Machine Learning

Basics, Models and Trends

Gang Chen

Copyright by

Gang Chen

2015

Abstract

This book is a beginner's guide to study machine learning, with focus on basic methods and algorithms. It aimed at senior undergraduates, graduate students and researchers in areas, such as computer science, bioinformatics, statistics and psychology. It is helpful for readers to be familiar with elementary calculus, linear algebra and probability before understanding the concepts and contents in this book. To make it easy to understand, we also provider basic mathematic reference in Appendix A and B.

Our focus is on machine learning basics, models and also the recent trends. More specifically, we provide the most widely used mathematical models, derivation and optimization techniques. We intentionally avoid the experiments and evaluations because machine learning models are sensitive to various settings and datasets. Instead, this book is more like a tutorial of machine learning methods and algorithms, with the hope that readers can understand the basics and learn how to derive equations and optimize a given objective functions. This means this book will mainly present methods and approaches to different machine learning problems.

In addition, this book covers most machine learning topics, such as surprised learning, unsupervised learning and semi-supervised learning. Considering metrics playing a vital role to learn models, we start with similarity measures and build all topics based on these fundamentals. Most chapters will introduce a distinct family of machine learning models given different training inputs, with focus on understanding the models throughly. While we cannot reflect the most advances in machine learning, the mathematic methods and logics will lay solid foundations for readers to learn and handle more complex situations in different applications.

We hope you enjoy and like the book.

Table of Contents

List of Tables

List of Figures

List of Symbols

Symbol Usage

δ	indicator function
ϵ	eror parameters
ξ	SVM slack non negative variable
b	bias in classifier, such as SVM and Perceptron
λ	learning rate or constant (to balance items in SVM)
$\mathcal{H}$	set of hypothesis
Φ	feature mapping
π	probability of Gaussian mixture models
μ	mean of a Gaussian distribution
σ, Σ	covariance matrix of a Gaussian distribution
D	feature dimension, degree matrix
d	feature dimension, distance
$\mathcal{D}$	observed training data
f	predictor, hypothesis (classifier or regressor)
$\mathbf{h}$	hidden variables or data
$\mathcal{L}$	objective function
ℓ	loss function
T	matrix transpose
p	probability distribution

q	auxiliary distribution
l	number of layers in neural network
W	weights in neural network or graph edge weights
$\mathbf{w}$	SVM or Perceptron weight/parameter
K, k	number of clusters
N, n	number of instances
L, l	number of layers
$\mathcal{X}$	instance domain
$\mathbf{v}, \mathbf{x}$	instance
$\mathcal{Y}$	target/label domain
y, z	label

Chapter 1

Introduction

1.1 The definition of machine learning

Machine learning is a research branch in computer science, which uses computer for pattern recognition. One of its definition which is widely quoted is from Tom M. Mitchell [1], with formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improved with experience E". The purpose of machine learning is to build a model from example inputs to make predictions or decisions on unseen data. More generally, we want computer do what we can do, and think what we can think.

From the model view, machine learning is derived from mathematics and statistics. Machine learning is a subfield in artificial intelligence, which need to study and design algorithms that can do learning and prediction. Such algorithms indeed requires mathematical models to learn from training data, and also require computational theory to analyze its time and space complexity. For example, we always use maximum likelihood for parameter estimation in statistics, and such technology also has been widely used to design objective functions and estimate parameters in machine learning. Another example is the probably approximately correct learning (PAC learning), which is a framework initially proposed to analyze the generalization error of support vector machines (SVMs) given a number of training samples. Machine learning also has strong ties to optimization, which includes methods and theory to handle convex and nonconvex objective functions. For instance, many machine learn-

machine learning methods		
Categories	Training dataset	
	x	y
supervised learning	*Yes*	*Yes*
semi-supervised learning	*Yes*	partial
unsupervised learning	*Yes*	*No*

Table 1.1. The figure shows the differences between supervised learning and unsupervised learning according to the availability of training datasets.

ing problems can be formulated as minimization of some loss function, and gradient based methods (descent/ascend) can be applied to optimize convex function to get the global minimum or maximum.

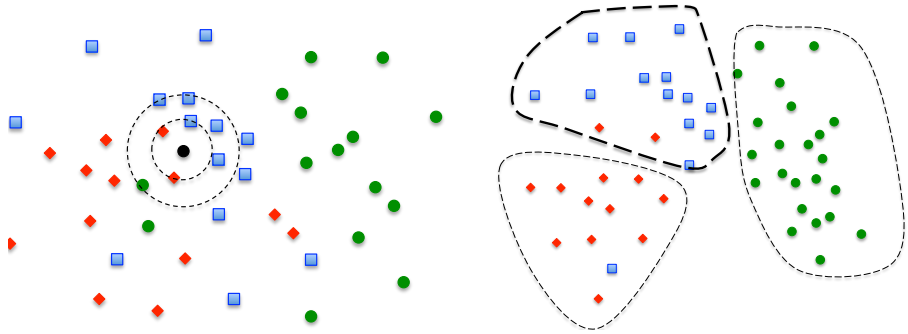
From the application view, how to apply computer science to handle real problems (i.e. relieve us from repeatable and tedious jobs) are the strong driving force for new machine learning models or algorithms. According to different training data, machine learning has evolved from discrete prediction to continuous and structural output prediction. For example, we use k-means and Gaussian mixture models to discovery the latent data structure without label information. If the training data has label information, perceptron training and SVMs are widely used for classification tasks. As for structured output (e.g. sequential labeling), hidden Markov model (HMM) can incorporate label correlation and is applied in speech recognition [2]. Later, conditional random fields (CRFs) [3] attracted significant attention in machine learning community, when it was first proposed to handle national language processing topics. Also more and more complex models have been proposed, from linear models (linear support vector machine) to nonlinear models (kernel methods and deep learning). In addition, the cross domain knowledge and disciplines requires machine learning methods to solve their corresponding problems, such as bioinformatics (protein structural prediction), marketing and economics.

1.2 Types of machine learning tasks

In general, machine learning tasks have two basic components: data and model (or algorithm). A training data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$ is from observations, which is a collection of instances or samples and act as the input to an learning algorithm. An training instance $\mathbf{x}_i$ is sampled independently from any under distribution, which is unknown to us. To assess the strength and utility of the learnt algorithm or model, we need a test set, which is unseen while training the model. And the purpose of machine learning is to design and learn a model which can fit the training data and generalize well on unseen data. According to different data types, we need to apply corresponding models. For example, for sequential data, we can use or design sequential models, which can capture more intrinsic information from the data.

Machine learning models can be divided into different paradigms, according to different criterions. Based on the target given or not, machine learning tasks can be classified into three categories in Table 1.1:

- **Supervised learning**: the computer or system has given input and output training pairs $(\mathbf{x}_i, y_i)_{i=1}^N$, with the purpose to learn a model which can make predication on new unseen data. More specifically, for each input instance $\mathbf{x}_i$, the system has desired label y_i, so that it can learn a mapping (linear or nonlinear) from the input to the output. Depending on the target domain (discrete or continuous), it can be further divided into classification and regression problems. For example, in binary classification task, the training data are divided into 2 classes, and more specifically, each instance has a label (e.g. YES or NO). Thus, in supervised case, the computer is given both its input and its target (YES or NO). Before we go ahead, we first look at the most simple case using k-NN as the classifier in Fig. 1.1(b). This example shows a training datasets belonging to 3 classes, which marked with 3 different colors (red, green and blue). The task here is that given the training data points, we need to predict the label of the dark point. If we use the small radius, there are three points inside this circle, with 2 red points and 1 blue points. According to the voting principle in kNN, we can assign the dark point to the red class. Considering we use kNN as the classifier here, we can vary the radius. If we use the large radius, then we have 8 points covered by the circle, with 2 red points, 4 blue points and 1 green point.



(a) (b)

Figure 1.1. (a) shows a supervised case for classification. The training dataset has 3 classes, which are marked with 3 different colors (red, green and blue). The dark colored point has unknown assignment, and we want to label it. It shows that when the radius of kNN changes, the classification results varies too. (b) shows a unsupervised case for clustering.

And we will classify the dark point as the blue class.

- **Unsupervised learning**: the system only has a bunch of data $\{\mathbf{x}_i\}_{i=1}^N$, without targets or labels, and the goal is to learn or discover hidden patterns from the data. Compared to supervised learning, no labels or targets are given to the unsupervised approaches. In clustering, we have a set of data points, and we want to use computer algorithms to divide them into several groups. The example in Fig. 1.1(b) shows a case using k-means clustering to group the data points. Unlike supervised learning, we do not know which group each data point belongs to in unsupervised learning. More specifically, each data point does not have the corresponding label to indicate the cluster it should be assigned. Common unsupervised learning tasks include clustering, outlier detection and dimension reduction.

- **Semi-supervised learning**: it is an learning strategy between supervised learning and unsupervised learning. Basically, the system only has incomplete training data (partial labeled data or side information) available, and it needs to learn a model for prediction (e.g. clustering or classification). Supervised algorithms require the labeling of the training instances, which is labor intensive and may also need expertise in certain fields. Unsupervised learning, however cannot yield desired performance. Thus, semi-supervised learning approaches make use of both (typical a small amount of) labeled and unlabeled data. Recently,

semi-supervised learning has attracted great attention and has been extensively used in semi-supervised classification (image and text categorization) and constrained clustering.

Another interesting research topic is reinforcement learning, in which computer is required to finish certain goal in a dynamic environment, without explicitly guidance from human. In this book, we will focus on the three topics mentioned above.

Based on mathematical models, machine learning algorithms can be categorized into generative learning and discriminative learning [4]. Given the training data $(\mathbf{x}, y)$, the generative algorithm learns the model by maximizing the joint distribution $p(\mathbf{x}, y)$, while the discriminative model maximizes the conditional likelihood $p(y|\mathbf{x})$. More specifically, the generative models learn parameters, which can generate the data from the model. The discriminative model is not interested how the data generated, and it only cares how to predicate the targets correctly.

Note that discriminative models or generative models are not only constrained in supervised learning cases, they can also be used in unsupervised learning. For example, Gaussian mixture model (GMM) is an generative model for unsupervised clustering.

1.3 Overview of the book

One of the key factors to improve machine learning performance is how to define or learn a good metric to evaluate the similarity between instances. In unsupervised learning, we hope the similar points should be grouped into the same cluster and points which are far away with each other should be assigned to different clusters. In supervised learning, we hope two points which have the same label should be mapping together, e.g. k-nearest neighbor classifier. Also, if we use SVMs for binary classification, we hope the inner product between the model and the data points from the positive class is as large as possible. Hence, we start the book with distance metric, which talks about different similarity measures. Then, we discuss supervised learning approaches, which learn a mapping from input $\mathbf{x}$ to its label y. Then we relax conditions and generalize supervised methods to handle partial labels, such as pairwise constraints. More specifically, we extend these approaches to handle semi-

supervised problems. And then considering it is time-consuming to acquire large scale labeled data, we focus on unsupervised learning approaches, which has no target information available. Lastly, we will introduce the recent trends on machine learning, especially deep learning. This book discuss the following topics:

(1) supervised learning: we will start with the most easy and intuitive perceptron training, and then extend it to linear SVMs and kernel SVMs. To handle large scale data, we will introduce online learning, which use stochastic gradient descent to update model. Perceptron training as a fundamental approach for supervised learning has significant influence on machine learning. For example, the widely used SVMs and kernel SVM from 1990-2010 is derived from Perceptron learning. Further the maximum margin techniques play a vital role in classification, such as passive aggressive and online SVM. Thus, we will discuss maximum margin techniques on classification problem. Beyond that, we will introduce regression problems, especially linear regression and logistic regression (it is called logistic regression, but it is for classification problem).

(2) unsupervised learning: we first discuss the widely used k-means, which is a hard label assignment approach for clustering. Basically, for each data instance, it will compute the distance to each clustering center, and assign it to the closest center; and further update clustering means with new assignments. Then we will discuss GMM, which uses soft label assignment. Especially, we will discuss Expectation-Maximization (EM) to learn model parameters. We will also discuss graph-based clustering approach, namely spectral clustering. Spectral clustering makes use of eigenvalue decomposition for dimension reduction, and then do clustering analysis in the low dimension space. Finally, we will introduce nonparametric clustering, where the model complexity grows with the training data. Compared to k-means and GMM, it can be thought as an infinite mixture model, which does not require the number of clusters as input.

(3) semi-supervised learning: it uses partial label or weakly labeled data for classification or clustering. Semi-supervised learning is a mix of supervised learning and unsupervised learning, with the purpose to leverage labeled data as less as possible with large amount of unlabeled data to improve prediction performance. We will extend the unsupervised clustering approaches to handle clustering or classification with partial label or side information. We will introduce to extend GMM/spectral

clustering with labeled data for classification. Especially, we will talk about transductive learning (SVM) for classification and clustering with pairwise constraints.

(4) structured output prediction: we will focus on hidden Markov models (HMM) and conditional random fields (CRFs). Both models can handle the structural output, such as sequential labeling and image segmentation problem. The former is a generative model, where the observations depends on hidden variables and EM algorithm is used to learn model parameters, while the latter is a discriminative model, which takes a total supervised approach to learn model parameters.

(5) Finally, we will introduce the latest trends in deep learning for both supervised/unsupervised tasks. We will focus on deep neural network(DNN), convolutional neural networks (CNN), deep belief networks (DBN), Autoencoder (AE) and recurrent neural networks (RNN). Especially, we will give a detail introduction on backpropagation, including all mathematical operations to derive gradients and learn model parameters.

In appendix A, we will discuss different methods to handle the quadratic optimization problem, especially for SVMs. Then we will talk about basics in probability, such as distribution, sets, correlation analysis (e.g. mean and variance) and likelihood in appendix B.

Chapter 2

Similarity Measures

Similarity is to measure how much alike two objects are, where the objects are points or vectors. In machine learning, similarity is usually evaluated with some metrics or described as a distance between (feature representations of) objects. If this distance is large, it shows the lower similarity; if it is small, there will be high degree of similarity. On the one hand, similarity is subjective and sensitive to different data type and domain knowledge, i.e. we prefer to use inner product to measure similarity for binary vectors. On the other hand, similarity is widely used in machine learning, and how to define similarity or distance function is vital in machine learning algorithms. For example, we need to compute distance, such as Euclidean or Gaussian similarity between data points to do clustering analysis (e.g. k-means and Gaussian mixture model). In linear classification, we generally use inner product to measure the orientation between weight vector and data points and then assign labels. Thus, similarity metrics are the fundamentals of machine learning algorithms, which can directly determine the final performance (good or bad). In this chapter, we will introduce some widely used similarity functions and their applications.

2.1 Introduction

The key component for successful machine learning algorithms is how to design or learn similarity function from data. No matter in supervised or unsupervised applications, we hope to discover the intrinsic measure, which can best capture the similarity

or distance between data points and generalize well on new data. The most common case is that we define the similarity function and then use it to do clustering or classification. For example, in k-means, we always use Euclidean distance to measure the similarity between data points, and then assign clustering labels based on such distance. Another research topic is to learn the hidden similarity function, which could be linear (e.g. Mahalanobis distance) or nonlinear (e.g. kernel or deep learning). In other words, we can learn such metric and then use it to do prediction. For example, in semi-supervised clustering case, we always use the pairwise constraints (must-link and cannot-link) to learn the hidden metric. In general, for any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^D$, the metrics used to evaluate the similarity should satisfy the following conditions:

$$
\begin{aligned}
&(1)\ d(\mathbf{x}, \mathbf{y}) \geq 0, \text{ non-negative} \\
&(2)\ d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}), \text{ symmetric} \\
&(3)\ d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}), \text{ triangle inequality}
\end{aligned}
$$

Most popular metrics satisfy the properties above, but there are many distance metrics lack one or more properties, which is out of scope here.

2.2 Distance measures on vectors

Suppose we have a dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$. To measure the similarity between two vectors, we list some similarity measures below.

2.2.1 Euclidean Distance

Euclidean distance is one of the most common similarity measures. Given any two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, the Euclidean distance s_{ij} between the two vectors is:

$$
s_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||^2 = \sum_k (\mathbf{x}_{ik} - \mathbf{x}_{jk})^2 \tag{2.1}
$$

More specifically, the Euclidean distance between two vectors is the summation of square difference, and is the best proximity measure when data is dense or continuous. The Euclidean distance is also known as L_2 distance.

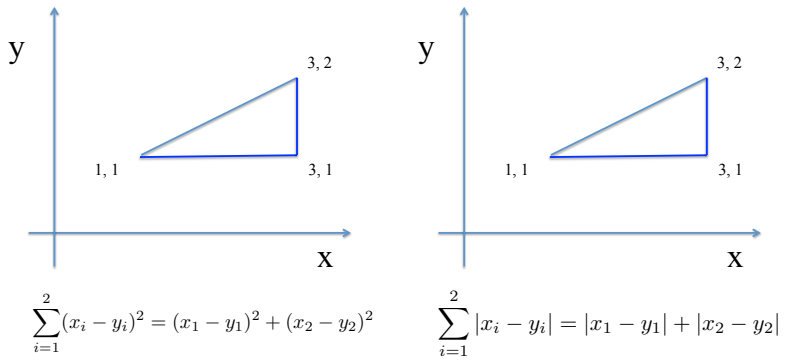


Figure 2.1. (a) The euclidean distance in a 2 dimensional case; (b) the Manhattan distance

For example in Fig. 2.1(a), we consider a two dimensional case (d=2) with $\mathbf{x}_1 = [1,1]$ and $\mathbf{x}_2 = [3,2]$, then the Euclidean distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ is $s_{12} = (1-3)^2 + (1-2)^2 = 5$.

2.2.2 Manhattan Distance

Manhattan distance is defined as the sum of the absolute differences of their Cartesian coordinates:

$$s_{ij} = \sum_{k=1}^{d} |\mathbf{x}_{ik} - \mathbf{x}_{jk}| \tag{2.2}$$

If the dimension $d = 2$ (x-axis and y-axis), we just have to sum up the absolute difference in x-axis and y-axis directions to calculate the Manhattan distance between them. This Manhattan distance metric is also known as Manhattan length, rectilinear distance, L_1 distance, L_1 norm, or city block distance.

