

Chapter 1 Introduction

Knowledge base approach to A.I.

a computer can reason automatically about statements
in formal languages using logical inference rules

Machine Learning

A.I. systems acquire our knowledge by extracting patterns from raw data

Representation Learning

use M.L. to discover not only the mapping from the output,
but also the representation itself.

→ discover a good set of features by itself ← e.g. represent data by

→ e.g. Autoencoder (encoder = input → representation
decoder = representation → output) ← Cartesian coordinates vs Polar coordinates

Deep Learning

→ solves representation learning by introducing representations expressed in simpler representations

→ depth enables the computer to learn a multi-step computer program

↳ measured either by the depth of the computational graph
or the depth of the probabilistic graph

→ involve a greater amount of
composition of either learned functions
or learned concepts than traditional M.L.

This book

1. Introduction

Part I

2. Linear Algebra → 3. Probability & Information Theory
↓ ↓
4. Numerical Computation → 5. Machine Learning Basics

Part II

6. Deep Feedforward Networks
7. Regularization
8. Optimization
9. CNNs
10. RNNs
↓
11. Practical Methodology
12. Applications

Part III

13. Linear Factor Model → 14. Autoencoders → 15. Representation Learning
↓ |
16. Structured Probabilistic Models → 17. Monte Carlo Methods
↓ |
19. Inference → 20. Deep Generative Models ← 18. Partition Function

Part II: established DL algorithms

Part III: speculative ideas

1.1 Who should read this book?1.2 Historical Trends in Deep Learning1.2.1 The many names and changing fortunes of Neural Networks

Cybernetics

Connectionism / Parallel distributed processing
↳ distributed representation1.2.2 Increasing Dataset Sizes1.2.3 Increasing Model Sizes1.2.4 Increasing Accuracy, Complexity, and Real-World ImpactChapter 2 Linear Algebra2.1 Scalars, Vectors, Matrices and Tensors

Tensors: arrays with more than 2 axes

Broadcasting: copying a vector to each row of a matrix

e.g. $C = A + \vec{b} \Rightarrow C_{ij} = A_{ij} + b_j$

2.2 Multiplying Matrices and VectorsMatrix Product AB vs Element-wise / Hadamard product $A \odot B$

$C_{ij} = \sum_k A_{ik} B_{kj}$ ↳ dot product of row i of A and column j of B

Dot product between two vectors $\vec{x}, \vec{y} = \vec{x}^T \vec{y}$ 2.3 Identity and Inverse Matrices

$A \vec{x} = \vec{b} \rightarrow \vec{x} = A^{-1} \vec{b}$ inverse matrix can solve the equation
many times with different \vec{b}

2.4 Linear Dependence and Span

A^{-1} exists only when exactly one solution \vec{x} exists for each \vec{b}
 (might have no solution
 or infinitely many solutions)
 → suppose \vec{x}, \vec{y} solution, then $a\vec{x} + (1-a)\vec{y}$ is a solution

 $A \vec{x} = \vec{b} \rightarrow$ columns of A specify different directions to travel from the originnumber of solutions $\vec{x} \rightarrow$ determine how many ways to reach \vec{b}
 each $x_i \rightarrow$ how far to move in the direction of column i decide whether solution \vec{x} exists →test whether \vec{b} is inthe span of
columns of A span of $\vec{v}^{(i)} =$
set of all points obtainable bylinear combination of $\vec{v}^{(i)}$

Column space / range of A

\leftarrow this particular span

Linear combination of v_1, v_2, \dots, v_n

- for $A\vec{x} = \vec{b}$ to have a solution for any $\vec{b} \in \mathbb{R}^m$ requires column space of $A = \mathbb{R}^m$
- $\rightarrow A$ contains at least one set of m linearly independent columns
 necessary and sufficient conditions
- far A^{-1} exists, must have at most one solution
- $\rightarrow A$ contains at most m columns
- $\rightarrow A$ is square and non-singular
- If linear dependent columns $\rightarrow A$ is singular
- If A is not square or is singular,
 can still solve $A\vec{x} = \vec{b}$, but cannot use matrix inversion
 since A^{-1} does not exist

2.5 Norms

Measure size of a vector

$$L^p \text{ norm: } \|\vec{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

Common:

$$\text{squared } L^2 \text{ norm: } \vec{x}^T \vec{x}$$

\hookrightarrow easy to work with computationally = e.g. each derivative depends elementwisely instead of the entire vector

\hookrightarrow problem: increase very slowly

near the origin: cannot discriminate between exactly zero vs small but non-zero elements

$$L^1 \text{ norm: } \|\vec{x}\|_1 = \sum_i |x_i|$$

vs small but non-zero elements

when this difference between zero vs non-zero elements is important

" L^0 norm": # of non-zero entries

incorrect terminology because this is not a norm,
 e.g. scaling by a does not change the number

$$L^\infty \text{ norm: max norm } \|\vec{x}\|_\infty = \max_i |x_i|$$

$$\text{Frobenius norm: } \|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$$

measures size of a matrix

analogous to the L^2 norm of vector

$$\text{Dot product } \vec{x}^T \vec{y} = \|\vec{x}\|_2 \|\vec{y}\|_2 \cos \theta \quad \text{angle between } \vec{x} \text{ and } \vec{y}$$

2.6 Special kinds of Matrices and Vectors

Diagonal matrix

Symmetric matrix

Unit vector : vector with unit norm $\|x\|_2 = 1$

\vec{x} and \vec{y} are orthogonal to each other if $\vec{x}^T \vec{y} = 0$

↳ if \vec{x} and \vec{y} are unit vectors, they are orthonormal

↳ in \mathbb{R}^n at most n vectors are mutually orthogonal with non-zero norms

Orthogonal matrix $A^T A = A A^T = I \Rightarrow A^{-1} = A^T$ easy to compute inverse matrix

square matrix, rows are mutually orthonormal

+ columns are mutually orthonormal

2.7 Eigendecomposition

Similar to integers decomposing to prime factors
(invariant to the representation "10" in base 10)

help us understand certain properties of integers / matrices

~ Decomposing matrices by eigendecomposition

$$A \vec{v} = \lambda \vec{v}$$

* Square matrix \vec{v} non-zero multiplication by A
 (right) Eigenvector only alters the scale of \vec{v} $\vec{v}^T A = \lambda \vec{v}^T$
 Eigenvalue left eigenvector

Eigendecomposition

$$A = V \text{ diag}(\lambda) V^{-1}$$

Suppose A has
n lin. indep. eigenvectors,
Concatenate to form a matrix
(one eigenvector per column)

Concatenate the eigenvalues to form
 $\lambda = [\lambda_1, \dots, \lambda_n]$
and get $\text{diag}[\lambda]$

$$V = [\vec{v}^{(1)}, \dots, \vec{v}^{(n)}]$$

Every real symmetric matrix can be decomposed into real-valued eigenvectors

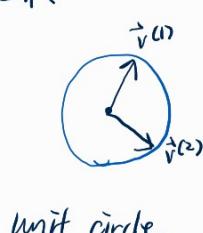
$$A = Q \Lambda Q^T$$

orthogonal

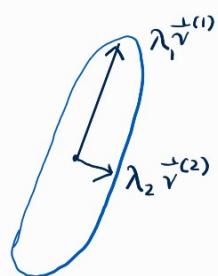
eigenvalue $\lambda_{i,i}$ associated with eigenvector $Q_{i,i}$
decomposition is unique only if all eigenvalues are unique

|| Suppose two eigenvectors share the same eigenvalue
then any linear combination gives a new eigenvector
with this eigenvalue

e.g. $A \in \mathbb{R}^{2 \times 2}$



unit circle



$A \vec{v}$: distort the unit circle

→ scale the space in direction $\vec{v}^{(1)}$
by λ_i

2.7 Optimize quadratic expressions

$$f(\vec{x}) = \vec{x}^T A \vec{x}$$

Subject to $\|\vec{x}\|_2 = 1$

Whenever \vec{x} is an eigenvector of A ,
f takes on the corresponding eigenvalue

$\max(f)$ within the constraint region

= max eigenvalue (similarly for $\min(f) = \min$ eigenvalue)

Positive definite \rightarrow eigenvalues all positive

Positive semidefinite \rightarrow eigenvalues all positive or zero-valued
guaranteed $\forall \vec{x}, \vec{x}^T A \vec{x} \geq 0$

2.8 Singular Value Decomposition (SVD)

Singular Value Decomposition another matrix factorization

into Singular vectors and Singular values

$$A = UDV^T$$

$m \times n$ $m \times m$ $n \times n$
 orthogonal diagonal orthogonal
 left-singular (may not right-singular
 vectors be square) vectors

Interpret SVD as
eigen decomposition of
functions of A

- * discovers some of the same kind of info from eigendecomposition
- * every real matrix has a SVD (more general)
 - e.g. not square matrix \rightarrow no eigendecomposition
- * Partially generalize matrix inversion to non-square matrix

2.9 The Moore-Penrose Pseudoinverse

Suppose want to find B s.t. $\text{left-inverse of } A$

$$A\vec{x} = \vec{y} \Rightarrow \vec{x} = B\vec{y}$$

$m \times n$

If $m > n \rightarrow$ may have no solution $\boxed{m \times n}$

If $m < n \rightarrow$ may have multiple solutions $\boxed{m \times n}$

Moore-Penrose Pseudoinverse

$$A^+ = \lim_{\alpha \rightarrow 0} (A^T A + \alpha I)^{-1} A^T$$

practical computation = $V D^+ U^T$ where U, V from singular value decomposition

D^+ = take reciprocal of D's non-zero elements,
then take transpose

When $m < n$, $\vec{x} = A^+ \vec{y}$ provides one of many solutions
with minimal Euclidean norm $\|\vec{x}\|_2$

