits the given data. Below, we give a very brief account of these classical analytical models but leave a more systematic study to Chapter 2. In theory, these analytical models have provided us with tremendous insights into the geometric and statistical properties of low-dimensional structures. They often give us closed-form solutions or efficient and scalable algorithms which are very useful for data whose distributions can be well approximated by such models. More importantly, for more general problems, they provide us with a general sense of how easy or difficult the problem of identifying low-dimensional structures can be, and what the basic ideas are to approach such a problem.

#### Linear Dynamical Systems

**Wiener Filter.** As we have discussed before in Section 1.2.1, a main task of intelligence is to learn what is predictable in sequences of observations. Probably the simplest class of predictable sequences, or signals, are generated via a *linear time-invariant* (LTI) process:

$$x[n] = h[n] * z[n] + \epsilon[n], \quad (1.3.1)$$

---

<sup>13</sup>Modern generative AI technologies such as (conditioned) image generation very much rely on this fact, as we will elaborate on in Chapter 6.

where  $z$  is the input and  $h$  is the impulse response function.<sup>14</sup> Here  $\epsilon[n]$  is some additive noise in the observations. The problem is that given the input process  $\{z[n]\}$  and observations of the output process  $\{x[n]\}$ , how to find the optimal  $h[n]$  such that  $\hat{x}[n] = h[n] * z[n]$  predicts  $x[n]$  is an optimal way. In general, we measure the goodness of the prediction by the minimum mean squared error (MMSE):

$$\min_h \mathbb{E}[\epsilon[n]^2] = \mathbb{E}[\|x[n] - h[n] * z[n]\|_2^2]. \quad (1.3.2)$$

The optimal solution  $h[n]$  is referred to as a (denoising) filter. Norbert Wiener, the same person initiated the Cybernetics movement, studied this problem in 1940s and gave an elegant closed-form solution known as the *Wiener filter* [Wie42; Wie49]. This became one of the most fundamental results in the field of Signal Processing.

**Kalman Filter.** The idea of denoising or filtering for a dynamic process was later extended to a linear time-invariant system described by a (finite-dimensional) state-space model by Rudolph Kalman in the 1960s:

$$z[n] = Az[n-1] + Bu[n] + \epsilon[n]. \quad (1.3.3)$$

The problem is how we can estimate the state of the system  $z[n]$  from noisy observations of the form:

$$x[n] = Cz[n] + w[n], \quad (1.3.4)$$

where  $w$  is some (white) noise. The optimal causal<sup>15</sup> state estimator that minimizes the MMSE-type prediction error

$$\min \mathbb{E}[\|x[n] - Cz[n]\|_2^2] \quad (1.3.5)$$

is given in closed-form by the so-called *Kalman filter* [Kal60]. This is one of the corner stones of modern Control Theory because it allows us to estimate the state of a dynamical system from its noisy observations. Then one can subsequently introduce a (linear) state feedback, say of the form  $u[n] = F\hat{z}[n]$ , and make the closed-loop system fully autonomous, as we saw in equation (1.2.13).

**Identification of Linear Dynamical Systems.** To derive the Kalman filter, the system parameters  $(A, B, C)$  are assumed to be known. If they are not given in advance, it would be a more challenging problem known as *system identification*: how to *learn* the parameters  $(A, B, C)$  from (many samples of) the input sequence  $\{u[n]\}$  and observation sequence  $\{x[n]\}$ . This is a classic problem in systems theory. If the system is linear, it can be shown that the input and output sequences  $\{u[n], x[n]\}$  would jointly lie on certain low-dimensional subspace<sup>16</sup>. Hence the identification problem is essentially equivalent to identifying this low-dimensional subspace [LV09; LV10; VM96].

Note that the above problems have two things in common: first, the (noise-free) sequences or signals are assumed to be generated by an explicit family of parametric models; second, these models essentially are all linear. So conceptually, let  $x_o$  be a random variable whose “true” distribution is supported on a low-dimensional linear subspace, say  $S$ . To a large extent, Wiener filter and Kalman filter all try to estimate such an  $x_o$  from its noisy observations:

$$x = x_o + \epsilon, \quad x_o \sim S, \quad (1.3.6)$$

where  $\epsilon$  is typically a random Gaussian noise (or process). Hence, essentially, their solutions all rely on identifying a low-dimensional linear subspace that best fits the observed (noisy) data. Then by projecting the data onto this subspace, one obtains the optimal denoising operations, all in closed form.

---

<sup>14</sup>Normally  $h$  is assumed to have certain nice structures, say finite length or banded spectrum.

<sup>15</sup>Which means the estimation can only use observations up to the current time step  $n$ . Kalman filter is always causal whereas Wiener filter needs not to be.

<sup>16</sup>which has the same dimension as the order of the state-space model (1.3.3).

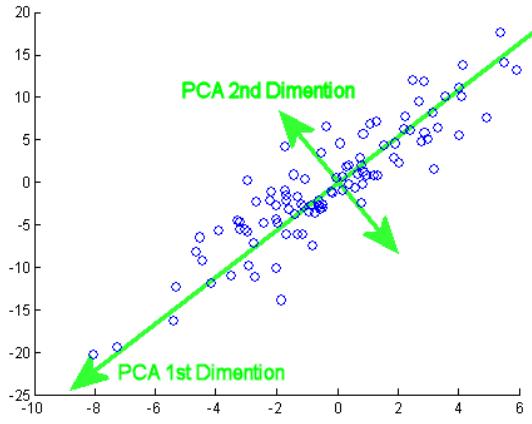


Figure 1.10: A distribution with two principal components.

### Linear and Mixed Linear Models

**Principal Component Analysis.** From the above problems in classical signal processing and system identification, we see that the main task behind of all these problems is to learn from noisy observations a *single* low-dimensional linear subspace. Mathematically, we may model such a structure as:

$$\mathbf{x} = \mathbf{u}_1 z_1 + \mathbf{u}_2 z_2 + \cdots + \mathbf{u}_d z_d + \boldsymbol{\epsilon} = \mathbf{U}\mathbf{z} + \boldsymbol{\epsilon}, \quad \mathbf{U} \in \mathbb{R}^{D \times d} \quad (1.3.7)$$

where  $\boldsymbol{\epsilon} \in \mathbb{R}^D$  is some small random noise. Figure 1.10 illustrates such a distribution with two principal components. The problem is to find the subspace basis  $\mathbf{U}$  from many samples of  $\mathbf{x}$ . A typical approach to estimate the subspace  $\mathbf{U}$  is to minimize the variance of the noise, also known as the minimum mean square error (MMSE):

$$\min_{\mathbf{U}} \mathbb{E}[\|\boldsymbol{\epsilon}\|_2^2] = \mathbb{E}[\|\mathbf{x} - \mathbf{U}\mathbf{z}\|_2^2]. \quad (1.3.8)$$

Notice that this is essentially a denoising task: once the basis  $\mathbf{U}$  is correctly found, we can denoise the noisy sample  $\mathbf{x}$  by projecting it onto the low-dimensional subspace spanned by  $\mathbf{U}$  as

$$\mathbf{x} \rightarrow \hat{\mathbf{x}} = \mathbf{U}\mathbf{U}^\top \mathbf{x}. \quad (1.3.9)$$

If the noise is small and if we learned the correct low-dimensional subspace  $\mathbf{U}$ , we should expect  $\mathbf{x} \approx \hat{\mathbf{x}}$ . That is, PCA is a special case of the auto-encoding:

$$\mathbf{x} \xrightarrow{\mathbf{U}^\top} \mathbf{z} \xrightarrow{\mathbf{U}} \hat{\mathbf{x}}. \quad (1.3.10)$$

Only here because of the simple data structure, the encoder  $\mathcal{E}$  and decoder  $\mathcal{D}$  both become simple linear operators ((projecting and lifting)).

This is a classic problem in statistics known as the Principal Component Analysis (PCA). It was first studied by Pearson in 1901 [Pea01] and later independently by Hotelling in 1933 [Hot33]. This topic is systematically summarized in the classic book [Jol02; Jol86]. In addition, one may explicitly assume the data  $\mathbf{x}$  is distributed according to a single low-dimensional Gaussian:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{U}\mathbf{U}^\top + \sigma\mathbf{I}), \quad \mathbf{U} \in \mathbb{R}^{D \times d}, \quad (1.3.11)$$

which is equivalent to assuming that  $\mathbf{z}$  in the above PCA model (1.3.7) is a standard normal distribution. This is known as Probabilistic PCA [TB99].

In this book, we will revisit PCA in Chapter 2, from the perspective of learning a low-dimensional distribution. Our goal is to use this simple and idealistic model to convey some of the most fundamental ideas for learning a compact representation for a low-dimensional distribution, including the important notion of compression via denoising and autoencoding for a consistent representation.

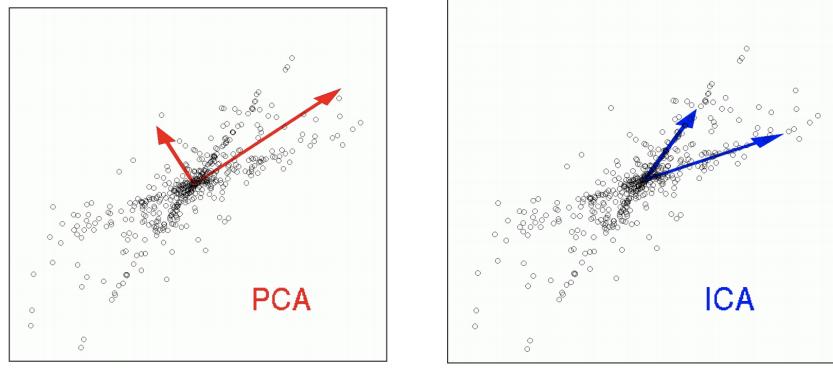


Figure 1.11: PCA (left) versus ICA (right).

**Independent Component Analysis.** Independent component analysis (ICA) was originally proposed by [BJC85] as a classic model for *learning a good representation*. In fact it was originally proposed as a simple mathematical model for our memory. The ICA model takes a deceptively similar form as the above PCA model (1.3.7) by assuming that the observed random variable  $\mathbf{x}$  is a linear superposition of multiple independent components  $z_i$ :

$$\mathbf{x} = \mathbf{u}_1 z_1 + \mathbf{u}_2 z_2 + \cdots + \mathbf{u}_d z_d + \boldsymbol{\epsilon} = \mathbf{U}\mathbf{z} + \boldsymbol{\epsilon}. \quad (1.3.12)$$

However, here the components  $z_i$  are assumed to be independent *non-Gaussian* variables. For example, a popular choice is

$$z_i = \sigma_i \cdot w_i, \quad \sigma_i \sim B(1, p), \quad (1.3.13)$$

where  $\sigma_i$  is a Bernoulli random variable and  $w_i$  could be a constant value or another random variable, say Gaussian.<sup>17</sup> The ICA problem is trying to identify both  $\mathbf{U}$  and  $\mathbf{z}$  from observed samples of  $\mathbf{x}$ . Figure 1.11 illustrates the difference between ICA and PCA.

Although the (decoding) mapping from  $\mathbf{z}$  to  $\mathbf{x}$  seems linear and easy once  $\mathbf{U}$  and  $\mathbf{z}$  are learned, the (encoding) mapping from  $\mathbf{x}$  to  $\mathbf{z}$  can be complicated and may not be represented by a simple linear mapping. Hence ICA generally gives an autoencoding of the form:

$$\mathbf{x} \xrightarrow{\mathcal{E}} \mathbf{z} \xrightarrow{\mathbf{U}} \hat{\mathbf{x}}. \quad (1.3.14)$$

Hence, unlike PCA, ICA is a little more difficult to analyze and solve. In 1990s, folks like Erkki Oja and Aapo Hyvärinen [HO97; HO00b] have made significant theoretical and algorithmic contributions to ICA. In Chapter 2, we will study and give a solution to ICA from which the encoding mapping  $\mathcal{E}$  will become clear.

**Sparse Structures and Compressive Sensing.** As one may see, if  $p$  in (1.3.13) is very small, the probability that any of the components is non-zero is small. In this case, we say  $\mathbf{x}$  is sparsely generated and it concentrates on a set of linear subspaces of dimension  $k = p \cdot d$ . Hence, to some extent, we may extend the above ICA model to a more general family of low-dimensional structures known as sparse models.

A  $k$ -sparse model is defined to be consisting of the set of all  $k$ -sparse vectors:

$$\mathcal{Z} = \{\mathbf{z} \in \mathbb{R}^n \mid \|\mathbf{z}\|_0 \leq k\}, \quad (1.3.15)$$

where  $\|\cdot\|_0$  is the  $\ell^0$ -norm that counts the number of non-zero entries in a vector  $\mathbf{z}$ . That is,  $\mathcal{Z}$  is the union of all  $k$ -dimensional subspaces that align with the coordinate axes, as illustrated in Figure 1.7 left. One important problem in classic signal processing and statistics is how to recover a sparse vector  $\mathbf{z}$  from its linear observations:

$$\mathbf{x} = \mathbf{A}\mathbf{z} + \boldsymbol{\epsilon}, \quad \mathbf{A} \in \mathbb{R}^{m \times n} \quad (1.3.16)$$

<sup>17</sup>Even if  $w$  is Gaussian,  $\sigma w$  is no longer a Gaussian variable!

where  $\mathbf{A}$  is given but typically  $m < n$  and  $\epsilon$  is some small noise. This seemingly benign problem turns out to be NP-hard to compute and it is even hard to approximate (see the book [WM22] for details).

So despite a very rich and long history of study which can be dated back as early as the 18th century [Bos50], there was no provably efficient algorithm to solve this class of problems, although many heuristic algorithms have been proposed and developed between the 1960s and the 1990s. Some are rather effective in practice but without any rigorous justification. A major breakthrough came in the early 2000s when a few renowned mathematicians including David Donoho, Emmanuel Candès, and Terence Tao [CT05a; CT05b; Don05] established a rigorous theoretical framework that allows us to characterize precise conditions under which the sparse recovery problem can be solved effectively and efficient, say via a convex  $\ell^1$  minimization:

$$\min \|\mathbf{z}\|_1 \quad \text{subject to} \quad \|\mathbf{x} - \mathbf{A}\mathbf{z}\|_2 \leq \epsilon, \quad (1.3.17)$$

where  $\|\cdot\|_1$  is the sparsity-promoting  $\ell^1$  norm of a vector and  $\epsilon$  is some small constant. Any solution to this problem essentially gives a sparse coding mapping:

$$\mathbf{x} \xrightarrow{\mathcal{E}} \mathbf{z}. \quad (1.3.18)$$

We will give a brief account of such an algorithm, hence mapping, in Chapter 2 which will reveal interesting fundamental connections between sparse coding and deep learning.<sup>18</sup>

As it turns out, conditions under which  $\ell^1$  minimization succeeds are surprisingly general. The minimum number of measurements  $m$  required for a successful recovery is only proportional to the intrinsic dimension of the data  $k$ . This is now known as the *compressive sensing* phenomenon [Can06]. Moreover, this phenomenon is not unique to sparse structures. It also applies to very broad families of low-dimensional structures such as low-rank matrices etc. These results have fundamentally changed our understanding about recovering low-dimensional structures in high-dimensional spaces. This dramatic reverse of fortune with high-dimensional data analysis was even celebrated as the “*blessing of dimensionality*” by David Donoho [D D00] as opposed to the typical pessimistic belief in “*curse of dimensionality*” for high-dimensional problems. This coherent and complete body of results has been systematically organized in the book [WM22].

From a computational perspective, one cannot over-estimate the significance of this new framework. It has fundamentally changed our views about an important class of problems previously believed to be largely intractable. It has enabled us to develop extremely efficient algorithms that scale gracefully with the dimension of the problem, hence making the problem of sparse recovery from:

$$\text{intractable} \implies \text{tractable} \implies \text{scalable}. \quad (1.3.19)$$

These algorithms also come with rigorous theoretical guarantees of their correctness given precise requirements in data and computation. The rigorous and precise nature of this approach is almost opposite to that of practicing deep neural networks, which is largely empirical. Yet, despite their seemingly opposite styles and standards, we now know both approaches share a common goal: *trying to pursue low-dimensional structures in high-dimensional spaces*.

**Dictionary Learning.** Conceptually, an even harder problem than the sparse coding problem (1.3.16) is when the observation matrix  $\mathbf{A}$  is not even known in advance and we need to learn  $\mathbf{A}$  from a set of (possibly noisy) observations, say  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ :

$$\mathbf{X} = \mathbf{A}\mathbf{Z} + \mathbf{E}, \quad \mathbf{A} \in \mathbb{R}^{m \times n}. \quad (1.3.20)$$

Here we are given only  $\mathbf{X}$  but not the corresponding  $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n]$  nor the noise term  $\mathbf{E} = [\epsilon_1, \epsilon_2, \dots, \epsilon_n]$ , except for knowing  $\mathbf{z}_i$ 's are sparse. This is known as the *dictionary learning* problem, which can be viewed as a generalization to the ICA problem (1.3.12) discussed earlier. In other words, given the distribution of the data  $\mathbf{X}$  to be the image of a standard sparse distribution  $\mathbf{Z}$  under a linear transform  $\mathbf{A}$ , we like to learn  $\mathbf{A}$  and its “inverse” mapping  $\mathcal{E}$  such that we obtain an autoencoder:

$$\mathbf{X} \xrightarrow{\mathcal{E}} \mathbf{Z} \xrightarrow{\mathbf{A}} \mathbf{X}? \quad (1.3.21)$$

---

<sup>18</sup>Although similarities between algorithms for sparse coding and deep networks were noticed as early as in 2010 [GL10].

One can see what is in common with PCA, ICA, and Dictionary Learning is that they all assume that the distribution of the data is supported around low-dimensional linear or mixed linear structures. They all require to learn a (global or local) basis of the linear structures, from probably noisy samples of the distribution. In Chapter 2, we will study how to identify low-dimensional structures through these classical models. In particular, we will see an interesting and important fact: all these low-dimensional (piecewise) linear models can be learned effectively and efficiently by the same type of fast algorithms, known as *power iteration* [ZMZ+20]. Although the above linear or mixed linear models are somewhat too simplistic or idealistic for most real-world data, understanding these models is an important first step toward understanding more general low-dimensional distributions.

### General Distributions

The distributions of real-world data such as images, videos, and audio are too complex to be modeled by above, somewhat idealistic, linear models or Gaussian processes. We normally do not know *a priori* they are generated from which family of parametric models.<sup>19</sup> In practice, we typically only have many samples from their distributions – the empirical distributions. Obviously, in such cases, we normally cannot expect to have any closed-form solutions for their low-dimensional structures, nor for the resulting denoising operators.<sup>20</sup> So we need to develop a more general solution to these empirical distributions, not necessarily in closed form but at least efficiently computable. If we did this correctly, solutions to the aforementioned linear models should become their special cases.

**Denoising.** In the 1950s, statisticians became interested in the problem of denoising data drawn from an arbitrary distribution. Let  $\mathbf{x}_o$  be a random variable with probability density function  $p_o(\cdot)$ . So if we observe a noisy version of  $\mathbf{x}_o$ :

$$\mathbf{x} = \mathbf{x}_o + \sigma \mathbf{g}, \quad (1.3.22)$$

where  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is standard Gaussian noise and  $\sigma$  is the noise level of the observation. Let  $p(\cdot)$  be the probability density function of  $\mathbf{x}$ ,<sup>21</sup> Amazingly, the posterior expectation of  $\mathbf{x}_o$  given  $\mathbf{x}$  can be calculated by an elegant formula, known as the Tweedie's formula [Rob56]:<sup>22</sup>

$$\hat{\mathbf{x}}_o = \mathbb{E}[\mathbf{x}_o | \mathbf{x}] = \mathbf{x} + \sigma^2 \nabla \log p(\mathbf{x}). \quad (1.3.23)$$

As can be seen, from the formula, the function  $\nabla \log p(\mathbf{x})$  plays a very special role in denoising the observation  $\mathbf{x}$  here. The noise  $\mathbf{g}$  can be explicitly estimated as

$$\hat{\mathbf{g}} = \frac{\mathbf{x} - \hat{\mathbf{x}}_o}{\sigma} = -\sigma \nabla \log p(\mathbf{x}), \quad (1.3.24)$$

for which we only need to know the distribution  $p(\cdot)$  of  $\mathbf{x}$  not the ground truth  $p_o(\cdot)$  for  $\mathbf{x}_o$ . An important implication of this result is that if we add Gaussian noise to any distribution, the denoising process can be done easily if we can somehow get hold of the function  $\nabla \log p(\mathbf{x})$ .

Because this is such an important and useful result, it has been rediscovered and used in many different contexts and areas. For example, after Tweedie's formula [Rob56], it was rediscovered a few years later by [Miy61]. In the early 2000s, the function  $\nabla \log p(\mathbf{x})$  was rediscovered again in the context of learning a general distribution and was named the “score function” by Aapo Hyvärinen [Hyv05]. But its connection to (empirical Bayesian) denoising was soon recognized by [Vin11]. Generalizations to other measurement distributions (beyond Gaussian noise) have been made by Eero Simoncelli's group [RS11], and later applied to image denoising [HJA20; KS21].

---

<sup>19</sup>Although in history there had been many attempts to develop analytical models for these data, such as random fields or stochastic processes for imagery data [MG99], as we have discussed in the previous section.

<sup>20</sup>Unlike the cases of PCA, Wiener filter, and Kalman filter.

<sup>21</sup>That is,  $p(\mathbf{x}) = \int_{-\infty}^{\infty} \varphi_\sigma(\mathbf{x} - \mathbf{z}) p_o(\mathbf{z}) d\mathbf{z}$ , where  $\varphi_\sigma$  is the density function of the Gaussian distribution  $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ .

<sup>22</sup>Herbert Robbins gave the credits of this formula to Maurice Kenneth Tweedie from their personal correspondence.

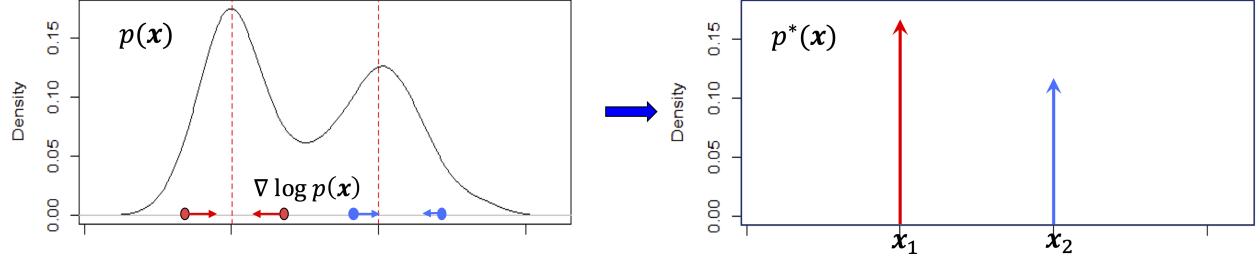


Figure 1.12: Geometric interpretation of a score function  $\nabla \log p(\mathbf{x})$  for a distribution with density  $p(\mathbf{x})$  on the left. The operation generated by the score function pushes the distribution towards areas of higher density. The goal is that, by a certain measure of compactness (e.g. entropy or coding length), the resulting distribution is more “compressed”. Eventually, the distribution converges onto one that is with a lower-dimensional support, as  $p^*(\mathbf{x})$  shown on the right.

**Entropy minimization.** In fact, this function has a very intuitive information-theoretic and geometric interpretation. Note that in information theory  $-\log p(\mathbf{x})$  generally corresponds to the number of bits needed to encode  $\mathbf{x}$ <sup>23</sup>. The gradient  $\nabla \log p(\mathbf{x})$  points to a direction in which the density is higher, as shown in Figure 1.12 left. The number of bits needed to encode  $\mathbf{x}$  decreases if it moves in that direction. Hence, the overall effect of the operator  $\nabla \log p(\mathbf{x})$  is to push the distribution to “shrink” towards areas of higher density. Actually, one can formally show that the (differential) entropy of the distribution

$$H(\mathbf{x}) = - \int p(\mathbf{w}) \log p(\mathbf{w}) d\mathbf{w} \quad (1.3.25)$$

indeed decreases under such an operation (see Chapter 3 and Appendix B). Therefore, if we encode it with an optimal code book, the overall coding length/rate of the resulting distribution is reduced, hence more “compressed.”. Intuitively, one can imagine that, if we repeat such a denoising process indefinitely, the distribution will eventually shrunk to one whose mass is concentrated on a support of lower dimension. For example, the distribution  $p(\mathbf{x})$  shown on the left of Figure 1.12, under the action of the score function  $\nabla \log p(\mathbf{x})$ , eventually will converge to the distribution  $p^*(\mathbf{x})$  on the right<sup>24</sup>:

$$H(\mathbf{x}) = - \int p(\mathbf{w}) \log p(\mathbf{w}) d\mathbf{w} \quad \xrightarrow{\text{decreasing}} \quad H^*(\mathbf{x}) = - \int p^*(\mathbf{w}) \log p^*(\mathbf{w}) d\mathbf{w}. \quad (1.3.26)$$

Strictly speaking, as the distribution converges to  $p^*(\mathbf{x})$ , its differential entropy converges to negative infinity. This is due to a technical difference between the definition of differential entropy for continuous random variable and discrete random variable, respectively. We will see how to resolve this technical difficulty in Chapter 3, using a more uniform measure of *rate distortion*.

We will discuss later in this chapter and in Chapter 3, how such a seemingly simple concept of denoising and compression leads to a very unifying and powerful method for learning general low-dimensional distributions in a high-dimensional space, including the distribution of natural images.

### 1.3.2 Empirical Approaches

In practice, for many important real-world data such as images, sounds, and texts, it is difficult to model them with idealistic linear or mixed linear models. For example, there has been a long and rich history in the fields of image processing and computer vision that tries to model the distribution of natural images analytically. David Mumford, a Fields Medalist, spent considerable effort in the 1990s trying to understand and model the statistics of natural images [Mum96]. He and his students, including Song-Chun Zhu, drew inspiration and techniques from statistical physics and proposed many statistical and stochastic models for the distribution

<sup>23</sup>at least in the case of a discrete variable, as we will explain in more details in Chapter 3.

<sup>24</sup>Strictly speaking,  $p^*(\mathbf{x})$  is a distribution whose density is a generalized function:  $p^*(\mathbf{x}) = p^*(\mathbf{x}_1)\delta(\mathbf{x}-\mathbf{x}_1) + p^*(\mathbf{x}_2)\delta(\mathbf{x}-\mathbf{x}_2)$ , with  $p^*(\mathbf{x}_1) + p^*(\mathbf{x}_2) = 1$ .

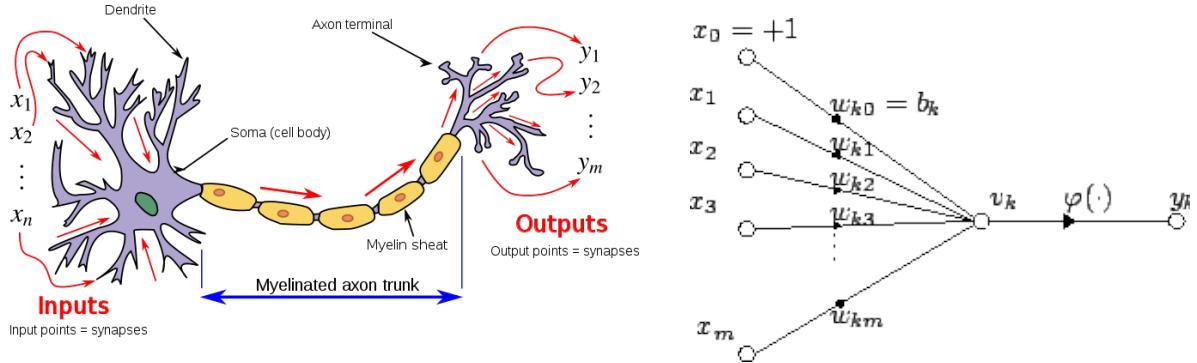


Figure 1.13: The first mathematical model of an artificial neuron (right) that emulates how a neuron (left) processes signals.

of natural images [HM99; LPM03; MG99; ZM97a; ZWM97; ZM97b]. However, these analytical models met with limited success in producing samples that closely resemble natural images. Obviously, for real-world data like images, we need to develop more powerful and unifying methods to pursue their more general low-dimensional structures.

Hence, historically, many empirical models have been proposed to model important real-world data, including images and texts. These models often drew inspiration from the characteristics of the biological nerve system because the brain of an animal or human seems to process these data extremely efficiently and effectively.

### Classic Artificial Neural Networks

**Artificial neuron.** Inspired by the nerve system in the brain, the mathematical model of the first artificial neuron<sup>25</sup> was proposed by Warren McCulloch<sup>26</sup> and Walter Pitts in 1943 [MP43]. It describes the relationship between the input  $x_i$  and output  $o_j$  as:

$$o_j = \varphi\left(\sum_i w_{ji} x_i\right), \quad (1.3.27)$$

where  $\varphi(\cdot)$  is some nonlinear activation, normally modeled by a threshold function. This model is illustrated in Figure 1.13. As we can see, this form already shares the main characteristics of a basic unit in modern deep neural networks. The model is derived from observations of how a single neuron works in our nerve system. However, people did not know exactly what functions a collection of such neurons wants to realize and perform. On a more technical level, neither were they sure what nonlinear activation function  $\varphi(\cdot)$  should be used. Hence, historically many variants have been proposed.<sup>27</sup>

**Artificial neural network.** In the 1950s, Frank Rosenblatt was the first to build a machine with a *network* of such artificial neurons, shown in Figure 1.15. The machine is called Mark I Perceptron which consists of an input layer, an output layer, and a single hidden layer consisting of 512 artificial neurons, as shown in Figure 1.15 left, which is similar to what is illustrated in Figure 1.14 left. It was designed to classify optical images of letters. However, the capacity of a single-layer network is limited and can only learn linearly separable patterns. In a 1969 book *Perceptrons: An Introduction to Computational Geometry* by Marvin Minsky and Seymour Papert [MP69], it was shown that the single-layer architecture of Mark I Perceptron cannot learn an XOR function. This result had significantly damped people's interest in artificial neural networks, even though it was later proven that a multi-layer network is able to learn an XOR function [RHW86a]. In fact, a (big enough) multi-layer network, as shown in Figure 1.14 right, consisting of such simple neurons

<sup>25</sup>known as the Linear Threshold Unit, or a perceptron.

<sup>26</sup>A professor of psychiatry at the University of Chicago at the time

<sup>27</sup>Step function, Hard or soft thresholding, Rectified Linear Unit (ReLU), sigmoid, etc.

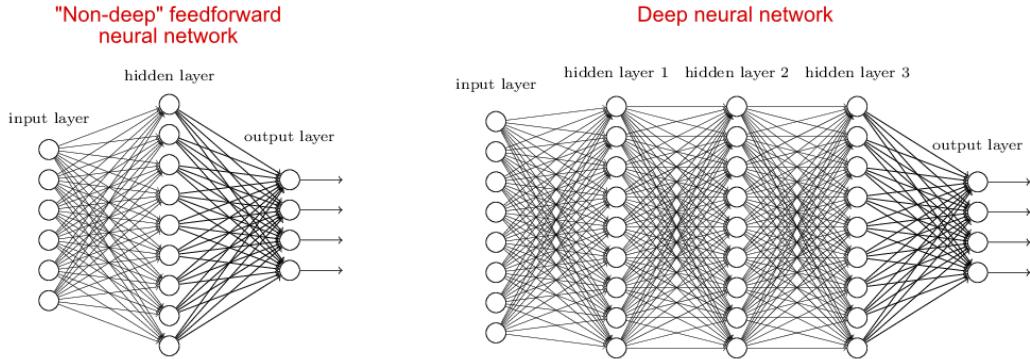


Figure 1.14: A network with one single hidden layer (left) versus a deep network (right).

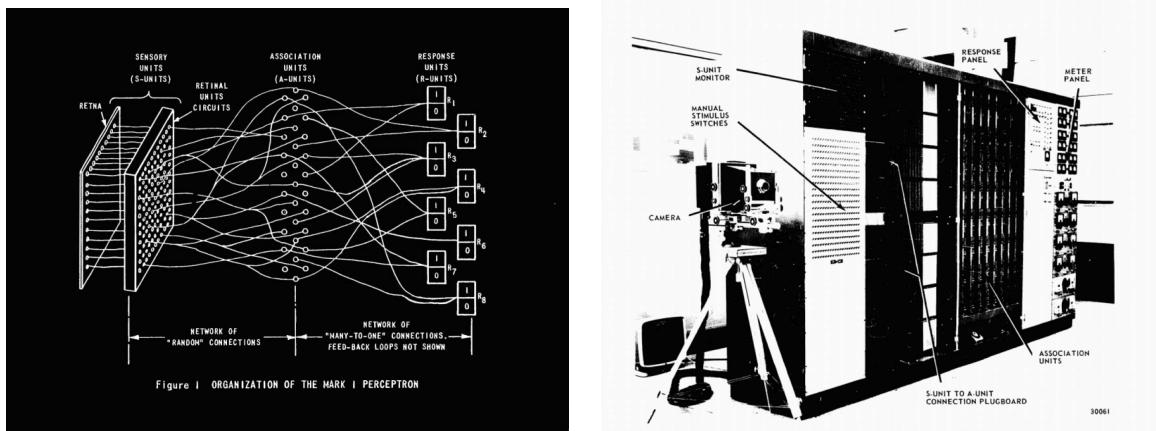


Figure 1.15: The Mark I Perceptron machine developed by Frank Rosenblatt in late 1950s.

can simulate any finite-state machine, even the universal Turing machine.<sup>28</sup> Nevertheless, subsequently, the study of artificial neural networks went into its first winter in the 1970s.

**Convolutional neural networks.** The somewhat disappointing early experimentation with artificial neural networks like Mark I Perceptron in the 50s and 60s suggested that it might not be enough to simply connect neurons in a general fashion as multi-layer perceptrons (MLP). In order to build effective and efficient networks, we need to understand what purpose or function neurons in a network need to achieve collectively so that they should be organized and learned in a certain special way. Once again, the study of machine intelligence turned to draw inspiration from how the animal's nerve system works.

It is known that most of our brain is dedicated to process visual information. In 1950s and 1960s, David Hubel and Torsten Wiesel systematically studied the visual cortices of cats. It was discovered that the visual cortex contains different types of cells (known as simple cells and complex cells), which are sensitive to visual stimuli of different orientations and locations [HW59]. Hubel and Wiesel won the 1981 Nobel Prize in Physiology or Medicine for their ground-breaking discovery.

On the artificial neural network side, Hubel and Wiesel's work had inspired Kunihiko Fukushima who designed the ‘neocognitron’ network in 1980 which consists of artificial neurons that emulate biological neurons in the visual cortices [Fuk80]. This is known as the first *convolutional neural network* (CNN), and its architecture is illustrated in Figure 1.16. Unlike the perceptron, the neocognitron had more than one hidden layer and could be viewed as a deep network, as compared in Figure 1.14 right.

<sup>28</sup>Do not confuse what neural networks are capable of doing in principle with whether it is tractable or easy to learn a neural network that realizes certain desired functions.

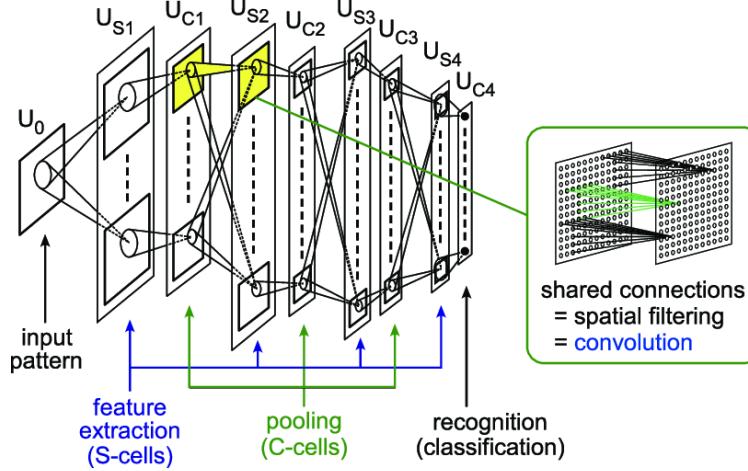


Figure 1.16: Origin of convolutional neural networks: the Neocognitron by Kunihiko Fukushima in 1980. Notice that the interleaving layers of convolutions and pooling try to emulate the functions of simple cells and complex cells discovered in the visual cortices of cats.

Also inspired by how neurons work in the cat's visual cortex, he was also the first to introduce the use of *rectified linear unit* (ReLU):

$$\varphi(x) = \max\{0, x\} = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0, \end{cases} \quad (1.3.28)$$

for the activation function  $\varphi(\cdot)$  in 1969 [Fuk69]. But not until recent years has the ReLU become widely used activation function in modern deep (convolutional) neural networks. We will learn from this book why this is a good choice once we explain the main operations that deep networks try to implement: compression.

CNN-type networks continued to evolve in the 1980s and many different variants had been introduced and studied. However, despite the remarkable capacities of deep networks and the improved architectures inspired by neuroscience, it remained extremely difficult to train such deep networks for a real task such as image classification. How to get a network to work depended on many unexplainable heuristics and tricks that really limited the appeal and applicability of neural networks. One major breakthrough came around 1989 when Yann LeCun successfully used *back propagation* (BP) to learn a deep convolutional neural network for recognizing hand-written digits [LBD+89], later known as the LeNet (see Figure 1.17). After several years' persistent development, his perseverance paid off: performance of the LeNet eventually became good enough for practical usage in late 1990s [LBB+98a]: it was used by the US Post Office for recognizing handwritten digits (for zip codes). The LeNet was considered as the “prototype” network for all modern deep neural networks, such as the AlexNet and ResNet, which we will discuss later. Due to this work, Yann LeCun was awarded the 2018 Turing Award.<sup>29</sup>

**Backpropagation.** In history, the fate of deep neural networks seems to be tied closely to how they can be trained easily and efficiently. Back propagation (BP) was introduced for this reason. We know that a multiple layer perceptron can be expressed as a composition of a sequence of linear mappings and nonlinear activations as follows:

$$h(\mathbf{W}_1, \dots, \mathbf{W}_L) = f^L(\mathbf{W}_L f^{L-1}(\mathbf{W}_{L-1} \cdots f^2(\mathbf{W}_2 f^1(\mathbf{W}_1 \mathbf{x}))). \quad (1.3.29)$$

In order to train the network weights  $\{\mathbf{W}_l\}_{l=1}^L$  to reduce the prediction/classification error based on a gradient descent algorithm, we need to evaluate the gradient  $\partial h / \partial \mathbf{W}_l$ . It has been well known, from the *chain rule* in calculus, that gradients can be computed efficiently for this type of functions, later known as back propagation (BP). See Appendix A for a detailed description. The technique of back propagation was already known and practiced by people in fields such as optimal control and dynamic programming

<sup>29</sup>Together with two other pioneers of deep networks, Yoshua Bengio and Geoffrey Hinton.

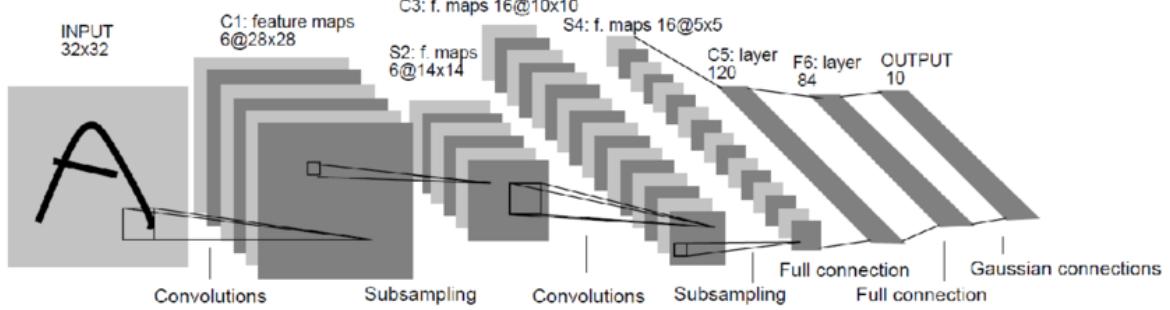


Figure 1.17: The LeNet-5 convolution neural network designed by Yann LeCun in 1989.

in the 1960s and 1970s. For example, it appeared in the 1974 PhD thesis of Dr. Paul Werbos [Wer74; Wer94]. In 1986, David Rumelhart et al. were the first to apply back propagation to train a multiple layer perceptron (MLP) network [RHW86a]. Since then, BP has become increasingly popular since it gives a *scalable* algorithm to learn large deep neural networks.<sup>30</sup> It is now an almost dominant technique for training deep neural networks today. However, it is believed that nature does not learn by back propagation, because such a mechanism is still way too expensive for physical implementation by nature<sup>31</sup>. This obviously leaves tremendous room for improvement in the future, as we will discuss more.

However, despite the above algorithmic progress and promising practice in the 1980s, training deep neural networks remained extremely finicky and expensive for computing systems in the 1980s and 1990s. In late 1990s, support vector machines (SVM) [CV95] had become very popular as they were viewed as a better alternative to neural networks for tasks such as classification.<sup>32</sup> There were two main reasons: first, SVM was based on a rigorous statistical learning framework known as the Vapnik–Chervonenkis (VC) theory; and second, it leads to rather efficient algorithms based on convex optimization [BV04]. The rise of SVM had brought a second winter to the study of neural networks around the early 2000s.

**Compressive autoencoding.** In the late 1980s and 1990s, artificial neural networks were already adopted to learn low-dimensional representations of high-dimensional data such as images. It had been shown that neural networks can be used to learn PCA from the data [BH89; Oja82], instead of using the classic methods discussed in Section 1.3.1. It was also argued during late 1980s that due to its capability to model nonlinear transforms, neural networks were suggested to learn low-dimensional representations for data with nonlinear distributions. Similar to the linear PCA case, one can try to simultaneously learn a nonlinear dimension-reduction encoder  $f$  and a decoder  $g$ , each modeled by a deep neural network [Kra91; RHW86a]:

$$\mathbf{X} \xrightarrow{f} \mathbf{Z} \xrightarrow{g} \hat{\mathbf{X}}. \quad (1.3.30)$$

By enforcing the decoded data  $\hat{\mathbf{X}}$  to be consistent with the original  $\mathbf{X}$ , say by minimizing an MMSE type reconstruction error<sup>33</sup>:

$$\min_{f,g} \|\mathbf{X} - \hat{\mathbf{X}}\|_2^2 = \|\mathbf{X} - g(f(\mathbf{X}))\|_2^2, \quad (1.3.31)$$

an autoencoder can be learned from the data  $\mathbf{X}$  themselves.

But how can we guarantee that such an autoencoding indeed captures the true low-dimensional structures in  $\mathbf{X}$  instead of giving a trivial redundant representation? For example, we can simply choose  $f$  and  $g$  to be the identity map and  $\mathbf{Z} = \mathbf{X}$ . So to ensure the autoencoding to be worthwhile, one wishes the resulting representation to be compressive, in terms of a certain computable measure of complexity. In 1993,

<sup>30</sup>as it can be efficiently implemented on computing platforms that facilitate parallel and distributed computing.

<sup>31</sup>As we have discussed earlier, nature almost ubiquitously learns to correct error via closed-loop feedback.

<sup>32</sup>In fact, similar ideas to solve classification problems can be traced back to the PhD dissertation work of Thomas Cover, which as condensed and published in a paper in 1964 [Cov64].

<sup>33</sup>Although MMSE type of errors are known to be problematic to imagery data that have complex nonlinear structures. As we will soon discuss, much of the recent work in generative methods, including GANs, has been to find surrogates of better distance functions between the original data  $\mathbf{X}$  and the regenerated  $\hat{\mathbf{X}}$ .

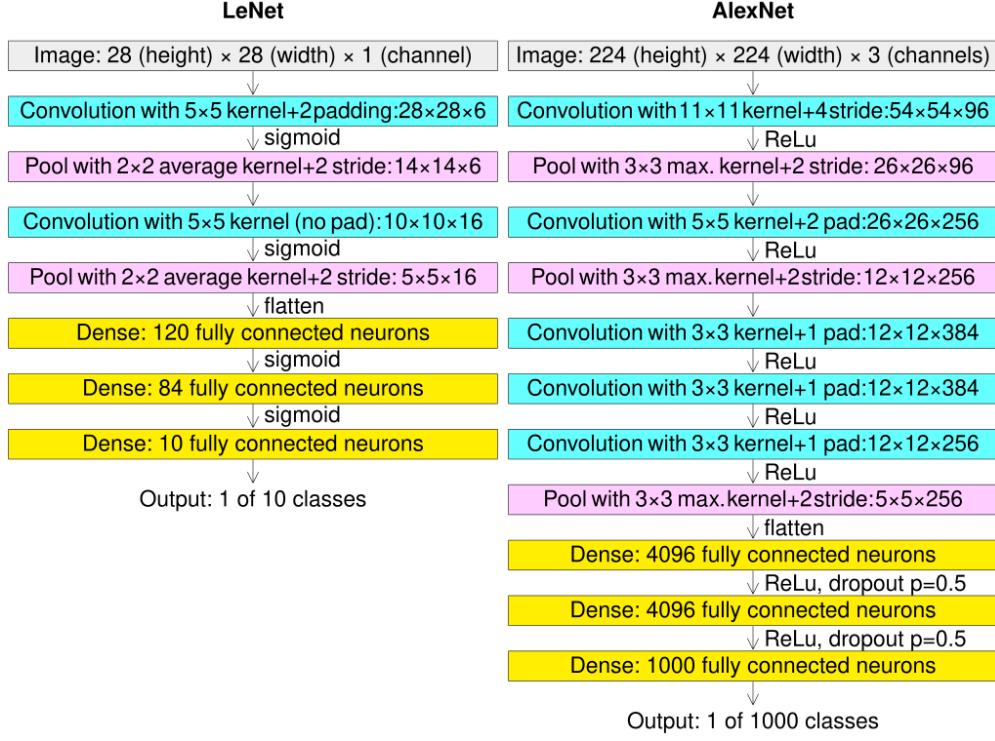


Figure 1.18: Architecture of the LeNet [LBD+89] versus that of the AlexNet [KSH12].

Geoffrey Hinton and colleagues proposed to use the coding length as such a measure, hence the objective of autoencoding became to find such a representation that minimizes the coding length [HZ93]. This work also established a fundamental connection between the principle of minimum description length [Ris78] and free (Helmholtz) energy minimization. Later work [HS06] from Hinton’s group showed empirically that such an autoencoding is capable of learning meaningful low-dimensional representations for real-world images. A more comprehensive survey of autoencoders was done by Pierre Baldi in 2011 [Bal11], just before deep networks became popular. We will discuss more about the measure of complexity and autoencoding later in Section 1.4, and give a systematic study of compressive autoencoding to Chapter 5, with a more unifying view.

### Modern Deep Neural Networks

For nearly 30 years, from 1980s to 2010s, for the study of machine learning and machine intelligence, neural networks were not considered seriously by the mainstream. Early (deep) neural networks, such as the LeNet, have shown promising performance for small-scale classification problems such as recognizing digits. However, the design and practice of the networks were rather empirical, datasets available at the time were small, and the BP algorithm was a huge computational burden for computers then. These factors had resulted in the lack of interest in neural networks, and progress had been stagnant, with only a handful of researchers working on it.

**Classification and recognition.** As it turns out, the tremendous potential of deep neural networks could only be unleashed once there are enough data and computing power. Fast forward to 2010s, much larger datasets such as ImageNet became available, and GPUs became powerful enough to make BP much more affordable, even for networks much larger than LeNet. Around 2012, a deep convolutional neural network known as the AlexNet drew attention as it surpassed extent classification methods by a significant margin

with the ImageNet dataset [KSH12].<sup>34</sup> Figure 1.18 shows the comparison between the AlexNet and the LeNet. The AlexNet shares many common characteristics as the LeNet, only it is bigger and has adopted ReLU as the nonlinear activation instead of the Sigmoid function used in LeNet. Partly due to the influence of this work, Geoffrey Hinton was awarded the 2018 Turing Award.

This early success inspired the machine intelligence community, in the next few years, to explore new variations and improvements to the network design. In particular, people had discovered empirically that the larger and deeper the networks, the better the performance in tasks such as image classification. Many deep network architectures have been tried, tested, and popularized. A few notable ones include VGG [SZ15], GoogLeNet [SLJ+14], ResNet [HZR+16a], and more recently Transformers [VSP+17] etc. Despite fast progress in empirical performance, there was a lack of theoretical explanation for these empirically discovered architectures, including relationships among them if any. One purpose of this book is to reveal what common objective all these networks may serve and why they share certain common characteristics, including multiple layers of linear operator interleaved with nonlinear activation (see Chapter 4).

**Reinforcement learning.** The early successes of deep networks were mainly for classification tasks in a supervised learning setting, such as speech recognition and image recognition. Deep networks were later adopted by the DeepMind team, led by Demis Hassabis, to learn decision-making or control policies for playing games. In this context, deep networks are used to model the optimal decision/control policy or the associated optimal value function, as shown in Figure 1.19. These network parameters are incrementally optimized<sup>35</sup> based on reward returned from the success or failure of playing the game with the current policy. This learning method is generally referred to as *reinforcement learning* [SB18], originated from the practice of control systems in the late 1960s [MM70; WF65]. Its earlier roots can be traced back to a much longer and richer history of *dynamic programming* by Richard Bellman [Bel57] and *trial-and-error learning* by Marvin Minsky [Min54] in the 1950s.

From an implementation perspective, the combination of deep networks and reinforcement learning turned out to be rather powerful: deep networks can be used to approximate control policy and value function for real-world environments that are difficult to model analytically. This practice had eventually led to the AlphaGo system, developed by the company DeepMind, which surprised the world in 2016 by beating a top human player Lee Sedol in the game Go and then the world champion Jie Ke in 2017.<sup>36</sup>

The success of AlphaGo came as a big surprise to the computing society which generally believes that the state space for search is too prohibitively large to admit any efficient solution, in terms of computation and sample size. The only reasonable explanation for its success is that there must be very good structures in the optimal value/policy function of the game Go. Their intrinsic dimension is not so high and they can be approximated well by a neural network, learnable from not so prohibitively many samples.

**Generation and prediction.** One may view early practices of deep networks in the 2010s focused more on extracting relevant information from the data  $\mathbf{X}$  and encoding it for certain task-specific representation  $\mathbf{Z}$  (say  $\mathbf{Z}$  represent class labels in classification tasks):

$$\mathbf{X} \xrightarrow{f} \mathbf{Z}. \quad (1.3.32)$$

In this setting, the mapping  $f$  to be learned does not need to preserve most distributional information about  $\mathbf{X}$  but only the sufficient statistics needed for a specific task. For example, a sample  $\mathbf{x}$  in  $\mathbf{X}$  could be an image of an apple, which is mapped by  $f$  to its class label  $\mathbf{z} = \text{"apple."}$  The *information bottleneck framework* [TZ15] was proposed in 2015 to analyze the role of deep networks in such a context.

However, in many modern situations such as those so-called large foundation models, people often need to decode  $\mathbf{Z}$  to recover the corresponding  $\mathbf{X}$  to a certain degree of precision:

$$\mathbf{Z} \xrightarrow{g} \hat{\mathbf{X}}. \quad (1.3.33)$$

<sup>34</sup>In fact, before this, deep networks had demonstrated state of the art performance on speech recognition tasks. But it did not receive so much attention till their success with image classification.

<sup>35</sup>Say based on back propagation (BP).

<sup>36</sup>In 1996, IBM's Deep Blue system made history by defeating Russian grandmaster Garry Kasparov in chess. It mainly used traditional machine learning techniques, such as tree search and pruning, which were not so scalable and had not been proven successful for more challenging games such as Go for over 20 years.

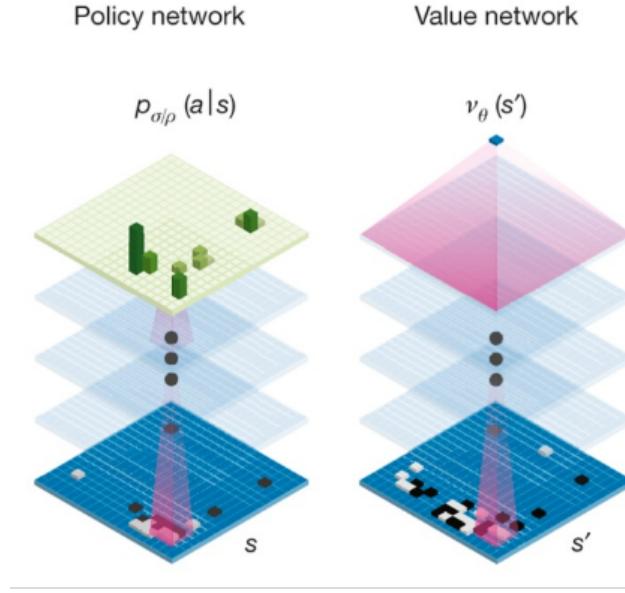


Figure 1.19: AlphaGo: Using deep neural networks to model the optimal policy or the optimal value function for the game Go.

As  $\mathbf{X}$  typically represents data observed from the external world, a good decoder would allow us to simulate or predict what happens in the world. For example, in a “text to image” or “text to video” task,  $\mathbf{z}$  normally represents the texts that describe the content of a desired image  $\mathbf{x}$ . The decoder should be able to generate an  $\hat{\mathbf{x}}$  that has the same content as  $\mathbf{x}$ . For example, given an object class  $\mathbf{z}$  = “apple”, the decoder  $g$  should generate an image  $\hat{\mathbf{x}}$  that looks like an apple, although not necessarily exactly the same as the original  $\mathbf{x}$ .

**Generation via discriminative approaches.** In order for the generated images  $\hat{\mathbf{X}}$  to be similar to the true natural images  $\mathbf{X}$ , we need to be able to evaluate and minimize some distance:

$$\min d(\mathbf{X}, \hat{\mathbf{X}}). \quad (1.3.34)$$

As it turns out, most theoretically motivated distances are extremely difficult, if not impossible, to compute and optimize for distributions in high-dimensional space but with a low intrinsic dimension.<sup>37</sup>

In 2007, Zhuowen Tu, a former student of Song-Chun Zhu, probably disappointed by early analytical attempts to model and generate natural images (discussed earlier), decided to try a drastically different approach. In a paper published in CVPR 2007 [Tu07], he was the first to suggest that one could learn a generative model for images via a discriminative approach. The idea is simple: if it is difficult to evaluate the distance  $d(\mathbf{X}, \hat{\mathbf{X}})$ , one could try to learn a discriminator  $d$  to separate  $\hat{\mathbf{X}}$  from  $\mathbf{X}$ :

$$\mathbf{Z} \xrightarrow{g} \hat{\mathbf{X}}, \mathbf{X} \xrightarrow{d} \mathbf{0}, \mathbf{1}, \quad (1.3.35)$$

where  $\mathbf{0}, \mathbf{1}$  indicate an image is generated or not. Intuitively, the harder we could separate  $\hat{\mathbf{X}}$  and  $\mathbf{X}$ , probably the closer they are.

Tu’s work [Tu07] was the first to demonstrate the feasibility of learning a generative model from a discriminative approach. However, the work adopted traditional methods to generate images and classify distributions (such as boosting), and they were slow and hard to implement. After 2012, deep neural networks became very popular for image classification. In 2014, Ian Goodfellow and colleagues proposed again to generate natural images with a discriminative approach [GPM+14a]. They suggested using deep

<sup>37</sup>This is the case even if a parametric family of distribution for  $\mathbf{X}$  is given. The distance often becomes ill-conditioned or ill-defined for distributions with low-dimensional supports. What could be even worse is that the chosen family might not be able to approximate well the true distribution of interest.

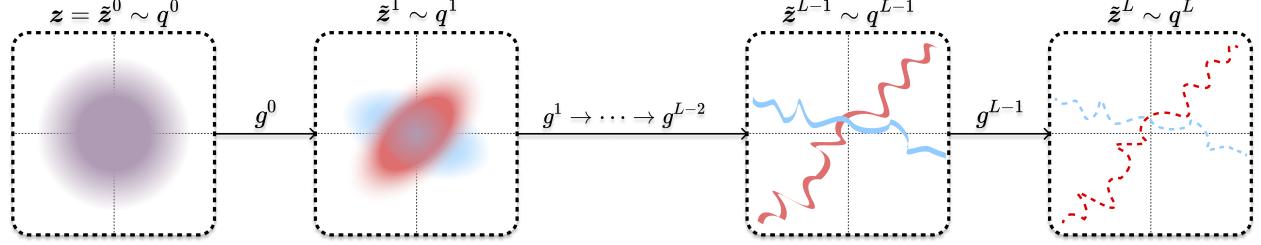


Figure 1.20: Illustration of an iterative denoising and compressing process that, starting from a Gaussian distribution  $q^0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ , converges to an arbitrary low-dimensional data distribution  $q^L = p(\mathbf{x})$ .

neural networks to model the generator  $g$  and the discriminator  $d$  instead. Moreover, they proposed to learn the generator  $g$  and discriminator  $d$  via a minimax game:

$$\min_g \max_d \ell(\mathbf{X}, \hat{\mathbf{X}}), \quad (1.3.36)$$

where  $\ell(\cdot)$  is some natural loss function associated with the classification. In words the discriminator  $d$  tries to maximize its success in separating  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ , whereas the generator  $g$  tries to do the opposite. For this reason, this approach is named as *generative adversarial networks* (GANs). It was shown that once trained on a large dataset, GANs can indeed generate photo-realistic images. Partially due to the influence of this work, Yoshua Bengio was awarded the 2018 Turing Award.

The discriminative approach seems to be a rather clever way to bypass a fundamental difficulty in distribution learning. However, rigorously speaking, this approach does not fully resolve this fundamental difficulty at all. It is shown by [GPM+14a] that with a properly chosen loss, the minimax formulation is mathematically equivalent to minimize the *Jensen-Shannon distance* (see [CT91]) between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ . This is known to be a hard problem for two low-dimensional distributions in a high-dimensional space. As a result, in practice, GANs typically rely on many heuristics and engineering tricks and often suffer from instability issues such as *mode collapsing*.<sup>38</sup> Overall, there has been a lack of theoretical guidance on how to improve the practice of GANs.

**Generation via denoising and diffusion.** In 2015, shortly after GAN was introduced and became popular, Surya Ganguli and his students realized and suggested that an iterative denoising process modeled by a deep network can be used to learn a general distribution, such as that of natural images [SWM+15]. Their method was inspired by properties of the special Gaussian and binomial processes, studied by William Feller back in 1949 [Fel49].<sup>39</sup>

Soon, denoising operators based on the score function [Hyv05], briefly introduced in Section 1.3.1, were shown to be more general and unified the denoising and diffusion processes and algorithms [HJA20; SE19; SSK+21]. Figure 1.20 gives an illustration of the process that transforms a generic Gaussian distribution  $q^0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$  to an (arbitrary) empirical distribution  $p(\mathbf{x})$  by performing a sequence of iterative denoising (or compressing) operations:

$$z^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \xrightarrow{g^0} z^1 \xrightarrow{g^1} \dots \xrightarrow{g^{L-1}} z^L \sim p(\mathbf{x}). \quad (1.3.37)$$

By now, denoising (and diffusion) have replaced GANs and become a mainstream method for learning distributions of images and videos, leading to popular commercial image generating engines such as Midjourney and Stability.ai. In Chapter 3 we will systematically introduce and study the denoising and diffusion method for learning a general low-dimensional distribution.

<sup>38</sup>Nevertheless, such a minimax formulation provides a practical approximation of the distance. It simplifies the implementation and avoids certain caveats in directly computing the distance.

<sup>39</sup>Again, in the magical era of 1940s!

## 1.4 A Unifying Approach

So far, we have given a brief account of the main objective and history of machine intelligence and many important ideas and approaches associated with it. In recent years, after the empirical success of deep neural networks, tremendous efforts have been made to develop theoretical frameworks that can help us understand all the empirically designed deep neural networks, either certain seemingly necessary components (e.g., dropout, normalization, attention, etc.) or their overall behaviors (e.g., double descent, neural collapse, etc.).

Partly motivated by this, this book aims to achieve several important and challenging goals:

- Develop a theoretical framework that would allow us to derive rigorous mathematical interpretation of deep neural networks.
- Ensure correctness of the learned data distribution and consistency with the learned representation.
- Demonstrate that the framework can lead to performant architectures and can guide further improvements in practice.

Within the past few years, there is mounting evidence that these goals can indeed be achieved, by leveraging the theory and solutions to the classical analytical low-dimensional models discussed briefly earlier (more thoroughly in Chapter 2) and by integrating fundamental ideas from a few related fields, namely information/coding theory, control/game theory, and optimization. This book aims to give a systematic introduction to this new approach.

### 1.4.1 Learning Parsimonious Representations

One necessary condition for any learning task to be possible is that the sequences of interest must be *computable*, at least in the sense of Alan Turing [Tur36]. That is, a sequence can be computed via a program on a typical computer.<sup>40</sup> In addition to being computable, we require computation be *tractable*.<sup>41</sup> That is, the computational cost (space and time) for learning and computing the sequence should not grow exponentially in length. Furthermore, as we see in nature (and in modern practice of machine intelligence), for most practical tasks, an intelligent system needs to learn what is predictable from massive data in a very high-dimensional space, such as from vision, sound, and touch. Hence, for intelligence, we do not need to consider all computable and tractable sequences or structures. We should focus only on predictable sequences and structures which admit *scalable* realizations of its learning and computing algorithms:

$$\text{computable} \implies \text{tractable} \implies \text{scalable}. \quad (1.4.1)$$

This is because whatever algorithms intelligent beings use to learn useful information must be *scalable*. More specifically, the computational complexity of the algorithms would better scale gracefully, typically linear or even sublinear, in the size and dimension of the data. On the technical level, this requires that the operations that the algorithms rely on to learn could only utilize oracle information that can be efficiently computed from the data. More specifically, when the dimension is high and the scale is large, the only oracle one could afford to compute is either the first-order geometric information about the data<sup>42</sup> or the second-order statistic information<sup>43</sup>. The main goal of this book is to develop a theoretical and computational framework within which we can systematically develop efficient and effective solutions or algorithms with such scalable oracles and operations to *learn* low-dimensional structures from the sampled data and subsequently the predictive function.

---

<sup>40</sup>There are indeed well-defined sequences that are not computable.

<sup>41</sup>We do not need to consider predicting things whose computational complexity is intractable, say grows exponentially in the length or dimension of the sequence.

<sup>42</sup>such as approximating a nonlinear structure locally with linear subspaces and computing the gradient of an objective function.

<sup>43</sup>such as covariance or correlation of the data or their features.

**Pursuing low-dimensionality via compression.** From the examples of sequences we gave in Section 1.2.1, it is clear that some sequences are easy to model and compute and others are more difficult. Obviously, the computational cost of a sequence depends on how complex the predicting function  $f$  is. The higher the degree of regression  $d$ , the more costly it is to compute.  $f$  can be a simple linear function, and it can also be a nonlinear function that can be arbitrarily difficult to specify and compute.

It is reasonable to believe that if a sequence is harder, by whatever measure we may choose, to specify and compute, then it will also be more difficult to learn from its sampled segments. Nevertheless, for any given predictable sequence, there are in fact many, often infinitely many, ways to specify it. For example, for a simple sequence  $x_{n+1} = ax_n$ , we could also define the same sequence with  $x_{n+1} = ax_n + bx_{n-1} - bx_{n-1}$ . Hence it would be very useful if we could develop an objective and rigorous notion of “complexity” for any given computable sequence.

Andrey Kolmogorov, a Russian mathematician, was one of the first to give a definition of complexity for any computable sequence.<sup>44</sup> He suggested that among all programs that can compute the same sequence, we may use the length of the shortest program as a measure for its complexity. This idea is very much in line with the famous “Occam’s Razor” principle of parsimony: *one always chooses the simplest among all theories that can explain the same observation*. To be more precise, let  $p$  represent a compute program that can generate a sequence  $S$  on a universal computer  $\mathcal{U}$ . The Kolmogorov complexity of the sequence  $S$  is defined to be:

$$K(S) = \min_{p : \mathcal{U}(p)=S} L(p). \quad (1.4.2)$$

Therefore, the complexity of a sequence is measured by how “parsimoniously” we can specify or compute it. This definition of complexity, or parsimony, for sequences is of great conceptual importance and has many interesting properties. Historically, it has inspired many profound studies in the theory of computation, especially in the field of algorithmic information theory.

The length of the shortest program can be viewed as the ultimate compression of the sequence considered, providing a quantitative measure of how much we have gained by having learned the correct generative mechanism of the sequence. However, despite its theoretical importance, the Kolmogorov complexity is in general not a computable function [CT91] (even intractable to approximate accurately). As a result, this measure of complexity is of little practical use. Neither can it tell us in advance how difficult it is to learn a given sequence nor can it tell us how well we have learned.

**Computable measure of parsimony.** Hence for practical purposes, we need an efficiently computable measure of complexity for sequences that are generated from the same predicting function.<sup>45</sup> Note that part of the reason why Kolmogorov complexity is not computable is because its definition is non-constructive.

So to introduce a computable measure of complexity, we may take a more constructive approach, as advocated by Claude Shannon through the framework of information theory [CT91; Sha48].<sup>46</sup> In essence, by assuming the sequence  $S$  is drawn from a probabilistic distribution  $p(S)$ , the so-called *entropy* of the distribution:<sup>47</sup>

$$h(S) \doteq - \int p(s) \log p(s) ds \quad (1.4.3)$$

gives a natural measure of its complexity. This measure also has a natural interpretation of the average number of binary bits needed to encode such a sequence, as we will see in Chapter 3.

To illustrate the main ideas of this view, let us take a large number of long sequence segments:

$$\{S_1, S_2, \dots, S_i, \dots, S_N\} \subset \mathbb{R}^D, \quad (1.4.4)$$

generated by a predicting function  $f$ . Note that without loss of generality, here we have assumed that all sequences are of the same length  $D$ . Therefore, each sequence can be viewed as a vector in  $\mathbb{R}^D$ . Secondly, we

<sup>44</sup>Many have contributed to this notion of sequence complexity, most notably including Ray Solomonoff and Greg Chaitin. All three are believed to have developed and studied algorithmic information theory independently, Ray Solomonoff in 1960, Andrey Kolmogorov in 1965 [Kol98] and Gregory Chaitin around 1966 [Cha66].

<sup>45</sup>Note that in practice, we typically care about learning the predicting function  $f$ , instead of any particular sequence generated by  $f$ .

<sup>46</sup>which has successfully guided the engineering practice of the communication industry for the past over 80 years.

<sup>47</sup>Here we consider differential entropy as we assume the sequence consists of continuous variables. If it consists of discrete variables, we could consider the entropy:  $H(S) = - \sum_i p(s_i) \log p(s_i)$ .

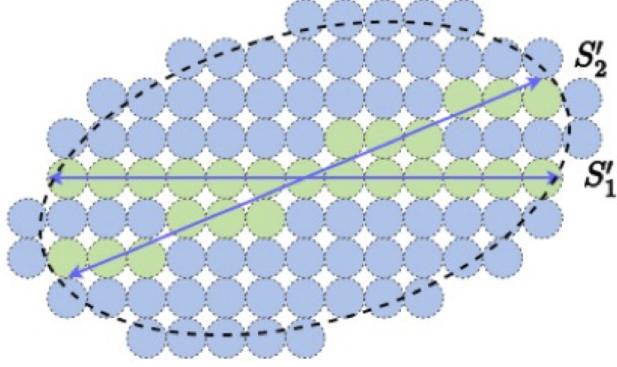


Figure 1.21: Comparison between two coding schemes. Image the true distribution of the data is around the two arrowed lines. One can code the samples from the two lines with the a code book consisting of all blue balls; one can also code the samples with a code book consisting of only the green balls. Obviously, the second coding schemes will result in much lower coding length/rate, subject to the same precision.

may introduce a coding scheme (with a code book), denoted as  $\mathcal{E}$ , that converts every segment  $S_i$  to a unique stream of binary bits  $\mathcal{E}(S_i)$ . The simplest coding scheme is to fill the space spanned by all segments with  $\epsilon$ -balls, as shown in Figure 1.21. We then number all the balls. Each sequence is encoded as the (binary) number of its closest ball. Hence, each segment can be recovered<sup>48</sup> from its corresponding bit stream.

Then the complexity of the predicting function  $f$  can be evaluated as the average coding length of all sequences, known as the coding rate:<sup>49</sup>

$$R(f | \mathcal{E}) = \mathbb{E}[L(\mathcal{E}(S))] \approx \frac{1}{N} \sum_{i=1}^N L(\mathcal{E}(S_i)). \quad (1.4.5)$$

Obviously, the coding rate measure will change if one uses a different coding scheme (or a code book). In practice, the better we know the low-dimensional structure around which the segments are distributed, the more efficient a code book we can design, as the example shown in Figure 1.21. Acute readers may have recognized that conceptually the denoising process illustrated in Figure 1.20 resembles closely going from the coding scheme with the blue balls to that with the green ones.

Given two coding schemes  $\mathcal{E}_1$  and  $\mathcal{E}_2$  for the segments, if the difference in the coding rates is positive:

$$R(f | \mathcal{E}_1) - R(f | \mathcal{E}_2) > 0, \quad (1.4.6)$$

we may say the coding scheme  $\mathcal{E}_2$  is better. This difference can be viewed as a measure of how much more information  $\mathcal{E}_2$  has over  $\mathcal{E}_1$  about the distribution of the data. To a large extent, the goal of learning  $f$  is equivalent to finding the most efficient encoding scheme that minimizes the coding rate:

$$\min_{\mathcal{E}} R(f | \mathcal{E}). \quad (1.4.7)$$

As we will see in Chapter 3, the achievable minimal rate is closely related to the notion of entropy  $H(S)$  (1.4.3).

*Remark 1.1.* The perspective of measuring data complexity with explicit encoding schemes has motivated several learning objectives that were proposed to revise the Kolmogorov complexity for better computability [WD99], including the minimum message length (MML) proposed later in 1968 [WB68] and the minimum description length (MDL) in 1978 [HY01; Ris78]. These objectives normally count the coding length for the coding scheme  $\mathcal{E}$  itself (including its code book) in addition to the data  $S$  of interest:  $L(\mathcal{E}(S)) + L(\mathcal{E})$ .

<sup>48</sup>up to precision  $\epsilon$  as such an encoding scheme is lossy.

<sup>49</sup>One may make this more precise by taking  $R(f | \mathcal{E})$  to be the expected coding length for all segments of length  $D$ .

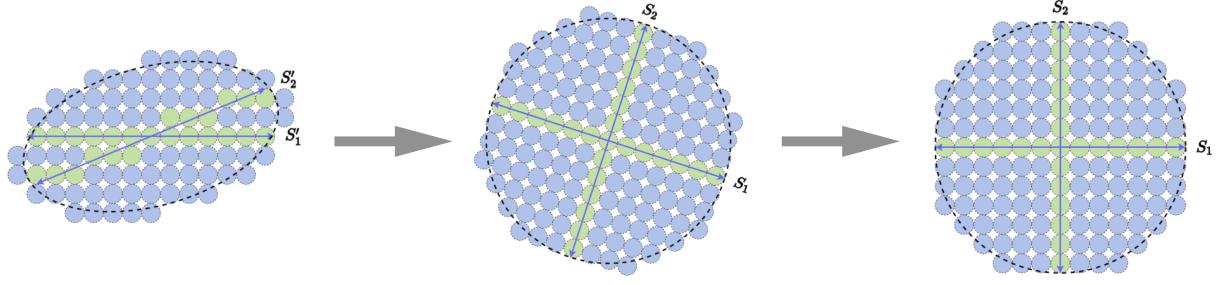


Figure 1.22: Transform the identified low-dimensional data distribution to a more informative and structured representation.

However, if the goal is to learn a finite-sized code book and apply it to a large number of sequences, the amortized cost of the code book can be ignored since

$$\frac{1}{N} \left( L(\mathcal{E}) + \sum_{i=1}^N L(\mathcal{E}(S_i)) \right) \approx \frac{1}{N} \sum_{i=1}^N L(\mathcal{E}(S_i))$$

as  $N$  becomes large.

Again, one may view the resulting optimal coding scheme as the one that achieves the best compression of the observed data. In general, compared to the Kolmogorov complexity, the coding length given by any encoding scheme will always be larger:

$$K(S) < L(\mathcal{E}(S)). \quad (1.4.8)$$

Therefore, minimizing the coding rate/length is essentially to minimize an upper bound of the otherwise uncomputable Kolmogorov complexity.

#### 1.4.2 Learning Informative Representations

Note that if the goal was simply to compress the given data just for the sake of compression, then in theory the optimal codes that approach the Kolmogorov complexity would become nearly random or structureless [Cha66].<sup>50</sup> However, our true purpose of learning the predictive function  $f$  is to use it repeatedly with ease in future predictions. Hence, while compression allows us to identify the low-dimensional distribution in the data, we would like to encode the distribution in a *structured and organized* way so that the resulting representation is very informative and efficient to use.<sup>51</sup> Figure 1.22 shows an example that explains intuitively why such a transformation is desired.

As we will show in Chapter 3, these desired structures in the final representation can be precisely promoted by choosing a natural measure of *information gain* based on the coding rates of the chosen coding schemes. As we see throughout this book, such an explicit and constructive coding approach provides a powerful computational framework for learning good representations of low-dimensional structures for real-world data, as in many cases of practical importance, the coding length function can be efficiently computed or approximated accurately. In some benign cases, we can even obtain closed-form formulae, e.g., subspace and Gaussian (see Chapter 3).

In addition, such a computational framework leads to a principled approach that naturally reveals the role that deep networks play in this learning process. As we will derive systematically in Chapter 4, the layers of a deep network are trying to perform operations that optimize the objective function of interest in an incremental manner. From this perspective, the role of deep networks can be precisely interpreted as to emulate a certain iterative optimization algorithm, say gradient descent, to optimize the objective of information gain. Layers of the resulting deep architectures can be endowed with precise statistical and geometric interpretation, namely performing incremental compressive encoding and decoding operations. As a result, the derived deep networks become transparent “white boxes” that are mathematically fully explainable.

<sup>50</sup>Because any codes with structures can be further compressed.

<sup>51</sup>For example, to sample the distribution under different conditions.

### 1.4.3 Learning Consistent Representations

To summarize our discussions so far, let us denote the data as:

$$\mathbf{X} = \{S_1, S_2, \dots, S_i, \dots, S_N\} \subset \mathbb{R}^D, \quad (1.4.9)$$

and let  $\mathbf{Z} = \mathcal{E}(\mathbf{X})$  be the codes of  $\mathbf{X}$  via some encoder  $\mathcal{E}$ :

$$\mathbf{X} \xrightarrow{\mathcal{E}} \mathbf{Z}. \quad (1.4.10)$$

In the machine learning context,  $\mathbf{Z}$  is often called “features” or “latent representation”. Note that without knowing the underlying distribution of  $\mathbf{X}$ , we do not know which encoder  $\mathcal{E}$  should be that can retain most useful information about the distribution of  $\mathbf{X}$ . In practice, people often start with trying a certain compact encoding scheme that serves well for a specific task. In particular, they would try to learn an encoder that optimizes certain (empirical) measure of parsimony for the learned representation:

$$\min \rho(\mathbf{Z}). \quad (1.4.11)$$

*Example 1.2.* For example, image classification is such a case: we assign all images in the same class to a single code and images in different classes to different codes, say “one-hot” vectors:

$$\mathbf{x} \mapsto \mathbf{z} \in \{[1, 0, 0, \dots, 0, 0], [0, 1, 0, \dots, 0, 0], \dots, [0, 0, 0, \dots, 0, 1]\}. \quad (1.4.12)$$

Now, a classifier  $f(\cdot)$  can be modeled as a function that predicts the probability of a given  $\mathbf{x}$  belonging to each of the  $k$  classes:  $\hat{\mathbf{z}} = f(\mathbf{x}) \in \mathbb{R}^K$ . Then the “goodness” of a classifier can be measured by the so-called *cross entropy*:<sup>52</sup>

$$L(\hat{\mathbf{z}}, \mathbf{z}) = \sum_{k=1}^K -z_k \log \hat{z}_k, \quad (1.4.13)$$

where  $z_k$  indicates the  $k$ -th entry of the vector  $\mathbf{z}$ . As the early practice of deep networks indicated [KSH12], if enough data are given, such an encoding scheme can often be represented by a deep network and learned in an end-to-end fashion by optimizing the cross-entropy. ■

The cross-entropy loss  $L(\hat{\mathbf{z}}, \mathbf{z})$  can be viewed as a special measure of parsimony  $\rho(\mathbf{z})$  associated with a particular family of encoding schemes that are suitable for classification. However, such an encoding is obviously *very lossy*. The learned  $\mathbf{z}$  does not contain any other information about  $\mathbf{x}$  except for its class type. For example, by assigning an image with (a code representing) the class label “apple”, we no longer know which specific type apple is in the original image from the label alone.

Of course, the other extreme is to require the coding scheme to be *lossless*. That is, there is a one-to-one mapping between  $\mathbf{x}$  and its code  $\mathbf{z}$ . However, as we will see in Chapter 3, lossless coding (or compression) is impractical unless  $\mathbf{x}$  is discrete. For a continuous random variable, we may only consider lossy coding schemes so that the coding length for the data can be finite. That is, we only encode the data up to a certain prescribed precision. As we will elaborate more in Chapter 3, lossy coding is not merely a practical choice, it plays a fundamental role in making learning the underlying continuous distribution possible from finite samples of the distribution.

For many purposes of learning, we want the feature  $\mathbf{z}$ , although *lossy*, to keep more information about  $\mathbf{x}$  than just its class type. In this book, we will introduce a more general measure of parsimony based on coding length/rate associated with a more general family of coding schemes – coding with a mixture of subspaces or Gaussians. This family has the capability to closely approximate arbitrary real-world distributions up to certain precision. As we will see in Chapter 3 and Chapter 4, such a measure will not only preserve most information about the distribution of  $\mathbf{X}$  but will also promote certain nice desired geometric and statistical structures for the learned representation  $\mathbf{Z}$ .

---

<sup>52</sup>The cross entropy can be viewed as a distance measure between the ground truth distribution of  $\mathbf{z}$  and that of the prediction  $\hat{\mathbf{z}}$ . It can also be viewed as the expected coding length of  $\mathbf{z}$  if we use the optimal code book for  $\hat{\mathbf{z}}$  to encode  $\mathbf{z}$ . The cross entropy reaches minimum when  $\mathbf{z}$  and  $\hat{\mathbf{z}}$  have the same distribution.

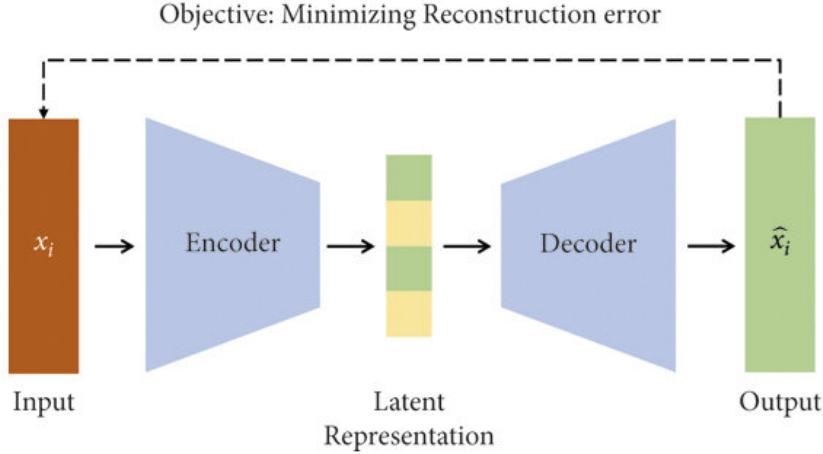


Figure 1.23: Illustration of the architecture of an autoencoder.

**Bidirectional Autoencoding for Consistency.** In a broader learning context, the main goal of a compressive coding scheme  $\mathcal{E}$  is to identify the low-dimensional structures in the data  $\mathbf{X}$  so that they can be used to predict things in the original data space. This requires that the learned encoding scheme  $\mathcal{E}$  allows an efficient decoding scheme, denoted as  $\mathcal{D}$ . It maps  $\mathbf{Z}$ , often known as a latent representation, back to the data space:

$$\mathbf{X} \xrightarrow{\mathcal{E}} \mathbf{Z} \xrightarrow{\mathcal{D}} \hat{\mathbf{X}}. \quad (1.4.14)$$

In general, we call such an encoding and decoding pair  $(\mathcal{E}, \mathcal{D})$  an *autoencoding*. Figure 1.23 illustrates the process of such an autoencoder.

Generally, we would prefer that the decoding is approximately an “inverse” to the encoding such that the data (distribution)  $\hat{\mathbf{X}}$  decoded from  $\mathbf{Z}$  would be similar to the original data (distribution)  $\mathbf{X}$  to some extent.<sup>53</sup> If so, we would be able to recover or predict from  $\mathbf{Z}$  what is going on in the original data space. In this case, we say the pair  $(\mathcal{E}, \mathcal{D})$  gives a *consistent* autoencoding. For most practical purposes, not only do we need such a decoding to exist, but also can be efficiently realized and physically implemented. For example, if  $x$  is a real-valued variable, quantification is needed in order for any decoding scheme to be realizable on a finite-state machine, as we will explain more in Chapter 3. Hence, in general, one should expect that realizable encoding and decoding schemes are necessarily lossy. A central question is how to learn a compact (lossy) representation  $\mathbf{Z}$  such that it can be used to predict  $\mathbf{X}$  well.

Generally speaking, as we will see, both encoder and decoder could be modeled and realized by deep networks and learned by solving an optimization problem of the following form:

$$\min d(\mathbf{X}, \hat{\mathbf{X}}) + \rho(\mathbf{Z}), \quad (1.4.15)$$

where  $d(\cdot, \cdot)$  is a certain distance function that promotes similarity between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ <sup>54</sup> and  $\rho(\mathbf{Z})$  is some measure that promotes parsimony and information richness of  $\mathbf{Z}$ . The classic principal component analysis (PCA) [Jol02] is a typical example of a consistent autoencoding, which we will study in great detail in Chapter 2, as a precursor to more general low-dimensional structures. In Chapter 5, we will study how to learn consistent autoencoding for general (say nonlinear) low-dimensional distributions.

#### 1.4.4 Learning Self-Consistent Representations

Note that in the above autoencoding objective, one needs to evaluate how close or consistent the decoded data  $\hat{\mathbf{X}}$  is to the original  $\mathbf{X}$ . This often requires some external supervision or knowledge on what similarity

<sup>53</sup>We will make it more precise what we mean by being similar later.

<sup>54</sup>Either sample-wise or distribution-wise similar, depending on the choice of the distance function  $d$ .

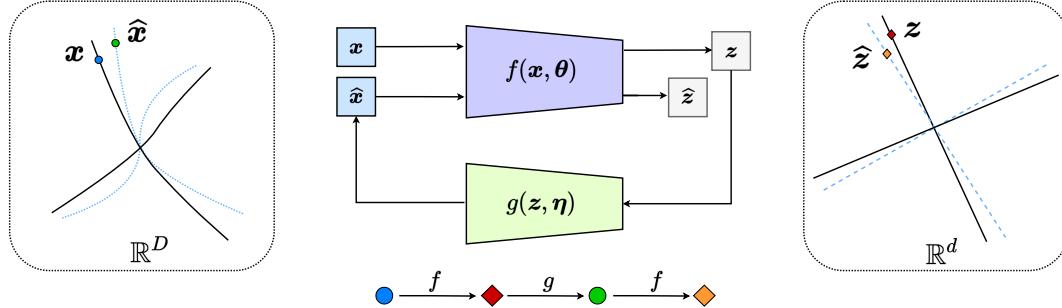


Figure 1.24: Illustration of a closed-loop transcription. Here we use a mapping  $f$  to represent the encoder  $\mathcal{E}$  and  $g$  to represent the decoder  $\mathcal{D}$ .

measure to use. Computing similarity between  $\hat{\mathbf{X}}$  and  $\mathbf{X}$  can be very expensive, if not entirely impossible or intractable.<sup>55</sup> Note that in nature, animals are capable of learning all by themselves without comparing their estimate  $\hat{\mathbf{X}}$  with the ground truth  $\mathbf{X}$  in the data space. They typically do not even have that option.

Then how is a system able to learn autonomously without external supervision or comparison? How can they know that  $\hat{\mathbf{X}}$  is consistent with  $\mathbf{X}$  even without directly comparing them? That leads to the idea of “closing the loop”. As it turns out, under the mild conditions that we will make precise in Chapter 5, to ensure  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  are consistent, one only has to encode  $\hat{\mathbf{X}}$  as  $\hat{\mathbf{Z}}$  and check if  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$  are consistent. We call this notion of consistency as *self-consistency*, which can be illustrated by the following diagram:

$$\mathbf{X} \xrightarrow{\mathcal{E}} \mathbf{Z} \xrightarrow{\mathcal{D}} \hat{\mathbf{X}} \xrightarrow{\mathcal{E}} \hat{\mathbf{Z}}, \quad (1.4.16)$$

We refer to this process as a *closed-loop transcription*,<sup>56</sup> which is illustrated in Figure 1.24.

It is arguably true that any autonomous intelligent being only needs to learn a self-consistent representation  $\mathbf{Z}$  of the observed data  $\mathbf{X}$ , because checking consistency in the original data space (often meaning in the external world) is either too expensive or even not physically feasible. The closed-loop formulation allows one to learn an optimal encoding  $f(\cdot, \theta)$  and decoding  $g(\cdot, \eta)$  via a minmax game that depends only on the internal (learned) feature  $\mathbf{Z}$ :

$$\max_{\theta} \min_{\eta} \ell(\mathbf{Z}, \hat{\mathbf{Z}}) + \rho(\mathbf{Z}), \quad (1.4.17)$$

where  $\ell(\mathbf{Z}, \hat{\mathbf{Z}})$  is a loss function based on coding rates of the features  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$ , which, as we will see, can be much easier to compute. Here again,  $\rho(\mathbf{Z})$  is some measure that promotes parsimony and information richness of  $\mathbf{Z}$ . Somewhat surprisingly, as we will see in Chapter 5, under rather mild conditions such as  $\mathbf{X}$  being sufficiently low-dimensional, self-consistency between  $(\mathbf{Z}, \hat{\mathbf{Z}})$  implies consistency in  $(\mathbf{X}, \hat{\mathbf{X}})$ ! In addition, we will also see that a closed-loop system will allow us to learn the distribution in a *continuous and incremental* manner,<sup>57</sup> without suffering from problems such as catastrophic forgetting associated with open-loop models.

## 1.5 Bridging Theory and Practice for Machine Intelligence

So far, we have introduced three related frameworks for learning a compact and structured representation  $\mathbf{Z}$  for a given data distribution  $\mathbf{X}$ :

- The open-ended encoding (1.4.10);
- The bi-directional autoencoding (1.4.14);
- The closed-loop transcription (1.4.16).

<sup>55</sup>Say one wants to minimize certain distributional distance between the two.

<sup>56</sup>Inspired by the transcription process between DNA and RNA or other proteins.

<sup>57</sup>That is to learn with one class at a time or even one sample at a time.

In this book, we will systematically study all three frameworks, one after another:

$$\text{open-ended} \implies \text{bi-directional} \implies \text{closed-loop}, \quad (1.5.1)$$

in Chapter 4, Section 5.1 and Section 5.2 of Chapter 5, respectively.

In the past few years, many theoretical frameworks have been proposed and developed to help understand deep networks. However, many were unable to provide scalable solutions that matched the performance of empirical methods on real-world data and tasks. Many theories do not provide useful guidance on how to further improve practice. Chapters 6 and 7 will show how the framework presented in this book may help bridge the gap between theory and practice. Chapter 6 will show how to use the learned distribution and its representation to conduct (Bayesian) inference for almost all practical tasks that depend on (conditional) generation, estimation, and prediction. Chapter 7 will provide convincing experimental evidence that networks and systems designed from the first principles can achieve competitive and potentially better performance on a variety of tasks such as visual representation learning, image classification, image completion, segmentation, and text generation.

**Back to Intelligence.** As we have mentioned in the beginning, a common and fundamental task of any intelligent being is to learn predictable information from its sensed data. Now we have understood a little about the computational nature of this task, and one should realize that this is a never-ending process, for the following reasons:

- The knowledge learned so far from the data, say by the encoding and decoding schemes, is unlikely to be correct or optimal. Intelligence should have the ability to improve if there are still errors in predicting new observations.
- The data observed so far do not yet cover all the predictable information. Intelligence should be able to recognize that current knowledge is inadequate and have the capability to learn and acquire new information whenever available.

Hence, intelligence is *not* about simply collecting all data in advance and training a model to memorize all the predictable information in the data. In contrast, it is about being equipped with computational mechanisms that can constantly improve current knowledge and acquire new information when available and needed. That is, a fundamental characteristic of any intelligent being or system<sup>58</sup> is *being able to continuously improve or gain information (or knowledge) on its own*. Conceptually, we may illustrate symbolically the relationship between intelligence and information (or knowledge) as follows:

$$\text{Intelligence}(t) = \frac{d}{dt} \text{Information}(t), \quad \text{Information}(t) = \int_0^t \text{Intelligence}(s) ds. \quad (1.5.2)$$

We believe that the closed-loop framework is a universal mechanism that enables self-improving and self-learning, via feedback<sup>59</sup> or gaming<sup>60</sup>. All intelligent beings or systems in nature utilize closed-loop mechanisms for learning at all levels and on all scales. Its ubiquity had inspired early studies that tried to model and emulate intelligence by machines and computers, particularly the Cybernetics movement initiated by Norbert Wiener in the 1940s.

We hope that this book will help people better understand the objectives, principles, and computational mechanisms behind intelligence. It serves as a foundation for further study of higher-level human intelligence, the true “artificial” intelligence, in the future, which we will layout several significant open problems in these new directions at the end of the book in Chapter 8.

---

<sup>58</sup>An animal, a human being, an intelligent robot, the scientific community, and even the entire civilization.

<sup>59</sup>Reinforcement can be viewed as a primitive form of feedback, say the natural selection by the nature.

<sup>60</sup>Scientific inquiries can be viewed as a most advanced form of gaming, through hypothesis formulation and verification.

## Chapter 2

# Learning Linear and Independent Structures

*“The art of doing mathematics consists in finding that special case which contains all the germs of generality.”*

– David Hilbert

*Real data has low-dimensional structure.* To see why this is true, let us consider the unassuming case of static on a TV when the satellite isn't working. At each frame (approximately every  $\frac{1}{30}$  seconds), the RGB static on a screen of size  $H \times W$  is, roughly, sampled independently from a uniform distribution on  $[0, 1]^{3 \times H \times W}$ . In theory, the static *could* resolve to a natural image on any given frame, but even if you spend a thousand years looking at the TV screen, it will not. This discrepancy is explained by the fact that the set of  $H \times W$  natural images takes up a vanishingly small fraction of the hypercube  $[0, 1]^{3 \times H \times W}$ . In particular, it is extremely low-dimensional, compared to the ambient space dimension. Similar phenomena occur for all other types of natural data, such as text, audio, and video. Thus, when we design systems and methods to process natural data and learn its structure or distribution, this is a central property of natural data which we need to take into account.

Therefore, our central task is to learn a distribution that has low intrinsic dimension in a high-dimensional space. In the remainder of this section, we discuss several *classical* methods to perform this task for several somewhat *idealistic* models for the distribution, namely models that are geometrically linear or statistically independent. While these models and methods are important and useful in their own right, we discuss them here as they motivate, inspire, and serve as a predecessor or analogue to more modern methods for more general distributions that involve deep (representation) learning.

Our main approach (and general problem formulation) can be summarized as:

**Problem:** *Given one or several (noisy or incomplete) observations of a ground truth sample from the data distribution, obtain an estimate of this sample.*

This approach underpins several classical methods for data processing, which we discuss in this chapter.

- Section 2.1 — Principal Components Analysis (PCA): Given noisy samples from a distribution supported on *one low-dimensional subspace*, obtain an estimate of the true sample that lies on this subspace.
- Section 2.2 — Complete Dictionary Learning and Independent Components Analysis (ICA): Given noisy samples from a distribution supported on *a union (not the span) of few low-dimensional subspaces*, obtain an estimate of the true samples.



Figure 2.1: Images of human faces and handwritten digits. Despite the seemingly large variety in their appearances, each set of those data spans (approximately) a very low-dimensional (nearly) linear subspace.

- Section 2.3 — Sparse Coding and Overcomplete Dictionary Learning: Given noisy samples from a distribution supported on combinations of a few incoherent vectors, such as the coordinate axes, obtain an estimate of the true sample, which also has this property.

As we will soon reveal in later chapters, in the deep learning era, modern methods essentially adopt the same approach to learn.

In this chapter, as described above, we make simplifying modeling assumptions that essentially assume the data have geometrically (nearly, piece-wise) linear structures and statistically independent components. In Chapter 1, we have referred to such models of the data as “analytical models”. These modeling assumptions allow us to derive efficient algorithms with provable efficiency guarantees<sup>1</sup> for processing data at scale. However, they are imperfect models for often-complex real-world data distributions, and so their underlying assumptions only approximately hold. This means that the guarantees afforded by detailed analysis of these algorithms also only approximately hold in the case of real data. Nonetheless, the techniques discussed in this chapter are useful in their own right, and beyond that, serve as the “special case with the germ of generality”, so to speak, in that they present a guiding motivation and intuition for the more general paradigms within (deep) learning of more general distributions that we later address.

## 2.1 A Low-Dimensional Subspace

### 2.1.1 Principal Components Analysis (PCA)

Perhaps the simplest modeling assumption possible for low-dimensional structure is the so-called *low-rank* assumption. Letting  $D$  be the dimension of our data space, we assume that our data belong to a low-dimensional subspace of dimension  $d \ll D$ , possibly plus some small disturbances. This ends up being a nearly valid assumption for some surprisingly complex data, such as images of handwritten digits and face data [RD03] as shown in Figure 2.1, yet as we will see, it will lend itself extremely well to comprehensive analysis.

**Problem formulation.** To write this in mathematical notation, we represent a subspace  $\mathcal{S} \subseteq \mathbb{R}^D$  of dimension  $d$  by an orthonormal matrix  $\mathbf{U} \in \mathrm{O}(D, d) \subseteq \mathbb{R}^{D \times d}$  such that the columns of  $\mathbf{U}$  span  $\mathcal{S}$ . Then, we say that our data  $\{\mathbf{x}_i\}_{i=1}^N \subseteq \mathbb{R}^D$  have (approximate) low-rank structure if there exists an orthonormal

<sup>1</sup>Both in terms of data and computational complexity.

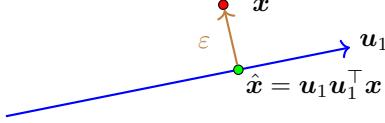


Figure 2.2: **Geometry of PCA.** A data point  $\mathbf{x}$  (red) is projected onto the one-dimensional learned subspace spanned by the unit basis vector  $\mathbf{u}_1$  (blue arrow). The projection  $\mathbf{U}\mathbf{U}^\top \mathbf{x} = \mathbf{u}_1\mathbf{u}_1^\top \mathbf{x}$  (green) is the denoised version of  $\mathbf{x}$  using the low-dimensional structure given by  $\mathbf{u}_1$ , and  $\boldsymbol{\varepsilon}$  (brown arrow) represents the projection residual or noise.

matrix  $\mathbf{U} \in \text{O}(D, d)$ , vectors  $\{\mathbf{z}_i\}_{i=1}^N \subseteq \mathbb{R}^d$ , and *small* vectors  $\{\boldsymbol{\varepsilon}_i\}_{i=1}^N \subseteq \mathbb{R}^D$  such that

$$\mathbf{x}_i = \mathbf{U}\mathbf{z}_i + \boldsymbol{\varepsilon}_i, \quad \forall i \in [N]. \quad (2.1.1)$$

Here  $\boldsymbol{\varepsilon}_i$  are disturbances that prevent the data from being perfectly low rank; their presence in our model allows us to quantify the degree to which our analysis remains relevant in the presence of deviations from our model. The true supporting subspace is  $\mathcal{S} \doteq \text{col}(\mathbf{U})$ , the span of the columns of  $\mathbf{U}$ . To process all we can from this data, we need to recover  $\mathcal{S}$ ; to do this it is sufficient to recover  $\mathbf{U}$ , also called the *principal components*. Fortunately, this is a computationally tractable task named **Principal Component Analysis**, and we discuss now how to solve it.

Given data  $\{\mathbf{x}_i\}_{i=1}^N$  distributed as in (2.1.1), we aim to recover the model  $\mathbf{U}$ . A natural approach is to find the subspace  $\mathbf{U}^*$  and latent vectors  $\{\mathbf{z}_i^*\}_{i=1}^N$  which yield the best approximation  $\mathbf{x}_i \approx \mathbf{U}^*\mathbf{z}_i^*$ . Namely, we aim to solve the problem

$$\min_{\tilde{\mathbf{U}}, \{\tilde{\mathbf{z}}_i\}_{i=1}^N} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{z}}_i\|_2^2, \quad (2.1.2)$$

where  $\tilde{\mathbf{U}}$  is constrained to be an orthonormal matrix, as above. We will omit this constraint in similar statements below for the sake of concision.

**Subspace encoding-decoding via denoising.** To simplify this problem, for a fixed  $\tilde{\mathbf{U}}$ , we have (proof as exercise):

$$\min_{\{\tilde{\mathbf{z}}_i\}_{i=1}^N} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{z}}_i\|_2^2 = \frac{1}{N} \sum_{i=1}^N \min_{\tilde{\mathbf{z}}_i} \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{z}}_i\|_2^2 \quad (2.1.3)$$

$$= \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}_i\|_2^2. \quad (2.1.4)$$

That is, the optimal solution  $(\mathbf{U}^*, \{\mathbf{z}_i^*\}_{i=1}^N)$  to the above optimization problem has  $\mathbf{z}_i^* = (\mathbf{U}^*)^\top \mathbf{x}_i$ .

Now, we can write the original optimization problem in  $\tilde{\mathbf{U}}^*$  and  $\{\tilde{\mathbf{z}}_i\}_{i=1}^N$  as an optimization problem just over  $\tilde{\mathbf{U}}$ , i.e., to obtain the basis  $\mathbf{U}^*$  and compact codes  $\{\mathbf{z}_i^*\}_{i=1}^N$  it suffices to solve either of the two following equivalent problems:

$$\min_{\tilde{\mathbf{U}}, \{\tilde{\mathbf{z}}_i\}_{i=1}^N} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{z}}_i\|_2^2 = \min_{\tilde{\mathbf{U}}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}_i\|_2^2. \quad (2.1.5)$$

Note that the problem on the right-hand side of (2.1.5) is a *denoising* problem: given noisy observations  $\mathbf{x}_i$  of low-rank data, we aim to find the *noise-free* copy of  $\mathbf{x}_i$ , which under the model (2.1.1) is  $\mathbf{z}_i$ . That is, the denoised input  $\hat{\mathbf{x}}_i = \mathbf{U}\mathbf{U}^\top \mathbf{x}_i$ . Notice that this is the point on the subspace that is closest to  $\mathbf{x}_i$ , as visualized in Figure 2.2. Here by solving the equivalent problem of finding the best subspace, parameterized by the learned basis  $\mathbf{U}^*$ , we learn an approximation to the *denoiser*, i.e., the projection matrix  $\mathbf{U}^*(\mathbf{U}^*)^\top \approx \mathbf{U}\mathbf{U}^\top$  that projects the noisy data point  $\mathbf{x}_i$  onto the subspace  $\mathcal{S}$ .

Putting the above process together, we essentially obtain a simple encoding-decoding scheme that maps a data point  $\mathbf{x}$  in  $\mathbb{R}^D$  to a lower-dimensional (latent) space  $\mathbb{R}^d$  and then back to  $\mathbb{R}^D$ :

$$\mathbf{x} \xrightarrow{\mathcal{E}=(\mathbf{U}^*)^\top} \mathbf{z} \xrightarrow{\mathcal{D}=\mathbf{U}^*} \hat{\mathbf{x}}. \quad (2.1.6)$$

Here,  $\mathbf{z} \in \mathbb{R}^d$  can be viewed as the low-dimensional compact code (or a latent representation) of a data point  $\mathbf{x} \in \mathbb{R}^D$  and the learned subspace basis  $\mathbf{U}^*$  as the associated codebook whose columns are the (learned) optimal code words. The process achieves the function of denoising  $\mathbf{x}$  by projecting it onto the subspace spanned by  $\mathbf{U}^*$ .

**Computing the subspace basis.** For now, we continue our calculation. Let  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$  be the matrix whose columns are the observations  $\mathbf{x}_i$ . We have (proof as exercise)

$$\arg \min_{\tilde{\mathbf{U}}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}_i\|_2^2 = \arg \max_{\tilde{\mathbf{U}}} \frac{1}{N} \sum_{i=1}^N \|\tilde{\mathbf{U}}^\top \mathbf{x}_i\|_F^2 \quad (2.1.7)$$

$$= \arg \max_{\tilde{\mathbf{U}}} \text{tr} \left\{ \tilde{\mathbf{U}}^\top \left( \frac{\mathbf{X}\mathbf{X}^\top}{N} \right) \tilde{\mathbf{U}} \right\}. \quad (2.1.8)$$

Thus, to compute the principal components, we find the orthogonal matrix  $\tilde{\mathbf{U}}$  which maximizes the term  $\text{tr}(\tilde{\mathbf{U}}^\top (\mathbf{X}\mathbf{X}^\top/N)\tilde{\mathbf{U}})$ . We can prove via induction that this matrix  $\mathbf{U}^*$  has columns which are the *top d unit eigenvectors* of  $\mathbf{X}\mathbf{X}^\top/N$ . We leave the whole proof to the reader in Exercise 2.1, but we handle the base case of the induction here. Suppose that  $d = 1$ . Then we only have a single unit vector  $\mathbf{u}$  to recover, so the above problem reduces to

$$\max_{\tilde{\mathbf{u}}: \|\tilde{\mathbf{u}}\|_2=1} \tilde{\mathbf{u}}^\top (\mathbf{X}\mathbf{X}^\top/N) \tilde{\mathbf{u}}. \quad (2.1.9)$$

This is the so-called *Rayleigh quotient* of  $\mathbf{X}\mathbf{X}^\top/N$ . By invoking the spectral theorem we diagonalize  $\mathbf{X}\mathbf{X}^\top/N = \mathbf{V}\Lambda\mathbf{V}^\top$ , where  $\mathbf{V}$  is orthogonal and  $\Lambda$  is diagonal with non-negative entries. Hence

$$\tilde{\mathbf{u}}^\top (\mathbf{X}\mathbf{X}^\top/N) \tilde{\mathbf{u}} = \tilde{\mathbf{u}}^\top \mathbf{V}\Lambda\mathbf{V}^\top \mathbf{u} = (\mathbf{V}^\top \tilde{\mathbf{u}})^\top \Lambda (\mathbf{V}^\top \tilde{\mathbf{u}}). \quad (2.1.10)$$

Since  $\mathbf{V}$  is an invertible orthogonal transformation,  $\mathbf{V}^\top \tilde{\mathbf{u}}$  is a unit vector, and optimizing over  $\tilde{\mathbf{u}}$  is equivalent to optimizing over  $\tilde{\mathbf{w}} \doteq \mathbf{V}^\top \tilde{\mathbf{u}}$ . Hence, we can write

$$\tilde{\mathbf{u}}^\top (\mathbf{X}\mathbf{X}^\top/N) \tilde{\mathbf{u}} = \tilde{\mathbf{w}}^\top \Lambda \tilde{\mathbf{w}}, \quad (2.1.11)$$

whose optimal solutions  $\mathbf{w}^*$  among unit vectors are one-hot vectors whose only nonzero (hence unit) entry is in one of the indices corresponding to the largest eigenvalue of  $\mathbf{X}\mathbf{X}^\top/N$ . This means that  $\tilde{\mathbf{u}} = \mathbf{V}\tilde{\mathbf{w}}$ , the optimal solution to the original problem, corresponds to a unit eigenvector of  $\mathbf{X}\mathbf{X}^\top/N$  (i.e., column of  $\mathbf{V}$ ) which corresponds to the largest eigenvalue. Suitably generalizing this to the case  $d > 1$ , and summarizing all the previous discussion, we have the following informal Theorem.

**Theorem 2.1.** Suppose that our dataset  $\{\mathbf{x}_i\}_{i=1}^N \subseteq \mathbb{R}^D$  can be written as

$$\mathbf{x}_i = \mathbf{U}\mathbf{z}_i + \boldsymbol{\varepsilon}_i, \quad \forall i \in [N], \quad (2.1.12)$$

where  $\mathbf{U} \in \text{O}(D, d)$  captures the low-rank structure,  $\{\mathbf{z}_i\}_{i=1}^N \subseteq \mathbb{R}^d$  are the compact codes of the data, and  $\{\boldsymbol{\varepsilon}_i\}_{i=1}^N \subseteq \mathbb{R}^D$  are small vectors indicating the deviation of our data from the low-rank model. Then the principal components  $\mathbf{U}^* \in \text{O}(D, d)$  of our dataset are given by the top  $d$  eigenvectors of  $\mathbf{X}\mathbf{X}^\top/N$ , where  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ , and approximately correspond to the optimal linear denoiser:  $\mathbf{U}^*(\mathbf{U}^*)^\top \approx \mathbf{U}\mathbf{U}^\top$ .

We do not give explicit rates of approximation here as they can become rather technical. In the special case that  $\boldsymbol{\varepsilon}_i = \mathbf{0}$  for all  $i$ , the learned  $\mathbf{U}^*$  spans the support of the samples  $\{\mathbf{x}_i\}_{i=1}^N$ . If in addition the  $\mathbf{z}_i$  are sufficiently diverse (say, spanning all of  $\mathbb{R}^d$ ) then we would have perfect recovery:  $\mathbf{U}^*(\mathbf{U}^*)^\top = \mathbf{U}\mathbf{U}^\top$ .

*Remark 2.1.* In some data analysis tasks, the data matrix  $\mathbf{X}$  is formatted such that each data point is a *row* rather than a *column* as is presented here. In this case the principal components are the top  $d$  eigenvectors of  $\mathbf{X}^\top \mathbf{X}/N$ .

*Remark 2.2 (Basis Selection via Denoising Eigenvalues).* In many cases, either our data will not truly be distributed according to a subspace-plus-noise model, or we will not know the true underlying dimension  $d$  of the subspace. In this case, we have to choose  $d$ ; this problem is called *model selection*. In the restricted

case of PCA, one way to perform model selection is to compute  $\mathbf{X}\mathbf{X}^\top/N$  and look for instances where adjacent eigenvalues sharply decrease; this is one indicator that the index of the larger eigenvalue is the “true dimension  $d$ ”, and the rest of the eigenvalues of  $\mathbf{X}\mathbf{X}^\top/N$  are contributed by the noise or disturbances  $\boldsymbol{\varepsilon}_i$ . Model selection is a difficult problem and, nowadays in the era of deep learning where it is called “hyperparameter optimization”, is usually done via brute force or Bayesian optimization.

*Remark 2.3* (Denoising Samples). The expression on the right-hand side of (2.1.5), that is,

$$\min_{\tilde{\mathbf{U}}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}_i\|_2^2, \quad (2.1.13)$$

is what’s known as a *denoising problem*, thusly named because it is an optimization problem whose solution *removes the noise from the samples so it fits on the subspace*. Denoising—learning a map which removes noise from noisy samples so that it fits on the data structure (such as in (2.1.1), but maybe more complicated)—is a common method for learning distributions that will be discussed in the sequel and throughout the manuscript. Note that we have already discussed this notion, but it bears repeating due to its central importance in later Chapters.

*Remark 2.4* (Neural Network Interpretation). If we do a PCA, we approximately recover the support of the distribution encoded by the parameter  $\mathbf{U}^*$ . The learned denoising map then takes the form  $\mathbf{U}^*(\mathbf{U}^*)^\top$ . On top of being a denoiser, this can be viewed as a *simple two-layer weight-tied linear neural network*: the first layer multiplies by  $(\mathbf{U}^*)^\top$ , and the second layer multiplies by  $\mathbf{U}^*$ , namely

$$\text{denoise}(\mathbf{x}) = \mathbf{U}^* \circ \underbrace{\text{id} \circ \underbrace{(\mathbf{U}^*)^\top \mathbf{x}}_{\text{first “layer”}}}_{\text{post-activation of first “layer”}} \underbrace{}_{\text{output of “NN”}} \quad (2.1.14)$$

Contrasting this to a standard two-layer neural network, we see a structural similarity:

$$\text{NN}(\mathbf{x}) = \mathbf{W}^* \circ \underbrace{\text{ReLU} \circ \underbrace{(\mathbf{U}^*)^\top \mathbf{x}}_{\text{first layer}}}_{\text{post-activation of first layer}} \underbrace{}_{\text{output of NN}} \quad (2.1.15)$$

In particular, PCA can be interpreted as *learning a simple two-layer denoising autoencoder*,<sup>2</sup> one of the simplest examples of a non-trivial neural network. In this framework, the *learned representations* would just be  $(\mathbf{U}^*)^\top \mathbf{x} \approx \mathbf{z}$ . In this way, PCA serves as a model problem for (deep) representation learning, which we will draw upon further in the monograph. Notice that in this analogy, the representations reflect, or are projections of, the input data towards a learned low-dimensional structure. This property will be particularly relevant in the future.

### 2.1.2 Pursuing Low-rank Structure via Power Iteration

There is a computationally efficient way to estimate the top eigenvectors of  $\mathbf{X}\mathbf{X}^\top/N$  or any symmetric positive semidefinite matrix  $\mathbf{M}$ , called *power iteration*. This method is the building block of several algorithmic approaches to high-dimensional data analysis that we discuss later in the Chapter, so we discuss it here.

Let  $\mathbf{M}$  be a symmetric positive semidefinite matrix. There exists an orthonormal basis for  $\mathbb{R}^D$  consisting of eigenvectors  $(\mathbf{w}_i)_{i=1}^D$  of  $\mathbf{M}$ , with corresponding eigenvalues  $\lambda_1 \geq \dots \geq \lambda_D \geq 0$ . By definition, any eigenvector  $\mathbf{w}_i$  satisfies  $\lambda_i \mathbf{w}_i = \mathbf{M}\mathbf{w}_i$ . Therefore, for any  $\lambda_i > 0$ ,  $\mathbf{w}_i$  is a “fixed point” to the following equation:

$$\mathbf{w} = \frac{\mathbf{M}\mathbf{w}}{\|\mathbf{M}\mathbf{w}\|_2}. \quad (2.1.16)$$

---

<sup>2</sup>In fact, as we have mentioned in the previous chapter, PCA was one of the first problems that neural networks were used to solve [BH89; Oja82].

**Theorem 2.2** (Power Iteration). *Assume that  $\lambda_1 > \lambda_i$  for all  $i > 1$ . If we compute the fixed point of (2.1.16) using the following iteration:*

$$\mathbf{v}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{1}), \quad \mathbf{v}_{t+1} \leftarrow \frac{\mathbf{M}\mathbf{v}_t}{\|\mathbf{M}\mathbf{v}_t\|_2}, \quad (2.1.17)$$

then, in the limit,  $\mathbf{v}_t$  will converge to a top unit-norm eigenvector of  $\mathbf{M}$ .

*Proof.* First, note that for all  $t$ , we have

$$\mathbf{v}_t = \frac{\mathbf{M}\mathbf{v}_{t-1}}{\|\mathbf{M}\mathbf{v}_{t-1}\|_2} = \frac{\mathbf{M}^2\mathbf{v}_{t-2}}{\|\mathbf{M}\mathbf{v}_{t-1}\|_2 \|\mathbf{M}\mathbf{v}_{t-2}\|_2} = \cdots = \frac{\mathbf{M}^t\mathbf{v}_0}{\prod_{s=1}^t \|\mathbf{M}\mathbf{v}_s\|_2}. \quad (2.1.18)$$

Thus,  $\mathbf{v}_t$  has the same direction as  $\mathbf{M}^t\mathbf{v}_0$  and is unit norm, so we can write

$$\mathbf{v}_t = \frac{\mathbf{M}^t\mathbf{v}_0}{\|\mathbf{M}^t\mathbf{v}_0\|_2}. \quad (2.1.19)$$

Because all the eigenvectors  $\mathbf{w}_i$  of  $\mathbf{M}$  form an orthonormal basis for  $\mathbb{R}^D$ , we can write

$$\mathbf{v}_0 = \sum_{i=1}^D \alpha_i \mathbf{w}_i, \quad (2.1.20)$$

where because  $\mathbf{v}_0$  is Gaussian the  $\alpha_i$  are all nonzero with probability 1. Thus, we can use our earlier expression for  $\mathbf{v}_t$  to write

$$\mathbf{v}_t = \frac{\mathbf{M}^t\mathbf{v}_0}{\|\mathbf{M}^t\mathbf{v}_0\|_2} = \frac{\sum_{i=1}^D \lambda_i^t \alpha_i \mathbf{w}_i}{\|\sum_{i=1}^D \lambda_i^t \alpha_i \mathbf{w}_i\|_2} = \frac{\sum_{i=1}^D \lambda_i^t \alpha_i \mathbf{w}_i}{\sum_{i=1}^D \lambda_i^t |\alpha_i|}. \quad (2.1.21)$$

Now, let us consider the case where  $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_D \geq 0$ . (The case with repeated top eigenvalues goes similarly.) Then we can write

$$\mathbf{v}_t = \frac{\alpha_1 \mathbf{w}_1 + \sum_{i=2}^D (\lambda_i/\lambda_1)^t \alpha_i \mathbf{w}_i}{|\alpha_1| + \sum_{i=2}^D (\lambda_i/\lambda_1)^t |\alpha_i|}. \quad (2.1.22)$$

Because  $\lambda_1 > \lambda_i$  for all  $i > 1$ , the terms inside the summation go to 0 exponentially fast, and the remainder is the limit

$$\lim_{t \rightarrow \infty} \mathbf{v}_t = \frac{\alpha_1}{|\alpha_1|} \mathbf{w}_1 = \text{sign}(\alpha_1) \mathbf{w}_1, \quad (2.1.23)$$

which is a top unit eigenvector of  $\mathbf{M}$ . The top eigenvalue  $\lambda_1$  of  $\mathbf{M}$  can be estimated by  $\mathbf{v}_t^\top \mathbf{M} \mathbf{v}_t$ , which converges similarly rapidly to  $\lambda_1$ .  $\square$

To find the second top eigenvector, we apply the power iteration algorithm to  $\mathbf{M} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^\top$ , which has eigenvectors  $(\mathbf{w}_i)_{i=2}^D$  and corresponding eigenvalues  $(\lambda_i)_{i=2}^D$ . By repeating this procedure  $d$  times in sequence, we can very efficiently estimate the top  $d$  eigenvectors of  $\mathbf{M}$  very quickly, for any symmetric positive semidefinite matrix  $\mathbf{M}$ . Thus we can also apply it to  $\mathbf{X}\mathbf{X}^\top/N$  to recover the top  $d$  principal components, which is what we were after in the first place. Notice that this approach recovers one principal component at a time; we will contrast this to other algorithmic approaches, such as gradient descent on global objective functions, in future sections.

### 2.1.3 Probabilistic PCA

Notice that the above formulation makes no statistical assumptions on the data-generating process. However, it is common to include statistical elements within a given data model, as it may add further enlightening interpretations about the result of the analysis. As such, we ask the natural question: *what is the statistical analogue to low-dimensional structure?* Our answer is that a low-dimensional *distribution* is one whose support is concentrated around a low-dimensional geometric structure.

To illustrate this point, we discuss *probabilistic principal component analysis (PPCA)*. This formulation can be viewed as a statistical variant of regular PCA. Mathematically, we now consider our data as samples from a random variable  $\mathbf{x}$  taking values in  $\mathbb{R}^D$  (also sometimes called a *random vector*). We say that  $\mathbf{x}$  has (approximate) low-rank statistical structure if and only if there exists an orthonormal matrix  $\mathbf{U} \in O(D, d)$ , a random variable  $\mathbf{z}$  taking values in  $\mathbb{R}^d$ , and a *small* random variable  $\boldsymbol{\varepsilon}$  taking values in  $\mathbb{R}^D$  such that  $\mathbf{z}$  and  $\boldsymbol{\varepsilon}$  are independent, and

$$\mathbf{x} = \mathbf{U}\mathbf{z} + \boldsymbol{\varepsilon}. \quad (2.1.24)$$

Our goal is again to recover  $\mathbf{U}$ . Towards this end, we set up the analogous problem as in Subsection (2.1.1), i.e., optimizing over subspace supports  $\tilde{\mathbf{U}}$  and random variables  $\mathbf{z}$  to solve the following problem:

$$\min_{\tilde{\mathbf{U}}, \tilde{\mathbf{z}}} \mathbb{E} \|\mathbf{x} - \tilde{\mathbf{U}}\tilde{\mathbf{z}}\|_2^2. \quad (2.1.25)$$

Since we are finding the best such random variable  $\mathbf{z}$  we can find its realization  $\mathbf{z}(\mathbf{x})$  separately for each value of  $\mathbf{x}$ . Performing the same calculations as in Subsection (2.1.1), we obtain

$$\min_{\tilde{\mathbf{U}}, \tilde{\mathbf{z}}} \mathbb{E} \|\mathbf{x} - \tilde{\mathbf{U}}\tilde{\mathbf{z}}\|_2^2 = \min_{\tilde{\mathbf{U}}} \mathbb{E} \min_{\tilde{\mathbf{z}}(\mathbf{x})} \|\mathbf{x} - \tilde{\mathbf{U}}\tilde{\mathbf{z}}(\mathbf{x})\|_2^2 = \min_{\tilde{\mathbf{U}}} \mathbb{E} \|\mathbf{x} - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}\|_2^2, \quad (2.1.26)$$

again re-emphasizing the fact that the estimated subspace with principal components  $\mathbf{U}^*$  corresponds to a denoiser  $\mathbf{U}^*(\mathbf{U}^*)^\top$  which projects onto that subspace. As before, we obtain

$$\arg \min_{\tilde{\mathbf{U}}} \mathbb{E} \|\mathbf{x} - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}\|_2^2 = \arg \max_{\tilde{\mathbf{U}}} \mathbb{E} \|\tilde{\mathbf{U}}^\top \mathbf{x}\|_2^2 \quad (2.1.27)$$

$$= \arg \max_{\tilde{\mathbf{U}}} \text{tr}(\tilde{\mathbf{U}}^\top \mathbb{E}[\mathbf{x}\mathbf{x}^\top] \tilde{\mathbf{U}}), \quad (2.1.28)$$

and the solution to the latter problem is just the top  $d$  principal components of the second moment matrix  $\mathbb{E}[\mathbf{x}\mathbf{x}^\top]$ . Actually, the above problems are visually very similar to the equations for computing the principal components in the previous Subsection, except with  $\mathbb{E}[\mathbf{x}\mathbf{x}^\top]$  replacing  $\mathbf{X}\mathbf{X}^\top/N$ . In fact, the latter quantity is an estimate for the former. Both formulations effectively do the same thing, and have the same practical solution—compute the left singular vectors of the data matrix  $\mathbf{X}$ , or equivalently the top eigenvectors of the estimated covariance matrix  $\mathbf{X}\mathbf{X}^\top/N$ . The statistical formulation, however, has an additional interpretation. Suppose that  $\mathbb{E}[\mathbf{z}] = \mathbf{0}$  and  $\mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}$ . We have

$$\mathbb{E}[\mathbf{x}] = \mathbf{U} \mathbb{E}[\mathbf{z}] + \mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}, \quad (2.1.29)$$

so that  $\text{Cov}[\mathbf{x}] = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]$ . Now working out  $\text{Cov}[\mathbf{x}]$  we have

$$\text{Cov}[\mathbf{x}] = \mathbf{U} \text{Cov}[\mathbf{z}] \mathbf{U}^\top + \text{Cov}[\boldsymbol{\varepsilon}] = \mathbf{U} \mathbb{E}[\mathbf{z}\mathbf{z}^\top] \mathbf{U}^\top + \text{Cov}[\boldsymbol{\varepsilon}]. \quad (2.1.30)$$

In particular, if  $\text{Cov}[\boldsymbol{\varepsilon}]$  is small, it holds that  $\text{Cov}[\mathbf{x}] = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]$  is approximately a low-rank matrix, in particular rank  $d$ . Thus the top  $d$  eigenvectors of  $\mathbb{E}[\mathbf{x}\mathbf{x}^\top]$  essentially summarize the whole covariance matrix. But they are also the principal components, so we can interpret principal component analysis as performing a low-rank decomposition of  $\text{Cov}[\mathbf{x}]$ .

*Remark 2.5.* By using the probabilistic viewpoint of PCA, we achieve a clearer and more quantitative understanding of how it relates to denoising. First, consider the denoising problem in (2.1.26), namely

$$\min_{\tilde{\mathbf{U}}} \mathbb{E} \|\mathbf{x} - \tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top \mathbf{x}\|_2^2. \quad (2.1.31)$$

It is not too hard to prove that if  $\tilde{\mathbf{U}}$  has  $d$  columns and if  $\boldsymbol{\varepsilon}$  is an isotropic Gaussian random variable, i.e., with distribution  $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ ,<sup>3</sup> then for *any* optimal solution  $\mathbf{U}^*$  to this problem, we have

$$\mathbf{U}^*(\mathbf{U}^*)^\top = \mathbf{U}\mathbf{U}^\top \quad (2.1.32)$$

---

<sup>3</sup>Other distributions work so long as they support all of  $\mathbb{R}^D$ , but the Gaussian is the easiest to work with here.

and so the true supporting subspace, say  $\mathcal{S} \doteq \text{col}(\mathbf{U})$ , is recovered as the span of columns of  $\mathbf{U}^*$ , since

$$\mathcal{S} = \text{col}(\mathbf{U}) = \text{col}(\mathbf{U}\mathbf{U}^\top) = \text{col}(\mathbf{U}^*(\mathbf{U}^*)^\top) = \text{col}(\mathbf{U}^*). \quad (2.1.33)$$

In particular, the learned *denoising map*  $\mathbf{U}^*(\mathbf{U}^*)^\top$  is an orthogonal projection onto  $\mathcal{S}$ , pushing noisy points onto the ground truth supporting subspace. We can establish a similar technical result in the case where we only have finite samples, as in Theorem 2.1, but this takes more effort and technicality. Summarizing this discussion, we have the following informal Theorem.

**Theorem 2.3.** *Suppose that the random variable  $\mathbf{x}$  can be written as*

$$\mathbf{x} = \mathbf{U}\mathbf{z} + \boldsymbol{\varepsilon} \quad (2.1.34)$$

where  $\mathbf{U} \in \text{O}(D, d)$  captures the low-rank structure,  $\mathbf{z}$  is a random vector taking values in  $\mathbb{R}^d$ , and  $\boldsymbol{\varepsilon}$  is a random vector taking values in  $\mathbb{R}^D$  such that  $\mathbf{z}$  and  $\boldsymbol{\varepsilon}$  are independent, and  $\boldsymbol{\varepsilon}$  is small. Then the principal components  $\mathbf{U}^* \in \text{O}(D, d)$  of our dataset are given by the top  $d$  eigenvectors of  $\mathbb{E}[\mathbf{x}\mathbf{x}^\top]$ , and approximately correspond to the optimal linear denoiser:  $\mathbf{U}^*(\mathbf{U}^*)^\top \approx \mathbf{U}\mathbf{U}^\top$ .

## 2.1.4 Matrix Completion

In the previous Subsections, we discussed the problem of *learning a low-rank geometric or statistical distribution*, where the data were sampled from a subspace with additive noise. But this is not the only type of disturbance from a low-dimensional distribution that is worthwhile to study. In this subsection, we introduce one more class of non-additive errors which become increasingly important in deep learning. Let us consider the case where we have some data  $\{\mathbf{x}_i\}_{i=1}^n$  generated according to (2.1.1). Now we arrange them into a matrix  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ . Unlike before, we do not observe  $\mathbf{X}$  directly; we instead imagine that our observation was corrupted en route and we obtained

$$\mathbf{Y} = \mathbf{M} \odot \mathbf{X}, \quad (2.1.35)$$

where  $\mathbf{M} \in \{0, 1\}^{D \times N}$  is a *mask* that is known by us, and  $\odot$  is element-wise multiplication. In this case, our goal is to recover  $\mathbf{X}$  (from which point we can use PCA to recover  $\mathbf{U}_o$ , etc), given only the corrupted observation  $\mathbf{Y}$ , the mask  $\mathbf{M}$ , and the knowledge that  $\mathbf{X}$  is (approximately) low-rank. This is called *low-rank matrix completion*.

There are many excellent resources discussing algorithms and approaches to solve this problem [WM22]. Indeed, this and similar generalizations of this low-rank structure recovery problem are resolved by “robust PCA”. We will not go into the solution method here. Instead, we will discuss under what conditions this problem is *plausible* to solve. On one hand, in the most absurd case, suppose that each entry of the matrix  $\mathbf{X}$  were chosen independently from all the others. Then there would be no hope of recovering  $\mathbf{X}$  exactly even if only one entry were missing and we had  $DN - 1$  entries. On the other hand, suppose that we knew that  $\mathbf{X}$  has rank 1 exactly, which is an extremely strong condition on the low-dimensional structure of the data, and we were dealing with the mask

$$\mathbf{M} = \begin{bmatrix} \mathbf{1}_{(D-1) \times 1} & \mathbf{0}_{(D-1) \times (N-1)} \\ 1 & \mathbf{1}_{1 \times (N-1)} \end{bmatrix}. \quad (2.1.36)$$

Then we know that the data were distributed on a line, and we know a vector on that line—it is just the first column of the matrix  $\mathbf{Y} = \mathbf{M} \odot \mathbf{X}$ . From the last coordinate of each column, also revealed to us by the mask, we can solve for each column since for each final coordinate there is only one vector on the line with this coordinate. Thus we can reconstruct  $\mathbf{X}$  with perfect accuracy, and we only need a linear number of observations  $D + N - 1$ .

In the real world, actual problems are somewhere in between the two limiting cases discussed above. Yet the differences between these two extremes, as well as the earlier discussion of PCA, reveal a general kernel of truth:

*The lower-dimensional and more structured the data distribution is, the easier it is to process, and the fewer observations are needed—provided that the algorithm effectively utilizes this low-dimensional structure.*

As is perhaps predictable, we will encounter this motif repeatedly in the remainder of the manuscript, starting in the very next section.

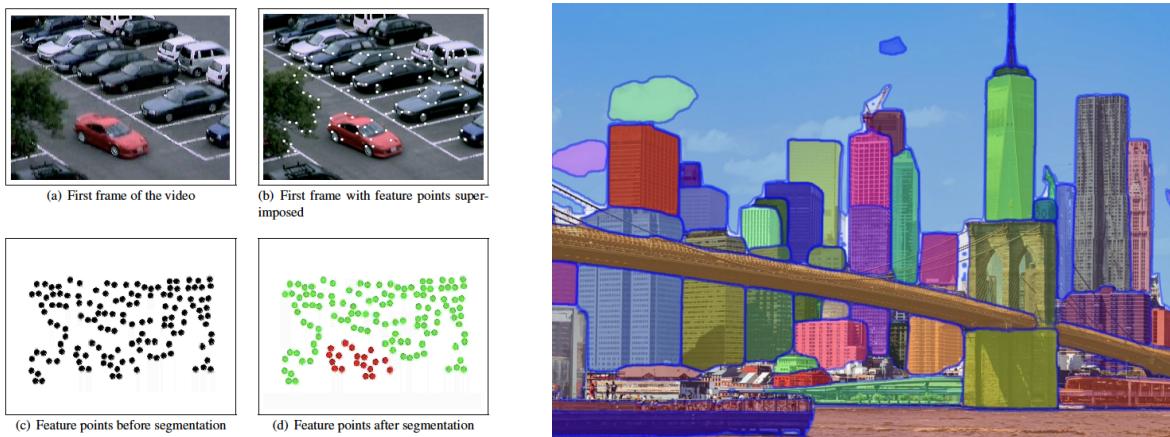


Figure 2.3: **Left:** features tracked on independently moving objects in a scene. **Right:** image patches associated with different regions of an image.

## 2.2 A Mixture of Complete Low-Dimensional Subspaces

As we have seen, low-rank signal models are rich enough to provide a full picture of the interplay between low-dimensionality in data and efficient and scalable computational algorithms for representation and recovery under errors. These models imply a *linear* and symmetric representation learning pipeline (2.1.6):

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) = \mathbf{U}^\top \mathbf{x}, \quad \hat{\mathbf{x}} = \mathcal{D}(\mathbf{z}) = \mathbf{U}\mathbf{z},$$

which can be provably learned from finite samples of  $\mathbf{x}$  with principal component analysis (solved efficiently, say, with the power method) whenever the distribution of  $\mathbf{x}$  truly is linear. This is a restrictive assumption—for as Harold Hotelling, the distinguished 20th century statistician,<sup>4</sup> objected following George Dantzig’s presentation of his theory of linear programming for the first time [Dan02],

“...we all know the world is nonlinear.”

Even accounting for its elegance and simplicity, the low-rank assumption is too restrictive to be broadly applicable to modeling real-world data. A key limitation is the assumption of a *single* linear subspace that is responsible for generating the structured observations. In many practical applications, structure generated by a *mixture* of distinct low-dimensional subspaces provides a more realistic model. For example, consider a video sequence that captures the motion of several distinct objects, each subject to its own independent displacement (Fig. 2.3 left). Under suitable assumptions on the individual motions, each object becomes responsible for an independent low-dimensional subspace in the concatenated sequence of video frames [VM04]. As another example, consider modeling natural images via learning a model for the distribution of *patches*, spatially-contiguous collections of pixels, within an image (Fig. 2.3 right). Unlike in the Eigenface example we saw previously, where images of faces with matched poses can be well-approximated by a single low-dimensional subspace, the patch at a specific location in a natural image can correspond to objects with very different properties—for example, distinct color or shape due to occlusion boundaries. Therefore, modeling the distribution of patches with a single subspace is futile, but a *mixture* of subspaces, one for each region, performs surprisingly well in practice, say for segmenting or compressing purposes [MRY+11].<sup>5</sup> We will see a concrete example in the next chapter (Example 3.12).

In this section, we will begin by discussing the conceptual and algorithmic foundations for structured representation learning when the data distribution can be modeled by a *mixture of low-dimensional subspaces*, as illustrated in Figure 2.4. In this setting, the decoder mapping will be almost as simple as the case of a

<sup>4</sup>By coincidence, also famous for his contributions to the development and naming of Principal Component Analysis [Hot33].

<sup>5</sup>We will return to this observation in Chapter 4, where we will show it can be significantly generalized to learn representations for large-scale modern datasets.

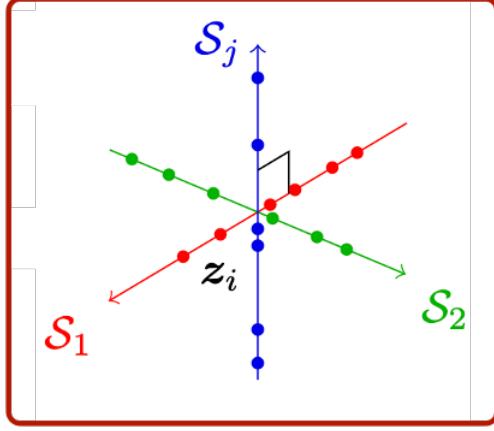


Figure 2.4: Data on a mixture of low-dimensional subspaces, say  $\mathcal{S}_j = \text{col}(\mathbf{U}_j)$ .

single subspace: we simply represent via

$$\hat{\mathbf{x}} = \mathcal{D}(\mathbf{z}) = \left( \sum_{k=1}^K \pi_k(\mathbf{z}) \mathbf{U}_k \right) \mathbf{z}, \quad (2.2.1)$$

where  $\pi_k : \mathbb{R}^d \rightarrow \{0, 1\}$  are a set of *sparse* weighting random variables, such that only a single subspace  $\mathcal{S}_k = \text{col}(\mathbf{U}_k)$  is selected in the sum. However, the task of encoding such data  $\mathbf{x}$  to suitable representations  $\mathbf{z}$ , and learning such an encoder-decoder pair from data, will prove to be more involved.

We will see how ideas from the rich literature on *sparse representation* and *independent component analysis* lead to a natural reformulation of the above decoder architecture through the lens of sparsity, the corresponding encoder architecture (obtained through a power-method-like algorithm analogous to that of principal component analysis), and strong guarantees of correctness and efficiency for learning such encoder-decoder pairs from data. In this sense, the case of mixed linear low-dimensional structure already leads to many of the key conceptual components of structured representation learning that we will develop in far greater generality in this book.

### 2.2.1 Mixtures of Subspaces and Sparsely-Used Dictionaries

Let  $\mathbf{U}_1, \dots, \mathbf{U}_K$ , each of size  $D \times d$ , denote a collection of orthonormal bases for  $K$  subspaces of dimension  $d$  in  $\mathbb{R}^D$ . To say that  $\mathbf{x}$  follows a mixture-of-subspaces distribution parameterized by  $\mathbf{U}_1, \dots, \mathbf{U}_K$  means, geometrically speaking, that

$$\mathbf{x} = \mathbf{U}_k \mathbf{z} \quad \text{for some } k \in [K], \quad \mathbf{z} \in \mathbb{R}^d. \quad (2.2.2)$$

The statistical analogue of this geometric model, as we saw for the case of PCA and linear structure, is that  $\mathbf{x}$  follows a *mixture of Gaussians* distribution: that is,

$$\mathbf{x} \sim \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{0}, \mathbf{U}_k \mathbf{U}_k^\top), \quad \text{for some } \pi_k \geq 0, \quad \sum_{k=1}^K \pi_k = 1. \quad (2.2.3)$$

In other words, for each  $k \in [K]$ ,  $\mathbf{x}$  is Gaussian on the low-dimensional subspace  $\text{col}(\mathbf{U}_k)$  with probability  $\pi_k$ .

*Remark 2.6* (A Mixture of Gaussians v.s. A Superposition of Gaussians). One should be aware that the above model (2.2.3) is a *mixture* of Gaussian distributions, not to be confused with a mixture of Gaussian variables by superposition, say

$$\mathbf{x} = \sum_{i=1}^n w_i \mathbf{x}_i, \quad \mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{U}_i \mathbf{U}_i^\top), \quad (2.2.4)$$

where  $\mathbf{x}_i$  are independent random Gaussian vectors and  $w_i$  are a set of fixed weights. As we know from the properties of Gaussian vectors, such a superposition  $\mathbf{x}$  will remain a Gaussian distribution.

For now, we focus on the geometric perspective offered by (2.2.2). There is an algebraically convenient alternative to this conditional representation. Consider a *lifted* representation vector  $\mathbf{z} = [\mathbf{z}_1^\top, \dots, \mathbf{z}_K^\top]^\top \in \mathbb{R}^{dK}$ , such that  $\mathbf{z}$  is *d-sparse* with support on one of the  $K$  consecutive non-overlapping blocks of  $d$  coordinates out of  $dK$ . Then (2.2.2) can be written equivalently as

$$\mathbf{x} = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{U}_1 & \dots & \mathbf{U}_K \\ | & \dots & | \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_K \end{bmatrix}}_{\mathbf{z}}, \quad \left\| \begin{bmatrix} \|\mathbf{z}_1\|_2 \\ \vdots \\ \|\mathbf{z}_K\|_2 \end{bmatrix} \right\|_0 = 1. \quad (2.2.5)$$

Here, the  $\ell^0$  “norm”  $\|\cdot\|_0$  measures sparsity by counting the number of nonzero entries:

$$\|\mathbf{z}\|_0 = |\{i \mid z_i \neq 0\}|, \quad (2.2.6)$$

and the matrix  $\mathbf{U} \in \mathbb{R}^{D \times Kd}$  is called a *dictionary* with all the  $\{\mathbf{U}_i\}_{i=1}^K$  as code words. In general, if the number of subspaces in the mixture  $K$  is large enough, there is no bound on the number of columns contained in the dictionary  $\mathbf{U}$ . In the case where  $Kd < D$ ,  $\mathbf{U}$  is called *undercomplete*; when  $Kd = D$ , it is called *complete*; and when  $Kd > D$ , it is called *overcomplete*.

Now, (2.2.5) suggests a convenient relaxation for tractability of analysis: rather than modeling  $\mathbf{x}$  as coming from a mixture of  $K$  *specific* subspaces  $\mathbf{U}_1, \dots, \mathbf{U}_K$ , we may instead start with a dictionary  $\mathbf{U} \in \mathbb{R}^{D \times m}$ , where  $m$  may be smaller or larger than  $D$ , and simply seek to represent  $\mathbf{x} = \mathbf{U}\mathbf{z}$  with the sparsity  $\|\mathbf{z}\|_0$  sufficiently small. This leads to the *sparse dictionary model* for  $\mathbf{x}$ :

$$\mathbf{x} = \mathbf{U}\mathbf{z} + \boldsymbol{\varepsilon}, \quad \|\mathbf{z}\|_0 \ll d, \quad (2.2.7)$$

where  $\boldsymbol{\varepsilon}$  represents an unknown noise vector. Geometrically, this implies that  $\mathbf{x}$  lies close to the span of a subset of  $\|\mathbf{z}\|_0$  columns of  $\mathbf{U}$ , making this an instantiation of the mixture-of-subspaces model (2.2.2) with a very large value of  $K$ , and specific correlations between the subspaces  $\mathbf{U}_k$ .

**Orthogonal dictionary for sparse coding.** Now we can formulate the structured representation learning problem for mixtures of low-dimensional subspaces that we will study in this section. We assume that we have samples  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  from an unknown sparse dictionary model (2.2.7), possibly with added noises  $\boldsymbol{\varepsilon}_i$ . Let us begin from the assumption that the dictionary  $\mathbf{U}$  in the sparse dictionary model (2.2.7) is complete and orthogonal,<sup>6</sup> and that each coefficient vector  $\mathbf{z}$  is *d-sparse*, with  $d \ll D$ . In this setting, representation learning amounts to correctly learning the orthogonal dictionary  $\mathbf{U}$  via optimization: we can then take  $\mathcal{E}(\mathbf{x}) = \mathbf{U}^\top \mathbf{x}$  as the encoder and  $\mathcal{D}(\mathbf{z}) = \mathbf{U}\mathbf{z}$  as the decoder, and  $\mathcal{D} = \mathcal{E}^{-1}$ . In diagram form:

$$\mathbf{x} \xrightarrow{\mathcal{E}=\mathbf{U}^\top} \mathbf{z} \xrightarrow{\mathcal{D}=\mathbf{U}} \hat{\mathbf{x}}. \quad (2.2.8)$$

We see that the autoencoding pair  $(\mathcal{E}, \mathcal{D})$  for complete dictionary learning is symmetric, as in the case of a single linear subspace, making the computational task of encoding and decoding no more difficult than in the linear case. On the other hand, the task of learning the dictionary  $\mathbf{U}$  is strictly more difficult than learning a single linear subspace by PCA. To see why we cannot simply use PCA to learn the orthogonal dictionary  $\mathbf{U}$  correctly, note that the loss function that gave rise to PCA, namely (2.1.5), is completely invariant to rotations of the rows of the matrix  $\mathbf{U}$ : that is, if  $\mathbf{Q}$  is any  $d \times d$  orthogonal matrix, then  $\mathbf{U}$  and  $\mathbf{U}\mathbf{Q}$  are both feasible and have an identical loss for (2.1.5). The sparse dictionary model is decidedly not invariant to such transformations: if we replaced  $\mathbf{U}$  by  $\mathbf{U}\mathbf{Q}$  and made a corresponding rotation  $\mathbf{Q}^\top \mathbf{z}$  of the representation coefficients  $\mathbf{z}$ , we would in general destroy the sparsity structure of  $\mathbf{z}$ , violating the modeling assumption. Thus, we need new algorithms for learning orthogonal dictionaries.

---

<sup>6</sup>It can be shown that for the complete case, we do not lose any generality by making the orthogonal assumption (Exercise 2.4).

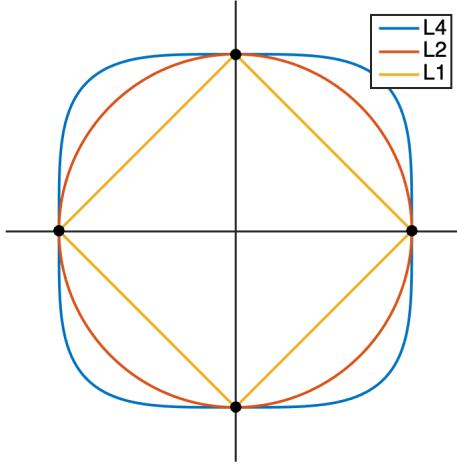


Figure 2.5: Maximizing  $\ell^4$  norm or minimizing  $\ell^1$  norm promotes sparsity (for vectors on the sphere).

### 2.2.2 Complete Dictionary Learning

In this section, we will derive algorithms for solving the orthogonal dictionary learning problem. To be more precise, we assume that the observed vector  $\mathbf{x} \in \mathbb{R}^D$  follows a statistical model

$$\mathbf{x} = \mathbf{U}\mathbf{z} + \boldsymbol{\varepsilon}, \quad (2.2.9)$$

where  $\mathbf{U} \in \mathbb{R}^{D \times D}$  is an unknown orthogonal dictionary,  $\mathbf{z}$  is a random vector with statistically independent components  $z_i$ , each with zero mean, and  $\boldsymbol{\varepsilon} \in \mathbb{R}^D$  is an independent random vector of small (Gaussian) noises. The goal is to recover  $\mathbf{U}$  (and hence  $\mathbf{z}$ ) from samples of  $\mathbf{x}$ .

Here we assume that each independent component  $z_i$  is distributed as

$$z_i \sim \text{Bern}(\theta) \cdot \mathcal{N}(0, 1/\theta).$$

That is, it is the product of a Bernoulli random variable with probability  $\theta$  of being 1 and  $1 - \theta$  of being 0, and an independent Gaussian random variable with variance  $1/\theta$ . This distribution is formally known as the *Bernoulli-Gaussian* distribution. The normalization is chosen so that  $\text{Var}(z_i) = 1$  and hence  $\mathbb{E}[\|\mathbf{z}\|_2^2] = d$ . This modeling assumption implies that the vector of independent components  $\mathbf{z}$  is typically very sparse: we calculate  $\mathbb{E}[\|\mathbf{z}\|_0] = d\theta$ , which is small when  $\theta$  is inversely proportional to  $d$ .

*Remark 2.7* (The Orthogonal Assumption). At first sight, the assumption that the dictionary  $\mathbf{U}$  is orthogonal might seem to be somewhat restrictive. But there is actually no loss of generality. One may consider a complete dictionary to be any square invertible matrix  $\mathbf{U}$ . With samples generated from this dictionary:  $\mathbf{X} = \mathbf{U}\mathbf{Z} \in \mathbb{R}^{D \times N}$ , it is easy to show that with some preconditioning of the data matrix  $\mathbf{X}$ :

$$\bar{\mathbf{X}} = \left( \frac{1}{N\theta} \mathbf{X} \mathbf{X}^\top \right)^{-\frac{1}{2}} \mathbf{X}, \quad (2.2.10)$$

then there exists an orthogonal matrix  $\mathbf{U}_o \in \text{O}(D)$  such that

$$\bar{\mathbf{X}} = \mathbf{U}_o \mathbf{Z}. \quad (2.2.11)$$

See Exercise 2.4 or [SQW17b] for more details.

**Dictionary learning via the MSP algorithm.** Now suppose that we are given a set of observations:

$$\mathbf{x}_i = \mathbf{U}\mathbf{z}_i + \boldsymbol{\varepsilon}_i, \quad \forall i \in [N]. \quad (2.2.12)$$

Let  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  and  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$ . The goal is to recover  $\mathbf{U}$  from the data  $\mathbf{X}$ . Therefore, given any orthogonal matrix  $\mathbf{A} \in \mathrm{O}(D)$ ,

$$\mathbf{A}\mathbf{x}_i = \mathbf{A}\mathbf{U}\mathbf{z}_i + \mathbf{A}\boldsymbol{\varepsilon}_i \quad (2.2.13)$$

would be nearly sparse if  $\mathbf{A} = \mathbf{U}^T$  (as by assumption the noise  $\boldsymbol{\varepsilon}_i$  is of small magnitude).

Also, given  $\mathbf{U}$  is orthogonal and the fact  $\boldsymbol{\varepsilon}$  is small, the vector  $\mathbf{x}$  has a predictable expected norm, i.e.,  $\mathbb{E}[\|\mathbf{x}\|_2^2] \approx \mathbb{E}[\|\mathbf{z}\|_2^2] = d$ . It is a known fact that for vectors on a sphere, maximizing the  $\ell^4$  norm is equivalent to minimizing the  $\ell^0$  norm (for promoting sparsity),

$$\arg \max_{\mathbf{z} \in \mathbb{S}^n} \|\mathbf{z}\|_4 \Leftrightarrow \arg \min_{\mathbf{z} \in \mathbb{S}^n} \|\mathbf{z}\|_0. \quad (2.2.14)$$

This is illustrated in Figure 2.5.

An orthogonal matrix  $\mathbf{A}$  preserves the Euclidean ( $\ell^2$ ) norm:  $\|\mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{x}\|_2^2$ . Therefore, to find the correct orthogonal dictionary  $\mathbf{U}$  from  $\mathbf{X}$ , we may try to solve the following optimization program:

$$\max_{\tilde{\mathbf{A}} \in \mathrm{O}(D)} \frac{1}{4} \|\tilde{\mathbf{A}}\mathbf{X}\|_4^4 = \frac{1}{4} \sum_{i=1}^N \|\tilde{\mathbf{A}}\mathbf{x}_i\|_4^4 \quad (2.2.15)$$

This is known as the  $\ell^4$  maximization problem [ZMZ+20]. After we find the solution  $\mathbf{A}^*$ , we can take the transpose  $\mathbf{U}^* = (\mathbf{A}^*)^\top$

*Remark 2.8.* It is also known that for vectors on a sphere, minimizing the  $\ell^1$  norm is equivalent to minimizing the  $\ell^0$  norm (for promoting sparsity),

$$\arg \min_{\mathbf{z} \in \mathbb{S}^n} \|\mathbf{z}\|_1 \Leftrightarrow \arg \min_{\mathbf{z} \in \mathbb{S}^n} \|\mathbf{z}\|_0,$$

which is also illustrated in Figure 2.5. This fact can also be exploited to learn the dictionary  $\mathbf{A}$  effectively and efficiently. This was actually explored earlier than the  $\ell^4$  norm used here. Interested readers may refer to the work of [QZL+20b].

Note that the above problem is equivalent to the following constrained optimization problem:

$$\min -\frac{1}{4} \|\tilde{\mathbf{A}}\mathbf{X}\|_4^4 \quad \text{subject to} \quad \tilde{\mathbf{A}}^\top \tilde{\mathbf{A}} = \mathbf{I}. \quad (2.2.16)$$

As shown in [WM22], using the Lagrange multiplier method, one can derive that the optimal solution to the problem should satisfy the following “fixed point” condition:

$$\mathbf{A}^* = \mathcal{P}_{\mathrm{O}(D)}[(\mathbf{A}^*\mathbf{X})^{\odot 3}\mathbf{X}^\top], \quad (2.2.17)$$

where  $\mathcal{P}_{\mathrm{O}(D)}[\cdot]$  is a projection onto the space of orthogonal matrices  $\mathrm{O}(D)$ .<sup>7</sup>

To compute the fixed point for the above equation, similar to how we computed eigenvectors for PCA (2.1.16), we may take the following power iteration:

$$\mathbf{A}_{t+1} = \mathcal{P}_{\mathrm{O}(D)}[(\mathbf{A}_t\mathbf{X})^{\odot 3}\mathbf{X}^\top]. \quad (2.2.18)$$

This is known as the *matching, stretching, and projection* (MSP) algorithm proposed by [ZMZ+20]. It was shown that under broad conditions such a greedy algorithm indeed converges to the correct solution at a superlinear rate.

*Remark 2.9* (Global Optimality of  $\ell^4$  Maximization). The constrained  $\ell^4$  maximization problem is a non-convex program. In general one should *not* expect that any greedy (say gradient-descent type) algorithms would converge to the globally optimal solution. Surprisingly, one can show that, unlike general nonconvex programs, the landscape of  $\ell^4$  maximization over a sphere

$$\min -\frac{1}{4} \|\mathbf{q}^\top \mathbf{X}\|_4^4 \quad \text{subject to} \quad \mathbf{q}^\top \mathbf{q} = 1. \quad (2.2.19)$$

is very benign: All local minima are close to the global optima and all critical points are saddle points with a direction of negative curvature. Hence, any descent method with the ability of escaping strict saddle points provably finds global optimal solutions. For more precise statements, interested readers may refer to [QZL+20a].

<sup>7</sup>For any matrix  $\mathbf{M}$  with SVD  $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$ ,  $\mathcal{P}_{\mathrm{O}(D)}[\mathbf{M}] = \mathbf{U}\mathbf{V}^\top$ . We leave this as an exercise for the reader.

*Remark 2.10* (Stable Deep Linear Network). The above iterative process of computing the dictionary has a natural incremental “deep learning” interpretation. Let us define  $\delta \mathbf{A}_{t+1} = \mathbf{A}_{t+1} \mathbf{A}_t^\top$  and  $\mathbf{Z}_t = \mathbf{A}_t \mathbf{X}$ , then it is easy to show that

$$\delta \mathbf{A}_{t+1} = \mathcal{P}_{\mathcal{O}(D)}[(\mathbf{Z}_t)^{\otimes 3} \mathbf{Z}_t^\top].$$

If  $\mathbf{A}_t$  converges to the correct dictionary  $\mathbf{D}_o$ , then the above iterative encoding process is essentially equivalent to a “deep linear network”:

$$\mathbf{Z} \leftarrow \mathbf{Z}_{t+1} = \underbrace{\delta \mathbf{A}_{t+1} \delta \mathbf{A}_t \dots \delta \mathbf{A}_1}_{\text{forward constructed layers}} \mathbf{X}.$$

Note that the computation of the increment transforms  $\delta \mathbf{A}_{t+1}$  at each layer depends only on the feature output from the previous layer  $\mathbf{Z}_t$ . The network is naturally stable as each layer is a norm-preserving orthogonal transform. Despite its resemblance to a linear deep network, backpropagation is unnecessary to learn each layer. All layers are learned in one forward pass!

### 2.2.3 Connection to ICA and Kurtosis

With the Bernoulli-Gaussian model, the variables  $z_i$  are independent and non-Gaussian. Then, there is a clear correspondence between the dictionary learning and the classic independent component analysis (ICA), to the extent that algorithms to solve one problem can be used to solve the other.<sup>8</sup>

Towards deriving an algorithm based on ICA, we focus on an objective function known as *kurtosis*, which is used in ICA as a direct consequence of the non-Gaussianity of the components. The *kurtosis*, or fourth-order cumulant, of a zero-mean random variable  $X$  is defined as

$$\text{kurt}(X) = \mathbb{E} X^4 - 3(\mathbb{E} X^2)^2. \quad (2.2.20)$$

If we have only finite samples from the random variable  $X$  arranged into a vector  $\mathbf{x} = [x_1, \dots, x_N]$ , we define kurtosis through their empirical average, which yields

$$\text{kurt}(\mathbf{x}) = \frac{1}{N} \|\mathbf{x}\|_4^4 - \frac{3}{N^2} \|\mathbf{x}\|_2^4. \quad (2.2.21)$$

Finally, for random vectors, we define their kurtosis as the sum of each component’s scalar kurtosis. Kurtosis is a natural loss function for ICA because for Gaussian  $X$ , kurtosis is zero; the reader can verify further that the Bernoulli-Gaussian distribution has positive kurtosis. Thus a natural procedure for seeking non-Gaussian independent components is to search for a set of mutually-orthogonal directions  $\mathbf{V} \in \mathbb{R}^{d \times k}$  such that  $\mathbf{V}^\top \mathbf{X}$  has maximal kurtosis, where  $\mathbf{X} = \mathbf{U} \mathbf{Z} \in \mathbb{R}^{D \times N}$  is the Bernoulli-Gaussian ICA data matrix. Formally, we seek to solve the problem

$$\max_{\mathbf{V}^\top \mathbf{V} = \mathbf{I}} \text{kurt}(\mathbf{V}^\top \mathbf{X}). \quad (2.2.22)$$

At one extreme, we may set  $k = D$  and seek to recover the entire dictionary  $\mathbf{U}$  in a single shot. It can be shown that this problem can be solved with the MSP algorithm we have seen previously. At the other extreme, we may set  $k = 1$  and seek to recover a single direction (column of  $\mathbf{U}$ ) at a time, performing *deflation*, i.e., replacing the data matrix  $\mathbf{X}$  by  $(\mathbf{I} - \mathbf{u}\mathbf{u}^\top)\mathbf{X}$ , after each step before finding another direction. There is a natural tradeoff between the scalability of the  $k = 1$  incremental approach and the efficiency and robustness of the  $k = D$  approach.

**Incremental ICA: correctness and FastICA algorithm.** The FastICA algorithm, advanced by Hyvärinen and Oja [HO97], is a fast fixed-point algorithm for solving the  $k = 1$  kurtosis maximization scheme for ICA. The problem at hand is

$$\max_{\|\mathbf{v}\|_2^2=1} \text{kurt}(\mathbf{X}^\top \mathbf{v}). \quad (2.2.23)$$

<sup>8</sup>We explore this issue in more depth in Exercise 2.3, where a connection between non-Gaussianity of the independent components and the purely geometric notion of symmetry is made. This issue is related to our observation above that PCA does not work for recovering sparsely-used orthogonal dictionaries: in the statistical setting, it can be related to rotational invariance of the Gaussian distribution (Exercise 2.2).

First, we will perform some very basic analysis of this objective to verify its correctness. Notice by the change of variables  $\mathbf{w} = \mathbf{U}^\top \mathbf{v}$  that this problem is equivalent to

$$\max_{\|\mathbf{w}\|_2^2=1} \text{kurt}(\mathbf{Z}^\top \mathbf{w}).$$

This objective is simple enough that we can make strong statements about its correctness as a formulation for recovering the dictionary  $\mathbf{U}$ . For example, in the population setting where  $N \rightarrow \infty$ , we may use additivity properties of the kurtosis (Exercise 2.5) and our assumed normalization on the independent components to write the previous problem equivalently as

$$\max_{\|\mathbf{w}\|_2^2=1} \sum_{i=1}^d \text{kurt}(z_i) w_i^4. \quad (2.2.24)$$

It can be shown that under the Bernoulli-Gaussian assumption, the optimization landscape of this problem is “benign” (Exercise 2.7)—meaning that all local maxima of the objective function correspond to the recovery of one of the independent components. One efficient and scalable way to compute one of these maxima is via first-order optimization algorithms, which iteratively follow the gradient of the objective function and project onto the constraint set  $\{\mathbf{w} \mid \|\mathbf{w}\|_2^2 = 1\}$ . Since we have assumed that each  $z_i$  satisfies  $\text{Var}(z_i) = 1$ , we have for large  $N$

$$\text{kurt}(\mathbf{X}^\top \mathbf{u}) \approx \frac{1}{N} \|\mathbf{X}^\top \mathbf{u}\|_4^4 - 3\|\mathbf{u}\|_2^4. \quad (2.2.25)$$

We can then derive a corresponding approximation to the gradient:

$$\nabla_{\mathbf{u}} \text{kurt}(\mathbf{X}^\top \mathbf{u}) \approx \frac{4}{N} \mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} - 12\|\mathbf{u}\|_2^2 \mathbf{u}.$$

The FastICA algorithm uses a fixed-point method to compute directions of maximum kurtosis. It starts from the first-order optimality conditions for the kurtosis maximization problem, given the preceding gradient approximation and the constraint set, which read

$$\mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} = \underbrace{\langle \mathbf{u}, \mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} \rangle}_{\lambda} \mathbf{u}, \quad (2.2.26)$$

where the specific value of  $\lambda$  is determined using the unit norm constraint on  $\mathbf{u}$ . Exercise 2.6 describes the mathematical background necessary to derive these optimality conditions from first principles. Equation (2.2.26) is satisfied by *any* critical point of the kurtosis maximization problem; we want to derive an equation satisfied by only the maximizers. After noticing that  $\lambda = \|\mathbf{X}^\top \mathbf{u}\|_4^4$ , we equivalently re-express (2.2.26) as the modified equation

$$\frac{1}{N} \mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} - 3\mathbf{u} = \left( \frac{\lambda}{N} - 3 \right) \mathbf{u}, \quad (2.2.27)$$

and realize that any maximizer of (2.2.23) must satisfy  $\lambda/N - 3 > 0$ , assuming that  $N$  is sufficiently large. Hence, we may *normalize* both sides of (2.2.27), giving the following fixed-point equation satisfied by every maximizer of (2.2.23):

$$\frac{\frac{1}{N} \mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} - 3\mathbf{u}}{\left\| \frac{1}{N} \mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} - 3\mathbf{u} \right\|_2} = \mathbf{u}. \quad (2.2.28)$$

Iterating the mapping defined by the lefthand side of this fixed point expression then yields the FastICA algorithm of Hyvärinen and Oja [HO97]:

$$\begin{aligned} \mathbf{v}^+ &= \frac{1}{N} \mathbf{X} (\mathbf{X}^\top \mathbf{u})^{\odot 3} - 3\mathbf{u}, \\ \mathbf{u}^+ &= \mathbf{v}^+ / \|\mathbf{v}^+\|_2. \end{aligned} \quad (2.2.29)$$

It turns out that the FastICA algorithm converges extremely rapidly (actually at a *cubic* rate) to columns of the dictionary  $\mathbf{U}$ ; interested readers may consult [HO97] for details. Comparing the FastICA algorithm (2.2.29) to the power method studied in 2.1.1 for the PCA problem and the MSP algorithm (2.2.18), we notice a striking similarity. Indeed, FastICA is essentially a modified power method, involving the gradient of the empirical kurtosis rather than the simpler linear gradient of the PCA objective.