For example in Fig. 2.1(b), we consider the same two dimensional case (d=2) with $\mathbf{x}_1 = [1,1]$ and $\mathbf{x}_2 = [3,2]$, then the Manhattan distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ is $s_{12} = |1-3| + |1-2| = 3$.

2.2.3 Minkowski distance

The Minkowski distance is a generalized metric form of Euclidean distance and Manhattan distance. It looks like this:

$$s_{ij} = \sum_{k=1}^{d} |\mathbf{x}_{ik} - \mathbf{x}_{jk}|^{\lambda} \tag{2.3}$$

where λ is the order of the Minkowski metric. Although it is defined for any $\lambda > 0$, it is rarely used for values other than 1, 2 and ∞. if $\lambda = 1$, then it is the Manhattan distance, a.k.a L_1-norm. if $\lambda = 2$, then it is Euclidean distance. When $\lambda = \infty$, it is the Chebyshev distance, which is a synonymy of L_{max}-Norm or Chessboard distance.

2.2.4 Gaussian similarity

The Gaussian similarity measure how close two objects are, which extends Euclidean distance. Given any two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, the Gaussian similarity is defined as

$$s_{ij} = exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/(2\sigma^2)) \tag{2.4}$$

it evaluates to 1 if the two input arguments are identical, and approaches 0 as the two vectors get very far apart. The Gaussian kernel similarity will be widely used in clustering analysis, especially spectral clustering.

2.2.5 Inner production

Inner production, or dot product or scalar product, is an algebraic operation that takes two coordinate vectors, multiplies corresponding elements in each dimension and returns a single number. The dot product is directly related to the cosine of the angle between two vectors in Euclidean space of any number of dimensions.

$$s_{ij} = \mathbf{x}_i^T \mathbf{x}_j = \sum_{k=1}^{d} \mathbf{x}_{ik} \mathbf{x}_{jk} \tag{2.5}$$

Larger s_{ij}, higher similarity between the two vectors. Note that in Euclidean distance, smaller s_{ij}, higher degree of similarity. Thus, the similarity measure of inner product

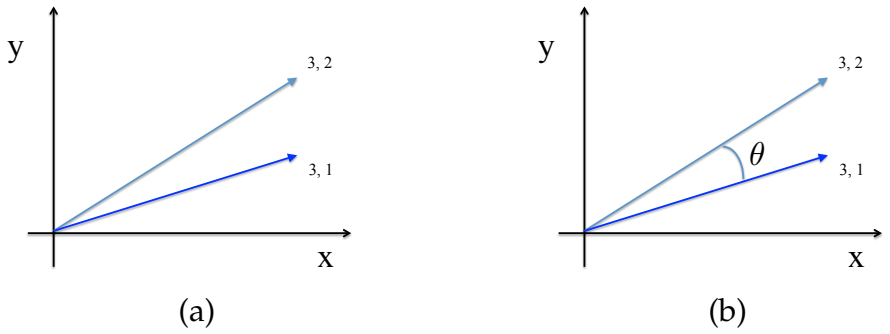


Figure 2.2. (a) it shows two vectors (2-dimension) [3, 2] and [3, 1]; (b) shows the same two vectors included the angle θ. The difference between inner production and cosine similarity is that the former calculates the magnitude (and orientation), while the latter only considers the relative distance derived from the angle.

is just opposite from Euclidean distance. In addition, inner production consider both orientation and magnitude, so it is widely used in machine learning. However, the inner production between two vectors does not consider normalization, it may not work well for not normalized data.

Example: given two points $\mathbf{x}_1 = [3, 2]$ and $\mathbf{x}_2 = [3, 1]$ shown in Fig. 2.2(a), then the inner production between $\mathbf{x}_1$ and $\mathbf{x}_2$: $s_{12} = [3, 2][3, 1]^T = 3 \times 3 + 2 \times 1 = 11$.

2.2.6 Cosine similarity

In Euclidean space, a vector is a geometrical object that possesses both a magnitude and a orientation. Then the dot product of two vectors can be defined by:

$$\mathbf{x}_i^T \mathbf{x}_j = ||\mathbf{x}_i|| ||\mathbf{x}_j|| \cos\theta \tag{2.6}$$

where θ is the angle between $\mathbf{x}_i$ and $\mathbf{x}_j$, and $||\mathbf{x}_i||$ magnitude of $\mathbf{x}_i$. The Cosine similarity metric finds the normalized dot product of the two vectors (or attributes), and is defined as:

$$s_{ij} = \cos\theta = \frac{\mathbf{x}_i^T \mathbf{x}_j}{||\mathbf{x}_i|| ||\mathbf{x}_j||} \tag{2.7}$$

where $s_{ij} \in [-1, 1]$. It is thus a judgement of orientation correlation between two vectors: two vectors with the same orientation have a cosine similarity of 1, and two orthogonal vectors have a similarity of 0. While for two vectors with opposite direc-

tions, its Cosine similarity is -1. Thus, Cosine similarity uses the angle between two vectors to evaluate its distance, independent of their magnitude.

Example: given two points $\mathbf{x}_1 = [3, 2]$ and $\mathbf{x}_2 = [3, 1]$ in Fig. 2.2(b), then the cosine similarity between $\mathbf{x}_1$ and $\mathbf{x}_2$: $s_{12} = \frac{[3,2][3,1]^T}{\sqrt{3^2+2^2}\sqrt{3^2+1^2}} = \frac{11}{\sqrt{130}}$.

2.2.7 Pearson correlation

Pearson correlation is the ratio between the covariance and the standard deviation of both objects, with the following definition:

$$s_{ij} = \frac{\big(\mathbf{x}_i - E(\mathbf{x}_i)\big)^T \big(\mathbf{x}_j - E(\mathbf{x}_j)\big)}{\sqrt{||\mathbf{x}_i - E(\mathbf{x}_i)||^2 ||\mathbf{x}_j - E(\mathbf{x}_j)||^2}} \tag{2.8}$$

Compared to Cosine similarity, it first normalize the data, and the compute the ratio above. A Pearson Correlation Coefficient of 1 indicates that the two data objects are perfectly correlated but in this case, a score of -1 means that the data objects are not correlated. In other words, the Pearson Correlation score quantifies how well two data objects fit a line.

2.2.8 Hamming distance

Given two objects (e.g. strings or binaries) with the same length, the Hamming distance is the number of dimensions at which the corresponding values are different. In another way, it measures the minimum number of substitutions required to change one value into the other, for example switching 1 int 0 or 0 into 1. In most cases, the Hamming distance is used to measures the minimum number of differences between two binary vectors, which can be fast computed with XOR operation.

If $\mathbf{x}_i$ and $\mathbf{x}_j$ are both binary vectors, the Hamming distance can be defined as:

$$s_{ij} = \sum_{k=1}^{d} \mathbf{x}_{ik} \oplus \mathbf{x}_{jk} \tag{2.9}$$

where $\oplus$ is XOR operation. Example: $\mathbf{x}_1 = 10001010$, $\mathbf{x}_2 = 00011000$, then the Hamming distance is: $s_{12} = 10001000 \oplus 00011000 = 1+0+0+1+0+0+1+0 = 3$ because we need to switch 3 times to make $\mathbf{x}_1$ and $\mathbf{x}_2$ equal.

2.3 Distance measures on probability

2.3.1 Chi-Square distance

Lets say that P and Q are distributions or histograms, which describe the probability belonging to different categories or classes and we want to measure if histogram P and histogram Q have same or different proportions of members across these categories, then we can use Chi-Square or KL-Divergence to measure their distance. More specifically, the Chi-Square between any two positive vectors is defined:

$$s_{chi} = \frac{1}{2}\sum_{k=1}^{d}\frac{\big(P(k)-Q(k)\big)^2}{\big(P(k)+Q(k)\big)} \tag{2.10}$$

Example: given $P = [0.5, 0.1, 0.4]$ and $Q = [0.6, 0.2, 0.2]$, then the Chi-Square distance between P and Q is: $s_{chi} = \frac{1}{2}[\frac{(0.5-0.6)^2}{0.5+0.6} + \frac{(0.1-0.2)^2}{0.1+0.2} + \frac{(0.4-0.2)^2}{0.4+0.2}] = \frac{3}{55}$

2.3.2 KL-Divergence

Kullback-Leibler divergence, also known as information divergence, information gain, relative entropy, or KL divergence, is a measure of the difference between two probability distributions P and Q.

For discrete probability distributions P and Q, the KullbackLeibler divergence of Q from P is defined to be

$$s_{KL}(P||Q) = \sum_{k} P(k)\log\frac{P(k)}{Q(k)} \tag{2.11}$$

For continuous probability, the sum should be changed into integration.

2.4 Distance measures on sets

So far, weve discussed some metrics to measure the similarity between objects, e.g. points, strings, or vectors. In this part, we introduce similarities between sets, and Jaccard Similarity to find similarities between sets. So first, lets learn the very basics of sets.

Sets: A set is (unordered) collection of unordered objects. For example, $\mathcal{S} = \{a, b, c\}$ contains 3 elements, a, b and c. In general, we use curly brackets $\{\}$ to represent a set, with elements separated by commas inside the pair curly brackets. Also, the elements in a set are unordered, so $\{a, b, c\} = \{b, a, c\} = \{c, a, b\}$.

Cardinality: The Cardinality of a set $\mathcal{S}$ (denoted by $|\mathcal{S}|$) counts how many elements are in $\mathcal{S}$.

Intersection: The intersection between two sets A and B is denoted $A \cap B$ and reveals all items which are in both sets A and B.

Union: The union between two sets A and B is denoted $A \cup B$ and reveals all items which are in either set. More details related to sets, please refer to Appendix B.

2.4.1 Jaccard similarity

Now going back to Jaccard similarity. The Jaccard similarity measures similarity between two finite sample sets (refer to Appendix B), and is defined as the cardinality of the intersection of sets divided by the cardinality of the union of the sample sets. Suppose you want to find Jaccard similarity between two sets A and B, it is the ratio of cardinality of $A \cup B$ and $A \cap B$. Jaccard similarity

$$s_J(A||B) = \frac{A \cap B}{A \cup B} \tag{2.12}$$

Example: $A = \{1, 3, 8, 11\}$ and B = { 0, 1, 2, 3, 6, 8, 9, 12 }, the Jaccard similarity is $S_J(A||B) = \frac{3}{9} = 0.333$. This gives us a single numeric score to measure the similarity between two vectors. In general, we can map encode A and B into binary representation, and then use And operation to calculate the similarity score. Note that $A \cup B = \{0, 1, 2, 3, 6, 8, 9, 12\}$, so if the i-th in A is in $A \cup B$, we set 1 and 0 otherwise. This captures the same data set that A represents, but make it sparse with binary representation. Thus we can represent A and B respectively as $A = [010101010]$, $B = [111111101]$, and further we have $A \cap B = \sum_i A(i) \wedge B(i) = 3$.

However, it is become cumbersome when the sets (or dimension) are quite large. Fortunately, there is a technique called min Hashing, which uses a randomized algorithm to quickly estimate the Jaccard similarity. More details, please refer to utah Jeff M. Phillips, Univ. of Utah.

2.4.2 Generalized Jaccard similarity

Generalized Jaccard similarity can be applied to positive real value vectors, which can be thought to be the same as Minhash.

$$s_{GJ}(A||B) = \frac{\sum_k \min(A(k), B(k))}{\sum_k \max(A(k), B(k))} \tag{2.13}$$

where $A(k)$ indicates the value in k-th dimension of vector A.

Example: $A = [0.5, 0.1, 0.4]$ and $B = [0.6, 0.2, 0.2]$, then the generalized Jaccard similarity is: $s_{GJ} = \frac{0.5+0.1+0.2}{0.6+0.2+0.4} = \frac{0.8}{1.2} = 0.667$

Chapter 3

Supervised Learning

Supervised learning is a subfield in machine learning, which learns a mapping from one space to another space. Let the domain of observations be $\mathbf{X}$, and the target domain be Y. And $P(\mathbf{X}, Y)$ is the unknown joint distribution on the instance-label space $\mathbf{X} \times Y$. In this framework, we need two spaces of objects $\mathbf{X}$ and Y, which usually appears as instance-label pairs in a supervised manner. Given the training pairs, and a hypothesis space $\mathcal{H}$ of functions $h : \mathbf{X} \mapsto Y$, the learner aims to select a function h which can output $y \in Y$, given $\mathbf{x} \in \mathbf{X}$.

In supervised learning, the goal here is to learn h given the training set consisting of N pairs $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$, and also predicts the label of unseen data. Ideally, we hope h fits the data exactly. However, if the training data has noise, it may not be the best choice as it may lead to a poor performance on unseen instances. Thus, the general idea of supervised learning is to look for possible models, which can fit well the data, but also as simple as possible (i.e. generalization).

In statistical machine learning, we need to assume that the future observations are related to the history ones. Furthermore, they are both sampled independent from the same distribution (i.i.d.). In other words, the observed phenomenon (i.e. training data) give information about the underlying probability distribution. To discover the patterns behind the phenomenon, we need to do at least three assumptions: (1) what is the joint distribution $P(\mathbf{x}, y)$; (2) what is the function formula, e.g. linear or nonlinear; (3) what the objective function. In a high level, we will first talk about objective function, and then introduce several linear classifiers.

3.1 Background

Assume the training set is sampled i.i.d. from a joint probability distribution $P(\mathbf{x}, y)$ over $\mathbf{X}$ and Y. Note that the joint probability assumption allows us to model uncertainty (i.e. noise) when we make prediction. Specifically, y is not a deterministic function of $\mathbf{x}$, but rather a random variable with conditional distribution $P(y|\mathbf{x})$ for a fixed $\mathbf{x}$. We also assume we are given a non-negative real value loss function $\ell(\hat{y}, y)$, which measure how different the prediction $\hat{y}$ of a hypothesis is from the true outcome y.

To learn the mapping h, we need a criterion to choose it. The criterion is a low prediction error. The risk associated with hypothesis $h(\mathbf{x})$ is then define as the expectation of the loss function

$$\mathbf{R}(h) = \mathbf{E}[\ell(h(\mathbf{x}), y)] = \int \ell(h(\mathbf{x}), y) P(\mathbf{x}, y) \tag{3.1}$$

A loss function commonly used in theory is the 0-1 loss function: $\ell(\hat{y}, y) = \delta(\hat{y}, y)$, where δ is the indicator function (return 0 if $\hat{y} = y$, otherwise 1). The ultimate goal of a learning algorithm is to find a hypothesis h^* among a fixed class of functions for which the risk is minimal:

$$h^* = argmin_{h \in \mathcal{H}} \mathbf{R}(h) \tag{3.2}$$

Empirical risk minimization in general, because the distribution $P(\mathbf{x}, y)$ is unknown to the learning algorithm (this situation is referred to as agnostic learning), we cannot compute the risk $\mathbf{R}(h)$, which is the expectation over $P(\mathbf{x}, y)$. We can only measure the agreement of a candidate function with the data. Fortunately, we can compute an approximation, called empirical risk, by averaging the loss function on the training set:

$$\mathbf{R}(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(\mathbf{x}_i), y_i) \tag{3.3}$$

Empirical risk minimization (ERM) principle states that the learning algorithm should choose a hypothesis which minimizes the empirical risk. Thus the learning algorithm defined by the ERM principle consists in solving the above optimization

problem.

Regularization Another to generalize the model well is to introduce regularization to the objective function. Then one has to minimize the regularized empirical risk:

$$h = argmin_{h \in \mathcal{H}} \mathbf{R}(h) + \lambda ||h||^2 \tag{3.4}$$

where λ is called the regularization parameters, which allows to choose the right trade-off between fit and complexity. In addition, we use L_2 in Eq. 3.4, other norm can be used too, such as L_1. Tuning λ is a hard task, and usually selected with extra validation set.

As we mentioned before, we will need to explicitly or implicitly specify or learn distance function on the input data space, no matter for supervised or unsupervised learning algorithm. For the supervised tasks, we hope to learn a model, which should be close to all instances with the desired output/target and far away to any instance with the undesired target. In this chapter, we will use inner product as the distance measure, and introduce some widely used supervised learning approaches, such as perceptron training, support vector machines (SVM), regression, etc.

Bias-variance tradeoff In supervised machine learning, the bias-variance dilemma is the problem to balance the tradeoff between bias and variance, given the training set. More specifically, the expected generalization error can be decomposed into the sum of bias and variance, and is hard to simultaneously minimize these two sources of errors.

In linear regression, we can decompose the generalization error below

$$\begin{aligned}
&E\big(y - \hat{f}(x)\big)^2 = E\big[y - E(\hat{f}(x)) + E(\hat{f}(x)) - \hat{f}(x)\big]^2 \\
=&E\Big[(y - E(\hat{f}(x)))^2 + 2E\big((y - E(\hat{f}(x)))(E(\hat{f}(x)) - \hat{f}(x))\big) + E(E(\hat{f}(x)) - \hat{f}(x))^2\Big] \\
=&E\Big[\big(y - E(\hat{f}(x))\big)^2\Big] + E\Big[\big(E(\hat{f}(x)) - \hat{f}(x)\big)^2\Big] \\
=&Var + Bias^2
\end{aligned} \tag{3.5}$$

The bias term is related to underfitting, where the model assumption cannot fit the model to observations well. In general, it is caused by erroneous model assumption

over the training data. The variance term is related to overfitting, where the model fit the observations well (even the noise data), but cannot generalize well to unseen data. The bias-variance tradeoff is an vital problem in supervised learning. In an ideal case, we want to captures the data distribution accurately in the training data, but also want to generalize well to unseen data. From the above decomposition, it is not possible to minimize the both errors simultaneously. High variance learning methods generally use complex models to represent training set, but at the risk of overfitting. In contrast, high bias algorithms use simple models to fit the training data, but in general cannot capture valuable data distribution.

3.2 Perceptron training

The Perceptron [5] is an algorithm for supervised learning of binary classifiers, which was invented in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory. It was one of the first artificial neural networks (the single-layer or multi-layer perceptron) to be produced and used for classification. In this part, we will first introduce its linear classifier for the binary case and then we will generalize it to multiclass. For the multi-layer perceptron, we will mention it the deep learning chapter 7.