When $m > n$, $\vec{x} = A^+ \vec{y}$ is the solution for $\arg \min_{\vec{x}} \|\vec{x}\|_2$
closest possible solution to \vec{y}

2.10 The Trace Operator

$T(A) = \sum_{i,j} a_{ij}$ trace of A = sum of diagonal elements

$\text{In}(A) = \sum_i A_{i,i}$ the sum of all diagonal entries of A

e.g. Frobenius norm: $\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} = \sqrt{\text{Tr}(AA^T)}$

Invariant to cyclic permutation

e.g. $\text{Tr}(ABC) = \text{Tr}(BCA) = \text{Tr}(CAB)$

2.11

Determinant

$\det(A) = \prod$ eigenvalues of matrix
product of

$|\det(A)|$ measures how much multiplication by the matrix
absolute value expands or contracts space

$\hookrightarrow |\det A| = 0$: space contracted completely along at least one dim
(lose all volume)

$\hookrightarrow |\det A| = 1$: the transformation preserves volume

2.12

Example: Principal Components Analysis (PCA)

A simple machine learning algorithm

Suppose: m points $\{\vec{x}^{(1)}, \dots, \vec{x}^{(m)}\}$ in \mathbb{R}^n

Want = apply lossy compression (store points with less memory)
 \downarrow but lose some precision
lose as little precision as possible

Want = find some encoding function to produce code vector $\vec{c}^{(i)} \in \mathbb{R}^l$ where $l < n$
 $f(\vec{x}) = \vec{c}$
and a decoding function that produces the reconstructed input given its code
 $g(\vec{c}) = g(f(\vec{x})) \approx \vec{x}$

Simple PCA: matrix multiplication for decoding =

let $g(\vec{c}) = D\vec{c}$ where $D \in \mathbb{R}^{n \times l}$

PCA: constrains the columns

to be orthogonal to each other

for unique solution

columns of D have unit norm

(may not be orthogonal matrix since may not be square)

Step ①: generate optimal code point \vec{c}^* for each \vec{x}

In PCA: L^2 norm $\rightarrow c^* = \underset{\vec{c}}{\arg\min} \|\vec{x} - g(\vec{c})\|_2$
or \downarrow same

squared L^2 norm $\rightarrow \underset{\vec{c}}{\arg\min} \|\vec{x} - g(\vec{c})\|_2^2 = \underset{\vec{c}}{\arg\min} \|\vec{x} - D\vec{c}\|_2^2$

solving: $\vec{c}^* = D^T \vec{x}$ $\because f(\vec{x}) = D^T \vec{x}$

Step ②: PCA reconstruction for the decoding step

$g(\vec{c}) = D D^T \vec{x}$

(similar idea)

Step ③: Choose encoding matrix D

$D^* = \underset{D}{\arg\min} \sqrt{\sum_{i,j} (x_j^{(i)} - g(f(\vec{x}^{(i)}))_j)^2}$ Frobenius norm
to compute matrix error

subject to $D^T D = I_l$

$$\begin{aligned} \vec{d}^* &= \underset{\vec{d}}{\operatorname{argmin}} \sum_i \left\| \frac{\vec{x}^{(i)}}{\|\vec{x}^{(i)}\|_2} - \frac{\vec{d}^T \vec{d}}{\|\vec{d}\|_2} \right\|_2^2 & \parallel \vec{d}^* &= \underset{\vec{d}}{\operatorname{argmin}} \|X - X \vec{d} \vec{d}^T\|_F^2 \\ \text{subject to } \|\vec{d}\|_2 &= 1 & \parallel \text{subject to } \vec{d}^T \vec{d} = 1 & \text{where } X_{i,:} = \frac{\vec{x}^{(i)T}}{\|\vec{x}^{(i)}\|_2} \end{aligned}$$

$$\vec{d}^* = \underset{\vec{d}}{\operatorname{argmax}} \operatorname{Tr}(\vec{d}^T X^T X \vec{d}) \text{ subject to } \vec{d}^T \vec{d} = 1$$

solution: \vec{d}^* = eigenvector of $X^T X$ with the largest eigenvalue
recovers only the first principal component

Chapter 3 Probability and Information Theory

Probability theory: means of quantifying uncertainty + axioms for deriving new uncertainty statements

- A.I. application:
- ① laws of probability \rightarrow how AI systems should respond
 - ② theoretically analyze the behavior of proposed AI systems

Information theory: quantify the amount of uncertainty in a probability distribution

3.1 Why Probability?

- 3 possible sources of uncertainty:
- ① Inherent stochasticity of the system
 - ② Incomplete observability
e.g. Monty Hall problem (3 doors win a prize)
 - ③ Incomplete modelling
treat as if same

Frequentist probability vs Bayesian probability

if an experiment happen many times, with probability p an event happen

*degree of belief: 1 absolute yes, 0 absolute no
qualitative levels of uncertainty*

3.2 Random Variables

Random variable \rightarrow variable that can take on different values randomly

e.g. X random variable, x_1, x_2 the possible values

- \rightarrow just a description of the states that are possible
- \rightarrow must be coupled with a probability distribution that specifies how likely each of these states are
- \rightarrow discrete/continuous

3.3 Probability Distributions

probability distribution \rightarrow describe how likely a random variable / set of random variables is to take on each of its possible states

3.3.1 Discrete variables and Probability Mass Functions

over discrete variables \rightarrow described with PMF (probability mass functions)

$\rightarrow P(x) \neq P(y)$ usually (different PMF)

to distinguish which PMF, write $P(x = x)$
or $x \sim P(x)$

over many variables \rightarrow joint probability distribution e.g. $P(x=x, y=y)$

PMF = $\forall x \in X, 0 \leq P(x) \leq 1, \sum_{x \in X} P(x) = 1$ (normalized) or $P(x, y)$

3.3.2 Continuous variables and Probability Density Functions

over Continuous variables \rightarrow described with PDF (probability density functions)

\rightarrow the probability of landing inside an infinitesimal region

PDF = $\forall x \in X, p(x) \geq 0, \int p(x) dx = 1$ with volume dx
not necessary $p(x) \leq 1$ is given by $p(x)dx$

the prob of x lies in some set S is given by the integral of $p(x)$ over that set.

e.g. for x lies in $[a, b]$, the prob is $\int_{[a,b]} p(x) dx$

e.g. Uniform Distribution on an interval of real numbers

$u(x; a, b)$ for $[a, b]$ with $b > a$

$\hookrightarrow u(x; a, b) = 0$ for all $x \notin [a, b]$ } non-negative

$\hookrightarrow u(x; a, b) = \frac{1}{b-a}$ for all $x \in [a, b]$ } integrates to 1

$x \sim U(a, b)$

3.4 Marginal Probability

To know the probability distribution over just a subset of variables

e.g. $\forall x \in X, P(x=x) = \sum_y P(x=x, y=y)$

or $p(x) = \int p(x, y) dy$

(imagine writing a grid to enumerate all possibles $P(x, y)$, then write in the margin for the sum of the row $p(x)$)

3.5 Conditional Probability

$P(y=y | x=x) = \frac{P(y=y, x=x)}{P(x=x)}$ only defined when $\neq 0$
 y conditions on x happens because cannot compute if event does not happen

do not confuse with Causal Modelling / Intervention Query

3.6 The Chain Rule of Conditional Probabilities

consequences of making an action

Chain Rule / Product Rule of Probability

$$P(x^{(1)}, \dots, x^{(n)}) = P(x^{(1)}) \cdot \prod_{i=2}^n P(x^{(i)} | x^{(1)}, \dots, x^{(i-1)})$$

3.7 Independence & Conditional Independence

Two random variables x and y are independent

if their probability distribution:

$P(x, y) = P(x)P(y)$ (indicates independence)

$$\forall x \in X, y \in Y \quad p(x=x, y=y) = p(x=x) p(y=y) \quad x \perp y$$

————— || ————— one conditionally independent

$$\text{if } \forall x \in X, y \in Y, z \in Z \quad p(x=x, y=y | z=z) = p(x=x | z=z) p(y=y | z=z) \rightarrow x \perp y | z$$

3.8 Expectation, Variance & Covariance

Expected value of some function $f(x)$ w.r.t. $P(x)$

| | | |
|--|---|-------------------|
| $\mathbb{E}_{x \sim P}[f(x)] =$ | $\sum_x P(x)f(x)$ | $\int p(x)f(x)dx$ |
| $= \bar{E}_x[f(x)] = \mathbb{E}[f(x)]$ | $\text{Expectation is linear e.g. } \mathbb{E}_x[\alpha f(x) + \beta g(x)] = \dots$ | |
| | $\mathbb{E}[\cdot] = \text{averages over the values of all the random variables inside the brackets}$ | |

Variance

measures how much the function values of a random variable x vary as diff values x are sampled from its probability distribution

$$\text{Var}(f(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$$

expected difference
between $f(x)$ and expected $f(x)$

$$\sqrt{\text{Var}(f(x))} : \text{Standard Deviation}$$

Covariance

measures how much two values are linearly related & the scale

$$\text{Cov}(f(x), g(y)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])]$$

expected difference
(between $f(x)$ and expected $f(x)$) \times (between $g(y)$ and expected $g(y)$)