## 2.3 A Mixture of Overcomplete Low-Dimensional Subspaces

As we have seen, complete dictionary learning enjoys an elegant computational theory in which we maintain a symmetric autoencoding structure  $\mathcal{E}(\mathbf{x}) = \mathbf{U}^\top \mathbf{x}$ ,  $\mathcal{D}(\mathbf{z}) = \mathbf{U}\mathbf{z}$ , with a scalable power-method-like algorithm (the MSP algorithm) for learning an orthogonal dictionary/codebook  $\mathbf{U}$  from data. Nevertheless, for learning representations of general high-dimensional data distributions, one must expand the size of the codebook beyond the orthogonality requirement—meaning that we must have  $\mathbf{A} \in \mathbb{R}^{D \times m}$ , with  $m \gg D$ , corresponding to the case of an *overcomplete* dictionary/codebook,<sup>9</sup> and the signal model

$$\mathbf{x} = \mathbf{A}\mathbf{z} + \boldsymbol{\varepsilon}, \quad \|\mathbf{z}\|_0 = d \ll m. \quad (2.3.1)$$

There are both geometric and physical/modeling motivations for passing to the overcomplete case. Geometrically, recall that in our original reduction from the mixture of subspace data model to the sparse dictionary model, a mixture of  $K$  subspaces in  $\mathbb{R}^D$ , each of dimension  $d$ , led to a dictionary of shape  $\mathbf{A} \in \mathbb{R}^{D \times Kd}$ . In other words, overcomplete dictionaries correspond to *richer* mixtures of subspaces, with more distinct modes of variability for modeling the high-dimensional data distribution. On the modeling side, we may run a computational experiment on real-world data that reveals the additional modeling power conferred by an overcomplete representation.

*Example 2.1.* Given sampled images of hand-written digits, Figure 2.6(a) shows the result of fitting an orthogonal dictionary to the dataset. In contrast, Figure 2.6(b) shows the result of running an optimization algorithm for learning overcomplete dictionaries (which we will present in detail later in the Chapter) on these samples. Notice that the representations become far sparser and the codebooks far more interpretable—they consist of fundamental primitives for the strokes composing the digits, i.e. oriented edges. ■

In fact, overcomplete dictionary learning was originally proposed as a biologically plausible algorithm for image representation based on empirical evidence of how early stages of the visual cortex represent visual stimuli [OF97; OF96].

In the remainder of this section, we will overview the conceptual and computational foundations of overcomplete dictionary learning. Supposing that the model (2.3.1) is satisfied with sparse codes  $\mathbf{z}$ , overcomplete dictionary  $\mathbf{A}$ , and sparsity level  $d$ , and given samples  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  of  $\mathbf{x}$ , we want to learn an encoder  $\mathcal{E} : \mathbb{R}^D \rightarrow \mathbb{R}^m$  mapping each  $\mathbf{x}$  to its sparse code  $\mathbf{z}$ , and a decoder  $\mathcal{D}(\mathbf{z}) = \mathbf{A}\mathbf{z}$  reconstructing each  $\mathbf{x}$  from its sparse code. In diagram form:

$$\mathbf{x} \xrightarrow{\mathcal{E}} \mathbf{z} \xrightarrow{\mathcal{D}=\mathbf{A}} \hat{\mathbf{x}}. \quad (2.3.2)$$

We will start from the construction of the encoder  $\mathcal{E}$ . We will work incrementally: first, *given the true dictionary  $\mathbf{A}$* , we will show how the problem of *sparse coding* gives an elegant, scalable, and provably-correct algorithm for recovering the sparse code  $\mathbf{z}$  of  $\mathbf{x}$ . Although this problem is NP-hard in the worst case, it can be solved efficiently and scalably for dictionaries  $\mathbf{A}$  which are *incoherent*, i.e. having columns that are not too correlated. The encoder architecture encompassed by this solution will no longer be symmetric: we will see it has the form of a primitive deep network, which depends on the dictionary  $\mathbf{A}$ .

Then we will proceed to the task of learning the decoder  $\mathcal{D}$ , or equivalently the overcomplete dictionary  $\mathbf{A}$ . We will derive an algorithm for overcomplete dictionary learning that allows us to simultaneously learn the codebook  $\mathbf{A}$  and the sparse codes  $\mathbf{z}$ , using ideas from sparse coding. Finally, we will discuss a more modern perspective on learnable sparse coding that leads us to a fully asymmetric encoder-decoder structure, as an alternative to (2.3.2). Here, the decoder will correspond to an incremental solution to the sparse dictionary learning problem, and yield a pair of deep network-like encoder decoders for sparse dictionary learning. This structure will foreshadow many developments to come in the remainder of the monograph, as we progress from analytical models to modern neural networks.

### 2.3.1 Sparse Coding with an Overcomplete Dictionary

In this section, we will consider the data model (2.3.1), which accommodates sparse linear combinations of many motifs, or *atoms*. Given data  $\{\mathbf{x}_i\}_{i=1}^N \subseteq \mathbb{R}^D$  satisfying this model, i.e. expressible as

$$\mathbf{x}_i = \mathbf{A}\mathbf{z}_i + \boldsymbol{\varepsilon}_i, \quad \forall i \in [N] \quad (2.3.3)$$

---

<sup>9</sup>We change the notation here from  $\mathbf{U}$  to  $\mathbf{A}$  in order to emphasize the non-orthogonality and non-square-shape of the overcomplete dictionary  $\mathbf{A}$ .

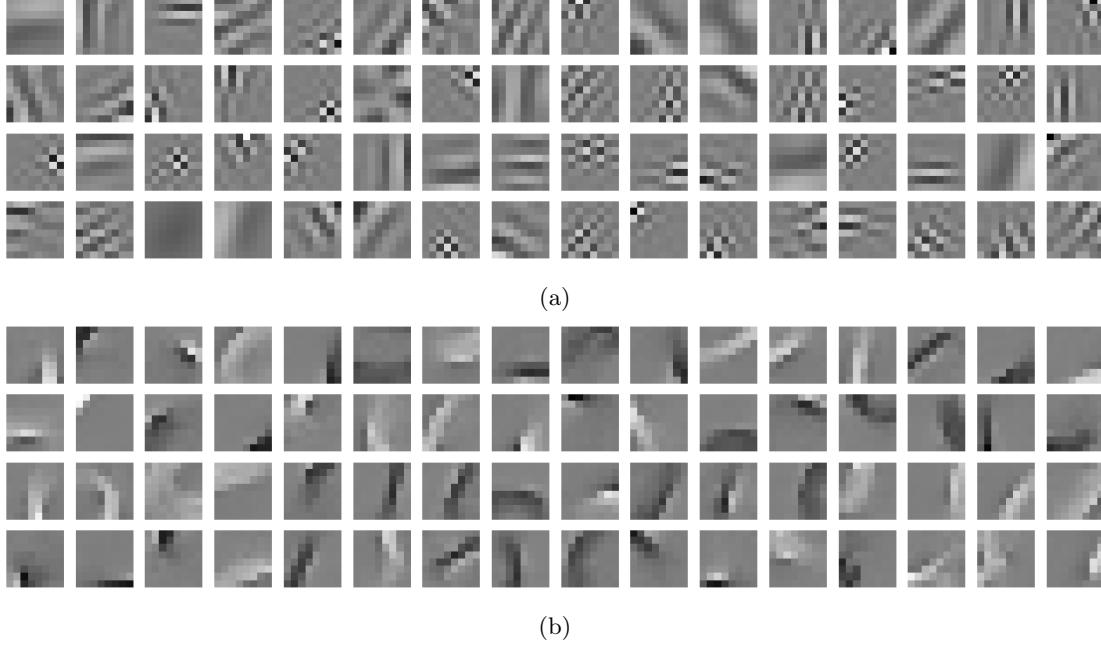


Figure 2.6: Comparison of learned dictionary atoms for complete (orthogonal) and overcomplete dictionaries, trained to reconstruct 8 by 8 patches taken from MNIST digits. Both dictionaries are trained for 6000 epochs on  $10^4$  random patches with nontrivial content, and sparse codes are computed with the LASSO objective and  $\lambda = 0.1$  (see (2.3.5)). Colormaps have black for negative values, and white for positive values. **Top:** An orthogonal dictionary learned with the MSP algorithm (2.2.18) is constrained to have no more than 64 atoms; the learned atoms roughly correspond to a “spike and slab” dictionary, and achieve relatively poor reconstruction sparsity levels on held-out test data (codes are approximately 17-sparse on average, with respect to a threshold of  $10^{-1}$ ). **Bottom:** In contrast, an overcomplete dictionary (here, with  $8^3$  atoms; we visualize a random subset of 64) learns semantically-meaningful dictionary atoms corresponding to signed oriented edges, which can be pieced together to create digit patches and achieve superior reconstruction and sparsity levels. Codes are approximately 20-sparse on average, while being 8 times larger than those of the orthogonal dictionary. To compute the dictionary, we use an optimizer based on proximal alternating linearized minimization on a suitably-regularized version of (2.3.17).

for some dictionary  $\mathbf{A} \in \mathbb{R}^{D \times m}$  with  $m$  atoms, sparse codes  $\mathbf{z}_i$  such that  $\|\mathbf{z}_i\|_0 \leq d$ , and small errors  $\varepsilon_i$ , the sparse coding problem is to recover the codes  $\mathbf{z}_i$  as accurately as possible from the data  $\mathbf{x}_i$ , given the dictionary  $\mathbf{A}$ . Efficient algorithms to solve this problem succeed when the dictionary  $\mathbf{A}$  is *incoherent* in the sense that the inner products  $\mathbf{a}_i^\top \mathbf{a}_j$  are uniformly small, hence the atoms are nearly orthogonal.<sup>10</sup>

Note that we can collect the  $\mathbf{x}_i$  into  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ , collect the  $\mathbf{z}_i$  into  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N] \in \mathbb{R}^{d \times N}$ , and collect the  $\varepsilon_i$  into  $\mathbf{E} = [\varepsilon_1, \dots, \varepsilon_N] \in \mathbb{R}^{D \times N}$ , to rewrite (2.3.3) as

$$\mathbf{X} = \mathbf{A}\mathbf{Z} + \mathbf{E}. \quad (2.3.4)$$

A natural approach to solving the sparse coding problem is to seek the sparsest signals that are consistent with our observations, and this naturally leads to the following optimization problem:

$$\min_{\mathbf{Z} \in \mathbb{R}^{d \times N}} \{\|\mathbf{X} - \mathbf{A}\mathbf{Z}\|_F^2 + \lambda\|\mathbf{Z}\|_1\}, \quad (2.3.5)$$

where the  $\ell^1$  norm  $\|\mathbf{Z}\|_1$ , taken elementwise, is known to promote sparsity of the solution [WM22]. This optimization problem is known as the LASSO problem.

<sup>10</sup>As it turns out, in a high-dimensional space, it is rather easy to pack a number of nearly orthogonal vectors that is much larger than the ambient dimension [WM22].

However, unlike PCA or the complete dictionary learning problem, there is no clear power iteration-type algorithm to recover  $\mathbf{Z}^*$ . A natural alternative is to consider solving the above optimization problem with gradient descent type algorithms. Let  $f(\mathbf{Z}) = \|\mathbf{X} - \mathbf{A}\mathbf{Z}\|_2^2 + \lambda\|\mathbf{Z}\|_1$ . Conceptually, we could try to find  $\mathbf{Z}^*$  with the following iterations:

$$\mathbf{Z}_{t+1} \leftarrow \mathbf{Z}_t + \eta \nabla f(\mathbf{Z}_t). \quad (2.3.6)$$

However, because the term associated with the  $\ell^1$  norm  $\|\mathbf{Z}\|_1$  is non-smooth, we cannot just run gradient descent. For this type of function, we need to replace the gradient  $\nabla f(\mathbf{Z})$  with something that generalizes the notion of gradient, known as the subgradient  $\partial f(\mathbf{Z})$ :

$$\mathbf{Z}_{t+1} \leftarrow \mathbf{Z}_t + \eta \partial f(\mathbf{Z}_t). \quad (2.3.7)$$

However, it is known that the convergence of subgradient descent is usually very slow. Hence, for this type of optimization problems, it is conventional to adopt a so-called *proximal gradient descent*-type algorithm. We give a technical overview of this method in Section A.1.3.

Applying proximal gradient to the LASSO objective function (2.3.5) leads to the classic *iterative shrinkage-thresholding algorithm* (ISTA), which implements the iteration

$$\mathbf{Z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (2.3.8)$$

$$\mathbf{Z}_{t+1} = S_{\eta\lambda}(\mathbf{Z}_t - 2\eta\mathbf{A}^\top(\mathbf{A}\mathbf{Z}_t - \mathbf{X})), \quad \forall t \geq 1, \quad (2.3.9)$$

with step size  $\eta \geq 0$ , and the soft-thresholding operator  $S_\alpha$  defined on scalars by

$$S_\alpha(x) \doteq \begin{cases} x - \alpha, & x \geq \alpha, \\ 0, & -\alpha < x < \alpha, \\ x + \alpha, & x \leq -\alpha \end{cases} \quad (2.3.10)$$

$$= \text{sign}(x) \max\{|x| - \alpha, 0\}, \quad (2.3.11)$$

and applied element-wise to the input matrix. As a proximal gradient descent algorithm applied to a convex problem, convergence to a global optimum is assured, and a precise convergence rate can be derived straightforwardly [WM22].

We now take a moment to remark on the functional form of the update operator in (2.3.9). It takes the form

$$\mathbf{Z}_{t+1} = \text{nonlinearity}(\mathbf{Z}_t + \text{linear}^\top(\text{linear}(\mathbf{Z}_t) + \text{bias})). \quad (2.3.12)$$

This functional form is very similar to that of a residual network layer, namely,

$$\mathbf{Z}_{t+1} = \mathbf{Z}_t + \text{linear}_1^\top(\text{nonlinearity}(\text{linear}_2(\mathbf{Z}_t) + \text{bias}_1) + \text{bias}_2), \quad (2.3.13)$$

only decoupling the linear maps (i.e., matrix multiplications), adding a bias, and moving the nonlinearity. The nonlinearity in (2.3.9) is notably similar to the commonly-used ReLU activation function  $x \mapsto \max\{x, 0\}$  in deep learning—in particular, the soft-thresholding operator is like a “signed” ReLU activation, which is also shifted by a bias. The ISTA, then, can be viewed as a forward pass of a primitive (recurrent one-layer) neural network. We argue in Chapter 4 that such operations are essential to deep representation learning.

### 2.3.2 Overcomplete Dictionary Learning

Recall that we have the data model

$$\mathbf{X} = \mathbf{A}\mathbf{Z} + \mathbf{E}, \quad (2.3.14)$$

where  $\mathbf{Z}$  is sparse, and our goal previously was to estimate  $\mathbf{Z}$  given knowledge of the data  $\mathbf{X}$  and the dictionary atoms  $\mathbf{A}$ . Now we turn to the more practical and more difficult case where we do not know either  $\mathbf{A}$  or  $\mathbf{Z}$  and seek to learn them from a large dataset.

A direct generalization of Equation (2.3.5) suggests solving the problem

$$\min_{\tilde{\mathbf{A}}, \tilde{\mathbf{Z}}} \left\{ \|\mathbf{X} - \tilde{\mathbf{A}}\tilde{\mathbf{Z}}\|_F^2 + \lambda\|\tilde{\mathbf{Z}}\|_1 \right\}. \quad (2.3.15)$$

However, the bilinear term in Equation (2.3.15) introduces a scale ambiguity that hinders convergence: given any point  $(\tilde{\mathbf{A}}, \tilde{\mathbf{Z}})$  and any constant  $c > 0$ , the substitution  $(c^{-1}\tilde{\mathbf{A}}, c\tilde{\mathbf{Z}})$  gives loss value

$$\|\mathbf{X} - \tilde{\mathbf{A}}\tilde{\mathbf{Z}}\|_F^2 + c\lambda\|\tilde{\mathbf{Z}}\|_1. \quad (2.3.16)$$

This loss is evidently minimized over  $c$  by taking  $c \rightarrow 0$ , which corresponds to making the rescaled dictionary  $c^{-1}\tilde{\mathbf{A}}$  go ‘to infinity’. In particular, the iterates of any optimization algorithm solving (2.3.15) will not converge.

This issue with (2.3.15) is dealt with by adding a constraint on the scale of the columns of the dictionary  $\tilde{\mathbf{A}}$ . For example, it is common to assume that each column  $\mathbf{A}_j$  of the dictionary  $\mathbf{A}$  in (2.3.14) has bounded  $\ell^2$  norm—say, without loss of generality, by 1. We then enforce this as a constraint, giving the objective

$$\min_{\tilde{\mathbf{Z}}, \tilde{\mathbf{A}} : \|\mathbf{A}_j\|_2 \leq 1} \left\{ \|\mathbf{X} - \tilde{\mathbf{A}}\tilde{\mathbf{Z}}\|_F^2 + \lambda\|\tilde{\mathbf{Z}}\|_1 \right\}. \quad (2.3.17)$$

Constrained optimization problems such as (2.3.17) can be solved by a host of sophisticated algorithms [NW06]. However, a simple and scalable one is actually furnished by the same proximal gradient descent algorithm that we used to solve the sparse coding problem in the previous section. We can encode each constraint as additional regularization term, via the characteristic function for the constraint set—details are given in Example A.1. Applying proximal gradient descent to the resulting regularized problem is equivalent to *projected gradient descent*, in which, at each iteration, the iterates after taking a gradient descent step are projected onto the constraint set.

*Remark 2.11* ( $\ell^4$  maximization versus  $\ell^1$  minimization). Note that the above problem formulation follows naturally from the LASSO formulation (2.3.5) for sparse coding. We promote the sparsity of the solution via the  $\ell^1$  norm. Nevertheless, if we are only interested in recovering the over-complete dictionary  $\mathbf{A}$ , the  $\ell^4$  maximization scheme introduced in Section 2.2.2 also generalizes to the over-complete case, without any significant modification. Interested readers may refer to the work of [QZL+20a].

The above problem (2.3.17), which we call *overcomplete dictionary learning*, is nonconvex as here both  $\mathbf{A}$  and  $\mathbf{Z}$  are unknown. It cannot be solved easily by the standard convex optimization toolkit. Nevertheless, because it is interesting, simple to state, and practically important, there have been many important works dedicated to different algorithms and theoretical analysis for this problem. Here, for the interest of this manuscript, we present an idiomatic approach to solve this problem which is closer to the spirit of deep learning.

From our experience with the LASSO problem above, it is easy to see that, for the two unknowns  $\mathbf{A}$  and  $\mathbf{Z}$ , if we fix one and optimize the other, each subproblem is in fact convex and easy to solve. This naturally suggests that we could attempt to solve the above program (2.3.17) by minimizing against  $\mathbf{Z}$  or  $\mathbf{A}$  alternatively, say using gradient descent. Coupled with a natural choice of initialization, this leads to the following iterative scheme:

$$\mathbf{Z}^{\ell+1} = S_{\eta\lambda}(\mathbf{Z}^\ell - 2\eta\mathbf{A}_+^\top(\mathbf{A}_+\mathbf{Z}^\ell - \mathbf{X})), \quad \mathbf{Z}^1 = \mathbf{0}, \quad \forall \ell \in [L] \quad (2.3.18)$$

$$\mathbf{Z}^+ = \mathbf{Z}^L, \quad (2.3.19)$$

$$\mathbf{A}_{t+1} = \text{proj}_{\|\cdot\|_2 \leq 1, \forall j} (\mathbf{A}_t - 2\nu(\mathbf{A}_t\mathbf{Z}^+ - \mathbf{X})(\mathbf{Z}^+)^{\top}), \quad (\mathbf{A}_1)_j \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \frac{1}{D}\mathbf{I}), \quad \forall j \in [m], \quad \forall t \in [T], \quad (2.3.20)$$

$$\mathbf{A}_+ = \mathbf{A}_T, \quad (2.3.21)$$

where the projection operation in the update for  $\mathbf{A}$  ensures each column has at most unit  $\ell^2$  norm, via  $\mathbf{A}_j \mapsto \mathbf{A}_j / \max\{\|\mathbf{A}_j\|_2, 1\}$ , and where  $\mathbf{A}_+$  is initialized with each column i.i.d.  $\mathcal{N}(\mathbf{0}, \frac{1}{D}\mathbf{I})$ . The above consists of one ‘block’ of alternating minimization, and we repeatedly perform such blocks, each with independent initializations, until convergence. Above, we have used two separate indices  $\{t\}$  and  $\{\ell\}$  to indicate the iterations. As we will see later, this allows us to interpret the two updates separately in the context of deep learning.

Despite the dictionary learning problem being a nonconvex problem, it has been shown that alternating minimization type algorithms indeed converge to the correct solution, at least locally. See, for example, [AAJ+16]. As a practical demonstration, the above algorithm (with  $L = T = 1$ ) was used to generate the results for overcomplete dictionary learning in Figure 2.6.

### 2.3.3 Learned Deep Sparse Coding

The main insight from the alternating minimization algorithm for overcomplete dictionary learning in the previous section (Equations (2.3.18) and (2.3.20)) is to notice that *when we fix  $\mathbf{A}$ , the ISTA update for  $\mathbf{Z}^\ell$  (2.3.18) looks like the forward pass of a deep neural network with weights given by  $\mathbf{A}$  (and  $\mathbf{A}^\top$ )*. But in general, we do not know the true  $\mathbf{A}$ , and the current estimate  $\mathbf{A}_+$  could be erroneous. Hence it needs to be further updated using (2.3.20) based on the residual of using the current estimate of the sparse codes  $\mathbf{Z}^+$  to reconstruct  $\mathbf{X}$ . The alternating minimization algorithm iterates these two procedures until convergence. But we can instead extrapolate, and design other learning procedures by combining these insights with techniques from deep learning. This leads to more interpretable network architectures, which will be a recurring theme throughout this manuscript.

**Learned ISTA.** The above deep-network interpretation of the alternating minimization is more conceptual than practical, as the process could be rather inefficient and take many layers or iterations to converge. But this is mainly because we try to infer both  $\mathbf{Z}$  and  $\mathbf{A}$  from  $\mathbf{X}$ . The problem can be significantly simplified and the above iterations can be made much more efficient in the *supervised* setting, where we have a dataset of input and output pairs  $(\mathbf{X}, \mathbf{Z})$  distributed according to (2.3.14) and we only seek to learn  $\mathbf{A}^\ell$  for the layerwise learnable sparse coding iterations (2.3.29):

$$\mathbf{Z}^{\ell+1} = S_{\eta\lambda}(\mathbf{Z}^\ell - 2\eta(\mathbf{A}^\ell)^\top(\mathbf{A}^\ell \mathbf{Z}^\ell - \mathbf{X})), \quad \forall \ell \in [L]. \quad (2.3.22)$$

If we denote the operator for each iteration as  $\mathbf{Z}^{\ell+1} = f(\mathbf{A}^\ell, \mathbf{Z}^\ell)$ , the above iteration can be illustrated in terms of a diagram:

$$\mathbf{X}, \mathbf{Z}^1 \xrightarrow{f(\mathbf{A}^1, \cdot)} \mathbf{Z}^2 \xrightarrow{f(\mathbf{A}^2, \cdot)} \mathbf{Z}^3 \xrightarrow{f(\mathbf{A}^3, \cdot)} \dots \mathbf{Z}^L \xrightarrow{f(\mathbf{A}^L, \cdot)} \mathbf{Z}^{L+1} \approx \mathbf{Z}.$$

Thus, given the sequential architecture, to learn the operator  $\mathbf{A}^\ell$  at each layer, it is completely natural to learn it, say via back propagation (BP),<sup>11</sup> by minimizing the error between the final code  $\mathbf{Z}^L$  and the ground truth  $\mathbf{Z}$ :

$$\min_{\{\mathbf{A}^\ell\}} \|\mathbf{Z}^L(\mathbf{A}^1, \dots, \mathbf{A}^L) - \mathbf{Z}\|_2^2. \quad (2.3.23)$$

This is the basis of the Learned ISTA (LISTA) algorithm [GL10], which can be viewed as the learning algorithm for a deep neural network, which tries to emulate the sparse encoding process from  $\mathbf{X}$  to  $\mathbf{Z}$ . In particular, it can be viewed as a *simple representation learning algorithm*. In fact, this same methodology can be used as a basis to understand the representations computed in more powerful network architectures, such as transformers. We develop these implications in detail in Chapter 4.

**Sparse Autoencoders.** The original motivation for overcomplete dictionary learning was to provide a simple generative model for high-dimensional data. We have seen with LISTA that, in addition, iterative algorithms for learning sparsely-used overcomplete dictionaries provide an interpretation for ReLU-like deep networks, which we will generalize in later chapters to more complex data distributions than (2.3.14). But it is also worth noting that even in the modern era of large models, the data generating model (2.3.14) provides a useful practical basis for *interpreting features in pretrained large-scale deep networks*, such as transformers, following the hypothesis that the (non-interpretable, *a priori*) features in these networks consist of sparse “superpositions” of underlying features, which are themselves interpretable [EHO+22b]. These *unsupervised* learning paradigms are generally more data friendly than LISTA, as well, which requires large amounts of labeled  $(\mathbf{X}, \mathbf{Z})$  pairs for supervised training.

We can use our development of the LISTA algorithm above to understand common practices in this field of research. In the most straightforward instantiation (see [GTT+25; HCS+24]), a large number of features from a pretrained deep network  $h$  are collected from different inputs  $\mathbf{x}_i$ , which themselves are chosen based on a desired interpretation task.<sup>12</sup> For simplicity, we will use  $h$  to denote the pre-selected feature map in

<sup>11</sup>See Appendix A.2 for a brief description of BP.

<sup>12</sup>For example, the inputs  $\mathbf{x}_i$  could correspond to texts containing samples of computer code in different programming languages, with our task being to try to identify interpretable features in a transformer feature map  $h$  corresponding to different salient aspects of the input, such as the specific programming language (distinct across input “classes”) or the need to insert a matching parenthesis at the current position (common across input “classes”). We discuss the use of deep networks, and in particular transformers, for text representation learning in greater detail in Chapter 7.

question, with  $D$ -dimensional features; given  $N$  sample inputs, let  $\mathbf{H} \in \mathbb{R}^{D \times N}$  denote the full matrix of features of  $h$ . Then a so-called sparse autoencoder  $f : \mathbb{R}^D \rightarrow \mathbb{R}^d$ , with decoder  $g : \mathbb{R}^d \rightarrow \mathbb{R}^D$ , is trained via the LASSO loss (2.3.5):

$$\min_{f,g} \|\mathbf{H} - g(f(\mathbf{H}))\|_F^2 + \lambda \|f(\mathbf{H})\|_1, \quad (2.3.24)$$

where the sparse autoencoder  $f$  takes the form of a one-layer neural network, i.e.  $f(\mathbf{h}_i) = \sigma(\mathbf{W}_{\text{enc}}(\mathbf{h}_i - \mathbf{b}) + \mathbf{b}_{\text{enc}})$ , where  $\sigma(x) = \max\{x, 0\}$  is the ReLU activation function, and the decoder  $g$  is linear, so that  $g(\mathbf{z}_i) = \mathbf{W}_{\text{dec}}\mathbf{z} + \mathbf{b}$ .

The parameterization and training procedure (2.3.24) may initially seem to be an arbitrary application of deep learning to the sparse coding problem, but it is actually highly aligned with the algorithms we have studied above for layerwise sparse coding with a learned dictionary. In particular, recall the LISTA architecture  $\mathbf{Z}^L = f(\mathbf{A}^L, f(\mathbf{A}^{L-1}, \dots, f(\mathbf{A}^1, \mathbf{X}) \dots))$ . In the special case  $L = 2$ , we have

$$\mathbf{Z}^2 = f(\mathbf{A}^1, \mathbf{X}) = S_{\eta\lambda}(\mathbf{Z}^1 - 2\eta(\mathbf{A}^1)^\top(\mathbf{A}^1\mathbf{Z}^1 - \mathbf{X})). \quad (2.3.25)$$

Let us assume that the sparse codes  $\mathbf{Z}$  in question are nonnegative, i.e., that  $\mathbf{Z} \geq \mathbf{0}$ .<sup>13</sup> Then (see Example A.3), we can consider an equivalent LISTA architecture obtained from the sparse coding objective with an additional nonnegativity constraint on  $\mathbf{Z}$  as

$$\mathbf{Z}^2 = f(\mathbf{A}^1, \mathbf{X}) = \max \{\mathbf{Z}^1 - 2\eta(\mathbf{A}^1)^\top(\mathbf{A}^1\mathbf{Z}^1 - \mathbf{X}) - \lambda\eta\mathbf{1}, 0\}, \quad (2.3.26)$$

and after some algebra, express this as

$$\mathbf{Z}^2 = f(\mathbf{A}^1, \mathbf{X}) = \max \{2\eta(\mathbf{A}^1)^\top + (\mathbf{Z}^1 - 2\eta(\mathbf{A}^1)^\top\mathbf{A}^1\mathbf{Z}^1 - \lambda\eta\mathbf{1}), 0\}. \quad (2.3.27)$$

Given the ability to change the sparse code initialization  $\mathbf{Z}^1$  as a learnable parameter (which, in the current framework, must have all columns equal to the same learnable vector), this has the form of a ReLU neural network with learnable bias—identical to the sparse autoencoder  $f$ ! Moreover, to *decode* the learned sparse codes  $\mathbf{Z}^2$ , it is natural to apply the learned dictionary  $\mathbf{Z}^2 \mapsto \mathbf{A}^1\mathbf{Z}^2$ . Then the only difference between this and the SAE decoder  $g$  is the additional bias  $\mathbf{b}$ , which can technically be absorbed into  $\mathbf{H}$  and  $f$  in the training objective (2.3.24).

Thus, the SAE parameterization and training procedure coincides with LISTA training with  $L = 1$ , and a modified training objective—using the LASSO objective (2.3.5), which remains *unsupervised*, instead of the supervised reconstruction loss (2.3.23) used in vanilla LISTA. In particular, we can understand the SAE architecture in terms of our interpretation of the LISTA architecture in terms of layerwise sparse coding in (2.3.29). This connection is suggestive of a host of new design strategies for improving practical interpretability methodology, many of which remain tantalizingly unexplored. We begin to lay out some connections to broader autoencoding methodology in Chapter 5.

**Layerwise learned sparse coding?** In the supervised setting, LISTA provides a deep neural network analogue of the sparse coding iteration, with layerwise-learned dictionaries, inspired by alternating minimization; even in the unsupervised setting, the same methodology can be applied to learning, as with sparse autoencoders. But the connection between low-dimensional-structure-seeking optimization algorithms and deep network architectures goes much deeper than this, and suggests an array of scalable and natural neural learning architectures which may even be usable without backpropagation.

As a simple illustration, we return the alternating minimization iterations (2.3.18) and (2.3.20). This scheme randomly re-initializes the dictionary  $\mathbf{A}_1$  on every such update. An improvement uses instead *warm starting*, where the residual is generated using the previous estimate  $\mathbf{A}_+$  for the dictionary. If we then view each ISTA update (2.3.18) as a layer and allow the associated dictionary, now coupled with the sparse code updates as  $\mathbf{A}_\ell$ , to update in time, this leads to a “layerwise-learnable” sparse coding scheme:

$$\mathbf{Z}^1 = \mathbf{0}, \quad (\mathbf{A}_1)_j \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \frac{1}{D}\mathbf{I}), \quad \forall j \in [m], \quad (2.3.28)$$

$$\mathbf{Z}^{\ell+1} = S_{\eta\lambda}(\mathbf{Z}^\ell - 2\eta(\mathbf{A}_\ell)^\top(\mathbf{A}_\ell\mathbf{Z}^\ell - \mathbf{X})), \quad (2.3.29)$$

---

<sup>13</sup>In the data generating model (2.3.14), an arbitrary dictionary-and-sparse-code pair  $(\mathbf{A}, \mathbf{Z})$  can be replaced by one in which  $\mathbf{Z} \geq \mathbf{0}$  simply by doubling the number of columns in  $\mathbf{A}$ , so from a modeling perspective, this is a very mild assumption.

$$\mathbf{A}_{\ell+1} = \mathbf{A}_\ell - 2\nu(\mathbf{A}_\ell \mathbf{Z}^{\ell+1} - \mathbf{X})(\mathbf{Z}^{\ell+1})^\top. \quad (2.3.30)$$

Note that this iteration corresponds to a relabeling of (2.3.18) and (2.3.20) for  $T = L = 1$ , over infinitely many blocks. Each of the ‘inner’ steps updating  $\mathbf{Z}$  can be considered as a one-layer forward pass, while each of the ‘outer’ steps updating  $\mathbf{A}$  can be considered as a one-layer backward pass, of a primitive deep neural network. In particular, this algorithm is the simplest case in which a clear divide between forward optimization and backward learning manifests. This divide is still observed in current neural networks and autoencoders—we will have much more to say about it in Chapter 4 and in Chapter 5.

Notice that the above layer-wise scheme also suggests a plausible alternative to the current end-to-end optimization strategy that primarily relies on back propagation (BP) detailed in the Appendix A.2.3. Freeing training large networks from BP would be one of the biggest challenges and opportunities in the future, as we will discuss more at the end of the book in Chapter 8.

## 2.4 Summary and Notes

The idealistic models we have presented in this chapter—PCA, ICA, and dictionary learning—were developed over the course of the twentieth century. Many books have been written solely about each method, so we will only attempt here to give a broad overview of the key works and history.

Jolliffe [Jol86] attributes principal component analysis to Pearson [Pea01], and independently Hotelling [Hot33]. In mathematics, the main result on the related problem of low-rank approximation in unitarily invariant norms is attributed to Eckart and Young [EY36], and to Mirsky for full generality [Mir60]. PCA continues to play an important role in research as perhaps the simplest model problem for unsupervised representation learning: as early as the 1980s, works such as Oja [Oja82] and Baldi and Hornik [BH89] used the problem to understand learning in primitive neural networks, and more recently, it has served as a tool for understanding more complex representation learning frameworks, such as diffusion models [WZZ+24].

Independent component analysis was proposed by B. Ans et al. [BJC85] and pioneered by Aapo Hyvärinen in the 1990s and early 2000s in a series of influential works: see Hyvärinen and Oja [HO00b] for a summary. As a simple model for structure that arises in practical data, it initially saw significant use in applications such as blind source separation, where each independent component  $z_i$  represents an independent source (such as sound associated to a distinct instrument in a musical recording) that is superimposed to produce the observation  $\mathbf{x} = \mathbf{U}\mathbf{z}$ .

The problem of dictionary learning can, in the complete or orthogonal case, be seen as one of the foundational problems of twentieth-century signal processing, particularly in linear systems theory, where the Fourier basis plays the key role; from the 1980s onward, the field of computational harmonic analysis crystallized around the study of alternate such dictionaries for classes of signals in which optimal approximation could only be realized in a basis other than Fourier (e.g., wavelets) [DVD+98]. However, the importance of the case of redundant bases, or overcomplete dictionaries, was only highlighted following the pioneering work of Olshausen and Field [OF97; OF96]. Early subsequent work established the conceptual and algorithmic foundations for learning sparsely-used overcomplete dictionaries, often aimed at representing natural images [AEB06; DM03; Don01; EA06; GJB15; MBP14; MK07]. Later, a significant amount of theoretical interest in the problem, as an important and nontrivial model problem for unsupervised representation learning, led to its study by the signal processing, theoretical machine learning, and theoretical computer science communities, in particular focused on conditions under which the problem could be provably and efficiently solved. A non-exhaustive list of notable works in this line include those of Spielman et al. [SWW12] and Sun et al. [SQW17a] on the complete case; Arora et al. [AGM+15]; Barak et al. [BKS15]; and Qu et al. [QZL+20a]. Many deep theoretical questions about this simple-to-state problem remain open, perhaps in part due to a tension with the problem’s worst-case NP-hardness (e.g., see Tillmann [Til15]).

One point that we wish to highlight from the study of these classical analytical models for low-dimensional structure is the common role played by various *generalized power methods*—algorithms that very rapidly converge, at least locally, to various types of low-dimensional structures. The terminology for this class of algorithms follows the work of M. Journée et al. [MYP+10]. At a high level, modeled on the classical power iteration for computation of the top eigenvector of a semidefinite matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , that is

$$\mathbf{u}_{t+1} = \frac{\mathbf{A}\mathbf{u}_t}{\|\mathbf{A}\mathbf{u}_t\|_2}, \quad (2.4.1)$$

Table 2.1: Summary of (generalized) power methods presented in the Chapter

Problem	Algorithm	Iteration	Type of Structure Enforced
PCA	Power Method	$\mathbf{u}_t = \frac{\mathbf{X}\mathbf{X}^\top \mathbf{u}_t}{\ \mathbf{X}\mathbf{X}^\top \mathbf{u}_t\ _2}$	1-dim. subspace (unit vector)
ICA	FastICA	$\mathbf{u}_{t+1} = \frac{\frac{1}{N}\mathbf{X}(\mathbf{X}^\top \mathbf{u}_t)^{\odot 3} - 3\mathbf{u}_t}{\left\  \frac{1}{N}\mathbf{X}(\mathbf{X}^\top \mathbf{u}_t)^{\odot 3} - 3\mathbf{u}_t \right\ _2}$	1-dim. subspace (unit vector)
Complete DL	MSP Algorithm	$\mathbf{U}_{t+1} = \mathcal{P}_{\mathcal{O}(D)}[(\mathbf{U}_t \mathbf{X})^{\odot 3} \mathbf{X}^\top]$	$D$ -dim. subspace (orthogonal matrix)

this class of algorithms consists of a “powering” operation involving a matrix  $\mathbf{A}$  associated to the data, along with a “projection” operation that enforces a desired type of structure. Table 2.1 presents a summary of the algorithms we have studied in this Chapter that follow this structure. The reader may appreciate the applicability of this methodology to different types of low-dimensional structure, and different losses (i.e., both the quadratic loss from PCA, and the kurtosis-type losses from ICA), as well as the lack of such an algorithm for overcomplete dictionary learning, despite the breadth of the literature on these algorithms. We see the development of power methods for further families of low-dimensional structures, particularly those relevant to applications where deep learning is prevalent, as one of the more important (and open) research questions suggested by this chapter.

The connection we make in Section 2.2.1 between the geometric mixture-of-subspaces distributional assumption and the more analytically-convenient sparse dictionary assumption has been mentioned in prior work, especially by those focused on generalized principal component analysis and applications such as subspace clustering, e.g. work of Vidal et al. [VMS16]. The mixture of subspaces assumption will continue to play a significant role throughout this manuscript, both as an analytical test case for different algorithmic paradigms, and as a foundation for deriving different deep network architectures, as with LISTA in Section 2.3.3, but which can scale to more complex data distributions.

## 2.5 Exercises and Extensions

*Exercise 2.1.* Prove that, for any symmetric matrix  $\mathbf{A}$ , the solution to the problem  $\max_{\mathbf{U} \in \mathcal{O}(D,d)} \text{tr}(\mathbf{U}^\top \mathbf{A} \mathbf{U})$  is the matrix  $\mathbf{U}^*$  whose columns are the top  $d$  unit eigenvectors of  $\mathbf{A}$ .

*Exercise 2.2.* Let  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$  be a Gaussian random variable with independent components, each with variance  $\sigma^2$ . Prove that for any orthogonal matrix  $\mathbf{Q}$  (i.e.,  $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ ), the random variable  $\mathbf{Q}\mathbf{z}$  is distributed identically to  $\mathbf{z}$ . (*Hint: recall the formula for the Gaussian probability density function, and the formula for the density of a linear function of a random variable.*)

*Exercise 2.3.* The notion of statistical identifiability discussed above can be related to *symmetries* of the model class, allowing estimation to be understood in a purely deterministic fashion without any statistical assumptions.

Consider the model  $\mathbf{X} = \mathbf{U}\mathbf{Z}$  for matrices  $\mathbf{X}, \mathbf{U}, \mathbf{Z}$  of compatible sizes.

1. Show that if  $\mathbf{A}$  is any square invertible matrix of compatible size, then the pair  $(\mathbf{U}\mathbf{A}, \mathbf{A}^{-1}\mathbf{Z})$  also equals  $\mathbf{X}$  under the model. We call this a *GL(d) symmetry*.
2. Suppose each column of  $\mathbf{Z}$  is an independent and identically distributed observation from a common statistical model  $\mathbf{z}$ , which moreover has zero mean and independent components  $z_i$  with positive variance. Show that for any square invertible matrix  $\mathbf{A}$ , if  $\mathbf{A}\mathbf{z}$  has uncorrelated components, then  $\mathbf{A}$  can be written as  $\mathbf{D}_1 \mathbf{Q} \mathbf{D}_2$ , where  $\mathbf{Q}$  is an orthogonal matrix and  $\mathbf{D}_1, \mathbf{D}_2$  are diagonal matrices. *This links the “independence” assumption in ICA to a “symmetry breaking” effect, which only allows scale and rotational symmetries.*

*Exercise 2.4.* Consider the model  $\mathbf{x} = \mathbf{U}\mathbf{z}$ , where  $\mathbf{U} \in \mathbb{R}^{D \times d}$  with  $D \geq d$  is fixed and has rank  $d$ , and  $\mathbf{z}$  is a zero-mean random variable. Let  $\mathbf{x}_1, \dots, \mathbf{x}_N$  denote i.i.d. observations from this model.

1. Show that the matrix  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  has rank no larger than  $d$ , and therefore there is an orthonormal matrix  $\mathbf{V} \in \mathbb{R}^{D \times d}$  so that  $\mathbf{X} = \mathbf{V}\mathbf{Y}$ , where  $\mathbf{Y} \in \mathbb{R}^{d \times N}$ . (*Hint: use PCA.*)
2. Show that the *whitened matrix*  $(\mathbf{Y}\mathbf{Y}^\top)^{-1/2}\mathbf{Y}$  exists in expectation whenever  $\text{Cov}(\mathbf{z})$  is nonsingular, and that it has identity empirical covariance.<sup>14</sup>
3. Show, by using the singular value decomposition of  $\mathbf{U}$ , that the matrix  $\mathbf{V}$  can be chosen so that the whitened matrix satisfies  $(\mathbf{Y}\mathbf{Y}^\top)^{-1/2}\mathbf{Y} = \mathbf{W}[\mathbf{z}_1, \dots, \mathbf{z}_N]$ , where  $\mathbf{W}$  is an orthonormal matrix.

*Exercise 2.5.* Let  $X$  and  $Y$  be zero-mean independent random variables.

1. Show that  $\text{kurt}(X + Y) = \text{kurt}(X) + \text{kurt}(Y)$ .
2. For any  $\alpha \in \mathbb{R}$ , show that  $\text{kurt}(\alpha X) = \alpha^4 \text{kurt}(X)$ .

*Exercise 2.6.* Let  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  be a given twice-continuously-differentiable objective function. Consider the spherically-constrained optimization problem

$$\max_{\|\mathbf{u}\|_2^2=1} f(\mathbf{u}). \quad (2.5.1)$$

In this exercise, we will derive the expressions we gave in the FastICA derivation for maximizing kurtosis over the sphere via a gradient ascent algorithm. These expressions are special cases of a rich theory of calculus and optimization on manifolds, of which the sphere is a particular example. A deep technical study of this field is out-of-scope for our purposes, so we only mention two key references for the interested reader: the pioneering textbook by Absil, Mahony, and Sepulchre [AMS09], and a more recent introductory treatise by Boumal [Bou23].

1. For any constraint set  $\mathcal{M}$  that is a differentiable submanifold of  $\mathbb{R}^d$ , the *tangent space* at a point  $\mathbf{u} \in \mathcal{M}$  is, informally, the best local linear approximation to the manifold  $\mathcal{M}$  at the point  $\mathbf{u}$ . In the important special case where  $\mathcal{M}$  is defined locally at  $\mathbf{u}$  as a level set of a function  $F : \mathbb{R}^d \rightarrow \mathbb{R}$ , that is

$$U \cap \mathcal{M} = F^{-1}(\{0\})$$

for some open set  $U \subset \mathcal{M}$  with  $\mathbf{u} \in U$ , the tangent space to  $\mathcal{M}$  at  $\mathbf{u}$  can be calculated via differentiation:

$$T_{\mathbf{u}}\mathcal{M} = \text{Ker}(DF_{\mathbf{u}}).$$

It is easily seen that the sphere has the defining equation  $F(\mathbf{u}) = \|\mathbf{u}\|_2^2 - 1$ . Show, using these facts, that the tangent space to the sphere at  $\mathbf{u}$  is given by

$$T_{\mathbf{u}}\mathbb{S}^{d-1} = \{\mathbf{v} \in \mathbb{R}^d \mid \langle \mathbf{v}, \mathbf{u} \rangle = 0\},$$

and that the orthogonal projection onto this subspace is  $\mathbf{P}_{\mathbf{u}}^\perp = \mathbf{I} - \mathbf{u}\mathbf{u}^\top$ .

2. The vector field

$$\text{grad } f(\mathbf{u}) = \mathbf{P}_{\mathbf{u}}^\perp \nabla f \quad (2.5.2)$$

is known as the *Riemannian gradient* of the function  $f$  restricted to the sphere. The *first order optimality conditions* for the optimization problem (2.5.1) can be expressed in terms of the Riemann gradient:

$$\text{grad } f(\mathbf{u}) = \mathbf{0}.$$

Geometrically, this says that the Euclidean gradient of  $f$  at  $\mathbf{u}$  must be orthogonal to the tangent space to the sphere at  $\mathbf{u}$ . Now suppose  $\mathbf{v} \in \mathbb{R}^d$  is nonzero. Show that

$$\text{proj}_{\mathbb{S}^{d-1}}(\mathbf{v}) \doteq \min_{\|\mathbf{u}\|_2^2=1} \|\mathbf{u} - \mathbf{v}\|_2 = \frac{\mathbf{v}}{\|\mathbf{v}\|_2},$$

using the first-order optimality conditions.

---

<sup>14</sup>In particular, it can be proved mathematically that this is enough to guarantee that the whitened matrix exists with high probability whenever  $\mathbf{z}$  satisfies a suitable concentration inequality and  $N$  is sufficiently large.

3. In optimization over  $\mathbb{R}^d$ , one checks the second-order optimality conditions (to determine whether a critical point is a maximizer, a minimizer, or a saddle point) using the *Hessian matrix*  $\nabla^2 f(\mathbf{u})$ . Show, by differentiating the Riemann gradient  $\text{grad } f(\mathbf{u})$  for the sphere with respect to  $\mathbf{u}$  as in the first part of this exercise, that the corresponding object for determining second-order optimality conditions for sphere-constrained optimization is the *Riemannian Hessian*, defined as

$$\text{Hess } f(\mathbf{u}) = \mathbf{P}_{\mathbf{u}}^\perp (\nabla^2 f(\mathbf{u}) - \langle \nabla f(\mathbf{u}), \mathbf{u} \rangle \mathbf{I}) \mathbf{P}_{\mathbf{u}}^\perp. \quad (2.5.3)$$

*Exercise 2.7.* In this exercise, we sketch an argument referred to in the literature as a *landscape analysis* for the spherically-constrained population kurtosis maximization problem (2.2.24). We will show that when there is at least one independent component with positive kurtosis, its global maximizers indeed lead to the recovery of one column of the dictionary  $\mathbf{U}$ . For simplicity, we will assume that  $\text{kurt}(z_i) \neq 0$  for each  $i = 1, \dots, d$ .

1. Using the results of Part 1 of Exercise 2.6, show that the first-order optimality condition for (2.2.24) is

$$\left( \sum_{i=1}^d \text{kurt}(z_i) w_i^4 \right) \mathbf{w} = \text{kurt}(\mathbf{z}) \odot \mathbf{w}^{\odot 3}, \quad (2.5.4)$$

where the kurtosis is calculated elementwise,  $\odot$  denotes elementwise multiplication of vectors and  $\mathbf{w}^{\odot 3}$  denotes the elementwise cube of its argument.

2. Show that the vectors  $\mathbf{w}$  with unit norm that also satisfy (2.5.4) all take the following form. Let  $S^+ = \{i \in [d] \mid \text{kurt}(z_i) > 0\}$ , and  $S^- = \{i \in [d] \mid \text{kurt}(z_i) < 0\}$ . Let  $S$  be a subset either of  $S^+$  or  $S^-$ . Then

$$\mathbf{w}_S = \sum_{i \in S} \pm \sqrt{\frac{1}{\text{kurt}(z_i) \sum_{j \in S} \frac{1}{\text{kurt}(z_j)}}} \mathbf{e}_i \quad (2.5.5)$$

satisfies (2.5.4), where  $\mathbf{e}_i$  is the vector with a 1 in the  $i$ -th position and 0s elsewhere, and the  $\pm$  sign denotes the choice of either a positive or negative sign.

3. Assume that there is at least one  $i$  such that  $\text{kurt}(z_i) > 0$ . Using the results of Part 2 of Exercise 2.6, show that the only local maxima of the objective of (2.2.24) are the signed one-sparse vectors  $\pm \mathbf{e}_i$  with  $i \in S^+$ . Conclude that the global maximizers of (2.2.24) are the signed one-sparse vectors corresponding to components with maximum kurtosis. (*Hint: count the number of positive and negative eigenvalues of the Riemannian Hessian (2.5.3) at each critical point.*)
4. Now assume that  $\text{kurt}(z_j) < 0$  for every  $j = 1, \dots, d$ . This corresponds to an “over-deflated” instantiation of the kurtosis maximization problem. Using again the results of Part 2 of Exercise 2.6, show that the only local maxima of the objective of (2.2.24) are the signed dense vectors  $\sum_{i=1}^d \pm \mathbf{e}_i$ . This shows that the optimization formulation (2.2.24) cannot be applied naively.

*Exercise 2.8.* This exercise follows the structure and formalism introduced in Exercise 2.6, but applies it instead to the orthogonal group  $O(d) = \{\mathbf{U} \in \mathbb{R}^{d \times d} \mid \mathbf{U}^\top \mathbf{U} = \mathbf{I}\}$ . Consult the description of Exercise 2.6 for the necessary conceptual background; the formalism applies identically to the case where the ambient space is the set of  $d \times d$  matrices as long as one recalls that the relevant inner product on matrices is  $\langle \mathbf{X}, \mathbf{Y} \rangle = \text{tr}(\mathbf{X}^\top \mathbf{Y})$ . An excellent general reference for facts about optimization on the orthogonal group is Edelman, Arias, and Smith [EAS98].

Let  $f : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$  be a given twice-continuously-differentiable objective function. Consider the orthogonally-constrained optimization problem

$$\max_{\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}} f(\mathbf{Q}). \quad (2.5.6)$$

1. It is easily seen that the orthogonal group has the defining equation  $F(\mathbf{Q}) = \mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ . Show, using this fact, that the tangent space to the orthogonal group at  $\mathbf{Q}$  is given by

$$T_{\mathbf{Q}} O(d) = \{\mathbf{Q} \Omega \in \mathbb{R}^{d \times d} \mid \Omega^\top = -\Omega\},$$

and that the orthogonal projection onto this subspace is

$$\mathcal{P}_{T_{\mathbf{Q}} \mathcal{O}(d)}(\Delta) = \mathbf{Q} \text{skew}(\mathbf{Q}^\top \Delta),$$

where  $\text{skew}(\Delta) = \frac{1}{2}(\Delta - \Delta^\top)$  is the orthogonal projection onto the set of skew-symmetric matrices. The vector field

$$\text{grad } f(\mathbf{Q}) = \mathcal{P}_{T_{\mathbf{Q}} \mathcal{O}(d)}(\nabla f(\mathbf{Q})) \quad (2.5.7)$$

is known as the *Riemannian gradient* of the function  $f$  restricted to the orthogonal group. The *first order optimality conditions* for the optimization problem (2.5.6) can be expressed in terms of the Riemann gradient:

$$\text{grad } f(\mathbf{Q}) = \mathbf{0}.$$

2. Show, by differentiating the Riemann gradient  $\text{grad } f(\mathbf{Q})$  for the orthogonal group with respect to  $\mathbf{Q}$  as in the first part of this exercise, that the *Riemannian Hessian* is given by

$$\text{Hess } f(\mathbf{Q}) = \mathcal{P}_{T_{\mathbf{Q}} \mathcal{O}(d)}(\nabla^2 f(\mathbf{Q}) - \text{sym}(\mathbf{Q}^\top \nabla f(\mathbf{Q})) \otimes \mathbf{I}) \mathcal{P}_{T_{\mathbf{Q}} \mathcal{O}(d)}, \quad (2.5.8)$$

where  $\text{sym}(\Delta) = \frac{1}{2}(\Delta + \Delta^\top)$  denotes the orthogonal projection onto the set of symmetric matrices, and  $\otimes$  denotes the Kronecker product of matrices. Take care to interpret the operators appearing in the previous expression as *linear transformations on  $d \times d$  matrices*, **not** as  $d \times d$  matrices themselves. The *second-order optimality conditions* for the optimization problem (2.5.6) can be expressed in terms of the Riemann Hessian:

$$\text{Hess } f(\mathbf{Q}) \preceq \mathbf{0}.$$

For a minimization problem, the sign is reversed.

(Hint: The key is to manipulate one's calculations to obtain the form (2.5.8), which is as compact as possible. To this end, make use of the following isomorphism of the Kronecker product: if  $\mathbf{A}$ ,  $\mathbf{X}$ , and  $\mathbf{B}$  are matrices of compatible sizes, then one has

$$(\mathbf{B}^\top \otimes \mathbf{A}) \text{vec}(\mathbf{X}) = \text{vec}(\mathbf{AXB}),$$

where  $\text{vec}(\mathbf{X})$  denotes the “left-to-right” stacking of the columns of the matrix argument into a vector. We use this isomorphism in (2.5.8) in order to define the Kronecker product of two matrices as an operator on matrices in a canonical way.)

3. Now suppose  $\mathbf{X} \in \mathbb{R}^{d \times d}$  is full-rank. In this and the next part of the exercise, we consider the projection onto the orthogonal group of  $\mathbf{X}$ :

$$\text{proj}_{\mathcal{O}(d)}(\mathbf{X}) \doteq \min_{\mathbf{Q} \in \mathcal{O}(d)} \|\mathbf{Q} - \mathbf{X}\|_F^2. \quad (2.5.9)$$

We will prove that the solution to this problem is given by

$$\text{proj}_{\mathcal{O}(d)}(\mathbf{X}) = \mathbf{UV}^\top,$$

where  $\mathbf{X} = \mathbf{USV}^\top$  is a singular value decomposition of  $\mathbf{X}$ .

- (a) Using the first and second-order optimality conditions, show that every local minimizer  $\mathbf{Q}$  of (2.5.9) satisfies

$$\begin{aligned} (\mathbf{Q}^\top \mathbf{X})^\top &= \mathbf{Q}^\top \mathbf{X}, \\ \mathbf{Q}^\top \mathbf{X} &\succeq \mathbf{0}. \end{aligned}$$

(Hint: use linearity of the Kronecker product in either of its two arguments when the other is fixed.)

- (b) Using these conditions, argue that at every local minimizer  $\mathbf{Q}$  of (2.5.9), one has  $\mathbf{Q}^\top \mathbf{X} = (\mathbf{X}^\top \mathbf{X})^{1/2}$ . (Hint: Use the fact from linear algebra that if  $\mathbf{S} \succeq \mathbf{0}$  is a symmetric positive semidefinite matrix, then  $(\mathbf{S}^\top \mathbf{S})^{1/2} = \mathbf{S}$ .)

- (c) Using the singular value decomposition  $\mathbf{X} = \mathbf{USV}^\top$ , conclude that

$$\mathbf{UV}^\top = \text{proj}_{\mathcal{O}(d)}(\mathbf{X}).$$

## Chapter 3

# Pursuing Low-Dimensional Distributions via Lossy Compression

“We compress to learn, and we learn to compress.”

— High-dimensional Data Analysis, Wright and Ma, 2022

In Chapter 2, we have shown how to learn simple classes of distributions whose supports are assumed to be either a single or a mixture of low-dimensional subspaces or low-rank Gaussians. For further simplicity, the different (hidden) linear or Gaussian modes are assumed to be orthogonal or independent<sup>1</sup>, as illustrated in Figure 2.4. As we have shown, for such special distributions, one can derive rather simple and effective learning algorithms with correctness and efficiency guarantees. The geometric and statistical interpretation of operations in the associated algorithms is also very clear.

In practice, both linearity and independence are rather idealistic assumptions that distributions of real-world high-dimensional data rarely satisfy. The only thing that we may assume is that the intrinsic dimension of the distribution is very low compared to the dimension of the ambient space in which the data are embedded. Hence, in this chapter, we show how to learn a more general class of low-dimensional distributions in a high-dimensional space that is not necessarily (piecewise) linear.

It is typical that the distribution of real data often contains multiple components or modes, say corresponding to different classes of objects in the case of images. These modes might not be statistically independent and they may even have different intrinsic dimensions. It is also typical that we have access to only a finite number of samples of the distribution. Therefore, in general, we may assume our data are distributed on a mixture of (nonlinear) low-dimensional submanifolds in a high-dimensional space. Figure 3.1 illustrates an example of such a distribution.

To learn such a distribution under such conditions, there are several fundamental questions that we need to address:

- What is a general approach to learn a general low-dimensional distribution in a high-dimensional space and represent the learned distribution?
- How do we measure the complexity of the resulting representation so that we can effectively exploit the low dimensionality to learn?
- How do we make the learning process computationally tractable and even scalable, as the ambient dimension is usually high and the number of samples typically large?

As we will see, the fundamental idea of *compression*, or *dimension reduction*, which has been shown to be very effective for the linear/independent case, still serves as a general principle for developing effective computational models and methods for learning general low-dimensional distributions.

---

<sup>1</sup>Or can be easily reduced to such idealistic cases.

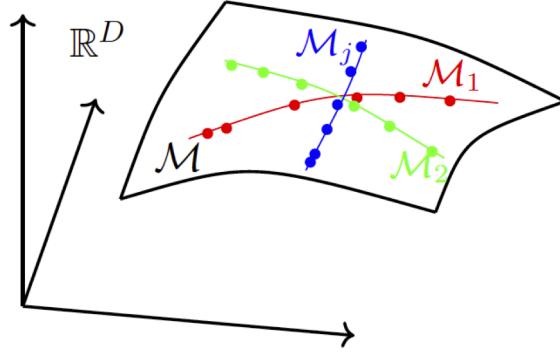


Figure 3.1: Data distributed on a mixture of low-dimensional submanifolds  $\cup_j \mathcal{M}_j$  in a very high-dimensional ambient space, say  $\mathbb{R}^D$ .

Due to its theoretical and practical significance, we will study in greater depth how this general framework of learning low-dimensional distributions via compression substantiates when the distribution of interest can be well-modeled or approximated by a mixture of low-dimensional subspaces or low-rank Gaussians.

## 3.1 Entropy Minimization and Compression

### 3.1.1 Entropy and Coding Rate

In Chapter 1, we have mentioned that the goal of learning is to find the simplest way to generate a given set of data. Conceptually, the Kolmogorov complexity was intended to provide such a measure of complexity but it is not computable and not associated with any implementable scheme that can actually reproduce the data. Hence we need an alternative, computable, and realizable, measure of complexity. That leads us to the notion of *entropy*, introduced by Shannon in 1948 [Sha48].

To illustrate the constructive nature of entropy, let us start with the simplest case. Suppose that we have a discrete random variable that takes  $N$  distinct values, or *tokens*,  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  with equal probability  $1/N$ . Then we could encode each token  $\mathbf{x}_i$  using the  $\log_2 N$ -bit binary representation of  $i$ . This coding scheme could be generalized to encoding arbitrary discrete distributions [CT91]: Given a distribution  $p$  such that  $\sum_{i=1}^N p(\mathbf{x}_i) = 1$ , one could assign each token  $\mathbf{x}_i$  with probability  $p(\mathbf{x}_i)$  to a binary code of size  $\log_2[1/p(\mathbf{x}_i)] = -\log_2 p(\mathbf{x}_i)$  bits. Hence the average number of bits, or the *coding rate*, needed to encode any sample from the distribution  $p(\cdot)$  is given by the expression:<sup>2</sup>

$$H(\mathbf{x}) \doteq \mathbb{E}[\log 1/p(\mathbf{x})] = - \sum_{i=1}^N p(\mathbf{x}_i) \log p(\mathbf{x}_i). \quad (3.1.1)$$

This is known as the *entropy* of the (discrete) distribution  $p(\cdot)$ . Note that this entropy is always nonnegative and it is zero if and only if  $p(\mathbf{x}_i) = 1$  for some  $\mathbf{x}_i$  with  $i \in [N]$ .<sup>3</sup>

### 3.1.2 Differential Entropy

When the random variable  $\mathbf{x} \in \mathbb{R}^D$  is continuous and has a probability density  $p$ , one may view that the limit of the above sum (3.1.1) is related to an integral:

$$h(\mathbf{x}) \doteq \mathbb{E}[\log 1/p(\mathbf{x})] = - \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log p(\boldsymbol{\xi}) d\boldsymbol{\xi}. \quad (3.1.2)$$

<sup>2</sup>By the convention of Information Theory [CT91], the log here is to the base 2. Hence entropy is measured in (binary) bits.

<sup>3</sup>Here notice that we use the fact  $\lim_{p \rightarrow 0} p \log p = 0$ .

More precisely, given a continuous variable  $\mathbf{x}$ , we may quantize it with a quantization size  $\epsilon > 0$ . Denote the resulting discrete variable as  $\mathbf{x}^\epsilon$ . Then one can show that  $H(\mathbf{x}^\epsilon) + \log(\epsilon) \approx h(\mathbf{x})$ . Hence, when  $\epsilon$  is small, the differential entropy  $h(\mathbf{x})$  can be negative. Interested readers may refer to [CT91] for a more detailed explanation.

*Example 3.1* (Entropy of Gaussian Distributions). Through direct calculation, it is possible to show that the entropy of a Gaussian distribution  $x \sim \mathcal{N}(\mu, \sigma^2)$  is given by:

$$h(x) = \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2}. \quad (3.1.3)$$

It is also known that the Gaussian distribution achieves the maximal entropy for all distributions with the same variance  $\sigma^2$ . The entropy of a multivariate Gaussian distribution  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  in  $\mathbb{R}^D$  is given by:

$$h(\mathbf{x}) = \frac{D}{2}(1 + \log(2\pi)) + \frac{1}{2} \log \det(\boldsymbol{\Sigma}). \quad (3.1.4)$$

■

Similar to the entropy for a discrete distribution, we would like the differential entropy to be associated with the coding rate of some realizable coding scheme. For example, as above, we may discretize the domain of the distribution with a grid of size  $\epsilon > 0$ . The coding rate of the resulting discrete distribution can be viewed as an approximation to the differential entropy [CT91].

Be aware that there are some caveats associated with the definition of differential entropy. For a distribution in a high-dimensional space, when its support becomes degenerate (low-dimensional), its differential entropy diverges to  $-\infty$ . This fact is proved in Theorem B.1 (we also recall the maximum entropy characterization of the Gaussian distribution mentioned above in Theorem B.1) but even in the simple explicit case of Gaussian distributions (3.1.4), when the covariance  $\boldsymbol{\Sigma}$  is singular, we can see that  $\log \det(\boldsymbol{\Sigma}) = -\infty$  so we have  $h(\mathbf{x}) = -\infty$ . In such a situation, it is not obvious how to properly quantize or encode such a distribution. Nevertheless, degenerate (Gaussian) distributions are precisely the simplest possible, and arguably the most important, instances of low-dimensional distributions in a high-dimensional space. In this chapter, we will discuss a complete resolution to this seeming difficulty with degeneracy.

### 3.1.3 Minimizing Coding Rate

Remember that the learning problem entails the recovery of a (potentially continuous) distribution  $p(\mathbf{x})$  from a set of samples  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  drawn from the distribution. For ease of exposition, we write  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ . Given that the distributions of interest here are (nearly) low-dimensional, we should expect that their (differential) entropy is very small. But unlike the situations that we have studied in the previous chapter, in general we do not know the family of (analytical) low-dimensional models to which the distribution  $p(\mathbf{x})$  belongs. So checking whether the entropy is small seems to be the only guideline that we can rely on to identify and model the distribution.

Now given the samples alone without knowing what  $p(\mathbf{x})$  is, in theory they could be interpreted as samples from any generic distribution. In particular, they could be interpreted as any of the following cases:

1. as samples from the empirical distribution  $p^{\mathbf{X}}$  itself, which assigns  $1/N$  probability each of the  $N$  samples  $\mathbf{x}_i, i = 1, \dots, N$ .
2. as samples from a standard normal distribution  $\mathbf{x}^n \sim p^n \doteq \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$  with a variance  $\sigma^2$  large enough (say larger than the sample norms);
3. as samples from a normal distribution  $\mathbf{x}^e \sim p^e \doteq \mathcal{N}(\mathbf{0}, \hat{\boldsymbol{\Sigma}})$  with a covariance  $\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \mathbf{X} \mathbf{X}^T$  being the empirical covariance of the samples;
4. as samples from a distribution  $\hat{\mathbf{x}} \sim \hat{q}(\mathbf{x})$  that closely approximates the ground truth distribution  $p$ .

Now the question is which one is better, in what sense? Suppose that you believe these data  $\mathbf{X}$  are drawn from a particular distribution  $q(\mathbf{x})$ , which may be one of the above distributions considered. Then we could

encode the data points with the optimal code book for the distribution  $q(\mathbf{x})$ . The required average coding length (or coding rate) is given by:

$$\frac{1}{N} \sum_{i=1}^N -\log q(\mathbf{x}_i) \approx - \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log q(\boldsymbol{\xi}) d\boldsymbol{\xi} \quad (3.1.5)$$

as the number of samples  $N$  becomes large. If we have identified the correct distribution  $p(\mathbf{x})$ , the coding rate is given by the entropy  $-\int p(\boldsymbol{\xi}) \log p(\boldsymbol{\xi}) d\boldsymbol{\xi}$ . It turns out that the above coding length  $-\int p(\boldsymbol{\xi}) \log q(\boldsymbol{\xi}) d\boldsymbol{\xi}$  is always larger than or equal to the entropy unless  $q(\mathbf{x}) = p(\mathbf{x})$ . Their difference, denoted as

$$\text{KL}(p \parallel q) \doteq - \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log q(\boldsymbol{\xi}) d\boldsymbol{\xi} - \left( - \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log p(\boldsymbol{\xi}) d\boldsymbol{\xi} \right) \quad (3.1.6)$$

$$= \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log \frac{p(\boldsymbol{\xi})}{q(\boldsymbol{\xi})} d\boldsymbol{\xi} \quad (3.1.7)$$

is known as the *Kullback-Leibler* (KL) divergence, or relative entropy. This quantity is always non-negative.

**Theorem 3.1** (Information Inequality). *Let  $p(\mathbf{x}), q(\mathbf{x})$  be two probability density functions (that have the same support). Then  $\text{KL}(p \parallel q) \geq 0$ , where the inequality becomes equality if and only if  $p = q$ .*<sup>4</sup>

*Proof.*

$$\begin{aligned} -\text{KL}(p \parallel q) &= - \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log \frac{p(\boldsymbol{\xi})}{q(\boldsymbol{\xi})} d\boldsymbol{\xi} = \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \log \frac{q(\boldsymbol{\xi})}{p(\boldsymbol{\xi})} d\boldsymbol{\xi} \\ &\leq \log \int_{\mathbb{R}^D} p(\boldsymbol{\xi}) \frac{q(\boldsymbol{\xi})}{p(\boldsymbol{\xi})} d\boldsymbol{\xi} = \log \int_{\mathbb{R}^D} q(\boldsymbol{\xi}) d\boldsymbol{\xi} = \log 1 = 0, \end{aligned}$$

where the first inequality follows from *Jensen's inequality* and the fact that the function  $\log(\cdot)$  is strictly concave. The equality holds if and only if  $p = q$ .  $\square$

Hence, given a set of sampled data  $\mathbf{X}$ , to determine which case is better among  $p^n$ ,  $p^e$ , and  $\hat{q}$ , we may compare their coding rates for  $\mathbf{X}$  and see which one gives the lowest rate. We know from the above that the (theoretically achievable) coding rate for a distribution is closely related to its entropy. In general, we have:

$$h(\mathbf{x}^n) > h(\mathbf{x}^e) > h(\hat{\mathbf{x}}). \quad (3.1.8)$$

Hence, if the data  $\mathbf{X}$  were encoded by the code book associated with each of these distributions, the coding rate for  $\mathbf{X}$  would in general decrease in the same order:

$$p(\mathbf{x}^n) \rightarrow p(\mathbf{x}^e) \rightarrow p(\hat{\mathbf{x}}). \quad (3.1.9)$$

This observation gives us a general guideline on how we may be able to pursue a distribution  $p(\mathbf{x})$  which has a low-dimensional structure. It suggests two possible approaches:

1. Starting with a general distribution (say a normal distribution) with high entropy, gradually transforming the distribution towards the (empirical) distribution of the data by reducing entropy.
2. Among a large family of (parametric or non-parametric) distributions with explicit coding schemes that encode the given data, progressively search for better coding schemes that give lower coding rates.

Conceptually, both approaches are essentially trying to do the same thing. For the first approach, we need to make sure such a path of transformation exists and is computable. For the second approach, it is necessary that the chosen family is rich enough and can closely approximate (or contain) the ground truth distribution. For either approach, we need to ensure that solutions with lower entropy or better coding rates can be efficiently computed and converge to the desired distribution quickly.<sup>5</sup> We will explore both approaches in the two remaining sections of this chapter.

<sup>4</sup>Technically, this equality should be taken to mean “almost everywhere”, i.e., except possibly on a set of zero measure (volume), since this set would not impact the value of any integral.

<sup>5</sup>Say the distribution of real-world data such as images and texts.

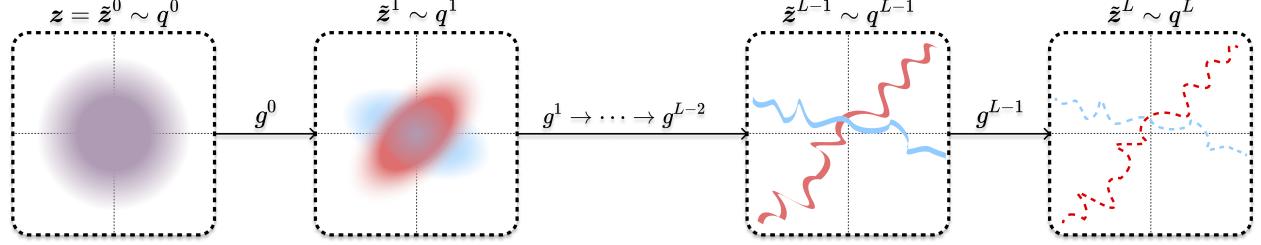


Figure 3.2: Illustration of an iterative denoising process that, starting from an isotropic Gaussian distribution, converges to an arbitrary data distribution.

## 3.2 Compression via Denoising

In this section, we will describe a *natural* and *computationally tractable* way to learn a distribution  $p(\mathbf{x})$  by way of learning a parametric encoding of our distribution such that the representation has the minimum entropy or coding rate, then using this encoding to transform high-entropy samples from a standard Gaussian into low-entropy samples from the target distribution, as illustrated in Figure 3.2. This presents a methodology that utilizes both approaches above in order to learn and sample from the distribution.

### 3.2.1 Diffusion and Denoising Processes

We first want to find a procedure to decrease the entropy of a given very noisy sample into a lower-entropy sample from the data distribution. Here, we describe a potential approach—one of many, but perhaps the most natural way to attack this problem. First, we find a way to *gradually increase* the entropy of existing samples from the data distribution. Then, we find an *approximate inverse* of this process. But in general, the operation of increasing entropy does not have an inverse, as information from the original distribution may be destroyed. We will thus tackle a special case where (1) the operation of adding entropy takes on a simple, computable, and reversible form; (2) we can obtain a (parametric) encoding of the data distribution, as alluded to in the above pair of approaches. As we will see, the above two factors will ensure that our approach is possible.

We will increase the entropy in arguably the simplest possible way, i.e., *adding isotropic Gaussian noise*. More precisely, given the random variable  $\mathbf{x}$ , we can consider the *stochastic process*  $(\mathbf{x}_t)_{t \in [0, T]}$  which adds gradual noise to it, i.e.,

$$\mathbf{x}_t \doteq \mathbf{x} + t\mathbf{g}, \quad \forall t \in [0, T], \quad (3.2.1)$$

where  $T \in [0, \infty]$  is a time horizon and  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is drawn independently of  $\mathbf{x}$ . This process is an example of a *diffusion process*, so-named because it spreads out the probability mass out over all of  $\mathbb{R}^D$  as time goes on, increasing the entropy over time. This intuition is confirmed graphically by Figure 3.3, and rigorously via the following theorem.

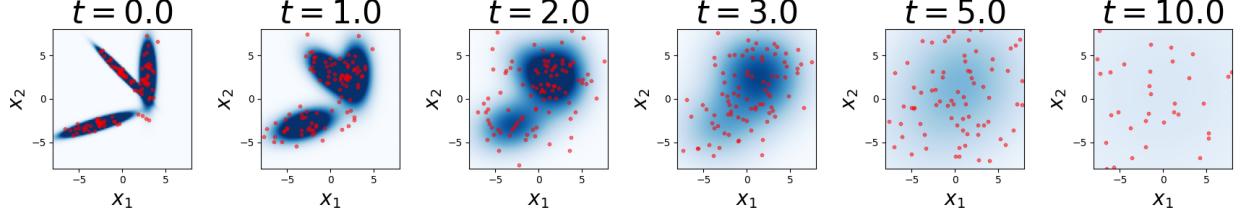
**Theorem 3.2** (Simplified Version of Theorem B.2). *Suppose that  $(\mathbf{x}_t)_{t \in [0, T]}$  follows the model (3.2.1). For any  $t \in (0, T]$ , the random variable  $\mathbf{x}_t$  has differential entropy  $h(\mathbf{x}_t) > -\infty$ . Moreover, under certain technical conditions on  $\mathbf{x}$ ,*

$$\frac{d}{dt} h(\mathbf{x}_t) > 0, \quad \forall t \in (0, T], \quad (3.2.2)$$

showing that the entropy of the noised  $\mathbf{x}$  increases over time  $t$ .

The proof is elementary, but it is rather long, so we postpone it to Section B.2.1. The main as-yet unstated implication of this result is that  $h(\mathbf{x}_t) > h(\mathbf{x})$  for every  $t > 0$ . To see this, note that if  $h(\mathbf{x}) = -\infty$  then  $h(\mathbf{x}_t) > -\infty$  for all  $t > 0$ , and if  $h(\mathbf{x}) > -\infty$  then  $h(\mathbf{x}_t) = h(\mathbf{x}) + \int_0^t [\frac{d}{ds} h(\mathbf{x}_s)] ds > h(\mathbf{x})$  by the fundamental theorem of calculus, so in both cases  $h(\mathbf{x}_t) > h(\mathbf{x})$  for every  $t > 0$ .

The inverse operation to adding noise is known as *denoising*. It is a classical and well-studied topic in signal processing and system theory, such as the Wiener filter and the Kalman filter. The several problems discussed in Chapter 2, such as PCA, ICA, and Dictionary Learning, are specific instances of the denoising



**Figure 3.3: Diffusing a mixture of Gaussians.** From left to right, we observe the evolution of the density as  $t$  grows from 0 to 10, along with some representative samples. Each region is colored by its density (0.0 is completely white,  $> 0.01$  is very dark blue, every other value maps to some shade of blue in between.) We observe that the probability mass gets less concentrated as  $t$  increases, signaling that entropy increases.

problem. For a fixed  $t$  and the additive Gaussian noise model (3.2.1), the denoising problem can be formulated as attempting to learn a function  $\bar{\mathbf{x}}^*(t, \cdot)$  which forms the best possible approximation (in expectation) of the true random variable  $\mathbf{x}$ , given both  $t$  and  $\mathbf{x}_t$ :

$$\bar{\mathbf{x}}^*(t, \cdot) \in \arg \min_{\bar{\mathbf{x}}(t, \cdot)} \mathbb{E}_{\mathbf{x}, \mathbf{x}_t} \|\mathbf{x} - \bar{\mathbf{x}}(t, \mathbf{x}_t)\|_2^2. \quad (3.2.3)$$

The solution to this problem, when optimizing  $\bar{\mathbf{x}}(t, \cdot)$  over all possible (square-integrable) functions, is the so-called *Bayes optimal denoiser*:

$$\bar{\mathbf{x}}^*(t, \xi) \doteq \mathbb{E}[\mathbf{x} \mid \mathbf{x}_t = \xi]. \quad (3.2.4)$$

This expression justifies the notation  $\bar{\mathbf{x}}$ , which is meant to compute a conditional expectation (i.e., conditional mean or conditional average). In short, it attempts to remove the noise from the noisy input, outputting the best possible guess (in expectation and w.r.t. the  $\ell^2$ -distance) of the (de-noised) original random variable.

*Example 3.2* (Denoising Gaussian Noise from a Mixture of Gaussians). In this example we compute the Bayes optimal denoiser for an incredibly important class of distributions, the Gaussian mixture model. To start, let us fix parameters for the distribution: mixture weights  $\boldsymbol{\pi} \in \mathbb{R}^K$ , component means  $\{\boldsymbol{\mu}_k\}_{k=1}^K \subseteq \mathbb{R}^D$ , and component covariances  $\{\boldsymbol{\Sigma}_k\}_{k=1}^K \subseteq \text{PSD}(D)$ , where  $\text{PSD}(D)$  is the set of  $D \times D$  symmetric positive semidefinite matrices. Now, suppose  $\mathbf{x}$  is generated by the following two-step procedure:

- First, an index (or *label*)  $y \in [K]$  is sampled such that  $y = k$  with probability  $\pi_k$ .
- Second,  $\mathbf{x}$  is sampled from the normal distribution  $\mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$ .

Then  $\mathbf{x}$  has distribution

$$\mathbf{x} \sim \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (3.2.5)$$

and so

$$\mathbf{x}_t = \mathbf{x} + t\mathbf{g} \sim \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2 \mathbf{I}). \quad (3.2.6)$$

Let us define  $\varphi(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$  as the probability density of  $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  evaluated at  $\mathbf{x}$ . In this notation, the density of  $\mathbf{x}_t$  is

$$p_t(\mathbf{x}_t) = \sum_{k=1}^K \pi_k \varphi(\mathbf{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2 \mathbf{I}). \quad (3.2.7)$$

Conditioned on  $y$ , the variables are jointly Gaussian: if we say that  $\mathbf{x} = \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_y^{1/2} \mathbf{u}$  where  $(\cdot)^{1/2}$  is the matrix square root and  $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  independently of  $y$  (and  $\mathbf{g}$ ), then we have

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{x}_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_y \\ \boldsymbol{\mu}_y \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Sigma}_y^{1/2} & \mathbf{0} \\ \boldsymbol{\Sigma}_y^{1/2} & t\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{g} \end{bmatrix}. \quad (3.2.8)$$

This shows that  $\mathbf{x}$  and  $\mathbf{x}_t$  are jointly Gaussian (conditioned on  $y$ ) as claimed. Thus we can write

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{x}_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_y \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_y & \boldsymbol{\Sigma}_y \\ \boldsymbol{\Sigma}_y & \boldsymbol{\Sigma}_y + t^2 \mathbf{I} \end{bmatrix}\right). \quad (3.2.9)$$

Thus the conditional expectation of  $\mathbf{x}$  given  $\mathbf{x}_t$  (i.e., the Bayes optimal denoiser conditioned on  $y$ ) is famously (Exercise 3.2)

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t, y] = \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_y (\boldsymbol{\Sigma}_y + t^2 \mathbf{I})^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_y). \quad (3.2.10)$$

To find the overall Bayes optimal denoiser, we use the law of iterated expectation, obtaining

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \mathbb{E}[\mathbf{x} | \mathbf{x}_t] \quad (3.2.11)$$

$$= \mathbb{E}[\mathbb{E}[\mathbf{x} | \mathbf{x}_t, y] | \mathbf{x}_t] \quad (3.2.12)$$

$$= \sum_{k=1}^K \mathbb{P}[y = k | \mathbf{x}_t] \mathbb{E}[\mathbf{x} | \mathbf{x}_t, y = k]. \quad (3.2.13)$$

The probability can be dealt with as follows. Let  $p_{t|y}$  be the probability density of  $\mathbf{x}_t$  conditioned on the value of  $y$ . Then

$$\mathbb{P}[y = k | \mathbf{x}_t] = \frac{p_{t|y}(\mathbf{x}_t | k) \pi_k}{p_t(\mathbf{x}_t)} \quad (3.2.14)$$

$$= \frac{\pi_k \varphi(\mathbf{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2 \mathbf{I})}{\sum_{i=1}^K \pi_i \varphi(\mathbf{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i + t^2 \mathbf{I})}. \quad (3.2.15)$$

On the other hand, the conditional expectation is as described before:

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t, y = k] = \boldsymbol{\mu}_k + \boldsymbol{\Sigma}_k (\boldsymbol{\Sigma}_k + t^2 \mathbf{I})^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_k). \quad (3.2.16)$$

So putting this all together, the true Bayes optimal denoiser is

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \sum_{k=1}^K \frac{\pi_k \varphi(\mathbf{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2 \mathbf{I})}{\sum_{i=1}^K \pi_i \varphi(\mathbf{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i + t^2 \mathbf{I})} \cdot (\boldsymbol{\mu}_k + \boldsymbol{\Sigma}_k (\boldsymbol{\Sigma}_k + t^2 \mathbf{I})^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_k)). \quad (3.2.17)$$

This example is particularly important, and several special cases will give us great conceptual insight later. For now, let us attempt to extract some geometric intuition from the functional form of the optimal denoiser (3.2.17).

To try to understand (3.2.17) intuitively, let us first set  $K = 1$  (i.e., one Gaussian) such that  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . Let us then diagonalize  $\boldsymbol{\Sigma} = \mathbf{V} \boldsymbol{\Lambda} \mathbf{V}^\top$ . Then the Bayes optimal denoiser is

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \boldsymbol{\mu} + \boldsymbol{\Sigma} (\boldsymbol{\Sigma} + t^2 \mathbf{I})^{-1} (\mathbf{x}_t - \boldsymbol{\mu}) = \boldsymbol{\mu} + \mathbf{V} \begin{bmatrix} \lambda_1 / (\lambda_1 + t^2) & & \\ & \ddots & \\ & & \lambda_D / (\lambda_D + t^2) \end{bmatrix} \mathbf{V}^\top (\mathbf{x}_t - \boldsymbol{\mu}), \quad (3.2.18)$$

where  $\lambda_1, \dots, \lambda_D$  are the eigenvalues of  $\boldsymbol{\Sigma}$ . We can observe that this denoiser has three steps:

- Translate the input  $\mathbf{x}_t$  by  $\boldsymbol{\mu}$ .
- Contract the (translated) input  $\mathbf{x}_t - \boldsymbol{\mu}$  in each eigenvector direction by a quantity  $\lambda_i / (\lambda_i + t^2)$ . If the translated input is low-rank and some eigenvalues of  $\boldsymbol{\Sigma}$  are zero, these directions get immediately contracted to 0 by the denoiser, ensuring that the output of the contraction is similarly low-rank.
- Translate the output back by  $\boldsymbol{\mu}$ .

It is easy to show that it contracts the current  $\mathbf{x}_t$  towards the mean  $\boldsymbol{\mu}$ :

$$\|\bar{\mathbf{x}}^*(t, \mathbf{x}_t) - \boldsymbol{\mu}\|_2 \leq \|\mathbf{x}_t - \boldsymbol{\mu}\|_2. \quad (3.2.19)$$

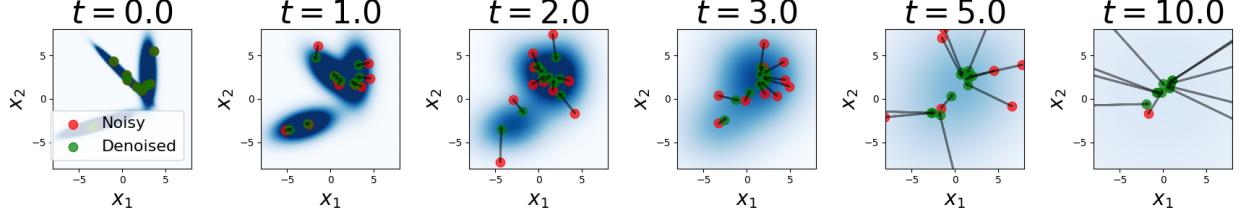


Figure 3.4: **Bayes optimal denoiser and score of a Gaussian mixture model.** In the same setting as Figure 3.3, we demonstrate the effect of the Bayes optimal denoiser  $\bar{x}^*$  by plotting  $\mathbf{x}_t$  (red) and  $\bar{x}^*(t, \mathbf{x}_t)$  (green) for some choice  $t$  and  $\mathbf{x}_t$ . By Tweedie's formula Theorem 3.3, the residual between them is proportional to the so-called (Hyvärinen) score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ . We can see that the score points towards the modes of the distribution of  $\mathbf{x}_t$ .

This is the geometric interpretation of the denoiser of a *single* Gaussian. The overall denoiser of the Gaussian mixture model (3.2.17) uses  $K$  such denoisers, weighting their output by the posterior probabilities  $\mathbb{P}[y = k \mid \mathbf{x}_t]$ . If the means of the Gaussians are well-separated, these posterior probabilities are very close to 0 or 1 near each mean or cluster. In this regime, the overall denoiser (3.2.17) has the same geometric interpretation as the above single Gaussian denoiser.

At first glance, such a contraction mapping (3.2.19) may appear similar to power iterations (see Section 2.1.2). However, the two are fundamentally different. Power iteration implements a contraction mapping towards a subspace—namely the subspace spanned by the first principal component. In contrast, the iterates in (3.2.19) converge to the mean  $\mu$  of the underlying distribution, which is a single point. ■

Intuitively, and as we can see from Example 3.2, the Bayes optimal denoiser  $\bar{x}^*(t, \cdot)$  should move its input  $\mathbf{x}_t$  towards the modes of the distribution of  $\mathbf{x}$ . It turns out that, actually, we can quantify this by showing that the Bayes optimal denoiser *takes a gradient ascent step* on the (log-)density of  $\mathbf{x}_t$ , which (recall) we denoted  $p_t$ . That is, following the denoiser means moving from the input iterate to a region of higher probability within this (perturbed) distribution. For small  $t$ , the perturbation is small so our initial intuition is therefore (almost) exactly right. The picture is visualized in Figure 3.4 and rigorously formulated as Tweedie's formula [Rob56].

**Theorem 3.3** (Tweedie's Formula). *Suppose that  $(\mathbf{x}_t)_{t \in [0, T]}$  obeys (3.2.1). Let  $p_t$  be the density of  $\mathbf{x}_t$  (as previously declared). Then*

$$\mathbb{E}[\mathbf{x} \mid \mathbf{x}_t] = \mathbf{x}_t + t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t). \quad (3.2.20)$$

*Proof.* For the proof let us suppose that  $\mathbf{x}$  has a density (even though the theorem is true without this assumption), and call this density  $p$ . Let  $p_{0|t}$  and  $p_{t|0}$  be the conditional densities of  $\mathbf{x} = \mathbf{x}_0$  given  $\mathbf{x}_t$  and  $\mathbf{x}_t$  given  $\mathbf{x}$  respectively. Let  $\varphi(\mathbf{x}; \mu, \Sigma)$  be the density of  $\mathcal{N}(\mu, \Sigma)$  evaluated at  $\mathbf{x}$ , so that  $p_{t|0}(\mathbf{x}_t \mid \mathbf{x}) = \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I})$ . Then a simple calculation gives

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t)}{p_t(\mathbf{x}_t)} \quad (3.2.21)$$

$$= \frac{1}{p_t(\mathbf{x}_t)} \nabla_{\mathbf{x}_t} \int_{\mathbb{R}^D} p(\mathbf{x}) p_{t|0}(\mathbf{x}_t \mid \mathbf{x}) d\mathbf{x} \quad (3.2.22)$$

$$= \frac{1}{p_t(\mathbf{x}_t)} \nabla_{\mathbf{x}_t} \int_{\mathbb{R}^D} p(\mathbf{x}) \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I}) d\mathbf{x} \quad (3.2.23)$$

$$= \frac{1}{p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p(\mathbf{x}) [\nabla_{\mathbf{x}_t} \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I})] d\mathbf{x} \quad (3.2.24)$$

$$= \frac{1}{p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p(\mathbf{x}) \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I}) \left[ -\frac{\mathbf{x}_t - \mathbf{x}}{t^2} \right] d\mathbf{x} \quad (3.2.25)$$

$$= \frac{1}{t^2 p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p(\mathbf{x}) \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I}) [\mathbf{x} - \mathbf{x}_t] d\mathbf{x} \quad (3.2.26)$$

$$= \frac{1}{t^2 p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p(\mathbf{x}) \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I}) \mathbf{x} d\mathbf{x} - \frac{\mathbf{x}_t}{t^2 p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p(\mathbf{x}) \varphi(\mathbf{x}_t; \mathbf{x}, t^2 \mathbf{I}) d\mathbf{x} \quad (3.2.27)$$

$$= \frac{1}{t^2 p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p(\mathbf{x}) p_{t|0}(\mathbf{x}_t \mid \mathbf{x}) \mathbf{x} d\mathbf{x} - \frac{\mathbf{x}_t}{t^2 p_t(\mathbf{x}_t)} p_t(\mathbf{x}_t) \quad (3.2.28)$$

$$= \frac{1}{t^2 p_t(\mathbf{x}_t)} \int_{\mathbb{R}^D} p_t(\mathbf{x}_t) p_{0|t}(\mathbf{x} | \mathbf{x}_t) \mathbf{x} d\mathbf{x} - \frac{\mathbf{x}_t}{t^2 p_t(\mathbf{x}_t)} p_t(\mathbf{x}_t) \quad (3.2.29)$$

$$= \frac{1}{t^2} \int_{\mathbb{R}^D} p_{0|t}(\mathbf{x} | \mathbf{x}_t) \mathbf{x} d\mathbf{x} - \frac{\mathbf{x}_t}{t^2} \quad (3.2.30)$$

$$= \frac{1}{t^2} \mathbb{E}[\mathbf{x} | \mathbf{x}_t] - \frac{\mathbf{x}_t}{t^2} \quad (3.2.31)$$

$$= \frac{\mathbb{E}[\mathbf{x} | \mathbf{x}_t] - \mathbf{x}_t}{t^2}. \quad (3.2.32)$$

Simple rearranging of the above equality proves the theorem.  $\square$

This result develops a connection between denoising and optimization: the Bayes-optimal denoiser takes a single step of gradient ascent on the perturbed data density  $p_t$ , and the step size adaptively becomes smaller (i.e., taking more precise steps) as the perturbation to the data distribution grows smaller. The quantity  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$  is called the (*Hyvärinen*) *score* and frequently appears in discussions about denoising, etc.; it first appeared in a paper of Aapo Hyvärinen in the context of ICA [Hyv05].

Similar to how one step of gradient descent is almost never sufficient to minimize an objective in practice when initializing far from the optimum, the output of the Bayes-optimal denoiser  $\bar{\mathbf{x}}^*(t, \cdot)$  is almost never contained in a high-probability region of the data distribution when  $t$  is large, *especially* when the data have low-dimensional structures. We illustrate this point explicitly in the following example.

*Example 3.3* (Denoising a Two-Point Mixture). Let  $x$  be uniform on the two-point set  $\{-1, +1\}$  and let  $(\mathbf{x}_t)_{t \in [0, T]}$  follow (3.2.1). This is precisely a degenerate Gaussian mixture model with priors equal to  $\frac{1}{2}$ , means  $\{-1, +1\}$ , and covariances both equal to 0. For a fixed  $t > 0$  we can use the calculation of the Bayes-optimal denoiser in (3.2.17) to obtain (proof as exercise)

$$\bar{x}^*(t, x_t) = \frac{\varphi(x_t; +1, t^2) - \varphi(x_t; -1, t^2)}{\varphi(x_t; 1, t^2) + \varphi(x_t; -1, t^2)} = \tanh\left(-\frac{x_t}{t^2}\right). \quad (3.2.33)$$

For  $t$  near 0, this quantity is near  $\{-1, +1\}$  for almost all inputs  $\bar{x}^*(t, x_t)$ . However, for  $t$  large, this quantity is not necessarily even approximately in the original support of  $x$ , which, remember, is  $\{-1, +1\}$ . In particular, for  $x_t \approx 0$  it holds  $\bar{x}^*(t, x_t) \approx 0$  which lies completely in between the two possible points. Thus  $\bar{x}^*$  will not output “realistic”  $x$ . Or more mathematically, the distribution of  $\bar{x}(t, x_t)$  is very different from the distribution of  $x$ . ■

Therefore, if we want to denoise the very noisy sample  $\mathbf{x}_T$  (where—recall— $T$  is the maximum time), we cannot just use the denoiser *once*. Instead, we must use the denoiser many times, analogously to gradient descent with *decaying step sizes*, to converge to a stationary point  $\hat{\mathbf{x}}$ . Namely, we shall use the denoiser to go from  $\mathbf{x}_T$  to  $\hat{\mathbf{x}}_{T-\delta}$  which approximates  $\mathbf{x}_{T-\delta}$ , then from  $\hat{\mathbf{x}}_{T-\delta}$  to  $\hat{\mathbf{x}}_{T-2\delta}$ , etc., etc., all the way from  $\hat{\mathbf{x}}_\delta$  to  $\hat{\mathbf{x}} = \hat{\mathbf{x}}_0$ . Each time we take a denoising step, the action of the denoiser becomes more like a gradient step on the original (log-)density.

More formally, we uniformly discretize  $[0, T]$  into  $L + 1$  timesteps  $0 = t_0 < t_1 < \dots < t_L = T$ , i.e.,

$$t_\ell = \frac{\ell}{L} T, \quad \ell \in \{0, 1, \dots, L\}. \quad (3.2.34)$$

Then for each  $\ell \in [L] = \{1, 2, \dots, L\}$ , going from  $\ell = L$  to  $\ell = 1$ , we can run the iteration

$$\hat{\mathbf{x}}_{t_{\ell-1}} = \mathbb{E}[\mathbf{x}_{t_{\ell-1}} | \mathbf{x}_{t_\ell} = \hat{\mathbf{x}}_{t_\ell}] \quad (3.2.35)$$

$$= \mathbb{E}[\mathbf{x} + t_{\ell-1} \mathbf{g} | \mathbf{x}_{t_\ell} = \hat{\mathbf{x}}_{t_\ell}] \quad (3.2.36)$$

$$= \mathbb{E}\left[\mathbf{x} + t_{\ell-1} \cdot \frac{\mathbf{x}_{t_\ell} - \mathbf{x}}{t_\ell} \mid \mathbf{x}_{t_\ell} = \hat{\mathbf{x}}_{t_\ell}\right] \quad (3.2.37)$$

$$= \frac{t_{\ell-1}}{t_\ell} \hat{\mathbf{x}}_{t_\ell} + \left(1 - \frac{t_{\ell-1}}{t_\ell}\right) \mathbb{E}[\mathbf{x} | \mathbf{x}_{t_\ell} = \hat{\mathbf{x}}_{t_\ell}] \quad (3.2.38)$$

$$= \left(1 - \frac{1}{\ell}\right) \cdot \hat{\mathbf{x}}_{t_\ell} + \frac{1}{\ell} \cdot \bar{\mathbf{x}}^*(t_\ell, \hat{\mathbf{x}}_{t_\ell}). \quad (3.2.39)$$

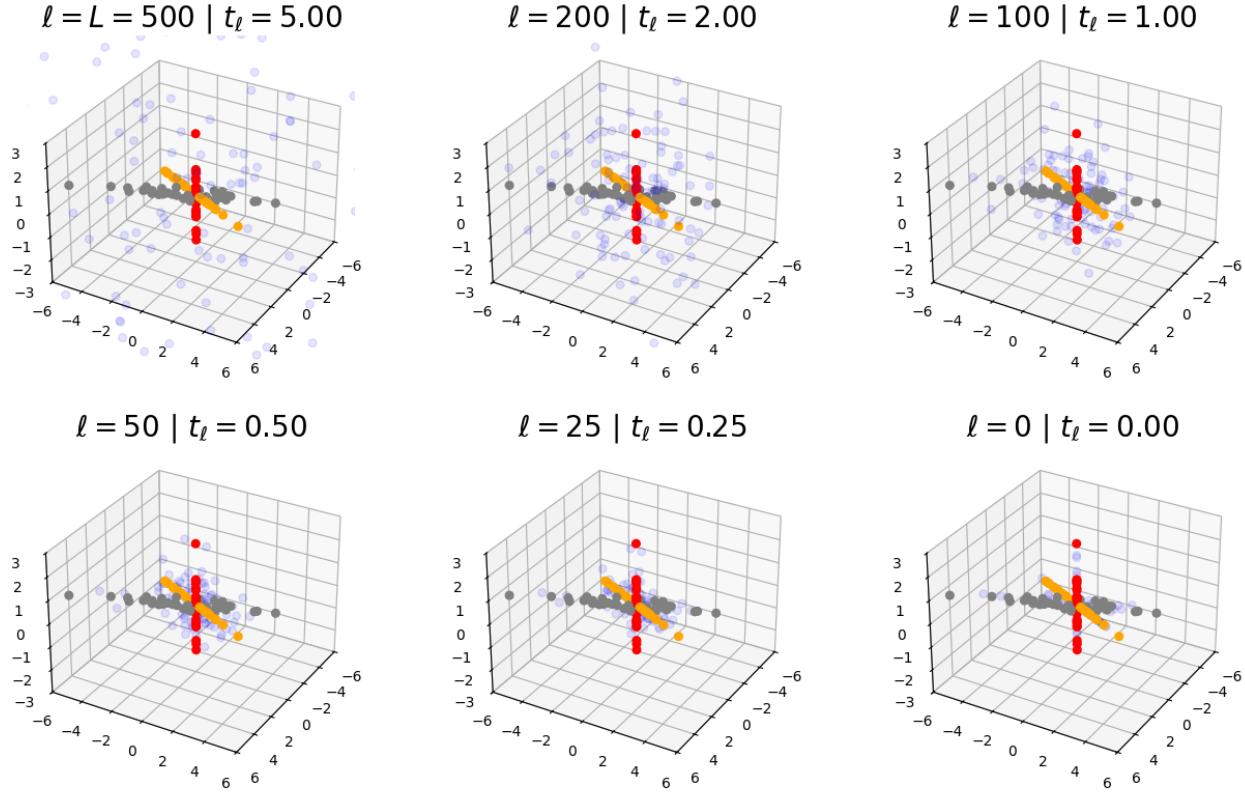


Figure 3.5: **Denoising a low-rank mixture of Gaussians.** Each figure represents samples from the true data distribution (gray, orange, red) and samples undergoing the denoising process (3.2.66) (light blue). At top left, the process has just started, and the noise is very large. As the process continues, the noise is pushed further towards the support of the low-rank data distribution. Finally, in the bottom right, the generated samples are perfectly aligned with the support of the data and look very much like samples drawn from the low-rank Gaussian mixture model.

The effect of this iteration is as follows. At the beginning of the iteration where  $\ell$  is large, we barely trust the output of the denoiser, and mostly keep the current iterate. This makes sense, as the denoiser can have huge variance (cf Example 3.3). When  $\ell$  is small, the denoiser will “lock on” to the modes of the data distribution as a denoising step basically takes a gradient step on the true distribution’s log-density, and we can trust it to not produce unreasonable samples, so the denoising step mostly involves the output of the denoiser. At  $\ell = 1$  we even throw away the current iterate and just keep the output of the denoiser.

The above is intuition for why we expect the denoising process to converge. We visualize the convergence process in  $\mathbb{R}^3$  in Figure 3.5. We will develop some rigorous results about convergence later. For now, recall that we wanted to build a process to reduce the entropy. While we did do this in a roundabout way by inverting a process which adds entropy, it is now time to pay the piper and confirm that our iterative denoising process reduces the entropy.

**Theorem 3.4** (Simplified Version of Theorem B.3). *Suppose that  $(\mathbf{x}_t)_{t \in [0, T]}$  obeys (3.2.1). Then, under certain technical conditions on  $\mathbf{x}$ , for every  $s < t$  with  $s, t \in (0, T]$ ,*

$$h(\mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t]) < h(\mathbf{x}_t). \quad (3.2.40)$$

The full statement of the theorem, and the proof itself, requires some technicality, so it is postponed to Section B.2.2.

The last thing we discuss here is that many times, we will *not be able to compute*  $\bar{\mathbf{x}}^*(t, \cdot)$  for any  $t$ , since we do not have the distribution  $p_t$ . But we can try to *learn one from data*. Recall that the denoiser  $\bar{\mathbf{x}}^*$  is defined in (3.2.3) as minimizing the mean-squared error  $\mathbb{E} \|\bar{\mathbf{x}}(t, \mathbf{x}_t) - \mathbf{x}\|_2^2$ . We can use this mean-squared

error as a loss or objective function to learn the denoiser. For example, we can parameterize  $\bar{\mathbf{x}}(t, \cdot)$  by a neural network, writing it as  $\bar{\mathbf{x}}_\theta(t, \cdot)$ , and optimize the loss over the parameter space  $\Theta$ :

$$\min_{\theta \in \Theta} \mathbb{E}_{\mathbf{x}, \mathbf{x}_t} \|\bar{\mathbf{x}}_\theta(t, \mathbf{x}_t) - \mathbf{x}\|_2^2. \quad (3.2.41)$$

The solution to this optimization problem, implemented via gradient descent or a similar algorithm, will give us a  $\bar{\mathbf{x}}_{\theta^*}(t, \cdot)$  which is a good approximation to  $\bar{\mathbf{x}}^*(t, \cdot)$  (at least if the training works) and which we will use as our denoiser.

What is a good architecture for this neural network  $\bar{\mathbf{x}}_{\theta^*}(t, \cdot)$ ? To answer this question, we will examine the ubiquitous case of a *Gaussian mixture model*, whose denoiser we computed in Example 3.2. This model is relevant because it can approximate many types of distributions: in particular, given a distribution for  $\mathbf{x}$ , there is a Gaussian mixture model that can approximate it arbitrarily well. So optimizing among the class of denoisers for Gaussian mixture models can give us something close to the optimal denoiser for the real data distribution.

In our case, we assume that  $\mathbf{x}$  is low-dimensional, which loosely translates into the requirement that  $\mathbf{x}$  is *approximately* distributed according to a *mixture of low-rank Gaussians*. Formally, we write

$$\mathbf{x} \sim \frac{1}{K} \sum_{k=1}^K \mathcal{N}(\mathbf{0}, \mathbf{U}_k \mathbf{U}_k^\top) \quad (3.2.42)$$

where  $\mathbf{U}_k \in \mathrm{O}(D, P) \subseteq \mathbb{R}^{D \times P}$  is an orthogonal matrix. Then the optimal denoiser under (3.2.1) is (from Example 3.2)

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \sum_{k=1}^K \frac{\varphi(\mathbf{x}_t; \mathbf{0}, \mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})}{\sum_{i=1}^K \varphi(\mathbf{x}_t; \mathbf{0}, \mathbf{U}_i \mathbf{U}_i^\top + t^2 \mathbf{I})} \cdot (\mathbf{U}_k \mathbf{U}_k^\top (\mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})^{-1} \mathbf{x}_t). \quad (3.2.43)$$

Notice that within the computation  $\varphi$  and outside of it, we compute the inverse  $(\mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})^{-1}$ . This is a low-rank perturbation of the full-rank matrix  $t^2 \mathbf{I}$ , and thus ripe for simplification via the *Sherman-Morrison-Woodbury identity*, i.e., for matrices  $\mathbf{A}, \mathbf{C}, \mathbf{U}, \mathbf{V}$  such that  $\mathbf{A}$  and  $\mathbf{C}$  are invertible,

$$(\mathbf{A} + \mathbf{U} \mathbf{C} \mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{U} (\mathbf{C}^{-1} + \mathbf{V} \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V} \mathbf{A}^{-1}. \quad (3.2.44)$$

We prove this identity in Exercise 3.3. For now we apply this identity with  $\mathbf{A} = t^2 \mathbf{I}$ ,  $\mathbf{U} = \mathbf{U}_k$ ,  $\mathbf{V} = \mathbf{U}_k^\top$ , and  $\mathbf{C} = \mathbf{I}$ , obtaining

$$(\mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})^{-1} = \frac{1}{t^2} \mathbf{I} - \frac{1}{t^4} \mathbf{U}_k \left( \mathbf{I} + \frac{1}{t^2} \mathbf{U}_k^\top \mathbf{U}_k \right)^{-1} \mathbf{U}_k^\top \quad (3.2.45)$$

$$= \frac{1}{t^2} \mathbf{I} - \frac{1}{t^4 (1 + \frac{1}{t^2})} \mathbf{U}_k \mathbf{U}_k^\top \quad (3.2.46)$$

$$= \frac{1}{t^2} \left( \mathbf{I} - \frac{1}{1 + t^2} \mathbf{U}_k \mathbf{U}_k^\top \right). \quad (3.2.47)$$

Then we can compute the posterior probabilities as follows. Note that since  $\mathbf{U}_k$ 's are all orthogonal,  $\det(\mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})$  are all the same for each  $k$ . So

$$\frac{\varphi(\mathbf{x}_t; \mathbf{0}, \mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})}{\sum_{i=1}^K \varphi(\mathbf{x}_t; \mathbf{0}, \mathbf{U}_i \mathbf{U}_i^\top + t^2 \mathbf{I})} = \frac{\exp(-\frac{1}{2} \mathbf{x}_t^\top (\mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})^{-1} \mathbf{x}_t)}{\sum_{i=1}^K \exp(-\frac{1}{2} \mathbf{x}_t^\top (\mathbf{U}_i \mathbf{U}_i^\top + t^2 \mathbf{I})^{-1} \mathbf{x}_t)} \quad (3.2.48)$$

$$= \frac{\exp\left(-\frac{1}{2t^2} \mathbf{x}_t^\top \left(\mathbf{I} - \frac{1}{1+t^2} \mathbf{U}_k \mathbf{U}_k^\top\right) \mathbf{x}_t\right)}{\sum_{i=1}^K \exp\left(-\frac{1}{2t^2} \mathbf{x}_t^\top \left(\mathbf{I} - \frac{1}{1+t^2} \mathbf{U}_i \mathbf{U}_i^\top\right) \mathbf{x}_t\right)} \quad (3.2.49)$$

$$= \frac{\exp\left(-\frac{1}{2t^2} \|\mathbf{x}_t\|_2^2 + \frac{1}{2t^2(1+t^2)} \|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(-\frac{1}{2t^2} \|\mathbf{x}_t\|_2^2 + \frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \quad (3.2.50)$$

$$= \frac{\exp\left(-\frac{1}{2t^2}\|\mathbf{x}_t\|_2^2\right) \exp\left(\frac{1}{2t^2(1+t^2)}\|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\exp\left(-\frac{1}{2t^2}\|\mathbf{x}_t\|_2^2\right) \sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)}\|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \quad (3.2.51)$$

$$= \frac{\exp\left(\frac{1}{2t^2(1+t^2)}\|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)}\|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)}. \quad (3.2.52)$$

This is a softmax operation weighted by the projection of  $\mathbf{x}_t$  onto each subspace measured by  $\|\mathbf{U}_i^\top \mathbf{x}_t\|_2$  (tempered by a temperature  $2t^2(1+t^2)$ ). Meanwhile, the component denoisers can be written as

$$\mathbf{U}_k \mathbf{U}_k^\top (\mathbf{U}_k \mathbf{U}_k^\top + t^2 \mathbf{I})^{-1} \mathbf{x}_t = \frac{1}{t^2} \mathbf{U}_k \mathbf{U}_k^\top \left( \mathbf{I} - \frac{1}{1+t^2} \mathbf{U}_k \mathbf{U}_k^\top \right) \mathbf{x}_t \quad (3.2.53)$$

$$= \frac{1}{t^2} \left( 1 - \frac{1}{1+t^2} \right) \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t \quad (3.2.54)$$

$$= \frac{1}{1+t^2} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t. \quad (3.2.55)$$

Putting these together, we have

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \frac{1}{1+t^2} \sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)}\|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)}\|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t, \quad (3.2.56)$$

i.e., a projection of  $\mathbf{x}_t$  onto each of  $K$  subspaces, weighted by a soft-max operation of a quadratic function of  $\mathbf{x}_t$ . This functional form is similar to an *attention mechanism* in a transformer architecture! As we will see in Chapter 4, this is no coincidence at all; the deep link between denoising and lossy compression (to be covered in Section 3.3) makes transformer denoisers so effective in practice. And so overall, our Gaussian mixture model theory motivates the use of transformer-like neural networks for denoising.

**Remark 3.1. Connections between denoising a distribution and probabilistic PCA.** Here, we would like to connect denoising a low-dimensional distribution to probabilistic PCA (see Section 2.1.3 for more details about probabilistic PCA). Suppose that we consider  $K = 1$  in (3.2.42), i.e.,  $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{U} \mathbf{U}^\top)$ , where  $\mathbf{U} \in \text{O}(D, P) \subseteq \mathbb{R}^{D \times P}$  is an orthogonal matrix. According to (3.2.56), the Bayes optimal denoiser is

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \frac{1}{1+t^2} \mathbf{U} \mathbf{U}^\top \mathbf{x}_t. \quad (3.2.57)$$

To learn this Bayes optimal denoiser, we can accordingly parameterize the denoising operator  $\bar{\mathbf{x}}(t, \mathbf{x}_t)$  as follows:

$$\bar{\mathbf{x}}(t, \mathbf{x}_t) = \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top \mathbf{x}_t, \quad (3.2.58)$$

where  $\mathbf{V} \in \text{O}(D, P)$  are learnable parameters. Substituting this into the training loss (3.2.3) yields

$$\min_{\mathbf{V} \in \text{O}(D, P)} \mathbb{E}_{\mathbf{x}, \mathbf{x}_t} \left\| \mathbf{x} - \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top \mathbf{x}_t \right\|_2^2 = \mathbb{E}_{\mathbf{x}, \mathbf{g}} \left\| \mathbf{x} - \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top (\mathbf{x} + t \mathbf{g}) \right\|_2^2, \quad (3.2.59)$$

where the equality is due to (3.2.1). Conditioned on  $\mathbf{x}$ , we compute

$$\mathbb{E}_{\mathbf{g}} \left\| \mathbf{x} - \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top (\mathbf{x} + t \mathbf{g}) \right\|_2^2 \quad (3.2.60)$$

$$= \left\| \mathbf{x} - \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top \mathbf{x} \right\|_2^2 - \frac{t}{1+t^2} \mathbb{E}_{\mathbf{g}} \left\langle \mathbf{x} - \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top \mathbf{x}, \mathbf{V} \mathbf{V}^\top \mathbf{g} \right\rangle + \frac{t^2}{(1+t^2)^2} \mathbb{E}_{\mathbf{g}} \|\mathbf{V} \mathbf{V}^\top \mathbf{g}\|_2^2 \quad (3.2.61)$$

$$= \left\| \mathbf{x} - \frac{1}{1+t^2} \mathbf{V} \mathbf{V}^\top \mathbf{x} \right\|_2^2 + \frac{t^2 P}{(1+t^2)^2} \quad (3.2.62)$$

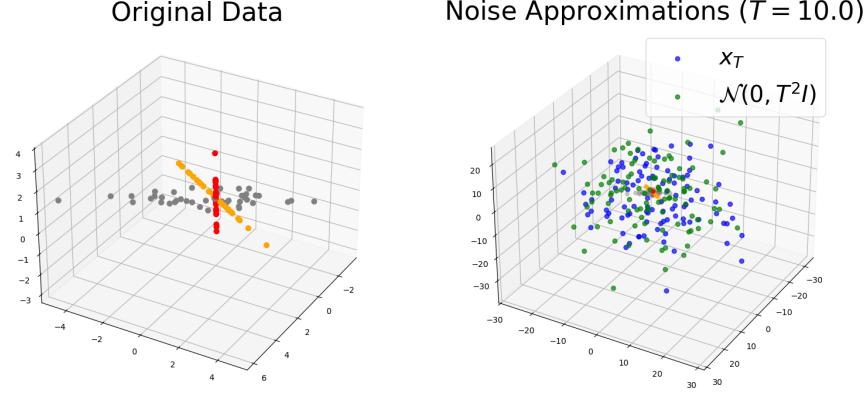


Figure 3.6: **Visualizing  $x_T$  versus  $\mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$ .** *Left:* A plot of Gaussian mixture model data  $\mathbf{x}$ . *Right:* A plot of  $\mathbf{x}$  as well as  $x_T$  and an independent sample of  $\mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$ , for  $T = 10$ . On the right plot,  $\mathbf{x}$  is plotted in the same colors as the left: however, samples from  $x_T$  and  $\mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$  are both much larger, on average, than samples from  $\mathbf{x}$ , and so it appears much smaller because of the scaling. Despite this large blow-up, we clearly observe the similarities in the distributions of  $x_T$  and  $\mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$ .

where the second equality follows from  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  and  $\mathbb{E}_{\mathbf{g}} \|\mathbf{V}\mathbf{V}^\top \mathbf{g}\|_2^2 = \mathbb{E}_{\mathbf{g}} \|\mathbf{V}^\top \mathbf{g}\|_2^2 = P$  due to  $\mathbf{V} \in \mathrm{O}(D, P)$ . Therefore, Problem (3.2.59) is equivalent to

$$\min_{\mathbf{V} \in \mathrm{O}(D, P)} \mathbb{E}_{\mathbf{x}} \left\| \mathbf{x} - \frac{1}{1+t^2} \mathbf{V}\mathbf{V}^\top \mathbf{x} \right\|_2^2 = \mathbb{E}_{\mathbf{x}} \|\mathbf{x}\|_2^2 + \left( \frac{1}{(1+t^2)^2} - \frac{2}{1+t^2} \right) \mathbb{E}_{\mathbf{x}} \|\mathbf{V}^\top \mathbf{x}\|_2^2. \quad (3.2.63)$$

This is further equivalent to

$$\max_{\mathbf{V} \in \mathrm{O}(D, P)} \mathbb{E}_{\mathbf{x}} \|\mathbf{V}^\top \mathbf{x}\|_2^2, \quad (3.2.64)$$

which is essentially Problem (2.1.27).

Overall, the learned denoiser forms an (implicit parametric) encoding scheme of the given data, since it can be used to denoise/project onto the data distribution. Training a denoiser is equivalent to finding a better coding scheme, and this partially fulfills one of the desiderata (the *second*) at the end of Section 3.1.3. In the sequel, we will discuss how to fulfill the other (the *first*).

### 3.2.2 Learning and Sampling a Distribution via Iterative Denoising

Remember that at the end of Section 3.1.3, we discussed a pair of desiderata for pursuing a distribution with low-dimensional structure. The first such desideratum is to start with a normal distribution, say with high entropy, and gradually reduce its entropy until it reaches the distribution of the data. We will call this procedure *sampling* since we are generating new samples. It is now time for us to discuss how to do this with the toolkit we have built up.

We know how to denoise very noisy samples  $x_T$  to attain approximations  $\hat{\mathbf{x}}$  which have similar distributions to the target random variable  $\mathbf{x}$ . But the desideratum says that, to sample, we want to start with a template distribution with *no* influence from the distribution of  $\mathbf{x}$ , and use the denoiser to guide the iterates towards the distribution of  $\mathbf{x}$ . How can we do this? One way is motivated as follows:

$$\frac{\mathbf{x}_T}{T} = \frac{\mathbf{x} + T\mathbf{g}}{T} = \frac{\mathbf{x}}{T} + \mathbf{g} \rightarrow \mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (3.2.65)$$

Thus,  $\mathbf{x}_T \approx \mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$ . This approximation is quite good for almost all practical distributions, and visualized in Figure 3.6.

So, discretizing  $[0, T]$  into  $0 = t_0 < t_1 < \dots < t_L = T$  uniformly using  $t_\ell = T\ell/L$  (as in the previous section), one possible way to sample from pure noise is:

- Sample  $\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$  (i.i.d. of everything else)
- Run the denoising iteration as in Section 3.2.1, i.e.,

$$\hat{\mathbf{x}}_{t_{\ell-1}} = \left(1 - \frac{1}{\ell}\right) \cdot \hat{\mathbf{x}}_{t_\ell} + \frac{1}{\ell} \cdot \bar{\mathbf{x}}^*(t_\ell, \hat{\mathbf{x}}_{t_\ell}). \quad (3.2.66)$$

- Output  $\hat{\mathbf{x}} = \hat{\mathbf{x}}_0$ .

This conceptually is all there is behind *diffusion models*, which transform noise into data samples in accordance with the first desideratum. However, there are a few steps left to take before we get models which can actually sample from real data distributions like images given practical resource constraints. In the sequel, we will introduce and motivate several such steps.

**Step 1: different discretizations.** The first step we do is motivated by the following point: *we do not need to spend so many denoising iterations* at large  $t$ . If we look at Figure 3.5, we observe that the first 200 or 300 iterations, out of the 500 iterations of the sampling process, are just spent contracting the noise towards the data distribution as a whole, before the remaining iterations push the samples towards a subspace. Given a fixed iteration count  $L$ , this signals that we should spend more timesteps  $t_\ell$  near  $t = 0$  compared to  $t = T$ . During sampling (and training), we can therefore use another discretization of  $[0, T]$  into  $0 \leq t_0 < t_1 < \dots < t_L \leq T$ , such as an *exponential discretization*:

$$t_\ell = C_1(e^{C_2 \ell} - 1), \quad \forall \ell \in \{0, 1, \dots, L\} \quad (3.2.67)$$

where  $C_1, C_2 > 0$  are constants which can be tuned for optimal performance in practice; theoretical analysis will often specify such optimal constants as well. Then the denoising/sampling iteration becomes

$$\hat{\mathbf{x}}_{t_{\ell-1}} \doteq \frac{t_{\ell-1}}{t_\ell} \hat{\mathbf{x}}_{t_\ell} + \left(1 - \frac{t_{\ell-1}}{t_\ell}\right) \bar{\mathbf{x}}^*(t_\ell, \hat{\mathbf{x}}_{t_\ell}), \quad (3.2.68)$$

with, again,  $\hat{\mathbf{x}}_{t_L} \sim \mathcal{N}(\mathbf{0}, t_L^2 \mathbf{I})$ .

**Step 2: different noise models.** The second step is to consider slightly different models compared to (3.2.1). The basic motivation for this is as follows. In practice, the noise distribution  $\mathcal{N}(\mathbf{0}, t_L^2 \mathbf{I})$  becomes an increasingly poor estimate of the true covariance in high dimensions, i.e., (3.2.65) becomes an increasingly worse approximation, especially with anisotropic high-dimensional data. The increased distance between  $\mathcal{N}(\mathbf{0}, t_L^2 \mathbf{I})$  and the true distribution of  $\mathbf{x}_{t_L}$  may cause the denoiser to perform worse in such circumstances. Theoretically,  $\mathbf{x}_{t_L}$  never converges to any distribution as  $t_L$  increases, so this setup is difficult to analyze end-to-end. In this case, our remedy is to *simultaneously add noise and shrink the contribution of  $\mathbf{x}$ , such that  $\mathbf{x}_T$  converges as  $T \rightarrow \infty$* . The rate of added noise is denoted  $\sigma: [0, T] \rightarrow \mathbb{R}_{\geq 0}$ , and the rate of shrinkage is denoted  $\alpha: [0, T] \rightarrow \mathbb{R}_{\geq 0}$ , such that  $\sigma$  is *increasing* and  $\alpha$  is (not strictly) *decreasing*, and

$$\mathbf{x}_t \doteq \alpha_t \mathbf{x} + \sigma_t \mathbf{g}, \quad \forall t \in [0, T]. \quad (3.2.69)$$

The previous setup has  $\alpha_t = 1$  and  $\sigma_t = t$ , and this is called the *variance-exploding (VE) process*. A popular choice which decreases the contribution of  $\mathbf{x}$ , as we described originally, has  $T = 1$  (so that  $t \in [0, 1]$ ),  $\alpha_t = \sqrt{1 - t^2}$  and  $\sigma_t = t$ ; this is the *variance-preserving (VP) process*. Note that under the VP process,  $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  exactly, so we can just sample from this standard distribution and iteratively denoise. As a result, the VP process is much easier to analyze theoretically and more stable empirically.<sup>6</sup>

With this more general setup, Tweedie's formula (3.2.20) becomes

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t] = \frac{1}{\alpha_t} (\mathbf{x}_t + \sigma_t^2 \nabla \log p_t(\mathbf{x})). \quad (3.2.70)$$

---

<sup>6</sup>Why use the whole  $\alpha, \sigma$  setup? As we will see in Exercise 3.5, it encapsulates and unifies many proposed processes, including the recently popular so-called *flow matching* process. Despite this, the VE and VP processes are still the most popular empirically and theoretically (so far), and so we will consider them in this Section.

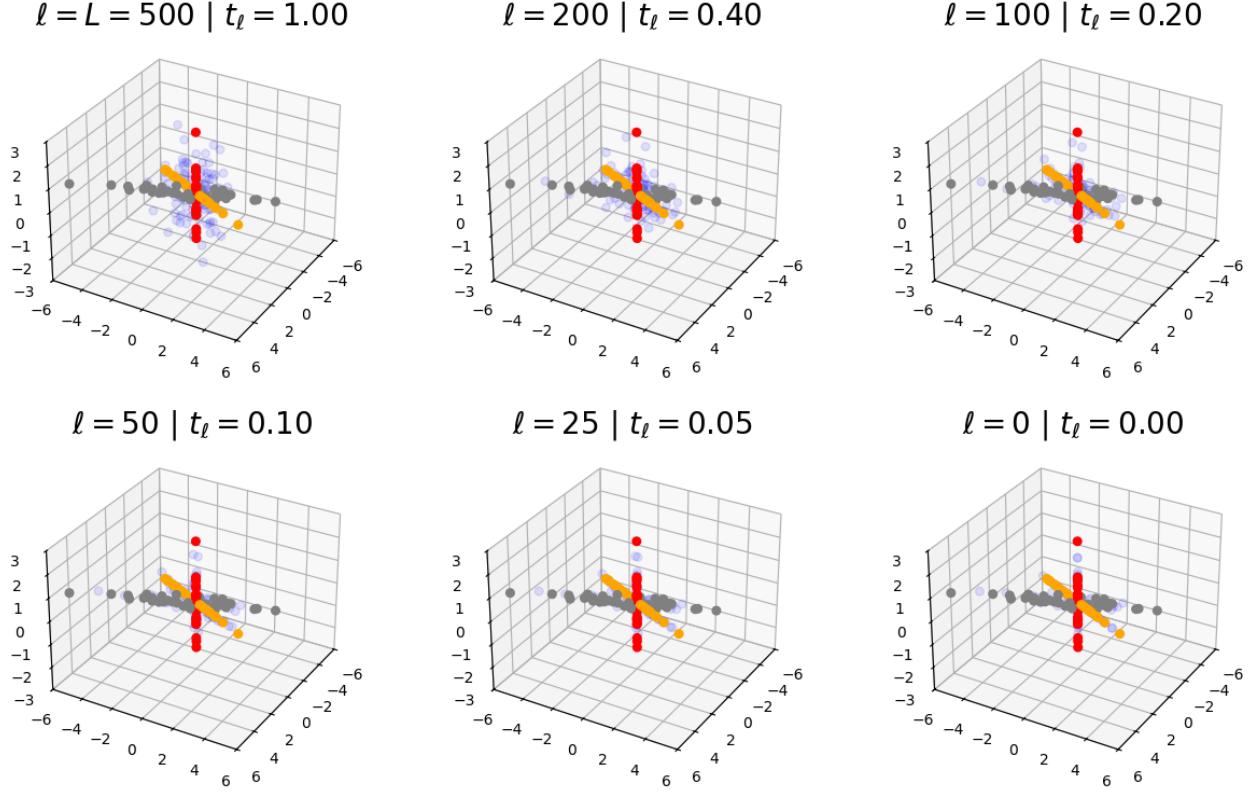


Figure 3.7: Denoising a mixture of Gaussians using the VP diffusion process. We use the same figure setup and data distribution as Figure 3.5. Note that compared to Figure 3.5, the noise distribution is much more concentrated around the origin.

The denoising iteration (3.2.68) becomes

$$\hat{\mathbf{x}}_{t_{\ell-1}} = \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \hat{\mathbf{x}}_{t_\ell} + \left( \alpha_{t_{\ell-1}} - \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \alpha_{t_\ell} \right) \bar{\mathbf{x}}^*(t_\ell, \hat{\mathbf{x}}_{t_\ell}). \quad (3.2.71)$$

Finally, the Gaussian mixture model denoiser (3.2.17) becomes

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \sum_{k=1}^K \frac{\pi_k \varphi(\mathbf{x}_t; \alpha_t \boldsymbol{\mu}_k, \alpha_t^2 \boldsymbol{\Sigma}_k + \sigma_t^2 \mathbf{I})}{\sum_{i=1}^K \pi_i \varphi(\mathbf{x}_t; \alpha_t \boldsymbol{\mu}_i, \alpha_t^2 \boldsymbol{\Sigma}_i + \sigma_t^2 \mathbf{I})} \cdot (\boldsymbol{\mu}_k + \alpha_t \boldsymbol{\Sigma}_k (\alpha_t^2 \boldsymbol{\Sigma}_k + \sigma_t^2 \mathbf{I})^{-1} (\mathbf{x}_t - \alpha_t \boldsymbol{\mu}_k)). \quad (3.2.72)$$

Figure 3.7 demonstrates iterations of the sampling procedure. Note that the denoising iteration (3.2.71) gives a sampling algorithm called the DDIM (“Denoising Diffusion Implicit Model”) sampler [SME20], and is one of the most popular sampling algorithms used today in diffusion models. We summarize it here in Algorithm 3.1.

**Step 3: optimizing training pipelines.** If we use the procedure dictated by Section 3.2.1 to learn a separate denoiser  $\bar{\mathbf{x}}(t, \cdot)$  for each time  $t$  to be used in the sampling algorithm, *we would have to learn  $L$  separate denoisers!* This is highly inefficient—the usual case is that we have to train  $L$  separate neural networks, taking up  $L$  times the training time and storage memory, and then be locked into using these timesteps for sampling forever. Instead, we can *train a single neural network* to denoise across all times  $t$ , taking as input the continuous variables  $\mathbf{x}_t$  and  $t$  (instead of just  $\mathbf{x}_t$  before). Mechanically, our training loss averages over  $t$ , i.e., solves the following problem:

$$\min_{\theta} \mathbb{E}_{t, \mathbf{x}, \mathbf{x}_t} \|\bar{\mathbf{x}}_{\theta}(t, \mathbf{x}_t) - \mathbf{x}\|_2^2. \quad (3.2.73)$$

---

**Algorithm 3.1** Sampling using a denoiser.

---

**Input:** An ordered list of timesteps  $0 \leq t_0 < \dots < t_L \leq T$  to use for sampling.

**Input:** A denoiser  $\bar{\mathbf{x}}: \{t_\ell\}_{\ell=1}^L \times \mathbb{R}^D \rightarrow \mathbb{R}^D$ .

**Input:** Scale and noise level functions  $\alpha, \sigma: \{t_\ell\}_{\ell=0}^L \rightarrow \mathbb{R}_{\geq 0}$ .

**Output:** A sample  $\hat{\mathbf{x}}$ , approximately from the distribution of  $\mathbf{x}$ .

```

1: function DDIMSAMPLER( $\bar{\mathbf{x}}, \{t_\ell\}_{\ell=0}^L$ )
2:   Initialize  $\tilde{\mathbf{x}}_{t_L} \sim$  approximate distribution of  $\mathbf{x}_{t_L}$             $\triangleright \text{VP} \implies \mathcal{N}(\mathbf{0}, \mathbf{I}), \text{VE} \implies \mathcal{N}(\mathbf{0}, t_L^2 \mathbf{I})$ .
3:   for  $\ell = L, L-1, \dots, 1$  do
4:     Compute
       
$$\hat{\mathbf{x}}_{t_{\ell-1}} \doteq \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \hat{\mathbf{x}}_{t_\ell} + \left( \alpha_{t_{\ell-1}} - \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \alpha_{t_\ell} \right) \bar{\mathbf{x}}(t_\ell, \hat{\mathbf{x}}_{t_\ell})$$

5:   end for
6:   return  $\hat{\mathbf{x}}_{t_0}$ 
7: end function
```

---

Similar to Step 1, where we used more timesteps closer to  $t = 0$  to ensure a better sampling process, we may want to ensure that the denoiser is higher quality closer to  $t = 0$ , and thereby *weight the loss* so that  $t$  near 0 has higher weight. Letting  $w_t$  be the weight at time  $t$ , the weighted loss would look like

$$\min_{\theta} \mathbb{E}_t w_t \mathbb{E}_{\mathbf{x}, \mathbf{x}_t} \|\bar{\mathbf{x}}_{\theta}(t, \mathbf{x}_t) - \mathbf{x}\|_2^2. \quad (3.2.74)$$

One reasonable choice of weight in practice is  $w_t = \alpha_t / \sigma_t$ . The precise reason will be covered in the next paragraph, but generally it serves to up-weight the losses corresponding to  $t$  near 0 while still remaining reasonably numerically stable. Also, of course, we cannot compute the expectation in practice, so we use the most straightforward Monte-Carlo average to estimate it. The series of changes made here have several conceptual and computational benefits: we do not need to train multiple denoisers, we can train on one set of timesteps and sample using a subset (or others entirely), etc. The full pipeline is discussed in Algorithm 3.2.

**(Optional) Step 4: changing the estimation target.** Note that it is common to instead reorient the whole denoising pipeline around *noise predictors*, i.e., estimates of  $\mathbb{E}[\mathbf{g} | \mathbf{x}_t]$ . In practice, noise predictors are slightly easier to train because their output is (almost) always of comparable size to a Gaussian random variable, so training is more numerically stable. Note that by (3.2.69) we have

$$\mathbf{x}_t = \alpha_t \mathbb{E}[\mathbf{x} | \mathbf{x}_t] + \sigma_t \mathbb{E}[\mathbf{g} | \mathbf{x}_t] \implies \mathbb{E}[\mathbf{g} | \mathbf{x}_t] = \frac{1}{\sigma_t} (\mathbf{x}_t - \alpha_t \mathbb{E}[\mathbf{x} | \mathbf{x}_t]), \quad (3.2.75)$$

Therefore any predictor for  $\mathbf{x}$  can be turned into a predictor for  $\mathbf{g}$  using the above relation, i.e.,

$$\bar{\mathbf{g}}(t, \mathbf{x}_t) = \frac{1}{\sigma_t} \mathbf{x}_t - \frac{\alpha_t}{\sigma_t} \bar{\mathbf{x}}(t, \mathbf{x}_t), \quad (3.2.76)$$

and vice-versa. Thus a good network for estimating  $\bar{\mathbf{g}}$  is the same as a good network for estimating  $\bar{\mathbf{x}}$  *plus a residual connection* (as seen in, e.g., transformers). Their losses are also the same as the denoiser, up to the factor of  $\alpha_t / \sigma_t$ , i.e.,

$$\mathbb{E}_t w_t \mathbb{E}_{\mathbf{g}, \mathbf{x}_t} \|\mathbf{g} - \bar{\mathbf{g}}(t, \mathbf{x}_t)\|_2^2 = \mathbb{E}_t w_t \frac{\alpha_t^2}{\sigma_t^2} \mathbb{E}_{\mathbf{x}, \mathbf{x}_t} \|\mathbf{x} - \bar{\mathbf{x}}(t, \mathbf{x}_t)\|_2^2. \quad (3.2.77)$$

For the sake of completeness we will mention that other targets have been proposed for different tasks, e.g.,  $\mathbb{E}[\frac{d}{dt} \mathbf{x}_t | \mathbf{x}_t]$  (called *v*-prediction or velocity prediction), etc., but denoising and noise prediction remain commonly used. Throughout the rest of this book we will only consider denoising.

We have made lots of changes to our original platonic noising/denoising process. To assure ourselves that the new process still works in practice, we can compute numerical examples (such as Figure 3.7). To assure ourselves that it is theoretically sound, we can prove a *bound on the error rate* for the sampling algorithm,

---

**Algorithm 3.2** Learning a denoiser from data.

---

**Input:** Dataset  $\mathcal{D} \subseteq \mathbb{R}^D$ .

**Input:** An ordered list of timesteps  $0 \leq t_0 < \dots < t_L \leq T$  to use for sampling.

**Input:** A weighting function  $w: \{t_\ell\}_{\ell=1}^L \rightarrow \mathbb{R}_{\geq 0}$ .

**Input:** Scale and noise level functions  $\alpha, \sigma: \{t_\ell\}_{\ell=0}^L \rightarrow \mathbb{R}_{\geq 0}$ .

**Input:** A parameter space  $\Theta$  and a denoiser architecture  $\bar{\mathbf{x}}_\theta$ .

**Input:** An optimization algorithm for the parameters.

**Input:** The number of optimization iterations  $M$ .

**Input:** The number of Monte-Carlo draws  $N$  per iteration (to approximate the expectation in (3.2.74))

**Output:** A trained denoiser  $\bar{\mathbf{x}}_{\theta^*}$ .

```

1: function TRAINDENOISER( $\mathcal{D}, \Theta$ )
2:   Initialize  $\theta^{(1)} \in \Theta$ 
3:   for  $i \in [M]$  do
4:     for  $n \in [N]$  do
5:        $\mathbf{x}_n^{(i)} \sim \mathcal{D}$                                  $\triangleright$  Draw a sample from the dataset.
6:        $t_n^{(i)} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}(\{t_\ell\}_{\ell=1}^L)$      $\triangleright$  Sample a timestep.
7:        $\mathbf{g}_n^{(i)} \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I})$        $\triangleright$  Sample a noise vector.
8:        $\mathbf{x}_{t,n}^{(i)} \doteq \alpha_{t_n^{(i)}} \mathbf{x}_n^{(i)} + \sigma_{t_n^{(i)}} \mathbf{g}_n^{(i)}$      $\triangleright$  Compute the noised sample.
9:        $w_n^{(i)} \doteq w_{t_n^{(i)}}$                                  $\triangleright$  Compute the loss weight.
10:      end for
11:       $\hat{\mathcal{L}}^{(i)} \doteq \frac{1}{N} \sum_{n=1}^N w_n^{(i)} \|\mathbf{x}_n^{(i)} - \bar{\mathbf{x}}_{\theta^{(i)}}(t_n^{(i)}, \mathbf{x}_{t,n}^{(i)})\|_2^2$      $\triangleright$  Compute the loss estimate.
12:       $\theta^{(i+1)} \doteq \text{OptimizationUpdate}^{(i)}(\theta^{(i)}, \nabla_{\theta^{(i)}} \hat{\mathcal{L}}^{(i)})$            $\triangleright$  Update parameters.
13:    end for
14:    return  $\bar{\mathbf{x}}_{\theta^{(K+1)}}$ 
15: end function

```

---

which shows that the error rate is small. We will now furnish such a rate from the literature, which shows that the output distribution of the sampler converges in the so-called *total variation (TV) distance* to the true distribution. The TV distance is defined between two random variables  $\mathbf{x}$  and  $\mathbf{y}$  as:

$$\text{TV}(\mathbf{x}, \mathbf{y}) \doteq \sup_{A \subseteq \mathbb{R}^d} |\mathbb{P}[\mathbf{x} \in A] - \mathbb{P}[\mathbf{y} \in A]|. \quad (3.2.78)$$

If  $\mathbf{x}$  and  $\mathbf{y}$  are very close (uniformly), then the supremum will be small. So the TV distance measures the closeness of random variables. (It is indeed a metric, as the name suggests: the proof is an exercise.)

**Theorem 3.5** ([LY24] Theorem 1, Simplified). *Suppose that  $\mathbb{E} \|\mathbf{x}\|_2 < \infty$ . If  $\mathbf{x}$  is denoised according to the VP process with an exponential discretization<sup>7</sup> as in (3.2.67), the output  $\hat{\mathbf{x}}$  of Algorithm 3.1 satisfies the total variation bound*

$$\text{TV}(\mathbf{x}, \hat{\mathbf{x}}) = \tilde{\mathcal{O}}\left(\underbrace{\frac{D}{L}}_{\text{discretization error}} + \underbrace{\sqrt{\frac{1}{L} \sum_{\ell=1}^L \frac{\alpha_{t_\ell}}{\sigma_{t_\ell}^2} \mathbb{E}_{\mathbf{x}, \mathbf{x}_{t_\ell}} \|\bar{\mathbf{x}}^*(t_\ell, \mathbf{x}_{t_\ell}) - \bar{\mathbf{x}}(t_\ell, \mathbf{x}_{t_\ell})\|_2^2}}}_{\text{average excess error of the denoiser}}\right) \quad (3.2.79)$$

where  $\bar{\mathbf{x}}^*$  is the Bayes optimal denoiser for  $\mathbf{x}$ , and  $\tilde{\mathcal{O}}$  is a version of the big- $\mathcal{O}$  notation which ignores logarithmic factors in  $L$ .

The very high-level proof technique is, as discussed earlier, to bound the error at each step, distinguish the error sources (between discretization and denoiser error) and carefully ensure that the errors do not accumulate too much (or even cancel out).

Note that, if  $L \rightarrow \infty$  and we correctly learn the Bayes optimal denoiser  $\bar{\mathbf{x}} = \bar{\mathbf{x}}^*$  (such that the excess error is 0), then the sampling process in Algorithm 3.1 yields a *perfect (in distribution) inverse* of the noising process, since the error rate in Theorem 3.5 goes to 0,<sup>8</sup> as heuristically argued previously.

*Remark 3.2.* What if the data is low-dimensional, say supported on a *low-rank subspace* of the high dimensional space  $\mathbb{R}^D$ ? If the data distribution is compactly supported—say if the data is normalized to the unit hypercube, which is often ensured as a pre-processing step for real data such as images—it is possible to do better. Namely, the authors of [LY24] also define a measure of *approximate intrinsic dimension* using the asymptotics of the so-called covering number, which is extremely similar in intuition (if not in implementation) to the rate distortion function presented in the next Section. Then they show that using a particular small modification of the DDIM sampler in Algorithm 3.1 (i.e., slightly perturbing the update coefficients), the discretization error becomes

$$\tilde{\mathcal{O}}\left(\frac{\text{approximate intrinsic dimension}}{L}\right) \quad (3.2.80)$$

instead of  $\frac{D}{L}$  like it was in Theorem 3.5. Therefore, using this modified algorithm,  $L$  does not have to be too large even as  $D$  reaches the thousands or millions, since real data has low-dimensional structure. However in practice we use the DDIM sampler instead, so  $L$  should have a mild dependence on  $D$  to achieve consistent error rates. The exact choice of  $L$  trades off between the computational complexity (e.g., runtime or memory consumption) of sampling and the statistical complexity of learning a denoiser for low-dimensional structures. The value of  $L$  is often different at training time (where a larger  $L$  allows better coverage of the interval  $[0, T]$ , which helps the network learn a relationship which generalizes over  $t$ ) and sampling time (where  $L$  being smaller means more efficient sampling). One can even pick the timesteps adaptively at sampling time in order to optimize this tradeoffs [BLZ+22].

*Remark 3.3.* Various other works define the reverse process as moving backward in the time index  $t$  using an explicit difference equation, or differential equation in the limit  $L \rightarrow \infty$ , or forward in time using the transformation  $\mathbf{y}_t = \mathbf{x}_{T-t}$ , such that if  $t$  increases then  $\mathbf{y}_t$  becomes closer to  $\mathbf{x}_0$ . In this work we strive to

<sup>7</sup>The precise definition is rather lengthy in our notation and only defined up to various absolute constants, so we omit it here for brevity. Of course it is in the original paper [LY24].

<sup>8</sup>There are similar results for VE processes, though none are as sharp as this to our knowledge.

keep consistency: we move forward in time to noise, and backward in time to denoise. If you are reading another work which is not clear on the time index, or trying to implement an algorithm which is similarly unclear, there is one way to do it right every time: the sampling process should always have a *positive* coefficient on both the denoiser term and the current iterate when moving from step to step. But in general many papers define their own notation and it is not user-friendly.

*Remark 3.4.* The theory presented at the end of the last Section 3.2.1 seems to suggest (loosely speaking) that in practice, using a transformer-like network is a good choice for learning or approximating a denoiser. This is reasonable, but what is the problem with using any old neural network (such as a multi-layer perceptron (MLP)) and just trying to scale it up to infinity? To observe the problem with this, let us look at another special case of the Gaussian mixture model studied in Example 3.2. Namely, the *empirical distribution* is an instance of a degenerate Gaussian mixture model, with  $K = N$  components  $\mathcal{N}(\mathbf{x}_i, \mathbf{0})$  sampled with equal probability  $\pi_i = \frac{1}{N}$ . In this case the Bayes optimal denoiser is

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \sum_{i=1}^N \frac{e^{-\|\mathbf{x}_t - \alpha_t \mathbf{x}_i\|_2^2 / (2\sigma_t^2)}}{\sum_{j=1}^N e^{-\|\mathbf{x}_t - \alpha_t \mathbf{x}_j\|_2^2 / (2\sigma_t^2)}} \mathbf{x}_i. \quad (3.2.81)$$

This is a convex combination of the data  $\mathbf{x}_i$ , and the coefficients get “sharper” (i.e., closer to 0 or 1) as  $t \rightarrow 0$ . Notice that this denoiser *optimally solves* the denoising optimization problem (3.2.74) when we compute the loss based on drawing  $\mathbf{x}$  uniformly at random from a fixed finite dataset  $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ , which is a very realistic setting. Thus, if our network architecture  $\bar{\mathbf{x}}_\theta$  is expressive enough such that optimal denoisers of the above form (3.2.81) may be well-approximated, then the learned denoiser may do just that. Then, our iterative denoising Algorithm 3.1 will sample exactly from the empirical distribution, re-generating samples in the training data, as certified by Theorem 3.5. This is a bad sampler, not really more interesting than a database of all samples, and so it is important to understand how to avoid this in practice. The key is to come up with a network architecture which can well-approximate the true denoiser (say corresponding to a low-rank distribution as in (3.2.56)) but not the empirical Bayesian denoiser as in (3.2.81). Some work has explored this fine line and why modern diffusion models, which use transformer- and convolutional-based network architectures, can memorize and generalize in different regimes [KG24; NZM+24].

At a high level, a denoiser which memorizes all the training points, as in (3.2.81), corresponds to a parametric model of the distribution which has minimal coding rate, and achieves this by just coding every sample separately. We will discuss this problem (and seeming paradox with our initial desiderata at the end of Section 3.1.3) from the perspective of information theory in the next section.

### 3.3 Compression via Lossy Coding

Let us recap what we have covered so far. We have discussed how to fit a *denoiser*  $\bar{\mathbf{x}}_\theta$  using finite samples. We showed that this denoiser encodes a distribution in that it is directly connected to its log-density via Tweedie’s formula (3.2.20). Then, we used it to gradually transform a pure noise (high-entropy) distribution towards the learned distribution via *iterative denoising*. Thus, we have developed the first way of learning or pursuing a distribution laid out at the end of Section 3.1.3.

Nevertheless, in this methodology the encoding of the distribution is implicit in the denoiser’s functional form and parameters, if any. In fact, acute readers might have noticed that, for a general distribution, we have never explicitly specified what the functional form for the denoiser is. In practice, people typically model it by some deep neural network with an empirically designed architecture. In addition, although we know the above denoising process reduces the entropy, we do not know by how much, nor do we know the entropy of the intermediate and resulting distributions.

Recall that our general goal is to model data from a (continuous) distribution with a low-dimensional support. If our goal is to identify the “simplest” model that generates the data, one could consider three typical measures of parsimony: the dimension, the volume, or the (differential) entropy. Well, if one uses the dimension, then obviously the best model for a given dataset is the empirical distribution itself which is zero-dimensional. For all distributions with low-dimensional supports, the differential entropy is always negative infinity; the volume of their supports are always zero. So, among all distributions of low-dimensional supports that could have generated the same data samples, how can we decide which one is better based

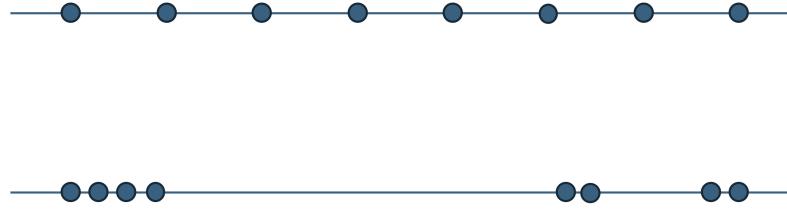


Figure 3.8: Eight points observed on a line.

on these measures of parsimony that cannot distinguish among low-dimensional models at all? This section aims to address this seemingly baffling situation.

In the remainder of this chapter, we discuss a framework that allows us to alleviate the above technical difficulty by associating the learned distribution with an explicit computable encoding and decoding scheme, following the second approach suggested at the end of Section 3.1.3. As we will see, such an approach essentially allows us to accurately approximate the entropy of the learned distributions in terms of a (lossy) coding length or coding rate associated with the coding scheme. With such a measure, not only can we accurately measure how much the entropy is reduced, hence information gained, by any processing (including denoising) of the distribution, but we can also derive an explicit form of the optimal operator that can conduct such operations in the most efficient way. As we will see in the next Chapter 4, this will lead to a principled explanation for the architecture of deep networks, as well as to more efficient deep-architecture designs.

### 3.3.1 Necessity of Lossy Coding

We have previously, multiple times, discussed a difficulty: if we learn the distribution from finite samples in the end, and our function class of denoisers contains enough functions, how do we ensure that we sample from the *true* distribution (with low-dimensional supports), instead of any other distribution that may produce those finite samples with high probability? Let us reveal some of the conceptual and technical difficulties with some concrete examples.

*Example 3.4* (Volume, Dimension, and Entropy). For the example shown on the top of Figure 3.8, suppose we have taken some samples from a uniform distribution on a line (say in a 2D plane). The volume of the line or the sample sets is zero. Geometrically, the empirical distribution on the produced finite sample set is the *minimum-dimension* one which can produce the finite sample set.<sup>9</sup> But this is in seemingly contrast with yet another measure of complexity: entropy. The (differential) entropy of the line is negative infinity but the (discrete) entropy of this sample set is finite and positive. So we seem to have a paradoxical situation according to these common measures of parsimony or complexity: they cannot properly differentiate among (models for) distributions of low-dimensional supports at all, and some seem to differentiate them even in exactly opposite manners.<sup>10</sup> ■

*Example 3.5* (Density). Consider the two sets of sampled data points shown in Figure 3.8. Geometrically, they are essentially the same: each set consists of eight points and each point has occurred with equal frequency 1/8th. The only difference is that for the second data set, some points are “close” enough to be viewed as having a higher density around their respective “cluster.” Which one is more relevant to the true distribution that may have generated the samples? How can we reconcile such ambiguity in interpreting this kind of (empirical) distributions? ■

There is yet another technical difficulty associated with constructing an explicit encoding and decoding scheme for a data set. Given a sampled data set in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , how to design a coding scheme that is implementable on machines with finite memory and computing resources? Note that even representing a general real number requires an infinite number of digits or bits. Therefore, one may wonder whether the

<sup>9</sup>A set of discrete samples are all of zero dimension whereas the supporting line is one dimension.

<sup>10</sup>Of course, strictly speaking, differential entropy and discrete entropy are not directly comparable.

entropy of a distribution is a direct measure for the complexity of its (optimal) coding scheme. We examine this matter with another simple example.

*Example 3.6 (Precision).* Consider a discrete distribution  $\mathbf{X} = [e, \pi]$  with equal probability 1/2 taking the values of the Euler number  $e \approx 2.71828$  or the number  $\pi \approx 3.14159$ . The entropy of this distribution is  $H = 1$ , which suggests that one may encode the two numbers by a one-bit digit 0 or 1, respectively. But can you realize a decoding scheme for this code on a finite-state machine? The answer is actually no, as it takes infinitely many bits to describe either number precisely. ■

Hence, it is generally impossible to have an encoding and decoding scheme that can precisely reproduce samples from an arbitrary real-valued distribution.<sup>11</sup> But there would be little practical value to encode a distribution without being able to decode for samples drawn from the same distribution.

So to ensure that any encoding/decoding scheme is computable and implementable with finite memory and computational resources, we need to quantify the sample  $\mathbf{x}$  and encode it only up to a certain precision, say  $\epsilon > 0$ . *By doing so, in essence, we treat any two data points equivalent if their distance is less than  $\epsilon$ .* More precisely, we would like to consider coding schemes

$$\mathbf{x} \mapsto \hat{\mathbf{x}} \quad (3.3.1)$$

such that the expected error caused by the quantization is bounded by  $\epsilon$ . It is mathematically more convenient, and conceptually almost identical, to bound the expected *squared* error by  $\epsilon^2$ , i.e.,

$$\mathbb{E}[d(\mathbf{x}, \hat{\mathbf{x}})^2] \leq \epsilon^2. \quad (3.3.2)$$

Typically, the distance  $d$  is chosen to be the Euclidean distance, or the 2-norm.<sup>12</sup> We will adopt this choice in the sequel.

### 3.3.2 Rate Distortion and Data Geometry

Of course, among all encoding schemes that satisfy the above constraint, we would like to choose the one that minimizes the resulting coding rate. For a given random variable  $\mathbf{x}$  and a precision  $\epsilon$ , this rate is known as the *rate distortion*, denoted as  $\mathcal{R}_\epsilon(\mathbf{x})$ . A deep theorem in information theory, originally proved by Shannon [Sha59], establishes that this rate can be expressed equivalently in purely probabilistic terms as

$$\mathcal{R}_\epsilon(\mathbf{x}) = \min_{p(\hat{\mathbf{x}}|\mathbf{x}): \mathbb{E}[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2] \leq \epsilon^2} I(\mathbf{x}; \hat{\mathbf{x}}), \quad (3.3.3)$$

where the quantity  $I(\mathbf{x}; \hat{\mathbf{x}})$  is known as the *mutual information*, defined by

$$I(\mathbf{x}; \hat{\mathbf{x}}) = \text{KL}(p(\mathbf{x}, \hat{\mathbf{x}}) \parallel p(\mathbf{x})p(\hat{\mathbf{x}})). \quad (3.3.4)$$

Note that the minimization in (3.3.3) is over all conditional distributions  $p(\hat{\mathbf{x}} | \mathbf{x})$  that satisfy the distortion constraint  $\mathbb{E}_{\mathbf{x}, \hat{\mathbf{x}}}[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2] \leq \epsilon^2$ . Each such conditional distribution induces a joint distribution  $p(\mathbf{x}, \hat{\mathbf{x}}) = p(\hat{\mathbf{x}} | \mathbf{x})p(\mathbf{x})$ , which determines the mutual information (3.3.4). Many convenient properties of the mutual information (and hence the rate distortion) are implied by corresponding properties of the KL divergence (recall Theorem 3.1). From the definition, we know that  $\mathcal{R}_\epsilon(\mathbf{x})$  is a *non-increasing* function in  $\epsilon$ .

*Remark 3.5.* As it turns out, the rate distortion is an implementable approximation to the entropy of  $\mathbf{x}$  in the following sense. Assume that  $\mathbf{x}$  and  $\hat{\mathbf{x}}$  are continuous random vectors. Then the mutual information can be written as

$$I(\mathbf{x}; \hat{\mathbf{x}}) = h(\mathbf{x}) - h(\mathbf{x} | \hat{\mathbf{x}}), \quad (3.3.5)$$

where  $h(\mathbf{x} | \hat{\mathbf{x}}) = \mathbb{E}[\log_2 p(\mathbf{x} | \hat{\mathbf{x}})]$  is the *conditional entropy* of  $\mathbf{x}$  given  $\hat{\mathbf{x}}$ . Hence, the minimal coding rate is achieved when the difference between the entropy of  $\mathbf{x}$  and the conditional entropy of  $\mathbf{x}$  given  $\hat{\mathbf{x}}$  is minimized among all distributions that satisfy the constraint:  $\mathbb{E}[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2] \leq \epsilon^2$ .

---

<sup>11</sup>That is, if one wants to encode such samples precisely, the only way is to memorize every single sample.

<sup>12</sup>More generally, we can replace  $d^2$  with any so-called *divergence*.

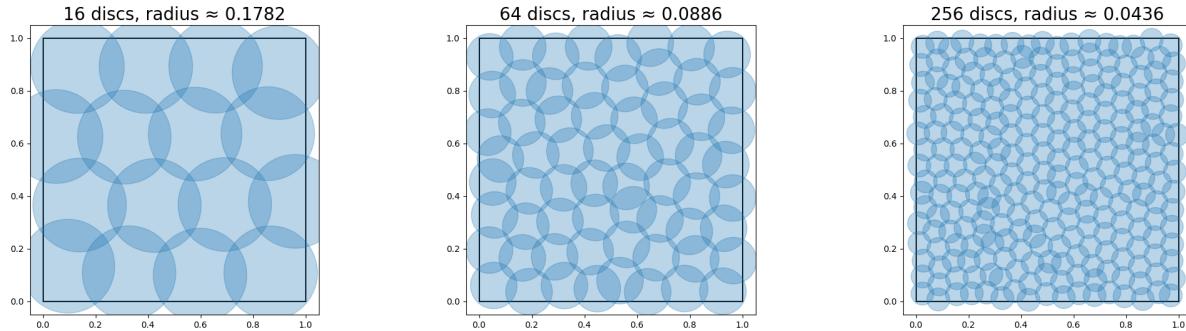


Figure 3.9: Approximations to the optimal solutions for  $2^4$ ,  $2^6$ , and  $2^8$  discs covering a square, along with the corresponding radii, calculated using a heuristic optimization algorithm.

In fact, it is not necessary to assume that  $\mathbf{x}$  and  $\hat{\mathbf{x}}$  are continuous to obtain the above type of conclusion. For example, if both random vectors are instead discrete, we have after a suitable interpretation of the KL divergence for discrete-valued random vectors that

$$I(\mathbf{x}; \hat{\mathbf{x}}) = H(\mathbf{x}) - H(\mathbf{x} | \hat{\mathbf{x}}). \quad (3.3.6)$$

More generally, advanced mathematical notions from measure theory can be used to define the mutual information (and hence the rate distortion) for arbitrary random variables  $\mathbf{x}$  and  $\hat{\mathbf{x}}$ , including those with rather exotic low-dimensional distributions; see Cover and Thomas [CT91, §8.5].

*Remark 3.6.* Given a set of data points in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ , one can always interpret them as samples from a uniform discrete distribution with equal probability  $1/N$  on these  $N$  vectors. The entropy for such a distribution is  $H(\mathbf{X}) = \frac{1}{N} \log_2 N$ .<sup>13</sup> However, even if  $\mathbf{X}$  is a uniform distribution on its samples, the coding rate  $\mathcal{R}_\epsilon(\mathbf{X})$  achievable with a lossy coding scheme could be significantly lower than  $H(\mathbf{X})$  if these samples are not so evenly distributed and many are clustered closely to each other. Therefore, for the second distribution shown in Figure 3.8, for a properly chosen quantization error  $\epsilon$ , the achievable lossy coding rate can be significantly lower than coding it as a uniform distribution.<sup>14</sup> Also notice that, with the notion of rate distortion, the difficulty discussed in Example 3.6 also disappears: We can choose two rational numbers that are close enough to each of the two irrational numbers. The resulting coding scheme will have a finite complexity.

*Example 3.7.* Sometimes, one may face an opposite situation when we want to fix the coding rate first and try to find a coding scheme that minimizes the distortion. For example, suppose that we only want to use a fixed number of codes for points sampled from a distribution, and we want to know how to design the codes such that the average or maximum distortion is minimized during the encoding/decoding scheme. For example, given a uniform distribution on a unit square, we wonder how precisely we can encode points drawn from this distribution, with say  $n$  bits. This problem is equivalent to asking what is the minimum radius (i.e., distortion) such that we can cover the unit square with  $2^n$  discs of this radius. Figure 3.9 shows approximately optimal coverings of a square with  $n = 4, 6, 8$ , so that  $2^n = 16, 64, 256$  discs, respectively. Notice that the optimal radii of the discs decreases as the number of discs  $2^n$  increases. ■

It turns out to be a notoriously hard problem to obtain closed-form expressions for the rate distortion function (3.3.3) for general distributions  $p(\mathbf{x})$ . However, as Example 3.7 suggests, there are important special cases where the *geometry* of the support of the distribution  $p(\mathbf{x})$  can be linked to the rate distortion function, and hence to the optimal coding rate at distortion level  $\epsilon$ . In fact, this example can be generalized to any setting where the support of  $p(\mathbf{x})$  is a sufficiently regular compact set—including low-dimensional distributions—and  $p(\mathbf{x})$  is uniformly distributed on its support. This covers a vast number of cases of practical interest. We formalize this notion in the following result, which establishes this property for a special case.