For the Perceptron, we need to learn a linear mapping function $f : \mathbb{R}^d \mapsto \{-1, 1\}$. Suppose we have learnt the model parameter $\mathbf{w} \in \mathbb{R}^d$ and bias b from the training set $\mathcal{D}$. Given an unseen instance $\mathbf{x}$, we can decide its label with the following decision function:

$$f(x) = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + b > 0 \\ -1, & \text{otherwise} \end{cases} \tag{3.6}$$

Note that $\mathbf{w}^T\mathbf{x} + b = [\mathbf{w}, b]^T[\mathbf{x}, 1]$, so we can incorporate the bias b into the model weight $\mathbf{w}$ by extending $\mathbf{x}$ into $[\mathbf{x}, 1]$.

According to Eq. 2.5, we hope to maximize the correlation $\mathbf{w}^T\mathbf{x}$ for $\mathbf{x}$ from positive instances. In other words, we need to find $\mathbf{w}$ which has higher positive correlation to instances in positive classes, and lower or even no correlation to negative instances. For the viewpoint of distance function in Eq. 2.5, we will find a vector $\mathbf{w}$, which are close to positive instances, while are far away from negative ones.

To learn the model, we need to define the loss function first. One widely used

function is zero-one loss

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} 1, & \text{if } f(\mathbf{x}) = y \\ -1, & \text{otherwise} \end{cases} \tag{3.7}$$

Then given the training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where y_i can be mapped into $\{1, -1\}$ for binary case, we can get the following empirical risk (or loss)

$$\mathcal{R}_{emp}(\mathbf{w}, b) = \sum_{i=1}^{N} \mathcal{L}(y_i, f(\mathbf{x}_i)) = \sum_{i=1}^{N} \delta(y_i, f(\mathbf{x}_i)) \tag{3.8}$$

However, it is not easy to calculate the gradient w.r.t. $\mathbf{w}$. So we can rewrite Eq. 3.8 as follows

$$\begin{aligned} \mathcal{R}_{emp}(\mathbf{w}, b) &= \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} -y_i f(\mathbf{x}_i) \\ &= \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} -y_i(\mathbf{w}^T \mathbf{x}_i + b) \end{aligned} \tag{3.9}$$

Note that if $y_i != f(\mathbf{x}_i)$, then $y_i f(\mathbf{x}_i) = -1$. So we need to add minus sign to construct the empirical loss in Eq. 3.9. This just says that correctly classified examples dont incur any loss at all, while incorrectly classified examples contribute $(\mathbf{w}^T \mathbf{x} + b)$, which is some sort of measure of confidence in the (incorrect) labeling.

Now, $\mathcal{R}_{emp}$ is differentiable in $\mathbf{w}$. The gradient of Eq. 3.9 is again a sum over the misclassified examples $\nabla_{\mathbf{w}} \mathcal{R}_{emp}(\mathbf{w}, b) = \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} -y_i \mathbf{x}_i$.

So we can compute the gradient w.r.t. $\mathbf{w}$ and use any gradient-based algorithm to learn it. If we use gradient descent to learn $\mathbf{w}$, we can update it with the following

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \mathcal{R}_{emp}(\mathbf{w}, b) \\ &= \mathbf{w}_t + \alpha \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} y_i \mathbf{x}_i \end{aligned} \tag{3.10}$$

where α, called the learning rate, is a parameter we need to specify in the algorithm.

Similarly, we can yield the following updating rule for b:

$$b_{t+1} = b_t + \alpha \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} y_i \tag{3.11}$$

As we have mentioned before, we can append a constant element 1 to each input vector $\mathbf{x}_i$ and incorporate b into $\mathbf{w}$. In other words, the dimensionality of all the vectors has increased by one, with an additional constant 1 appended, and the size of $\mathbf{w}$ also increases to $d + 1$. What we described above is the batch perceptron. However, when the training set is very large, it may be not scalable (i.e. it is not possible to store all the data in the memory). Thus the batch-based algorithm may be not feasible for large scale dataset.

The perceptron has an online learning algorithm, which can effectively learn the model in a stream way. In online learning there is no distinction between the training set and testing set. The input is a continuous stream of examples, and the algorithm has to make a prediction immediately after $\mathbf{x}_i$ arrives, and update the model if the prediction does not match its groundtruth. We will introduce its online learning algorithm below.

3.2.1 Parameters learning

Given the training data $\mathcal{D}$, we need to learn the weight $\mathbf{w}$ in Eq. 3.6. The learning algorithm of perceptron training is listed below:

Data: the training set $\mathcal{D}$, iterations T and learning rate α
Result: output the weight $\mathbf{w}$ and bias b
initialize the weights $\mathbf{w}$ and b;
set iteration index $t = 0$;
while $t < T$ **do**
    missed = 0 ;
    for *Each* $\mathbf{x}_i \in \mathcal{D}$ **do**
        predict $\hat{y}_i = \mathbf{w}^T \mathbf{x}_i + b$;
        if $\hat{y}_i != y_i$ **then**
            update the weights: $\mathbf{w} = \mathbf{w} + \alpha(y_i - \hat{y}_i)\mathbf{x}_i$;
            missed++;
        end
    end
    if *missed* $==$ N **then**
        break;
    end
    t++;
end

Algorithm 1: Perceptron online learning algorithm

In each iteration, the algorithm will go over all training instances, and update the model if the prediction does not match its groundtruth. Apparently, the vial step in Algorithm 1 is to update the model $\mathbf{w}$ when the prediction is wrong:

$$\mathbf{w} = \mathbf{w} + \alpha(y_i - \hat{y}_i)\mathbf{x}_i \tag{3.12}$$

According to the updating rule, the model $\mathbf{w}$ is the linear combination of the training set $\{\mathbf{x}_i\}_{i=1}^N$. This formulation can be written as:

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i \tag{3.13}$$

where α_i is the weighing coefficient (it is a real value) for each instance $\mathbf{x}_i$. In the late chapters, this formula will appear on other linear classifiers, such as support vector

machines (SVM). If $y_i \in \{-1, 1\}$, then we can rewrite Eq. 3.13 as follows:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{3.14}$$

We can think that it just absorbs the sign $y_i \in \{-1, 1\}$ into α_i. In other words, the final model $\mathbf{w}$ is a linear combination of the training data.

3.2.2 Regularized Perceptron learning

As mentioned in Eq. 3.4, we can incorporate regularization term to Eq. 3.9, then we get the following objective function

$$\begin{aligned} \mathcal{R}_{emp}(\mathbf{w}, b) &= \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} -y_i f(\mathbf{x}_i) + \frac{\lambda}{2}||\mathbf{w}||^2 \\ &= \sum_{i \in \{1,2,\ldots,N\}: y_i != f(\mathbf{x}_i)} -y_i(\mathbf{w}^T \mathbf{x}_i + b) + \frac{\lambda}{2}||\mathbf{w}||^2 \end{aligned} \tag{3.15}$$

Similarly, we can take gradient w.r.t. $\mathbf{w}$, and use gradient descent to update it. In fact, we will introduce more in linear SVMs, which generalizes Perceptron training and more robust in classification.

3.2.3 Correlation analysis

Given the training pair $(\mathbf{x}_i, y_i)$, when its prediction $\hat{y}_i$ is wrong, the algorithm will add $\mathbf{x}_i$ into the weights $\mathbf{w}$ with a learning rate α in Eq. 3.12. Thus, it will increase the correlation between $\mathbf{w}$ and $\mathbf{x}$, and move the prediction $\mathbf{w}^T \mathbf{x}_i + b$ toward the target.

More specifically, assume that $\mathbf{x}_i$ has the positive label ($y_i = 1$). Then, we consider the following two cases:

- If $\hat{y}_i = 0$, then it indicates the model makes the wrong prediction. So we need to correct the mistake by moving $\mathbf{w}$ towards $\mathbf{x}_i$. Remember the correlation between two vectors in Eq. 2.5, thus, we always have $\mathbf{x}_i^T \mathbf{x}_i \geq 0$ for any real vector $\mathbf{x}_i$. If we can add $\mathbf{x}_i$ into the model $\mathbf{w}$, then $(\mathbf{w} + \alpha \mathbf{x}_i)^T \mathbf{x}_i = \mathbf{w}^T \mathbf{x}_i + \alpha \mathbf{x}_i^T \mathbf{x}_i$. Note that $\alpha \mathbf{x}_i^T \mathbf{x}_i$ is positive for non zero vector, thus Eq. 3.12 will always move $f(\mathbf{x}_i)$ into the positive direction.

- If $\hat{y}_i = 1$, it shows that we make the right prediction, then no update needed to the model $\mathbf{w}$.

Thus, the linear class $f(x) = \mathbf{w}^T\mathbf{x} + b$ attempts to learn a model $\mathbf{w}$, which has higher correlation to its positive training instances, and lower correlation to its negative training sets.

3.2.4 Convergence analysis

If the training set $\mathcal{D}$ is separable, then perceptron training can converge and predict all $\mathcal{D}$ correctly. While the perceptron algorithm is guaranteed to converge on some solutions in the case of a linearly separable training set, it may still pick any feasible solution. Note that perceptron training is an online learning algorithm, where the training instances should be randomized to improve the robustness and generalize well on the new dataset. Because the random order of the training set, it may converge (i.e. no wrong prediction) and problems may admit many solutions of varying quality.

However, if the training set $\mathcal{D}$ is not linearly separable, it will never predict all the input vectors classified correctly because the perceptron model is a linear model.

3.2.5 Multiclass perceptron

Let $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$ ($\mathbf{x}_i \in \mathbb{R}^d$) be a set of N examples, which belongs to K classes called. The purpose is to learn mapping function $f : \mathbb{R}^d \to \mathbb{R}^K$ with weights $\mathbf{w}^k \in \mathbb{R}^d$ for $k = [1, K]$, to make sure the prediction on the training set as accurate as possible. Just like the multi-class classification problems [6], we use the joint feature representation $\Phi(\mathbf{x}, y)$ for each $(\mathbf{x}, y) \in \mathcal{D} \times \mathcal{K}$

$$\Phi(\mathbf{x}, y) = \begin{bmatrix} \mathbf{x} \cdot \delta(y = 1) \\ \cdots \\ \mathbf{x} \cdot \delta(y = K) \end{bmatrix} \tag{3.16}$$

where δ is the indicator function (1 if the equation holds, otherwise 0). Thus, $\Phi(\mathbf{x}, y)$ function maps each possible input/output pair to a finite-dimensional real-valued feature vector belong to $\mathbb{R}^{(K \times d) \times 1}$. Correspondently, the hyperplanes for the K classes

can be parameterized by the weight vector $\mathbf{W} \in \mathbb{R}^{(K\times d)\times 1}$, which is the concatenation of weights $\mathbf{w}^k$, for $k = \{1, ..., K\}$. In other words, $\mathbf{W}[(k-1)\times d + 1 : k \times d] = \mathbf{w}^k$. The predication of testing examples is done in the same manner as follows,

$$\max_{y\in[1,K]} f(x, y) = \mathbf{W}^T \Phi(\mathbf{x}, y) \tag{3.17}$$

For inference, we first project data $\mathbf{x}$ into K dimension variables (output) with function f and then decide its class which has the largest value in the output in Eq. 3.17. The problem left is how to learn the weight parameter $\mathbf{W}$.

The learning procedure for the multi-class perceptron is similar to its binary case:

Data: the training set $\mathcal{D}$, iterations T and learning rate α

Result: output the weight $\mathbf{w}$ and bias b

initialize the weights $\mathbf{w}$ and b;

set iteration index t = 0;

while $t < T$ **do**

  missed = 0 ;

  for *Each* $\mathbf{x}_i \in \mathcal{D}$ **do**

    predict $\hat{y}_i = max_{y\in\{1,K\}} f(\mathbf{x}_i, y_i)$ according to Eq. 3.17 ;

    if $\hat{y}_i != y_i$ **then**

      update the weights: $\mathbf{W} = \mathbf{W} + \alpha\big(\Phi(\mathbf{x}_i, y_i) - \Phi(\mathbf{x}_i, \hat{y}_i)\big)$;

      missed++;

    end

  end

  if $missed == N$ **then**

    break;

  end

  t++;

end

Algorithm 2: Perceptron learning for multi class

Compared to its binary algorithm, the multiclass perceptron can use the joint feature representation in Eq. 3.16, so that it can has the same learning formula as its binary algorithm.

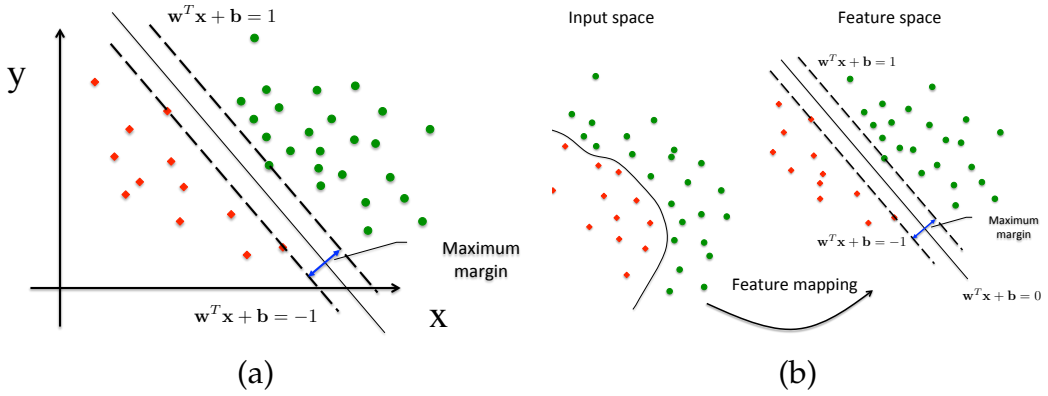


Figure 3.1. (a) shows a linear SVM for classification, where the data is linear separable; (b) shows a kernel SVM for binary classification, where the original data can be linear separable in the mapping space.

3.3 Support vector machines

A support vector machine (SVM) learns a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. In general the larger the margin, the lower the generalization error of the classifier. Thus, compared to perceptron training, SVM attempts to construct a classifier, which keeps large margins between positive instances and negative instances. The original SVM algorithm [7] was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis. And later soft margin was proposed by Corinna Cortes and Vapnik in 1993, and the kernel trick was applied to learn maximum-margin hyperplanes to handle nonlinear separable dataset. In the following parts, we will introduce linear SVM and kernel (nonlinear) SVM.

3.3.1 Linear SVM

We first consider linear SVM classifier in Fig. 3.1(a). Given the training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)|x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\}_{i=1}^N$, where y_i is either 1 or -1, indicating the class to which the point belongs. We want to find the hyperplane, such that the largest score for assigning $\mathbf{x}_i$ into its class should be greater than that for assigning it into undesired class by at least 1 (soft margin can be applied here too, which will be introduced later).

Thus, we have the following classification model:

$$f(x) = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 1 \\ -1, & \text{otherwise } \mathbf{w}^T\mathbf{x} + b \leq -1 \end{cases} \tag{3.18}$$

If the training data is linearly separable, we want to find two hyperplanes, which can predict all training data correctly. In other words, for the positive vectors (or points), we have $\mathbf{w}^T\mathbf{x}_i + b \geq 1, \forall y_i = 1$; while for all negative vectors, it should satisfy $\mathbf{w}^T\mathbf{x}_i + b \leq -1, \forall y_i = -1$. So the two hyperplanes separate the data well and there are no points between them (or misclassified). Note that the two hyperplanes are parallel, so its geometrical distance is $\frac{2}{||\mathbf{w}||}$. To maximize the margin between the two hyperplanes, we should minimize $||\mathbf{w}||$ (which in turn will maximize $\frac{2}{||\mathbf{w}||}$). In addition, we also hope all instances are predicted correctly (or minimize the classification error), so the model $\mathbf{w}$ also need to satisfy $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \forall i \in [1, N]$.

Linear SVM with hinge loss: If the prediction $(\mathbf{w}^T\mathbf{x} + b)$ matches its true output y, then $y(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$, and we have $\ell(y, f(x)) = 0$. Otherwise $y(\mathbf{w}^T\mathbf{x} + b) < 1$, and it induces a loss $\ell(y, f(x)) = 1 - y(\mathbf{w}^T\mathbf{x} + b)$. Let's define the hinge loss $\ell(y, f(x)) = max\big(0, 1 - y(\mathbf{w}^T\mathbf{x} + b)\big)$, which measures the loss when there are misclassified instances.

Then, we need to minimize the hinge loss, and meanwhile we also need to maximize the margin between the two hydroplanes, so we minimize the following objective function

$$\mathbf{w} = \underset{\mathbf{w}}{\text{argmin}} \, ||\mathbf{w}||^2 + C \sum_{i=1}^{N} max\big(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\big) \tag{3.19}$$

As mentioned in the perceptron learning in Eq. 3.9, we can compute gradients w.r.t. $\mathbf{w}$, and use gradient descent to optimize it.

$$\frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}} = \mathbf{w} - C \sum_{\substack{i=1 \\ 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0}}^{N} y_i\mathbf{x}_i \tag{3.20}$$

Then, we can update $\mathbf{w}$ with gradient descent:

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}} \tag{3.21}$$

where α is the learning rate.

Also we can use stochastic gradient descent to update $\mathbf{w}$. More specifically, for the current training data $(\mathbf{x}_i, y_i)$, if $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$, then it indicates wrong prediction. Thus, we can update $\mathbf{w}$

$$\mathbf{w} = \mathbf{w} + \alpha C y_i \mathbf{x}_i \tag{3.22}$$

The model to update $\mathbf{w}$ depends on the misclassified instances, which shares the same formula as Perception training.

Linear SVM with Lagrange multipliers: To learn linear SVM, we can optimize the following problem

$$\begin{aligned} &\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}} ||\mathbf{w}||^2 \\ &s.t.\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq= 1,\ \forall i \in [1, N] \end{aligned} \tag{3.23}$$

where both $||\mathbf{w}||^2$ and $\mathbf{w}^T\mathbf{x}_i + b$ have continuous first partial derivatives, with respect to $\mathbf{w}$. It is a quadratic programming with linear constraints. To optimize it, we can introduce Lagrange multiplier to handle the linear constraints. More specifically, we can rewrite the optimization problem above in order to apply the method of Lagrange multipliers

$$\begin{aligned} &\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}} ||\mathbf{w}||^2 \\ &s.t.\ 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0,\ \forall i \in [1, N] \end{aligned} \tag{3.24}$$

To incorporate these conditions into one equation, we can introduce a new variable α_i called a Lagrange multiplier and study the Lagrange function (or Lagrangian) defined by

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 + \sum_{i=1}^{N} \alpha_i (1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$

$$s.t.\ \alpha_i \geq 0,\ \forall i \in [1, N] \tag{3.25}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^N$ is the concatenation of α_i for $i = \{1, 2, ..., N\}$, and it satisfies the following condition $\alpha_i(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) = 0$ for any $i \in [1, N]$.