→ Correlation normalizes the contribution of each variable

→ Zero Covariance → no linear dependence for the two variables

Independence → above & no non-linear relationships -
e.g. stronger requirement

$$\text{continuous } x \in [-1, 1], f(x) = u(x) = \frac{1}{2}$$

$$\text{discrete } S = \{-1, 1\}, g(s=-1) = g(s=1) = \frac{1}{2}$$

let $y = sx$ be a random variable

$$\text{Cov}(x, y) = 0 \text{ but } x \text{ completely determines } y$$

Covariance Matrix of a random vector $x \in \mathbb{R}^n$

$$\text{Cov}(\vec{x})_{ij} = \text{Cov}(\vec{x}_i, \vec{x}_j) \quad \text{diag cov}(\vec{x}_i, \vec{x}_i) = \text{Var}(\vec{x}_i)$$

3.9 Common Probability Distributions

3.9.1 Bernoulli Distribution

→ over a single binary random variable

Choose ϕ to be controlled by $\phi \in [0,1]$

Logistic sigmoid

$$\delta(x) = \frac{1}{1 + \exp(-x)}$$

$$P(x=1) = \phi$$

$$P(x=0) = 1 - \phi$$

$$P(x=x) = \phi^x (1-\phi)^{1-x}$$

$$\mathbb{E}[x] = \phi$$

$$\text{Var}(x) = \phi(1-\phi)$$

3.9.2 Multinoulli Distribution (Categorical Distribution)

→ over a single discrete variable with k different states (finite)

Predict probabilities with

Softmax function

$$\text{ith: } \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

↳ special case of multinomial distribution

($n=1$ for how many times each of the k categories is visited when n samples are drawn from a multinoulli distribution)

→ parametrized by $\vec{p} \in [0,1]^k$ p_i gives the prob of the i th state

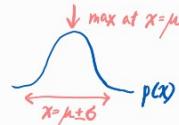
the prob of final state = $1 - \vec{1}^T \vec{p}$ must constraint $\vec{1}^T \vec{p} \leq 1$

3.9.3 Gaussian Distribution (Normal Distribution)

→ commonly use over real numbers

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

where $\mu \in \mathbb{R}$, $\sigma \in (0, \infty)$ $\mathbb{E}(x) = \mu$ $\text{Var}(x) = \sigma^2$

Choose μ, β to be

Softplus function

$$\xi(x) = \log(1 + \exp(x))$$

→ To control precision = $\beta \in (0, \infty)$

$$\mathcal{N}(x; \mu, \frac{1}{\beta}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x-\mu)^2\right)$$

★ Central Limit Theorem

the sum of many independent variables \sim normally distributed

★ Encodes the max. amount of uncertainty
out of all possible prob. distributions with the same variance

→ Multivariate Normal Distribution (generalize to \mathbb{R}^n)

$$\mathcal{N}(\vec{x}; \vec{\mu}, \Sigma) = \sqrt{\frac{1}{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})\right)$$

where Σ a positive definite symmetric matrix

$$\mathbb{E}(\vec{x}) = \vec{\mu}$$

$$\text{Cov}(\vec{x}) = \Sigma$$

→ Simpler = fix Σ diagonal control variance along axis-aligned direction

Set $\text{Cov}(\vec{x}) = \alpha I$ (isotropic Gaussian distribution)

some variance along each direction

3.9.4 Exponential and Laplace Distributions

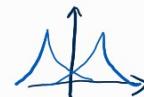
Exponential distribution

$$p(x; \lambda) = \lambda \int_{x>0} \exp(-\lambda x)$$

sharp point at $x=0$ 

Laplace distribution

$$\text{Laplace}(x; \mu, \sigma) = \frac{1}{2\sigma} \exp\left(-\frac{|x-\mu|}{\sigma}\right)$$

sharp peak of probability mass at arbitrary μ 

3.9.5

Ti D P r i T p r o b a b i l i t y

3.1.1 The Dirac Distribution & Empirical Distribution

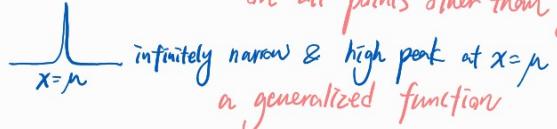
→ specify all the mass in a probability distribution clusters around a single point

→ a PDF with the Dirac delta function $\delta(x) =$

$$p(x) = \delta(x - \mu) \quad \text{limit point of a series of functions}$$

→ zero-valued everywhere except $x = \mu = 0$ that put less & less density

→ integrate to 1 on all points other than zero



Empirical Distribution

Over continuous variables: use Dirac delta distribution

$$\hat{p}(\vec{x}) = \frac{1}{m} \sum_{i=1}^m \delta(\vec{x} - \vec{x}^{(i)})$$

put probability mass $\frac{1}{m}$ on each of the m points $\vec{x}^{(1)}, \dots, \vec{x}^{(m)}$
forming the given collection of samples

Over discrete variables:

a multinoulli distribution whose probability with each value equal to the empirical frequency of that in the training set

* the probability density that maximizes the likelihood of the training data

3.9.6 Mixtures of Distributions

one way of combining probability distributions

Mixture Distribution

Generate the sample x :

$$P(x) = \sum_i P(c=i) P(x | c=i)$$

where $P(c)$: the multinoulli distribution over component identities

E.g. the empirical distribution over real-valued variables

= mixture distribution with one Dirac component for each sample

Latent variable

a random variable that we cannot observe directly

e.g. component identity variable c

Gaussian mixture model

$p(\vec{x} | c=i)$ are Gaussian distributions — each has $\mu^{(i)}$ and $\Sigma^{(i)}$

→ Some mixtures can have more constraints

e.g. shared covariance $\Sigma^{(i)} = \Sigma \forall i$

→ the parameters of the Gaussian mixture specify

the prior probability $\alpha_i = P(c=i)$ for each component i before observing x

the posterior probability $P(c|\vec{x})$ after the observation of x

→ a Universal Approximator

can approximate any smooth density within specified error
with enough components

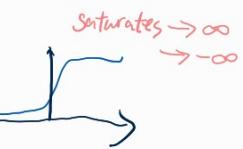
3.10 Useful Properties of Common Functions

Logistic sigmoid

$$\delta(x) = \frac{1}{1 + \exp(-x)}$$

Range is $(0, 1)$

✓ produce ϕ in Bernoulli distribution



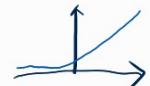
Softplus function

$$\zeta(x) = \log(1 + \exp(x))$$

Range is $(0, \infty)$

✓ produce β or δ of Gaussian distribution

* Softplus version of $x^+ = \max(0, x)$



$$\delta(x) = \frac{\exp(x)}{1 + \exp(x)}$$

$$\frac{d}{dx} \delta(x) = \delta(x) \cdot (1 - \delta(x))$$

$$1 - \delta(x) = \delta(-x)$$

$$\forall x \in (0, 1), \delta^{-1}(x) = \log\left(\frac{x}{1-x}\right)$$

$$\log \delta(x) = -\zeta(-x)$$

$$\frac{d}{dx} \zeta(x) = \delta(x)$$

$$\zeta(x) = \int_{-\infty}^x \delta(y) dy$$

e.g. $x^+ + x^- = x$

$$\zeta(x) - \zeta(-x) = x$$

$$\forall x > 0, \zeta^{-1}(x) = \log(\exp(x) - 1)$$

3.11 Bayes' Rule

$$\text{Bayes' Rule } p(x|y) = \frac{p(x)p(y|x)}{p(y)}$$

3.12 Technical Details of Continuous Variables

Measure Theory: provides a characterization of the set of sets that can compute probability without paradoxes

e.g. by infinite precision of real numbers/
fractal-shaped sets / transformation of rational numbers

e.g. The Banach-Tarski theorem 

provides a rigorous way of describing
a set of points that is negligibly small i.e. measure zero
occupies no volume e.g. line in \mathbb{R}^2

Almost Everywhere: Some important theory holds for discrete values
but hold "almost everywhere" only for continuous values

Handle Continuous Random Variables that are deterministic functions of one another

$$\vec{y} = g(\vec{x}) \xrightarrow{\text{assume invertible } g} p_y(\vec{y}) = p_x(g^{-1}(\vec{y}))$$

$$p_x(x) = p_y(g(x)) \left| \frac{\partial g(x)}{\partial x} \right| \quad | \quad p_x(\vec{x}) = p_y(g(\vec{x})) \left| \det \left(\frac{\partial g(\vec{x})}{\partial \vec{x}} \right) \right|$$

det of Jacobian matrix

$$J_{i,j} = \frac{\partial x_i}{\partial y_j}$$

3.13 Information Theory

Quantity how much info is present in a signal

application: send msg from discrete alphabets over noisy channel

Communication via radio transmission

Information theory design optimal code

in ML = to continuous variables

calculate expected length of msg. sampled from different probability distributions with diff encoding schemes

characterize probability distributions
quantify similarity between probability distributions

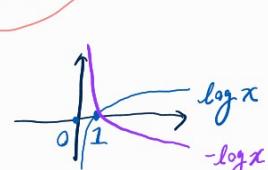
Basic Intuition:

learning that an unlikely event has occurredis more informative than a likely event has occurred

- quantify st. an event guarantee to happen
= no information content
- less likely event → higher information content
- independent events → additive information content