<sup>13</sup>Note again, even if we can encode these vectors with this coding rate, we cannot decode them with an arbitrary precision.

<sup>14</sup>Nevertheless, for this discrete uniform distribution, when  $\epsilon$  is small enough, we always have  $H(\mathbf{X}) \approx \mathcal{R}_\epsilon(\mathbf{X})$ .

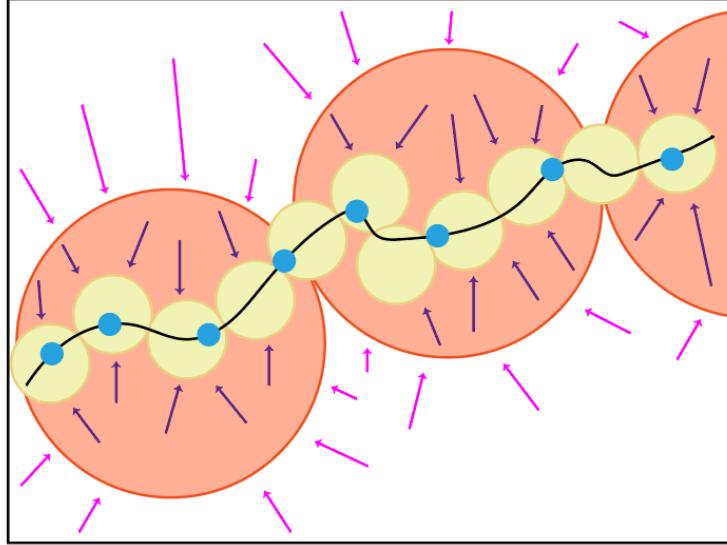


Figure 3.10: **The approximation of a low-dimensional distribution by  $\epsilon$  balls.** We can see that as the  $\epsilon$  parameter shrinks, the union of  $\epsilon$ -balls approximates the support of the true distribution (black) increasingly well. Furthermore, the associated denoisers (whose input-output mapping is given by the provided arrows) obtained by approximating the true distribution by a mixture of Gaussians, each with covariance  $(\epsilon^2/D)\mathbf{I}$ , increasingly well-approximate the true denoisers. At large  $\epsilon$ , such denoisers do not point near the true distribution at all, whereas at small  $\epsilon$  they closely approximate the true denoisers. Theorem 3.6 establishes that this approximation characterizes the rate distortion function at small distortions  $\epsilon$ , unifying the parallel approaches of coding rate minimization and denoising for learning low-dimensional distributions without pathologies.

**Theorem 3.6.** Suppose that  $\mathbf{x}$  is a random variable such that its support  $K \doteq \text{Supp}(\mathbf{x})$  is a compact set. Define the covering number  $\mathcal{N}_\epsilon(K)$  as the minimum number of balls of radius  $\epsilon$  that can cover  $K$ , i.e.,

$$\mathcal{N}_\epsilon(K) \doteq \min \left\{ n \in \mathbb{N}: \exists \mathbf{p}_1, \dots, \mathbf{p}_n \in K \text{ s.t. } K \subseteq \bigcup_{i=1}^n B_\epsilon(\mathbf{p}_i) \right\}, \quad (3.3.7)$$

where  $B_\epsilon(\mathbf{p}) = \{\boldsymbol{\xi} \in \mathbb{R}^D \mid \|\boldsymbol{\xi} - \mathbf{p}\|_2 \leq \epsilon\}$  is the Euclidean ball of radius  $\epsilon$  centered at  $\mathbf{p}$ . Then it holds

$$\mathcal{R}_\epsilon(\mathbf{x}) \leq \log_2 \mathcal{N}_\epsilon(K). \quad (3.3.8)$$

If, in addition,  $\mathbf{x}$  is uniformly distributed on  $K$  and  $K$  is a mixture of mutually orthogonal low-rank subspaces,<sup>15</sup> then a matching lower bound holds:

$$\mathcal{R}_\epsilon(\mathbf{x}) \geq \log_2 \mathcal{N}_\epsilon(K) - O(D). \quad (3.3.9)$$

*Proof.* A proof of this theorem is beyond the scope of this book and we defer it to Section B.3. □

The implication of Theorem 3.6 can be summarized as follows: for sufficiently accurate coding of the distribution of  $\mathbf{x}$ , the minimum rate distortion coding framework is completely characterized by the sphere packing problem on the support of  $\mathbf{x}$ . The core of the proof of Theorem 3.6 can indeed be generalized to more complex distributions such as sufficiently incoherent mixtures of manifolds, but we leave this for a future study. So the rate distortion can be thought of as a “probability-aware” way to approximate the support of the distribution of  $\mathbf{x}$  by a mixture of many small balls.

We now discuss another connection between this and the denoising-diffusion-entropy complexity hierarchy we discussed earlier in the Chapter.

<sup>15</sup>In fact, it is possible to treat highly irregular  $K$ , such as fractals, with a parallel result, but its statement becomes far more technical: c.f. Riegler et al. [RBK18; RKB23]. We give a simple proof in Section B.3 which shows the result for mixtures of subspaces.

*Remark 3.7.* The key ingredient in the proof of the lower bound in Theorem 3.6 is an important result from information theory known as the *Shannon lower bound* for the rate distortion, named after Claude Shannon, who first derived it in a special case [Sha59]. It asserts the following estimate for the rate distortion function, for any random variable  $\mathbf{x}$  with a density  $p(\mathbf{x})$  and finite expected squared norm [LZ94]:

$$\mathcal{R}_\epsilon(\mathbf{x}) \geq h(\mathbf{x}) - \log_2 \text{vol}(B_\epsilon) - C_D, \quad (3.3.10)$$

where  $C_D > 0$  is a constant depending only on  $D$ . Moreover, this lower bound is actually sharp as  $\epsilon \rightarrow 0$ : that is,

$$\lim_{\epsilon \rightarrow 0} \mathcal{R}_\epsilon(\mathbf{x}) - [h(\mathbf{x}) - \log_2 \text{vol}(B_\epsilon) - C_D] = 0. \quad (3.3.11)$$

So when the distortion  $\epsilon$  is small, we can think solely in terms of the Shannon lower bound, rather than the (generally intractable) optimization problem defining the rate distortion (3.3.3).

The Shannon lower bound is the bridge between the coding rate, entropy minimization/denoising, and geometric sphere packing approaches for learning low-dimensional distributions. Notice that in the special case of a uniform density  $p(\mathbf{x})$ , (3.3.10) becomes

$$\mathcal{R}_\epsilon(\mathbf{x}) \geq - \int_K \frac{1}{\text{vol}(K)} \log_2 \frac{1}{\text{vol}(K)} d\xi - \log_2 \text{vol}(B_\epsilon) - C_d \quad (3.3.12)$$

$$= \log_2 \text{vol}(K) / \text{vol}(B_\epsilon) - C_d. \quad (3.3.13)$$

The ratio  $\text{vol}(K) / \text{vol}(B_\epsilon)$  approximates the number of  $\epsilon$ -balls needed to cover  $K$  by a worst-case argument, which is accurate for sufficiently regular sets  $K$  when  $\epsilon$  is small (see Section B.3 for details). Meanwhile, recall the Gaussian denoising model  $\mathbf{x}_\epsilon = \mathbf{x} + \epsilon \mathbf{g}$  from earlier in the Chapter, where  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is independent of  $\mathbf{x}$ . Interestingly, the differential entropy of the joint distribution  $(\mathbf{x}, \mathbf{g})$  can be calculated as

$$h(\mathbf{x}, \mathbf{g}) = - \int p(\xi) p(\gamma) \log_2 p(\xi) p(\gamma) d\xi d\gamma \quad (3.3.14)$$

$$= h(\mathbf{x}) + h(\epsilon \mathbf{g}). \quad (3.3.15)$$

We have seen the Gaussian entropy calculated in Equation (3.1.4): when  $\epsilon$  is small, it is equal, up to additive constants, to the volumetric quantity  $-\log_2 \text{vol}(B_\epsilon)$  we have seen in the Shannon lower bound. In certain special cases (e.g., data supported on incoherent low-rank subspaces), when  $\epsilon$  is small and the support of  $\mathbf{x}$  is sufficiently regular, the distribution of  $\mathbf{x}_\epsilon$  can even be well-approximated locally by the product of the distributions  $p(\mathbf{x})$  and  $p(\mathbf{g})$ , justifying the above computation. Hence the Gaussian denoising process yields yet another interpretation of the Shannon lower bound, as arising from the entropy of a noisy version of  $\mathbf{x}$ , with noise level proportional to the distortion level  $\epsilon$ .

Thus, this finite rate distortion approach via sphere covering re-enables or generalizes all previous measures of complexity of the distribution, allowing us to differentiate between and rank different distributions in a unified way. These interrelated viewpoints are visualized in Figure 3.10.

For a general distribution at finite distortion levels, it is typically impossible to find its rate distortion function in an analytical form. One must often resort to numerical computation<sup>16</sup>. Nevertheless, as we will see, in our context, we often need to know the rate distortion as an explicit function of a set of data points or their representations. This is because we want to use the coding rate as a measure of the goodness of the representations. An explicit analytical form makes it easy to determine how to transform the data distribution to improve the representation. So, we should work with distributions whose rate distortion functions take explicit analytical forms. To this end, we start with the simplest, and also the most important, family of distributions.

### 3.3.3 Lossy Coding Rate for a Low-Dimensional Gaussian

Now suppose we are given a set of data samples in  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$  from any distribution.<sup>17</sup> We would like to come up with a constructive scheme that can encode the data up to certain precision, say

$$\mathbf{x}_i \mapsto \hat{\mathbf{x}}_i, \quad \text{subject to } \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2 \leq \epsilon. \quad (3.3.16)$$

---

<sup>16</sup> Interested readers may refer to [Bla72] for a classic algorithm that computes rate distortion function numerically for a discrete distribution.

<sup>17</sup> Or these data points could be viewed as an (empirical) distribution themselves.

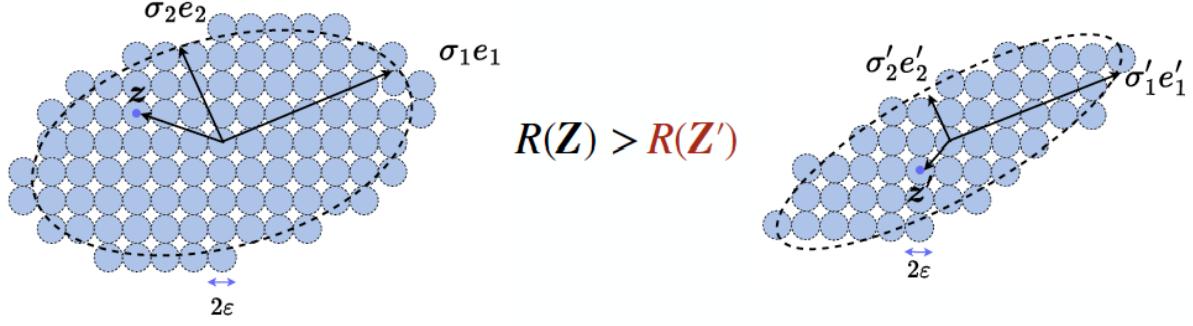


Figure 3.11: Covering the region spanned by the data vectors using  $\epsilon$ -balls. The larger the volume of the space, the more balls are needed, hence the more bits are needed to encode and enumerate the balls. Each real-valued vector  $\mathbf{x}$  can be encoded as the number of the ball which it falls into.

Notice that this is a sufficient, explicit, and interpretable condition which ensures that the data are encoded such that  $\frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 \leq \epsilon^2$ . This latter inequality is exactly the rate distortion constraint for the provided empirical distribution and its encoding. For example, in Example 3.7, we used this simplified criterion to explicitly find the minimum distortion and explicit coding scheme for a given coding rate.

Without loss of generality, let us assume the mean of  $\mathbf{X}$  is zero, i.e.,  $\frac{1}{N} \sum_{i=1}^N \mathbf{x}_i = \mathbf{0}$ . Without any prior knowledge about the nature of the distribution behind  $\mathbf{X}$ , we may view  $\mathbf{X}$  as sampled from a Gaussian distribution  $\mathcal{N}(\mathbf{0}, \Sigma)$  with the covariance<sup>18</sup>

$$\Sigma = \frac{1}{N} \mathbf{X} \mathbf{X}^\top. \quad (3.3.17)$$

Notice that geometrically  $\Sigma$  characterizes an ellipsoidal region where most of the samples  $\mathbf{x}_i$  reside.

We may view  $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_N]$  as a noisy version of  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ :

$$\hat{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{w}_i, \quad (3.3.18)$$

where  $\mathbf{w}_i$  is a Gaussian noise  $\mathbf{w}_i \sim \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{I}/D)$  independent of  $\mathbf{x}_i$ . Then the covariance of  $\hat{\mathbf{x}}_i$  is given by

$$\hat{\Sigma} = \mathbb{E} [\hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^\top] = \frac{\epsilon^2}{D} \mathbf{I} + \frac{1}{N} \mathbf{X} \mathbf{X}^\top. \quad (3.3.19)$$

Note that the volume of the region spanned by the vectors  $\mathbf{x}_i$  is proportional to the square root of the determinant of the covariance matrix

$$\text{volume}(\hat{\mathbf{x}}_i) \propto \sqrt{\det(\hat{\Sigma})} = \sqrt{\det\left(\frac{\epsilon^2}{D} \mathbf{I} + \frac{1}{N} \mathbf{X} \mathbf{X}^\top\right)}. \quad (3.3.20)$$

The volume spanned by each random vector  $\mathbf{w}_i$  is proportional to

$$\text{volume}(\mathbf{w}_i) \propto \sqrt{\det\left(\frac{\epsilon^2}{D} \mathbf{I}\right)}. \quad (3.3.21)$$

To encode vectors that fall into the region spanned by  $\hat{\mathbf{x}}_i$ , we can cover the region with non-overlapping balls of radius  $\epsilon$ , as illustrated in Figure 3.11. When the volume of the region spanned by  $\hat{\mathbf{x}}_i$  is significantly larger than the volume of the  $\epsilon$ -ball, the total number of balls that we need to cover the region is approximately equal to the ratio of the two volumes:

$$\#\epsilon\text{-balls} \approx \frac{\text{volume}(\hat{\mathbf{x}}_i)}{\text{volume}(\mathbf{w}_i)} = \sqrt{\det\left(\mathbf{I} + \frac{D}{N\epsilon^2} \mathbf{X} \mathbf{X}^\top\right)}. \quad (3.3.22)$$

<sup>18</sup>It is known that given a fixed variance, the Gaussian achieves the maximal entropy. That is, it gives an upper bound for what the worst case could be in terms of possible coding rate.

If we use binary numbers to label all the  $\epsilon$ -balls in the region of interest, the total number of binary bits needed is thus

$$\mathcal{R}_\epsilon(\mathbf{X}) \approx \log_2(\# \epsilon\text{-balls}) \approx R_\epsilon(\mathbf{X}) \doteq \frac{1}{2} \log \det \left( \mathbf{I} + \frac{D}{N\epsilon^2} \mathbf{X} \mathbf{X}^\top \right). \quad (3.3.23)$$

*Example 3.8.* Figure 3.11 shows an example of a 2D distribution with an ellipsoidal support – approximating the support of a 2D Gaussian distribution. The region is covered by small balls of size  $\epsilon$ . All the balls are numbered from 1 to say  $n$ . Then given any vector  $\mathbf{x}$  in this region, we only need to determine to which  $\epsilon$ -ball center it is the closest, denoted as  $\text{ball}_\epsilon(\mathbf{x})$ . To remember  $\mathbf{x}$ , we only need to remember the number of this ball, which takes  $\log(n)$  bits to store. If we need to decode  $\mathbf{x}$  from this number, we simply take  $\hat{\mathbf{x}}$  as the center of the ball. This leads to an explicit encoding and decoding scheme:

$$\mathbf{x} \longrightarrow \text{ball}_\epsilon(\mathbf{x}) \longrightarrow \hat{\mathbf{x}} = \text{center of } \text{ball}_\epsilon(\mathbf{x}). \quad (3.3.24)$$

One may refer to these ball centers as “codes” of a code book or a dictionary for the encoding scheme. It is easy to see that the accuracy of this (lossy) encoding-decoding scheme is about the radius of the ball  $\epsilon$ . Clearly  $\mathcal{R}_\epsilon(\mathbf{Z})$  is the average number of bits required to encode the ball number of each vector  $\mathbf{z}$  with this coding scheme, and hence can be called the *coding rate* associated with this scheme. ■

From the above derivation, we know that the coding rate  $\mathcal{R}_\epsilon(\mathbf{X})$  is (approximately) achievable with an explicit encoding (and decoding) scheme. It has two interesting properties:

- First, one may notice that  $R_\epsilon(\mathbf{X})$  closely resembles the rate distortion function of a Gaussian source [CT91]. Indeed, when  $\epsilon$  is small, the above expression is a close approximation to the rate distortion of a Gaussian source, as pointed out by [MDH+07a].
- Second, the same closed-form coding rate  $R_\epsilon(\mathbf{X})$  can be derived as an approximation of  $\mathcal{R}_\epsilon(\mathbf{X})$  if the data  $\mathbf{X}$  are assumed to be from a linear subspace. This can be shown by properly quantifying the singular value decomposition (SVD) of  $\mathbf{X} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^\top$  and constructing a lossy coding scheme for vectors in the subspace spanned by  $\mathbf{U}$  [MDH+07a].

In our context, the closed-form expression  $R_\epsilon(\mathbf{X})$  is rather fundamental: it is the coding rate associated with an explicit and natural lossy coding scheme for data drawn from either a Gaussian distribution or a linear subspace. As we will see in the next chapter, this formula plays an important role in understanding the architecture of deep neural networks.

### 3.3.4 Clustering a Mixture of Low-Dimensional Gaussians

As we have discussed before, the given dataset  $\mathbf{X}$  often has low-dimensional intrinsic structures. Hence, encoding it as a general Gaussian would be very redundant. If we can identify those intrinsic structures in  $\mathbf{X}$ , we could design much better coding schemes that give much lower coding rates. Or equivalently, the codes used to encode such  $\mathbf{X}$  can be compressed. We will see that compression gives a unifying computable way to identify such structures. In this section, we demonstrate this important idea with the most basic family of low-dimensional structures: a mixture of (low-dimensional) Gaussians or subspaces.

*Example 3.9.* Figure 3.12 shows an example in which the data  $\mathbf{X}$  are distributed around two subspaces (or low-dimensional Gaussians). If they are viewed and coded together as one single Gaussian, the associated discrete (lossy) code book, represented by all the blue balls, is obviously very redundant. We can try to identify the locations of the two subspaces, denoted by  $S_1$  and  $S_2$ , and design a code book that only covers the two subspaces, i.e., the green balls. If we can correctly partition samples in the data  $\mathbf{X}$  into the two subspaces:  $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2] \boldsymbol{\Pi}$  with  $\mathbf{X}_1 \in S_1$  and  $\mathbf{X}_2 \in S_2$ , where  $\boldsymbol{\Pi}$  denotes a permutation matrix, then the resulting coding rate for the data will be much lower. This gives a more parsimonious, hence more desirable, representation of the data. ■

So, more generally speaking, if the data are drawn from any mixture of subspaces or low-dimensional Gaussians, it would be desirable to identify those components and encode the data based on the intrinsic dimensions of those components. It turns out that we do not lose much generality by assuming that the data are drawn from a mixture of low-dimensional Gaussians. This is because a mixture of Gaussians can closely approximate most general distributions [BDS16].

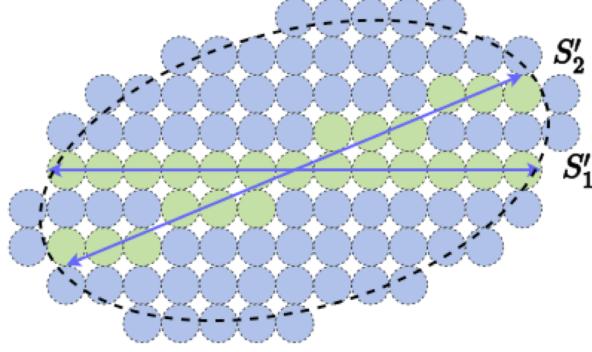


Figure 3.12: Comparison of two lossy coding schemes for data that are distributed around two subspaces. One is to pack (blue)  $\epsilon$ -balls for the entire space spanned by the two subspaces; the other is to pack balls only in a tabular neighborhood around the two subspaces. The latter obviously has a much smaller code book and results in a much lower coding rate for samples on the subspaces.

**The clustering problem.** Now for this specific family of distributions, how can we effectively and efficiently identify those low-dimensional components from a set of samples

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N], \quad (3.3.25)$$

drawn from them? In other words, given the whole data set  $\mathbf{X}$ , we want to partition, or cluster, it into multiple, say  $K$ , subsets:

$$\mathbf{X}\boldsymbol{\Pi} = [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K], \quad (3.3.26)$$

where each subset consists of samples drawn from only one low-dimensional Gaussian or subspace and  $\boldsymbol{\Pi}$  is a permutation matrix to indicate membership of the partition. Note that, depending the situation, the partition could be either deterministic or probabilistic. As shown in [MDH+07b], for mixture of Gaussians, probabilistic partition does not lead to a lower coding rate. So for simplicity, we here consider a deterministic partition only.

**Clustering via lossy compression.** The main difficulty in solving the above clustering problem is that we normally do not know the number of clusters  $K$ , nor do we know the dimension of each component. There has been a long history for the study of this clustering problem. The textbook [VMS16] gives a systematic and comprehensive coverage of different approaches to this problem. To find an effective approach to this problem, we first need to understand and clarify why we want to cluster. In other words, what exactly do we gain from clustering the data, compared with not to? How do we measure the gain? From the perspective of data compression, a correct clustering should lead to a more efficient encoding (and decoding) scheme.

For any given data set  $\mathbf{X}$ , there are already two obvious encoding schemes as the baseline. They represent two extreme ways to encode the data:

- Simply view all the samples together drawn as from one single Gaussian. The associated coding rate is, as derived before, given by:

$$\mathcal{R}_\epsilon(\mathbf{X}) \approx R_\epsilon(\mathbf{X}) = \frac{1}{2} \log \det \left( \mathbf{I} + \frac{D}{N\epsilon^2} \mathbf{X} \mathbf{X}^\top \right). \quad (3.3.27)$$

- Simply memorize all the samples separately by assigning a different number to each sample. The coding rate would be:

$$\mathcal{R}_0(\mathbf{X}) = \log(N). \quad (3.3.28)$$

Note that either coding scheme can become the “optimal” solution for certain (extreme) choice of the quantization error  $\epsilon$ :

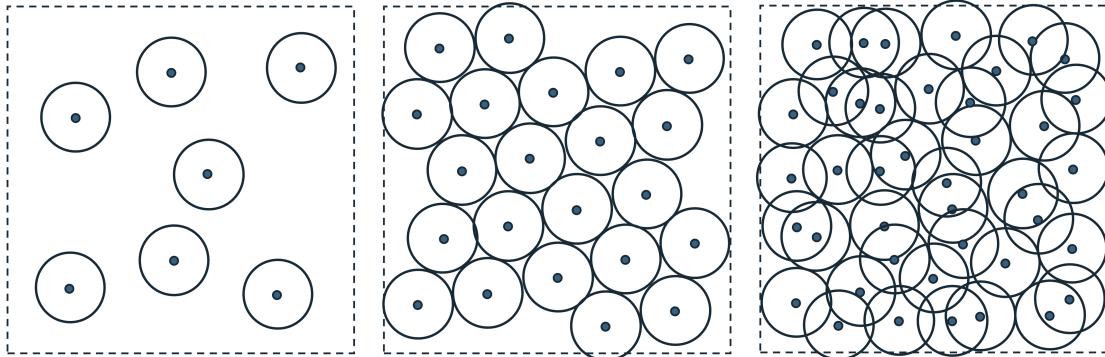


Figure 3.13: A number of random samples on a 2D plane. Consider an  $\epsilon$ -disc assigned to each sample with the sample as its center. The density of the samples increases from left to right.

1. *Lazy Regime*: If we choose  $\epsilon$  to be extremely large, all samples in  $\mathbf{X}$  can be covered by a single ball. The rate is  $\lim_{\epsilon \rightarrow \infty} \mathcal{R}_\epsilon \rightarrow \frac{1}{2} \log \det(\mathbf{I}) = 0$ .
2. *Memorization Regime*: If  $\epsilon$  is extremely small, every sample in  $\mathbf{X}$  is covered by a different  $\epsilon$ -ball, hence the total is  $N$ . The rate is  $\lim_{\epsilon \rightarrow 0} \mathcal{R}_\epsilon \rightarrow \log(N)$ .

Note that the first scheme corresponds to the scenario when one does not care about anything interesting about the distribution at all. One does not want to spare any bit for anything informative. We call this the “lazy regime.” The second scheme corresponds to the scenario when one wants to decode every sample with an extremely high precision. So one would better “memorize” every sample. We call this the “memorization regime.”

*Example 3.10.* To see when the memorization regime is preferred or not, let us consider a number, say  $N$ , of samples randomly distributed in a unit area on a 2D plane.<sup>19</sup> Imagine we try to design a lossy coding scheme with a fixed quantization error  $\epsilon$ . This is equivalent to putting an  $\epsilon$ -disc around each sample, as shown in Figure 3.13. When  $N$  is small, the chance that all the discs overlap with each other is zero. A codebook of size  $N$  is necessary and optimal in this case. When  $N$  or the density reaches a certain critical value  $N_c$ , with high probability all the discs start to overlap and connect into one cluster that covers the whole plane—this phenomenon is known as continuum “percolation” [Gil61; MM12]. When  $N$  becomes larger than this value, the discs overlap heavily. The number  $N$  of discs becomes very redundant because we only want to encode points on the plane up to the given precision  $\epsilon$ . The number of discs needed to cover all the samples is much less than  $N$ .<sup>20</sup> ■

Both the lazy and memorization regimes are somewhat trivial and perhaps are of little theoretical or practical interest. Either scheme would be far from optimal when used to encode a large number of samples drawn from a distribution that has a *compact and low-dimensional support*. The interesting regime exists in between these two.

*Example 3.11.* Figure 3.14 shows an example with noisy samples drawn from two lines and one plane in  $\mathbb{R}^3$ . As we notice from the plot (c) on the right, the optimal coding rate decreases monotonically as we increase  $\epsilon$ , as anticipated from the property of the rate distortion function. The plots (a) and (b) show, when varying  $\epsilon$  from very small (near zero) to very large (towards infinite), the optimal number of clusters when the coding rate is minimal. We can clearly see the lazy regime and the memorization regime on the two ends of the plots. But one can also notice in plot (b), when the quantization error  $\epsilon$  is chosen to be around the level of the true noise variance  $\epsilon_0$ , the optimal number of clusters is the “correct” number three that represents two planes and one subspace. We informally refer to this middle regime as the “generalization regime”. Notice that a sharp phase transition takes place between these regimes.<sup>21</sup> ■

<sup>19</sup>Say the points are drawn by a Poisson process with density  $N$  points per unit area.

<sup>20</sup>In fact, there are efficient algorithms to find such a covering [BBF+01].

<sup>21</sup>So far, to our best knowledge, there is no rigorous theoretical justification for these phase transition behaviors.

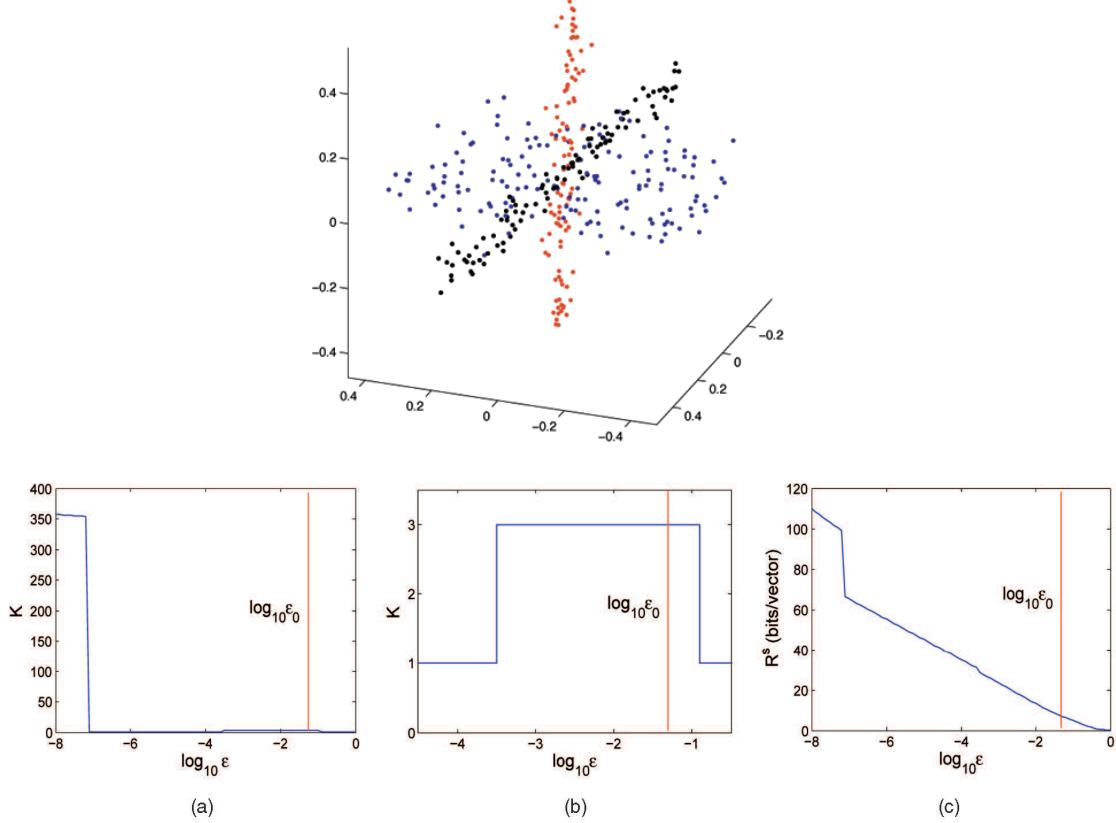


Figure 3.14: Top: 358 noisy samples drawn from two lines and one plane in  $\mathbb{R}^3$ . Bottom: the effect of varying  $\epsilon$  on the clustering result and the coding rate. The red line marks the variance  $\epsilon_0$  of the Gaussian noise added to the samples.

From the above discussion and examples, we see that, when the quantization error relative to the sample density<sup>22</sup> is in a proper range, minimizing the lossy coding rate would allow us to uncover the underlying (low-dimensional) distribution of the sampled data. *Hence, quantization, started as a choice of practicality, seems to be becoming necessary for learning a continuous distribution from its empirical distribution with finite samples.* Although a rigorous theory for explaining this phenomenon remains elusive, here, for learning purposes, we care about how to exploit the phenomenon to design algorithms that can find the correct distribution.

Let us use the simple example shown in Figure 3.12 to illustrate the basic ideas. If one can partition all samples in  $\mathbf{X}$  into two clusters in  $\mathbf{X}_1$  and  $\mathbf{X}_2$ , with  $N_1$  and  $N_2$  samples respectively, then the associated coding rate would be<sup>23</sup>

$$R_\epsilon^c(\mathbf{X} | \boldsymbol{\Pi}) = \frac{N_1}{N} R_\epsilon(\mathbf{X}_1) + \frac{N_2}{N} R_\epsilon(\mathbf{X}_2), \quad (3.3.29)$$

where we use  $\boldsymbol{\Pi}$  to indicate membership of the partition. If the partition respects the low-dimensional structures of the distribution, in this case  $\mathbf{X}_1$  and  $\mathbf{X}_2$  belonging to the two subspaces respectively, then the resulting coding rate should be significantly smaller than the above two basic schemes:

$$R_\epsilon^c(\mathbf{X} | \boldsymbol{\Pi}) \ll R_\epsilon(\mathbf{X}), \quad R_\epsilon^c(\mathbf{X} | \boldsymbol{\Pi}) \ll R_0(\mathbf{X}). \quad (3.3.30)$$

<sup>22</sup>or the sample density relative to the quantization error

<sup>23</sup>We here ignore some overhead bits needed to encode the membership for each sample, say via the Huffman coding.

In general, we can cast the clustering problem into an optimization problem that minimizes the coding rate:

$$\min_{\boldsymbol{\Pi}} \left\{ R_{\epsilon}^c(\mathbf{X} \mid \boldsymbol{\Pi}) \doteq \sum_{k=1}^K \frac{N_k}{N} R_{\epsilon}(\mathbf{X}_k) \right\}. \quad (3.3.31)$$

**Optimization strategies to cluster.** The remaining question is how we optimize the above coding rate objective to find the optimal clusters. There are three natural approaches to this objective:

1. We may start with the whole set  $\mathbf{X}$  as a single cluster (i.e. the lazy regime) and then search (say randomly) to partition it so that it would lead to a smaller coding rate.
2. Inversely, we may start with each sample  $\mathbf{x}_i$  as its own cluster (i.e. the memorization regime) and search to merge clusters that would result in a smaller coding rate.
3. Alternatively, if we could represent (or approximate) the membership  $\boldsymbol{\Pi}$  as some continuous parameters, we may use optimization methods such as gradient descent (GD).

The first approach is not so appealing computationally as the number of possible partitions that one needs to try is exponential in the number of samples. For example, the number of partitions of  $\mathbf{X}$  into two subsets of equal size is  $\binom{N}{N/2}$  which explodes as  $N$  becomes large. We will explore the third approach in the next Chapter 4. There, we will see how the role of deep neural networks, transformers in particular, are connected with the coding rate objective.

The second approach was originally suggested in the work of [MDH+07b]. It demonstrates the benefit of being able to evaluate the coding rate efficiently (say with an analytical form). With it, the (low-dimensional) clusters of the data can be found rather efficiently and effectively via the principle of minimizing coding length (MCL). Note that for a cluster  $\mathbf{X}_k$  with  $N_k$  samples, the length of binary bits needed to encode all the samples in  $\mathbf{X}_k$  is given by:<sup>24</sup>

$$L(\mathbf{X}_k) = N_k R_{\epsilon}(\mathbf{X}_k). \quad (3.3.32)$$

If we have two clusters  $\mathbf{X}_k$  and  $\mathbf{X}_l$ , if we want to code the samples as two separate clusters, the length of binary bits needed is

$$L^c(\mathbf{X}_k, \mathbf{X}_l) = N_k R_{\epsilon}(\mathbf{X}_k) + N_l R_{\epsilon}(\mathbf{X}_l) - N_k \log \frac{N_k}{N_k + N_l} - N_l \log \frac{N_l}{N_k + N_l}.$$

The last two terms are the number of bits needed to encode the memberships of samples according to the Huffman code.

Then, given any two separate clusters  $\mathbf{X}_1$  and  $\mathbf{X}_2$ , we can decide whether to merge them or not based on the difference between the two coding lengths:

$$L(\mathbf{X}_k \cup \mathbf{X}_l) - L^c(\mathbf{X}_k, \mathbf{X}_l) \quad (3.3.33)$$

is positive or negative and  $\mathbf{X}_k \cup \mathbf{X}_l$  denotes the union of the sets of samples in  $\mathbf{X}_k$  and  $\mathbf{X}_l$ . If it is negative, it means the coding length would become smaller if we merge the two clusters into one. This simple fact leads to the following clustering algorithm proposed by [MDH+07b]:

Note that this algorithm is tractable as the total number of (pairwise) comparisons and merges is about  $O(N^2 \log N)$ . However, due to its greedy nature, there is no theoretical guarantee that the process will converge to the globally optimal clustering solution. Nevertheless, as reported in [MDH+07b], in practice, this seemingly simple algorithm works extremely well. The clustering results plotted in Figure 3.14 were actually computed by this algorithm.

*Example 3.12* (Image Segmentation). The above measure of coding length and the associated clustering algorithm assume the data distribution is a mixture of (low-dimensional) Gaussians. Although this seems somewhat idealistic, the measure and algorithm can already be very useful and even powerful in scenarios when the model is (approximately) valid.

---

<sup>24</sup>In fact, a more accurate estimate of the coding length is  $L(\mathbf{X}_k) = (N_k + D)R_{\epsilon}(\mathbf{X}_k)$  where the extra bits are used to encode the basis of the subspace [MDH+07b]. Here we omit this overhead for simplicity.

**Algorithm 3.3** Pairwise Steepest Descent of Coding Length

---

**Input:**  $N$  data points  $\{\mathbf{x}_i\}_{i=1}^N$

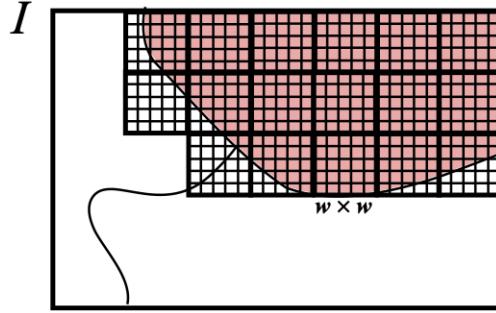
**Output:** A set  $\mathcal{C}$  of clusters

```

1: procedure PAIRWISESTEEPESTDESCENTOFCODINGLENGTH( $\{\mathbf{x}_i\}_{i=1}^N$ )
2:    $\mathcal{C} \leftarrow \{\{\mathbf{x}_i\}_{i=1}^N\}$                                  $\triangleright$  Initialize  $N$  clusters  $\mathbf{X}_k$  with one element each
3:   while  $|\mathcal{C}| > 1$  do
4:     if  $\min_{\mathbf{X}_k, \mathbf{X}_l \in \mathcal{C}} [L(\mathbf{X}_k \cup \mathbf{X}_l) - L^c(\mathbf{X}_k, \mathbf{X}_l)] \geq 0$  then           $\triangleright$  If no bits are saved by any merging
5:       return  $\mathcal{C}$                                                $\triangleright$  Early return  $\mathcal{C}$  and exit
6:     else
7:        $\mathbf{X}_{k^*}, \mathbf{X}_{l^*} \leftarrow \arg \min_{\mathbf{X}_k, \mathbf{X}_l \in \mathcal{C}} [L(\mathbf{X}_k \cup \mathbf{X}_l) - L^c(\mathbf{X}_k, \mathbf{X}_l)]$      $\triangleright$  Merge clusters which save the most bits
8:        $\mathcal{C} \leftarrow [\mathcal{C} \setminus \{\mathbf{X}_{k^*}, \mathbf{X}_{l^*}\}] \cup \{\mathbf{X}_{k^*} \cup \mathbf{X}_{l^*}\}$      $\triangleright$  Remove unmerged clusters and add back the merged
      one
9:     end if
10:    end while
11:    return  $\mathcal{C}$                                                $\triangleright$  If all merges yield savings, return one cluster
12: end procedure

```

---

Figure 3.15: Image patches with a size of  $w \times w$  pixels.

For example, a natural image typically consists of multiple regions with nearly homogeneous textures. If we take many small windows from each region, they should resemble samples drawn from a (low-dimensional) Gaussian, as illustrated in Figure 3.15. Figure 3.16 shows the results of image segmentation based on applying the above clustering algorithm to the image patches directly. More technical details regarding customizing the algorithm to the image segmentation problem can be found in [MRY+11]. ■

## 3.4 Maximizing Information Gain

So far in this chapter, we have discussed how to identify a distribution with low-dimensional structures through the principle of compression. As we have seen from the previous two sections, computational compression can be realized through either the denoising operation or through clustering. Figure 3.17 illustrates this concept with our favorite example. Of course, the ultimate goal for identifying a data distribution is to use it to facilitate certain subsequent tasks such as segmentation, classification, or generation (of images). Hence, how the resulting distribution is “represented” matters tremendously with respect to how information related to these subsequent tasks can be efficiently and effectively retrieved and utilized. This naturally raises a fundamental question: *what makes a representation truly “good” for downstream use?* In the following, we will explore the essential properties that a meaningful and useful representation should possess, and how these properties can be explicitly characterized and pursued via maximizing information gain.

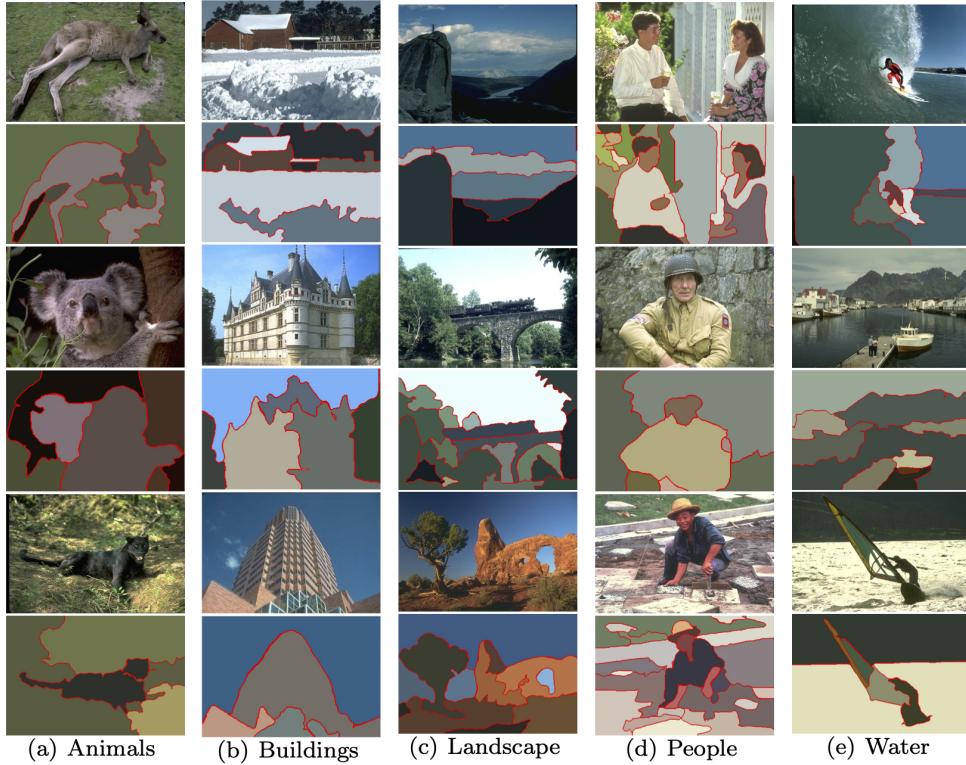


Figure 3.16: Segmentation results based on the clustering algorithm applied to the image patches.

**How to measure the goodness of representations.** One may view a given dataset as samples of a random vector  $\mathbf{x}$  with a certain distribution in a high-dimensional space, say  $\mathbb{R}^D$ . Typically, the distribution of  $\mathbf{x}$  has a much lower intrinsic dimension than the ambient space. Generally speaking, *learning a representation* refers to learning a continuous mapping, say  $f(\cdot)$ , that transforms  $\mathbf{x}$  to a so-called *feature vector*  $\mathbf{z}$  in another (typically lower-dimensional) space, say  $\mathbb{R}^d$ , where  $d < D$ . It is hopeful that through such a mapping

$$\mathbf{x} \in \mathbb{R}^D \xrightarrow{f(\mathbf{x})} \mathbf{z} \in \mathbb{R}^d, \quad (3.4.1)$$

the low-dimensional intrinsic structures of  $\mathbf{x}$  are identified and represented by  $\mathbf{z}$  in a more compact and structured way so as to facilitate subsequent tasks such as classification or generation. The feature  $\mathbf{z}$  can be viewed as a (learned) compact code for the original data  $\mathbf{x}$ , so the mapping  $f$  is also called an *encoder*. The fundamental question of representation learning is

*What is a principled and effective measure for the goodness of representations?*

Conceptually, the quality of a representation  $\mathbf{z}$  depends on how well it identifies the most relevant and sufficient information of  $\mathbf{x}$  for subsequent tasks and how efficiently it represents this information. For a long time, it was believed and argued that the “sufficiency” or “goodness” of a learned feature representation should be defined in terms of a specific task. For example,  $\mathbf{z}$  just needs to be sufficient for predicting the class label  $\mathbf{y}$  in a classification problem. Below, let us start with the classic problem of image classification and argue why such a notion of a task-specific “representation” is limited and needs to be generalized.

### 3.4.1 Linear Discriminative Representations

Suppose that  $\mathbf{x} \in \mathbb{R}^D$  is a random vector drawn from a mixture of  $K$  (component) distributions  $\mathcal{D} = \{\mathcal{D}_k\}_{k=1}^K$ . Give a finite set of i.i.d. samples  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$  of the random vector  $\mathbf{x}$ , we seek a good representation through a continuous mapping  $f(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^d$  that captures intrinsic structures of  $\mathbf{x}$  and best

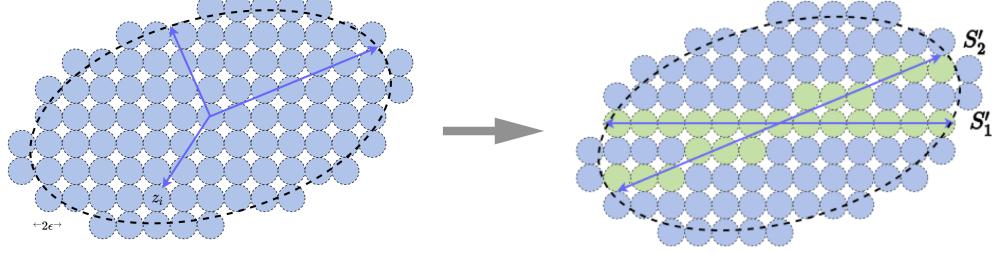


Figure 3.17: Identify a low-dimensional distribution with two subspaces (left) via denoising or clustering, starting from a generic random Gaussian distribution (right).

facilitates the subsequent classification task.<sup>25</sup> To ease the task of learning distribution  $\mathcal{D}$ , in the popular supervised classification setting, a true class label (or a code word for each class), usually represented by a one-hot vector  $\mathbf{y}_i \in \mathbb{R}^K$ , is given for each sample  $\mathbf{x}_i$ .

**Encoding class information via cross entropy.** Extensive studies have shown that for many practical datasets (e.g., images, audio, and natural languages), the (encoding) mapping from the data  $\mathbf{x}$  to its class label  $\mathbf{y}$  can be effectively modeled by training a deep network,<sup>26</sup> here denoted as

$$f(\mathbf{x}, \theta) : \mathbf{x} \mapsto \mathbf{y}$$

with network parameters  $\theta \in \Theta$ , where  $\Theta$  denotes the parameter space. For the output  $f(\mathbf{x}, \theta)$  to match well with the label  $\mathbf{y}$ , we like to minimize the *cross-entropy loss* over a training set  $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ :

$$\min_{\theta \in \Theta} -\mathbb{E}[\langle \mathbf{y}, \log(f(\mathbf{x}, \theta)) \rangle] \approx -\frac{1}{N} \sum_{i=1}^N \langle \mathbf{y}_i, \log(f(\mathbf{x}_i, \theta)) \rangle. \quad (3.4.2)$$

The optimal network parameters  $\theta$  is typically found by optimizing the above objective through an efficient gradient descent scheme, with gradients computed via back propagation (BP), as described in Section A.2.3 of Appendix A.

Despite its effectiveness and enormous popularity, there are two serious limitations with this approach: 1) It aims only to predict the labels  $\mathbf{y}$  even if they might be mislabeled. Empirical studies show that deep networks, used as a “black box,” can even fit random labels [ZBH+17]. 2) With such an end-to-end data fitting, despite plenty of empirical efforts in trying to interpret the so-learned features, it is not clear to what extent the intermediate features learned by the network capture the intrinsic structures of the data that make meaningful classification possible in the first place. The precise geometric and statistical properties of the learned features are also often obscured, which leads to the lack of interpretability and subsequent performance guarantees (e.g., generalizability, transferability, and robustness, etc.) in deep learning. Therefore, *one of the goals of this section is to address such limitations by reformulating the objective towards learning explicitly meaningful and useful representations for the data  $\mathbf{x}$ , not limited to classification.*

**Minimal discriminative features via information bottleneck.** One popular approach to interpret the role of deep networks is to view outputs of intermediate layers of the network as selecting certain latent features  $\mathbf{z} = f(\mathbf{x}, \theta) \in \mathbb{R}^d$  of the data that are discriminative among multiple classes. Learned representations  $\mathbf{z}$  then facilitate the subsequent classification task for predicting the class label  $\mathbf{y}$  by optimizing a classifier  $g(\mathbf{z})$ :

$$\mathbf{x} \xrightarrow{f(\mathbf{x}, \theta)} \mathbf{z} \xrightarrow{g(\mathbf{z})} \mathbf{y}. \quad (3.4.3)$$

<sup>25</sup>Classification is the domain where deep learning demonstrated the initial success, sparking the explosive interest in deep networks. Although our study focuses on classification, we believe the ideas and principles can be naturally generalized to other settings, such as regression.

<sup>26</sup>Here let us not worry about yet which network we should use here and why. The purpose here is to consider any empirically tested deep network. We will leave the justification of the network architectures to the next chapter.

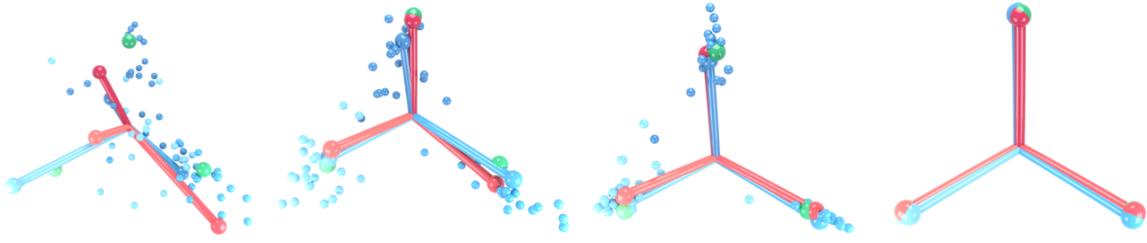


Figure 3.18: Evolution of penultimate layer outputs of a VGG13 neural network when trained on the CIFAR10 dataset with 3 randomly selected classes. Figure from [PHD20].

We know from information theory [CT91] that *the mutual information* between two random variables, say  $\mathbf{x}, \mathbf{z}$ , is defined to be

$$I(\mathbf{x}; \mathbf{z}) = H(\mathbf{x}) - H(\mathbf{x} | \mathbf{z}), \quad (3.4.4)$$

where  $H(\mathbf{x} | \mathbf{z})$  is the conditional entropy of  $\mathbf{x}$  given  $\mathbf{z}$ . The mutual information is also known as *the information gain*: It measures how much the entropy of the random variable  $\mathbf{x}$  can be reduced once  $\mathbf{z}$  is given. Or equivalently, it measures how much information  $\mathbf{z}$  contains about  $\mathbf{x}$ . The *information bottleneck* (IB) formulation [TZ15] further hypothesizes that the role of the network is to learn  $\mathbf{z}$  as the minimal sufficient statistics for predicting  $\mathbf{y}$ . Formally, it seeks to maximize the mutual information  $I(\mathbf{z}, \mathbf{y})$  between  $\mathbf{z}$  and  $\mathbf{y}$  while minimizing the mutual information  $I(\mathbf{x}, \mathbf{z})$  between  $\mathbf{x}$  and  $\mathbf{z}$ :

$$\max_{\theta \in \Theta} \text{IB}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \doteq I(\mathbf{z}; \mathbf{y}) - \beta I(\mathbf{x}; \mathbf{z}) \quad \text{s.t. } \mathbf{z} = f(\mathbf{x}, \theta), \quad (3.4.5)$$

where  $\beta > 0$ .

Given one can overcome some caveats associated with this framework [KTV18], such as how to accurately evaluate mutual information with finite samples of degenerate distributions, this framework can be helpful in explaining certain behaviors of deep networks. For example, recent work [PHD20] indeed shows that the representations learned via the cross-entropy loss (3.4.2) exhibit a *neural collapse* phenomenon. That is, features of each class are mapped to a one-dimensional vector whereas all other information of the class is suppressed, as illustrated in Figure 3.18.

*Remark 3.8.* Neural collapse refers to a phenomenon observed in deep neural networks trained for classification, where the learned feature representations and classifier weights exhibit highly symmetric and structured behavior during the terminal phase of training [PHD20; ZDZ+21]. Specifically, within each class, features collapse to their class mean, and across classes, these means become maximally separated, forming a simplex equiangular configuration. The linear classifier aligns with the class mean up to rescaling. Additionally, the last-layer classifier converges to choosing whichever class has the nearest train class mean. Neural collapse reveals deep connections between optimization dynamics, generalization, and geometric structures arising in supervised learning.

From the above example of classification, we see that the so-learned representation gives a very simple encoder that essentially maps each class of data to only one code word: the one-hot vector representing each class. From the lossy compression perspective, such an encoder is too lossy to preserve information in the data distribution. Other information, such as that useful for tasks such as image generation, is severely lost in such a supervised learning process. To remedy this situation, we want to learn a different encoding scheme such that the resulting feature representation can capture much richer information about the data distribution, not limited to that useful for classification alone.

**Linear discriminative representations.** Whether the given data  $\mathbf{X}$  of a mixed distribution  $\mathcal{D}$  can be effectively classified or clustered depends on how separable (or discriminative) the component distributions  $\mathcal{D}_k$  are (or can be made). One popular working assumption is that the distribution of each class has relatively *low-dimensional* intrinsic structures. Hence we may assume that the distribution  $\mathcal{D}_k$  of each class has a support on a low-dimensional submanifold, say  $\mathcal{M}_k$  with dimension  $d_k \ll D$ , and the distribution  $\mathcal{D}$

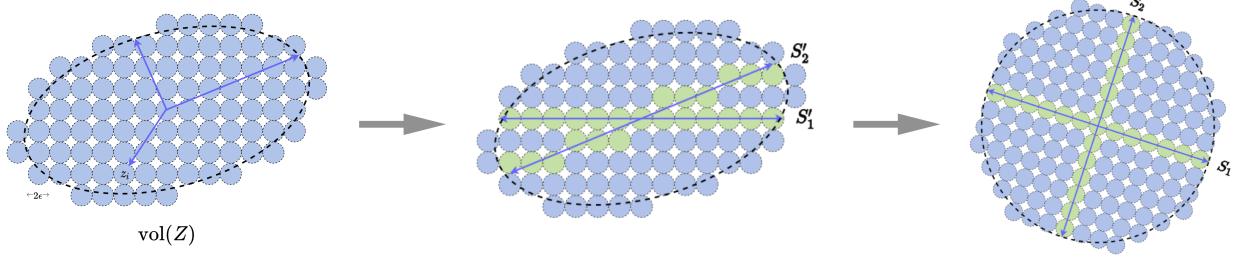


Figure 3.19: After identifying the low-dimensional data distribution, we would like to further transform the data distribution to a more informative structure representation:  $R$  is the number of  $\epsilon$ -balls covering the whole space and  $R^c$  is the sum of the numbers for all the subspaces (the green balls).  $\Delta R$  is their difference (the number of blue balls).

of  $\mathbf{x}$  is supported on the mixture of those submanifolds,  $\mathcal{M} = \cup_{k=1}^K \mathcal{M}_k$ , in the high-dimensional ambient space  $\mathbb{R}^D$ .

Not only do we need to identify the low-dimensional distribution, but we also want to represent the distribution in a form that best facilitates subsequent tasks such as classification, clustering, and conditioned generation (as we will see in the future). To do so, we require our learned feature representations to have the following properties:

1. *Within-Class Compressible*: Features of samples from the same class should be strongly *correlated* in the sense that they belong to a low-dimensional linear subspace.
2. *Between-Class Discriminative*: Features of samples from different classes should be highly *uncorrelated* and belong to different low-dimensional linear subspaces.
3. *Maximally Diverse Representation*: Dimension (or variance) of the features of each class should be *as large as possible* as long as they are incoherent to the other classes.

We refer to such a representation the *linear discriminative representation* (LDR). Notice that the first property aligns well with the objective of the classic *principal component analysis* (PCA) that we have discussed in Chapter 2.1.1. The second property resembles that of the classic *linear discriminant analysis* (LDA) [HTF09]. Figure 3.19 illustrates these properties with a simple example when the data distribution is actually a mixture of two subspaces. Through compression (denoising or clustering), we first identify that the true data distribution is a mixture of two low-dimensional subspaces (middle) instead of a generic Gaussian distribution (left). We then would like to transform the distribution so that the two subspaces eventually become mutually incoherent/independent (right).

*Remark 3.9.* Linear discriminant analysis (LDA) [HTF09] is a supervised dimensionality reduction technique that aims to find a linear projection of data that maximizes class separability. Specifically, given labeled data, LDA seeks a linear transformation that projects high-dimensional inputs onto a lower-dimensional space where the classes are maximally separated. Note that PCA is an unsupervised method that projects data onto directions of maximum variance without considering class labels. While PCA focuses purely on preserving global variance structure, LDA explicitly exploits label information to enhance discriminative power; see the comparison in Figure 3.20.

The third property is also important because we want the learned features to reveal all possible causes of why one class is different from all other classes. For example, to tell “apple” from “orange”, we care not only about color but also shape and the leaves. Ideally, the dimension of each subspace  $\{\mathcal{S}_k\}$  should be equal to that of the corresponding submanifold  $\mathcal{M}_k$ . This property will be important if we would like the map  $f(\mathbf{x}, \theta)$  to be *invertible* for tasks such as image generation. For example, if we draw different sample points from the feature subspace for “apple”, we should be able to decode them to generate diverse images of apples. The feature learned from minimizing the cross entropy (3.4.2) clearly does not have this property.

In general, although the intrinsic structures of each class/cluster may be low-dimensional, they are by no means simply linear (or Gaussian) in their original representation  $\mathbf{x}$  and they need to be made linear first,

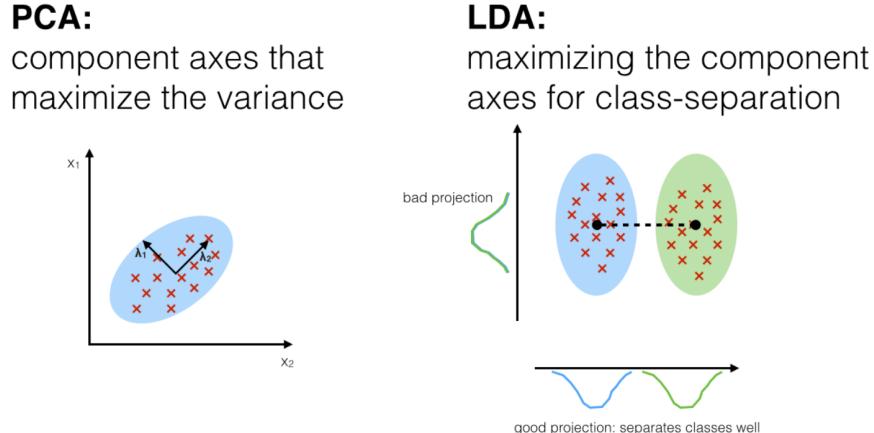


Figure 3.20: Comparison between PCA and LDA. Figures adopted from [https://sebastianraschka.com/Articles/2014\\_python\\_lda.html](https://sebastianraschka.com/Articles/2014_python_lda.html).

through some nonlinear transformation.<sup>27</sup> Therefore, overall, we use the nonlinear transformation  $f(\mathbf{x}, \theta)$  to seek a representation of the data such that the subspaces that represent all the classes are maximally incoherent linear subspaces. To be more precise, we want to learn a mapping  $\mathbf{z} = f(\mathbf{x}, \theta)$  that maps each of the submanifolds  $\mathcal{M}_k \subset \mathbb{R}^D$  (Figure 3.21 left) to a *linear* subspace  $\mathcal{S}_k \subset \mathbb{R}^d$  (Figure 3.21 right). To some extent, the resulting multiple subspaces  $\{\mathcal{S}_k\}$  can be viewed as discriminative *generalized principal components* [VMS16] or, if orthogonal, *independent components* [HO00a] of the resulting features  $\mathbf{z}$  for the original data  $\mathbf{x}$ . As we will see in the next Chapter 4, deep networks precisely play the role of modeling and realizing this nonlinear transformation from the data distribution to linear discriminative representations.

### 3.4.2 The Principle of Maximal Coding Rate Reduction

Although the three properties—*between-class discriminative*, *within-class compressible*, and *maximally diverse representation*—for linear discriminative representations (LDRs) are all highly desired properties of the learned representation  $\mathbf{z}$ , they are by no means easy to obtain: Are these properties compatible so that we can expect to achieve them all at once? If so, is there a *simple but principled* objective that can measure the goodness of the resulting representations in terms of all these properties? The key to these questions is to find a principled “measure of compactness” or “information gain” for the distribution of a random variable  $\mathbf{z}$  or from its finite samples  $\{\mathbf{z}_i\}_{i=1}^N$ . Such a measure should directly and accurately characterize intrinsic geometric or statistical properties of the distribution, in terms of its intrinsic dimension or volume. Unlike the cross entropy (3.4.2) or information bottleneck (3.4.5), such a measure should not depend exclusively on class labels so that it can work in more general settings such as supervised, self-supervised, semi-supervised, and unsupervised settings.

Without loss of generality, assume that the distribution  $\mathcal{D}$  of the random vector  $\mathbf{x}$  is supported on a mixture of distributions, i.e.,  $\mathcal{D} = \cup_{k=1}^K \mathcal{D}_k$ , where each  $\mathcal{D}_k \subset \mathbb{R}^D$  has a low intrinsic dimension in the high-dimensional ambient space  $\mathbb{R}^D$ . Let  $\mathbf{X}_k \in \mathbb{R}^{D \times N_k}$  denote the data matrix whose columns are samples drawn from the distribution  $\mathcal{D}_k$ , where  $N_k$  denotes the number of samples for each  $k = 1, \dots, K$ . Then, we use  $\mathbf{X} = [\mathbf{X}_1, \dots, \mathbf{X}_K] \in \mathbb{R}^{D \times N}$  to denote all the samples, where  $N = \sum_{k=1}^K N_k$ . Recall that we also use  $\mathbf{x}_i$  to denote the  $i$ -th sample of  $\mathbf{X}$ , i.e.,  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ . Under an encoding mapping:

$$\mathbf{x} \xrightarrow{f(\mathbf{x})} \mathbf{z}, \quad (3.4.6)$$

the input samples are mapped to  $\mathbf{z}_i = f(\mathbf{x}_i)$  for each  $i = 1, \dots, N$ . With an abuse of notation, we also write  $\mathbf{Z}_k = f(\mathbf{X}_k)$  and  $\mathbf{Z} = f(\mathbf{X})$ . Therefore, we have  $\mathbf{Z} = [\mathbf{Z}_1, \dots, \mathbf{Z}_K]$  and  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$ .

On one hand, for learned features to be discriminative, features of different classes/clusters are preferred to be *maximally incoherent* to each other. Hence, they together should span a space of the largest possible

<sup>27</sup>We will discuss how this can be done explicitly in Chapter 5.

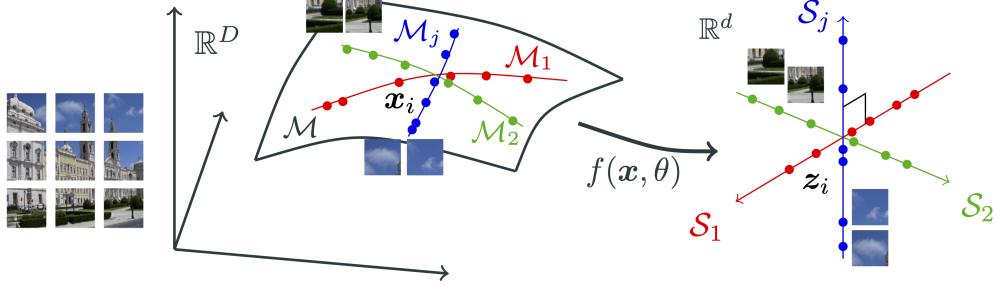


Figure 3.21: The distribution  $\mathcal{D}$  of high-dimensional data  $\mathbf{x} \in \mathbb{R}^D$  is supported on a manifold  $\mathcal{M}$  and its classes on low-dimensional submanifolds  $\mathcal{M}_k$ . We aim to learn a mapping  $f(\mathbf{x}, \theta)$  parameterized by  $\theta$  such that  $\mathbf{z}_i = f(\mathbf{x}_i, \theta)$  lie on a union of maximally uncorrelated subspaces  $\{\mathcal{S}_k\}$ .

volume (or dimension) and the coding rate of the whole set  $\mathbf{Z}$  should be as large as possible. On the other hand, learned features of the same class/cluster should be highly correlated and coherent. Hence, each class/cluster should only span a space (or subspace) of a very small volume and the coding rate should be as small as possible. Now, we will introduce how to measure the coding rate of the learned features.

**Coding rate of features.** Notably, a practical challenge in evaluating the coding rate is that the underlying distribution of the feature representations  $\mathbf{Z}$  is typically unknown. To address this, we may approximate the features  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$  as samples drawn from a multivariate Gaussian distribution. Under this assumption, as discussed in Chapter 3.3.3, the compactness of the features  $\mathbf{Z}$  as a whole can be measured in terms of the average coding length per sample, referred to as the *coding rate*, subject to a precision level  $\epsilon > 0$  (see (3.3.23)) defined as follows:

$$R_\epsilon(\mathbf{Z}) = \frac{1}{2} \log \det \left( \mathbf{I} + \frac{d}{N\epsilon^2} \mathbf{Z} \mathbf{Z}^\top \right). \quad (3.4.7)$$

On the other hand, we hope that a nonlinear transformation  $f(\mathbf{x})$  maps each class-specific submanifold  $\mathcal{M}_k \subset \mathbb{R}^D$  to a maximally incoherent linear subspace  $\mathcal{S}_k \subset \mathbb{R}^d$  such that the learned features  $\mathbf{Z}$  lie in a union of low-dimensional subspaces. This structure allows for a more accurate evaluation of the coding rate by analyzing each subspace separately. Recall that the columns of  $\mathbf{Z}_k$  denotes the features of the samples in  $\mathbf{X}_k$  for each  $k = 1, \dots, K$ . The coding rate for the features in  $\mathbf{Z}_k$  can be computed as follows:

$$R_\epsilon(\mathbf{Z}_k) = \frac{N_k}{2N} \log \det \left( \mathbf{I} + \frac{d}{N_k \epsilon^2} \mathbf{Z}_k \mathbf{Z}_k^\top \right) \quad (3.4.8)$$

Then, the sum of the average coding rates of features in each class is

$$R_\epsilon^c(\mathbf{Z}) \doteq \sum_{k=1}^K R_\epsilon(\mathbf{Z}_k), \quad (3.4.9)$$

Therefore, a good representation  $\mathbf{Z}$  of  $\mathbf{X}$  is the one that achieves a large difference between the coding rate for the whole and that for all the classes:

$$\Delta R_\epsilon(\mathbf{Z}) \doteq R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z}). \quad (3.4.10)$$

Notice that, as per our discussions earlier in this chapter, this difference can be interpreted as the amount of “information gained” by identifying the correct low-dimensional clusters  $\mathbf{Z}_k$  within the overall set  $\mathbf{Z}$ .

If we choose our feature mapping  $f(\cdot)$  to be a deep neural network  $f(\cdot, \theta)$  with network parameters  $\theta$ , the overall process of the feature representation and the resulting rate reduction can be illustrated by the following diagram:

$$\mathbf{X} \xrightarrow{f(\mathbf{x}, \theta)} \mathbf{Z} \xrightarrow{\epsilon} \Delta R_\epsilon(\mathbf{Z}). \quad (3.4.11)$$

Note that  $\Delta R_\epsilon$  is *monotonic* in the scale of the features  $\mathbf{Z}$ . To ensure fair comparison across different representations, it is essential to *normalize the scale* of the learned features. This can be achieved by either imposing the Frobenius norm of each class  $\mathbf{Z}_k$  to scale with the number of features in  $\mathbf{Z}_k \in \mathbb{R}^{d \times N_k}$ , i.e.,  $\|\mathbf{Z}_k\|_F^2 = N_k$ , or by normalizing each feature to be on the unit sphere, i.e.,  $\mathbf{z}_i \in \mathbb{S}^{d-1}$ , where  $N_k = \text{tr}(\mathbf{\Pi}_k)$  denotes the number of samples in the  $k$ -th class. This formulation offers a natural justification for the need for “batch normalization” in the practice of training deep neural networks [IS15].

Once the representations are comparable, the goal becomes to learn a set of features  $\mathbf{Z} = f(\mathbf{X}, \theta)$  such that they maximize the reduction between the coding rate of all features and that of the sum of features w.r.t. their classes:

$$\begin{aligned} \max_{\theta} \Delta R_\epsilon(\mathbf{Z}) &\doteq R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z}), \\ \text{s.t. } \mathbf{Z} &= f(\mathbf{X}, \theta), \quad \|\mathbf{Z}_k\|_F^2 = N_k, \quad k = 1, \dots, K. \end{aligned} \tag{3.4.12}$$

We refer to this as the principle of *maximal coding rate reduction* (MCR<sup>2</sup>), a true embodiment of Aristotle’s famous quote:

“The whole is greater than the sum of its parts.”

To learn the best representation, we require that *the whole is maximally greater than the sum of its parts*. Let us examine the example shown in Figure 3.19 again. From a compression perspective, the representation on the right is *the most compact one* in the sense that the difference between the coding rate when all features are encoded as a single Gaussian (blue) and that when the features are properly clustered and encoded as two separate subspaces (green) is maximal.<sup>28</sup>

Note that the above MCR<sup>2</sup> principle is designed for supervised learning problems, where the group memberships (or class labels) are known. However, this principle can be naturally extended to unsupervised learning problems by introducing a membership matrix, which encodes the (potentially soft) assignment of each data point to latent groups or clusters. Specifically, let  $\mathbf{\Pi} = \{\mathbf{\Pi}_k\}_{k=1}^K \subset \mathbb{R}^{N \times N}$  be a set of diagonal matrices whose diagonal entries encode the membership of the  $N$  samples into  $K$  classes. That is,  $\mathbf{\Pi}$  lies in a simplex  $\Omega \doteq \{\mathbf{\Pi} : \mathbf{\Pi}_k \geq \mathbf{0} : \sum_{k=1}^K \mathbf{\Pi}_k = \mathbf{I}_N\}$ . Then, we can define the average coding rate with respect to the partition  $\mathbf{\Pi}$  as

$$R_\epsilon^c(\mathbf{Z} | \mathbf{\Pi}) \doteq \sum_{k=1}^K \frac{\text{tr}(\mathbf{\Pi}_k)}{2N} \log \det \left( \mathbf{I} + \frac{d}{\text{tr}(\mathbf{\Pi}_k)\epsilon^2} \mathbf{Z}\mathbf{\Pi}_k\mathbf{Z}^\top \right). \tag{3.4.13}$$

When  $\mathbf{Z}$  is given,  $R_\epsilon^c(\mathbf{Z} | \mathbf{\Pi})$  is a concave function of  $\mathbf{\Pi}$ . Then the MCR<sup>2</sup> principle for unsupervised learning problems becomes as follows:

$$\begin{aligned} \max_{\mathbf{\Pi}, \theta} \Delta R_\epsilon(\mathbf{Z} | \mathbf{\Pi}) &\doteq R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z} | \mathbf{\Pi}) \\ \text{s.t. } \mathbf{Z} &= f(\mathbf{X}, \theta), \quad \|\mathbf{Z}\mathbf{\Pi}_k\|_F^2 = N_k, \quad k = 1, \dots, K, \quad \mathbf{\Pi} \in \Omega. \end{aligned} \tag{3.4.14}$$

Compared to (3.4.12), the formulation here allows for the joint optimization of both the group memberships and the network parameters. In particular, when  $\mathbf{\Pi}$  is fixed to a group membership matrix that assigns  $N$  data points into  $K$  groups, Problem (3.4.14) can recover Problem (3.4.12).

### 3.4.3 Optimization Properties of Coding Rate Reduction

In this subsection, we study the optimization properties of the MCR<sup>2</sup> function by analyzing its optimal solutions and the structure of its optimization landscape. To get around the technical difficulty introduced by the neural networks, we consider a simplified version of Problem (3.4.12) as follows:

$$\max_{\mathbf{Z}} R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z}) \quad \text{s.t. } \|\mathbf{Z}_k\|_F^2 = N_k, \quad k = 1, \dots, K. \tag{3.4.15}$$

In theory, the MCR<sup>2</sup> principle (3.4.15) benefits from great generalizability and can be applied to representations  $\mathbf{Z}$  of *any* distributions as long as the rates  $R_\epsilon$  and  $R_\epsilon^c$  for the distributions can be accurately and

---

<sup>28</sup>Intuitively, the ratio between the “volume” of the whole space spanned by all features and that actually occupied by the features is maximal.

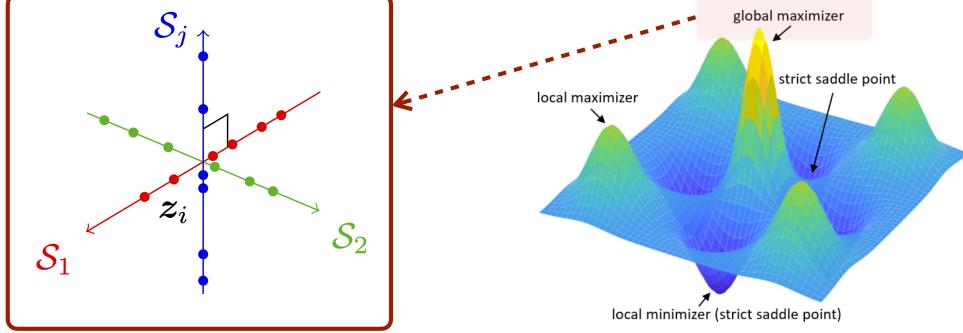


Figure 3.22: **Local optimization landscape:** According to Theorem 3.7, the global maximum of the rate reduction objective corresponds to a solution with mutually incoherent subspaces.

efficiently evaluated. The optimal representation  $\mathbf{Z}^*$  should have some interesting geometric and statistical properties. We here reveal nice properties of the optimal representation with the special case of subspaces, which have many important use cases in machine learning. When the desired representation for  $\mathbf{Z}$  is multiple subspaces, the rates  $R_\epsilon$  and  $R_\epsilon^c$  in (3.4.15) are given by (3.4.7) and (3.4.9), respectively. At the maximal rate reduction, MCR<sup>2</sup> achieves its optimal representations, denoted as  $\mathbf{Z}^* = [\mathbf{Z}_1^*, \dots, \mathbf{Z}_K^*]$  with  $\text{rank}(\mathbf{Z}_k^*) \leq d_k$ . One can show that  $\mathbf{Z}^*$  has the following desired properties (see [YCY+20] for a formal statement and detailed proofs).

**Theorem 3.7 (Characterization of Global Optimal Solutions).** *Suppose  $\mathbf{Z}^* = [\mathbf{Z}_1^*, \dots, \mathbf{Z}_K^*]$  is a global optimal solution of Problem (3.4.15). The following statements hold:*

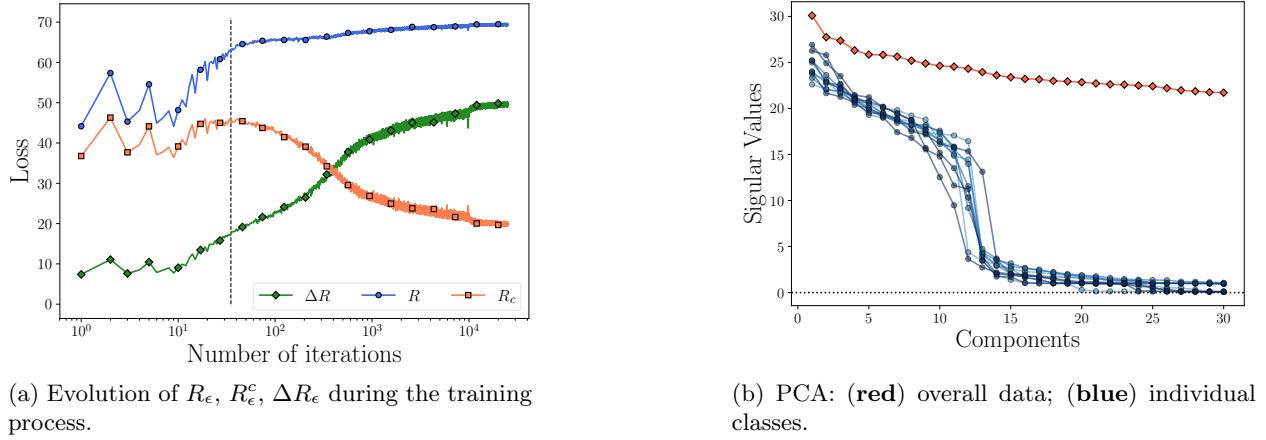
- Between-Class Discriminative: *As long as the ambient space is adequately large ( $d \geq \sum_{k=1}^K d_k$ ), the subspaces are all orthogonal to each other, i.e.,  $(\mathbf{Z}_k^*)^\top \mathbf{Z}_l^* = \mathbf{0}$  for  $k \neq l$ .*
- Maximally Diverse Representation: *As long as the coding precision is adequately high, i.e.,  $\epsilon^4 < c \cdot \min_k \left\{ \frac{N_k}{N} \frac{d_k^2}{d^2} \right\}$ , where  $c > 0$  is a constant. Each subspace achieves its maximal dimension, i.e.  $\text{rank}(\mathbf{Z}_k^*) = d_k$ . In addition, the largest  $d_k - 1$  singular values of  $\mathbf{Z}_k^*$  are equal.*

This theorem indicates that the MCR<sup>2</sup> principle promotes embedding of data into multiple independent subspaces (as illustrated in Figure 3.22), with features distributed *isotropically* in each subspace (except for possibly one dimension). Notably, this theorem also confirms that the features learned by the MCR<sup>2</sup> principle exhibit the desired low-dimensional discriminative properties discussed in Section 3.4.1. In addition, among all such discriminative representations, it prefers the one with the highest dimensions in the ambient space. This is substantially different from the objective of information bottleneck (3.4.5).

*Example 3.13 (Classification of Images on CIFAR-10).* We here present how the MCR<sup>2</sup> objective helps learn better representations than the cross entropy (3.4.2) for image classification. Here we adopt the popular neural network architecture, the ResNet-18 [HZR+16b], to model the feature mapping  $\mathbf{z} = f(\mathbf{x}, \theta)$ . We optimize the neural network parameters  $\theta$  to maximize the coding rate reduction. We evaluate the performance with the CIFAR10 image classification dataset [KH+09].

Figure 3.23a illustrates how the two rates and their difference (for both training and test data) evolves over epochs of training: After an initial phase,  $R_\epsilon$  gradually increases while  $R_\epsilon^c$  decreases, indicating that features  $\mathbf{Z}$  are expanding as a whole while each class  $\mathbf{Z}_k$  is being compressed. Figure 3.23b shows the distribution of singular values per  $\mathbf{Z}_k$ . Figure 3.24 shows the cosine similarities between the learned features sorted by class. We compare the similarities of the learned features by using the cross-entropy (3.4.2) and the MCR<sup>2</sup> objective (3.4.12). From the plots, one can clearly see that the representations learned by using MCR<sup>2</sup> loss are much more diverse than the ones learned by using cross-entropy loss. More details of this experiment can be found in [CYY+22]. ■

However, there has been an apparent lack of justification of the network architectures used in the above experiments. It is yet unclear why the network adopted here (the ResNet-18) is suitable for representing

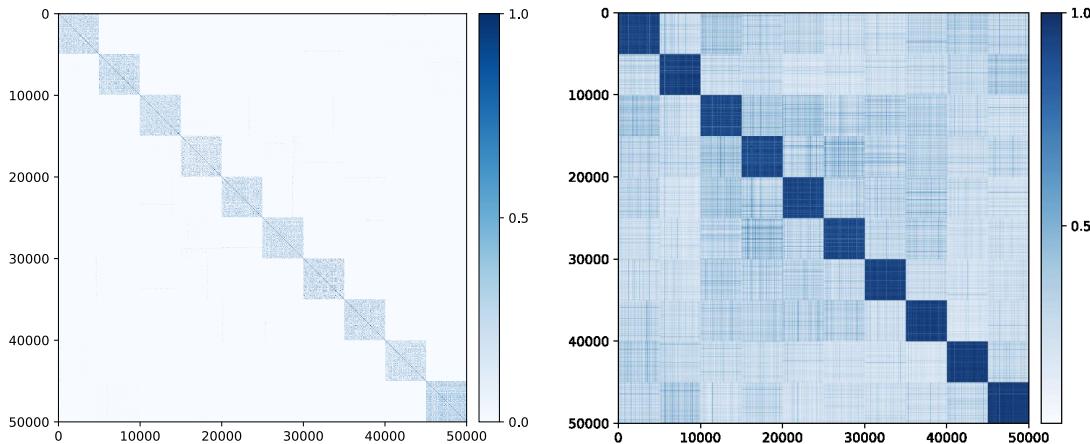
Figure 3.23: Evolution of the rates of  $MCR^2$  in the training process, the principal components of learned features.

the map  $f(\mathbf{x}, \theta)$ , let alone for interpreting the layer operators and parameters  $\theta$  learned inside. In the next chapter, we will show how to derive network architectures and components entirely as a “white box” from the desired objective (say the rate reduction).

**Regularized  $MCR^2$ .** The above theorem characterizes properties of the global optima of the rate reduction objectives. What about other optima, such as local ones? Due to the constraints of the Frobenius norm, it is a difficult task to analyze Problem (3.4.15) from an optimization-theoretic perspective. Therefore, we consider the Lagrangian formulation of (3.4.15). This can be viewed as a tight relaxation or even an equivalent problem of (3.4.15) whose optimal solutions agree under specific settings of the regularization parameter; see [WLP+24, Proposition 1]. Specifically, the formulation we study, referred to henceforth as the *regularized  $MCR^2$  problem*, is as follows:

$$\max_{\mathbf{Z}} R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z}) - \frac{\lambda}{2} \|\mathbf{Z}\|_F^2, \quad (3.4.16)$$

where  $\lambda > 0$  is the regularization parameter. Although the program (3.4.16) is highly nonconcave and involves matrix inverses in its gradient computation, we can still explicitly characterize its local and global optima as follows.

Figure 3.24: Cosine similarity between learned features by using the  $MCR^2$  objective (left) and CE loss (right).

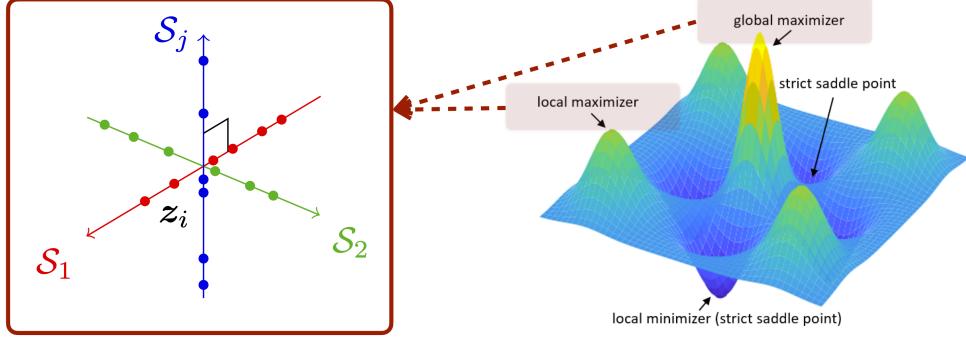


Figure 3.25: **Global optimization landscape:** According to [LSJ+16; SQW15], Theorems 3.8 and 3.9, both global and local maxima of the (regularized) rate reduction objective correspond to a solution with mutually incoherent subspaces. All other critical points are strict saddle points.

**Theorem 3.8 (Local and Global Optima).** Let  $N_k$  denote the number of training samples in the  $k$ -th class for each  $k \in \{1, \dots, K\}$ ,  $N_{\max} \doteq \max\{N_1, \dots, N_K\}$ ,  $\alpha = d/(N\epsilon^2)$ , and  $\alpha_k = d/(N_k\epsilon^2)$  for each  $k \in \{1, \dots, K\}$ . Given a coding precision  $\epsilon > 0$ , if the regularization parameter satisfies

$$\lambda \in \left(0, \frac{d(\sqrt{N/N_{\max}} - 1)}{N(\sqrt{N/N_{\max}} + 1)\epsilon^2}\right], \quad (3.4.17)$$

then the following statements hold:

(i) (**Local maximizers**)  $\mathbf{Z}^* = [\mathbf{Z}_1^*, \dots, \mathbf{Z}_K^*]$  is a local maximizer of Problem (3.4.16) if and only if the  $k$ -th block admits the following decomposition

$$\mathbf{Z}_k^* = \left( \frac{\eta_k + \sqrt{\eta_k^2 - 4\lambda^2 N/N_k}}{2\lambda\alpha_k} \right)^{1/2} \mathbf{U}_k \mathbf{V}_k^\top, \quad (3.4.18)$$

where (a)  $r_k = \text{rank}(\mathbf{Z}_k^*)$  satisfies  $r_k \in [0, \min\{N_k, d\}]$  and  $\sum_{k=1}^K r_k \leq \min\{N, d\}$ , (b)  $\mathbf{U}_k \in \mathcal{O}^{d \times r_k}$  satisfies  $\mathbf{U}_k^\top \mathbf{U}_l = \mathbf{0}$  for all  $k \neq l$ ,  $\mathbf{V}_k \in \mathcal{O}^{N_k \times r_k}$ , and (c)  $\eta_k = (\alpha_k - \alpha) - \lambda(N/N_k + 1)$  for each  $k \in \{1, \dots, K\}$ .

(ii) (**Global maximizers**)  $\mathbf{Z}^* = [\mathbf{Z}_1^*, \dots, \mathbf{Z}_K^*]$  is a global maximizer of Problem (3.4.16) if and only if (a) it satisfies the above all conditions and  $\sum_{k=1}^K r_k = \min\{m, d\}$ , and (b) for all  $k \neq l \in [K]$  satisfying  $N_k < N_l$  and  $r_l > 0$ , we have  $r_k = \min\{N_k, d\}$ .

This theorem explicitly characterizes the local and global optima of problem (3.4.16). Intuitively, this shows that the features represented by each local maximizer of Problem (3.4.16) are low-dimensional and discriminative. Although we have characterized the local and global optimal solutions in Theorem 3.8, it remains unknown whether these solutions can be efficiently computed using GD to solve the problem (3.4.16), since GD may get stuck at other critical points such as a saddle point. Fortunately, [LSJ+16; SQW15] showed that if a function is twice continuously differentiable and satisfies the *strict saddle property*, i.e., each critical point is either a local minimizer or a strict saddle point<sup>29</sup>, GD converges to its local minimizer almost surely with random initialization. We investigate the global optimization landscape of the problem (3.4.16) by characterizing all its critical points as follows.

**Theorem 3.9 (Benign Global Optimization Landscape).** Given a coding precision  $\epsilon > 0$ , if the regularization parameter satisfies (3.4.17), it holds that any critical point  $\mathbf{Z}$  of the problem (3.4.16) is either a local maximizer or a strict saddle point.

Together, the above two theorems show that the learned features associated with each local maximizer of the rate reduction objective—not just global maximizers—are structured as incoherent low-dimensional

<sup>29</sup>We say that a critical point is a strict saddle point of Problem (3.4.16) if it has a direction with strictly positive curvature [SQW15]. This includes classical saddle points with strictly positive curvature as well as local minimizers.

subspaces. Furthermore, the (regularized) rate reduction objective (3.4.12) has a very benign landscape with only local maxima and strict saddles as critical points, as illustrated in Figure 3.25. According to [LSJ+16; SQW15], Theorems 3.8 and 3.9 imply that low-dimensional and discriminative representations (LDRs) can be efficiently found by applying (stochastic) gradient descent to the rate reduction objective (3.4.12) from random initialization. These results also indirectly explain why in Example 3.13, if the chosen network is expressive enough and trained well, the resulting representation typically gives an incoherent linear representation which likely corresponds to the globally optimal solution. Interested readers are referred to [WLP+24] for proofs.

## 3.5 Summary and Notes

The use of denoising and diffusion for sampling has a rich history. The first work which is clearly about a diffusion model is probably [SWM+15], but before this there are many works about denoising as a computational and statistical problem. The most relevant of these is probably [Hyr05], which explicitly uses the score function to denoise (as well as perform independent component analysis). The most popular follow-ups are basically co-occurring: [HJA20; SE19]. Since then, thousands of papers have built on diffusion models; we will revisit this topic in Chapter 5.

Many of these works use a different stochastic process than the simple linear combination (3.2.69). In fact, all works listed above emphasize the need to add *independent* Gaussian noise at the beginning of each step of the forward process. Theoretically-minded work actually uses Brownian motion or stochastic differential equations to formulate the forward process [SSK+21]. However, since linear combinations of Gaussians still result in Gaussians, the *marginal distributions* of such processes still take the form of (3.2.69). Most of our discussion requires only that the marginal distributions are what they are, and hence our overly simplistic model is actually quite enough for almost everything. In fact, the only time where marginal distributions are not enough is when we derive an expression for  $\mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t]$  in terms of  $\mathbb{E}[\mathbf{x} \mid \mathbf{x}_t]$ . Different (noising) processes give different such expressions, which can be used for sampling (and of course there are other ways to derive efficient samplers, such as the ever-popular DDPM sampler). The process in (3.2.69) is a bona fide stochastic process, however, whose “natural” denoising iteration takes the form of the popular DDIM algorithm [SME20]. (Even this equivalence is not trivial; we cite [DGG+25] as a justification.)

On top of the theoretical work [LY24] covered in Section 1.3.1, and the lineage of work that it builds on, which studies the *sampling* efficiency of diffusion models when the data has low-dimensional structure, there is a large body of work which studies the *training* efficiency of diffusion models when the data has low-dimensional structure. Specifically, Chen et al. [CHZ+23] and [WZZ+24] characterized the approximation and estimation error of denoisers when the data belongs to a mixture of low-rank Gaussians, showing that the number of training samples required to accurately learn the distribution scales with the intrinsic dimension of the data rather than the ambient distribution. There is considerable *methodological* work which attempts to utilize the low-dimensional structure of the data in order to do various things with diffusion models. We highlight three here: image editing [CZG+24], watermarking [LZQ24], and unlearning [CZL+25], though as always this is an inexhaustive list.

## 3.6 Exercises and Extensions

*Exercise 3.1.* Please show that (3.2.4) is the optimal solution of Problem (3.2.3).

*Exercise 3.2.* Consider random vectors  $\mathbf{x} \in \mathbb{R}^D$  and  $\mathbf{y} \in \mathbb{R}^d$ , such that the pair  $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{D+d}$  is jointly Gaussian. This means that

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} \boldsymbol{\mu}_{\mathbf{x}} \\ \boldsymbol{\mu}_{\mathbf{y}} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{x}} & \boldsymbol{\Sigma}_{\mathbf{x}\mathbf{y}} \\ \boldsymbol{\Sigma}_{\mathbf{x}\mathbf{y}}^\top & \boldsymbol{\Sigma}_{\mathbf{y}} \end{bmatrix} \right),$$

where the mean and covariance parameters are given by

$$\boldsymbol{\mu}_{\mathbf{x}} = \mathbb{E}[\mathbf{x}], \quad \boldsymbol{\mu}_{\mathbf{y}} = \mathbb{E}[\mathbf{y}], \quad \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{x}} & \boldsymbol{\Sigma}_{\mathbf{x}\mathbf{y}} \\ \boldsymbol{\Sigma}_{\mathbf{x}\mathbf{y}}^\top & \boldsymbol{\Sigma}_{\mathbf{y}} \end{bmatrix} = \mathbb{E} \left[ \begin{bmatrix} \mathbf{x} - \mathbb{E}[\mathbf{x}] \\ \mathbf{y} - \mathbb{E}[\mathbf{y}] \end{bmatrix} \begin{bmatrix} \mathbf{x} - \mathbb{E}[\mathbf{x}] \\ \mathbf{y} - \mathbb{E}[\mathbf{y}] \end{bmatrix}^\top \right]$$

Assume that  $\Sigma_y$  is positive definite (hence invertible); then positive semidefiniteness of the covariance matrix is equivalent to the Schur complement condition  $\Sigma_x - \Sigma_{xy}\Sigma_y^{-1}\Sigma_{xy}^\top \succeq 0$ .

In this exercise, we will prove that the conditional distribution  $p_{x|y}$  is Gaussian: namely,

$$p_{x|y} \sim \mathcal{N}(\mu_x + \Sigma_{xy}\Sigma_y^{-1}(y - \mu_y), \Sigma_x - \Sigma_{xy}\Sigma_y^{-1}\Sigma_{xy}^\top). \quad (3.6.1)$$

A direct path to prove this result manipulates the defining ratio of densities  $p_{x,y}/p_y$ . We sketch an algebraically-concise argument of this form below.

1. Verify the following matrix identity for the covariance:

$$\begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{xy}^\top & \Sigma_y \end{bmatrix} = \begin{bmatrix} I_D & \Sigma_{xy}\Sigma_y^{-1} \\ 0 & I_d \end{bmatrix} \begin{bmatrix} \Sigma_x - \Sigma_{xy}\Sigma_y^{-1}\Sigma_{xy}^\top & 0 \\ 0 & \Sigma_y \end{bmatrix} \begin{bmatrix} I_D & 0 \\ \Sigma_y^{-1}\Sigma_{xy}^\top & I_d \end{bmatrix}. \quad (3.6.2)$$

One arrives at this identity by performing two rounds of (block) Gaussian elimination on the covariance matrix.

2. Based on the previous identity, show that

$$\begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{xy}^\top & \Sigma_y \end{bmatrix}^{-1} = \begin{bmatrix} I_D & 0 \\ -\Sigma_y^{-1}\Sigma_{xy}^\top & I_d \end{bmatrix} \begin{bmatrix} (\Sigma_x - \Sigma_{xy}\Sigma_y^{-1}\Sigma_{xy}^\top)^{-1} & 0 \\ 0 & \Sigma_y^{-1} \end{bmatrix} \begin{bmatrix} I_D & -\Sigma_{xy}\Sigma_y^{-1} \\ 0 & I_d \end{bmatrix} \quad (3.6.3)$$

whenever the relevant inverses are defined.<sup>30</sup> Conclude that

$$\begin{bmatrix} x - \mu_x \\ y - \mu_y \end{bmatrix}^\top \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{xy}^\top & \Sigma_y \end{bmatrix}^{-1} \begin{bmatrix} x - \mu_x \\ y - \mu_y \end{bmatrix} \quad (3.6.4)$$

$$= \begin{bmatrix} x - (\mu_x + \Sigma_{xy}\Sigma_y^{-1}(y - \mu_y)) \\ y - \mu_y \end{bmatrix}^\top \begin{bmatrix} (\Sigma_x - \Sigma_{xy}\Sigma_y^{-1}\Sigma_{xy}^\top)^{-1} & 0 \\ 0 & \Sigma_y^{-1} \end{bmatrix} \begin{bmatrix} x - (\mu_x + \Sigma_{xy}\Sigma_y^{-1}(y - \mu_y)) \\ y - \mu_y \end{bmatrix}. \quad (3.6.5)$$

(Hint: To economize algebraic manipulations, note that the first and last matrices on the RHS of Equation (3.6.2) are transposes of one another.)

3. By dividing  $p_{x,y}/p_y$ , prove Equation (3.6.1). (Hint: Using the previous identities, only minimal algebra should be necessary. For the normalizing constant, use Equation (3.6.3) to factor the determinant similarly.)

*Exercise 3.3.* Show the Sherman-Morrison-Woodbury identity, i.e., for matrices  $\mathbf{A}$ ,  $\mathbf{C}$ ,  $\mathbf{U}$ ,  $\mathbf{V}$  such that  $\mathbf{A}$ ,  $\mathbf{C}$ , and  $\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V}$  are invertible,

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1} \quad (3.6.6)$$

*Exercise 3.4.* Rederive the following, assuming  $x_t$  follows the generalized noise model (3.2.69).

- Tweedie's formula: (3.2.70).
- The DDIM iteration: (3.2.71).
- The Bayes optimal denoiser for a Gaussian mixture model: (3.2.72).

*Exercise 3.5.* 1. Implement the formulae derived in Exercise 3.4, building a sampler for Gaussian mixtures.

2. Reproduce Figure 3.4 and Figure 3.7.

---

<sup>30</sup>In cases where the Schur complement term is not invertible, the same result holds with its inverse replaced by the Moore-Penrose pseudoinverse. In particular, the conditional distribution (3.6.1) becomes a degenerate Gaussian distribution.

3. We now introduce a separate process called *Flow Matching (FM)*, as follows:

$$\alpha_t = 1 - t, \quad \sigma_t = t. \quad (3.6.7)$$

Implement this process using the same framework, and test it for sampling in high dimensions. Which process seems to give better or more stable results?

*Exercise 3.6.* Please show the following properties of the  $\log \det(\cdot)$  function.

1. Show that

$$f(\mathbf{X}) = \log \det(\mathbf{X})$$

is a concave function. (**Hint:** The function  $f(\mathbf{x})$  is convex if and only if the function  $f(\mathbf{x} + t\mathbf{h})$  for all  $\mathbf{x}$  and  $\mathbf{h}$ .)

2. Show that:

$$\log \det(\mathbf{I} + \mathbf{X}^\top \mathbf{X}) = \log \det(\mathbf{I} + \mathbf{X} \mathbf{X}^\top)$$

3. Let  $\mathbf{A} \in \mathbb{R}^{n \times n}$  be a positive definite matrix. Please show that:

$$\log \det(\mathbf{A}) = \sum_{i=1}^n \log(\lambda_i), \quad (3.6.8)$$

where  $\lambda_1, \lambda_2, \dots, \lambda_n$  are the eigenvalues of  $\mathbf{A}$ .

## Chapter 4

# Deep Representations from Unrolled Optimization

“What I cannot create, I do not understand.”

— Richard Feynman

In previous chapters, we have shown how to identify low-dimensional structures in high-dimensional spaces, *mainly focusing on linear structures*. For example, we introduced principal component analysis (PCA) to learn the linear denoiser  $\hat{\mathbf{U}}^\top$  when the observed data  $\mathbf{x}$  follow the statistical model  $\mathbf{x} = \mathbf{U}\mathbf{z} + \boldsymbol{\varepsilon}$ . In this setting, the learned representations are linearly transformed input data  $\hat{\mathbf{U}}^\top \mathbf{x}$ . Under the linear model assumption, one can learn the low-dimensional linear structure with efficient optimization algorithms and strong theoretical guarantees. Moreover, the linear model assumption covers a wide range of applications and problems, including face recognition, magnetic resonance image recovery, and structure texture recovery [WM22].

On the other hand, the linear model can be limited when dealing with real-world applications, especially when the input data  $\mathbf{x}$  is complex, such as speech and natural languages, images and videos, and robotic motions. The low-dimensional distributions of such data are typically nonlinear. How to deal with nonlinearity has a long history across different disciplines such as control theory, signal processing, and pattern recognition. There have been considerable efforts that try to extend methods and solutions for linear models to handle nonlinearity, including early effort to extend PCA to nonlinear PCA (as we will study in more detail in Chapter 5). In most cases, the methods are designed based on certain assumptions about the data distributions and tailored to specific problems.

More recently, deep neural networks have achieved remarkable success across a wide range of data and applications. A neural network

$$f(\cdot, \boldsymbol{\theta}): \mathbf{x} \xrightarrow{f^0} \mathbf{z}^0 \rightarrow \cdots \rightarrow \mathbf{z}^\ell \xrightarrow{f^\ell} \mathbf{z}^{\ell+1} \rightarrow \cdots \rightarrow \mathbf{z}^L = \mathbf{z}. \quad (4.0.1)$$

can learn effective features/representations for downstream applications. For example, a trained deep neural network  $f(\cdot, \boldsymbol{\theta})$  can be applied to map images to feature vectors, that is,  $\mathbf{z}_i = f(\mathbf{x}_i, \boldsymbol{\theta})$ , while a linear classifier can be learned on top of such representations  $\{\mathbf{z}_i\}$ . One notable breakthrough is AlexNet [KSH12], a deep convolutional neural network trained with more than a million natural images, outperforming all previous approaches that were based on hand-crafted features. One of the key differences between AlexNet and previous approaches is that the former *learns parameters of the nonlinear transformation from massive amounts of data* trained with back-propagation (BP) [RHW86b], as detailed in Section A.2.3 of Chapter A.

Subsequent popular practice models the mapping  $f$  with other empirically designed artificial deep neural networks and learns the parameters  $\boldsymbol{\theta}$  from random initialization via BP. Starting with the AlexNet [KSH12], the architectures of modern deep networks continue to be empirically revised and improved. Network architectures such as VGG [SZ14], ResNet [HZR+16b], DenseNet [HLV+17], CNN, RNN or LSTM [HS97], Transformer [VSP+17], and a mixture of experts (MoE) [FZS22; SMM+17], etc. have continued to push the

performance envelope. As part of the effort to improve the performance of deep networks, almost every component of the networks has been empirically scrutinized, and various revisions and improvements have been proposed. They are not limited to nonlinear activation functions [KUM+17; MHN13; NIG+18; XWC+15], skip connections [HZR+16b; RFB15], normalizations [BKH16; IS15; MKK+18; UVL16; WH18], up/down sampling or pooling [SMB10], convolutions [KSH12; LBB+98b], etc. However, almost all such modifications have been developed through years of empirical trial and error or ablation studies. Some recent practices even take to the extreme by searching for effective network structures and training strategies through extensive random search techniques, such as Neural Architecture Search [BGN+17; ZL17], AutoML [HKV19], and Learning to Learn [ADG+16].

Despite the wide application of deep neural networks, it is not clear what the underlying design principles of such a constructed network are. In particular, it is not clear what mathematical function each layer of the network performs. In this chapter, based on the results from previous chapters, we develop a principled framework that will provide a fully rigorous mathematical interpretation of the role of a deep network, including its individual layers and the network as a whole.

To understand deep networks and how they should be better designed, we must start with the objective of representation learning. In previous chapters, we have argued that the objective is to identify the intrinsically low-dimensional data distribution and then transform it to a compact and structured (say piecewise linear) representation. As we have seen in the previous chapter, the general approach to identifying a low-dimensional data distribution is through a compression process that progressively minimizes the entropy or coding rate of the distribution. However, up to this point, we have been using empirically designed deep networks to model or approximate the operations that aim to optimize these objectives, such as the score function for denoising (in Section 1.3.1) or the transformation that maximizes the rate reduction (in Section 3.4.3).

As we have argued in the previous chapter, Section 3.4 in particular, one can measure the goodness of the resulting representation by the information of the representation gained from a “lazy” representation which models all data as one big Gaussian.<sup>1</sup> In particular, if *we use a mixture of Gaussians (subspaces)<sup>2</sup> as prototypical distributions to approximate the non-linear distribution of interest*, then we can efficiently measure the coding rate of such a representation using the (sum of) rate distortion functions of the associated Gaussians. Then the amount of *information gained* or (relative) entropy reduced with such a modeling can be measured by the difference between the coding rate for the lazy representation and that for the more refined representation. Then, the objective of representation learning is to maximize this information gain, also known as the rate reduction objective.

As we will see in this chapter, once the objective of representation learning is clear, the role of a deep neural network is precisely to help optimize the objective iteratively. Each layer of a deep neural network can be naturally derived as an iterative optimization step to incrementally maximize the information gain, including the popular architectures of ResNet, CNN, and Transformer, and other more advanced variants. In particular, this chapter aims to answer the following questions about deep networks:

- Section 4.1 — given a measure of goodness for a learned representation, how to construct the nonlinear mapping from the data to the optimal representation via unrolled optimization for the objective?
- Section 4.2 — how would the above unrolling approach provide a principled interpretation of the popular transformer architectures; if so, what are the associated objective and optimization mechanisms?
- Section 4.3 — how would this framework guide us to design more efficient or more parsimonious deep architectures?

## 4.1 White-Box Deep Networks via Unrolled Optimization

Now, if we agree that maximizing the rate reduction or information gain leads to the desired representation as discussed in Section 3.4, the remaining question is how to construct and learn a (nonlinear) mapping from the data  $\mathbf{X}$  to the optimal representation  $\mathbf{Z}^*$ . This involves designing a network architecture and learning

---

<sup>1</sup>that we have seen in the previous chapter as one particular choice of interpretation of the sampled dataset.

<sup>2</sup>which we have studied thoroughly in the previous chapter.

algorithm that can effectively capture the underlying structures in the data and faithfully realize the optimal representation.

### 4.1.1 Deep Networks from Unrolled Gradient Descent

In the previous chapter, we presented the rate reduction objective (3.4.12) as a principled objective for learning linear discriminative representations of the data. We have, however, not specified the architecture of the feature mapping  $\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta})$  for extracting such representations from input data  $\mathbf{x}$ . A straightforward choice is to use a conventional deep network, such as ResNet, for implementing  $f(\mathbf{x}, \boldsymbol{\theta})$ . As we have seen in Example 3.13, such a choice often leads to decent performance empirically. Nonetheless, there remain several unanswered problems with adopting an arbitrary deep network. Although the learned feature representation is now more interpretable, the network itself is still *not*. It is unclear why any chosen “black-box” network is able to optimize the desired MCR<sup>2</sup> objective at all. The good empirical results (say with a ResNet) do not necessarily justify the particular choice in architectures and operators of the network: Why is a deep layered model even necessary; what do additional layers try to improve or simplify; how wide and deep is adequate; or is there any rigorous justification for the convolutions (in a popular multi-channel form) and nonlinear operators (e.g. ReLU or softmax) used?

In this chapter, we show that using gradient ascent to maximize the rate reduction  $\Delta R_\epsilon(\mathbf{Z} \mid \boldsymbol{\Pi})$  as defined in (3.4.12) naturally leads to a “white-box” deep network that realizes the desired mapping. All network layers, linear/nonlinear operators, and parameters are *explicitly constructed in a purely forward propagation fashion*. Moreover, such network architectures resemble existing empirically-designed deep networks, providing principled justifications for their design.

**Gradient Ascent for Coding Rate Reduction.** From the previous chapter, we see that to seek a linear discriminative representation (LDR), mathematically, we are essentially seeking a continuous mapping  $f(\cdot) : \mathbf{x} \mapsto \mathbf{z}$  from the data  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$  (or initial features extracted from the data<sup>3</sup>) to an optimal representation  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N] \in \mathbb{R}^{d \times N}$  that maximizes the following coding rate reduction objective:

$$\Delta R_\epsilon(\mathbf{Z} \mid \boldsymbol{\Pi}) \doteq \underbrace{\frac{1}{2} \log \det (\mathbf{I} + \alpha \mathbf{Z} \mathbf{Z}^\top)}_{R_\epsilon(\mathbf{Z})} - \underbrace{\sum_{k=1}^K \frac{\gamma_k}{2} \log \det (\mathbf{I} + \alpha_k \mathbf{Z} \boldsymbol{\Pi}_k \mathbf{Z}^\top)}_{R_\epsilon^c(\mathbf{Z} \mid \boldsymbol{\Pi})}, \quad (4.1.1)$$

where  $\epsilon > 0$  is a prescribed quantization error and for simplicity we denote<sup>4</sup>

$$\alpha \doteq \frac{d}{N\epsilon^2}, \quad \alpha_k \doteq \frac{d}{\text{tr}(\boldsymbol{\Pi}_k)\epsilon^2}, \quad \gamma_k \doteq \frac{\text{tr}(\boldsymbol{\Pi}_k)}{N}, \quad \text{for } k = 1, \dots, K.$$

The question really boils down to whether there is a *constructive* way of finding such a continuous mapping  $f(\cdot, \boldsymbol{\theta})$  from  $\mathbf{x}$  to  $\mathbf{z}$ ? To this end, let us consider incrementally maximizing the objective  $\Delta R_\epsilon$  as a function of  $\mathbf{Z} \subseteq \mathbb{S}^{d-1}$ . Although there might be many optimization schemes to choose from, for simplicity we first consider the arguably simplest projected gradient ascent (PGA) scheme.<sup>5</sup>

$$\mathbf{Z}^{\ell+1} \propto \mathbf{Z}^\ell + \eta \cdot \frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}}(\mathbf{Z}^\ell) \quad \text{s.t.} \quad \mathbf{Z}^{\ell+1} \subseteq \mathbb{S}^{d-1}, \quad \ell = 1, 2, \dots, \quad (4.1.2)$$

for some step size  $\eta > 0$  and the iterate starts with the given data  $\mathbf{Z}^0 = \mathbf{X}$ .<sup>6</sup> This scheme can be interpreted as how one should incrementally adjust locations of the current features  $\mathbf{Z}^\ell$ , initialized as the input data  $\mathbf{X}$ , in order for the resulting  $\mathbf{Z}^{\ell+1}$  to improve the rate reduction  $\Delta R_\epsilon$ , as illustrated in Figure 4.1.

<sup>3</sup>As we will see the necessity of such a feature extraction in the next section.

<sup>4</sup>Notice our use of slightly simplified notation compared to Chapter 3.

<sup>5</sup>Notice that we use subscript  $j$  on  $\mathbf{Z}_j$  to indicate features in the  $j$ -th class and superscript  $\ell$  on  $\mathbf{Z}^\ell$  to indicate all features at  $\ell$ -th iteration or layer.

<sup>6</sup>Again, for simplicity, we here first assume the initial features  $\mathbf{Z}^1$  are the data themselves. Note that here  $\ell$  denotes the number of iterations. Hence, the data and the features have the same dimension  $d$ . This needs not to be the case though. As we will see in the next section, the initial features can be some (lifted) features of the data to begin with and could in principle have a different (much higher) dimension. All subsequent iterates have the same dimension.

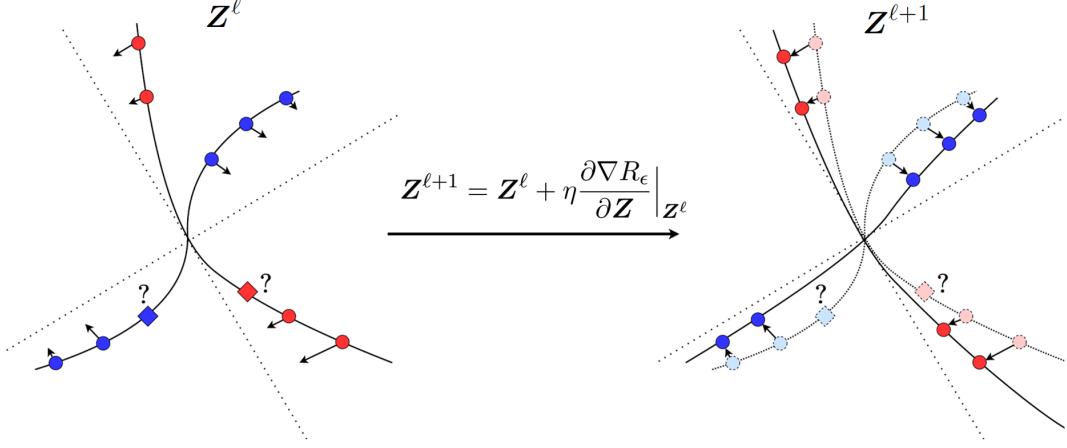


Figure 4.1: Incremental deformation via gradient flow to both flatten data of each class into a subspace and push different classes apart.

Simple calculation shows that the gradient  $\partial \Delta R_\epsilon / \partial \mathbf{Z}$  entails evaluating the following derivatives of the two terms in  $\Delta R_\epsilon$ :

$$\frac{1}{2} \frac{\partial \log \det(\mathbf{I} + \alpha \mathbf{Z} \mathbf{Z}^\top)}{\partial \mathbf{Z}}(\mathbf{Z}^\ell) = \underbrace{\alpha (\mathbf{I} + \alpha \mathbf{Z}^\ell (\mathbf{Z}^\ell)^\top)^{-1} \mathbf{Z}^\ell}_{\mathbf{E}^\ell \in \mathbb{R}^{d \times d}}, \quad (4.1.3)$$

$$\frac{1}{2} \frac{\partial (\gamma_k \log \det(\mathbf{I} + \alpha_k \mathbf{Z} \boldsymbol{\Pi}_k \mathbf{Z}^\top))}{\partial \mathbf{Z}}(\mathbf{Z}^\ell) = \gamma_k \underbrace{\alpha_k (\mathbf{I} + \alpha_k \mathbf{Z}^\ell \boldsymbol{\Pi}_k (\mathbf{Z}^\ell)^\top)^{-1} \mathbf{Z}^\ell \boldsymbol{\Pi}_k}_{\mathbf{C}_k^\ell \in \mathbb{R}^{d \times d}}. \quad (4.1.4)$$

Notice that in the above, the matrix  $\mathbf{E}^\ell$  only depends on  $\mathbf{Z}^\ell$  and it aims to *expand* all the features to increase the overall coding rate; the matrix  $\mathbf{C}_k^\ell$  depends on features from the  $k$ -class and aims to *compress* them to reduce the coding rate of each class. Then the complete gradient  $\frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}}(\mathbf{Z}^\ell) \in \mathbb{R}^{d \times N}$  is of the form:

$$\frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}}(\mathbf{Z}^\ell) = \underbrace{\mathbf{E}^\ell}_{\text{Expansion}} \mathbf{Z}^\ell - \sum_{k=1}^K \gamma_k \underbrace{\mathbf{C}_k^\ell}_{\text{Compression}} \mathbf{Z}^\ell \boldsymbol{\Pi}_k. \quad (4.1.5)$$

*Remark 4.1* (Interpretation of  $\mathbf{E}^\ell$  and  $\mathbf{C}_j^\ell$  as linear operators). For any  $\mathbf{z}^\ell \in \mathbb{R}^d$ ,

$$\mathbf{E}^\ell \mathbf{z}^\ell = \alpha(\mathbf{z}^\ell - \mathbf{Z}^\ell \mathbf{q}_*^\ell), \quad \text{where} \quad \mathbf{q}_*^\ell \doteq \arg \min_{\mathbf{q}^\ell} \{\alpha \|\mathbf{z}^\ell - \mathbf{Z}^\ell \mathbf{q}^\ell\|_2^2 + \|\mathbf{q}^\ell\|_2^2\}. \quad (4.1.6)$$

Notice that  $\mathbf{q}_*^\ell$  is exactly the solution to the ridge regression by all the data points  $\mathbf{Z}^\ell$  concerned. Therefore,  $\mathbf{E}^\ell$  (similarly for  $\mathbf{C}_k^\ell$ ) is approximately (i.e., when  $N$  is large enough) the projection onto the orthogonal complement of the subspace spanned by columns of  $\mathbf{Z}^\ell$ . Another way to interpret the matrix  $\mathbf{E}^\ell$  is through eigenvalue decomposition of the covariance matrix  $\mathbf{Z}^\ell (\mathbf{Z}^\ell)^\top$ . Assuming that  $\mathbf{Z}^\ell (\mathbf{Z}^\ell)^\top \doteq \mathbf{U}^\ell \boldsymbol{\Lambda}^\ell (\mathbf{U}^\ell)^\top$  where  $\boldsymbol{\Lambda}^\ell \doteq \text{diag}(\lambda_1^\ell, \dots, \lambda_d^\ell)$ , we have

$$\mathbf{E}^\ell = \alpha \mathbf{U}^\ell \text{ diag} \left( \frac{1}{1 + \alpha \lambda_1^\ell}, \dots, \frac{1}{1 + \alpha \lambda_d^\ell} \right) (\mathbf{U}^\ell)^\top. \quad (4.1.7)$$

Therefore, the matrix  $\mathbf{E}^\ell$  operates on a vector  $\mathbf{z}^\ell$  by stretching in a way that directions of large variance are shrunk while directions of vanishing variance are kept. These are exactly the directions (4.1.3) in which we move the features so that the overall volume expands and the coding rate will increase, hence the positive sign. To the opposite effect, the directions associated with (4.1.4) are “residuals” of features of each class deviate from the subspace to which they are supposed to belong. These are exactly the directions in which the features need to be compressed back onto their respective subspace, hence the negative sign (see Figure 4.2).

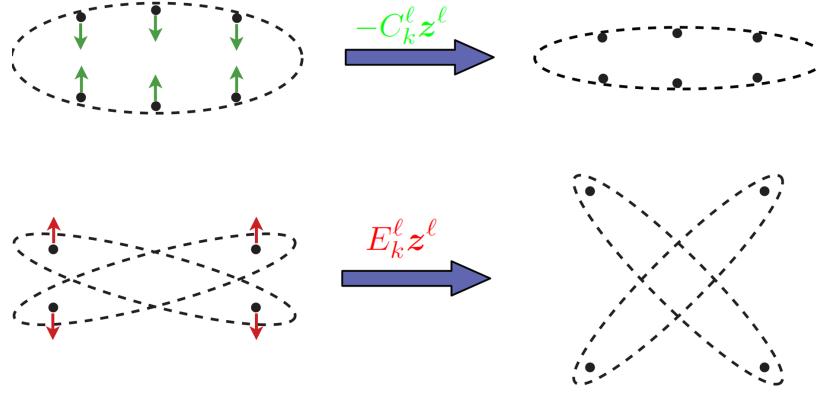


Figure 4.2: Interpretation of  $C_k^l$  and  $E^l$ :  $C_k^l$  compresses each class by contracting the features to a low-dimensional subspace;  $E^l$  expands all features by contrasting and repelling features across different classes.

Essentially, the linear operations  $E^l$  and  $C_k^l$  in gradient ascend for rate reduction are determined by training data conducting “auto-regressions”. The recent renewed understanding about ridge regression in an over-parameterized setting [WX20; YYY+20] indicates that using seemingly redundantly sampled data (from each subspace) as regressors does not lead to overfitting.

**Gradient-Guided Feature Map Increment.** Notice that in the above, the gradient ascent considers all the features  $\mathbf{Z}^\ell = [\mathbf{z}_1^\ell, \dots, \mathbf{z}_N^\ell]$  as free variables. The increment  $\mathbf{Z}^{\ell+1} - \mathbf{Z}^\ell = \eta \frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}}(\mathbf{Z}^\ell)$  does not yet give a transformation on the entire feature domain  $\mathbf{z}^\ell \in \mathbb{R}^d$ . According to equation (4.1.5), the gradient cannot be evaluated at a point whose membership is not known, as illustrated in Figure 4.1. Hence, in order to find the optimal  $f(\mathbf{x}, \boldsymbol{\theta})$  explicitly, we may consider constructing a small increment transform  $g(\cdot, \boldsymbol{\theta}^\ell)$  on the  $\ell$ -th layer feature  $\mathbf{z}^\ell$  to emulate the above (projected) gradient scheme:

$$\mathbf{z}^{\ell+1} \propto \mathbf{z}^\ell + \eta \cdot g(\mathbf{z}^\ell, \boldsymbol{\theta}^\ell) \quad \text{subject to } \mathbf{z}^{\ell+1} \in \mathbb{S}^{d-1} \quad (4.1.8)$$

such that  $[g(\mathbf{z}_1^\ell, \boldsymbol{\theta}^\ell), \dots, g(\mathbf{z}_N^\ell, \boldsymbol{\theta}^\ell)] \approx \frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}}(\mathbf{Z}^\ell)$ . That is, we need to approximate the gradient flow  $\frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}}$  that locally deforms all (training) features  $\{\mathbf{z}_i^\ell\}_{i=1}^N$  with a continuous mapping  $g(\mathbf{z}, \boldsymbol{\theta})$  defined on the entire feature space  $\mathbf{z}^\ell \in \mathbb{R}^d$ . Notice that one may interpret the increment (4.1.8) as a discretized version of a continuous differential equation:

$$\dot{\mathbf{z}} = g(\mathbf{z}, \boldsymbol{\theta}). \quad (4.1.9)$$

Hence the (deep) network so constructed can be interpreted as certain neural ODE [CRB+18]. Nevertheless, unlike neural ODE where the flow  $g$  is chosen to be some generic structures, here our  $g(\mathbf{z}, \boldsymbol{\theta})$  is to emulate the gradient flow of the rate reduction on the feature set (as shown in Figure 4.1):

$$\dot{\mathbf{Z}} = \frac{\partial \Delta R_\epsilon}{\partial \mathbf{Z}},$$

and its structure is entirely derived and fully determined from this objective, without any other priors or heuristics.

By inspecting the structure of the gradient (4.1.5), it suggests that a natural candidate for the increment transform  $g(\mathbf{z}^\ell, \boldsymbol{\theta}^\ell)$  is of the form:

$$g(\mathbf{z}^\ell, \boldsymbol{\theta}^\ell) \doteq \mathbf{E}^\ell \mathbf{z}^\ell - \sum_{k=1}^K \gamma_k \pi_k(\mathbf{z}^\ell) \mathbf{C}_k^\ell \mathbf{z}^\ell \in \mathbb{R}^d, \quad (4.1.10)$$

where  $\pi_k(\mathbf{z}^\ell) \in [0, 1]$  indicates the probability of  $\mathbf{z}^\ell$  belonging to the  $k$ -th class. The increment map parameters  $\boldsymbol{\theta}^\ell$  depend on: First, a set of linear maps represented by  $\mathbf{E}^\ell$  and  $\{\mathbf{C}_k^\ell\}_{k=1}^K$  that depend only on statistics of features of the training  $\mathbf{Z}^\ell$ ; Second, the membership  $\{\pi_k(\mathbf{z}^\ell)\}_{k=1}^K$  of any feature  $\mathbf{z}^\ell$ . Notice that

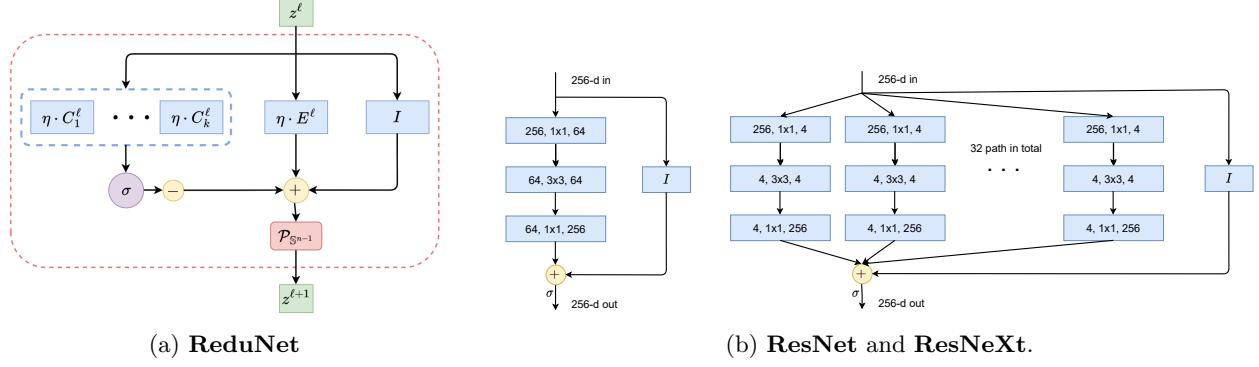


Figure 4.3: Network Architectures of the ReduNet and comparison with others. (a): Layer structure of the **ReduNet** derived from one iteration of gradient ascent for optimizing rate reduction. (b) (left): A layer of ResNet [HZR+16b]; and (b) (right): A layer of ResNeXt [XGD+17]. As we will see in Section 4.1.2, the linear operators  $\mathbf{E}^\ell$  and  $\mathbf{C}_k^\ell$  of the ReduNet naturally become (multi-channel) convolutions when shift-invariance is imposed.

on the training samples  $\mathbf{Z}^\ell$ , for which the memberships  $\Pi_k$  are known, the so defined  $g(\mathbf{z}^\ell, \theta)$  gives exactly the values for the gradient  $\frac{\partial \Delta R_e}{\partial \mathbf{Z}}(\mathbf{Z}^\ell)$ .

Since we only have the membership for the training samples, the function  $g(\cdot)$  defined in (4.1.10) can only be evaluated on the training. To extrapolate  $g(\cdot)$  to the entire feature space, we need to estimate  $\pi_k(\mathbf{z}^\ell)$  in its second term. In conventional deep learning, this map is typically modeled as a deep network and learned from the training data, say via *back propagation*. Nevertheless, our goal here is not to learn a precise classifier  $\pi_k(\mathbf{z}^\ell)$  already. Instead, we only need a good enough estimate of the class information in order for  $g(\cdot)$  to approximate the gradient  $\frac{\partial \Delta R_e}{\partial \mathbf{Z}}$  well.

From the geometric interpretation of the linear maps  $\mathbf{E}^\ell$  and  $\mathbf{C}_k^\ell$  given by Remark 4.1, the term  $\mathbf{p}_k^\ell \doteq \mathbf{C}_k^\ell \mathbf{z}^\ell$  can be viewed as (approximately) the projection of  $\mathbf{z}^\ell$  onto the orthogonal complement of each class  $j$ . Therefore,  $\|\mathbf{p}_j^\ell\|_2$  is small if  $\mathbf{z}^\ell$  is in class  $j$  and large otherwise. This motivates us to estimate its membership based on the following softmax function:

$$\widehat{\pi}(\mathbf{z}^\ell) \doteq \text{softmax}\left(-\lambda \begin{bmatrix} \|\mathbf{C}_1^\ell \mathbf{z}^\ell\|_2 \\ \vdots \\ \|\mathbf{C}_K^\ell \mathbf{z}^\ell\|_2 \end{bmatrix}\right) = \frac{1}{\sum_{k=1}^K \exp(-\lambda \|\mathbf{C}_k^\ell \mathbf{z}^\ell\|_2)} \begin{bmatrix} \exp(-\lambda \|\mathbf{C}_1^\ell \mathbf{z}^\ell\|_2) \\ \vdots \\ \exp(-\lambda \|\mathbf{C}_K^\ell \mathbf{z}^\ell\|_2) \end{bmatrix} \in [0, 1]^K. \quad (4.1.11)$$

Hence, the second term of (4.1.10) can be approximated by this estimated membership:

$$\sum_{k=1}^K \gamma_k \pi_k(\mathbf{z}^\ell) \mathbf{C}_k^\ell \mathbf{z}^\ell \approx \sum_{k=1}^K \gamma_k \widehat{\pi}_k(\mathbf{z}^\ell) \mathbf{C}_k^\ell \mathbf{z}^\ell \doteq \boldsymbol{\sigma}([\mathbf{C}_1^\ell \mathbf{z}^\ell, \dots, \mathbf{C}_K^\ell \mathbf{z}^\ell]), \quad (4.1.12)$$

which is denoted as a nonlinear operator  $\boldsymbol{\sigma}(\cdot)$  on outputs of the feature  $\mathbf{z}^\ell$  through  $K$  groups of filters:  $[\mathbf{C}_1^\ell, \dots, \mathbf{C}_K^\ell]$ . Notice that the nonlinearity arises due to a “soft” assignment of class membership based on the feature responses from those filters.

Overall, combining (4.1.8), (4.1.10), and (4.1.12), the increment feature transform from  $\mathbf{z}^\ell$  to  $\mathbf{z}^{\ell+1}$  now becomes

$$\begin{aligned} \mathbf{z}^{\ell+1} &\propto \mathbf{z}^\ell + \eta \cdot \mathbf{E}^\ell \mathbf{z}^\ell - \eta \cdot \boldsymbol{\sigma}([\mathbf{C}_1^\ell \mathbf{z}^\ell, \dots, \mathbf{C}_K^\ell \mathbf{z}^\ell]) \\ &= \mathbf{z}^\ell + \eta \cdot g(\mathbf{z}^\ell, \theta^\ell) \quad \text{s.t. } \mathbf{z}^{\ell+1} \in \mathbb{S}^{d-1}, \end{aligned} \quad (4.1.13)$$

with the nonlinear function  $\boldsymbol{\sigma}(\cdot)$  defined above and  $\theta^\ell$  collecting all the layer-wise parameters. That is  $\theta^\ell = \{\mathbf{E}^\ell, \mathbf{C}_1^\ell, \dots, \mathbf{C}_K^\ell, \gamma_k, \lambda\}$ . Note features at each layer are always “normalized” by projecting onto the unit sphere  $\mathbb{S}^{d-1}$ , denoted as  $\mathcal{P}_{\mathbb{S}^{d-1}}$ . The form of increment in (4.1.13) can be illustrated by a diagram in Figure 4.3(a).

**Deep Network for Optimizing Rate Reduction.** Notice that the increment is constructed to emulate the gradient ascent for the rate reduction  $\Delta R_e$ . Hence by transforming the features iteratively via the

**Algorithm 4.1** Training algorithm for ReduNet

---

**Input:**  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ ,  $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}_k\}_{k=1}^K$ ,  $\epsilon > 0$ ,  $\lambda$ , and a learning rate  $\eta$ .

**Output:** The learned parameters  $\{\mathbf{E}^\ell\}_{\ell=1}^L$ ,  $\{\{\mathbf{C}_k^\ell\}_{k=1}^K\}_{\ell=1}^L$ ,  $\{\gamma_k\}_{k=1}^K$

```

1: procedure REDUNETTRAINING( $\mathbf{X}, \boldsymbol{\Pi}, \epsilon, \lambda, \eta$ )
    # Define constants
    2:    $\alpha \leftarrow d/(N\epsilon^2)$ 
    3:   for  $k \in \{1, \dots, K\}$  do
    4:      $\alpha_k \leftarrow D/(\text{tr}(\boldsymbol{\Pi}_k)\epsilon^2)$ 
    5:      $\gamma_k \leftarrow \text{tr}(\boldsymbol{\Pi}_k)/D$ 
    6:   end for

    # ReduNet layer-by-layer iteration
    7:    $\mathbf{Z}^1 = [\mathbf{z}_1^1, \dots, \mathbf{z}_N^1] \leftarrow \mathbf{X}$                                  $\triangleright$  Initialize the ReduNet per-layer iteration
    8:   for  $\ell \in \{1, \dots, L\}$  do
        # Step 1: Compute network parameters  $\mathbf{E}^\ell, \{\mathbf{C}_k^\ell\}_{k=1}^K$ 
        9:      $\mathbf{E}^\ell \leftarrow \alpha (\mathbf{I} + \alpha \mathbf{Z}^\ell (\mathbf{Z}^\ell)^\top)^{-1} \in \mathbb{R}^{d \times d}$ 
        10:    for  $k \in \{1, \dots, K\}$  do
        11:       $\mathbf{C}_k^\ell \leftarrow \alpha_k (\mathbf{I} + \alpha_k \mathbf{Z}^\ell \boldsymbol{\Pi}_k (\mathbf{Z}^\ell)^\top)^{-1} \in \mathbb{R}^{d \times d}$ 
        12:    end for

        # Step 2: Update features  $\mathbf{Z}^\ell$ 
        13:    for  $i \in \{1, \dots, N\}$  do
        14:       $\hat{\pi}(\mathbf{z}_i^\ell) \leftarrow \text{softmax}(-\lambda[\|\mathbf{C}_1^\ell \mathbf{z}_i^\ell\|_2, \dots, \|\mathbf{C}_K^\ell \mathbf{z}_i^\ell\|_2]) \in [0, 1]^K$      $\triangleright$  Compute soft assignments  $\hat{\pi}(\mathbf{z}_i^\ell)$ 
        15:       $\mathbf{z}_i^{\ell+1} \leftarrow \mathcal{P}_{\mathbb{S}^{d-1}} \left( \mathbf{z}_i^\ell + \eta \left( \mathbf{E}^\ell \mathbf{z}_i^\ell - \sum_{k=1}^K \gamma_k \hat{\pi}_k(\mathbf{z}_i^\ell) \mathbf{C}_k^\ell \mathbf{z}_i^\ell \right) \right) \in \mathbb{R}^d$      $\triangleright$  Update features  $\mathbf{z}_i^{\ell+1}$  from  $\mathbf{z}_i^\ell$ 
        16:    end for
        17:  end for
        18:  return  $\{\mathbf{E}^\ell\}_{\ell=1}^L, \{\{\mathbf{C}_k^\ell\}_{k=1}^K\}_{\ell=1}^L, \{\gamma_k\}_{k=1}^K$                                  $\triangleright$  Return all network parameters.
    19: end procedure

```

---

above process, we expect the rate reduction to increase, as we will see in the experimental section. This iterative process, once converged say after  $L$  iterations, gives the desired feature map  $f(\mathbf{x}, \boldsymbol{\theta})$  on the input  $\mathbf{x} = \mathbf{z} \dots 0$ , precisely in the form of a *deep network*, in which each layer has the structure shown in Figure 4.3 left:

$$\begin{aligned} f(\mathbf{x}, \boldsymbol{\theta}) &= f^L \circ f^{L-1} \circ \dots \circ f^1 \circ f^0(\mathbf{z}^0), \\ f^\ell(\mathbf{z}^\ell, \boldsymbol{\theta}^\ell) &\doteq \mathbf{z}^{\ell+1} = \mathcal{P}_{\mathbb{S}^{d-1}}[\mathbf{z}^\ell + \eta \cdot g(\mathbf{z}^\ell, \boldsymbol{\theta}^\ell)], \\ g(\mathbf{z}^\ell, \boldsymbol{\theta}^\ell) &= \mathbf{E}^\ell \mathbf{z}^\ell - \sigma([\mathbf{C}_1^\ell \mathbf{z}^\ell, \dots, \mathbf{C}_K^\ell \mathbf{z}^\ell]). \end{aligned} \quad (4.1.14)$$

As this deep network is derived from maximizing the rate **reduction**, we call it the **ReduNet**. By comparing the architecture of ReduNet with those of popular empirically designed networks, **ResNet** and **NesNeXt** shown in Figure 4.3, the similarity is somewhat uncanny. Conceptually, ReduNet could also be used to justify the popular mixture of experts (**MoE**) architecture [SMM+17] as each parallel channel,  $\mathbf{C}_k^\ell$ , can be viewed as an “expert” trained for each class of objects.

We summarize the training and evaluation of ReduNet in Algorithm 4.1 and Algorithm 4.2, respectively. Notice that all parameters of the network are explicitly constructed layer by layer in a *forward propagation* fashion. The construction does not need any back propagation! The so-learned features can be directly used for classification, say via a nearest subspace classifier.

*Example 4.1.* To provide some intuition on how ReduNet transforms the features, we provide a simple example with mixed 3D Gaussians and visualize how the features are transformed in Figure 4.5. Consider a mixture of three Gaussian distributions in  $\mathbb{R}^3$  that is projected onto  $\mathbb{S}^2$ . We first generate data points for 3 classes: for  $k = 1, 2, 3$ ,  $\mathbf{X}_k = [\mathbf{x}_{k,1}, \dots, \mathbf{x}_{k,m}] \in \mathbb{R}^{3 \times m}$ ,  $\mathbf{x}_{k,i} \sim \mathcal{N}(\boldsymbol{\mu}_k, \sigma_k^2 \mathbf{I})$ , and  $\pi(\mathbf{x}_{k,i}) = k$ . We

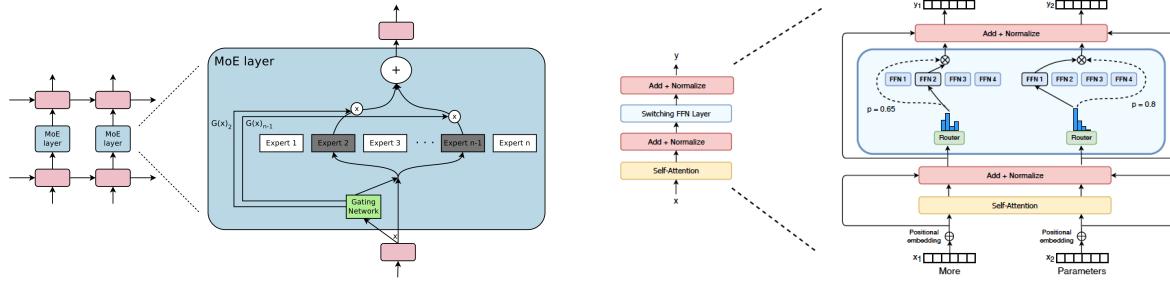


Figure 4.4: Left: a mixture of experts (MoE) deep network [SMM+17]. Right: a sparsity-promoting Switch Transformer [FZS22], used to implement MoE with 1.7 trillion parameters.

---

**Algorithm 4.2** Evaluation algorithm for ReduNet

---

**Input:** Input  $\mathbf{x} \in \mathbb{R}^D$ , network parameters  $\{\mathbf{E}^\ell\}_{\ell=1}^L, \{\{\mathbf{C}_k^\ell\}_{k=1}^K\}_{\ell=1}^L, \{\gamma_k\}_{k=1}^K$ , learning rate  $\lambda$

**Output:** feature  $\mathbf{z}^{L+1}$

```

1: procedure REDUNETEVALUATION( $\mathbf{x}$ )
2:    $\mathbf{z}^1 \leftarrow \mathbf{x} \in \mathbb{R}^D$                                  $\triangleright$  Initialize the ReduNet per-layer iteration
3:   for  $\ell \in \{1, \dots, L\}$  do
4:      $\hat{\pi}(\mathbf{z}^\ell) \leftarrow \text{softmax}(-\lambda [\|\mathbf{C}_1^\ell \mathbf{z}^\ell\|_2, \dots, \|\mathbf{C}_K^\ell \mathbf{z}^\ell\|_2]) \in [0, 1]^K$      $\triangleright$  Compute soft assignments  $\hat{\pi}(\mathbf{z}^\ell)$ 
5:      $\mathbf{z}^{\ell+1} \leftarrow \mathcal{P}_{\mathbb{S}^{d-1}} \left( \mathbf{z}^\ell + \eta \left( \mathbf{E}^\ell \mathbf{z}^\ell - \sum_{k=1}^K \gamma_k \hat{\pi}_k(\mathbf{z}^\ell) \mathbf{C}_k^\ell \mathbf{z}^\ell \right) \right) \in \mathbb{R}^d$      $\triangleright$  Update feature  $\mathbf{z}^{\ell+1}$  using  $\mathbf{z}^\ell$ 
6:   end for
7:   return  $\mathbf{z}^{L+1}$                                  $\triangleright$  Return the output features
8: end procedure

```

---

set  $m = 500, \sigma_1 = \sigma_2 = \sigma_3 = 0.1$ , and  $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3 \in \mathbb{S}^2$ . Then we project all the data points onto  $\mathbb{S}^2$ , i.e.,  $\mathbf{x}_{k,i}/\|\mathbf{x}_{k,i}\|_2$ . To construct the network (computing  $\mathbf{E}^\ell, \mathbf{C}_k^\ell$  for the  $\ell$ -th layer), we set the number of iterations/layers  $L = 2,000$ , step size  $\eta = 0.5$ , and precision  $\epsilon = 0.1$ . We do this only to demonstrate that our framework leads to stable deep networks even with thousands of layers. In practice, thousands of layers may not be necessary and one can stop whenever adding new layers gives diminishing returns. For this example, a couple of hundred layers is sufficient. Hence, the clear optimization objective gives a natural criterion for the depth of the network needed.

As shown in Figure 4.5, we can observe that after the mapping  $f(\cdot, \boldsymbol{\theta})$ , samples from the same class are highly compressed and converge to a single cluster and the angle between two different clusters is approximately  $\pi/2$ , which is well aligned with the optimal solution  $\mathbf{Z}^*$  of the MCR<sup>2</sup> loss in  $\mathbb{S}^2$ . MCR<sup>2</sup> loss of features on different layers can be found in Figure 4.5(c). Empirically, we find that the constructed ReduNet is able to maximize MCR<sup>2</sup> loss and converges stably and samples from the same class converge to one cluster and different clusters are orthogonal to each other. Moreover, when sampling new data points from the same distributions, we find that new samples from the same class consistently converge to the same cluster center as the training samples. ■

### 4.1.2 Convolutional Networks from Invariant Rate Reduction

In the previous section, we derived the layer-wise architecture of a deep network, the ReduNet, using unrolled optimization for the rate reduction objective. Specifically, the compression term  $R_\epsilon^c(\mathbf{Z} \mid \boldsymbol{\Pi})$  in (4.1.1) is designed to compress representations from the same class. However, this formulation does not account for possible domain transformation or deformation of the input data. For instance, shifting an object slightly to the right does not change the semantic label of an image. In this section, we will demonstrate how convolutional layers can be derived by maximizing a rate reduction objective that is invariant to certain domain deformations, such as image rotations and translations.

For many clustering or classification tasks (such as object detection in images), we consider two samples

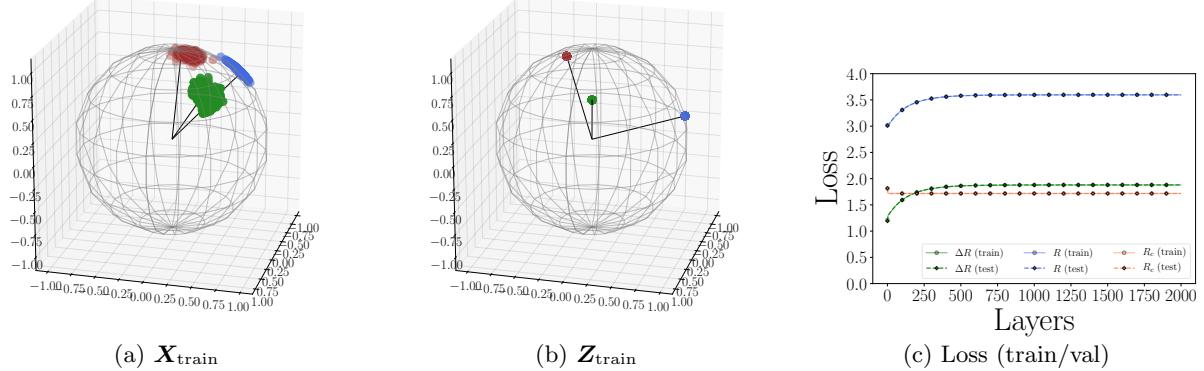


Figure 4.5: Original samples and learned representations for 3D Mixture of Gaussians. We visualize data points  $\mathbf{X}$  (before mapping  $f(\cdot, \theta)$ ) in (a) and learned features  $\mathbf{Z}$  (after mapping  $f(\cdot, \theta)$ ) in (b) by scatter plot. In each scatter plot, each color represents one class of samples. In (c), we also show the plots for the progression of values of the objective functions.

as *equivalent* if they differ by certain classes of domain deformations or augmentations  $\mathcal{T} = \{\tau\}$ . Hence, we are only interested in low-dimensional structures that are *invariant* to such deformations (i.e.,  $\mathbf{x} \in \mathcal{M}$  iff  $\tau(\mathbf{x}) \in \mathcal{M}$  for all  $\tau \in \mathcal{T}$ ), which are known to have sophisticated geometric and topological structures and can be difficult to learn precisely in practice, even with rigorously designed CNNs [CW16a]. In this framework, this can be formulated in a very natural way: all equivariant instances are to be embedded into the same subspace, so that the subspace itself is invariant to the transformations under consideration.

In many applications, such as serial data or imagery data, the semantic meaning (labels) of the data are *invariant* to certain transformations  $\mathbf{g} \in \mathbb{G}$  (for some group  $\mathbb{G}$ ) [CW16c; ZKR+17]. For example, the meaning of an audio signal is invariant to shift in time; and the identity of an object in an image is invariant to translation in the image plane. Hence, we prefer the feature mapping  $f(\mathbf{x}, \theta)$  is rigorously invariant to such transformations:

$$\text{Group Invariance: } f(\mathbf{x} \circ \mathbf{g}, \theta) \sim f(\mathbf{x}, \theta), \forall \mathbf{g} \in \mathbb{G}, \quad (4.1.15)$$

where “ $\sim$ ” indicates two features belonging to the same equivalent class. Although to ensure invariance or equivariance, convolutional operators have been common practice in deep networks [CW16c], it remains challenging in practice to train an (empirically designed) convolution network from scratch that can *guarantee* invariance even to simple transformations such as translation and rotation [AW18; ETT+17]. An alternative approach is to carefully design convolution filters of each layer so as to ensure translational invariance for a wide range of signals, say using wavelets as in ScatteringNet [BM13] and followup works [WB18]. However, in order to ensure invariance to generic signals, the number of convolutions needed usually grows exponentially with network depth. That is the reason why this type of network cannot be constructed so deep, usually only several layers.

Now, we show that the MCR<sup>2</sup> principle is compatible with invariance in a natural and precise way: we only need to assign all transformed versions  $\{\mathbf{x} \circ \mathbf{g} \mid \mathbf{g} \in \mathbb{G}\}$  into the same class as the data  $\mathbf{x}$  and map their features  $\mathbf{z}$  all to the same subspace  $\mathcal{S}$ . Hence, all group equivariant information is encoded only inside the subspace, and any classifier defined on the resulting set of subspaces will be automatically invariant to such group transformations. See Figure 4.6 for an illustration of the examples of 1D rotation and 2D translation. Next, we will rigorously show that when the group  $\mathbb{G}$  is circular 1D shifting, the resulting deep network naturally becomes a *multi-channel convolution network*. Because the so-constructed network only needs to ensure invariance for the given data  $\mathbf{X}$  or their features  $\mathbf{Z}$ , the number of convolutions needed actually remains constant through a very deep network, as opposed to the ScatteringNet.

**1D Serial Data and Shift Invariance** To classify one-dimensional data  $\mathbf{x} = [x(0), x(1), \dots, x(D-1)] \in \mathbb{R}^D$  invariant under shifting, we take  $\mathbb{G}$  to be the group of all circular shifts. Each observation  $\mathbf{x}_i$  generates a family  $\{\mathbf{x}_i \circ \mathbf{g} \mid \mathbf{g} \in \mathbb{G}\}$  of shifted copies, which are the columns of the circulant matrix  $\text{circ}(\mathbf{x}_i) \in \mathbb{R}^{D \times D}$

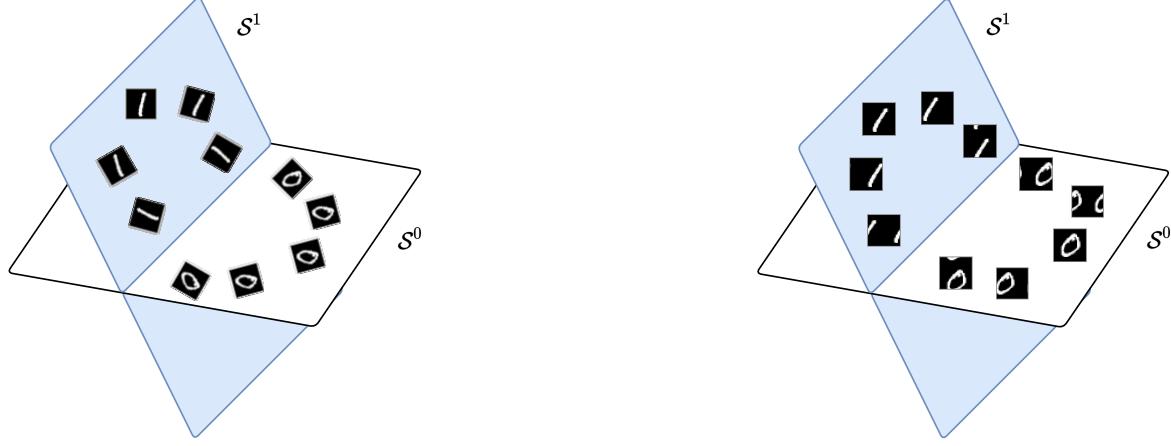


Figure 4.6: Illustration of the sought representation that is equivariant/invariant to image rotation (left) or translation (right): all transformed images of each class are mapped into the same subspace that is incoherent to other subspaces. The features embedded in each subspace are equivariant to the transformation group whereas each subspace is invariant to such transformations.

given by

$$\text{circ}(\mathbf{x}) \doteq \begin{bmatrix} x(0) & x(D-1) & \dots & x(2) & x(1) \\ x(1) & x(0) & x(D-1) & \dots & x(2) \\ \vdots & x(1) & x(0) & \ddots & \vdots \\ x(D-2) & \vdots & \ddots & \ddots & x(D-1) \\ x(D-1) & x(D-2) & \dots & x(1) & x(0) \end{bmatrix} \in \mathbb{R}^{D \times D}. \quad (4.1.16)$$

We refer the reader to [KS12] for properties of circulant matrices. For simplicity, let  $\mathbf{Z}^1 \doteq [\mathbf{z}_1^1, \dots, \mathbf{z}_N^1] = \mathbf{X} \in \mathbb{R}^{d \times N}$ .<sup>7</sup> Then what happens if we construct the ReduNet from their circulant families  $\text{circ}(\mathbf{Z}^1) = [\text{circ}(\mathbf{z}_1^1), \dots, \text{circ}(\mathbf{z}_N^1)] \in \mathbb{R}^{d \times (dN)}$ ? That is, we want to compress and map all these into the same subspace by the ReduNet.

Notice that now the data covariance matrix:

$$\begin{aligned} \text{circ}(\mathbf{Z}^1)\text{circ}(\mathbf{Z}^1)^\top &= [\text{circ}(\mathbf{z}_1^1), \dots, \text{circ}(\mathbf{z}_N^1)] [\text{circ}(\mathbf{z}_1^1), \dots, \text{circ}(\mathbf{z}_N^1)]^\top \\ &= \sum_{i=1}^N \text{circ}(\mathbf{z}_i^1)\text{circ}(\mathbf{z}_i^1)^\top \in \mathbb{R}^{d \times d} \end{aligned} \quad (4.1.17)$$

associated with this family of samples is *automatically* a (symmetric) circulant matrix. Moreover, because the circulant property is preserved under sums, inverses, and products, the matrices  $\mathbf{E}^1$  and  $\mathbf{C}_k^1$  are also automatically circulant matrices, whose application to a feature vector  $\mathbf{z} \in \mathbb{R}^d$  can be implemented using circular convolution “ $\circledast$ ”. Specifically, we have the following proposition.

**Proposition 4.1** (Convolution structures of  $\mathbf{E}^1$  and  $\mathbf{C}_k^1$ ). *The matrix*

$$\mathbf{E}^1 = \alpha(\mathbf{I} + \alpha\text{circ}(\mathbf{Z}^1)\text{circ}(\mathbf{Z}^1)^\top)^{-1} \quad (4.1.18)$$

*is a circulant matrix and represents a circular convolution:*

$$\mathbf{E}^1 \mathbf{z} = \mathbf{e}_1 \circledast \mathbf{z},$$

*where  $\mathbf{e}_1 \in \mathbb{R}^d$  is the first column vector of  $\mathbf{E}^1$  and “ $\circledast$ ” is circular convolution defined as*

$$(\mathbf{e}_1 \circledast \mathbf{z})_i \doteq \sum_{j=0}^{d-1} \mathbf{e}_1(j)x(i+d-j \bmod d).$$

<sup>7</sup>Again, to simplify discussion, we assume for now that the initial features  $\mathbf{Z}^1$  are  $\mathbf{X}$  themselves hence have the same dimension  $d$ , i.e.,  $D = d$ . But that does not need to be the case as we will soon see that we need to lift  $\mathbf{X}$  to a higher dimension.

Similarly, the matrices  $\mathbf{C}_k^1$  associated with any subsets of  $\mathbf{Z}^1$  are also circular convolutions.

Not only do the first-layer parameters  $\mathbf{E}^1$  and  $\mathbf{C}_k^1$  of the ReduNet become circulant convolutions but also the next-layer features remain circulant matrices. That is, the incremental feature transform in (4.1.13) applied to all shifted versions of a  $\mathbf{z}^1 \in \mathbb{R}^d$ , given by

$$\text{circ}(\mathbf{z}^1) + \eta \cdot \mathbf{E}^1 \text{circ}(\mathbf{z}^1) - \eta \cdot \boldsymbol{\sigma}([\mathbf{C}_1^1 \text{circ}(\mathbf{z}^1), \dots, \mathbf{C}_K^1 \text{circ}(\mathbf{z}^1)]), \quad (4.1.19)$$

is a circulant matrix. This implies that there is no need to construct circulant families from the second layer features as we did for the first layer. By denoting

$$\mathbf{z}^2 \propto \mathbf{z}^1 + \eta \cdot g(\mathbf{z}^1, \boldsymbol{\theta}^1) = \mathbf{z}^1 + \eta \cdot \mathbf{e}_1 \circledast \mathbf{z}^1 - \eta \cdot \boldsymbol{\sigma}([\mathbf{c}_1^1 \circledast \mathbf{z}^1, \dots, \mathbf{c}_K^1 \circledast \mathbf{z}^1]), \quad (4.1.20)$$

the features at the next level can be written as

$$\text{circ}(\mathbf{Z}^2) = [\text{circ}(\mathbf{z}_1^1 + \eta g(\mathbf{z}_1^1, \boldsymbol{\theta}^1)), \dots, \text{circ}(\mathbf{z}_N^1 + \eta g(\mathbf{z}_N^1, \boldsymbol{\theta}^1))].$$

Continuing inductively, we see that all matrices  $\mathbf{E}^\ell$  and  $\mathbf{C}_k^\ell$  based on such  $\text{circ}(\mathbf{Z}^\ell)$  are circulant, and so are all features. By virtue of the properties of the data, ReduNet has taken the form of a convolutional network, *with no need to explicitly choose this structure!*

**A Fundamental Trade-off between Invariance and Sparsity.** There is one problem though: In general, the set of all circular permutations of a vector  $\mathbf{z}$  gives a full-rank matrix. That is, the  $d$  “augmented” features associated with each sample (hence each class) typically already span the entire space  $\mathbb{R}^d$ . For instance, all shifted versions of a delta function  $\delta(d)$  can generate any other signal as their (dense) weighted superposition. The MCR<sup>2</sup> objective (3.4.12) will not be able to distinguish classes as different subspaces.

One natural remedy is to improve the separability of the data by “lifting” the original signal to a higher dimensional space, e.g., by taking their responses to multiple, filters  $\mathbf{k}_1, \dots, \mathbf{k}_C \in \mathbb{R}^d$ :

$$\mathbf{z}[c] = \mathbf{k}_c \circledast \mathbf{x} = \text{circ}(\mathbf{k}_c) \mathbf{x} \in \mathbb{R}^d, \quad c = 1, \dots, C. \quad (4.1.21)$$

The filters can be pre-designed invariance-promoting filters,<sup>8</sup> or adaptively learned from the data,<sup>9</sup> or randomly selected as we do in our experiments. This operation lifts each original signal  $\mathbf{x} \in \mathbb{R}^d$  to a  $C$ -channel feature, denoted as  $\bar{\mathbf{z}} \doteq [\mathbf{z}[1], \dots, \mathbf{z}[C]]^\top \in \mathbb{R}^{C \times d}$ . Then, we may construct the ReduNet on vector representations of  $\bar{\mathbf{z}}$ , denoted as  $(\bar{\mathbf{z}}) \doteq [\mathbf{z}[1]^\top, \dots, \mathbf{z}[C]^\top] \in \mathbb{R}^{dC}$ . The associated circulant version  $\text{circ}(\bar{\mathbf{z}})$  and its data covariance matrix, denoted as  $\bar{\Sigma}(\bar{\mathbf{z}})$ , for all its shifted versions are given as:

$$\text{circ}(\bar{\mathbf{z}}) \doteq \begin{bmatrix} \text{circ}(\mathbf{z}[1]) \\ \vdots \\ \text{circ}(\mathbf{z}[C]) \end{bmatrix} \in \mathbb{R}^{dC \times d}, \quad \bar{\Sigma}(\bar{\mathbf{z}}) \doteq \begin{bmatrix} \text{circ}(\mathbf{z}[1]) \\ \vdots \\ \text{circ}(\mathbf{z}[C]) \end{bmatrix} [\text{circ}(\mathbf{z}[1])^\top, \dots, \text{circ}(\mathbf{z}[C])^\top] \in \mathbb{R}^{dC \times dC}, \quad (4.1.22)$$

where  $\text{circ}(\mathbf{z}[c]) \in \mathbb{R}^{d \times d}$  with  $c \in [C]$  is the circulant version of the  $c$ -th channel of the feature  $\bar{\mathbf{z}}$ . Then the columns of  $\text{circ}(\bar{\mathbf{z}})$  will only span at most a  $d$ -dimensional proper subspace in  $\mathbb{R}^{dC}$ . However, this simple lifting operation (if linear) is not sufficient to render the classes separable yet—features associated with other classes will span the *same*  $d$ -dimensional subspace. This reflects a fundamental conflict between invariance and linear (subspace) modeling: *one cannot hope for arbitrarily shifted and superposed signals to belong to the same class.*

One way of resolving this conflict is to leverage additional structure within each class, in the form of *sparsity*: signals within each class are not generated as an arbitrary linear superposition of some base atoms (or motifs), but only *sparse* combinations of them and their shifted versions, as shown in Figure 4.7. More

<sup>8</sup>For 1D signals like audio, one may consider the conventional short-time Fourier transform (STFT); for 2D images, one may consider 2D wavelets as in the ScatteringNet [BM13].

<sup>9</sup>For learned filters, one can learn filters as the principal components of samples as in the PCANet [CJG+15] or from convolution dictionary learning [LB19; QLZ19].