Primal form: Fix $\mathbf{w}$ and maximize w.r.t. the Lagrange multipliers

$$\hat{\mathcal{L}}(\mathbf{w}, b) = \max_{\boldsymbol{\alpha}} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) \tag{3.26}$$

$$= \begin{cases} \frac{1}{2}||\mathbf{w}||^2, & \text{if } 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0,\ \forall\ i \\ +\infty, & \text{otherwise} \end{cases} \tag{3.27}$$

If $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0,\ \forall\ i$, then the optimal α_i are all zeros; otherwise if $\exists\ i$, such that $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$, then it is unbound above as $\alpha_i \to \infty$.

Dual form: Fix Lagrange multipliers $\boldsymbol{\alpha}$ and minimize w.r.t. $\mathbf{w}$ and b

$$\hat{\mathcal{L}}(\boldsymbol{\alpha}) = \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) \tag{3.28}$$

To minimize the auxiliary function above, we can set its gradients to zero and solve

$$\begin{aligned} &\nabla_{\mathbf{w},b}\mathcal{L}(\mathbf{w}, b)) = 0 \\ \implies &\mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \\ \implies &\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \end{aligned} \tag{3.29}$$

Similarly, we can calculate the gradient w.r.t. b, and get

$$\sum_i \alpha_i y_i = 0 \tag{3.30}$$

According to $\mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$, then we have

$$\mathbf{w}^T(\mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i) = 0 \tag{3.31}$$

$$\implies \mathbf{w}^T\mathbf{w} - \sum_i \alpha_i y_i \mathbf{w}^T\mathbf{x}_i = 0 \tag{3.32}$$

further we have

$$\mathbf{w}^T\mathbf{w} = \sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i \tag{3.33}$$

substituting $||\mathbf{w}||^2$ and $\mathbf{w}$ with Eq. 3.33 and 3.29, plug them into Eq. 3.25, we can yield

$$\begin{aligned}
&= \frac{1}{2}\sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i + b\sum_i \alpha_i y_i - \sum_i \alpha_i [y_i \mathbf{w}^T \mathbf{x}_i - 1] \\
&= \sum_i \alpha_i - \frac{1}{2}\sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i \\
&= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i y_i \alpha_j y_j {\mathbf{x}_j}^T \mathbf{x}_i \\
&= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i y_i \alpha_j y_j {\mathbf{x}_i}^T \mathbf{x}_j \qquad (3.34) \\
&= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i y_i \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) \qquad (3.35)
\end{aligned}$$

where $k(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel defined by $k(\mathbf{x}_i, \mathbf{x}_j) = {\mathbf{x}_i}^T \mathbf{x_j}$. Finally, we can get the dual form

$$\begin{aligned}
\hat{\mathcal{L}}(\boldsymbol{\alpha}) &= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i y_i \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) \qquad (3.36) \\
s.t. \sum_{i=1}^{N} &\alpha_i y_i = 0 \\
&\alpha_i \geq 0, \forall i \in [1, N]
\end{aligned}$$

where $\boldsymbol{\alpha} = [\alpha_1, ..., \alpha_N]$. We can solve $\boldsymbol{\alpha}$ with quadratic programming, and then further solve $\mathbf{w}$ according Eq. 3.29. To solve the bias b, we either put the bias into $\mathbf{w}$ (by extending $\mathbf{x}$ into [$\mathbf{x}$, 1]), or else we solve it as follows: According to Eq. 3.25, we do the following analysis: If $\alpha_i = 0$, then $(\mathbf{x}_i, y_i)$ will not be active (or constrained) in the optimization problem. Otherwise, if $\alpha_i > 0$, then we have $1 - y_i(\mathbf{w}^T\mathbf{x} + b) = 0$ and $(\mathbf{x}_i, y_i)$ are called the **support vectors** since they support the hyperplanes on both sides of the margin, please refer to Fig 3.1(a). Then $b = y_i - \mathbf{w}^T\mathbf{x}_i$. Any i such that

$\alpha_i > 0$ can be used to calculate b, but in proactive, it is better to average the results obtained from support vectors.

3.3.2 Linear SVM with soft margin

Considering that the training set might be nonlinear separable, it should allow misclassified cases for the two hyperplanes discussed above. Thus, the soft margin method was proposed by In 1995, Corinna Cortes and Vladimir N. Vapnik, which introduces a nonnegative ξ_i to handle the misclassified examples, but at the same time keep a large margin.

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \forall i \in [1, N] \tag{3.37}$$

For the correct predictions, $\xi_i = 0$; while for the misclassified examples, we hope to minimize ξ_i, then maximize $1 - \xi_i$. Because $1 - \xi_i$ is the low bound of $y_i(\mathbf{w}^T\mathbf{x}_i - b)$ in Eq. 3.37, we can also maximize the distance to the nearest cleanly split examples.

Thus the objective function becomes:

$$\begin{aligned} &\underset{\mathbf{w},\xi,b}{\operatorname{argmin}}\{\frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i\} \\ &s.t.\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0\ \forall i \in [1, N] \end{aligned} \tag{3.38}$$

By incorporating the constraints, we can get the following auxiliary objective

$$\begin{aligned} \underset{\mathbf{w},\xi,b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i + \sum_i \alpha_i\big(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) - \xi_i\big) - \sum_i \beta_i\xi_i \\ s.t.\ \alpha_i \geq 0,\ \forall i \in [1, N] \end{aligned} \tag{3.39}$$

According to KKT conditions, by setting $\nabla_{\mathbf{w},\boldsymbol{\xi},b}\mathcal{L}(\mathbf{w}, \boldsymbol{\xi}, b)) = 0$, we yield

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{3.40}$$

$$\sum_i \alpha_i y_i = 0 \tag{3.41}$$

$$\alpha_i + \beta_i = C \tag{3.42}$$

This is a quadratic programming problems, which can be solved with many optimization methods, such as sequential minimal optimization (SMO), interior point methods and active set methods. We will introduce quadratic programming in the Appendix A.

Plug Eqs. 3.40, 3.41 and 3.42 back into objective function Eq. 3.38, we yield

$$\begin{aligned}
&\frac{1}{2}\sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i + C\sum_i \xi_i + \sum_i \alpha_i\big(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) - \xi_i\big) - \sum_i \beta_i \xi_i \\
=&\frac{1}{2}\sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i + \sum_i (C - \alpha_i - \beta_i)\xi_i + \sum_i \alpha_i - \sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i - b\sum_i \alpha_i y_i \\
=&\sum \alpha_i - \frac{1}{2}\sum_i \alpha_i y_i \mathbf{w}^T \mathbf{x}_i
\end{aligned} \tag{3.43}$$

According to $\alpha_i + \beta_i = C$, $\alpha_i \geq 0$ and $\beta_i \geq 0$, we have $0 \leq \alpha_i \leq C$. Further, we have

$$\begin{aligned}
\hat{\mathcal{L}}(\boldsymbol{\alpha}) = \arg\max_{\boldsymbol{\alpha}} &\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i y_i \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) \\
s.t.\ &\sum_{i=1}^{N} \alpha_i y_i = 0 \\
&0 \leq \alpha_i \leq C, \forall i \in [1, N]
\end{aligned} \tag{3.44}$$

where $k(\mathbf{x}_i, \mathbf{x}_j) = {\mathbf{x}_i}^T \mathbf{x_j}$. This equation above has the same objective function as Eq. 3.36, and the only difference is that it has bounded constraints on α_i. We can further rewrite the objective function Eq. 3.44 as follows:

$$\begin{aligned}
\hat{\mathcal{L}}(\boldsymbol{\alpha}) = \arg\max_{\boldsymbol{\alpha}} &\boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T H \boldsymbol{\alpha} \\
s.t.\ &\boldsymbol{\alpha}^T \mathbf{y} = \mathbf{0} \\
&0 \leq \alpha_i \leq C, \forall i \in [1, N]
\end{aligned} \tag{3.45}$$

where $H_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$. To optimize this quadratic programming above, we can use a varieties of methods, such as gradient descent, sequential quadratic programming and active set methods. More details to optimize the above objective, please refer to Appendix A. In the following part, we will introduce kernel SVM, which generalizes

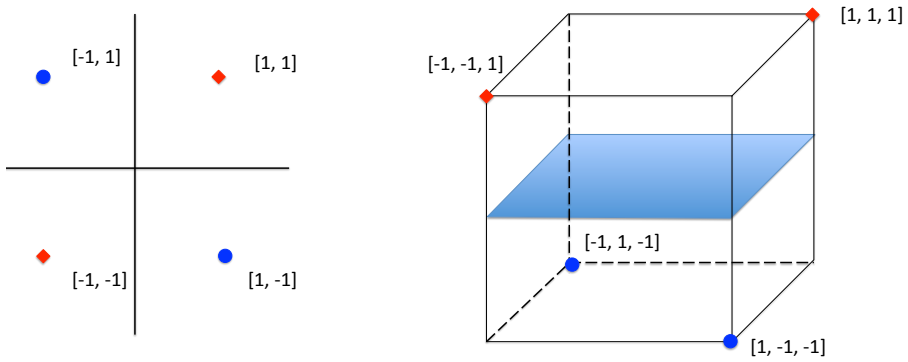


Figure 3.2. (a) It is a not linear separable in the 2-dimension case, which shows four data points belonging to two classes (red and blue respectively); (b) it shows to map the original 2-dimension into 3 dimension, where the original data can be linear separable in the mapping space.

K, without explicitly feature mapping function.

3.3.3 Kernel SVM

Before we discuss SVM with kernel tricks, we will introduce feature expansion first. If the SVM classifier cannot separate the data linearly, then we can explicitly map $\mathbf{x}$ with some nonlinear function ϕ, so that decision boundary could be linear in the mapping space such as $\mathbf{w}^T \cdot \phi(\mathbf{x}) + \mathbf{b} = 0$, as shown Fig 3.1(b). For example $\mathbf{x} \in \mathbb{R}^2$ and we has 4 points, with two positive training examples [-1, -1], [1, 1] and two negative examples [-1, 1], [1, -1], shown in Fig. 3.2. In the two dimensional space, we cannot separate them linearly in the original 2-d case. But, we can map the training instances into the following space $\phi(\mathbf{x}) = [a, b, ab]$, then it becomes linear separable in the 3 dimensional space, where there exists many 3-d hyperplanes, which can easily separate the red points from blue ones.

Here are some popular feature sets, which are widely used for feature mapping:

- polynomials of degree of k: $1, a, a^2, b, b^2, ...$
- neural nets (sigmoids) $1/(1+e^{-a})$, $tanh(3a + 2b - 1)$
- RBFs of radius σ: $\exp\{(-\frac{1}{\sigma^2}((a-a_0)^2 + (b-b_0)^2))\}$

Thus, we can map the original data into high dimension case, which can be linear separable. However, there are many problems with feature mapping: (1) it yields lots

of features, e.g. polynomial of degree k on d original features yields $O(d^k)$ expanded features. More specifically, the feature vector for even simple kernels can blow up in size, such as the RBF kernel, in which the corresponding feature vector is infinite dimensional. In contrast, kernel methods do the similar mapping, which can handle infinite feature space and only require a user-specified kernel, i.e. a similarity function (e.g. Gaussian kernel) over pairs of data points in raw representation. (2) It is hard to define feature mapping, while it is easy to define similarity matrices. Moreover, many matrices can be written to only use dot products, and in turn we can replace the dot products with kernels (or similarity matrix). By doing so, we don't have to explicit define the feature vector at all. This introduces the kernel trick, which basically make use of kernel functions. The kernel can map the date into high dimensional space, without explicitly computing the coordinates of the data in that space. We will introduce the kernel trick in the following.

Now we consider the simple case with inner product, the kernel matrix is $K = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 & \mathbf{x}_1^T\mathbf{x}_2 & \mathbf{x}_1^T\mathbf{x}_3 & \dots & \mathbf{x}_1^T\mathbf{x}_n \\ \mathbf{x}_2^T\mathbf{x}_1 & \mathbf{x}_2^T\mathbf{x}_2 & \mathbf{x}_2^T\mathbf{x}_3 & \dots & \mathbf{x}_2^T\mathbf{x}_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n^T\mathbf{x}_1 & \mathbf{x}_n^T\mathbf{x}_2 & \mathbf{x}_n^T\mathbf{x}_3 & \dots & \mathbf{x}_n^T\mathbf{x}_n \end{bmatrix} = X^TX$, where $X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$

if we use ϕ for data mapping, then

$K = \begin{bmatrix} \phi(\mathbf{x}_1^T)\phi(\mathbf{x}_1) & \phi(\mathbf{x}_1^T)\phi(\mathbf{x}_2) & \dots & \phi(\mathbf{x}_1^T)\phi(\mathbf{x}_n) \\ \phi(\mathbf{x}_2^T)\phi(\mathbf{x}_1) & \phi(\mathbf{x}_2^T)\phi(\mathbf{x}_2) & \dots & \mathbf{x}_2^T\mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_1) & \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_2) & \dots & \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_n) \end{bmatrix} = \phi(X^T)\phi(X)$ But, we need to know ϕ. Instead of mapping via ϕ, we can do it in one operation, leaving the mapping completely implicit. In fact, we do not even need to know ϕ, all we need to know is how to define a kernel K, which specifies how to compute the similarity between different data points. We call K the kernel matrix because it contains the value of the kernel for every pair of data points, K also is called Gram matrix.

We can define kernel matrix, but we need to satisfy the positive semidefinite matrix, which is from Mercer's Theorem. Mercer's theorem is generalized from linear algebra which associates an inner product to any positive-definite matrix. A real-valued function $K(\mathbf{x}_i, \mathbf{x}_j)$ is said to satisfy Mercer's condition if for all square inte-

grable functions $g(\mathbf{x})$, it has

$$\int\int g(\mathbf{x}_i)K(\mathbf{x}_i,\mathbf{x}_j)g(\mathbf{x}_j)^T d_{x_i} d_{x_j} \geq 0 \tag{3.46}$$

Analogously, for the desecrate case, if K is square matrix of dimension n, and $\forall g$, it has

$$\sum_{i=1}^{n}\sum_{j=1}^{n} g(\mathbf{x}_i)K(\mathbf{x}_i,\mathbf{x}_j)g(\mathbf{x}_j)^T \geq 0 \tag{3.47}$$

If Eq. 3.47 holds for any choice g, then K satisfies Mercer's condition.

(1) A symmetric function $K(x_i, x_j)$ can be expressed as an inner product $K(\mathbf{x}_i,\mathbf{x}_j) = \langle\phi(\mathbf{x}_i),\phi(\mathbf{x}_j)\rangle$ for some ϕ, if and only if $K(\mathbf{x}_i,\mathbf{x}_j)$ is positive semidefinite, i.e. for any mapping $\forall g$.

proof:

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{j=1}^{n} g(\mathbf{x}_i)K(\mathbf{x}_i,\mathbf{x}_j)g(\mathbf{x}_j)^T = \sum_{i=1}^{n}\sum_{j=1}^{n} g(\mathbf{x}_i)\phi(\mathbf{x}_i)\phi(\mathbf{x}_j)^T g(\mathbf{x}_j)^T \\ =&\sum_{i=1}^{n}\sum_{j=1}^{n} g(\mathbf{x}_i)\phi(\mathbf{x}_i)\big(g(\mathbf{x}_j)\phi(\mathbf{x}_j)\big)^T \geq 0\end{aligned} \tag{3.48}$$

(2) A positive definite matrix is a symmetric matrix K, for which all eigenvalues are positive. If $K = K^T$, then we can write

$$K = Q\Lambda Q^T = Q\begin{bmatrix}\lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n\end{bmatrix}Q^T \tag{3.49}$$

proof: if Λ in Eq. 3.49 has non-negative eigenvalues, then we can decompose it $\Lambda = LL^T$. Further, we can rewrite Eq. 3.49 as

$$K = Q\Lambda Q^T = QLL^TQ^T = QL(QL)^T = AA^T \tag{3.50}$$

Then, we can infer that K is semi positive definite, and satisfy Mercer's condition above. In practice, the choice of K that does not satisfy Mercer's condition may still

perform reasonably if K at least defines some kind of similarity between points.

In the following, we will discuss about the learning and inference in the dual form for the kernel SVM.