Self-information

for an event $x = x$: $I(x) = -\log P(x)$



Units of nats = one nat = $I(x)=1 \Leftrightarrow P(x)=\frac{1}{e}$
one bit / shannon $\Leftrightarrow P(x)=\frac{1}{2}$

for continuous variable,

zero information → an event with unit density
(instead of must occur)

deals with single outcome only

Shannon entropy

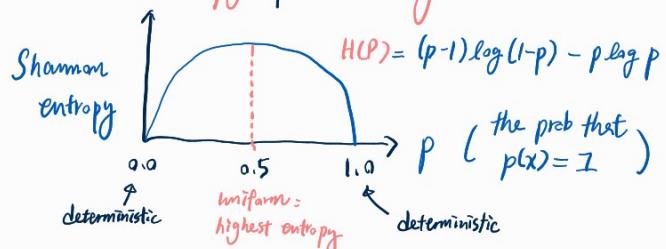
the amount of uncertainty in the entire probability distribution

$$H(x) = \mathbb{E}_{x \sim p} [I(x)] \\ = -\mathbb{E}_{x \sim p} [\log P(x)] = H(p)$$

i.e. the expected amount of information in an event drawn from that distribution

* the lower-bound on # bits on average to encode symbols drawn from that distribution

e.g. Shannon entropy of a binary random variable

When x is continuous → Differential Entropy

Kullback-Leibler (KL) divergence

Same random variable x , different probability distribution → measure difference

$$D_{KL}(P \parallel Q) = \mathbb{E}_{x \sim P} \left[\log \frac{P(x)}{Q(x)} \right] = \mathbb{E}_{x \sim P} [\underline{\log P(x)} - \log Q(x)]$$

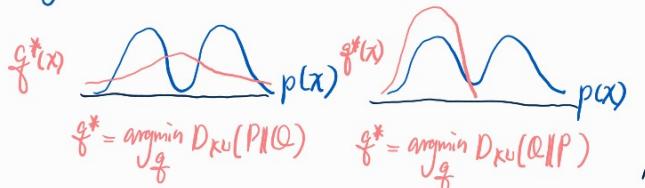
In ML =

the extra amount of information needed (in nats/bits)
to send a message containing symbols drawn from P , when we
use a code that was designed to minimize the length of messages drawn from Q

$D_{KL}(P \parallel Q) = 0 \rightarrow$ same distribution (discrete variables)

non-negative or equal almost everywhere (continuous variables)

asymmetric: $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$ not a distance measure!



select depends on application
left: find q s.t.
high $p(x) \rightarrow$ high $q(x)$
right: find q s.t.
low $p(x) \rightarrow$ low $q(x)$

Cross-Entropy

$$H(P, Q) = H(P) + D_{KL}(P \parallel Q) \quad & \text{similar to KL Divergence} \\ = - \mathbb{E}_{x \sim P} [\log Q(x)] \quad & \text{without the term on the left}$$

* minimize cross-entropy w.r.t. Q = minimize the KL Divergence

$O \log Q$

in information theory, treat as $\lim_{x \rightarrow 0} x \log x = 0$

3.14 Structured Probabilistic Models

Motivation: don't use one single function to represent a probability distribution,
split it to multiple ones by factorization over fewer variables

e.g. $p(a, b, c) = \underbrace{p(a)}_{\text{reduce cost of representing}} \cdot \underbrace{p(b|a)}_{\text{a distribution}} \cdot \underbrace{p(c|b)}_{\text{a distribution}}$

Structured Probabilistic Model / Graphical Model

represent the factorization with a graph (in context of graph theory)

Graph G : node \rightarrow random variable

only a description edge \rightarrow the probability distribution is able to represent
of prob distribution, not exclusive direct interaction between the two random variables

Directed Model to each other

$$p(\vec{x}) = \prod_i p(x_i | \text{parent}_G(x_i))$$

$$\begin{array}{c} \textcircled{a} \rightarrow \textcircled{b} \\ \downarrow \quad \downarrow \\ \textcircled{c} \quad \textcircled{d} \\ \downarrow \\ \textcircled{e} \end{array} \quad \begin{aligned} p(a, b, c, d, e) \\ = p(a) \cdot p(b|a) \\ \cdot p(c|a, b) \cdot p(d|b) \\ \cdot p(e|c) \end{aligned}$$

Undirected Model

$$p(\vec{x}) = \frac{1}{Z} \prod_i \phi^{(i)}(C^{(i)})$$

$$\begin{array}{c} \textcircled{a} \text{---} \textcircled{b} \\ \textcircled{a} \text{---} \textcircled{c} \\ \textcircled{c} \text{---} \textcircled{d} \\ \textcircled{c} \text{---} \textcircled{e} \\ \textcircled{d} \text{---} \textcircled{e} \end{array} \quad \begin{aligned} p(a, b, c, d, e) \\ = \frac{1}{Z} \phi^{(1)}(a, b, c) \cdot \phi^{(2)}(b, d) \cdot \phi^{(3)}(c, e) \end{aligned}$$

Clique $C^{(i)}$: set of nodes connecting to each other

associated with a factor $\phi^{(i)}(C^{(i)})$

factor: just functions
 \neq probability distribution

$$\frac{1}{Z} = \text{normalizing}$$

\rightarrow non-negative

\rightarrow no negative + ...

Chapter 4 Numerical Computation

Algorithms that solve math problems by methods that update estimates of the solution via an iterative process

4.1 Overflows and Underflow

Real numbers \rightarrow approximation error \rightarrow rounding error

Underflow = e.g. division by zero \rightarrow not-a-number error

Overflow $> \infty$ or $-\infty \rightarrow$ not-a-number error

Softmax function

predict probabilities associated with a multinoulli distribution

$$\text{softmax}(\vec{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

prone to underflows/overflows

e.g. suppose all $x_i = c \Rightarrow \text{softmax} = \frac{\exp(c)}{\sum \exp(c)} = \frac{1}{n}$ analytically
but when c is very negative/positive

$\rightarrow \exp(c)$ underflow $\rightarrow \text{softmax} = \frac{0}{0}$
/ overflow $\rightarrow \text{softmax} = \frac{\infty}{\infty} >$ undefined

e.g. suppose get softmax = 0 $\Rightarrow \log(\text{softmax}) = \log(0) \rightarrow$ undefined

Solution:

evaluate $\text{softmax}(\vec{z})$ where $\vec{z} = \vec{x} - \max_i x_i$; implement $\log(\text{softmax})$
in a numerically stable way

4.2 Poor Conditioning

Conditioning = how rapidly a function changes w.r.t. small changes in its inputs perturbed slightly

Condition number of $A \in \mathbb{R}^{n \times n}$ for $f(\vec{x}) = A^{-1}(\vec{x})$

$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$ ratio of max eigenvalue to min eigenvalue \rightarrow large
intrinsic property \rightarrow sensitive to errors in inputs

4.3 Gradient-based Optimization

Objective function = the function we wish to maximize/minimize

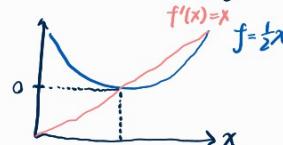
Cost / Loss / Error function = when minimizing

* optimization
in continuous space

Gradient Descent:

minimize $f(\vec{x})$ by moving \vec{x} with the opposite sign of the derivative in small steps

vs hill climbing



$$\vec{x}' = \vec{x} - \epsilon \nabla_{\vec{x}} f(\vec{x})$$

Steepest descent.

(in Discrete space)

\leftarrow post course
 $<0: x \uparrow \uparrow >0: x \downarrow \downarrow$

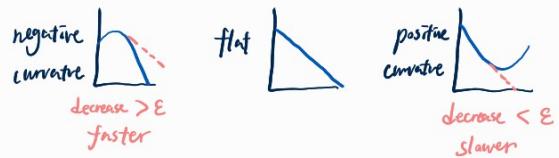
Critical point = local minimum/maximum, saddle point ↗

Line search = evaluate several ε (learning rate)

4.3.1 Beyond the Gradient: Jacobian & Hessian Matrices

Jacobian matrix: $J_{i,j} = \frac{\partial}{\partial x_j} f(\vec{x})_i$ first derivative

Hessian matrix: $H(f)(\vec{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\vec{x})$ second derivative
 ↳ measures curvature



↳ symmetric (and real) → eigen decomposition

Taylor Series approximation

Second order, with $\vec{x} = \vec{x}^{(0)} - \varepsilon \cdot \vec{g}$ \vec{g} = gradient

$$f(\vec{x}) \approx \underbrace{f(\vec{x}^{(0)})}_{\text{original value}} - \underbrace{\varepsilon \vec{g}^T \vec{g}}_{\text{expected improvement from slope of the function}} + \underbrace{\frac{1}{2} \varepsilon^2 \vec{g}^T H \vec{g}}_{\text{correction we must apply to account for the curvature of the function}}$$

Optimizing ε gives

$$\varepsilon^* = -\frac{\vec{g}^T \vec{g}}{\vec{g}^T H \vec{g}} \checkmark \text{only when this is positive}$$

Worst case when \vec{g} = eigenvector of H with the max eigenvalue,
 $\varepsilon^* = \frac{1}{\lambda_{\max}}$

* Scale of the learning rate → determined by eigenvalues of Hessian
 when function can be approximated well by quadratic function (second-order)

Second Derivative test

$$f'(x) = 0 \text{ and}$$

$f''(x) > 0 \rightarrow$ local minimum

$f''(x) < 0 \rightarrow$ local maximum

$f''(x) = 0 \rightarrow$ inconclusive

[Multiple dimensions:
 examine eigenvalues of Hessian]