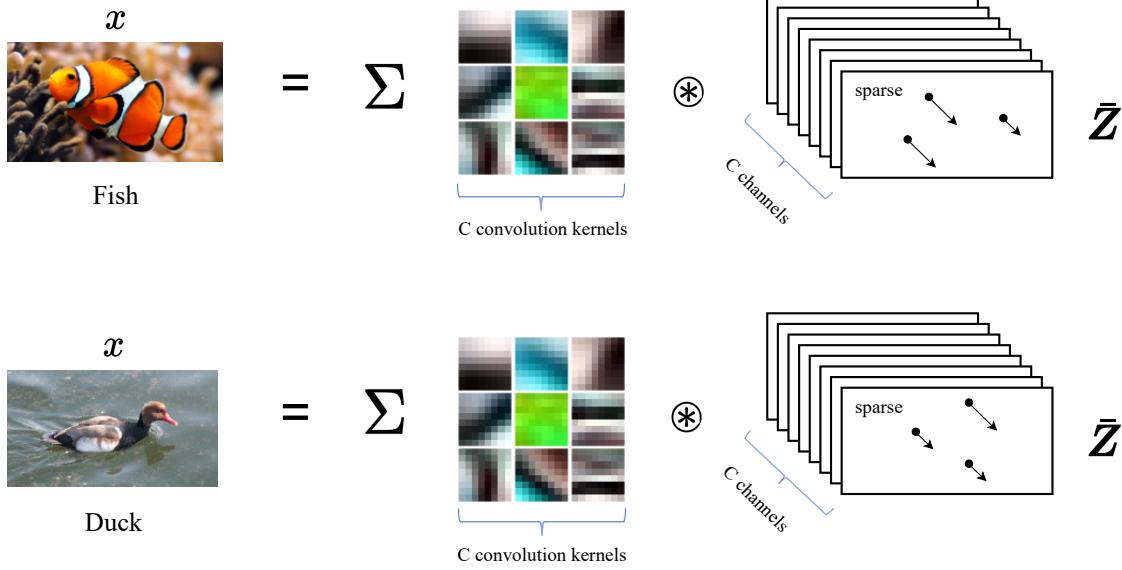


Figure 4.7: Each input signal  $x$  (an image here) can be represented as a superposition of sparse convolutions with multiple kernels  $d_c$  in a dictionary  $D$ .

precisely, let  $D_k = [d_{k,1}, \dots, d_{k,C}]$  denote a matrix with a collection of atoms associated for class  $k$ , also known as a dictionary, then each signal  $x$  in this class is sparsely generated as:

$$x = d_{k,1} \circledast z_1 + \dots + d_{k,C} \circledast z_C = \text{circ}(D_k)z, \quad (4.1.23)$$

for some sparse vector  $z$ . Signals in different classes are then generated by different dictionaries whose atoms (or motifs) are incoherent from one another. Due to incoherence, signals in one class are unlikely to be sparsely represented by atoms in any other class. Hence all signals can be represented as

$$x = [\text{circ}(D_1), \text{circ}(D_2), \dots, \text{circ}(D_K)]\bar{z}, \quad (4.1.24)$$

where  $\bar{z}$  is sparse.<sup>10</sup> There is a vast literature on how to learn the most compact and optimal sparsifying dictionaries from sample data, e.g. [LB19; QLZ19] and subsequently solve the inverse problem and compute the associated sparse code  $z$  or  $\bar{z}$ . Recent studies of [QLZ20; QZL+20a] even show that under broad conditions the convolution dictionary learning problem can be solved effectively and efficiently.

Nevertheless, for tasks such as classification, we are not necessarily interested in the precise optimal dictionary nor the precise sparse code for each individual signal. We are mainly interested if collectively the set of sparse codes for each class are adequately separable from those of other classes. Under the assumption of the sparse generative model, if the convolution kernels  $\{\mathbf{k}_c\}_{c=1}^C$  match well with the ‘‘transpose’’ or ‘‘inverse’’ of the above sparsifying dictionaries  $D = [D_1, \dots, D_K]$ , also known as the *analysis filters* [NDE+13; RE14], signals in one class will only have high responses to a small subset of those filters and low responses to others (due to the incoherence assumption). Nevertheless, in practice, often a sufficiently large number of, say  $C$ , random filters  $\{\mathbf{k}_c\}_{c=1}^C$  suffices to ensure that the extracted  $C$ -channel features

$$[\mathbf{k}_1 \circledast x, \mathbf{k}_2 \circledast x, \dots, \mathbf{k}_C \circledast x]^\top = [\text{circ}(\mathbf{k}_1)x, \dots, \text{circ}(\mathbf{k}_C)x]^\top \in \mathbb{R}^{C \times d} \quad (4.1.25)$$

for different classes have different response patterns to different filters hence make different classes separable [CJG+15].

Therefore, in our framework, to a large extent the number of channels (or the width of the network) truly plays the role as the *statistical resource* whereas the number of layers (the depth of the network) plays

<sup>10</sup>Notice that similar sparse representation models have long been proposed and used for classification purposes in applications such as face recognition, demonstrating excellent effectiveness [WWG+12; WYG+09]. Recently, the convolution sparse coding model has been proposed by [PRE17] as a framework for interpreting the structures of deep convolution networks.

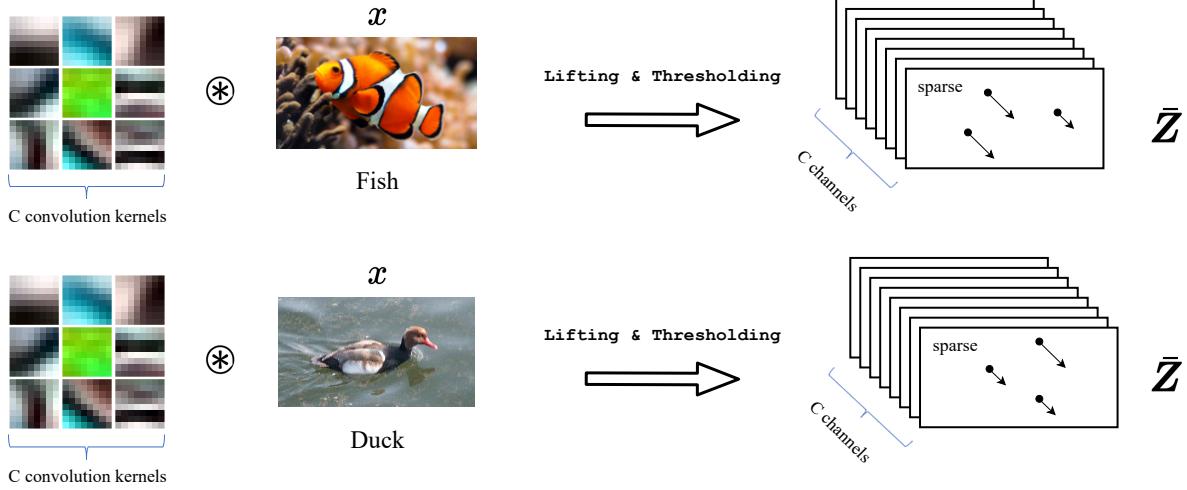


Figure 4.8: Estimate the sparse code  $\bar{z}$  of an input signal  $x$  (an image here) by taking convolutions with multiple kernels  $k_c$  and then sparsifying.

the role as the *computational resource*. The theory of compressive sensing precisely characterizes how many measurements are needed in order to preserve the intrinsic low-dimensional structures (including separability) of the data [WM21].

The multi-channel responses  $\bar{z}$  should be sparse. So to approximate the sparse code  $\bar{z}$ , we may take an entry-wise *sparsity-promoting nonlinear thresholding*, say  $\tau(\cdot)$ , on the above filter outputs by setting low (say absolute value below  $\epsilon$ ) or negative responses to be zero:

$$\bar{z} \doteq \tau \left( [\text{circ}(k_1)x, \dots, \text{circ}(k_C)x]^\top \right) \in \mathbb{R}^{C \times d}. \quad (4.1.26)$$

Figure 4.8 illustrates the basic ideas. One may refer to [RE14] for a more systematical study on the design of the sparsifying thresholding operator. Nevertheless, here we are not so interested in obtaining the best sparse codes as long as the codes are sufficiently separable. Hence the nonlinear operator  $\tau$  can be simply chosen to be a soft thresholding or a ReLU. These presumably sparse features  $\bar{z}$  can be assumed to lie on a lower-dimensional (nonlinear) submanifold of  $\mathbb{R}^{dC}$ , which can be linearized and separated from the other classes by subsequent ReduNet layers, as illustrated later in Figure 4.9.

The ReduNet constructed from circulant version of these multi-channel features  $\bar{Z} \doteq [\bar{z}_1, \dots, \bar{z}_N] \in \mathbb{R}^{C \times d \times N}$ , i.e.,  $\text{circ}(\bar{Z}) \doteq [\text{circ}(\bar{z}_1), \dots, \text{circ}(\bar{z}_N)] \in \mathbb{R}^{dC \times dN}$ , retains the good invariance properties described above: the linear operators, now denoted as  $\bar{E}$  and  $\bar{C}_k$ , remain block circulant, and represent *multi-channel 1D circular convolutions*. Specifically, we have the following result.

**Proposition 4.2** (Multi-channel convolution structures of  $\bar{E}$  and  $\bar{C}_k$ ). *The matrix*

$$\bar{E} \doteq \alpha (\mathbf{I} + \alpha \text{circ}(\bar{Z})\text{circ}(\bar{Z})^\top)^{-1} \quad (4.1.27)$$

is block circulant, i.e.,

$$\bar{E} = \begin{bmatrix} \bar{E}_{1,1} & \cdots & \bar{E}_{1,C} \\ \vdots & \ddots & \vdots \\ \bar{E}_{C,1} & \cdots & \bar{E}_{C,C} \end{bmatrix} \in \mathbb{R}^{dC \times dC},$$

where each  $\bar{E}_{c,c'} \in \mathbb{R}^{d \times d}$  is a circulant matrix. Moreover,  $\bar{E}$  represents a multi-channel circular convolution, i.e., for any multi-channel signal  $\bar{z} \in \mathbb{R}^{C \times n}$  we have

$$\bar{E} \cdot \text{vec}(\bar{z}) = \text{vec}(\bar{e} \circledast \bar{z}).$$

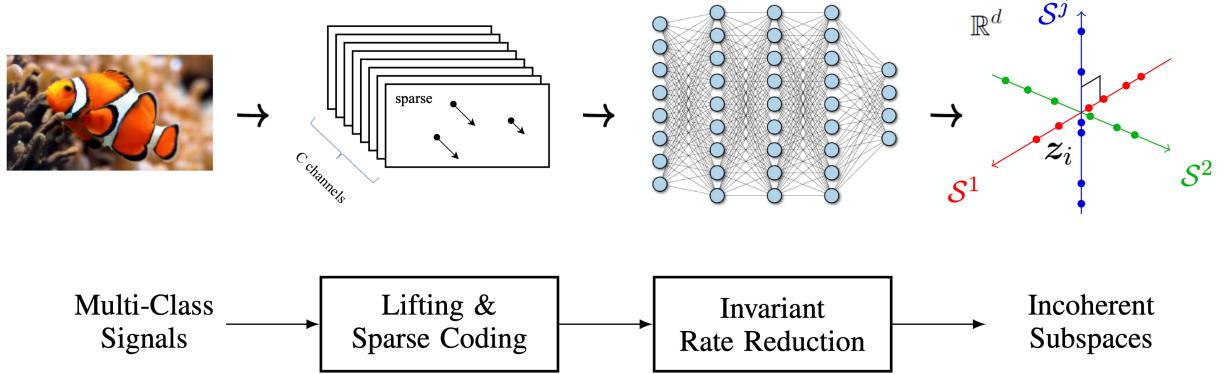


Figure 4.9: The overall process for classifying multi-class signals with shift invariance: Multi-channel lifting, sparse coding, followed by a multi-channel convolution ReduNet for invariant rate reduction. These components are *necessary* in order to map shift-invariant multi-class signals to incoherent (linear) subspaces as an LDR. Note that the architectures of most modern deep neural networks resemble this process. The so-learned LDR facilitates subsequent tasks such as classification.

In above,  $\bar{\mathbf{e}} \in \mathbb{R}^{C \times C \times d}$  is a multi-channel convolutional kernel with  $\bar{\mathbf{e}}[c, c'] \in \mathbb{R}^d$  being the first column vector of  $\bar{\mathbf{E}}_{c, c'}$ , and  $\bar{\mathbf{e}} \circledast \bar{\mathbf{z}} \in \mathbb{R}^{C \times d}$  is the multi-channel circular convolution defined as

$$(\bar{\mathbf{e}} \circledast \bar{\mathbf{z}})[c] \doteq \sum_{c'=1}^C \bar{\mathbf{e}}[c, c'] \circledast \bar{\mathbf{z}}[c'], \quad \forall c = 1, \dots, C.$$

Similarly, the matrices  $\bar{\mathbf{C}}_k$  associated with any subsets of  $\bar{\mathbf{Z}}$  are also multi-channel circular convolutions.

From Proposition 4.2, shift invariant ReduNet is a deep convolutional network for multi-channel 1D signals by construction. Notice that even if the initial lifting kernels are separated (4.1.26), the matrix inverse in (4.1.27) for computing  $\bar{\mathbf{E}}$  (similarly for  $\bar{\mathbf{C}}_k$ ) introduces “cross talk” among all  $C$  channels. Hence, these multi-channel convolutions in general are *not* depth-wise separable, unlike the Xception nets [Cho17] that were once suggested to simplify multi-channel convolutional neural networks.<sup>11</sup>

*Remark 4.2* (Reducing Computational Complexity in the Frequency Domain). The calculation of  $\bar{\mathbf{E}}$  in (4.1.27) requires inverting a matrix of size  $dC \times dC$ , which in general has complexity  $O(d^3 C^3)$ . Nevertheless, by using the fact that a circulant matrix can be diagonalized by the Discrete Fourier Transform (DFT) matrix, the complexity can be significantly reduced. As shown in [CYY+22], to compute  $\bar{\mathbf{E}}$  and  $\bar{\mathbf{C}}_k \in \mathbb{R}^{dC \times dC}$ , we only need to compute in the frequency domain the inverse of  $C \times C$  blocks for  $d$  times hence the overall complexity becomes  $O(dC^3)$ .

**Overall Network Architecture and Comparison.** Following the above derivation, we see that in order to find a linear discriminative representation (LDR) for multiple classes of signals/images that is invariant to translation, sparse coding, a multi-layer architecture with multi-channel convolutions, different nonlinear activation, and spectrum computing all become *necessary* components for achieving the objective effectively and efficiently. Figure 4.9 illustrates the overall process of learning such a representation via invariant rate reduction on the input sparse codes.

*Example 4.2* (Invariant Classification of Digits). We next provide an empirical performance of the ReduNet on learning *rotation* invariant features on the real 10-class MNIST dataset. We impose a polar grid on the image  $\mathbf{x} \in \mathbb{R}^{H \times W}$ , with its geometric center being the center of the 2D polar grid (as illustrated in Figure 4.10). For each radius  $r_i$ ,  $i \in [C]$ , we can sample  $\Gamma$  pixels with respect to each angle  $\gamma_l = l \cdot (2\pi/\Gamma)$  with  $l \in [\Gamma]$ . Then given a sample image  $\mathbf{x}$  from the dataset, we represent the image in the (sampled) polar coordinate as a multi-channel signal  $\mathbf{x}_p \in \mathbb{R}^{\Gamma \times C}$ . The goal here is to learn a rotation invariant representation, i.e., we

<sup>11</sup>It remains open what additional structures on the data would lead to depth-wise separable convolutions.

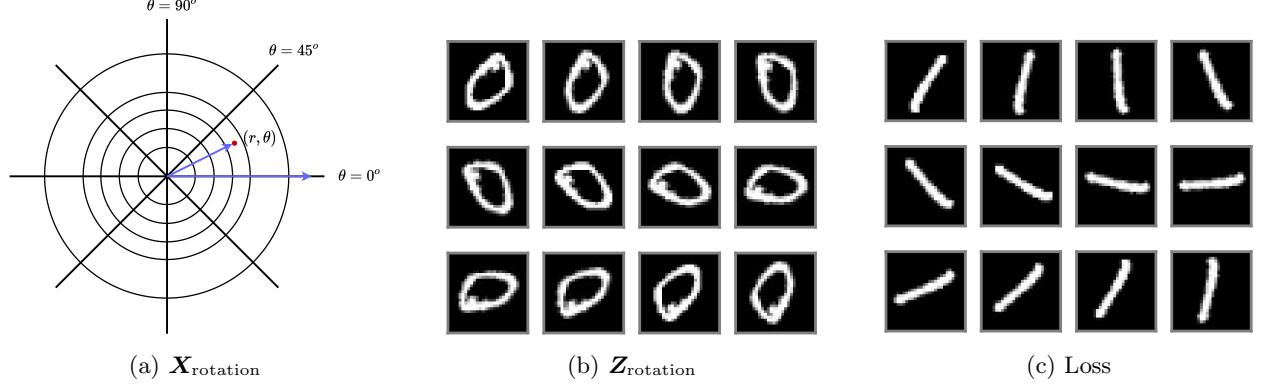


Figure 4.10: Examples of rotated images of MNIST digits, each by  $18^\circ$ . (Left) Diagram for polar coordinate representation; (Right) Rotated images of digit ‘0’ and ‘1’.

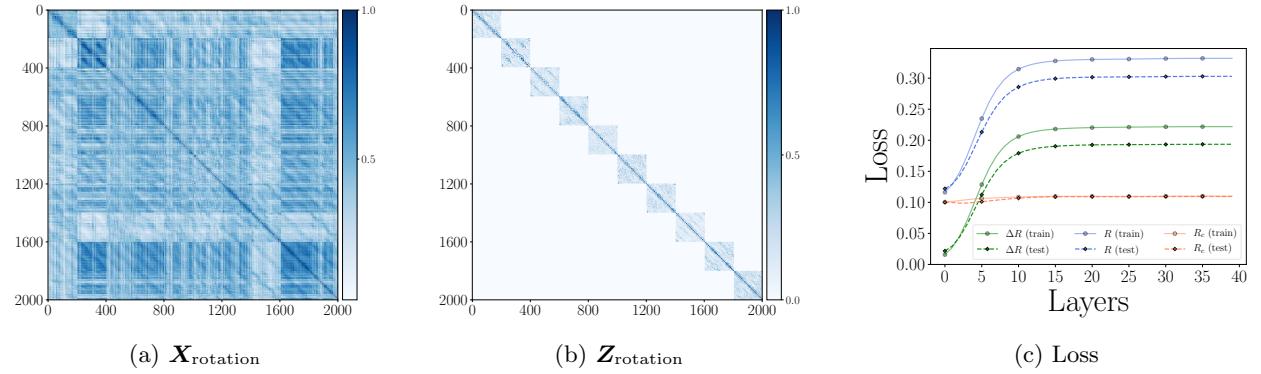


Figure 4.11: (a)(b) are heatmaps of cosine similarity among rotated training data  $\mathbf{X}_{\text{rotation}}$  and learned features  $\bar{\mathbf{Z}}_{\text{rotation}}$  for rotation invariance. (d) visualizes the training/val MCR<sup>2</sup> losses across layers.

expect to learn  $f(\cdot, \theta)$  such that  $\{f(\mathbf{x}_p \circ \mathbf{g}, \theta)\}_{\mathbf{g} \in \mathbb{G}}$  lie in the same subspace, where  $\mathbf{g}$  is the cyclic-shift in polar angle. We use  $N = 100$  training samples (10 from each class) and set  $\Gamma = 200$ ,  $C = 15$  for polar sampling. By performing the above sampling in polar coordinate, we can obtain the data matrix  $\mathbf{X}_p \in \mathbb{R}^{(\Gamma \cdot C) \times N}$ . For the ReduNet, we set the number of layers/iterations  $L = 40$ , precision  $\epsilon = 0.1$ , step size  $\eta = 0.5$ . Before the first layer, we perform lifting of the input by 1D circulant-convolution with 20 random Gaussian kernels of size 5.

To evaluate the learned representation, each training sample is augmented by 20 of its rotated version, each shifted with stride=10. We compute the cosine similarities among the  $m \times 20$  augmented training inputs  $\mathbf{X}_{\text{rotation}}$  and the results are shown in Figure 4.11 (a). We compare the cosine similarities among the learned features of all the augmented versions, i.e.,  $\bar{\mathbf{Z}}_{\text{rotation}}$  and summarize the results in Figure 4.11 (b). As we see, the so constructed rotation-invariant ReduNet is able to map the training data (as well as all its rotated versions) from the 10 different classes into 10 nearly orthogonal subspaces. That is, the learnt subspaces are truly invariant to shift transformation in polar angle. Next, we randomly draw another 100 test samples followed by the same augmentation procedure. In Figure 4.11 (c), we visualize the MCR<sup>2</sup> loss on the  $\ell$ -th layer representation of the ReduNet on the training and test dataset. From these results, we can find that the constructed ReduNet is indeed able to maximize the MCR<sup>2</sup> loss as well as generalize to the test data. ■

## 4.2 White-Box Transformers from Unrolled Optimization

As we have seen in the previous section, we use the problem of classification to provide a rigorous interpretation for main architectural characteristics of popular deep networks such as the ResNet and the CNN: each

layer of such networks can be viewed as to imitate a gradient step which increases the rate reduction (or information gain) objective. This perspective also leads to a somewhat surprising fact: the parameters and operators of the layers of such a deep network, the ReduNet, can be computed in a purely forward fashion.

Despite the theoretical and conceptual importance of the ReduNet, several factors limit it from being very practical. First, as we have discussed in the above, the computational cost of computing the matrix operators in each layer in a forward fashion can be very high. Second, the so-computed operators may not be so effective in optimizing the objective and it might take thousands of iterations (hence layers). As we have seen in Section 2.3.3 for LISTA, these two issues can be addressed by allowing to optimize those operators and make them learnable via back-propagation.<sup>12</sup>

The supervised classification setting in which the ReduNet was derived is also somewhat limiting. In practice, an image might not belong to a single class as it may contain multiple objects. Hence it would be more general to assume that different regions of the image belong to different low-dimensional models (say a Gaussian or a subspace). As we will see, such a generalization would lead to a both simple and general architecture which unifies the rate reduction and the denoising operations that we have seen in the previous chapter. Moreover, the so-obtained architecture resembles the popular Transformer architecture.

#### 4.2.1 Unrolled Optimization for Sparse Rate Reduction

We consider a general learning setup associated with real-world signals. Let  $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$  denote random variables representing our data source. In vision tasks, each  $\mathbf{x}_i \in \mathbb{R}^D$  is interpreted as a *token*, typically corresponding to an image patch. In language tasks, each  $\mathbf{x}_i \in \mathbb{R}^D$  is interpreted as an *token embedding*, i.e., a continuous vector representation of a discrete token such as a word or subword.<sup>13</sup> The  $\mathbf{x}_i$ 's may have arbitrary correlation structures. We use  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N] \in \mathbb{R}^{d \times N}$  to denote the random variables that defines our representations, where  $\mathbf{z}_i \in \mathbb{R}^d$  is the representation of the corresponding token  $\mathbf{x}_i \in \mathbb{R}^D$ .

*Remark 4.3.* In transformers, each input sample is typically converted into a sequence of *tokens*. A token is a basic unit of information derived from the raw input: in natural language processing, tokens are typically words or subwords; in computer vision, they correspond to image patches; and in other modalities, they may represent time steps, spatial locations, or other domain-specific units. A *token embedding* is a continuous vector representation of a token that serves as the input to a transformer. It maps each token to a point in a high-dimensional space, enabling the model to process symbolic inputs using numerical computation. A *token representation* is a vector that encodes the semantic or structural information of a token, typically produced by the intermediate or final layers of a transformer. These representations are designed to capture meaningful features of the input that are useful for downstream tasks such as classification, generation, or regression. Please refer to Section 7.2 for more details about these concepts in implementations.

**Objective for Learning a Structured and Compact Representation.** Following the framework of rate reduction Section 4.1, we contend that the goal of representation learning is to find a feature mapping  $f: \mathbf{X} \in \mathbb{R}^{D \times N} \rightarrow \mathbf{Z} \in \mathbb{R}^{d \times N}$  which transforms input tokens  $\{\mathbf{x}_i\}_{i=1}^N \subset \mathbb{R}^D$  with a potentially nonlinear and multi-modal distribution to a (piecewise) *linearized and compact* token representations  $\{\mathbf{z}_i\}_{i=1}^N \subset \mathbb{R}^d$ . While the joint distribution of tokens representations  $\{\mathbf{z}_i\}_{i=1}^N$  may be sophisticated (and task-specific), we further contend that it is reasonable and practical to require that the target marginal distribution of individual token representations should be highly compressed and structured, amenable for compact coding. Particularly, we require the distribution to be a *mixture of low-dimensional (say  $K$ ) Gaussian distributions*, such that the  $k$ -th Gaussian has mean  $\mathbf{0} \in \mathbb{R}^d$ , covariance  $\Sigma_k \succeq \mathbf{0} \in \mathbb{R}^{d \times d}$ , and support spanned by the orthonormal basis  $\mathbf{U}_k \in \mathbb{R}^{d \times p}$ . We denote  $\mathbf{U}_{[K]} = \{\mathbf{U}_k\}_{k=1}^K$  to be the set of bases of all Gaussians. Hence, to maximize the *information gain* [MTS22] for the final token representations, we wish to maximize their rate reduction (see Section 3.4.2), i.e.,

$$\max_{\mathbf{Z} \in \mathbb{R}^{d \times N}} \Delta R_\epsilon(\mathbf{Z} | \mathbf{U}_{[K]}) \doteq R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z} | \mathbf{U}_{[K]}). \quad (4.2.1)$$

<sup>12</sup>Or, perhaps, by a mixture of both forward and backward optimization.

<sup>13</sup>With a slight abuse of terminology, we refer to both the discrete tokens and their associated embeddings simply as tokens throughout this chapter for convenience.

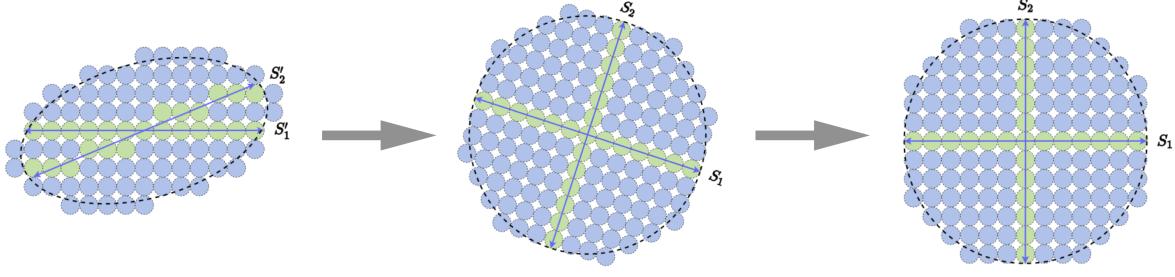


Figure 4.12: **Comparison of three sets of representations via rate reduction and sparsity.** Each  $S_i$  represents one linear subspace, and the number of blue balls represents the difference between the coding rates  $\Delta R_\epsilon(\mathbf{Z} \mid \mathbf{U}_{[K]}) = R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z} \mid \mathbf{U}_{[K]})$ .

Here, the first term  $R_\epsilon$  is an estimate of the lossy coding rate for the whole set of token representations. More specifically, if we view the token representations  $\{\mathbf{z}_i\}_{i=1}^N$  as i.i.d. samples from a single zero-mean Gaussian, their lossy coding rate subject to a quantization precision  $\epsilon > 0$  is given as

$$R_\epsilon(\mathbf{Z}) \doteq \frac{1}{2} \log \det \left( \mathbf{I} + \frac{d}{N\epsilon^2} \mathbf{Z}^\top \mathbf{Z} \right) = \frac{1}{2} \log \det \left( \mathbf{I} + \frac{d}{N\epsilon^2} \mathbf{Z} \mathbf{Z}^\top \right). \quad (4.2.2)$$

The second term  $R_\epsilon^c$  is an estimate of the lossy coding rate under the codebook  $\mathbf{U}_{[K]}$ , which is given as

$$R_\epsilon^c(\mathbf{Z} \mid \mathbf{U}_{[K]}) \doteq \sum_{k=1}^K R_\epsilon(\mathbf{U}_k^\top \mathbf{Z}) = \frac{1}{2} \sum_{k=1}^K \log \det \left( \mathbf{I} + \frac{p}{N\epsilon^2} (\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z}) \right). \quad (4.2.3)$$

*Remark 4.4.* The expression (4.2.3) for the coding rate can be viewed as a generalization of the coding rate  $R_\epsilon^c$  used in the original rate reduction objective (3.4.13). In particular, the original objective is defined with respect to a set of known membership labels  $\{\Pi_k\}$  specific to the particular data realization  $\mathbf{X}$ . In contrast, the current objective is defined with respect to subspaces  $\mathbf{U}_{[K]}$ , which are independent of any particular realization but are assumed to support the distribution of token representations. Suppose that a token representation  $\mathbf{z}_i$  belongs to a subspace  $\mathbf{U}_k$  and these subspaces are approximately orthogonal to each other, i.e.,  $\mathbf{U}_k^\top \mathbf{U}_l \approx \mathbf{0}$  for all  $k \neq l$ . Then, one can verify that the projections  $\mathbf{U}_k \mathbf{U}_k^\top \mathbf{z}_i = \mathbf{z}_i$  and  $\mathbf{U}_l \mathbf{U}_l^\top \mathbf{z}_i \approx \mathbf{0}$  for all  $l \neq k$ . These orthogonal projections effectively serve as implicit membership labels, identifying the subspace to which each token representation belongs.

**Sparse Rate Reduction.** Note that the rate reduction objective (4.2.1) is invariant to arbitrary joint rotations of the representations and subspaces. In particular, optimizing the rate reduction objective may not naturally lead to axis-aligned (i.e., *sparse*) representations. For instance, consider the three sets of learned representations in Figure 4.12. The coding rate reduction increases from (a) to (b), but because it is invariant under rotations, remains the same from (b) to (c). Therefore, we would like to transform the representations (and their supporting subspaces) so that the representations  $\mathbf{Z}$  eventually become sparse<sup>14</sup> with respect to the standard coordinates of the resulting representation space as in Figure 4.12(c). Therefore, to ensure the final representations are amenable to more compact coding, we would like to transform the representations (and their supporting subspaces) so that they become *sparse* with respect to the standard coordinates of the resulting representation space.<sup>15</sup> Computationally, we may combine the above two goals into a unified objective for optimization:

$$\max_{f \in \mathcal{F}} [\Delta R_\epsilon(\mathbf{Z} \mid \mathbf{U}_{[K]}) - \lambda \|\mathbf{Z}\|_0] \quad \text{s.t. } \mathbf{Z} = f(\mathbf{X}), \quad (4.2.4)$$

where  $\mathcal{F}$  denotes a general function class and the  $\ell_0$  norm  $\|\mathbf{Z}\|_0$  promotes the sparsity of the final token representations  $\mathbf{Z} = f(\mathbf{X})$ .

<sup>14</sup>Concretely, having few nonzero entries.

<sup>15</sup>That is, having the fewest nonzero entries.

In practice, the  $\ell_0$  norm is often relaxed to the  $\ell_1$  norm to improve computational traceability and enable convex optimization techniques [WM22]. Motivated by this, we relax Problem (4.2.4) accordingly, leading to a formulation that remains faithful to the original sparsity objective while being more amenable to efficient algorithms as follow:

$$\max_{f \in \mathcal{F}} [\Delta R_\epsilon(\mathbf{Z} | \mathbf{U}_{[K]}) - \lambda \|\mathbf{Z}\|_1] \quad \text{s.t. } \mathbf{Z} = f(\mathbf{X}), \quad (4.2.5)$$

With a slight abuse of terminology, we often refer to this objective function also as the *sparse rate reduction*.

**White-Box Network Architecture via Unrolled Optimization.** Although easy to state, each term in the above objective is computationally challenging to optimize [WM22]. Hence it is natural to adopt an approximation approach that realizes the global transformation  $f$  to optimize (4.2.4) through a concatenation of multiple, say  $L$ , simple *incremental and local* operations  $f^\ell$  that push the representation distribution towards the desired parsimonious model distribution:

$$f: \mathbf{X} = \mathbf{Z}^0 \xrightarrow{f^0} \mathbf{Z}^1 \rightarrow \dots \rightarrow \mathbf{Z}^\ell \xrightarrow{f^\ell} \mathbf{Z}^{\ell+1} \rightarrow \dots \xrightarrow{f^{L-1}} \mathbf{Z}^L = \mathbf{Z}, \quad (4.2.6)$$

where  $f^0: \mathbb{R}^D \rightarrow \mathbb{R}^d$  is the pre-processing mapping that transforms each input token  $\mathbf{x}_i \in \mathbb{R}^D$  to the initial token representations  $\mathbf{z}_i^1 \in \mathbb{R}^d$ . Each incremental *forward mapping*  $\mathbf{Z}^{\ell+1} = f^\ell(\mathbf{Z}^\ell)$ , or a “layer”, transforms the token distribution to *optimize* the above sparse rate reduction objective (4.2.4), conditioned on the distribution of its input  $\mathbf{Z}^\ell$ .

*Remark 4.5.* In contrast to other unrolled optimization approaches such as the ReduNet (see Section 4.1), we *explicitly model* the distribution of  $\mathbf{Z}^\ell$  at each layer, say as a mixture of linear subspaces or sparsely generated from a dictionary. The model parameters are learned from data (say via *backward propagation* with end-to-end training). This separation between forward “optimization” and backward “learning” clarifies the mathematical role of each layer as an operator that transforms the distribution of its input, whereas the input distribution is in turn modeled (and subsequently learned) by the parameters of the layer.

Now, we show how to derive these incremental and local operations through an unrolled optimization perspective to solve Problem (4.2.5). Once we decide on using an incremental approach to optimizing Problem (4.2.5), there are a variety of possible choices to achieve the optimization. Given a model for  $\mathbf{Z}^\ell$ , say a mixture of subspaces  $\mathbf{U}_{[K]}$ , we opt for a two-step *alternating minimization* method with a strong conceptual basis. First, we *compress* the tokens  $\mathbf{Z}^\ell$  via a gradient descent to minimize the coding rate term  $R_\epsilon^c(\mathbf{Z} | \mathbf{U}_{[K]}^\ell)$ . Specifically, we take a gradient step on  $R_\epsilon^c$  with a learning rate  $\kappa$  as follows:

$$\mathbf{Z}^{\ell+1/2} = \mathbf{Z}^\ell - \kappa \nabla_{\mathbf{Z}} R_\epsilon^c(\mathbf{Z} | \mathbf{U}_{[K]}^\ell). \quad (4.2.7)$$

Next, we *sparsify* the compressed tokens, generating  $\mathbf{Z}^{\ell+1}$  via a suitably-relaxed proximal gradient step to minimize the remaining term  $\lambda \|\mathbf{Z}\|_1 - R_\epsilon(\mathbf{Z})$ . As we will argue in detail later, we can find such a  $\mathbf{Z}^{\ell+1}$  by solving a sparse presentation problem with respect to a dictionary  $\mathbf{D}^\ell$ :

$$\mathbf{Z}^{\ell+1} = \arg \min_{\mathbf{Z}} \left\{ \lambda \|\mathbf{Z}\|_1 + \frac{1}{2} \|\mathbf{Z}^{\ell+1/2} - \mathbf{D}^\ell \mathbf{Z}\|_F^2 \right\}. \quad (4.2.8)$$

In the following, we provide technical details for each of the two steps above and derive efficient updates for their implementation.

**Self-Attention as Gradient Descent on Coding Rate of Token Representations.** For the first step (4.2.7), the gradient of the coding rate  $\nabla_{\mathbf{Z}} R_\epsilon^c$  is costly to compute, as it involves  $K$  separate matrix inverses, one for each of the  $K$  subspaces with basis  $\mathbf{U}_k^\ell$ :

$$\nabla_{\mathbf{Z}} R_\epsilon^c(\mathbf{Z} | \mathbf{U}_{[K]}) = \frac{p}{N\epsilon^2} \sum_{k=1}^K \mathbf{U}_k \mathbf{U}_k^\top \mathbf{Z} \left( \mathbf{I} + \frac{p}{N\epsilon^2} (\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z}) \right)^{-1}. \quad (4.2.9)$$

Now, we demonstrate that this gradient can be naturally approximated using a so-called multi-head subspace self-attention (MSSA) operator, which has a similar functional form to the multi-head self-attention operator

[VSP+17] with  $K$  heads (i.e., one for each subspace, coming from each matrix inverse). Here, we approximate the gradient (4.2.9) using the first-order Neumann series (see Exercise 4.2):

$$\begin{aligned}\nabla_{\mathbf{Z}} R_{\epsilon}^c(\mathbf{Z} \mid \mathbf{U}_{[K]}) &\approx \frac{p}{N\epsilon^2} \sum_{k=1}^K \mathbf{U}_k \mathbf{U}_k^\top \mathbf{Z} \left( \mathbf{I} - \frac{p}{N\epsilon^2} (\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z}) \right) \\ &= \frac{p}{N\epsilon^2} \left( \sum_{k=1}^K \mathbf{U}_k \mathbf{U}_k^\top \right) \mathbf{Z} - \left( \frac{p}{N\epsilon^2} \right)^2 \sum_{k=1}^K \mathbf{U}_k (\mathbf{U}_k^\top \mathbf{Z}) (\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z}).\end{aligned}\quad (4.2.10)$$

In this approximation, we compute the similarity between projected token representations  $\{\mathbf{U}_k^\top \mathbf{z}_i\}$  through an auto-correlation among the projected features as  $(\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z})$  and convert it to a distribution of membership with a softmax, namely softmax  $(\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z})$ . Suppose that a union of subspaces  $\mathbf{U}_{[K]}$  spans the whole space. Then, we have  $\sum_{k=1}^K \mathbf{U}_k \mathbf{U}_k^\top = \mathbf{I}$ . Hence, (4.2.10) becomes

$$\nabla_{\mathbf{Z}} R_{\epsilon}^c(\mathbf{Z} \mid \mathbf{U}_{[K]}) \approx \frac{p}{N\epsilon^2} \mathbf{Z} - \left( \frac{p}{N\epsilon^2} \right)^2 \text{MSSA} \left( \mathbf{Z}^\ell \mid \mathbf{U}_{[K]}^\ell \right), \quad (4.2.11)$$

where MSSA is defined through an SSA operator as follows:

$$\text{SSA}(\mathbf{Z} \mid \mathbf{U}_k) \doteq (\mathbf{U}_k^\top \mathbf{Z}) \text{softmax}((\mathbf{U}_k^\top \mathbf{Z})^\top (\mathbf{U}_k^\top \mathbf{Z})), \quad \forall k \in [K], \quad (4.2.12)$$

$$\text{MSSA}(\mathbf{Z} \mid \mathbf{U}_{[K]}) \doteq \frac{p}{N\epsilon^2} [\mathbf{U}_1, \dots, \mathbf{U}_K] \begin{bmatrix} \text{SSA}(\mathbf{Z} \mid \mathbf{U}_1) \\ \vdots \\ \text{SSA}(\mathbf{Z} \mid \mathbf{U}_K) \end{bmatrix}. \quad (4.2.13)$$

Substituting (4.2.11) into (4.2.7) yields that it can naturally be approximated by

$$\mathbf{Z}^{\ell+1/2} = \left( 1 - \frac{\kappa p}{N\epsilon^2} \right) \mathbf{Z}^\ell + \frac{\kappa p}{N\epsilon^2} \text{MSSA} \left( \mathbf{Z}^\ell \mid \mathbf{U}_{[K]}^\ell \right). \quad (4.2.14)$$

*Remark 4.6.* The SSA operator in (4.2.12) resembles the *attention operator* in a typical transformer [VSP+17], except that here the linear operators of value, key, and query are all set to be *the same* as the subspace basis, i.e.,  $\mathbf{V}_k = \mathbf{K}_k = \mathbf{Q}_k = \mathbf{U}_k^*$ . Hence, we name  $\text{SSA}(\cdot \mid \mathbf{U}_k) : \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^{p \times n}$  the **Subspace Self-Attention (SSA)** operator. Then, the whole MSSA operator in (4.2.13), formally defined as  $\text{MSSA}(\cdot \mid \mathbf{U}_{[K]}) : \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^{d \times n}$  and called the **Multi-Head Subspace Self-Attention (MSSA)** operator, aggregates the attention head outputs by averaging using model-dependent weights, similar in concept to the popular multi-head self-attention operator in existing transformer networks. The overall gradient step (4.2.14) resembles the multi-head self-attention implemented with a skip connection in transformers.

**MLP as Proximal Gradient Descent for Sparse Coding of Token Representations.** For the second step of alternating minimization, we need to minimize  $\lambda \|\mathbf{Z}\|_1 - R_\epsilon(\mathbf{Z})$ . Note that the gradient  $\nabla R_\epsilon(\mathbf{Z})$  involves a matrix inverse, and thus naive proximal gradient (see Section A.1.3) to optimize this problem becomes intractable on large-scale problems. We therefore take a different, simplifying approach to trading off between representational diversity and sparsification: we posit a (complete) incoherent or orthogonal dictionary  $\mathbf{D}^\ell \in \mathbb{R}^{d \times d}$ , and ask to sparsify the intermediate iterates  $\mathbf{Z}^{\ell+1/2}$  with respect to  $\mathbf{D}^\ell$ . That is,  $\mathbf{Z}^{\ell+1/2} \approx \mathbf{D}^\ell \mathbf{Z}^{\ell+1}$  where  $\mathbf{Z}^{\ell+1}$  is more sparse; that is, it is a *sparse encoding* of  $\mathbf{Z}^{\ell+1/2}$ . The dictionary  $\mathbf{D}^\ell$  is used to sparsify all tokens simultaneously. By the incoherence assumption, we have  $(\mathbf{D}^\ell)^\top (\mathbf{D}^\ell) \approx \mathbf{I}$ . Thus from (4.2.2) we have

$$R_\epsilon(\mathbf{Z}^{\ell+1/2}) \approx R_\epsilon(\mathbf{D}^\ell \mathbf{Z}^{\ell+1}) \approx R_\epsilon(\mathbf{Z}^{\ell+1}). \quad (4.2.15)$$

To solve  $\lambda \|\mathbf{Z}\|_1 - R_\epsilon(\mathbf{Z})$ , we optimize the following problem

$$\mathbf{Z}^{\ell+1} \approx \arg \min_{\mathbf{Z}} \|\mathbf{Z}\|_1 \quad \text{subject to} \quad \mathbf{Z}^{\ell+1/2} = \mathbf{D}^\ell \mathbf{Z}.$$

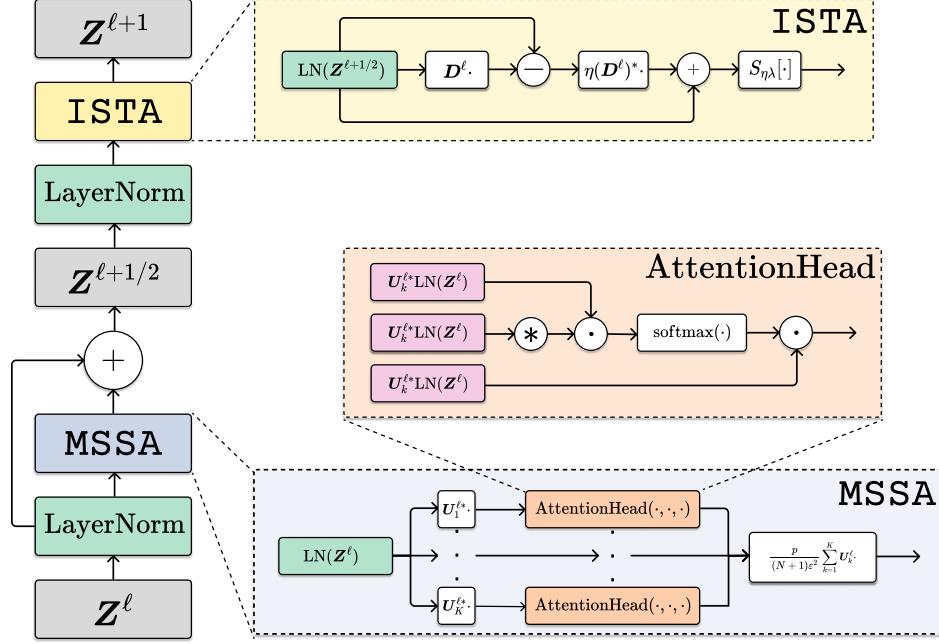


Figure 4.13: **One layer of the CRATE encoder architecture.** The full architecture is simply a concatenation of such layers, with some initial tokenizer, pre-processing head, and final task-specific head (i.e., a classification head).

The above sparse representation program is usually solved by relaxing it to an unconstrained convex program, known as LASSO [WM22]:

$$\mathbf{Z}^{\ell+1} \approx \arg \min_{\mathbf{Z}} \left[ \lambda \|\mathbf{Z}\|_1 + \frac{1}{2} \|\mathbf{Z}^{\ell+1/2} - \mathbf{D}^\ell \mathbf{Z}\|_F^2 \right]. \quad (4.2.16)$$

In our implementation, we also add a non-negative constraint to  $\mathbf{Z}^{\ell+1}$ , and solve the corresponding non-negative LASSO:

$$\mathbf{Z}^{\ell+1} \approx \arg \min_{\mathbf{Z} \geq 0} \left[ \lambda \|\mathbf{Z}\|_1 + \frac{1}{2} \|\mathbf{Z}^{\ell+1/2} - \mathbf{D}^\ell \mathbf{Z}\|_F^2 \right]. \quad (4.2.17)$$

Then, we incrementally optimize Equation (4.2.17) by performing an unrolled *proximal gradient descent* step, known as an ISTA step [BT09], to give the update:

$$\mathbf{Z}^{\ell+1} = \text{ISTA}(\mathbf{Z}^{\ell+1/2} | \mathbf{D}^\ell), \quad (4.2.18)$$

$$\text{where } \text{ISTA}(\mathbf{Z} | \mathbf{D}) \doteq \text{ReLU}(\mathbf{Z} - \eta \mathbf{D}^\top (\mathbf{DZ} - \mathbf{Z}) - \eta \lambda \mathbf{1}). \quad (4.2.19)$$

## 4.2.2 Overall White-Box Transformer Architecture: CRATE

We now design a white-box transformer architecture, named the Coding RATE Transformer (CRATE), by unrolling the above updates. By combining the above two steps (4.2.14) and (4.2.18):

1. Local compression of tokens within a sample towards a mixture-of-subspace structure, leading to the multi-head subspace self-attention block – MSSA;
2. Global sparsification of token sets across all samples through sparse coding, leading to the sparsification block – ISTA;

we can get the following rate-reduction-based transformer layer, illustrated in Figure 4.13,

$$\mathbf{Z}^{\ell+1/2} \doteq \mathbf{Z}^\ell + \text{MSSA}(\mathbf{Z}^\ell | \mathbf{U}_{[K]}^\ell), \quad \mathbf{Z}^{\ell+1} \doteq \text{ISTA}(\mathbf{Z}^{\ell+1/2} | \mathbf{D}^\ell). \quad (4.2.20)$$

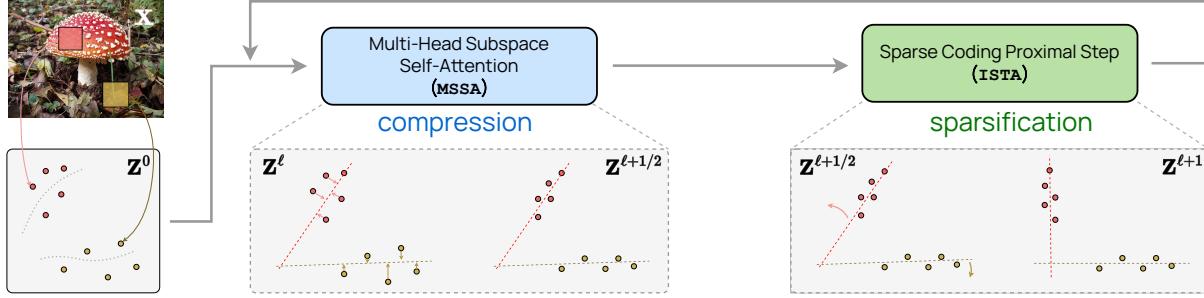


Figure 4.14: The ‘main loop’ of the crate white-box deep network design. After encoding input data as a sequence of tokens  $Z^0$ , CRATE constructs a deep network that transforms the data to a canonical configuration of low-dimensional subspaces by successive **compression** against a local model for the distribution, generating  $Z^{l+1/2}$ , and **sparsification** against a global dictionary, generating  $Z^{l+1}$ . Repeatedly stacking these blocks and training the model parameters via backpropagation yields a powerful and interpretable representation of the data.

Composing multiple such layers following the incremental construction of our representation in (4.2.6), we obtain a white-box transformer architecture that transforms the data tokens towards a compact and sparse union of incoherent subspaces, where  $f^{\text{pre}} : \mathbb{R}^{D \times N} \rightarrow \mathbb{R}^{d \times N}$  is the pre-processing mapping that transforms the input tokens  $X \in \mathbb{R}^{D \times N}$  to first-layer representations  $Z^1 \in \mathbb{R}^{d \times N}$ . An overall flow of this architecture was shown in Figure 4.14.

*Remark 4.7 (The roles of the forward pass and backward propagation).* In contrast to other unrolled optimization approaches such as the ReduNet [CYY+22], we *explicitly model* the distribution of each  $Z^\ell$  and  $Z^{l+1/2}$  at each layer, either by a mixture of linear subspaces or sparsely generated from a dictionary. We introduced the interpretation that at each layer  $\ell$ , the learned bases for the subspaces  $U_{[K]}^\ell$  and the learned dictionaries  $D^\ell$  together serve as a *codebook* or *analysis filter* that encodes and transforms the intermediate representations at each layer  $\ell$ . Since the input distribution to layer  $\ell$  is first modeled by  $U_{[K]}^\ell$  then transformed by  $D^\ell$ , the input distribution to each layer is different, and so we require a separate code book at each layer to obtain the most parsimonious encoding. Parameters of these codebooks (i.e., the subspace bases and the dictionaries), heretofore assumed as being perfectly known, are actually learned from data (say via *backward propagation* within end-to-end training).

Hence, our methodology features a clear conceptual separation between forward “optimization” and backward “learning” for the so-derived white-box deep neural network. Namely, in its forward pass, we interpret each layer as an operator which, conditioned on a learned model (i.e., a codebook) for the distribution of its input, transforms this distribution towards a more parsimonious representation. In its backward propagation, the codebook of this model, for the distribution of the input to each layer, is updated to better fit a certain (supervised) input-output relationship, as illustrated in Figure 4.15. This conceptual interpretation implies a certain agnosticism of the model representations towards the particular task and loss; in particular, many types of tasks and losses will ensure that the models at each layer are fit, which ensures that the model produces parsimonious representations.

We now present the empirical performance of the proposed networks CRATE by measuring their top-1 classification accuracy on ImageNet-1K as well as transfer learning performance on several widely used downstream datasets. We summarize the results in Table 4.1. The transfer learning methodology is to fine-tune using cross-entropy loss initializing from the pre-trained networks. As the designed white-box transformer architecture leverages parameter sharing in both the attention block (MSSA) and the nonlinearity block (ISTA), the CRATE-Base model (22.80 million) has a similar number of parameters to the ViT-Small (22.05 million) [DBK+21], and less than 30% of the parameters of an identically configured ViT-Base (86.54 million). From Table 4.1, we find that with a similar number of model parameters, our proposed network achieves similar ImageNet-1K and transfer learning performance as ViT, while having a simple and principled design. Moreover, with the same set of training hyperparameters, we observe promising scaling behavior in CRATE—we consistently improve the performance by scaling up the model size. To summarize, CRATE

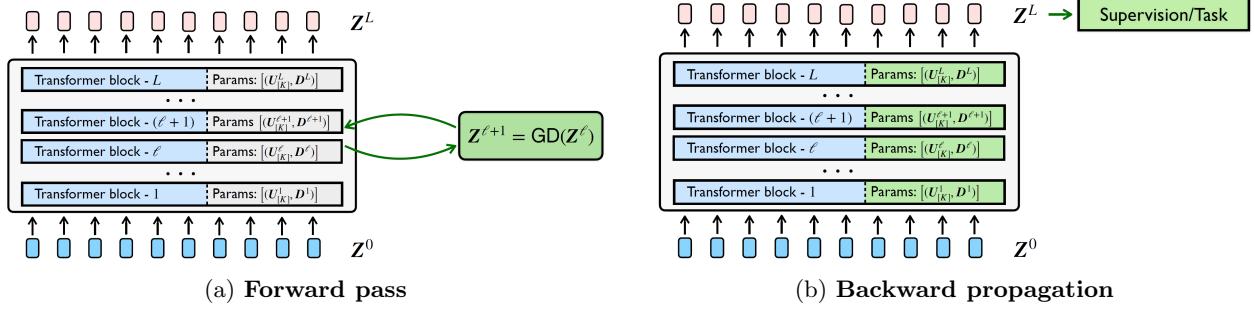


Figure 4.15: **The roles of forward pass and backward propagation in deep networks.** (a) Given fixed subspaces and dictionaries  $\{(U_{[K]}^\ell, D^\ell)\}_{\ell=1}^L$ , each layer performs compression and sparsification on representations in the forward pass; (b) Backpropagation learn subspaces and dictionaries  $\{(U_{[K]}^\ell, D^\ell)\}_{\ell=1}^L$  from training data.

Table 4.1: Top-1 classification accuracy of CRATE on various datasets with different model scales when pre-trained on ImageNet-1K. For ImageNet-1K/ImageNet-1K ReaL, we directly evaluate the top-1 accuracy. For other datasets, we use models that are pre-trained on ImageNet as initialization and the evaluate the transfer learning performance via fine-tuning.

Model	CRATE-T	CRATE-S	CRATE-B	CRATE-L	ViT-T	ViT-S
# parameters	6.09M	13.12M	22.80M	77.64M	5.72M	22.05M
ImageNet-1K	66.7	69.2	70.8	71.3	71.5	72.4
ImageNet-1K ReaL	74.0	76.0	76.5	77.4	78.3	78.4
CIFAR10	95.5	96.0	96.8	97.2	96.6	97.2
CIFAR100	78.9	81.0	82.7	83.6	81.8	83.2
Oxford Flowers-102	84.6	87.1	88.7	88.3	85.1	88.5
Oxford-IIIT-Pets	81.4	84.9	85.3	87.4	88.5	88.6

achieve promising performance on real-world large-scale datasets by directly implementing our principled architecture. We will provide more details of the implementation and analysis of the experimental results on image classification in the final application Chapter 7.

## 4.3 Variants of Deep Architectures by Design

So far, we wish that we have provided compelling evidence that the role of (popular) deep networks is to realize certain optimization algorithms for minimizing the coding rate (or maximizing the information gain) of the learned representations. However, readers who are familiar with optimization methods might have noticed that the above architectures (the ReduNet or the CRATE) correspond to rather basic optimization techniques. They may have plenty of room for improvement in efficiency or effectiveness. Moreover, if we believe the proposed theoretical framework for interpreting deep networks is correct, it should not only help explain existing architectures, it should guide us develop more efficient and effective architectures. In this section, we show this could be the case: the resulting new architectures are not only fully interpretable but also with guaranteed correctness and improved efficiency.

### 4.3.1 Attention-Only Transformer Architecture

In this subsection, we propose a minimalistic transformer architecture consisting of interpretable layers based on the MSSA operator. To derive a fully interpretable transformer architecture with only necessary components, we contend that the goal of representation learning is to compress a set of noisy initial token representations towards a mixture of low-dimensional subspaces. Here, we assume that the initial token representations  $Z^{(1)}$  are sampled from a mixture of low-rank Gaussians perturbed by noise as follows:

**Definition 4.1.** Let  $C_1, \dots, C_K$  be a partition of the index set  $[N]$  and  $\mathbf{U}_k \in \mathcal{O}^{d \times p_k}$  denote the orthonormal basis of the  $k$ -th subspace for each  $k \in [K]$ . We say that the token representations  $\{\mathbf{z}_i\}_{i=1}^N \subseteq \mathbb{R}^d$  are sampled from a mixture of noisy low-rank Gaussian distributions if for each  $k \in [K]$ ,

$$\mathbf{z}_i = \underbrace{\mathbf{U}_k \mathbf{a}_i}_{\text{signal}} + \underbrace{\sum_{j \neq k}^K \mathbf{U}_j \mathbf{e}_{i,j}}_{\text{noise}}, \quad \forall i \in C_k, \quad (4.3.1)$$

where  $\mathbf{a}_i \stackrel{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_{p_k})$  and  $\mathbf{e}_{i,j} \stackrel{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, \delta^2 \mathbf{I}_{p_j})$  for all  $i \in C_k$  and  $k \in [K]$ ,  $\{\mathbf{a}_i\}$  and  $\{\mathbf{e}_{i,j}\}$  are respectively mutually independent, and  $\{\mathbf{a}_i\}$  is independent of  $\{\mathbf{e}_{i,j}\}$ .

This model serves as an idealized framework for approximating token representations in real-world pre-trained LLMs. It assumes that the token representations are sampled from a mixture of multiple low-rank Gaussian distributions with noise. Under this model, the goal of representation learning is to compress a set of noisy initial token presentations into the corresponding subspace. In addition, this model aligns well with two well-established hypotheses about the structure of token representations in pretrained large language models: the “linear representation hypothesis” [JRR+24; PCV24] and the “superposition hypothesis” [EHO+22a; YCO+21].

*Remark 4.8.* The linear representation hypothesis posits that token representations in LLMs lie in low-dimensional linear subspaces that encode semantic features. Similarly, the superposition hypothesis suggests that these representations can be approximately expressed as a sparse linear combination of these feature vectors. In Definition 4.1, each basis  $\mathbf{U}_k$  of the subspaces can be interpreted as a set of semantic features, where each feature corresponds to a specific aspect of the token’s meaning. Token representations are then approximately expressed as sparse linear combinations of these subspace bases, capturing the essential semantic components of the token while ignoring irrelevant dimensions.

**Denoising Operator for Token Representations.** Now, we show that the MSSA operator (see (4.2.13)) can incrementally denoise token representations generated from the above model. Specifically, we consider for each  $\ell = 1, \dots, L$ ,

$$\mathbf{Z}^{(\ell+1)} = \mathbf{Z}^{(\ell)} + \eta \sum_{k=1}^K \mathbf{U}_k \mathbf{U}_k^T \mathbf{Z}^{(\ell)} \varphi \left( \mathbf{Z}^{(\ell)T} \mathbf{U}_k \mathbf{U}_k^T \mathbf{Z}^{(\ell)} \right), \quad (4.3.2)$$

where  $\{\mathbf{U}_k\}_{k=1}^K$  is defined in Definition 4.1,  $\eta > 0$  is the step size, and  $\varphi$  is an element-wise operator, such as soft-max, ReLU, or other functions. To simplify our development, we assume that the subspaces in Definition 4.1 are orthogonal to each other, i.e.,  $\mathbf{U}_k^T \mathbf{U}_j = \mathbf{0}$  for all  $k \neq j$ . Note that this assumption is not restrictive, as in high-dimensional spaces, random low-dimensional subspaces are incoherent to each other with high probability, i.e.,  $\mathbf{U}_k^T \mathbf{U}_j \approx \mathbf{0}$  [WM21].<sup>16</sup>

Now, let the columns of  $\mathbf{Z}_k^{(\ell)}$  denotes the token representations from the  $k$ -th subspace at the  $\ell$ -th layer. To quantify the denoising capability, we define the signal-to-noise ratio (SNR) for each block of the token representations at the  $\ell$ -th layer as follows:

$$\text{SNR}(\mathbf{Z}_k^{(\ell)}) \doteq \frac{\|\mathbf{U}_k \mathbf{U}_k^T \mathbf{Z}_k^{(\ell)}\|_F}{\|(\mathbf{I} - \mathbf{U}_k \mathbf{U}_k^T) \mathbf{Z}_k^{(\ell)}\|_F}, \quad \forall k \in [K]. \quad (4.3.3)$$

To simplify our analysis, we assume that  $p = p_1 = \dots = p_K$ ,  $N_1 = \dots = N_K = N/K$ , and

$$[\mathbf{U}_1 \quad \dots \quad \mathbf{U}_K] \in \mathcal{O}^{d \times Kp}. \quad (4.3.4)$$

With the above setup, we now characterize the denoising performance of the MSSA operator.

---

<sup>16</sup>One may straightforwardly generalize our results to non-orthogonal subspaces, with slightly more sophisticated analysis.

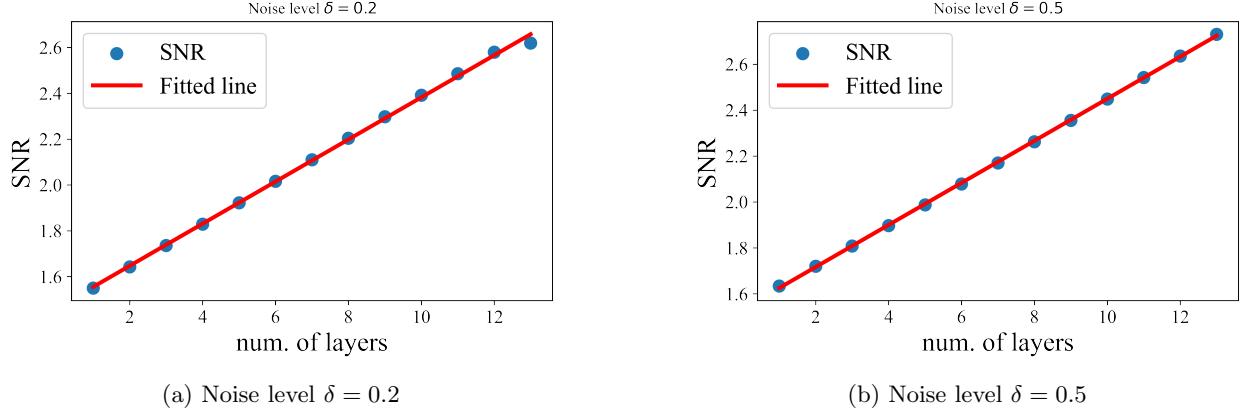


Figure 4.16: **Denoising performance of the attention-only transformer.** Here, we sample initial token representations from a mixture of low-rank Gaussians in Definition 4.1. Then, we apply (4.3.2) to update token representations and report the SNR at each layer.

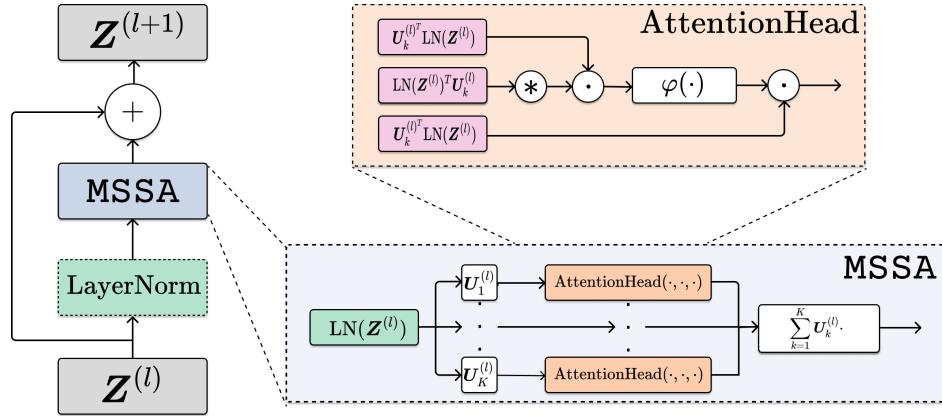


Figure 4.17: **Details of the attention-only transformer architecture.** Each layer consists of the MSSA operator and a skip connection. In addition, LayeNnorm is included only for language tasks. In practice, backpropagation is applied to train the model parameters using training samples.

**Theorem 4.1.** Let  $Z^{(1)}$  be defined in Definition 4.1 and  $\varphi(\cdot)$  in Eq. (4.3.2) be  $\varphi(x) = h(\sigma(x))$ , where  $\sigma : \mathbb{R}^N \rightarrow \mathbb{R}^N$  is the soft-max function and  $h : \mathbb{R}^N \rightarrow \mathbb{R}^N$  is an element-wise thresholding function with  $h(x) = \tau \mathbb{I}\{x > \tau\}$  for each  $i \in [N]$ . Suppose that  $p \gtrsim \log N$ ,  $\delta \lesssim \sqrt{\log N}/\sqrt{p}$ , and

$$\tau \in \left( \frac{1}{2}, \frac{1}{1 + N \exp(-9p/32)} \right].$$

For sufficiently large  $N$ , it holds with probability at least  $1 - KLN^{-\Omega(1)}$  that for each  $\ell \in [L]$ ,

$$\text{SNR}(Z_k^{(\ell+1)}) = (1 + \eta\tau)\text{SNR}(Z_k^{(\ell)}), \quad \forall k \in [K]. \quad (4.3.5)$$

This theorem demonstrates that when the initial token representations are sampled from a mixture of low-rank Gaussian distributions with a noise level  $O(\sqrt{\log N}/\sqrt{p})$ , we show that each layer of the proposed transformer denoises token representations at a linear rate. This indicates the MSSA operator's efficiency in reducing noise across layers. Notably, our theoretical results are well-supported by experimental observations in Figure 4.16. This theorem provides a theoretical foundation for the practical denoising capability of the transformer architecture derived by unrolling (4.3.2).

*Remark 4.9.* Under this model, the goal of representation learning is to compress a set of noisy initial token presentations into the corresponding subspace. However, we should point out that in real-world applications, where token representations exhibit more complicated structures, the goal of representation learning is to find a compact and structured representation by compressing token sets.

**Attention-Only Transformer.** Now, we formally propose an attention-only transformer architecture. Specifically, by unrolling the iterative optimization steps (4.3.2) as layers of a deep network, we construct a transformer architecture in Figure 4.17. Each layer of the proposed architecture only consists of the MSSA operator and a skip connection. For language tasks, we additionally incorporate LayerNorm before the MSSA operator to improve performance. The complete architecture is built by stacking such layers, along with essential task-specific pre-processing and post-processing steps, such as positional encoding, token embedding, and final task-specific head to adapt to different applications.

Generally speaking, the standard decoder-only transformer architecture is composed of the following key components [VSP+17]: (1) positional encoding, (2) multi-head QKV self-attention mechanisms, (3) feed-forward MLP networks, (4) layer normalization, and (5) residual connections. In contrast, our proposed transformer architecture adopts a streamlined design by incorporating several key simplifications. Specifically, it employs shared-QKV subspace self-attention mechanisms, excludes MLP layers, and reduces the frequency of LayerNorm.

### 4.3.2 Linear-Time Attention: Token Statistics Transformer

In this subsection, we propose a new transformer attention operator whose computational complexity scales linearly with the number of tokens based on the coding rate reduction objective. Specifically, we derive a novel variational form of the MCR<sup>2</sup> objective and show that the architecture that results from unrolled gradient descent of this variational objective leads to a new attention module called Token Statistics Self-Attention (TSSA). TSSA has *linear computational and memory complexity* and radically departs from the typical attention architecture that computes pairwise similarities between tokens. Recall from (3.4.14) that  $\boldsymbol{\Pi} = [\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_K] \in \mathbb{R}^{N \times K}$  denotes a stochastic “group assignment” matrix (i.e.,  $\boldsymbol{\Pi}\mathbf{1} = \mathbf{1}$  and  $\Pi_{ik} \geq 0, \forall (i, k) \in [N] \times [K]$ ), where  $\Pi_{ik}$  denotes the probability of assigning the  $i$ -th token to the  $k$ -th group.

**A New Variational Form for Coding Rates.** To begin, we consider a general form of MCR<sup>2</sup>-like objectives based on concave functions of the spectrum of a matrix. Namely, for a given PSD matrix  $\mathbf{M} \in \text{PSD}(d)$  and any scalar  $c \geq 0$  we have that  $\log \det(\mathbf{I} + c\mathbf{M}) = \sum_{i=1}^d \log(1 + c\lambda_i(\mathbf{M}))$ , where  $\lambda_i(\mathbf{M})$  is the  $i$ -th largest eigenvalue of  $\mathbf{M}$ . Further, note that  $\log(1 + c\sigma)$  is a concave non-decreasing function of  $\sigma$ . Thus, we describe our results in terms of a more general form of MCR<sup>2</sup> based on general spectral functions of PSD matrices of the form  $F(\mathbf{M}) = \sum_{i=1}^d f(\lambda_i(\mathbf{M}))$ , where  $f$  is concave and non-decreasing. In particular, recall from our above discussion that the attention mechanism arises from unrolling the compression component of MCR<sup>2</sup>, so we consider a more general MCR<sup>2</sup>-style compression function:

$$R_{c,f}(\mathbf{Z}, \boldsymbol{\Pi}) \doteq \frac{1}{2} \sum_{k=1}^K \frac{N_k}{N} F\left(\frac{1}{N_k} \mathbf{Z} \text{Diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top\right). \quad (4.3.6)$$

For the above objective, we now note the following result:

**Theorem 4.2.** Let  $f: [0, \infty) \rightarrow \mathbb{R}$  be non-decreasing, concave, and obey  $f(0) = 0$ , and let  $F: \text{PSD}(d) \rightarrow \mathbb{R}$  have the form  $F(\mathbf{M}) = \sum_{i=1}^d f(\lambda_i(\mathbf{M}))$ . Then for each  $\mathbf{M} \in \text{PSD}(d)$  and  $\mathbf{Q} \in \text{O}(d)$ , we have

$$F(\mathbf{M}) \leq \sum_{i=1}^d f((\mathbf{Q}^\top \mathbf{M} \mathbf{Q})_{ii}). \quad (4.3.7)$$

Further, the inequality in (4.3.7) is achieved with equality for any  $\mathbf{Q}$  which diagonalizes  $\mathbf{M}$ , and if  $f$  is strictly concave then the inequality in (4.3.7) is achieved with equality if and only if  $\mathbf{Q}$  diagonalizes  $\mathbf{M}$ .

Using the above result, we can replace (4.3.6) with an equivalent variational objective with form

$$R_{c,f}^{\text{var}}(\mathbf{Z}, \boldsymbol{\Pi} \mid \mathbf{U}_{[K]}) \doteq \frac{1}{2} \sum_{k=1}^K \frac{N_k}{N} \sum_{i=1}^d f\left(\frac{1}{N_k} (\mathbf{U}_k^\top \mathbf{Z} \text{Diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top \mathbf{U}_k)_{ii}\right), \quad (4.3.8)$$

where the equivalence is in the sense that for an optimal choice of  $\{\mathbf{U}_k \in \text{O}(d)\}_{k=1}^K$  matrices as described in Theorem 4.2 (i.e., orthogonal matrices which diagonalize each  $\mathbf{Z} \text{Diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top$ ) we will achieve a tight bound with  $R_{c,f}^{\text{var}}(\mathbf{Z}, \boldsymbol{\Pi} \mid \mathbf{U}_{[K]}) = R_{c,f}(\mathbf{Z}, \boldsymbol{\Pi})$ . Note that in general, achieving this bound would require selecting, for each sampled instance of  $\mathbf{Z}$ , a new optimal set of  $\mathbf{U}_k$  parameter matrices which diagonalize each  $\mathbf{Z} \text{Diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top$ , which is clearly impractical for network architecture. Instead, as an alternative viewpoint, rather than considering the data ( $\mathbf{Z}$ ) as fixed and trying to optimize the  $\mathbf{U}_k$  parameters to achieve the tight variational bound, we can instead take the algorithmic unrolling design principle described above and design an operator to perturb  $\mathbf{Z}$  to incrementally minimize  $R_{c,f}^{\text{var}}(\cdot \mid \mathbf{U}_{[K]})$ . To make this point explicit, each variational bound becomes tight when the eigenspaces of  $\mathbf{Z} \text{Diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top$  align with the columns of  $\mathbf{U}_k$ , so by rotating the appropriate columns of  $\mathbf{Z}$  (namely, those which correspond to large entries in  $\boldsymbol{\pi}_k$ ) to align with  $\mathbf{U}_k$  we can approach a tight variational bound. That is, instead of rotating  $\mathbf{U}_k$  to align with the data for each instance of  $\mathbf{Z}$ , we can instead rotate the token features in each  $\mathbf{Z}$  to align with  $\mathbf{U}_k$ .

Following this approach, we compute a gradient descent step on  $R_{c,f}^{\text{var}}$  w.r.t.  $\mathbf{Z}$ . To begin this computation, first let  $\boldsymbol{\pi} \in \mathbb{R}^N$  be any element-wise non-negative vector. Then we have

$$\nabla_{\mathbf{Z}} \frac{1}{2} \sum_{i=1}^d f((\mathbf{Z} \text{Diag}(\boldsymbol{\pi}) \mathbf{Z}^\top)_{ii}) = \text{Diag}(\nabla f[\mathbf{Z}^{\odot 2} \boldsymbol{\pi}]) \mathbf{Z} \text{Diag}(\boldsymbol{\pi}), \quad (4.3.9)$$

where  $\nabla f$  is the gradient of  $f$ , and (recall)  $\nabla f[\cdot]$  applies  $\nabla f$  to each element of the vector in the bracket. In particular, for  $f(x) = \log(1 + (d/\epsilon^2)x)$ ,  $\nabla f(x) = (d/\epsilon^2)(1 + (d/\epsilon^2)x)^{-1}$  is simply a non-linear activation. Also, (recall)  $N_k = \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle$ . Thus, the gradient of  $R_{c,f}^{\text{var}}$  w.r.t.  $\mathbf{Z}$  is:

$$\nabla_{\mathbf{Z}} R_{c,f}^{\text{var}}(\mathbf{Z}, \boldsymbol{\Pi} \mid \mathbf{U}_{[K]}) = \frac{1}{n} \sum_{k=1}^K \mathbf{U}_k \underbrace{\text{Diag}\left(\nabla f\left[(\mathbf{U}_k^\top \mathbf{Z})^{\odot 2} \frac{\boldsymbol{\pi}_k}{\langle \boldsymbol{\pi}_k, \mathbf{1} \rangle}\right]\right)}_{\doteq \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k)} \mathbf{U}_k^\top \mathbf{Z} \text{Diag}(\boldsymbol{\pi}_k). \quad (4.3.10)$$

(Note that the  $1/N$  constant arises from a  $(N_k/N) \cdot (1/N_k) = 1/N$  constant in each term of the sum.) If we now consider a gradient step w.r.t. the  $j$ -th token  $\mathbf{z}_j$ , we arrive at our proposed incremental compression operator, i.e., our surrogate for a *self attention* + residual operator:

$$\mathbf{z}_j^+ = \mathbf{z}_j - \tau \nabla_{\mathbf{z}_j} R_{c,f}^{\text{var}}(\mathbf{Z}, \boldsymbol{\Pi} \mid \mathbf{U}_{[K]}) = \mathbf{z}_j - \frac{\tau}{N} \sum_{k=1}^K \Pi_{jk} \mathbf{U}_k \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k) \mathbf{U}_k^\top \mathbf{z}_j \quad (4.3.11)$$

for each  $j \in [n]$ , where  $\tau > 0$  is a step size parameter for the incremental optimization. Then, we can construct a layer of TOST in Figure 4.18.

**Model interpretation.** Given the proposed attention operator in (4.3.11), first recall that the rows of  $\boldsymbol{\Pi}$  are non-negative and sum to 1, so our operator takes a weighted average of  $K$  “attention head”-esque operators and then adds a residual connection. Using that  $\sum_{k=1}^K \Pi_{jk} = 1$ , we can rewrite (4.3.11) as:

$$\mathbf{z}_j^+ = \sum_{k=1}^K \Pi_{jk} \left[ \mathbf{z}_j - \underbrace{\frac{\tau}{n} \mathbf{U}_k \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k) \mathbf{U}_k^\top \mathbf{z}_j}_{\text{action of one attention head}} \right]. \quad (4.3.12)$$

That is, we can view each attention head as first projecting the token features onto the basis  $\mathbf{U}_k$  via multiplying by  $\mathbf{U}_k^\top$ , multiplying by the diagonal matrix  $\mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k)$  (abbreviated as  $\mathbf{D}_k$ ), projecting back into the standard basis via multiplying by  $\mathbf{U}_k$ , and subtracting this from the original token features via the residual connection. The core aspect of our attention layer is the computation of  $\mathbf{D}_k$ . Namely,  $\Pi_{jk} \geq 0$ , so  $\boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle \in \mathbb{R}^N$  forms a probability distribution over which tokens belong to the  $k^{\text{th}}$  group. As a result,  $(\mathbf{U}_k^\top \mathbf{Z})^{\odot 2} \boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle$  estimates the second moment of  $\mathbf{U}_k^\top \mathbf{Z}$  under the distribution given by  $\boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle$ . Further, since  $f$  is a concave non-decreasing function,  $\nabla f(x)$  monotonically decreases towards

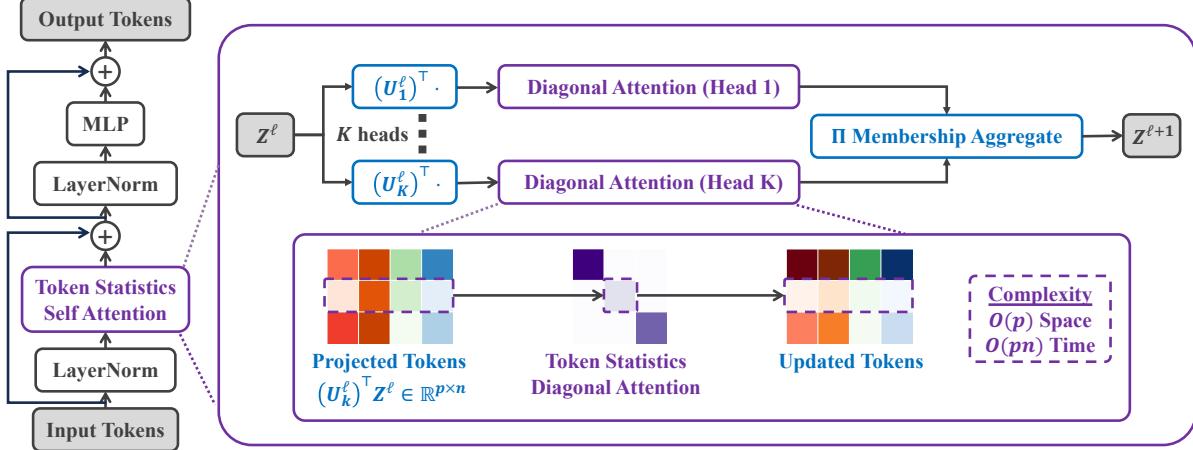


Figure 4.18: **One layer  $\ell$  of the proposed Token Statistics Transformer (ToST).** Notably, the self-attention of ToST transforms tokens  $Z^\ell$  efficiently to  $Z^{\ell+1}$ , via multiplying each row of the projected token by *only a scalar*. This leads to reduced complexity of the attention: it has  $O(p)$  space and  $O(pn)$  time complexity, where  $p$  is the dimension of the projected tokens of each head, and  $n$  is the number of tokens.

0 as  $x$  increases, so the entries of  $D_k$  (which have form  $\nabla f(x)$ ) achieve their maximum at  $x = 0$  and decay monotonically to 0 as  $x$  increases.

From this, we arrive at the core interpretation of our attention head + residual operators  $[I - (\tau/n)U_k D_k U_k^\top]$ . Namely, this operator does an approximate low-rank data-dependent projection, where directions which have a large amount of “power” after the projection  $U_k^\top Z$  (i.e., directions which have a large second moment  $(U_k^\top Z)^{\odot 2} \pi_k / \langle \pi_k, 1 \rangle$ ) are preserved, while directions which do not are suppressed. To see this, recall that the entries of  $D_k$  decrease monotonically to 0 as the second moment increases, so for directions with large second moments the attention + residual operator acts largely as the identity operator. Conversely, for directions with a small second moment, our operator subtracts a projection of the tokens along those directions, resulting in those directions being suppressed. Compared to the standard self-attention operator, our method clearly does not compute any pairwise similarities between tokens. Rather, the interactions between the tokens in  $Z$  impact the operator solely through their contribution to the second moment statistic used to construct the  $D_k$ ’s. Nevertheless, similar to the standard self-attention operator, our method still has a clear interpretation as performing a form of compression towards a data-dependent low-rank structure, in the sense that it performs an approximate low-rank projection, where the specific directions that are suppressed are those which are not strongly aligned with other tokens in the group.

**Practical Implementation Details.** Having introduced our proposed attention operator, we now discuss further practical considerations. First, until this point in the presentation, we have avoided discussion of how tokens are “grouped” into various attention heads via the  $\Pi$  matrix, but clearly a means of constructing  $\Pi$  is needed to implement our method. Additionally, our variational form in Theorem 4.2 requires the  $U$  matrices to be square and orthogonal, but one would ideally like to use smaller matrices (i.e., reduce the number of columns in  $U$ ) for efficiency as well as drop the orthogonality constraints.

In practice, we do not enforce the orthogonality constraints. To reduce the number of columns in the  $U$  matrices, we note that similar to CRATE [YBP+23], if we assume the features  $Z$  within group  $k$  are (approximately) clustered around a low-dimensional subspace — say of dimension  $p$  — then the within-group- $k$  covariance  $Z \text{Diag}(\pi_k) Z^\top$  is low-rank, where recall that [YCY+20] shows that the optimal geometry of  $Z$  will be for each group to be a low-rank subspace, orthogonal to the other groups. We can thus explicitly find a low-dimensional orthonormal basis for the image of this covariance, i.e., the linear span of the data in group  $k$ . If the dimension is  $p \leq d$ , the basis can be represented by a  $d \times p$  orthogonal matrix  $U_k \in O(d, p)$ . In this case, we can more efficiently upper-bound  $R_{c,f}$  using these low-rank orthogonal basis matrices. To show this, we use a more general version of Theorem 4.2 to yield the following corollary.

**Corollary 4.1.** Let  $f: [0, \infty) \rightarrow \mathbb{R}$  be non-decreasing, concave, and obey  $f(0) = 0$ , and let  $F: \text{PSD}(p) \rightarrow \mathbb{R}$  have the form  $F(\mathbf{M}) = \sum_{i=1}^p f(\lambda_i(\mathbf{M}))$ . Let  $\mathbf{Z}, \boldsymbol{\Pi}$  be fixed. Then, for all  $\mathbf{U}_1, \dots, \mathbf{U}_K \in \text{O}(d, p)$  such that  $\text{image}(\mathbf{Z} \text{ diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top) \subset \text{image}(\mathbf{U}_k)$  for all  $k \in [K]$ , we have

$$R_{c,f}(\mathbf{Z}, \boldsymbol{\Pi}) \leq R_{c,f}^{\text{var}}(\mathbf{Z}, \boldsymbol{\Pi} \mid \mathbf{U}_{[K]}), \quad (4.3.13)$$

where  $R_{c,f}^{\text{var}}$  is formally defined in (4.3.8). Equality holds if  $\mathbf{U}_k$  diagonalizes  $\mathbf{Z} \text{ diag}(\boldsymbol{\pi}_k) \mathbf{Z}^\top$  for each  $k \in [K]$ , and if  $f$  is strongly concave then this equality condition becomes an “if and only if.”

The final step to define our attention operator is to estimate the group membership  $\boldsymbol{\Pi}$ . For this we posit a simple model of how each feature  $\mathbf{z}_j$  deviates from its supporting subspace and then find the optimal subspace assignment. [YBP+23] show that if we independently model each  $\mathbf{z}_j$  as belonging to a low-dimensional Gaussian mixture model, where each Gaussian has a covariance matrix with identical spectrum and is supported on a subspace with orthonormal basis  $\mathbf{U}_k$ , plus independent Gaussian noise with covariance  $\eta\mathbf{I}$ , then the posterior probability that each token  $\mathbf{z}_j$  belongs to each subspace is given by the assignment matrix  $\boldsymbol{\Pi} = \boldsymbol{\Pi}(\mathbf{Z} \mid \mathbf{U}_{[K]})$  as follows:

$$\boldsymbol{\Pi} = \begin{bmatrix} \boldsymbol{\nu}(\mathbf{z}_1 \mid \mathbf{U}_{[K]})^\top \\ \vdots \\ \boldsymbol{\nu}(\mathbf{z}_n \mid \mathbf{U}_{[K]})^\top \end{bmatrix}, \quad \text{where } \boldsymbol{\nu}(\mathbf{z}_j \mid \mathbf{U}_{[K]}) \doteq \text{softmax} \left( \frac{1}{2\eta} \begin{bmatrix} \|\mathbf{U}_1^\top \mathbf{z}_j\|_2^2 \\ \vdots \\ \|\mathbf{U}_K^\top \mathbf{z}_j\|_2^2 \end{bmatrix} \right), \quad \forall j \in [n], \quad (4.3.14)$$

where  $\eta$  becomes a learnable temperature parameter. Thus, given an input feature  $\mathbf{Z}$ , we estimate  $\boldsymbol{\Pi}$  using (4.3.14) and then compute the attention operator. Combining the construction of  $\boldsymbol{\Pi}$  in (4.3.14) with (4.3.11), we obtain the *Token Statistics Self-Attention* operator:

$$\text{TSSA}(\mathbf{Z} \mid \mathbf{U}_{[K]}) \doteq -\frac{\tau}{n} \sum_{k=1}^K \mathbf{U}_k \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k) \mathbf{U}_k^\top \mathbf{Z} \text{ diag}(\boldsymbol{\pi}_k), \quad (4.3.15)$$

where  $\boldsymbol{\pi}_k$  are the columns of  $\boldsymbol{\Pi} = \boldsymbol{\Pi}(\mathbf{Z} \mid \mathbf{U}_{[K]})$  defined in (4.3.14) and  $\mathbf{D}$  is defined in (4.3.10).

## 4.4 Summary and Notes

The materials presented in this chapter are based on a series of recent works on this topic, including [CYY+22; WLP+24; WLY+25; WDL+25; YBP+23]. These contributions encompass both theoretical advances and practical methodologies for constructing interpretable deep networks through unrolled optimization. Many of the key results and proofs discussed in this chapter are derived directly from, or inspired by, these foundational works.

The idea of unrolling an optimization algorithm to construct a neural network traces back to the seminal work [GL10]. In this work, the authors demonstrated that sparse coding algorithms—such as the Iterative Shrinkage-Thresholding Algorithm (ISTA)—can be unrolled to form multilayer perceptrons (MLPs), effectively bridging iterative optimization and neural network design. Notably, [MLE19] demonstrated that such unrolled networks are more interpretable, parameter-efficient, and effective compared to generic networks. In this chapter, we build on this perspective to develop principled, white-box deep network architectures by unrolling optimization algorithms that are designed to minimize well-motivated objectives—such as the (sparse) rate reduction objective introduced earlier. This approach not only clarifies the role of each layer in the network but also offers theoretical grounding for architectural choices, moving beyond empirical trial-and-error toward interpretable and goal-driven design. In the following, we compare conventional DNNs, which are typically constructed through empirical design and heuristic tuning, with our mathematically grounded ReduNet architectures:

	Conventional DNNs	ReduNets
Objectives	input/output fitting	information gain
Deep architectures	trial & error	iterative optimization
Layer operators	empirical	projected gradient
Shift invariance	CNNs+augmentation	invariant ReduNets
Initializations	random/pre-design	forward unrolled
Training/fine-tuning	back prop	forward/back prop
Interpretability	black box	white box
Representations	hidden/latent	incoherent subspaces

## 4.5 Exercises and Extensions

*Exercise 4.1.* Let  $\mathbf{Z} = [\mathbf{Z}_1, \dots, \mathbf{Z}_K] \in \mathbb{R}^{d \times m}$  with  $\mathbf{Z}_k \in \mathbb{R}^{d \times m_k}$  for each  $k \in [K]$ . For some  $\alpha > 0$ , let

$$R(\mathbf{Z}) = \log \det (\mathbf{I} + \alpha \mathbf{Z} \mathbf{Z}^T).$$

1. Given any direction  $\mathbf{D} \in \mathbb{R}^{d \times m}$ , please show that  $\nabla R(\mathbf{Z}) = \alpha \mathbf{X}^{-1} \mathbf{Z}$  and

$$\nabla^2 R(\mathbf{Z})[\mathbf{D}, \mathbf{D}] = \alpha \text{Tr}(\mathbf{X}^{-1} \mathbf{D} \mathbf{D}^T) - \frac{\alpha^2}{2} \text{Tr}(\mathbf{X}^{-1} (\mathbf{Z} \mathbf{D}^T + \mathbf{D} \mathbf{Z}^T) \mathbf{X}^{-1} (\mathbf{Z} \mathbf{D}^T + \mathbf{D} \mathbf{Z}^T)),$$

where  $\mathbf{X} \doteq \mathbf{I} + \alpha \mathbf{Z} \mathbf{Z}^T$ . *Hint:* Note that

$$\nabla^2 R(\mathbf{Z})[\mathbf{D}, \mathbf{D}] \doteq \left\langle \lim_{t \rightarrow 0} \frac{\nabla R(\mathbf{Z} + t\mathbf{D}) - \nabla R(\mathbf{Z})}{t}, \mathbf{D} \right\rangle.$$

2. Please show that

$$R(\mathbf{Z}) \leq \sum_{k=1}^K \log \det (\mathbf{I} + \alpha \mathbf{Z}_k \mathbf{Z}_k^T),$$

where the equality holds if and only if  $\mathbf{Z}_k^T \mathbf{Z}_l = \mathbf{0}$  for all  $k \neq l \in [K]$ .

3. Given some  $\alpha > 0$ , let  $\alpha_k = m\alpha/m_k$  for each  $k \in [K]$ . Please derive the closed-form for the first-order critical point of the following function:

$$f(\mathbf{Z}_k) = \frac{1}{2} \log \det (\mathbf{I} + \alpha \mathbf{Z}_k \mathbf{Z}_k^T) - \frac{m_k}{2m} \log \det (\mathbf{I} + \alpha_k \mathbf{Z}_k \mathbf{Z}_k^T) - \frac{\lambda}{2} \|\mathbf{Z}_k\|_F^2.$$

*Hint:* Let  $r_k = \text{rank}(\mathbf{Z}_k)$ . Consider the following singular value decomposition of  $\mathbf{Z}_k$ :

$$\mathbf{Z}_k = \mathbf{P}_k \boldsymbol{\Sigma}_k \mathbf{Q}_k^T = [\mathbf{P}_{k,1} \quad \mathbf{P}_{k,2}] \begin{bmatrix} \tilde{\boldsymbol{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{k,1}^T \\ \mathbf{Q}_{k,2}^T \end{bmatrix},$$

where  $\mathbf{P}_k \in \mathcal{O}^d$  with  $\mathbf{P}_{k,1} \in \mathbb{R}^{d \times r_k}$  and  $\mathbf{P}_{k,2} \in \mathbb{R}^{d \times (d-r_k)}$ ,  $\boldsymbol{\Sigma}_k \in \mathbb{R}^{d \times m_k}$  with  $\tilde{\boldsymbol{\Sigma}}_k \in \mathbb{R}^{r_k \times r_k}$  being a diagonal matrix, and  $\mathbf{Q}_k \in \mathcal{O}^{m_k}$  with  $\mathbf{Q}_{k,1} \in \mathbb{R}^{m_k \times r_k}$  and  $\mathbf{P}_{k,2} \in \mathbb{R}^{m_k \times (m_k-r_k)}$ .

*Exercise 4.2* (Neumann series for matrix inverse). Let  $\mathbf{A} \in \mathbb{R}^{n \times n}$ . If  $\|\mathbf{A}\| < 1$ , please show

$$(\mathbf{I} - \mathbf{A})^{-1} = \sum_{k=1}^{\infty} \mathbf{A}^k. \tag{4.5.1}$$

*Hint:* The proof consists of two steps.

- (i) **Step 1:** Please show that the infinite series  $\sum_{k=1}^{\infty} \mathbf{A}^k$  converges when  $\|\mathbf{A}\| < 1$  using  $\|\mathbf{A}^k\| \leq \|\mathbf{A}\|^k$ .
- (ii) **Step 2:** Compute the matrix product  $(\mathbf{I} - \mathbf{A}) \sum_{k=1}^{\infty} \mathbf{A}^k$ .

*Exercise 4.3.* Please compute the gradients in (4.3.9) and (4.3.10).

*Exercise 4.4.* Please show Corollary 4.1 when  $Kp \leq d$ .



# Chapter 5

## Consistent and Self-Consistent Representations

“*Everything should be made as simple as possible, but not any simpler.*”

— Albert Einstein

In the past chapters, we have established a basic fact that the fundamental goal of learning is to learn a data distribution with low-dimensional supports and transform it to a compact and structured representation. Such a representation reveals intrinsic low-dimensional structures of the data distribution and facilitates subsequent tasks such as classification and generation.

A fundamental approach to learning a good representation of such a distribution is through *compression*. To make the goal of compression measurable and computable, it can be done explicitly by learning a coding scheme that minimizes the coding rate (entropy) or maximizes the information gain (coding rate reduction). In this context, the fundamental role of a deep neural network is to realize a certain iterative optimization algorithm that incrementally optimizes the learned representations in terms of those measures:

$$f: \mathbf{X} \xrightarrow{f^0} \mathbf{Z}^0 \rightarrow \cdots \rightarrow \mathbf{Z}^\ell \xrightarrow{f^\ell} \mathbf{Z}^{\ell+1} \rightarrow \cdots \rightarrow \mathbf{Z}^L = \mathbf{Z}. \quad (5.0.1)$$

In the preceding chapter, we have shown that main architectural characteristics of almost all popular deep networks (ResNet, CNN, and Transformer) can be derived and interpreted from this perspective.

However, when we try to achieve a certain objective through optimization, there is no guarantee that the solution  $\mathbf{Z}$  found in the end by incremental optimization is the correct solution. In fact, even if the optimization process manages to find the globally optimal solution  $\mathbf{Z}^*$ , there is no guarantee that the solution corresponds to a complete representation of the data distribution.<sup>1</sup> Therefore, an outstanding question is how we can ensure that the learned representation of the data distribution is correct or good enough?

Of course, the only way to verify this is to see whether there exists a decoding map, say  $g$ , that can decode the learned representation to reproduce the original data (distribution) well enough:

$$\mathbf{X} \xrightarrow{\mathcal{E}=f} \mathbf{Z} \xrightarrow{\mathcal{D}=g} \hat{\mathbf{X}} \quad (5.0.2)$$

in terms of some measure of similarity:

$$d(\mathbf{X}, \hat{\mathbf{X}}). \quad (5.0.3)$$

This leads to the concept of *consistent representation*. As we have briefly alluded to in Chapter 1 (see Figure 1.23), *autoencoding*, which integrates the encoding and decoding processes, is a natural framework to learn such a representation. We have studied some important special cases in Chapter 2. In this chapter,

---

<sup>1</sup>This could be due to many reasons: for example, the data available for learning the distribution might not be sufficient, or formulation of the optimization program fails to consider some additional constraints or conditions.

Section 5.1, we will study how to extend autoencoding to more general classes of distributions by enforcing the consistency between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ .

In many practical and natural learning scenarios, it can be difficult or even impossible to compare distributions of the data  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ . We are left with the only option to compare in the learned feature  $\mathbf{Z}$  with its image  $\hat{\mathbf{Z}}$  under the encoder  $f$ :

$$\mathbf{X} \xrightarrow{\mathcal{E}=f} \mathbf{Z} \xrightarrow{\mathcal{D}=g} \hat{\mathbf{X}} \xrightarrow{\mathcal{E}=f} \hat{\mathbf{Z}}? \quad (5.0.4)$$

This leads to the notion of a *self-consistent representation*. In Section 5.2, we will study when and how we can learn a consistent representation by enforcing the self-consistency between  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$  only through a closed-loop transcription framework.

Furthermore, in many practical and natural learning scenarios, we normally do not have sufficient samples of the data distribution all at once. For example, animals and humans develop their visual memory through continuously taking in increments of observations all their life. In Section 5.3, we will study how to extend the closed-loop transcription framework to learn a self-consistent representation in a *continuous learning* setting.

Of course, a fundamental motivation why we ever want to identify the low-dimensional structures in a data distribution and find a good representation is to make it easy to use the data for various tasks of intelligence, such as classification, completion, and prediction. Therefore, the resulting joint representation  $(\mathbf{x}, \mathbf{z})$  must be structured in such a way that is best suited for these tasks. In next chapter, we will see how the learned representation can be structured to facilitate conditioned completion or generation tasks.

## 5.1 Learning Consistent Representations

Here we give a formal definition of consistent representations, which are closely related to the concept of autoencoding.

**Definition 5.1** (Consistent Representations). Given data  $\mathbf{X}$ , an *consistent representation* is a pair of functions  $(f: \mathcal{X} \rightarrow \mathcal{Z}, g: \mathcal{Z} \rightarrow \mathcal{X})$ , such that the *features*  $\mathbf{Z} = f(\mathbf{X})$  are compact and structured, and the *autoencoding*

$$\hat{\mathbf{X}} \doteq g(\mathbf{Z}) = g(f(\mathbf{X})) \quad (5.1.1)$$

is *close* to  $\mathbf{X}$  according to either the following two measures:

1. We say that it is *sample-wise* consistent if  $\mathbf{X} \approx \hat{\mathbf{X}}$  in certain norm with high probability.
2. We say that the representation is *distributionally consistent* if  $\text{Law}(\mathbf{X}) \approx \text{Law}(\hat{\mathbf{X}})$ .

Acute readers may have noticed that if we do not impose certain requirements on the representation  $\mathbf{Z}$  sought, the above problem has a trivial solution: One may simply choose the functions  $f$  and  $g$  to be the identity map! Hence, the true purpose of seeking for an autoencoding is to try to ensure that so obtained  $\mathbf{Z}$  is both more compact and more structured than  $\mathbf{X}$ . Firstly, for compactness,  $\mathbf{Z}$  should better reveal the intrinsic low-dimensionality of  $\mathbf{X}$ . Therefore, the representation should maximize a certain information gain, say, measured by the rate reduction

$$\Delta R_\epsilon(\mathbf{Z}) \quad (5.1.2)$$

introduced in Section 3.4.2. Secondly, the main purpose of learning a good representation of the data distribution is to facilitate tasks that exploit the low-dimensionality of its distribution. Hence, the distribution of  $\mathbf{Z}$  should be better structured. For example, the distribution of  $\mathbf{Z}$  is piecewise linear or Gaussian, and its components are largely incoherent or independent etc. These independent components can represent different clusters or classes and can also be easily used as conditions for decoding the corresponding data  $\mathbf{x}$ .

From the definition of consistent representation, it requires that the representation  $\mathbf{Z}$  is sufficient to recover the original data distribution  $\mathbf{X}$  to some degree of accuracy. For sample-wise consistency, a typical choice is to minimize the expected reconstruction error:

$$d(\mathbf{X}, \hat{\mathbf{X}}) = \mathbb{E}[\|\mathbf{X} - \hat{\mathbf{X}}\|_2^2]. \quad (5.1.3)$$

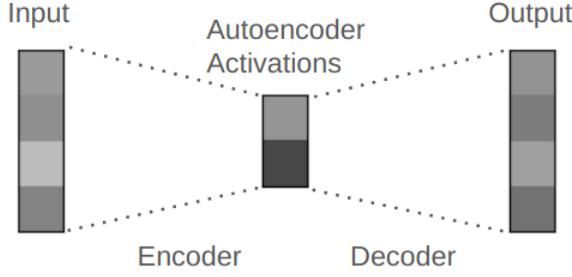


Figure 5.1: Illustration of a typical autoencoder such as PCA, seeking a low-dimensional representation  $\mathbf{z}$  of high-dimensional data  $\mathbf{x}$ .

For consistency in distribution, a typical choice is to minimize a certain distributional distance such as the KL divergence<sup>2</sup>:

$$d(\mathbf{X}, \hat{\mathbf{X}}) = \mathcal{D}_{KL}(\mathbf{X} \parallel \hat{\mathbf{X}}). \quad (5.1.4)$$

Hence, computation aside, when we seek a good autoencoding for a data distribution  $\mathbf{X}$ , conceptually we try to find an encoder  $f$  and decoder  $g$  such that

$$\min_{f,g} [-\Delta R_\epsilon(\mathbf{Z}) + d(\mathbf{X}, \hat{\mathbf{X}})]. \quad (5.1.5)$$

For the rest of this chapter, we will study how to solve such autoencoding problems under different conditions, from simple and ideal cases to increasingly more challenging and realistic conditions.

### 5.1.1 Linear Autoencoding via PCA

According to [Bal11], the phrase “autoencoder” was first introduced by Hinton and Rumelhart [RHW86a] so that a deep representation can be learned via back propagation (BP) in a self-supervision fashion—reconstructing the original data is the self-supervising task. However, the very same concept of seeking a compact and consistent representation has been rooted in many classic studies. As we have already seen in Chapter 2, the classical PCA, ICA, and sparse dictionary learning all share a similar goal. The only difference is when the underlying data distribution is simple (linear and independent), the encoding or decoding mappings become easy to represent and learn: they do not need to be deep and often can be computed in closed form or with an explicit algorithm.

It is instructive to see how the notion of consistency we have defined plays out in the simple case of PCA: here, the consistent encoding and decoding mappings are given by a single-layer linear transform:

$$\mathbf{X} \xrightarrow{\mathcal{E}=\mathbf{U}^\top} \mathbf{Z} \xrightarrow{\mathcal{D}=\mathbf{U}} \hat{\mathbf{X}}, \quad (5.1.6)$$

where  $\mathbf{U} \in \mathbb{R}^{D \times d}$  typically with  $d \ll D$ . Hence  $\mathbf{U}^\top$  represents a projection from a higher-dimensional space  $\mathbb{R}^D$  to a lower one  $\mathbb{R}^d$ , as illustrated in Figure 5.1.

As we saw in Chapter 2, when the distribution of  $\mathbf{x}$  is indeed supported on a low-dimensional subspace  $\mathbf{U}_o$ , the compactness of the representation  $\mathbf{z}$  produced by  $\mathcal{E}$  is a direct consequence of correctly estimating (and enforcing) the dimension of this subspace. Finally, recall that gradient descent on the reconstruction criterion exactly yields these sample-wise consistent mappings: indeed, the optimal solution to the problem

$$\min_{\mathbf{U}^\top \mathbf{U} = \mathbf{I}} \mathbb{E}_{\mathbf{x}} \left[ \|\mathbf{x} - \mathbf{U}\mathbf{U}^\top \mathbf{x}\|_2^2 \right] \quad (5.1.7)$$

precisely coincides with  $\mathbf{U}_o$  when the dimension of the representation is sufficiently large. In this case, we obtain sample-wise consistency for free, since this guarantees that  $\mathbf{U}_o \mathbf{U}_o^\top \mathbf{x} = \mathbf{x}$ . Notice that in the case of

<sup>2</sup>Note that for distributions without common support, which is typical for degenerate distributions, KL divergence may not even be well-defined. In fact, much of the distribution learning literature is trying to address this technical difficulty by replacing or approximating it with something well-defined and efficiently computable.

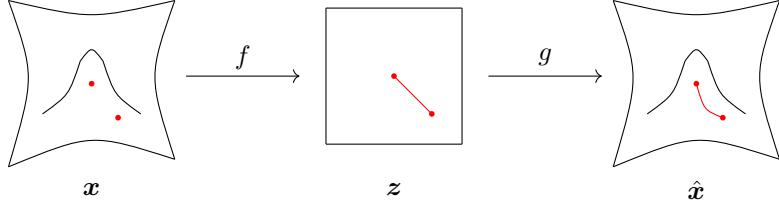


Figure 5.2: A depiction of interpolation through manifold flattening on a manifold in  $\mathbb{R}^3$  of dimension  $d = 2$ . To interpolate two points on the data manifold, map them through the flattening map  $f$  to the flattened space, take their convex interpolants, and then map them back to the data manifold through the reconstruction map  $g$ .

PCA, the rate reduction term in (5.1.5) becomes void as the regularization on the representation  $\mathbf{Z}$  sought is explicit: It spans the entire subspace of lower dimension<sup>3</sup>.

**Online PCA.** Notice that in the above construction, the linear transform  $\mathbf{U}$  used for the encoding and decoding is computed “offline” from all the input data before hand. One question is whether this transform can be learned “online” as the input data come in order? This question was answered by the work of Oja in 1982 [Oja82].

*Example 5.1* (Normalized Hebbian learning scheme for PCA). Consider a sequence of i.i.d. random vectors  $\mathbf{x}_1, \dots, \mathbf{x}_i, \dots \in \mathbb{R}^n$  with covariance  $\Sigma \in \mathbb{R}^{n \times n}$ . Let  $\mathbf{u}_0 \in \mathbb{R}^n$  and define the response of an input vector  $\mathbf{x}_i$  against a weight vector  $\mathbf{u}_i$  to be their inner product:

$$\eta_i = \mathbf{u}_i^T \mathbf{x}_i \quad (5.1.8)$$

and we update the weight vector according to the following scheme:

$$\mathbf{u}_{i+1} = \frac{\mathbf{u}_i + \gamma \eta_i \mathbf{x}_i}{\|\mathbf{u}_i + \gamma \eta_i \mathbf{x}_i\|_2} \quad (5.1.9)$$

for some small gain  $\gamma > 0$ . This update scheme can be viewed as a normalized Hebbian scheme, in which the weights of connections between neurons become stronger if (products of) both the input  $\mathbf{x}$  and output  $\eta$  are strong. One may view the vector of weights  $\mathbf{u}$  are “learned” based on a form of feedback from the output  $\eta$ . Then, under reasonable assumptions, Oja [Oja82] has shown that  $\mathbf{u}_i$  converges to the eigenvector associated with the large eigenvalue of  $\Sigma$ . ■

The normalized Hebbian scheme (5.1.9) can be interpreted as a first-order approximation to a *stochastic* projected gradient descent scheme on the objective of the problem (5.1.7) (with batch size 1, and with the number of columns of  $\mathbf{U}$  equal to 1) as long as  $\|\mathbf{u}\|_2 = 1$ , which is maintained by the projection operation in (5.1.9). It is worth keeping its existence in the back of one’s mind, both as a proof of correctness for the use of stochastic gradient methods for optimizing reconstruction costs such as (5.1.7), and for its suggestion that *simpler algorithms than (end-to-end) back propagation can succeed in learning consistent autoencoders*.

### 5.1.2 Nonlinear PCA and Autoencoding

Of course, one should expect that things will no longer be so simple when we deal with more complex distributions whose underlying low-dimensional structure could be nonlinear.

**Data on a Nonlinear Submanifold.** So, to move beyond the linear structure addressed by PCA, we may assume that the data distribution lies on a (smooth) submanifold  $\mathcal{M}$ . The intrinsic dimension of the submanifold, say  $d$ , is typically much lower than the dimension of the ambient space  $\mathbb{R}^D$ . From this geometric perspective, we typically want to find a nonlinear mapping  $f$  such that the resulting manifold  $f(\mathcal{M})$  is flattened, as illustrated by the example shown in Figure 5.2. The resulting feature  $z$ -space is

<sup>3</sup>One may view  $\Delta R = 0$  in this case.

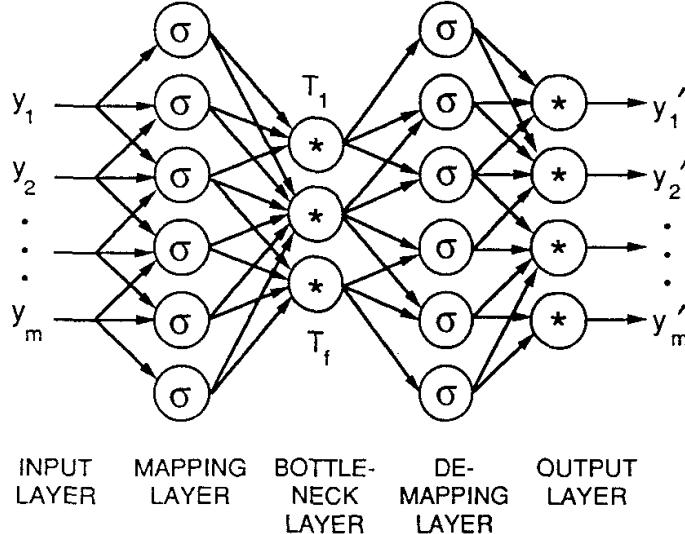


Figure 5.3: Nonlinear PCA by autoassociative neural networks of depth two for both the encoding and decoding mappings, suggested by Kramer [Kra91].

typically more compact (of lower-dimension) than the  $\mathbf{x}$ -space, and the manifold is flat. From the statistical perspective, which is complementary to the geometric perspective but distinct in general, we may also want to ensure that the data distribution on  $\mathcal{M}$  is mapped to a sufficiently regular distribution, say a Gaussian or a uniform distribution (with a very low-dimensional support), in the  $\mathbf{z}$ -space. These two properties ensure that sampling and interpolation in the  $\mathbf{z}$ -space are as easy as possible, and they are mathematical formalizations of the desirable notions of compact and structured features in the low-dimensional manifold model for the data distribution. In general, the problem of learning such an autoencoding mapping for this class of data distributions is known as *nonlinear principal component analysis* (NLPCA).

**A Classical Attempt via a Two-Layer Network.** As we have seen above, in the case of PCA, a one-layer linear neural network is sufficient. That is no longer the case for NLPCA. In 1991, Kramer [Kra91] proposed to solve NLPCA by using a two-layer neural network to represent the encoder mapping  $f$  (or its inverse  $g$ ) based on the universal representation property of two-layer networks with sigmoid activation:

$$\mathbf{z} = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}), \quad (5.1.10)$$

where  $\sigma(\cdot)$  is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (5.1.11)$$

Cybenko [Cyb89] showed that functions of the above form (with enough hidden nodes) can approximate any smooth nonlinear function, say the encoder  $f(\cdot)$ , to an arbitrary precision. In particular, they can represent the flattening and reconstruction maps for data distributions supported on (unions of) low-dimensional manifolds, as in Figure 5.2. The overall architecture of the original networks proposed by Kramer is illustrated in Figure 5.3.

Unfortunately, unlike the above case of PCA, there is in general no closed-form learning scheme for the parameters  $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$  of these networks. Hence it was proposed to train the network via back propagation with the supervision of reconstruction error:

$$\min_{\boldsymbol{\theta}} \mathbb{E}[\|\mathbf{x} - \hat{\mathbf{x}}(\boldsymbol{\theta})\|_2^2]. \quad (5.1.12)$$

Compared to the simple case of PCA, we utilize the same reconstruction objective for learning, but a far more complex nonlinear class of models for parameterizing and learning the encoder and decoder. Although

universal approximation properties such as Cybenko's suggest that *in principle* learning consistent autoencoders via this framework is possible—because for any random sample of data, given enough parameters, such autoencoding pairs exist—one often finds it highly nontrivial to find them with gradient descent. Moreover, to obtain an informative enough reconstruction objective and model for the distribution of high-dimensional real-world data such as images, the required number of samples and hidden nodes can be huge. In addition, as a measure of the compactness of the learned representation, the (lower) dimension of  $\mathbf{z}$  for the bottleneck layer is often chosen heuristically.<sup>4</sup>

**Manifold Flattening via a Deeper Network.** Based on the modern practice of deep networks, such classical shallow and wide network architectures are known to be rather difficult to train effectively and efficiently via back propagation (BP), partly due to the diminishing gradient of the sigmoid function. Hence, the modern practice normally suggests to further decompose the nonlinear transform  $f$  (or  $g$ ) into a composition of many more layers of simpler transforms, resulting a deeper network architecture [HS06], as illustrated in Figure 5.4. In the modern context, further elaborations over the basic reconstruction cost (5.1.12) also prove necessary to achieve good performance on complex real-world data distributions such as images.

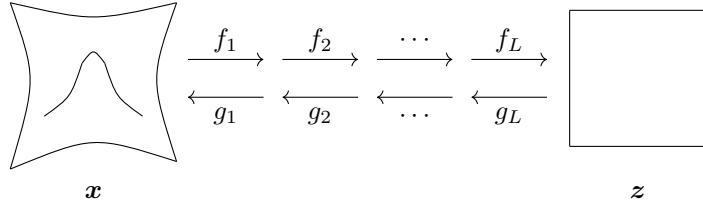


Figure 5.4: A depiction of the construction process of the flattening and reconstruction pair  $(f, g)$ , where the encoder  $f = f_L \circ f_{L-1} \circ \dots \circ f_1$  is constructed from composing flattening layers, and the decoder  $g = g_1 \circ g_2 \circ \dots \circ g_L$  is composed of inversions of each  $f_\ell$ .

In light of universal approximation theorems such as Cybenko's, one may initially wonder why, conceptually, deeper autoencoders should be preferred to shallow ones. From purely an expressivity perspective, we can understand this phenomenon through a geometric angle related to the task of flattening the nonlinear manifold on which our hypothesized data distribution is supported. A purely constructive approach to flattening the manifold proceeds incrementally, in parallel to what we have seen in Chapters 3 and 4 with the interaction between diffusion, denoising, and compression. In the geometric setting, the incremental *flattening process* corresponding to  $f_\ell$  takes the form of transforming a neighborhood of one point of the manifold to be closer to a flat manifold (i.e., a subspace), and enforcing local consistency with the rest of the data samples; the corresponding incremental operation in the decoder,  $g_\ell$ , undoes this transformation. This procedure precisely incorporates curvature information about the underlying manifold, which is estimated from data samples. Given enough samples from the manifold and a careful instantiation of this conceptual process, it is possible to implement this procedure as a computational procedure that verifiably flattens nonlinear manifolds in a white-box fashion [PPR+24]. However, the approach is limited in its applicability to high-dimensional data distributions such as images due to unfavorable scalability, motivating the development of more flexible methods to incremental autoencoding.

### 5.1.3 Sparse Autoencoding

In the above autoencoding schemes, the dimension of the feature space  $d$  is typically chosen to be much lower than that the original data space  $D$  so as to explicitly enforce or promote the learned representation to be low-dimensional. However, in practice, we normally do not know the intrinsic dimension of the data distribution. Hence, the choice of the feature space dimension for autoencoding is often done empirically. In more general situations, the data distribution can be a mixture of a few low-dimensional subspaces or submanifolds. In these cases, it is no longer feasible to enforce a single low-dimensional space for all the features together.

<sup>4</sup>In the later work [HZ93], Hinton et. al. suggested to use the minimum description length (MDL) principle to promote the compactness of the learned coding scheme, in a spirit very similar to the rate distortion measure introduced in this book.

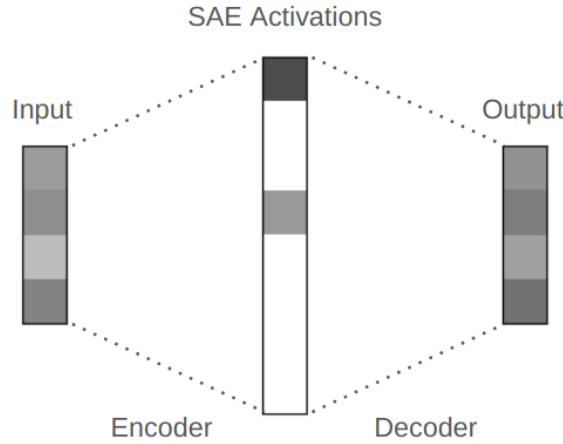


Figure 5.5: Illustration of a sparse autoencoder (SAE), compared to that of a typical autoencoder (AE) in Figure 5.1.

The sparse autoencoder is meant to resolve some of these limitations. In particular, the dimension of the feature space can be equal to or even higher than that of the data space, as illustrated in Figure 5.5. However, the features are required to be highly sparse in the feature space. So if we impose sparsity as the measure of parsimony in addition to the rate reduction in the objective (5.1.5), we obtain a new objective for the sparse autoencoding:

$$\min_{f,g} [\|\mathbf{Z}\|_0 - \Delta R_\epsilon(\mathbf{Z}) + d(\mathbf{X}, \hat{\mathbf{X}})], \quad (5.1.13)$$

where the  $\ell^0$ -“norm”  $\|\cdot\|_0$  is known to promote sparsity. This is very similar to the sparse rate reduction objective (4.2.4) which we used in the previous Chapter 4 to derive the white-box CRATE architecture.

As a method for learning autoencoding pairs in an end-to-end fashion, sparse autoencoding has been practiced in the past [LRM+12; RPC+06], but nearly all modern autoencoding frameworks are instead based on a different, probabilistic autoencoding framework, which we will study now.

### 5.1.4 Variational Autoencoding

In the classical conception of autoencoding, following Hinton and Rumelhart [RHW86a], the data distribution plays a very minor role in the formulation, in spite of its centrality to the representation we ultimately learn. Indeed, in the naive framework, one hopes that by training a deep network to reconstruct samples from the data distribution with a suitably-configured bottleneck for the representation  $\mathbf{z}$ , the learned encoders  $f$  and  $g$  will naturally end up corresponding to a compact and structured feature representation for the data. This is rarely the case in practice. An improved, more modern methodology for autoencoding that still finds significant application to this day is *variational autoencoding* [KW13; KW19]. We will see how this framework, which trains a variational autoencoder (VAE) derived through probabilistic modeling considerations, both generalizes the classical autoencoder training (via minimization of the reconstruction loss), and stabilizes it with appropriate regularization. Later, we will see how to begin to go further and go beyond the black-box nature of the deep networks used to represent the encoding and decoding mappings  $f$  and  $g$ .

#### Probabilistic Perspective on Autoencoding

In the manifold model for the data distribution, the key mathematical objects are the *support* of the data distribution, namely the manifold  $\mathcal{M}$ , and the density of the data on the support, say  $p$ . When we formulate autoencoding from the probabilistic perspective, we often think of the high-dimensional input  $\mathbf{x}$  as having a density  $p$  with support on  $\mathbb{R}^D$ ; one can think of adding a very small amount of noise to the (degenerate) distribution supported on the manifold  $\mathcal{M}$  to obtain this density  $p$ , in line with our denoising-diffusion

constructions in Chapter 3. Then the goal of generative probabilistic modeling is to learn the density  $p$  from samples  $\mathbf{x}$ , say from a class of models  $p(\mathbf{x}; \boldsymbol{\theta})$  parameterized by  $\boldsymbol{\theta}$ . As we have recalled in Chapter 3, a classical approach to achieving this is via maximum likelihood estimation:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}}[\log p(\mathbf{x}; \boldsymbol{\theta})].$$

For certain representative classes of data distributions  $p$  and sufficiently-expressive classes of models  $p(\mathbf{x}; \boldsymbol{\theta})$ , even simpler learning problems than the maximum likelihood estimation problem are known to be statistically hard [YB99]. Hence it is desirable to exploit the knowledge that  $\mathbf{x}$  has low-dimensional structure by seeking to factor the distribution  $p(\mathbf{x}; \boldsymbol{\theta})$  according to a low-dimensional “latent” variable model  $\mathbf{z}$ . Indeed, we may write by conditioning  $p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) = p(\mathbf{z}; \boldsymbol{\theta})p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})$ , and

$$\begin{aligned} p(\mathbf{x}; \boldsymbol{\theta}) &= \int p(\mathbf{z}; \boldsymbol{\theta})p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z} \sim p(\cdot; \boldsymbol{\theta})}[p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})]. \end{aligned}$$

Classically, our model for the data distribution  $p(\mathbf{x}; \boldsymbol{\theta})$  corresponds to a choice of the latent distribution  $p(\mathbf{z}; \boldsymbol{\theta})$  and the conditional distribution  $p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})$ . Even so, computing the model for the data distribution from these latent distributions is intractable except for in special cases, analogous to those we have studied in Chapter 2. By the same token, computing the posterior  $p(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})$  from data, allowing us to *encode* samples to their corresponding latents, is generally intractable. There is hence a tradeoff between the expressivity of our generative model, its computational tractability, and the accuracy of any approximations we make to the underlying probabilistic framework for the sake of computational tractability. In navigating this tradeoff, one also needs a flexible computational approach for learning the model parameters  $\boldsymbol{\theta}$  from data, analogous to the maximum likelihood objective.

In the variational autoencoding framework, we navigate this tradeoff through three key insights:

1. We posit *simple* distributions for  $\mathbf{z}$  and  $\mathbf{x}$  conditional on  $\mathbf{z}$ , but make their parameters depend in a highly flexible way on the input data  $\mathbf{x}$  (where relevant) using deep networks.
2. We replace the posterior  $p(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})$ , used for encoding and whose form is implied (by Bayes rule) by our modeling choices for  $\mathbf{z}$ , with a tractable approximation  $q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})$ , which has its own parameters  $\boldsymbol{\eta}$ .
3. We jointly learn  $\boldsymbol{\theta}$  and  $\boldsymbol{\eta}$  via maximizing a tractable lower bound for the likelihood  $p(\mathbf{x}; \boldsymbol{\theta})$ , known as the evidence lower bound (ELBO).

We will focus on the most useful instantiation of the VAE framework for practice, namely where the prior  $p(\mathbf{z}; \boldsymbol{\theta})$  and the conditional  $p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})$  are both Gaussian. Namely, we use the following Gaussian distributions:

$$\begin{aligned} \mathbf{z} &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \\ \mathbf{x} | \mathbf{z} &\sim \mathcal{N}(g_1(\mathbf{z}), \text{diag}(e^{g_2(\mathbf{z})})\mathbf{I}), \end{aligned}$$

where  $g = (g_1, g_2)$  are deep networks with parameters  $\boldsymbol{\theta}$ , which correspond to the *decoder* in the autoencoder. Similarly, for the approximate posterior  $q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})$ , we use a special Gaussian distribution with parameters given by an encoder MLP  $f = (f_1, f_2)$  with parameters  $\boldsymbol{\eta}$ :

$$\mathbf{z} | \mathbf{x} \sim \mathcal{N}(f_1(\mathbf{x}), \text{diag}(e^{f_2(\mathbf{x})})\mathbf{I}).$$

This makes probabilistic encoding and decoding simple: we simply map our data  $\mathbf{x}$  to mean and variance parameters of a Gaussian distribution for encoding, or vice versa. For learning the encoder and decoder, we start from the maximum likelihood objective Section 5.1.4, and derive a convenient lower bound known as the evidence lower bound, or ELBO. Starting from simple algebraic manipulations

$$\log p(\mathbf{x}; \boldsymbol{\theta}) = \log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{p(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})}$$

$$= \log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})} + \log \frac{q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})}{p(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})},$$

we take expectations with respect to  $\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})$  and use the Gibbs inequality (Theorem 3.1) to get

$$\log p(\mathbf{x}; \boldsymbol{\theta}) \geq \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} \left[ \log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})} \right].$$

The righthand side of this bound is the ELBO; as a lower bound for the pointwise log-likelihood of  $p(\mathbf{x}; \boldsymbol{\theta})$ , its maximization offers a principled compromise between the maximum likelihood objective and computational tractability. Interestingly, the derivation above shows that its tightness depends on the KL divergence between the approximate posterior  $q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})$  and the true posterior  $p(\mathbf{z} | \mathbf{x}; \boldsymbol{\theta})$ , implying that the more accurate our approximate posterior is, the more maximization of the ELBO leads to maximization of the underlying objective of interest, the likelihood of the data. Thus, the VAE objective is:

$$\max_{\boldsymbol{\theta}, \boldsymbol{\eta}} \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} \left[ \log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})} \right]. \quad (5.1.14)$$

By our derivation, maximization of this objective corresponds to a tradeoff between maximizing the likelihood function of the data and minimizing the KL divergence between the approximate and true posterior, which is a highly sensible objective given the VAE modeling assumptions.

### VAE Training as Probabilistic Autoencoding

There is a general methodology for maximizing the ELBO objective in Equation (5.1.14) using stochastic gradient descent and various tractable Monte Carlo estimators for the associated gradients. However, the task is simpler under the Gaussian assumptions we have made above. In this case, the ELBO reads

$$\begin{aligned} & \max_{\boldsymbol{\theta}, \boldsymbol{\eta}} \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} \left[ \log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{q(\mathbf{z} | \mathbf{x}; \boldsymbol{\eta})} \right] \\ &= \max_{\boldsymbol{\theta}, \boldsymbol{\eta}} \left\{ \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} [\log p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})] + \frac{d \log(2\pi e)}{2} + \frac{1}{2} \langle \mathbb{E}_{\mathbf{x}} [f_2(\mathbf{x})], \mathbf{1} \rangle \right\} \\ &\equiv \max_{\boldsymbol{\theta}, \boldsymbol{\eta}} \left\{ \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} [\log p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta}) + \log p(\mathbf{z})] + \frac{1}{2} \langle \mathbb{E}_{\mathbf{x}} [f_2(\mathbf{x})], \mathbf{1} \rangle \right\} \\ &\equiv \max_{\boldsymbol{\theta}, \boldsymbol{\eta}} \left\{ 2 \mathbb{E}_{\mathbf{x}} \left[ \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} [\log p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})] + \langle f_2(\mathbf{x}) - e^{f_2(\mathbf{x})}, \mathbf{1} \rangle - \|f_1(\mathbf{x})\|_2^2 \right] \right\}, \end{aligned} \quad (5.1.15)$$

following Theorem B.1 and the Gaussian entropy calculation done in Chapter B, where  $\equiv$  denotes equivalence of optimization objectives (in each case, we remove some additive constants that do not change the optimization problem's solutions). The remaining term corresponds to the “autoencoding” part of the ELBO objective: intuitively, it seeks to maximize the likelihood of data generated by the decoder when the latents  $\mathbf{z}$  are distributed according to the approximate posterior (generated by the encoder applied to a data sample  $\mathbf{x}$ ), which is a probabilistic form of autoencoding. To see this more directly, consider the special case where the approximate posterior concentrates on its mean  $f_1(\mathbf{x})$ , for every  $\mathbf{x}$ : this is the limit where its log-variance  $f_{2,i}(\mathbf{x}) \rightarrow -\infty$  for each coordinate  $i = 1, \dots, d$ . For simplicity, assume moreover that  $g_2(\mathbf{x}) = \mathbf{1}$ , giving the decoder constant unit variance on each coordinate. Then the loss term in question converges to

$$\begin{aligned} \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})} [\log p(\mathbf{x} | \mathbf{z}; \boldsymbol{\theta})] &\rightarrow \mathbb{E}_{\mathbf{x}} [\log p(\mathbf{x} | f_1(\mathbf{x}); \boldsymbol{\theta})] \\ &\equiv -\frac{1}{2} \mathbb{E}_{\mathbf{x}} [\|\mathbf{x} - g_1 \circ f_1(\mathbf{x})\|_2^2]. \end{aligned}$$

So with a highly confident encoder, which deterministically maps each sample  $\mathbf{x}$  to a single point  $f_1(\mathbf{x})$  in  $\mathbf{z}$ -space, and an isotropic decoder, the “autoencoding” part of the ELBO maximization problem indeed becomes a classical autoencoding objective!<sup>5</sup> At the same time, note that this special case is actually excluded

---

<sup>5</sup>In the general case where  $g_2$  is not fixed as  $\mathbf{1}$ , the reader can verify that the autoencoding term in the ELBO objective converges to a *regularized* classical autoencoding objective.

by the extra terms in Equation (5.1.15)—these effectively correspond to regularization terms that discourage the encoder from collapsing. The general ELBO loss (Equations (5.1.14) and (5.1.15)) is therefore a strict generalization of the classical autoencoding reconstruction objective (Equation (5.1.12)), both in terms of its data fidelity term and in terms of the inclusion of regularization terms.

### Training a VAE

VAEs are typically trained by alternating stochastic gradient ascent on the ELBO objective (Equation (5.1.15)), given individual samples  $\mathbf{x}$  from the true data distribution and from  $\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})$ . In particular, it is standard to collect and train on many independently-generated samples  $\mathbf{z}^i$  for each sample  $\mathbf{x}$ . To take gradients of Equation (5.1.15) with respect to the encoder parameters  $\boldsymbol{\eta}$ , one makes use of the so-called reparameterization trick by writing  $\mathbf{z} \sim q(\cdot | \mathbf{x}; \boldsymbol{\eta})$  as

$$\mathbf{z} =_{\text{d}} f_1(\mathbf{x}) + \text{diag}(e^{\frac{1}{2}f_2(\mathbf{x})})\mathbf{g}, \quad \mathbf{g} \sim \mathcal{N}(0, \mathbf{I}),$$

where  $=_{\text{d}}$  denotes equality in distribution. Then one can simply take many independent standard Gaussian samples  $\mathbf{g}^i$  for each data sample  $\mathbf{x}$ , generate the corresponding samples from the approximate posterior  $\mathbf{z}^i$ , and compute gradients with respect to  $\boldsymbol{\eta}$  using automatic differentiation without any issues.

## 5.2 Learning Self-Consistent Representations

In earlier chapters, we have studied methods that would allow us to learn a low-dimensional distribution via (lossy) compression. As we have mentioned in Chapter 1 and demonstrated in the previous chapters, the progresses made in machine intelligence largely rely on finding computationally feasible and efficient solutions to realize the desired compression, not only computable or tractable in theory, but also scalable in practice:

$$\text{computable} \implies \text{tractable} \implies \text{scalable}. \quad (5.2.1)$$

It is even fair to say that the tremendous advancement in machine intelligence made in the past decade or so is largely attributed to the development of scalable models and methods, say by training deep networks via back propagation. Large models with billions of parameters trained with trillions of data points on tens of thousands of powerful GPUs have demonstrated super-human capabilities in memorizing existing knowledge. This has led many to believe that the “intelligence” of such models will continue to improve as their scale continues to go up.

While we celebrate the engineering marvels of such large man-made machine learning systems, we also must admit that, compared to intelligence in nature, this approach to improve machine intelligence is unnecessarily resource demanding. Natural intelligent beings, including animals and humans, simply cannot afford such a brute force solution to learn because they must operate with a very limited budget in energy, space and time, subject to many strict physical constraints.

Firstly, there is strong scientific evidence that our brain does not conduct global end-to-end back propagation to improve or correct its predictions. Instead, it was long known in neuroscience that our brain corrects errors with local closed-loop feedback, such as predictive coding. This was the scientific basis that had inspired Norbert Wiener to develop the theory of feedback control and the Cybernetics program back in the 1940s.

Secondly, we saw in the previous sections that in order to learn a consistent representation, one needs to learn a bi-directional autoencoding:

$$\mathbf{X} \xrightarrow{\mathcal{E}=f} \mathbf{Z} \xrightarrow{\mathcal{D}=g} \hat{\mathbf{X}}. \quad (5.2.2)$$

It requires to enforce the observed input data  $\mathbf{X}$  and the decoded  $\hat{\mathbf{X}}$  to be close by some measure of similarity  $d(\mathbf{X}, \hat{\mathbf{X}})$ . In nature, animals or humans rarely have direct access to the ground truth  $\mathbf{X}$ . For example, we never have direct access to true 3D shape, distance, or dynamics of objects in a scene. Yet, we have all learned to estimate and predict them very accurately and efficiently. Hence, an outstanding question is how we can learn the true distribution of the  $\mathbf{X}$ , even though we cannot directly compare our estimate  $\hat{\mathbf{x}}$  with the ground truth  $\mathbf{x}$ . As we will see in this chapter, the answer also lies in the closed-loop feedback, as well as the low-dimensionality of the data distribution.

As we know from the previous chapter, in order to ensure a representation is consistent, we need to compare the generated  $\hat{\mathbf{X}} \sim p(\hat{\mathbf{x}})$  and the original  $\mathbf{X} \sim p(\mathbf{x})$ , at least in distribution. Even when we do have access to  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ , technically, computing and minimizing distance of two distributions can be problematic, especially when the support of the distributions is low-dimensional. The KL-divergence introduced in Chapter 3 is not even well-defined between two distributions that do not have overlap supports.

As an early attempt to alleviate the above difficulty in computing and minimizing the distance between two (low-dimensional) distributions, people had suggested to learn the generator/decoder  $g$  via discriminative approaches [Tu07]. This line of thought has led to the idea of *Generative Adversarial Nets (GAN)* [GPM+14b]. It introduces a discriminator  $d$ , usually modeled by a deep network, to discern differences between the generated samples  $\hat{\mathbf{X}}$  and the real ones  $\mathbf{X}$ :

$$\mathbf{Z} \xrightarrow{g(\mathbf{z}, \eta)} \hat{\mathbf{X}}, \mathbf{X} \xrightarrow{d(\mathbf{x}, \theta)} \{\mathbf{0}, \mathbf{1}\}. \quad (5.2.3)$$

It is suggested that we may attempt to align the distributions between  $\hat{\mathbf{x}}$  and  $\mathbf{x}$  via a *Stackelberg game* between the generator  $g$  and the discriminator  $d$ :

$$\max_{\theta} \min_{\eta} \mathbb{E}_{p(\mathbf{x})} [\log d(\mathbf{x}, \theta)] + \mathbb{E}_{p(\mathbf{z})} [1 - \log d(\underbrace{g(\mathbf{z}, \eta)}, \theta)]. \quad (5.2.4)$$

That is, the discriminator  $d$  is trying to minimize the cross entropy between the true samples  $\mathbf{X}$  and the generated ones  $\hat{\mathbf{X}}$  while the generator  $g$  is trying to do the opposite.

One may show that finding an equilibrium for the above Stackelberg game is equivalent to minimizing the *Jensen-Shannon divergence*:

$$\mathcal{D}_{JS}(p(\mathbf{x}), p_g(\hat{\mathbf{x}})) = \mathcal{D}_{KL}(p\|(p + p_g)/2) + \mathcal{D}_{KL}(p_g\|(p + p_g)/2). \quad (5.2.5)$$

Note that, compared to the KL-divergence, the JS-divergence is well-defined even if the supports of the two distributions are non-overlapping. However, JS-divergence does not have a closed-form expression even between two Gaussians, whereas KL-divergence does. In addition, since the data distributions are low-dimensional, the JS-divergence can be highly ill-conditioned to optimize.<sup>6</sup> This may explain why many additional heuristics are typically used in many subsequent variants of GAN.

So, instead, it has also been suggested to replace JS-divergence with the earth mover (EM) distance or the Wasserstein distance.<sup>7</sup> However, both the JS-divergence and Wasserstein distance can only be approximately computed between two general distributions.<sup>8</sup> Furthermore, neither the JS-divergence nor the Wasserstein distance have closed-form formulae, even for the Gaussian distributions.<sup>9</sup>

If it is difficult to compare distributions of the data  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ , would it possible to compare in the learned feature  $\mathbf{Z}$  with its image  $\hat{\mathbf{Z}}$  under the encoder  $f$ :

$$\mathbf{X} \xrightarrow{\mathcal{E}=f} \mathbf{Z} \xrightarrow{\mathcal{D}=g} \hat{\mathbf{X}} \xrightarrow{\mathcal{E}=f} \hat{\mathbf{Z}}? \quad (5.2.6)$$

This leads to the notion of *self-consistent representation*.

**Definition 5.2** (Self-Consistent Representation). Given data  $\mathbf{X}$ , we call an *self-consistent representation* to be a pair of functions  $(f: \mathcal{X} \rightarrow \mathcal{Z}, g: \mathcal{Z} \rightarrow \mathcal{X})$ , such that the *features*  $\mathbf{Z} = f(\mathbf{X})$  are compact and structured, and the *autoencoding features*  $\hat{\mathbf{Z}} \doteq f \circ g(\mathbf{Z})$  is *close* to  $\mathbf{Z}$ .

1. We say that it is *sample-wise* self-consistent if  $\mathbf{Z} \approx \hat{\mathbf{Z}}$  in a certain norm with high probability.
2. We say that the representation is *distributionally self-consistent* if  $\text{Law}(\mathbf{Z}) \approx \text{Law}(\hat{\mathbf{Z}})$ .

<sup>6</sup>as shown in [ACB17].

<sup>7</sup>Roughly speaking, for distributions with potentially non-overlapping low-dimensional supports, the JS-divergence behaves like the  $\ell^0$ -norm, and the EM-distance behaves like the  $\ell^1$ -norm.

<sup>8</sup>For instance, the Wasserstein distance requires one to compute the maximal difference between expectations of the two distributions over all 1-Lipschitz functions.

<sup>9</sup>The ( $\ell^1$ -norm) Wasserstein distance can be bounded by the ( $\ell^2$ -norm) Wasserstein distance which has a closed-form [DDS22]. However, as is well-known in high-dimensional geometry,  $\ell^1$ -norm and  $\ell^2$  norm deviate significantly in terms of their geometric and statistical properties as the dimension becomes high [WM21]. The bound can become very loose.

### 5.2.1 Closed-Loop Transcription via Stackelberg Games

How do we try to ensure a learned representation is self-consistent? As usual, let us assume  $\mathbf{X} = \cup_{k=1}^K \mathbf{X}_k$  with each subset of samples  $\mathbf{X}_k$  belonging to a low-dimensional submanifold:  $\mathbf{X}_k \subset \mathcal{M}_k, k = 1, \dots, K$ . Following the notation in Chapter 3, we use a matrix  $\boldsymbol{\Pi}_k(i, i) = 1$  to denote the membership of sample  $i$  belonging to class  $k$  and  $\boldsymbol{\Pi}_k(i, j) = 0$  otherwise. One seeks a continuous mapping  $f(\cdot, \boldsymbol{\theta}) : \mathbf{x} \mapsto \mathbf{z}$  from  $\mathbf{X}$  to an optimal representation  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N] \subset \mathbb{R}^{d \times N}$ :

$$\mathbf{X} \xrightarrow{f(\mathbf{x}, \boldsymbol{\theta})} \mathbf{Z}, \quad (5.2.7)$$

which maximizes the following coding rate-reduction (MCR<sup>2</sup>) objective:

$$\max_{\mathbf{Z}} \Delta R_\epsilon(\mathbf{Z} | \boldsymbol{\Pi}) \doteq \underbrace{\frac{1}{2} \log \det (\mathbf{I} + \alpha \mathbf{Z} \mathbf{Z}^\top)}_{R_\epsilon(\mathbf{Z})} - \underbrace{\sum_{k=1}^K \frac{\gamma_k}{2} \log \det (\mathbf{I} + \alpha_k \mathbf{Z} \boldsymbol{\Pi}^j \mathbf{Z}^\top)}_{R_\epsilon^c(\mathbf{Z} | \boldsymbol{\Pi})}, \quad (5.2.8)$$

where  $\alpha = d/(N\epsilon^2)$ ,  $\alpha_k = d/(\text{tr}(\boldsymbol{\Pi}_k)\epsilon^2)$ ,  $\gamma_k = \text{tr}(\boldsymbol{\Pi}_k)/N$  for each  $k = 1, \dots, K$ .

One issue with learning such a one-sided mapping (5.2.7) via maximizing the above objective (5.2.8) is that it tends to expand the dimension of the learned subspace for features in each class<sup>10</sup>, as we have seen in Section 3.4 of Chapter 3. To verify whether the learned features are consistent, meaning neither over-estimating nor under-estimating the intrinsic data structure, we may consider learning a decoder  $g(\cdot, \boldsymbol{\eta}) : \mathbf{z} \mapsto \mathbf{x}$  from the representation  $\mathbf{Z} = f(\mathbf{X}, \boldsymbol{\theta})$  back to the data space  $\mathbf{x}$ :  $\hat{\mathbf{X}} = g(\mathbf{Z}, \boldsymbol{\eta})$ :

$$\mathbf{X} \xrightarrow{f(\mathbf{x}, \boldsymbol{\theta})} \mathbf{Z} \xrightarrow{g(\mathbf{z}, \boldsymbol{\eta})} \hat{\mathbf{X}}, \quad (5.2.9)$$

and check how close  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  are. This forms an autoencoding which is what we have studied extensively in the previous Chapter 5.

**Measuring distance in the feature space.** However, as we have discussed above, if we do not have the option to compute the distance between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$ , we are left with the option of comparing their corresponding features  $\mathbf{Z}$  and  $\hat{\mathbf{Z}} = f(\hat{\mathbf{X}}, \boldsymbol{\theta})$ . Notice that under the MCR<sup>2</sup> objective, the distributions of the resulting  $\mathbf{Z}$  or  $\hat{\mathbf{Z}}$  tend to be piecewise linear so their distance can be computed more easily. More precisely, since the features of each class,  $\mathbf{Z}_k$  and  $\hat{\mathbf{Z}}_k$ , are close to be a low-dimensional subspace/Gaussian, their “distance” can be measured by the rate reduction, with (5.2.8) restricted to two sets of equal size:

$$\Delta R_\epsilon(\mathbf{Z}_k, \hat{\mathbf{Z}}_k) \doteq R_\epsilon(\mathbf{Z}_k \cup \hat{\mathbf{Z}}_k) - \frac{1}{2}(R_\epsilon(\mathbf{Z}_k) + R_\epsilon(\hat{\mathbf{Z}}_k)). \quad (5.2.10)$$

The overall distance between  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$  is given by:

$$d(\mathbf{Z}, \hat{\mathbf{Z}}) \doteq \sum_{k=1}^K \Delta R_\epsilon(\mathbf{Z}_k, \hat{\mathbf{Z}}_k) = \sum_{k=1}^K \Delta R_\epsilon(\mathbf{Z}_k, f(g(\mathbf{Z}_k, \boldsymbol{\eta}), \boldsymbol{\theta})). \quad (5.2.11)$$

If we are interested in discerning *any* differences in the distributions of the original data  $\mathbf{X}$  and their decoded  $\hat{\mathbf{X}}$ , we may view the feature encoder  $f(\cdot, \boldsymbol{\theta})$  as a “discriminator” to *magnify* any difference between all pairs of  $\mathbf{X}_k$  and  $\hat{\mathbf{X}}_k$ , by simply maximizing the distance  $d(\mathbf{X}, \hat{\mathbf{X}})$ :

$$\max_f d(\mathbf{Z}, \hat{\mathbf{Z}}) = \max_{\boldsymbol{\theta}} \sum_{k=1}^K \Delta R_\epsilon(\mathbf{Z}_k, \hat{\mathbf{Z}}_k) = \max_{\boldsymbol{\theta}} \sum_{k=1}^K \Delta R_\epsilon(f(\mathbf{X}_k, \boldsymbol{\theta}), f(\hat{\mathbf{X}}_k, \boldsymbol{\theta})). \quad (5.2.12)$$

<sup>10</sup>if the dimension of the feature space  $d$  is too high, maximizing the rate reduction may over-estimate the dimension of each class. Hence, to learn a good representation, one needs to pre-select a proper dimension for the feature space, as achieved in the experiments in [YCY+20]. In fact the same “model selection” problem persists even in the simplest single-subspace case, which is the classic PCA [Jol86]. To our best knowledge, selecting the correct number of principal components in a heterogeneous noisy situation remains an active research topic [HSD20].

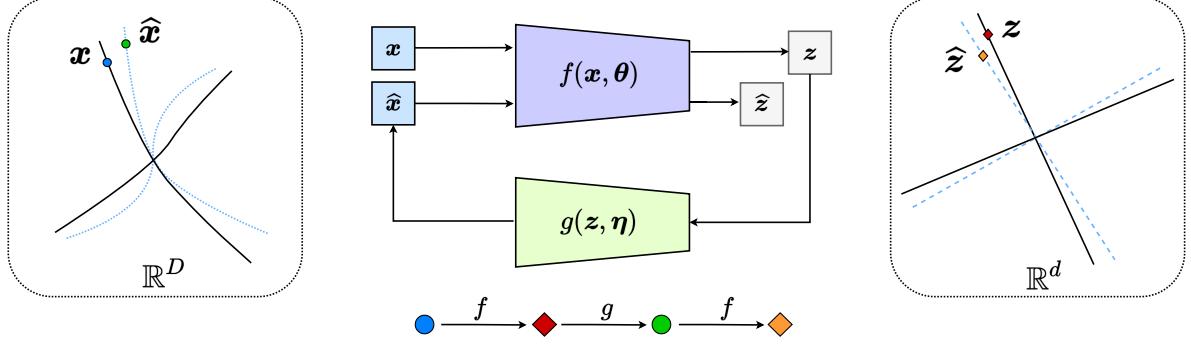


Figure 5.6: **A Closed-loop Transcription.** The encoder  $f$  has dual roles: it learns a representation  $z$  for the data  $\mathbf{x}$  via maximizing the rate reduction of  $z$  and it is also a “feedback sensor” for any discrepancy between the data  $\mathbf{x}$  and the decoded  $\hat{\mathbf{x}}$ . The decoder  $g$  also has dual roles: it is a “controller” that corrects the discrepancy between  $\mathbf{x}$  and  $\hat{\mathbf{x}}$  and it also aims to minimize the overall coding rate for the learned representation.

That is, a “distance” between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  can be measured as the maximally achievable rate reduction between all pairs of classes in these two sets. In a way, this measures how well or badly the decoded  $\hat{\mathbf{X}}$  aligns with the original data  $\mathbf{X}$ —hence measuring the goodness of “injectivity” of the encoder  $f$ . Notice that such a discriminative measure is consistent with the idea of GAN (5.2.3) that tries to separate  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  into two classes, measured by the cross-entropy.

Nevertheless, here the discriminative encoder  $f$  naturally generalizes to cases when the data distributions are multi-class and multi-modal, and the discriminativeness is measured with a more refined measure—the rate reduction—instead of the typical two-class loss (e.g., cross entropy) used in GANs. That is, we may view the encoder  $f$  as a generalized discriminator that replaces the binary classifier  $d$  in (5.2.3):

$$\mathbf{Z} \xrightarrow{g(\mathbf{z}, \eta)} \hat{\mathbf{X}}, \mathbf{X} \xrightarrow{f(\mathbf{x}, \theta)} \{\hat{\mathbf{Z}}, \mathbf{Z}\}. \quad (5.2.13)$$

The overall pipeline can be illustrated by the following “closed-loop” diagram:

$$\mathbf{X} \xrightarrow{f(\mathbf{x}, \theta)} \mathbf{Z} \xrightarrow{g(\mathbf{z}, \eta)} \hat{\mathbf{X}} \xrightarrow{f(\mathbf{x}, \theta)} \hat{\mathbf{Z}}, \quad (5.2.14)$$

where the overall model has parameters:  $\Theta = \{\theta, \eta\}$ . Figure 5.6 shows the overall process.

**Encoder and decoder as a two-player game.** Obviously, to ensure the learned auto-encoding to be self-consistent, the main goal of the decoder  $g(\cdot, \eta)$  is to *minimize* the distance between  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$ . That is, to learn  $g$ , we want to minimize the distance  $d(\mathbf{Z}, \hat{\mathbf{Z}})$ :

$$\min_g d(\mathbf{Z}, \hat{\mathbf{Z}}) \doteq \min_{\eta} \sum_{k=1}^K \Delta R(\mathbf{Z}_k, \hat{\mathbf{Z}}_k) = \min_{\eta} \sum_{k=1}^K \Delta R(\mathbf{Z}_k, f(g(\mathbf{Z}_k, \eta), \theta)), \quad (5.2.15)$$

where  $\mathbf{Z}_k = f(\mathbf{X}_k, \theta)$  and  $\hat{\mathbf{Z}}_k = f(\hat{\mathbf{X}}_k, \theta)$ .

*Example 5.2.* One may wonder why we need the mapping  $f(\cdot, \theta)$  to function as a discriminator between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  by maximizing  $\max_{\theta} \Delta R_e(f(\mathbf{X}, \theta), f(\hat{\mathbf{X}}, \theta))$ . Figure 5.7 gives a simple illustration: there might be many decoders  $g$  such that  $f \circ g$  is an identity (Id) mapping.  $f \circ g(z) = z$  for all  $z$  in the subspace  $S_z$  in the feature space. However,  $g \circ f$  is not necessarily an auto-encoding map for  $\mathbf{x}$  in the original distribution  $S_x$  (here for simplicity drawn as a subspace). That is,  $g \circ f(S_x) \not\subset S_x$ , let alone  $g \circ f(S_x) = S_x$  or  $g \circ f(\mathbf{x}) = \mathbf{x}$ . One should expect, without careful control of the image of  $g$ , with high probability, this would be the case, especially when the support of the distribution of  $\mathbf{x}$  is extremely low-dimensional in the original high-dimensional data space. ■

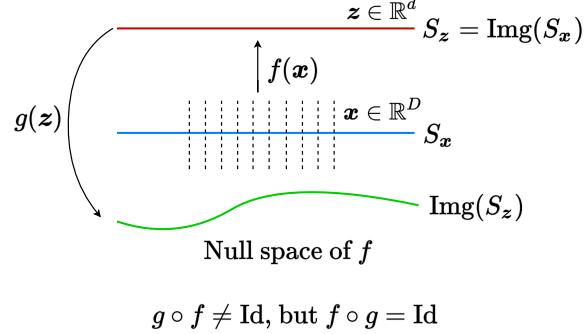


Figure 5.7: **Embeddings of low-dim submanifolds in a high-dim space.**  $S_{\mathbf{x}}$  (blue) is the submanifold for the original data  $\mathbf{x}$ ;  $S_{\mathbf{z}}$  (red) is the image of  $S_{\mathbf{x}}$  under the mapping  $f$ , representing the learned feature  $\mathbf{z}$ ; and the green curve is the image of the feature  $\mathbf{z}$  under the decoding mapping  $g$ .

Comparing the contractive and contrastive nature of (5.2.15) and (5.2.12) on the same distance measure, we see the roles of the encoder  $f(\cdot, \theta)$  and the decoder  $g(\cdot, \eta)$  naturally as “**a two-player game**”: *while the encoder  $f$  tries to magnify the difference between the original data and their transcribed data, the decoder  $g$  aims to minimize the difference*. Now for convenience, let us define the “closed-loop encoding” function:

$$h(\mathbf{x}, \theta, \eta) \doteq f(g(f(\mathbf{x}, \theta), \eta), \theta) : \mathbf{x} \mapsto \hat{\mathbf{z}}. \quad (5.2.16)$$

Ideally, we want this function to be very close to  $f(\mathbf{x}, \theta)$  or at least the distributions of their images should be close. With this notation, combining (5.2.15) and (5.2.12), a closed-loop notion of “distance” between  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  can be computed as *an equilibrium point* to the following Stackelberg game (cf Section A.3) for the same utility in terms of rate reduction:

$$\mathcal{D}(\mathbf{X}, \hat{\mathbf{X}}) \doteq \max_{\theta} \min_{\eta} \sum_{k=1}^K \Delta R_{\epsilon}(f(\mathbf{X}_k, \theta), h(\mathbf{X}_k, \theta, \eta)). \quad (5.2.17)$$

Notice that this only measures the difference between (features of) the original data and its transcribed version. It does not measure how good the representation  $\mathbf{Z}$  (or  $\hat{\mathbf{Z}}$ ) is for the multiple classes within  $\mathbf{X}$  (or  $\hat{\mathbf{X}}$ ). To this end, we may combine the above distance with the original MCR<sup>2</sup>-type objectives (5.2.8): namely, the rate reduction  $\Delta R_{\epsilon}(\mathbf{Z})$  and  $\Delta R_{\epsilon}(\hat{\mathbf{Z}})$  for the learned LDR  $\mathbf{Z}$  for  $\mathbf{X}$  and  $\hat{\mathbf{Z}}$  for the decoded  $\hat{\mathbf{X}}$ . Notice that although the encoder  $f$  tries to *maximize* the multi-class rate reduction of the features  $\mathbf{Z}$  of the data  $\mathbf{X}$ , the decoder  $g$  should *minimize* the rate reduction of the multi-class features  $\hat{\mathbf{Z}}$  of the decoded  $\hat{\mathbf{X}}$ . That is, the decoder  $g$  tries to use a minimal coding rate needed to achieve a good decoding quality.

Hence, the overall “multi-class” Stackelberg game for learning the closed-loop transcription, named CTRL-Multi, is

$$\max_{\theta} \min_{\eta} \mathcal{T}_{\mathbf{X}}(\theta, \eta) \quad (5.2.18)$$

$$\doteq \underbrace{\Delta R_{\epsilon}(f(\mathbf{X}, \theta))}_{\text{Expansive encode}} + \underbrace{\Delta R_{\epsilon}(h(\mathbf{X}, \theta, \eta))}_{\text{Compressive decode}} + \sum_{k=1}^K \underbrace{\Delta R_{\epsilon}(f(\mathbf{X}_k, \theta), h(\mathbf{X}_k, \theta, \eta))}_{\text{Contrastive encode \& Contractive decode}} \quad (5.2.19)$$

$$= \Delta R_{\epsilon}(\mathbf{Z}(\theta)) + \Delta R_{\epsilon}(\hat{\mathbf{Z}}(\theta, \eta)) + \sum_{k=1}^K \Delta R_{\epsilon}(\mathbf{Z}_k(\theta), \hat{\mathbf{Z}}_k(\theta, \eta)), \quad (5.2.20)$$

subject to certain constraints (upper or lower bounds) on the first term and the second term.

Notice that, without the terms associated with the generative part  $h$  or with all such terms fixed as constant, the above objective is precisely the original MCR<sup>2</sup> objective introduced in Chapter 3. In an unsupervised setting, if we view each sample (and its augmentations) as its own class, the above formulation remains exactly the same. The number of classes  $k$  is simply the number of independent samples. In addition,

notice that the above game’s objective function depends only on (features of) the data  $\mathbf{X}$ , hence one can learn the encoder and decoder (parameters) without the need for sampling or matching any additional distribution (as typically needed in GANs or VAEs).

As a special case, if  $\mathbf{X}$  only has one class, the Stackelberg game reduces to a special “two-class” or “binary” form,<sup>11</sup> named CTRL-Binary,

$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \mathcal{T}_{\mathbf{X}}^b(\boldsymbol{\theta}, \boldsymbol{\eta}) \doteq \Delta R_\epsilon(f(\mathbf{X}, \boldsymbol{\theta}), h(\mathbf{X}, \boldsymbol{\theta}, \boldsymbol{\eta})) = \Delta R_\epsilon(\mathbf{Z}(\boldsymbol{\theta}), \hat{\mathbf{Z}}(\boldsymbol{\theta}, \boldsymbol{\eta})), \quad (5.2.21)$$

between  $\mathbf{X}$  and the decoded  $\hat{\mathbf{X}}$  by viewing  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  as two classes  $\{\mathbf{0}, \mathbf{1}\}$ . Notice that this binary case resembles the formulation of the original GAN (5.2.3). Instead of using cross entropy, our formulation adopts a more refined rate-reduction measure, which has been shown to promote diversity in the learned representation in Chapter 3.

Sometimes, even when  $\mathbf{X}$  contains multiple classes/modes, one could still view all classes together as one class. Then, the above binary objective is to align the union distribution of all classes with their decoded  $\hat{\mathbf{X}}$ . This is typically a simpler task to achieve than the multi-class one (5.2.20), since it does not require learning of a more refined multi-class CTRL for the data, as we will later see in experiments. Notice that one good characteristic of the above formulation is that *all quantities in the objectives are measured in terms of rate reduction for the learned features* (assuming features eventually become subspace Gaussians).

One may notice that the above learning framework draws inspiration from closed-loop error correction widely practiced in feedback control systems. The closed-loop mechanism is used to form an overall feedback system between the two encoding and decoding networks for correcting any “error” in the distributions between the data  $\mathbf{x}$  and the decoded  $\hat{\mathbf{x}}$ . Using terminology from control theory, one may view the encoding network  $f$  as a “sensor” for error feedback while the decoding network  $g$  as a “controller” for error correction. However, notice that here the “target” for control is not a scalar nor a finite dimensional vector, but a continuous distribution—in order for the distribution of  $\hat{\mathbf{x}}$  to match that of the data  $\mathbf{x}$ . This is in general a control problem in an infinite dimensional space. The space of possible diffeomorphisms of submanifolds that  $f$  tries to model is infinite-dimensional [Lee02]. Ideally, we hope when the sensor  $f$  and the controller  $g$  are optimal, the distribution of  $\mathbf{x}$  becomes a “fixed point” for the closed loop while the distribution of  $\mathbf{z}$  reaches a compact linear discriminative representation. Hence, the minimax programs (5.2.20) and (5.2.21) can also be interpreted as games between an error-feedback sensor and an error-reducing controller.

The remaining question is whether the above framework can indeed learn a good (autoencoding) presentation of a given dataset? Before we give some formal theoretical justification (in the next subsection), we present some empirical results.

**Visualizing correlation of features  $\mathbf{Z}$  and decoded features  $\hat{\mathbf{Z}}$ .** We visualize the cosine similarity between  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$  learned from the multi-class objective (5.2.20) on MNIST, CIFAR-10 and ImageNet (10 classes), which indicates how close  $\hat{\mathbf{z}} = f \circ g(\mathbf{z})$  is from  $\mathbf{z}$ . Results in Figure 5.8 show that  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$  are aligned very well within each class. The block-diagonal patterns for MNIST are sharper than those for CIFAR-10 and ImageNet, as images in CIFAR-10 and ImageNet have more diverse visual appearances.

**Visualizing auto-encoding of the data  $\mathbf{X}$  and the decoded  $\hat{\mathbf{X}}$ .** We compare some representative  $\mathbf{X}$  and  $\hat{\mathbf{X}}$  on MNIST, CIFAR-10 and ImageNet (10 classes) to verify how close  $\hat{\mathbf{x}} = g \circ f(\mathbf{x})$  is to  $\mathbf{x}$ . The results are shown in Figure 5.9, and visualizations are created from training samples. Visually, the auto-encoded  $\hat{\mathbf{x}}$  faithfully captures major visual features from its respective training sample  $\mathbf{x}$ , especially the pose, shape, and layout. For the simpler dataset such as MNIST, auto-encoded images are almost identical to the original.