Learning: given the kernel K matrix, we need to solve the objective function in Eq. 3.45. In this section, we will introduce Newton method (coordinate descent [8]) to optimize it. More specifically, we want to solve the i-th coordinate given all other coordinates. Suppose we have the current solution $\boldsymbol{\alpha}$, and we want to find the optimal update d for the i-th coordinate, which can be stated as the one-variable sub-problem

$$\max_{d} \hat{\mathcal{L}}(\boldsymbol{\alpha} + d\mathbf{e}_i), \ s.t. \ 0 \leq \alpha_i + d\mathbf{e}_i \leq C \tag{3.51}$$

where $\mathbf{e}_i = [0, ..., 1, ...0]^T$. Note that $\hat{\mathcal{L}}(\boldsymbol{\alpha})$ in Eq. 3.45 is differentiable w.r.t. $\boldsymbol{\alpha}$. Take the derivative w.r.t. $\boldsymbol{\alpha}$, then we get the first order gradient

$$\begin{aligned} &\nabla\hat{\mathcal{L}}(\boldsymbol{\alpha}) = \mathbf{1} - H\boldsymbol{\alpha} \\ \Longrightarrow &\nabla_i\hat{\mathcal{L}}(\boldsymbol{\alpha}) = 1 - (H\boldsymbol{\alpha})_i = 1 - \sum_{j=1}^{N} H_{ij}\alpha_j \end{aligned} \tag{3.52}$$

And further, we can yield its second order gradient

$$\nabla^2\hat{\mathcal{L}}(\boldsymbol{\alpha}) = -H \tag{3.53}$$

Then, we can get its Taylor expansion of Eq. 3.51 as follows:

$$\hat{\mathcal{L}}(\boldsymbol{\alpha} + d\mathbf{e}_i) = \frac{d^2}{2}H + d\nabla_i\hat{\mathcal{L}}(\boldsymbol{\alpha}) + const. \tag{3.54}$$

where $\nabla_i\hat{\mathcal{L}}$ is the i-th component of the gradient $\nabla\hat{\mathcal{L}}$, see further. It is easy to see that when $d = 0$, the objective function in Eq. 3.54 achieves its optimum. Take the gradient w.r.t. d in Eq. 3.54, and set it to zero

$$\begin{aligned} &\frac{\partial\hat{\mathcal{L}}(\boldsymbol{\alpha} + d\mathbf{e}_i)}{\partial d} = 0 \\ \Longrightarrow \ &dH + \nabla_i\hat{\mathcal{L}}(\boldsymbol{\alpha}) = 0 \\ \Longrightarrow \ &d = -\frac{\nabla_i\hat{\mathcal{L}}(\boldsymbol{\alpha})}{H} \end{aligned} \tag{3.55}$$

Apparently, Eq. 3.55 is exactly from Newton method for quadratic programming. If we assume the linear SVM case, then we can induce further for $\nabla_i \hat{\mathcal{L}}(\boldsymbol{\alpha})$

$$\begin{aligned}\nabla_i \hat{\mathcal{L}}(\boldsymbol{\alpha}) &= 1 - \sum_{j=1}^{N} H_{ij}\alpha_j = 1 - \sum_{j=1}^{N} y_i \mathbf{x}_i^T y_j \mathbf{x}_j \alpha_j \\ &= 1 - y_i \mathbf{x}_i^T \sum_{j=1}^{N} (y_j \mathbf{x}_j \alpha_j) \qquad (3.56) \\ &= 1 - y_i \mathbf{x}_i^T \mathbf{w} \qquad (3.57)\end{aligned}$$

where we use $\mathbf{w} = \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j$. According to Newton method, we can update α_i

$$\alpha_i = \min(\max(\alpha_i - d, 0), C) \qquad (3.58)$$

where d is from Eq. 3.55. Of course, we also need to update $\mathbf{w}$, which can will be used to update d,

$$\mathbf{w} = \mathbf{w}_{old} + (\alpha_i - \alpha_{old}) y_i x_i \qquad (3.59)$$

Note that if we have the constraint in Eq. 3.41, we can only need to update α_i for $i = \{1, ..., N-1\}$ and α_N can be derived from constraint in Eq. 3.41.

Prediction: for the linear case, we have

$$y = \mathbf{x}^T \mathbf{w} = \mathbf{x}^T \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = \sum_i \alpha_i y_i \mathbf{x}^T \mathbf{x}_i \qquad (3.60)$$

And for any kernel mapping (assuming $\phi(\mathbf{x})$, which does not need to know explicitly), then we have

$$y = \phi(\mathbf{x})^T \mathbf{w} = \phi(\mathbf{x})^T \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i) = \sum_i \alpha_i y_i \phi(\mathbf{x})^T \phi(\mathbf{x}_i)) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) \qquad (3.61)$$

For more details for quadratic programming, please refer to Appendix A and [**?**].

3.4 Passive aggressive algorithm

We have talked about Perceptron training and SVM. The advantage of perceptron training is that it is an online learning algorithm, which can effectively handle large scale dataset. SVM extends perceptron learning with maximum margin, feature extension and kernel trick to make it powerful for classification tasks. In this part, we will take about online learning algorithm under maximum margin framework, namely Passive aggressive (PA), which also extends the perceptron with large margin techniques [9].

Similar to perceptron training, PA algorithm observes instances in a sequential manner. Given the current observation, the algorithm infers an outcome. This outcome can be as simple as a yes/no (+/-) decision, as in the case of binary classification problems, and as complex as a string over a large alphabet. Once the algorithm has made a prediction, it will compare it with the desired outcome. If the prediction did not match the grouthtruth, the online algorithm suffers an instantaneous loss and then modify its prediction mechanism (update the model), presumably improving the chances of making an accurate prediction on subsequent rounds. In the following parts, we will introduce linear classifier, multi-label (or ranking SVM) and deep ranking SVM.

3.4.1 Linear classifier with maximum margin

Binary classifier: Assume that we have instance-label pairs $(\mathbf{x}_i, y_i)$ with binary labels, the passive aggressive algorithm learn a mapping $f : \mathbb{R}^D \rightarrow \{-1, 1\}$

$$f(\mathbf{w}; (\mathbf{x}_i, y_i)) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x}_i \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{3.62}$$

the magnitude $|\mathbf{w}^T x|$ can be interpreted as the degree of confidence in this prediction. Note that we have discussed the correlation (or inner product) in Eq. 2.5, which measures how the instance $\mathbf{x}$ matches the model $\mathbf{w}$. Compared to Perceptron training, the Passive aggressive considers to achieve a margin of at least 1 as much as possible, in order to predict with high confidence. Similarly, it uses the following hinge-loss function as SVM

$$\ell\big(\mathbf{w};(\mathbf{x},y)\big) = \begin{cases} 0 & \text{if } y(\mathbf{w}^T\mathbf{x}) \geq 1 \\ 1 - y(\mathbf{w}^T\mathbf{x}) & \text{otherwise} \end{cases} \tag{3.63}$$

For the binary case, the Passive aggressive optimize the following problem:

$$\mathbf{w}_{t+1} = \text{argmin}\, ||\mathbf{w} - \mathbf{w}_t||^2 \quad s.t.\ \ell\big(\mathbf{w};(\mathbf{x},y)\big) = 0 \tag{3.64}$$

Geometrically, we want $\mathbf{w}_{t+1}$ to be set as the projection of $\mathbf{w}_t$ on the half-space of vectors which attain the hinge-loss $\ell\big(\mathbf{w};(\mathbf{x},y)\big) = 0$ on the current example. The resulting algorithm is passive whenever the hinge-loss is zero, that is, $\mathbf{w}_{t+1} = \mathbf{w}_t$ whenever $\ell\big(\mathbf{w}_t;(\mathbf{x}_t,y_t)\big) = 0$. In contrast, on those rounds where the loss is positive, the algorithm aggressively forces $\mathbf{w}_{t+1}$ to satisfy the constraint $\ell\big(\mathbf{w}_{t+1};(\mathbf{x}_t,y_t)\big) = 0$ regardless of the step-size required. This is the reason why the algorithm is named Passive-Aggressive or PA for short in [9].

To incorporate the constraints into the objective, we use Lagrange multipliers

$$\mathbf{w}_{t+1} = \text{argmin}\, ||\mathbf{w} - \mathbf{w}_t||^2 - \alpha_i \ell(\mathbf{w};(\mathbf{x}_i,y_i)) \tag{3.65}$$

where $\alpha_i \geq 0$. According to KKT conditions, we can get the following equations (compute the gradient w.r.t. $\mathbf{w}$, and set it to zero)

$$\mathbf{w} - \mathbf{w}_t + \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \mathbf{w}_t - \alpha_i y_i \mathbf{x}_i \tag{3.66}$$

Plugging the above back into Eq. 3.65 we get

$$\begin{aligned} \mathcal{L}(\alpha_i) &= \frac{1}{2}{\alpha_i}^2||\mathbf{x}_i||^2 - \alpha_i(1 - y_i\mathbf{w}^T\mathbf{x}_i) \\ &\implies \frac{1}{2}{\alpha_i}^2||\mathbf{x}_i||^2 - \alpha_i\big(1 - y_i({\mathbf{w}_t}^T - \alpha_i y_i \mathbf{x}_i)\mathbf{x}_i\big) \\ &\implies \frac{1}{2}{\alpha_i}^2||\mathbf{x}_i||^2 - \alpha_i(1 - y_i{\mathbf{w}_t}^T\mathbf{x}_i) - {\alpha_i}^2||\mathbf{x}_i||^2 \\ &\implies -\frac{1}{2}{\alpha_i}^2||\mathbf{x}_i||^2 - \alpha_i(1 - y_i{\mathbf{w}_t}^T\mathbf{x}_i) \end{aligned} \tag{3.67}$$

Taking the derivative of $\mathcal{L}(\alpha_i)$ w.r.t. α_i and setting it to zero, we get

$$\begin{aligned}\frac{\partial \mathcal{L}(\alpha_i)}{\partial \alpha_i} = 0 \implies & -\alpha_i ||\mathbf{x}_i||^2 - (1 - y_i \mathbf{w}_t{}^T x_i) = 0 \\ \implies & \alpha_i = -\frac{1 - y_i \mathbf{w}_t{}^T \mathbf{x}_i}{||\mathbf{x}_i||^2}\end{aligned} \tag{3.68}$$

Finally, put Eq. 3.68 into Eq. 3.67, we get the updating equation for $\mathbf{w}$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_i y_i \mathbf{x}_i \quad \text{where } \alpha_i = \frac{\ell(\mathbf{w}; (\mathbf{x}_i, y_i))}{||\mathbf{x}_i{}^2||} \tag{3.69}$$

As mentioned, the objective function above is just for binary classifier. In the following, we consider to handle the multiclass case.

Linear Multiclass: it is easy to extend the binary case into multiclass situation. We define $\mathbf{\Theta} = [\mathbf{w}_1, ..., \mathbf{w}_K]$ a parameter vector by concatenating all the parameters $\{\mathbf{w}_k\}_{k=1}^K$ (that means $\mathbf{\Theta}^{y_t}$ is y_t-th block in $\mathbf{\Theta}$, or says $\mathbf{\Theta}^{y_t} = \mathbf{w}_{y_t}$), and $\mathbf{\Phi}(\mathbf{x}_t, y_t)$ is a feature vector relating input $\mathbf{x}_t$ and output y_t, which is composed of K blocks, and all blocks but the y_t-th are set to be the zero vector while the y_t-th block is set to be $\mathbf{x}_t$. We denote by $\mathbf{\Theta}_t$ the weight vector used by the algorithm on round t, and predict the label for $\mathbf{x}$ by

$$\hat{y} = \arg\max_{y \in [1,K]} \mathbf{\Theta} \cdot \mathbf{\Phi}(\mathbf{x}, y) \tag{3.70}$$

and refer to the term

$$\gamma(\mathbf{\Theta}_t; (\mathbf{x}_t, y_t)) = \mathbf{\Theta}_t \cdot \mathbf{\Phi}(\mathbf{x}_t, y_t) - \mathbf{\Theta}_t \cdot \mathbf{\Phi}(\mathbf{x}_t, \hat{y}_t) \tag{3.71}$$

as the (signed) margin attained on round t. In other words, we hope for any instance $\mathbf{x}_t$, its score of the truth label y_t should be larger than the runner up $\hat{z}_t$ with margin at least 1, where $\hat{z}_t = \arg\max_{z_t \in [1,K], z_t \neq y_t} \mathbf{\Theta} \cdot \mathbf{\Phi}(\mathbf{x}_t, y_t)$. Similarly, we use the hinge-loss function, which is defined by the following,

$$\ell(\mathbf{\Theta}; (\mathbf{x}_t, y_t)) = \begin{cases} 0 & \text{if } \gamma(\mathbf{\Theta}_t; (\mathbf{x}_t, y_t)) \geq 1 \\ 1 - \gamma(\mathbf{\Theta}_t; (\mathbf{x}_t, y_t)) & \text{otherwise} \end{cases} \tag{3.72}$$

where $\gamma(\mathbf{\Theta}_t; (\mathbf{x}_t, y_t)) \geq 1$ indicates that the correct class for each instance $\mathbf{x}_t$ has a score

higher than the incorrect classes (i.e. the runner up) by a fixed margin 1 according to Eqs. 3.71 and 3.72.

Thus we can optimize the objective function:

$$\begin{aligned} \mathbf{\Theta}_{t+1} &= \arg\min_{\mathbf{\Theta}} \frac{1}{2}||\mathbf{\Theta} - \mathbf{\Theta}_t||^2 + C\xi \\ &s.t.\ \ell(\mathbf{\Theta}; (\mathbf{x}_t, y_t)) \leq \xi \end{aligned} \tag{3.73}$$

where the l_2 norm of $\mathbf{\Theta}$ on the right hand size can be thought as Gaussian prior. If there's loss, then the updates of PA-1 has the following closed form

$$\begin{aligned} \mathbf{\Theta}_{t+1}^{y_t} &= \mathbf{\Theta}_t^{y_t} + \tau_t \mathbf{x}_t, \\ \mathbf{\Theta}_{t+1}^{\hat{y}_t} &= \mathbf{\Theta}_t^{\hat{y}_t} - \tau_t \mathbf{x}_t, \end{aligned} \tag{3.74}$$

where $\hat{y}_t = \arg\max_{y \in [1,K]} \mathbf{\Theta}_t \cdot \mathbf{\Phi}(\mathbf{x}_t, y_t)$ and $\tau_t = \min\{C, \frac{\ell(\mathbf{\Theta}_t; (\mathbf{x}_t, y_t))}{||\mathbf{x}_t||^2}\}$. If the prediction $\hat{y}_t$ mismatches the groundtruth y_t, then it indicates that $\mathbf{x}_t$ is misclassified. Given the label (the ground truth assignment) for $\mathbf{x}_t$, we update our parameter $\mathbf{\Theta}$ using the above Eq. 3.74. Thus, our parameter updating takes almost the similar formula as perceptron training, except the threshold constant C, which controls the updating step size. For convergence analysis and time complexity, refer to [9].

3.4.2 Ranking SVM

Suppose we have n training pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ belonging to the total K classes. For each training data, we have a feature vector $\mathbf{x}_i \in \mathbb{R}^D$ and a corresponding label vector $\mathbf{y}_i \in \{-1, 1\}^K$, with each $\mathbf{y}_i^j$ encoded into the binary case, namely $\mathbf{y}_i^j \in \{-1, 1\}$. For any $j \in [1, K]$, $y_i^j = 1$ will denote that $\mathbf{x}_i$ belongs to the j-th class, whereas $y_i^j = -1$ indicates the j-th label is "off" or say $\mathbf{x}_i$ does not belong to the j-th class. Assume we have the score function $f : \mathbb{R}^D \to \mathbb{R}^K$ to make prediction for K classes. If we use linear SVM with weight $\mathbf{w} \in \mathbb{R}^{D \times K}$, then for each $\mathbf{x}_i$, we have

$$f(\mathbf{x}_i; \mathbf{w}) = \mathbf{w}^T \mathbf{x}_i + \mathbf{b} \tag{3.75}$$

where $\mathbf{w}$ is the weight matrix, and $\mathbf{b}$ is the bias. Before we talk about our model, we will first introduce Ranking SVM, and then we will discuss our model which can

effectively incorporate context information in our new ranking function.

3.4.2.1 Linear ranking SVM

The basic idea of Ranking SVM [9, 10] is to rank every relevant label above every irrelevant label. According to Eq. 3.70, the r-th class score can be expressed as $f^r(\mathbf{x}_i) = \mathbf{w}(.,r)^T\mathbf{x}_i + \mathbf{b}(r)$, which is a linear function. Assuming further $\mathbf{Y}_i^+ = \{j|\mathbf{y}_i^j = 1\}$ denotes the set of positive labels for each instance $\mathbf{x}_i$, and $\mathbf{Y}_i^- = \{j|\mathbf{y}_i^j = -1\}$ denotes the set of negative labels for each instance $\mathbf{x}_i$. The Ranking SVM minimizes the following objective function:

$$\begin{aligned} &\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i,r,s}\xi_{i,r,s} \\ &s.t.\ f^r(\mathbf{x}_i) - f^s(\mathbf{x}_i) > 1 - \xi_{i,r,s}, \\ &\forall\xi_{i,r,s} \geq 0 \\ &\forall r \in \mathbf{Y}_i^+, \forall s \in \mathbf{Y}_i^- \end{aligned} \tag{3.76}$$

where C is the constant to balance the weight between the regularization term and the loss term. Basically, for a pair of labels $(r,s) \in [1,K]$, if $r \in \mathbf{Y}_i^+$ and $s \in \mathbf{Y}_i^-$, then the score $f^r(x_i)$ is greater by at least 1 than the score $f^s(x_j)$ with soft margin, and then we say that label r is ranked higher than label s. Although Ranking SVM can be extended with kernel functions [11], it does not scale well, and it also cannot incorporate the context information to help the label prediction. In the following part, we will introduce our deep ranking SVM.

3.4.2.2 Deep ranking SVM

Non-linear mapping: instead of learning a linear mapping in Ranking SVM, we are interested in a non-linear mapping function. To make it easy to understand, suppose we have learned a nonlinear mapping function $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$. For each instance $\mathbf{x} \in \mathcal{X}$, we can get its prediction $f(\mathbf{x})$ using deep learning with several layers of non-linear mapping. The output of $f(\cdot)$ is a scoring function of the instance $\mathbf{x}$, that produces a vector of activations to specify which class is "on" or "off".

Suppose the non-linear mapping function f is defined with L-layers neural net-

work, s.t.

$$f(\mathbf{x}_i;\boldsymbol{\theta}) = \underbrace{f_L \circ f_{L-1} \circ \cdots \circ f_1}_{L \text{ times}}(\mathbf{x}_i) \tag{3.77}$$

where $\boldsymbol{\theta} = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_L\}$, and $\circ$ indicates the function composition, and f_l is the sigmoid function with the weight parameter $\mathbf{w}_l$ respectively for each layer $l = \{1, .., L-1\}$ and the top layer is linear mapping with weight $\mathbf{w}_L$. For more details to learn $\boldsymbol{\theta}$, refer to Sec. 5.5.2. With a little abuse of symbols, for any input $\mathbf{x}$, we denote the output of the l-th layer as $f_{1\to l}(\mathbf{x})$. Then for each layer in the neural network, we have the following definition:

$$f_l(\mathbf{x}) = \begin{cases} \sigma\big(\mathbf{w}_l^T f_{1\to(l-1)}(\mathbf{x})\big) & \text{if } l \in \{2, ..., L-1\} \\ \mathbf{w}_L^T[f_{1\to(L-1)}(\mathbf{x})] + \mathbf{b} & \text{if } l = L \end{cases} \tag{3.78}$$

where $\sigma(x) = 1/(1+exp(-x))$ is sigmoid function. Note that other non-linear functions, such as rectified linear unit (ReLU), can be used to model each layer too. Thus, the output prediction in Eq. 3.77 can be shown as

$$\hat{f}(\mathbf{x}_i;\boldsymbol{\theta}) = \mathbf{w}_L^T[f_{1\to(L-1)}(\mathbf{x}_i)] + \mathbf{b}, \tag{3.79}$$

where $f_{1\to(L-1)}$ indicates $L-1$ layers's of nonlinear mapping, and the top layer can be thought as a linear SVM.