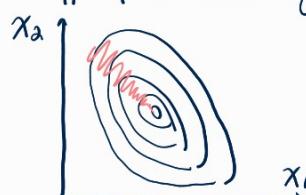
| | |
|---|---|
| } | positive definite (all eigenvalues > 0) |
| | negative definite (all eigenvalues < 0) |
| | Saddle point (+- eigenvalues) inconclusive (same-sign eigenvalues with same zeros) |

Poor-conditioned Hessian

Hessian with a poor condition number

→ one derivative changes more rapidly than another

→ difficult to choose a good step size too large → overshooting at one direction
 too small → converges slowly



direction ↓ direction ↓
 (the ratio of λ_{\max} to λ_{\min})

e.g. condition number 5

→ direction of most curvature is 5-times more
 → waste time descending in the steepest feature

Resolve above by using info from the Hessian to guide the search:

$$\vec{x}^* = \vec{x}^{(0)} - H(\vec{x})(\vec{x}^{(0)})^{-1} \nabla_{\vec{x}} f(\vec{x}^{(0)})$$

f: positive definite quadratic function
directly jumps to the minimum of the function
converge quicker than gradient descent \star but not good for saddle points

First-order optimization (e.g. Gradient descent)

Second-order optimization (e.g. Newton's Method, also uses Hessian)

Obtain some guarantees by restricting to

Lipschitz continuous functions

$$A\vec{x}, A\vec{y}$$

$$|f(\vec{x}) - f(\vec{y})| \leq L \|\vec{x} - \vec{y}\|_2$$

a function with

rate of change bounded

by a Lipschitz constant L

Useful to quantify assumption that \star only a weak constraint

a small change in the input \rightarrow a small change in the output

\star applicable for many DL algo

Convex Optimization

Stronger guarantees by stronger restrictions

Convex functions \rightarrow Hessian is positive semidefinite everywhere

Well-behaved

\rightarrow lack saddle points

\rightarrow local minima = global minima

\star difficult to apply to most DL algo

4.4 Constrained Optimization

Constrained Optimization

Only maximize/minimize $f(\vec{x})$ for values of \vec{x} in some set S

Feasible points: \vec{x} that lie within the set S

Often = find small solution

Norm constraints = e.g. $\|\vec{x}\| \leq 1$

Simple approach

modify gradient descent = make step, project back into S

line search = search only step size ϵ that yields feasible points / project back into S

Sophisticated approach

\downarrow design an unconstrained optimization problem,

convert the solution into that of the original constrained problem

Karush-Kuhn-Tucker (KKT)

introduce the generalized Lagrangian (function)

① Describe S as

$$S = \left\{ \vec{x} \mid \begin{array}{l} \forall i, g^{(i)}(\vec{x}) = 0 \\ \forall j, h^{(j)}(\vec{x}) \leq 0 \end{array} \right\}$$

← Equality constraints
also an "active" constraint
← Inequality constraints

② Introduce KKT multipliers

new variables λ_i and α_j for each constraint (Lagrange multipliers - does not allow inequality constraints)

③ Define the generalized Lagrangian

$$L(\vec{x}, \vec{\lambda}, \vec{\alpha}) = f(\vec{x}) + \sum_i \lambda_i g^{(i)}(\vec{x}) + \sum_j \alpha_j h^{(j)}(\vec{x})$$

④ Unconstrained optimization

$$\min_{\vec{x}} \max_{\vec{\lambda}} \max_{\vec{\alpha} \geq 0} L(\vec{x}, \vec{\lambda}, \vec{\alpha}) \quad \leftrightarrow \quad \min_{\vec{x} \in S} f(\vec{x})$$

when satisfied = equal $f(\vec{x})$
if violate = equal ∞

* Solution of convergence

remains a stationary point whether or not the inactive constraints are included
i.e. the inequality has no influence on the solution

and represented by zeroing out the KKT multipliers
(but inactive constraint can exclude some solutions)

K.K.T conditions

necessary (but not sufficient) for optimal points

→ ∇L is zero

→ All constraints on \vec{x} and KKT multipliers are satisfied

→ The inequality constraints exhibit Complementary Slackness
 $\vec{\alpha} \odot h(\vec{x}) = \vec{0}$

4.5 Example: Linear Least Square

$$\boxed{\min f(\vec{x}) = \frac{1}{2} \|A\vec{x} - b\|_2^2}$$

Gradient-based optimization:

① Obtain gradient:

$$\nabla_{\vec{x}} f(\vec{x}) = A^T A \vec{x} - A^T b$$

② Gradient Descent:

while $\| \nabla_{\vec{x}} f(\vec{x}) \| > \epsilon$ do

$\vec{x} \leftarrow \vec{x} - \epsilon \cdot \| \nabla_{\vec{x}} f(\vec{x}) \|$

end while

or Newton's method:

since the function is quadratic,
the approximation by Newton's method is
exact → one step reaches min.

Suppose now subject to $\vec{x}^T \vec{x} \leq 1$:

① Introduce the Lagrangian

$$L(\vec{x}, \lambda) = f(\vec{x}) + \lambda (\vec{x}^T \vec{x} - 1)$$

② Solve

$$\min_{\vec{x}} \max_{\lambda \geq 0} L(\vec{x}, \lambda)$$

(3a) Solve by Moore-Penrose Pseudoinverse $\vec{x} = A^+ \vec{b}$
→ smallest-norm solution, if the point is feasible; otherwise

(3b) Solve by
differentiating the Lagrange w.r.t. \vec{x} → $\vec{x} = (A^T A + 2\lambda I)^{-1} A^T \vec{b}$
gradient ascent on λ until \vec{x} has the correct norm & derivative on $\lambda = 0$
 $\frac{\partial}{\partial \lambda} L(\vec{x}, \lambda) = \vec{x}^T \vec{x} - 1$

Chapter 5 Machine Learning Basics

5.1. Learning Algorithms

Mitchell (1997) : learn from experience E
w.r.t. some class of tasks T
and performance measure P

5.1.1 The Task T

Classification with missing inputs

e.g. learn a joint distribution for the 2^n classification functions with n (possible missing) inputs

Structured output

e.g. sentence → grammatical tree

Density / PMF estimation

learn the PDF / PMF on the space where the examples are drawn

5.1.2 The Performance Measure P

5.1.3 The Experience E

Supervised vs Unsupervised learning → blurred definition

Re-inforcement learning e.g. learning a joint probability distribution (unsupervised) by factorization (multiple supervised)

interact with the environment with feedback loop
(not only a fixed dataset)

5.1.4 Example: Linear Regression

$\hat{y} = \vec{w}^T \vec{x} + b$ b: bias (the "bias" towards b when input is absent)
affine function: prediction plot is a line not necessarily passing through origin

$MSE_{test} = \frac{1}{m} \sum_i (\hat{y}^{(test)} - y^{(test)})^2$ mean squared error

5.2 Capacity, Overfitting and Underfitting

Data-generating distribution: assumption that the underlying train, test distributions i.i.d. assumptions → are identically distributed and samples are independent

Underfitting: struggle to reduce training error

Overfitting: struggle to reduce test error

Alter Capacity by choosing the hypothesis space

e.g. increase cap by generalize linear regression to include polynomial functions (on top of linear functions)
 $\hat{y} = b + w_1 x \rightarrow \hat{y} = b + w_1 x + \underbrace{w_2 x^2}_{\text{quadratic function of inputs}}$
 \Rightarrow change # input features and add new parameters still linear function of the parameters

Representational Capacity

the family of functions the model can choose from when varying the parameters

Effective Capacity

less than representational capacity often due to imperfection of the algo

Occam's razor

improve generalization of a model by choosing the "simplest" one

Vapnik - Chervonenkis dimension

VC dimension measures the capacity of a binary classifier

* deciding the bounds of difference between training vs test error
rarely applied to DL algo

Nonparametric models

can have arbitrary high capacity

can design by making their complexity a function of the training set size
e.g. Nearest Neighbor Regression

Bayes error

the error incurred by an oracle making predictions from the true distribution
happen when some noise in the distribution

5.2.1 The No Free Lunch Theorem

(Wolpert, 1996) Averaged over all possible data-generating distributions,
no ML algorithm is universally better than any other.

→ Goal of ML research

understand what kind of distributions are relevant to the real world,
and what kind of ML algorithms perform well on them

5.2.2 Regularization

another way to control a model's capacity

by expressing preferences for one function over another
no free-lunch thm → must design for the particular task

through adding regularizer / penalty

e.g. $J(\vec{w}) = \text{MSE}_{\text{train}} + \lambda \vec{w}^T \vec{w}$ preference for smaller squared L^2 norm

5.3 Hyperparameters and Validation Sets

* In practise, when the same test set is repeatedly evaluated for different hyperparameters

different algorithms over many years, (esp for research community)
we end up having over-optimistic evaluations with the test set
⇒ Benchmark became stale, solution = community move on to newer dataset

5.3.1 Cross-Validation

On trial i , the i -th subset of the data → test set

Test error = est. by taking the average test error across k trials

5.4 Estimators, Bias and Variance

5.4.1 Point Estimation

Estimate the single "best" prediction of some quantity of interest

e.g. weights in linear regression a single parameter a vector of parameters in parametric model

θ : true underlying generator, fixed

$\hat{\theta}$: a function of the data

Some data is drawn from a random process,
any function of the data is random → $\hat{\theta}$ a random variable

Function Estimation

assume $\exists f(x)$ that describes the approx. relationship between a variable y given x
a point estimator in the function space