## 5.2.2 A Mixture of Low-Dimensional Gaussians

In the above, we have argued that it is possible to formulate the problem of learning a data distribution as a closed-loop autoencoding problem. We also saw empirically that such a scheme seems to work. The remaining question is when and why such a scheme should work. It is difficult to answer this question for the most general cases with arbitrary data distributions. Nevertheless, as usual, let us see if we can

---

<sup>11</sup>as the first two rate reduction terms automatically become zero

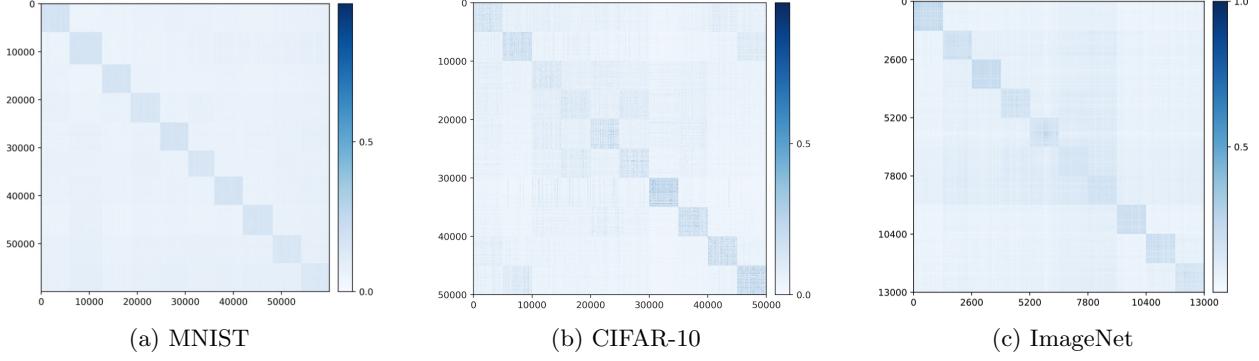


Figure 5.8: Visualizing the alignment between  $\mathbf{Z}$  and  $\hat{\mathbf{Z}}$ :  $|\mathbf{Z}^\top \hat{\mathbf{Z}}|$  and in the feature space for (a) MNIST, (b) CIFAR-10, and (c) ImageNet-10-Class.

arrive at a rigorous justification for the ideal case when the data distribution is a mixture of low-dimensional subspaces or low-rank Gaussians. A clear characterization and understanding of this important special case would shed light on the more general cases.<sup>12</sup>

To this end, let us first suppose that  $\mathbf{X}$  is distributed according to a mixture of low-dimensional Gaussians, and the label (i.e., subspace assignment) for  $\mathbf{X}$  is given by  $\mathbf{y}$ . Then, let us set up a minimax optimization problem to learn the data distribution, say through learning an encoding of  $\mathbf{X}$  into representations  $\mathbf{Z}$  which are supported on a mixture of *orthogonal* subspaces, and a decoding of  $\mathbf{Z}$  back into  $\mathbf{X}$ . Then, the representation we want to achieve is maximized by the earlier-discussed version of the information gain, i.e.,  $\Delta R_\epsilon(\mathbf{Z}) = R_\epsilon(\mathbf{Z}) - \sum_{k=1}^K R_\epsilon(\mathbf{Z}_k)$ , which enforces that the representation  $\mathbf{Z}_k$  of each class  $k$  spans a subspace which is orthogonal to the supporting subspaces of other classes. The way to measure the consistency of the decoding is, as before, given by  $\sum_{k=1}^K \Delta R_\epsilon(\mathbf{Z}_k, \hat{\mathbf{Z}}_k)$ , which enforces that the representation  $\mathbf{Z}_k$  and its autoencoding  $\hat{\mathbf{Z}}_k$  for each class  $k$  span the same subspace. Thus, we can set up a simplified Stackelberg game:

$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \left\{ \Delta R_\epsilon(\mathbf{Z}(\boldsymbol{\theta})) + \sum_{k=1}^K \Delta R_\epsilon(\mathbf{Z}_k(\boldsymbol{\theta}), \hat{\mathbf{Z}}_k(\boldsymbol{\theta}, \boldsymbol{\eta})) \right\} \quad (5.2.22)$$

Notice that this is a simpler setup than what's used in practice—there's no  $\Delta R_\epsilon(\hat{\mathbf{Z}})$  term, for instance, and we work in the supervised setting with class labels (although the techniques used to prove the following result are easy to extend to unsupervised formulations). Also, the consistency of the representations is only measured in a distribution-wise sense via  $\Delta R_\epsilon$  (though this may be substituted with a sample-wise distance metric such as the  $\ell_2$  norm if desired, and equivalent conclusions may be drawn, *mutatis mutandis*).

Under mild conditions, in order to realize the desired encoder and decoder which realize  $\mathbf{Z}$  from a data source  $\mathbf{X}$  that is already distributed according to a mixture of correlated low-dimensional Gaussians, we only require a linear encoder and decoder to disentangle and whiten the Gaussians. We then study this setting in the case where  $\boldsymbol{\theta}$  and  $\boldsymbol{\eta}$  parameterize matrices whose operator norm is constrained.

We want to understand what kinds of optima are learned in this setting. One suitable solution concept for this kind of game, where the decoder's optimal solution is defined solely with respect to the encoder (and not, in particular, with respect to some other intrinsic property of the decoder), is a *Stackelberg equilibrium* where the decoder follows the encoder. Namely, at such an equilibrium, the decoder should optimize its objective; meanwhile the encoder should optimize its objective, given that whatever it picks, the decoder will pick an optimal response (which may affect the encoder objective). In game-theoretic terms, it is like the decoder goes *second*: it chooses its weights after the encoder, and both the encoder and decoder attempt to optimize their objective in light of this. It is computationally tractable to learn sequential equilibria via gradient methods via *alternating optimization*, where each side uses different learning rates. Detailed exposition of sequential equilibria is beyond the scope of this book and we provide more technical details in Section A.3. In this setting, we have the following result:

---

<sup>12</sup>As most distributions with low-dimensional structures can be well-approximated by this family of distributions.

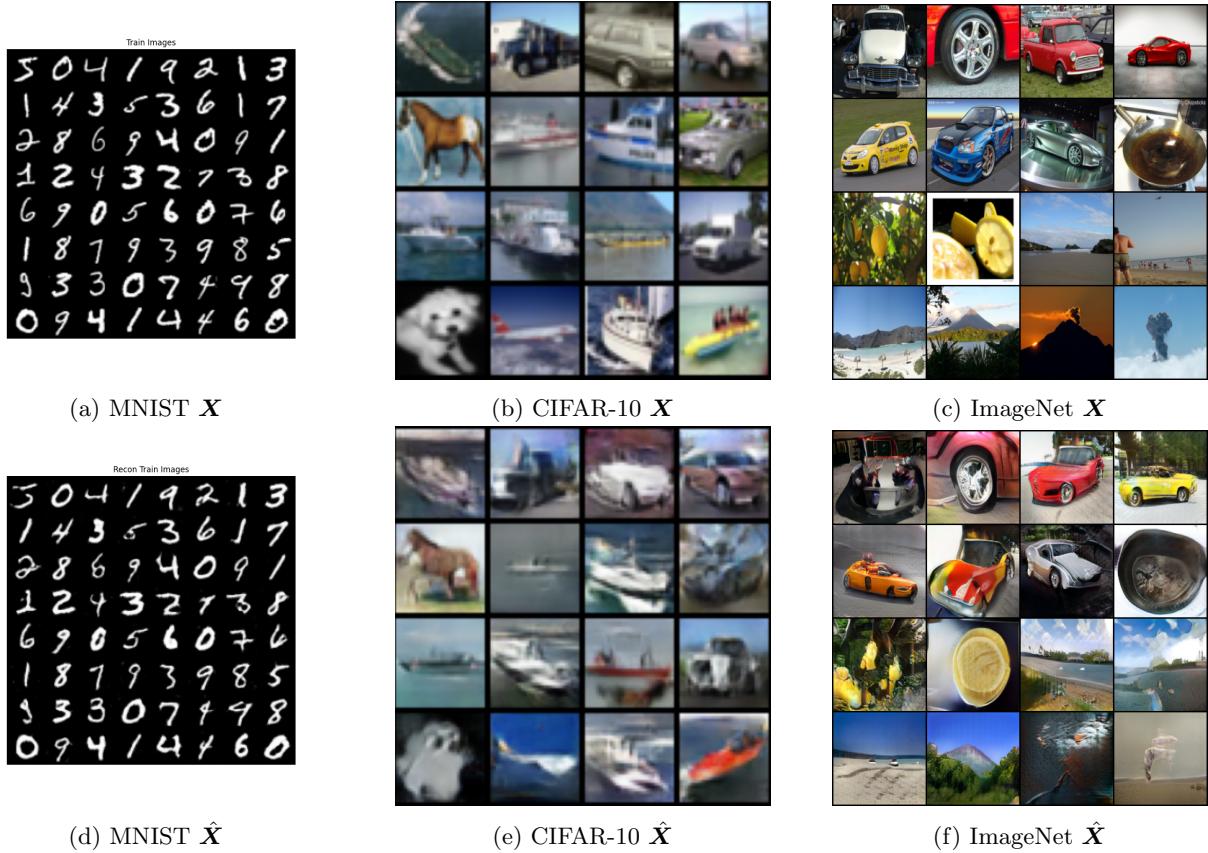


Figure 5.9: Visualizing the auto-encoding property of the learned closed-loop transcription ( $\mathbf{x} \approx \hat{\mathbf{x}} = g \circ f(\mathbf{x})$ ) on MNIST, CIFAR-10, and ImageNet (zoom in for better visualization).

**Theorem 5.1** ([PPC+23], Abridged). *Suppose that  $\mathbf{X}$  is distributed on a mixture of subspaces. Under certain realistic yet technical conditions, it holds that all sequential equilibria of (5.2.22) obey:*

- The  $\mathbf{Z}_k$  lie on orthogonal subspaces and are isotropic on those subspaces, i.e., maximizing the information gain.
- The autoencoding is self-consistent, i.e., the subspaces spanned by  $\mathbf{Z}_k$  and  $\hat{\mathbf{Z}}_k$  are the same for all  $k$ .

This notion of self-consistency is the most one can expect if there are only geometric assumptions on the data, i.e., there are no statistical assumptions. If we assume that the columns of  $\mathbf{X}$  is drawn from a low-rank Gaussian mixture model, then analogous versions of this theorem certify that  $\mathbf{Z}_k$  are also low-rank Gaussians whose covariance is isotropic, for instance. Essentially, this result validates, via the simple case of Gaussian mixtures on subspaces, that minimax games to optimize the information gain and self-consistency may achieve optimal solutions.

## 5.3 Continuous Learning Self-Consistent Representations

### 5.3.1 Class-wise Incremental Learning

As we have seen, deep neural networks have demonstrated a great ability to learn representations for hundreds or even thousands of classes of objects, in both discriminative and generative contexts. However, networks typically must be trained offline, with uniformly sampled data from all classes simultaneously. It has been known that when an (open-loop) network is updated to learn new classes without data from the old ones, previously learned knowledge will fall victim to the problem of *catastrophic forgetting* [MC89]. This is known

in neuroscience as the stability-plasticity dilemma: the challenge of ensuring that a neural system can learn from a new environment while retaining essential knowledge from previous ones [Gro87].

In contrast, natural neural systems (e.g. animal brains) do not seem to suffer from such catastrophic forgetting at all. They are capable of developing new memory of new objects while retaining memory of previously learned objects. This ability, for either natural or artificial neural systems, is often referred to as *incremental learning*, *continual learning*, *sequential learning*, or *life-long learning* [AR20].

While many recent works have highlighted how artificial neural systems can be trained in more flexible ways, the strongest existing efforts toward answering the stability-plasticity dilemma for artificial neural networks typically require either storing raw exemplars [CRE+19; RKS+17] or providing external mechanisms [KPR+17]. Raw exemplars, particularly in the case of high-dimensional inputs like images, are costly and difficult to scale, while external mechanisms—which typically include secondary networks and representation spaces for generative replay, incremental allocation of network resources, network duplication, or explicit isolation of used and unused parts of the network—require heuristics and incur hidden costs.

Here we are interested in an incremental learning setting that is similar to nature. It counters these existing practices with two key qualities.

1. The first is that it should be *memory-based*. When learning new classes, no raw exemplars of old classes are available to train the network together with new data. This implies that one has to rely on a compact and thus structured “memory” learned for old classes, such as incrementally learned generative representations of the old classes, as well as the associated encoding and decoding mappings [KK18].
2. The second is that it should be *self-contained*. Incremental learning takes place in a single neural system with a fixed capacity, and in a common representation space. The ability to minimize forgetting is implied by optimizing an overall learning objective, without external networks, architectural modifications, or resource allocation mechanisms.

The incoherent linear structures for features of different classes closely resemble how objects are encoded in different areas of the inferotemporal cortex of animal brains [BSM+20; CT17]. The closed-loop transcription  $\mathbf{X} \rightarrow \mathbf{Z} \rightarrow \hat{\mathbf{X}} \rightarrow \hat{\mathbf{Z}}$  also resembles popularly hypothesized mechanisms for memory formation [JT20; VST+20]. This leads to a question: since memory in the brains is formed in an incremental fashion, can the above closed-loop transcription framework also support incremental learning?

**LDR memory sampling and replay.** The simple linear *structures* of LDR make it uniquely suited for incremental learning: the distribution of features  $\mathbf{Z}_j$  of each previously learned class can be explicitly and concisely represented by a principal subspace  $\mathcal{S}_j$  in the feature space. To preserve the memory of an old class  $j$ , we only need to preserve the subspace while learning new classes. To this end, we simply sample  $m$  representative prototype features on the subspace along its top  $r$  principal components, and denote these features as  $\mathbf{Z}_{j,old}$ . Because of the simple linear structures of LDR, we can sample from  $\mathbf{Z}_{j,old}$  by calculating the mean and covariance of  $\mathbf{Z}_{j,old}$  after learning class  $j$ . The storage required is extremely small, since we only need to store means and covariances, which are sampled from as needed. Suppose a total of  $t$  old classes have been learned so far. If prototype features, denoted  $\mathbf{Z}_{old} \doteq [\mathbf{Z}_{old}^1, \dots, \mathbf{Z}_{old}^t]$ , for all of these classes can be preserved when learning new classes, the subspaces  $\{\mathcal{S}_j\}_{j=1}^t$  representing past memory will be preserved as well. Details about sampling and calculating mean and covariance can be found in the work of [TDW+23].

**Incremental learning LDR with an old-memory constraint.** Notice that, with the learned auto-encoding (5.2.9), one can replay and use the images, say  $\hat{\mathbf{X}}_{old} = g(\mathbf{Z}_{old}, \boldsymbol{\eta})$ , associated with the memory features to avoid forgetting while learning new classes. This is typically how generative models have been used for prior incremental learning methods. However, with the closed-loop framework, explicitly replaying images from the features is not necessary. Past memory can be effectively preserved through optimization exclusively on the features themselves.

Consider the task of incrementally learning a new class of objects.<sup>13</sup> We denote a corresponding new sample set as  $\mathbf{X}_{new}$ . The features of  $\mathbf{X}_{new}$  are denoted as  $\mathbf{Z}_{new}(\boldsymbol{\theta}) = f(\mathbf{X}_{new}, \boldsymbol{\theta})$ . We concatenate them

---

<sup>13</sup>Of course, one may also consider the more general setting where the task contains a small batch of new classes, without serious modification.

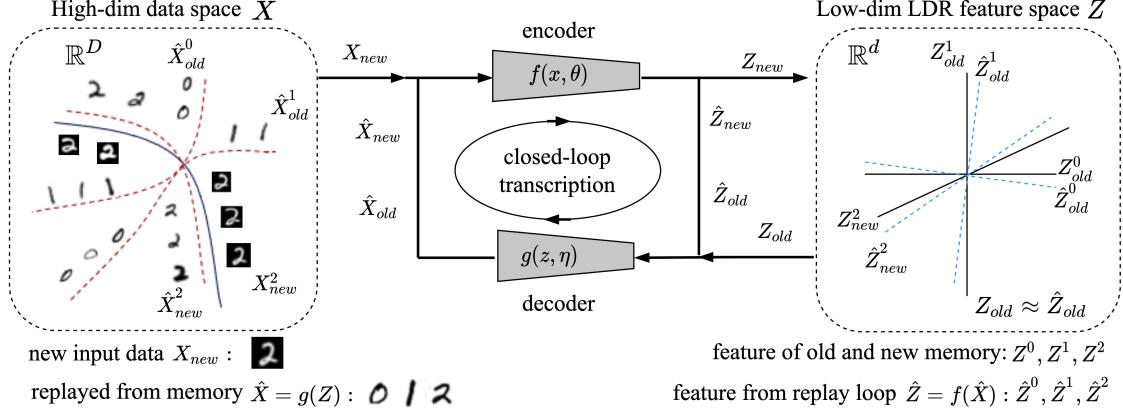


Figure 5.10: **Overall framework** of our closed-loop transcription-based incremental learning for a structured LDR memory. Only a single, entirely self-contained, encoding-decoding network is needed: for a new data class  $\mathbf{X}_{new}$ , a new LDR memory  $\mathbf{Z}_{new}$  is incrementally learned as a minimax game between the encoder and decoder subject to the constraint that old memory of past classes  $\mathbf{Z}_{old}$  is intact through the closed-loop transcription (or replay):  $\mathbf{Z}_{old} \approx \hat{\mathbf{Z}}_{old} = f(g(\mathbf{Z}_{old}))$ .

together with the prototype features of the old classes  $\mathbf{Z}_{old}$  and form  $\mathbf{Z} = [\mathbf{Z}_{new}(\boldsymbol{\theta}), \mathbf{Z}_{old}]$ . We denote the replayed images from all features as  $\hat{\mathbf{X}} = [\hat{\mathbf{X}}_{new}(\boldsymbol{\theta}, \boldsymbol{\eta}), \hat{\mathbf{X}}_{old}(\boldsymbol{\eta})]$  although we do not actually need to compute or use them explicitly. We only need features of replayed images, denoted  $\hat{\mathbf{Z}} = f(\hat{\mathbf{X}}, \boldsymbol{\theta}) = [\hat{\mathbf{Z}}_{new}(\boldsymbol{\theta}, \boldsymbol{\eta}), \hat{\mathbf{Z}}_{old}(\boldsymbol{\theta}, \boldsymbol{\eta})]$ .

Mirroring the motivation for the multi-class CTRL objective (5.2.20), we would like the features of the new class  $\mathbf{Z}_{new}$  to be incoherent to all of the old ones  $\mathbf{Z}_{old}$ . As  $\mathbf{Z}_{new}$  is the only new class whose features needs to be learned, the objective (5.2.20) reduces to the case where  $k = 1$ :

$$\min_{\boldsymbol{\eta}} \max_{\boldsymbol{\theta}} \Delta R_{\epsilon}(\mathbf{Z}) + \Delta R_{\epsilon}(\hat{\mathbf{Z}}) + \Delta R_{\epsilon}(\mathbf{Z}_{new}, \hat{\mathbf{Z}}_{new}). \quad (5.3.1)$$

However, when we update the network parameters  $(\boldsymbol{\theta}, \boldsymbol{\eta})$  to optimize the features for the new class, the updated mappings  $f$  and  $g$  will change features of the old classes too. Hence, to minimize the distortion of the old class representations, we can try to enforce  $\text{Cov}(\mathbf{Z}_{j,old}) = \text{Cov}(\hat{\mathbf{Z}}_{j,old})$ . In other words, while learning new classes, we enforce the memory of old classes remain “self-consistent” through the transcription loop:

$$\mathbf{Z}_{old} \xrightarrow{g(\mathbf{z}, \boldsymbol{\eta})} \hat{\mathbf{X}}_{old} \xrightarrow{f(\mathbf{x}, \boldsymbol{\theta})} \hat{\mathbf{Z}}_{old}. \quad (5.3.2)$$

Mathematically, this is equivalent to setting

$$\Delta R_{\epsilon}(\mathbf{Z}_{old}, \hat{\mathbf{Z}}_{old}) \doteq \sum_{j=1}^t \Delta R_{\epsilon}(\mathbf{Z}_{j,old}, \hat{\mathbf{Z}}_{j,old}) = 0.$$

Hence, the above minimax program (5.3.1) is revised as a *constrained* minimax game, which we refer to as *incremental closed-loop transcription* (i-CTRL). The objective of this game is identical to the standard multi-class CTRL objective (5.2.20), but includes just one additional constraint:

$$\begin{aligned} \min_{\boldsymbol{\eta}} \max_{\boldsymbol{\theta}} \quad & \Delta R_{\epsilon}(\mathbf{Z}) + \Delta R_{\epsilon}(\hat{\mathbf{Z}}) + \Delta R_{\epsilon}(\mathbf{Z}_{new}, \hat{\mathbf{Z}}_{new}) \\ \text{subject to} \quad & \Delta R(\mathbf{Z}_{old}, \hat{\mathbf{Z}}_{old}) = 0. \end{aligned} \quad (5.3.3)$$

In practice, the constrained minimax program can be solved by *alternating* minimization and maximization between the encoder  $f(\cdot, \boldsymbol{\theta})$  and decoder  $g(\cdot, \boldsymbol{\eta})$  as follows:

$$\max_{\boldsymbol{\theta}} \quad \Delta R_{\epsilon}(\mathbf{Z}) + \Delta R_{\epsilon}(\hat{\mathbf{Z}}) + \lambda \cdot \Delta R_{\epsilon}(\mathbf{Z}_{new}, \hat{\mathbf{Z}}_{new}) - \gamma \cdot \Delta R_{\epsilon}(\mathbf{Z}_{old}, \hat{\mathbf{Z}}_{old}), \quad (5.3.4)$$

$$\min_{\boldsymbol{\eta}} \quad \Delta R_{\epsilon}(\mathbf{Z}) + \Delta R_{\epsilon}(\hat{\mathbf{Z}}) + \lambda \cdot \Delta R_{\epsilon}(\mathbf{Z}_{new}, \hat{\mathbf{Z}}_{new}) + \gamma \cdot \Delta R_{\epsilon}(\mathbf{Z}_{old}, \hat{\mathbf{Z}}_{old}); \quad (5.3.5)$$

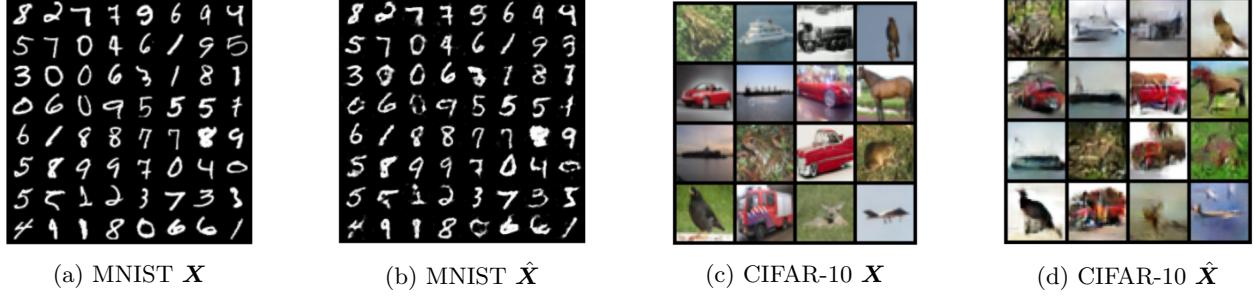


Figure 5.11: Visualizing the auto-encoding property of the learned ( $\hat{\mathbf{X}} = g \circ f(\mathbf{X})$ ).

where the constraint  $\Delta R_\epsilon(\mathbf{Z}_{old}, \hat{\mathbf{Z}}_{old}) = 0$  in (5.3.3) has been converted (and relaxed) to a Lagrangian term with a corresponding coefficient  $\gamma$  and sign. We additionally introduce another coefficient  $\lambda$  for weighting the rate reduction term associated with the new data.

**Jointly optimal memory via incremental reviewing.** As we will see, the above constrained minimax program can already achieve state of the art performance for incremental learning. Nevertheless, developing an optimal memory for *all classes* cannot rely on graceful forgetting alone. Even for humans, if an object class is learned only once, we should expect the learned memory to fade as we continue to learn new others, unless the memory can be consolidated by reviewing old object classes.

To emulate this phase of memory forming, after incrementally learning a whole dataset, we may go back to review all classes again, one class at a time. We refer to going through all classes once as one reviewing “cycle”.<sup>14</sup> If needed, multiple reviewing cycles can be conducted. It is quite expected that reviewing can improve the learned (LDR) memory. But somewhat surprisingly, the closed-loop framework allows us to review even in a “class-unsupervised” manner: when reviewing data of an old class say  $\mathbf{X}_j$ , the system does not need the class label and can simply treat  $\mathbf{X}_j$  as a new class  $\mathbf{X}_{new}$ . That is, the system optimizes the same constrained mini-max program (5.3.3) without any modification; after the system is optimized, one can identify the newly learned subspace spanned by  $\mathbf{Z}_{new}$ , and use it to replace or merge with the old subspace  $\mathcal{S}_j$ . As our experiments show, such an class-unsupervised incremental review process can gradually improve both discriminative and generative performance of the LDR memory, eventually converging to that of a jointly-learned memory.

**Experimental verification.** We show some experimental results on the following datasets: MNIST [LBB+98b] and CIFAR-10 [KNH14]. All experiments are conducted for the more challenging class-IL setting. For both MNIST and CIFAR-10, the 10 classes are split into 5 tasks with 2 classes each or 10 tasks with 1 class each. For the encoder  $f$  and decoder  $g$ , we adopt a very simple network architecture modified from DCGAN [RMC16], which is merely a *four-layer* convolutional network. Here we only show some qualitative visual results and more experiments and analytical analysis can be found in the work [TDW+23].

**Visualizing auto-encoding properties.** We begin by qualitatively visualizing some representative images  $\mathbf{X}$  and the corresponding replayed  $\hat{\mathbf{X}}$  on MNIST and CIFAR-10. The model is learned incrementally with the datasets split into 5 tasks. Results are shown in Figure 5.11, where we observe that the reconstructed  $\hat{\mathbf{X}}$  preserves the main visual characteristics of  $\mathbf{X}$  including shapes and textures. For a simpler dataset like MNIST, the replayed  $\hat{\mathbf{X}}$  are almost identical to the input  $\mathbf{X}$ ! This is rather remarkable given: (1) our method does not explicitly enforce  $\hat{\mathbf{x}} \approx \mathbf{x}$  for individual samples as most autoencoding methods do, and (2) after having incrementally learned all classes, the generator has not forgotten how to generate digits learned earlier, such as 0, 1, 2. For a more complex dataset like CIFAR-10, we also demonstrate good visual quality, faithfully capturing the essence of each image.

<sup>14</sup>to distinguish from the term “epoch” used in the conventional joint learning setting.

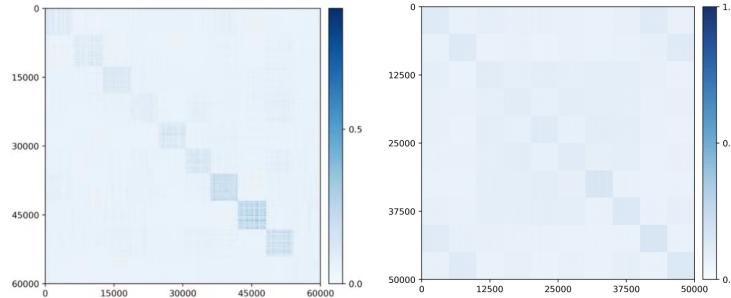


Figure 5.12: Block diagonal structure of  $|Z^T Z|$  in the feature space for MNIST (left) and CIFAR-10 (right).

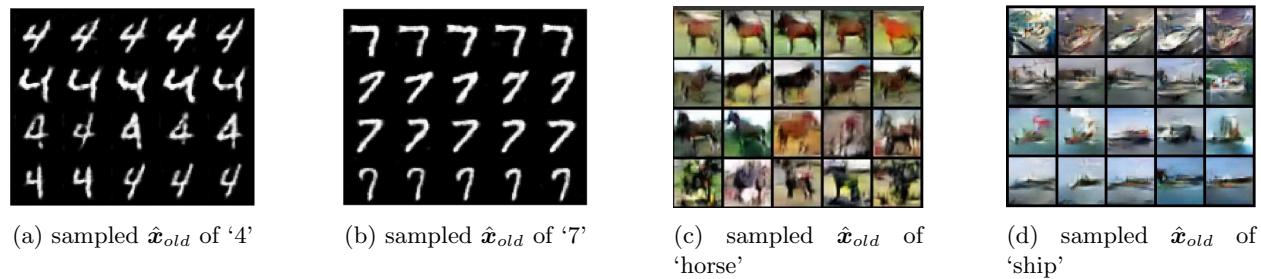


Figure 5.13: Visualization of 5 reconstructed  $\hat{x} = g(z)$  from  $z$ 's with the closest distance to (top-4) principal components of learned features for MNIST (class ‘4’ and class ‘7’) and CIFAR-10 (class ‘horse’ and ‘ship’).

**Principal subspaces of the learned features.** Most generative memory-based methods utilize autoencoders, VAEs, or GANs for replay purposes. The structure or distribution of the learned features  $Z_j$  for each class is unclear in the feature space. The features  $Z_j$  of the LDR memory, on the other hand, have a clear linear structure. Figure 5.12 visualizes correlations among all learned features  $|Z^T Z|$ , in which we observe clear block-diagonal patterns for both datasets.<sup>15</sup> This indicates the features for different classes  $Z_j$  indeed lie on subspaces that are incoherent from one another. Hence, features of each class can be well modeled as a principal subspace in the feature space.

**Replay images of samples from principal components.** Since features of each class can be modeled as a principal subspace, we further visualize the individual principal components within each of those subspaces. Figure 5.13 shows the images replayed from sampled features along the top-4 principal components for different classes, on MNIST and CIFAR-10 respectively. Each row represents samples along one principal component and they clearly show similar visual characteristics but distinctively different from those in other rows. We see that the model remembers different poses of ‘4’ after having learned all remaining classes. For CIFAR-10, the incrementally learned memory remembers representative poses and shapes of horses and ships.

**Effectiveness of incremental reviewing.** We verify how the incrementally learned LDR memory can be further consolidated with an unsupervised incremental reviewing phase described before. Experiments are conducted on CIFAR-10, with 10 steps. Figure 5.14 left shows replayed images of the first class ‘airplane’ at the end of incremental learning of all ten classes, sampled along the top-3 principal components – every two rows (16 images) are along one principal direction. Their visual quality remains very decent – observed almost no forgetting. The right figure shows replayed images after reviewing the first class once. We notice a significant improvement in visual quality after the reviewing, and principal components of the features in the subspace start to correspond to distinctively different visual attributes within the same class.

<sup>15</sup>Notice that these patterns closely resemble the similarity matrix of response profiles of object categories from different areas of the inferotemporal cortex, as shown in Extended DataFig.3 of [BSM+20].



Figure 5.14: Visualization of replayed images  $\hat{\mathbf{x}}_{old}$  of class 1-‘airplane’ in CIFAR-10, before (left) and after (right) one reviewing cycle.

### 5.3.2 Sample-wise Continuous Unsupervised Learning

As we know, the closed-loop CTRL formulation can already learn a decent autoencoding, even without class information, with the CTRL-Binary program:

$$\max_{\theta} \min_{\eta} \Delta R_\epsilon(\mathbf{Z}, \hat{\mathbf{Z}}) \quad (5.3.6)$$

However, note that (5.3.6) is practically limited because it only aligns the dataset  $\mathbf{X}$  and the regenerated  $\hat{\mathbf{X}}$  at the distribution level. There is no guarantee that for each sample  $\mathbf{x}$  would be close to the decoded  $\hat{\mathbf{x}} = g(f(\mathbf{x}))$ .

**Sample-wise constraints for unsupervised transcription.** To improve discriminative and generative properties of representations learned in the unsupervised setting, we propose two additional mechanisms for the above CTRL-Binary maximin game (5.3.6). For simplicity and uniformity, here these will be formulated as equality constraints over rate reduction measures, but in practice they can be enforced softly during optimization.

**Sample-wise self-consistency via closed-loop transcription.** First, to address the issue that CTRL-Binary does not learn a sample-wise consistent autoencoding, we need to promote  $\hat{\mathbf{x}}$  to be close to  $\mathbf{x}$  for each sample. In the CTRL framework, this can be achieved by enforcing their corresponding features  $\mathbf{z} = f(\mathbf{x})$  and  $\hat{\mathbf{z}} = f(\hat{\mathbf{x}})$  to be close. To promote sample-wise self-consistency, where  $\hat{\mathbf{x}} = g(f(\mathbf{x}))$  is close to  $\mathbf{x}$ , we want the distance between  $\mathbf{z}$  and  $\hat{\mathbf{z}}$  to be zero or small, for all  $N$  samples. This distance can be measured by the rate reduction:

$$\sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \hat{\mathbf{z}}^i) = 0. \quad (5.3.7)$$

Note that this again avoids measuring differences in the image space.

**Self-supervision via compressing augmented samples.** Since we do not know any class label information between samples in the unsupervised settings, the best we can do is to view every sample and its augmentations (say via translation, rotation, occlusion etc) as one “class”—a basic idea behind almost all self-supervised learning methods. In the rate reduction framework, it is natural to compress the features of each sample and its augmentations. In this work, we adopt the standard transformations in SimCLR [CKN+20] and denote such a transformation as  $\tau$ . We denote each augmented sample  $\mathbf{x}_a = \tau(\mathbf{x})$ , and its corresponding feature as  $\mathbf{z}_a = f(\mathbf{x}_a, \theta)$ . For discriminative purposes, we hope the classifier is *invariant* to such transformations. Hence it is natural to enforce that the features  $\mathbf{z}_a$  of all augmentations are the same

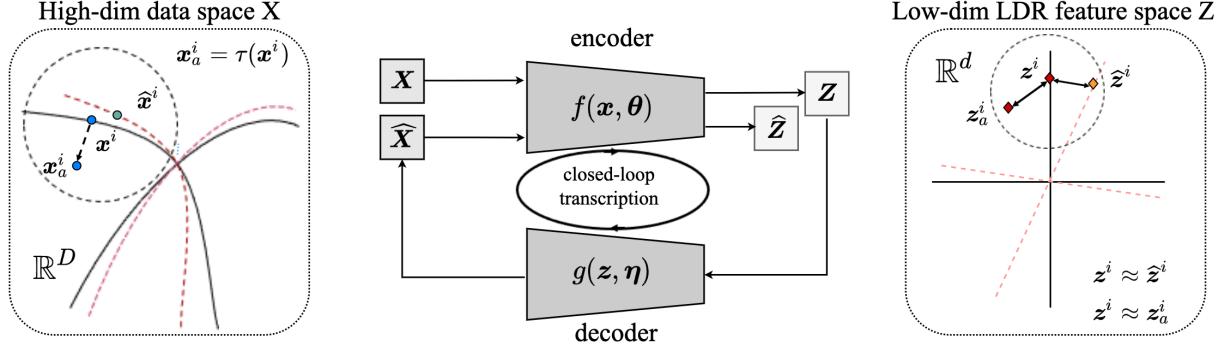


Figure 5.15: **Overall framework** of closed-loop transcription for unsupervised learning. Two additional constraints are imposed on the Binary-CTRL method: 1) self-consistency for sample-wise features  $\mathbf{z}^i$  and  $\hat{\mathbf{z}}^i$ , say  $\mathbf{z}^i \approx \hat{\mathbf{z}}^i$ ; and 2) invariance/similarity among features of augmented samples  $\mathbf{z}^i$  and  $\mathbf{z}_a^i$ , say  $\mathbf{z}^i \approx \mathbf{z}_a^i = f(\tau(\mathbf{x}^i), \theta)$ , where  $\mathbf{x}_a^i = \tau(\mathbf{x}^i)$  is an augmentation of sample  $\mathbf{x}^i$  via some transformation  $\tau(\cdot)$ .

as that  $\mathbf{z}$  of the original sample  $\mathbf{x}$ . This is equivalent to requiring the distance between  $\mathbf{z}$  and  $\mathbf{z}_a$ , measured in terms of rate reduction again, to be zero (or small) for all  $N$  samples:

$$\sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \mathbf{z}_a^i) = 0. \quad (5.3.8)$$

**Unsupervised representation learning via closed-loop transcription.** So far, we know the CTRL-Binary objective  $\Delta R_\epsilon(\mathbf{Z}, \hat{\mathbf{Z}})$  in (5.3.6) helps align the distributions while sample-wise self-consistency (5.3.7) and sample-wise augmentation (5.3.8) help align and compress features associated with each sample. Besides consistency, we also want learned representations are maximally discriminative for different samples (here viewed as different “classes”). Notice that the rate distortion term  $R_\epsilon(\mathbf{Z})$  measures the coding rate (hence volume) of all features.

**Unsupervised CTRL.** Putting these elements together, we propose to learn a representation via the following constrained maximin program, which we refer to as *unsupervised CTRL* (u-CTRL):

$$\begin{aligned} & \max_{\theta} \min_{\eta} R_\epsilon(\mathbf{Z}) + \Delta R_\epsilon(\mathbf{Z}, \hat{\mathbf{Z}}) \\ & \text{subject to } \sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \hat{\mathbf{z}}^i) = 0, \text{ and } \sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \mathbf{z}_a^i) = 0. \end{aligned} \quad (5.3.9)$$

Figure 5.15 illustrate the overall architecture of the closed-loop system associated with this program.

In practice, the above program can be optimized by alternating maximization and minimization between the encoder  $f(\cdot, \theta)$  and the decoder  $g(\cdot, \eta)$ . We adopt the following optimization strategy that works well in practice, which is used for all subsequent experiments on real image datasets:

$$\max_{\theta} R_\epsilon(\mathbf{Z}) + \Delta R_\epsilon(\mathbf{Z}, \hat{\mathbf{Z}}) - \lambda_1 \sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \mathbf{z}_a^i) - \lambda_2 \sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \hat{\mathbf{z}}^i); \quad (5.3.10)$$

$$\min_{\eta} R_\epsilon(\mathbf{Z}) + \Delta R_\epsilon(\mathbf{Z}, \hat{\mathbf{Z}}) + \lambda_1 \sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \mathbf{z}_a^i) + \lambda_2 \sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \hat{\mathbf{z}}^i), \quad (5.3.11)$$

where the constraints  $\sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \hat{\mathbf{z}}^i) = 0$  and  $\sum_{i \in N} \Delta R_\epsilon(\mathbf{z}^i, \mathbf{z}_a^i) = 0$  in (5.3.9) have been converted (and relaxed) to Lagrangian terms with corresponding coefficients  $\lambda_1$  and  $\lambda_2$ .<sup>16</sup>

The above representation is learned without class information. In order to facilitate discriminative or generative tasks, it must be highly structured. It has been verified experimentally that this is indeed the case

<sup>16</sup>Notice that computing the rate reduction terms  $\Delta R$  for all samples or a batch of samples requires computing the expensive  $\log \det$  of large matrices. In practice, from the geometric meaning of  $\Delta R$  for two vectors,  $\Delta R$  can be approximated with an  $\ell^2$  norm or the cosine distance between two vectors.

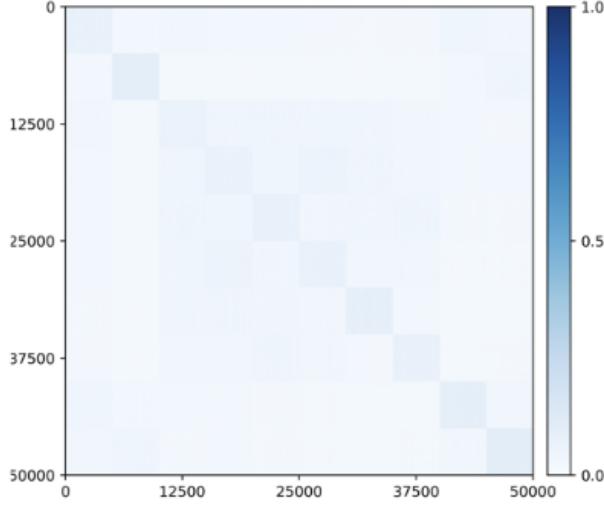


Figure 5.16: Emergence of block-diagonal structures of  $|Z^T Z|$  in the feature space for CIFAR-10.

and u-CTRL demonstrates significant advantages over other incremental or unsupervised learning methods [TDC+24]. We here only illustrate some qualitative results with the experiment on the CIFAR-10 dataset [KNH14], with standard augmentations for self-supervised learning [CKN+20]. One may refer to [TDC+24] for experiments on more and larger datasets and their quantitative evaluations.

As one can see from the experiments, specific and unique structure indeed emerges naturally in the representations learned using u-CTRL: globally, features of images in the same class tend to be clustered well together and separated from other classes, as shown in Figure 5.16; locally, features around individual samples exhibit approximately piecewise linear low-dimensional structures, as shown in Figure 5.17.

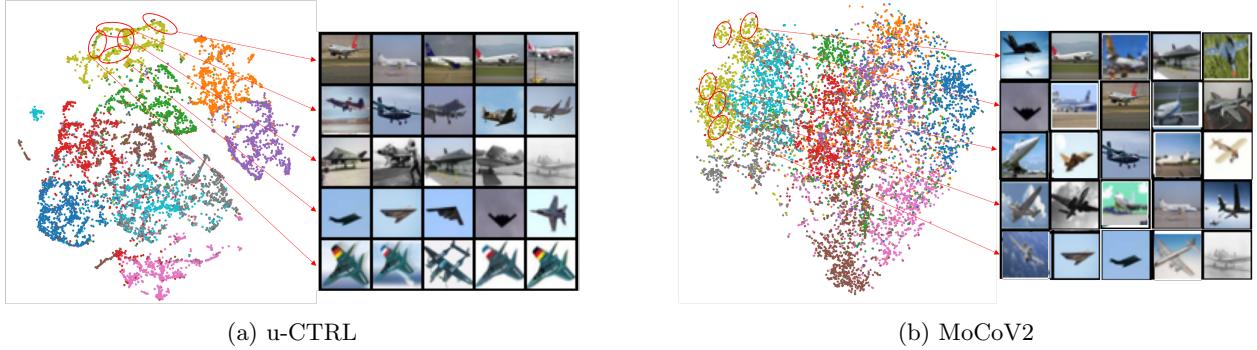


Figure 5.17: t-SNE visualizations of learned features of CIFAR-10 with different models.

**Unsupervised conditional image generation via rate reduction.** The highly-structured feature distribution also suggests that the learned representation can be very useful for generative purposes. For example, we can organize the sample features into meaningful clusters, and model them with low-dimensional (Gaussian) distributions or subspaces. By sampling from these compact models, we can conditionally regenerate meaningful samples from computed clusters. This is known as *unsupervised conditional image generation* [HKJ+21].

To cluster features, we exploit the fact that the rate reduction framework (3.4.12) is inspired by unsupervised clustering via compression [MDH+07b], which provides a principled way to find the membership  $\Pi$ . Concretely, we maximize the same rate reduction objective (3.4.12) over  $\Pi$ , but fix the learned representation  $Z$  instead. We simply view the membership  $\Pi$  as a nonlinear function of the features  $Z$ , say



Figure 5.18: Unsupervised conditional image generation from each cluster of CIFAR-10, using u-CTRL. Images from different rows mean generation from different principal components of each cluster.

$h_{\pi}(\cdot, \xi) : \mathbf{Z} \mapsto \mathbf{\Pi}$  with parameters  $\xi$ . In practice, we model this function with a simple neural network, such as an MLP head right after the output feature  $\mathbf{z}$ . To estimate a “pseudo” membership  $\hat{\mathbf{\Pi}}$  of the samples, we solve the following optimization problem over  $\mathbf{\Pi}$ :

$$\hat{\mathbf{\Pi}} = \arg \max_{\xi} \Delta R_e(\mathbf{Z} | \mathbf{\Pi}(\xi)). \quad (5.3.12)$$

In Figure 5.18, we visualize images generated from the ten unsupervised clusters from (5.3.12). Each block represents one cluster and each row represents one principal component for each cluster. Despite learning and training without labels, the model not only organizes samples into correct clusters, but is also able to preserve statistical diversities within each cluster/class. We can easily recover the diversity within each cluster by computing different principal components and then sample and generate accordingly. While the experiments presented here are somewhat limited in scale, we will explore more direct and powerful methods that utilize the learned data distributions and representations for conditional generation and estimation in the next chapter.

## 5.4 Summary and Notes

Historically, autoencoding has been one of the important drivers of research innovation in neural networks for learning, although the most practically impressive demonstrations of deep learning have probably been in other domains (such as discriminative classification, with AlexNet [KSH12], or generative modeling with GPT architectures [BMR+20]). Works we have featured throughout the chapter, especially the work of [HS06], served as catalysts of research interest in neural networks during times when they were otherwise not prominent in the machine learning research landscape. In modern practice, autoencoders remain core components of many large-scale systems for generating highly structured data such as visual data, speech data, and molecular data, especially the VQ-VAE approach [OVK17], which builds on the variational autoencoder methodology we discussed in Section 5.1.4. The core problem of autoencoding remains of paramount intellectual importance due to its close connection with representation learning, and we anticipate that it will reappear on the radar of practical researchers in the future as efficiency in training and deploying large models continue to become more important.

Materials presented in the second half of this chapter are based on a series of recent work on the topic of closed-loop transcription: [DTL+22], [PPC+23], [TDW+23], and [TDC+24]. In particular, Section 5.2.1 is based on the pioneering work of [DTL+22]. After that, the work of [PPC+23] has provided strong theoretical justifications for the closed-loop framework, at least for an ideal case. Section 5.3.1 and Section 5.3.2 are based on the works of [TDW+23] and [TDC+24], respectively. They demonstrate that the closed-loop framework naturally supports incremental and continuous learning, either in a class-wise or sample-wise setting. The reader may refer to these papers for more technical and experimental details.

**Shallow vs. deep neural networks, for autoencoding and more.** In Section 5.1.2, we discussed Cybenko’s universal approximation theorem and how it states that in principle, a neural network with a single hidden layer (and suitable elementwise nonlinearities) is sufficient to approximate any suitably regular target function. Of course, in practice, the major architectural reason for the dominance of neural

networks in practice has been the refinement of techniques for training *deeper* neural networks. Why is depth necessary? From a fundamental point of view, the issue of depth separations, which construct settings where a deeper neural network can approximate a given class of target functions with exponentially-superior efficiency relative to a shallow network, has been studied at great length in the theoretical literature: examples include [BN20; Tel16; VJO+21]. The ease of training deeper networks in practice has not received as satisfying of an answer from the perspective of theory. ResNets [HZR+16b] represent the pioneering empirical work of making deeper networks more easily trainable, used in nearly all modern architectures in some form. Theoretical studies have focused heavily on trainability of very deep networks, quantified via the initial neural tangent kernel [BGW21; MBD+21], but these studies have not given significant insight into the trainability benefits of deeper networks in middle and late stages of training (but see [YH21] for the root of a line of research attempting to address this).

## 5.5 Exercises and Extensions

*Exercise 5.1* (Conceptual Understanding of Manifold Flattening). Consider data lying on a curved manifold  $\mathcal{M}$  embedded in  $\mathbb{R}^D$  (like a curved surface in 3-dimensional space), as discussed in the manifold flattening subsection of Section 5.1.2. In this exercise, we will describe the basic ingredients of the manifold flattening algorithm from [PPR+24]. A manifold is called flat if it is an open set in Euclidean space (or more generally, an open set in a subspace).

1. Suppose the manifold  $\mathcal{M}$  is a *graph*: this means that  $\mathcal{M} \subset \mathbb{R}^{D_1} \times \mathbb{R}^{D_2}$  (say), and that there is a function  $F : \mathbb{R}^{D_1} \rightarrow \mathbb{R}^{D_2}$  such that

$$\mathcal{M} = \{(\mathbf{x}, F(\mathbf{x}) \mid \mathbf{x} \in \mathbb{R}^{D_1})\}. \quad (5.5.1)$$

Give an integer  $D_3$  and a map  $f : \mathbb{R}^{D_1} \times \mathbb{R}^{D_2} \rightarrow \mathbb{R}^{D_3}$  that flattens  $\mathcal{M}$ , and describe the corresponding (lossless) reconstruction procedure from the flattened representation.

2. Now suppose that  $\mathcal{M}$  is a general smooth manifold. Smooth manifolds have the property that they are locally well-approximated by subspaces near each point. Describe in intuitive terms how to flatten the manifold  $\mathcal{M}$  locally at each point, by relating it to a graph. (The algorithm of [PPR+24] performs a refined version of this local flattening process in a way that allows them to be glued together to form a global flattening.)

*Exercise 5.2* (Reproduce Closed-Loop Transcription). Implement a closed-loop transcription pipeline for representation learning on the CIFAR-10 dataset following the methodology in Section 5.2.1. Reference [DTL+22] for useful hyperparameters and architecture settings. Reproduce the result in Figure 5.8.

# Chapter 6

## Inference with Low-Dimensional Distributions

“Mathematics is the art of giving the same name to different things.”

— Henri Poincaré

In the previous chapters of this book, we have studied how to effectively and efficiently learn a representation for a variable  $\mathbf{x}$  in the world with a distribution  $p(\mathbf{x})$  that has a low-dimensional support in a high-dimensional space. So far, we have mainly developed the methodology for learning representation and autoencoding in a general, distribution or task-agnostic fashion. With such a learned representation, one can already use it to perform some generic and basic tasks such as classification (if the encoding is supervised with the class) and generation of random samples that have the same distribution as the given data (say natural images or natural languages).

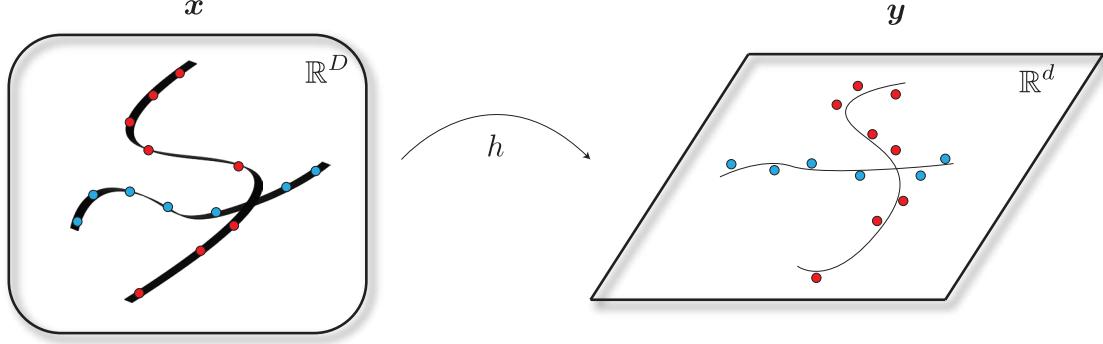
More generally, however, the universality and scalability of the theoretical and computational framework presented in this book have enabled us to learn the distribution of a variety of important real-world high-dimensional data such as natural languages, human poses, natural images, videos, and even 3D scenes. Once the intrinsic rich and low-dimensional structures of these real data can be learned and represented correctly, they start to enable a broad family of powerful, often seemingly miraculous, tasks. Hence, here onwards, we will start to show how to connect and tailor the general methods presented in previous chapters to learn useful representations for specific structured data distributions and for many popular tasks in modern practice of machine intelligence.

### 6.1 Bayesian Inference and Constrained Optimization

**Leveraging Low-dimensionality for Stable and Robust Inference.** Generally speaking, a good representation or autoencoding should enable us to utilize the learned low-dimensional distribution of the data  $\mathbf{x}$  and its representation  $\mathbf{z}$  for various subsequent classification, estimation, and generation tasks under different conditions. As we have alluded to earlier in Chapter 1 Section 1.2.2, the importance of the *low-dimensionality* of the distribution is the key for us to conduct stable and robust inference related to the data  $\mathbf{x}$ , as illustrated by the few simple examples in Figure 1.9, from incomplete, noisy, and even corrupted observations. As it turns out, the very same concept carries over to real-world high-dimensional data whose distributions have a low-dimensional support, such as natural images and languages.

Despite a dazzling variety of applications in the practice of machine learning with data such as languages, images, videos and many other modalities, almost all practical applications can be viewed as a special case of the following inference problem: given an observation  $\mathbf{y}$  that depends on  $\mathbf{x}$ , say

$$\mathbf{y} = h(\mathbf{x}) + \mathbf{w}, \tag{6.1.1}$$



**Figure 6.1: Inference with low-dimensional distributions.** This is the generic picture for this chapter: we have a low-dimensional distribution for  $\mathbf{x} \in \mathbb{R}^D$  (here depicted as a union of two 2-dimensional manifolds in  $\mathbb{R}^3$ ) and a measurement model  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w} \in \mathbb{R}^d$ . We want to infer various things about this model, including the conditional distribution of  $\mathbf{x}$  given  $\mathbf{y}$ , or the conditional expectation  $\mathbb{E}[\mathbf{x} | \mathbf{y}]$ , given various information about the model and (potentially finite) samples of either  $\mathbf{x}$  or  $\mathbf{y}$ .

where  $h(\cdot)$  represents measurements of a part of  $\mathbf{x}$  or its certain observed attributes and  $\mathbf{w}$  represents some measurement noise and even (sparse) corruptions, solve the “inverse problem” of obtaining a most likely estimate  $\hat{\mathbf{x}}(\mathbf{y})$  of  $\mathbf{x}$  or generating a sample  $\hat{\mathbf{x}}$  that is at least consistent with the observation  $\mathbf{y} \approx h(\hat{\mathbf{x}})$ . Figure 6.1 illustrates the general relationship between  $\mathbf{x}$  and  $\mathbf{y}$ .

*Example 6.1* (Image Completion and Text Prediction). The popular natural image completion and natural language prediction are two typical tasks that require us to recover a full data  $\mathbf{x}$  from its partial observations  $\mathbf{y}$ , with parts of  $\mathbf{x}$  masked out and to be completed based on the rest. Figure 6.2 shows some examples of such tasks. In fact, it is precisely these tasks which have inspired how to train modern large models for text generation (such as GPT) and image completion (such as the masked autoencoder) that we will study in greater details later. ■

**Statistical interpretation via Bayes’ rule.** Generally speaking, to accomplish such tasks well, we need to get ahold of the conditional distribution  $p(\mathbf{x} | \mathbf{y})$ . If we had this, then we would be able to find the maximal likelihood estimate (prediction):

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} p(\mathbf{x} | \mathbf{y}); \quad (6.1.2)$$

compute the conditional expectation estimate:

$$\hat{\mathbf{x}} = \mathbb{E}[\mathbf{x} | \mathbf{y}] = \int \mathbf{x} p(\mathbf{x} | \mathbf{y}) d\mathbf{x}; \quad (6.1.3)$$

and sample from the conditional distribution:

$$\hat{\mathbf{x}} \sim p(\mathbf{x} | \mathbf{y}). \quad (6.1.4)$$

Notice that from Bayes’ rule, we have

$$p(\mathbf{x} | \mathbf{y}) = \frac{p(\mathbf{y} | \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}. \quad (6.1.5)$$

For instance, the maximal likelihood estimate can be computed by solving the following (maximal log likelihood) program:

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} [\log p(\mathbf{y} | \mathbf{x}) + \log p(\mathbf{x})], \quad (6.1.6)$$

say via gradient ascent:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \cdot (\nabla_{\mathbf{x}} \log p(\mathbf{y} | \mathbf{x}) + \nabla_{\mathbf{x}} \log p(\mathbf{x})). \quad (6.1.7)$$

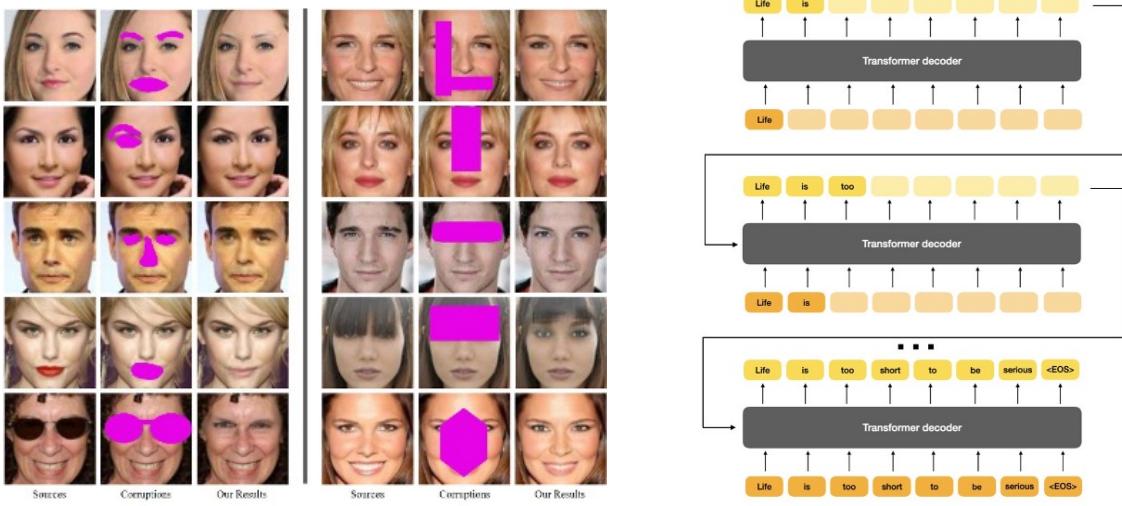


Figure 6.2: Left: image completion. Right: text prediction. In particular, text prediction is the inspiration for the popular Generative Pre-trained Transformer (GPT).

Efficiently computing the conditional distribution  $p(\mathbf{x} | \mathbf{y})$  naturally depends on how we learn and exploit the low-dimensional distribution  $p(\mathbf{x})$  of the data  $\mathbf{x}$  and the observation model  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w}$  that determines the conditional distribution  $p(\mathbf{y} | \mathbf{x})$ .

*Remark 6.1* (End-to-End versus Bayesian). In the modern practices of data-driven machine learning, for certain popular tasks people often directly learn the conditional distribution  $p(\mathbf{x} | \mathbf{y})$  or a (probabilistic) mapping or a regressor. Such a mapping is often modeled by some deep networks and trained end-to-end with sufficient paired samples  $(\mathbf{x}, \mathbf{y})$ . Such an approach is very different from the above Bayesian approach in which both the distribution of  $\mathbf{x} \sim p(\mathbf{x})$  and the (observation) mapping are needed. The benefits of the Bayesian approach is that the learned distribution  $p(\mathbf{x})$  can facilitate many different tasks with very observation models and conditions.

**Geometric interpretation as constrained optimization.** As the support  $\mathcal{S}_{\mathbf{x}}$  of the distribution of  $\mathbf{x}$  is low-dimensional, we may assume that there exists a function  $F$  such that

$$F(\mathbf{x}) = \mathbf{0} \iff \mathbf{x} \in \mathcal{S}_{\mathbf{x}} \quad (6.1.8)$$

such that  $\mathcal{S}_{\mathbf{x}} = F^{-1}(\{\mathbf{0}\})$  is the low-dimensional support of the distribution  $p(\mathbf{x})$ . Geometrically, one natural choice of  $F(\mathbf{x})$  is the ‘‘distance function’’ to the support  $\mathcal{S}_{\mathbf{x}}$ :<sup>1</sup>

$$F(\mathbf{x}) = \min_{\mathbf{x}_p \in \mathcal{S}_{\mathbf{x}}} \|\mathbf{x} - \mathbf{x}_p\|_2. \quad (6.1.9)$$

Now given  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w}$ , to solve for  $\mathbf{x}$ , we can solve the following constrained optimization problem:

$$\max_{\mathbf{x}} -\frac{1}{2} \|h(\mathbf{x}) - \mathbf{y}\|_2^2 \quad \text{s.t.} \quad F(\mathbf{x}) = \mathbf{0}. \quad (6.1.10)$$

Using the method of augmented Lagrange multipliers, we can solve the following unconstrained program:

$$\max_{\mathbf{x}} \left[ -\frac{1}{2} \|h(\mathbf{x}) - \mathbf{y}\|_2^2 + \boldsymbol{\lambda}^\top F(\mathbf{x}) - \frac{\mu}{2} \|F(\mathbf{x})\|_2^2 \right] \quad (6.1.11)$$

<sup>1</sup>Notice that, in reality, we only have discrete samples on the support of the distribution. In the same spirit of continuation, through diffusion or lossy coding studied in Chapter 3, we may approximate the distance function as  $F(\mathbf{x}) \approx \min_{\mathbf{x}_p \in \mathcal{C}_{\mathbf{x}}^\epsilon} \|\mathbf{x} - \mathbf{x}_p\|_2$  where  $\mathcal{S}_{\mathbf{x}}$  is replaced by a covering  $\mathcal{C}_{\mathbf{x}}^\epsilon$  of the samples with  $\epsilon$ -balls.

for some constant Lagrange multipliers  $\boldsymbol{\lambda}$ . This is equivalent to the following program:

$$\max_{\mathbf{x}} \left[ \log \exp \left( -\frac{1}{2} \|h(\mathbf{x}) - \mathbf{y}\|_2^2 \right) + \log \exp \left( -\frac{\mu}{2} \|F(\mathbf{x}) - \boldsymbol{\lambda}/\mu\|_2^2 \right) \right], \quad (6.1.12)$$

where  $\mathbf{c} = \boldsymbol{\lambda}/\mu$  can be viewed as a “mean” for the constraint function. As  $\mu$  becomes large when enforcing the constraint via continuation<sup>2</sup>,  $\|\mathbf{c}\|_2$  becomes increasingly small.

The above program may be interpreted in two different ways. Firstly, one may view the first term as the conditional probability of  $\mathbf{y}$  given  $\mathbf{x}$ , and the second term as a probability density for  $\mathbf{x}$ :

$$p(\mathbf{y} | \mathbf{x}) \propto \exp \left( -\frac{1}{2} \|h(\mathbf{x}) - \mathbf{y}\|_2^2 \right), \quad p(\mathbf{x}) \propto \exp \left( -\frac{\mu}{2} \|F(\mathbf{x}) - \mathbf{c}\|_2^2 \right). \quad (6.1.13)$$

Hence, solving the constrained optimization for the inverse problem is equivalent to conducting Bayes inference with the above probability densities. Hence solving the above program (6.1.12) via gradient ascent is equivalent to the above maximal likelihood estimate (6.1.7), in which the gradient takes the form:

$$\nabla_{\mathbf{x}} \log p(\mathbf{y} | \mathbf{x}) + \nabla_{\mathbf{x}} \log p(\mathbf{x}) = \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x})(\mathbf{y} - h(\mathbf{x})) + \mu \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x})(\mathbf{c} - F(\mathbf{x})), \quad (6.1.14)$$

where  $\frac{\partial h}{\partial \mathbf{x}}(\mathbf{x})$  and  $\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x})$  are the Jacobians of  $h(\mathbf{x})$  and  $F(\mathbf{x})$ , respectively.

Secondly, notice that the above program (6.1.12) is equivalent to:

$$\min_{\mathbf{x}} \frac{1}{2} \|h(\mathbf{x}) - \mathbf{y}\|_2^2 + \frac{\mu}{2} \|F(\mathbf{x}) - \boldsymbol{\lambda}/\mu\|_2^2, \quad (6.1.15)$$

Due to the conspicuous quadratic form of the two terms, they can also be interpreted as certain “energy” functions. Such a formulation is often referred to as “Energy Minimization” in the machine learning literature.

**Several representative practical settings for inference.** In practice, however, initial information about the distributions of  $\mathbf{x}$  and the relationship between  $\mathbf{x}$  and  $\mathbf{y}$  can be given in many different ways or forms. In general, they can mostly be categorized into four cases, which are, conceptually, increasingly more challenging:

- *Case 1:* Both a model for the distribution of  $\mathbf{x}$  and the observation model  $\mathbf{y} = h(\mathbf{x}) (+\mathbf{w})$  is known, even with an analytical form. This is typically the case for many classic signal processing problems, such as signal denoising, the sparse vector recovery problem we saw in Chapter 2 and the low-rank matrix recovery problem to be introduced below.
- *Case 2:* We do not have a model for the distribution but only samples  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  of  $\mathbf{x}$ , and the observation model  $\mathbf{y} = h(\mathbf{x}) (+\mathbf{w})$  is known.<sup>3</sup> A model for the distribution  $p(\mathbf{x})$  of  $\mathbf{x}$  needs to be learned, and subsequently the conditional distribution  $p(\mathbf{x} | \mathbf{y})$ . Natural image completion or natural language completion (e.g., BERT and GPT) are typical examples for this class of problems.
- *Case 3:* We only have the paired samples:  $(\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$  of the two variables  $(\mathbf{x}, \mathbf{y})$ . The distributions of  $\mathbf{x}$  and  $\mathbf{y}$  and their relationship  $h(\cdot)$  need to be learned from these paired sample data. For example, given many images and their captions, learning to conduct text-conditioned image generation is one such problem.
- *Case 4:* We only have the samples  $\mathbf{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$  of the observations  $\mathbf{y}$ , and the observation model  $h(\cdot)$  needs to be known, at least in some parametric family  $h(\cdot, \boldsymbol{\theta})$ . The distribution  $p(\mathbf{x})$  and  $p(\mathbf{x} | \mathbf{y})$  need to be learned from  $\hat{\mathbf{x}}$ , estimated from  $\mathbf{Y}$ . For example, learning to render a new view from a sequence of calibrated or uncalibrated views is one such problem.

<sup>2</sup>In the same spirit of continuation in Chapter 3 where we obtained better approximations of our distribution by sending  $\varepsilon \rightarrow 0$ , here we send  $\mu \rightarrow \infty$ . Larger values of  $\mu$  will constrain  $F$  to take smaller and smaller values at optimum, meaning that the optimum lies within a smaller and smaller neighborhood of the support  $S_{\mathbf{x}}$ . Interestingly, the theory of Lagrange multipliers hints that, under certain benign conditions on  $F$  and other terms in the objective, we only need to make  $\mu$  large enough in order to ensure  $F(\mathbf{x}) = \mathbf{0}$  at optimum, meaning that at *finite* penalty we get *perfect* approximation of the support. In general, we should have the intuition that  $\mu$  plays the same role as  $\epsilon^{-1}$ .

<sup>3</sup>In the literature, this setting is sometimes referred to as the *empirical Bayesian inference*.

In this chapter, we will discuss general approaches to learn the desired distributions and solve the associated conditional estimation or generation for these cases, typically with a representative problem. Throughout the chapter, you should keep Figure 6.1 in mind.

## 6.2 Conditional Inference with a Known Data Distribution

Notice that in the setting we have discussed in previous Chapters, the autoencoding network is trained to reconstruct a set of samples of the random vector  $\mathbf{x}$ . This would allow us to regenerate samples from the learned (low-dimensional) distribution. In practice, the low-dimensionality of the distribution, once given or learned, can be exploited for stable and robust recovery, completion, or prediction tasks. That is, under rather mild conditions, one can recover  $\mathbf{x}$ , from highly compressive, partial, noisy or even corrupted measures of  $x$  of the kind:

$$\mathbf{y} = h(\mathbf{x}) + \mathbf{w}, \quad (6.2.1)$$

where  $\mathbf{y}$  is typically an observation of  $\mathbf{x}$  that is of much lower dimension than  $\mathbf{x}$  and  $\mathbf{w}$  can be random noise or even sparse gross corruptions. This is a class of problems that have been extensively studied in the classical signal processing literature, for low-dimensional structures such as sparse vectors, low-rank matrices, and beyond. Interested readers may see [WM22] for a complete exposition of this topic.

Here to put the classic work in a more general modern setting, we illustrate the basic idea and facts through the arguably simplest task of data (and particularly image) completion. That is, we consider the problem of recovering a sample  $\mathbf{x}$  when parts of it are missing (or even corrupted). We want to recover or predict the rest of  $\mathbf{x}$  from observing only a fraction of it:

$$f : \mathcal{P}_\Omega(\mathbf{x}) \mapsto \hat{\mathbf{x}}, \quad (6.2.2)$$

where  $\mathcal{P}_\Omega(\cdot)$  represents a masking operation (see Figure 6.3 for an example).

In this section and the next, we will study the completion task under two different scenarios: One is when the distribution of the data  $\mathbf{x}$  of interest is already given *a priori*, even in a certain analytical form. This is the case that prevails in classic signal processing where the structures of the signals are assumed to be known, for example, band-limited, sparse or low-rank. The other is when only raw samples of  $\mathbf{x}$  are available and we need to learn the low-dimensional distribution from the samples in order to solve the completion task well. This is the case for the tasks of natural image completion or video frame prediction. As a precursor to the rest of the chapter, we start with the simplest case of image completion: when the image to be completed can be well modeled as a low-rank matrix. We will move on to increasingly more general cases and more challenging settings later.

**Low-rank matrix completion.** The low-rank *matrix completion* problem is a classical problem for data completion when its distribution is low-dimensional and known. Consider a random sample of a matrix  $\mathbf{X}_o = [\mathbf{x}_1, \dots, \mathbf{x}_n] \in \mathbb{R}^{m \times n}$  from the space of all matrices of rank  $r$ . In general, we assume the rank of the matrix is

$$\text{rank}(\mathbf{X}_o) = r < \min\{m, n\}. \quad (6.2.3)$$

So it is clear that locally the intrinsic dimension of the space of all matrix of rank  $r$  is much lower than the ambient space  $mn$ .

Now, let  $\Omega$  indicate a set indices of observed entries of the matrix  $\mathbf{X}_o$ . Let the observed entries to be:

$$\mathbf{Y} = \mathcal{P}_\Omega(\mathbf{X}_o). \quad (6.2.4)$$

The remaining entries supported on  $\Omega^c$  are unobserved or missing. The problem is whether we can recover from  $\mathbf{Y}$  the missing entries of  $\mathbf{X}$  correctly and efficiently. Figure 6.3 shows one example of completing such a matrix.

Notice that the fundamental reason why such a matrix can be completed is because columns and rows of the matrix are highly correlated and they all lie on a low-dimensional subspace. For the example shown in Figure 6.3, the dimension or the rank of the matrix completed is only two. Hence the fundamental idea



Figure 6.3: Illustration of completing an image as low-rank matrix with some entries masked or corrupted. Left: the masked/corrupted image  $\mathbf{Y}$ ; middle: the mask  $\Omega$ ; right: the completed image  $\hat{\mathbf{X}}$ .

to recover such a matrix is to seek a matrix that has the lowest rank among all matrices that have entries agreeing with the observed ones:

$$\min_{\mathbf{X}} \text{rank}(\mathbf{X}) \quad \text{subject to} \quad \mathbf{Y} = \mathcal{P}_{\Omega}(\mathbf{X}). \quad (6.2.5)$$

This is known as the *low-rank matrix completion* problem. See [WM22] for a full characterization of the space of all low-rank matrices. As the rank function is discontinuous and rank minimization is in general an NP-hard problem, we like to relax it with something easier to optimize.

Based on our knowledge about compression from Chapter 3, we could promote the low-rankness of the recovered matrix  $\mathbf{X}$  by enforcing the lossy coding rate (or the volume spanned by  $\mathbf{X}$ ) of the data in  $\mathbf{X}$  to be small:

$$\min R_{\epsilon}(\mathbf{X}) = \frac{1}{2} \log \det (\mathbf{I} + \alpha \mathbf{X} \mathbf{X}^{\top}) \quad \text{subject to} \quad \mathbf{Y} = \mathcal{P}_{\Omega}(\mathbf{X}). \quad (6.2.6)$$

The problem can viewed as a continuous relaxation of the above low-rank matrix completion problem (6.2.5) and it can be solved via gradient descent. One can show that the gradient descent operator for the log det objective is precisely minimizing a close surrogate of the rank of the matrix  $\mathbf{X} \mathbf{X}^{\top}$ .

The rate distortion function is a nonconvex function, and its gradient descent does not always guarantee finding the globally optimal solution. Nevertheless, since the underlying structure sought for  $\mathbf{X}$  is piecewise linear, the rank function admits a rather effective convex relaxation: the nuclear norm—the sum of all singular values of the matrix  $\mathbf{X}$ . As shown in the compressive sensing literature, under fairly broad conditions,<sup>4</sup> the matrix completion problem (6.2.5) can be effectively solved by the following convex program:

$$\min \|\mathbf{X}\|_* \quad \text{subject to} \quad \mathbf{Y} = \mathcal{P}_{\Omega}(\mathbf{X}), \quad (6.2.7)$$

where the nuclear norm  $\|\mathbf{X}\|_*$  is the sum of singular values of  $\mathbf{X}$ . In practice, we often convert the above constrained convex optimization program to an unconstrained one:

$$\min \|\mathbf{X}\|_* + \lambda \|\mathbf{Y} - \mathcal{P}_{\Omega}(\mathbf{X})\|_F^2, \quad (6.2.8)$$

for some properly chosen  $\lambda > 0$ . Interested readers may refer to [WM22] for how to develop algorithms that can solve the above programs efficiently and effectively. Figure 6.3 shows a real example in which the matrix  $\hat{\mathbf{X}}$  is actually recovered by solving the above program.

**Further extensions.** It has been shown that images (or more accurately textures) and 3D scenes with low-rank structures can be very effectively completed via solving optimization programs of the above kind, even if there is additional corruption and distortion [LRZ+12; YZB+23; ZLG+10]:

$$\mathbf{Y} \circ \tau = \mathbf{X}_o + \mathbf{E}, \quad (6.2.9)$$

---

<sup>4</sup>Typically, such conditions specify the necessary and sufficient amount of entries needed for the completion to be computationally feasible. These conditions have been systematically characterized in [WM22].

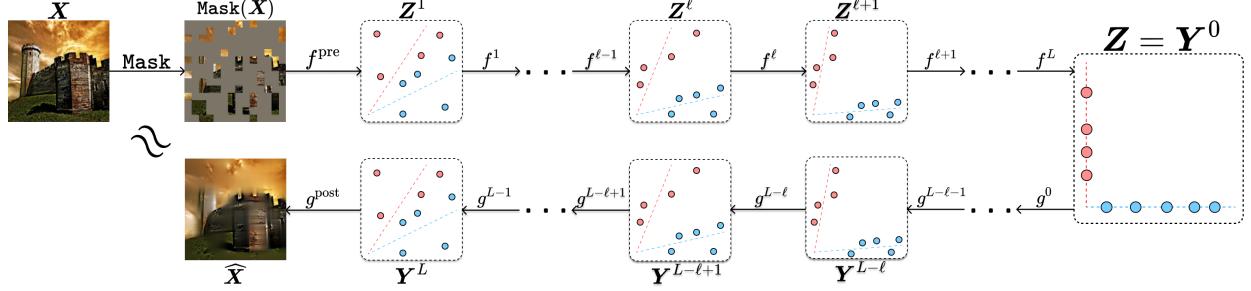


Figure 6.4: **Diagram of the overall (masked) autoencoding process.** The (image) token representations are transformed iteratively towards a parsimonious (e.g., compressed and sparse) representation by each encoder layer  $f^\ell$ . Furthermore, such representations are transformed back to the original image by the decoder layers  $g^\ell$ . Each encoder layer  $f^\ell$  is meant to be (partially) inverted by a corresponding decoder layer  $g^{L-\ell}$ .

where  $\tau$  is some unknown nonlinear distortion of the image and  $\mathbf{E}$  is an unknown matrix that models some (sparse) occlusion and corruption. Again, interested readers may refer to [WM22] for a more detailed account.

## 6.3 Conditional Inference with a Learned Data Representation

In the previous subsection, the reason we can infer  $\mathbf{x}$  from the partial observation  $\mathbf{y}$  is because (support of) the distribution of  $\mathbf{X}$  is known or specified *a priori*, say as the set of all low-rank matrices. For many practical dataset, we do not have its distribution in an analytical form as the low-rank matrices, say the set to of all natural images. Nevertheless, if we have sufficient samples of the data  $\mathbf{x}$ , we should be able to learn its low-dimensional distribution first and leverage it for future inference tasks based on an observation  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w}$ . In this section, we assume the observation model  $h(\cdot)$  is given and known. We will study the case when  $h(\cdot)$  is not explicitly given in the next section.

### 6.3.1 Image Completion with Masked Auto-Encoding

For a general image  $\mathbf{X}$  such as the one shown on the left of Figure 6.4, we can no longer view it as a low-rank matrix. However, humans still demonstrate remarkable ability to complete a scene and recognize familiar objects despite severe occlusion. This suggests that our brain has learned the low-dimensional distribution of natural images and can use it for completion, hence recognition. However, the distribution of all natural images is not as simple as a low-dimensional linear subspace. Hence a natural question is whether we can learn the more sophisticated distribution of natural images and use it to perform image completion?

One empirical approach to the image completion task is to find an encoding and decoding scheme by solving the following *masked autoencoding* (MAE) program that minimizes the reconstruction loss:

$$\min_{f,g} L_{\text{MAE}}(f,g) \doteq \mathbb{E}_{f,g} [\|(g \circ f)(\mathcal{P}_\Omega(\mathbf{X})) - \mathbf{X}\|_2^2]. \quad (6.3.1)$$

Unlike the matrix completion problem which has a simple underlying structure, we should no longer expect that the encoding and decoding mappings admit simple closed forms or the program can be solved by explicit algorithms.

For a general natural image, we can no longer assume its columns or rows are sampled from a low-dimensional subspace or a low-rank Gaussian. However, it is reasonable to assume that the image consists of multiple regions. Image patches in each region are similar and can be modeled as one (low-rank) Gaussian or subspace. Hence, to exploit the low-dimensionality of the distribution, the objective of the encoder  $f$  is to transform  $\mathbf{X}$  to a representation  $\mathbf{Z}$ :

$$f : \mathbf{X} \mapsto \mathbf{Z} \quad (6.3.2)$$

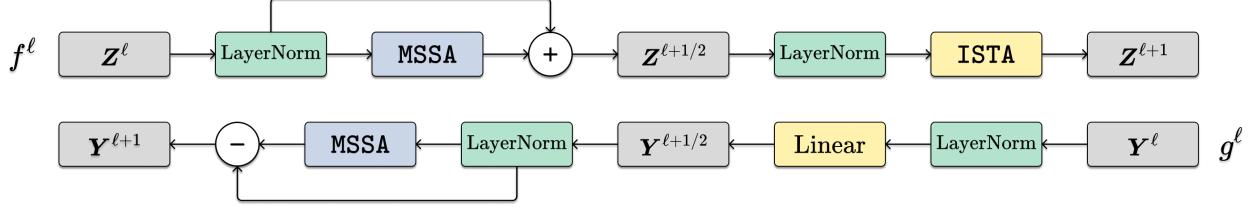


Figure 6.5: **Diagram of each encoder layer (top) and decoder layer (bottom).** Notice that the two layers are highly anti-parallel — each is constructed to do the operations of the other in reverse order. That is, in the decoder layer  $g^\ell$ , the ISTA block of  $f^{L-\ell}$  is partially inverted first using a linear layer, then the MSSA block of  $f^{L-\ell}$  is reversed; this order unravels the transformation done in  $f^{L-\ell}$ .



Figure 6.6: **Autoencoding visualizations of CRATE-Base and ViT-MAE-Base [HCX+22] with 75% patches masked.** We observe that the reconstructions from CRATE-Base are on par with the reconstructions from ViT-MAE-Base, despite using  $< 1/3$  of the parameters.

such that the distribution of  $\mathbf{Z}$  can be well modeled as a mixture of subspaces, say  $\{\mathbf{U}_{[K]}\}$ , such that the rate reduction is maximized while the sparsity is minimized:

$$\mathbb{E}_{\mathbf{Z}=f(\mathbf{X})}[\Delta R_\epsilon(\mathbf{Z} \mid \mathbf{U}_{[K]}) - \lambda \|\mathbf{Z}\|_0] = \mathbb{E}_{\mathbf{Z}=f(\mathbf{X})}[R_\epsilon(\mathbf{Z}) - R_\epsilon^c(\mathbf{Z} \mid \mathbf{U}_{[K]}) - \lambda \|\mathbf{Z}\|_0], \quad (6.3.3)$$

where the functions  $R_\epsilon(\cdot)$  and  $R_\epsilon^c(\cdot)$  are defined in (4.2.2) and (4.2.3), respectively.

As we have shown in the previous Chapter 4, the encoder  $f$  that minimizes the above objective can be constructed by a sequence of transformer-like operators. As shown in the work of [PBW+24], the decoder  $g$  can be viewed and hence constructed explicitly as the inverse process of the encoder  $f$ . Figure 6.5 illustrates the overall architectures of both the encoder and the corresponding decoder at each layer. The parameters of the encoder  $f$  and decoder  $g$  can be learned by optimizing the reconstruction loss (6.3.1) via gradient descent.

Figure 6.6 shows some representative results of the so-designed masked auto-encoder. More implementation details and results of the masked autoencoder for natural image completion can be found in Chapter 7.

### 6.3.2 Conditional Sampling with Measurement Matching

The above (masked) autoencoding problem aims to generate a sample image that is consistent with certain observations or conditions. But let us examine the approach more closely: Given the visual part of an image  $\mathbf{X}_v = \mathcal{P}_\Omega(\mathbf{X})$ , we try to estimate the masked part  $\mathbf{X}_m = \mathcal{P}_{\Omega^c}(\mathbf{X})$ . For realizations  $(\Xi_v, \Xi_m)$  of the random variable  $\mathbf{X} = (\mathbf{X}_v, \mathbf{X}_m)$ , let

$$p_{\mathbf{X}_m | \mathbf{X}_v}(\Xi_m \mid \Xi_v)$$

be the conditional distribution of  $\mathbf{X}_m$  given  $\mathbf{X}_v$ . It is easy to show that the optimal solution to the MAE formulation (6.3.1) is given by the conditional expectation:

$$\arg \min_{h=g \circ f} L_{\text{MAE}}(h) = \Xi_v \mapsto \Xi_v + \mathbb{E}[\mathbf{X}_m \mid \mathbf{X}_v = \Xi_v]. \quad (6.3.4)$$



Figure 6.7: **Sampling visualizations from models trained via ambient diffusion [DSD+23b] with 80% of the pixels masked.** Using a similar ratio of masked pixels as in Figure 6.6, the ambient diffusion sampling algorithm recovers a much sharper image than the blurry image recovered by the MAE-based method. The former method samples from the distribution of natural images, while the latter approximates the conditional expectation (i.e., average) of this distribution given the observation; this averaging causes the blurriness.

In general, however, this expectation may not even be on the low-dimensional distribution of natural images! This partially explains why some of the recovered patches in Figure 6.6 are a little blurry.

For many practical purposes, we would like to learn (a representation of) the conditional distribution  $p_{\mathbf{X}_m|\mathbf{X}_v}$ , or equivalently  $p_{\mathbf{X}|\mathbf{X}_v}$ , and then get a clear (most likely) sample from this distribution directly. Notice that, when the distribution of  $\mathbf{X}$  is low-dimensional, it is possible that if a sufficient part of  $\mathbf{X}$ ,  $\mathbf{X}_v$ , is observed, it fully determines  $\mathbf{X}$  hence the missing part  $\mathbf{X}_m$ . In other words, the distribution  $p_{\mathbf{X}|\mathbf{X}_v}$  is a generalized function—if  $\mathbf{X}$  is fully determined by  $\mathbf{X}_v$  it is the delta function, and more generally one of its exotic cousins.

Hence, instead of solving the completion task as a conditional estimation problem, we should address it as a conditional sampling problem. To that end, we should first learn the (low-dimensional) distribution of all natural images  $\mathbf{X}$ . If we have sufficient samples of natural images, we can learn the distribution via a denoising process  $\mathbf{X}_t$  described in Chapter 3. Then the problem of recovering  $\mathbf{X}$  from its partial observation  $\mathbf{Y} = \mathcal{P}_\Omega(\mathbf{x}) + \mathbf{w}$  becomes a conditional generation problem – to sample the distribution conditioned on the observation.

**General linear measurements.** In fact, we may even consider recovering  $\mathbf{X}$  from a more general linear observation model:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}_0, \quad \mathbf{X}_t = \mathbf{X}_0 + \sigma_t \mathbf{G}, \quad (6.3.5)$$

where  $\mathbf{A}$  is a linear operator on matrix space<sup>5</sup> and  $\mathbf{G} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . The masking operator  $\mathcal{P}_\Omega(\cdot)$  in the image completion task is one example of such a linear model. Then it has been shown by [DSD+23a] that

$$\hat{\mathbf{X}}_* = \arg \min_{\hat{\mathbf{X}}} \mathbb{E}[\|\mathbf{A}(\hat{\mathbf{X}}(\mathbf{A}\mathbf{X}_t, \mathbf{A}) - \mathbf{X}_0)\|^2] \quad (6.3.6)$$

satisfies the condition that:

$$\mathbf{A}\hat{\mathbf{X}}_*(\mathbf{A}(\mathbf{X}_t), \mathbf{A}) = \mathbf{A}\mathbb{E}[\mathbf{X}_0 | \mathbf{A}\mathbf{X}_t, \mathbf{A}]. \quad (6.3.7)$$

Notice that in the special case when  $\mathbf{A}$  is of full column rank, we have  $\mathbb{E}[\mathbf{X}_0 | \mathbf{A}\mathbf{X}_t, \mathbf{A}] = \mathbb{E}[\mathbf{X}_0 | \mathbf{X}_t]$ . Hence, in the more general case, it has been suggested by [DSD+23a] that one could still use the so obtained  $\mathbb{E}[\mathbf{X}_0 | \mathbf{A}(\mathbf{X}_t), \mathbf{A}]$  to replace the  $\mathbb{E}[\mathbf{X}_0 | \mathbf{X}_t]$  in the normal denoising process for  $\mathbf{X}_t$ :

$$\mathbf{X}_{t-s} = \gamma_t \mathbf{X}_t + (1 - \gamma_t) \mathbb{E}[\mathbf{X}_0 | \mathbf{A}\mathbf{X}_t, \mathbf{A}]. \quad (6.3.8)$$

This usually works very well in practice, say for many image restoration tasks, as shown in [DSD+23a]. Compared to the blurry images recovered from MAE, the images recovered by the above method are much sharper as it leverages a learned distribution of natural image and samples a (sharp) image from the distribution that is consistent with the measurement, as shown in Figure 6.7 (cf Figure 6.6).

<sup>5</sup>i.e., if we imagine unrolling  $\mathbf{X}$  into a long vector then  $\mathbf{A}$  takes the role of a matrix on  $\mathbf{X}$ -space.

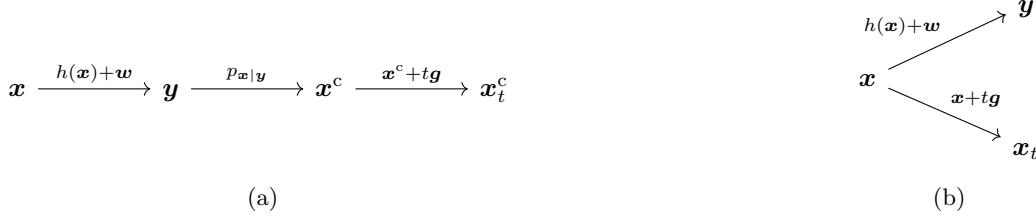


Figure 6.8: Statistical dependency diagrams for the conditional sampling process. **Left:** In a direct (conceptual) application of the diffusion-denoising scheme we have developed in Chapter 3 to conditional sampling, we use samples from the posterior  $p_{\mathbf{x}|\mathbf{y}}$  to train denoisers directly on the posterior at different noise levels, then use them to generate new samples. In practice, however, we do not normally have direct samples from the posterior, but rather paired samples  $(\mathbf{x}, \mathbf{y})$  from the joint. **Right:** It turns out that it suffices to have only noisy observations of  $\mathbf{x}$  to realize the denoisers corresponding to  $p_{\mathbf{x}_t^c|\mathbf{x}^c}$ : this follows from conditional independence of  $\mathbf{x}_t$  and  $\mathbf{y}$  given  $\mathbf{x}$ . It implies that  $p_{\mathbf{x}_t^c} = p_{\mathbf{x}_t|\mathbf{y}}$ , which gives a score function for denoising that consists of the unconditional score function, plus a correction term that enforces measurement consistency.

**General nonlinear measurements.** To generalize the above (image) completion problems and make things more rigorous, we may consider that a random vector  $\mathbf{x} \sim p$  is partially observed through a more general observation function:

$$\mathbf{y} = h(\mathbf{x}) + \mathbf{w}, \quad (6.3.9)$$

where  $\mathbf{w}$  usually stands for some random measurement noise, say of a Gaussian distribution  $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ . It is easy to see that, for so related  $\mathbf{x}$  and  $\mathbf{y}$ , their joint distribution  $p(\mathbf{x}, \mathbf{y})$  is naturally nearly degenerate if the noise  $\mathbf{w}$  is small. To a large extent, we may view  $p(\mathbf{x}, \mathbf{y})$  as a noisy version of a hypersurface defined by the function  $\mathbf{y} = h(\mathbf{x})$  in the joint space  $(\mathbf{x}, \mathbf{y})$ . Practically speaking, we will consider a setting more akin to masked autoencoding than to pure matrix completion, where we always have access to a corresponding clean sample  $\mathbf{x}$  for every observation  $\mathbf{y}$  we receive.<sup>6</sup>

Like image/matrix completion, we are often faced with a setting where  $\mathbf{y}$  denotes a degraded or otherwise “lossy” observation of the input  $\mathbf{x}$ . This can manifest in quite different forms. For example, in various scientific or medical imaging problems, the measured data  $\mathbf{y}$  may be a compressed and corrupted observation of the underlying data  $\mathbf{x}$ ; whereas in 3D vision tasks,  $\mathbf{y}$  may represent an image captured by a camera of a physical object with an unknown (low-dimensional) pose  $\mathbf{x}$ . Generally, by virtue of mathematical modeling (and, in some cases, co-design of the measurement system), we know  $h$  and can evaluate it on any input, and we can exploit this knowledge to help reconstruct and sample  $\mathbf{x}$ .

At a technical level, we want the learned representation of the data to facilitate us to sample the conditional distribution  $p_{\mathbf{x}|\mathbf{y}}$ , also known as the posterior, effectively and efficiently. More precisely, write  $\boldsymbol{\nu}$  to denote a realization of the random variable  $\mathbf{y}$ . We want to generate samples  $\hat{\mathbf{x}}$  such that:

$$\hat{\mathbf{x}} \sim p_{\mathbf{x}|\mathbf{y}}(\cdot \mid \mathbf{y} = \boldsymbol{\nu}). \quad (6.3.10)$$

Recall that in Section 3.2, we have developed a natural and effective way to produce *unconditional* samples of the data distribution  $p$ . The ingredients are the denoisers  $\bar{\mathbf{x}}^*(t, \xi) = \mathbb{E}[\mathbf{x} \mid \mathbf{x}_t = \xi]$ , or their learned approximations  $\bar{\mathbf{x}}_\theta(t, \xi)$ , for different levels of noisy observations  $\mathbf{x}_t = \mathbf{x} + t\mathbf{g}$  (and  $\xi$  for their realizations) under Gaussian noise  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , and  $t \in [0, T]$  with a choice of times  $0 = t_1 < \dots < t_L = T$  at which to perform the iterative denoising, starting from  $\hat{\mathbf{x}}_{t_L} \sim \mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$  (recall Equation (3.2.66)).<sup>7</sup> We could directly apply this scheme to generate samples from the posterior  $p_{\mathbf{x}|\mathbf{y}}$  if we had access to a dataset of samples  $\mathbf{x}^c \sim p_{\mathbf{x}|\mathbf{y}}(\cdot \mid \boldsymbol{\nu})$  for each realization  $\boldsymbol{\nu}$  of  $\mathbf{y}$ , by generating noisy observations  $\mathbf{x}_t^c$  and training denoisers to approximate  $\mathbb{E}[\mathbf{x}^c \mid \mathbf{x}_t^c = \cdot, \mathbf{y} = \boldsymbol{\nu}]$ , the mean of the posterior under the noisy observation

<sup>6</sup>In some more specialized applications, in particular in scientific imaging, it is of interest to be able to generate samples from the posterior  $p_{\mathbf{x}|\mathbf{y}}$  without access to any clean/ground-truth samples of  $\mathbf{x}$ . We give a brief overview of methods for this setting in the end-of-chapter notes.

<sup>7</sup>Recall from our discussion in Section 3.2.2 that a few small improvements to this basic iterative denoising scheme are sufficient to bring competitive practical performance. For clarity as we develop conditional sampling, we will focus here on the simplest instantiation.

(see Figure 6.8(a)). However, performing this resampling given only paired samples  $(\mathbf{x}, \mathbf{y})$  from the joint distribution (say by binning the samples over values of  $\mathbf{y}$ ) requires prohibitively many samples for high-dimensional data, and alternate approaches explicitly or implicitly rely on density estimation, which similarly suffers from the curse of dimensionality.

Fortunately, it turns out that this is not necessary. Consider the alternate statistical dependency diagram in Figure 6.8(b), which corresponds to the random variables in the usual denoising-diffusion process, together with the measurement  $\mathbf{y}$ . Because our assumed observation model (6.3.9) implies that  $\mathbf{x}_t$  and  $\mathbf{y}$  are independent conditioned on  $\mathbf{x}$ , we have for any realization  $\boldsymbol{\nu}$  of  $\mathbf{y}$

$$\begin{aligned} p_{\mathbf{x}_t^c|\mathbf{y}}(\cdot | \boldsymbol{\nu}) &= \int \underbrace{p_{\mathbf{x}_t^c|\mathbf{x}^c}(\cdot | \boldsymbol{\xi})}_{=\mathcal{N}(\boldsymbol{\xi}, t^2 \mathbf{I})} \cdot \underbrace{p_{\mathbf{x}^c|\mathbf{y}}(\boldsymbol{\xi} | \boldsymbol{\nu})}_{=p_{\mathbf{x}|\mathbf{y}}} d\boldsymbol{\xi} \\ &= \int p_{\mathbf{x}_t|\mathbf{x}, \mathbf{y}}(\cdot | \boldsymbol{\xi}, \boldsymbol{\nu}) \cdot p_{\mathbf{x}|\mathbf{y}}(\boldsymbol{\xi} | \boldsymbol{\nu}) d\boldsymbol{\xi} \\ &= \int p_{\mathbf{x}_t, \mathbf{x}|\mathbf{y}}(\cdot, \boldsymbol{\xi} | \boldsymbol{\nu}) d\boldsymbol{\xi} \\ &= p_{\mathbf{x}_t|\mathbf{y}}(\cdot | \boldsymbol{\nu}). \end{aligned} \quad (6.3.11)$$

Above, the first line recognizes an equivalence between the distributions arising in Figure 6.8 (a,b); the second line applies this together with conditional independence of  $\mathbf{x}_t$  and  $\mathbf{y}$  given  $\mathbf{x}$ ; the third line uses the definition of conditional probability; and the final line marginalizes over  $\mathbf{x}$ . Thus, the denoisers from the conceptual posterior sampling process are equal to  $\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \cdot, \mathbf{y} = \boldsymbol{\nu}]$ , which we can learn solely from paired samples  $(\mathbf{x}, \mathbf{y})$ , and by Tweedie's formula (Theorem 3.3), we can express these denoisers in terms of the score function of  $p_{\mathbf{x}_t|\mathbf{y}}$ , which, by Bayes' rule, satisfies

$$p_{\mathbf{x}_t|\mathbf{y}}(\boldsymbol{\xi} | \boldsymbol{\nu}) = \frac{p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \boldsymbol{\xi}) p_{\mathbf{x}_t}(\boldsymbol{\xi})}{p_{\mathbf{y}}(\boldsymbol{\nu})}. \quad (6.3.12)$$

Recall that the density of  $\mathbf{x}_t$  is given by  $p_t = \varphi_t * p$ , where  $\varphi_t$  denotes the standard Gaussian density with zero mean and covariance  $t^2 \mathbf{I}$  and  $*$  denotes convolution. This is nothing but the *unconditional score function* obtained from the standard diffusion training that we developed in Section 3.2! The conditional score function then satisfies, for any realization  $(\boldsymbol{\xi}, \boldsymbol{\nu})$  of  $(\mathbf{x}_t, \mathbf{y})$ ,

$$\nabla_{\boldsymbol{\xi}} \log p_{\mathbf{x}_t|\mathbf{y}}(\boldsymbol{\xi} | \boldsymbol{\nu}) = \underbrace{\nabla_{\boldsymbol{\xi}} \log p_t(\boldsymbol{\xi})}_{\text{score matching}} + \underbrace{\nabla_{\boldsymbol{\xi}} \log p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \boldsymbol{\xi})}_{\text{measurement matching}}, \quad (6.3.13)$$

giving (by Tweedie's formula) our proposed denoisers as

$$\begin{aligned} \mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}, \mathbf{y} = \boldsymbol{\nu}] &= \boldsymbol{\xi} + t^2 \nabla_{\boldsymbol{\xi}} \log p_t(\boldsymbol{\xi}) + t^2 \nabla_{\boldsymbol{\xi}} \log p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \boldsymbol{\xi}) \\ &= \mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}] + t^2 \nabla_{\boldsymbol{\xi}} \log p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \boldsymbol{\xi}). \end{aligned} \quad (6.3.14)$$

The resulting operators are interpretable as a *corrected* version of the unconditional denoiser for the noisy observation, where the correction term (the so-called “measurement matching” term) enforces consistency with the observations  $\mathbf{y}$ . The reader should take care to note to which argument the gradient operators are applying in the above score functions in order to fully grasp the meaning of this operator.

The key remaining issue in making this procedure computational is to prescribe how to compute the measurement matching correction, since in general we do not have a closed-form expression for the likelihood  $p_{\mathbf{y}|\mathbf{x}_t}$  except for when  $t = 0$ . Before taking up this problem, we discuss an illustrative concrete example of the entire process, continuing from those we have developed in Section 3.2.

*Example 6.2.* Consider the case where the data distribution is Gaussian with mean  $\boldsymbol{\mu} \in \mathbb{R}^D$  and covariance  $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$ , i.e.,  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . Assume that  $\boldsymbol{\Sigma} \succeq \mathbf{0}$  is nonzero. Moreover, in the measurement model (6.3.9), suppose we obtain linear measurements of  $\mathbf{x}$  with independent Gaussian noise, where  $\mathbf{A} \in \mathbb{R}^{d \times D}$  and  $\mathbf{y} = \mathbf{Ax} + \sigma \mathbf{w}$  with  $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  independent of  $\mathbf{x}$ . Then  $\mathbf{x} =_d \boldsymbol{\Sigma}^{1/2} \mathbf{g} + \boldsymbol{\mu}$ , where  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is independent

of  $\mathbf{w}$  and  $\Sigma^{1/2}$  is the unique positive square root of the covariance matrix  $\Sigma$ , and after some algebra, we can then write

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} =_d \begin{bmatrix} \Sigma^{1/2} & \mathbf{0} \\ \mathbf{A}\Sigma^{1/2} & \sigma\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \mathbf{w} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\mu} \\ \mathbf{A}\boldsymbol{\mu} \end{bmatrix}.$$

By independence, we have that  $(\mathbf{g}, \mathbf{w})$  is jointly Gaussian, which means that  $(\mathbf{x}, \mathbf{y})$  is also jointly Gaussian, as the affine image of a jointly Gaussian vector. Its covariance matrix is given by

$$\begin{bmatrix} \Sigma^{1/2} & \mathbf{0} \\ \mathbf{A}\Sigma^{1/2} & \sigma\mathbf{I} \end{bmatrix} \begin{bmatrix} \Sigma^{1/2} & \mathbf{0} \\ \mathbf{A}\Sigma^{1/2} & \sigma\mathbf{I} \end{bmatrix}^\top = \begin{bmatrix} \Sigma & \Sigma\mathbf{A}^\top \\ \mathbf{A}\Sigma & \mathbf{A}\Sigma\mathbf{A}^\top + \sigma^2\mathbf{I} \end{bmatrix}.$$

Now, we apply the fact that conditioning a random vector with joint Gaussian distribution on a subset of coordinates is again a Gaussian distribution (Exercise 3.2). By this, we obtain that

$$p_{\mathbf{x}|\mathbf{y}}(\cdot | \boldsymbol{\nu}) = \mathcal{N} \left( \underbrace{\boldsymbol{\mu} + \Sigma\mathbf{A}^\top (\mathbf{A}\Sigma\mathbf{A}^\top + \sigma^2\mathbf{I})^{-1} (\boldsymbol{\nu} - \mathbf{A}\boldsymbol{\mu})}_{\boldsymbol{\mu}_{\mathbf{x}|\mathbf{y}}(\boldsymbol{\nu})}, \underbrace{\Sigma - \Sigma\mathbf{A}^\top (\mathbf{A}\Sigma\mathbf{A}^\top + \sigma^2\mathbf{I})^{-1} \mathbf{A}\Sigma}_{\Sigma_{\mathbf{x}|\mathbf{y}}} \right). \quad (6.3.15)$$

By the equivalence we have derived above, we get by another application of Exercise 3.2

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}, \mathbf{y} = \boldsymbol{\nu}] = \boldsymbol{\mu}_{\mathbf{x}|\mathbf{y}}(\boldsymbol{\nu}) + \Sigma_{\mathbf{x}|\mathbf{y}} (\Sigma_{\mathbf{x}|\mathbf{y}} + t^2\mathbf{I})^{-1} (\boldsymbol{\xi} - \boldsymbol{\mu}_{\mathbf{x}|\mathbf{y}}(\boldsymbol{\nu})). \quad (6.3.16)$$

The functional form of this denoiser is quite simple, but it carries an unwieldy dependence on the problem data  $\boldsymbol{\mu}$ ,  $\Sigma$ ,  $\mathbf{A}$ , and  $\sigma^2$ . We can gain further insight into its behavior by comparing it with Equation (6.3.14). We have as usual

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}] = \boldsymbol{\mu} + \Sigma (\Sigma + t^2\mathbf{I})^{-1} (\boldsymbol{\xi} - \boldsymbol{\mu}), \quad (6.3.17)$$

which is rather simple—suggesting that the measurement matching term is rather complicated. To confirm this, we can calculate the likelihood  $p_{\mathbf{y}|\mathbf{x}_t}$  directly using the following expression for the joint distribution of  $(\mathbf{x}_t, \mathbf{y})$ :

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{x}_t \\ \mathbf{y} \end{bmatrix} =_d \begin{bmatrix} \Sigma^{1/2} & \mathbf{0} & \mathbf{0} \\ \Sigma^{1/2} & t\mathbf{I} & \mathbf{0} \\ \mathbf{A}\Sigma^{1/2} & \mathbf{0} & \sigma\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \mathbf{g}' \\ \mathbf{w} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \\ \mathbf{A}\boldsymbol{\mu} \end{bmatrix}, \quad (6.3.18)$$

where  $\mathbf{g}' \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  independent of the other Gaussians. This is again a jointly Gaussian distribution; restricting to only the final two rows, we have the covariance

$$\begin{bmatrix} \Sigma^{1/2} & t\mathbf{I} & \mathbf{0} \\ \mathbf{A}\Sigma^{1/2} & \mathbf{0} & \sigma\mathbf{I} \end{bmatrix} \begin{bmatrix} \Sigma^{1/2} & t\mathbf{I} & \mathbf{0} \\ \mathbf{A}\Sigma^{1/2} & \mathbf{0} & \sigma\mathbf{I} \end{bmatrix}^\top = \begin{bmatrix} \Sigma + t^2\mathbf{I} & \Sigma\mathbf{A}^\top \\ \mathbf{A}\Sigma & \mathbf{A}\Sigma\mathbf{A}^\top + \sigma^2\mathbf{I} \end{bmatrix}.$$

Another application of Exercise 3.2 then gives us

$$p_{\mathbf{y}|\mathbf{x}_t}(\cdot | \boldsymbol{\xi}) = \mathcal{N} \left( \underbrace{\mathbf{A}\boldsymbol{\mu} + \mathbf{A}\Sigma (\Sigma + t^2\mathbf{I})^{-1} (\boldsymbol{\xi} - \boldsymbol{\mu})}_{\boldsymbol{\mu}_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\xi})}, \underbrace{\mathbf{A}\Sigma\mathbf{A}^\top + \sigma^2\mathbf{I} - \mathbf{A}\Sigma (\Sigma + t^2\mathbf{I})^{-1} \Sigma\mathbf{A}^\top}_{\Sigma_{\mathbf{y}|\mathbf{x}_t}} \right). \quad (6.3.19)$$

Now notice that  $\boldsymbol{\mu}_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\xi}) = \mathbf{A}\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}]$ . So, by the chain rule,

$$\begin{aligned} t^2 \nabla_{\boldsymbol{\xi}} \log p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \boldsymbol{\xi}) &= t^2 \nabla_{\boldsymbol{\xi}} \left[ -\frac{1}{2} (\boldsymbol{\nu} - \mathbf{A}\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}])^\top \Sigma_{\mathbf{y}|\mathbf{x}_t}^{-1} (\boldsymbol{\nu} - \mathbf{A}\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}]) \right] \\ &= t^2 (\Sigma + t^2\mathbf{I})^{-1} \Sigma\mathbf{A}^\top \Sigma_{\mathbf{y}|\mathbf{x}_t}^{-1} (\boldsymbol{\nu} - \mathbf{A}\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}]). \end{aligned} \quad (6.3.20)$$

This gives us a more interpretable decomposition of the conditional posterior denoiser (6.3.16): following Equation (6.3.14), it is the sum of the unconditional posterior denoiser (6.3.17) and the measurement matching term (6.3.20). We can further analyze the measurement matching term. Notice that

$$\Sigma_{\mathbf{y}|\mathbf{x}_t} = \sigma^2\mathbf{I} + \mathbf{A}\Sigma^{1/2} \left( \mathbf{I} - \Sigma^{1/2} (\Sigma + t^2\mathbf{I})^{-1} \Sigma^{1/2} \right) \Sigma\mathbf{A}^\top. \quad (6.3.21)$$

If we let  $\Sigma = \mathbf{V}\Lambda\mathbf{V}^\top$  denote an eigenvalue decomposition of  $\Sigma$ , where  $(\mathbf{v}_i)$  are the columns of  $\mathbf{V}$ , we can further write

$$\Sigma^{1/2} \left( \mathbf{I} - \Sigma^{1/2} (\Sigma + t^2 \mathbf{I})^{-1} \Sigma^{1/2} \right) \Sigma^{1/2} = t^2 \mathbf{V} \Lambda^{1/2} (\Lambda + t^2 \mathbf{I})^{-1} \Lambda^{1/2} \mathbf{V}^\top \quad (6.3.22)$$

$$= t^2 \sum_{i=1}^D \frac{\lambda_i}{\lambda_i + t^2} \mathbf{v}_i \mathbf{v}_i^*. \quad (6.3.23)$$

Then for any eigenvalue of  $\Sigma$  equal to zero, the corresponding summand is zero; and writing  $\lambda_{\min}(\Sigma)$  for the smallest positive eigenvalue of  $\Sigma$  (it has at least one positive eigenvalue, by assumption), we have (in a sense that can be made quantitatively precise) that whenever  $t \ll \sqrt{\lambda_{\min}(\Sigma)}$ , it holds

$$\frac{\lambda_i t^2}{\lambda_i + t^2} \approx 0. \quad (6.3.24)$$

So, when  $t \ll \sqrt{\lambda_{\min}(\Sigma)}$ , we have the approximation

$$\Sigma_{\mathbf{y}|\mathbf{x}_t} \approx \sigma^2 \mathbf{I}. \quad (6.3.25)$$

The righthand side of this approximation is equal to  $\Sigma_{\mathbf{y}|\mathbf{x}}$ . So we have in turn

$$\nabla_{\xi} \log p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \xi) \approx \nabla_{\xi} \log p_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\nu} | \mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi]). \quad (6.3.26)$$

Equation (6.3.26) is, of course, a direct consequence of our calculations above. However, notice that if we directly interpret this approximation, it is *ab initio* tractable: the likelihood  $p_{\mathbf{y}|\mathbf{x}} = \mathcal{N}(\mathbf{A}\mathbf{x}, \sigma^2 \mathbf{I})$  is a simple Gaussian distribution centered at the observation, and the approximation to the measurement matching term that we arrive at can be interpreted as simply evaluating the log-likelihood at the conditional expectation  $\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi]$ , then taking gradients with respect to  $\xi$  (and backpropagating through the conditional expectation, which is given here by Equation (6.3.17)). Nevertheless, note that the approximation in Equation (6.3.26) requires  $t \ll \sqrt{\lambda_{\min}(\Sigma)}$ , and that it is never accurate in general when this condition does not hold, even in this Gaussian setting.

To gain insight into the effect of the convenient approximation (6.3.26), we implement and simulate a simple numerical experiment in the Gaussian setting in Figure 6.9. The sampler we implement is a direct implementation of the simple scheme (3.2.66) we have developed in Chapter 3 and recalled above, using the true conditional posterior denoiser, i.e. Equation (6.3.16) (top row of Figure 6.9), and the convenient approximation to this denoiser made with the decomposition (6.3.14), the posterior denoiser (6.3.17), and the measurement matching approximation (6.3.26) (bottom row of Figure 6.9). We see that even in the simple Gaussian setting, the approximation to the measurement matching term we have made is not without its drawbacks—specifically, at small noise levels  $\sigma^2 \ll 1$ , it leads to rapid collapse of the variance of the sampling distribution along directions that are parallel to the rows of the linear measurement operator  $\mathbf{A}$ , which cannot be corrected by later iterations of sampling. We can intuit this from the approximation (6.3.26) and the definition of the denoising iteration (3.2.66), given Equation (6.3.14): for  $\sigma^2 \ll 1$ , early steps of sampling effectively take gradient descent steps with a very large step size on the likelihood, via Equation (6.3.26), which leads the sampling distribution to get “stuck” in a collapsed state. ■

Example 6.2 suggests a convenient approximation for the measurement matching term (6.3.26), which can be made beyond the Gaussian setting of the example. To motivate this approximation in greater generality, notice that by conditional independence of  $\mathbf{y}$  and  $\mathbf{x}_t$  given  $\mathbf{x}$ , we can write

$$p_{\mathbf{y}|\mathbf{x}_t}(\boldsymbol{\nu} | \xi) = \int p_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\nu} | \xi') p_{\mathbf{x}|\mathbf{x}_t}(\xi' | \xi) d\xi'. \quad (6.3.27)$$

Formally, when the posterior  $p_{\mathbf{x}|\mathbf{x}_t}$  is a delta function centered at its mean  $\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi]$ , the approximation (6.3.26) is exact. More generally, when the posterior  $p_{\mathbf{x}|\mathbf{x}_t}$  is highly concentrated around its mean, the approximation (6.3.26) is accurate. This holds, for example, for sufficiently small  $t$ , which we saw explicitly in the Gaussian setting of Example 6.2. Although the numerical simulation in Figure 6.9 suggests that

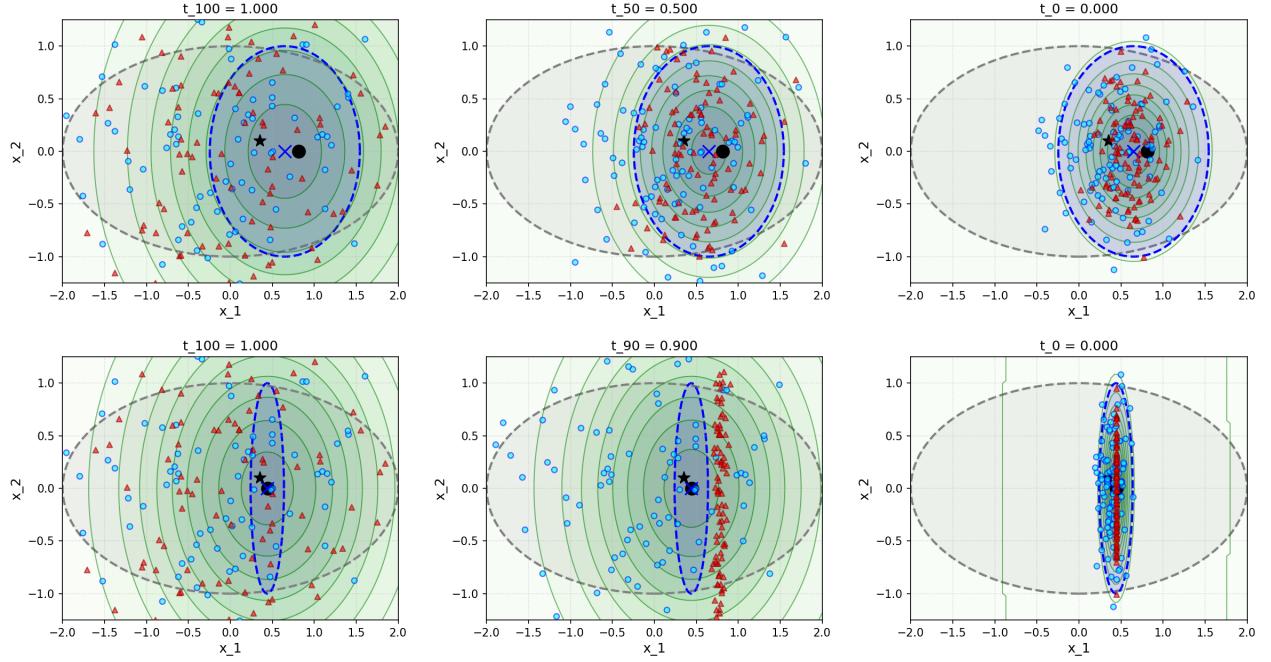


Figure 6.9: Numerical simulation of the conditional sampling setup (6.3.9), with Gaussian data, linear measurements, and Gaussian noise. We simulate  $D = 2$  and  $d = 1$ , with  $\Sigma = \mathbf{e}_1\mathbf{e}_1^\top + \frac{1}{4}\mathbf{e}_2\mathbf{e}_2^\top$ ,  $\mu = \mathbf{0}$ , and  $\mathbf{A} = \mathbf{e}_1^\top$ . The underlying signal  $\mathbf{x}$  is marked with a black star, and the measurement  $\mathbf{y}$  is marked with a black circle. Each individual plot corresponds to a different value of sampler time  $t_\ell$ , with different rows corresponding to different observation noise levels  $\sigma^2$ . In each plot, the covariance matrix of  $\mathbf{x}$  is plotted in gray, the posterior covariance matrix and posterior mean of  $p_{\mathbf{x}|y}$  are plotted in blue (with the posterior mean marked by a blue “x”), and contours for  $p_{\mathbf{x}_{t_\ell}|y}$  are drawn in green. The sampler hyperparameters are  $T = 1$ ,  $L = 100$ , and we draw 100 independent samples to initialize the samplers. Samplers are implemented with the closed-form denoisers derived in Example 6.2, with those using the approximation (6.3.26) marked with red triangles, and those using the exact conditional posterior denoiser marked with blue circles. **Top:** For large observation noise  $\sigma = 0.5$ , both the exact conditional posterior denoiser and the approximate one do a good job of converging to the posterior  $p_{\mathbf{x}|y}$ . Sampling time (corresponding to time in the “forward process”, so larger times mean larger noise) decreases from left to right. The convergence dynamics for the exact and approximate measurement matching term are similar. **Bottom:** For smaller observation noise  $\sigma = 0.1$ , the approximate measurement matching term leads to extreme bias in the sampler (red triangles): samples rapidly converge to an affine subspace of points that are consistent, modulo some shrinkage from the posterior mean denoiser, with the measured ground truth, and later sampling iterations are unable to recover the lost posterior variance along this dimension. Note that different times  $t_\ell$  are plotted in the bottom row, compared to the top row, to show the rapid collapse of the approximation to the posterior along the measurement dimension.

this approximation is not without its caveats in certain regimes, it has proved to be a reliable baseline in practice, after being proposed by Chung et al. as ‘‘Diffusion Posterior Sampling’’ (DPS) [CKM+23]. In addition, there are even principled and generalizable approaches to improve it by incorporating better estimates of the posterior variance (which turn out to be exact in the Gaussian setting of Example 6.2), which we discuss further in the end-of-chapter summary.

Thus, with the DPS approximation, we arrive at the following approximation for the conditional posterior denoisers  $\mathbb{E}[\mathbf{x} | \mathbf{y}, \mathbf{x}_t]$ , via Equation (6.3.14):

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}, \mathbf{y} = \boldsymbol{\nu}] \approx \mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}] + t^2 \nabla_{\boldsymbol{\xi}} \log p_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\nu} | \mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}]). \quad (6.3.28)$$

And, for a neural network or other model  $\bar{\mathbf{x}}_\theta(t, \boldsymbol{\xi})$  trained as in Section 3.2 to approximate the denoisers  $\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \boldsymbol{\xi}]$  for each  $t \in [0, T]$ , we arrive at the learned conditional posterior denoisers

$$\bar{\mathbf{x}}_\theta(t, \boldsymbol{\xi}, \boldsymbol{\nu}) = \bar{\mathbf{x}}_\theta(t, \boldsymbol{\xi}) + t^2 \nabla_{\boldsymbol{\xi}} \log p_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\nu} | \bar{\mathbf{x}}_\theta(t, \boldsymbol{\xi})). \quad (6.3.29)$$

Note that the approximation (6.3.28) is valid for arbitrary forward models  $h$  in the observation model (6.3.9), including nonlinear  $h$ , and even to arbitrary noise models for which a clean expression for the likelihood  $p_{\mathbf{y}|\mathbf{x}}$  is known. Indeed, in the case of Gaussian noise, we have

$$p_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\nu} | \boldsymbol{\xi}) \propto \exp\left(-\frac{1}{2\sigma^2} \|h(\boldsymbol{\xi}) - \boldsymbol{\nu}\|_2^2\right). \quad (6.3.30)$$

Hence, evaluating the righthand side of (6.3.29) requires only

1. A pretrained denoiser  $\bar{\mathbf{x}}_\theta(t, \boldsymbol{\xi})$  for the data distribution  $p$  (of  $\mathbf{x}$ ), learned as in Section 3.2 via Algorithm 3.2;
2. Forward and backward pass access to the forward model  $h$  for the measurements (6.3.9);
3. A forward and backward pass through  $\bar{\mathbf{x}}_\theta(t, \boldsymbol{\xi})$ , which can be evaluated efficiently using (say) backpropagation.

---

**Algorithm 6.1** Conditional sampling under measurements (6.3.9), with an unconditional denoiser and DPS.

---

**Input:** An ordered list of timesteps  $0 \leq t_0 < \dots < t_L \leq T$  to use for sampling.

**Input:** An unconditional denoiser  $\bar{\mathbf{x}}_\theta: \{t_\ell\}_{\ell=1}^L \times \mathbb{R}^D \rightarrow \mathbb{R}^D$  for  $p_{\mathbf{x}}$ .

**Input:** Measurement realization  $\boldsymbol{\nu}$  of  $\mathbf{y}$  (Equation (6.3.9)) to condition on.

**Input:** Forward model  $h: \mathbb{R}^D \rightarrow \mathbb{R}^d$  and measurement noise variance  $\sigma^2 > 0$ .

**Input:** Scale and noise level functions  $\alpha, \sigma: \{t_\ell\}_{\ell=0}^L \rightarrow \mathbb{R}_{\geq 0}$ .

**Output:** A sample  $\hat{\mathbf{x}}$ , approximately from  $p_{\mathbf{x}|\mathbf{y}}$ .

1: **function** DDIMSAMPLERCONDITIONALDPS( $\bar{\mathbf{x}}_\theta, \boldsymbol{\nu}, h, \sigma^2, (t_\ell)_{\ell=0}^L$ )  
2:     Initialize  $\hat{\mathbf{x}}_{t_L} \sim$  approximate distribution of  $\mathbf{x}_{t_L}$  ▷ VP  $\implies \mathcal{N}(\mathbf{0}, \mathbf{I})$ , VE  $\implies \mathcal{N}(\mathbf{0}, t_L^2 \mathbf{I})$ .

3:     **for**  $\ell = L, L-1, \dots, 1$  **do**

4:         Compute

$$\hat{\mathbf{x}}_{t_{\ell-1}} \doteq \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \hat{\mathbf{x}}_{t_\ell} + \left( \alpha_{t_{\ell-1}} - \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \alpha_{t_\ell} \right) \left( \bar{\mathbf{x}}_\theta(t_\ell, \hat{\mathbf{x}}_{t_\ell}) - \frac{\sigma_{t_\ell}^2}{2\alpha_{t_\ell}\sigma^2} \nabla_{\boldsymbol{\xi}} \left[ \|h(\bar{\mathbf{x}}_\theta(t_\ell, \boldsymbol{\xi})) - \boldsymbol{\nu}\|_2^2 \right] \Big|_{\boldsymbol{\xi}=\hat{\mathbf{x}}_{t_\ell}} \right)$$

5:     **end for**

6:     **return**  $\hat{\mathbf{x}}_{t_0}$

7: **end function**

---

Combining this scheme with the basic implementation of unconditional sampling we developed in Section 3.2, we obtain a practical algorithm for conditional sampling of the posterior  $p_{\mathbf{x}|\mathbf{y}}$  given measurements following (6.3.9). Algorithm 6.1 records this scheme for the case of Gaussian observation noise with known standard deviation  $\sigma$ , with minor modifications to extend to a general noising process, as in Equation (3.2.69) and the surrounding discussion in Chapter 3 (our discussion above made the simplifying choices  $\alpha_t = 1$ ,  $\sigma_t = t$ , and  $t_\ell = T\ell/L$ , as for Equation (3.2.66) in Section 3.2).

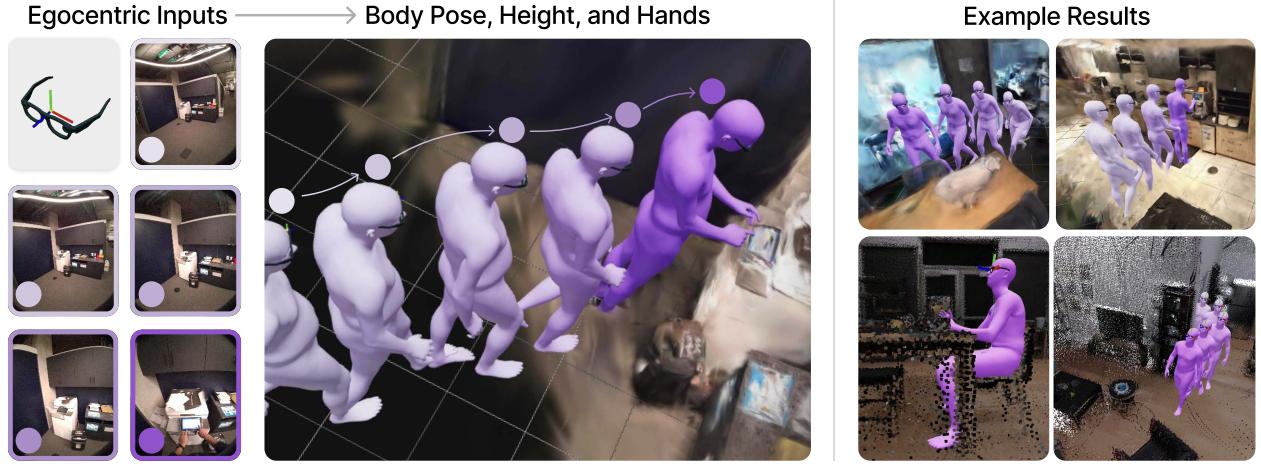


Figure 6.10: A system that estimates human body height, pose, and hand parameters (middle), conditioned on ego-centric SLAM poses and images (left). Outputs capture the wearer’s actions in the allocentric reference frame of the scene, which we visualize here with 3D reconstructions (right).

### 6.3.3 Body Pose Generation Conditioned on Head and Hands

The type of conditional estimation or generation problems arise rather naturally in many practical applications. A typical problem of this kind is how to estimate and generate body pose and hand gesture conditioned on a given head pose and egocentric images, as illustrated in Figure 6.10. This is often the problem we need to solve when one is wearing a head-mounted device such as the Vision-pro from Apple or the Project Aria from Meta. The pose of the whole body and the gesture of the hands need to be inferred so that we can use the information to control virtual objects that the person interacts with.

Notice that in this case, one only has the head pose provided by the device and a very limited field of view for part of one’s hands and upper limbs. The pose of the rest of the body needs to “inferred” or “completed” based on such partial information. The only way one can estimate the body pose over time is by learning the joint distribution of the head and body pose sequences in advance and then sample this prior distribution conditioned on the real-time partial inputs. Figure 6.11 outlines a system called EgoAllo [YYZ+24] to solve this problem based on a learned conditional diffusion-denoising model.

Figure 6.12 compares some ground truth motion sequences with sampled results generated by the EgoAllo. Although the figure shows one result for each input head pose sequence, different runs can generate different body pose sequences that are consistent with the given head pose, all drawing from the distribution of natural full-body motion sequences.

Strictly speaking, the solution proposed in EgoAllo [YYZ+24] does not enforce measurement matching using the techniques introduced above. Instead it heuristically enforces the condition by utilizing the cross-attention mechanism in a transformer architecture. As we will describe with more precision in the paired data setting in Section 6.4.2, there is reason to believe that the cross-attention mechanism is in a way approximately realizing the conditional sampling of the denoising a posteriori. We believe the more principled techniques introduced here, if properly implemented, can lead to better methods that further improve the body pose and hand gesture estimation.

## 6.4 Conditional Inference with Paired Data and Measurements

In many practical applications, we do not know either the distribution of the data  $\mathbf{x}$  of interest nor the explicit relationship between the data and certain observed attributes  $\mathbf{y}$  of the data. We only have a (large) set of paired samples  $(\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$  from which we need to infer the data distribution and a mapping that models their relationship:

$$h : \mathbf{x} \mapsto \mathbf{y}. \quad (6.4.1)$$

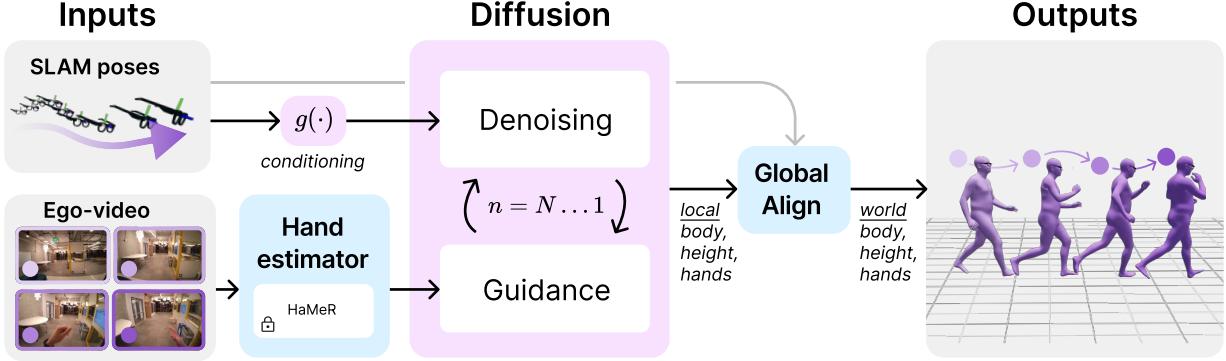


Figure 6.11: **Overview of technical components of EgoAllo [YYZ+24]**. A diffusion model is pre-trained that can generate body pose sequence based on local body parameters (middle). An invariant parameterization  $g(\cdot)$  of SLAM poses (left) is used to condition the diffusion model. These can be placed into the global coordinate frame via global alignment to input poses. When available, egocentric video is used for hand detection (left) via HaMeR [PSR+23], which can be incorporated into samples via guidance by the generated gesture.

The problem of image classification can be viewed as one such an example. In a sense, the classification problem is to learn a (extremely lossy) compressive encoder for natural images. Say, given a random sample of an image  $\mathbf{x}$ , we would like to predict its class label  $\mathbf{y}$  that best correlates the content in  $\mathbf{x}$ . We know the distribution of natural images of objects is low-dimensional compared to the dimension of the pixel space. From the previous chapters, we have learned that given sufficient samples, in principle, we can learn a structured low-dimensional representation  $\mathbf{z}$  for  $\mathbf{x}$  through a learned compressive encoding:

$$f : \mathbf{x} \mapsto \mathbf{z}. \quad (6.4.2)$$

The representation  $\mathbf{z}$  can also be viewed as a learned (lossy but structured) code for  $\mathbf{x}$ . It is rather reasonable to assume that if the class assignment  $\mathbf{y}$  truly depends on the low-dimensional structures of  $\mathbf{x}$  and the learned code  $\mathbf{z}$  truly reflect such structures,  $\mathbf{y}$  and  $\mathbf{z}$  can be made highly correlated hence their joint distribution  $p(\mathbf{z}, \mathbf{y})$  should be extremely low-dimensional. Therefore, we may combine the two desired codes  $\mathbf{y}$  and  $\mathbf{z}$  together and try to learn a combined encoder:

$$f : \mathbf{x} \mapsto (\mathbf{z}, \mathbf{y}) \quad (6.4.3)$$

where the joint distribution of  $(\mathbf{z}, \mathbf{y})$  is highly low-dimensional.

From our study in previous chapters, the mapping  $f$  is usually learned as a sequence of compression or denoising operators in the same space. Hence to leverage on such family of operations, we may introduce an auxiliary vector  $\mathbf{w}$  that can be viewed as an initial random guess of the class label  $\mathbf{y}$ . In this way, we can learn a compression or denoising mapping:

$$f : (\mathbf{x}, \mathbf{w}) \mapsto (\mathbf{z}, \mathbf{y}) \quad (6.4.4)$$

within a common space. In fact, the common practice of introducing an auxiliary ‘‘class token’’ in the training of a Transformer for classification tasks, such as in ViT, can be viewed as learning such a representation by compressing (the coding rate of) given (noisy) samples of  $(\mathbf{x}, \mathbf{w})$ . If the distribution of the data  $\mathbf{x}$  is already a mixture of (low-dimensional) Gaussians, the work [WTL+08] has shown that classification can be done effectively by directly minimizing the *(lossy) coding length* associated with the given samples.

#### 6.4.1 Class Conditioned Image Generation

While a learned classifier allows us to classify a given image  $\mathbf{x}$  to its corresponding class, we often like to generate an image of a given class, by sampling the learned distribution of natural images. To some extent,

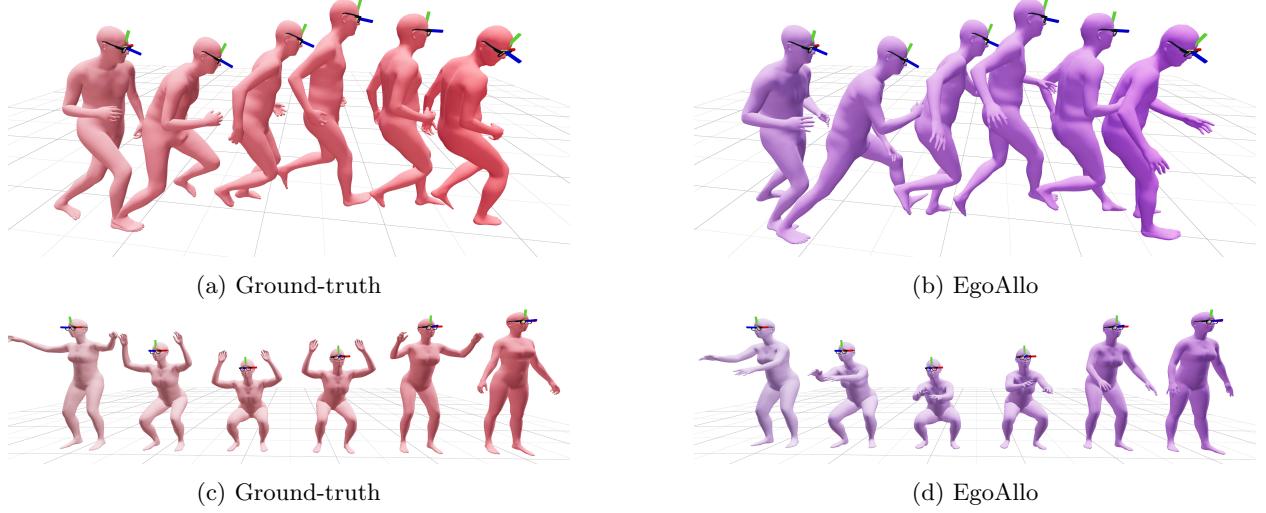


Figure 6.12: **Egocentric human motion estimation for a running (top) and squatting (bottom) sequence.** The ground-truth motion is compared with one output from EgoAllo that is consistent with the given head pose sequence.

this can be viewed as the “inverse” problem to image classification. Let  $p_{\mathbf{x}}$  denote the distribution of natural images, say modeled by a diffusion-denoising process. Given a class label random variable  $y \in [K]$  with realization  $\nu$ , say an “Apple”, we would like to sample the conditional distribution  $p_{\mathbf{x}|y}(\cdot | \nu)$  to generate an image of an apple:

$$\hat{\mathbf{x}} \sim p_{\mathbf{x}|y}(\cdot | \nu). \quad (6.4.5)$$

We call this *class-conditioned image generation*.

In Section 6.3.2, we have seen how to use the denoising-diffusion paradigm for conditional sampling from the posterior  $p_{\mathbf{x}|y}$  given *model-based* measurements  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w}$  (Equation (6.3.9)), culminating in the DPS algorithm (6.1). This is a powerful framework, but it does not apply to the class (or text) conditioned image generation problem here, where an explicit generative model  $h$  for the observations/attributes  $y$  is not available due to the intractability of analytical modeling. In this section, we will present techniques for extending conditional sampling to this setting.

Thus, we now assume only that we have access to samples from the joint distribution of  $(\mathbf{x}, y)$ :

$$(\mathbf{x}, y) \sim p_{\mathbf{x}, y}. \quad (6.4.6)$$

As in the previous section, we define  $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \mathbf{g}$  with  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  independent of  $(\mathbf{x}, y)$ , as in Equation (3.2.69) in Chapter 3, and we will repeatedly use the notation  $\xi$  to denote realizations of  $\mathbf{x}$  and  $\mathbf{x}_t$ .

To proceed, we note that our development of conditional sampling under measurements  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w}$  only explicitly used the forward model  $h$  in making the DPS approximation (6.3.26). In particular, the conditional posterior denoiser decomposition (6.3.14) *still holds in the paired data setting*, by virtue of Bayes’ rule and conditional independence of  $\mathbf{y}$  and  $\mathbf{x}_t$  given  $\mathbf{x}$  (recall Figure 6.8). Thus we can still write in the paired data setting

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi, y = \nu] = \mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi] + \frac{\sigma_t^2}{\alpha_t} \nabla_\xi \log p_{y|\mathbf{x}_t}(\nu | \xi). \quad (6.4.7)$$

A natural ideal is then to directly implement the likelihood correction term in (6.4.7) using a deep network  $f_{\theta_c}$  with parameters  $\theta_c$ , as in Equation (6.4.4):

$$f_{\theta_c} : (t, \mathbf{x}_t) \mapsto \text{softmax}(\mathbf{W}_{\text{head}} \mathbf{z}(t, \mathbf{x}_t)). \quad (6.4.8)$$

This expression combines the final representations  $\mathbf{z}(t, \mathbf{x}_t)$  (which also depend on  $\theta_c$ ) of the noisy inputs  $\mathbf{x}_t$  with a classification head  $\mathbf{W}_{\text{head}} \in \mathbb{R}^{K \times d}$ , which maps the representations to a probability distribution over

the  $K$  possible classes. As is common in practice, it also takes the time  $t$  in the noising process as input. Thus, with appropriate training, it provides an approximation to the log-likelihood  $\log p_{y|\mathbf{x}_t}$ , and differentiating  $\log f_{\theta_c}$  with respect to its input  $\mathbf{x}_t$  allows an approximation to the second term in Equation (6.4.7):

$$\bar{\mathbf{x}}_\theta^{\text{naive}}(t, \mathbf{x}_t, y) = \bar{\mathbf{x}}_{\theta_d}(t, \mathbf{x}_t) + \frac{\sigma_t^2}{\alpha_t} \nabla_{\mathbf{x}_t} \langle \log f_{\theta_c}(t, \mathbf{x}_t), \mathbf{e}_y \rangle \quad (6.4.9)$$

where, as usual, we approximate the first term in Equation (6.4.7) via a learned unconditional denoiser for  $\mathbf{x}_t$  with parameters  $\theta_d$ , and where we write  $\mathbf{e}_k$  for  $k \in [K]$  to denote the  $k$ -th canonical basis vector for  $\mathbb{R}^K$  (i.e., the vector with a one in the  $k$ -th position, and zeros elsewhere). The reader should note that the conditional denoiser  $\bar{\mathbf{x}}_\theta$  requires two separate training runs, with separate losses: one for the classifier parameters  $\theta_c$ , on a classification loss,<sup>8</sup> and one for the denoiser parameters  $\theta_d$ , on a denoising loss. Such an approach to conditional sampling was already recognized and exploited to perform conditional sampling in pioneering early works on diffusion models, notably those by Sohl-Dickstein et al. [SWM+15] and by Song et al. [SSK+21].

However, this straightforward methodology has two key drawbacks (which is why we label it as “naive”). The first is that, empirically, such a trained deep network classifier frequently does not provide a strong enough guidance signal (in Equation (6.4.7)) to ensure that generated samples reflect the conditioning information  $y$ . This was first emphasized by Dhariwal and Nichol [DN21], who noted that in the setting of class-conditional ImageNet generation, the learned deep network classifier’s probability outputs for the class  $y$  being conditioned on were frequently around 0.5—large enough to be the dominant class, but not large enough to provide a strong guidance signal—and that upon inspection, generations were not consistent with the conditioning class  $y$ . Dhariwal and Nichol [DN21] proposed to address this heuristically by incorporating an “inverse temperature” hyperparameter  $\gamma > 0$  into the definition of the naive conditional denoiser (6.4.9), referring to the resulting conditional denoiser as having incorporated “classifier guidance” (CG):

$$\bar{\mathbf{x}}_\theta^{\text{CG}}(t, \mathbf{x}_t, y) = \bar{\mathbf{x}}_{\theta_d}(t, \mathbf{x}_t) + \gamma \frac{\sigma_t^2}{\alpha_t} \nabla_{\mathbf{x}_t} \langle \log f_{\theta_c}(t, \mathbf{x}_t), \mathbf{e}_y \rangle \quad (6.4.10)$$

with the case  $\gamma = 1$  coinciding with (6.4.9). Dhariwal and Nichol [DN21] found that a setting  $\gamma > 1$  performed best empirically. One possible interpretation for this is as follows: note that, in the context of the *true* likelihood term Equation (6.4.7), scaling by  $\gamma$  gives equivalently

$$\gamma \frac{\sigma_t^2}{\alpha_t} \nabla_{\xi} \log p_{y|\mathbf{x}_t}(\boldsymbol{\nu} | \xi) = \frac{\sigma_t^2}{\alpha_t} \nabla_{\xi} \log (p_{y|\mathbf{x}_t}(\boldsymbol{\nu} | \xi)^\gamma), \quad (6.4.11)$$

which suggests the natural interpretation of the parameter  $\gamma$  performing (inverse) *temperature scaling* on the likelihood  $p_{y|\mathbf{x}_t}$ , which is precise if we consider the renormalized distribution  $p_{y|\mathbf{x}_t}(\boldsymbol{\nu} | \xi)^\gamma / \int p_{y|\mathbf{x}_t}(\boldsymbol{\nu}' | \xi)^\gamma d\boldsymbol{\nu}'$ . However, note that this *is not* a rigorous interpretation in the context of Equation (6.4.7), because the gradients are taken with respect to  $\xi$ , and the normalization constant in the temperature-scaled distribution is in general a function of  $\xi$ . Instead, the parameter  $\gamma$  should simply be understood as amplifying large values of the deep network classifier’s output probabilities  $f_{\theta_c}(t, \mathbf{x}_t)$  relative to smaller ones, which effectively amplifies the guidance signal provided in cases where the deep network  $f$  assigns it the largest probability among the  $K$  classes.

Nevertheless, classifier guidance does not address the second key drawback of the naive methodology: it is both cumbersome and wasteful to have to train an auxiliary classifier  $f_{\theta_c}$  in addition to the unconditional denoiser  $\bar{\mathbf{x}}_{\theta_d}$ , given that it is not possible to directly adapt a pretrained classifier due to the need for it to work well on noisy inputs  $\mathbf{x}_t$  and incorporate other empirically-motivated architecture modifications. In particular, Dhariwal and Nichol [DN21] found that it was necessary to explicitly design the architecture of the deep network implementing the classifier to match that of the denoiser. Moreover, from a purely practical perspective—trying to obtain the best possible performance from the resulting sampler—the best-performing configuration of classifier guidance-based sampling departs even further from the idealized and conceptually sound framework we have presented above. To obtain the best performance, Dhariwal and Nichol [DN21] found it necessary to provide the class label  $y$  as an additional input to the denoiser  $\bar{\mathbf{x}}_{\theta_d}$ .

---

<sup>8</sup>In Chapter 7, we review the process of training such a classifier in full detail.

As a result, the idealized classifier-guided denoiser (6.4.10), derived by Dhariwal and Nichol [DN21] as we have done above from the conditional posterior denoiser decomposition (6.4.7), is not exactly reflective of the best-performing denoiser in practice—such a denoiser actually combines a *conditional* denoiser for  $\mathbf{x}_t$  given  $y$  with an additional guidance signal from an auxiliary classifier!

This state of affairs, empirically-motivated as it is, led Ho and Salimans [HS22] in subsequent work to propose a more empirically pragmatic methodology, known as classifier-free guidance (CFG). Instead of representing the conditional denoiser (6.4.7) as a weighted sum of an unconditional denoiser for  $\mathbf{x}_t$  with a log-likelihood correction term (with possibly modified weights, as in classifier guidance), they accept the apparent necessity of training a conditional denoiser for  $\mathbf{x}_t$  given  $y$ , as demonstrated by the experimental results of Dhariwal and Nichol [DN21], and replace the log-likelihood gradient term with a correctly-weighted sum of this conditional denoiser with an *unconditional* denoiser for  $\mathbf{x}$  given  $\mathbf{x}_t$ .<sup>9</sup> To see how this structure arises, we begin with an ‘idealized’ version of the classifier guidance denoiser  $\bar{\mathbf{x}}_\theta^{\text{CG}}$  defined in (6.4.10), for which the denoiser  $\bar{\mathbf{x}}_{\theta_d}$  and the classifier  $f_{\theta_c}$  perfectly approximate their targets, via (6.4.7):

$$\bar{\mathbf{x}}_\theta^{\text{CG, ideal}}(t, \xi, \nu) = \mathbb{E}[\mathbf{x} \mid \mathbf{x}_t = \xi] + \gamma \frac{\sigma_t^2}{\alpha_t} \nabla_\xi \log p_{y|\mathbf{x}_t}(\nu \mid \xi). \quad (6.4.12)$$

We then use Bayes’ rule, in the form

$$\log p_{y|\mathbf{x}_t} = \log p_{\mathbf{x}_t|y} + \log p_y - \log p_{\mathbf{x}_t}, \quad (6.4.13)$$

together with Tweedie’s formula (Theorem 3.3, modified as in Equation (3.2.70)) to convert between score functions and denoisers, to obtain

$$\begin{aligned} \bar{\mathbf{x}}_\theta^{\text{CG, ideal}}(t, \xi, \nu) &= \frac{1}{\alpha_t} \xi + (1 - \gamma) \frac{\sigma_t^2}{\alpha_t} \nabla_\xi \log p_{\mathbf{x}_t}(\xi) + \gamma \frac{\sigma_t^2}{\alpha_t} \nabla_\xi \log p_{\mathbf{x}_t|y}(\xi \mid \nu) \\ &= (1 - \gamma) \mathbb{E}[\mathbf{x} \mid \mathbf{x}_t = \xi] + \gamma \mathbb{E}[\mathbf{x} \mid \mathbf{x}_t = \xi, y = \nu], \end{aligned} \quad (6.4.14)$$

where in the last line, we apply Equation (6.3.11). Now, Equation (6.4.14) suggests a natural approximation strategy: we combine a learned unconditional denoiser for  $\mathbf{x}$  given  $\mathbf{x}_t$ , as previously, with a learned *conditional* denoiser for  $\mathbf{x}$  given  $\mathbf{x}_t$  and  $y$ .

However, following Ho and Salimans [HS22] and the common practice of training deep network denoisers, it is standard to use the *same* deep network to represent both the conditional and unconditional denoisers, by introducing an additional label, which we will denote by  $\emptyset$ , to denote the “unconditional” case. This leads to the form of the CFG denoiser:

$$\bar{\mathbf{x}}_\theta^{\text{CFG}}(t, \mathbf{x}_t, y) = (1 - \gamma) \bar{\mathbf{x}}_\theta(t, \mathbf{x}_t, \emptyset) + \gamma \bar{\mathbf{x}}_\theta(t, \mathbf{x}_t, y). \quad (6.4.15)$$

To train a denoiser  $\bar{\mathbf{x}}_\theta(t, \mathbf{x}_t, y^+)$  for use with classifier-free guidance sampling, where  $y^+ \in \{1, \dots, K, \emptyset\}$ , we proceed almost identically to the unconditional training procedure in Algorithm 3.2, but with two modifications:

1. When we sample from the dataset, we sample a pair  $(\mathbf{x}, y)$  rather than just a sample  $\mathbf{x}$ .
2. Every time we sample a pair from the dataset, we sample the augmented label  $y^+$  via

$$y^+ = \begin{cases} \emptyset & \text{with probability } p_{\text{uncond}}; \\ y & \text{else.} \end{cases} \quad (6.4.16)$$

Here,  $p_{\text{uncond}} \in [0, 1]$  is a new hyperparameter. This can be viewed as a form of dropout [SHK+14].

In this way, we train a conditional denoiser suitable for use in classifier-free guidance sampling. We summarize the overall sampling process for class-conditioned sampling with classifier-free guidance in Algorithm 6.2.

<sup>9</sup>That said, Ho and Salimans [HS22] actually proposed to use a different weighting than what we present here, based on the fact that Dhariwal and Nichol [DN21] heuristically replaced the unconditional denoiser in (6.4.7) with a conditional denoiser. In fact, the weighting we derive and present here reflects modern practice, and in particular is used in state-of-the-art diffusion models such as Stable Diffusion 3.5 [EKB+24].

---

**Algorithm 6.2** Conditional sampling with classification data, using class-conditioned denoiser.

---

**Input:** An ordered list of timesteps  $0 \leq t_0 < \dots < t_L \leq T$  to use for sampling.  
**Input:** Class label  $\nu \in \{1, \dots, K\}$  to condition on.  
**Input:** A denoiser  $\bar{\mathbf{x}}_\theta: \{t_\ell\}_{\ell=1}^L \times \mathbb{R}^D \times \{1, \dots, K, \emptyset\} \rightarrow \mathbb{R}^D$  for  $p_{\mathbf{x}|y}$  and  $p_{\mathbf{x}}$  (input  $\emptyset$  for  $p_{\mathbf{x}}$ ).  
**Input:** Scale and noise level functions  $\alpha, \sigma: \{t_\ell\}_{\ell=0}^L \rightarrow \mathbb{R}_{\geq 0}$ .  
**Input:** Guidance strength  $\gamma \geq 0$  ( $\gamma > 1$  preferred for performance).  
**Output:** A sample  $\hat{\mathbf{x}}$ , approximately from  $p_{\mathbf{x}|y}(\cdot | \nu)$ .

```

1: function DDIMSAMPLERCONDITIONALCFG( $\bar{\mathbf{x}}_\theta, \nu, \gamma, (t_\ell)_{\ell=0}^L$ )
2:   Initialize  $\hat{\mathbf{x}}_{t_L} \sim$  approximate distribution of  $\mathbf{x}_{t_L}$  ▷ VP  $\implies \mathcal{N}(\mathbf{0}, \mathbf{I})$ , VE  $\implies \mathcal{N}(\mathbf{0}, t_L^2 \mathbf{I})$ .
3:   for  $\ell = L, L-1, \dots, 1$  do
4:     Compute
      
$$\hat{\mathbf{x}}_{t_{\ell-1}} \doteq \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \hat{\mathbf{x}}_{t_\ell} + \left( \alpha_{t_{\ell-1}} - \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \alpha_{t_\ell} \right) ((1-\gamma)\bar{\mathbf{x}}_\theta(t_\ell, \hat{\mathbf{x}}_{t_\ell}, \emptyset) + \gamma\bar{\mathbf{x}}_\theta(t_\ell, \hat{\mathbf{x}}_{t_\ell}, \nu))$$

5:   end for
6:   return  $\hat{\mathbf{x}}_{t_0}$ 
7: end function
```

---

Ho and Salimans [HS22] report strong empirical performance for class-conditional image generation with classifier-free guidance, and it has become a mainstay of the largest-scale practical diffusion models, such as Stable Diffusion [RBL+22] and its derivatives. At the same time, its derivation is rather opaque and empirically-motivated, giving little insight into the mechanisms behind its strong performance. A number of theoretical works have studied this, providing explanations for some parts of the overall CFG methodology [BN24b; LWQ25; WCL+24]—itself encompassing denoiser parameterization and training, as well as configuration of the guidance strength and performance at sampling time. Below, we will give an interpretation in the simplifying setting of a Gaussian mixture model data distribution and denoiser, which will demonstrate an insight into the *parameterization* of the denoiser in the presence of such low-dimensional structures.

*Example 6.3.* Let us recall the low-rank mixture of Gaussians data generating process we studied in Example 3.2 (and specifically, the form in Equation (3.2.42)). Given  $K \in \mathbb{N}$  classes, we assume that

$$\mathbf{x} \sim \frac{1}{K} \sum_{k=1}^K \mathcal{N}(\mathbf{0}, \mathbf{U}_k \mathbf{U}_k^\top), \quad (6.4.17)$$

where each  $\mathbf{U}_k \in \mathrm{O}(D, P) \subseteq \mathbb{R}^{D \times P}$  is a matrix with orthogonal columns, and  $P \ll D$ . Moreover, we assume that the class label  $y \in [K]$  is a deterministic function of  $\mathbf{x}$  mapping an example to its corresponding mixture component. Applying the analysis in Example 3.2 (and the subsequent analysis of the low-rank case, culminating in Equation (3.2.56)), we obtain for the class-conditional optimal denoisers

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi, y = \nu] = \frac{1}{1+t^2} \mathbf{U}_\nu \mathbf{U}_\nu^\top \xi \quad (6.4.18)$$

for each  $\nu \in [K]$ , and for the optimal unconditional denoiser, we obtain

$$\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi] = \frac{1}{1+t^2} \sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_k^\top \xi\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \xi\|_2^2\right)} \mathbf{U}_k \mathbf{U}_k^\top \xi. \quad (6.4.19)$$

As a result, we can express the CFG denoiser with guidance strength  $\gamma > 1$  as

$$\bar{\mathbf{x}}^{\text{CFG, ideal}}(t, \mathbf{x}_t, y) = \frac{1}{1+t^2} \left( (1-\gamma) \sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \mathbf{U}_k \mathbf{U}_k^\top + \gamma \mathbf{U}_y \mathbf{U}_y^\top \right) \mathbf{x}_t. \quad (6.4.20)$$

This denoiser has a simple, interpretable form. The first term, corresponding to the unconditional denoiser, performs denoising of the signal  $\mathbf{x}_t$  against an average of the denoisers associated to each subspace, weighted

by how correlated  $\mathbf{x}_t$  is to each subspace. The second term, corresponding to the conditional denoiser, simply performs denoising with the conditioning class's denoiser. The CFG scheme further averages these two denoisers: the effect can be gleaned from the refactoring

$$\begin{aligned} \bar{\mathbf{x}}^{\text{CFG, ideal}}(t, \mathbf{x}_t, y) &= \frac{1}{1+t^2} \left( \left[ \gamma + (1-\gamma) \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_y^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \right] \mathbf{U}_y \mathbf{U}_y^\top \right. \\ &\quad \left. + (1-\gamma) \sum_{k \neq y} \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \mathbf{U}_k \mathbf{U}_k^\top \right) \mathbf{x}_t. \end{aligned} \quad (6.4.21)$$

We have

$$\sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_k^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} = 1, \quad (6.4.22)$$

and each summand is nonnegative, hence also bounded above by 1. So we can conclude two regimes for the terms in Equation (6.4.21):

1. **Well-correlated regime:** If  $\mathbf{x}_t$  correlates well with  $\mathbf{U}_y$ , then the normalized weight corresponding to the  $k = y$  summand in the unconditional denoiser is near to 1. Then

$$\gamma + (1-\gamma) \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_y^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \approx 1, \quad (6.4.23)$$

all other weights are necessarily near to zero, and the CFG denoiser is approximately equal to the denoiser associated to the conditioning class  $y$ .

2. **Poorly-correlated regime:** In contrast, if  $\mathbf{x}_t$  does not correlate well with  $\mathbf{U}_y$  (say because  $t$  is large), then the normalized weight corresponding to the  $k = y$  summand in the unconditional denoiser is near to 0. As a result,

$$\gamma + (1-\gamma) \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_y^\top \mathbf{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_i^\top \mathbf{x}_t\|_2^2\right)} \approx \gamma, \quad (6.4.24)$$

and thus the guidance strength  $\gamma \gg 1$  places a large positive weight on the denoiser associated to  $y$ . Meanwhile, in the second term of Equation (6.4.21), any classes  $k \neq y$  that are well-correlated with  $\mathbf{x}_t$  receive a large *negative* weight from the  $1 - \gamma$  coefficient. This simultaneously has the effect of making the denoised signal vastly more correlated with the conditioning class  $y$ , and making it negatively correlated with the previous iterate (i.e., the iterate before denoising). In other words, CFG steers the iterative denoising process towards the conditioning class and away from the previous iterate, a different dynamics from purely conditional sampling (i.e., the case  $\gamma = 1$ ).

We now perform a further analysis of the form of this guided denoiser in order to make some inferences about the role of CFG. Many of these insights will be relevant to *general* data distributions with low-dimensional geometric structure, as well. First, notice that the CFG denoiser (6.4.20) takes a simple form in the setting where  $\mathbf{x}_t$  correlates significantly more strongly with a single subspace  $\mathbf{U}_y$  than any other  $\mathbf{U}_{y'}$ . Indeed, because the ratio of weights in the class-conditional denoiser is given by

$$\frac{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_y^\top \mathbf{x}_t\|_2^2\right)}{\exp\left(\frac{1}{2t^2(1+t^2)} \|\mathbf{U}_{y'}^\top \mathbf{x}_t\|_2^2\right)} = \exp\left(\frac{1}{2t^2(1+t^2)} (\|\mathbf{U}_y^\top \mathbf{x}_t\|_2^2 - \|\mathbf{U}_{y'}^\top \mathbf{x}_t\|_2^2)\right), \quad (6.4.25)$$

a large separation between the correlation of  $\mathbf{x}_t$  with  $\mathbf{U}_y$  and other subspaces  $\mathbf{U}_{y'}$  implies that the sum over  $k$  concentrates on the  $k = y$  summand, giving that *the CFG denoiser remains equal to the class-conditioned*

*denoiser.* Moreover, when  $t \approx 0$ , the magnitude of any such gap is amplified in the exponential, making this concentration on the  $k = y$  summand even stronger. In particular, for small times  $t$  (i.e., near to the support of the data distribution), CFG denoising is no different from standard class-conditional denoising—implying that it will converge stably once it has reached such a configuration. Thus, the empirical benefits of CFG should be due to its behavior in cases where  $\mathbf{x}_t$  is not unambiguously from a single class.

Next, we consider the problem of parameterizing a learnable denoiser  $\mathbf{x}_\theta^{\text{CFG}}$  to represent the optimal denoiser (6.4.20). Here, it may initially seem that the setting of classification of a mixture distribution is too much of a special case relative to learning practical data distributions, as the ideal denoiser (6.4.20) has in this setting the simple form of a *hard assignment* of the noisy signal  $\mathbf{x}_t$  to the (denoiser associated to) the subspace  $\mathbf{U}_y$  corresponding to the true class label  $y$  of  $\mathbf{x}$ , averaged with the *soft assignment* denoiser associated to all subspaces  $\mathbf{U}_k$ , with weights given by the correlations of  $\mathbf{x}_t$  with these different subspaces. However, we can extract a more general form for the class-conditional denoiser in this example which is relevant for practical parameterization using the geometric structure of the mixture of Gaussians distribution, which actually parallels the kinds of geometric structure common in real-world data. More precisely, we add an additional assumption associated to the subspaces  $\mathbf{U}_k$  being ‘distinguishable’ from one another, which is natural in practice: specifically, we assume that for any pair of indices  $k, k' \in [K]$  with  $k \neq k'$ , we can find a set of  $K$  directions  $\mathbf{v}_k \in \mathbb{R}^D$  such that

$$\mathbf{U}_k \mathbf{U}_k^\top \mathbf{v}_k = \mathbf{v}_k, \quad \mathbf{U}_{k'} \mathbf{U}_{k'}^\top \mathbf{v}_k = \mathbf{0}, \quad k' \neq k. \quad (6.4.26)$$

This is a slightly stronger assumption than simple distinguishability, but it should be noted that it is not overly restrictive: for example, it still allows the subspaces  $\mathbf{U}_k$  to have significant correlations with one another.<sup>10</sup> These vectors  $\mathbf{v}_k$  can then be thought of as *embeddings* of the class label  $y \in [K]$ , and we can use them to define a more general operator that can represent both the unconditional and class-conditional denoisers. More precisely, consider the mapping

$$(\mathbf{x}_t, \mathbf{v}) \mapsto \sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_k \mathbf{U}_k^\top \mathbf{v}\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_i \mathbf{U}_i^\top \mathbf{v}\right)} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t. \quad (6.4.27)$$

If we substitute  $\mathbf{v} = \mathbf{v}_y$  for some  $y \in [K]$ , we get

$$\begin{aligned} \sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_k \mathbf{U}_k^\top \mathbf{v}_y\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_i \mathbf{U}_i^\top \mathbf{v}_y\right)} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t &= \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{v}_y\right)}{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{v}_y\right) + K - 1} \mathbf{U}_y \mathbf{U}_y^\top \mathbf{x}_t \\ &\quad + \sum_{k \neq y} \frac{1}{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{v}_y\right) + K - 1} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t. \end{aligned} \quad (6.4.28)$$

Now, because  $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \mathbf{g}$ , if the subspace dimension  $P$  is sufficiently large—for example, if we consider a large-scale, asymptotic regime where  $P, D \rightarrow \infty$  with their ratio  $P/D$  converging to a fixed constant—we have for  $t \approx 0$  that  $\|\mathbf{x}_t\|_2$  is close to  $\sqrt{P}$ , by the concentration of measure phenomenon<sup>11</sup>. Then by the argument in the previous paragraph, we have in this regime that for *almost all* realizations of  $\mathbf{x}_t$ , the following approximation holds:

$$\sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_k \mathbf{U}_k^\top \mathbf{v}_y\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_i \mathbf{U}_i^\top \mathbf{v}_y\right)} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t \approx \mathbf{U}_y \mathbf{U}_y^\top \mathbf{x}_t. \quad (6.4.29)$$

This argument shows that the operator (6.4.27) is, with overwhelming probability, *equal to the optimal class-conditional denoiser* (6.4.18) for  $y$  when  $\mathbf{v} = \mathbf{v}_y$ ! In intuitive terms, at small noise levels  $t \approx 0$ —corresponding to the structure-enforcing portion of the denoising process—plugging in the embedding for

<sup>10</sup>More generally, this assumption is naturally formulated as an *incoherence condition* between the subspaces  $\mathbf{U}_{[K]}$ , a familiar notion from the theory of compressive sensing.