Objective function: given the posterior mapping score $\hat{f}(\mathbf{x};\boldsymbol{\theta})$, we propose the following deep ranking SVM:

$$\mathcal{L}(D;\boldsymbol{\theta}) = \sum_i \ell(\boldsymbol{\theta};\mathbf{x}_i,\mathbf{y}_i) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \tag{3.80}$$

where the hinge loss $\ell(\boldsymbol{\theta},\mathbf{A};\mathbf{x}_i,\mathbf{y}_i)$ is defined as

$$\ell(\boldsymbol{\theta};\mathbf{x}_i,\mathbf{y}_i) = \max\big(1 + \max_{s\in\mathbf{Y}_i^-} \hat{f}^s(\mathbf{x}_i) - \min_{r\in\mathbf{Y}_i^+} \hat{f}^r(\mathbf{x}_i), 0\big) \tag{3.81}$$

where $\hat{f}(\mathbf{x}_i)$ is determined via Eq. 3.79, where we ignore the parameters $\boldsymbol{\theta}$ for clarity. Compared to Eq. 3.70, the scoring function here considers the non-linear mapping which will learn better discriminative features.

To learn the model parameters $\boldsymbol{\theta}$, we can use backpropagation, which will be introduced in Chapter 7.

3.5 Regression analysis

Given training set consists of N pairs $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$, where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$, the task of regression is to fit a real valued function $f : \mathbb{R}^d \to \mathbb{R}$. Compared to classification, the regression finds a mapping from a finite dimensional space to a real value (continuous space), while classification maps to a discrete space. As mentioned in Sec 3.1, especially Eqs. 3.1 and 3.2, we need the following conditions:

- assume the mapping function formula: linear or nonlinear
- define a loss function ℓ
- using the loss function define the empirical risk $R_{emp}(\mathbf{w})$
- solve the parameters by minimizing the empirical risk $R_{emp}(\mathbf{w})$

In the following parts, we will assume linear function to do regression analysis, and introduce optimization methods to estimate model parameters.

3.5.1 linear regression

Revisit the least square objective function for linear regression: given the training data $D = \{x_i, y_i\}_{i=1}^N$, where $x \in \mathbb{R}$, and we can minimize the following objective function

$$\mathcal{L}(\alpha, \beta) = \min_{\alpha,\beta} \sum_{i=1}^{N} (y_i - (\alpha + \beta x_i))^2 \tag{3.82}$$

According to KKT conditions, we can get

$$\frac{\partial \mathcal{L}(\alpha, \beta)}{\partial \alpha} = 0 \implies \sum_{i=1}^{N} (y_i - (\alpha + \beta x_i)) = 0 \tag{3.83}$$

$$\frac{\partial \mathcal{L}(\alpha, \beta)}{\partial \beta} = 0 \implies \sum_{i=1}^{N} (y_i - (\alpha + \beta x_i)) x_i = 0 \tag{3.84}$$

Further, we can have

$$
\begin{aligned}
N\alpha + \beta \sum_{i=1}^{N} x_i &= \sum_{i=1}^{N} y_i \\
\alpha \sum_{i=1}^{N} x_i = \beta \sum_{i=1}^{N} x_i^2 &= \sum_{i=1}^{N} x_i y_i
\end{aligned} \tag{3.85}
$$

Solve the above linear equations, and we get estimations below

$$
\begin{aligned}
\hat{\alpha} &= \bar{y} - \hat{\beta}\bar{x} \\
\hat{\beta} &= \frac{\sum_i x_i y_i - N\bar{x}\bar{y}}{\sum_{i=1}^{N} x_i^2 - N\bar{x}^2}
\end{aligned}
$$

where $\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$, and $\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N}$. And then we predict $\hat{y} = \hat{\alpha} + \hat{\beta}x$.

Before we introduce the variance decomposition, we first talk about some properties related to linear regression. In particular, the residual is defined to be $y_i - \hat{y}_i$, which is the distance from the original data point to the predicted value. The properties related to the learnt model $\hat{\alpha}$ and $\hat{\beta}$ are listed below:

1. the sum of the residuals is zero
2. the sum of observed y_i equals to the sum of predicted value $\hat{y}_i$
3. the sum of the weighted residuals is zero, when i-th residual is weighed by x_i

Property 1: Define i-th residual as $e_i = y_i - \hat{y}_i$, then the sum of the residual is zero.
Proof:

$$
\begin{aligned}
&\sum_{i=1}^{N} e_i = \sum_{i=1}^{N} (y_i - \hat{\alpha} - \hat{\beta}x_i) = \sum_i y_i - \sum_i (\hat{\alpha} + \hat{\beta}x_i) = 0 \\
&\implies \sum_{i=1}^{N} (y_i - \hat{y}_i) = 0 \implies \sum_i y_i = \sum_i \hat{y}_i
\end{aligned}
$$

where the above inference is from Eq. 3.83. And the above equation also proves the property 2.

Further we can prove the Property 3:

$$\begin{aligned} &\sum_i x_i e_i = \sum_i x_i (y_i - \hat{\alpha} - \hat{\beta} x_i) = 0 \\ &\implies \sum_i x_i (y_i - \hat{y}_i) = 0 \end{aligned} \tag{3.86}$$

where the above inference is from Eq. 3.84.

Now consider to decompose the total sum of difference $\sum_{i=1}^{N} (y_i - \bar{y})^2$,

$$\begin{aligned} y_i - \bar{y} = &\underbrace{(\hat{y}_i - \bar{y})}_{\text{explained by the model}} + \underbrace{y_i - \hat{y}_i}_{\text{difference not explained}} \\ \implies &\sum_i \underbrace{(y_i - \bar{y})^2}_{\text{SST}} = \sum_i \underbrace{(\hat{y}_i - \bar{y})^2}_{\text{SSM}} + \sum_i \underbrace{(y_i - \hat{y}_i)^2}_{\text{SSE}} \end{aligned} \tag{3.87}$$

where "SST" stands for "total sum of squares", "SSM": sum of squares associated to model, "SSE": sum of squares associated to error. These terms are often abbreviated as SST, SSM and SSE respectively.

To prove Eq. 3.87, we have

$$\begin{aligned} \sum_i (y_i - \bar{y})^2 &= \sum_i (\hat{y}_i - \bar{y} + y_i - \hat{y}_i)^2 \\ &= \sum_i (\hat{y}_i - \bar{y})^2 + \sum_i (y_i - \hat{y}_i)^2 + 2 \sum_i (\hat{y}_i - \bar{y})(y_i - \hat{y}_i) \end{aligned} \tag{3.88}$$

Thus, we need to prove $\sum_i (\hat{y}_i - \bar{y})(y_i - \hat{y}_i) = 0$. Note that

$$\sum_i (\hat{y}_i - \bar{y})(y_i - \hat{y}_i) = \sum_i \hat{y}_i (y_i - \hat{y}_i) - \sum_i \bar{y} (y_i - \hat{y}_i) \tag{3.89}$$

$$= \sum_i \hat{y}_i (y_i - \hat{y}_i) - \bar{y} \sum_i (y_i - \hat{y}_i) \tag{3.90}$$

$$= \sum_i (\hat{\alpha} + \hat{\beta} x_i) e_i = \hat{\alpha} \sum_i e_i + \hat{\beta} \sum_i x_i e_i = 0 \tag{3.91}$$

where we extract the common factor $\bar{y}$ from Eq. 3.89 to Eq. 3.90. In Eq. 3.90, we have $\sum_i (y_i - \hat{y}_i) = 0$, which is from Property 2 of linear regression. And Eq. 3.91 is set to zero from Property 3.

If our model is doing a good job, then it should explain most of the difference from

y_i, and the first term should be bigger than the second term in Eq. 3.87. If the second term is much bigger, then the model is probably not as useful. If we divide through by SST, we obtain

$$\begin{aligned} 1 &= \frac{SSM}{SST} + \frac{SSE}{SST} \\ \implies \frac{SSM}{SST} &= 1 - \frac{SSE}{SST} \end{aligned}$$

$\frac{SSM}{SST} \in [0,1]$ is the fraction of variability in the data that is explained by the model, and we can use to valuate a model's performance. $\frac{SSM}{SST} = 0$ indicates no variance of y is explained by x, and 1 means that all variance of y is explained well by our model. More specifically, larger $\frac{SSM}{SST}$, more tightly our observations lines around the regression line.

general linear regression with multiple factors Suppose we use Euclidean distance (or minimize least square) and have the ℓ_2 regularization, then

$$\mathcal{L}(x, y; \mathbf{w}) = \min_{\mathbf{w}} \sum_i (y_i - \mathbf{w}^T x_i)^2 + \frac{1}{2}\lambda \mathbf{w}^2 \tag{3.92}$$

take the derivative w.r.t. $\mathbf{w}$ and set it to zero, we can get the following close form solution:

$$\mathbf{w} = (X^T X + \lambda)^{-1} X^T Y \tag{3.93}$$

where $X = [x_1, x_2, ..x_N]$ and $Y = [y_1, ..., y_N]$

3.5.2 Logistic regression

Given the training pair $(\mathbf{x}_i, y_i)$ for classes problem, y_i= 1 or 0, we can estimate model parameters with maximum likelihood,

$$\mathcal{L}(\mathbf{w}, b) = p(Y|X; \mathbf{w}, b) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i) = \prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i} (1 - p(\mathbf{x}_i))^{1-y_i} \tag{3.94}$$

where $p(\mathbf{x}_i)$ is from the following likelihood:

$$p(\mathbf{x}_i|\mathbf{w}, b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}_i + b)}} \tag{3.95}$$

We can take the log to $\mathcal{L}$, and we can get the following log-likelihood as

$$\begin{aligned}
\log\mathcal{L}(\mathbf{w}, b) &= log\prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i}(1 - p(\mathbf{x}_i))^{(1-y_i)} \\
&= \sum_{i=1}^{n} y_i log p(\mathbf{x}_i) + (1 - y_i) log(1 - p(\mathbf{x}_i)) \\
&= \sum_{i=1}^{n} y_i log p(\mathbf{x}_i) + log(1 - p(\mathbf{x}_i)) - y_i log(1 - p(\mathbf{x}_i)) \\
&= \sum_{i=1}^{n} y_i log \frac{p(\mathbf{x}_i)}{1 - p(\mathbf{x}_i)} + log(1 - p(\mathbf{x}_i)) \\
&= \sum_{i=1}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b) + log(1 + e^{\mathbf{w}^T\mathbf{x}_i + b})
\end{aligned} \tag{3.96}$$

To find the maximum likelihood estimation, we can minimize the negative log likelihood

$$\mathbf{w}, b = \underset{\mathbf{w},b}{\operatorname{argmin}} -\log\mathcal{L}(\mathbf{w}, b) \tag{3.97}$$

Take the derivative w.r.t. $\mathbf{w}$ and set it to zero

$$\begin{aligned}
\frac{\partial - \log\mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}} &= -\Big[- \sum_{i=1}^{n} \frac{e^{\mathbf{w}^T\mathbf{x}_i + b}}{1 + e^{\mathbf{w}^T\mathbf{x}_i + b}}\mathbf{x} + \sum_{i=1}^{N} y_i\mathbf{x}_i \Big] \\
&= -\sum_{i=1}^{N}(y_i - p_i(\mathbf{x}_i|w))\mathbf{x}_i
\end{aligned} \tag{3.98}$$

There is no closed-form solution according to Eq. 3.98 because the gradient w.r.t. $\mathbf{w}$ relies on p_i, which in turn depends on $\mathbf{w}$. Fortunately, we can leverage fix-point method to solve it.

More specifically, we can use Newton's method. Assume that we have a local

minimum $\mathbf{w}^*$, and then we can use Taylor expansion

$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \frac{L'(\mathbf{w}^{(n)})}{L''(\mathbf{w}^{(n)})} \tag{3.99}$$

$$\mathcal{L}(\mathbf{w}, b) = \mathcal{L}(\mathbf{w}^*, b) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^2 \frac{d^2\mathcal{L}}{d\mathbf{w}^2} \tag{3.100}$$

where Eq. 3.99 is from Newton method to update model parameters with Hessian matrix, please refer to Eq. 3.55. Given the optimal solution $\mathbf{w}^*$, the first gradient $\partial\mathcal{L}(\mathbf{w})/\partial\mathbf{w}^* = 0$, so Eq. 3.100 holds too.

Proof: suppose we denote the gradient w.r.t. $\mathbf{w}$ as: $g(\mathbf{w}) = \frac{\partial\mathcal{L}(\mathbf{w})}{\partial\mathbf{w}} = \mathcal{L}'(\mathbf{w})$, then the first order Taylor series approximate $g(\mathbf{w})$ around $\mathbf{w}_k$

$$g(\mathbf{w}) \approx g(\mathbf{w}_k) + H_k(\mathbf{w} - \mathbf{w}_k) \tag{3.101}$$

At the minimum $\mathbf{w}^*$, $g(\mathbf{w}^*) = 0$, then we have

$$\mathbf{w}^* = \mathbf{w}_k - H_k^{-1} g(\mathbf{w}_k) \implies \mathbf{w}_{k+1} = \mathbf{w}_k - H_k^{-1} g(\mathbf{w}_k) \tag{3.102}$$

Learning:
Data: The training dataset $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$, iterations T, convergence threshold ϵ
Result: The model parameters $\mathbf{w}, b$
initialization $\mathbf{w}_0$;
while *not convergence* **do**
  Evaluate $g_k = \nabla f(x_k)$, $H_k = \nabla^2 f(x_k)$;
  Solve $d_k = -H^{-1} g_k$;
  Use line search to find stepsize α_k;
  $x_k = x_k + \alpha_k d_k$;
end

Algorithm 3: Newton's method for minimizing objective function.

So the iterative updating need to compute H_k, which is the Hessian matrix of $\mathcal{L}$ (it is the matrix of second partial derivatives). However, to compute H^{-1} is usually time-consuming, unless it happens to be a diagonal matrix. Note that $g(\mathbf{w}) = \sum_{i=1}^{N}(y_i -$

$p_i(\mathbf{x}_i|w))\mathbf{x}_i$, then we have

$$H = \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1,j=1}^{N} (1 - p_i(\mathbf{x}_i|w))p_j(\mathbf{x}_j|w)\mathbf{x}_i^T\mathbf{x}_j \tag{3.103}$$

$$\implies H = X^T S X \tag{3.104}$$

where $S \stackrel{\text{def}}{=} diag\big((1-p_1)p_1, (1-p_2)p_2, (1-p_N)p_N\big)$. Further we can define $g(\mathbf{w}) = X^T(\mathbf{y} - \mathbf{p})$, where $\mathbf{y} = [y_1, ..., y_N]$ and $\mathbf{p} = [p_1, ..., p_N]$.

According to iterative reweighted least square (IRLS)

$$\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k - H_k^{-1} g(\mathbf{w}_k) \\
&= \mathbf{w}_t + (X^T S_t X)^{-1} X^T(\mathbf{y} - \mathbf{p}_t) \\
&= (X^T S_t X)^{-1}((X^T S_t X)\mathbf{w}_t + X^T(\mathbf{y} - \mathbf{p}_t)) \\
&= (X^T S_t X)^{-1} X^T[S_t X \mathbf{w}_t + \mathbf{y} - \mathbf{p}_t] \\
\mathbf{w}_{t+1} &= (X^T S_t X)^{-1} X^T S_t z_t
\end{aligned} \tag{3.105}$$

where $z_t \stackrel{\text{def}}{=} X\mathbf{w}_t + S^{-1}(\mathbf{y} - \mathbf{p}_t)$

Relationship to SVM , we can compute the following ratio

$$\frac{p(y|x)}{1 - p(y|x)} = e^{(\mathbf{w}^T x + b)} \implies log\frac{p(y|x)}{1 - p(y|x)} = \mathbf{w}^T x + b \tag{3.106}$$

Thus, if $\mathbf{w} > 0$, then increasing $\mathbf{x}$ makes $y = 1$ more likely and decreasing $\mathbf{x}$ make y=0 more likely. if $\mathbf{w} = 0$, $\mathbf{x}$ has no impact on the output. This indicates that it is a linear model. To minimize the misclassification rate, we should predict $y = 1$ when $p \geq 0.5$ and $y = 0$ when $p < 0.5$. Thus by setting $p(y|\mathbf{x}; \mathbf{w}, b) = 1$, we can get

$$\frac{1}{1 + e^{-(\mathbf{w}^T x + b)}} = 0.5 \implies e^{-(\mathbf{w}^T x + b)} = 1 \implies \mathbf{w}^T x + b = 0 \tag{3.107}$$

Thus logistic regression gives us a linear classifier, with the decision boundary $\mathbf{w}^T x + b = 0$. In a sense, logistic regression has the similar linear decision boundary as linear SVM, but with probabilistic prediction.

Because logistic regression predicts probabilities, rather just classes, we can fit it using likelihood. For the binary case, e.g. $y_i = 1$ *or* 0, the prediction probability of x_i

is either p if $y_i = 1$, or $1 - p$ if $y_i = 0$. The likelihood is then

Chapter 4

Unsupervised Learning

Unsupervised learning is to learn a model from one space X to another space Y, or a function $f : X \mapsto Y$. However, compared to supervised learning, we are not given Y as the desired outcome. Thus, unsupervised learning is more challenging because we have no guidance. On the other hand, unsupervised learning is more common in real practice, due to the lack of labels.

In unsupervised learning, we need to unfold the inner distribution or to describe hidden structure from unlabeled data. In general, there are two major subfields: one is to cluster the data into categories (clustering analysis), and the other direction is to learn a mapping, called feature learning. The purpose of feature learning is to discover useful representations, which can improve clustering or classification performance. As for unsupervised feature learning, there are many dimension reduction methods, such as principal component analysis (PCA), independent component analysis (ICA) and dictionary learning.