5.4.2 Bias

Bias of an estimator: bias($\hat{\theta}_m$) = $E(\hat{\theta}_m) - \theta$

Unbias: bias($\hat{\theta}_m$) = 0

Asymptotically unbiased:

$$\lim_{m \rightarrow \infty} \text{bias}(\hat{\theta}_m) = 0 \quad \text{or} \quad \lim_{m \rightarrow \infty} E(\hat{\theta}_m) = \theta$$

expectation over the sampled data from a random variable

true θ to define the data-generating distribution

e.g. Bernoulli Distribution

$X^{(i)}$ i.i.d from Bernoulli

$$P(X^{(i)}; \theta) = \theta^{X^{(i)}} (1-\theta)^{1-X^{(i)}}$$

Common unbiased estimator:

$$\hat{\theta}_m = \frac{1}{m} \sum_{i=1}^m X^{(i)}$$

mean of the training samples

Proof of unbias =

$$\text{bias}(\hat{\theta}_m)$$

$$= E\left[\frac{1}{m} \sum_{i=1}^m X^{(i)}\right] - \theta$$

$$= \left(\frac{1}{m} \sum_{i=1}^m E[X^{(i)}]\right) - \theta$$

$$= \left(\frac{1}{m} \sum_{i=1}^m [p(X^{(i)}; \theta)]\right) - \theta$$

e.g. Gaussian Distribution Estimator of the Mean

$X^{(i)}$ i.i.d from Gaussian

$$p(X^{(i)}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(X^{(i)} - \mu)^2}{\sigma^2}\right)$$

Common unbiased estimator

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m X^{(i)}$$

the sample mean

$$\text{Proof of unbias: } \text{bias}(\hat{\mu}_m) = E\left[\frac{1}{m} \sum_{i=1}^m X^{(i)}\right] - \mu$$

$$= \left(\frac{1}{m} \sum_{i=1}^m E[X^{(i)}]\right) - \mu$$

$$= \frac{1}{m} (m \cdot \mu) - \mu = 0$$

e.g. Gaussian ... Estimator of the Variance

$$\begin{aligned} & \left[\sum_{i=1}^m p(x^{(i)} = 1) + p(x^{(i)} = 1) \cdot 1 \right] - \theta \\ & = \left(\frac{1}{m} \sum_{i=1}^m [0 \cdot (1-\theta)^{1-i}] \right) - \theta \\ & = \frac{1}{m} (m \cdot \theta) - \theta = 0 \end{aligned}$$

Bias Unbias \star

$$\hat{\sigma}_m^2 = \frac{1}{m} \sum_{i=1}^m (\bar{x}^{(i)} - \hat{\mu}_m)^2 \quad | \quad \hat{\sigma}_m^2 = \frac{1}{m-1} \sum_{i=1}^m (\bar{x}^{(i)} - \hat{\mu}_m)^2$$

5.4.3 Variance and Standard Error

Measure of how we expect the estimator we compute from data change as we independently resample the dataset from the underlying data-generating process

Variance of an estimator: $\text{Var}(\hat{\theta})$

Standard Error of an estimator: $SE(\hat{\theta}) = \sqrt{\text{Var}(\hat{\theta})}$

Standard Error of the mean: $SE(\hat{\mu}_m)$

given by using

an estimate of σ

$$= \sqrt{\text{Var}\left(\frac{1}{m} \sum_{i=1}^m \bar{x}^{(i)}\right)} = \frac{\sigma}{\sqrt{m}} \hat{\sigma}$$

σ^2 : true variance
of the samples $x^{(i)}$

but both the two cannot provide an unbiased estimator in this case

→ very useful in ML but the RHS (unbiased one) gives reasonable result when m is large

→ estimate test error by

computing the sample mean of [the error on the test set]

→ by Central Limit theorem,

the test error \sim distributed with a normal distribution
with a mean and a variance

→ Compute $SE(\hat{\mu}_m)$ → get confidence interval

$$\text{e.g. } 95\% \text{ is } \hat{\mu}_m \pm 1.96 SE(\hat{\mu}_m)$$

→ Algo A is better than Algo B with 95% confidence

$$\text{if } \hat{\mu}_{m_B} + 1.96 SE(\hat{\mu}_{m_B}) < \hat{\mu}_{m_A} - 1.96 SE(\hat{\mu}_{m_A})$$

B upper bound of sample mean of test error lower bound of sample mean of test error

A

Common: variance of a popular estimator $\downarrow \downarrow$ as sample size $m \uparrow \uparrow$

5.4.4 Trading off Bias and Variance to Minimize Mean Squared Error

Two sources of error in an estimator:

Bias: expected deviation from the true function/parameter

Variance: deviation from the expected estimator value due to sampling

Negotiate the trade-off:

① Cross validation (empirically)

② Mean squared error (MSE) of the estimates

$$\text{underfit} \quad \text{overfit} \quad \text{test} \quad \text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2]$$

keep both small



5.4.5 Consistency

(Weak consistency) $\underset{m \rightarrow \infty}{\text{plim}} \hat{\theta}_m = \theta$ i.e. $P(|\hat{\theta}_m - \theta| > \varepsilon) \rightarrow 0$ as $m \rightarrow \infty$

(Strong consistency) Almost sure convergence of $\hat{\theta}$ to θ sequence of random variables $X^{(m)}$
 $i.e. P(\lim_{m \rightarrow \infty} X^{(m)} = x) = 1$

Consistency: ensures bias diminishes when # sample grows
 however, asymptotical unbias $\not\rightarrow$ consistency
 $i.e. \hat{\theta}_m \not\rightarrow \theta$ as $m \rightarrow \infty$

5.5 Maximum Likelihood Estimation

Maximum Likelihood Estimation Principle

$$\begin{aligned}\hat{\theta}_{\text{MLE}} &= \underset{\theta}{\operatorname{argmax}} p_{\text{model}}(X_i; \theta) & &= \underset{\theta}{\operatorname{argmax}} \frac{1}{m} \sum_{i=1}^m \log p_{\text{model}}(\vec{x}^{(i)}; \theta) && (\text{divided by } m) \\ &= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^m p_{\text{model}}(\vec{x}^{(i)}; \theta) & &= \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{\vec{x} \sim p_{\text{data}}} \log p_{\text{model}}(\vec{x}^{(i)}; \theta) && (\text{view as expectation}) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^m \log p_{\text{model}}(\vec{x}^{(i)}; \theta) & &= \underset{\theta}{\operatorname{argmin}} - \mathbb{E}_{\vec{x} \sim p_{\text{data}}} \log p_{\text{model}}(\vec{x}^{(i)}; \theta)\end{aligned}$$

Maximize log likelihood \Leftrightarrow Minimize Cross Entropy (or KL Divergence $D_{\text{KL}}(\hat{p}_{\text{data}} \| p_{\text{model}})$)

where $p_{\text{model}}(\vec{x}; \theta)$: a parametric family of probability distributions over the same space indexed by θ

\rightarrow more "likely" to \vec{x} \rightarrow maps any configuration \vec{x} to a real number close to $p_{\text{data}}(\vec{x}; \theta)$ \rightarrow estimating the true probability $p_{\text{data}}(\vec{x})$

Any loss with a negative log-likelihood is a cross-entropy \downarrow minimize the dissimilarity (NLL)
 not only for Bernoulli or softmax distribution!

between the empirical distribution and the probability distribution (by train set) (by model)

in software, view as "minimizing KL divergence" is helpful

\rightarrow KL divergence's min is 0 (but NLL can be negative when x is real-valued)

5.5.1 Conditional Log-Likelihood & Mean Squared Error

Conditional Maximum likelihood estimator: $\hat{\theta}_{\text{ML}} = \underset{\theta}{\operatorname{argmax}} P(Y|X; \theta)$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^m \log P(\vec{y}^{(i)} | \vec{x}^{(i)}; \theta) \quad (\text{Assume i.i.d.})$$

Example: Linear Regression as Maximum Likelihood

mapping from \vec{x} to \vec{y} \rightarrow chosen to be minimizing mean squared error arbitrary?

imagine:

an infinitely large training set \rightarrow examples might have same \vec{x} value

but produce different y

Goal

predict the conditional distribution $p(y|\vec{x})$ instead of a single \hat{y}

define:

Assume Gaussian $p(y|\vec{x}) = \mathcal{N}(y; \hat{y}(\vec{x}; \vec{w}), \sigma^2)$ assume variance fixed by the user
 prediction of the mean μ

⇒ then this choice of $p(y|\vec{x})$: of the Gaussian

$$p(y|\vec{x}) = \sum_{i=1}^m \log p(y^{(i)}|\vec{x}^{(i)}; \theta) \quad \begin{array}{l} \text{Assumed i.i.d;} \\ \text{Conditional log likelihood} \end{array}$$

$$\text{log-likelihood} = -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^m \frac{\|\hat{y}^{(i)} - y^{(i)}\|^2}{2\sigma^2}$$

So comparing with the mean squared error

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^m \|\hat{y}^{(i)} - y^{(i)}\|^2 \quad \begin{array}{l} \text{Maximizing log likelihood w.r.t. } \vec{w} \\ \text{Same estimates of } \vec{w} \text{ as minimizing MSE} \end{array}$$

5.5.2 Properties of Maximum Likelihood

→ Main appeal: maximum likelihood estimator
 → best estimator asymptotically as $m \rightarrow \infty$ in terms of its rate of convergence

Condition of Consistency (parameter → true parameter as $m \rightarrow \infty$)

① True P_{data} must lie within model family $P_{\text{model}}(\cdot; \theta)$

② True P_{data} must correspond to exactly one value of θ

(otherwise maximum likelihood cannot decide which θ)