<sup>11</sup>One of a handful of *blessings of dimensionality*—see Wright and Ma [WM22].

a given class  $y$  to the second argument of the operator (6.4.27) leads the resulting function of  $\mathbf{x}_t$  to well-approximate the optimal class-conditional denoiser (6.4.18) for  $y$ . Moreover, it is evident that plugging in  $\mathbf{v} = \mathbf{x}_t$  to the operator (6.4.27) yields (exactly) the optimal unconditional denoiser (6.4.19) for  $\mathbf{x}_t$ . Thus, this operator provides a unified way to parameterize the constituent operators in the optimal denoiser for  $\mathbf{x}$  *within a single ‘network’*: it is enough to add the output of an instantiation of (6.4.27) with input  $(\mathbf{x}_t, \mathbf{x}_t)$  to an instantiation with input  $(\mathbf{x}_t, \mathbf{v}_y)$ . The resulting operator is a function of  $(\mathbf{x}_t, y)$ , and computationally, the subspaces  $(\mathbf{U}_k)_{k=1}^K$  and embeddings  $y \mapsto \mathbf{v}_y$  become its learnable parameters. ■

Example 6.3 shows that in the special case of a low-rank mixture of Gaussians data distribution for  $\mathbf{x}$  with incoherent components, operators of the form

$$(\mathbf{x}_t, \mathbf{v}) \mapsto \sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_k \mathbf{U}_k^\top \mathbf{v}\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)} \mathbf{x}_t^\top \mathbf{U}_i \mathbf{U}_i^\top \mathbf{v}\right)} \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}_t \quad (6.4.30)$$

provide a sufficiently rich class of operators to parameterize the MMSE-optimal denoiser for noisy observations  $\mathbf{x}_t$  of  $\mathbf{x}$ , in the setting of classifier-free guidance where one network is to be used to represent both the unconditional and class-conditional denoisers for  $\mathbf{x}_t$ . For such operators, the auxiliary input  $\mathbf{v}$  can be taken as either  $\mathbf{x}_t$  or a suitable embedding of the class label  $y \mapsto \mathbf{v}_y$  in order to realize such a denoiser. Based on the framework in Chapter 4, which develops deep network architectures suitable for transforming more general data distributions to structured representations using the low-rank mixture of Gaussians model as a primitive, it is natural to imagine that operators of the type (6.4.30) may be leveraged in denoisers for general data distributions  $\mathbf{x}$  with low-dimensional geometrically-structured components that are sufficiently distinguishable (say, incoherent) from one another. The next section demonstrates that this is indeed the case.

## 6.4.2 Caption Conditioned Image Generation

In the previous subsection, we have formulated denoisers for class-conditional denoising with classifier-free guidance, a ubiquitous practical methodology used in the largest-scale diffusion models, and shown how to parameterize them (in Example 6.3) in the special case of a low-rank Gaussian mixture model data distribution. One interesting byproduct of this example is that it highlights the crucial role of *embeddings* of the class label  $y$  into a common space with the image  $\mathbf{x}$  in order to provide a concise and unified scheme for parameterizing the optimal denoisers (conditional and unconditional). Below, we will describe an early such instantiation, which formed the basis for the original open-source Stable Diffusion implementation [RBL+22]. In this setting, the embedding and subsequent conditioning is performed not on a class label, but a text prompt, which describes the desired image content (Figure 6.13). We denote the raw tokenized text prompt as  $\mathbf{Y} \in \mathbb{R}^{D_{\text{text}} \times N}$  in this context, since it corresponds to a sequence of vectors—in Section 7.4, we describe the process of encoding a text sequence as a vector representation in detail.

Stable Diffusion follows the conditional generation methodology we outline in Section 6.4.1, with two key modifications: (i) The conditioning signal is a tokenized text prompt  $\mathbf{Y}$ , rather than a class label; (ii) Image denoising is performed in “latent” space rather than on raw pixels, using a specialized, pretrained variational autoencoder pair  $f : \mathbb{R}^{D_{\text{img}}} \rightarrow \mathbb{R}^{d_{\text{img}}}, g : \mathbb{R}^{d_{\text{img}}} \rightarrow \mathbb{R}^{D_{\text{img}}}$  (see Section 5.1.4), where  $f$  is the encoder and  $g$  the decoder. Subsequent model development has shown that point (ii) is an efficiency issue, rather than a core conceptual one, so we will not focus on it, other than to mention that it simply leads to the following straightforward modifications to the text-to-image pipeline sketched in Figure 6.13:

1. **At training time**, the encoder  $f : \mathbf{x} \mapsto \mathbf{z}$  is used to generate the denoising targets, and all denoising is performed on the encoded representations  $\mathbf{z}_t \in \mathbb{R}^{d_{\text{img}}}$ ;
2. **At generation time**, sampling is performed on the representations  $\hat{\mathbf{z}}_t$ , and the final image is generated by applying the decoder  $g(\hat{\mathbf{z}}_0)$ .

In contrast, issue (i) is essential, and the approach proposed to treat it represents one of the lasting methodological innovations of Rombach et al. [RBL+22]. In the context of the iterative conditional denoising frame-

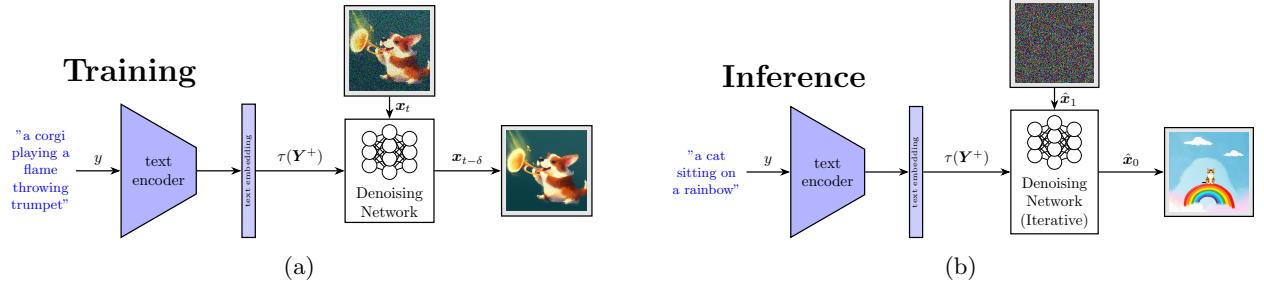


Figure 6.13: A high-level schematic of training and applying a text-to-image generative model, via conditional generation with a text prompt. **Left:** To train a text-to-image model, a large dataset of images paired with corresponding text captions is used. An encoder is used to map the captions to sequences of vectors, which are used as conditioning signals for a conditional denoiser, trained as described in Section 6.4.1. The text encoder may be pretrained and frozen, or jointly trained with the denoiser. **Right:** When applying a trained model, a desired text prompt is used as conditioning, then sampling is performed with the trained model, as in Algorithm 6.2 (*mutatis mutandis* for use with an encoded text prompt). For full details of the process of encoding text to a sequence of vectors, see Section 7.4.

work we have developed in Section 6.4.1, this concerns the parameterization of the denoisers  $\bar{z}_\theta(t, \mathbf{z}_t, \mathbf{Y}^+)$ .<sup>12</sup> Rombach et al. [RBL+22] implement text conditioning in the denoiser using a layer known as cross attention, inspired by the original encoder-decoder transformer architecture of Vaswani et al. [VSP+17]. Cross attention is implemented as follows. We let  $\tau : \mathbb{R}^{D_{\text{text}} \times N} \rightarrow \mathbb{R}^{d_{\text{model}} \times N_{\text{text}}}$  denote an encoding network for the text embeddings (often a causal transformer—see Section 7.4), and let  $\psi : \mathbb{R}^{d_{\text{img}}} \rightarrow \mathbb{R}^{d_{\text{model}} \times N_{\text{img}}}$  denote the mapping corresponding to one of the intermediate representations in the denoiser.<sup>13</sup> Here,  $N_{\text{text}}$  is the maximum tokenized text prompt length, and  $N_{\text{img}}$  roughly corresponds to the number of image channels (layer-dependent) in the representation, which is fixed if the input image resolution is fixed. Cross attention (with  $K$  heads, and no bias) is defined as

$$\text{MHCA}(\mathbf{z}_t, \mathbf{Y}^+) = \mathbf{U}_{\text{out}} \begin{bmatrix} \text{SA}([\mathbf{U}_{\text{qry}}^1]^\top \psi(\mathbf{z}_t), [\mathbf{U}_{\text{key}}^1]^\top \tau(\mathbf{Y}^+), [\mathbf{U}_{\text{val}}^1]^\top \tau(\mathbf{Y}^+)) \\ \vdots \\ \text{SA}([\mathbf{U}_{\text{qry}}^K]^\top \psi(\mathbf{z}_t), [\mathbf{U}_{\text{key}}^K]^\top \tau(\mathbf{Y}^+), [\mathbf{U}_{\text{val}}^K]^\top \tau(\mathbf{Y}^+)) \end{bmatrix}, \quad (6.4.31)$$

where SA denotes the ubiquitous self attention operation in the transformer (which we recall in detail in Chapter 7: see Equations (7.2.15) and (7.2.16)), and  $\mathbf{U}_*^k \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$  for  $* \in \{\text{qry}, \text{key}, \text{val}\}$  (as well as the output projection  $\mathbf{U}_{\text{out}}$ ) are the learnable parameters of the layer.

Notice that, by the definition of the self-attention operation, cross attention *outputs linear combinations of the value-projected text embeddings, weighted by correlations between the image features and the text embeddings*. In the denoiser architecture used by Rombach et al. [RBL+22], self-attention residual blocks in the denoiser architecture, applied to the image representation at the current layer and defined analogously to those in Equation (7.2.13) for the vision transformer, are followed by cross attention residual blocks of the form (6.4.31). Such a structure requires the text encoder  $\tau$  to, in a certain sense, share some structure in its output with the image feature embedding  $\psi$ : this can be enforced either by appropriate joint text-image pretraining (such as with CLIP [RKH+21]), or by joint training with the denoiser itself (which was proposed and demonstrated by Rombach et al. [RBL+22], but has fallen out of favor due to high data and training costs for strong performance). Conceptually, this joint text-image embedding space and the cross attention layer itself bear a strong resemblance to the conditional mixture of Gaussians denoiser that we derived in the previous section (recall (6.4.27)), in the special case of a single token sequence. Deeper connections can be drawn in the multi-token setting following the rate reduction framework for deriving deep

<sup>12</sup>In the setting of text conditioning, the ‘augmented’ label  $\mathbf{Y}^+$ , which is either the encoded text prompt or  $\emptyset$ , denoting unconditional denoising, is often implemented by mapping  $\emptyset$  to the empty string “”, then encoding this text prompt with the tokenizer as usual. This gives a simple, unified way to treat conditional and unconditional denoising with text conditioning.

<sup>13</sup>In practice, text-conditioned denoisers add cross attention layers at regular intervals within the forward pass of the denoiser, so  $\psi$  should be seen as layer-dependent, in contrast to  $\tau$ . See Rombach et al. [RBL+22] for details.

network architectures discussed in Chapter 4, and manifested in the derivation of the CRATE transformer-like architecture.

This same basic design has been further scaled to even larger model and dataset sizes, in particular in modern instantiations of Stable Diffusion [EKB+24], as well as in competing models such as FLUX.1 [LBB+25], Imagen [SCS+22], and DALL-E [RDN+22]. The conditioning mechanism of cross attention has also become ubiquitous in other applications, as in EgoAllo (Section 6.3.3) for conditioned pose generation and in Michelangelo [ZLC+23] for conditional 3D shape generation based on images or texts.

## 6.5 Conditional Inference with Measurement Self-Consistency

In this last section, we consider the more extreme, but actually ubiquitous, case for distribution learning in which we only have a set of observed samples  $\mathbf{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$  of the data  $\mathbf{x}$ , but no samples of  $\mathbf{x}$  directly! In general, the observation  $\mathbf{y} \in \mathbb{R}^d$  is of lower dimension than  $\mathbf{x} \in \mathbb{R}^D$ . To make the problem well-defined, we do assume that the observation model between  $\mathbf{y}$  and  $\mathbf{x}$  is known to belong to a certain family of analytical models, denoted as  $\mathbf{y} = h(\mathbf{x}, \theta) + \mathbf{w}$ , with  $\theta$  either known or not known.

Let us first try to understand the problem conceptually with the simple case when the measurement function  $h$  is known and the observed  $\mathbf{y} = h(\mathbf{x}) + \mathbf{w}$  is informative about  $\mathbf{x}$ . That is, we assume that  $h$  is surjective from the space of  $\mathbf{x}$  to that of  $\mathbf{y}$  and the support of the distribution  $\mathbf{y}_0 = h(\mathbf{x}_0)$  is low-dimensional. This typically requires the *extrinsic* dimension  $d$  of  $\mathbf{y}$  is higher than the *intrinsic* dimension of the support of the distribution of  $\mathbf{x}$ . Without loss of generality, we may assume that there exist functions:

$$F(\mathbf{x}) = \mathbf{0}, \quad G(\mathbf{y}) = \mathbf{0}. \quad (6.5.1)$$

Notice that here we may assume that we know  $G(\mathbf{y})$  but not  $F(\mathbf{x})$ . Let  $\mathcal{S}_{\mathbf{y}} \doteq \{\mathbf{y} \mid G(\mathbf{y}) = \mathbf{0}\}$  be the support of  $p(\mathbf{y})$ . In general,  $h^{-1}(\mathcal{S}_{\mathbf{y}}) = \{\mathbf{x} \mid G(h(\mathbf{x})) = \mathbf{0}\}$  is a super set of  $\mathcal{S}_{\mathbf{x}} \doteq \{\mathbf{x} \mid F(\mathbf{x}) = \mathbf{0}\}$ . That is, we have  $h(\mathcal{S}_{\mathbf{x}}) \subseteq \mathcal{S}_{\mathbf{y}}$ .

### 6.5.1 Linear Measurement Models

First, for simplicity, let us consider the measurement is a linear function of the data  $\mathbf{x}$  of interest:

$$\mathbf{y} = \mathbf{A}\mathbf{x}. \quad (6.5.2)$$

Here the matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  is of full row rank and  $m$  is typically smaller than  $n$ . We assume  $\mathbf{A}$  is known for now. We are interested in how to learn the distribution of  $\mathbf{x}$  from such measurements. Since we no longer have direct samples of  $\mathbf{x}$ , we wonder whether we can still develop a denoiser for  $\mathbf{x}$  with observations  $\mathbf{y}$ . Let us consider the following diffusion process:

$$\mathbf{y}_t = \mathbf{y}_0 + t\mathbf{g}, \quad \mathbf{y}_0 = \mathbf{A}(\mathbf{x}_0), \quad (6.5.3)$$

where  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

Without loss of generality, we assume  $\mathbf{A}$  is of full row rank, i.e., under-determined. Let us define the corresponding process  $\mathbf{x}_t$  as one that satisfies:

$$\mathbf{y}_t = \mathbf{A}\mathbf{x}_t. \quad (6.5.4)$$

From the denoising process of  $\mathbf{y}_t$ , we have

$$\mathbf{y}_{t-s} \approx \mathbf{y}_t + st\nabla \log p_t(\mathbf{y}_t). \quad (6.5.5)$$

Then we have:

$$\mathbf{A}\mathbf{x}_{t-s} \approx \mathbf{A}\mathbf{x}_t + st\nabla \log p_t(\mathbf{A}\mathbf{x}_t), \quad (6.5.6)$$

for a small  $s > 0$ . So  $\mathbf{x}_{t-s}$  and  $\mathbf{x}_t$  need to satisfy:

$$\mathbf{A}(\mathbf{x}_{t-s} - \mathbf{x}_t) \approx st\nabla \log p_t(\mathbf{A}\mathbf{x}_t). \quad (6.5.7)$$

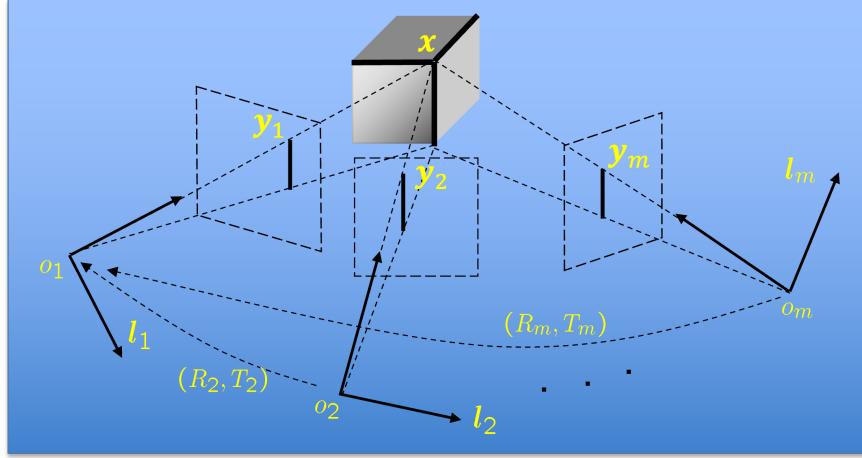


Figure 6.14: Relationship between a 3D object/scene and its 2D projections. Here we illustrate the projection of a point  $\mathbf{x}$  and a line intersecting the point.

Among all  $\mathbf{x}_{t-s}$  that satisfy the above constraint, we arbitrarily choose the one that minimizes the distance  $\|\mathbf{x}_{t-s} - \mathbf{x}_t\|_2^2$ . Therefore, we obtain a “denoising” process for  $\mathbf{x}_t$ :

$$\mathbf{x}_{t-s} \approx \mathbf{x}_t + st\mathbf{A}^\dagger \nabla \log p_t(\mathbf{A}\mathbf{x}_t). \quad (6.5.8)$$

Notice that this process does not sample from the distribution of  $\mathbf{x}_t$ . In particular, there are components of  $\mathbf{x}$  in the null space/kernel of  $\mathbf{A}$  which can never be recovered from observations. Thus more information is needed to recover the full distribution of  $\mathbf{x}$ , strictly speaking. But this recovers the component of  $\mathbf{x}$  that is orthogonal to the null space of  $\mathbf{A}$ .

### 6.5.2 3D Visual Model from Calibrated Images

In practice, the measurement model is often nonlinear or only partially known. A typical problem of this kind is actually behind how we can learn a working model of the external world from the images perceived, say through our eyes, telescopes or microscopes. In particular, humans and animals are able to build a model of the 3D world ((or 4D for a dynamical world) through a sequence of its 2D projections – a sequence of 2D images (or stereo image pairs). The mathematical or geometric model of the projection is generally known:

$$\mathbf{y}^i = h(\mathbf{x}, \theta^i) + \mathbf{w}^i, \quad (6.5.9)$$

where  $h(\cdot)$  represents a (perspective) projection of the 3D (or 4D) scene from a certain camera view at time  $t_i$  to a 2D image (or a stereo pair) and  $\mathbf{w}$  is some possibly additive small measurement noise. Figure 6.14 illustrates this relationship concretely, while Figure 6.15 illustrates the model problem in the abstract. A full exposition of geometry related to multiple 2D views of a 3D scene is beyond the scope of this book. Interested readers may refer to the book [MKS+04]. For now, all we need to proceed is that such projections are well understood and multiple images of a scene contains sufficient information about the scene.

In general, we would like to learn the distribution  $p(\mathbf{x})$  of the 3D (or 4D) world scene  $\mathbf{x}$ <sup>14</sup> from the perceived 2D images of the world so far. The primary function of such a (visual) world model is to allow us to recognize places where we had been before or predict what the current scene would look alike in a future time at a new viewpoint.

Let us first examine the special but important case of stereo vision. In this case, we have two calibrated views of the 3D scene  $\mathbf{x}$ :

$$\mathbf{y}^0 = h(\mathbf{x}, \theta^0) + \mathbf{w}^0, \quad \mathbf{y}^1 = h(\mathbf{x}, \theta^1) + \mathbf{w}^1, \quad (6.5.10)$$

<sup>14</sup>Here by abuse of notation, we use  $\mathbf{x}$  to represent either a point in 3D or a sample of an entire 3D object or a scene which consists of many points.

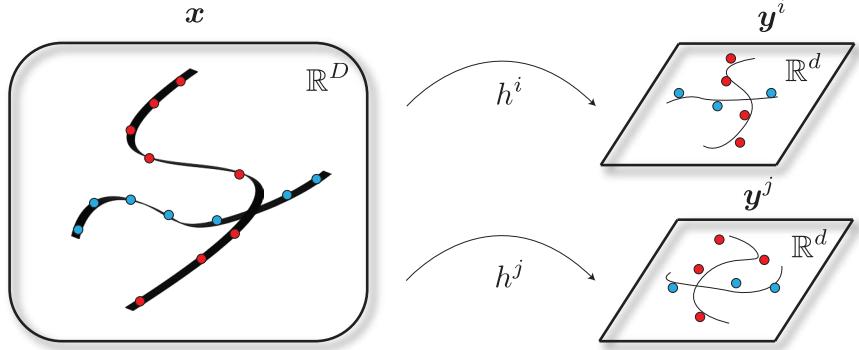


Figure 6.15: **Inference with distributed measurements.** We have a low-dimensional distribution  $\mathbf{x}$  (here, similarly to Figure 6.1, depicted as a union of two 2-dimensional manifolds in  $\mathbb{R}^3$ ) and a measurement model  $\mathbf{y}^i = h^i(\mathbf{x}) + \mathbf{w}^i$ . As before, we want to infer various properties of the conditional distribution of  $\mathbf{x}$  given  $\mathbf{y}$ , where  $\mathbf{y}$  is the collection of all the measurements  $\mathbf{y}^i$ .

where parameters  $\theta_0$  and  $\theta_1$  for the view poses can be assumed to be known.  $\mathbf{y}^0$  and  $\mathbf{y}^1$  are two 2D-projections of the 3D scene  $\mathbf{x}$ . We may also assume that they have the same marginal distribution  $p(\mathbf{y})$  and we have learned a diffusion and denoising model for it. That, we know the denoiser:

$$\mathbb{E}[\mathbf{y} \mid \mathbf{y}_t = \boldsymbol{\nu}] = \boldsymbol{\nu} + t^2 \nabla_{\boldsymbol{\nu}} \log p_t(\boldsymbol{\nu}). \quad (6.5.11)$$

Or, furthermore, we may assume that we have a sufficient number of samples of stereo pairs  $(\mathbf{y}^0, \mathbf{y}^1)$  and have also learned the joint distribution of the pairs. By a little abuse of notation, we also use  $\mathbf{y} = h(\mathbf{x})$  to indicate the pair  $\mathbf{y} = (\mathbf{y}^0, \mathbf{y}^1)$  and  $p(\mathbf{y})$  as the learned probability distribution of the pair (say via a denoiser as above).

The main question now is: How to learn (a representation for) the distribution of the 3D scene  $\mathbf{x}$  from its two projections with known relationships? People might question the rationale for doing this: why this is necessary if the function  $h(\cdot)$  is largely invertible? That is, the observation  $\mathbf{y}$  can largely determine the unknown  $\mathbf{x}$ , which is kind of the case for stereo – in general, two (calibrated) images contain sufficient information about the scene depth, from the given vintage point. However, 2D images are far from the most compact representation of the 3D scene as the same scene can produce infinite many (highly correlated) 2D images or image pairs. In fact, a good representation of a 3D scene should be invariant to the view point. Hence, a correct representation of the distribution of 3D scenes should be much more compact and structured than the distribution of 2D images, stereo pairs, or image-depth pairs.

Consider the (inverse) denoising processing for the diffusion:  $\mathbf{y}_t = \mathbf{y} + t\mathbf{g}$  in (6.5.11), where  $\mathbf{g}$  is standard Gaussian. From the denoising process of (6.5.11), we have

$$\mathbf{y}_{t-s} = \mathbf{y}_t + st \nabla_{\mathbf{y}} \log p_t(\mathbf{y}_t). \quad (6.5.12)$$

We try to find a corresponding “denoising” process of  $\mathbf{x}_t$  such that  $\mathbf{x}$  is related to  $\mathbf{y}$  as:

$$\mathbf{y} = h(\mathbf{x}). \quad (6.5.13)$$

Then we have:

$$h(\mathbf{x}_{t-s}) \approx h(\mathbf{x}_t) + st \nabla_{\mathbf{y}} \log p_t(h(\mathbf{x}_t)), \quad (6.5.14)$$

for a small  $s > 0$ . Suppose  $\mathbf{x}_{t-s} = \mathbf{x}_t + s\mathbf{v}$  for some vector  $\mathbf{v}$  and small increment  $s$ . We have

$$h(\mathbf{x}_{t-s}) \approx h(\mathbf{x}_t) + \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}_t) \cdot s\mathbf{v} \doteq h(\mathbf{x}_t) + \mathbf{A}(\mathbf{x}_t)s\mathbf{v}. \quad (6.5.15)$$

Hence, we have

$$\mathbf{A}(\mathbf{x}_t)\mathbf{v} = t \nabla_{\mathbf{y}} \log p_t(h(\mathbf{x}_t)). \quad (6.5.16)$$

Geometrically the vector  $\mathbf{v}$  in the domain of  $\mathbf{x}$  can be viewed as the pullback of the vector field  $t\nabla \log p_t(\mathbf{y})$  under the map  $\mathbf{y} = h(\mathbf{x})$ . In general, as before, we may (arbitrarily) choose  $\mathbf{v}$  to be the minimum 2-norm vector that satisfies the pullback relationship. Hence, we can express  $\hat{\mathbf{x}}_{t-s}$  approximately as:

$$\hat{\mathbf{x}}_{t-s} \approx \mathbf{x}_t + st\mathbf{A}(\mathbf{x}_t)^\dagger \nabla_{\mathbf{y}} \log p_t(h(\mathbf{x}_t)). \quad (6.5.17)$$

*Remark 6.2* (Parallel Sensing and Distributed Denoising.). There is something very interesting about the above equation (6.5.17). It seems to suggest we could try to learn the distribution of  $\mathbf{x}$  through a process that coupled with (many of) its (partial) observations:

$$\mathbf{y}^i = h^i(\mathbf{x}) + \mathbf{w}^i, i = 1, \dots, K. \quad (6.5.18)$$

In this case, we obtain a set of equations that the vector field  $\mathbf{v}$  in the domain of  $\mathbf{x}$  should satisfy:

$$\mathbf{A}^i(\mathbf{x}_t)\mathbf{v} = t\nabla_{\mathbf{y}^i} \log p_t(h^i(\mathbf{x}_t)), \quad (6.5.19)$$

where  $\mathbf{A}^i(\mathbf{x}_t) = \frac{\partial h^i}{\partial \mathbf{x}}(\mathbf{x}_t)$ . The final  $\mathbf{v}$  can be chosen as a “centralized” solution that satisfies all the above equations, or it could be chosen as a certain (stochastically) “aggregated” version of all  $\mathbf{v}^i$ :

$$\mathbf{v}^i = t\mathbf{A}^i(\mathbf{x}_t)^\dagger [\nabla_{\mathbf{y}^i} \log p_t(h^i(\mathbf{x}_t))], \quad i = 1, \dots, K, \quad (6.5.20)$$

that are computed in a parallel and distributed fashion? An open question here is, exactly what the so-defined “denoising” process for  $\mathbf{x}_t$  converges to, even in the linear measurement model case? When would it converge to a distribution that has the same low-dimensional support as the original  $\mathbf{x}_0$ , as  $\mathbf{y}_t$  converges to  $\mathbf{y} = h(\mathbf{x}_0)$ ?

**Visual World Model from Uncalibrated Image Sequences** In the above derivation, we have assumed that the measurement model  $h(\cdot)$  is fully known. In the case of stereo vision, this is rather reasonable as the relative pose (and calibration) of the two camera views (or two eyes<sup>15</sup>) are usually known in advance. Hence, through the stereo image pairs, in principle we should be able to learn the distribution of 3D scenes, at least the ego-centric distribution of 3D scenes. However, the low-dimensional structures of the so-called learned distribution contains variation caused by changing the viewpoints. That is, the appearance of the stereo images varies when we change our viewpoints with respect to the same 3D scene. For many practical vision tasks (such as localization and navigation), it is important if we can decouple this variation of viewpoints from an invariant representation of (the distribution of) 3D scenes.

*Remark 6.3.* Note that the above goal aligns well with Klein’s Erlangen Program for modern geometry, which is to study invariants of a manifold under a group of transformations. Here, we may view the manifold of interest as the distribution of ego-centric representations of 3D scenes. We have learned that it admits a group of three-dimensional rigid-body motion acting on it. It is remarkable that our brain has learned to effectively decouple such transformations from the observed 3D world.

Notice that we have studied learning representations that are invariant to translation and rotation in a limited setting in Chapter 4. We know that the associated compression operators take the necessary form of (multi-channel) convolutions, hence leading to the (deep) convolution neural networks. Nevertheless, operators that are associated with compression or denoising that are invariant to more general transformation groups remain elusive to characterize [CW16b]. For the 3D Vision problem in its most general setting, we know the change of our viewpoints can be well modeled as a rigid-body motion. However, the exact relative motion of our eyes between different viewpoints is usually not known. More generally, there could also be objects (e.g., cars, humans, hands) moving in the scene and we normally do not know their motion either. How can we generalize the problem of learning the distribution of 3D scenes with calibrated stereo pairs to such more general settings? More precisely, we want to learn a compact representation  $\mathbf{x}$  of the 3D scenes that is invariant to the camera/eye motions. Once such a representation is learned, we could sample and generate a 3D scene and render images or stereo pairs from arbitrary poses.

To this end, not that we can model a sequence of stereo pairs as:

$$\mathbf{y}^k = h(\mathbf{x}^k, \theta^k), \quad k = 1, \dots, K, \quad (6.5.21)$$

<sup>15</sup>The relative pose of our two eyes is well known to our brain.

where  $h(\cdot)$  represents the projection map from 3D to 2D.  $\theta^k$  denotes the rigid-body motion parameters of the  $k$ th view, with respect to some canonical frame in the world.  $\mathbf{x}^k$  represents the 3D scene at time  $k$ . If the scene is static,  $\mathbf{x}^k$  should all be the same  $\mathbf{x}^k = \mathbf{x}$ . To simplify the notation, we may denote the set of  $k$  equations as one:

$$\mathbf{Y} = H(\mathbf{x}, \Theta). \quad (6.5.22)$$

We may assume that we are given many samples of such stereo image sequences  $\{\mathbf{Y}_i\}$ . The problem is how to recover the associated motion sequence  $\{\Theta_i\}$  and learn the distribution of the scene  $\mathbf{x}$  (that is invariant to the motion). To our best knowledge, this remains an open challenging problem, probably as the final frontier for the 3D Vision problem.

## 6.6 Summary and Notes

**Measurement matching without clean samples.** In our development of conditional sampling, we considered measurement matching under an observation model (6.3.9), where we assume that we have paired data  $(\mathbf{x}, \mathbf{y})$ —i.e., ground truth for each observation  $\mathbf{y}$ . In many practically-relevant inverse problems, this is not the case: one of the most fundamental examples is in the context of compressed sensing, which we recalled in Chapter 2, where we need to reconstruct  $\mathbf{x}$  from  $\mathbf{y}$  using prior knowledge about  $\mathbf{x}$  (i.e., sparsity). In the setting of denoising-diffusion, we have access to an implicit prior for  $\mathbf{x}$  via the learned denoisers  $\bar{\mathbf{x}}_\theta(t, \xi)$ . Can we still perform conditional sampling without access to ground truth samples  $\mathbf{x}$ ?

For intuition as to why this might be possible, we recall a classical example from statistics known as Stein’s unbiased risk estimator (SURE). Under an observation model  $\mathbf{x}_t = \mathbf{x} + t\mathbf{g}$  with  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  and  $t > 0$ , it turns out that for any weakly differentiable  $f : \mathbb{R}^D \rightarrow \mathbb{R}^D$ ,

$$\mathbb{E}_{\mathbf{g}} \left[ \|\mathbf{x} - f(\mathbf{x} + t\mathbf{g})\|_2^2 \right] = \mathbb{E}_{\mathbf{g}} \left[ \|\mathbf{x} + t\mathbf{g} - f(\mathbf{x} + t\mathbf{g})\|_2^2 + 2t^2 \nabla \cdot f(\mathbf{x} + t\mathbf{g}) \right] - t^2 D, \quad (6.6.1)$$

where  $\nabla \cdot$  denotes the divergence operator:

$$\nabla \cdot f = \sum_{i=1}^D \partial_i f_i.$$

The  $\mathbf{x}$ -dependent part of the RHS of Equation (6.6.1) is called Stein’s unbiased risk estimator (SURE). If we take expectations over  $\mathbf{x}$  in Equation (6.6.1), note that the RHS can be written as an expectation with respect to  $\mathbf{x}_t$ —in particular, the mean-squared error of *any denoiser*  $f$  can be estimated *solely from noisy samples!* This remarkable fact, in refined forms, constitutes the basis for many practical techniques for performing image restoration, denoising-diffusion, etc. using only noisy data: notable examples include the “noise2noise” paradigm [LMH+18] and Ambient Diffusion [DSD+23a].

As a fun aside, we point out that Equation (6.6.1) leads to an alternate proof of Tweedie’s formula (Theorem 3.3). At a high level, one takes expectations over  $\mathbf{x}$  and expresses the main part of the RHS of Equation (6.6.1) equivalently, via integration by parts, as

$$\mathbb{E}_{\mathbf{x}_t} \left[ \|\mathbf{x}_t - f(\mathbf{x}_t)\|_2^2 + 2t^2 \nabla \cdot f(\mathbf{x}_t) \right] = \mathbb{E}_{\mathbf{x}_t} \left[ \|\mathbf{x}_t - f(\mathbf{x}_t)\|_2^2 \right] - 2t^2 \int \langle \nabla p_{\mathbf{x}_t}(\xi), f(\xi) \rangle d\xi. \quad (6.6.2)$$

This is a quadratic function of  $f$ , and formally taking derivatives gives that the optimal  $f$  satisfies Tweedie’s formula (Theorem 3.3). This argument can be made rigorous using basic ideas from the calculus of variations.

**Corrections to the Diffusion Posterior Sampling (DPS) approximation.** In Example 6.2 and in particular in Figure 6.9, we pointed out a limitation of the DPS approximation Equation (6.3.26) at small levels of measurement noise. This limitation is well-understood, and a principled approach to ameliorating it has been proposed by Rozet et al. [RAL+24]. The approach involves incorporating an additional estimate for the variance of the noisy posterior  $p_{\mathbf{x}|\mathbf{x}_t}$  to Equation (6.3.26)—we refer to the paper for details. Natural estimates for the posterior variance are slightly less scalable than DPS itself, due to the need to invert an affine transformation of the Jacobian of the posterior denoiser  $\mathbb{E}[\mathbf{x} | \mathbf{x}_t = \xi]$  (a large matrix). This is done relatively efficiently by Rozet et al. [RAL+24] using autodifferentiation and an approximation for the inverse based on conjugate gradients. It seems that it should be possible to improve further over this approach (say, using classical ideas from second-order optimization).

**More about measurement matching and diffusion models for inverse problems.** Diffusion models have become an extremely popular tool for solving inverse problems arising in scientific applications. Many more methods beyond the simple DPS algorithm we have presented in Algorithm 6.1 have been developed and continue to be developed, as the area is evolving rapidly. Popular and performant classes of approaches beyond DPS, which we have presented due to its generality, include variable splitting approaches like DAPS [ZCB+24], which allow for specific measurement constraints to be enforced much more strongly than in DPS, and exact approaches that can avoid the use of approximations like in DPS, such as TDS [WTN+23]. For more on this area, we recommend [ZCZ+25], which functions simultaneously as a survey and a benchmark of several popular methods on specific scientific inverse problem datasets.

## 6.7 Exercises and Extensions

*Exercise 6.1* (Posterior Variance Correction to DPS). 1. Using the code provided in the book GitHub for implementing Figure 6.9, implement the posterior variance correction proposed by Rozet et al. [RAL+24].

2. Verify that it ameliorates the posterior collapse at low noise variance issue observed in Figure 6.9.
3. Discuss any issues of sampling correctness that are retained or introduced by the corrected method, as well as its efficiency, relative to diffusion posterior sampling (DPS).

*Exercise 6.2* (Conditional Sampling on MNIST). 1. Train a simple classifier for the MNIST dataset, using an architecture of your choice. Additionally train a denoiser suitable for use in conditional sampling (Algorithm 6.2, since this denoiser can be used for unconditional denoising as well).

2. Integrate the classifier into a conditional sampler based on classifier guidance, as described in the first part of Section 6.4.1. Evaluate the resulting samples in terms of faithfulness to the conditioning class (visually; in terms of nearest neighbor; in terms of the output of the classifier).
3. Integrate the classifier into a conditional sampler based on classifier-free guidance, as described in Section 6.4.1 and Algorithm 6.2. Perform the same evaluation as in the previous step, and compare the results.
4. Repeat the experiment on the CIFAR-10 dataset.



# Chapter 7

# Learning Representations for Real-World Data

*“The best theory is inspired by practice, and the best practice is inspired by theory.”*

— Donald Knuth

The previous chapters have presented a systematic introduction to mathematical problems, computational frameworks, and practical algorithms associated with learning low-dimensional distributions from high-dimensional data. Although most theoretical justifications of these methods have been established for idealistic models of data such as (mixtures of) subspaces and/or Gaussians, the principles and ideas behind these computational methods are nevertheless powerful and general, and they are in fact meant to be applicable for real-world datasets and tasks.

To help readers understand the material in this book better and learn how to apply what you have learned so far to real-world data, towards the end of the book, we provide some demonstrations and vignettes of several representative applications. Each application proposes a solution to a real-world task with a real-world dataset (such as visual data and text data), using the methods we have introduced in this book. The results presented in this chapter are meant to serve the following two purposes:

- firstly, to provide additional experimental details and empirical evidence which validate the methods presented earlier in the book, and demonstrate their significant potential in real-world contexts;
- secondly, to introduce the reader to certain modern empirical methods and tasks in deep learning which are not well-documented outside of research or production codebases.

However, in our honest opinion, the solutions and results given here are designed simply to verify that the methodology works. As such, there is great room for future improvement, both in engineering and theoretical understanding, to potentially improve the state-of-the-art. We will discuss some future directions in Chapter 8.

## 7.1 Technical Setup and Outline of the Chapter

In previous chapters, we alluded to different setups in which we used representation-learning techniques to process real data at scale. In this chapter, we will describe such setups in great detail. The objective of this section is to get you, the reader, to be able to reproduce any experiment discussed in this section (or indeed the book) using just the description we will give in the book, the principles introduced in previous chapters and expanded on in this chapter, and hyperparameters taken from a smattering of papers whose results are discussed in this chapter. To this end, we will *precisely* describe all procedures in a detailed language, pseudocode, or mathematical notation that can be directly implemented in code. Wherever possible, we will discuss how the concrete implementations connect to the principles presented earlier in the book.

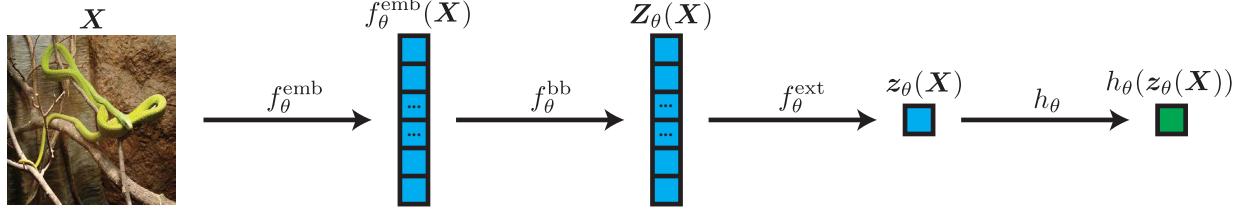


Figure 7.1: **A diagram of the encoder pipeline.** Data  $\mathbf{X} \in \mathcal{D}$  is fed through the embedding  $f_\theta^{\text{emb}}$  to get a sequence in  $(\mathbb{R}^d)^*$ . The embedding is fed through a backbone  $f_\theta^{\text{bb}}$  to get features  $Z_\theta(\mathbf{X})$  for each token. We can extract an aggregate feature  $z_\theta(\mathbf{X})$  using the extraction map  $f_\theta^{\text{ext}}$ . Finally, to use the aggregate feature in downstream tasks, we can use the task-specific head  $h_\theta$ .

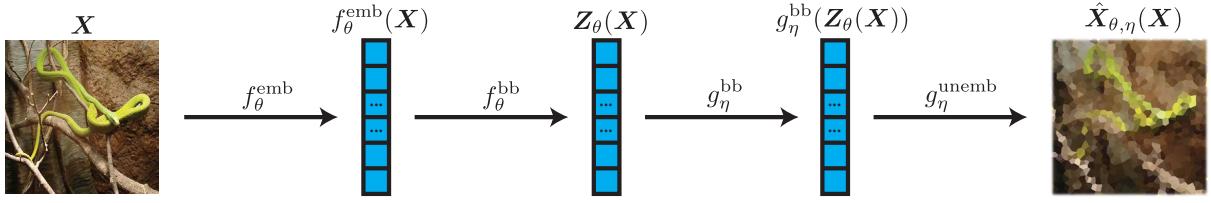


Figure 7.2: **A diagram of the autoencoder pipeline.** Data  $\mathbf{X} \in \mathcal{D}$  is fed through the embedding  $f_\theta^{\text{emb}}$  to get a sequence in  $(\mathbb{R}^d)^*$ . The embedding is fed through an encoder backbone  $f_\theta^{\text{bb}}$  to get features  $Z_\theta(\mathbf{X})$  for each token. To decode  $Z_\theta(\mathbf{X})$ , we pass it through a decoder backbone  $g_\eta^{\text{bb}}$ . To map the decoder backbone output back to data space  $\mathcal{D}$ , we use an *unembedding* layer  $g_\eta^{\text{unemb}}$ , overall obtaining a reconstruction  $\hat{X}_{\theta,\eta}(\mathbf{X})$  (here stylized to be a pixelated reconstruction of the input).

Let us define the set of possible data as  $\mathcal{D}$  (eventually this will be the set of images  $\mathcal{I}$ , for example, or the set of text  $\mathcal{T}$ ), and the set of finite sequences of tokens in  $\mathbb{R}^d$  (i.e., the set of matrices with  $d$  rows) as  $(\mathbb{R}^d)^* \doteq \bigcup_{T=1}^{\infty} \mathbb{R}^{d \times T}$ . In order to discuss the wealth of applications we introduce in this chapter, we first recall that in the rest of the book, we discuss two different types of model architectures.

- An *encoder* architecture, parameterized by  $\theta$ , which is composed of several components:
  - An *embedding*  $f_\theta^{\text{emb}}: \mathcal{D} \rightarrow (\mathbb{R}^d)^*$ , which converts the input data  $\mathcal{D}$  into a series of *tokens* which are mapped into, or *embedded* in,  $D$ -dimensional space. *In the rest of the chapter, we will often identify tokens and embeddings with each other.*
  - An *encoder backbone*  $f_\theta^{\text{bb}}: (\mathbb{R}^d)^* \rightarrow (\mathbb{R}^d)^*$ , which processes the series of embeddings using a sequence-to-sequence operation. This backbone is implemented by the network architectures discussed in the previous chapters, but we will give a more formal description as we go along.
  - A *aggregate feature extractor*  $f_\theta^{\text{ext}}: (\mathbb{R}^d)^* \rightarrow \mathbb{R}^d$ , which extracts an aggregate representation of the whole sequence. This is used to define a single feature for the entire data sample.
  - A *task-specific head*  $h_\theta: \mathbb{R}^d \rightarrow \mathbb{R}^m$ , which extracts an  $m$ -dimensional output for prediction.

We also define  $f_\theta \doteq f_\theta^{\text{bb}} \circ f_\theta^{\text{emb}}: \mathcal{D} \rightarrow (\mathbb{R}^d)^*$ . Given an input  $\mathbf{X}$ , we write  $Z_\theta(\mathbf{X}) \doteq f_\theta(\mathbf{X})$  and  $z_\theta(\mathbf{X}) \doteq f_\theta^{\text{ext}}(Z_\theta(\mathbf{X}))$ . The overall pipeline is depicted in Figure 7.1.

- An *autoencoder* architecture, which is composed of several components:
  - An *embedding*  $f_\theta^{\text{emb}}: \mathcal{D} \rightarrow (\mathbb{R}^d)^*$ , which converts the input data  $\mathcal{D}$  into a series of *tokens* which are embedded in  $D$ -dimensional space.
  - An *encoder backbone*  $f_\theta^{\text{bb}}: (\mathbb{R}^d)^* \rightarrow (\mathbb{R}^d)^*$ , which processes the series of embeddings using a sequence-to-sequence operation.
  - A *decoder backbone*  $g_\eta^{\text{bb}}: (\mathbb{R}^d)^* \rightarrow (\mathbb{R}^d)^*$ , which conceptually undoes the operation of the encoder backbone.

- An *unembedding*  $g_\eta^{\text{unemb}} : (\mathbb{R}^d)^* \rightarrow \mathcal{D}$ , which acts as an inverse of the embedding.

We also define  $f_\theta \doteq f_\theta^{\text{bb}} \circ f_\theta^{\text{emb}} : \mathcal{D} \rightarrow (\mathbb{R}^d)^*$  and  $g_\eta \doteq g_\eta^{\text{unemb}} \circ g_\eta^{\text{bb}} : (\mathbb{R}^d)^* \rightarrow \mathcal{D}$ . Given an input  $\mathbf{X}$ , we write  $\mathbf{Z}_\theta(\mathbf{X}) \doteq f_\theta(\mathbf{X})$  and  $\hat{\mathbf{X}}_{\theta,\eta}(\mathbf{X}) \doteq g_\eta(\mathbf{Z}_\theta(\mathbf{X}))$ . The overall pipeline is depicted in Figure 7.2.

We will repeatedly use this notation many times in this chapter, so please feel free to refer back to it if something doesn't make sense. This decomposition of our networks also closely mirrors most code implementations, and you can start your coding projects by defining these networks.

In this chapter, we will discuss applications of the book's principles to contrastive learning in Section 7.2. This will serve as both an introduction to imagery data, data augmentation techniques, and the common architecture known as the transformer, *as well as* a first demonstration of the drastic kinds of simplifications we can make using the demonstrated principles. We will continue with modifications to the *network architecture* in Sections 7.3 and 7.4, which demonstrate the capabilities of simplified architectures for *encoding* within the image and text domains. We then demonstrate simplified architectures for *autoencoding* in Section 7.6.

## 7.2 Simplified Contrastive Learning

Learning high-quality and faithful representations of data is a fundamental problem in deep learning, known as *self-supervised learning*. There have been many approaches proposed for this task, many of which do not evidently use the techniques and principles outlined in this manuscript. One such approach is called *contrastive learning*, so named because the learning objective is (roughly speaking) about ensuring that features of “similar” data are similar, and features of “dis-similar” data are far apart. Contrastive learning solutions are often highly-engineered, empirically designed approaches. In this section, we will describe one such approach named DINO [CTM+21], and use the principles described in the previous chapters to drastically simplify their design decisions while improving the learned representations.

### 7.2.1 Data

The data that we will use to explore and simplify the DINO methodology is all 2-dimensional image data. For *training*, we will use the ImageNet-1K and ImageNet-21K datasets. Each sample in the dataset is an RGB image, of varying resolution, and a label indicating the object or scene that the image contains (i.e., the *class* of the image). The ImageNet-1K dataset contains 1.28M training images and 50K validation images partitioned into 1K classes. The ImageNet-21K dataset contains 14.2M training images and 21.8K classes, but the classes are not disjoint (i.e., some classes are subsets of others). Since we are doing self-supervised learning, the labels will not be used during training, only during evaluation. For *evaluation*, we will use a litany of datasets. Of these, the most common is CIFAR-10. CIFAR-10 is a dataset of 60K RGB  $32 \times 32$  natural images partitioned into 10 classes, with a pre-established training set of 50K samples and a validation set of 10K samples. The purpose of using CIFAR-10 will be to ensure that models which train on one distribution of images (ImageNet) can generalize to another distribution of images (CIFAR-10). We also refer to other similar datasets, such as CIFAR-100 (disjoint from CIFAR-10), Oxford Flowers, and Oxford Pets. Exemplars of ImageNet-1K and CIFAR-10 data are shown in Figure 7.3. In order to increase robustness of our model, we often apply *small data augmentations* to each image during processing, such as flips, added small random noise, or random large crops; we do not include this in our notation, as each augmentation of a natural image is itself (very close to) a natural image in our dataset.

On a slightly more formal level, our data  $\mathbf{X}$  will be images; we let  $\mathcal{I}$  be the set of all images. Since an image is a rectangular array of pixels, and each pixel has a color given by RGB, CMYK, or another color format, we say that an image is an element of  $\mathbb{R}^{c \times h \times w}$  — here  $c$  is the number of channels (i.e., 3 for RGB and 4 for CMYK),  $h$  is the image height, and  $w$  is the image width. Consequently, the set of all images  $\mathcal{I} \doteq \bigcup_{c,h,w=1}^{\infty} \mathbb{R}^{c \times h \times w}$  is the set of all possible such data. Again, we will use this notation repeatedly.

### 7.2.2 Task and Objective Function

Our task is to learn a good representation of the data. Contrastive learning, by and large, does this by defining what properties of the input image we wish the features to reflect, construct images which share

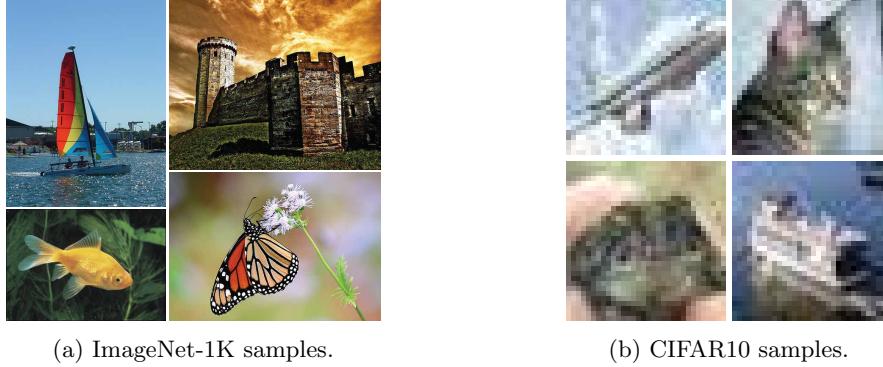


Figure 7.3: **Images from ImageNet-1K (left) and CIFAR-10 (right).** Notice that the CIFAR-10 images are much lower resolution, generally speaking, reducing the complexity of learning that distribution.

these properties but vary others, and set up a loss which promotes that the features of images with shared properties are close and images with different properties are different. The naturally optimal solution to this learning problem is that the learned features preserve the desired properties of the input. However, there are many practical and empirical complications that arise in the course of training contrastive models.

In the case of DINO, the authors propose to use a methodology which produces a single feature vector for the whole image and desires the feature vector to contain “global” (i.e., image-level) information. Accordingly, the loss will promote that images with similar global information have similar features and images with different global information have different features.

This seems intuitive, but as previously mentioned, there are several empirical considerations, even while setting up the loss. First and foremost, how should we promote similarities and differences? The answer of DINO [CTM+21] is<sup>1</sup> to convert the output features into “logits” corresponding to some probability distribution and take their cross-entropy. More specifically, let  $\Delta_m \doteq \{\mathbf{x} \in \mathbb{R}^m : x_i \geq 0 \forall i \in [m], \sum_{i=1}^m x_i = 1\}$  be the space of probability vectors in  $\mathbb{R}^m$  and define the function  $d_{CE} : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$  by

$$d_{CE}(\mathbf{p}, \mathbf{q}) \doteq \text{CE}(\mathbf{p}, \mathbf{q}), \quad \forall \mathbf{p}, \mathbf{q} \in \Delta_m \quad (7.2.1)$$

where  $\text{CE} : \Delta_m \times \Delta_m \rightarrow \mathbb{R}$  is the cross-entropy, defined as

$$\text{CE}(\mathbf{p}, \mathbf{q}) \doteq - \sum_{i=1}^m p_i \log q_i, \quad \forall \mathbf{p} = (p_1, \dots, p_m), \mathbf{q} = (q_1, \dots, q_m) \in \Delta_m. \quad (7.2.2)$$

Before we continue our discussion, let us build some intuition about this distance function. We have, in particular,

$$\text{CE}(\mathbf{p}, \mathbf{q}) = - \sum_{i=1}^m p_i \log q_i = \sum_{i=1}^m p_i \log(p_i/q_i) - \sum_{i=1}^m p_i \log p_i = \text{KL}(\mathbf{p} \parallel \mathbf{q}) + H(\mathbf{p}) \quad (7.2.3)$$

where  $\text{KL} : \Delta_m \times \Delta_m \rightarrow \mathbb{R}$  is the KL divergence, defined as

$$\text{KL}(\mathbf{p} \parallel \mathbf{q}) \doteq \sum_{i=1}^m p_i \log(p_i/q_i), \quad (7.2.4)$$

and  $H : \Delta_m \rightarrow \mathbb{R}$  is the entropy of a random variable. Note that  $\text{KL}(\mathbf{p} \parallel \mathbf{q})$  is minimized if and only if  $\mathbf{p} = \mathbf{q}$ . So minimizing  $d_{CE}$  does two things: it makes  $\mathbf{p} = \mathbf{q}$ , and it makes  $\mathbf{p}$  and  $\mathbf{q}$  have minimal entropy (i.e., vectors with 1 in one component and 0 elsewhere — these are called *one-hot vectors*). Overall, the goal of this objective is not *just* to match  $\mathbf{p}$  and  $\mathbf{q}$  but also to shape them in a certain way to make them low-entropy. Keep this in mind when we discuss the formulation.

<sup>1</sup>In the author’s view, inexplicably...

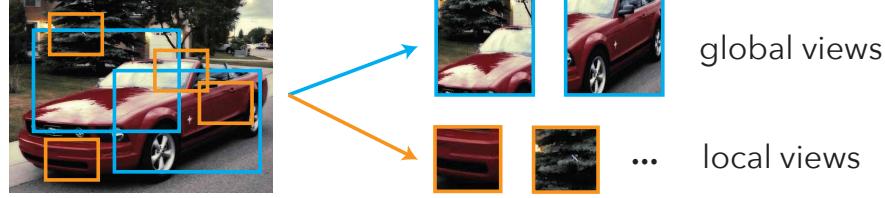


Figure 7.4: **Local and global views in DINO.** Local views and global views take a rectangular crop of the input image and resize it to a square shape, which is then input into the network for processing.

The next question is, how should we obtain samples with similar global information? The answer of DINO (as well as nearly all contrastive learning) is *data augmentation* — from each sample, make several correlated samples which share the desired properties. In the DINO case, we use different crops or *views* of the input image. Recall that we model an image as an element of the set  $\mathcal{I}$ . In this notation, a view is a function  $v: \mathcal{I} \rightarrow \mathcal{I}$ . In the DINO case, the view is a *random resized crop*: it takes a randomly chosen rectangular crop of the image (which has a fixed percentage  $p_v \in [0, 1]$  of the total area of the image), resize it proportionally so that the *shorter edge* is  $S_{\text{rsz}}$  pixels long, then resizes it to a fixed shape  $(C, S_v, S_v)$  where  $S_v \geq 1$  is the size of the view and  $C$  is the number of channels in the original image.

There are two types of views we want to use, depicted in Figure 7.4:

- *global views*, which are random resized crops with area percentage parameter  $p_{\text{glo}} \in [0, 1]$  and output shape  $(C, S_{\text{glo}}, S_{\text{glo}})$ ;
- and *local views*, which are random resized crops with area percentage parameter  $p_{\text{loc}} \in [0, 1]$  and output shape  $(C, S_{\text{loc}}, S_{\text{loc}})$ . Here  $p_{\text{loc}} < p_{\text{glo}}$  and  $S_{\text{loc}} < S_{\text{glo}}$ .

DINO desires that the aggregate features  $\mathbf{z}_\theta(\mathbf{X}_v) \doteq (f_\theta^{\text{ext}} \circ f_\theta)(\mathbf{X}_v)$  of all views  $\mathbf{X}_v \doteq v(\mathbf{X})$  of an input image  $\mathbf{X}$  are consistent with each other. DINO does this by using a “DINO head”<sup>2</sup>  $h_{\mathbf{W}, \mu}$ , parameterized by a matrix  $\mathbf{W} \in \mathbb{R}^{s \times d}$  and a vector  $\mu \in \mathbb{R}^s$ , to extract a probability vector  $\mathbf{p}_{\theta, \mathbf{W}, \mu}(\mathbf{X}_v) \doteq h_{\mathbf{W}, \mu}(\mathbf{z}_\theta(\mathbf{X}_v))$  from the aggregate feature  $\mathbf{z}_\theta(\mathbf{X}_v)$ , using the following simple recipe:

$$h_{\mathbf{W}, \mu}(\mathbf{z}) \doteq \text{softmax}([\mathbf{W}\mathbf{z} - \mu]/\tau), \quad \forall \mathbf{z} \in \mathbb{R}^d, \quad (7.2.5)$$

where the softmax:  $\mathbb{R}^s \rightarrow \Delta_s$  function is defined by

$$\text{softmax}\left(\begin{bmatrix} x_1 \\ \vdots \\ x_s \end{bmatrix}\right) \doteq \frac{1}{\sum_{i=1}^s e^{x_i}} \begin{bmatrix} e^{x_1} \\ \vdots \\ e^{x_s} \end{bmatrix} \quad (7.2.6)$$

and  $\tau > 0$  is a “temperature” parameter which controls the entropy of the softmax’s output.

In particular, DINO minimizes the difference between the probability vector  $\mathbf{p}_{\theta, \mathbf{W}, \mu}(\mathbf{X}_g) \doteq h_{\mathbf{W}, \mu}(\mathbf{z}_\theta(\mathbf{X}_g))$  for each global view  $\mathbf{X}_g \doteq v_g(\mathbf{X})$  and the probability vector  $\mathbf{p}_{\theta, \mathbf{W}}(\mathbf{X}_c) \doteq h_{\mathbf{W}, \mathbf{o}_m}(\mathbf{z}_\theta(\mathbf{X}_c))$  for each view  $\mathbf{X}_c \doteq v_c(\mathbf{X})$ . Here,  $v_c$  can either be a local view or a global view. We will discuss the implementation of  $f_\theta$  and  $f_\theta^{\text{ext}}$  shortly in Section 7.2.3. Overall, DINO solves the problem

$$\min_{\theta, \mathbf{W}, \mu} \mathcal{L}_{\text{DINO}}(\theta, \mathbf{W}, \mu) \quad \text{where} \quad \mathcal{L}_{\text{DINO}}(\theta, \mathbf{W}, \mu) \doteq \mathbb{E}[d_{\text{CE}}(\mathbf{p}_{\theta, \mathbf{W}, \mu}(\mathbf{X}_g), \mathbf{p}_{\theta, \mathbf{W}}(\mathbf{X}_c))], \quad (7.2.7)$$

where the expectation is over data  $\mathbf{X}$ , global views  $v_g$ , and other views  $v_c$ .

In this specific case, however, if you try to implement (7.2.7) and optimize it on a real network, it is very likely that you run into a problem: after running a few iterations of the learning algorithm, the feature mapping  $f_\theta^{\text{ext}} \circ f_\theta$  will become the constant function! This certainly optimizes the above loss since it minimizes the distance between features of different views of the same image. But we obviously do not want to learn this solution.

<sup>2</sup>Note that  $h_{\mathbf{W}, \mu}$  is the task-specific head, which in Section 7.1 is parameterized only by  $\theta$  as opposed to any specific parameters, but since we use two invocations of  $h$  with different values of the second parameter, we keep the specified notation.



Figure 7.5: **An example of an image turned into  $5 \times 5$  square patches, which are placed in raster order.** Each patch is of the same size, and the grid of patches is of shape  $(N_H, N_W) = (5, 5)$ . The grid of patches is then unrolled into a sequence of length  $5 \times 5 = 25$  in raster order.

Actually avoiding collapse is a very common consideration in contrastive learning. So how to do it in this case? The solution of DINO, again, is empirically designed, and carefully tunes the optimization of the parameter  $\mu$  (which is updated using all samples in the batch) and a “temperature” hyperparameter  $\tau$  which is part of the implementation of  $h_{W,\mu}$  and discussed in Section 7.2.3. Given a certain special set of hyperparameters that work well, this is indeed enough to ensure non-collapse of the representation. However, outside of this special configuration, training models to converge is difficult, and the training is highly unstable.

Towards amending this state of affairs, let us discuss simplifications to the formulation. First, instead of computing a probability vector using a learned transformation  $h_{W,\mu}$  of the aggregate features  $\mathbf{z}_\theta$ , we can *directly use the aggregate representation*, ignoring the task-specific head (or equivalently, setting it to the identity mapping). But now we need a way to compare the vectors directly. Using our hypothesis from Chapter 4 that good representations should have Euclidean (subspace) geometry, a much more natural measure of difference is the *squared  $\ell^2$  distance*  $d_{\ell^2} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ , defined as

$$d_{\ell^2}(\mathbf{x}, \mathbf{y}) \doteq \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|_2^2, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d. \quad (7.2.8)$$

This distance-based score is even more efficient to compute than the cross-entropy score. Thus,  $d_{\ell^2}$  takes the place of  $d_{CE}$  in our simplification.

Before, collapse was avoided by using tricks to update  $\mu$  and  $\tau$ . In our simplification, if we compare the features within the representation space instead of converting them to probabilities, we do not have either of these parameters and so must consider a different way to avoid collapse. To do this, we return to the fundamentals. The basic idea of avoiding collapse is that in order to make sure that all samples do not return the same exact same features, we need different samples to have different features. In other words, we would like the *covariance* of the features to be *large* in some sense. But from Chapters 3 and 4, we already have a quantity which measures the size of the covariance matrix. Namely, we use the straightforward (population-level) *Gaussian coding rate*  $R$  to ensure that the features of global views of different images, which have different global information, are well-separated and not collapsed (hence expanded). The overall modified loss  $\mathcal{L}_{\text{SimDINO}}$  becomes:

$$\mathcal{L}_{\text{SimDINO}}(\theta) \doteq \mathbb{E}[d_{\ell^2}(\mathbf{z}_\theta(\mathbf{X}_g), \mathbf{z}_\theta(\mathbf{X}_c))] - \frac{\gamma}{2} \log \det \left( \mathbf{I} + \frac{d}{\varepsilon^2} \text{Cov}(\mathbf{z}_\theta(\mathbf{X}_g)) \right), \quad (7.2.9)$$

where  $\varepsilon > 0$  is fixed and the appropriate expectations are, as before, taken over data  $\mathbf{X}$ , global view  $v_g$ , and other (local or global) view  $v_c$ . The loss in (7.2.9) is the loss used for the simplified DINO (“SimDINO”). As we will see, when properly implemented, it works at least as well as the original DINO.

### 7.2.3 Architecture: Vision Transformer

For the architecture, we use a standard vision transformer. Here is how such an architecture works, formally, in the context of image data. Recall from Section 7.1 that there are four components to an encoder architecture, namely an embedding, a backbone, a feature extractor, and a task-specific head. We discuss these three parts presently.



Figure 7.6: **The transformer embedding pipeline.** Given a sequence of unrolled patches in raster order  $\mathbf{X}^{\text{patch}}$ , each unrolled patch is linearly projected into the feature space, and equipped with an (additive) positional encoding and an additional token known as the class token. The output is the first-layer-input feature  $\mathbf{Z}_{\theta}^1(\mathbf{X}) = f_{\theta}^{\text{emb}}(\mathbf{X})$ .

**Embedding.** Given imagery data  $\mathbf{X} \in \mathcal{I}$ , we embed it as a sequence of tokens in  $\mathbb{R}^d$  using the map  $f_{\theta}^{\text{emb}}$ , as follows. The first two steps are depicted in Figure 7.5, and the latter two are depicted in Figure 7.6.

1. First, we turn the image data  $\mathbf{X}$  into a sequence of patches of shape  $(C, P_H, P_W)$  where  $P_H$  and  $P_W$  are the patch dimensions. We assume that  $P_H$  and  $P_W$  evenly divide the height and width of  $\mathbf{X}$ , respectively (in the notation of Section 7.2.2 we assume that  $P_H$  and  $P_W$  evenly divide  $S_{\text{loc}}$  and  $S_{\text{glo}}$ ). Let the resulting grid of patches have  $N_H$  rows and  $N_W$  columns.
2. We unroll each patch into a vector of length  $D \doteq C P_H P_W$ . There are  $N \doteq N_H N_W$  patch vectors, which we place in “raster order” (top left → top right → bottom left → bottom right) into a matrix  $\mathbf{X}^{\text{patch}} \in \mathbb{R}^{D \times N}$ , where  $\mathbf{X}^{\text{patch}} \doteq f^{\text{patch}}(\mathbf{X})$ . Notice that  $D$  only depends on the patch size and number of channels. Since the latter quantity is normally constant among samples in the same dataset,  $D$  is the same for all images in the dataset, while  $N$  is different for larger and smaller images.
3. We then perform the following operation on  $\mathbf{X}^{\text{patch}} \in \mathbb{R}^{D \times N}$  to project it to  $\mathbb{R}^{d \times n}$  where  $n \doteq N + 1$ :

$$\mathbf{X}^{\text{patch}} \mapsto [z_{\text{cls}}^1, \mathbf{W}^{\text{emb}} \mathbf{X}] + \mathbf{E}^{\text{pos}}. \quad (7.2.10)$$

Here we have three trainable parameters  $\mathbf{W}^{\text{emb}}$ ,  $z_{\text{cls}}^1$ , and  $\mathbf{E}^{\text{pos}}$  whose purpose is as follows:

- $\mathbf{W}^{\text{emb}} \in \mathbb{R}^{d \times D}$  is a matrix which projects each patch vector to a token feature.
- $z_{\text{cls}}^1 \in \mathbb{R}^d$  is a so-called *class token* or *register token*. The class token heuristically holds global information of the whole data and is used for downstream tasks. In the framework of compressive deep networks from Chapter 4, we expect that the class token is projected onto the same subspaces as the salient or semantically relevant tokens during the progression of the forward pass.
- $\mathbf{E}^{\text{pos}} \in \mathbb{R}^{d \times N}$  is a so-called *positional encoding* which distinguishes tokens of different patches from each other. That is, token features should have positional information, so that the overall map  $f^{\text{pre}}$  is not invariant to permutations of the patches, and  $\mathbf{E}^{\text{pos}}$  inserts this positional information.
  - In this DINO case, where the transformer receives differently-sized images, we learn a positional encoding for the largest size received during training, and interpolate to get the positional encodings for smaller-sized inputs.

Thus, in the end we have

$$f_{\theta}^{\text{emb}}(\mathbf{X}) \doteq [z_{\text{cls}}^1, \mathbf{W}^{\text{emb}} f^{\text{patch}}(\mathbf{X}) + \mathbf{E}^{\text{pos}}]. \quad (7.2.11)$$

All parameters  $z_{\text{cls}}^1, \mathbf{W}^{\text{emb}}, \mathbf{E}^{\text{pos}}$  are contained in the parameter set  $\theta$ .

**Backbone.** Given a sequence of embeddings  $\mathbf{Z}_{\theta}^1(\mathbf{X}) \doteq f_{\theta}^{\text{emb}}(\mathbf{X}) \in (\mathbb{R}^d)^*$ , we process it using the backbone map  $f_{\theta}^{\text{bb}}$  as follows and as depicted in Figure 7.7. The function  $f_{\theta}^{\text{bb}}$  is composed of  $L$  layers  $f_{\theta}^{\ell}$ , i.e.,

$$f_{\theta}^{\text{bb}} = f_{\theta}^L \circ \cdots \circ f_{\theta}^1. \quad (7.2.12)$$

The layer  $f_{\theta}^{\ell}$  has the following implementation:

$$\mathbf{Z}_{\theta}^{\ell+1/2}(\mathbf{X}) = \mathbf{Z}_{\theta}^{\ell}(\mathbf{X}) + \text{MHSA}_{\theta}^{\ell}(\text{LN}_{\theta}^{1,\ell}(\mathbf{Z}_{\theta}^{\ell}(\mathbf{X}))) \quad (7.2.13)$$

$$\mathbf{Z}_{\theta}^{\ell+1}(\mathbf{X}) = \mathbf{Z}_{\theta}^{\ell+1/2}(\mathbf{X}) + \text{MLP}_{\theta}^{\ell}(\text{LN}_{\theta}^{2,\ell}(\mathbf{Z}_{\theta}^{\ell+1/2}(\mathbf{X}))) \quad (7.2.14)$$

and  $f_{\theta}^{\ell}$  is defined such that  $f_{\theta}^{\ell}(\mathbf{Z}_{\theta}^{\ell}(\mathbf{X})) \doteq \mathbf{Z}_{\theta}^{\ell+1}(\mathbf{X})$ . Here we have used some operators, such as  $\text{MHSA}_{\theta}^{\ell}$ ,  $\text{MLP}_{\theta}^{\ell}$  and  $\text{LN}_{\theta}^{i,\ell}$  that are defined as follows:

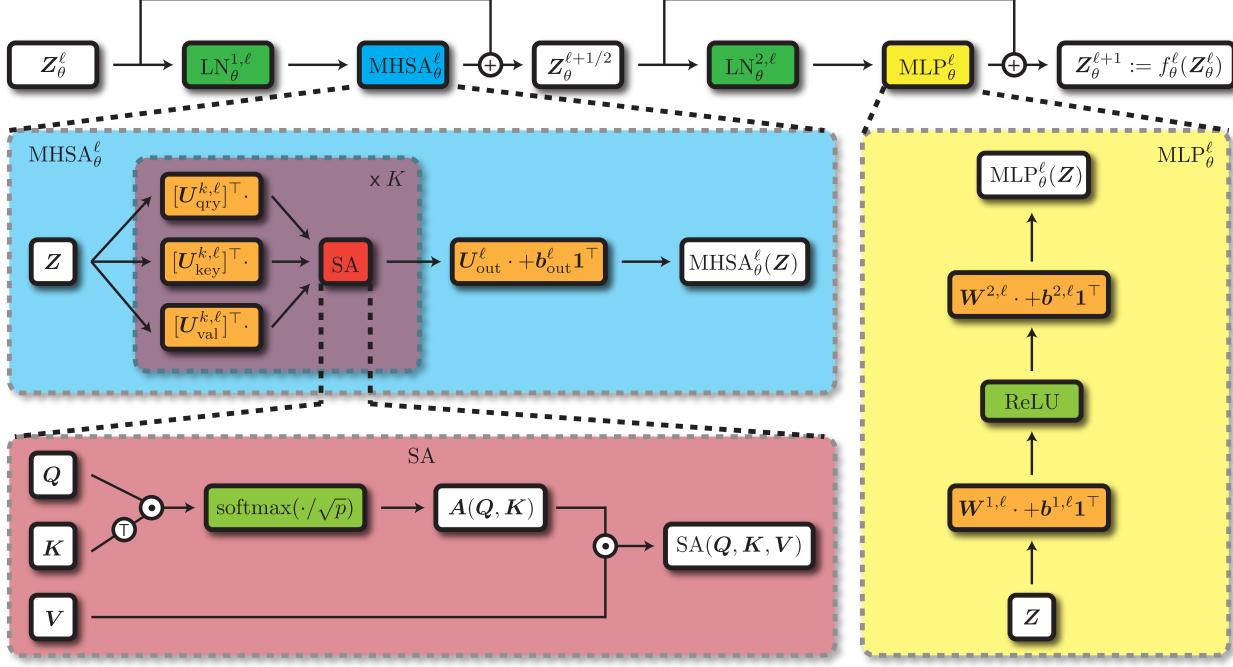


Figure 7.7: **One layer  $f_\theta^\ell$  of the transformer backbone.** The input features go through layer-normalization, multi-head self-attention, and multi-layer perceptron blocks in sequence to form the output features of the layer.

- The  $\text{MHSA}_\theta^\ell$  operator is multi-head-self-attention, the predecessor of the multi-head subspace self-attention (cf Chapter 4). The formulation is as follows:

$$\text{MHSA}_\theta^\ell(\mathbf{Z}) \doteq \mathbf{U}_{\text{out}}^\ell \begin{bmatrix} \text{SA}([\mathbf{U}_{\text{qry}}^{1,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{key}}^{1,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{val}}^{1,\ell}]^\top \mathbf{Z}) \\ \vdots \\ \text{SA}([\mathbf{U}_{\text{qry}}^{K,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{key}}^{K,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{val}}^{K,\ell}]^\top \mathbf{Z}) \end{bmatrix} + \mathbf{b}_{\text{out}}^\ell \mathbf{1}_n^\top, \quad (7.2.15)$$

$$\text{where } \text{SA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \doteq \mathbf{V} \underbrace{\text{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{p}}\right)}_{\doteq \mathbf{A}(\mathbf{Q}, \mathbf{K})} \quad (7.2.16)$$

where  $p$  is a positive integer,  $\mathbf{U}_{\text{qry}}^{k,\ell}, \mathbf{U}_{\text{key}}^{k,\ell}, \mathbf{U}_{\text{val}}^{k,\ell} \in \mathbb{R}^{d \times p}$ ,  $\mathbf{U}_{\text{out}}^\ell \in \mathbb{R}^{d \times Kp}$ , and  $\mathbf{b}_{\text{out}}^\ell \in \mathbb{R}^d$  are trainable parameters contained in the parameter set  $\theta$ , and the softmax is defined column-wise as

$$\text{softmax}(\mathbf{M}) \doteq [\text{softmax}(\mathbf{m}_1) \ \cdots \ \text{softmax}(\mathbf{m}_p)], \quad (7.2.17)$$

$$\forall \mathbf{M} = [\mathbf{m}_1, \dots, \mathbf{m}_p] \in \mathbb{R}^{n \times p}. \quad (7.2.18)$$

In practice, the dimensions are usually picked such that  $Kp = d$ . The terms

$$\mathbf{A}_\theta^{k,\ell}(\mathbf{Z}) \doteq \mathbf{A}([\mathbf{U}_{\text{qry}}^{k,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{key}}^{k,\ell}]^\top \mathbf{Z}), \quad \text{SA}_\theta^{k,\ell}(\mathbf{Z}) \doteq \text{SA}([\mathbf{U}_{\text{qry}}^{k,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{key}}^{k,\ell}]^\top \mathbf{Z}, [\mathbf{U}_{\text{val}}^{k,\ell}]^\top \mathbf{Z}) \quad (7.2.19)$$

are also known as the  $k^{\text{th}}$  *attention map* and  $k^{\text{th}}$  *attention head output* at layer  $\ell$ , respectively. Furthermore, the operation  $\text{SA}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$  can be computed extremely efficiently using specialized software such as FlashAttention [SBZ+25].

- The  $\text{MLP}_\theta^\ell$  is a two-layer perceptron, a regular nonlinearity used in deep networks, and has the form

$$\text{MLP}_\theta^\ell(\mathbf{Z}) \doteq \mathbf{W}_{\text{down}}^\ell \text{ReLU}(\mathbf{W}_{\text{up}}^\ell \mathbf{Z} + \mathbf{b}_{\text{up}}^\ell \mathbf{1}_n^\top) + \mathbf{b}_{\text{down}}^\ell \mathbf{1}_n^\top \quad (7.2.20)$$

where  $\mathbf{W}_{\text{up}}^\ell \in \mathbb{R}^{q \times d}$ ,  $\mathbf{W}_{\text{down}}^\ell \in \mathbb{R}^{d \times q}$ ,  $\mathbf{b}_{\text{up}}^\ell \in \mathbb{R}^q$ ,  $\mathbf{b}_{\text{down}}^\ell \in \mathbb{R}^d$  are trainable parameters also contained in the parameter set  $\theta$ , and  $\text{ReLU}$  is the element-wise  $\text{ReLU}$  nonlinearity, i.e.,  $\text{ReLU}(\mathbf{M})_{ij} = \max\{M_{ij}, 0\}$ .

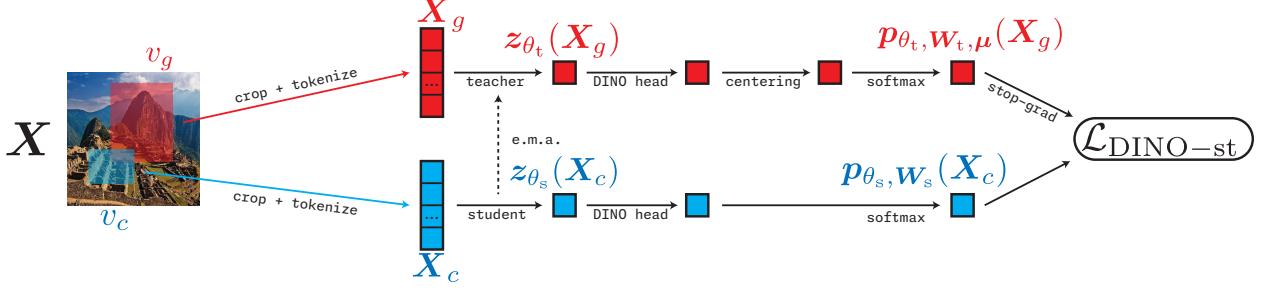


Figure 7.8: **The DINO pipeline.** Student features and teacher features are computed for each input. The objective attempts to align the student features with the teacher features by projecting both sets of features into a high-dimensional probability simplex and computing a cross-entropy loss. Notably, because of the “stop-grad”, the gradient is only computed w.r.t. the *student parameters’ outputs*.

- Each layer-norm  $\text{LN}_{\theta}^{i,\ell}$  for  $i \in \{1, 2\}$  is a standard normalization, which applies column-wise to each token feature independently:

$$\text{LN}_{\theta}^{i,\ell}(\mathbf{Z}) = \text{LN}_{\theta}^{i,\ell}([\mathbf{z}_1, \dots, \mathbf{z}_n]) = [\text{LN}_{\theta}^{i,\ell}(\mathbf{z}_1), \dots, \text{LN}_{\theta}^{i,\ell}(\mathbf{z}_n)] \quad (7.2.21)$$

and has the form

$$\text{LN}_{\theta}^{i,\ell}(\mathbf{z}) = \frac{\mathbf{z} - \text{mean}(\mathbf{z})\mathbf{1}_d}{\|\mathbf{z} - \text{mean}(\mathbf{z})\mathbf{1}_d\|_2} \odot \boldsymbol{\alpha}^{i,\ell} + \boldsymbol{\beta}^{i,\ell} \quad \text{where} \quad \text{mean}(\mathbf{z}) = \frac{1}{d}\mathbf{1}_d^T \mathbf{z} \quad (7.2.22)$$

where  $\odot$  denotes element-wise multiplication, and  $\boldsymbol{\alpha}^{i,\ell}, \boldsymbol{\beta}^{i,\ell} \in \mathbb{R}^d$  are trainable parameters contained in the parameter set  $\theta$ . The layer-norm operator serves as a sort of normalization on each token, where the scale of each token afterwards is learnable and shared amongst all tokens.

The transformer is one of the most popular neural network architectures in history, powering applications in almost all fields of deep learning.

**Feature extractor.** We use a post-processing step  $f_{\theta}^{\text{ext}}$  which extracts the *class token feature*, which (recall) is the feature meant to contain aggregate information about the input image, and applies an MLP and normalization to it. Namely, we have

$$\mathbf{z}_{\theta}(\mathbf{X}) \doteq f_{\theta}^{\text{ext}}(\mathbf{Z}_{\theta}(\mathbf{X})) = f_{\theta}^{\text{ext}}([\mathbf{z}_{\theta}^1(\mathbf{X}), \dots, \mathbf{z}_{\theta}^n(\mathbf{X})]) \doteq \frac{\text{MLP}_{\theta}^{\text{ext}}(\mathbf{z}_{\theta}^1(\mathbf{X}))}{\|\text{MLP}_{\theta}^{\text{ext}}(\mathbf{z}_{\theta}^1(\mathbf{X}))\|_2}. \quad (7.2.23)$$

**Task-specific (“DINO”) head.** For DINO, we use the task-specific DINO head  $h_{\mathbf{W}, \mu}$ . For SimDINO, we use *no* task-specific head *at all*, as previously described.

## 7.2.4 Optimization Strategy

**Optimizing DINO.** We have a loss function and an architecture, so we now discuss the optimization strategy. The optimization strategy for DINO uses *two sets of weights for the same architecture*: *student* weights  $\theta_s$  and *teacher* weights  $\theta_t$ . These correspond to two different neural networks, called the teacher network and student network, with the same architecture. The teacher network encodes all global views, while the student network encodes all “other” views. The goal of the loss is to distill teacher outputs into the student model. Namely, we train on the loss  $\mathcal{L}_{\text{DINO-st}}$ :

$$\mathcal{L}_{\text{DINO-st}}(\theta_s, \theta_t, \mathbf{W}_s, \mathbf{W}_t, \mu) \doteq \mathbb{E}[d_{\text{CE}}(\mathbf{p}_{\theta_t, \mathbf{W}_t, \mu}(\mathbf{X}_g), \mathbf{p}_{\theta_s, \mathbf{W}_s}(\mathbf{X}_c))]. \quad (7.2.24)$$

Now, we can fully describe the overall pipeline of DINO, depicted in Figure 7.8.

While it is easy to reason about (7.2.24), it is impossible in practice to implement optimization algorithms such as gradient descent with a loss given by  $\mathcal{L}_{\text{DINO-st}}$ . This is because the expectations in the loss are impossible to evaluate, much less take the gradient of. In this extremely frequent case, we approximate the expectation via finite samples. That is, at each timestep  $k$  we:

- Subsample  $B$  data points from our dataset  $\{\mathbf{X}_1^{(k)}, \dots, \mathbf{X}_B^{(k)}\} \subset \mathcal{I}$ .
- For each data point  $\mathbf{X}_b^{(k)}$ , sample  $M_{\text{glo}}$  global views  $v_{b,g}^{(k),i}$  and  $M_{\text{loc}}$  local views  $v_{b,\ell}^{(k),i}$ . Apply the views to  $\mathbf{X}_b^{(k)}$  to obtain  $\mathbf{X}_{b,g}^{(k),i} \doteq v_{b,g}^{(k),i}(\mathbf{X}_b^{(k)})$  and  $\mathbf{X}_{b,\ell}^{(k),i} \doteq v_{b,\ell}^{(k),i}(\mathbf{X}_b^{(k)})$ .
- For each *local* view  $\mathbf{X}_{b,\ell}^{(k),i}$  compute the following quantities:

$$\mathbf{z}_{\theta_s}(\mathbf{X}_{b,\ell}^{(k),i}) \doteq (f_{\theta_s}^{\text{ext}} \circ f_{\theta_s})(\mathbf{X}_{b,\ell}^{(k),i}), \quad \mathbf{p}_{\theta_s, \mathbf{W}_s}(\mathbf{X}_{b,\ell}^{(k),i}) \doteq h_{\mathbf{W}_s, \mathbf{o}_m}(\mathbf{z}_{\theta_s}(\mathbf{X}_{b,\ell}^{(k),i}(\theta))) \quad (7.2.25)$$

and for each *global* view  $\mathbf{X}_{b,g}^{(k),i}$  compute the following quantities (by an abuse of notation):

$$\mathbf{z}_{\theta_s}(\mathbf{X}_{b,g}^{(k),i}) \doteq (f_{\theta_s}^{\text{ext}} \circ f_{\theta_s})(\mathbf{X}_{b,g}^{(k),i}), \quad \mathbf{p}_{\theta_s, \mathbf{W}_s}(\mathbf{X}_{b,g}^{(k),i}) \doteq h_{\mathbf{W}_s, \mathbf{o}_m}(\mathbf{z}_{\theta_s}(\mathbf{X}_{b,g}^{(k),i})), \quad (7.2.26)$$

$$\mathbf{z}_{\theta_t}(\mathbf{X}_{b,g}^{(k),i}) \doteq (f_{\theta_t}^{\text{ext}} \circ f_{\theta_t})(\mathbf{X}_{b,g}^{(k),i}), \quad \mathbf{p}_{\theta_t, \mathbf{W}_t, \boldsymbol{\mu}}(\mathbf{X}_{b,g}^{(k),i}) \doteq h_{\mathbf{W}_t, \boldsymbol{\mu}}(\mathbf{Z}_{\theta_t}(\mathbf{X}_{b,g}^{(k),i})). \quad (7.2.27)$$

- Compute the *surrogate, approximate loss*  $\hat{\mathcal{L}}_{\text{DINO-st}}^{(k)}$ , defined as follows:

$$\begin{aligned} \hat{\mathcal{L}}_{\text{DINO-st}}^{(k)}(\theta_s, \theta_t, \mathbf{W}_s, \mathbf{W}_t, \boldsymbol{\mu}) &\doteq \frac{1}{BM_{\text{glo}}(M_{\text{glo}} + M_{\text{loc}} - 1)} \sum_{b=1}^B \sum_{i=1}^{M_{\text{glo}}} \\ &\left[ \sum_{j=1}^{M_{\text{loc}}} d_{\text{CE}}(\mathbf{p}_{\theta_t, \mathbf{W}_t, \boldsymbol{\mu}}(\mathbf{X}_{b,g}^{(k),i}), \mathbf{p}_{\theta_s, \mathbf{W}_s}(\mathbf{X}_{b,\ell}^{(k),j})) + \sum_{\substack{j=1 \\ j \neq i}}^{M_{\text{glo}}} d_{\text{CE}}(\mathbf{p}_{\theta_t, \mathbf{W}_t, \boldsymbol{\mu}}(\mathbf{X}_{b,g}^{(k),i}), \mathbf{p}_{\theta_s, \mathbf{W}_s}(\mathbf{X}_{b,g}^{(k),j})) \right] \end{aligned} \quad (7.2.28)$$

as well as its gradients with respect to  $\theta_s$  and  $\mathbf{W}_s$ , which should be computed under the assumption that  $\theta_t$ ,  $\mathbf{W}_t$ , and  $\boldsymbol{\mu}$  are constants — namely that they are *detached from the computational graph* and not dependent on  $\theta_s$  and  $\mathbf{W}_s$ .

- Update the student parameters  $\theta_s$  and  $\mathbf{W}_s$  via an iterative gradient-based optimization algorithm, and update  $\theta_t$ ,  $\mathbf{W}_t$ , and  $\boldsymbol{\mu}$  via exponential moving averages with decay parameters  $\nu^{(k)}$ ,  $\nu^{(k)}$ , and  $\rho^{(k)}$  respectively, i.e.,

$$(\theta_s^{(k+1)}, \mathbf{W}_s^{(k+1)}) = \text{OPTUPDATE}^{(k)}(\theta_s^{(k)}, \mathbf{W}_s^{(k)}; \nabla_{(\theta_s, \mathbf{W}_s)} \hat{\mathcal{L}}_{\text{DINO-st}}^{(k)}) \quad (7.2.29)$$

$$\theta_t^{(k+1)} = \nu^{(k)} \theta_t^{(k)} + (1 - \nu^{(k)}) \theta_s^{(k+1)} \quad (7.2.30)$$

$$\mathbf{W}_t^{(k+1)} = \nu^{(k)} \mathbf{W}_t^{(k)} + (1 - \nu^{(k)}) \mathbf{W}_s^{(k+1)} \quad (7.2.31)$$

$$\boldsymbol{\mu}^{(k+1)} = \rho^{(k)} \boldsymbol{\mu}^{(k)} + (1 - \rho^{(k)}) \cdot \frac{1}{BM_{\text{glo}}} \sum_{b=1}^B \sum_{i=1}^{M_{\text{glo}}} \mathbf{W}^{(k)} \mathbf{z}_{\theta_t}(\mathbf{X}_{b,g}^{(k),i}), \quad (7.2.32)$$

For example, if the chosen optimization algorithm were stochastic gradient descent, we would have the update  $\theta_s^{(k+1)} \doteq \theta_s^{(k)} - \delta^{(k)} \nabla_{\theta_s} \hat{\mathcal{L}}_{\text{DINO-st}}^{(k)}$ , and so on.

Notice that the optimization procedure is rather irregular: although all four parameters change at each iteration, only two of them are directly updated from a gradient-based method. The other two are updated from exponential moving averages, and indeed treated as constants when computing any gradients. After training, we discard the student weights and use the teacher weights for our trained network  $f$ , as this exponentially moving average has been empirically shown to stabilize the resulting model (this idea is known as Polyak averaging or iterate averaging).

The way that  $\nu$  and  $\rho$  change over the optimization trajectory (i.e., the functions  $k \mapsto \nu^{(k)}$  and  $k \mapsto \rho^{(k)}$ ) are hyperparameters or design decisions, with  $\nu^{(1)} < 1$  and  $\lim_{k \rightarrow \infty} \nu^{(k)} = 1$  usually, and similar for  $\rho$ . The temperature hyperparameter  $\tau$ , used in the DINO head  $h_{\mathbf{W}, \boldsymbol{\mu}}$ , also changes over the optimization trajectory (though this dependence is not explicitly notated).

Using the surrogate (“empirical”) loss transforms our intractable optimization problem, as in optimizing the loss in (7.2.24), into a tractable stochastic optimization problem which is run to train essentially every deep learning model in the world. This conversion is extremely natural once you have seen some examples, and we will hopefully give these examples throughout the chapter.

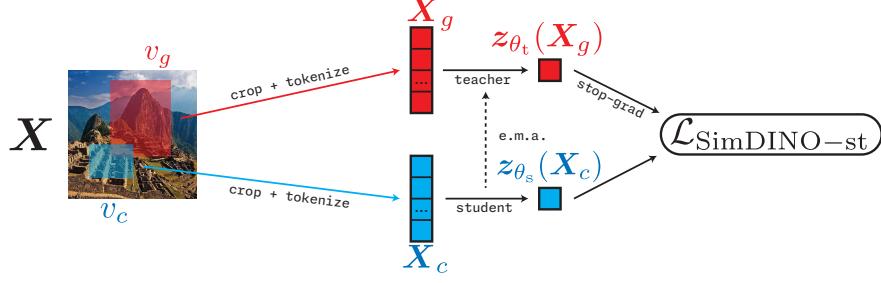


Figure 7.9: **The SimDINO pipeline.** Here, in contrast to the DINO pipeline in Figure 7.8, the loss is computed directly on the features without need of further manipulation. This shaves off several large matrices’ worth of parameters and simplifies the pipeline, simultaneously making it more stable to train.

**Optimizing SimDINO.** The simplified DINO population-level objective is very similar in spirit but much simpler in execution, i.e.,

$$\mathcal{L}_{\text{SimDINO-st}}(\theta_s, \theta_t) \doteq \mathbb{E}[d_{\ell^2}(\mathbf{z}_{\theta_t}(\mathbf{X}_g), \mathbf{z}_{\theta_s}(\mathbf{X}_c))] - \frac{\gamma}{2} \log \det \left( \mathbf{I} + \frac{d}{\varepsilon^2} \text{Cov}(\mathbf{z}_{\theta_s}(\mathbf{X}_g)) \right). \quad (7.2.33)$$

Thus, as elaborated in Figure 7.9, the SimDINO pipeline is strictly simpler than the DINO pipeline. We can use a simpler version of the DINO training pipeline to optimize SimDINO. At each timestep  $k$ , we:

- Subsample  $B$  data points from our dataset  $\{\mathbf{X}_1^{(k)}, \dots, \mathbf{X}_B^{(k)}\} \subset \mathcal{I}$ .
- For each data point  $\mathbf{X}_b^{(k)}$ , sample  $M_{\text{glo}}$  global views  $v_{b,g}^{(k),i}$  and  $M_{\text{loc}}$  local views  $v_{b,\ell}^{(k),i}$ . Apply the views to  $\mathbf{X}_b^{(k)}$  to obtain  $\mathbf{X}_{b,g}^{(k),i} \doteq v_{b,g}^{(k),i}(\mathbf{X}_b^{(k)})$  and  $\mathbf{X}_{b,\ell}^{(k),i} \doteq v_{b,\ell}^{(k),i}(\mathbf{X}_b^{(k)})$ .
- For each *local* view  $\mathbf{X}_{b,\ell}^{(k),i}$  compute  $\mathbf{z}_{\theta_s}(\mathbf{X}_{b,\ell}^{(k),i}) \doteq (f_{\theta_s}^{\text{ext}} \circ f_{\theta_s})(\mathbf{X}_{b,\ell}^{(k),i})$ . For each *global* view  $\mathbf{X}_{b,g}^{(k),i}$  compute  $\mathbf{z}_{\theta_s}(\mathbf{X}_{b,g}^{(k),i}) \doteq (f_{\theta_s}^{\text{ext}} \circ f_{\theta_s})(\mathbf{X}_{b,g}^{(k),i})$  and  $\mathbf{z}_{\theta_t}(\mathbf{X}_{b,g}^{(k),i}) \doteq (f_{\theta_t}^{\text{ext}} \circ f_{\theta_t})(\mathbf{X}_{b,g}^{(k),i})$ .
- Compute the *surrogate, approximate loss*  $\hat{\mathcal{L}}_{\text{SimDINO-st}}^{(k)}$ , defined as follows:

$$\begin{aligned} \hat{\mathcal{L}}_{\text{SimDINO-st}}^{(k)}(\theta_s, \theta_t) &\doteq \frac{1}{BM_{\text{glo}}(M_{\text{glo}} + M_{\text{loc}} - 1)} \sum_{b=1}^B \sum_{i=1}^{M_{\text{glo}}} \left[ \sum_{j=1}^{M_{\text{loc}}} d_{\ell^2}(\mathbf{z}_{\theta_t}(\mathbf{X}_{b,g}^{(k),i}), \mathbf{z}_{\theta_s}(\mathbf{X}_{b,\ell}^{(k),j})) \right. \\ &\quad \left. + \sum_{j=1}^{M_{\text{glo}}} d_{\ell^2}(\mathbf{z}_{\theta_t}(\mathbf{X}_{b,g}^{(k),i}), \mathbf{z}_{\theta_s}(\mathbf{X}_{b,g}^{(k),j})) \right] - \frac{\gamma}{M_{\text{glo}}} \sum_{i=1}^{M_{\text{glo}}} R_{\varepsilon}([\mathbf{z}_{\theta_s}(\mathbf{X}_{1,g}^{(k),i}), \dots, \mathbf{z}_{\theta_s}(\mathbf{X}_{B,g}^{(k),i})]) \end{aligned} \quad (7.2.34)$$

where  $R_{\varepsilon}$  is the Gaussian coding rate estimated on finite samples, described in Chapter 4. The gradient of  $\hat{\mathcal{L}}_{\text{SimDINO-st}}^{(k)}$  with respect to  $\theta_s$  should (again) be computed, under the assumption that  $\theta_t$  is constant.

- Update the student parameters  $\theta_s$  via an iterative gradient-based optimization algorithm, and update  $\theta_t$  via an exponential moving average with decay parameter  $\nu^{(k)}$ , i.e.,

$$\theta_s^{(k+1)} = \text{OPTUPDATE}^{(k)}(\theta_s^{(k)}; \nabla_{\theta_s} \hat{\mathcal{L}}_{\text{SimDINO-st}}^{(k)}) \quad (7.2.35)$$

$$\theta_t^{(k+1)} = \nu^{(k)} \theta_t^{(k)} + (1 - \nu^{(k)}) \theta_s^{(k+1)}. \quad (7.2.36)$$

Again, we re-iterate that the gradient is only taken w.r.t.  $\theta_s$ , treating  $\theta_t$  as a constant. Here, note that while the choice of  $\nu$  is still a design decision, the hyperparameters  $\rho$  and  $\tau$  is removed.

### 7.2.5 Evaluation Methodology

There are several ways to evaluate a trained transformer model. We highlight two in this section. Let us define the *center crop* view  $v_{cc}: \mathcal{I} \rightarrow \mathcal{I}$  which is a *a deterministic resized crop*:

- it resizes the image so that the shortest edge is of size  $S_{rsz}$  (similar to random resized crops with area percentage parameter 1);
- then it takes the *center*  $S_{cc} \times S_{cc}$  crop;

so that the final shape is  $(C, S_{cc}, S_{cc})$ . Notice that the view  $v_{cc}$  is completely deterministic given an input. For an input  $\mathbf{X}$  we write  $\mathbf{X}_{cc} \doteq v_{cc}(\mathbf{X})$ . Here  $S_{cc} \leq S_{rsz}$ .

**Linear probing.** The first, and most architecture-agnostic, way to evaluate an encoder model  $\mathbf{X} \mapsto \mathbf{z}_\theta(\mathbf{X})$  is to employ *linear probing*. Linear probing is, in a sentence, running logistic regression on the aggregate features computed by the encoder. This tells us how much semantic information exists in the representations, as well as how easily extractable this information is. (That is: to what extent do the features of images with different semantics live on different subspaces of the feature space?)

More formally, let us suppose that we want to evaluate the quality and faithfulness of the features of the encoder on image-label data  $(\mathbf{X}, \mathbf{y})$ , where there are  $N_{cls}$  classes and  $\mathbf{y} \in \{0, 1\}^{N_{cls}}$  is a “one-hot encoding” (namely, zeros in all positions except a 1 in the  $i^{\text{th}}$  position if  $\mathbf{X}$  is in the  $i^{\text{th}}$  class). One way to do this is to solve the logistic regression problem

$$\min_{\mathbf{W} \in \mathbb{R}^{N_{cls} \times d}} \mathbb{E}[\text{CE}(\mathbf{y}, \mathbf{W}\mathbf{z}_\theta(\mathbf{X}_{cc}))]. \quad (7.2.37)$$

More practically, if we have labeled data  $\{(\mathbf{X}_b, \mathbf{y}_b)\}_{b=1}^B$ , we can solve the *empirical* logistic regression problem (akin to (7.2.24) vs. (7.2.28)) given by

$$\min_{\mathbf{W} \in \mathbb{R}^{N_{cls} \times d}} \frac{1}{B} \sum_{b=1}^B \text{CE}(\mathbf{y}_b, \mathbf{W}\mathbf{z}_\theta(\mathbf{X}_{b,cc})). \quad (7.2.38)$$

This problem is a convex optimization problem in  $\mathbf{W}$ , and thus can be solved efficiently via (stochastic) gradient descent or a litany of other algorithms. This linear probe, together with the encoder, may be used as a classifier, and we can evaluate the classification accuracy. The usual practice is to train the model first on a large dataset (such as ImageNet-1K), then train the linear probe on a dataset (such as the training dataset of CIFAR-10), and evaluate it on a third (“holdout”) dataset which is drawn from the same distribution as the second one (such as the evaluation dataset of CIFAR-10).

**$k$ -nearest neighbors.** We can also evaluate the performance of the features on classification tasks *without needing to explicitly train a classifier* by using the  $k$ -nearest neighbor algorithm to get an average predicted label. Namely, given a dataset  $\{\mathbf{z}_b\}_{b=1}^B \subseteq \mathbb{R}^d$ , define the  $k$ -nearest neighbors of another point  $\mathbf{z} \in \mathbb{R}^d$  as  $\text{NN}_k(\mathbf{z}, \{\mathbf{z}_b\}_{b=1}^B)$ . Using this notation, we can compute the predicted label  $\hat{\mathbf{y}}_\theta(\mathbf{X} \mid \{(\mathbf{X}_b, \mathbf{y}_b)\}_{b=1}^B)$  as

$$\hat{\mathbf{y}}_\theta(\mathbf{X} \mid \{(\mathbf{X}_b, \mathbf{y}_b)\}_{b=1}^B) = \mathbf{1}(i^*) \quad \text{where} \quad i^* \doteq \arg \max_{i \in [Q]} \sum_{b=1}^B \mathbf{y}_b \mathbf{1}[\mathbf{z}_\theta(\mathbf{X}_{cc,b}) \in \text{NN}_k(\mathbf{z}_\theta(\mathbf{X}_{cc}))]. \quad (7.2.39)$$

Here,  $\mathbf{1}(i) \in \Delta_{N_{cls}}$  is (by an abuse of notation, c.f. indicator variables) the one-hot probability vector supported at  $i$ , i.e., 1 in the  $i^{\text{th}}$  coordinate and 0 elsewhere. That is, this procedure takes the most common label among the  $k$  nearest points in feature space. The  $k$ -nearest neighbor classification accuracy is just the accuracy of this predicted label, namely,

$$\mathbb{E}_{\mathbf{X}, \mathbf{y}} [\mathbf{1}(\hat{\mathbf{y}}_\theta(\mathbf{X} \mid \{(\mathbf{X}_b, \mathbf{y}_b)\}_{b=1}^B) = \mathbf{y})] \quad (7.2.40)$$

or more commonly its corresponding empirical version, where  $(\mathbf{X}, \mathbf{y})$  ranges over a finite dataset (*not* the existing samples  $(\mathbf{X}_b, \mathbf{y}_b)$  which are used for the  $k$  neighbors).

**Fidelity of the attention maps.** Another way to check the performance of the representations, for a transformer-based encoder, is to examine the fidelity of the attention maps  $\mathbf{A}^{L,k} \in \mathbb{R}^{n \times n}$  as defined in Equation (7.2.19), at the last layer  $L$ , and given by the following pipeline:

$$\mathbf{X} \mapsto \dots \mapsto \mathbf{Z}^{L-1} = [\underbrace{\mathbf{z}_1^{L-1}}_{\text{class token}}, \underbrace{\mathbf{z}_2^{L-1}, \dots, \mathbf{z}_n^{L-1}}_{\text{patch tokens}}] \mapsto \mathbf{A}^{k,L} = \begin{bmatrix} \mathbf{A}_{1,1}^{k,L} & \mathbf{A}_{1,2}^{k,L} \\ \mathbf{A}_{2,1}^{k,L} & \mathbf{A}_{2,2}^{k,L} \end{bmatrix}. \quad (7.2.41)$$

In particular, we examine what the attention maps for a given input reveal about the salient objects in the input image, i.e., which parts of the image provide the most globally-relevant information to the class token. One particular way to do this is to examine the component of the attention map where the class token is extracted as the query and removed from the value matrix, i.e.,  $\mathbf{A}_{2,:1}^{k,L} \in \mathbb{R}^{1 \times (n-1)} = \mathbb{R}^{1 \times N}$  or its transpose  $\mathbf{a}^{k,L} = (\mathbf{A}_{2,:1}^{k,L})^\top \in \mathbb{R}^N$ . Notice that this vector  $\mathbf{a}^{k,L}$ , which we label as the “*saliency vector* at the  $k^{\text{th}}$  attention head at layer  $L$ ,” has a value for every patch,  $1, \dots, N$ , and we use this value to describe how relevant each patch is towards the global information. In particular for visualization’s sake we create a new image where each patch is replaced by its corresponding value in the saliency vector, showcasing the contribution of each patch; we call this image the “*saliency map* at the  $k^{\text{th}}$  attention head at layer  $L$ ”. To visualize the total relevance of each patch towards the global information across all heads, we can average the saliency vector, i.e.,  $\tilde{\mathbf{a}}^L \doteq \frac{1}{K} \sum_{k=1}^K \mathbf{a}^{k,L}$  and expand into the *average saliency map*. The average saliency maps should highlight the relevant parts of the input image.

**Object detection and segmentation.** We can evaluate how the representations capture the fine-grained (i.e., smaller or more detailed) properties of the input by using them for *semantic segmentation*. Roughly, this means that we use the features to construct bounding boxes for all objects in the input. There are several ways to do this, and several ways to score the resulting bounding boxes compared to a ground-truth. Each combination of methods corresponds to a particular segmentation metric. We do not formally describe them here as they are not particularly insightful, but the DINO paper [CTM+21] and DINOv2 paper [ODM+23] contain references to all metrics that are used in practice.

### 7.2.6 Experimental Setup and Results

Since SimDINO is directly built upon DINO, we compare the optimal settings for DINO as given by their original paper [CTM+21] with the same settings applied to SimDINO for fair comparison.

**Objective function.** We use 10 local views (i.e.,  $M_{\text{loc}} = 10$ ) of resolution  $96 \times 96$  (i.e.,  $S_{\text{loc}} = 96$ ) and 2 global views (i.e.,  $M_{\text{glo}} = 2$ ) of resolution  $224 \times 224$  (i.e.,  $S_{\text{glo}} = 224$ ) for all experiments. The corresponding portions of the original images cropped for local and global views are  $p_{\text{loc}} \in [\frac{1}{20}, \frac{3}{10}]$  and  $p_{\text{glo}} \in [\frac{3}{10}, 1]$  (chosen randomly per-view). The smaller edge size within the resized crops is  $S_{\text{rsz}} = 256$ , and the center crop (evaluation) view edge size is  $S_{\text{cc}} = 224$ . All of these settings apply to both DINO and SimDINO.

**Model architecture.** For all inputs we set the patch size to be  $16 \times 16$  (namely,  $P_H = P_W = 16$ ). We use the small, base, and large models of the ViT [DBK+21] architecture as the embedding and backbone. The feature extractor is a three-layer MLP with a hidden size of 2048 and an output dimension of 256, followed by an  $\ell^2$ -normalization, as specified in Section 7.2.3. For DINO architectures (i.e., not SimDINO architectures), the DINO head  $\mathbf{W}$  is a matrix in  $\mathbb{R}^{65536 \times 256}$ , and the parameter  $\mu$  is a vector in  $\mathbb{R}^{65536}$ .

**Datasets and optimization.** For pre-training, both our DINO reproduction and SimDINO use the ImageNet-1K dataset across all methods. We use AdamW [LH17] as the optimizer, which is a very standard choice. We follow the following hyperparameter recommendations:

- The batch size is  $B = 1024$ .
- The learning rate (for AdamW and the student model) has “base” value  $2 \times 10^{-3}$ . In the first 10 epochs the learning rate linearly increases from 0 to the base value (i.e., at the  $i^{\text{th}}$  epoch the learning

Method	Model	Epochs	20-NN	Linear Probing
DINO	ViT-B	100	72.9	76.3
SimDINO	ViT-B	100	<b>74.9</b>	<b>77.3</b>
DINO	ViT-L	100	—	—
SimDINO	ViT-L	100	<b>75.6</b>	<b>77.4</b>
SwAV	ViT-S	800	66.3	73.5
MoCov3	ViT-B	300	—	76.7

Table 7.1: **Classification performance** on hold-out test data for DINO and SimDINO, using both  $k$ -nearest neighbor accuracy ( $k = 20$ ) and linear probing. At the same number of iterations (100), SimDINO is clearly better in terms of performance, and is more stable (the DINO training running on ViT-L backbone with the provided settings has very unstable optimization and obtains NaN loss in short order). We also compare to other standout methods, namely SwAV and MoCov3, which DINO was built on.

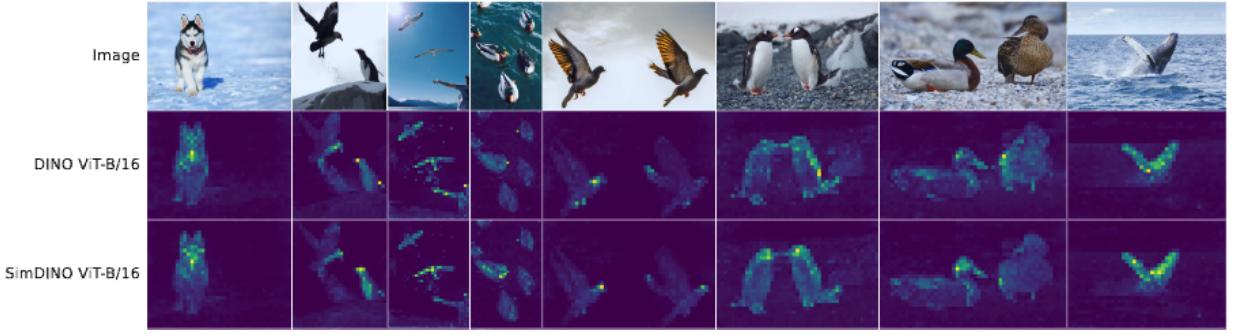


Figure 7.10: **A qualitative comparison of saliency maps** generated by DINO (middle row) and by SimDINO (bottom row). For each image, we compute and display the average saliency map in the last layer  $L$ . The saliency maps are similar across models, meaning that all models converge to a similar notion of what objects are important. Note that although  $X_{\text{eval}}$  is a square image, it is interpolated back into rectangular shape to make this visualization.

rate is  $(i/10) \cdot 2 \times 10^{-3}$ , for  $1 \leq i \leq 10$ ). Then over the next 90 epochs the learning rate decays via a so-called *cosine schedule* back down to 0. The definition of a cosine schedule is given in many places, including [PyTorch documentation](#), and it is commonly used when training deep vision models.

- The weight decay (the  $W$  in AdamW) follows a cosine schedule from 0.04 to 0.4 over training.
- The EMA rate  $\nu$  follows a cosine schedule from 0.996 to 1.0 over training. Specifically for DINO, the centering EMA rate  $\rho$  is fixed at 0.9.
- Specifically for DINO, the teacher temperature  $\tau_t$  is fixed at 0.1, while the student temperature  $\tau_s$  linearly increases from 0.04 to 0.07 during the first 30 epochs and is fixed at 0.07 thereafter.

We use some (essentially information-preserving) data augmentations, such as flips, color jittering, Gaussian blur, and solarization, for each seen image during training, before taking the local and global views. The exact hyperparameters governing these are not listed here, but are referenced in the DINO paper [CTM+21].

For linear probing, the linear probe is usually trained using the AdamW optimizer with learning rate  $2 \times 10^{-4}$ , weight decay 0.01, and batch size 512, but these are often modified on a case-by-case basis to minimize the loss.

**Evaluation results.** In terms of downstream classification performance, we obtain the performance in Table 7.1. We observe that the performance of SimDINO is much higher than in DINO under fair comparison. Also, it is much more stable: the prescribed settings of DINO cannot train a ViT-L(arge) model. On the other hand, Figure 7.10 shows visualizations of the average saliency maps in DINO and our simplified DINO, observing that the saliency maps look quite similar across models, indicating that the models learn

Method	Model	Detection $\uparrow$			Segmentation $\uparrow$		
		AP <sub>50</sub>	AP <sub>75</sub>	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP
SimDINO	ViT-L/16	<b>5.4</b>	1.9	2.4	4.5	1.4	1.9
SimDINO	ViT-B/16	5.2	<b>2.0</b>	<b>2.5</b>	<b>4.7</b>	<b>1.5</b>	<b>2.0</b>
DINO	ViT-B/16	3.9	1.5	1.8	3.1	1.0	1.4
DINO	ViT-B/8	5.1	2.3	2.5	4.1	1.3	1.8

Table 7.2: **Segmentation performance** of pre-trained DINO and SimDINO models on COCO val2017 [LMB+14], a segmentation dataset which contains object location metadata. We do not train on COCO, merely using the pre-trained embedding and backbone, and the bounding boxes are extracted from the features via a method called MaskCut [WGY+23]. Nevertheless, SimDINO surpasses DINO at object detection and segmentation at fair comparison, and even surpasses DINO with smaller patch size (side length 8 instead of 16). Smaller patch sizes are known to help performance, especially with detection and segmentation tasks, so this result is quite surprising and encouraging.

features which are at least as good at capturing fine-grained details. The segmentation and object detection performances in Table 7.2 confirm this claim quantitatively, where SimDINO features show substantive improvement over those of DINO.

## 7.3 Image Classification

In the previous section, we simplified an overly complex learning objective using our intuition of representation learning through the lens of compression. However, many of the most popular learning procedures are incredibly simple. In these cases, it is difficult to further simplify the objective. Thus, in this and future sections, we will focus on principled ways to modify the *deep network architectures* for a variety of tasks.

Let us first start with arguably the most classical task in machine learning: *image classification*, which is often used as a standard task to evaluate pattern recognition algorithms or deep network architectures. From our discussion of white-box architectures in Chapter 4, we only need a semantically meaningful task to learn good representations with white-box architectures. We will validate this idea in this section.

First, the dataset stays largely the same as Section 7.2.1. Both the training and test data consist of labeled images, i.e., image-label pairs  $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{C \times H \times W} \times \{0, 1\}^{N_{cls}}$ . We still apply various data augmentations (e.g., flips, Gaussian blurring, solarization, etc.) to each sample in each new batch.

### 7.3.1 Task and Objective

Unlike before, our task is not just to learn a good representation of the data, but also simultaneously build a classifier. Formally, we have labeled data pairs  $(\mathbf{X}, \mathbf{y})$ , where  $\mathbf{y} \in \{0, 1\}^{N_{cls}}$  is a one-hot vector denoting the class membership of  $\mathbf{X}$ . We consider a deterministic *center crop view*  $v_{cc}$  of the input data  $\mathbf{X}$  (cf Section 7.2.2) of the input data  $\mathbf{X}$ . We want to jointly train a feature mapping ( $f_\theta, f_\theta^{\text{ext}}$ ) and a *classification head*  $h_\theta$ , defined as follows:

$$h_\theta(\mathbf{z}) \doteq \text{softmax}(\mathbf{W}^{\text{head}} \mathbf{z} + \mathbf{b}^{\text{head}}), \quad \forall \mathbf{z} \in \mathbb{R}^d \quad (7.3.1)$$

where  $(\mathbf{W}^{\text{head}}, \mathbf{b}^{\text{head}}) \in \mathbb{R}^{N_{cls} \times d} \times \mathbb{R}^{N_{cls}}$  are trainable parameters in the parameter set  $\theta$ , such that the map  $\mathbf{X}_{cc} \mapsto \mathbf{p}_\theta(\mathbf{X}_{cc}) \doteq h_\theta(\mathbf{z}_\theta(\mathbf{X}_{cc}))$  predicts a smoothed label for the view  $\mathbf{X}_{cc} = v_{cc}(\mathbf{X})$  of the input  $\mathbf{X}$ . The learning problem attempts to minimize the distance between  $\mathbf{p}_\theta$  and  $\mathbf{y}$  measured through cross-entropy:

$$\min_{\theta} \{\mathcal{L}_{\text{CE}}(\theta) \doteq \mathbb{E}[\text{CE}(\mathbf{y}, \mathbf{p}_\theta(\mathbf{X}_{cc}))]\}. \quad (7.3.2)$$

### 7.3.2 The CRATE Architecture

The architecture that we use is the CRATE architecture, described in some level of detail in Chapter 4. The overall setup is similar to that of the regular transformer in Section 7.2.3, with a few changes. While the embedding step is the same as both DINO and SimDINO in Section 7.2.3, the feature extraction step is the

same as SimDINO in Section 7.2.3 as it just extracts the feature corresponding to the class token, and the classification head is described in Section 7.3.1, the backbone architecture is different. Each layer takes the form

$$\mathbf{Z}_\theta^{\ell+1/2}(\mathbf{X}) = \mathbf{Z}_\theta^\ell(\mathbf{X}) + \text{MSSA}_\theta^\ell(\text{LN}_\theta^{1,\ell}(\mathbf{Z}_\theta^\ell(\mathbf{X}))), \quad (7.3.3)$$

$$\mathbf{Z}_\theta^{\ell+1}(\mathbf{X}) = \text{ISTA}_\theta^\ell(\text{LN}_\theta^{2,\ell}(\mathbf{Z}_\theta^{\ell+1/2}(\mathbf{X}))), \quad (7.3.4)$$

where the  $\text{MSSA}_\theta^\ell$  and  $\text{ISTA}_\theta^\ell$  blocks are as described in Chapter 4, namely:

- The MSSA operator is multi-head-subspace-self-attention, defined as follows:

$$\text{MSSA}_\theta^\ell(\mathbf{Z}) \doteq \mathbf{U}_{\text{out}}^\ell \begin{bmatrix} \text{SA}([\mathbf{U}^{1,\ell}]^\top \mathbf{Z}, [\mathbf{U}^{1,\ell}]^\top \mathbf{Z}, [\mathbf{U}^{1,\ell}]^\top \mathbf{Z}) \\ \vdots \\ \text{SA}([\mathbf{U}^{K,\ell}]^\top \mathbf{Z}, [\mathbf{U}^{K,\ell}]^\top \mathbf{Z}, [\mathbf{U}^{1,\ell}]^\top \mathbf{Z}) \end{bmatrix} + \mathbf{b}_{\text{out}}^\ell \mathbf{1}_n^\top \quad (7.3.5)$$

where  $\mathbf{U}^{k,\ell} \in \mathbb{R}^{d \times p}$ ,  $\mathbf{U}_{\text{out}}^\ell \in \mathbb{R}^{d \times Kp}$ , and  $\mathbf{b}_{\text{out}}^\ell \in \mathbb{R}^d$  are trainable parameters belonging to the parameter set  $\theta$ , and (recall) the self-attention operator  $\text{SA}$  is defined in (7.2.16).

- The ISTA operator is the iterative-shrinkage-thresholding-algorithm operator, defined as follows:

$$\text{ISTA}_\theta^\ell(\mathbf{Z}) \doteq \text{ReLU}(\mathbf{Z} - \beta(\mathbf{D}^\ell)^\top (\mathbf{D}^\ell \mathbf{Z} - \mathbf{Z}) + \beta\lambda \mathbf{1}_d \mathbf{1}_n^\top), \quad (7.3.6)$$

so named because the map  $\mathbf{X} \mapsto \text{ReLU}(\mathbf{X} - \beta\mathbf{D}^\top(\mathbf{D}\mathbf{X} - \mathbf{Z}) + \beta\lambda \mathbf{1}_d \mathbf{1}_n^\top)$  is one step of the well-established ISTA algorithm to find an element-wise non-negative sparse representation for  $\mathbf{Z}$  with respect to the complete dictionary  $\mathbf{D}$  (cf Section 2.3).

We call this architecture CRATE, and a layer of the backbone is depicted in Figure 4.13. CRATE models, on top of being interpretable, are generally also highly performant as well as parameter-efficient.

### 7.3.3 Optimization

We train our classifier using a simple end-to-end stochastic optimization procedure, where we sub-sample data and views, compute the average loss and its gradient over these samples, and use an optimization algorithm to change the parameters. At each timestep  $k$ , we:

- Subsample  $B$  different labeled samples  $\{(\mathbf{X}_b^{(k)}, \mathbf{y}_b^{(k)})\}_{b=1}^B \subseteq \mathcal{I} \times \{0, 1\}^{N_{\text{cls}}}$ .
- For each labeled sample  $(\mathbf{X}_b^{(k)}, \mathbf{y}_b^{(k)})$ , compute the central crop view  $v_{b,\text{cc}}^{(k)}$  and apply it to  $\mathbf{X}_b^{(k)}$  to get  $\mathbf{X}_{b,\text{cc}}^{(k)} \doteq v_{b,\text{cc}}^{(k)}(\mathbf{X}_b^{(k)})$ .
- Compute the predictions  $\mathbf{p}_\theta(\mathbf{X}_{b,\text{cc}}^{(k)}) \doteq (h_\theta \circ f_\theta^{\text{ext}} \circ f_\theta)(\mathbf{X}_{b,\text{cc}}^{(k)})$ .
- Form the surrogate stochastic loss

$$\hat{\mathcal{L}}_{\text{CE}}^{(k)}(\theta) \doteq \frac{1}{B} \sum_{b=1}^B \text{CE}(\mathbf{y}_b^{(k)}, \mathbf{p}_\theta(\mathbf{X}_{b,\text{cc}}^{(k)})). \quad (7.3.7)$$

- Compute one step of an optimization algorithm on  $\theta$ , giving the following iteration:

$$\theta^{(k+1)} \doteq \text{OPTUPDATE}^{(k)}(\theta^{(k)}; \nabla_\theta \hat{\mathcal{L}}_{\text{CE}}^{(k)}). \quad (7.3.8)$$

### 7.3.4 Evaluation Methodology

We use the same evaluation procedure as Section 7.2.5. To summarize, for all evaluations (as well as training) we use a center crop view  $v_{\text{cc}}$  which reshapes the input image and takes a large central crop of size  $(C, S_{\text{cc}}, S_{\text{cc}})$  where  $C$  is the number of channels in the input image. We can then do linear probing, attention map visualization, and detection/segmentation benchmarks, given the output of this view.

Model	CRATE-T	CRATE-S	CRATE-B	CRATE-L	ViT-T	ViT-S
# parameters	6.09M	13.12M	22.80M	77.64M	5.72M	22.05M
ImageNet-1K	66.7	69.2	70.8	71.3	71.5	72.4
ImageNet-1K ReaL	74.0	76.0	76.5	77.4	78.3	78.4
CIFAR10	95.5	96.0	96.8	97.2	96.6	97.2
CIFAR100	78.9	81.0	82.7	83.6	81.8	83.2
Oxford Flowers-102	84.6	87.1	88.7	88.3	85.1	88.5
Oxford-IIIT-Pets	81.4	84.9	85.3	87.4	88.5	88.6

Table 7.3: **Linear probing classification accuracy of CRATE and ViT** on various datasets with different model sizes when the backbone is pre-trained for classification on ImageNet-1K. We observe that given the same model configuration, CRATE has comparable classification performance with a simpler, more principled, and more parameter-efficient design.

Model	Detection ( $\uparrow$ )			Segmentation ( $\uparrow$ )		
	AP <sub>50</sub>	AP <sub>75</sub>	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP
CRATE-S/8	<b>2.9</b>	<b>1.0</b>	1.1	1.8	<b>0.7</b>	0.8
CRATE-B/8	<b>2.9</b>	<b>1.0</b>	<b>1.3</b>	<b>2.2</b>	<b>0.7</b>	<b>1.0</b>
ViT-S/8	0.1	0.1	0.0	0.0	0.0	0.0
ViT-B/8	0.8	0.2	0.4	0.7	0.5	0.4

Table 7.4: **Object detection and fine-grained segmentation via MaskCut on COCO val2017** [LMB+14]. Here all models are trained with patch size 8 instead of 16. CRATE conclusively performs better than the ViT at detection and segmentation metrics when both are trained using supervised classification.

### 7.3.5 Experimental Setup and Results

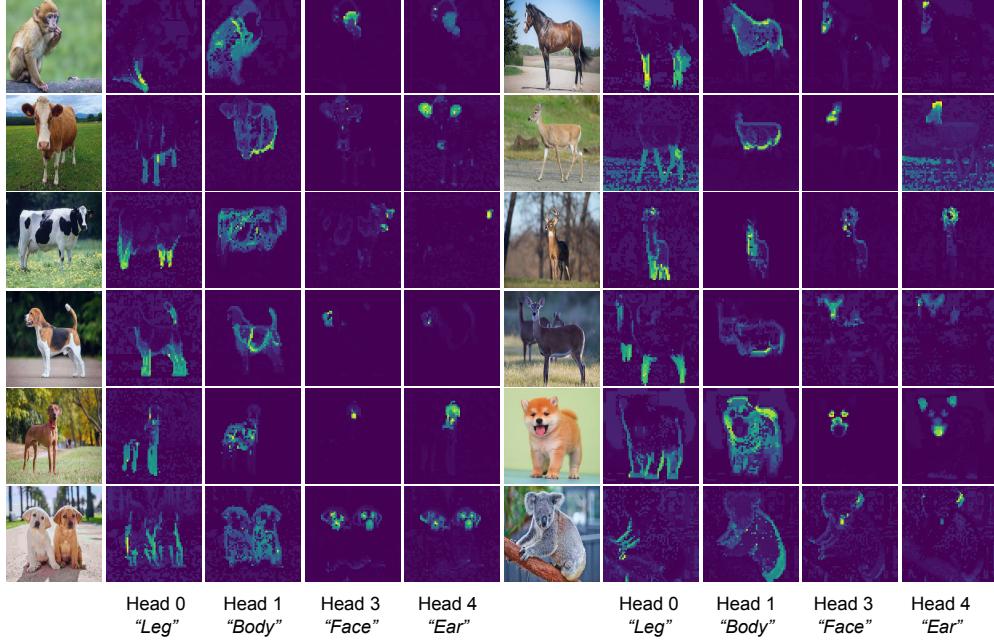
Since CRATE is directly based on the transformer, we compare the optimal settings for ViT as given by [DBK+21; TCD+20] with the same settings applied to CRATE for fair comparison.

**Model architecture.** The center crop resizes the whole image so that the shorter edge is of size 256 (i.e.,  $S_{\text{rsz}} = 256$ ) before taking a center crop of size  $224 \times 224$  (i.e.,  $S_{\text{cc}} = 224$ ), both in evaluation and training. We take patch size 16 (i.e.,  $P_H = P_W = 16$ ). We use the tiny, small, base, and large models of the ViT [DBK+21] architecture as the embedding and backbone, swapping out the MHSA and MLP components for MSSA and ISTA respectively, using the same number of heads and head dimension in the case of MSSA, and therefore reducing the number of training parameters drastically. For CRATE, we set  $(\beta, \lambda) = (1, 0.1)$ .

**Datasets and optimization.** For pre-training, we use the ImageNet-1K dataset. We use the LION optimizer [CLH+24] to pre-train both our ViT replication as well as CRATE. We set the base learning rate as  $2.4 \times 10^{-4}$ , the weight decay as 0.5, and batch size as  $B = 2048$ . Our learning rate schedule increases the learning rate linearly to the base learning rate over the first 5 epochs, and decreases to 0 using a cosine schedule over the next 145 epochs (training all models for 150 epochs each). For pre-training, we apply a usual regime of data augmentations (flips, Gaussian blurs, solarization, etc) to the image data, and also add small noise to the labels (this is called *label smoothing* [MKH19]).

For linear probing, we use several evaluation datasets such as CIFAR10, Oxford-Flowers, and Oxford-IIIT-Pets. We use the AdamW optimizer to train the linear probe, using learning rate  $5 \times 10^{-5}$ , weight decay 0.01, and batch size  $B = 256$ . We also apply the aforementioned data augmentations to the image data.

**Experiment results.** Table 7.3 demonstrates that CRATE models achieve parity or improvement compared to the popular Vision Transformer (ViT) architecture at similar parameter counts, at least in terms of the linear separability of their features w.r.t. different classes. In terms of attention map fidelity, Figure 7.11 demonstrates a truly extraordinary result: without needing to train on any segmentation or object detection



**Figure 7.11: Interpretable saliency maps in CRATE** with patch size 8. When given images with similar properties (perhaps but not necessarily from the same class), the saliency maps corresponding to different attention heads in the last layer each highlight a specific property. One can observe that the average saliency map (not included) then highlights all relevant objects in the image, showing that it uses all fine-grained details of the input image for classification. This is the *first* machine learning system to do this, to the authors' knowledge, much less automatically without training on any segmentation data.

data, *not only* do the saliency maps effectively capture all relevant parts of the input image, the saliency maps *self-organize* to each correspond to a discrete set of concepts, even across samples and classes! This is the first system to do this, to the authors' knowledge, and it can do this without using any extra data except for the image classification data. Table 7.4 confirms these qualitative insights quantitatively, showing significant improvement over ViTs trained in the same supervised classification setup.

## 7.4 Causal Language Modeling

We now study *causal language modeling*, a method for training large language models (LLMs). This is the same setup used to train, among many others, GPT-2 and many more language models.

### 7.4.1 Data

The data we will use to investigate the performance of CRATE for language tasks will be OpenWebText (OWT) [GC19], an open-source reproduction of the unreleased WebText dataset used by OpenAI to train GPT2. Each sample in OWT is a web document, typically sourced from high-quality web pages, blogs, articles, or online discussions, that is written in well-formed natural language. The OpenWebText dataset contains around 8.01M documents of varying lengths, totaling around 41.70GB of text. For evaluation, we will use several datasets, such as WikiText [MXB+16]<sup>3</sup>, LAMBADA [PKL+16]<sup>4</sup>, and PTB [MSM93]. PTB and OWT are generally easier compared to other datasets. PTB focuses on simpler journalistic text, ideal for traditional language modeling, while OWT is diverse and informal, covering various topics but with less complexity in language structure or long-range dependencies. WikiText, with its formal structure

<sup>3</sup>For WikiText2 and WikiText103 [MXB+16], the test splits are the same, so we merge them as a single dataset referred to as WikiText.

<sup>4</sup>To obtain the accuracy on LAMBADA dataset, we use greedy decoding.

and domain-specific content, requires a more complex understanding than OWT but remains manageable. LAMBADA is the most challenging, as it involves long-range dependencies, requiring the model to grasp broader contextual information to complete sentences accurately.

On a more formal level, our data  $\mathbf{X}$  will be text, or strings of characters; we let  $\mathcal{T}$  be the set of all strings.

### 7.4.2 Task and Objective

For causal language modeling pre-training, the idea is that we want to *train the model to output human-like text*. The most popular way to do this by far is to use a two-stage training process:<sup>5</sup>

- *First*, we wish to *learn* a way to optimally encode documents as a sequence of basic (“building block”) strings, called *tokens*. This is called *tokenization*, and we build a *tokenizer*.
- *Second*, we wish to *learn* a way to *predict the distribution of a token given all previous tokens*. This is called *next-token prediction*, and we build a *language model*.

This procedure actually dates back to Markov who first noticed that natural language could be modeled by the eponymous Markov chain structure [Mar06] given an appropriate tokenization, and then Shannon who proposed doing this exact language modeling setup with a character-level tokenizer (i.e., each character is a token) and so-called “ $n$ -gram” (i.e., an explicit look-up table, calculated from training data, for the distribution of a token given the  $n$  previous tokens) in place of the language model [Sha48].<sup>6</sup>

#### Training a Tokenizer

To build a tokenizer, it amounts to building a vocabulary  $\mathcal{V}$ , which is a set of tokens and has some pre-specified size  $V$ . There are several methods to do this. One popular algorithm is known as Byte Pair Encoding (BPE), which can be described as:

- Start with a list of all unique characters in your training data, and their frequencies. Ensure that there are fewer than  $V$  such characters, and add each character as a separate string (“token”) to the vocabulary along with its frequency.
- Until there are  $V$  tokens in the vocabulary:
  - Construct a token by taking the two most frequent existing tokens and merging them.
  - Compute this token’s frequency in the dataset.
  - Add it to the vocabulary (along with its frequency).
- At this point, the frequency information is no longer needed and can be discarded.

The overall process of BPE is in Figure 7.12. Note that this procedure is a modification of a classical information-theoretic compression procedure for *learning a lossless encoding* of bytestream data (such as text), and as such, one can interpret it as finding an optimal lossless compression of the data. Notice that this is possible because (unlike images), the data here are fundamentally discrete and noise-free hence. After such a vocabulary is built, a tokenizer can break down a document into tokens (i.e., “tokenize” it). BPE uses a similar procedure to tokenize data as in training:

- Separate the document into a long list of one-character-long tokens. That is, if the document is “Hello” then the initial list is ‘H’, ‘e’, ‘l’, ‘l’, ‘o’.
- While any two adjacent tokens can be concatenated and their concatenation is another token, we do it, i.e., we replace this pair of tokens with the merged token. Namely, if ‘He’ is a token in the vocabulary, ‘H’, ‘e’, ‘l’, ‘l’, ‘o’ would become ‘He’, ‘l’, ‘l’, ‘o’.

<sup>5</sup>Modern language model training has several additional training steps which demand different data distributions and algorithm approaches. However, training a model to merely mimic human writing only requires these few presented steps.

<sup>6</sup>A recent study [LMZ+24] scaling up  $n$ -gram models has shown that they are able to model text reasonably well for large  $n$ , but of course the memory required to store such a lookup table is of order  $V^n$  and hence completely intractable.

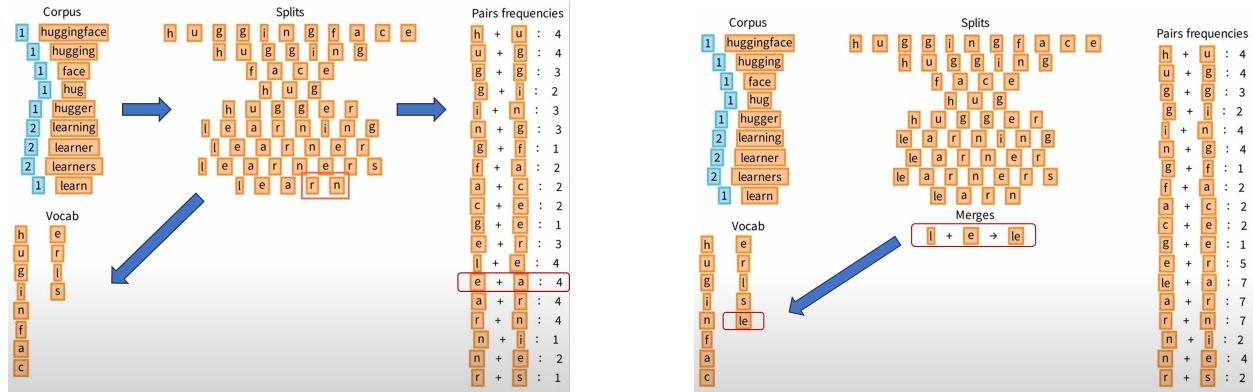


Figure 7.12: The process of tokenizing text data using BPE. (Image credit to <https://huggingface.co/learn/nlp-course/chapter6/5>). (Left) We begin by analyzing the given text corpus and constructing an initial vocabulary that consists of individual characters (or bytes in the case of byte-level BPE). Then, we compute the frequencies of adjacent character pairs in the corpus. This involves scanning the entire text and counting how often each two-character sequence (bigram) appears. (Right) After computing the frequencies of adjacent character pairs, we identify the most frequent pair in the corpus. This pair is then merged into a new subword unit, which is added to the vocabulary as a single token. This process is repeated iteratively until the predefined vocabulary size is reached.

- Repeat the above process until no more merges can be done. At this point, the document is partitioned into the final list (sequence) of tokens.

There are many practical and efficiency-based considerations to take into account during tokenization. The above algorithm, as presented, is *very far from optimal* if naively implemented, for instance. We do not cover this topic in great detail; there are many resources online to learn more, such as [HuggingFace tutorials](#).

For instance, each token has a corresponding *index* which is just its index in the vocabulary (which after all is just a list of length  $V$ ). Thus, the output of most tokenizers is a list of indices, say an element of  $[V]^*$ . Keep in mind that they correspond to substrings of the original document, as shown above.

Once a tokenizer is learned, it can be used as a black box by any language model. For instance, many models have the same (OpenAI-based) tokenizer based off of the `tiktoken` library. In the remainder of this section, we will use such a fixed and pre-built tokenizer for everything, and thus identify each text document  $\mathbf{X} \in \mathcal{T}$  with its tokenized version in  $[V]^*$ . Therefore, we may as well consider the text space  $\mathcal{T}$  as *equal* to the space of token sequences  $[V]^*$  (and lose nothing essential).

### Training a Language Model

Once we have each document as a sequence of tokens  $\mathbf{X} \in [V]^N \subseteq [V]^* = \mathcal{T}$ , we wish to perform next-token prediction. That is, given a *context*  $\mathbf{X}_{:n} \in [V]^n$  (i.e., the first  $n$  tokens  $\mathbf{x}_1, \dots, \mathbf{x}_n \in [V]$  in the document)<sup>7</sup>, we wish to predict the token  $\mathbf{x}_{n+1} \in [V]$  at position  $n + 1$ . To do this, we compute the aggregate feature of  $\mathbf{X}_{:n}$  via  $\mathbf{z}_\theta(\mathbf{X}_{:n}) \doteq (f_\theta^{\text{ext}} \circ f_\theta)(\mathbf{X}_{:n}) \in \mathbb{R}^d$ , and use a classification head  $h_\theta: \mathbb{R}^d \rightarrow \Delta_V$  (implemented as either a linear layer, MLP, or something slightly more complicated) to project this feature into the  $V$ -dimensional probability simplex  $\Delta_V$ . This projection  $\mathbf{p}_\theta(\mathbf{X}_{:n}) \doteq h_\theta(\mathbf{z}_\theta(\mathbf{X}_{:n}))$  serves as an estimated probability distribution of the next token. Then, using the notation  $\mathbf{1}(\mathbf{x}_{n+1}) \in \Delta_V$  to be 1 in the  $\mathbf{x}_{n+1}$ -th component and 0 elsewhere, the causal language modeling loss is

$$\min_{\theta} \left\{ \mathcal{L}_{\text{CLM}}(\theta) \doteq \mathbb{E}_{\mathbf{X}} \left[ \frac{1}{N-1} \sum_{n=1}^{N-1} \text{CE}(\mathbf{1}(\mathbf{x}_{n+1}), \mathbf{p}_\theta(\mathbf{X}_{:n})) \right] \right\} \quad (7.4.1)$$

Note how similar this is to a classification loss (say for images); one uses the cross-entropy and tries to align a predicted probability vector with the ground truth. The major difference between these two losses is that

<sup>7</sup>Note the incongruity with Python notation: here the notation *includes* index  $n$ .

in this one, we compute the loss on a whole sequence, where each prediction is correlated with each other (unlike in the i.i.d. classification case).

Optimizing this loss is usually called “pre-training” in the language model community (contrast with “post-training” and, more recently, “mid-training”, which are methodologies to modify a next-token-predictor for useful tasks).

*Side note:* Why does the first term of (7.4.1) predict  $\mathbf{1}(\mathbf{x}_2)$ , and there is no term which measures the loss to predict the first token? It’s because if we wanted to predict the first token, we would have the *empty sequence* as context, and therefore make this first token prediction using a qualitatively different mechanism than that which applies to the other tokens. So actually this model is not trained to predict the *very first* token of any document. The reason this is OK is due to an implementation detail of the tokenizer: often, after building the tokenizer, we insert a *special* token into its vocabulary, called the *beginning-of-string (or document)* token and labeled  $\langle \text{bos} \rangle$ .<sup>8</sup> Then, while processing each document, we add the  $\langle \text{bos} \rangle$  token at the beginning of the document’s token sequence, increasing the length of the tokenized sequence by 1. Thus the above causal language modeling objective has a term which involves trying to predict the first token of the document given only the  $\langle \text{bos} \rangle$  token as context, and so it is a conceptually correct loss.

### 7.4.3 Architecture: Causal CRATE

For the architecture, we use a standard GPT-2 style transformer, substituting CRATE layers in for the transformer layers.<sup>9</sup> For completeness, we specify the architecture here.

**Embedding.** We first embed the token sequence  $\mathbf{X} \in [V]^N$  to Euclidean space. This is often done by associating each index in  $[V]$  with a vector in  $\mathbb{R}^d$  using a *massive*<sup>10</sup> array  $\mathbf{E} \in \mathbb{R}^{V \times d}$ , and directly forming the sequence  $[\mathbf{E}_{\mathbf{x}_1}, \dots, \mathbf{E}_{\mathbf{x}_N}] \in \mathbb{R}^{d \times N}$ . The full embedding map  $f_\theta^{\text{emb}}$  also applies a positional encoding  $\mathbf{E}^{\text{pos}} \in \mathbb{R}^{d \times N_{\max}}$  where  $N_{\max}$  is the maximum number of tokens which are possible to process,<sup>11</sup> which yields the embedding map

$$f_\theta^{\text{emb}}(\mathbf{X}) \doteq [\mathbf{E}_{\mathbf{x}_1}, \dots, \mathbf{E}_{\mathbf{x}_N}] + \mathbf{E}_{:N}^{\text{pos}} \quad (7.4.2)$$

The parameters  $\mathbf{E}$  and  $\mathbf{E}^{\text{pos}}$  are directly trainable. Since  $\mathbf{E}$  is so large (and the gradient update is very sparse w.r.t. it since only a small fraction of the vocabulary is used in each sample), specialized software is used to make sure the memory updates are not too onerous. Notice also that we do not use a class token like in the other sections; more on this later.

**Backbone.** We process the embeddings using a CRATE-like backbone which uses causal masking. To motivate causal masking, consider the causal language modeling loss  $\mathcal{L}_{\text{CLM}}$  defined in (7.4.1). The most naive implementation would require us to compute the forward pass  $N$  times in order to backpropagate once. Obviously this is extremely inefficient, since  $N$  can often be in the thousands. In order to scale training with this loss efficiently, we impose a *causal* constraint, i.e.,

$$\mathbf{Z}_\theta(\mathbf{X}_{:n}) = \mathbf{Z}_\theta(\mathbf{X})_{:n} \quad (7.4.3)$$

i.e., the  $n$  columns of the token features  $\mathbf{Z}_\theta(\mathbf{X}_{:n}) \in \mathbb{R}^{d \times n}$  should be the same as the first  $n$  columns of the token features  $\mathbf{Z}_\theta(\mathbf{X}) \in \mathbb{R}^{d \times N}$  regardless of the positive values of  $n$  and  $N$  such that  $N \geq n$ . In effect, this means we can apply the backbone *once* to the whole sequence and compute  $\mathbf{Z}_\theta(\mathbf{X})$ , then apply  $f_\theta^{\text{ext}}$  to each increasing subset  $\mathbf{Z}_\theta(\mathbf{X}_{:n}) = \mathbf{Z}_\theta(\mathbf{X})_{:n}$  as  $n$  grows to the sequence length  $N$ . Then we can use all of those to compute the loss.

<sup>8</sup>There are usually several special tokens for different purposes. Existing text containing the special tokens are specially processed.

<sup>9</sup>In direct contravention of the conventions in this book and those of many other communities, the NLP community calls such GPT-2 style transformers (encompassing nearly all current LLMs) as “decoder-only” transformers. “Encoder-only” transformers have a different architecture, and “encoder-decoder” transformers concatenate an “encoder-only” transformer with a “decoder-only” transformer. This despite the fact that “decoder-only” transformers *also* compute an encoding of the data!

<sup>10</sup>By “massive” we mean that such a structure is often a large fraction of the language model’s total size.

<sup>11</sup>Modern positional encoding methods have since taken care of this issue and allowed for (in theory) infinite extrapolation, but such methods are more complex to develop, and for simplicity we only introduce the absolute additive positional encoding here.

So now that we want a causal architecture for the backbone, how can we get it? Since the MLP and layer normalizations inside each transformer layer affect each token individually, the only thing that matters for causality is the attention block (or MSSA in terms of CRATE). In order to make MSSA causal, we define the CausalMSSA block as

$$\text{CausalMSSA}_{\theta}^{\ell}(\mathbf{Z}) \doteq \mathbf{U}_{\text{out}}^{\ell} \begin{bmatrix} \text{CausalSA}([\mathbf{U}^{1,\ell}]^{\top} \mathbf{Z}, [\mathbf{U}^{1,\ell}]^{\top} \mathbf{Z}, [\mathbf{U}^{1,\ell}]^{\top} \mathbf{Z}) \\ \vdots \\ \text{CausalSA}([\mathbf{U}^{K,\ell}]^{\top} \mathbf{Z}, [\mathbf{U}^{K,\ell}]^{\top} \mathbf{Z}, [\mathbf{U}^{1,\ell}]^{\top} \mathbf{Z}) \end{bmatrix} + \mathbf{b}_{\text{out}}^{\ell} \mathbf{1}_N^{\top} \quad (7.4.4)$$

$$\text{where } \text{CausalSA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \doteq \mathbf{V} \text{ softmax}\left(\frac{\text{CausalMask}(\mathbf{K}^{\top} \mathbf{Q})}{\sqrt{p}}\right) \quad (7.4.5)$$

$$\text{where } \text{CausalMask}(\mathbf{M})_{ij} = \begin{cases} M_{ij}, & \text{if } i \geq j, \\ -\infty, & \text{if } i < j. \end{cases} \quad (7.4.6)$$

Here, practitioners say that the causal mask *allows future tokens  $i$  to attend to past tokens  $j$  but not vice versa*. To see why, let us write out the expression for the  $t^{\text{th}}$  column of  $\text{CausalSA}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ :

$$\text{CausalSA}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_t = \sum_{i=1}^t \mathbf{V}_i \text{ softmax}([\mathbf{K}_{:t}]^{\top} \mathbf{Q}_t)_i \quad (7.4.7)$$

(where here the non-colon subscript denotes the column). This expression for the  $t$ -th token uses no information about any token beyond index  $t$ . Therefore CausalSA, hence CausalMSSA, hence the whole causal CRATE backbone is causal in terms of the definition in (7.4.3), and we unlock the considerable efficiency gains that we were promised.

**Feature extractor.** We use a post-processing step  $f_{\theta}^{\text{ext}}$  which extracts the *feature vector of the last known token* so as to predict the next token. In theory this means that each token  $\mathbf{Z}_{\theta}(\mathbf{X})_n$  should contain rich information about all tokens that come before or on index  $n$ , i.e.,  $\mathbf{x}_1, \dots, \mathbf{x}_n$ , as all of this information should be available for predicting the next token at index  $n+1$ . In practice, only a few of these tokens are really needed for each prediction task. Anyways, the equation for  $f_{\theta}^{\text{ext}}$  is

$$f_{\theta}^{\text{ext}}(\mathbf{Z}_{\theta}(\mathbf{X}_{:n})) \doteq (\mathbf{Z}_{\theta}(\mathbf{X}))_n \quad (7.4.8)$$

where (again) the non-colon subscript is the column. In this case, as promised, we just directly extract the feature vector of the last token in the sequence.

**Task-specific head.** For our classification head  $h_{\theta}$ , the GPT-2 architecture uses a simple linear layer and a softmax to get the desired probability vectors:

$$h_{\theta}(\mathbf{z}) \doteq \text{softmax}(\mathbf{W}^{\text{out}} \mathbf{z} + \mathbf{b}^{\text{out}}), \quad (7.4.9)$$

where  $\mathbf{W}^{\text{out}} \in \mathbb{R}^{V \times d}$ ,  $\mathbf{b}^{\text{out}} \in \mathbb{R}^V$ . Some other more modern architectures use small MLPs and layer normalizations, but the idea is very much the same. Note that this linear layers also have large memory usage (because  $V$  is very large), and form a bottleneck in training; there has been significant effort attempting to circumvent it.

All these architectural choices mean that causal training is extremely efficient relative to non-causal training:

- We only need *one forward pass* through the backbone to compute the loss for the whole sequence.
- The feature extraction is basically *free*.
- All tokens can be pushed through the task-specific head *in parallel*.

### 7.4.4 Optimization Strategy

We train our language model using end-to-end stochastic optimization. One remaining issue is that, in practice, different documents in a batch have different lengths (in terms of the number of tokens required for each sequence), but as of the time of writing this book, the main deep learning frameworks by-and-large allow only “rectangular” tensors, which do not accommodate this behavior. To try to resolve this issue, we just insert a special padding token  $\text{<}|\text{pad}| \text{>}$  for all shorter samples in the batch, so that we can batch process everything using rectangular tensors. At each timestep  $k$ , we:

- Subsample  $B$  different tokenized documents  $\{\mathbf{X}_b^{(k)}\}_{b=1}^B \subseteq \mathcal{T} = [V]^*$ , each with length  $N_b^{(k)}$ .
- Compute  $N_{\max}^{(k)} \doteq \max_{b \in [B]} N_b^{(k)}$  and pad each  $\mathbf{X}_b^{(k)}$  to length  $N_{\max}^{(k)}$  using a special padding token.
- Compute the features  $\mathbf{Z}_\theta(\mathbf{X}_b^{(k)})$ .
- Compute the predicted distributions  $\mathbf{p}_\theta(\mathbf{X}_{b,:n}^{(k)}) \doteq (h_\theta \circ f_\theta^{\text{ext}})(\mathbf{Z}_\theta(\mathbf{X}_b^{(k)})_{:,n})$ .
- Form the surrogate stochastic loss

$$\hat{\mathcal{L}}_{\text{CLM}}^{(k)}(\theta) \doteq \frac{1}{B(N_{\max}^{(k)} - 1)} \sum_{b=1}^B \sum_{n=1}^{N_{\max}^{(k)} - 1} \text{CE}(\mathbf{1}(\mathbf{x}_{b,n+1}^{(k)}), \mathbf{p}_\theta(\mathbf{X}_{b,:n}^{(k)})). \quad (7.4.10)$$

- Compute one step of an optimization algorithm on  $\theta$ , giving the following iteration:

$$\theta^{(k+1)} \doteq \text{OPTUPDATE}^{(k)}(\theta^{(k)}; \nabla_\theta \hat{\mathcal{L}}_{\text{CLM}}^{(k)}). \quad (7.4.11)$$

### 7.4.5 Evaluation Methodology

There are several ways to evaluate a trained transformer language model.

- On a holdout dataset of arbitrary text, we can evaluate  $\mathcal{L}_{\text{CLM}}$  on it; lower losses are better since they mean the model’s sampling yields better performance.
- On a multiple choice question dataset, for each question we can put it as the context and check the estimated probability of the correct answer being generated.
- We can also test the *text generation* capabilities. Namely, we can repeatedly sample from the model’s probability distribution over the next token given the context. Each time we sample we generate a new token, which we print and add to the context. This allows us to sample from the LLM, and judge the generated samples however we please.<sup>12</sup>

In this section, we perform the first kind of evaluation exclusively.

### 7.4.6 Experimental Setup and Results

Since our causal CRATE architecture is directly built upon GPT-2, we compare the optimal settings for GPT-2 as given by the NanoGPT repository [Kar22a] with the same settings applied to CRATE for fair comparison.

**Model architecture.** We use the GPT-2 tokenizer, which has vocabulary size  $V = 50257$ , including a special token for  $\text{<}|\text{pad}| \text{>}^{13}$ . The context length is  $N_{\max} = 1024$ . The backbone model follows the GPT2-Base architecture [RWC+19] with the appropriate alterations to have causal CRATE layers, and we compare against GPT2-Small and GPT2-Base.

<sup>12</sup>Having to re-run the model on each token every time can become prohibitively expensive. Clever storages of different internal features of the language model (such as the so-called *K-V cache*), along with the causality of the architecture, can dramatically reduce the cost of sampling.

<sup>13</sup>The  $\text{<}|\text{bos}| \text{>}$  token is not included in this setup, although it is very common in modern language models.

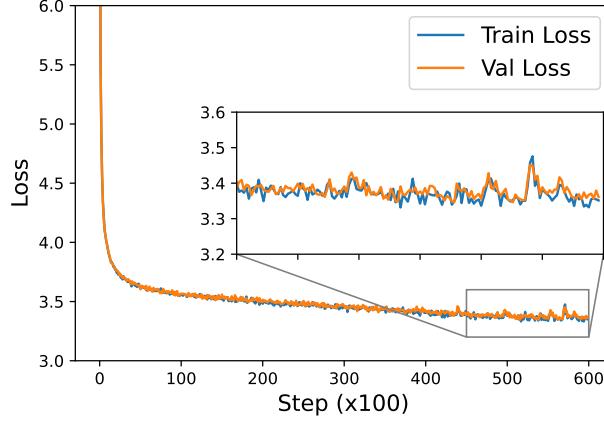


Figure 7.13: The loss curve of CRATE-GPT-Base trained on the OpenWebText dataset.

Table 7.5: Zero-shot cross-entropy loss of the CRATE-GPT2-Base model and GPT2-Small, GPT2-Base model evaluated on the test split of the datasets ( $\downarrow$  lower is better).

	#parameters	OWT	LAMBADA	WikiText	PTB	Avg
GPT2-Base	124M	2.85 $\downarrow$	4.12 $\downarrow$	3.89 $\downarrow$	4.63 $\downarrow$	3.87 $\downarrow$
GPT2-Small	64M	3.04	4.49	4.31	5.15	4.25
Causal-CRATE-Base	60M	3.37	4.91	4.61	5.53	4.61

**Datasets and optimization.** For training causal CRATE, we follow the implementations in the NanoGPT repository [Kar22a]. Specifically, we use a batch size of 384 and train for 600,000 steps with the Adam optimizer [KB14]. For the Adam optimizer, we use  $(\beta_1, \beta_2) = (0.9, 0.95)$  and weight decay of 0.1. For the learning rate schedule, we apply a linear warm-up and cosine decay, with a peak value of  $\eta = 6 \times 10^{-4}$  at the 2,000 iteration, and minimum value  $6 \times 10^{-5}$ . The training and validation losses over iterations are shown in Figure 7.13. The training/validation loss converges around 3.37 after training with a batch size of 384 and 600,000 iterations. In comparison, the open GPT-2 implementation is pre-trained on OpenWebText with a batch size of 512 and 600,000 steps and converges to a validation loss of 2.85 [Kar22a].

**Experiment results.** Table 7.5 demonstrates that CRATE models achieve reasonable performance on the causal language modeling loss across a variety of datasets compared to GPT-2 models with similar parameter counts and similar architectures.

## 7.5 Scaling White-Box Transformers

In this section, we will discuss three ways in which various parts of CRATE-type models can be scaled up or made more efficient while still remaining white-box. These developments mix both conceptual and empirical insights, and can be viewed as case studies about how to use white-box understanding to improve deep learning models in practice. The tasks that we use to evaluate the methods will be image classification and next-token-prediction, the data will be ImageNet and OpenWebText respectively, the optimization procedure will be the same backpropagation, and the only thing that changes is the architecture.

### 7.5.1 Increasing Network Width: CRATE- $\alpha$

One design decision enforced by the CRATE framework is that the *width* of the nonlinearity in the network. In a regular transformer, the width is usually set to 4, 8, or  $\frac{11}{3}$  times the feature dimension. However, CRATE enforces that the width is exactly equal to the feature dimension, i.e., the dictionaries  $D^\ell$  are square, which

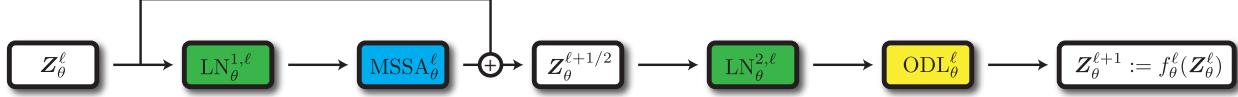


Figure 7.14: **One layer of the CRATE- $\alpha$  backbone.** The difference from CRATE is that the  $\text{ISTA}_\theta^\ell$  block is replaced by the  $\text{ODL}_\theta^\ell$  block, which performs several ISTA steps with an overcomplete dictionary.

could lead to reduced performance. The fundamental reason that the CRATE framework constrains us to this choice is as follows:

- The ISTA block takes a *single* step of optimization for dictionary learning.
- Usually one step of any iterative optimization algorithm cannot effectively optimize the objective. So then why does this work?
- Optimization algorithms usually converge very quickly if and only if they have good initializations, or *warm starts*. The ISTA block has a warm start — it treats the input features as an initialization to the resulting sparse codes.
- This enforces that the input features and sparse codes are have the same dimension. Namely, ISTA learns a complete sparsifying dictionary (cf Chapter 2).

Thus if we want to use a wide dictionary, we need ISTA to perform *overcomplete* dictionary learning. This means we cannot have the same warm start (as our sparse codes have larger dimension than our features), and need more iterations to converge to a sparse code. Hence the step from features  $Z_\theta^{\ell+1/2}$  to sparse codes  $Z_\theta^{\ell+1}$  would no longer be

$$Z_\theta^{\ell+1} = \text{ISTA}_\theta^\ell(Z_\theta^{\ell+1/2} | Z_\theta^{\ell+1/2}) \quad (7.5.1)$$

where the  $\text{ISTA}_\theta^\ell$  function is defined as (by an abuse of notation from earlier sections)

$$\text{ISTA}_\theta^\ell(\mathbf{Z} | \mathbf{Y}) \doteq \text{ReLU}(\mathbf{Z} - \beta(\mathbf{D}^\ell)^\top(\mathbf{D}^\ell \mathbf{Z} - \mathbf{Y}) + \beta\lambda \mathbf{1}_s \mathbf{1}_n^\top) \quad (7.5.2)$$

but rather the following iteration:

$$Z_\theta^{\ell+1} = \mathbf{A}_\theta^{\ell,T}; \quad \mathbf{A}_\theta^{\ell,t+1} = \text{ISTA}_\theta^\ell(\mathbf{A}_\theta^{\ell,t} | Z_\theta^{\ell+1/2}) \quad \forall 0 \leq t < T; \quad \mathbf{A}_\theta^{\ell,0} = \mathbf{0}_{s \times n}, \quad (7.5.3)$$

i.e., running proximal gradient on the LASSO objective for  $T \geq 1$  steps in the forward pass at each layer, initialized at  $\mathbf{0}_{s \times n}$ . In this circumstance, the dictionary can be as wide as needed, i.e.,  $\mathbf{D}^\ell \in \mathbb{R}^{s \times d}$  where  $s \geq d$  (usually one takes  $s = 4d$  in practice).