In this chapter, we will focus on clustering analysis. The purpose of clustering is to group a set of instances (or observations) into clusters, where instances within the same group are similar according to some predesignated criterion (distance function described in Chapter 2), while observations drawn from different clusters are dissimilar and should be far away from each other. Thus, the fundamental problem of clustering is to define similarity metrics, learn useful features and infer the assignment or label for each instance. In practice, we can infer the clustering assignment for each instance, and then update the model parameters in an alternative manner.

Notice that unsupervised clustering is closely related to supervised classification.

As for unsupervised clustering, no label is given and we need to learn a mapping $f : \mathbf{x} \mapsto y$ to infer label for each instance. Apparently, supervised classification does the same thing (e.g. learning a function f), the only difference is that y is given to each instance $\mathbf{x}$. Thus, in unsupervised clustering, we can introduce a latent variable y (label) associated with $\mathbf{x}$ and infer the label of each insurance with maximum likelihood.

In the following parts, we will introduce some widely used clustering methods, such as clustering, Gaussian mixture models (GMM) and spectral clustering. In addition, we will also introduce nonparametric clustering, where the model complexity will grow with data.

4.1 K-means clustering

k-means is a popular clustering algorithm used in many applications including vector quantization (e.g. hidden Markov models and bag of words representation in vision), initialization of more computational expensive algorithms (Gaussian mixture models). The aim of k-means is to group the given observations (e.g. N data instances) into k clusters, where each instance is assigned to its nearest mean. k-means is a difficult problem (NP-hard). However, many heuristic methods have been proposed, which can converge quickly into a local optimum. In this section, we will introduce k-means algorithm.

Given the training data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$, we want to group it into K clusters. More specifically, we need to learn K centers μ_k for $k = [1, K]$ by minimizing the following objective

$$\mathcal{L}(D; \{\mu_k\}_{k=1}^K) = \sum_i \min_{k \in [1,K]} ||x_i - \mu_k||^2 \tag{4.1}$$

The k-means algorithm alternates between label assignment and component parameter updating. More specifically, given the component parameters, it will loop over all instances and assign its clustering label with shortest distance (or highest similarity). For example, when we want to assign a data point with black color in Fig. 4.1 to the given 3 clusters, we will compute the distance between this point to all the 3 cluster centers and decide its assignment to the nearest center. For example

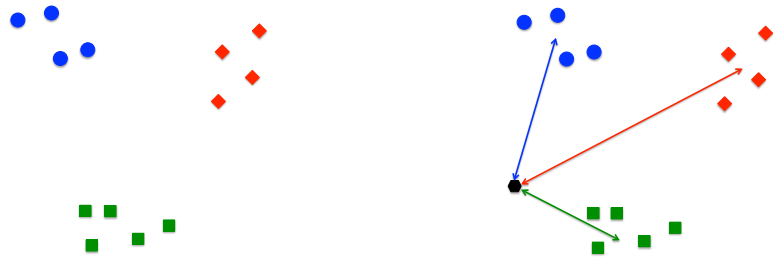


Figure 4.1. (a) the left figure shows a few data points belonging to 3 clusters (red, green and blue respectively); (b) the right shows how to decide the label of the new point (with dark color), given the 3 clusters.

in Fig. 4.1, we can assign it to the green cluster, which the dark point is closest to. Note that we need to loop over all data points and assign them to the nearest cluster center. Then, we can update component parameters in a supervised manner (because all labels are given for all instances). The two stages of k-means clustering are:

Inference (label assignment): for each instance $\mathbf{x}_i$,

$$y_i = \underset{k}{\operatorname{argmin}} ||x_i - \mu_k||^2 \tag{4.2}$$

Learning (component parameter): for each center μ_k:

$$\frac{\partial \mathcal{L}(D; \{\mu_k\}_{k=1}^K)}{\partial \mu_k} = 0 \implies \mu_k = \frac{\sum_i \delta(y_i, k)\mathbf{x}_i}{\sum_i \delta(y_i, k)} \tag{4.3}$$

where δ is a indicator function, which is set to 1 if its two parameters are equal, otherwise 0.

The objective function in Eqs. 4.1 and 4.2 leverages Euclidean distance to measure the similarity between two instances (each observation and the clustering center). In practice, different similarity measures can be used, refer Chapter 2 for more details.

4.1.1 Algorithm

k-means is to group the observations in the data space into K subspaces, where the assignment of each observation in its corresponding subspace relies on similarity. More specifically, for each observation, we need to compute its similarity to all K subspaces

(in practice, cluster center or mean is used), determine its label, and then update the representation of each subspace. The most common k-means algorithm uses an iterative refinement technique with two steps: label assignment and component parameter updating. The detail algorithm is below:

Data: the training set $\mathcal{D}$ and iterations T
Result: output the centers $\{\mu_k\}_{k=1}^{K}$
initialize the clustering centers μ_k, for $k = \{1, 2..., K\}$;
initialize the assignment vector $Y = \{y_1, y_2, ..., y_N\}$;
set iteration index t = 0;
while $t < T$ **do**
    missed = 0 ;
    for *Each* $\mathbf{x}_i \in \mathcal{D}$ **do**
        $Mindist = +\infty$ // the minimum distance ;
        for *Each* $\mathbf{c}_k$ **do**
            calculate the similarity between $\mathbf{x}_i$ and μ_k, denoted as $d_{i,k}$;
            if $Mindist < d_{i,k}$ **then**
                update the minimal distance: $Mindist = d_{i,k}$;
                update its assignment $y_i = k$;
            end
        end
    end
    update the new centers according to $\mathcal{D}$ and Y with the following;
    for *Each* μ_k **do**
        $\mu_k = \frac{\sum_i \delta(y_i == k)\mathbf{x}_i}{\sum_i \delta(y_i == k)}$;
    end
    t++;
end

Algorithm 4: kmeans clustering

time complexity analysis: for each instance, the time complexity to calculate the similarity is $O(Kd)$, so the total time is $O(NKd)$ on the whole dataset.

4.1.2 Convergence analysis

the convergence of k-means has been proved in [12]. The basic idea is below: k-means algorithm assumes there is hidden variable y_i which indicates the assignment of each instance $\mathbf{x}_i$. Then it defines the following cost function

$$Q(\mu', \mu) = \sum_i (\mathbf{x}_i - \mu'_{y_i})^2 \tag{4.4}$$

where y_i is the best assignment for $\mathbf{x}_i$, and μ' are the centers which will be calculated in the next step. As we mentioned before we can find the best ${y_i}'$ via Eq. 4.3. Since ${y_i}'$ is by definition the best assignment of observation $\mathbf{x}_i$ to the centers μ', we have

$$\mathcal{L}(D, \mu') - Q(\mu', \mu) = \sum_i (\mathbf{x}_i - \mu'_{y_i'})^2 - \sum_i (\mathbf{x}_i - \mu'_{y_i})^2 \leq 0 \tag{4.5}$$

$$\begin{aligned} \mathcal{L}(D, \mu') - \mathcal{L}(D, \mu) &= \mathcal{L}(D, \mu') - Q(\mu', \mu) + Q(\mu', \mu) - \mathcal{L}(D, \mu) \\ &= \mathcal{L}(D, \mu') - Q(\mu', \mu) + Q(\mu', \mu) - Q(\mu, \mu) \\ &\leq Q(\mu', \mu) - Q(\mu, \mu) \leq 0 \end{aligned} \tag{4.6}$$

On the one hand the square error $\mathcal{L}$ is positive. On the other hand, the algorithm shown in Eq. 4.6 will decrease its value in each iteration. Thus, the algorithm will converge to a local minimum, where μ never change.

4.2 Gaussian mixture models

Gaussian mixture models (GMM) is another method to partition the data into K clusters. Note that we have discussed k-means, which does the hard label assignment. Compared to k-means, GMM does the soft label assignment. Specifically, for each observation, we compute its probability belonging to K Gaussian components. Thus, GMM is a parametric probability density function represented as a weighted sum of K Gaussian components. Given the training data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$, we model it with Gaussian mixture models. For any instance $x \in \mathcal{D}$, we have the following represen-

tation

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k) \tag{4.7}$$

where π_k is the mixing coefficient (prior distribution), such that $\pi_k \geq 0$ and $\sum_{k=1}^{K} \pi_k = 1$, and $\mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$ is the normal distribution,

$$\mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{m/2}|\Sigma_k|^{1/2}} exp[-\frac{1}{2}(\mathbf{x} - \mu_k)^T \Sigma_k^{-1}(\mathbf{x} - \mu_k)] \tag{4.8}$$

Understand GMM as a generative model: for each instance $i = \{1, 2, ..., N\}$
(1) sample from 1 of K clusters with multinomial distribution $\{\pi_1, ..., \pi_K\}$;
(2) given $k \in [1, K]$, draw $\mathbf{x}_i$ from normal distrubution $\mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$

4.2.1 Parameter learning

If we use GMM to model the training data, the key problem is to estimate all parameters $\{\pi_k, \mu_k, \Sigma_k\}$, for $k = \{1, 2, ..., K\}$. The algorithm of learning GMM parameters is known as the iterative Expectation-Maximization (EM), which is consisted of E-step and M-step:

E-step is to do the soft assignment for the current observation $\mathbf{x}_i$, given the model

$$p(y_i = k|\mathbf{x}_j, \lambda_k) = p(\mathbf{x}_j|\mu_k, \Sigma_k)\pi_k \tag{4.9}$$

where the likelihood $p(\mathbf{x}_j|\mu_k, \Sigma_k)$ is from Gaussian distribution:

$$p(\mathbf{x}_j|\mu_k, \Sigma_k) \propto exp(-\frac{1}{2\Sigma_k}||x_j - \mu||^2) \tag{4.10}$$

While **M-step** is to estimate the model parameters, given the probability assignment.

$$\mu_k = \frac{\sum_{j=1}^{N} p(y_j = k|\mathbf{x}_j; \mu_k, \Sigma_k)\mathbf{x}_j}{\sum_{j=1}^{N} p(y_j = k|\mathbf{x}_j; \mu_k, \Sigma_k)} \tag{4.11}$$

$$\Sigma_k = \frac{\sum_{j=1}^{N} p(y_j = k|\mathbf{x}_j)(\mathbf{x}_j - \mu_k)(\mathbf{x}_j - \mu_k)^T}{\sum_{j=1}^{N} p(y_j = k|\mathbf{x}_j)} \tag{4.12}$$

$$\pi_k = \frac{\sum_{j=1}^{N} p(y_i = k|\mathbf{x}_j, \lambda_k)}{N} \tag{4.13}$$

The outline of algorithm to learn parameters is below:
Data: the training set $\mathcal{D}$ and iterations T
Result: output the centers$\{\pi_{k=1}^{K}\}$, $\{\mathbf{u}_{k=1}^{K}\}$, $\{\boldsymbol{\sigma}_{k=1}^{K}\}$
initialize the clustering centers $\mathbf{u}_k$, for $k = \{1, 2..., K\}$;
initialize the prior probability for each cluster π_k, for $k = \{1, 2..., K\}$;
initialize the assignment vector $Y = \{y_1, y_2, ..., y_N\}$;
set iteration index t = 0;
while $t < T$ **do**
  E-step:
  for *Each* $\mathbf{x}_i \in \mathcal{D}$ **do**
    calculate $p(y_i = k|\mathbf{x}_j, \lambda_k)$ according to Eq. 4.9 ;
  end
  M-step:
  for *Each component* **do**
    update the mean and variance (μ_k, Σ_k) via Eq. 4.12;
    update the prior distribution π_k according to Eq. 4.13 ;
  end
end

Algorithm 5: Gaussian mixture models

4.2.2 Understanding EM

In this part, we will introduce theoretic background of EM algorithm. Given the observed data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{N}$, the EM algorithm to estimate the Gaussian mixture model

by maximizing the likelihood

$$\prod_i p(\mathbf{x}_i;\theta) = \prod_i \sum_{y_i=1}^{K} p(\mathbf{x}_i, y_i;\theta) \tag{4.14}$$

where y_i is the latent variable for each instance and $\theta = \{\pi_k, \mu_k, \Sigma_k\}$, for $k = \{1, 2, ..., K\}$.

Data: the training set $\mathcal{D}$ and iterations T

Result: output K components: θ

set iteration index t = 0;

while $t < T$ **do**

  E-step:

  for *Each* $\mathbf{x}_i \in \mathcal{D}$ **do**

    calculate $q(y_i) = p(y_i|\mathbf{x}_i;\theta)$;

  end

  M-step:

  $\theta = \arg\max_\theta \sum_i \sum_k q(y_i) \log \frac{p(y_i,\mathbf{x}_i;\theta)}{q(y_i)}$;

  if (converged) break;

end

Algorithm 6: EM algorithm

Take the logarithm of Eq. 4.14, we can get the following log likelihood

$$\sum_{i=1}^{N} \log p(\mathbf{x}_i;\theta) = \sum_{i=1}^{N} \log \sum_{y_i=1}^{K} p(\mathbf{x}_i, y_i;\theta) \tag{4.15}$$

$$= \sum_{i=1}^{N} \log \sum_{y_i=1}^{K} q(y_i) \frac{p(\mathbf{x}_i, y_i;\theta)}{q(y_i)} \tag{4.16}$$

$$\geq \sum_{i=1}^{N} \sum_{y_i=1}^{K} q(y_i) \log \frac{p(\mathbf{x}_i, y_i;\theta)}{q(y_i)} \tag{4.17}$$

$$= \sum_{i=1}^{N} \sum_{y_i=1}^{K} q(y_i) \log \frac{p(\mathbf{x}_i|y_i;\theta)p(y_i;\theta)}{q(y_i)} \tag{4.18}$$

Jensen's inequality: for $p_i \geq 0$, and $\sum_{i=1}^{K} p_i = 1$, we have

$$\log \sum_{i=1}^{K} p_i f(x_i) \geq \sum_{i=1}^{K} p_i \log f(x_i)$$

$$\log E(f(x_i)) \geq E(\log f(x_i)) \tag{4.19}$$

where we use Jensen's inequality from Eq. 4.19

To make the equation holds in Eq. 4.17, we requires $\frac{p(\mathbf{x}_i, y_i; \theta)}{q(y_i)} = c$, then

$$q(y_i) \propto p(\mathbf{x}_i, y_i; \theta) \tag{4.20}$$

Note that $\sum_{y_i=1}^{K} q(y_i) = 1$, thus we have

$$q(y_i) = \frac{p(\mathbf{x}_i, y_i; \theta)}{\sum_{y_i} p(\mathbf{x}_i, y_i; \theta)} \tag{4.21}$$

$$= p(y_i|\mathbf{x}_i; \theta) \tag{4.22}$$

Thus, $q(y_i)$ is the posterior $p(y_i|\mathbf{x}_i; \theta)$ given the current parameter θ. Plugging Eq. 4.21 back into Eq. 4.17, we can get

$$\mathcal{L}(\theta) \geq \sum_{i=1}^{N} \sum_{y_i=1}^{K} q(y_i) \log \frac{p(y_i|\mathbf{x}_i; \theta) p(\mathbf{x}_i; \theta)}{q(y_i)} \tag{4.23}$$

$$= \sum_{i=1}^{N} \sum_{y_i=1}^{K} p(y_i|\mathbf{x}_i; \theta) \log \frac{p(y_i|\mathbf{x}_i; \theta) p(\mathbf{x}_i; \theta)}{p(y_i|\mathbf{x}_i; \theta)} \tag{4.24}$$

$$= \sum_{i=1}^{N} \sum_{y_i=1}^{K} p(y_i|\mathbf{x}_i; \theta) \log p(\mathbf{x}_i; \theta) \tag{4.25}$$

$$= \sum_{i=1}^{N} \log p(\mathbf{x}_i; \theta) \tag{4.26}$$

Therefore, as we maximize the objective function, we are also maximizing the log likelihood function. In practice, we need to maximize the low bound in the M-step:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \sum_{y_i=1}^{K} q(y_i) \log \frac{p(\mathbf{x}_i|y_i; \theta) p(y_i; \theta)}{q(y_i)} \tag{4.27}$$

$$= \sum_{i=1}^{N} E_q(\log p(\mathbf{x}_i|y_i; \theta)) + E_q(\log p(y_i; \theta)) - E_q(\log q(y_i)) \tag{4.28}$$

where $p(y_i)$ is the prior distribution $p(y_i) = \pi_i$ and $p(\mathbf{x}_i|y_i; \theta)$ is Gaussian distribu-

tion.