Consistent estimators (other than maximum likelihood)

diff by their statistical efficiency i.e. lower test error for fixed m

studied by Parametric mean squared error (how close we are to the true parameter)

Cramér-Rao lower bound (1945, 1946)

No consistent estimator has a lower MSE than the maximum likelihood estimator

✓ consistent ✓ efficient

5.6 Bayesian Statistics

Frequentist statistics

the observed dataset is random
 find $\hat{\theta}$ (fancier)

estimating one θ , make all predictions based on that θ

true θ is fixed but unknown, the point estimate $\hat{\theta}$ is a random variable

account of it being a function of the dataset (thus random)

Bayesian statistics

the observed dataset is NOT random
 find true θ

consider all possible values of θ when making a prediction

probability reflect degrees of certainty / belief in states of knowledge

the dataset is directly observed & not random, the true θ is uncertain thus random

Prior probability distribution

"the prior" $p(\theta)$ = represent knowledge of θ
 usually choose to be of *high entropy* before observing any data

Recover the effect of data on our belief on θ

$$\text{Baye's rule: } p(\theta | x^{(1)}, \dots, x^{(m)}) = \frac{p(x^{(1)}, \dots, x^{(m)} | \theta) p(\theta)}{p(x^{(1)}, \dots, x^{(m)})}$$

→ causes the *posterior* to *lose entropy* and concentrates around a few highly likely values of parameters

Bayesian vs Maximum Likelihood Estimation

↳ make predictions using a full distribution of θ (instead of a point estimate of θ)

$$\text{e.g. } p(x^{(mtl)} | x^{(1)}, \dots, x^{(m)}) = \int p(x^{(mtl)} | \theta) p(\theta | x^{(1)}, \dots, x^{(m)}) d\theta$$

← each value of θ
 with pos. probability density
 contributes to the prediction

↳ Integration over $\theta \rightarrow$ address the uncertainty

of a given point estimate of θ

(Frequentist approach: evaluate the estimator's variance)

↳ the prior has influence:

shift PMD towards regions of the parameter spaces that are preferred a priori.

in practice, the prior → a preference for models that are simpler or smooth

"a source of subjective human judgement affecting the predictions"

✓ Generalize better ✗ High computational cost

Example: Bayesian Linear Regression

$$\hat{y}^{(train)} = X^{(train)} \vec{w}$$

$$p(\vec{y} | X, \vec{w}) = \mathcal{N}(\vec{y}; X\vec{w}, I) \text{ assume a Gaussian conditional distribution on } y^{(train)}$$

$$\propto \exp(-\frac{1}{2}(\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w}))$$

① Specify a prior (reflect naive belief)

Real-valued parameters → common to use Gaussian

$$p(\vec{w}) = \mathcal{N}(\vec{w}; \mu_0, \Lambda_0)$$

$$\propto \exp(-\frac{1}{2}(\vec{w} - \mu_0)^T \Lambda_0^{-1} (\vec{w} - \mu_0))$$

μ_0 : mean vector of the prior distribution

Λ_0 : covariance matrix = $\text{diag}(\lambda_0)$

② Decide the *posterior* distribution over \vec{w}

$$p(\vec{w} | X, \vec{y}) \propto p(\vec{y} | X, \vec{w}) \cdot p(\vec{w})$$

$$\propto \exp(-\frac{1}{2}(\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w})) \cdot \exp(-\frac{1}{2}(\vec{w} - \mu_0)^T \Lambda_0^{-1} (\vec{w} - \mu_0))$$

rewritten as

$$\text{another Gaussian} \propto \exp(-\frac{1}{2}(\vec{w} - \mu_m)^T \Lambda_m^{-1} (\vec{w} - \mu_m))$$

$$\Lambda_m = (X^T X + \Lambda_{\mu_0}^{-1})^{-1}$$

Insight: if set $\mu_0 = 0$, $\Lambda_0 = \frac{1}{\alpha} I$

then $\mu_m = \text{Same estimates of } \vec{w} \text{ as does frequentist estimation}$

$$\mathcal{L}_m = \mathcal{L}_m(\vec{x}^T \vec{y} + \lambda_0^{-1} \mu_0)$$

with a weight decay penalty of $\alpha w^T w$

\mathcal{L}_m : covariance matrix showing how likely each value of w are (not only μ_m)

5.6.1 Maximum a Posteriori (MAP) estimation

Motivation: still desirable to give a single point estimate
 → allow the prior to influence the point estimate offers tractable approximation

Maximum a Posteriori (MAP) point estimate

chooses the point of maximal posterior probability (continuous θ = maximal probability density)

$$\theta_{\text{MAP}} = \arg \max_{\theta} p(\theta | \vec{x}) = \arg \max_{\theta} p(\vec{x} | \theta) \cdot p(\theta)$$

$$= \arg \max_{\theta} \underbrace{\log p(\vec{x} | \theta)}_{\text{standard log likelihood}} + \underbrace{\log p(\theta)}_{\text{prior}}$$

Assume Gaussian prior
 $N(\vec{w}; 0, \frac{1}{\lambda} I^d)$
 then prior $\propto \lambda w^T w$ \rightarrow weight decay penalty

MAP bayesian inference with a Gaussian prior on the weight correspond to

✓ Leverage info from prior (not found in training data)

✓ Reduce Variance in this point estimate (compared to ML estimate)

✗ At the cost of increasing Variance

Many regularized estimation strategies with an extra term correspond to $\log p(\theta)$

→ can be interpreted as making the MAP estimation to Bayesian Inference

5.7 Supervised Learning Algorithms

5.7.1 Probabilistic Supervised Learning

Finding $p(y | \vec{x}) \rightarrow$ use maximum likelihood estimation

to find the best θ for a parametric family of distributions $p(y | \vec{x}; \theta)$

Linear Regression: $p(y | \vec{x}; \theta) = N(y; \theta^T \vec{x}, I)$

What if a classification problem?

① Specify only probability of one of these classes

② if binary variable → constraints its mean to between 0 and 1

Instead of arbitrary means

Logistic Regression: $p(y=1 | \vec{x}; \theta) = \delta(\theta^T \vec{x})$ the logistic sigmoid function

no closed form of solution

minimize negative likelihood by gradient descent

$$\frac{1}{1 + \exp(-x)} = \frac{\exp(x)}{1 + \exp(x)}$$

5.7.2 Support Vector Machines

Output a class identity: $\vec{w}^T \vec{x} + b = \begin{cases} > 0 & \rightarrow \text{positive} \\ < 0 & \rightarrow \text{negative} \end{cases}$

Kernel trick (Algo: Kernel machines / Kernel methods)

Motivation: observe that many ML algs can be written exclusively in terms of dot products between examples

Rewrite:

$$\vec{w}^T \vec{x} + b = b + \sum_{i=1}^m d_i \vec{x}^T \vec{x}^{(i)}$$

Replace:

$$\vec{x} \rightarrow \phi(\vec{x}) \text{ feature function}$$

$$\vec{x}^T \vec{x} \rightarrow k(\vec{x}, \vec{x}^{(i)}) = \phi(\vec{x}) \cdot \phi(\vec{x}^{(i)}) \text{ the kernel}$$

Outcome:

$$f(\vec{x}) = b + \sum_{i=1}^m a_i k(\vec{x}, \vec{x}^{(i)})$$

inner product, finite dimensional space \rightarrow product
infinite dimensional space \rightarrow integration

Non-linear for $f(\vec{x})$ w.r.t. \vec{x}
linear between $f(\vec{x})$ and $\phi(\vec{x})$

equivalent to preprocessing the data
by $\phi(\vec{x})$ then learn a linear model

linear in a diff. space

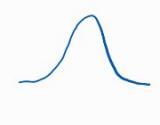
- ✓ learn models that are non-linear as a function of \vec{x}
using convex optimization techniques that are guaranteed to converge efficiently
- ✓ the kernel function k is sig. more computationally efficient
e.g. $\phi(\vec{x})$ is intractable, but $k(\vec{x}, \vec{x}')$ is a non-linear, tractable function of \vec{x}
(or infinite dimensional)
- ✗ cost of evaluation is linear in # training samples by a_i for each i th sample
SVM \rightarrow mitigate by learning an almost-zero \vec{x}
only a few non-zero a_i to evaluate (support vectors)
- ✗ high computational cost of training when the dataset is large
- ✗ kernel machines with generic kernel struggle to generalize well
- !!! modern DL begins (2006) when NN outperform RBF kernel SVM on MNIST

Gaussian kernel / RBF kernel (Radius basis function)

$$k(\vec{u}, \vec{v}) = \mathcal{N}(\vec{u} - \vec{v}; 0, \sigma^2 I)$$

↑
standard normal density

Value decreases along lines in \vec{v} space
radiating outward from \vec{u}



A dot product in an infinite-dimensional space

\rightarrow performing a template matching

when a new test point \vec{x}' is near a training sample \vec{x} (Euclidean distance),

RBF kernel has a large response $\rightarrow \vec{x}'$ similar to \vec{x}

the model then put a larger weight to the training label y associated with \vec{x}

prediction: combine many training labels weighted by the similarity

5.7.3 Other Simple Supervised Learning Algorithms

k -nearest neighbours

non-probabilistic, can be used for regression/classification

not restricted to a fixed number of parameters

\hookrightarrow actually, just performing an "average" of the y values for the

k -nearest neighbours of test \vec{x}' from the training dataset X

\hookrightarrow classification

- ✓ achieve high capacity
- 1-nearest neighbour \rightarrow 2. Bayes Error (error by the oracle on true distribution)
- If all neighbours, as $m \rightarrow \infty$ the \hat{x} comes from 2 neighbours
converge to Bayes error having a tie in distance and have to pick random
- high accuracy given a large dataset
- X at cost of high computational cost
- X generalize badly at small finite training set
- X cannot learn that one feature (of \hat{x}) is more discriminative than another
e.g. if only x_1 is relevant \rightarrow cannot detect