However, this presents an empirical problem. Using the above configuration, if  $Z^{\ell+1/2} \in \mathbb{R}^{d \times n}$  then  $Z^{\ell+1} \in \mathbb{R}^{s \times n}$ , which can have arbitrarily large feature dimension. In practice, we want the feature dimension at each layer to be the same. So this sets up a practical trichotomy for designing the network, namely, we *cannot* have *all* of the following desiderata:

Model	Detection				Segmentation	
	AP <sub>50</sub> ↑	AP <sub>75</sub> ↑	AP ↑	AP <sub>50</sub> ↑	AP <sub>75</sub> ↑	AP ↑
CRATE- $\alpha$ -B/8	3.5	1.1	1.5	2.2	1.0	1.1
CRATE- $\alpha$ -L/8	<b>4.0</b>	<b>1.7</b>	<b>2.0</b>	<b>2.7</b>	<b>1.1</b>	<b>1.4</b>
CRATE-B/8	2.9	1.0	1.3	2.2	0.7	1.0
ViT-B/8	0.8	0.2	0.4	0.7	0.5	0.4

Table 7.6: **Object detection and fine-grained segmentation via MaskCut on COCO val2017 [LMB+14].** Here all models are trained with patch size 8 instead of 16. Compared with existing models such as CRATE and ViT, the CRATE- $\alpha$  model family noticeably has improved performance as well as scalability.

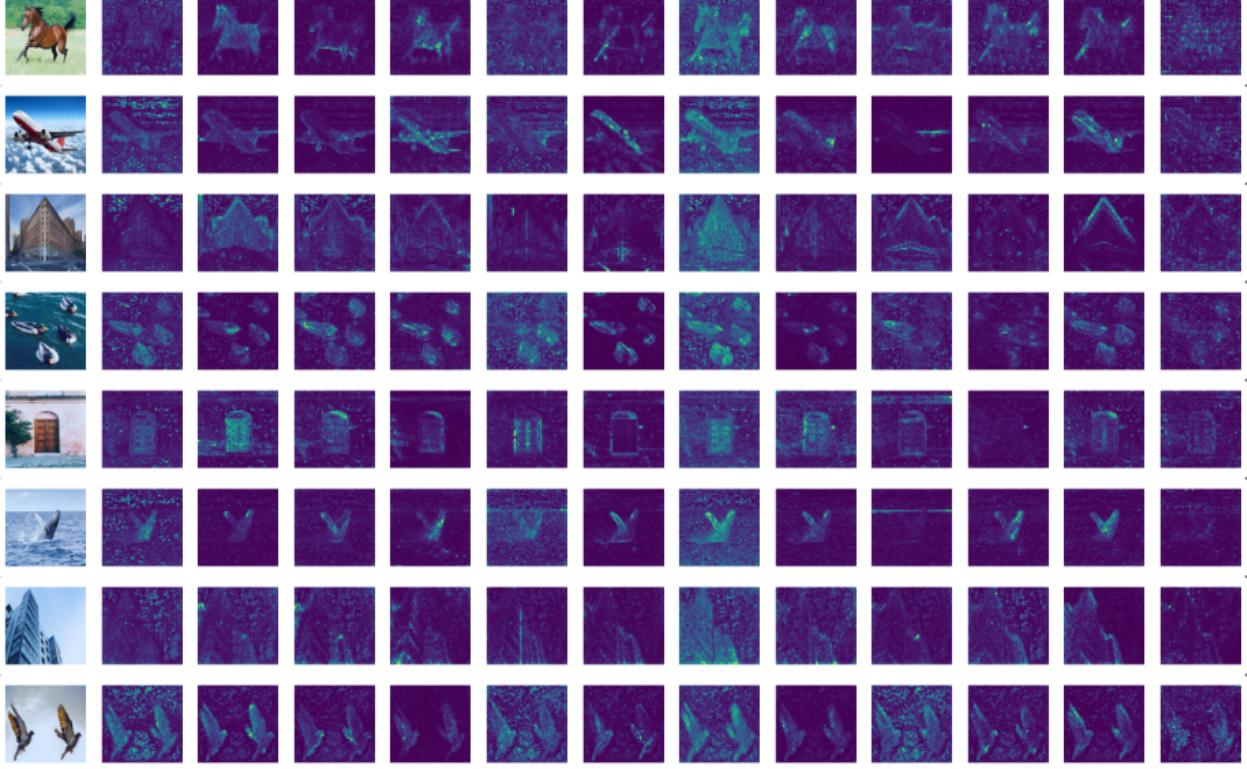


Figure 7.15: **Saliency maps from CRATE- $\alpha$  with patch size 8.** Each row is a different image and each column corresponds to a different attention head in the last layer. We observe that the saliency maps strongly correspond to the objects in the input image.

1. The feature dimension at each layer is the same.
2. The dictionary is wide, i.e., overcomplete.
3. The output of the nonlinearity is the sparse codes of the input w.r.t. the dictionary.

In practice, giving (1) is less tractable for efficiency reasons. Giving up (2) leads to the usual CRATE framework. Giving up (3) leads to a wide version of CRATE, i.e., CRATE- $\alpha$ , which has the following nonlinearity to get from  $\mathbf{Z}^{\ell+1/2}$  to  $\mathbf{Z}^{\ell+1}$ :

$$\mathbf{Z}_\theta^{\ell+1} = \mathbf{D}^\ell \mathbf{A}_\theta^{\ell,T}; \quad \mathbf{A}_\theta^{\ell,t+1} = \text{ISTA}_\theta^\ell(\mathbf{A}_\theta^{\ell,t} | \mathbf{Z}_\theta^{\ell+1/2}); \quad \mathbf{A}_\theta^{\ell,0} = \mathbf{0}, \quad (7.5.4)$$

i.e., takes the sparse codes obtained via proximal gradient descent and multiplies by the dictionary, to get the denoised version of the input. Thus CRATE- $\alpha$ 's nonlinearity computes a denoised version of the input which is amenable to sparse coding, not the actual sparse codes themselves. The map from  $\mathbf{Z}_\theta^{\ell+1/2}$  to  $\mathbf{Z}_\theta^{\ell+1}$  here is called the Overcomplete Dictionary Learning (ODL) block and denoted  $\text{ODL}_\theta^\ell$ , i.e.,

$$\mathbf{Z}_\theta^{\ell+1}(\mathbf{X}) \doteq \text{ODL}_\theta^\ell(\mathbf{Z}_\theta^{\ell+1/2}(\mathbf{X})). \quad (7.5.5)$$

The CRATE- $\alpha$  layer is shown in Figure 7.14. In practice this modification of CRATE performs very well at larger scales. For example, when we pre-train CRATE- $\alpha$  models on ImageNet-21K, unsupervised tasks like segmentation (see Figure 7.15 and Table 7.6) generally have significantly improved performance compared to CRATE. Similar trends are present in language model training using causal self-attention (see Table 7.7). Overall, it is a promising avenue to scaling up the performance to match black-box models such as transformers.<sup>14</sup>

<sup>14</sup>Note that the experimental results in this section use a slightly different model architecture, which add very slight empirical gains. The changes are: (1) an additional residual connection on the ODL block, (2) modifying ISTA to use two separate dictionaries instead of  $\mathbf{D}^\ell$  and  $(\mathbf{D}^\ell)^\top$ .

Model	GPT-2-B(ase)	CRATE-B	CRATE- $\alpha$ -S(mall)	CRATE- $\alpha$ -B
# parameters	124M	60M	57M	120M
OWT val. loss	2.85	3.37	3.28	3.14

Table 7.7: **Validation loss in language modeling.** Here all models are pre-trained on most of OpenWebText, and the validation cross-entropy loss is measured on a hold-out subset of OpenWebText. CRATE- $\alpha$  shows significant improvement over the CRATE design, though there still exists a gap with traditional transformers like GPT-2.

Datasets	ToST-T(iny)	ToST-S(mall)	ToST-M(edium)	XCiT-S	XCiT-M	ViT-S	ViT-B(ase)
# parameters	5.8M	22.6M	68.1M	24.9M	80.2M	22.1M	86.6 M
ImageNet	67.3	77.9	80.3	80.5	81.5	79.8	81.8
ImageNet ReaL	72.2	84.1	85.6	85.6	85.9	85.6	86.7
CIFAR10	95.5	96.5	97.5	98.1	98.3	98.6	98.8
CIFAR100	78.3	82.7	84.5	86.1	87.6	88.8	89.3
Oxford Flowers-102	88.6	92.8	94.2	93.9	94.0	94.0	95.7
Oxford-IIIT-Pets	85.6	91.1	92.8	92.9	94.0	92.8	94.1

Table 7.8: **Linear probing classification accuracy of ToST** on various datasets with different model sizes when the backbone is pre-trained for ImageNet-1K classification. We observe that, compared to the XCiT (a empirically-designed transformer-like architecture specialized for efficient processing of long sequences) and the ViT, ToST maintains relatively similar performance, even while enjoying benefits like faster runtime and white-box design.

### 7.5.2 Linear Time Complexity Transformers

In practice, deep learning models suffer bottlenecks to space and time complexity, representing problem sizes which they cannot scale beyond given fixed resources. One such bottleneck, particularly meaningful when dealing with data where each sample is itself high-dimensional and rich (such as long streams of text or videos), is the *time complexity* of processing long sequences of data. In order to alleviate the time-complexity of processing data using transformers, in Section 4.3.2 we proposed a *token statistics self-attention* operator TSSA $_{\theta}^{\ell}$ . We now build a *token statistics transformer*, called ToST, around it, which we can use for long-context tasks. In particular, we can use the following layer (depicted in Figure 7.16) as a drop-in replacement for a backbone layer in CRATE:

$$\mathbf{Z}_{\theta}^{\ell+1/2}(\mathbf{X}) = \mathbf{Z}_{\theta}^{\ell}(\mathbf{X}) + \text{TSSA}_{\theta}^{\ell}(\text{LN}_{\theta}^{1,\ell}(\mathbf{Z}_{\theta}^{\ell}(\mathbf{X}))) \quad (7.5.6)$$

$$\mathbf{Z}_{\theta}^{\ell+1}(\mathbf{X}) = \mathbf{Z}_{\theta}^{\ell+1/2}(\mathbf{X}) + \text{MLP}_{\theta}^{\ell}(\text{LN}_{\theta}^{2,\ell}(\mathbf{Z}_{\theta}^{\ell+1/2}(\mathbf{X}))) \quad (7.5.7)$$

where the TSSA block is defined as in Section 4.3.2. Notice that this is exactly the same as a vision transformer architecture as discussed in Section 7.2.3, except that TSSA replaces the conventional multi-head self-attention block MHSAs. Regardless, the computational complexity of the forward pass of this layer is linear in all problem variables — sequence length, feature dimension, number of heads, and head dimension.

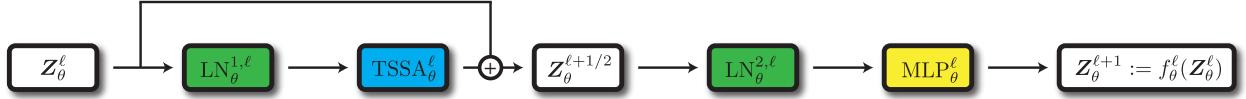


Figure 7.16: **One layer of the ToST backbone.** Token representations go through layer-norms, the token statistics self-attention (TSSA) operator, and an MLP, in order to form the layer’s output.

Moreover, the proposed architecture, named ToST (for “Token Statistics Transformer”) performs well at vision tasks (i.e., Table 7.8) and language tasks (i.e., Table 7.9). This is especially true for long-sequence-length tasks (cf Table 7.10), where it is both more performant and much more efficient than conventional transformers and all other transformer-like architectures.

Model	# params	OWT	Lambada	Wikitext	PTB	Avg ↓
GPT-2-Base	124M	2.84	4.32	4.13	5.75	4.26
ToST-Base	110M	3.20	4.98	4.77	6.39	4.84
ToST-Medium	304M	2.88	4.45	4.30	5.64	4.32
ToST-Large	655M	2.72	4.32	3.99	5.03	4.02

Table 7.9: **Language modeling validation loss** computed on (holdout sets of) a variety of natural language datasets, after pre-training the model on that dataset. We observe that ToST scales well, so that ToST-Large surpasses the baseline GPT-2-Base in causal language modeling, while enjoying superior efficiency in long contexts.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Avg
Reformer	<b>37.27</b>	56.10	53.40	38.07	68.50	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	54.17
LinFormer	16.13	<u>65.90</u>	53.09	42.34	<u>75.30</u>	50.46
Performer	18.01	65.40	53.82	42.77	<b>77.05</b>	51.18
Transformer	37.11	65.21	<u>79.14</u>	<u>42.94</u>	71.83	<u>59.24</u>
ToST	<u>37.25</u>	<b>66.75</b>	<b>79.46</b>	<b>46.62</b>	69.41	<b>59.90</b>

Table 7.10: **Long-Range Arena (LRA) performance comparison of ToST(-B) versus the top transformer variants optimized for long-context.** Long-Range Arena is a family of benchmarks that test the long sequence modeling capability of algorithms and architectures, by fixing the dataset and evaluation mechanism. ToST scores at the top of the leaderboard compared to all known transformer variants, including XCiT and the regular (ViT) transformer (cf Table 7.8). Moreover, ToST has the lowest time- and space-complexity inference. (In this table, the best score for a particular benchmark is bolded, and the second-best score is underlined.)

### 7.5.3 Attention-Only Transformers

Another bottleneck to remove from deep learning models, specifically transformer-like architectures, is the memory bottleneck which comes from massive matrix multiplications in MLPs, where the internal dimension is far greater than the feature dimension  $d$ . It thus is an interesting and important question to ask: do we *really* need the MLP inside a transformer, and how good can the performance get without it? To explore this question, we use the attention-only-transformer (AoT) architecture (see Section 4.3.1), depicted in Figure 7.17. Namely, each layer is simply of the form

$$\mathbf{Z}_\theta^{\ell+1}(\mathbf{X}) = \mathbf{Z}_\theta^\ell(\mathbf{X}) + \text{MSSA}_\theta^\ell(\text{LN}_\theta^{\ell, \ell}(\mathbf{Z}_\theta^\ell(\mathbf{X}))). \quad (7.5.8)$$

In our implementation, we also experimented with using multi-head self-attention (MHSA) in place of MSSA. It turns out that this architecture is *also* viable, though the depth of the network needs to be much deeper in order to achieve equivalent performance as the usual CRATE or transformer architecture.

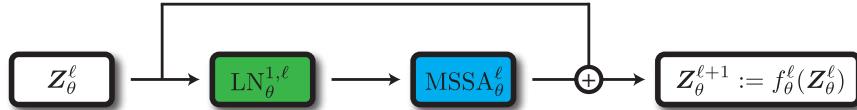


Figure 7.17: **One layer of the AoT backbone.** Token representations merely go through a layer-norm and the multi-head (subspace) self-attention operator to form the layer’s output. Notice that there is no token-wise nonlinearity such as MLP or ISTA or ODL.

We conduct the experiments using the proposed AoT architecture and demonstrate its potential. We pre-train the AoT-MSSA and AoT-MHSA models of different sizes, along with GPT-2, on OpenWebText [GC19]. We plot the training loss and validation loss against the number of training iterations in Figure 7.18(a) and (b), respectively. It is observed that medium- and large-sized AoT-based models achieve training and validation losses comparable to those of the GPT-2 base model. In addition, compared to the GPT-2

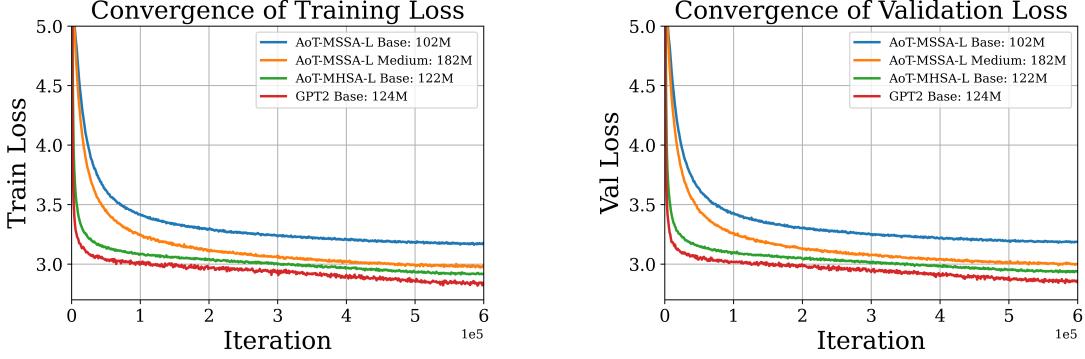


Figure 7.18: **Evaluating models on language tasks.** We plot the training loss (left) and validation loss (right) of the AoT and GPT-2 models pretrained on OpenWebText.

base model, the AoT-MHSA model is identical to the GPT-2 base model, except for the absence of MLP layers in the architecture. As shown in Figure 7.18, incorporating MLP layers can accelerate the training process. Using the above pre-trained models, we compute the cross-entropy validation loss without training on different datasets in Table 7.11. It is observed that the AoT models with medium and large parameter sizes can achieve comparable performance to the GPT-2 base model. Moreover, we found that adding MLP layers to AoT does not improve the zero-shot performance. These results highlight the potential of attention-only models to achieve competitive results while maintaining interpretability.

Table 7.11: Zero-shot results on several language benchmark datasets and tasks: Evaluation of different sizes of AoT with the MSSA and MHSA operators and comparison to the GPT2 model.

Models # of parameters	LAMBADA (val loss) ↓	PTB (val loss) ↓	WikiText (val loss) ↓	LAMBADA (acc) ↑	CBT CN (acc) ↑	CBT NE (acc) ↑
AoT-MSSA Base (102M)	4.70	6.03	4.65	0.25	0.80	0.74
AoT-MSSA Medium (182M)	4.47	5.08	4.22	0.29	0.84	0.77
AoT-MHSA Base (122M)	4.42	5.52	4.19	0.38	0.86	0.82
GPT-2 Base (124M)	4.32	5.75	4.13	0.40	0.87	0.84

## 7.6 Masked Autoencoding for Imagery Data

The second application we discuss is *nonlinear image completion*, also known as *masked autoencoding* (MAE), which is a direct generalization of the low-rank matrix completion problem discussed in Chapter 2. Masked autoencoding, since its introduction in the deep learning context by [HCX+22] has been a staple and simple self-supervised representation learning method, which aims to endow each patch feature within  $Z_\theta$  with aggregate information as well as information about its neighbors, such that both the patch feature and aggregate features are rich sources of information for the whole sample.

The dataset is kept to be the same imagery datasets as discussed in Section 7.2.1. As usual, we still apply data augmentations to each sample in each new batch.

### 7.6.1 Task and Objective

As the name suggests, masked autoencoding involves a view  $v_m$  which, given an input, performs a random resized crop (cf Section 7.2.2) to turn the input image into a square image of size  $(C, S_{\text{mask}}, S_{\text{mask}})$ , then *masks* (i.e., sets to zero) a fixed percentage  $p_{\text{mask}} \in [0, 1]$  of pixels in the input. For efficiency reasons<sup>15</sup>,

<sup>15</sup>The original implementation of MAE by [HCX+22] embeds the whole image, removes the tokens that would be masked, feeds the resulting token set through the encoder, adds back learned placeholder tokens in the masked spots and adds back the appropriate positional encoding, and feeds the resulting token set through the decoder to get the autoencoding prediction. This

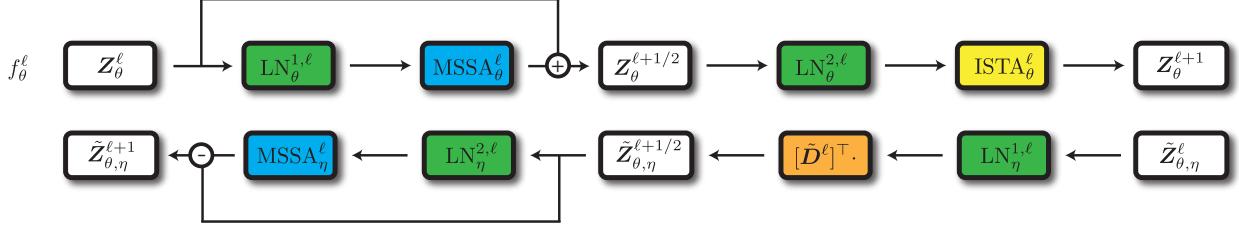


Figure 7.19: **One layer of the encoder and decoder in a CRATE autoencoder backbone.** The encoder and decoder layers both feed their inputs through multi-head subspace self-attention and a dictionary learning or dictionary encoding step. Note that the encoder and decoder layers are symmetrically designed; the conceptual goal of each decoder layer is to invert an encoder layer, so this symmetry is very much by design (see e.g., Chapter 5).

the masking is done patch-wise, i.e., after embedding the whole image,  $p_{\text{mask}}$  percentage of *patches* are set to zero. The goal of MAE is to train an encoder  $f_\theta: \mathcal{I} \rightarrow (\mathbb{R}^d)^*$  and a decoder  $g_\eta: (\mathbb{R}^d)^* \rightarrow \mathcal{I}$  which can reconstruct an input from its masking, i.e., writing  $\hat{\mathbf{X}}_{\theta,\eta} \doteq g_\eta \circ f_\theta$ , we have

$$\min_{\theta, \eta} \left\{ \mathcal{L}_{\text{MAE}}(\theta, \eta) \doteq \mathbb{E} \| \hat{\mathbf{X}}_{\theta,\eta}(\mathbf{X}_m) - \mathbf{X} \|_F^2 \right\} \quad (7.6.1)$$

Essentially this means that the features  $\mathbf{Z}_\theta(\mathbf{X}_m)$  of the view  $\mathbf{X}_m \doteq v_m(\mathbf{X})$  must contain information about the *masked patches* as well as the existing patches. From the perspective of the compression-based white-box models in Chapter 4, if a white-box autoencoder  $(f_\theta, g_\eta)$  succeeds at this task, it means that the learned subspaces and dictionaries perform a *redundant* encoding of the data such that it can reconstruct missing parts of the data from encoded other parts of the data. This means that information about each patch is stored in other patches. Therefore, each patch feature should contain both information about the patch and information about the statistics of the whole image. Thus, again, we expect that the representations should contain both local and global semantically relevant information, and the therefore representations of different patches with similar local and global information should be related (i.e., on the same subspace or encoded together by a dictionary).

## 7.6.2 Architecture

We use a CRATE encoder and decoder, depicted in Figure 7.7, though of course it is possible to use a regular transformer encoder and decoder. Details follow now.

**The encoder.** The encoder is the same as the CRATE encoder in Section 7.3.2, with the caveat that there is no feature extractor  $f_\theta^{\text{ext}}$ . However, both the embedding  $f_\theta^{\text{emb}}$  and the backbone  $f_\theta^{\text{bb}}$  are the same.

**The decoder backbone.** The decoder backbone is the CRATE decoder described in Chapter 5. For completeness' sake, we describe it now. Given a feature sequence  $\mathbf{Z}_\theta(\mathbf{X}) \doteq f_\theta(\mathbf{X}) \in (\mathbb{R}^d)^*$ , we can process it using the decoder backbone  $g_\eta^{\text{bb}}$  as follows. The function  $g_\eta^{\text{bb}}$  is composed of  $L$  layers  $g_\eta^\ell$ , i.e.,

$$g_\eta^{\text{bb}} = g_\eta^L \circ \cdots \circ g_\eta^1. \quad (7.6.2)$$

The layer  $g_\eta^\ell$  has the following implementation. First, define  $\tilde{\mathbf{Z}}_{\theta,\eta}^1(\mathbf{X}) \doteq \mathbf{Z}_\theta(\mathbf{X})$ . Then, we obtain

$$\tilde{\mathbf{Z}}_{\theta,\eta}^{\ell+1/2}(\mathbf{X}) = [\tilde{\mathbf{D}}^\ell]^\top \text{LN}_\eta^{1,\ell}(\tilde{\mathbf{Z}}_{\theta,\eta}^\ell(\mathbf{X})) \quad (7.6.3)$$

$$\tilde{\mathbf{Z}}_{\theta,\eta}^{\ell+1}(\mathbf{X}) = \tilde{\mathbf{Z}}_{\theta,\eta}^{\ell+1/2}(\mathbf{X}) - \text{MSSA}_\eta^\ell(\text{LN}_\eta^{2,\ell}(\tilde{\mathbf{Z}}_{\theta,\eta}^{\ell+1/2})) \quad (7.6.4)$$

and  $g_\eta^\ell$  is defined such that  $g_\eta^\ell(\tilde{\mathbf{Z}}_{\theta,\eta}^\ell) \doteq \tilde{\mathbf{Z}}_{\theta,\eta}^{\ell+1}(\mathbf{X})$ . Here, the relevant concept is that  $g_\eta^\ell$  should learn an approximate inverse of  $f_\theta^{L+1-\ell}$ , as discretizations of a forward- and reverse-time diffusion process, respectively.

is more efficient since the encoder has fewer tokens to go through, but conceptually is the same as the method discussed in the text, and the resulting models' performance in the masked autoencoding task and downstream evaluations is very similar.

In particular,  $\tilde{\mathbf{D}}^\ell$  should approximate  $\mathbf{D}^{L+1-\ell}$ , and similarly the  $\text{MSSA}_\eta^\ell$  parameters should be similar to the parameters of  $\text{MSSA}_\theta^{L+1-\ell}$ . The output is  $\tilde{\mathbf{Z}}_{\theta,\eta} \doteq \tilde{\mathbf{Z}}_{\theta,\eta}^{L+1}$ .

**The un-embedding module.** To transform  $\tilde{\mathbf{Z}}_{\theta,\eta}(\mathbf{X})$  back into an estimate for  $\mathbf{X}$ , we need to undo the effect of the embedding module  $f_\theta^{\text{emb}}$  using the unembedding module  $g_\eta^{\text{unemb}}$ . As such, harkening back to the functional form of the embedding module in (7.2.11), i.e.,

$$f_\theta^{\text{emb}}(\mathbf{X}) \doteq [\mathbf{z}_{\text{cls}}^1, \mathbf{W}^{\text{emb}} f^{\text{patch}}(\mathbf{X}) + \mathbf{E}^{\text{pos}}] \quad (7.6.5)$$

it implies that our inverse operation  $g_\eta^{\text{unemb}}$  looks like the following:

$$g_\eta^{\text{unemb}}(\tilde{\mathbf{Z}}) \doteq g_\eta^{\text{unemb}}([\tilde{\mathbf{z}}^1, \dots, \tilde{\mathbf{z}}^n]) = g^{\text{unpatch}}(\mathbf{W}^{\text{unemb}}([\tilde{\mathbf{z}}^2, \dots, \tilde{\mathbf{z}}^n] - \tilde{\mathbf{E}}^{\text{pos}})), \quad (7.6.6)$$

where  $g^{\text{unpatch}}$  does the inverse operation of the unrolling and flattening operation that  $f^{\text{patch}}$  does.<sup>16</sup>

This architecture is a white-box autoencoder  $(f_\theta, g_\eta)$  where (recall)  $f_\theta = f_\theta^{\text{bb}} \circ f_\theta^{\text{emb}}$  and  $g_\eta = g_\eta^{\text{unemb}} \circ g_\eta^{\text{bb}}$ . In particular, we can use it to compute an estimate for a masked view  $\hat{\mathbf{X}}_{\theta,\eta}(\mathbf{X}_m) = (g_\eta \circ f_\eta)(\mathbf{X}_m)$  which should approximately equal  $\mathbf{X}$  itself.

### 7.6.3 Optimization

As in Section 7.3.3, we use a simple optimization setup: we sample images and masks, compute the loss on those samples and the gradients of this loss, and update the parameters using a generic optimization algorithm and the aforementioned gradients. For each timestep  $k$ , we:

- Subsample  $B$  different samples  $\{\mathbf{X}_b^{(k)}\}_{b=1}^B \subseteq \mathcal{I}$ .
- For each sample  $\mathbf{X}_b^{(k)}$ , compute a different randomized resized crop and mask  $v_{b,m}^{(k)}$  and apply it to  $\mathbf{X}_b^{(k)}$  to get  $\mathbf{X}_{b,m}^{(k)} \doteq v_{b,m}^t(\mathbf{X}_b^{(k)})$ .
- Compute the estimated autoencoding  $\hat{\mathbf{X}}_{\theta,\eta}(\mathbf{X}_{b,r}^{(k)}) \doteq (g_\eta \circ f_\theta)(\mathbf{X}_{b,r}^{(k)})$ .
- Form the surrogate stochastic loss

$$\hat{\mathcal{L}}_{\text{MAE}}^{(k)}(\theta, \eta) \doteq \frac{1}{B} \sum_{b=1}^B \|\hat{\mathbf{X}}_{\theta,\eta}(\mathbf{X}_{b,r}^{(k)}) - \mathbf{X}_b^{(k)}\|_F^2. \quad (7.6.7)$$

- Compute one step of an optimization algorithm on  $(\theta, \eta)$ , giving the following iteration:

$$(\theta^{(k+1)}, \eta^{(k+1)}) \doteq \text{OPTUPDATE}^{(k)}(\theta^{(k)}, \eta^{(k)}; \nabla_{(\theta,\eta)} \hat{\mathcal{L}}_{\text{MAE}}^{(k)}). \quad (7.6.8)$$

### 7.6.4 Evaluation

This is the first autoencoder network we discuss in this chapter. We use the same center crop view  $v_{\text{cc}}$  as in Sections 7.2.5 and 7.3.4, resizing the final image to a square with side length  $S_{\text{cc}} = S_{\text{mask}}$  pixels so as to match the shapes of the input images seen during training.

On top of evaluating the masked autoencoding loss itself, it is also possible to evaluate the features  $\mathbf{Z}_\theta(\mathbf{X}_{\text{cc}})$  of the view  $\mathbf{X}_{\text{cc}} \doteq v_{\text{cc}}(\mathbf{X})$  of the data  $\mathbf{X}$  directly. For the sake of attention map fidelity evaluation, just obtaining  $\mathbf{Z}_\theta(\mathbf{X}_{\text{cc}})$  is enough, but for linear probing we need to extract a summarized or aggregate feature from  $\mathbf{Z}_\theta$ . To do this, we can use a (parameter-free) feature extraction map which just returns the feature corresponding to the class token, i.e.,

$$f_\theta^{\text{ext}}(\mathbf{Z}) \doteq f_\theta^{\text{ext}}([\mathbf{z}^1, \dots, \mathbf{z}^n]) = \mathbf{z}^1, \quad (7.6.9)$$

as in (for example) Sections 7.3.1 and 7.3.2. With this, we have a way to obtain aggregate features  $\mathbf{z}_\theta(\mathbf{X}_{\text{cc}}) \doteq (f_\theta^{\text{ext}} \circ f_\theta)(\mathbf{X}_{\text{cc}})$ , at which point we can perform linear probing, segmentation evaluations, and so on.

---

<sup>16</sup>Again, the “inverse positional encoding”  $\tilde{\mathbf{E}}^{\text{pos}}$  is learned for a large input, and for smaller inputs may be interpolated. It is even possible to directly set  $\tilde{\mathbf{E}}^{\text{pos}}$  equal to the positional encoding  $\mathbf{E}^{\text{pos}}$  and use the same interpolated positional encodings for each input in both the encoder and decoder.

Model	CRATE-MAE-S(mall)	CRATE-MAE-B(ase)	ViT-MAE-S	ViT-MAE-B
# parameters	25.4M	44.6M	47.6M	143.8M
CIFAR10	79.4	80.9	79.9	87.9
CIFAR100	56.6	60.1	62.3	68.0
Oxford Flowers-102	57.7	61.8	66.8	66.4
Oxford-IIIT-Pets	40.6	46.2	51.8	80.1

Table 7.12: **Linear probing classification accuracy of CRATE-MAE and ViT-MAE** on various datasets with different model sizes when the backbone is pre-trained for masked autoencoding on ImageNet-1K. Given the same parameter count, CRATE-MAE achieves roughly similar performance, while simultaneously enjoying a simpler and more principled architecture design.

### 7.6.5 Experiments

Since CRATE-MAE is directly based on the ViT-MAE, we compare the optimal settings for ViT-MAE as given by [H CX+22] with the same settings applied to CRATE-MAE for fair comparison.

**Model architecture.** During training, the masked crop  $v_m$  resizes the whole image so that the shorter edge is of size 256 (i.e.,  $S_{\text{rsz}} = 256$ ) before taking a random crop of size  $224 \times 224$  (i.e.,  $S_{\text{mask}} = 224$ ), and masking  $p_{\text{mask}} = \frac{3}{4}$  of the patches. We take patch size 16 (i.e.,  $P_H = P_W = 16$ ). We use the small and base variants of the ViT-MAE architecture as the embedding and backbone for both the encoder and decoder, swapping out the MHSA and MLP components for MSSA, ISTA, and linear layers respectively. We use the same number of heads and head dimension in the case of MSSA. However, the original ViT-MAE uses an encoder which uses nearly all the total layers and a decoder which only uses a few layers; we allocate half the total number of layers (which stays the same from ViT-MAE to CRATE-MAE) to our encoder and decoder, as suggested by the conceptual and theoretical framework in Chapter 5. For CRATE-MAE we set  $(\beta, \lambda) = (1, 0.1)$ .

**Datasets and optimization.** For pre-training, we use the ImageNet-1K dataset. We use the AdamW optimizer to pre-train both our ViT-MAE replication as well as CRATE-MAE. We set the base learning rate as  $3 \times 10^{-5}$ , the weight decay as 0.1, and batch size as  $B = 4096$ . Our learning rate schedule increases the learning rate linearly to the base learning rate over the first 40 epochs, and decreases to 0 using a cosine schedule over the next 760 epochs (training all models for 800 epochs each). For pre-training, we apply the usual regime of data augmentations (flips, Gaussian blurs, solarization, etc) to the image data.

For linear probing, we use several evaluation datasets such as CIFAR10, CIFAR100, Oxford-Flowers, and Oxford-IIIT-Pets. For linear probing, we precompute the feature of all samples in the target dataset and apply a fast linear regression solver, e.g., from a standard package such as Scikit-Learn.

**Experiment results.** Table 7.12 demonstrates that CRATE-MAE models achieve, roughly speaking, parity compared to the popular ViT-MAE architecture at similar parameter counts, and also that the feature learning performance (as measured by performance on downstream classification tasks) increases with scale. Meanwhile, Figure 7.20 demonstrates that the encoder saliency maps (and therefore the fine-grained features learned by the encoder) indeed isolate and highlight the key parts of the input image.

## 7.7 Summary and Notes

All work in this chapter is downstream of the Transformer architecture, which was introduced by Vaswani et al. [VSP+17]. The Transformer architecture is formally described in Section 7.2. A main empirical innovation in recent years, spurred by the prevalence and performance of the transformer architecture, is to formulate a given learning problem as a sequence-to-sequence problem and apply the transformer architecture. This has enabled the transformer architecture to be essentially ubiquitous in (almost) all deep learning applications. As such, direct improvements to the transformer can propagate to become solutions

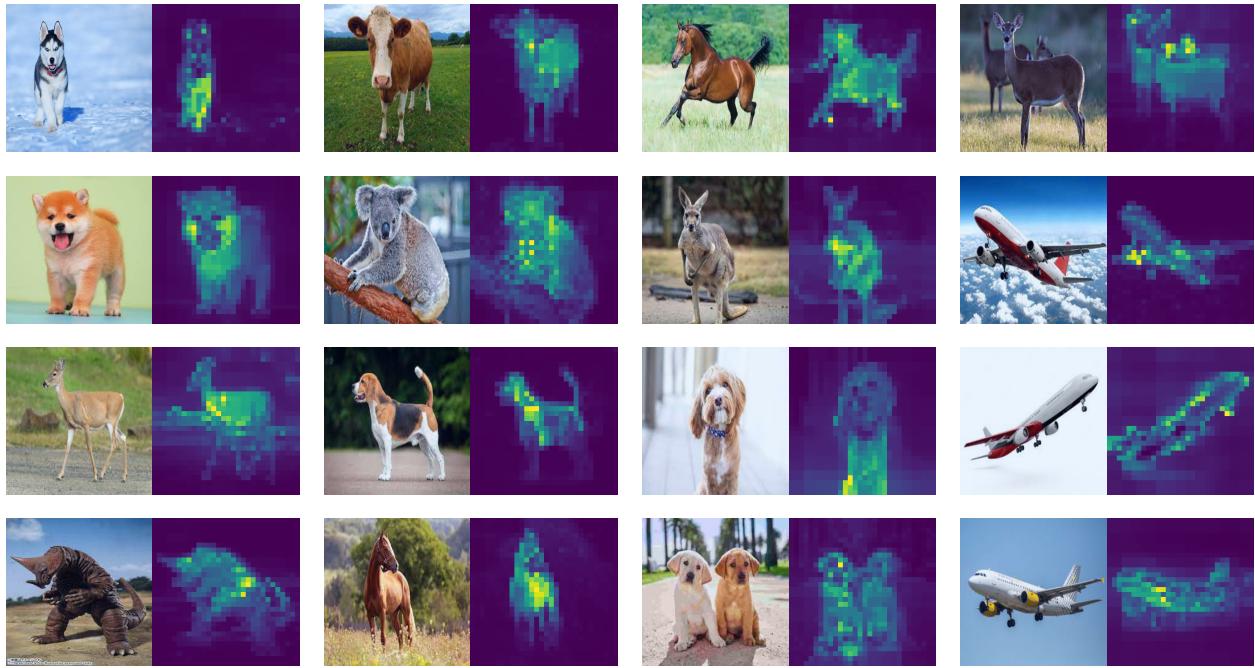


Figure 7.20: **Saliency maps of CRATE-MAE.** Each pair of images consists of the original image (left) and a selected saliency map (right) corresponding to an attention head in the last layer. As is usual for CRATE models, but unusual for general transformer-like models, the saliency maps correspond to the objects in the input image.

to many problems and have considerable impact; similarly, we can apply our white-box understanding of transformer-like architectures to many modern problems. The material covered in this Chapter is merely a subset of the work which has already been done; other work includes masked completion for text data (i.e., BERT) [DCL+19; YBP+24], (mechanistic) interpretability of language and vision models [BM24], and error correcting coding [ZLG+]. There is much more to do.

There is also much more theory specifically about the practice of scaling neural networks, which is enormously practically viable, and we at least remark on it here. This line of work was popularized by the “Tensor Programs” line of work [YHB+22]. The basic prescription is that we want the initial gradient updates in a transformer to be constant size, and by working through the backpropagation equations (Chapter A) carefully, we can determine the scale of the initialization and learning rates (chosen layer-wise) that are required to achieve this. In practice, such prescriptions greatly increase the stability and convergence of training at large scales; they also prescribe a way to find the “optimal”<sup>17</sup> hyperparameters for large-scale training using only small-scale training. Follow-ups to this work attempt to accommodate the feature geometry [BN24a], which could be informed by the work in this book about representation learning. Other follow-ups incorporate this weight-wise information into the optimizer itself to obtain these scaling benefits automatically, obtaining optimizers such as Muon [JJB+], which have recently been used for training trillion-parameter models very stably [Tea]. Overall, the two approaches to deep learning theory are orthogonal or complementary.

## 7.8 Exercises and Extensions

*Exercise 7.1.* Read the DINO paper [CTM+21].

*Exercise 7.2.* DINO v2 [ODM+23] uses everything from DINO v1 but also, during the data augmentation phase, randomly masks out patches within each view. This kind of augmentation should enforce that the

---

<sup>17</sup>The word “optimal” is used in quotes because the work on this merely uses some desiderata about the weight size, feature size, and gradient size at initialization to determine “optimality”, as opposed to, say, the test loss at convergence.

features of images with similar local information are similar. Formulate an optimization problem which promotes this in the encoder, and implement it.

*Exercise 7.3.* This exercise considers the implementation of stochastic optimization algorithms to minimize losses involving expectations.

- (a) Propose an alternative to the term involving  $R_\varepsilon$  in (7.2.34) for approximating the covariance regularization term in (7.2.33). Evaluate the time complexity required to compute your proposed term and its gradient. Include analysis for computing it on a single compute node vs. multiple nodes.
- (b) Evaluate the time complexity required to compute the existing term in (7.2.34) and its gradient.

*Exercise 7.4.* Prove that (7.2.37) and (7.2.38) are convex optimization problems.

*Exercise 7.5.*

- (a) Implement the CRATE and CRATE- $\alpha$  models.
- (b) Compare their performance and efficiency on the CIFAR-10 dataset.
- (c) Compare their interpretability in two ways:
  - The sparsity  $\|\mathbf{Z}\|_0$  of the representation  $\mathbf{Z}$ .
  - The attention maps  $\mathbf{a}_\theta^{k,\ell}$ .

## Chapter 8

# Future Study of Intelligence

*“The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problem now reserved for humans, and improve themselves.”*

– Proposal for the Dartmouth AI program, 1956

Generally speaking, this manuscript is meant to systematically introduce mathematical principles and computational mechanisms for how memory or knowledge can be developed from empirical observations. The capability to seek parsimony in a seemingly random world is a fundamental characteristic of any intelligence, natural or man-made. We believe that the principles and mechanisms presented in this book are rather unifying and universal and are applicable to both animals and machines.

We hope that this book can help the readers fully clarify the mystery around modern practices of artificial deep neural networks by developing a rigorous understanding of their functions and roles in achieving the objective of learning low-dimensional distributions from high-dimensional data. With such a understanding, we should have become rather clear both capabilities and limitations of existing AI models and systems:

1. Existing models and systems are short of being complete in terms of a memory system that is capable of self-learning and self-improving.
2. Existing realizations of these functions are still rather primitive and brute force and certainly far from optimal in terms of optimization strategies hence network architectures.
3. Existing AI models only learn the data distribution and conduct inductive (Bayesian) inference, which is different from the high-level human intelligence.

One of the goals of this book is for people to establish an objective and systematic understanding of current machine intelligence technologies and to realize what open problems and challenges remain ahead for further advancement of machine intelligence. In the last chapter of the book, we provide some of our views and projections for the future.

### 8.1 Towards Autonomous Intelligence: Close the Loop?

From the practice of machine intelligence in the past decade, it has become clear that, if there were sufficient data and computational resources, one could build a large enough model and pre-train it to learn the *a priori* distribution of all the data, say  $p(\mathbf{x})$ . Theoretically, such a large model can memorize almost all existing knowledge about the world that has been encoded in massive languages and texts. As we have discussed at the beginning of the book, in a way, such a large model plays a similar role as DNAs with which life uses to record and pass on knowledge about the world.

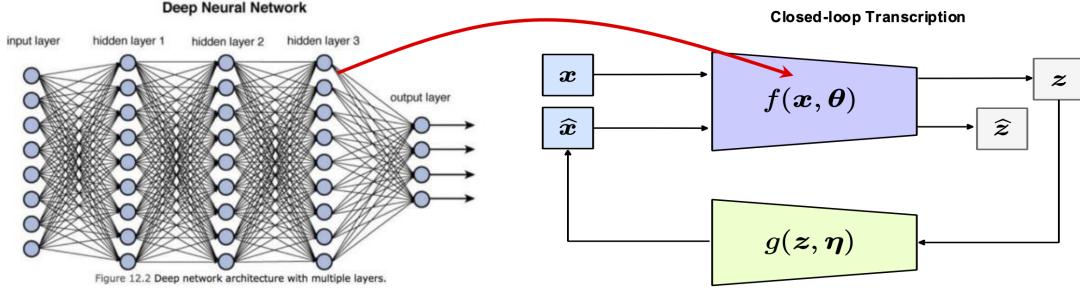


Figure 8.1: From an open-ended deep network to a closed-loop system.

The so learned model and distribution can then be used to regenerate new data samples based on the same distribution. One can also use the model to conduct inference (e.g., estimation, prediction) with the memorized knowledge under various conditions, say by sampling the *a posteriori* distribution  $p(\mathbf{x} | \mathbf{y})$  under a new observation  $\mathbf{y}$ . Strictly speaking, such inference is statistical.

Any pre-trained model, however large, cannot guarantee that the distribution that it has learned so far is entirely correct or complete. In case our samples  $\hat{\mathbf{x}}_t$  from the current *a priori*  $p_t(\mathbf{x})$  or estimates  $\hat{\mathbf{x}}_t(\mathbf{y})$  based on the *a posteriori*  $p_t(\mathbf{x} | \mathbf{y})$  are inconsistent with the truth  $\mathbf{x}$ , we would very much like to correct the learned distributions:

$$p_t(\mathbf{x}) \rightarrow p_{t+1}(\mathbf{x}) \quad \text{or} \quad p_t(\mathbf{x} | \mathbf{y}) \rightarrow p_{t+1}(\mathbf{x} | \mathbf{y}), \quad (8.1.1)$$

based on the error  $\mathbf{e}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ . This is known as error correction based on error feedback, an ubiquitous mechanism in nature for continuous learning. However, as we know, any open-ended model itself does not have the mechanisms to revise or improve the learned distribution when it is incorrect or incomplete. Improving current AI models still largely depends on human involvement: experimentation, evaluation, and selection. We may call this process as “artificial selection” of large models, as opposed to the natural selection for the evolution of lives.

As we have studied earlier in this book (Chapter 5 in particular), closed-loop systems allows to align an internal representation with the (sensed) observations of the external world. It can continue to improve the internally learned distribution and its representation to achieve consistency or self-consistency. An immediate step forward for the future is to develop and build truly closed-loop memory systems, as shown in Figure 8.1, that are capable of learning and improving more general data distributions autonomously and continuously based on error feedback.

Therefore, transition from the current popular end-to-end trained open-loop models to continuously-learning closed-loop systems:

$$\text{open-ended models} \implies \text{closed-loop systems} \quad (8.1.2)$$

is the key for machines to truly emulate how the (animal) brain learns and applies knowledge in an open world. We believe that

*open-ended models are for a closed world, however large;  
closed-loop systems are for an open world, however small.*

In fact, “general intelligence” could never be achieved by simply having memorized all existing knowledge of the world. Instead, general intelligence can only be achieved by having the mechanisms to improve its existing memory so as to be able to adapt to any new environments and tasks.

## 8.2 Towards Intelligence of Nature: Beyond Back Propagation?

The practice of machine intelligence in the past few years has led many to believe that one needs to build a single large model to learn the distribution of all data and memorize all knowledge. Even if this might be technologically possible, it is likely that such a solution is far from necessary and efficient. As we have

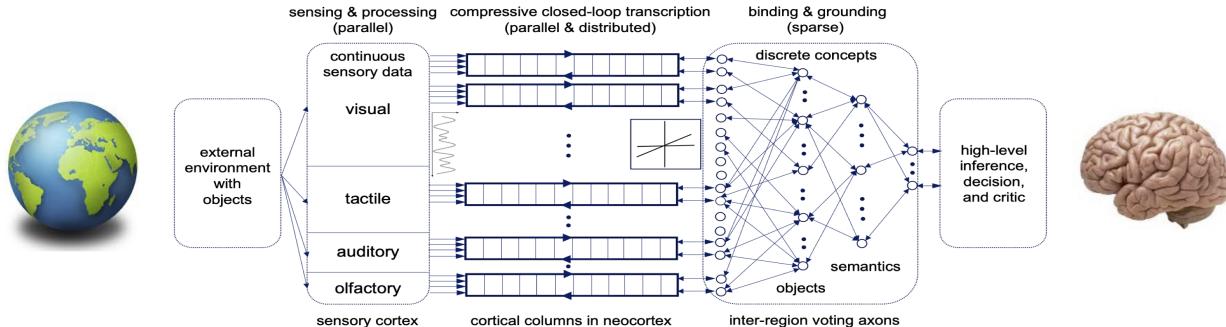


Figure 8.2: Conjectured architecture of the brain cortex: The cortex is a massively parallel and distributed auto-encoding system that consists of a hierarchy of closed-loop auto-encoders that extract information from multiple senses and maximize the information gain of the resulting representations at multiple levels of hierarchy and granularity.

known from the practice of training deep networks, the only known scalable method to train such networks at scale is through back propagation (BP) [RHW86b]. Although BP has offered a way to correct errors via gradient signals propagated back through the whole model, it is nevertheless rather brute force and differs significantly from how nature learns: BP is an option that nature cannot afford in terms of its high cost or simply cannot implement due to physical limitations.

More generally, we cannot truly understand intelligence unless we also understand how it can be efficiently implemented. That is, one needs to address the computational complexity of realizing mechanisms associated with achieving the objectives of intelligence. Note that, historically, our understanding of (machine) intelligence has precisely evolved through several phases, from the incomputable Kolmogorov complexity to Shannon’s entropy, from Turing’s computability to later understanding of tractability,<sup>1</sup> and to the strong emphasis on algorithm scalability in modern practice of artificial intelligence. This evolution can be summarized as the following diagram:

$$\text{incomputable} \implies \text{computable} \implies \text{tractable} \implies \text{scalable}. \quad (8.2.1)$$

To a large extent, the success and popularity of deep learning and back propagation is precisely because they have offered a scalable implementation with modern computing platforms (such as GPUs) for processing and compressing massive data. Nevertheless, such an implementation is still way too more expensive compared to how nature realizes intelligence.

There remains a huge room for improvement of the efficiency of machine intelligence so that it can emulate the level of efficiency of natural intelligence, which should be of magnitudes more efficient than the current brute-force implementations. To this end, we need to discover new learning architectures and optimization mechanisms that enable learning data distributions under natural physical conditions and resource constraints, similar to those for intelligent beings in nature, say, without accessing all data at once or updating all model parameters at once (by BP).

The principled framework and approach laid out in this book can guide us to discover such new architectures and mechanisms. These new architectures and mechanisms should enable online continuous learning and can be updated through highly localized and sparse forward or backward optimization. So far, for learning a distribution, the only case for which we know such a solution exists is the simplest case of PCA, with the online PCA method introduced in Chapter 5.

As we have learned from neuroscience, the cortex of our brain consists of tens of thousands of cortical columns. All cortical columns have similar physical structures and functions. They are highly parallel and distributed, though sparsely interconnected. Hence, we believe that in order to develop a more scalable and structured memory system, we need to consider architectures that emulate that of the cortex. Figure 8.2 shows such a hypothesized architecture, a massively distributed and hierarchical system that consists of many largely parallel closed-loop auto-encoding modules. These modules learn to encode different sensory

<sup>1</sup>We say a problem is tractable if it allows an algorithm whose complexity is polynomial in the size of the problem.

modalities or many projections of data from each sensory modality. Our discussion in Section 6.5 of Chapter 6 suggests that such a parallel sensing and learning of a low-dimensional distribution is theoretically possible. Higher-level (lossy) autoencoders can then be learned based on outputs of lower-level ones to develop more sparse and higher-level “abstractions” of the representations learned by the lower levels.

The distributed, hierarchical, and closed-loop system architecture illustrated in Figure 8.2 shares many characteristics of cortex of the brain. Such a system architecture may open up many more possibilities than the current single large-model architecture. It makes exploring much more efficient learning and optimization mechanisms possible, and resulting more structured modular organization of the learned data distribution and knowledge. This would allow us to bring the implementation of machine intelligence to the next level of evolution:

$$\text{incomputable} \Rightarrow \text{computable} \Rightarrow \text{tractable} \Rightarrow \text{scalable} \Rightarrow \text{natural}. \quad (8.2.2)$$

### 8.3 Towards Artificial Intelligence of Human: Beyond the Turing Test?

As we have discussed at the beginning of this book, Chapter 1, intelligence in nature has evolved through multiple phases and manifested in four different forms:

$$\text{phylogenetic} \Rightarrow \text{ontogenetic} \Rightarrow \text{societal} \Rightarrow \text{artificial intelligence}. \quad (8.3.1)$$

All forms of intelligence share the common objective of learning useful knowledge as certain low-dimensional distributions of sensed high-dimensional data about the world. However, they may differ significantly in the specific coding schemes adopted, the information encoded, computational mechanisms for learning and improving, and physical implementations of such mechanisms. Using the concepts and terminologies developed in this book, from the perspective of learning and representing information or knowledge from the distribution of the sensed data, the above four stages of intelligence developed in nature differ in the following three aspects:

1. The codebook that one uses to learn and encode the intended information or knowledge.
2. The information or knowledge that are encoded and represented using the codebook.
3. The optimization mechanisms used to improve the information or knowledge encoded.

More specifically, the following table summarizes their main characteristics in the above three aspects:

	<b>Phylogenetic</b>	<b>Ontogenetic</b>	<b>Societal</b>	<b>Artificial</b>
<b>Codebook</b>	Amino Acids	Neurons	Alphabet & Words	Mathematics/Logic
<b>Information</b>	Genes/DNAs	Memory	Languages/Texts	Scientific Facts
<b>Optimization</b>	Reinforce Learning	Error Feedback	Trial & Error	Hypothesis Testing

As we now know, humans have achieved two quantum leaps in intelligence in history. The first is the development of spoken and written languages about five to ten thousand years ago. That has enabled human to share and pass on learned knowledge for generations, similar to the role of DNAs in nature. The second is the development of mathematics and logic about three thousand years ago, which have become a precise language for modern science. This new language has freed us from summarizing knowledge from observations in empirical forms and allowed us to formalize knowledge as verifiable or falsifiable theories either through mathematical deduction or experimental verification. Through hypothesis formulating, logic deduction, and experimental testing, we are able to proactively discover new knowledge that was previously impossible by passively learning from data distribution.<sup>2</sup>

As we have discussed in the Introduction (Chapter 1), the 1956 “artificial intelligence” (AI) program precisely aimed to study high-level functions such as mathematical abstractions, logical inference, and problem solving that are believed to differentiate humans from animals:

$$\text{low-level (animal) intelligence} \Rightarrow \text{high-level (human) intelligence}. \quad (8.3.2)$$

---

<sup>2</sup>such as causal relationships

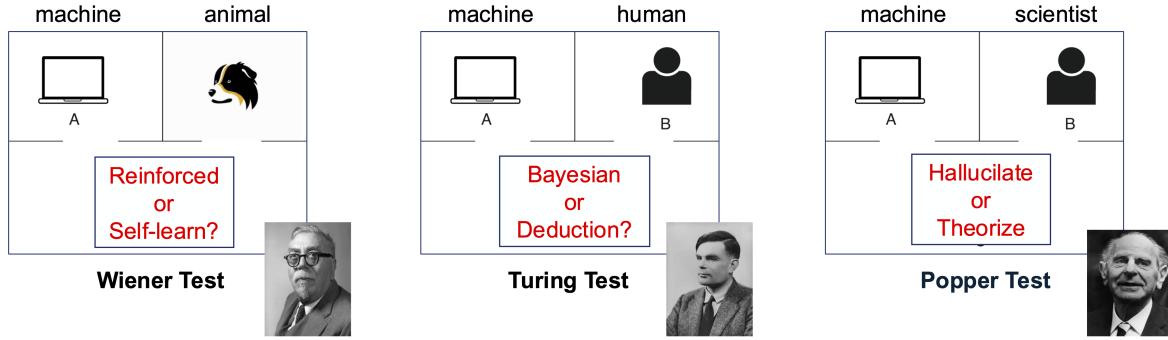


Figure 8.3: Three tests for different levels or types of intelligence capabilities: the Wiener test, the Turing test, and the Popper test.

As we have clarified repeatedly in this book, much of the technological advances in machine intelligence in the past decades, although carried out under the name “AI”, is actually more closely related to the low-level intelligence shared by both animals and humans, which is mainly inductive. So far, there has been no evidence suggesting that these mechanisms alone would suffice to achieve high-level human intelligence that the original AI program truly aims to understand and emulate.

In fact, we know little about how to rigorously verify whether a system is truly capable of certain high-level intelligence, despite the fact that the Turing test has been proposed since 1950 [Tur50].<sup>3</sup> For long such a test was not deemed necessary since the capabilities of machines were far below that of a human (or even animal). However, given recent technological advances, many models and systems have claimed to reach and even surpass human intelligence. Therefore, it is high time to give a scientific and executable definition of the Turing test. That is, how do we systematically and objectively evaluate the level of intelligence for a given model or system? For example, how can we rigorously verify whether an intelligent system has truly grasped a certain abstract concept, such as the notion of natural or real numbers, or it has merely memorized a large number of instances? Note that the state-of-the-art large language models still struggle with simple mathematical questions like: “Whether 3.11 is larger or smaller than 3.9?”<sup>4</sup> How to verify whether a system truly understands the rules of logic and knows how to apply them rigorously or has simply memorized a large number of instances of practicing logic? Furthermore, is such a system even capable of correcting its own knowledge or developing new knowledge such as physical laws, mathematical concepts, or causal relationships? In summary, it is high time that we develop rigorous evaluation methods that can tell a system/model’s seemingly intelligent capability belongs to which of the following:

1. simply having memorized the distribution of some knowledge-carrying data and regenerating them;
2. being able to autonomously and continuously develop new knowledge from new observations;
3. truly having understood certain abstract knowledge and knowing how to apply it correctly;
4. being able to generate new scientific hypotheses or mathematical conjectures and verify them.

Figure 8.3 illustrate that probably there should be at least three different types of tests to evaluate and differentiate different types of intelligence capabilities:

1. *The Norbert Wiener Test:* The evaluate whether a system is capable of improving and developing new knowledge of its own or simply receives information through reinforced or supervised learning;
2. *The Alan Turing Test:* The evaluate whether a system can understand abstract knowledge or simply learns its statistics and uses it for Bayesian inference.

<sup>3</sup>In Turning’s proposal, the evaluator is a human. However, most human evaluators’ scientific training and knowledge can be limited and their conclusions can be subjective.

<sup>4</sup>Note that some models have corrected their answers to questions of this kind via post engineering. Or some models have incorporated additional reasoning mechanisms based on verifying the immediate answers produced and correct them during reasoning. However, we leave it to the reader as an exercise to rigorously test whether any of the state-of-the-art language models truly understand the notion of numbers (natural, rational, real, and complex) and the associated arithmetic.

3. *The Karl Popper Test:* To evaluate whether a system is capable of exploring new knowledge through forming and verifying new theories based on self-consistency.

We believe that, for such evaluation methods, the evaluator should not be a human but a scientifically sound protocol and process, instead.

As we have seen throughout this entire book, *compression* has played a most fundamental role in learning. It is the governing principle and a unified mechanism for identifying an (empirical) data distribution and organizing information encoded therein. To a large extent, it explains most of the practice of “artificial intelligence” in the past decade or so. Here the word “artificial” largely means “man-made.” An outstanding question for future study is whether *compression alone* is sufficient to achieve all higher-level capabilities of intelligence listed above?

*Is compression all there is?*

Whether abstraction, causal inference, logical reasoning, and hypothesis generating and the subsequent deduction are certain extended or extreme forms of compression? Is there some fundamental difference between identifying empirical) data distributions through compression from forming high-level abstract concepts and theories? Philosopher Sir Karl Popper has once suggested:

*“Science may be described as the art of systematic oversimplification.”*

To a large extent, Science, and its associated codebook Mathematics, can be viewed as the most advanced form of intelligence, hence the true “artificial” part of our intelligence. Here the word “artificial” means what is unique to educated and enlightened humans, almost like a form of high art. We believe that uncovering and understanding the underlying mathematical principles and computational mechanisms of such higher-level intelligence will be the final frontier for Science, Mathematics, and Computation altogether!

# Appendix A

# Optimization Methods

*“Since the building of all the universe is perfect and is created by the wisdom creator, nothing arises in the universe in which one cannot see the sense of some maximum or minimum.”*

– L. Euler

In this chapter, we give a brief introduction to some of the most basic but important optimization algorithms used in this book. The purpose is only to help the reader apply these algorithms to problems studied in this book, not to gain a deep understanding about these algorithms. Hence, we will not provide a thorough justification for the algorithms introduced, in terms of performance guarantees.

## A.1 Steepest Descent

Optimization is concerned with the question of how to find where a function, say  $L(\theta)$ , reaches its minimum value. Mathematically, this is stated as a problem:

$$\arg \min_{\theta \in \Theta} \mathcal{L}(\theta), \quad (\text{A.1.1})$$

where  $\Theta$  represents a domain to which the argument  $\theta$  is confined. Often (and unless otherwise mentioned, in this chapter)  $\Theta$  is simply  $\mathbb{R}^n$ . Without loss of generality, we assume that here the function  $\mathcal{L}(\theta)$  is smooth<sup>1</sup>.

The efficiency of finding the (global) minima depends on what information we have about the function  $\mathcal{L}$ . For most optimization problems considered in this book, the dimension of  $\theta$ , say  $n$ , is very large. That makes computing or accessing local information about  $\mathcal{L}$  expensive. In particular, since the gradient  $\nabla \mathcal{L}$  has  $n$  entries, it is often reasonable to compute; however, the Hessian  $\nabla^2 \mathcal{L}$  has  $n^2$  entries which is often wildly impractical to compute (and the same goes for higher-order derivatives). Hence, it is typical to assume that we have the zeroth-order information, i.e., we are able to evaluate  $\mathcal{L}(\theta)$ , and the first-order information, i.e., we are able to evaluate  $\nabla \mathcal{L}(\theta)$ . Optimization theorists may rephrase this as saying we have a “first-order oracle.” All optimization algorithms that we introduce in this section only use a first-order oracle.<sup>2</sup>

### A.1.1 Vanilla Gradient Descent for Smooth Problems

The simplest and most widely used method for optimization is *gradient descent* (GD). It was first introduced by Cauchy in 1847. The idea is very simple: starting from an initial state, we iteratively take small steps such that each step reduces the value of the function  $\mathcal{L}(\theta)$ .

Suppose that the current state is  $\theta$ . We want to take a small step, say of distance  $h$ , in a direction, indicated by a vector  $v$ , to reach a new state  $\theta + hv$  such that the value of the function decreases:

$$\mathcal{L}(\theta + hv) \leq \mathcal{L}(\theta). \quad (\text{A.1.2})$$

---

<sup>1</sup>In case the function  $\mathcal{L}$  is not smooth, we replace its gradient with a so-called *subgradient*.

<sup>2</sup>We refer the readers to the book by [WM22] for a more systematic introduction to optimization algorithms in a high-dimensional space, including algorithms assuming higher-order oracles.

To find such a direction  $\mathbf{v}$ , we can approximate  $\mathcal{L}(\theta + h\mathbf{v})$  through a Taylor expansion around  $h = 0$ :

$$\mathcal{L}(\theta + h\mathbf{v}) = \mathcal{L}(\theta) + h\langle \nabla \mathcal{L}(\theta), \mathbf{v} \rangle + o(h), \quad (\text{A.1.3})$$

where the inner product here (and in this chapter) will be the  $\ell^2$  inner product, i.e.,  $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$ . To find the direction of *steepest descent*, we attempt to minimize this Taylor expansion among unit vectors  $\mathbf{v}$ . If  $\nabla \mathcal{L}(\theta) = \mathbf{0}$ , then the second term above is 0 regardless of the value of  $\mathbf{v}$ , so we cannot attempt to make progress, i.e., the algorithm has converged. On the other hand, if  $\nabla \mathcal{L}(\theta) \neq \mathbf{0}$  then it holds

$$\arg \min_{\substack{\mathbf{v} \in \mathbb{R}^d \\ \|\mathbf{v}\|_2=1}} [\mathcal{L}(\theta) + h\langle \nabla \mathcal{L}(\theta), \mathbf{v} \rangle] = \arg \min_{\substack{\mathbf{v} \in \mathbb{R}^d \\ \|\mathbf{v}\|_2=1}} \langle \nabla \mathcal{L}(\theta), \mathbf{v} \rangle = -\frac{\nabla \mathcal{L}(\theta)}{\|\nabla \mathcal{L}(\theta)\|_2}, \quad (\text{A.1.4})$$

In words, this means that the value of  $\mathcal{L}(\theta + h\mathbf{v})$  decreases the fastest along the direction  $\mathbf{v} = -\nabla \mathcal{L}(\theta)/\|\nabla \mathcal{L}(\theta)\|_2$ , for small enough  $h$ . This leads to the gradient descent method: From the current state  $\theta_k$  ( $k = 0, 1, \dots$ ), we take a step of size  $h$  in the direction of the negative gradient to reach the next iterate,

$$\theta_{k+1} = \theta_k - h\nabla \mathcal{L}(\theta_k). \quad (\text{A.1.5})$$

The step size  $h$  is also called the *learning rate* in machine learning contexts.

### Step-Size Selection

The remaining question is what the step size  $h$  should be? If we choose  $h$  to be too small, the value of the function may decrease very slowly, as shown by the plot in the middle in Figure A.1. If  $h$  is too large, the value might not even decrease at all, as shown by the plot on the right in Figure A.1.

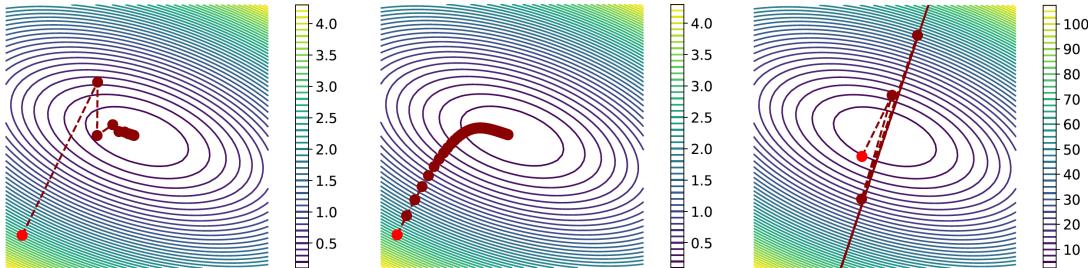


Figure A.1: The effect of step size  $h$  on the convergence of the gradient descent method.

So the step size  $h$  should be chosen based on the landscape of the function  $\mathcal{L}(\theta_k)$ . Ideally, to choose the best step size  $h$ , we can solve the following optimization problem over a single variable  $h$ :

$$h = \arg \min_{h \geq 0} \mathcal{L}(\theta_k - h\nabla \mathcal{L}(\theta_k)). \quad (\text{A.1.6})$$

This method of choosing the step size is called *line search*. However, when the function  $L(\theta_k)$  is complicated, which is usually the case for training a deep neural network, this one-dimensional optimization is very difficult to solve at each iteration of gradient descent.

Then how should we choose a proper step size  $h$ ? One common and classical approach is to try to obtain a good approximation of the local landscape around the current state  $\theta$  based on some knowledge about the overall landscape of the function  $\mathcal{L}(\theta)$ .

Common conditions for the landscape of  $\mathcal{L}(\theta)$  include:

- $\alpha$ -Strong Convexity. Recall that  $\mathcal{L}$  is  $\alpha$ -strongly convex if its graph lies above a global quadratic lower bound of slope  $\alpha$ , i.e.,

$$\mathcal{L}(\theta) \geq l_{\theta_0, \alpha}(\theta) \doteq \mathcal{L}(\theta_0) + \langle \nabla \mathcal{L}(\theta_0), \theta - \theta_0 \rangle + \frac{\alpha}{2} \|\theta - \theta_0\|_2^2 \quad (\text{A.1.7})$$

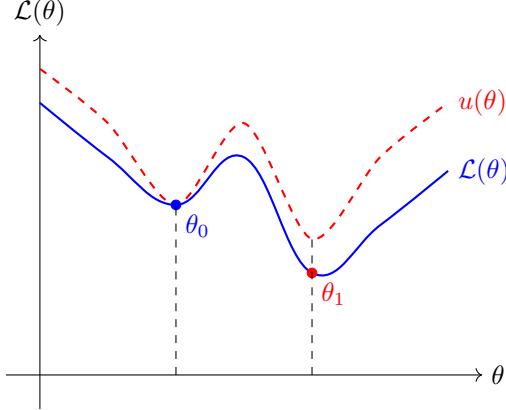


Figure A.2: **Majorization-minimization scheme and intuition.** A function  $\mathcal{L}: \Theta \rightarrow \mathbb{R}$  has a global upper bound  $u: \Theta \rightarrow \mathbb{R}$  which meets  $L$  at at least one point  $\theta_0$ . Then, finding the  $\theta_1$  which minimizes  $u$  will improve the value of  $\mathcal{L}$  from  $\mathcal{L}(\theta_0)$ . Note that similar results can be shown about local bounds.

for any “base point”  $\theta_0$ . We say that  $\mathcal{L}$  is *convex* if it is 0-strongly convex, i.e., its graph lies above its tangents. It is easy to show (proof as exercise) that strongly convex functions have unique global minima. Another important fact (proof as exercise) is that  $\alpha$ -strongly convex twice-differentiable functions  $\mathcal{L}$  have (symmetric) Hessians  $\nabla^2 \mathcal{L}$  whose minimum eigenvalue is  $\geq \alpha$ . For  $\alpha > 0$  this implies the Hessian is symmetric positive definite, and for  $\alpha = 0$  (i.e.,  $\mathcal{L}$  is convex) this implies that the Hessian is symmetric positive semidefinite.

- $\beta$ -Lipschitz Gradient (also called  $\beta$ -Smoothness). Recall that  $\mathcal{L}$  has  $\beta$ -Lipschitz gradient if  $\nabla \mathcal{L}$  exists and is  $\beta$ -Lipschitz, i.e.,

$$\|\nabla \mathcal{L}(\theta) - \nabla \mathcal{L}(\theta_0)\|_2 \leq \beta \|\theta - \theta_0\|_2. \quad (\text{A.1.8})$$

for any “base point”  $\theta_0$ . It is easy to show (proof as exercise) that this is equivalent to  $\mathcal{L}$  having a global quadratic upper bound of slope  $\beta$ , i.e.,

$$\mathcal{L}(\theta) \leq u_{\theta_0, \beta}(\theta) \doteq \mathcal{L}(\theta_0) + \langle \nabla \mathcal{L}(\theta_0), \theta - \theta_0 \rangle + \frac{\beta}{2} \|\theta - \theta_0\|_2^2. \quad (\text{A.1.9})$$

for any “base point”  $\theta_0$ . Another important fact (proof as exercise) is that convex  $\beta$ -Lipschitz gradient twice-differentiable functions have (symmetric) Hessians  $\nabla^2 \mathcal{L}$  whose largest eigenvalue is  $\leq \beta$ .

First, let us suppose that  $\mathcal{L}$  has  $\beta$ -Lipschitz gradient (but is not necessarily even convex). We will use this occasion to introduce a common optimization theme: *to minimize  $\mathcal{L}$ , we can minimize an upper bound on  $\mathcal{L}$* , which is justified by the following lemma visualized in Figure A.2.

**Lemma A.1** (Majorization-Minimization). *Suppose that  $u: \Theta \rightarrow \mathbb{R}$  is a global upper bound on  $\mathcal{L}$ , namely  $\mathcal{L}(\theta) \leq u(\theta)$  for all  $\theta \in \Theta$ . Suppose that they meet with equality at  $\theta_0$ , i.e.,  $\mathcal{L}(\theta_0) = u(\theta_0)$ . Then*

$$\theta_1 \in \arg \min_{\theta \in \Theta} u(\theta) \implies \mathcal{L}(\theta_1) \leq u(\theta_1) \leq u(\theta_0) = \mathcal{L}(\theta_0). \quad (\text{A.1.10})$$

We will use this lemma to show that we can use the Lipschitz gradient property to ensure that each gradient step cannot worsen the value of  $\mathcal{L}$ . Indeed, at every base point  $\theta_0$ , we have that  $u_{\theta_0, \beta}$  is a global upper bound on  $\mathcal{L}$ , and  $u_{\theta_0, \beta}(\theta_0) = \mathcal{L}(\theta_0)$ . Hence by Lemma A.1

$$\text{if } \theta \text{ minimizes } u_{\theta_0, \beta} \text{ then } \mathcal{L}(\theta) \leq u_{\theta_0, \beta}(\theta) \leq u_{\theta_0, \beta}(\theta_0) = \mathcal{L}(\theta_0). \quad (\text{A.1.11})$$

This motivates us, when finding an update to obtain  $\theta_{k+1}$  from  $\theta_k$ , we can instead minimize the upper bound  $u_{\theta_k, \beta}$  over  $\theta$  and set that to be  $\theta_{k+1}$ . By minimizing  $u_{\theta_k, \beta}$  (proof as exercise) we get

$$\theta_{k+1} = \theta_k - \frac{1}{\beta} \nabla \mathcal{L}(\theta_k) \implies \mathcal{L}(\theta_{k+1}) \leq \mathcal{L}(\theta_k). \quad (\text{A.1.12})$$

This implies that a step size  $h = 1/\beta$  is a usable learning rate, but it does not provide a convergence rate or certify that  $L(\theta_k)$  actually converges to  $\min_{\theta} \mathcal{L}(\theta)$ . This requires a little more rigor, which we now pursue.

Now, let us suppose that  $\mathcal{L}$  is  $\alpha$ -strongly convex, has  $\beta$ -Lipschitz gradient, and has global optimum  $\theta^*$ . We will show that  $\theta_k$  will converge directly to the unique global optimum  $\theta^*$ , which is a very strong form of convergence. In particular, we will bound  $\|\theta^* - \theta_k\|_2$  using both strong convexity and Lipschitzness of the gradient of  $\mathcal{L}$ , i.e., taking a look at the neighborhood around  $\theta_k$ :<sup>3</sup>

$$\|\theta^* - \theta_{k+1}\|_2^2 \leq \|\theta^* - \theta_k + h\nabla\mathcal{L}(\theta_k)\|_2^2 \quad (\text{A.1.13})$$

$$= \|\theta^* - \theta_k\|_2^2 + 2h\langle \nabla\mathcal{L}(\theta_k), \theta^* - \theta_k \rangle + h^2\|\nabla\mathcal{L}(\theta_k)\|_2^2 \quad (\text{A.1.14})$$

$$\leq \|\theta^* - \theta_k\|_2^2 + 2h \left( \mathcal{L}(\theta^*) - \mathcal{L}(\theta_k) - \frac{\alpha}{2}\|\theta^* - \theta_k\|_2^2 \right) + h^2\|\nabla\mathcal{L}(\theta_k)\|_2^2 \quad (\alpha\text{-SC}) \quad (\text{A.1.15})$$

$$= (1 - \alpha h)\|\theta^* - \theta_k\|_2^2 + 2h(\mathcal{L}(\theta^*) - \mathcal{L}(\theta_k)) + h^2\|\nabla\mathcal{L}(\theta_k)\|_2^2 \quad (\text{A.1.16})$$

$$\leq (1 - \alpha h)\|\theta^* - \theta_k\|_2^2 + 2h(\mathcal{L}(\theta^k) - \mathcal{L}(\theta^*)) + 2h^2\beta(\mathcal{L}(\theta_k) - \mathcal{L}(\theta^*)) \quad (\beta\text{-LG}) \quad (\text{A.1.17})$$

$$= (1 - \alpha h)\|\theta^* - \theta_k\|_2^2 - 2h(1 - \beta h)(\mathcal{L}(\theta_k) - \mathcal{L}(\theta^*)). \quad (\text{A.1.18})$$

In order to ensure that the gradient descent iteration makes progress we must pick the step size so that  $1 - \beta h \geq 0$ , i.e.,  $h \leq 1/\beta$ . If such a setting occurs, then

$$\|\theta^* - \theta_{k+1}\|_2^2 \leq (1 - \alpha h)\|\theta^* - \theta_k\|_2^2 \leq (1 - \alpha h)^2\|\theta^* - \theta_{k-1}\|_2^2 \leq \dots \quad (\text{A.1.19})$$

$$\leq (1 - \alpha h)^{k+1}\|\theta^* - \theta_0\|_2^2. \quad (\text{A.1.20})$$

In order to minimize the right-hand side, we can set  $h = 1/\beta$ , which obtains

$$\|\theta^* - \theta_{k+1}\|_2^2 \leq (1 - \alpha/\beta)^{k+1}\|\theta^* - \theta_0\|_2^2, \quad (\text{A.1.21})$$

showing convergence to global optimum with exponentially decaying error. Notice that here we used a convergence rate to obtain a favorable *step size* of  $h = 1/\beta$ . This motif will re-occur in this section.

We end this section with a caveat: learning a global optimum is (usually) impractically hard. Under certain conditions, we can ensure that the gradient descent iterates converge to a *local optimum*. Also, under more relaxed conditions, we can ensure *local* convergence, i.e., that the iterates converge to a (global or local) optimum if the sequence is initialized close enough to the optimum.

### A.1.2 Preconditioned Gradient Descent for Badly-Conditioned Problems

#### Newton's Method

There are some smooth problems and strongly convex problems on which gradient descent nonetheless does quite poorly. For example, let  $\lambda \geq 0$  and let  $\mathcal{L}_\lambda: \mathbb{R}^2 \rightarrow \mathbb{R}$  of the form

$$\mathcal{L}_\lambda(\theta) = \mathcal{L}_\lambda\left(\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}\right) \doteq \frac{1}{2} \{(1 + \lambda)\theta_1^2 + \theta_2^2\} = \frac{1}{2} \theta^\top \begin{bmatrix} 1 + \lambda & 0 \\ 0 & 1 \end{bmatrix} \theta. \quad (\text{A.1.22})$$

This problem is 1-strongly convex and has  $(1 + \lambda)$ -Lipschitz gradient. The convergence rate is then geometric with rate  $1 - 1/(1 + \lambda)$ . For large  $\lambda$ , this is still not very fast. In this section, we will introduce a class of optimization problems which can successfully optimize such badly-conditioned functions.

The key lies in the objective's *curvature*, which is given by the Hessian. Suppose that (as a counterfactual) we had a *second-order* oracle which would allow us to compute  $\mathcal{L}(\theta)$ ,  $\nabla\mathcal{L}(\theta)$ , and  $\nabla^2\mathcal{L}(\theta)$ . Then, instead of picking a descent direction  $\mathbf{v}$  to optimize the first-order Taylor expansion around  $\theta$ , we could optimize the second-order Taylor expansion instead. Intuitively this would allow us to incorporate curvature information into the update.

Let us carry out this computation. The second-order Taylor expansion of  $\mathcal{L}(\theta + h\mathbf{v})$  around  $h = 0$  is

$$\mathcal{L}(\theta + h\mathbf{v}) = \mathcal{L}(\theta) + h\langle \nabla\mathcal{L}(\theta), \mathbf{v} \rangle + \frac{1}{2}h^2\langle [\nabla^2\mathcal{L}(\theta)]\mathbf{v}, \mathbf{v} \rangle + o(h^2). \quad (\text{A.1.23})$$

---

<sup>3</sup>In this proof the  $\beta$ -Lipschitz Gradient invocation step is a little non-trivial. We also leave this step as an exercise, with the hint to plug in  $\theta = \theta_0 - h\nabla\mathcal{L}(\theta_0)$  into the Lipschitz gradient identity.

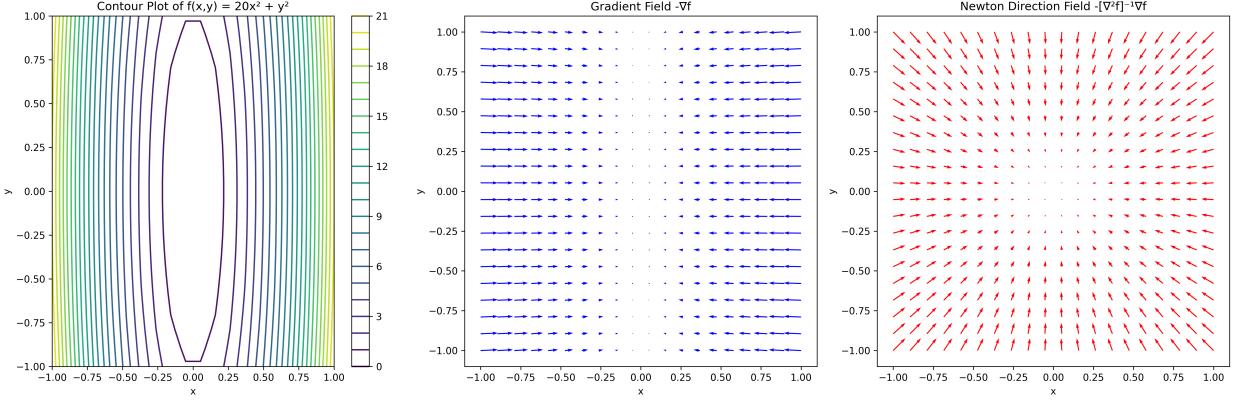


Figure A.3: **The negative gradient  $-\nabla \mathcal{L}_\lambda$  and pre-conditioned (Newton's method step) vector field  $-[\nabla^2 \mathcal{L}_\lambda]^{-1}[\nabla \mathcal{L}_\lambda]$**  where  $\lambda = 19$ . There is a section of the space where following the negative gradient vector field makes very little progress towards finding the minimum, but in all cases following the Newton's method vector field achieves equal speed of progress towards the optimum since the gradient is whitened. Since the Hessian here is diagonal, adaptive learning rate algorithms (e.g. Adam, as will be discussed later in the section) can make similar progress as Newton's method, but a non-axis-aligned Hessian may even prevent Adam from succeeding quickly.

Then we can compute the descent direction:

$$\arg \min_{\substack{\mathbf{v} \in \mathbb{R}^n \\ \|\mathbf{v}\|_2=1}} \left[ \mathcal{L}(\theta) + h \langle \nabla \mathcal{L}(\theta), \mathbf{v} \rangle + \frac{1}{2} h^2 \langle [\nabla^2 \mathcal{L}(\theta)] \mathbf{v}, \mathbf{v} \rangle \right] = \arg \min_{\substack{\mathbf{v} \in \mathbb{R}^n \\ \|\mathbf{v}\|_2=1}} \left[ \langle \nabla \mathcal{L}(\theta), \mathbf{v} \rangle + \frac{1}{2} h \langle [\nabla^2 \mathcal{L}(\theta)] \mathbf{v}, \mathbf{v} \rangle \right]. \quad (\text{A.1.24})$$

This optimization problem is a little difficult to solve because of the constraint  $\|\mathbf{v}\|_2 = 1$ . But in practice we never normalize the descent direction  $\mathbf{v}$  and use the step size  $h$  to control the size of the update. So let us just solve the above problem over all vectors  $\mathbf{v} \in \mathbb{R}^n$ :<sup>4</sup>

$$\arg \min_{\mathbf{v} \in \mathbb{R}^n} \left[ \langle \nabla \mathcal{L}(\theta), \mathbf{v} \rangle + \frac{1}{2} h \langle [\nabla^2 \mathcal{L}(\theta)] \mathbf{v}, \mathbf{v} \rangle \right] = -\frac{1}{h} [\nabla^2 \mathcal{L}(\theta)]^{-1} [\nabla \mathcal{L}(\theta)]. \quad (\text{A.1.25})$$

We can thus use the steepest descent iteration

$$\theta_{k+1} = \theta_k - [\nabla^2 \mathcal{L}(\theta_k)]^{-1} [\nabla \mathcal{L}(\theta_k)], \quad (\text{A.1.26})$$

(this is the celebrated *Newton's method*), or

$$\theta_{k+1} = \theta_k - h [\nabla^2 \mathcal{L}(\theta_k)]^{-1} [\nabla \mathcal{L}(\theta_k)], \quad (\text{A.1.27})$$

(which is called *underdamped Newton's method*). Since the second-order quadratic  $\mathcal{L}_\lambda$  is equal to its second-order Taylor expansion, if we run Newton's method for *one step*, we will achieve the global minimum in *one step* no matter how large  $\lambda$  is. Figure A.3 gives some intuition about poorly conditioned functions and the gradient steps versus Newton's steps.

## PGD

In practice, we do *not* have a second-order oracle which allows us to compute  $\nabla^2 \mathcal{L}(\theta)$ . Instead, we can attempt to *learn an approximation to it* alongside the parameter update  $\theta_{k+1}$  from  $\theta_k$ .

How do we learn an approximation to it? We shall find some equations which the Hessian's inverse satisfies and then try to update our approximation so that it satisfies the equations. Namely, taking the Taylor series of  $\nabla \mathcal{L}(\theta + \delta_\theta)$  around point  $\theta$ , we obtain

$$\underbrace{\nabla \mathcal{L}(\theta + \delta_\theta) - \nabla \mathcal{L}(\theta)}_{\doteq \delta_g} = [\nabla^2 \mathcal{L}(\theta)] \delta_\theta + o(\|\delta_\theta\|_2). \quad (\text{A.1.28})$$

<sup>4</sup>If  $\nabla^2 \mathcal{L}(\theta)$  is not invertible, then we can replace  $[\nabla^2 \mathcal{L}(\theta)]^{-1}$  with the Moore-Penrose pseudoinverse of  $\nabla^2 \mathcal{L}(\theta)$ .

In this case we have

$$\delta_g \approx [\nabla^2 \mathcal{L}(\theta)]\delta_\theta \implies \delta_\theta \approx [\nabla^2 \mathcal{L}(\theta)]^{-1}\delta_g \quad (\text{A.1.29})$$

We can now try to learn a symmetric positive semidefinite pre-conditioner  $P \in \mathbb{R}^{n \times n}$  such that

$$\delta_\theta \approx P\delta_g, \quad (\text{A.1.30})$$

updating it at each iteration along with  $\theta_k$ . Namely, we have the *PSGD* iteration

$$P_k = \text{PreconditionerUpdate}(P_{k-1}; \theta_k, \nabla \mathcal{L}(\theta_k)) \quad (\text{A.1.31})$$

$$\theta_{k+1} = \theta_k - hP_k \nabla \mathcal{L}(\theta_k). \quad (\text{A.1.32})$$

This update has two problems: how can we even use  $P$  (since we already said we cannot store an  $n \times n$  matrix) and how can we *update*  $P$  at each iteration? The answers are very related; we can never materialize  $P$  in computer memory, but we can represent it using a low-rank factorization (or comparable methods such as *Kronecker factorization* which is particularly suited to the form of deep neural networks). Then the preconditioner update step is designed to exploit the structure of the preconditioner representation.

We end this subsection with a caveat: in deep learning, for example,  $\mathcal{L}$  is not a convex function and so Newton's method (and approximations to it) do not make sense. In this case we look at the geometric intuition of Newton's method on convex functions, say from Figure A.3: the inverse Hessian *whitens* the gradients. Thus instead of a Hessian-approximating preconditioner, we can adjust the above procedures to learn a more general whitening transformation for the gradient. This is the idea behind the original proposal of PSGD [Li17], which contains more information about how to store and update the preconditioner, and more modern optimizers like Muon [LSY+25].

### A.1.3 Proximal Gradient Descent for Non-Smooth Problems

Even in very toy problems, however, such as LASSO or dictionary learning, the problem is not strongly convex but rather just convex, and the objective is no longer just smooth but rather the sum of a smooth function and a non-smooth regularizer (such as the  $\ell^1$  norm). Such problems are solved by *proximal optimization algorithms*, which generalize steepest descent to non-smooth objectives.

Formally, let us say that

$$\mathcal{L}(\theta) \doteq \mathcal{S}(\theta) + \mathcal{R}(\theta) \quad (\text{A.1.33})$$

where  $\mathcal{S}$  is smooth, say with  $\beta$ -Lipschitz gradient, and  $\mathcal{R}$  is non-smooth (i.e., rough). The proximal gradient algorithm generalizes the steepest descent algorithm, by using the majorization-minimization framework (i.e., Lemma A.1) with a different global upper bound. Namely, we construct such an upper bound by asking: what if we take the Lipschitz gradient upper bound of  $\mathcal{S}$  but *leave  $\mathcal{R}$  alone*? Namely, we have

$$\mathcal{L}(\theta_1) = \mathcal{S}(\theta_1) + \mathcal{R}(\theta_1) \leq u_{\theta_0, \beta}(\theta_1) \doteq \mathcal{S}(\theta_0) + \langle \nabla \mathcal{S}(\theta_0), \theta_1 - \theta_0 \rangle + \frac{\beta}{2} \|\theta_1 - \theta_0\|_2^2 + \mathcal{R}(\theta_1). \quad (\text{A.1.34})$$

Note that (proof as exercise)

$$\arg \min_{\theta_1} u_{\theta_0, \beta}(\theta_1) = \arg \min_{\theta_1} \left[ \frac{\beta}{2} \left\| \theta_1 - \left( \theta_0 - \frac{1}{\beta} \nabla \mathcal{S}(\theta_0) \right) \right\|_2^2 + \mathcal{R}(\theta_1) \right]. \quad (\text{A.1.35})$$

Now if we try to minimize the upper bound  $u_{\theta_0, \beta}$ , we are picking a  $\theta_1$  that:

- is close to the gradient update  $\theta_0 - \frac{1}{\beta} \nabla \mathcal{S}(\theta_0)$ ;
- has a small value of the regularizer  $\mathcal{R}(\theta_1)$

and trades off these properties according to the smoothness parameter  $\beta$ . Accordingly, let us define the proximal operator

$$\text{prox}_{h, \mathcal{R}}(\theta) \doteq \arg \min_{\theta_1} \left[ \frac{1}{2h} \|\theta_1 - \theta\|_2^2 + \mathcal{R}(\theta_1) \right]. \quad (\text{A.1.36})$$

Then, we can define the *proximal gradient descent* iteration which, at each iteration, minimizes the upper bound  $u_{\theta_k, h^{-1}}$ , i.e.,

$$\theta_{k+1} = \text{prox}_{h, \mathcal{R}}(\theta_k - h \nabla \mathcal{S}(\theta_k)). \quad (\text{A.1.37})$$

Convergence proofs are possible when  $h \leq 1/\beta$ , but we do not give any in this section.

One remaining question is: how can we compute the proximal operator? At first glance, it seems like we have traded one intractable minimization problem for another. Since we have not made any assumption on  $\mathcal{R}$  so far, the framework works even when  $\mathcal{R}$  is a very complex function (such as a neural network loss), which would require us to solve a neural network training problem in order to compute a single proximal operator. However, in practice, for simple regularizers  $\mathcal{R}$  such as those we use in this manuscript, there exist proximal operators which are easy to compute or even in closed-form. We give a few below (the proofs are an exercise).

*Example A.1.* Let  $\Gamma \subseteq \Theta$  be a set, and let  $\chi_\Gamma$  be the characteristic function on  $\Gamma$ , i.e.,

$$\chi_\Gamma(\theta) \doteq \begin{cases} 0, & \text{if } \theta \in \Gamma \\ +\infty, & \text{if } \theta \notin \Gamma. \end{cases} \quad (\text{A.1.38})$$

Then the proximal operator of  $\chi_\Gamma$  is a projection, i.e.,

$$\text{prox}_{h, \chi_\Gamma}(\theta) = \arg \min_{\theta_1 \in \Gamma} \frac{1}{2} \|\theta_1 - \theta\|_2^2 = \arg \min_{\theta_1 \in \Gamma} \|\theta_1 - \theta\|_2. \quad (\text{A.1.39})$$

■

*Example A.2.* The  $\ell^1$  norm has a proximal operator which performs soft thresholding:

$$S_h(\theta) \doteq \text{prox}_{h, \lambda \|\cdot\|_1}(\theta) = \arg \min_{\theta_1} \left[ \frac{1}{2h} \|\theta_1 - \theta\|_2^2 + \lambda \|\theta_1\|_1 \right] \quad (\text{A.1.40})$$

then  $S_h(\theta)$  is defined by

$$S_h(\theta)_i = \begin{cases} \theta_i - h\lambda, & \text{if } \theta_i \geq h\lambda \\ 0, & \text{if } \theta_i \in [-h\lambda, h\lambda] \\ \theta_i + h\lambda, & \text{if } \theta_i \leq -h\lambda \end{cases} = \begin{cases} \max\{|\theta_i| - h\lambda, 0\} \text{ sign}(\theta_i), & \text{if } |\theta_i| \geq h\lambda \\ 0, & \text{if } |\theta_i| < h\lambda. \end{cases} \quad (\text{A.1.41})$$

The proximal gradient operation with the smooth part of the objective being least-squares and the non-smooth part being the  $\ell^1$  norm (hence using this soft thresholding proximal operator) is called the Iterative Shrinkage-Thresholding Algorithm (ISTA). ■

*Example A.3.* In Chapter 4 we use a proximal operator corresponding to the  $\ell^1$  norm plus the characteristic function for the positive orthant  $\mathbb{R}_+^n \doteq \{\boldsymbol{x} \in \mathbb{R}^n : x_i \geq 0 \forall i\}$ , namely

$$T_h(\theta) \doteq \text{prox}_{h, \lambda \|\cdot\|_1 + \chi_{\mathbb{R}_+^n}}(\theta) = \arg \min_{\theta_1 \in \mathbb{R}_+^n} \left[ \frac{1}{2h} \|\theta_1 - \theta\|_2^2 + \lambda \|\theta_1\|_1 \right], \quad (\text{A.1.42})$$

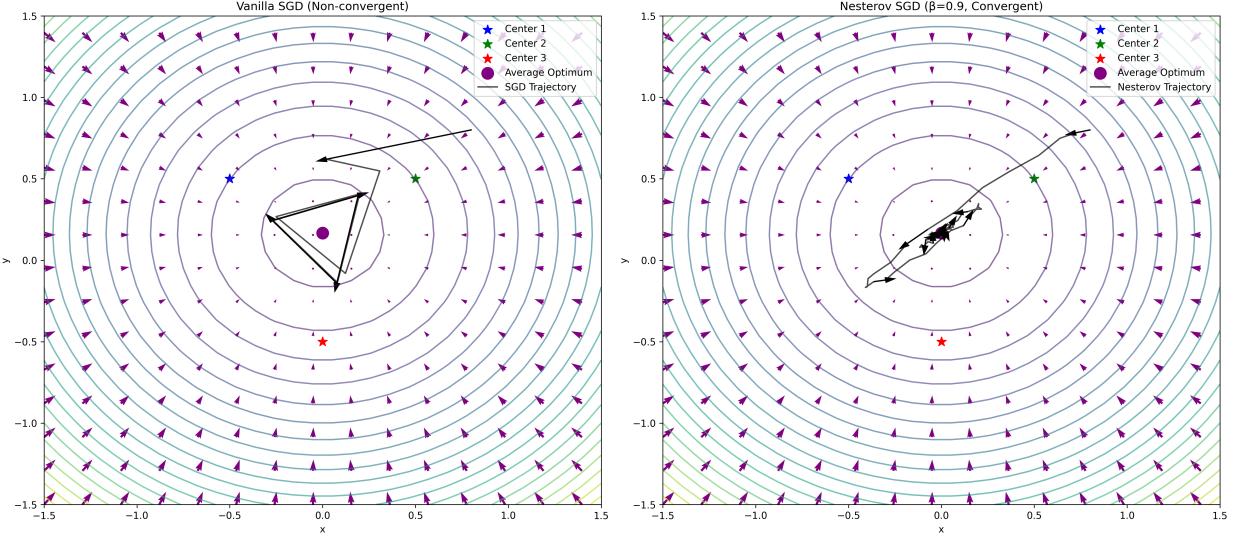
then  $T_h$  is defined as

$$T_h(\theta)_i \doteq \max\{\theta_i - h\lambda, 0\}. \quad (\text{A.1.43})$$

This proximal operator yields the non-negative ISTA that is invoked in Chapter 4 and beyond. ■

#### A.1.4 Stochastic Gradient Descent for Large-Scale Problems

In deep learning, the objective function  $\mathcal{L}$  usually cannot be computed exactly, and instead at each optimization step it is *estimated* using finite samples (say, using a mini-batch). A common way to model this situation is to define a *stochastic loss function*  $\mathcal{L}_\omega(\theta)$  where  $\omega$  is some “source of randomness”. For example,  $\omega$  could contain the indices of the samples in a batch over which to compute the loss. Then, we would like to minimize  $\mathcal{L}(\theta) \doteq \mathbb{E}_\omega[\mathcal{L}_\omega(\theta)]$  over  $\theta$ , given access to a *stochastic first-order oracle*: given  $\theta$ , we can sample  $\omega$  and compute  $\mathcal{L}_\omega(\theta)$  and  $\nabla_\theta \mathcal{L}_\omega(\theta)$ . This minimization problem is called a *stochastic optimization problem*.



**Figure A.4: Stochastic gradient descent may not converge, even for very benign objectives, but Nesterov gradient converges.** For even simple quadratic objectives, stochastic gradient descent iterates may pinball around the global optimum, whereas Nesterov gradients align to point to the optimal value.

The basic first-order stochastic algorithm is *stochastic gradient descent*: at each iteration  $k$  we sample  $\omega_k$ , define  $\mathcal{L}_k \doteq \mathcal{L}_{\omega_k}$ , and perform a gradient step on  $\mathcal{L}_k$ , i.e.,

$$\theta_{k+1} = \theta_k - h \nabla \mathcal{L}_k(\theta_k). \quad (\text{A.1.44})$$

However, even for very simple problems we cannot expect the same type of convergence as we obtained in gradient descent. For example, suppose that there are  $m$  possible values for  $\omega \in \{1, \dots, m\}$  which it takes with equal probability, and there are  $m$  possible targets  $\xi_1, \dots, \xi_m$ , such that the loss function  $\mathcal{L}_\omega$  is

$$\mathcal{L}_\omega(\theta) \doteq \frac{1}{2} \|\theta - \xi_\omega\|_2^2. \quad (\text{A.1.45})$$

Then  $\arg \min_\theta \mathbb{E}[\mathcal{L}_\omega(\theta)] = \frac{1}{m} \sum_{i=1}^m \xi_i$ , but stochastic gradient descent can “pinball” around the global optimum value, and not converge, as visualized in Figure A.4.

In order to fix this, we can either average the parameters  $\theta_k$  or average the gradients  $\nabla \mathcal{L}_k(\theta_k)$  over time. If we average the parameters  $\theta_k$ , then (using Figure A.4 as a mental model) the issue of pinballing is straightforwardly not possible, since the average iterate will grow closer to the center. As such, most theoretical convergence proofs consider the convergence of the average iterate  $\frac{1}{k} \sum_{i=0}^k \theta_i$  to the global minimum. If we average the gradients, we will eventually learn an average gradient  $\frac{1}{k} \sum_{i=0}^k \nabla \mathcal{L}_k(\theta_k)$  which does not change much at each iteration and therefore does not pinball.

In practice, instead of using an arithmetic average, we take an *exponentially moving average* (EMA) of the parameters (this is called *Polyak momentum*) or of the gradients (this is called *Nesterov momentum*). Nesterov momentum is more popular and we will study it here.

A stochastic gradient descent iteration with Nesterov momentum is as follows:

$$\mathbf{g}_k = \beta \mathbf{g}_{k-1} + (1 - \beta) \nabla \mathcal{L}_k(\theta_k) \quad (\text{A.1.46})$$

$$\theta_{k+1} = \theta_k - h \mathbf{g}_k. \quad (\text{A.1.47})$$

We do not go through a convergence proof (see Chapter 7 of [GG23] for an example). However, Nesterov momentum handles our toy case in Figure A.4 easily (see the right-hand figure): it stops pinballing and eventually converges to the global optimum.

We end with a caveat: one can show that Polyak momentum and Nesterov momentum are equivalent, for certain choices of parameter settings. Then it is also possible to show that a decaying learning rate

schedule (i.e., the learning rate  $h$  depends on the iteration  $k$ , and its limit is  $h_k \rightarrow 0$  as  $k \rightarrow \infty$ ) with plain SGD (or PSGD) can mimic the effect of momentum. Namely, [DCM+23] shows that if the SGD algorithm lasts  $K$  iterations, the gradient norms are bounded  $\|\nabla \mathcal{L}(\theta_k)\|_2 \leq G$ , and we define  $D \doteq \|\theta_0 - \theta^*\|_2$ , then plain SGD iterates  $\theta_k$  satisfy the rate  $\mathbb{E}[\mathcal{L}(\theta_k) - \mathcal{L}(\theta^*)] \leq DG/\sqrt{K}$  — but only so long as the learning rate  $h_k = (D/[G\sqrt{K}])(1 - k/K)$  decays *linearly* with time. This matches learning rate schedules used in practice. Indeed, surprisingly, such a theory of convex optimization can predict many empirical phenomena in deep networks [SHT+25], despite deep learning optimization being highly non-convex and non-smooth in the worst case. It is so far unclear why this is the case.

### A.1.5 Putting Everything Together: Adam

The gradient descent scheme proposes an iteration of the form

$$\theta_{k+1} = \theta_k + h\mathbf{v}_k, \quad (\text{A.1.48})$$

where (recall)  $\mathbf{v}_k$  is chosen to be (proportional to) the steepest descent vector in the Euclidean norm:

$$\mathbf{v}_k = -\frac{\nabla \mathcal{L}(\theta_k)}{\|\nabla \mathcal{L}(\theta_k)\|_2} \in \arg \min_{\substack{\mathbf{v} \in \mathbb{R}^n \\ \|\mathbf{v}\|_2=1}} \langle \nabla \mathcal{L}(\theta_k), \mathbf{v} \rangle. \quad (\text{A.1.49})$$

However, in the context of deep learning optimization, there is absolutely nothing which implies that we have to use the Euclidean norm; indeed the “natural geometry” of the space of parameters is not well-respected by the Euclidean norm, since small changes in the parameter space can lead to very large differences in the output space, for a particular fixed input to the network. If we were instead to use a generic norm  $\|\cdot\|$  on the parameter space  $\mathbb{R}^n$ , we would get some other quantity corresponding to the so-called *dual norm*:

$$\mathbf{v}_k \in \arg \min_{\substack{\mathbf{v} \in \mathbb{R}^n \\ \|\mathbf{v}\|=1}} \langle \nabla \mathcal{L}(\theta_k), \mathbf{v} \rangle. \quad (\text{A.1.50})$$

For instance, if we were to use the  $\ell^\infty$ -norm, it is possible to show that

$$\mathbf{v}_k = -\text{sign}(\nabla \mathcal{L}(\theta_k)) \in \arg \min_{\substack{\mathbf{v} \in \mathbb{R}^n \\ \|\mathbf{v}\|_\infty=1}} \langle \nabla \mathcal{L}(\theta_k), \mathbf{v} \rangle, \quad (\text{A.1.51})$$

where  $\text{sign}(\mathbf{x})_i = \text{sign}(x_i) \in \{-1, 0, 1\}$ . Thus if we were so-inclined, we could use the so-called *sign-gradient descent*:

$$\theta_{k+1} = \theta_k - h \text{sign}(\nabla \mathcal{L}(\theta_k)). \quad (\text{A.1.52})$$

From sign-gradient descent, we can derive the famous Adam optimization algorithm. Note that for a scalar  $x \in \mathbb{R}$  we can write

$$\text{sign}(x) = \frac{x}{|x|} = \frac{x}{\sqrt{x^2}}. \quad (\text{A.1.53})$$

Similarly, for a vector  $\mathbf{x} \in \mathbb{R}^n$  we write (where  $\odot$  and  $\oslash$  are element-wise multiplication and division)

$$\text{sign}(\mathbf{x}) = \mathbf{x} \oslash [\mathbf{x}^{\odot 2}]^{\odot(1/2)}. \quad (\text{A.1.54})$$

Using this representation we can write (A.1.52) as

$$\theta_{k+1} = \theta_k - h([\nabla \mathcal{L}(\theta_k)] \oslash [\nabla \mathcal{L}(\theta_k)^{\odot 2}]^{\odot \frac{1}{2}}). \quad (\text{A.1.55})$$

Now consider the stochastic regime where we are optimizing a different loss  $L_k$  at each iteration. In SGD, we “tracked” (i.e., took an average of) the gradients using Nesterov momentum. Here, we can track both the gradient and the squared gradient using momentum, i.e.,

$$\mathbf{g}_k = \beta^1 \mathbf{g}_{k-1} + (1 - \beta^1) \nabla L_k(\theta_k) \quad (\text{A.1.56})$$

$$\mathbf{s}_k = \beta^2 \mathbf{s}_{k-1} + (1 - \beta^2) [\nabla L_k(\theta_k)]^{\odot 2} \quad (\text{A.1.57})$$

$$\theta_{k+1} = \theta_k - h \mathbf{g}_k \oslash \mathbf{s}_k^{\odot \frac{1}{2}}, \quad (\text{A.1.58})$$

where  $\beta^i \in [0, 1]$  are the momentum parameters. The algorithm presented by this iteration is the celebrated *Adam* optimizer,<sup>5</sup> which is the most-used optimizer in deep learning. While convergence proofs of Adam are

---

<sup>5</sup>In order to avoid division-by-zero errors, we divide by  $\mathbf{s}_k^{\odot(1/2)} + \varepsilon \mathbf{1}_n$  where  $\varepsilon$  is small, say on the order of  $10^{-8}$ .

more involved, it falls out of the same steepest descent principle we used so far, and so we should expect that given a small enough learning rate, each update should improve the loss.

Another way to view Adam, which partially explains its empirical success, is that it dynamically updates the learning rates for each parameter based on the squared gradients. In particular, notice that we can write

$$\theta_{k+1} = \theta_k - \eta_k \odot \mathbf{g}_k \quad \text{where} \quad \eta_k = h \mathbf{s}_k^{\odot(-\frac{1}{2})} \quad (\text{A.1.59})$$

where  $\eta_k$  is the parameter-wise adaptively set learning rate. This scheme is called adaptive because if the gradient of a particular parameter is large up to iteration  $k$ , then the learning rate for this parameter becomes smaller to compensate, and vice versa, as can be seen from the above equation.

## A.2 Computing Gradients via Automatic Differentiation

Above, we discussed several optimization algorithms for deep networks which assumed access to a *first-order oracle*, i.e., a device which would allow us to compute  $\mathcal{L}(\theta)$  and  $\nabla \mathcal{L}(\theta)$ . For simple functions  $\mathcal{L}$ , it is possible to do this by hand. However, for deep neural networks, this quickly becomes tedious, and hinders rapid experimentation. Hence we require a general algorithm which would allow us to efficiently compute the gradients of arbitrary (sub)differentiable network architectures.

In this section, we introduce the basics of *automatic differentiation* (AD or *autodiff*), which is a computationally efficient way to compute gradients and Jacobians of general functions  $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ . We will show how this leads to the backpropagation algorithm for computing gradients of loss functions involving neural networks. A summary of the structure of this section is as follows:

1. We introduce **differentials**, a convenient formalism for calculating and organizing the derivatives of functions between high-dimensional parameter spaces (which may themselves be products of other spaces involving matrices, tensors, etc.).
2. We describe the basics of **forward-mode** and **reverse-mode** automatic differentiation, which involves considerations that are important for efficient computation of gradients/Jacobians for different kinds of functions arising in machine learning applications.
3. We describe **backpropagation** in the special case of a loss function applied to a stack-of-layers neural network as an instantiation of reverse-mode automatic differentiation.

Our treatment will err on the mathematical side, to give the reader a deep understanding of the underlying mathematics. The reader should ensure to couple this understanding with a strong grasp of practical aspects of automatic differentiation for deep learning, for example as manifested in the outstanding tutorial of Karpathy [Kar22b].

### A.2.1 Differentials

A full accounting of this subsection is given in the excellent guide [BEJ25]. To motivate differentials, let us first consider the simple example of a differentiable function  $\mathcal{L} : \mathbb{R} \rightarrow \mathbb{R}$  acting on a parameter  $\theta$ . We can write

$$\mathcal{L}(\theta) - \mathcal{L}(\theta_0) = \mathcal{L}'(\theta_0) \cdot (\theta - \theta_0) + o(|\theta - \theta_0|). \quad (\text{A.2.1})$$

If we take  $\delta\theta \doteq \theta - \theta_0$  and  $\delta\mathcal{L} \doteq \mathcal{L}(\theta_0 + \delta\theta) - \mathcal{L}(\theta_0)$ , we can write

$$\delta\mathcal{L} = \mathcal{L}'(\theta_0) \cdot \delta\theta + o(|\delta\theta|). \quad (\text{A.2.2})$$

We will (non-rigorously) define  $d\theta$  and  $d\mathcal{L}$ , i.e., the *differentials* of  $\theta$  and  $\mathcal{L}$ , to be *infinitesimally small* changes in  $\theta$  and  $\mathcal{L}$ . Think of them as what one gets when  $\delta\theta$  (and therefore  $\delta\mathcal{L}$ ) are extremely small. The goal of differential calculus, in some sense, is to study the relationships between the differentials  $d\theta$  and  $d\mathcal{L}$ , namely, seeing how small changes in the input of a function change the output. We should note that the differential  $d\theta$  is the *same shape* as  $\theta$ , and the differential  $d\mathcal{L}$  is the *same shape* as  $\mathcal{L}$ . In particular, we can write

$$d\mathcal{L} = \mathcal{L}'(\theta) \cdot d\theta, \quad (\text{A.2.3})$$

whereby we have that all higher powers of  $|d\theta|$ , such as  $(d\theta)^2$ , are 0.

Let's see how this works for a higher dimensions, i.e.,  $\mathcal{L}: \mathbb{R}^n \rightarrow \mathbb{R}$ . Then we still have

$$d\mathcal{L} = \mathcal{L}'(\theta) \cdot d\theta \quad (\text{A.2.4})$$

for some notion of a derivative  $\mathcal{L}'(\theta)$ . Since  $\theta$  (hence  $d\theta$ ) is a column vector here and  $\mathcal{L}$  (hence  $d\mathcal{L}$ ) is a scalar, we must have that  $\mathcal{L}'(\theta)$  is a row vector. In this case,  $\mathcal{L}'(\theta)$  is the Jacobian of  $\mathcal{L}$  w.r.t.  $\theta$ . Here notice that we have set all higher powers and products of coordinates of  $d\theta$  to 0. In sum,

*All products and powers  $\geq 2$  of differentials are equal to 0.*

Now consider a higher-order tensor function  $\mathcal{L}: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{p \times q}$ . Then our basic linearization equation is insufficient for this case:  $d\mathcal{L} = \mathcal{L}'(\theta) \cdot d\theta$  does not make sense since  $\theta$  is an  $m \times n$  matrix but  $d\mathcal{L}$  is a  $p \times q$  matrix, so there is no possible vector or matrix shape for  $\mathcal{L}'(\theta)$  that works in general (as no matrix can multiply a  $m \times n$  matrix to form a  $p \times q$  matrix unless  $m = p$ ). So we must have a slightly more advanced interpretation.

Namely, we consider  $\mathcal{L}'(\theta)$  as a *linear transformation* whose input is  $\theta$ -space and whose output is  $\mathcal{L}$ -space, which takes in a small change in  $\theta$  and outputs the corresponding small change in  $\mathcal{L}$ . Namely, we can write

$$d\mathcal{L} = \mathcal{L}'(\theta)[d\theta]. \quad (\text{A.2.5})$$

In the previous cases,  $\mathcal{L}'(\theta)$  was first a linear operator  $\mathbb{R} \rightarrow \mathbb{R}$  whose action was to multiply its input by the scalar derivative of  $\mathcal{L}$  with respect to  $\theta$ , and then a linear operator  $\mathbb{R}^n \rightarrow \mathbb{R}$  whose action was to multiply its input by the Jacobian derivative of  $\mathcal{L}$  with respect to  $\theta$ . In general  $\mathcal{L}'(\theta)$  is the “derivative” of  $\mathcal{L}$  w.r.t.  $\theta$ . Think of  $\mathcal{L}'$  as a generalized version of the Jacobian of  $\mathcal{L}$ . As such, it follows some simple derivative rules, most crucially the chain rule.

**Theorem A.1** (Differential Chain Rule). *Suppose  $\mathcal{L} = f \circ g$  where  $f$  and  $g$  are differentiable. Then*

$$d\mathcal{L} = f'(g(\theta))g'(\theta)[d\theta], \quad (\text{A.2.6})$$

where (as usual) multiplication indicates composition of linear operators. In particular,

$$\mathcal{L}'(\theta) = f'(g(\theta))g'(\theta) \quad (\text{A.2.7})$$

in the sense of equality of linear operators.

It is productive to think of the multivariate chain rule in functorial terms: composition of functions gets ‘turned into’ matrix multiplication of Jacobians (composition of linear operators!). We illustrate the power of this result and this perspective through several examples.

*Example A.4.* Consider the function  $f(\mathbf{X}) = \mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}^\top$ . Then

$$df = f(\mathbf{X} + d\mathbf{X}) - f(\mathbf{X}) = [\mathbf{W}(\mathbf{X} + d\mathbf{X}) + \mathbf{b}\mathbf{1}^\top] - [\mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}^\top] = \mathbf{W}d\mathbf{X}. \quad (\text{A.2.8})$$

Thus the derivative of an affine function w.r.t. its input is

$$f'(\mathbf{X})[d\mathbf{X}] = \mathbf{W}d\mathbf{X} \implies f'(\mathbf{X}) = \mathbf{W}. \quad (\text{A.2.9})$$

Notice that  $f'$  is *constant*. On the other hand, consider the function  $g(\mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}^\top$ . Then

$$dg = g(\mathbf{W} + d\mathbf{W}, \mathbf{b} + d\mathbf{b}) - g(\mathbf{W}, \mathbf{b}) = [(\mathbf{W} + d\mathbf{W})\mathbf{X} + (\mathbf{b} + d\mathbf{b})\mathbf{1}^\top] - [\mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}^\top] \quad (\text{A.2.10})$$

$$= (d\mathbf{W})\mathbf{X} + (d\mathbf{b})\mathbf{1}^\top = g'(\mathbf{W}, \mathbf{b})[d\mathbf{W}, d\mathbf{b}]. \quad (\text{A.2.11})$$

Notice that this derivative is *constant* in  $\mathbf{W}, \mathbf{b}$  (which makes sense since  $g$  itself is linear) and linear in the differential inputs  $d\mathbf{W}, d\mathbf{b}$ . ■

*Example A.5.* Consider the function  $f = gh$  where  $g, h$  are differentiable functions whose outputs can multiply together. Then  $f = p \circ v$  where  $v = (g, h)$  and  $p(a, b) = ab$ . Applying the chain rule we have

$$df = p'(v(x))v'(x)[dx]. \quad (\text{A.2.12})$$

To compute  $v'(x)$  we can compute

$$dv = v'(x)[dx] = v(x + dx) - v(x) = \begin{bmatrix} g(x + dx) - g(x) \\ h(x + dx) - h(x) \end{bmatrix} = \begin{bmatrix} g'(x)[dx] \\ h'(x)[dx] \end{bmatrix}. \quad (\text{A.2.13})$$

To compute  $p'$  we can compute

$$dp = p'(a, b)[da, db] = p(a + da, b + db) - p(a, b) = (a + da)(b + db) - ab \quad (\text{A.2.14})$$

$$= (da)b + a(db) + (da)(db) = (da)b + a(db), \quad (\text{A.2.15})$$

where (recall) the product of the differentials  $da$  and  $db$  is set to 0. Therefore

$$p'(a, b)[da, db] = (da)b + (db)a. \quad (\text{A.2.16})$$

Putting these together, we find

$$f'(x)[dx] = p'(v(x))v'(x)[dx] = p'(g(x), h(x))[g'(x)[dx], h'(x)[dx]] \quad (\text{A.2.17})$$

$$= (g'(x)[dx])h(x) + g(x)(h'(x)[dx]). \quad (\text{A.2.18})$$

This gives

$$f'(x)[dx] = (g'(x)[dx])h(x) + g(x)(h'(x)[dx]). \quad (\text{A.2.19})$$

If for example we say that  $f, g, h: \mathbb{R} \rightarrow \mathbb{R}$  then everything commutes so

$$f'(x)[dx] = (g'(x)h(x) + g(x)h'(x))[dx] \implies f'(x) = g'(x)h(x) + g(x)h'(x) \quad (\text{A.2.20})$$

which is the familiar product rule. ■

*Example A.6.* Consider the function  $f(\mathbf{A}) = \mathbf{A}^\top \mathbf{B} \mathbf{A}$  where  $\mathbf{A}$  is a matrix and  $\mathbf{B}$  is a constant matrix. Then, letting  $f = gh$  where  $g(\mathbf{A}) = \mathbf{A}^\top \mathbf{A}$  and  $h(\mathbf{A}) = \mathbf{B} \mathbf{A}$ , we can use the product rule to obtain

$$f'(\mathbf{A})[\mathbf{dA}] = (g'(\mathbf{A})[\mathbf{dA}])h(\mathbf{A}) + g(\mathbf{A})(h'(\mathbf{A})[\mathbf{dA}]) \quad (\text{A.2.21})$$

$$= ((\mathbf{dA})^\top \mathbf{A} + \mathbf{A}^\top (\mathbf{dA}))\mathbf{B} \mathbf{A} + \mathbf{A}^\top \mathbf{B} \mathbf{A}(\mathbf{dA}). \quad (\text{A.2.22})$$

■

*Example A.7.* Consider the function  $f: \mathbb{R}^{m \times n \times k} \rightarrow \mathbb{R}^{m \times n}$  given by

$$f(\mathbf{A})_{ij} = \sum_{t=1}^k A_{ijt}. \quad (\text{A.2.23})$$

We cannot write a (matrix-valued) Jacobian or gradient for this function. But we can compute its differential just fine:

$$\mathbf{d}f_{ij} = [f(\mathbf{A} + \mathbf{dA}) - f(\mathbf{A})]_{ij} = \sum_{t=1}^k \mathbf{dA}_{ijt} = \mathbf{1}_k^\top (\mathbf{dA})_{ij}. \quad (\text{A.2.24})$$

So

$$(f'(\mathbf{A})[\mathbf{dA}])_{ij} = \mathbf{1}_k^\top (\mathbf{dA})_{ij}, \quad (\text{A.2.25})$$

which represents a higher-order tensor multiplication operation that is nonetheless well-defined. ■

This gives us all the technology we need to compute differentials of everything. The last thing we cover in this section is a method to compute gradients using the differential. Namely, for a function  $\mathcal{L}$  whose output is a scalar, the gradient  $\nabla\mathcal{L}$  is defined as

$$d\mathcal{L} = \mathcal{L}'(\theta)[d\theta] = \langle \nabla\mathcal{L}(\theta), d\theta \rangle, \quad (\text{A.2.26})$$

where the inner product here is the “standard” inner product for the specified objects (i.e., for vectors it’s the  $\ell^2$  inner product, whereas for matrices it’s the Frobenius inner product, and for higher-order tensors it’s the analogous sum-of-coordinates inner product). This definition is the correct generalization of the ‘familiar’ example of the gradient of a function from  $\mathbb{R}^n$  to  $\mathbb{R}$  as the vector of partial derivatives—a version of Taylor’s theorem for general functions  $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$  makes this connection rigorous. So one way to compute the gradient  $\nabla\mathcal{L}$  is to compute the differential  $d\mathcal{L}$  and rewrite it in the form  $\langle \text{something}, d\theta \rangle$ , then that “something” is the gradient.

## A.2.2 Automatic Differentiation

The main idea of AD is to compute the chain rule efficiently. The basic problem we need to cope with is the following. In the optimization section of the appendix, we considered that the parameter space  $\Theta$  was an abstract Euclidean space like  $\mathbb{R}^n$ . In practice the parameters are really some collection of vectors, matrices, and higher-order objects:  $\Theta = \mathbb{R}^{m \times n} \times \mathbb{R}^n \times \mathbb{R}^{r \times q \times p} \times \mathbb{R}^{r \times q} \times \dots$ . While in theory this is the same thing as a large parameter space  $\mathbb{R}^{n'}$  for some (very large)  $n'$ , computationally efficient algorithms for differentiation must treat these two spaces differently. Forward and reverse mode automatic differentiation are two different schemes for performing this computation.

Let us do a simple example to start. Let  $\mathcal{L}$  be defined by  $\mathcal{L} = a \circ b \circ c$  where  $a, b, c$  are differentiable. Then the *chain rule* gives

$$\mathcal{L}'(\theta) = a'(b(c(\theta)))b'(c(\theta))c'(\theta). \quad (\text{A.2.27})$$

To compute  $\mathcal{L}(\theta)$ , we first compute  $c(\theta)$  then  $b(c(\theta))$  then  $a(b(c(\theta)))$ , and store them all. There are two ways to compute  $\mathcal{L}'(\theta)$ . The *forward-mode AD* will compute

$$c'(\theta) \implies b'(c(\theta))c'(\theta) \implies a'(b(c(\theta)))b'(c(\theta))c'(\theta) \quad (\text{A.2.28})$$

i.e., computing the derivatives “from the bottom-up”. The *reverse mode AD* will compute

$$a'(b(c(\theta))) \implies a'(b(c(\theta)))b'(c(\theta)) \implies a'(b(c(\theta)))b'(c(\theta))c'(\theta), \quad (\text{A.2.29})$$

i.e., computing the derivatives “from the top down”. To see why this matters, suppose that  $f : \mathbb{R}^p \rightarrow \mathbb{R}^s$  is given by  $f = a \circ b \circ c$  where  $a : \mathbb{R}^r \rightarrow \mathbb{R}^s$ ,  $b : \mathbb{R}^q \rightarrow \mathbb{R}^r$ ,  $c : \mathbb{R}^p \rightarrow \mathbb{R}^q$ . Then the chain rule is:

$$f'(\mathbf{x}) = a'(b(c(\mathbf{x})))b'(c(\mathbf{x}))c'(\mathbf{x}) \quad (\text{A.2.30})$$

where (recall)  $f'$  is the derivative, in this case the Jacobian (since the input and output of each function are both vectors). Assuming that computing each Jacobian is trivial and the only cost is multiplying the Jacobians together, forward-mode AD has the following computational costs (assuming that multiplying  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times k}$  takes  $\mathcal{O}(mnk)$  time):

$$\text{computing } c'(\mathbf{x}) \in \mathbb{R}^{q \times p} \text{ takes negligible time} \quad (\text{A.2.31})$$

$$\text{computing } b'(c(\mathbf{x}))c'(\mathbf{x}) \in \mathbb{R}^{r \times p} \text{ takes } \mathcal{O}(pqr) \text{ time} \quad (\text{A.2.32})$$

$$\text{computing } a'(b(c(\mathbf{x})))b'(c(\mathbf{x}))c'(\mathbf{x}) \in \mathbb{R}^{s \times p} \text{ takes } \mathcal{O}(pqr + prs) \text{ time.} \quad (\text{A.2.33})$$

Meanwhile, doing reverse-mode AD has the following computational costs:

$$\text{computing } a'(b(c(\mathbf{x}))) \in \mathbb{R}^{s \times r} \text{ takes negligible time} \quad (\text{A.2.34})$$

$$\text{computing } a'(b(c(\mathbf{x})))b'(c(\mathbf{x})) \in \mathbb{R}^{s \times q} \text{ takes } \mathcal{O}(qrs) \text{ time} \quad (\text{A.2.35})$$

$$\text{computing } a'(b(c(\mathbf{x})))b'(c(\mathbf{x}))c'(\mathbf{x}) \in \mathbb{R}^{s \times p} \text{ takes } \mathcal{O}(qrs + pqs) \text{ time.} \quad (\text{A.2.36})$$

In other words, the forward-mode AD takes  $\mathcal{O}(p(qr+rs))$  time, and the reverse-mode AD takes  $\mathcal{O}(s(pq+qr))$  time. *These take a different amount of time!*

More generally, suppose that  $f = f^L \circ \dots \circ f^1$  where each  $f^\ell: \mathbb{R}^{d^{\ell-1}} \rightarrow \mathbb{R}^{d^\ell}$ , so that  $f: \mathbb{R}^{d^0} \rightarrow \mathbb{R}^{d^L}$ . Then the forward-mode AD takes  $\mathcal{O}(d^0(\sum_{\ell=2}^L d^{\ell-1}d^\ell))$  time while the reverse-mode AD takes  $\mathcal{O}(d^L(\sum_{\ell=1}^{L-1} d^{\ell-1}d^\ell))$  time. From the above rates, we see that all else equal:

- If the function to optimize has *more outputs than inputs* (i.e.,  $d^L > d^0$ ), use *forward-mode AD*.
- If the function to optimize has *more inputs than outputs* (i.e.,  $d^0 > d^L$ ), use *reverse-mode AD*.

In a neural network, we compute the gradient of a *loss function*  $\mathcal{L}: \Theta \rightarrow \mathbb{R}$ , where the parameter space  $\Theta$  is usually very high-dimensional. So in practice we always use reverse-mode AD for training neural networks. Reverse-mode AD, in the context of training neural networks, is called *backpropagation*.

### A.2.3 Back Propagation

In this section, we will discuss algorithmic backpropagation using a simple yet completely practical example. Suppose that we fix an input-label pair  $(\mathbf{X}, \mathbf{y})$ , and fix a network architecture  $f_\theta = f_\theta^L \circ \dots \circ f_\theta^1 \circ f_\theta^{\text{emb}}$  where  $\theta = (\theta^{\text{emb}}, \theta^1, \dots, \theta^m, \theta^{\text{head}})$  and task-specific head  $h_{\theta^{\text{head}}}$ , and write

$$\mathbf{Z}_\theta^1(\mathbf{X}) \doteq f_\theta^{\text{emb}}(\mathbf{X}), \quad (\text{A.2.37})$$

$$\mathbf{Z}_\theta^{\ell+1}(\mathbf{X}) \doteq f_\theta^\ell(\mathbf{Z}_\theta^\ell(\mathbf{X})), \quad \forall \ell \in \{1, \dots, L\}, \quad (\text{A.2.38})$$

$$\hat{\mathbf{y}}_{\theta^{\text{head}}}(\mathbf{X}) \doteq h_\theta(\mathbf{Z}_\theta^{L+1}(\mathbf{X})). \quad (\text{A.2.39})$$

Then, we can define the loss on this one term by

$$\mathcal{L}(\theta) \doteq \mathsf{L}(\mathbf{y}, \hat{\mathbf{y}}_\theta(\mathbf{X})), \quad (\text{A.2.40})$$

where  $\mathsf{L}$  is a differentiable function of its second argument. Then the backward-mode AD instructs us to compute the derivatives in the order  $\theta^{\text{head}}, \theta^L, \dots, \theta^1, \theta^{\text{emb}}$ .

To carry out this computation, let us make some changes to the notation.

- First, let us change the notation to emphasize the dependency structure between the variables. Namely,

$$\mathbf{Z}^1 \doteq f^{\text{emb}}(\mathbf{X}, \theta^{\text{emb}}) \quad (\text{A.2.41})$$

$$\mathbf{Z}^{\ell+1} \doteq f^\ell(\mathbf{Z}^\ell, \theta^\ell), \quad \forall \ell \in \{1, \dots, L\}, \quad (\text{A.2.42})$$

$$\hat{\mathbf{y}} \doteq h(\mathbf{Z}^{L+1}, \theta^{\text{head}}), \quad (\text{A.2.43})$$

$$\mathcal{L} \doteq \mathsf{L}(\mathbf{y}, \hat{\mathbf{y}}). \quad (\text{A.2.44})$$

- Then, instead of having the derivative be  $f'$ , we explicitly notate the independent variable and write the derivative as  $\frac{df}{d\theta^\ell}$ , for example. This is because there are many variables in our model and we only care about one at a time.

We can start by computing the appropriate differentials. First, for  $\theta^{\text{head}}$  we have

$$d\mathcal{L} = d\mathsf{L} \quad (\text{A.2.45})$$

$$= \frac{d\mathsf{L}}{d\hat{\mathbf{y}}} \cdot d\hat{\mathbf{y}} \quad (\text{A.2.46})$$

$$= \frac{d\mathsf{L}}{d\hat{\mathbf{y}}} \cdot d(h(\mathbf{Z}^{L+1}, \theta^{\text{head}})) \quad (\text{A.2.47})$$

$$= \frac{d\mathsf{L}}{d\hat{\mathbf{y}}} \left[ \frac{dh}{d\mathbf{Z}^{L+1}} \cdot d\mathbf{Z}^{L+1} + \frac{dh}{d\theta^{\text{head}}} \cdot d\theta^{\text{head}} \right]. \quad (\text{A.2.48})$$

Now since  $\mathbf{Z}^{L+1}$  does not depend on  $\theta^{\text{head}}$ , we have  $d\mathbf{Z}^{L+1} = 0$ , so in the end it holds (using the fact that the gradient is the transpose of the derivative for a function  $\mathsf{L}$  whose codomain is  $\mathbb{R}$ ):

$$d\mathcal{L} = \frac{d\mathsf{L}}{d\hat{\mathbf{y}}} \cdot \frac{dh}{d\theta^{\text{head}}} \cdot d\theta^{\text{head}} \quad (\text{A.2.49})$$

$$= [\nabla_{\hat{\mathbf{y}}} \mathcal{L}]^\top \frac{dh}{d\theta^{\text{head}}} \cdot d\theta^{\text{head}} \quad (\text{A.2.50})$$

$$= \left\langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, \frac{dh}{d\theta^{\text{head}}} \cdot d\theta^{\text{head}} \right\rangle \quad (\text{A.2.51})$$

$$= \left\langle \left( \frac{dh}{d\theta^{\text{head}}} \right)^* \nabla_{\hat{\mathbf{y}}} \mathcal{L}, d\theta^{\text{head}} \right\rangle \quad (\text{A.2.52})$$

$$= \langle \nabla_{\theta^{\text{head}}} \mathcal{L}, d\theta^{\text{head}} \rangle. \quad (\text{A.2.53})$$

Thus to compute  $\nabla_{\theta^{\text{head}}} \mathcal{L}$ , we compute the gradient  $\nabla_{\hat{\mathbf{y}}} \mathcal{L}$  and the *adjoint*<sup>6</sup> of the derivative  $\frac{dh}{d\theta^{\text{head}}}$  and multiply (i.e., apply the adjoint linear transformation to the gradient). In practice, both derivatives can be computed by hand, but many modern computational frameworks can *automatically* define the derivatives (and/or their adjoints) given code for the “forward pass,” i.e., the loss function computation. While extending this automatic derivative definition to as many functions as possible is an area of active research, the resource [BEJ25] describes one basic approach to do it in some detail. By the way, backpropagation is also called the *adjoint method* for this reason — i.e., that we use adjoints derivatives to compute the gradient.

Now let us compute the differentials w.r.t. some  $\theta^\ell$ :

$$d\mathcal{L} = \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \cdot d\mathbf{Z}^{\ell+1} \quad (\text{A.2.54})$$

$$= \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \cdot d(f^\ell(\mathbf{Z}^\ell, \theta^\ell)) \quad (\text{A.2.55})$$

$$= \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \left[ \frac{df^\ell}{d\mathbf{Z}^\ell} \cdot d\mathbf{Z}^\ell + \frac{df^\ell}{d\theta^\ell} \cdot d\theta^\ell \right] \quad (\text{A.2.56})$$

$$= \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \cdot \frac{df^\ell}{d\theta^\ell} \cdot d\theta^\ell \quad (\text{b/c } \mathbf{Z}^\ell \text{ isn't fn. of } \theta^\ell \text{ so } d\mathbf{Z}^\ell = 0) \quad (\text{A.2.57})$$

$$= [\nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}]^\top \frac{df^\ell}{d\theta^\ell} \cdot d\theta^\ell \quad (\text{A.2.58})$$

$$= \left\langle \nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}, \frac{df^\ell}{d\theta^\ell} \cdot d\theta^\ell \right\rangle \quad (\text{A.2.59})$$

$$= \left\langle \left( \frac{df^\ell}{d\theta^\ell} \right)^* \nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}, d\theta^\ell \right\rangle \quad (\text{A.2.60})$$

$$= \langle \nabla_{\theta^\ell} \mathcal{L}, d\theta^\ell \rangle. \quad (\text{A.2.61})$$

Thus to compute  $\nabla_{\theta^\ell} \mathcal{L}$  we compute  $\nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}$  then apply the adjoint derivative  $\left( \frac{df^\ell}{d\theta^\ell} \right)^*$  to it. Since  $\nabla_{\theta^\ell} \mathcal{L}$  depends on  $\nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}$ , we also want to be able to compute the gradients w.r.t.  $\mathbf{Z}^\ell$ . This can be computed in the exact same way:

$$d\mathcal{L} = \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \cdot d\mathbf{Z}^{\ell+1} \quad (\text{A.2.62})$$

$$= \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \cdot d(f^\ell(\mathbf{Z}^\ell, \theta^\ell)) \quad (\text{A.2.63})$$

$$= \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \left[ \frac{df^\ell}{d\mathbf{Z}^\ell} \cdot d\mathbf{Z}^\ell + \frac{df^\ell}{d\theta^\ell} \cdot d\theta^\ell \right] \quad (\text{A.2.64})$$

$$= \frac{d\mathcal{L}}{d\mathbf{Z}^{\ell+1}} \cdot \frac{df^\ell}{d\mathbf{Z}^\ell} \cdot d\mathbf{Z}^\ell \quad (\text{b/c } \theta^\ell \text{ isn't fn. of } \mathbf{Z}^\ell \text{ so } d\theta^\ell = 0 \text{ this time}) \quad (\text{A.2.65})$$

$$= \left\langle \left( \frac{df^\ell}{d\mathbf{Z}^\ell} \right)^* \nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}, d\mathbf{Z}^\ell \right\rangle \quad (\text{same machine}) \quad (\text{A.2.66})$$

---

<sup>6</sup>The adjoint is like a generalized transpose for more general linear transformations. Particularly, for a given pair of inner product spaces and linear transformation  $T$  between those spaces, the adjoint  $T^*$  is defined by the identity  $\langle T\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, T^*\mathbf{y} \rangle$ . In finite dimensions (i.e., all cases relevant to this book) the adjoint exists and is unique.

$$= \langle \nabla_{\mathbf{Z}^\ell} \mathcal{L}, d\mathbf{Z}^\ell \rangle. \quad (\text{A.2.67})$$

Thus to compute  $\nabla_{\mathbf{Z}^\ell} \mathcal{L}$  we compute  $\nabla_{\mathbf{Z}^{\ell+1}} \mathcal{L}$  then apply the adjoint derivative  $\left(\frac{df^\ell}{d\mathbf{Z}^\ell}\right)^*$  to it. So we have a recursion to compute  $\nabla_{\mathbf{Z}^\ell} \mathcal{L}$  for all  $\ell$ , with base case  $\nabla_{\mathbf{Z}^{L+1}} \mathcal{L}$ , which is given by

$$d\mathcal{L} = d\mathbf{L} \quad (\text{A.2.68})$$

$$= \frac{d\mathbf{L}}{d\hat{\mathbf{y}}} \cdot d\hat{\mathbf{y}} \quad (\text{A.2.69})$$

$$= \frac{d\mathbf{L}}{d\hat{\mathbf{y}}} \cdot dh(\mathbf{Z}^{L+1}, \theta^{\text{head}}) \quad (\text{A.2.70})$$

$$= \frac{d\mathbf{L}}{d\hat{\mathbf{y}}} \left[ \frac{dh}{d\mathbf{Z}^{L+1}} \cdot d\mathbf{Z}^{L+1} + \frac{dh}{d\theta^{\text{head}}} \cdot d\theta^{\text{head}} \right] \quad (\text{A.2.71})$$

$$= \frac{d\mathbf{L}}{d\hat{\mathbf{y}}} \cdot \frac{dh}{d\mathbf{Z}^{L+1}} \cdot d\mathbf{Z}^{L+1} \quad (\text{A.2.72})$$

$$= \left\langle \left( \frac{dh}{d\mathbf{Z}^{L+1}} \right)^* \nabla_{\hat{\mathbf{y}}} \mathbf{L}, d\mathbf{Z}^{L+1} \right\rangle \quad (\text{A.2.73})$$

$$= \langle \nabla_{\mathbf{Z}^{L+1}} \mathcal{L}, d\mathbf{Z}^{L+1} \rangle. \quad (\text{A.2.74})$$

Thus we have the recursion:

$$\nabla_{\mathbf{Z}^{L+1}} \mathcal{L} = \left( \frac{dh}{d\mathbf{Z}^{L+1}} \right)^* \nabla_{\hat{\mathbf{y}}} \mathbf{L} \quad (\text{A.2.75})$$

$$\nabla_{\mathbf{Z}^L} \mathcal{L} = \left( \frac{df^L}{d\mathbf{Z}^L} \right)^* \nabla_{\mathbf{Z}^{L+1}} \mathcal{L} \quad (\text{A.2.76})$$

$$\nabla_{\theta^L} \mathcal{L} = \left( \frac{df^L}{d\theta^L} \right)^* \nabla_{\mathbf{Z}^{L+1}} \mathcal{L} \quad (\text{A.2.77})$$

$$\vdots \quad (\text{A.2.78})$$

$$\nabla_{\mathbf{Z}^1} \mathcal{L} = \left( \frac{df^1}{d\mathbf{Z}^1} \right)^* \nabla_{\mathbf{Z}^2} \mathcal{L} \quad (\text{A.2.79})$$

$$\nabla_{\theta^1} \mathcal{L} = \left( \frac{df^1}{d\theta^1} \right)^* \nabla_{\mathbf{Z}^2} \mathcal{L} \quad (\text{A.2.80})$$

$$\nabla_{\theta^{\text{emb}}} \mathcal{L} = \left( \frac{df^{\text{emb}}}{d\theta^{\text{emb}}} \right)^* \nabla_{\mathbf{Z}^1} \mathcal{L}. \quad (\text{A.2.81})$$

This gives us a computationally efficient algorithm to find all gradients in the whole network. We'll finish this section by computing the adjoint derivative for a simple layer.

*Example A.8.* Consider the “linear” (affine) layer  $f^\ell$

$$f^\ell(\mathbf{Z}, \mathbf{W}^\ell, \mathbf{b}^\ell) \doteq \mathbf{W}^\ell \mathbf{Z} + \mathbf{b}^\ell \mathbf{1}^\top = [\mathbf{W}^\ell \quad \mathbf{b}^\ell]. \quad (\text{A.2.82})$$

We can compute the differential w.r.t. both parameters as

$$df^\ell = [(\mathbf{W}^\ell + d\mathbf{W}^\ell)\mathbf{Z} + (\mathbf{b}^\ell + db^\ell)\mathbf{1}^\top] - [\mathbf{W}^\ell \mathbf{Z} + \mathbf{b}^\ell] \quad (\text{A.2.83})$$

$$= (d\mathbf{W}^\ell)\mathbf{Z} + (db^\ell)\mathbf{1}^\top. \quad (\text{A.2.84})$$

Thus the derivative of this transformation is

$$\frac{df^\ell}{d(\mathbf{W}^\ell, \mathbf{b}^\ell)} [d\mathbf{W}^\ell, db^\ell] = (d\mathbf{W}^\ell)\mathbf{Z} + (db^\ell)\mathbf{1}^\top, \quad (\text{A.2.85})$$

namely, representing the following linear transformation from  $\mathbb{R}^{m \times d} \times \mathbb{R}^m$  to  $\mathbb{R}^{m \times n}$ :

$$T[\mathbf{A}, \mathbf{u}] = \mathbf{A}\mathbf{Z} + \mathbf{u}\mathbf{1}^\top. \quad (\text{A.2.86})$$

We calculate the adjoint  $T^* : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times d} \times \mathbb{R}^m$  w.r.t. the sum-over-coordinates (Frobenius) inner product by the following procedure:

$$\langle T[\mathbf{A}, \mathbf{u}], \mathbf{B} \rangle_{\mathbb{R}^{m \times n}} = \text{tr}((\mathbf{A}\mathbf{Z} + \mathbf{u}\mathbf{1}^\top)\mathbf{B}^\top) \quad (\text{A.2.87})$$

$$= \text{tr}(\mathbf{A}\mathbf{Z}\mathbf{B}^\top + \mathbf{u}\mathbf{1}^\top\mathbf{B}^\top) \quad (\text{A.2.88})$$

$$= \text{tr}(\mathbf{A}\mathbf{Z}\mathbf{B}^\top) + \text{tr}(\mathbf{u}\mathbf{1}^\top\mathbf{B}^\top) \quad (\text{A.2.89})$$

$$= \text{tr}(\mathbf{B}\mathbf{Z}^\top\mathbf{A}^\top) + \text{tr}(\mathbf{1}^\top\mathbf{B}^\top\mathbf{u}) \quad (\text{A.2.90})$$

$$= \langle \mathbf{B}\mathbf{Z}^\top, \mathbf{A} \rangle_{\mathbb{R}^{m \times d}} + \langle \mathbf{B}\mathbf{1}, \mathbf{u} \rangle_{\mathbb{R}^m} \quad (\text{A.2.91})$$

$$= \langle \mathbf{B}(\mathbf{Z}^\top, \mathbf{1}), (\mathbf{A}, \mathbf{u}) \rangle_{\mathbb{R}^{m \times d} \times \mathbb{R}^m} \quad (\text{A.2.92})$$

So  $T^*\mathbf{B} = \mathbf{B}(\mathbf{Z}^\top, \mathbf{1})$ . ■

Note that as a simple application of chain rule, both backpropagation and automatic differentiation work over general “computational graphs”, i.e., compositions of (simple) functions. We give all examples as neural network layers because this is the most common example in practice.

## A.3 Game Theory and Minimax Optimization

In certain cases, such as in Chapter 5, a learning problem cannot be reduced to a single optimization problem but rather represents multiple potentially opposing components of the system try to each minimize their own objective. Examples of this paradigm include distribution learning via generative adversarial networks (GAN) and closed-loop transcription (CTRL). We will denote such a system as a *two-player game*, where we have two “players” (i.e., components) called Player 1 and Player 2 trying to minimize their objectives  $\mathcal{L}^1$  and  $\mathcal{L}^2$  respectively. Player 1 picks parameters  $\theta \in \Theta$  and Player 2 picks parameters  $\eta \in \mathcal{H}$ . In this book we consider the special case of *zero-sum games*, i.e., defining a common objective  $\mathcal{L}$  such that  $\mathcal{L} = -\mathcal{L}^1 = \mathcal{L}^2$ .

Our first, very preliminary example is as follows. Suppose that there exists functions  $u(\theta)$  and  $v(\eta)$  such that

$$\mathcal{L}(\theta, \eta) = -u(\theta) + v(\eta). \quad (\text{A.3.1})$$

Then both players’ objectives are independent of the other player, and the players should try to achieve their respective optima:

$$\theta^* \in \arg \min_{\theta \in \Theta} u(\theta), \quad \eta^* \in \arg \min_{\eta \in \mathcal{H}} v(\eta). \quad (\text{A.3.2})$$

The pair  $(\theta^*, \eta^*)$  is a straightforward special case of an *equilibrium*: a situation where neither player will want to move, given the chance, since moving will end up making their own situation worse. However, not all games are so trivial; many have more complicated objectives and information structures.

In this book, the relevant game-theoretic formalism is a Stackelberg game (variously called sequential game). In this formalism, one player (without loss of generality Player 1, and also described as a *leader*) picks their parameters before the other (i.e., Player 2, also described as a *follower*), and the follower can use the full knowledge of the leader’s choice to make their own choice. The correct notion of equilibrium for a Stackelberg game is a *Stackelberg equilibrium*. To explain this equilibrium, note that since Player 2 (i.e., the follower) can choose  $\eta$  reactively to the choice  $\theta_1$  made by Player 1 (i.e., the leader), Player 2 would of course choose the  $\eta$  which minimizes  $\mathcal{L}(\theta_1, \cdot)$ . But of course a rational Player 1 would realize this, and so pick a  $\theta_1$  such that the worst-case  $\eta$  picked by Player 2 according to this rule is not too bad. More formally, let  $\mathcal{S}(\theta) \doteq \arg \min_{\eta \in \mathcal{H}} \mathcal{L}(\theta, \eta)$  be the set of  $\eta$  minimizing  $\mathcal{L}(\theta, \eta)$ , i.e., the set of all  $\eta$  which Player 2 is liable to play given that Player 1 has played  $\theta$ . Then  $(\theta^*, \eta^*)$  is a Stackelberg equilibrium if

$$\theta^* \in \arg \max_{\theta \in \Theta} \min_{\eta \in \mathcal{S}(\theta)} \mathcal{L}(\theta, \eta), \quad \eta^* \in \arg \min_{\eta \in \mathcal{H}} \mathcal{L}(\theta^*, \eta). \quad (\text{A.3.3})$$

Actually (proof as exercise), one can show that in the context of two-player zero-sum Stackelberg games,  $(\theta^*, \eta^*)$  is a Stackelberg equilibrium if and only if

$$\theta^* \in \arg \max_{\theta \in \Theta} \min_{\eta \in \mathcal{H}} \mathcal{L}(\theta, \eta), \quad \eta^* \in \arg \min_{\eta \in \mathcal{H}} \mathcal{L}(\theta^*, \eta), \quad (\text{A.3.4})$$

(note that the notation  $\mathcal{S}(\theta)$  is not used nor needed).

*Proof.* Note that

$$\mathcal{S}(\theta) = \arg \min_{\eta \in H} \mathcal{L}(\theta, \eta), \quad (\text{A.3.5})$$

so it holds

$$\min_{\eta \in \mathcal{S}(\theta)} \mathcal{L}(\theta, \eta) = \min_{\eta \in \arg \min_{\eta' \in H} \mathcal{L}(\theta, \eta')} \mathcal{L}(\theta, \eta) = \min_{\eta \in H} \mathcal{L}(\theta, \eta). \quad (\text{A.3.6})$$

□

In the rest of the section, we will briefly discuss some algorithmic approaches to learn Stackelberg equilibria. The intuition you should have is that learning an equilibrium is like letting the different parts of the system automatically figure out tradeoffs between the different objectives they want to optimize.

We end this section with a caveat: in two-player zero-sum games, if it holds that

$$\max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta) = \min_{\eta \in H} \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta) \quad (\text{A.3.7})$$

then every Stackelberg equilibrium is a saddle point,<sup>7</sup> i.e.,

$$\theta^* \in \arg \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*), \quad \eta^* \in \arg \min_{\eta \in H} \mathcal{L}(\theta^*, \eta), \quad (\text{A.3.8})$$

and vice versa, and furthermore each Stackelberg equilibrium has the (same) objective value

$$\max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta).$$

*Proof.* Suppose that indeed

$$\max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta) = \min_{\eta \in H} \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta). \quad (\text{A.3.9})$$

First suppose that  $(\theta^*, \eta^*)$  is a saddle point. We will show it is a Stackelberg equilibrium. By definition we have

$$\min_{\eta \in H} \mathcal{L}(\theta^*, \eta) = \mathcal{L}(\theta^*, \eta^*) = \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*). \quad (\text{A.3.10})$$

It then holds for any  $\theta$  and  $\eta$  that

$$\min_{\eta \in H} \mathcal{L}(\theta, \eta) \leq \mathcal{L}(\theta, \eta^*) \leq \mathcal{L}(\theta^*, \eta^*). \quad (\text{A.3.11})$$

Therefore

$$\max_{\theta \in \Theta} \mathcal{L}(\theta, \eta) \leq \mathcal{L}(\theta^*, \eta^*). \quad (\text{A.3.12})$$

Completely symmetrically,

$$\mathcal{L}(\theta^*, \eta^*) \leq \mathcal{L}(\theta^*, \eta) \leq \max_{\theta \in \Theta} \mathcal{L}(\theta^*, \eta) \implies \mathcal{L}(\theta^*, \eta^*) \leq \min_{\eta \in H} \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta). \quad (\text{A.3.13})$$

Therefore since  $\max \min = \min \max$  we have

$$\max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta) = \mathcal{L}(\theta^*, \eta^*) = \min_{\eta \in H} \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta). \quad (\text{A.3.14})$$

In particular, it holds that

$$\theta^* \in \arg \max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta). \quad (\text{A.3.15})$$

From the saddle point condition we have  $\eta^* \in \arg \min_{\eta \in H} \mathcal{L}(\theta^*, \eta)$ . So  $(\theta^*, \eta^*)$  is a Stackelberg equilibrium. Furthermore we have also proved that all saddle points obey (A.3.14).

Now let  $(\theta^*, \eta^*)$  be a Stackelberg equilibrium. We claim that it is a saddle point, which completes the proof. By the definition of minimax equilibrium,

$$\max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta) = \mathcal{L}(\theta^*, \eta^*). \quad (\text{A.3.16})$$

---

<sup>7</sup>Famously called a *Nash equilibrium*.

Then by the  $\min \max = \max \min$  assumption we have

$$\max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta) = \mathcal{L}(\theta^*, \eta^*) = \min_{\eta \in H} \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta). \quad (\text{A.3.17})$$

This proves that all minimax equilibria have the desired objective value  $\mathcal{L}(\theta^*, \eta^*)$ . Now we want to show that

$$\theta^* \in \arg \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*), \quad \eta^* \in \arg \min_{\eta \in H} \mathcal{L}(\theta^*, \eta). \quad (\text{A.3.18})$$

Indeed the latter assertion holds by definition of the minimax equilibrium, so only the former need be proved. Namely, we will show that

$$\max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*) = \mathcal{L}(\theta^*, \eta^*). \quad (\text{A.3.19})$$

To show this note that by definition of the max

$$\mathcal{L}(\theta^*, \eta^*) \leq \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*), \quad (\text{A.3.20})$$

meanwhile we have  $\min_{\eta \in H} \mathcal{L}(\theta, \eta) \leq \mathcal{L}(\theta, \eta^*)$  so

$$\mathcal{L}(\theta^*, \eta^*) = \max_{\theta \in \Theta} \min_{\eta \in H} \mathcal{L}(\theta, \eta) \leq \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*). \quad (\text{A.3.21})$$

Therefore it holds

$$\mathcal{L}(\theta^*, \eta^*) = \max_{\theta \in \Theta} \mathcal{L}(\theta, \eta^*), \quad (\text{A.3.22})$$

and the proof is complete.  $\square$

Conditions under which the  $\min \max = \max \min$  equality holds are given by so-called *minimax theorems*; the most famous of these is a theorem of von Neumann. However, in the cases we think about, this property usually does not hold.

### A.3.1 Learning Stackelberg Equilibria

How can we learn Stackelberg equilibria via GDA? In general this is clearly impossible, since learning Stackelberg equilibria via GDA is obviously at least as hard as computing a global minimizer of a loss function (say by setting the shared objective  $\mathcal{L}(\theta, \eta)$  to only be a function of  $\eta$ ). As such, we can achieve two types of convergence guarantees:

- When  $\mathcal{L}$  is (strongly) concave in the first argument  $\theta$  and (strongly) convex in the second argument  $\eta$  (as well as having Lipschitz gradients in both arguments), we can achieve exponentially fast convergence to a Stackelberg equilibrium.
- When  $\mathcal{L}$  is not concave or convex in either argument, we can achieve *local convergence guarantees*: namely, if we initialize the parameter values near a (local) Stackelberg equilibrium and the optimization geometry is good then we can learn that equilibrium efficiently.

The former situation is exactly analogous to the case of single-player optimization, where we proved that gradient descent converges exponentially fast for strongly convex objectives which have Lipschitz gradient. The latter situation is also analogous to the case of single-player optimization, although we did not cover it in depth due to technical difficulty; indeed there exist local convergence guarantees for nonconvex objectives which have locally nice geometry.

The algorithm in these two cases is the same algorithm, called Gradient Descent-Ascent (GDA). To motivate GDA, suppose we are trying to learn  $\theta^*$ . We could do gradient ascent on the function  $\theta \mapsto \min_{\eta \in H} \mathcal{L}(\theta, \eta)$ . But then we would need to take the derivative in  $\theta$  of this function. To see how to do this, suppose that  $\mathcal{L}$  is strongly convex in  $\eta$  so that there is one minimizer  $\eta^*(\theta)$  of  $\mathcal{L}(\theta, \cdot)$ . Then, *Danskin's theorem* says that

$$\nabla_{\theta} \left[ \min_{\eta \in H} \mathcal{L}(\theta, \eta) \right] = \nabla_{\theta} \mathcal{L}(\theta, \text{stop\_grad}(\eta^*(\theta))), \quad (\text{A.3.23})$$

where the gradient is *only with respect to the first argument* (i.e., not a total derivative which would require computing the Jacobian of  $\eta^*(\theta)$  with respect to  $\theta$ ), indicated by the stop-gradient operator.<sup>8</sup> In order to take the derivative in  $\theta$ , we need to set up a *secondary process* to also optimize  $\eta$  to obtain an approximation for  $\eta^*(\theta)$ . We can do this through the following algorithmic template:

$$\eta_{k+1} = \eta_{k+1}^{T+1}; \quad \eta_{k+1}^{t+1} = \eta_{k+1}^t - h \nabla_\eta \mathcal{L}(\theta_k, \eta_{k+1}^t), \quad \forall t \in [T]; \quad \eta_{k+1}^1 = \eta_k \quad (\text{A.3.24})$$

$$\theta_{k+1} = \theta_k + h \nabla_\theta \mathcal{L}(\theta_k, \eta_{k+1}). \quad (\text{A.3.25})$$

That is, we take  $T$  steps of gradient descent to update  $\eta_k$  (hopefully to the minimizer of  $\mathcal{L}(\theta_k, \cdot)$ ), and then take a gradient ascent step to update  $\theta_k$ . As a bonus, on top of estimating  $\theta_K \approx \arg \max_\theta \min_\eta \mathcal{L}(\theta, \eta)$ , we also learn an  $\eta_K \approx \arg \min_\eta \mathcal{L}(\theta_K, \eta)$  — this is an approximate Stackelberg equilibrium.

This method is often not done in practice, as it requires  $T + 1$  total gradient descent iterations to update  $\theta$  once. Instead, we use the so-called (*simultaneous*) *Gradient Descent-Ascent* (GDA) iteration

$$\theta_{k+1} = \theta_k + h \nabla_\theta \mathcal{L}(\theta_k, \eta_k) \quad (\text{A.3.26})$$

$$\eta_{k+1} = \eta_k - Th \nabla_\eta \mathcal{L}(\theta_k, \eta_k), \quad (\text{A.3.27})$$

which can be implemented efficiently via a single gradient step on  $(\theta, \eta)$ . The crucial idea here is, to make our method close to the inefficient iteration above, we use an  $\eta$  update which is  $T$  times faster than the  $\theta$  update (these can be seen as nearly the same by taking a linearization of the dynamics).

It is crucial to pick  $T$  sensibly. How can we do that? In the sequel, we discuss two configurations of  $T$  which lead to convergence of GDA to a Stackelberg equilibrium under different assumptions.

### Convergence of One-Timescale GDA to Stackelberg Equilibrium

If  $T = 1$  (i.e., named *one-timescale* because both  $\theta$  and  $\eta$  updates are of the same scale), then the GDA algorithm becomes

$$\theta_{k+1} = \theta_k + h \nabla_\theta \mathcal{L}(\theta_k, \eta_k), \quad \eta_{k+1} = \eta_k - h \nabla_\eta \mathcal{L}(\theta_k, \eta_k). \quad (\text{A.3.28})$$

If  $\mathcal{L}$  has Lipschitz gradients in  $\theta$  and  $\eta$ , and is strongly concave in  $\theta$  and strongly convex in  $\eta$ , and is coercive (i.e.,  $\lim_{\|\theta\|_2, \|\eta\|_2 \rightarrow \infty} \mathcal{L}(\theta, \eta) = \infty$ ), then all saddle points are Stackelberg equilibria and vice versa. The work [ZAK24] shows that GDA (again, with  $T = 1$ ) with sufficiently small step size  $h$  converges to a saddle point (hence Stackelberg equilibrium) exponentially fast if one exists, analogously to gradient descent for strongly convex functions with Lipschitz gradient.<sup>9</sup> To our knowledge, this flavor of results constitute the only known rigorous justification for single-timescale GDA.

### Local Convergence of Two-Timescale GDA to Stackelberg Equilibrium

Strong convexity/concavity is a *global* property, and none of the games we look into in this book have objectives which are globally strongly concave/strongly convex. In this case, the best we can hope for is *local* convergence to Stackelberg equilibria: if the parameters are initialized close to a Stackelberg equilibrium, then GDA can converge onto it, given an appropriate step size  $h$  and timescale  $T$ .

In fact, our results also hold for a version of the *local* Stackelberg equilibrium called the *differential Stackelberg equilibrium*, which was introduced in [FCR19] (though we use the precise definition in [LFD+22]), and which we define as follows. A point  $(\theta^*, \eta^*)$  is a differential Stackelberg equilibrium if:

- $\nabla_\eta \mathcal{L}(\theta^*, \eta^*) = \mathbf{0}$ ;
- $\nabla_\eta^2 \mathcal{L}(\theta^*, \eta^*)$  is symmetric positive definite;
- $\nabla_\theta \mathcal{L}(\theta^*, \eta^*) = \mathbf{0}$ ;

<sup>8</sup>In the case that  $\mathcal{L}$  is not strongly convex in  $\eta$  but rather just convex, Danskin's theorem can be stated in terms of subdifferentials. If  $\mathcal{L}$  is not convex at all in  $\eta$ , then this derivative may not be well-defined, but one can obtain (local) convergence guarantees for the resulting algorithm anyways. Hence we use Danskin's theorem as a motivation and not a justification for our algorithms. Danskin's theorems can be generalized into more relaxed circumstances by the so-called *envelope theorems*.

<sup>9</sup>We do not provide the step size or exponential base since they are complicated functions of the strong convexity/concavity/Lipschitz constant of the gradient. Of course, the paper [ZAK24] provides the precise parameter values.

- $(\nabla_\theta^2 \mathcal{L} + [\frac{d}{d\theta} \nabla_\eta \mathcal{L}] [\nabla_\eta^2 \mathcal{L}]^{-1} [\frac{d}{d\eta} \nabla_\theta \mathcal{L}]) (\theta^*, \eta^*)$  is symmetric negative definite.

Notice that the last condition asks for the (total) Hessian

$$\nabla^2 \mathcal{L}(\theta, \eta) = \begin{bmatrix} \nabla_\theta^2 \mathcal{L}(\theta, \eta) & \frac{d}{d\eta} \nabla_\theta^2 \mathcal{L}(\theta, \eta) \\ \frac{d}{d\theta} \nabla_\eta^2 \mathcal{L}(\theta, \eta) & \nabla_\eta^2 \mathcal{L}(\theta) \end{bmatrix} \quad (\text{A.3.29})$$

or, equivalently, its Schur complement to be negative definite. If we look at the computation of  $\nabla_\theta [\min_{\eta \in H} \mathcal{L}(\theta, \eta)]$  furnished by Danskin's theorem, the last two criteria are actually constraints on the gradient and Hessian of the function  $\theta \mapsto \min_{\eta \in H} \mathcal{L}(\theta, \eta)$ , ensuring that the gradient is 0 and the Hessian is negative semidefinite. This intuition tells us that we can expect that each Stackelberg equilibrium is a differential Stackelberg equilibrium; [FCR20] confirms this rigorously (up to some technical conditions).

Analogously to the notion of strict local optimum in single-player optimization (where we require  $\nabla^2 \mathcal{L}(\theta^*)$  to be positive semidefinite), the definition of differential Stackelberg equilibrium *implies that  $\mathcal{L}(\theta, \cdot)$  is locally (strictly) convex in a neighborhood of the equilibrium, and that  $\min_{\eta \in H} \mathcal{L}(\cdot, \eta)$  is locally (strictly) concave in the same region.*

In this context, we present the result from [LFD+22]. Let  $(\theta^*, \eta^*)$  be a differential Stackelberg equilibrium. Suppose that  $\mathcal{L}$  has Lipschitz gradients, i.e.,

$$\max \left\{ \|\nabla_\theta^2 \mathcal{L}(\theta^*, \eta^*)\|_2, \|\nabla_\eta^2 \mathcal{L}(\theta^*, \eta^*)\|_2, \left\| \frac{d}{d\eta} \nabla_\theta \mathcal{L}(\theta^*, \eta^*) \right\|_2 \right\} \leq \beta. \quad (\text{A.3.30})$$

Further define the local strong convexity/concavity parameters of  $\mathcal{L}(\theta, \cdot)$  and  $\min_{\eta} \mathcal{L}(\cdot, \eta)$  respectively as

$$\mu_\eta = \lambda_{\min}(\nabla_\eta^2 \mathcal{L}(\theta^*, \eta^*)), \quad \mu_\theta = \min \left\{ \beta, - \left( \nabla_\theta^2 \mathcal{L} + \left[ \frac{d}{d\theta} \nabla_\eta \mathcal{L} \right] [\nabla_\eta^2 \mathcal{L}]^{-1} \left[ \frac{d}{d\eta} \nabla_\theta \mathcal{L} \right] \right) (\theta^*, \eta^*) \right\}. \quad (\text{A.3.31})$$

Then define the local condition numbers as

$$\kappa_\eta = L/\mu_\eta, \quad \kappa_\theta = L/\mu_\theta. \quad (\text{A.3.32})$$

The paper [LFD+22] says that if we take step size  $h = \frac{1}{4\beta T}$  and take  $T \geq 2\kappa$ , so that the algorithm is

$$\theta_{k+1} = \theta_k + \frac{1}{4\beta T} \nabla_\theta \mathcal{L}(\theta_k, \eta_k), \quad (\text{A.3.33})$$

$$\eta_{k+1} = \eta_k - \frac{1}{4\beta} \nabla_\eta \mathcal{L}(\theta_k, \eta_k), \quad (\text{A.3.34})$$

the total Hessian  $\nabla^2 \mathcal{L}(\theta^*, \eta^*)$  is diagonalizable, and we initialize  $(\theta_0, \eta_0)$  close enough to  $(\theta^*, \eta^*)$ , then there are positive constants  $c_0, c_1 > 0$  such that

$$\|(\theta_k, \eta_k) - (\theta^*, \eta^*)\|_2 \leq c_0 \left( 1 - \frac{c_1}{T\kappa_\theta} \right)^k \|(\theta_0, \eta_0) - (\theta^*, \eta^*)\|_2, \quad (\text{A.3.35})$$

implying exponential convergence to the differential Stackelberg equilibrium.

### A.3.2 Practical Considerations when Learning Stackelberg Equilibria

In practice, we do not know how to initialize parameters close to a (differential) Stackelberg equilibrium. Due to symmetries within the objective, including those induced by overparameterization of the neural networks being trained, one can (heuristically) expect that most initializations are close to a Stackelberg equilibrium. Also, we do not know how to compute the step size  $h$  or the timescale  $T$ , since they are dependent on properties of the loss  $\mathcal{L}$  at the equilibrium. In practice, there are some common approaches:

- Take  $T = 1$  (equal step-sizes), and use updates for  $\theta$  and  $\eta$  that are derived from a learning-rate-adaptive optimizer like Adam (as opposed to vanilla GD). Here, you *hope* (but do not *know*) that the optimizer can adjust the learning rates to learn a good equilibrium.

- Take  $T$  to be some constant like  $T = 10^6$  which implies that  $\eta$  equilibrates  $10^6$  times as fast as  $\theta$ . Here you can also use Adam-style updates, and hope that it fixes the time scale.
- Let  $T$  depend on the iteration  $k$ , and let  $T_k \rightarrow \infty$  as  $k \rightarrow \infty$ . This schedule was studied (also in the case of noise) by Borkar in [Bor97].

For example, you can use this while training CTRL-style models (see Chapter 5), where the encoder is Player 1 and the decoder is Player 2. Some theory about CTRL is given in Theorem 5.1.

## A.4 Exercises

*Exercise A.1.* We have shown that for a smooth function  $f$ , gradient descent converges linearly to the global optimum if it is strongly convex. However, in general nonconvex optimization, we do not have convexity, let alone strong convexity. Fortunately, in some cases,  $f$  satisfies the so-called  $\mu$ -Polyak-Łojasiewicz (PL) inequality, i.e., there exists a constant  $\mu > 0$  such that for all  $\theta$ ,

$$\frac{1}{2} \|\nabla f(\theta)\|_2^2 \geq \mu (f(\theta) - f(\theta^*)) ,$$

where  $\theta^*$  is a minimizer of  $f$ .

Please show that under the PL inequality and the assumption that  $f$  is  $\beta$ -smooth, gradient descent (A.1.12) converges linearly to  $\theta^*$ .

*Exercise A.2.* Compute the differential and adjoint derivative of the softmax function, defined as follows.

$$\text{softmax} \left( \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \right) = \frac{1}{\sum_{i=1}^n e^{x_i}} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}. \quad (\text{A.4.1})$$

*Exercise A.3.* Carry through the backpropagation computation for a  $L$ -layer MLP, as defined in Section 7.2.3.

*Exercise A.4.* Carry through the backpropagation computation for a  $L$ -layer transformer, as defined in Section 7.2.3.

*Exercise A.5.* Carry through the backpropagation computation for an autoencoder with  $L$  encoder layers and  $L$  decoder layers (without necessarily specifying an architecture).

## Appendix B

# Entropy, Diffusion, Denoising, and Lossy Coding

*“The increase of disorder or entropy with time is one example of what is called an arrow of time, something that distinguishes the past from the future, giving a direction to time.”*

– A Brief History of Time, Stephen Hawking

In this appendix we provide proofs for several facts, mentioned in Chapter 3, which are related to differential entropy, how it evolves under diffusion processes, and its connections to lossy coding. We will make the following mild assumption about the random variable representing the data source, denoted  $\mathbf{x}$ .

**Assumption B.1.**  $\mathbf{x}$  is supported on a compact set  $\mathcal{S} \subseteq \mathbb{R}^D$  of radius at most  $R$ , i.e.,  $R \doteq \sup_{\xi \in \mathcal{S}} \|\xi\|_2$ .

In particular, since compact sets in Euclidean space are bounded, it holds  $R < \infty$ . We will consistently use the notation  $B_r(\xi) \doteq \{\mathbf{u} \in \mathbb{R}^D : \|\xi - \mathbf{u}\|_2 \leq r\}$  to denote the Euclidean ball of radius  $r$  centered at  $\xi$ . In this sense, Assumption B.1 has  $\mathcal{S} \subseteq B_R(\mathbf{0})$ .

Notice that this assumption holds for (almost) all variables we care about in practice, as it is (often) imposed by a normalization step during data pre-processing.

### B.1 Differential Entropy of Low-Dimensional Distributions

In this short appendix we discuss the differential entropy of low-dimensional distributions. By definition, the differential entropy of a random variable  $\mathbf{x}$  which does not have a density is  $-\infty$ ; this includes all random variables supported on low-dimensional sets. The objective of this section is to discuss why this is a “morally correct” value.

In fact, let  $\mathbf{x}$  be any random variable such that Assumption B.1 holds, the support  $\mathcal{S}$  of  $\mathbf{x}$  has 0 volume.<sup>1</sup> We will consider the case that  $\mathbf{x}$  is uniform on  $\mathcal{S}$ .<sup>2</sup> Our goal is to compute  $h(\mathbf{x})$ .

In this case,  $\mathbf{x}$  would not have a density; in the counterfactual world where we did not know  $h(\mathbf{x}) = -\infty$ , we could not directly define it using the standard definition of differential entropy. Instead, as in the rest of analysis and information theory it would be reasonable to consider the limit of entropies of successively better approximations  $\mathbf{x}_\varepsilon$  of  $\mathbf{x}$  which have densities, i.e.,

$$h(\mathbf{x}) \text{ “=} \lim_{\varepsilon \searrow 0} h(\mathbf{x}_\varepsilon). \quad (\text{B.1.1})$$

To this end, the basic idea is to take an  $\varepsilon$ -thickening of  $\mathcal{S}$ , say  $\mathcal{S}_\varepsilon$  defined as

$$\mathcal{S}_\varepsilon = \bigcup_{\xi \in \mathcal{S}} B_\varepsilon(\xi) \quad (\text{B.1.2})$$

---

<sup>1</sup>Formally this means that  $\mathcal{S}$  is Borel measurable with Borel measure 0.

<sup>2</sup>Say, w.r.t. the Hausdorff measure on  $\mathcal{S}$ .

and visualized in Figure B.1. We will work with random variables whose support is  $\mathcal{S}_\varepsilon$ , which is fully-

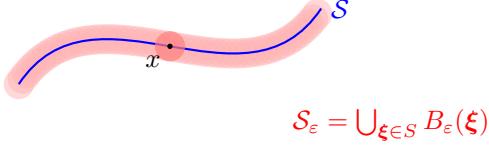


Figure B.1: Illustration of the  $\varepsilon$ -thickening  $\mathcal{S}_\varepsilon$  of a curve  $\mathcal{S} \subseteq \mathbb{R}^2$ .

dimensional, and take the limit as  $\varepsilon \rightarrow 0$ . Indeed, define  $\mathbf{x}_\varepsilon \sim \mathcal{U}(\mathcal{S}_\varepsilon)$ . Since  $\mathcal{S}_\varepsilon$  has positive volume,  $\mathbf{x}_\varepsilon$  has a density  $p_\varepsilon$  equal to

$$p_\varepsilon(\boldsymbol{\xi}) = \mathbf{1}(\boldsymbol{\xi} \in \mathcal{S}_\varepsilon) \cdot \frac{1}{\text{vol}(\mathcal{S}_\varepsilon)}. \quad (\text{B.1.3})$$

Computing the entropy of  $\mathbf{x}_\varepsilon$  using the convention that  $0 \log 0 = 0$ , it holds

$$h(\mathbf{x}_\varepsilon) = - \int_{\mathbb{R}^D} p_\varepsilon(x) \log p_\varepsilon(x) dx \quad (\text{B.1.4})$$

$$= - \int_{\mathcal{S}_\varepsilon} \frac{1}{\text{vol}(\mathcal{S}_\varepsilon)} \log \left( \frac{1}{\text{vol}(\mathcal{S}_\varepsilon)} \right) d\boldsymbol{\xi} \quad (\text{B.1.5})$$

$$= \frac{\log(\text{vol}(\mathcal{S}_\varepsilon))}{\text{vol}(\mathcal{S}_\varepsilon)} \int_{\mathcal{S}_\varepsilon} d\boldsymbol{\xi} \quad (\text{B.1.6})$$

$$= \log(\text{vol}(\mathcal{S}_\varepsilon)). \quad (\text{B.1.7})$$

Since  $\mathcal{S}$  is compact  $\text{vol}(\mathcal{S}_\varepsilon)$  is finite and tends to 0 as  $\varepsilon \searrow 0$ . Thus

$$h(\mathbf{x}) = \lim_{\varepsilon \searrow 0} h(\mathbf{x}_\varepsilon) = \lim_{\varepsilon \searrow 0} \log(\text{vol}(\mathcal{S}_\varepsilon)) = -\infty, \quad (\text{B.1.8})$$

as desired.

The above calculation is actually a corollary of a much more famous and celebrated set of results about the maximum possible entropy of  $\mathbf{x}$  subject to certain constraints on the distribution of  $\mathbf{x}$ . We would be remiss to not provide the results here; the proofs are provided in Chapter 2 of [PW22], for example.

**Theorem B.1.** *Let  $\mathbf{x}$  be a random variable on  $\mathbb{R}^D$ .*

1. *If  $\mathbf{x}$  is supported on a compact set  $\mathcal{S} \subseteq \mathbb{R}^D$  (i.e., Assumption B.1) then*

$$h(\mathbf{x}) \leq h(\mathcal{U}(\mathcal{S})) = \log \text{vol}(\mathcal{S}). \quad (\text{B.1.9})$$

2. *If  $\mathbf{x}$  has finite covariance such that, for a PSD matrix  $\boldsymbol{\Sigma} \in \text{PSD}(D)$ , it holds  $\text{Cov}(\mathbf{x}) \preceq \boldsymbol{\Sigma}$  (w.r.t. the PSD ordering, i.e.,  $\boldsymbol{\Sigma} - \text{Cov}(\mathbf{x})$  is PSD), then*

$$h(\mathbf{x}) \leq h(\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})) = \frac{1}{2} \log((2\pi e)^D \det \boldsymbol{\Sigma}). \quad (\text{B.1.10})$$

3. *If  $\mathbf{x}$  has finite second moment such that, for a constant  $a \geq 0$ , it holds  $\mathbb{E} \|\mathbf{x}\|_2^2 \leq a$ , then*

$$h(\mathbf{x}) \leq h\left(\mathcal{N}\left(\mathbf{0}, \frac{a}{D} \mathbf{I}\right)\right) = \frac{D}{2} \log \frac{2\pi e a}{D}. \quad (\text{B.1.11})$$

## B.2 Diffusion and Denoising Processes

In the main body (Chapter 3), we considered a random variable  $\mathbf{x}$ , and a stochastic process defined by (3.2.1), i.e.,

$$\mathbf{x}_t = \mathbf{x} + t\mathbf{g}, \quad \forall t \in [0, T] \quad (\text{B.2.1})$$

where  $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  independently of  $\mathbf{x}$ .

The structure of this section is as follows. In Section B.2.1 we provide a formal theorem and crisp proof which shows that under Equation (B.2.1) the entropy increases, i.e.,  $\frac{d}{dt} h(\mathbf{x}_t) > 0$ . In Section B.2.2 we provide a formal theorem and crisp proof which shows that under Equation (B.2.1), the entropy decreases during denoising, i.e.,  $h(\mathbb{E}[\mathbf{x}_s | \mathbf{x}_t]) < h(\mathbf{x}_t)$  for all  $s < t$ . In Section B.2.3 we provide proofs for technical lemmas that are needed to establish the claims in the previous subsections.

Before we start, we introduce some key notations. First, let  $\varphi_t$  be the density of  $\mathcal{N}(\mathbf{0}, t^2 \mathbf{I})$ , i.e.,

$$\varphi_t(\boldsymbol{\xi}) \doteq \frac{1}{(2\pi)^{D/2} t^D} \exp\left(-\frac{\|\boldsymbol{\xi}\|_2^2}{2t^2}\right). \quad (\text{B.2.2})$$

Next,  $\mathbf{x}_t$  is supported on all of  $\mathbb{R}^D$ , so it has a *density*, which we denote  $p_t$  (as in the main body). A quick calculation shows that

$$p_t(\boldsymbol{\xi}) = \mathbb{E}[\varphi_t(\boldsymbol{\xi} - \mathbf{x})], \quad (\text{B.2.3})$$

and from this representation it is possible to deduce (i.e., from Proposition B.4) that  $p_t$  is smooth (i.e., infinitely differentiable) in  $\boldsymbol{\xi}$ , also smooth in  $t$ , and positive everywhere. This fact is somewhat remarkable at first sight: even for a completely irregular random variable  $\mathbf{x}$  (say, a Bernoulli random variable, which does not have a density), its Gaussian smoothing admits a density for every (arbitrarily small)  $t > 0$ . The proof is left as an exercise for readers well-versed in mathematical analysis.

However, we also need to add an assumption about the *smoothness* of the distribution of  $\mathbf{x}$ , which will eliminate some technical problems that occur around  $t = 0$  with low-dimensional distributions.<sup>3</sup> Despite this, we expect that our results hold under milder assumptions with additional work. For now, let us assume:

**Assumption B.2.**  $\mathbf{x}$  has a twice continuously differentiable density, denoted  $p$ .

### B.2.1 Diffusion Process Increases Entropy Over Time

In this section appendix we provide a proof of Theorem B.2. For convenience, we restate it as follows.

**Theorem B.2** (Diffusion Increases Entropy). *Let  $\mathbf{x}$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). Then*

$$h(\mathbf{x}_s) < h(\mathbf{x}_t), \quad \forall s, t: 0 \leq s < t \leq T. \quad (\text{B.2.4})$$

*Proof.* Before we start, we must ask: when does the inequality in (B.2.4) make sense? We will show in Lemma B.1 that under our assumptions, the differential entropy is well-defined, is never  $+\infty$ , and for  $t > 0$  is finite, so the (strict) inequality in (B.2.4) makes sense.

The question of well-definedness aside, the crux of this proof is to show that the density  $p_t$  of  $\mathbf{x}_t$  satisfies a particular partial differential equation, which is very similar to the *heat equation*. The heat equation is a famous PDE which describes the diffusion of heat through space. This intuitively should make sense, and paints a mental picture: as the time  $t$  increases, the probability from the original (perhaps tightly concentrated)  $\mathbf{x}$  disperses across all of  $\mathbb{R}^D$  like heat radiating from a source in a vacuum.

Such PDEs for  $p_t$ , known as *Fokker-Planck equations* for more general stochastic processes, are very powerful tools, as they allow us to describe the instantaneous temporal derivatives of  $p_t$  in terms of the instantaneous spatial derivatives of  $p_t$ , and vice versa, providing a concise description of the regularity and dynamics of  $p_t$ . Once we obtain dynamics for  $p_t$ , we can then use the system to obtain another one which describes the dynamics of  $h(\mathbf{x}_t)$ , which after all is just a functional of  $p_t$ .

The description of the PDE involves a mathematical object called the Laplacian  $\Delta$ . Recall from your multivariable calculus class that the Laplacian operating on a differentiable-in-time and twice-differentiable-in-space function  $f: (0, T) \times \mathbb{R}^D \rightarrow \mathbb{R}$  is given by

$$\Delta f_t(\boldsymbol{\xi}) = \text{tr}(\nabla^2 f_t(\boldsymbol{\xi})) = \sum_{i=1}^D \frac{\partial^2 f_t}{\partial \xi_i^2}(\boldsymbol{\xi}). \quad (\text{B.2.5})$$

---

<sup>3</sup>As then various quantities become highly irregular and dealing with them would require significant additional analysis.

Namely, from using the integral representation of  $p_t$  and differentiating under the integral, we can compute the derivatives of  $p_t$  (which we do in Proposition B.1) and observe that  $p_t$  satisfies the heat-like PDE

$$\frac{\partial p_t}{\partial t}(\boldsymbol{\xi}) = t \Delta p_t(\boldsymbol{\xi}). \quad (\text{B.2.6})$$

Then for finding the dynamics of  $h(\mathbf{x}_t)$ , we can use Proposition B.3 again as well as the heat-like PDE to get

$$\frac{d}{dt} h(\mathbf{x}_t) = - \frac{d}{dt} \int_{\mathbb{R}^D} p_t(\boldsymbol{\xi}) \log p_t(\boldsymbol{\xi}) d\boldsymbol{\xi} \quad (\text{B.2.7})$$

$$= - \int_{\mathbb{R}^D} \frac{\partial}{\partial t} [p_t(\boldsymbol{\xi}) \log p_t(\boldsymbol{\xi})] d\boldsymbol{\xi} \quad (\text{B.2.8})$$

$$= - \int_{\mathbb{R}^D} \frac{\partial p_t}{\partial t}(\boldsymbol{\xi}) [1 + \log p_t(\boldsymbol{\xi})] d\boldsymbol{\xi} \quad (\text{B.2.9})$$

$$= -t \int_{\mathbb{R}^D} \Delta p_t(\boldsymbol{\xi}) [1 + \log p_t(\boldsymbol{\xi})] d\boldsymbol{\xi}. \quad (\text{B.2.10})$$

By using a slightly involved integration by parts argument (Lemma B.2), we obtain

$$\frac{d}{dt} h(\mathbf{x}_t) = t \int_{\mathbb{R}^D} \langle \nabla \log p_t(\boldsymbol{\xi}), \nabla p_t(\boldsymbol{\xi}) \rangle d\boldsymbol{\xi} \quad (\text{B.2.11})$$

$$= t \int_{\mathbb{R}^D} \frac{\|\nabla p_t(\boldsymbol{\xi})\|_2^2}{p_t(\boldsymbol{\xi})} d\boldsymbol{\xi} \quad (\text{B.2.12})$$

$$> 0 \quad (\text{B.2.13})$$

where strict inequality holds in the last line because, for it to not hold,  $\nabla p_t(\boldsymbol{\xi})$  would need to vanish almost everywhere (i.e., everywhere except possibly on a set of zero volume), but this would imply that  $p_t$  would be constant almost everywhere, a contradiction with a fact that  $p_t$  is a density.

To complete the proof we just use the fundamental theorem of calculus

$$h(\mathbf{x}_t) = h(\mathbf{x}_s) + \int_s^t \frac{d}{du} h(\mathbf{x}_u) du > h(\mathbf{x}_s), \quad (\text{B.2.14})$$

which proves the claim. (Note that this does not make sense when  $h(\mathbf{x}_s) = -\infty$ , which can only happen when  $s = 0$  and  $h(\mathbf{x}) = -\infty$ , but in this case  $h(\mathbf{x}_t) > -\infty$  so the claim is vacuously true anyways.)  $\square$

## B.2.2 Denoising Process Reduces Entropy Over Time

Recall that in Section 3.2.1 we start with the random variable  $\mathbf{x}_T$  and iteratively denoise it using iterations of the form

$$\hat{\mathbf{x}}_s \doteq \mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t = \hat{\mathbf{x}}_t] = \frac{s}{t} \hat{\mathbf{x}}_t + \left(1 - \frac{s}{t}\right) \bar{\mathbf{x}}^*(t, \hat{\mathbf{x}}_t). \quad (\text{B.2.15})$$

for  $s, t \in \{t_0, t_1, \dots, t_L\}$  with  $s < t$  and  $\mathbf{x}_T = \hat{\mathbf{x}}_T$ . We wish to prove that  $h(\hat{\mathbf{x}}_s) < h(\hat{\mathbf{x}}_t)$ , showing that the denoising process actually reduces the entropy.

Before we go about doing this, we make several remarks about the problem statement. First, Tweedie's formula (3.2.20) says that

$$\bar{\mathbf{x}}^*(t, \mathbf{x}_t) = \mathbf{x}_t + t^2 \nabla p_t(\mathbf{x}_t), \quad (\text{B.2.16})$$

which likens a full denoising step from time  $t$  to time 0 to a gradient step on the log-density of  $\mathbf{x}_t$ . Can we get a similar result for the full denoising step from time  $t$  to time  $s$  in (B.2.15)? It turns out that indeed we can, and it is pretty simple. By using (B.2.15) and Tweedie's formula (3.2.20), we obtain

$$\mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t] = \frac{s}{t} \mathbf{x}_t + \left(1 - \frac{s}{t}\right) (\mathbf{x}_t + t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)) = \mathbf{x}_t + \left(1 - \frac{s}{t}\right) t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t). \quad (\text{B.2.17})$$

So this iterative denoising step is again a gradient step on the perturbed log-density  $\log p_t$  with a shrunken step size. In particular, this step can be seen as a perturbation of the distribution of the random variable

$\mathbf{x}_t$  by the *score function vector field*, suggesting a connection to stochastic differential equations (SDEs) and the theory of diffusion models [SSK+21]. Indeed, a proof of the following result Theorem B.3 can be developed using this powerful machinery and a limiting argument (e.g., following the technical approach in the exposition of [CCL+23]). We will give a simpler proof here, which will use only elementary tools and thereby illuminate some of the key quantities behind the process of entropy reduction via denoising. On the other hand, we will need to deal with some slightly technical calculations due to the fact that the denoising process in Theorem B.3 does *not* correspond exactly to the *reverse* process associated to the noise addition process that generates the observation  $\mathbf{x}_t$ .<sup>4</sup>

We want to prove that  $h(\mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t]) < h(\mathbf{x}_t)$ , i.e., formally:

**Theorem B.3.** *Let  $\mathbf{x}$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). Then*

$$h(\mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t]) < h(\mathbf{x}_t), \quad \forall s, t \in [0, T]: \quad 0 < t \leq \frac{R}{\sqrt{2D}}, \quad 0 \leq s < t \cdot \min \left\{ 1, \frac{R^2/D - 2t^2}{R^2/D - t^2} \right\}. \quad (\text{B.2.18})$$

*Proof.* This proof uses two main ideas:

1. First, write down a density for  $\mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t]$  using a change-of-variables formula.
2. Second, bound this density to control the entropy.

The change of variables is justified by Corollary B.1, which was originally derived in [Gri11].

We execute these ideas now. From Corollary B.1, we obtain that the function  $\bar{\mathbf{x}}$  defined as  $\bar{\mathbf{x}}(\xi) \doteq \mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t = \xi]$  is differentiable, injective, and thus invertible on its range, which we henceforth denote  $\mathcal{X} \subseteq \mathbb{R}^D$ . We denote its inverse as  $\bar{\mathbf{x}}^{-1}$ . Using a change-of-variables formula, the density  $\bar{p}$  of  $\bar{\mathbf{x}}(\mathbf{x}_t)$  is given by

$$\bar{p}(\xi) \doteq \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))}, \quad (\text{B.2.19})$$

where (recall, from Section A.2)  $\bar{\mathbf{x}}'$  is the Jacobian of  $\bar{\mathbf{x}}$ . Since from Lemma B.3 we know  $\bar{\mathbf{x}}'$  is a positive definite matrix, the determinant is positive and so the whole density is positive. Then it follows that

$$h(\bar{\mathbf{x}}(\mathbf{x}_t)) = - \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \log \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} d\xi \quad (\text{B.2.20})$$

$$= - \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \log((p_t \circ \bar{\mathbf{x}}^{-1})(\xi)) d\xi \quad (\text{B.2.21})$$

$$+ \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \log \det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi))) d\xi \quad (\text{B.2.22})$$

$$= - \int_{\mathbb{R}^D} p_t(\xi) \log p_t(\xi) d\xi + \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \log \det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi))) d\xi \quad (\text{B.2.23})$$

$$= h(\mathbf{x}_t) - \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \log \left( \frac{1}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \right) d\xi. \quad (\text{B.2.24})$$

We will study the last term (including the  $-$ ), and show that it is negative.

By concavity, one has  $-x \log x \leq 1 - x$  for every  $x \geq 0$ . Hence

$$h(\bar{\mathbf{x}}(\mathbf{x}_t)) - h(\mathbf{x}_t) = - \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \log \left( \frac{1}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \right) d\xi \quad (\text{B.2.25})$$

$$\leq \int_{\mathcal{X}} (p_t \circ \bar{\mathbf{x}}^{-1})(\xi) \cdot \left( 1 - \frac{1}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} \right) d\xi \quad (\text{B.2.26})$$

<sup>4</sup>For those familiar with diffusion models, we refer here to the time-reversed forward process not coinciding with the sequence of iterates generated by the process defined by Theorem B.3. These processes coincide in a certain limit where infinitely many steps of Theorem B.3 are taken with infinitely small levels of noise added at each step; for general, finite steps, we must introduce some approximations regardless of the level of sophistication of our tools.

$$= \int_{\mathcal{X}} (p_t \circ \bar{\mathbf{x}}^{-1})(\xi) d\xi - \int_{\mathcal{X}} \frac{(p_t \circ \bar{\mathbf{x}}^{-1})(\xi)}{\det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi)))} d\xi \quad (\text{B.2.27})$$

$$= \int_{\mathbb{R}^D} p_t(\xi) \det(\bar{\mathbf{x}}'(\bar{\mathbf{x}}^{-1}(\xi))) d\xi - \int_{\mathcal{X}} \bar{p}(\xi) d\xi \quad (\text{B.2.28})$$

$$= \int_{\mathbb{R}^D} p_t(\xi) \det\left(\mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 \nabla^2 \log p_t(\xi)\right) d\xi - 1. \quad (\text{B.2.29})$$

Now, by the AM-GM inequality on eigenvalues, we have for any symmetric positive definite matrix  $\mathbf{M} \in \text{PSD}(D)$  the bound

$$\det(\mathbf{M})^{1/D} = \prod_{i=1}^D \lambda_i(\mathbf{M})^{1/D} \leq \frac{\sum_{i=1}^D \lambda_i(\mathbf{M})}{D} = \frac{\text{tr}(\mathbf{M})}{D}, \quad (\text{B.2.30})$$

which we can apply to the above expression and obtain

$$\int_{\mathbb{R}^D} p_t(\xi) \det\left(\mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 \nabla^2 \log p_t(\xi)\right) d\xi \quad (\text{B.2.31})$$

$$\leq \int_{\mathbb{R}^D} p_t(\xi) \text{tr}\left(\frac{1}{D} \left[\mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 \nabla^2 \log p_t(\xi)\right]\right)^D d\xi \quad (\text{B.2.32})$$

$$= \int_{\mathbb{R}^D} p_t(\xi) \left(1 + \frac{\left(1 - \frac{s}{t}\right) t^2}{D} \text{tr}(\nabla^2 \log p_t(\xi))\right)^D d\xi \quad (\text{B.2.33})$$

$$= \int_{\mathbb{R}^D} p_t(\xi) \left(1 + \frac{\left(1 - \frac{s}{t}\right) t^2}{D} \Delta \log p_t(\xi)\right)^D d\xi. \quad (\text{B.2.34})$$

From Lemma B.5, it holds (where, recall,  $R$  is the radius of the support of  $\mathbf{x}$  as in Assumption B.1)

$$|\Delta \log p_t(\xi)| \leq \max\left(\frac{D}{t^2}, \left|\frac{R^2}{t^4} - \frac{D}{t^2}\right|\right) =: U_t. \quad (\text{B.2.35})$$

Then it holds

$$-\frac{\left(1 - \frac{s}{t}\right) t^2}{D} U_t \leq \frac{\left(1 - \frac{s}{t}\right) t^2}{D} \Delta \log p_t(\xi) \leq \frac{\left(1 - \frac{s}{t}\right) t^2}{D} U_t. \quad (\text{B.2.36})$$

Meanwhile, the function  $x \mapsto (1+x)^D$  is convex on  $[-1, \infty)$ , so for  $-(1-s/t)t^2U_t/D \leq x \leq (1-s/t)t^2U_t/D$  we have

$$(1+x)^d \leq \left(1 - \frac{\left(1 - \frac{s}{t}\right) t^2 U_t}{D}\right)^D + \underbrace{\left[\left(1 + \frac{\left(1 - \frac{s}{t}\right) t^2 U_t}{D}\right)^D - \left(1 - \frac{\left(1 - \frac{s}{t}\right) t^2 U_t}{D}\right)^D\right]}_{M(s,t,D)} x \quad (\text{B.2.37})$$

$$\leq 1 + M(s, t, D)x. \quad (\text{B.2.38})$$

Here  $M(s, t, D) > 0$  since  $U_t > 0$ . In the above bound, we need to verify that the lower bound for  $x$  is  $\geq -1$ . Indeed,

$$-\frac{\left(1 - \frac{s}{t}\right) t^2}{D} U_t = -\frac{\left(1 - \frac{s}{t}\right) t^2}{D} \max\left(\frac{D}{t^2}, \left|\frac{R^2}{t^4} - \frac{D}{t^2}\right|\right) \quad (\text{B.2.39})$$

$$= -\left(1 - \frac{s}{t}\right) \max\left(1, \left|\frac{R^2}{Dt^2} - 1\right|\right) \quad (\text{B.2.40})$$

Notice that this is  $\geq -1$  if and only if  $\left(1 - \frac{s}{t}\right) \cdot \left(\frac{R^2}{Dt^2} - 1\right) \geq 1$ , i.e.,  $0 < t < R/\sqrt{2D}$  and  $0 \leq s \leq t \cdot \frac{R^2/D - 2t^2}{R^2/D - t^2}$ , as granted by the assumptions.

Applying this bound, we obtain

$$\int_{\mathbb{R}^D} p_t(\xi) \left( 1 + \frac{(1 - \frac{s}{t}) t^2}{D} \Delta \log p_t(\xi) \right)^D d\xi \quad (\text{B.2.41})$$

$$\leq \int_{\mathbb{R}^D} p_t(\xi) (1 + M(s, t, D) \Delta \log p_t(\xi)) d\xi \quad (\text{B.2.42})$$

$$= 1 + M(s, t, D) \int_{\mathbb{R}^D} p_t(\xi) \Delta \log p_t(\xi) d\xi \quad (\text{B.2.43})$$

$$= 1 - M(s, t, D) \int_{\mathbb{R}^D} \langle \nabla p_t(\xi), \nabla \log p_t(\xi) \rangle d\xi \quad (\text{B.2.44})$$

$$= 1 - M(s, t, D) \int_{\mathbb{R}^D} \frac{\|\nabla p_t(\xi)\|_2^2}{p_t(\xi)} d\xi, \quad (\text{B.2.45})$$

where the last few lines are the same as in the proof of Theorem B.2. Combining this result with our previous estimate,

$$h(\bar{x}(\mathbf{x}_t)) - h(\mathbf{x}_t) \leq -M(s, t, D) \int_{\mathbb{R}^D} \frac{\|\nabla p_t(\xi)\|_2^2}{p_t(\xi)} d\xi < 0 \quad (\text{B.2.46})$$

where the inequality is strict by the same argument as in Theorem B.2.  $\square$

Notice that the bounds for  $s$  and  $t$  depend on the radius  $R$  of the data distribution, and are not so general as the bounds in Theorem B.2. The result is actually “as general as needed” in the following sense. Note that if  $\mathbf{x}$  has a twice continuously differentiable density supported on the ball of radius  $R$  centered at  $\mathbf{0}$ , then it does for  $2R$ , and  $3R$ , and so on, i.e., for any ball of radius  $R' > R$ . Thus one strategy to get the appropriate denoising guarantee is: fix a data dimension  $D$  and discretization schedule, and then set (in the analysis) the data radius  $R$  to be very large such that each denoising step satisfies the requirements for entropy decrease given in Theorem B.3. Then each step of the denoising process will indeed reduce the entropy, as desired.

### B.2.3 Technical Lemmas and Intermediate Results

In this subsection we present technical results which power our main two conceptual theorems. Our presentation will be more or less standard for mathematics; we will start with the higher-level results first, and gradually move back to the more incremental results. The higher-level results will use the incremental results, and in this way we have an easy-to-read dependency ordering of the results: no result depends on those before it. Results which do not depend on each other are generally ordered by the place they appear in the above pair of proofs.

#### Finiteness of the Differential Entropy

We first show that the entropy exists along the stochastic process and is finite.

**Lemma B.1.** *Let  $\mathbf{x}$  be any random variable, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1).*

1. *For  $t > 0$ , the differential entropy  $h(\mathbf{x}_t)$  exists and is  $> -\infty$ .*
2. *If in addition Assumption B.1 holds for  $\mathbf{x}$ , then  $h(\mathbf{x}) < \infty$  and  $h(\mathbf{x}_t) < \infty$ .*

*Proof.* To prove Lemma B.1.1, we use a classic yet tedious analysis argument. Since  $\mathbf{x}_t$  has a density, we can write

$$h(\mathbf{x}_t) = - \int_{\mathbb{R}^D} p_t(\xi) \log p_t(\xi) d\xi. \quad (\text{B.2.47})$$

Accordingly, let  $g: \mathbb{R}^D \rightarrow \mathbb{R}$  be defined as

$$g(\xi) \doteq -p_t(\xi) \log p_t(\xi) \implies h(\mathbf{x}_t) = \int_{\mathbb{R}^D} g(\xi) d\xi. \quad (\text{B.2.48})$$

As usual to bound the value of an integral in analysis, define

$$g_+(\xi) \doteq \max(g(\xi), 0), \quad g_-(\xi) \doteq \max(-g(\xi), 0) \implies g = g_+ - g_- \quad \text{and} \quad g_+, g_- \geq 0. \quad (\text{B.2.49})$$

Then

$$h(\mathbf{x}_t) = \int_{\mathbb{R}^D} g_+(\xi) d\xi - \int_{\mathbb{R}^D} g_-(\xi) d\xi, \quad (\text{B.2.50})$$

and both integrals are guaranteed to be non-negative since their integrands are.

In order to show that  $h(\mathbf{x}_t)$  is well-defined, we need to show that  $\int_{\mathbb{R}^D} g_+(\xi) d\xi < \infty$  or  $\int_{\mathbb{R}^D} g_-(\xi) d\xi < \infty$ . To show that  $h(\mathbf{x}_t) > -\infty$ , it merely suffices to show that  $\int_{\mathbb{R}^D} g_-(\xi) d\xi < \infty$ . To bound the integral of  $g_-$  we need to understand the quantity  $g_-$ , namely, we want to characterize when  $g$  is negative.

$$g(\xi) \leq 0 \iff p_t(\xi) \log p_t(\xi) \geq 0 \iff \log p_t(\xi) \geq 0 \iff p_t(\xi) \geq 1. \quad (\text{B.2.51})$$

Thus, it holds that

$$g_-(\xi) = \mathbf{1}(p_t(\xi) \geq 1) \cdot (-g(\xi)) = \mathbf{1}(p_t(\xi) \geq 1) p_t(\xi) \log p_t(\xi). \quad (\text{B.2.52})$$

In order to bound the integral of  $g_-(\xi)$ , we need to show that  $p_t$  is “not too concentrated,” namely that  $p_t$  is not too large. To prove this, in this case we are lucky enough to be able to bound the function  $g_-(\xi)$  itself. Namely, notice that

$$\max_{\xi \in \mathbb{R}^D} \varphi_t(\xi - \mathbf{x}) = \varphi_t(\mathbf{0}) = \frac{1}{(2\pi)^{D/2} t^D} =: C_t. \quad (\text{B.2.53})$$

which blows up as  $t \rightarrow 0$  but is finite for all finite  $t$ . Therefore

$$p_t(\xi) = \mathbb{E} \varphi_t(\xi - \mathbf{x}) \leq \mathbb{E} C_t = C_t. \quad (\text{B.2.54})$$

Now there are two cases.

- If  $C_t < 1$ , then  $p_t(\xi) < 1$ , so the indicator is never 1, hence  $g_- = 0$  identically and its integral is also 0.
- If  $C_t \geq 1$ , then  $\log C_t \geq 0$ , so since the logarithm is monotonically increasing,

$$\int_{\mathbb{R}^D} g_-(\xi) d\xi = \int_{\mathbb{R}^D} \mathbf{1}(p_t(\xi) \geq 1) p_t(\xi) \log p_t(\xi) d\xi \quad (\text{B.2.55})$$

$$= \mathbb{E}[\mathbf{1}(p_t(\mathbf{x}_t) \geq 1) \log p_t(\mathbf{x}_t)] \quad (\text{B.2.56})$$

$$\leq \mathbb{E}[\mathbf{1}(p_t(\mathbf{x}_t) \geq 1) \log C_t] \quad (\text{B.2.57})$$

$$= \mathbb{P}[p_t(\mathbf{x}_t) \geq 1] \log C_t. \quad (\text{B.2.58})$$

Hence we have  $\int_{\mathbb{R}^D} g_-(\xi) d\xi < \infty$ , so the differential entropy  $h(\mathbf{x}_t)$  exists and is  $> -\infty$ .

To prove Lemma B.1.2, suppose that Assumption B.1 holds. We want to show that  $h(\mathbf{x}) < \infty$  and  $h(\mathbf{x}_t) < \infty$ . The mechanism for doing this is the same, and involves the maximum entropy result Theorem B.1. Namely, since  $\mathbf{x}$  is absolutely bounded, it has a finite covariance which we will denote  $\Sigma$ . Then the covariance of  $\mathbf{x}_t$  is  $\Sigma + t^2 \mathbf{I}$ . Thus the entropy of  $\mathbf{x}$  and  $\mathbf{x}_t$  can be upper bounded by the entropy of normal distributions with the respective covariances, i.e.,  $\log[(2\pi e)^D \det(\Sigma)]$  or  $\log[(2\pi e)^D \det(\Sigma + t^2 \mathbf{I})]$ , and both are  $< \infty$ .  $\square$

### Integration by Parts in De Bruijn Identity

Finally, we fill in the integration-by-parts argument alluded to in the proofs of Theorems B.2 and B.3. The argument is conceptually pretty simple but requires some technical estimates to show that the boundary term in the integration-by-parts vanishes.

**Lemma B.2.** *Let  $\mathbf{x}$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). For  $t \geq 0$ , let  $p_t$  be the density of  $\mathbf{x}_t$ . Then for a constant  $c \in \mathbb{R}$  it holds*

$$\int_{\mathbb{R}^D} \Delta p_t(\xi) [c + \log p_t(\xi)] d\xi = - \int_{\mathbb{R}^D} \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle d\xi. \quad (\text{B.2.59})$$

*Proof.* The basic idea of this proof is in two steps:

- First, apply Green's theorem to do integration by parts over a compact set;
- Second, send the radius of this compact set to  $+\infty$ , to get integrals over all of  $\mathbb{R}^D$ .

Green's theorem says that for any compact set  $\mathcal{K} \subseteq \mathbb{R}^D$ , twice continuously differentiable  $\phi: \mathbb{R}^D \rightarrow \mathbb{R}$ , and continuously differentiable  $\psi: \mathbb{R}^D \rightarrow \mathbb{R}$ ,

$$\int_{\mathcal{K}} \{\psi(\xi) \Delta \phi(\xi) + \langle \nabla \psi(\xi), \nabla \phi(\xi) \rangle\} d\xi = \int_{\partial \mathcal{K}} \psi(\xi) \langle \nabla \phi(\xi), \mathbf{n}(\xi) \rangle d\sigma(\xi) \quad (\text{B.2.60})$$

where  $d\sigma(\xi)$  denotes an integral over the “surface measure”, i.e., the inherited measure on  $\partial \mathcal{K}$ , namely the boundary of  $\mathcal{K}$ , and accordingly  $\xi$  takes values on this surface and  $\mathbf{n}(\xi)$  is the unit normal vector to  $\mathcal{K}$  at the surface point  $\xi$ . Now, taking  $\phi(\xi) \doteq p_t(\xi)$  and  $\psi(\xi) \doteq c + \log p_t(\xi)$ , over a ball  $B_r(\mathbf{0})$  of radius  $r > 0$  centered at  $\mathbf{0}$  (so that  $\partial B_r(\mathbf{0})$  is the sphere of radius  $r$  centered at  $\mathbf{0}$  and  $\mathbf{n}(\xi) = \xi / \|\xi\|_2 = \xi / r$ ):

$$\int_{B_r(\mathbf{0})} \{\Delta p_t(\xi) [c + \log p_t(\xi)] + \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle\} d\xi \quad (\text{B.2.61})$$

$$= \int_{\partial B_r(\mathbf{0})} [c + \log p_t(\xi)] \left\langle \nabla p_t(\xi), \frac{\xi}{r} \right\rangle d\sigma(\xi) \quad (\text{B.2.62})$$

$$= \frac{1}{r} \int_{\partial B_r(\mathbf{0})} [c + \log p_t(\xi)] \langle \nabla p_t(\xi), \xi \rangle d\sigma(\xi). \quad (\text{B.2.63})$$

Sending  $r \rightarrow \infty$ , it holds that

$$\int_{\mathbb{R}^D} \{\Delta p_t(\xi) [c + \log p_t(\xi)] + \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle\} d\xi \quad (\text{B.2.64})$$

$$= \lim_{r \rightarrow \infty} \int_{B_r(\mathbf{0})} \{\Delta p_t(\xi) [c + \log p_t(\xi)] + \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle\} d\xi \quad (\text{B.2.65})$$

$$= \lim_{r \rightarrow \infty} \int_{\partial B_r(\mathbf{0})} \{\Delta p_t(\xi) [c + \log p_t(\xi)] + \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle\} d\xi \quad (\text{B.2.66})$$

$$= \lim_{r \rightarrow \infty} \frac{1}{r} \int_{\partial B_r(\mathbf{0})} [c + \log p_t(\xi)] \langle \nabla p_t(\xi), \xi \rangle d\sigma(\xi), \quad (\text{B.2.67})$$

where the first inequality follows by dominated convergence on the integrand. It remains to compute the last limit. For this, we take asymptotic expansions of each term. The main device is as follows: for  $\xi \in \partial B_r(\mathbf{0})$ , we have  $\|\xi\|_2 = r$ , so

$$p_t(\xi) = \mathbb{E}[\varphi_t(\xi - \mathbf{x})] \quad (\text{B.2.68})$$

$$= \mathbb{E} \left[ \underbrace{\frac{1}{(2\pi)^{D/2} t^D}}_{\doteq C_t} e^{-\|\xi - \mathbf{x}\|_2^2 / (2t^2)} \right] \quad (\text{B.2.69})$$

$$= C_t \mathbb{E} \left[ e^{-(\|\xi\|_2^2 - 2\langle \xi, \mathbf{x} \rangle + \|\mathbf{x}\|_2^2) / (2t^2)} \right] \quad (\text{B.2.70})$$

$$= C_t \mathbb{E} \left[ e^{-(r^2 - 2\langle \xi, \mathbf{x} \rangle + \|\mathbf{x}\|_2^2) / (2t^2)} \right] \quad (\text{B.2.71})$$

$$= C_t e^{-r^2 / (2t^2)} \mathbb{E} \left[ e^{(2\langle \xi, \mathbf{x} \rangle - \|\mathbf{x}\|_2^2) / (2t^2)} \right]. \quad (\text{B.2.72})$$

Note that because  $\|\xi\|_2 = r$ , we have by Cauchy-Schwarz that

$$-2r\|\mathbf{x}\|_2 - \|\mathbf{x}\|_2^2 \leq 2\langle \xi, \mathbf{x} \rangle - \|\mathbf{x}\|_2^2 \leq 2r\|\mathbf{x}\|_2 - \|\mathbf{x}\|_2^2. \quad (\text{B.2.73})$$

Recall that by Assumption B.1,  $\mathbf{x}$  is supported on a compact set  $\mathcal{S}$  of radius  $R$ . Thus

$$-2R(r + R) \leq 2\langle \xi, \mathbf{x} \rangle - \|\mathbf{x}\|_2^2 \leq 2Rr. \quad (\text{B.2.74})$$

In other words, it holds

$$C_t e^{-[r^2+2R(r+R)]/(2t^2)} \leq p_t(\xi) \leq C_t e^{-[r^2+2Rr]/(2t^2)}. \quad (\text{B.2.75})$$

Now to compute the gradient, we can use Proposition B.1 and linearity of expectation to compute

$$\langle \nabla p_t(\xi), \xi \rangle = \left\langle -\frac{1}{t^2} \mathbb{E}[(\xi - \mathbf{x}) \varphi_t(\xi - \mathbf{x})], \xi \right\rangle \quad (\text{B.2.76})$$

$$= -\frac{1}{t^2} \mathbb{E}[\langle \xi - \mathbf{x}, \xi \rangle \varphi_t(\xi - \mathbf{x})] \quad (\text{B.2.77})$$

$$= -\frac{1}{t^2} \mathbb{E}[(\|\xi\|_2^2 - \langle \xi, \mathbf{x} \rangle) \varphi_t(\xi - \mathbf{x})] \quad (\text{B.2.78})$$

$$= -\frac{1}{t^2} \mathbb{E}[(r^2 - \langle \xi, \mathbf{x} \rangle) \varphi_t(\xi - \mathbf{x})] \quad (\text{B.2.79})$$

$$= \frac{1}{t^2} \mathbb{E}[(\langle \xi, \mathbf{x} \rangle - r^2) \varphi_t(\xi - \mathbf{x})]. \quad (\text{B.2.80})$$

Using Cauchy-Schwarz and the representation  $p_t(\xi) \doteq \mathbb{E}[\varphi_t(\xi - \mathbf{x})]$  again, it holds

$$\frac{1}{t^2} \mathbb{E}[(-Rr - r^2) \varphi_t(\xi - \mathbf{x})] \leq \langle \nabla p_t(\xi), \xi \rangle \leq \frac{1}{t^2} \mathbb{E}[(Rr - r^2) \varphi_t(\xi - \mathbf{x})] \quad (\text{B.2.81})$$

$$\frac{1}{t^2} (-Rr - r^2) \mathbb{E}[\varphi_t(\xi - \mathbf{x})] \leq \langle \nabla p_t(\xi), \xi \rangle \leq \frac{1}{t^2} (Rr - r^2) \mathbb{E}[\varphi_t(\xi - \mathbf{x})] \quad (\text{B.2.82})$$

$$-\frac{r(R+r)}{t^2} p_t(\xi) \leq \langle \nabla p_t(\xi), \xi \rangle \leq -\frac{r(r-R)}{t^2} p_t(\xi). \quad (\text{B.2.83})$$

For  $r > R > 0$  (as is suitable, because we are going to take the limit  $r \rightarrow \infty$  while  $R$  is fixed), both sides are negative. This makes sense: most of the probability mass is contained within the ball of radius  $R$  and thus the score points inwards, having a negative inner product with the outward-pointing vector  $\xi$ . Thus using the appropriate bounds for  $p_t(\xi)$ ,

$$-\frac{r(R+r)}{t^2} \cdot C_t e^{-[r^2+2Rr]/(2t^2)} \leq \langle \nabla p_t(\xi), \xi \rangle \leq -\frac{r(r-R)}{t^2} \cdot C_t e^{-[r^2+2R(r+R)]/(2t^2)}. \quad (\text{B.2.84})$$

Then, noting that  $C_t = \text{poly}(t^{-1})$ , we can compute

$$[c + \log p_t(\xi)] \langle \nabla p_t(\xi), \xi \rangle = \text{poly}(r, R, t^{-1}, c) e^{-\Theta_r(r^2)} \quad (\text{B.2.85})$$

So one can see that, letting the surface area of  $\partial B_r(\mathbf{0})$  be  $\omega_D r^{D-1}$  where  $\omega_D$  is a function of  $D$ , it holds

$$\frac{1}{r} \int_{\partial B_r(\mathbf{0})} [c + \log p_t(\xi)] \langle \nabla p_t(\xi), \xi \rangle d\xi = \text{poly}(r, R, t^{-1}, c) e^{-\Theta_r(r^2)} \quad (\text{B.2.86})$$

and therefore the exponentially decaying tails mean

$$\lim_{r \rightarrow \infty} \frac{1}{r} \int_{\partial B_r(\mathbf{0})} [c + \log p_t(\xi)] \langle \nabla p_t(\xi), \xi \rangle d\xi = 0. \quad (\text{B.2.87})$$

Finally, plugging into (B.2.64), we have

$$\int_{\mathbb{R}^D} \{\Delta p_t(\xi)[c + \log p_t(\xi)] + \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle\} d\xi = 0 \quad (\text{B.2.88})$$

$$\implies \int_{\mathbb{R}^D} \Delta p_t(\xi)[c + \log p_t(\xi)] d\xi = - \int_{\mathbb{R}^D} \langle \nabla \log p_t(\xi), \nabla p_t(\xi) \rangle d\xi \quad (\text{B.2.89})$$

as claimed.  $\square$

### Local Invertibility of the Denoiser $\bar{x}$

Here we provide some results used in the proof of Theorem B.3 which are appropriate generalizations of corresponding results in [Gri11].

**Lemma B.3** (Generalization of [Gri11], Lemma A.1). *Let  $x$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(x_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). Let  $s, t \in [0, T]$  be such that  $0 \leq s < t \leq T$ , and let  $\bar{x}(\xi) \doteq \mathbb{E}[x_s | x_t = \xi]$ . The Jacobian  $\bar{x}'(\xi)$  is symmetric positive definite.*

*Proof.* We have

$$\bar{x}'(\xi) = \mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 \nabla^2 \log p_t(\xi). \quad (\text{B.2.90})$$

Here we expand

$$\nabla^2 \log p_t(\xi) = \frac{p_t(\xi) \nabla^2 p_t(\xi) - (\nabla p_t(\xi))(\nabla p_t(\xi))^{\top}}{p_t(\xi)^2}. \quad (\text{B.2.91})$$

So we need to ensure that

$$\bar{x}'(\xi) = \mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 \frac{p_t(\xi) \nabla^2 p_t(\xi) - (\nabla p_t(\xi))(\nabla p_t(\xi))^{\top}}{p_t(\xi)^2} \quad (\text{B.2.92})$$

$$= \frac{p_t(\xi)^2 \mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 [p_t(\xi) \nabla^2 p_t(\xi) - (\nabla p_t(\xi))(\nabla p_t(\xi))^{\top}]}{p_t(\xi)^2} \quad (\text{B.2.93})$$

is symmetric positive semidefinite. Indeed it is obviously symmetric (by Clairaut's theorem). To show its positive semidefiniteness, we plug in the expectation representation of  $p_t$  given by (B.2.3) (and  $\nabla p_t$ ,  $\Delta p_t$  by Proposition B.1) to obtain (where  $x$  is as defined and  $y$  is i.i.d. as  $x$ ),

$$\mathbf{v}^{\top} [\bar{x}'(\xi)] \mathbf{v} \quad (\text{B.2.94})$$

$$= p_t(\xi)^{-2} \mathbf{v}^{\top} \left\{ p_t(\xi)^2 \mathbf{I} + \left(1 - \frac{s}{t}\right) t^2 \mathbb{E}[\varphi_t(\xi - x)] \mathbb{E} \left[ \varphi_t(\xi - x) \cdot \frac{(\xi - x)(\xi - x)^{\top} - t^2 \mathbf{I}}{t^4} \right] \right\} \quad (\text{B.2.95})$$

$$- \left(1 - \frac{s}{t}\right) t^2 \mathbb{E} \left[ -\varphi_t(\xi - x) \cdot \frac{\xi - x}{t^2} \right] \mathbb{E} \left[ -\varphi_t(\xi - x) \cdot \frac{\xi - x}{t^2} \right]^{\top} \} \mathbf{v} \quad (\text{B.2.96})$$

$$= p_t(\xi)^{-2} \mathbf{v}^{\top} \left\{ \mathbb{E}[\varphi_t(\xi - x) \varphi_t(\xi - y) \mathbf{I}] \right\} \quad (\text{B.2.97})$$

$$+ \left(1 - \frac{s}{t}\right) t^2 \mathbb{E} \left[ \varphi_t(\xi - x) \varphi_t(\xi - y) \cdot \frac{(\xi - y)(\xi - y)^{\top} - t^2 \mathbf{I}}{t^4} \right] \quad (\text{B.2.98})$$

$$- \left(1 - \frac{s}{t}\right) t^2 \mathbb{E} \left[ \varphi_t(\xi - x) \varphi_t(\xi - y) \cdot \frac{(\xi - x)(\xi - y)^{\top}}{t^4} \right] \} \quad (\text{B.2.99})$$

$$= \frac{1 - s/t}{p_t(\xi)^2} \mathbf{v}^{\top} \mathbb{E} \left[ \varphi_t(\xi - x) \varphi_t(\xi - y) \left\{ \frac{1}{1 - s/t} \mathbf{I} + \frac{(\xi - y)(\xi - y)^{\top}}{t^2} - \mathbf{I} - \frac{(\xi - x)(\xi - y)^{\top}}{t^2} \right\} \right] \mathbf{v} \quad (\text{B.2.100})$$

$$= \frac{t - s}{tp_t(\xi)^2} \mathbf{v}^{\top} \mathbb{E} \left[ \frac{s}{t - s} \mathbf{I} + \frac{(\xi - x)(\xi - x)^{\top}}{2t^2} + \frac{(\xi - y)(\xi - y)^{\top}}{2t^2} - \frac{(\xi - x)(\xi - y)^{\top}}{t^2} \right] \mathbf{v} \quad (\text{B.2.101})$$

$$= \frac{t - s}{tp_t(\xi)^2} \mathbf{v}^{\top} \mathbb{E} \left[ \frac{s}{t - s} \mathbf{I} + \frac{1}{2t^2} ((\xi - x)(\xi - x)^{\top} + (\xi - y)(\xi - y)^{\top} - 2(\xi - x)(\xi - y)^{\top}) \right] \mathbf{v} \quad (\text{B.2.102})$$

$$= \frac{t - s}{tp_t(\xi)^2} \mathbb{E} \left[ \frac{s}{t - s} \|\mathbf{v}\|_2^2 + \frac{1}{2t^2} ([(\xi - x)^{\top} \mathbf{v}]^2 + [(\xi - y)^{\top} \mathbf{v}]^2 - 2[(\xi - x)^{\top} \mathbf{v}] [(\xi - y)^{\top} \mathbf{v}]) \right] \quad (\text{B.2.103})$$

$$= \frac{t - s}{tp_t(\xi)^2} \mathbb{E} \left[ \frac{s}{t - s} \|\mathbf{v}\|_2^2 + \frac{1}{2t^2} ([(\xi - x)^{\top} \mathbf{v}]^2 + [(\xi - y)^{\top} \mathbf{v}]^2 - 2[(\xi - x)^{\top} \mathbf{v}] [(\xi - y)^{\top} \mathbf{v}]) \right] \quad (\text{B.2.104})$$

$$= \frac{t - s}{tp_t(\xi)^2} \mathbb{E} \left[ \frac{s}{t - s} \|\mathbf{v}\|_2^2 + \frac{1}{2t^2} ([(\xi - x)^{\top} \mathbf{v}] - [(\xi - y)^{\top} \mathbf{v}])^2 \right] \quad (\text{B.2.105})$$

$$= \frac{t-s}{tp_t(\xi)^2} \mathbb{E} \left[ \frac{s}{t-s} \|\mathbf{v}\|_2^2 + \frac{1}{2t^2} [(\mathbf{y} - \mathbf{x})^\top \mathbf{v}]^2 \right] \quad (\text{B.2.106})$$

$$= \frac{s}{tp_t(\xi)^2} \|\mathbf{v}\|_2^2 + \frac{t-s}{2t^3 p_t(\xi)} \mathbb{E}[\{(\mathbf{y} - \mathbf{x})^\top \mathbf{v}\}^2] \quad (\text{B.2.107})$$

Since  $\mathbf{x}$  and  $\mathbf{y}$  are i.i.d., the whole integral (i.e., the original quadratic form) is 0 if and only if  $s = 0$  and  $\mathbf{x}$  has support entirely contained in an affine subspace which is orthogonal to  $\mathbf{v}$ . But this is ruled out by assumption (i.e., that  $\mathbf{x}$  has a density on  $\mathbb{R}^D$ ), so the Jacobian  $\bar{\mathbf{x}}'(\xi)$  is symmetric positive definite.  $\square$

**Lemma B.4** (Generalization of [Gri11] Corollary A.2, Part 1). *Let  $f: \mathbb{R}^D \rightarrow \mathbb{R}^D$  be any differentiable function whose Jacobian  $f'(\mathbf{x})$  is symmetric positive definite. Then  $f$  is injective, and hence invertible as a function  $\mathbb{R}^D \rightarrow \mathcal{R}(f)$  where  $\mathcal{R}(f)$  is the range of  $f$ .*

*Proof.* Suppose that  $f$  were not injective, i.e., there exists  $\mathbf{x}, \mathbf{x}'$  such that  $f(\mathbf{x}) = f(\mathbf{x}')$  while  $\mathbf{x} \neq \mathbf{x}'$ . Define  $\mathbf{v} \doteq (\mathbf{x}' - \mathbf{x})/\|\mathbf{x}' - \mathbf{x}\|_2$ . Define the function  $g: \mathbb{R} \rightarrow \mathbb{R}$  as  $g(t) \doteq \mathbf{v}^\top f(\mathbf{x} + t\mathbf{v})$ . Then  $g(0) = \mathbf{v}^\top f(\mathbf{x}) = \mathbf{v}^\top f(\mathbf{x}') = g(\|\mathbf{x}' - \mathbf{x}\|_2)$ . Since  $f$  is differentiable,  $g$  is differentiable, so the derivative  $g'$  must vanish for some  $t^* \in (0, \|\mathbf{x}' - \mathbf{x}\|_2)$  by the mean value theorem. However,

$$g'(t^*) \doteq \mathbf{v}^\top [f'(\mathbf{x} + t^*\mathbf{v})] \mathbf{v} > 0 \quad (\text{B.2.108})$$

since the Jacobian is positive definite. Thus we arrive at a contradiction, as claimed.  $\square$

Combining the above two results, we obtain the following crucial result.

**Corollary B.1** (Generalization of [Gri11] Corollary A.2, Part 2). *Let  $\mathbf{x}$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). Let  $s, t \in [0, T]$  be such that  $0 \leq s < t \leq T$ , and let  $\bar{\mathbf{x}}(\xi) \doteq \mathbb{E}[\mathbf{x}_s \mid \mathbf{x}_t = \xi]$ . Then  $\bar{\mathbf{x}}$  is injective, and therefore invertible onto its range.*

*Proof.* The only thing left to show is that  $\bar{\mathbf{x}}$  is differentiable, but this is immediate from Tweedie's formula (Theorem 3.3) which shows that  $\bar{\mathbf{x}}$  is differentiable if and only if  $\nabla \log p_t$  is differentiable, and this is provided by Equation (B.2.3).  $\square$

### Controlling the Laplacian $\Delta \log p_t$

Finally, we develop a technical estimate which is required for the proof of Theorem B.3 and actually motivates the assumption for the viable  $t$ .

**Lemma B.5.** *Let  $\mathbf{x}$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). Let  $p_t$  be the density of  $\mathbf{x}_t$ . Then, for  $t > 0$  it holds*

$$\sup_{\xi \in \mathbb{R}^D} |\Delta \log p_t(\xi)| \leq \max \left( \frac{D}{t^2}, \left| \frac{R}{t^4} - \frac{D}{t^2} \right| \right). \quad (\text{B.2.109})$$

*Proof.* By chain rule, a simple exercise computes

$$\Delta \log p_t(\xi) = \frac{\Delta p_t(\xi)}{p_t(\xi)} - \frac{\|\nabla p_t(\xi)\|_2^2}{p_t(\xi)^2}. \quad (\text{B.2.110})$$

Using Proposition B.1 to write the terms in  $\Delta p_t(\xi)$ , we obtain

$$\frac{\Delta p_t(\xi)}{p_t(\xi)} = \frac{\mathbb{E} \left[ \frac{\|\xi - \mathbf{x}\|_2^2 - Dt^2}{t^4} \cdot \varphi_t(\xi - \mathbf{x}) \right]}{\mathbb{E}[\varphi_t(\xi - \mathbf{x})]} \quad (\text{B.2.111})$$

$$= \frac{\int_{\mathbb{R}^D} \left\{ \frac{\|\xi - \mathbf{u}\|_2^2 - Dt^2}{t^4} \right\} \varphi_t(\xi - \mathbf{u}) p(\mathbf{u}) d\mathbf{u}}{\int_{\mathbb{R}^D} \varphi_t(\xi - \mathbf{u}) p(\mathbf{u}) d\mathbf{u}}. \quad (\text{B.2.112})$$

This looks like a Bayesian marginalization, so let us define the appropriate normalized density

$$q_{\xi}(\mathbf{u}) = \frac{\varphi_t(\xi - \mathbf{u})p(\mathbf{u})}{\int_{\mathbb{R}^D} \varphi_t(\xi - \mathbf{v})p(\mathbf{v})d\mathbf{v}} = \frac{\varphi_t(\xi - \mathbf{u})p(\mathbf{u})}{\mathbb{E}[\varphi_t(\xi - \mathbf{x})]} = \frac{\varphi_t(\xi - \mathbf{u})p(\mathbf{u})}{p_t(\xi)} \quad (\text{B.2.113})$$

Then, defining  $\mathbf{y}_{\xi} \sim q_{\xi}$ , we can write

$$\frac{\Delta p_t(\xi)}{p_t(\xi)} = \int_{\mathbb{R}^D} \left\{ \frac{\|\xi - \mathbf{u}\|_2^2 - Dt^2}{t^4} \right\} q_{\xi}(\mathbf{u})d\mathbf{u} = \frac{1}{t^4} \mathbb{E}[\|\xi - \mathbf{y}_{\xi}\|_2^2] - \frac{D}{t^2}. \quad (\text{B.2.114})$$

Similarly, writing out the second term (non-squared) we obtain

$$\frac{\nabla p_t(\xi)}{p_t(\xi)} = -\frac{\xi - \mathbb{E}[\mathbf{y}_{\xi}]}{t^2}. \quad (\text{B.2.115})$$

Letting  $\mathbf{z}_{\xi} \doteq \mathbf{y}_{\xi} - \xi$ , it holds

$$\frac{\Delta p_t(\xi)}{p_t(\xi)} = \frac{\mathbb{E}[\|\mathbf{z}_{\xi}\|_2^2]}{t^4} - \frac{D}{t^2}, \quad \frac{\nabla p_t(\xi)}{p_t(\xi)} = \frac{\mathbb{E}[\mathbf{z}_{\xi}]}{t^2}. \quad (\text{B.2.116})$$

Thus writing  $\Delta \log p_t$  out fully, we have

$$\Delta \log p_t(\xi) = \frac{\mathbb{E}[\|\mathbf{z}_{\xi}\|_2^2]}{t^4} - \frac{D}{t^2} - \frac{\|\mathbb{E}[\mathbf{z}_{\xi}]\|_2^2}{t^4} \quad (\text{B.2.117})$$

$$= \frac{\mathbb{E}[\|\mathbf{z}_{\xi}\|_2^2] - \|\mathbb{E}[\mathbf{z}_{\xi}]\|_2^2}{t^4} - \frac{D}{t^2} \quad (\text{B.2.118})$$

$$= \frac{\text{tr}(\text{Cov}(\mathbf{z}_{\xi}))}{t^4} - \frac{D}{t^2} \quad (\text{B.2.119})$$

$$= \frac{\text{tr}(\text{Cov}(\mathbf{y}_{\xi}))}{t^4} - \frac{D}{t^2}. \quad (\text{B.2.120})$$

A trivial lower bound on this trace is 0, since covariance matrices are positive semidefinite. To find an upper bound, note that  $\mathbf{y}_{\xi}$  takes values only in the support of  $\mathbf{x}$  (since  $p$  is a factor of the density  $q_{\xi}$  of  $\mathbf{y}_{\xi}$ ), which by Assumption B.1 is a compact set  $\mathcal{S}$  with radius  $R \doteq \sup_{\xi \in \mathbb{R}^D} \|\xi\|_2$ . So

$$\text{tr}(\text{Cov}(\mathbf{y}_{\xi})) = \mathbb{E}[\|\mathbf{y}_{\xi}\|_2^2] - \|\mathbb{E}[\mathbf{y}_{\xi}]\|_2^2 \leq \mathbb{E}[\|\mathbf{y}_{\xi}\|_2^2] \leq R^2. \quad (\text{B.2.121})$$

Therefore

$$-\frac{D}{t^2} \leq \Delta \log p_t(\xi) \leq \frac{R^2}{t^4} - \frac{D}{t^2}, \quad (\text{B.2.122})$$

which shows the claim.  $\square$

### Derivative Computations

Here we calculate some useful derivatives which will be reused throughout the appendix.

**Proposition B.1.** *Let  $\mathbf{x}$  be any random variable such that Assumptions B.1 and B.2 hold, and let  $(\mathbf{x}_t)_{t \in [0, T]}$  be the stochastic process (B.2.1). For  $t \geq 0$ , let  $p_t$  be the density of  $\mathbf{x}_t$ . Then*

$$\frac{\partial p_t}{\partial t}(\xi) = \mathbb{E} \left[ \varphi_t(\xi - \mathbf{x}) \cdot \frac{\|\xi - \mathbf{x}\|_2^2 - Dt^2}{t^3} \right] \quad (\text{B.2.123})$$

$$\nabla p_t(\xi) = -\mathbb{E} \left[ \varphi_t(\xi - \mathbf{x}) \cdot \frac{\xi - \mathbf{x}}{t^2} \right] \quad (\text{B.2.124})$$

$$\nabla^2 p_t(\xi) = \mathbb{E} \left[ \varphi_t(\xi - \mathbf{x}) \cdot \frac{(\xi - \mathbf{x})(\xi - \mathbf{x})^\top - t^2 \mathbf{I}}{t^4} \right] \quad (\text{B.2.125})$$

$$\Delta p_t(\xi) = \mathbb{E} \left[ \varphi_t(\xi - \mathbf{x}) \cdot \frac{\|\xi - \mathbf{x}\|_2^2 - Dt^2}{t^4} \right]. \quad (\text{B.2.126})$$

*Proof.* We use the convolution representation of  $p_t$ , namely (B.2.3). First taking the time derivative, a computation yields that Proposition B.3 applies,<sup>5</sup> so we can bring the derivative inside the integral/expectation as:

$$\frac{\partial p_t}{\partial t}(\boldsymbol{\xi}) = \frac{\partial}{\partial t} \mathbb{E}[\varphi_t(\boldsymbol{\xi} - \mathbf{x})] = \mathbb{E}\left[\frac{\partial}{\partial t} \varphi_t(\boldsymbol{\xi} - \mathbf{x})\right] = \frac{\partial \varphi_t}{\partial t} * p. \quad (\text{B.2.127})$$

Meanwhile, by properties of convolutions (Proposition B.4) and using the fact that  $p$  is compactly supported (Assumption B.1),

$$p_t = \varphi_t * p \implies \nabla p_t = \nabla \varphi_t * p \implies \nabla^2 p_t = \nabla^2 \varphi_t * p \implies \Delta p_t = \Delta \varphi_t * p. \quad (\text{B.2.128})$$

The rest of the computation follows from Proposition B.2.  $\square$

**Proposition B.2.** *For  $t > 0$  and  $\boldsymbol{\xi} \in \mathbb{R}^D$  it holds*

$$\frac{\partial}{\partial t} \varphi_t(\boldsymbol{\xi}) = \varphi_t(\boldsymbol{\xi}) \cdot \frac{\|\boldsymbol{\xi}\|_2^2 - Dt^2}{t^3} \quad (\text{B.2.129})$$

$$\nabla \varphi_t(\boldsymbol{\xi}) = -\varphi_t(\boldsymbol{\xi}) \cdot \frac{\boldsymbol{\xi}}{t^2} \quad (\text{B.2.130})$$

$$\nabla^2 \varphi_t(\boldsymbol{\xi}) = \varphi_t(\boldsymbol{\xi}) \cdot \frac{\boldsymbol{\xi} \boldsymbol{\xi}^\top - t^2 \mathbf{I}}{t^4} \quad (\text{B.2.131})$$

$$\Delta \varphi_t(\boldsymbol{\xi}) = \varphi_t(\boldsymbol{\xi}) \cdot \frac{\|\boldsymbol{\xi}\|_2^2 - Dt^2}{t^4}. \quad (\text{B.2.132})$$

*Proof.* Direct computation.  $\square$

### Differentiating Under the Integral Sign

In this appendix, we differentiate under the integral sign many times, and it is important to know when we can do this. There are two kinds of differentiating under the integral sign:

1. Differentiating an integral  $\int f_t(\boldsymbol{\xi}) d\boldsymbol{\xi}$  with respect to the auxiliary parameter  $t$ .
2. Differentiating a convolution  $(f * g)(\boldsymbol{\xi}) = \int f(\boldsymbol{\xi})g(\boldsymbol{\xi} - \mathbf{u}) du$  with respect to the variable  $\boldsymbol{\xi}$ .

For the first category, we give a concrete result, stated without proof but attributable to [the linked source](#), which derives the following result as a special case of a more general theorem about the interaction of differential operators and tempered distributions, much beyond the scope of the book. A full formal reference can be found in [Jon82].

**Proposition B.3** ([Jon82], Section 11.12). *Let  $f: (0, T) \times \mathbb{R}^D \rightarrow \mathbb{R}$  be such that:*

- *$f$  is a jointly measurable function of  $(t, \boldsymbol{\xi})$ ;*
- *For Lebesgue-almost every  $\boldsymbol{\xi} \in \mathbb{R}^D$ , the function  $t \mapsto f_t(\boldsymbol{\xi})$  is absolutely continuous;*
- *$\frac{\partial f_t}{\partial t}$  is locally integrable, i.e., for every  $[t_{\min}, t_{\max}] \subseteq (0, T)$  it holds*

$$\int_{t_{\min}}^{t_{\max}} \int_{\mathbb{R}^D} \left| \frac{\partial f_t}{\partial t}(\boldsymbol{\xi}) \right| d\boldsymbol{\xi} dt < \infty. \quad (\text{B.2.133})$$

*Then  $t \mapsto \int_{\mathbb{R}^D} f_t(\boldsymbol{\xi}) d\boldsymbol{\xi}$  is an absolutely continuous function on  $(0, T)$ , and its derivative is*

$$\frac{d}{dt} \int_{\mathbb{R}^D} f_t(\boldsymbol{\xi}) d\boldsymbol{\xi} = \int_{\mathbb{R}^D} \frac{\partial}{\partial t} f_t(\boldsymbol{\xi}) d\boldsymbol{\xi}, \quad (\text{B.2.134})$$

*defined for almost every  $t \in (0, T)$ .*

---

<sup>5</sup>We use  $f_t(\boldsymbol{\xi}) = p(\boldsymbol{\xi})\varphi_t(\boldsymbol{\xi} - \mathbf{x})$ , noting that it is twice continuously differentiable in  $\boldsymbol{\xi}$  and (more than) twice continuously differentiable in  $t$ . Then to check the local integrability of  $f_t$  we compute  $\frac{\partial f_t}{\partial t}(\boldsymbol{\xi}) = f_t(\boldsymbol{\xi}) \cdot \frac{1}{t^3}(\|\boldsymbol{\xi} - \mathbf{x}\|_2^2 - Dt^2)$ , which is is easy to check integrable over  $\boldsymbol{\xi}$  and  $t \in [t_{\min}, t_{\max}]$  where  $t_{\min} > 0$ . (Indeed,  $f_t$  has exponentially decaying tails, so the quadratic term in the product is of no issue.)

For the second category, we give another concrete result, stated without proof but fully formalized in [BB11].

**Proposition B.4** ([BB11], Proposition 4.20). *Let  $f$  be  $k$ -times continuously differentiable with compact support, and let  $g$  be locally integrable. Then the convolution  $f * g$  defined by*

$$(f * g)(\boldsymbol{\xi}) \doteq \int_{\mathbb{R}^D} f(\mathbf{u})g(\boldsymbol{\xi} - \mathbf{u})d\mathbf{u} \quad (\text{B.2.135})$$

*is  $k$ -times continuously differentiable, and its derivative of order  $k$  is*

$$\nabla^k(f * g) = (\nabla^k f) * g. \quad (\text{B.2.136})$$

Although not in the book, a simple integration by parts argument shows that if  $g$  is also  $k$ -times differentiable, then we can “trade off” the regularity:

$$\nabla^k(f * g) = f * (\nabla^k g). \quad (\text{B.2.137})$$

## B.3 Lossy Coding and Sphere Packing

In this section, we prove Theorem 3.6. Following our conventions throughout this appendix, we write  $\mathcal{S} = \text{Supp}(\mathbf{x})$  for the compact support of the random variable  $\mathbf{x}$ .

As foreshadowed, we will make a regularity assumption on the support set  $\mathcal{S}$  to prove Theorem 3.6. One possibility for proceeding under minimal assumptions would be to instantiate the results of [RBK18; RKB23] in our setting, since these results apply to sets  $\mathcal{S}$  with very low regularity (e.g., Cantor-like sets with fractal structure). However, we have found precisely computing the constants in these results, a necessary endeavor to assert a conclusion like Theorem 3.6, to be somewhat onerous in our setting. Our approach is therefore to add a geometric regularity assumption on the set  $\mathcal{S}$  that sacrifices some generality, but allows us to develop a more transparent argument. To avoid sacrificing too much generality, we must ensure that low-dimensionality in the set  $\mathcal{S}$  is not prohibited. We therefore consider the running example we have used throughout the book, the mixture of low-rank Gaussian distributions. In this geometric setting, we will enforce this via assuming that  $\mathcal{S}$  is a union of hyperspheres, which is equivalent to the Gaussian assumption in high dimensions with overwhelming probability.

**Assumption B.3.** The support  $\mathcal{S} \subset \mathbb{R}^D$  of the random variable  $\mathbf{x}$  is a finite union of  $K$  spheres, each with dimension  $d_k$ ,  $k \in [K]$ . The probability that  $\mathbf{x}$  is drawn from the  $k$ -th sphere is given by  $\pi_k \in [0, 1]$ , and conditional on being drawn from the  $k$ -th sphere,  $\mathbf{x}$  is uniformly distributed on that sphere. The supports satisfy that each sphere is mutually orthogonal with all others.

We proceed under the simplifying Assumption B.3 in order to simplify excessive technicality, and to connect to an important running example used throughout the monograph. We believe our results can be generalized to support  $\mathcal{S}$  from the class of *sets with positive reach* with additional technical effort, but leave this for the future.

### B.3.1 Proof of Relationship Between Rate Distortion and Covering

We briefly sketch the proof, then proceed to establishing three fundamental lemmas, then give the proof. The proof will depend on notions introduced in the sketch below.

Obtaining an upper bound on the rate distortion function (3.3.3) is straightforward: by the rate characterization (i.e., the rate distortion function is the minimum rate of a code for  $\mathbf{x}$  with expected squared  $\ell^2$  distortion  $\epsilon$ ), upper bounding  $\mathcal{R}_\epsilon(\mathbf{x})$  only requires demonstrating one code for  $\mathbf{x}$  that achieves this target distortion, and any  $\epsilon$ -covering of  $\text{Supp}(\mathbf{x})$  achieves this, with rate equal to the base-2 logarithm of the cardinality of the covering. The lower bound is more subtle. We make use of the Shannon lower bound, discussed in Remark 3.7: working out the constants in [LZ94, §III, (22)] gives a more precise version of the

result quoted in Equation (3.3.10) (in bits, of course): for any random variable  $\mathbf{x}$  with compact support and a density, it holds

$$\mathcal{R}_\epsilon(\mathbf{x}) \geq h(\mathbf{x}) - \log \text{vol}(B_\epsilon) + \log \left( \frac{2}{D\Gamma(D/2)} \left( \frac{D}{2e} \right)^{D/2} \right), \quad (\text{B.3.1})$$

with entropy (etc.) in nats in this expression. The constant can be easily estimated using Stirling's approximation. A quantitative form of Stirling's approximation which is often useful gives for any  $x > 0$  [Jam15]

$$\Gamma(x) \leq \sqrt{2\pi} x^{x-1/2} e^{-x} e^{1/(12x)}. \quad (\text{B.3.2})$$

We will apply this bound to  $\Gamma(D/2)$  in Equation (B.3.1). We get

$$\log \left( \frac{2}{D\Gamma(D/2)} \left( \frac{D}{2e} \right)^{D/2} \right) \geq -\frac{1}{6D} + \log \left( \frac{2}{D} \left( \frac{D}{2e} \right)^{D/2} \cdot \sqrt{\frac{D}{4\pi}} \left( \frac{D}{2e} \right)^{-D/2} \right) \quad (\text{B.3.3})$$

$$= -\frac{1}{6D} - \frac{1}{2} \log D\pi, \quad (\text{B.3.4})$$

which we can take for the explicit value of the constant  $C_D$  in Equation (3.3.10). Summarizing the fully quantified Shannon lower bound (in bits):

$$\mathcal{R}_\epsilon(\mathbf{x}) \geq h(\mathbf{x}) - \log_2 \text{vol}(B_\epsilon) - O(\log D). \quad (\text{B.3.5})$$

Now, the important constraint for our current purposes is that the Shannon lower bound requires the random variable  $\mathbf{x}$  to have a density, which rules out many low-dimensional distributions of interest. But let us momentarily consider the situation when  $\mathbf{x}$  does admit a density. The assumption that  $\mathbf{x}$  is uniformly distributed on its support is easily formalized in this setting: for any Borel set  $A \subset \mathcal{S}$ , we have

$$\mathbb{P}[\mathbf{x} \in A] = \int_A \frac{1}{\text{vol}(\mathcal{S})} d\mathbf{x}. \quad (\text{B.3.6})$$

Then the entropy  $h(\mathbf{x})$  is just

$$h(\mathbf{x}) = \log_2 \text{vol}(\mathcal{S}). \quad (\text{B.3.7})$$

The proof then concludes with a lemma that relates the ratio  $\text{vol}(\mathcal{S}) / \text{vol}(B_\epsilon)$  to the  $\epsilon$ -covering number of  $\mathcal{S}$  by  $\epsilon$  balls.

To extend the program above to degenerate distributions satisfying Assumption B.3, our proof of the lower bound in Theorem 3.6 will leverage an approximation argument of the actual low-dimensional distribution  $\mathbf{x}$  by “nearby” distributions which have densities, similarly but not exactly the same as the proof sketch preceding Theorem B.1. We will then link the parameter introduced in the approximating sequence to the distortion parameter  $\epsilon$  in order to obtain the desired conclusion in Theorem 3.6.

**Definition B.1.** Let  $\mathcal{S}$  be a compact set. For any  $\delta > 0$ , define the  $\delta$ -thickening of  $\mathcal{S}$ , denoted  $\mathcal{S}_\delta$ , by

$$\mathcal{S}_\delta = \{\boldsymbol{\xi} \in \mathbb{R}^D \mid \text{dist}(\boldsymbol{\xi}, \mathcal{S}) \leq \delta\}. \quad (\text{B.3.8})$$

The distance function referenced in Definition B.1 is defined by

$$\text{dist}(\boldsymbol{\xi}, \mathcal{S}) = \inf_{\boldsymbol{\xi}' \in \mathcal{S}} \|\boldsymbol{\xi} - \boldsymbol{\xi}'\|_2. \quad (\text{B.3.9})$$

For a compact set  $\mathcal{S}$ , Weierstrass's theorem implies that for any  $\boldsymbol{\xi} \in \mathbb{R}^D$ , there is always some  $\boldsymbol{\xi}' \in \mathcal{S}$  attaining the infimum in the distance function. Compactness of  $\mathcal{S}_\delta$  follows readily from compactness of  $\mathcal{S}$ , so  $\text{vol}(\mathcal{S}_\delta)$  is finite for any  $\delta > 0$ . It is then possible to make the following definition of a thickened random variable, specialized to Assumption B.3.

**Definition B.2.** Let  $\mathbf{x}$  be a random variable such that  $\text{Supp}(\mathbf{x}) = \mathcal{S}$  is a union of  $K$  hyperspheres, distributed as in Assumption B.3. Denote the support of each component of the mixture by  $\mathcal{S}_k$ . Define the thickened random variable  $\mathbf{x}_\delta$  as the mixture of measures where each component measure is uniform on the thickened set  $\mathcal{S}_{k,\delta}$  (Definition B.1), for  $k \in [K]$ , with mixing weights  $\pi_k$ .

**Lemma B.6.** Suppose the random variable  $\mathbf{x}$  satisfies Assumption B.3. Then if  $0 < \delta < \frac{1}{2}$ , the thickened random variable  $\mathbf{x}_\delta$  (Definition B.2) satisfies for any  $\epsilon > 0$

$$R_{\delta+\epsilon}(\mathbf{x}_\delta) \leq R_\epsilon(\mathbf{x}). \quad (\text{B.3.10})$$

The proof of Lemma B.6 is deferred to Section B.3.2. Using Lemma B.6, the above program can be realized, because the random variable  $\mathbf{x}_\delta$  has a density that is uniform with respect to the Lebesgue measure.

(*Proof of Theorem 3.6*). The upper bound is readily shown. If  $S$  is any  $\epsilon$ -cover of the support of  $\mathbf{x}$  with cardinality  $\mathcal{N}_\epsilon(\text{Supp}(\mathbf{x}))$ , then consider the coding scheme assigning to each  $\xi \in \text{Supp}(\mathbf{x})$  the reconstruction  $\hat{\xi} = \arg \min_{\xi' \in S} \|\xi - \xi'\|_2$ , with ties broken arbitrarily. Then ties occur with probability zero, and the fact that  $S$  covers  $\text{Supp}(\mathbf{x})$  at scale  $\epsilon$  guarantees distortion no larger than  $\epsilon$ ; the rate of this scheme is  $\log_2 \mathcal{N}_\epsilon(\text{Supp}(\mathbf{x}))$ .

For the lower bound, let  $0 < \delta < \frac{1}{2}$ , and consider the thickened random variable  $\mathbf{x}_\delta$ . By Lemma B.6, we have

$$R_{\delta+\epsilon}(\mathbf{x}_\delta) \leq R_\epsilon(\mathbf{x}). \quad (\text{B.3.11})$$

Since  $\mathbf{x}_\delta$  has a Lebesgue density that is uniform, we can then apply the Shannon lower bound, in the form (B.3.5), to get

$$\log_2 \text{vol}(\text{Supp}(\mathbf{x}_\delta)) - \log_2 \text{vol}(B_{\delta+\epsilon}) - O(\log D) \leq R_\epsilon(\mathbf{x}). \quad (\text{B.3.12})$$

Finally, we need to lower bound the ratio

$$\frac{\text{vol}(\text{Supp}(\mathbf{x}_\delta))}{\text{vol}(B_{\delta+\epsilon})} \quad (\text{B.3.13})$$

in terms of the covering number. Since  $\text{Supp}(\mathbf{x}_\delta) = \text{Supp}(\mathbf{x}) + B_\delta$ , where  $+$  here denotes the Minkowski sum, a standard application of volume bound arguments (see e.g. [Ver18, Proposition 4.2.12]) gives

$$\text{vol}(\text{Supp}(\mathbf{x}_\delta)) \geq \mathcal{N}_{2\delta}(\text{Supp}(\mathbf{x})) \text{vol}(B_\delta). \quad (\text{B.3.14})$$

Hence

$$\frac{\text{vol}(\text{Supp}(\mathbf{x}_\delta))}{\text{vol}(B_{\delta+\epsilon})} \geq \mathcal{N}_{2\delta}(\text{Supp}(\mathbf{x})) \frac{\text{vol}(B_\delta)}{\text{vol}(B_{\delta+\epsilon})} \quad (\text{B.3.15})$$

$$= \mathcal{N}_{2\delta}(\text{Supp}(\mathbf{x})) \left( \frac{\delta}{\delta + \epsilon} \right)^D. \quad (\text{B.3.16})$$

Choosing  $\delta = \epsilon/2$  gives from the Shannon lower bound (B.3.12) and the above estimates

$$\log_2 \mathcal{N}_\epsilon(\text{Supp}(\mathbf{x})) - O(D) \leq R_\epsilon(\mathbf{x}), \quad (\text{B.3.17})$$

as was to be shown. □

### B.3.2 Proof of Lemma B.6

(*Proof of Lemma B.6*). It suffices to show that any code for  $\mathbf{x}$  with expected squared distortion  $\epsilon^2$  produces a code for  $\mathbf{x}_\delta$  with the same rate and distortion not much larger, for a suitable choice of  $\delta$ . So fix such a code for  $\mathbf{x}$ , achieving rate  $R$  and expected squared distortion  $\epsilon^2$ . We write  $\hat{\mathbf{x}}$  for the reconstructed random variable using this code, and  $q : \text{Supp}(\mathbf{x}) \rightarrow \text{Supp}(\mathbf{x})$  for the associated encoding-decoding mapping (i.e.,  $\hat{\mathbf{x}} = q(\mathbf{x})$ ).

Now let  $\mathcal{S}_k$  denote the  $k$ -th hypersphere in the support of  $\mathbf{x}$ . There is an orthonormal basis  $\mathbf{U}_k \in \mathbb{R}^{D \times d_k}$  such that  $\text{Span}(\mathcal{S}_k) = \text{Span}(\mathbf{U}_k)$ . The following orthogonal decomposition of the support set  $\mathcal{S}$  will be used repeatedly throughout the proof. We have

$$\mathcal{S}_\delta = \{\xi \in \mathbb{R}^D \mid \exists k \in [K] : \text{dist}(\xi, \mathcal{S}_k) \leq \delta\} \quad (\text{B.3.18})$$

$$= \bigcup_{k \in [K]} \{\xi \in \mathbb{R}^D \mid \text{dist}(\xi, \mathcal{S}_k) \leq \delta\}. \quad (\text{B.3.19})$$

By orthogonal projection, for any  $k \in [K]$  any  $\xi \in \mathbb{R}^D$  can be written as  $\xi = \xi^\parallel + \xi^\perp$ , with  $\xi^\parallel \in \text{Span}(\mathcal{S}_k)$  and  $\langle \xi^\parallel, \xi^\perp \rangle = 0$ . Then for any  $\xi' \in \mathcal{S}_k$ , we have

$$\|\xi - \xi'\|_2^2 = \left\| \xi^\parallel + \xi^\perp - \xi' \right\|_2^2 = \left\| \xi^\parallel \right\|_2^2 + \left\| \xi^\perp \right\|_2^2 + \left\| \xi' \right\|_2^2 - 2 \langle \xi^\parallel, \xi' \rangle \quad (\text{B.3.20})$$

$$\geq \left\| \xi^\parallel \right\|_2^2 + \left\| \xi' \right\|_2^2 - 2 \langle \xi^\parallel, \xi' \rangle \quad (\text{B.3.21})$$

$$= \left\| \xi^\parallel - \xi' \right\|_2^2. \quad (\text{B.3.22})$$

Further, it is known that for any nonzero  $\xi^\parallel \in \text{Span}(\mathcal{S}_k)$ ,

$$\inf_{\xi' \in \mathcal{S}_k} \left\| \xi^\parallel - \xi' \right\|_2^2 = \left\| \xi^\parallel - \frac{\xi^\parallel}{\|\xi^\parallel\|_2} \right\|_2^2. \quad (\text{B.3.23})$$

If  $\xi^\parallel$  is zero, it is clear that the above distance is equal to 1 for every  $\xi' \in \mathcal{S}_k$ . Hence, if we define a projection mapping  $\pi_{\mathcal{S}_k}(\xi)$  by

$$\pi_{\mathcal{S}_k}(\xi) = \frac{\mathbf{U}_k \mathbf{U}_k^\top \xi}{\|\mathbf{U}_k^\top \xi\|_2} \quad (\text{B.3.24})$$

for any  $\xi \in \mathbb{R}^D$  with  $\mathbf{U}_k^\top \xi \neq \mathbf{0}$ , then  $\pi_{\mathcal{S}_k}(\xi) = \arg \min_{\xi' \in \mathcal{S}_k} \|\xi' - \xi\|_2$ . We choose  $0 < \delta < 1$ , so that the thickened set  $\mathcal{S}_\delta$  contains no points  $\xi \in \mathbb{R}^D$  at which any of the projection maps  $\pi_{\mathcal{S}_k}$  is not well-defined. So the thickened set  $\mathcal{S}_\delta$  satisfies

$$\mathcal{S}_\delta = \bigcup_{k \in [K]} \left\{ \xi \in \mathbb{R}^D \mid \left\| \xi - \frac{\mathbf{U}_k \mathbf{U}_k^\top \xi}{\|\mathbf{U}_k^\top \xi\|_2} \right\|_2 \leq \delta \right\}. \quad (\text{B.3.25})$$

These distances can be rewritten in terms of the orthogonal decomposition as

$$\left\| \xi - \frac{\mathbf{U}_k \mathbf{U}_k^\top \xi}{\|\mathbf{U}_k^\top \xi\|_2} \right\|_2^2 = \|\xi\|_2^2 - 2\|\mathbf{U}_k^\top \xi\|_2 + 1 \quad (\text{B.3.26})$$

$$= \left\| \xi^\parallel \right\|_2^2 + \left\| \xi^\perp \right\|_2^2 - 2\|\mathbf{U}_k^\top \xi^\parallel\|_2 + 1 \quad (\text{B.3.27})$$

$$= \left\| \xi^\perp \right\|_2^2 + \left( \left\| \xi^\parallel \right\|_2 - 1 \right)^2. \quad (\text{B.3.28})$$

We are going to show next that every such  $\xi \in \mathcal{S}_\delta$  can be uniquely associated to a projection onto a single subspace in the mixture, which will allow us to define a corresponding projection onto  $\mathcal{S}$ . Given a  $\xi \in \mathcal{S}_\delta$ , by the above, we can find a subspace  $\mathbf{U}_k$  such that the orthogonal decomposition  $\xi = \xi_k^\parallel + \xi_k^\perp$  satisfies

$$\left\| \xi_k^\perp \right\|_2^2 + \left( \left\| \xi_k^\parallel \right\|_2 - 1 \right)^2 \leq \delta^2. \quad (\text{B.3.29})$$

Consider the decomposition  $\xi = \xi_j^\parallel + \xi_j^\perp$  for some  $j \neq k$ . We have

$$\left\| \xi_j^\parallel \right\|_2 = \left\| \mathbf{U}_j \mathbf{U}_j^\top \xi \right\|_2 = \left\| \mathbf{U}_j \mathbf{U}_j^\top (\mathbf{U}_k \mathbf{U}_k^\top \xi + (\mathbf{I} - \mathbf{U}_k \mathbf{U}_k^\top) \xi) \right\|_2 \quad (\text{B.3.30})$$

$$= \left\| \mathbf{U}_j \mathbf{U}_j^\top (\mathbf{I} - \mathbf{U}_k \mathbf{U}_k^\top) \xi \right\|_2 \quad (\text{B.3.31})$$

$$\leq \left\| (\mathbf{I} - \mathbf{U}_k \mathbf{U}_k^\top) \xi \right\|_2 = \left\| \xi_k^\perp \right\|_2 \leq \delta, \quad (\text{B.3.32})$$

where the second line uses the orthogonality assumption on the subspaces  $\mathbf{U}_k$ , and the third uses the fact that orthogonal projections are nonexpansive. Hence, the  $j$ -th distance satisfies

$$\left\| \xi_j^\perp \right\|_2^2 + \left( \left\| \xi_j^\parallel \right\|_2 - 1 \right)^2 \geq (1 - \delta)^2. \quad (\text{B.3.33})$$

This implies that if  $0 < \delta < 1/2$ , every  $\xi \in \mathcal{S}_\delta$  has a unique closest subspace in the mixture. Hence, under this condition, the following mapping  $\pi_{\mathcal{S}} : \mathcal{S}_\delta \rightarrow \mathcal{S}$  is well-defined:

$$\pi_{\mathcal{S}}(\xi) = \pi_{\mathcal{S}_{k_\star}}(\xi), \text{ where } k_\star = \arg \min_{k \in [K]} \text{dist}(\xi, \mathcal{S}_k). \quad (\text{B.3.34})$$

Now, we define a code for  $\mathbf{x}_\delta$  by

$$\hat{\mathbf{x}}_\delta = q(\pi_{\mathcal{S}}(\mathbf{x}_\delta)). \quad (\text{B.3.35})$$

Clearly this is associated to a rate- $R$  code for  $\mathbf{x}_\delta$ , because it uses the encoding-decoding mappings from the rate- $R$  code for  $\mathbf{x}$ . We have to show that it achieves small distortion. We calculate

$$\mathbb{E} [\|\mathbf{x}_\delta - \hat{\mathbf{x}}_\delta\|_2^2] = \mathbb{E} [\|\mathbf{x}_\delta - q(\pi_{\mathcal{S}}(\mathbf{x}_\delta))\|_2^2] \quad (\text{B.3.36})$$

$$\leq \left( \mathbb{E} [\|\mathbf{x}_\delta - \pi_{\mathcal{S}}(\mathbf{x}_\delta)\|_2^2]^{1/2} + \mathbb{E} [\|\pi_{\mathcal{S}}(\mathbf{x}_\delta) - q(\pi_{\mathcal{S}}(\mathbf{x}_\delta))\|_2^2]^{1/2} \right)^2, \quad (\text{B.3.37})$$

where the inequality uses the Minkowski inequality. Now, by Definitions B.1 and B.2, we have deterministically

$$\|\mathbf{x}_\delta - \pi_{\mathcal{S}}(\mathbf{x}_\delta)\|_2^2 \leq \delta^2, \quad (\text{B.3.38})$$

so the expectation also satisfies this estimate. For the second term, it will suffice to characterize the density of the random variable  $\pi_{\mathcal{S}}(\mathbf{x}_\delta)$  as being sufficiently close to the density of  $\mathbf{x}$ —which, as Assumption B.3 implies, is a mixture of uniform distributions on each sub-sphere  $\mathcal{S}_k$ . By the argument above, every point  $\xi \in \mathcal{S}_\delta$  can be associated to one and only one subspace  $\mathbf{U}_k$ , which means that the mixture components in the definition of  $\mathcal{S}_\delta$  (recall Definition B.2) do not overlap. Hence, the density  $\pi_{\mathcal{S}}(\mathbf{x}_\delta)$  can be characterized by studying the effect of  $\pi_{\mathcal{S}_k}$  on the conditional random variable  $\mathbf{x}_\delta$ , conditioned on being drawn from  $\mathcal{S}_{k,\delta}$ . Denote this measure by  $\mu_{k,\delta}$ . We claim that the pushforward of this measure under  $\pi_{\mathcal{S}_k}$  is uniform on  $\mathcal{S}_k$ . To see that this holds, we recall Equation (B.3.28), which gives the characterization

$$\mathcal{S}_{k,\delta} = \left\{ \xi^\parallel + \xi^\perp \mid \xi^\parallel \in \text{Span}(\mathbf{U}_k), \xi^\perp \in \text{Span}(\mathbf{U}_k)^\perp, \|\xi^\perp\|_2^2 + (\|\xi^\parallel\|_2 - 1)^2 \leq \delta \right\}. \quad (\text{B.3.39})$$

The conditional distribution in question is uniform on this set; we need to show that the projection  $\pi_{\mathcal{S}_k}$  applied to this conditional random variable yields a random variable that is uniform on  $\mathcal{S}_k$ . With respect to these coordinates, we have seen that  $\pi_{\mathcal{S}_k}(\xi^\parallel + \xi^\perp) = \xi^\parallel / \|\xi^\parallel\|_2$ . Hence, for any  $\xi \in \mathcal{S}_k$ , we have that the preimage of  $\xi$  in  $\mathcal{S}_{k,\delta}$  under  $\pi_{\mathcal{S}_k}$  is

$$\pi_{\mathcal{S}_k}^{-1}(\xi) = \left\{ r\xi + \xi^\perp \mid r > 0, \xi^\perp \in \text{Span}(\mathbf{U}_k)^\perp, \|\xi^\perp\|_2^2 + (r-1)^2 \leq \delta \right\}. \quad (\text{B.3.40})$$

To show that  $(\pi_{\mathcal{S}_k})_\# \mu_{k,\delta}$  is uniform, we need to decompose the integral of the uniform density on  $\mathcal{S}_{k,\delta}$  in a way that makes it clear that each of the fibers  $\pi_{\mathcal{S}_k}^{-1}(\xi)$  (for each  $\xi \in \mathcal{S}_k$ ) “contributes” equally to the integral.<sup>6</sup> We have by Definition B.2

$$\text{vol}(\mathcal{S}_{k,\delta}) = \iint_{\text{Span}(\mathbf{U}_k) \times \text{Span}(\mathbf{U}_k)^\perp} \mathbf{1}_{\|\xi^\perp\|_2^2 + (\|\xi^\parallel\|_2 - 1)^2 \leq \delta} d\xi^\parallel d\xi^\perp. \quad (\text{B.3.41})$$

In particular, the integration over the orthogonal coordinates factors. Let  $d\theta^d$  denote the uniform (Haar) measure on the sphere of radius 1 in  $\mathbb{R}^d$ . Converting the  $\xi^\parallel$  integral to polar coordinates, we have

$$\text{vol}(\mathcal{S}_{k,\delta}) = \int_{[0,\infty)} \int_{\mathbb{S}^{d_k-1}} \int_{\text{Span}(\mathbf{U}_k)^\perp} r^{d_k-1} \mathbf{1}_{\|\xi^\perp\|_2^2 + (r-1)^2 \leq \delta} dr d\theta^{d_k} d\xi^\perp. \quad (\text{B.3.42})$$

Comparing to the fiber representation (B.3.40), we see that we need to “integrate out” over the  $r$  and  $\xi^\perp$  components of the preceding integral in order to verify that the pushforward is uniform. But this is evident,

<sup>6</sup>More rigorously, this corresponds to decomposing the uniform density on  $\mathcal{S}_{k,\delta}$  into a regular conditional density corresponding to  $\xi \in \mathcal{S}_k$ , and showing that the corresponding density on  $\xi$  is uniform. The proof makes it clear this is true.

as the previous expression shows that the value of this integral is independent of  $\xi^{\parallel}$ —or, equivalently in context, the value of the spherical component  $\theta^{d_k}$ .

Thus it follows from the above argument that  $\pi_S(\mathbf{x}_\delta)$  is uniform. Because the assumption on  $\delta$  implies that the mixture components in the distribution of  $\mathbf{x}_\delta$  do not overlap, the mixing weights  $\pi_k$  are also preserved in the image  $\pi_S(\mathbf{x}_\delta)$ , and in particular, the distribution of  $\pi_S(\mathbf{x}_\delta)$  is equal to the distribution of  $\mathbf{x}$ . Hence the second term in Equation (B.3.37) satisfies

$$\mathbb{E} \left[ \|\pi_S(\mathbf{x}_\delta) - q(\pi_S(\mathbf{x}_\delta))\|_2^2 \right] = \mathbb{E} \left[ \|\mathbf{x} - q(\mathbf{x})\|_2^2 \right] \leq \epsilon^2, \quad (\text{B.3.43})$$

because  $q$  is a distortion- $\epsilon$  code for  $\mathbf{x}$ .

We have thus shown that the hypothesized rate- $R$ , (expected squared) distortion- $\epsilon^2$  code for  $\mathbf{x}$  produces a rate- $R$ , (expected squared) distortion  $\delta + \epsilon$  code for  $\mathbf{x}_\delta$ . This establishes that

$$R_{\delta+\epsilon}(\mathbf{x}_\delta) \leq R_\epsilon(\mathbf{x}), \quad (\text{B.3.44})$$

as was to be shown. □

# Bibliography

- [AMS09] P-A Absil, R Mahony, and R Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Apr. 2009.
- [AAJ+16] Alekh Agarwal, Animashree Anandkumar, Prateek Jain, and Praneeth Netrapalli. “Learning Sparsely Used Overcomplete Dictionaries via Alternating Minimization”. *SIAM Journal on Optimization* 26.4 (2016), pp. 2775–2799. eprint: <https://doi.org/10.1137/140979861>.
- [AEB06] Michal Aharon, Michael Elad, and Alfred Bruckstein. “K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation”. *IEEE Transactions on signal processing* 54.11 (2006), pp. 4311–4322.
- [AR20] Jason M. Allred and Kaushik Roy. “Controlled Forgetting: Targeted Stimulation and Dopaminergic Plasticity Modulation for Unsupervised Lifelong Learning in Spiking Neural Networks”. *Frontiers in Neuroscience* 14 (2020).
- [ADG+16] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. “Learning to learn by gradient descent by gradient descent”. *Advances in neural information processing systems*. 2016, pp. 3981–3989.
- [ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein generative adversarial networks”. *International conference on machine learning*. PMLR. 2017, pp. 214–223.
- [AGM+15] Sanjeev Arora, Rong Ge, Tengyu Ma, and Ankur Moitra. “Simple, Efficient, and Neural Algorithms for Sparse Coding”. *Proceedings of The 28th Conference on Learning Theory*. Ed. by Peter Grünwald, Elad Hazan, and Satyen Kale. Vol. 40. Proceedings of Machine Learning Research. Paris, France: PMLR, July 2015, pp. 113–149.
- [AW18] Aharon Azulay and Yair Weiss. “Why do deep convolutional networks generalize so poorly to small image transformations?” *arXiv preprint arXiv:1805.12177* (2018).
- [BJC85] B. Ans, J. Hérault, and C. Jutten. “Architectures neuromimétiques adaptatives : Détection de primitives”. *Cognitiva* 2 (1985), pp. 593–597.
- [BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. *arXiv preprint arXiv:1607.06450* (2016).
- [BM24] Hao Bai and Yi Ma. “Improving neuron-level interpretability with white-box language models”. *arXiv preprint arXiv:2410.16443* (2024).
- [BGN+17] Bowen Baker, Otkrist Gupta, N. Naik, and R. Raskar. “Designing Neural Network Architectures using Reinforcement Learning”. *ArXiv* abs/1611.02167 (2017).
- [Bal11] Pierre Baldi. “Autoencoders, unsupervised learning and deep architectures”. *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*. UTLW’11. Washington, USA: JMLR.org, 2011, pp. 37–50.
- [BH89] Pierre Baldi and Kurt Hornik. “Neural networks and principal component analysis: Learning from examples without local minima”. *Neural networks* 2.1 (1989), pp. 53–58.
- [BLZ+22] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. “Analytic-dpm: an analytic estimate of the optimal reverse variance in diffusion probabilistic models”. *arXiv preprint arXiv:2201.06503* (2022).

- [BSM+20] Pinglei Bao, Liang She, Mason McGill, and Doris Y. Tsao. “A map of object space in primate inferotemporal cortex”. *Nature* 583 (2020), pp. 103–108.
- [BKS15] Boaz Barak, Jonathan A Kelner, and David Steurer. “Dictionary learning and tensor decomposition via the sum-of-squares method”. *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. New York, NY, USA: ACM, June 2015.
- [BT09] Amir Beck and Marc Teboulle. “A fast iterative shrinkage-thresholding algorithm for linear inverse problems”. *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.
- [Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [BN24a] Jeremy Bernstein and Laker Newhouse. *Old Optimizer, New Norm: An Anthology*. 2024. arXiv: 2409.20325 [cs.LG].
- [Bla72] R. Blahut. “Computation of channel capacity and rate-distortion functions”. *IEEE Transactions on Information Theory* 18.4 (1972), pp. 460–473.
- [BBF+01] Lorna Booth, Jehoshua Bruck, M. Franceschetti, and Ronald Meester. “Covering Algorithms, Continuum Percolation and the Geometry of Wireless Networks”. *Ann. Appl. Probab.* 13 (July 2001).
- [Bor97] Vivek S Borkar. “Stochastic approximation with two time scales”. *Systems & Control Letters* 29.5 (1997), pp. 291–294.
- [BDS16] Vivek S Borkar, Raaz Dwivedi, and Neeraja Sahasrabudhe. “Gaussian approximations in high dimensional estimation”. *Systems & Control Letters* 92 (2016), pp. 42–45.
- [Bos50] R. Boscovich. *De calculo probabilitatum que respondent diversis valoribus summe errorum post plures observationes, quarum singule possint esse erronee certa quadam quantitate*. 1750.
- [Bou23] Nicolas Boumal. *An Introduction to Optimization on Smooth Manifolds*. Cambridge University Press, Mar. 2023.
- [BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [BN24b] Arwen Bradley and Preetum Nakkiran. “Classifier-Free Guidance is a Predictor-Corrector”. *arXiv [cs.LG]* (Aug. 2024). arXiv: 2408.09000 [cs.LG].
- [BN20] Guy Bresler and Dheeraj Nagaraj. “Sharp representation theorems for relu networks with precise dependence on depth”. *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS’20 Article 897. Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 10697–10706.
- [BB11] Haim Brezis and Haim Brézis. *Functional analysis, Sobolev spaces and partial differential equations*. Vol. 2. 3. Springer, 2011.
- [BEJ25] Paige Bright, Alan Edelman, and Steven G Johnson. “Matrix Calculus (for Machine Learning and Beyond)”. *arXiv preprint arXiv:2501.14787* (2025).
- [BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. “Language models are few-shot learners”. *arXiv preprint arXiv:2005.14165* (2020).
- [BM13] Joan Bruna and Stéphane Mallat. “Invariant Scattering Convolution Networks”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1872–1886.
- [BGW21] Sam Buchanan, Dar Gilboa, and John Wright. “Deep Networks and the Multiple Manifold Problem”. *International Conference on Learning Representations*. 2021.
- [CD91] M. Frank Callier and A. Charles Desoer. *Linear System Theory*. Springer-Verlag, 1991.

- [Can06] E. Candès. “Compressive sampling”. *Proceedings of the International Congress of Mathematicians*. 2006.
- [CT05a] E. Candès and T. Tao. “Decoding by linear programming”. *IEEE Transactions on Information Theory* 51.12 (2005).
- [CT05b] E. Candès and T. Tao. “Error Correction via Linear Programming”. *IEEE Symposium on FOCS* (2005), pp. 295–308.
- [CTM+21] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. “Emerging properties in self-supervised vision transformers”. *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9650–9660.
- [Cha66] Gregory J. Chaitin. “On the Length of Programs for Computing Finite Binary Sequences”. *J. ACM* 13.4 (Oct. 1966), pp. 547–569.
- [CYY+22] Kwan Ho Ryan Chan, Yaodong Yu, Chong You, Haozhi Qi, John Wright, and Yi Ma. “ReduNet: A White-box Deep Network from the Principle of Maximizing Rate Reduction”. *Journal of Machine Learning Research* 23.114 (2022), pp. 1–103.
- [CJG+15] Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, and Yi Ma. “PCANet: A simple deep learning baseline for image classification?” *TIP* (2015).
- [CT17] Le Chang and Doris Tsao. “The Code for Facial Identity in the Primate Brain”. *Cell* 169 (June 2017), 1013–1028.e14.
- [CRE+19] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc’Aurelio Ranzato. “On tiny episodic memories in continual learning”. *arXiv preprint arXiv:1902.10486* (2019).
- [CHZ+23] Minshuo Chen, Kaixuan Huang, Tuo Zhao, and Mengdi Wang. “Score approximation, estimation and distribution recovery of diffusion models on low-dimensional data”. *International Conference on Machine Learning*. PMLR. 2023, pp. 4672–4712.
- [CRB+18] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. “Neural ordinary differential equations”. *Advances in neural information processing systems* 31 (2018).
- [CCL+23] Sitan Chen, Sinho Chewi, Jerry Li, Yuanzhi Li, Adil Salim, and Anru Zhang. “Sampling is as easy as learning the score: theory for diffusion models with minimal data assumptions”. *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.
- [CZG+24] Siyi Chen, Huijie Zhang, Minzhe Guo, Yifu Lu, Peng Wang, and Qing Qu. “Exploring low-dimensional subspace in diffusion models for controllable image editing”. *Advances in neural information processing systems* 37 (2024), pp. 27340–27371.
- [CZL+25] Siyi Chen, Yimeng Zhang, Sijia Liu, and Qing Qu. “The Dual Power of Interpretable Token Embeddings: Jailbreaking Attacks and Defenses for Diffusion Model Unlearning”. *arXiv preprint arXiv:2504.21307* (2025).
- [CKN+20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. “A simple framework for contrastive learning of visual representations”. *arXiv preprint arXiv:2002.05709* (2020).
- [CLH+24] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. “Symbolic discovery of optimization algorithms”. *Advances in neural information processing systems* 36 (2024).
- [Cho17] Francois Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1800–1807.
- [CKM+23] Hyungjin Chung, Jeongsol Kim, Michael Thompson Mccann, Marc Louis Klasky, and Jong Chul Ye. “Diffusion Posterior Sampling for General Noisy Inverse Problems”. *The Eleventh International Conference on Learning Representations*. 2023.
- [CW16a] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. *International Conference on Machine Learning*. 2016, pp. 2990–2999.

- [CW16b] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.
- [CW16c] Taco S. Cohen and Max Welling. “Group Equivariant Convolutional Networks”. *CoRR* abs/1602.07576 (2016). arXiv: [1602.07576](#).
- [CV95] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. *Mach. Learn.* 20.3 (1995), pp. 273–297.
- [CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications, 1991.
- [Cov64] Thomas Cover. “Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition”. *IEEE TRANSACTIONS ON ELECTRONIC COMPUTERS* (1964).
- [Cyb89] George V. Cybenko. “Approximation by superpositions of a sigmoidal function”. *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314.
- [D00] D. Donoho. “High-dimensional data analysis: The curses and blessings of dimensionality”. *AMS Math Challenges Lecture* (2000).
- [DM03] D. Donoho and M. Elad. “Optimally sparse representation in general (nonorthogonal) dictionaries via  $\ell^1$  minimization”. *PNAS* 100.5 (2003), pp. 2197–2202.
- [DTL+22] Xili Dai, Shengbang Tong, Mingyang Li, Ziyang Wu, Michael Psenka, Kwan Ho Ryan Chan, Pengyuan Zhai, Yaodong Yu, Xiaojun Yuan, Heung-Yeung Shum, and Yi Ma. “CTRL: Closed-Loop Transcription to an LDR via Minimaxing Rate Reduction”. *Entropy* 24.4 (2022).
- [Dan02] George B Dantzig. “Linear Programming”. *Operations Research* 50.1 (2002), pp. 42–47.
- [DSD+23a] Giannis Daras, Kulin Shah, Yuval Dagan, Aravind Gollakota, Alex Dimakis, and Adam Klivans. “Ambient Diffusion: Learning Clean Distributions from Corrupted Data”. *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.
- [DSD+23b] Giannis Daras, Kulin Shah, Yuval Dagan, Aravind Gollakota, Alex Dimakis, and Adam Klivans. “Ambient Diffusion: Learning Clean Distributions from Corrupted Data”. *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 288–313.
- [DGG+25] Valentin De Bortoli, Alexandre Galashov, J Swaroop Guntupalli, Guangyao Zhou, Kevin Murphy, Arthur Gretton, and Arnaud Doucet. “Distributional Diffusion Models with Scoring Rules”. *arXiv preprint arXiv:2502.02483* (2025).
- [DCM+23] Aaron Defazio, Ashok Cutkosky, Harsh Mehta, and Konstantin Mishchenko. “Optimal linear decay learning rate schedules and further refinements”. *arXiv preprint arXiv:2310.07831* (2023).
- [DDS22] Julie Delon, Agnes Desolneux, and Antoine Salmona. “Gromov–Wasserstein distances between Gaussian distributions”. *Journal of Applied Probability* 59.4 (2022), pp. 1178–1198.
- [DCL+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.
- [DN21] Prafulla Dhariwal and Alexander Quinn Nichol. “Diffusion Models Beat GANs on Image Synthesis”. *Advances in Neural Information Processing Systems*. Ed. by A Beygelzimer, Y Dauphin, P Liang, and J Wortman Vaughan. 2021.
- [Don01] D L Donoho. “Sparse components of images and optimal atomic decompositions”. *Constructive approximation* 17.3 (Jan. 2001), pp. 353–382.
- [DVD+98] D L Donoho, M Vetterli, R A DeVore, and I Daubechies. “Data compression and harmonic analysis”. *IEEE transactions on information theory / Professional Technical Group on Information Theory* 44.6 (Oct. 1998), pp. 2435–2476.

- [Don05] David L Donoho. “Neighborly polytopes and sparse solutions of underdetermined linear equations”. *Stanford Technical Report 2005-04* (2005).
- [DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [EY36] Carl Eckart and Gale Young. “The approximation of one matrix by another of lower rank”. *Psychometrika* 1.3 (Sept. 1936), pp. 211–218.
- [EAS98] A Edelman, T Arias, and S Smith. “The Geometry of Algorithms with Orthogonality Constraints”. *SIAM Journal on Matrix Analysis and Applications* 20.2 (Jan. 1998), pp. 303–353.
- [EA06] Michael Elad and Michal Aharon. “Image denoising via sparse and redundant representations over learned dictionaries”. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society* 15.12 (Dec. 2006), pp. 3736–3745.
- [EHO+22a] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. “Toy models of superposition”. *arXiv preprint arXiv:2209.10652* (2022).
- [EHO+22b] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. “Toy Models of Superposition”. *Transformer Circuits Thread* (2022).
- [ETT+17] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. “A rotation and a translation suffice: Fooling CNNs with simple transformations”. *arXiv preprint arXiv:1712.02779* (2017).
- [EKB+24] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. “Scaling rectified flow transformers for high-resolution image synthesis”. *Forty-first international conference on machine learning*. 2024.
- [FZS22] William Fedus, Barret Zoph, and Noam Shazeer. “Switch transformers: scaling to trillion parameter models with simple and efficient sparsity”. *J. Mach. Learn. Res.* 23.1 (Jan. 2022).
- [Fel49] William Feller. “On the Theory of Stochastic Processes, with Particular Reference to Applications”. 1949.
- [FCR20] Tanner Fiez, Benjamin Chasnov, and Lillian Ratliff. “Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study”. *International Conference on Machine Learning*. PMLR. 2020, pp. 3133–3144.
- [FCR19] Tanner Fiez, Benjamin Chasnov, and Lillian J Ratliff. “Convergence of learning dynamics in stackelberg games”. *arXiv preprint arXiv:1906.01217* (2019).
- [Fuk69] Kunihiko Fukushima. “Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements”. *IEEE Transactions on Systems Science and Cybernetics* 5.4 (1969), pp. 322–333.
- [Fuk80] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. *Biological Cybernetics* 36 (1980), pp. 193–202.
- [GTT+25] Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. “Scaling and evaluating sparse autoencoders”. *The Thirteenth International Conference on Learning Representations*. 2025.
- [GG23] Guillaume Garrigos and Robert M Gower. “Handbook of convergence theorems for (stochastic) gradient methods”. *arXiv preprint arXiv:2301.11235* (2023).

- [Gil61] E. N. Gilbert. “Random Plane Networks”. *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 533–543. eprint: <https://doi.org/10.1137/0109045>.
- [GC19] Aaron Gokaslan and Vanya Cohen. *OpenWebText Corpus*. <http://Skylion007.github.io/OpenWebTextCorpus>. 2019.
- [GPM+14a] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative Adversarial Nets”. *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014.
- [GPM+14b] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial nets”. *Advances in neural information processing systems*. 2014, pp. 2672–2680.
- [GL10] Karol Gregor and Yann LeCun. “Learning fast approximations of sparse coding”. *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010, pp. 399–406.
- [Gri11] Rémi Gribonval. “Should Penalized Least Squares Regression be Interpreted as Maximum A Posteriori Estimation?” *IEEE transactions on signal processing: a publication of the IEEE Signal Processing Society* 59.5 (May 2011), pp. 2405–2410.
- [GJB15] Remi Gribonval, Rodolphe Jenatton, and Francis Bach. “Sparse and spurious: Dictionary learning with noise and outliers”. *IEEE transactions on information theory* 61.11 (Nov. 2015), pp. 6298–6319.
- [Gro87] Stephen Grossberg. “Competitive Learning: From Interactive Activation to Adaptive Resonance”. *Cogn. Sci.* 11 (1987), pp. 23–63.
- [HY01] M.H. Hansen and B. Yu. “Model Selection and the Principle of Minimum Description Length”. *Journal of American Statistical Association* 96 (2001), pp. 746–774.
- [HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Second. Springer, 2009.
- [HCX+22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. “Masked autoencoders are scalable vision learners”. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16000–16009.
- [HZR+16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.
- [HZR+16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [HS06] G. E. Hinton and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. *Science* 313.5786 (2006), pp. 504–507. eprint: <https://science.sciencemag.org/content/313/5786/504.full.pdf>.
- [HZ93] Geoffrey E. Hinton and Richard S. Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS’93. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 3–10.
- [HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.
- [HS22] Jonathan Ho and Tim Salimans. “Classifier-Free Diffusion Guidance”. *arXiv [cs.LG]* (July 2022). arXiv: [2207.12598 \[cs.LG\]](https://arxiv.org/abs/2207.12598).
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. *Neural computation* 9 (Dec. 1997), pp. 1735–80.

- [HSD20] David Hong, Yue Sheng, and Edgar Dobriban. *Selecting the number of components in PCA via random signflips*. 2020. arXiv: 2012.02985 [math.ST].
- [Hot33] H. Hotelling. “Analysis of a Complex of Statistical Variables into Principal Components”. *Journal of Educational Psychology* (1933).
- [HLV+17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. “Densely Connected Convolutional Networks”. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269.
- [HM99] Jinggang Huang and David Mumford. “Statistics of natural images and models”. *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 1. IEEE. 1999, pp. 541–547.
- [HW59] D.H. Hubel and T.N. Wiesel. “Receptive fields of single neurones in the cat’s striate cortex”. *J. Physiol.* 148.3 (1959), pp. 574–591.
- [HCS+24] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. “Sparse Autoencoders Find Highly Interpretable Features in Language Models”. *The Twelfth International Conference on Learning Representations*. 2024.
- [HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.
- [HKJ+21] Uiwon Hwang, Heeseung Kim, Dahuin Jung, Hyemi Jang, Hyungyu Lee, and Sungroh Yoon. “Stein Latent Optimization for Generative Adversarial Networks”. *arXiv preprint arXiv:2106.05319* (2021).
- [Hyv05] Aapo Hyvärinen. “Estimation of Non-Normalized Statistical Models by Score Matching”. *Journal of Machine Learning Research* 6.24 (2005), pp. 695–709.
- [HO97] Aapo Hyvärinen and Erkki Oja. “A Fast Fixed-Point Algorithm for Independent Component Analysis”. *Neural Computation* 9.7 (1997), pp. 1483–1492.
- [HO00a] Aapo Hyvärinen and Erkki Oja. “Independent Component Analysis: Algorithms and Applications”. *Neural Networks* 13.4-5 (2000), pp. 411–430.
- [HO00b] Aapo Hyvärinen and Erkki Oja. “Independent component analysis: algorithms and applications”. *Neural Networks* 13.4 (2000), pp. 411–430.
- [IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. *ICML*. 2015, pp. 448–456.
- [Jam15] G J O Jameson. “A simple proof of Stirling’s formula for the gamma function”. *The Mathematical Gazette* 99.544 (Mar. 2015), pp. 68–74.
- [JRR+24] Yibo Jiang, Goutham Rajendran, Pradeep Kumar Ravikumar, Bryon Aragam, and Victor Veitch. “On the Origins of Linear Representations in Large Language Models”. *International Conference on Machine Learning*. Vol. 235. PMLR, 2024, pp. 21879–21911.
- [Jol02] I. Jolliffe. *Principal Component Analysis*. 2nd. Springer-Verlag, 2002.
- [Jol86] I. Jolliffe. *Principal Component Analysis*. New York, NY: Springer-Verlag, 1986.
- [Jon82] Douglas Samuel Jones. *The theory of generalised functions*. Cambridge University Press, 1982.
- [JJB+] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cecista, Laker Newhouse, and Jeremy Bernstein. “Muon: An optimizer for hidden layers in neural networks, 2024”. URL <https://kellerjordan.github.io/posts/muon> 6 () .
- [JT20] Sheena A. Josselyn and Susumu Tonegawa. “Memory engrams: Recalling the past and imagining the future”. *Science* 367 (2020).
- [KS21] Z Kadkhodaie and E P Simoncelli. “Stochastic solutions for linear inverse problems using the prior implicit in a denoiser”. *Adv. Neural Information Processing Systems (NeurIPS)*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021.

- [Kal60] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems” (1960).
- [KG24] Mason Kamb and Surya Ganguli. “An analytic theory of creativity in convolutional diffusion models”. *arXiv preprint arXiv:2412.20292* (2024).
- [Kar22a] Andrej Karpathy. *nanoGPT*. <https://github.com/karpathy/nanoGPT>. 2022.
- [Kar22b] Andrej Karpathy. *The spelled-out intro to neural networks and backpropagation: building micrograd*. YouTube. Aug. 16, 2022. URL: <https://www.youtube.com/watch?v=VMj-3S1tku0> (visited on 08/17/2025).
- [KK18] Ronald Kemker and Christopher Kanan. “FearNet: Brain-Inspired Model for Incremental Learning”. *International Conference on Learning Representations*. 2018.
- [KB14] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980* (2014).
- [KW13] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. *arXiv [stat.ML]* (Dec. 2013). arXiv: [1312.6114v11 \[stat.ML\]](https://arxiv.org/abs/1312.6114v11).
- [KW19] Diederik P Kingma and Max Welling. “An introduction to variational autoencoders”. *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.
- [KPR+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. “Overcoming catastrophic forgetting in neural networks”. *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.
- [KUM+17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. “Self-normalizing neural networks”. *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 972–981.
- [KTV18] Artemy Kolchinsky, Brendan D Tracey, and Steven Van Kuyk. “Caveats for information bottleneck in deterministic scenarios”. *arXiv preprint arXiv:1808.07593* (2018).
- [Kol98] Andrei N. Kolmogorov. “On Tables of Random Numbers (Reprinted from ”Sankhya: The Indian Journal of Statistics”, Series A, Vol. 25 Part 4, 1963)”. *Theor. Comput. Sci.* 207 (1998), pp. 387–395.
- [KS12] Irwin Kra and Santiago R Simanca. “On Circulant Matrices”. *Notices of the American Mathematical Society* 59 (2012), pp. 368–377.
- [Kra91] Mark A Kramer. “Nonlinear principal component analysis using autoassociative neural networks”. *AIChe Journal* 37.2 (1991), pp. 233–243.
- [KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images” (2009).
- [KNH14] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “The CIFAR-10 dataset”. online: <http://www.cs.toronto.edu/~kriz/cifar.html> 55 (2014).
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [LBB+25] Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. “FLUX.1 Kontext: Flow matching for in-context image generation and editing in latent space”. *arXiv [cs.GR]* (June 2025). arXiv: [2506.15742 \[cs.GR\]](https://arxiv.org/abs/2506.15742).
- [LRM+12] Quoc V. Le, Marc’Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. “Building high-level features using large scale unsupervised learning”. *Proceedings of the 29th International Conference on International Conference on Machine Learning*. ICML’12. Edinburgh, Scotland: Omnipress, 2012, pp. 507–514.

- [LBD+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. *Neural Computation* 1.4 (1989), pp. 541–551.
- [LBB+98a] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [LBB+98b] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [LPM03] Ann Lee, Kim Pedersen, and David Mumford. “The Nonlinear Statistics of High-Contrast Patches in Natural Images”. *International Journal of Computer Vision* 54 (Aug. 2003).
- [LSJ+16] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. “Gradient descent only converges to minimizers”. *Conference on learning theory*. PMLR. 2016, pp. 1246–1257.
- [Lee02] John M. Lee. “Introduction to Smooth Manifolds”. 2002.
- [LMH+18] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. “Noise2Noise: Learning Image Restoration without Clean Data”. *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 2965–2974.
- [LY24] Gen Li and Yuling Yan. “O (d/T) convergence theory for diffusion probabilistic models under minimal assumptions”. *arXiv preprint arXiv:2409.18959* (2024).
- [LFD+22] Haochuan Li, Farzan Farnia, Subhro Das, and Ali Jadbabaie. “On convergence of gradient descent ascent: A tight local analysis”. *International Conference on Machine Learning*. PMLR. 2022, pp. 12717–12740.
- [Li17] Xi-Lin Li. “Preconditioned stochastic gradient descent”. *IEEE transactions on neural networks and learning systems* 29.5 (2017), pp. 1454–1466.
- [LZQ24] Wenda Li, Huijie Zhang, and Qing Qu. “Shallow diffuse: Robust and invisible watermarking through low-dimensional subspaces in diffusion models”. *arXiv preprint arXiv:2410.21088* (2024).
- [LWQ25] Xiang Li, Rongrong Wang, and Qing Qu. “Towards Understanding the Mechanisms of Classifier-Free Guidance”. *arXiv [cs.CV]* (May 2025). arXiv: [2505.19210 \[cs.CV\]](#).
- [LB19] Yanjun Li and Yoram Bresler. “Multichannel sparse blind deconvolution on the sphere”. *IEEE Transactions on Information Theory* 65.11 (2019), pp. 7415–7436.
- [LRZ+12] Xiao Liang, Xiang Ren, Zhengdong Zhang, and Yi Ma. “Repairing Sparse Low-Rank Texture”. *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 482–495.
- [LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context”. *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.
- [LZ94] T Linder and R Zamir. “On the asymptotic tightness of the Shannon lower bound”. *IEEE transactions on information theory* 40.6 (1994), pp. 2026–2031.
- [LMZ+24] Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. “Infinigram: Scaling unbounded n-gram language models to a trillion tokens”. *arXiv preprint arXiv:2401.17377* (2024).
- [LSY+25] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. “Muon is Scalable for LLM Training”. *arXiv preprint arXiv:2502.16982* (2025).
- [LV09] Z. Liu and L. Vandenberghe. “Semidefinite programming methods for system realization and identification”. *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 2009, pp. 4676–4681.

- [LV10] Zhang. Liu and Lieven. Vandenberghe. “Interior-Point Method for Nuclear Norm Approximation with Application to System Identification”. *SIAM Journal on Matrix Analysis and Applications* 31.3 (2010), pp. 1235–1256. eprint: <https://doi.org/10.1137/090755436>.
- [LH17] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. *International Conference on Learning Representations*. 2017.
- [MYP+10] M. Journée, Y. Nesterov, P. Richtárik, and R. Sepulchre. “Generalized power method for sparse principal component analysis”. *Journal of Machine Learning Research* 11 (2010), pp. 517–553.
- [MDH+07a] Y. Ma, H. Derksen, W. Hong, and J. Wright. “Segmentation of multivariate mixed data via lossy coding and compression”. *To appear in IEEE Transactions on Pattern Analysis and Machine Intelligence* (2007).
- [MKS+04] Y. Ma, J. Košecká, S. Soatto, and S. Sastry. *An Invitation to 3-D Vision, From Images to Models*. New York: Springer-Verlag, 2004.
- [MDH+07b] Yi Ma, Harm Derksen, Wei Hong, and John Wright. “Segmentation of multivariate mixed data via lossy data coding and compression”. *IEEE transactions on pattern analysis and machine intelligence* 29.9 (2007), pp. 1546–1562.
- [MTS22] Yi Ma, Doris Tsao, and Heung-Yeung Shum. “On the principles of Parsimony and Self-consistency for the emergence of intelligence”. *Frontiers Inf. Technol. Electron. Eng.* 23.9 (2022), pp. 1298–1323.
- [MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. *Proc. ICML*. Vol. 30. Citeseer. 2013, p. 3.
- [MBP14] Julien Mairal, Francis Bach, and Jean Ponce. “Sparse Modeling for Image and Vision Processing”. *Foundations and Trends® in Computer Graphics and Vision* 8.2-3 (2014), pp. 85–283.
- [MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. “Building a Large Annotated Corpus of English: The Penn Treebank”. *Computational Linguistics* 19.2 (1993). Ed. by Julia Hirschberg, pp. 313–330.
- [Mar06] Andrei Andreevich Markov. “An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains”. *Science in Context* 19.4 (2006), pp. 591–600.
- [MBD+21] James Martens, Andy Ballard, Guillaume Desjardins, Grzegorz Swirszcz, Valentin Dalibard, Jascha Sohl-Dickstein, and Samuel S Schoenholz. “Rapid training of deep neural networks without skip connections or normalization layers using Deep Kernel Shaping”. *arXiv [cs.LG]* (Oct. 2021). arXiv: [2110.01765 \[cs.LG\]](https://arxiv.org/abs/2110.01765).
- [MC89] Michael McCloskey and Neal J Cohen. “Catastrophic interference in connectionist networks: The sequential learning problem”. *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.
- [MP43] Warren McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.
- [MM70] Jerry M. Mendel and Robert W. McLaren. “Reinforcement-learning control and pattern recognition systems”. In Mendel, J. M. and Fu, K. S., editors, *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications* (1970), pp. 287–318.
- [MXB+16] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. *Pointer Sentinel Mixture Models*. 2016. arXiv: [1609.07843 \[cs.CL\]](https://arxiv.org/abs/1609.07843).
- [MM12] Stephan Mertens and Christopher Moore. “Continuum percolation thresholds in two dimensions”. *Phys. Rev. E* 86 (6 Dec. 2012), p. 061109.
- [Min54] Marvin Minsky. “Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem”. PhD thesis. Princeton University, 1954.
- [MP69] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. the MIT Press, 1969.

- [Mir60] L Mirsky. “SYMMETRIC GAUGE FUNCTIONS AND UNITARILY INVARIANT NORMS”. *The Quarterly Journal of Mathematics* 11.1 (Jan. 1960), pp. 50–59.
- [Miy61] K. Miyasawa. “An empirical bayes estimator of the mean of a normal population”. *Bull. Inst. Internat. Statist.* 38 (1961).
- [MKK+18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. “Spectral normalization for generative adversarial networks”. *arXiv preprint arXiv:1802.05957* (2018).
- [MRY+11] Hossein Mobahi, Shankar Rao, Allen Yang, Shankar Sastry, and Yi Ma. “Segmentation of Natural Images by Texture and Boundary Compression”. *the International Journal of Computer Vision* 95.1 (2011), pp. 86–98.
- [MLE19] Vishal Monga, Yuelong Li, and Yonina C Eldar. “Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing”. *arXiv preprint arXiv:1912.10557* (2019).
- [MKH19] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. “When does label smoothing help?” *Advances in neural information processing systems* 32 (2019).
- [Mum96] David Mumford. “The Statistical Description of Visual Signals”. 1996.
- [MG99] David Mumford and Basilis Gidas. “Stochastic Models for Generic Images”. *Quarterly of Applied Mathematics* 59 (July 1999).
- [MK07] Joseph F Murray and Kenneth Kreutz-Delgado. “Learning sparse overcomplete codes for images”. *The Journal of VLSI Signal Processing Systems for Signal Image and Video Technology* 46.1 (Mar. 2007), pp. 1–13.
- [NDE+13] S. Nam, M.E. Davies, M. Elad, and R. Gribonval. “The cosparse analysis model and algorithms”. *Applied and Computational Harmonic Analysis* 34.1 (2013), pp. 30–56.
- [NZM+24] Matthew Niedoba, Berend Zwartenberg, Kevin Murphy, and Frank Wood. “Towards a Mechanistic Explanation of Diffusion Model Generalization”. *arXiv preprint arXiv:2411.19339* (2024).
- [NW06] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- [NIG+18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. “Activation functions: Comparison of trends in practice and research for deep learning”. *arXiv preprint arXiv:1811.03378* (2018).
- [Oja82] Erkki Oja. “A simplified neuron model as a principal component analyzer”. *Journal of Mathematical Biology* 15 (1982), pp. 267–273.
- [OF97] B A Olshausen and D J Field. “Sparse coding with an overcomplete basis set: a strategy employed by V1?” *Vision research* 37.23 (Dec. 1997), pp. 3311–3325.
- [OF96] Bruno A Olshausen and David J Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. *Nature* 381 (June 1996), p. 607.
- [OVK17] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural discrete representation learning”. *arXiv [cs.LG]* (Nov. 2017). arXiv: [1711.00937 \[cs.LG\]](https://arxiv.org/abs/1711.00937).
- [ODM+23] Maxime Oquab, Timothée Darzet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khaldov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. “Dinov2: Learning robust visual features without supervision”. *arXiv preprint arXiv:2304.07193* (2023).
- [PBW+24] Druv Pai, Sam Buchanan, Ziyang Wu, Yaodong Yu, and Yi Ma. “Masked Completion via Structured Diffusion with White-Box Transformers”. *The Twelfth International Conference on Learning Representations*. 2024.
- [PPC+23] Druv Pai, Michael Psenka, Chih-Yuan Chiu, Manxi Wu, Edgar Dobriban, and Yi Ma. “Pursuit of a discriminative representation for multiple subspaces via sequential games”. *Journal of the Franklin Institute* 360.6 (2023), pp. 4135–4171.

- [PKL+16] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. *The LAMBADA dataset: Word prediction requiring a broad discourse context*. 2016. arXiv: [1606.06031 \[cs.CL\]](#).
- [PHD20] Vardan Petyan, XY Han, and David L Donoho. “Prevalence of Neural Collapse during the terminal phase of deep learning training”. *arXiv preprint arXiv:2008.08186* (2020).
- [PRE17] Vardan Petyan, Yaniv Romano, and Michael Elad. “Convolutional neural networks analyzed via convolutional sparse coding”. *The Journal of Machine Learning Research* 18.1 (2017), pp. 2887–2938.
- [PCV24] Kiho Park, Yo Joong Choe, and Victor Veitch. “The Linear Representation Hypothesis and the Geometry of Large Language Models”. *International Conference on Machine Learning*. PMLR. 2024, pp. 39643–39666.
- [PSR+23] Georgios Pavlakos, Dandan Shan, Ilija Radosavovic, Angjoo Kanazawa, David Fouhey, and Jitendra Malik. “Reconstructing Hands in 3D with Transformers”. *arxiv*. 2023.
- [Pea01] K. Pearson. “On Lines and Planes of Closest Fit to Systems of Points in Space”. *Philosophical Magazine* 2.6 (1901), pp. 559–572.
- [PW22] Yuri Poliyanski and Yihong Wu. *Information Theory: From Coding to Learning*. Cambridge University Press, 2022.
- [PPR+24] Michael Psenka, Druv Pai, Vishal Raman, Shankar Sastry, and Yi Ma. “Representation Learning via Manifold Flattening and Reconstruction”. *Journal of Machine Learning Research* 25.132 (2024), pp. 1–47.
- [QLZ19] Qing Qu, Xiao Li, and Zhihui Zhu. “A nonconvex approach for exact and efficient multichannel sparse blind deconvolution”. *Advances in Neural Information Processing Systems*. 2019, pp. 4017–4028.
- [QLZ20] Qing Qu, Xiao Li, and Zhihui Zhu. “Exact Recovery of Multichannel Sparse Blind Deconvolution via Gradient Descent”. *SIAM Journal on Imaging Sciences* 13.3 (2020), pp. 1630–1652.
- [QZL+20a] Qing Qu, Yuexiang Zhai, Xiao Li, Yuqian Zhang, and Zhihui Zhu. “Geometric Analysis of Nonconvex Optimization Landscapes for Overcomplete Learning”. *International Conference on Learning Representations*. 2020.
- [QZL+20b] Qing Qu, Zhihui Zhu, Xiao Li, Manolis C. Tsakiris, John Wright, and René Vidal. *Finding the Sparsest Vectors in a Subspace: Theory, Algorithms, and Applications*. 2020. arXiv: [2001.06970 \[cs.LG\]](#).
- [RD03] R. Basri and D. Jacobs. “Lambertian reflectance and linear subspaces”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.2 (2003), pp. 218–233.
- [RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. “Learning Transferable Visual Models From Natural Language Supervision”. *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8748–8763.
- [RMC16] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. *arXiv preprint arXiv:1511.06434* (2016). arXiv: [1511.06434 \[cs.LG\]](#).
- [RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. “Language models are unsupervised multitask learners”. *OpenAI blog* 1.8 (2019), p. 9.
- [RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. “Hierarchical Text-Conditional Image Generation with CLIP Latents”. *arXiv [cs.CV]* (Apr. 2022). arXiv: [2204.06125 \[cs.CV\]](#).

- [RPC+06] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Cun. "Efficient Learning of Sparse Representations with an Energy-Based Model". *Advances in Neural Information Processing Systems*. Ed. by B Schölkopf, J Platt, and T Hoffman. Vol. 19. MIT Press, 2006.
- [RS11] M Raphan and E P Simoncelli. "Least squares estimation without priors or supervision". *Neural Computation* 23.2 (2011). Published online, Nov 2010., pp. 374–420.
- [RKS+17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. "iCaRL: Incremental classifier and representation learning". *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.
- [RBK18] Erwin Riegler, Helmut Bölcskei, and Gunther Koliander. "Rate-distortion theory for general sets and measures". *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, June 2018, pp. 101–105.
- [RKB23] Erwin Riegler, Günther Koliander, and Helmut Bölcskei. "Lossy compression of general random variables". *Information and inference: a journal of the IMA* 12.3 (Apr. 2023), pp. 1759–1829.
- [Ris78] J. Rissanen. "Paper: Modeling by shortest data description". *Automatica* 14.5 (1978), pp. 465–471.
- [Rob56] Herbert E. Robbins. "An Empirical Bayes Approach to Statistics". 1956.
- [RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. "High-Resolution Image Synthesis with Latent Diffusion Models". *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pp. 10674–10685.
- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.
- [RAL+24] François Rozet, Gérôme Andry, Francois Lanusse, and Gilles Louppe. "Learning Diffusion Priors from Observations by Expectation Maximization". *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.
- [RE14] R. Rubinstein and M. Elad. "Dictionary Learning for Analysis-Synthesis Thresholding". *IEEE Transactions on Signal Processing* 62.22 (2014), pp. 5962–5972.
- [RHW86a] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning internal representations by error propagation". *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.
- [RHW86b] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". *Nature* 323.6088 (Oct. 1986), pp. 533–536.
- [SCS+22] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasempour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". *NeurIPS*. 2022.
- [Sas99] Shankar Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer, 1999.
- [SHT+25] Fabian Schaipp, Alexander Hägele, Adrien Taylor, Umut Simsekli, and Francis Bach. "The Surprising Agreement Between Convex Optimization Theory and Learning-Rate Scheduling for Large Model Training". *arXiv preprint arXiv:2501.18965* (2025).
- [SMB10] Dominik Scherer, Andreas Müller, and Sven Behnke. "Evaluation of pooling operations in convolutional architectures for object recognition". *International conference on artificial neural networks*. Springer, 2010, pp. 92–101.
- [SBZ+25] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. "Flashattention-3: Fast and accurate attention with asynchrony and low-precision". *Advances in Neural Information Processing Systems* 37 (2025), pp. 68658–68685.

- [Sha48] C. E. Shannon. “A mathematical theory of communication”. *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.
- [Sha59] Claude E Shannon. “Coding theorems for a discrete source with a fidelity criterion”. *IRE Nat. Conv. Rec* 4.142-163 (1959), p. 1.
- [SMM+17] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer”. *ICLR*. 2017.
- [SZ14] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556* (2014).
- [SZ15] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. *International Conference on Learning Representations*. 2015.
- [SWM+15] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 2256–2265.
- [SME20] Jiaming Song, Chenlin Meng, and Stefano Ermon. “Denoising diffusion implicit models”. *arXiv preprint arXiv:2010.02502* (2020).
- [SE19] Yang Song and Stefano Ermon. “Generative Modeling by Estimating Gradients of the Data Distribution”. *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.
- [SSK+21] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. “Score-Based Generative Modeling through Stochastic Differential Equations”. *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [SWW12] Daniel A Spielman, Huan Wang, and John Wright. “Exact Recovery of Sparsely-Used Dictionaries”. *Proceedings of the 25th Annual Conference on Learning Theory*. Ed. by Shie Mannor, Nathan Srebro, and Robert C Williamson. Vol. 23. Proceedings of Machine Learning Research. Edinburgh, Scotland: PMLR, 2012, pp. 37.1–37.18.
- [SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [SQW15] Ju Sun, Qing Qu, and John Wright. “When are nonconvex problems not scary?” *arXiv preprint arXiv:1510.06096* (2015).
- [SQW17a] Ju Sun, Qing Qu, and John Wright. “Complete Dictionary Recovery Over the Sphere I: Overview and the Geometric Picture”. *IEEE Transactions on Information Theory* 63.2 (2017), pp. 853–884.
- [SQW17b] Ju Sun, Qing Qu, and John Wright. “Complete dictionary recovery over the sphere I: Overview and the geometric picture”. *IEEE Transactions on Information Theory* 63.2 (2017), pp. 853–884.
- [SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.
- [SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), pp. 1–9.
- [Tea] Moonshot Team.
- [Tel16] Matus Telgarsky. “Benefits of depth in neural networks”. *29th Annual Conference on Learning Theory*. Ed. by Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir. Vol. 49. Proceedings of Machine Learning Research. Columbia University, New York, New York, USA: PMLR, 2016, pp. 1517–1539.

- [Til15] Andreas M Tillmann. “On the computational intractability of exact and approximate dictionary learning”. *IEEE signal processing letters* 22.1 (Jan. 2015), pp. 45–49.
- [TB99] M. Tipping and C. Bishop. “Probabilistic principal component analysis”. *Journal of Royal Statistical Society: Series B* 61.3 (1999), pp. 611–622.
- [TZ15] Naftali Tishby and Noga Zaslavsky. “Deep learning and the information bottleneck principle”. *2015 IEEE Information Theory Workshop (ITW)*. IEEE. 2015, pp. 1–5.
- [TDC+24] Shengbang Tong, Xili Dai, Yubei Chen, Mingyang Li, ZENGYI LI, Brent Yi, Yann LeCun, and Yi Ma. “Unsupervised Learning of Structured Representation via Closed-Loop Transcription”. *Conference on Parsimony and Learning*. Ed. by Yuejie Chi, Gintare Karolina Dziugaite, Qing Qu, Atlas Wang Wang, and Zhihui Zhu. Vol. 234. Proceedings of Machine Learning Research. PMLR, Jan. 2024, pp. 440–457.
- [TDW+23] Shengbang Tong, Xili Dai, Ziyang Wu, Mingyang Li, Brent Yi, and Yi Ma. “Incremental Learning of Structured Memory via Closed-Loop Transcription”. *The Eleventh International Conference on Learning Representations*. 2023.
- [TCD+20] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. “Training data-efficient image transformers & distillation through attention. arXiv 2020”. *arXiv preprint arXiv:2012.12877* 2.3 (2020).
- [Tu07] Zhuowen Tu. “Learning Generative Models via Discriminative Approaches”. *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8.
- [Tur50] Alan Turing. “Computing Machinery and Intelligence”. *Mind* 59 (1950), pp. 433–460.
- [Tur36] Alan M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. *Proceedings of the London Mathematical Society* 2.42 (1936), pp. 230–265.
- [UVL16] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Instance normalization: The missing ingredient for fast stylization”. *arXiv preprint arXiv:1607.08022* (2016).
- [VM96] P. Van Overschee and B. de Moor. *Subspace Identification for Linear Systems*. Kluwer Academic, 1996.
- [VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. *Advances in neural information processing systems* 30 (2017).
- [VST+20] Gido M Ven, Hava T Siegelmann, Andreas S Tolias, et al. “Brain-inspired replay for continual learning with artificial neural networks”. *Nature Communications* 11.1 (2020), pp. 1–14.
- [VJO+21] Luca Venturi, Samy Jelassi, Tristan Ozuch, and Joan Bruna. “Depth separation beyond radial functions”. *Journal of machine learning research: JMLR* 23 (Feb. 2021), 122:1–122:56. eprint: [2102.01621](#).
- [Ver18] Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press, Sept. 2018.
- [VM04] R. Vidal and Y. Ma. “A unified algebraic approach to 2-D and 3-D motion segmentation”. *Proceedings of the European Conference on Computer Vision*. 2004.
- [VMS16] Rene Vidal, Yi Ma, and S. S. Sastry. *Generalized Principal Component Analysis*. 1st. Springer Publishing Company, Incorporated, 2016.
- [VMS05] Rene Vidal, Yi Ma, and Shankar Sastry. “Generalized principal component analysis”. *IEEE transactions on pattern analysis and machine intelligence* 27.12 (2005), pp. 1945–1959.
- [Vin11] Pascal Vincent. “A Connection Between Score Matching and Denoising Autoencoders”. *Neural Computation* 23.7 (2011), pp. 1661–1674.
- [WWG+12] Andrew Wagner, John Wright, Arvind Ganesh, Zihan Zhou, Hossein Mobahi, and Yi Ma. “Toward a practical face recognition system: Robust alignment and illumination by sparse representation”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.2 (2012), pp. 372–386.

- [WB68] C. Wallace and D. Boulton. “An Information Measure for Classification”. *The Computer Journal* 11 (1968), pp. 185–194.
- [WD99] C. Wallace and D. Dowe. “Minimum message length and Kolmogorov complexity”. *The Computer Journal* 42.4 (1999), pp. 270–283.
- [WF65] Marion Dwain Waltz and King-Sun Fu. “A heuristic approach to reinforcement learning control systems”. *IEEE Transactions on Automatic Control* 10 (1965), pp. 390–398.
- [WLP+24] Peng Wang, Huikang Liu, Druv Pai, Yaodong Yu, Zhihui Zhu, Qing Qu, and Yi Ma. “A Global Geometric Analysis of Maximal Coding Rate Reduction”. *arXiv preprint arXiv:2406.01909* (2024).
- [WLY+25] Peng Wang, Yifu Lu, Yaodong Yu, Druv Pai, Qing Qu, and Yi Ma. “Attention-Only Transformers via Unrolled Subspace Denoising”. *arXiv preprint arXiv:2506.03790* (2025).
- [WZZ+24] Peng Wang, Huijie Zhang, Zekai Zhang, Siyi Chen, Yi Ma, and Qing Qu. “Diffusion Models Learn Low-Dimensional Distributions via Subspace Clustering”. *arXiv preprint arXiv:2409.02426* (2024).
- [WGY+23] Xudong Wang, Rohit Girdhar, Stella X Yu, and Ishan Misra. “Cut and learn for unsupervised object detection and instance segmentation”. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 3124–3134.
- [Wer74] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD Thesis, Applied Mathematics Dept., Harvard Univ., 1974.
- [Wer94] Paul J. Werbos. “The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting”. 1994.
- [WB18] T. Wiatowski and H. Bölcskei. “A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction”. *IEEE Transactions on Information Theory* (2018).
- [Wie42] Norbert Wiener. “The interpolation, extrapolation and smoothing of stationary time series”. *Report of the Services 19, Research Project DIC-6037 MIT* (1942).
- [Wie48] Norbert Wiener. *Cybernetics: Or Control and Communication in the Animal and the Machine*. the MIT Press, 1948.
- [Wie49] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. New York: Wiley, 1949.
- [Wie61] Norbert Wiener. *Cybernetics: Or Control and Communication in the Animal and the Machine*. 2nd ed. the MIT Press, 1961.
- [WM21] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2021.
- [WM22] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2022.
- [WTL+08] John Wright, Yangyu Tao, Zhouchen Lin, Yi Ma, and Heung-Yeung Shum. “Classification via minimum incremental coding length (MICL)”. *Advances in Neural Information Processing Systems*. 2008, pp. 1633–1640.
- [WYG+09] John Wright, Allen Y. Yang, Arvind Ganesh, S. Shankar Sastry, and Yi Ma. “Robust Face Recognition via Sparse Representation”. *IEEE Trans. Pattern Anal. Mach. Intell.* 31.2 (Feb. 2009), pp. 210–227.
- [WX20] Denny Wu and J. Xu. “On the Optimal Weighted  $\ell_2$  Regularization in Overparameterized Linear Regression”. *ArXiv abs/2006.05800* (2020).
- [WTN+23] Luhuan Wu, Brian L. Trippe, Christian A Naesseth, John Patrick Cunningham, and David Blei. “Practical and Asymptotically Exact Conditional Sampling in Diffusion Models”. *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

- [WCL+24] Yuchen Wu, Minshuo Chen, Zihao Li, Mengdi Wang, and Yuting Wei. “Theoretical Insights for Diffusion Guidance: A Case Study for Gaussian Mixture Models”. *arXiv [cs.LG]* (Mar. 2024). arXiv: [2403.01639 \[cs.LG\]](#).
- [WH18] Yuxin Wu and Kaiming He. “Group normalization”. *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.
- [WDL+25] Ziyang Wu, Tianjiao Ding, Yifu Lu, Druv Pai, Jingyuan Zhang, Weida Wang, Yaodong Yu, Yi Ma, and Benjamin David Haeffele. “Token Statistics Transformer: Linear-Time Attention via Variational Rate Reduction”. *The Thirteenth International Conference on Learning Representations*. 2025.
- [XGD+17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. “Aggregated Residual Transformations for Deep Neural Networks”. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5987–5995.
- [XWC+15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. “Empirical evaluation of rectified activations in convolutional network”. *arXiv preprint arXiv:1505.00853* (2015).
- [YHB+22] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. “Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer”. *arXiv preprint arXiv:2203.03466* (2022).
- [YH21] Greg Yang and J E Hu. “Tensor Programs IV: Feature learning in infinite-width neural networks”. *International Conference on Machine Learning* 139 (2021). Ed. by Marina Meila and Tong Zhang, pp. 11727–11737.
- [YB99] Yuhong Yang and Andrew Barron. “Information-theoretic determination of minimax rates of convergence”. *Annals of statistics* 27.5 (Oct. 1999), pp. 1564–1599.
- [YYY+20] Zitong Yang, Yaodong Yu, Chong You, Jacob Steinhardt, and Yi Ma. “Rethinking Bias-Variance Trade-off for Generalization of Neural Networks”. *International Conference on Machine Learning*. 2020.
- [YYZ+24] Brent Yi, Vickie Ye, Maya Zheng, Lea Müller, Georgios Pavlakos, Yi Ma, Jitendra Malik, and Angjoo Kanazawa. “Estimating Body and Hand Motion in an Ego-sensed World”. *arXiv preprint arXiv:2410.03665* (2024).
- [YZB+23] Brent Yi, Weijia Zeng, Sam Buchanan, and Yi Ma. “Canonical Factors for Hybrid Neural Fields”. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 3414–3426.
- [YBP+24] Yaodong Yu, Sam Buchanan, Druv Pai, Tianzhe Chu, Ziyang Wu, Shengbang Tong, Hao Bai, Yuexiang Zhai, Benjamin D Haeffele, and Yi Ma. “White-Box Transformers via Sparse Rate Reduction: Compression Is All There Is?” *Journal of Machine Learning Research* 25.300 (2024), pp. 1–128.
- [YBP+23] Yaodong Yu, Sam Buchanan, Druv Pai, Tianzhe Chu, Ziyang Wu, Shengbang Tong, Benjamin Haeffele, and Yi Ma. “White-box transformers via sparse rate reduction”. *Advances in Neural Information Processing Systems* 36 (2023).
- [YCY+20] Yaodong Yu, Kwan Ho Ryan Chan, Chong You, Chaobing Song, and Yi Ma. “Learning Diverse and Discriminative Representations via the Principle of Maximal Coding Rate Reduction”. *Advances in neural information processing systems*. 2020.
- [YCO+21] Zeyu Yun, Yubei Chen, Bruno A Olshausen, and Yann LeCun. “Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors”. *arXiv preprint arXiv:2103.15949* (2021).
- [ZKR+17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. “Deep Sets”. *Advances in Neural Information Processing Systems* 30. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 3391–3401.

- [ZAK24] Moslem Zamani, Hadi Abbaszadehpeivasti, and Etienne de Klerk. “Convergence rate analysis of the gradient descent–ascent method for convex–concave saddle-point problems”. *Optimization Methods and Software* 39.5 (2024), pp. 967–989.
- [MZM+20] Yuexiang Zhai, Hermish Mehta, Zhengyuan Zhou, and Yi Ma. “Understanding l4-based Dictionary Learning: Interpretation, Stability, and Robustness”. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
- [ZCB+24] Bingliang Zhang, Wenda Chu, Julius Berner, Chenlin Meng, Anima Anandkumar, and Yang Song. “Improving diffusion inverse problem solving with decoupled noise annealing”. *arXiv [cs.LG]* (July 2024). arXiv: 2407.01521 [cs.LG].
- [ZBH+17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. “Understanding deep learning requires rethinking generalization”. *International Conference on Learning Representations*. 2017.
- [ZLG+10] Zhengdong Zhang, Xiao Liang, Arvind Ganesh, and Yi Ma. “TILT: Transform Invariant Low-Rank Textures”. *International Journal of Computer Vision* 99 (2010), pp. 1–24.
- [ZLC+23] Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, BIN FU, Tao Chen, Gang YU, and Shenghua Gao. “Michelangelo: Conditional 3D Shape Generation based on Shape-Image-Text Aligned Latent Representation”. *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.
- [ZCZ+25] Hongkai Zheng, Wenda Chu, Bingliang Zhang, Zihui Wu, Austin Wang, Berthy Feng, Caifeng Zou, Yu Sun, Nikola Borislavov Kovachki, Zachary E Ross, Katherine Bouman, and Yisong Yue. “InverseBench: Benchmarking Plug-and-Play Diffusion Priors for Inverse Problems in Physical Sciences”. *The Thirteenth International Conference on Learning Representations*. 2025.
- [ZLG+] Ziyan Zheng, Chin Wa Lau, Nian Guo, Xiang Shi, and Shao-Lun Huang. “White-box error correction code transformer”. *The Second Conference on Parsimony and Learning (Proceedings Track)*.
- [ZM97a] Song Chun Zhu and David Mumford. “Prior Learning and Gibbs Reaction-Diffusion”. *IEEE Trans. Pattern Anal. Mach. Intell.* 19.11 (1997), pp. 1236–1250.
- [ZWM97] Song Chun Zhu, Ying Nian Wu, and David Mumford. “Minimax Entropy Principle and Its Application to Texture Modeling”. *Neural Computation* 9.8 (1997), pp. 1627–1660.
- [ZM97b] Song-Chun Zhu and David Mumford. “Learning generic prior models for visual computation”. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (1997), pp. 463–469.
- [ZDZ+21] Zhihui Zhu, Tianyu Ding, Jinxin Zhou, Xiao Li, Chong You, Jeremias Sulam, and Qing Qu. “A geometric analysis of neural collapse with unconstrained features”. *Advances in Neural Information Processing Systems* 34 (2021), pp. 29820–29834.
- [ZL17] Barret Zoph and Quoc V. Le. “Neural Architecture Search with Reinforcement Learning”. 2017.