Decision Trees

Result is a piecewise-constant function
 Internal node \rightarrow break into subregion in the input space
 axis-aligned splits \rightarrow non-overlapping \rightarrow 1:1 correspondent of a leaf node
 usually regularized with size constraint \rightarrow parametrized model \leftrightarrow output

X struggle to solve easy problem w.r.t. logistic regression
 e.g. 2-class problem of \hat{x} $\begin{cases} > 0 & \text{when } x_2 > x_1 \\ < 0 & \text{otherwise} \end{cases}$ decision boundary is not axis aligned \rightarrow require many nodes

5.8 Unsupervised Learning Algorithms

Informally, extract info from a distribution that do not require human labor to annotate examples

Classic Task = find the "best" representation of data

while obeying some penalty/constraint aiming to keep the representation "simpler"

e.g. lower-dimensional representations \rightarrow compress \hat{x}
 sparse representations \rightarrow increases the dimensionality of \hat{x} to become mostly 0
 independent representations \rightarrow disentangle the source of variations underlying

Representations \rightarrow central themes of DL St. the dimensions are statistically independent

5.8.1 Principal Component Analysis

- A means of compressing data \leftrightarrow unsupervised learning to learn a representation of data
- ✓ lower-dimensionality than the original
 - ✓ no linear correlation between each elements
 - * full independence \rightarrow must also remove non-linear relationship between variables
 - ✓ PCA de-correlates (has a diagonal covariance matrix $\Sigma^2 \rightarrow$ uncorrelated)
when projecting \hat{x} to \hat{f} via linear transformation W
disentangle the unknown factors of variation underlying the data
 - X requires other algo to learn representations

that distinguish more complicated feature dependences (other than simple linear transformation) (that provided by)

5.8.2 k-means Clustering

divides the training set into k clusters of examples that are close to each other

⇒ providing a k -dimensional one-hot code vector \vec{h} representing an input \vec{x}

① Initialize k centroids $\mu^{(1)}, \dots, \mu^{(k)}$ $h_i = 1$ if $\vec{x} \in \text{cluster } i$

② each training example assigned to cluster i (the nearest centroid)

③ each centroid $\mu^{(i)}$ is updated to the mean of all training examples assigned to cluster i

✓ sparse representation

✓ statistical advantages (same cluster → similar)

✓ computational advantage (one-hot code vector → captured by an integer)

✗ inherently ill-posed

can measure Euclidean distance but no real world meanings

e.g. images of Red cars, Red trucks, Gray cars, Gray trucks

⇒ prefer distributed representation

than one-hot representation (e.g. (color, car type))

✗ not clear what the optimal representation is

✓ reduces burden on the algo to guess the single attribute we care

5.9

Stochastic Gradient Descent (SGD)

Large training sets → good generalization ✓

more computational expensive ✗

$$J(\theta) = \mathbb{E}_{\vec{x}, y \sim \hat{P}_{\text{data}}} [L(\vec{x}, y, \theta)] = \frac{1}{m} \sum_{i=1}^m L(\vec{x}^{(i)}, y^{(i)}, \theta)$$

Negative
 Conditional
 likelihood

$$\nabla J(\theta) = -\log p(y|\vec{x}; \theta) \quad \frac{1}{m} \nabla_{\theta} L(\vec{x}^{(i)}, y^{(i)}; \theta)$$

- log $p(y|\vec{x}; \theta)$

$O(m)$ computational cost

SGD → sample a minibatch of size m' (usually fixed even if m grows)

✓ Main way to train large linear models on large dataset

✓ SGD convergence rate does not improve when $m \rightarrow \infty$ (already converge to best test error before SGD sample all examples)

✓ DL = scalable way of training non-linear models on large dataset

Past: Kernel machines for nonlinear models, but constructing kernel take $O(m^2)$ → inefficient

5.10

Building a Machine Learning Algorithm

Recipe = a specification of dataset

typically includes + a cost function

A term to perform statistical est. + an optimization procedure

X and y Linear Regression

$$J(\vec{w}, b) = -\mathbb{E}_{\vec{x}, y \sim \hat{P}_{\text{data}}} \log p_{\text{model}}(y|\vec{x})$$

Solving for when Cost is 0

- ① e.g. negative log-likelihood + a model
 - ② regularization terms ↗ if model is non-linear,
e.g. weight decay $\|\vec{w}\|_2^2$ no closed-form solution → iterative numerical procedure
- $P_{\text{model}}(y|\vec{x}) = \mathcal{N}(y; \vec{x}^T \vec{w} + b, I)$
- Unsupervised: e.g. PCA loss function $J(\vec{w}) = \mathbb{E}_{x \sim p_{\text{data}}} \|\vec{x} - r(\vec{x}; \vec{w})\|_2^2$
- Special-case optimizers: e.g. decision trees
 k -means
- flat region, inappropriate for Gradient-based optimizers for minimization
- reconstruction function $(\vec{w}^T \vec{x}) \vec{w}$

5.11 Challenges Motivating Deep Learning

Generalizing to new examples → exponentially more difficult when working with high-dimensional data

Traditional ML algorithm → insufficient to learn complicated functions in high-dimensional spaces

5.11.1 The Curse of Dimensionality

possible distinct configurations increases exponentially as # variable increases
→ much larger than # training examples

5.11.2 Local Constancy and Smoothness Regularization

To generalize well, guide ML algo by prior beliefs

Explicit Prior Beliefs: e.g. probability distribution (over parameters) of models about what kind of function they should learn ← (directly influencing the function)
only indirectly for parameters

Implicit Prior Beliefs: e.g. choosing algo that bias towards some class of functions ↓ most widely used

* Smoothness prior / Local constancy prior *

the function should not change very much within a small region

↳ many methods are designed to encourage

$$f^*(\vec{x}) = f^*(\vec{x} + \vec{\epsilon})$$

e.g. k-nearest neighbour

e.g. local kernels (similarity function that performs template matching, $k(\vec{u}, \vec{v}) \neq 0$ when $\vec{u} = \vec{v}$)

e.g. decision-trees

✗ suffer from the limitations of exclusively smoothness-based learning

↳ $O(k)$ regions in input space
 $O(k)$ examples for training → multiple of n to achieve some statistical confidence of the prediction

← require too many for complex target f. to ...

$O(k)$ parameters with $O(1)$ parameter for each of the $O(k)$ regions.

- ? Is it possible to represent a complicated function efficiently?
- ? Is it possible for the estimated function to generalize well to new inputs?

YES, a very large number of regions e.g. $O(2^k)$
can be defined with $O(k)$ examples
as long as we introduce some dependencies between the regions through
additional assumptions about the underlying data-generating distributions
→ can generalize non-locally

Deep learning → usually not include strong, task-specific assumptions
(such as periodicity), so as to improve generalization

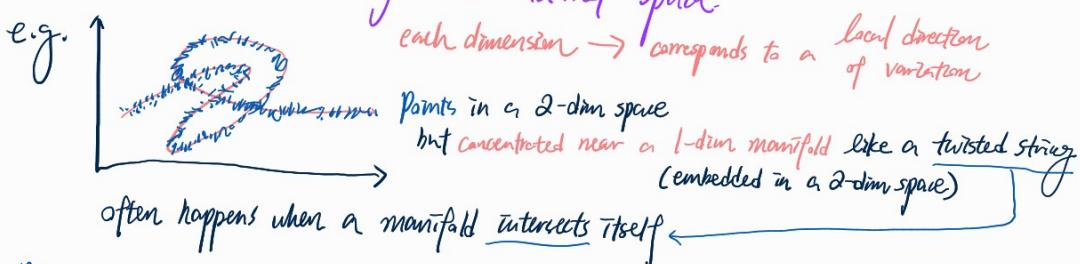
Care idea: assume the data was generated by allows exponential gains
the composition of factors potentially in a hierarchy

5.11.3 Manifold Learning

Manifold: a connected region (a set of points associated with a neighborhood around each point)
from any given point, the manifold locally appears to be a Euclidean space
e.g. ground floor as 2D-plane, but in fact a Spherical manifold in 3D space

In DL context,

a connected set of points that can be approximated well
by considering only a small number of degrees of freedom/dimensions
embedded in a higher-dimensional space.



Manifold Learning Algorithms

assume that most of \mathbb{R}^n consists of invalid inputs,
the interesting inputs occur only along a collection of manifolds
containing a small subset of points

the interesting variations in the output of the learned function
occur only along directions that lie on the manifold

Manifold assumptions / hypothesis: probability mass is highly concentrated
or when moving from one manifold to another

Argue that it is at least approximately correct for AI tasks e.g. processing text sound images

① real world is true e.g. uniform noise X resembles structured inputs

② can imagine that each examples are connected by highly similar examples
that can be reached by small steps

... can be reduced by applying Transformations
to traverse the manifold

Extract Manifold coordinates

e.g. roads \rightarrow 1-dim manifold embedded in 3-dim space
we give directions to address in terms of 1-dim roads
of coordinates in the 3-dim world



e.g. manifold structure of a dataset consisting of faces QMUL Multiview Face Dataset
photos cover 2-dim manifold corresponding to
 Δ angles of face rotations \Rightarrow hope the algo is
able to discover + disentangle such manifold coordinates