Given the posterior inference from E-step in Eq. 4.21, we can find the parameters which maximize Eq. 4.27

$$
\begin{aligned}
&\mathcal{L}(\theta) = \sum_{i=1}^{N}\sum_{y_i=1}^{K} q(y_i)\log\frac{p(\mathbf{x}_i|y_i;\theta)p(y_i;\theta)}{q(y_i)} \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} q(y_i=k)\log\frac{p(\mathbf{x}_i|y_i=k;\theta)p(y_i=k;\theta)}{q(y_i=k)} \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\log\left\{\frac{1}{(2\pi)^{m/2}|\Sigma_k|^{1/2}} exp[-\frac{1}{2}(\mathbf{x}_i-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}_i-\mu_k)]\pi_k\right\} - \sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\log q_{ik} \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\left\{-\frac{1}{2}(\mathbf{x}_i-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}_i-\mu_k)] - \frac{1}{2}\log|\Sigma_k| + \log\pi_k\right\} + C
\end{aligned} \tag{4.29}
$$

Take the derivative w.r.t. μ_k and set it to zero

$$
\begin{aligned}
&\frac{\partial\mathcal{L}(\theta)}{\partial\mu_k} = 0 \implies \sum_{i=1}^{N} -q_{ik}\Sigma_k^{-1}(\mathbf{x}_i-\mu_k) = 0 \\
&\implies \sum_{i=1}^{N} q_{ik}\mathbf{x}_i - \sum_{i=1}^{N} q_{ik}\mu_k = 0 \implies \mu_k = \frac{\sum_{i=1}^{N} q_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} q_{ik}}
\end{aligned} \tag{4.30}
$$

Take the derivative w.r.t. Σ_k and set it to zero

$$
\begin{aligned}
&\frac{\partial\mathcal{L}(\theta)}{\partial\Sigma_k} = 0 \\
&\implies \sum_{i=1}^{N} q_{ik}[\Sigma_k^{-1}(\mathbf{x}_i-\mu_k)(\mathbf{x}_i-\mu_k)^T\Sigma_k^{-1} - {\Sigma_k}^{-1}] = 0 \\
&\implies \Sigma_k = \frac{\sum_{i=1}^{N} q_{ik}(\mathbf{x}_i-\mu_k)(\mathbf{x}_i-\mu_k)^T}{\sum_{i=1}^{N} q_{ik}}
\end{aligned} \tag{4.31}
$$

As for the prior π_k, we have $\sum_{k=1}^{K}\pi_k = 1$. Thus we use the Lagrange multipliers and get the following objective function $\mathcal{L}(\pi_k) = \sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\pi_k + \lambda(\sum_{i=1}^{K}\pi_k - 1)$

$$
\frac{\partial\mathcal{L}(\pi_k)}{\partial\pi_k} = 0
$$

$$\implies \sum_{i=1}^{N} q_{ik}\pi_k + \lambda = 0$$
$$\implies \pi_k = \frac{\sum_{i=1}^{N} q_{ik}}{-\lambda} \tag{4.32}$$

According to $\sum_{k=1}^{K} \pi_k = 1$, we have

$$\sum_k \pi_k = \sum_k \frac{\sum_{i=1}^{N} q_{ik}}{-\lambda} = 1$$
$$\implies \frac{\sum_{i=1}^{N}\sum_k q_{ik}}{-\lambda} = 1$$
$$\implies -\lambda = N \tag{4.33}$$

Finally, we can get $\pi_k = \frac{\sum_{i=1}^{N} q_{ik}}{-\lambda} \implies \pi_k = \frac{\sum_{i=1}^{N} q_{ik}}{N}$. Note that q_{ik} is the posterior probability from Eq. 4.21, with the following formula

$$q_{ik} = p(y_i = k|\mathbf{x}_i, \theta) \propto p(\mathbf{x}_i|y_i = k, \theta)p(y_i = k) \tag{4.34}$$
$$= \pi_k p(\mathbf{x}_i|y_i = k, \theta) \tag{4.35}$$

4.3 Spectral clustering

The previous clustering algorithms, such as k-means and GMM, groups the data with compactness. In this section, we will introduce spectral clustering, which is a graph-based clustering approach by partitioning the data with connectivity [13, 14]. Specifically, it uses eigenvalue decomposition of the similarity graph of the data to do dimension reduction before clustering in the low dimensional space. Compared to k-means, spectral clustering can be formulated as a weighted kernel k-means problem.

4.3.1 Similarity graphs

Given a set of observations $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, we can define a similarity matrix to describe the connectivity between any pair $(\mathbf{x}_i, \mathbf{x}_j)$. Then, based on the graph connectivity, we can divide the instances into several groups, such that instances in the same group are similar and points in different groups are dissimilar to each other.

Similarity matrix S: We can construct a graph $G = (V, S)$, where each vertex v_i is corresponding to the instance $\mathbf{x}_i$ in the training data $\mathcal{D}$, and S_{ij} is the weight or similarity between any vertex pair $(\mathbf{x}_i, \mathbf{x}_j)$. Thus, we use the similarity graph **S** to represent the similarity between instances, and define the non-negative similarity matrix **S** as

$$S_{ij} = \begin{cases} exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2), & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases} \tag{4.36}$$

where we use Gaussian kernel in Eq. 4.36. $S_{ij} = 0$ means that the instances $\mathbf{x}_i$ and $\mathbf{x}_j$ are not connected by an edge. We can also note that S is a symmetry matrix with $S_{ij} = S_{ji}$. Notice that we can also define other similarity matrices, which takes a similar strategy as kernel SVM.

The degree of a vertex: the degree of a vertex $v_i \in V$ is defined as $d_i = \sum_{j=1}^{N} S_{ij}$. Then, we have the degree matrix **D** as follows

$$\mathbf{D} = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_N \end{pmatrix}$$

4.3.2 Algorithm

Data: the training set $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and the number of clusters K

Result: output the K clusters$\{C_1, ..., C_K\}$

1. Construct the similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$:

$$S_{ij} = \begin{cases} exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2), & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases}$$

2. Define $\mathbf{D}$ (degree matrix) to be the diagonal matrix with: $d_i = \sum_{j=1}^{N} S_{ij}$ then, compute unnormalized graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{S}$;

3. Do component analysis to $\mathbf{L}$ with SVD, and yield the K largest eigenvectors of: $\{u_1, u_2, ..., u_K\}$; and construct matrix $\mathbf{U} = [u_1, u_2, ..., u_K]$ with eigenvectors as columns ;

4. treating each row of $\mathbf{U}$ as an observation in $\mathbb{R}^K$, clustering them into K groups $C_1, ..., C_K$ via k-means or any other algorithm;

5. Finally, assign the original point $\mathbf{x}_i$ to cluster C_j if and only if row of matrix $\mathbf{U}$ was assigned to cluster C_j.

Algorithm 7: Spectral clustering algorithm

Note that the step 2 to compute $\mathbf{L}$ is important to the clustering results in Algorithm 7. There are quite a few methods to use normalized Laplacian matrix to replace $\mathbf{L}$ and then forward it to SVD and further to do clustering analysis.

4.3.3 Algorithm understanding

Recall that graph Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{S}$. Then, according the similarity of $\mathbf{S}$, we have $\mathbf{L}^T = \mathbf{D}^T - \mathbf{S}^T = \mathbf{D} - \mathbf{S}$. Thus, we can see that $\mathbf{L}$ is a symmetric matrix.

For any vector $f \in \mathbb{R}^N$, we have

$$f^T \mathbf{L} f = \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 \tag{4.37}$$

Proof: by the definition of **D**, we have

$$\begin{aligned} f^T\mathbf{L}f &= f^T\mathbf{D}f - f^T\mathbf{S}f = \sum_{i=1}^{N} d_i f_i^2 - \sum_{i,j=1}^{N} S_{ij} f_i f_j \\ &= \frac{1}{2}\Big(\sum_{i=1}^{N} d_i f_i^2 - 2\sum_{i,j=1}^{N} S_{ij} f_i f_j + \sum_{j=1}^{N} d_j f_j^2 \Big) \\ &= \frac{1}{2}\sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 \geq 0 \end{aligned} \tag{4.38}$$

We can conclude:

- **L** is symmetric and positive semi-definite,
- **L** has N non-negative real-valued eigenvalues, $\lambda_1 \geq \lambda_2 \geq ...\lambda_N \geq 0$, and it can be decomposed into a weighted sum of outer products:

$$\mathbf{L} = \sum_{i=1}^{N} \lambda_i \phi_i \phi_i^T \tag{4.39}$$

  where ϕ_i is the corresponding eigenvector of λ_i, and $\phi_i^T \phi_j = 0$, if $i \neq j$.

- **L** is semi-definite, so the smallest eigenvalue of **L** is 0, the corresponding eigenvector consists of equal value, indicating the connectivity.

According to Eq. 4.38, we can infer that **L** is positive semi-definite. Further, it is obvious that it has N non-negative real-valued eigenvalues.

As the weights S_{ij} are non-negative, this sum in Eq. 4.38 can only vanish if all terms $S_{ij}(f_i - f_j)^2$ vanish $\forall i,\ j$. So if two vertices v_i and v_j are connected (i.e., $S_{ij} > 0$), then f_i needs to equal f_j. With this argument, we can see that f needs to be constant vector with the same value over all N dimensions. And it further indicates all vertices can be connected by a path in the graph. In other words, the graph consists of only one fully connected component.

As we have mentioned, the whole graph is fully connected, if only if **L** has the eigenvalue 0. Now we consider the ideal case that the graph can be partitioned into K connected components. Without loss of generality, we assume that the vertices

are ordered according to the connected components they belong to. In this case, the adjacency matrix $\mathbf{S}$ has a block diagonal form, and the same is true for the matrix $\mathbf{L}$:

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_1 & & & \\ & \mathbf{L}_2 & & \\ & & \ddots & \\ & & & \mathbf{L}_K \end{pmatrix}$$

where $\mathbf{L}_k$ is a submatrix (or subgraph) from $\mathbf{L}$. we also use $\mathbf{S}^k$ to represent the same region corresponding to $\mathbf{L}_k$.

Analogically, we can yield the following equation according to Eq. 4.38

$$f^T \mathbf{L} f = \sum_{k=1}^{K} f^T \mathbf{L}_k f = \frac{1}{2} \sum_{k=1}^{K} \sum_{i,j=1}^{N} S_{ij}^k (f_i - f_j)^2 \tag{4.40}$$

we quoted the sentence below from [14]. "Note that each of the blocks $\mathbf{L}_k$ is a proper graph Laplacian on its own, namely the Laplacian corresponding to the subgraph of the k-th connected component. As it is the case for all block diagonal matrices, we know that the spectrum of $\mathbf{L}$ is given by the union of the spectra of $\mathbf{L}_k$, and the corresponding eigenvectors of $\mathbf{L}$ are the eigenvectors of $\mathbf{L}_k$, filled with 0 at the positions of the other blocks. As each $\mathbf{L}_k$ is a graph Laplacian of a connected graph, we know that every $\mathbf{L}_k$ has eigenvalue 0 with multiplicity 1, and the corresponding eigenvector is the constant one vector on the k-th connected component. Thus, the matrix $\mathbf{L}$ has as many eigenvalues 0 as there are connected components, and the corresponding eigenvectors are the indicator vectors of the connected components."

Example: we can consider a very easy case with 3 components

$$\mathbf{L} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Then we can do SVD to $\mathbf{L}$, and we get eigenvalues = $\{2, 1, 1\}$ and all eigenvectors

$$\mathbf{U} = \begin{pmatrix} -0.7071 & 0 & 0 & -0.7071 \\ -0.7071 & 0 & 0 & 0.7071 \\ 0 & 0 & 1.0 & 0 \\ 0 & 1.0 & 0 & 0 \end{pmatrix}$$

If we want to group it into 3 clusters, we can pick 3 eigenvectors corresponding to the top 3 eigenvalues

$$\mathbf{U} = \begin{pmatrix} -0.7071 & 0 & 0 \\ -0.7071 & 0 & 0 \\ 0 & 0 & 1.0 \\ 0 & 1.0 & 0 \end{pmatrix}$$

And we can do k-means clustering to $\mathbf{U}$ with each row as an instance (or observation), and we can get 3 clusters below

$$\mathbf{C}_1 = \begin{pmatrix} -0.7071 & 0 & 0 \\ -0.7071 & 0 & 0 \end{pmatrix}$$

$\mathbf{C}_2 = \{0, 0, 1\}$ and $\mathbf{C}_3 = \{0, 1, 0\}$

4.3.4 The normalized graph Laplacians

The normalized Laplacian matrix is defined as

$$\mathbf{L}_{norm} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{S})\mathbf{D}^{-1/2} \tag{4.41}$$

Then, for any $f \in \mathbb{R}^N$, we have

$$\begin{aligned} f^T\mathbf{L}_{norm}f &= f^T\mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{S})\mathbf{D}^{-1/2}f \\ &= \hat{f}^T(\mathbf{D} - \mathbf{S})\hat{f} \qquad (4.42) \\ &= \frac{1}{2}\sum_{i,j=1}^{N} S_{ij}(\hat{f}_i - \hat{f}_j)^2 \qquad (4.43) \end{aligned}$$

$$= \frac{1}{2} \sum_{i,j=1}^{N} S_{ij} (\frac{f_i}{\sqrt{d}_i} - \frac{f_j}{\sqrt{d}_j})^2 \tag{4.44}$$

Note that we define

$$\hat{f} = \mathbf{D}^{-1/2} f = [\frac{f_1}{\sqrt{d}_1}, ..., \frac{f_N}{\sqrt{d}_N}] \tag{4.45}$$

so we can derive Eq. 4.43 from Eq. 4.42 using Eq. 4.45. Also, Eq. 4.43 is from Eq. 4.38. Lastly, we plug Eq. 4.45 back into Eq. 4.43, we can get our final equation 4.44.

Similarly, we can consider the eigenvector with eigenvalue 0 (for the case with fully connected graph). Because S_{ij} is non-negative, we need to set $\frac{f_i}{\sqrt{d}_i} = \frac{f_j}{\sqrt{d}_j}, \forall i,\ j$ in order to make Eq. 4.44 zero. Thus, 0 is an eigenvalue of $\mathbf{L}_{norm}$ with eigenvector $\mathbf{D}^{1/2}\mathbf{1}$. And the same formula can be applied to the graph with K components.

4.4 Hierarchical clustering

We have talked about flat clustering methods, such as k-means and GMM. However, these flat clustering algorithms exist a vital flaw, which namely require a predefined number of clusters as input. On the contrary, hierarchical clustering, as a widely used data analysis method, can return a hierarchical tree structure or (dendrogram) and is more informative to describe the data itself. In general, it is implemented with a bottom-up manner, which treats each instance as a singleton cluster at the leaf, and then successively merge nearest pairs to the next level until one single cluster left. The clustering tree allows you to decide which level or scale of clustering is most fitting to your data or application. Of course, these advantages of hierarchical clustering come at the cost of computational complexity. The most common hierarchical clustering algorithms have a complexity that is at least quadratic in the number of instances compared to the linear complexity of k-means and EM.

In hierarchical clustering, one vital step is to pre specify the similarity measure (distance metric) to measure the distance between two clusters C_1 and C_2. There are many possible ways to define the distance between two sets. For example, we can define it to be the closest pair of points respectively in C_1 and C_2, or the fairest pair, or the average distance of the two clusters. We can describe the simple distance between

two clusters as

$$d(C_1, C_2) = \min_{\mathbf{x}_1 \in C_1, \mathbf{x}_2 \in C_2} d(\mathbf{x}_1, \mathbf{x}_2) \tag{4.46}$$

where $d(\mathbf{x}_1, \mathbf{x}_2)$ can be any distance metrics define in Chapter 1.

Or we can define the two clusters' distance as

$$\begin{aligned} & d(C_1, C_2) = d(\mathbf{x}_1, \mathbf{x}_2) \\ & s.t.\ \mathbf{x}_1 = \sum_{i=1, \mathbf{x}_i \in C_1}^{|C_1|} \frac{\mathbf{x}_i}{|C_1|},\ \mathbf{x}_2 = \sum_{j=1, \mathbf{x}_j \in C_2}^{|C_2|} \frac{\mathbf{x}_j}{|C_2|} \end{aligned} \tag{4.47}$$

where $|C_1|$ is the cardinality or the number of instances in C_1. There are many other distance measures over two sets, which we will not cover in this part.

Now we go to the algorithm: the outline of algorithm to hierarchical agglomerative clustering

Data: the training set $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$ and a similarity measure $d(\cdot, \cdot)$
Result: output a binary hierarchical tree which merges from leaf to root
initialize each instance $\mathbf{x}_i$ as a cluster ;
initialize the number of cluster $K = N$;
while $K > 1$ **do**
    Find the two closest cluster pair C_1 and C_2 based on the $d(C_1, C_2)$;
    Merge C_1 and C_2 into one cluster C, and update the instances in the new cluster (and the cluster center);
    Update $K = K - 1$;
end

Algorithm 8: hierarchical agglomerative clustering

4.5 Nonparametric clustering

Nonparametric clustering makes no assumption about size or complexity of the underline models. More specifically, it grows the number of parameters with the amount of training data. Compared to parametric models, non-parametric models sometimes are called infinite model, while parametric model has a fixed number of parameters.

Bayesian nonparametric models [15, 16, 17, 18] have received a lot of attention in

the machine learning community. The attractive property of these models is that the number of components can be learned automatically from data, without being specified in advance. One of the most popular nonparametric models for clustering is Dirichlet process mixture model (DPM) [19, 20, 21, 22], which has been widely used on character recognition and document categorization [18, 23]. For DPM, the prior imposed by Dirichlet process (DP) is defined by two parameters: the concentration parameter α and the base measure G_0 respectively. In a DP mixture model, both two parameters heavily influence the model selection and clustering performance. In general, the concentration parameter can be learned from the data adaptively [17, 24] and the component parameters can be estimated by maximizing posterior probability [17]. However, it remains challenging to estimate these parameters due to the appearance of intractable normalizing constants in the likelihood. One possible trend is to either assume conjugate priors or employ approximate methods [18, 23] to accelerate the inference. Another trend that catches great attention recently is to learn the parameters with the discriminative models [3, 7]. An important question is whether it is possible to learn a discriminative model for nonparametric clustering; and whether the modeling performance is weakened without conjugate prior assumption.

In the following part, we will introduce Dirichlet process and Dirichlet process mixture model (DPM). Then we will talk about a maximum margin approach for nonparametric clustering.

4.5.1 Dirichlet Process

The Dirichlet process (DP) [25] defines a distribution over distributions. Let Θ denote a probability space, G_0 be a distribution (or base measure) over Θ. A Dirichlet process is parameterized by G_0 and a positive scaling parameter α. Writing a draw from the Dirichlet process as

$$G|G_0, \alpha \sim \mathrm{DP}(\alpha, G_0) \tag{4.48}$$

where the base measure G_0 is basically the mean of DP.

For a distribution over probability measures to be a DP, its marginal distributions have to take on a specific form. Specifically, for any finite measurable partition

$A_1, ..., A_K$ of Θ, such that

$$\cup_{i=k}^{K} A_i = \Theta,\ A_i \cap A_j = \emptyset,\ \forall i \neq j \tag{4.49}$$

we say G is Dirichlet process distributed with base distribution G_0 and α, if

$$\big(G(A_1), ..., G(A_K)\big) \sim Dir\big(\alpha G_0(A_1), ..., \alpha G_0(A_K)\big) \tag{4.50}$$

Specifically, for any region $A \subset \Theta$, we have $E(G(A)) = G_0(A)$. And α is the concentration parameter. Larger α, more concentrated its mass around the mean.

Proof. Given the measurable region A, we have the following via Eq. 4.50

$$\big(G(A), G(A^c)\big) \sim Dir\big(\alpha G_0(A), \alpha G_0(A^c)\big) = Dir\big(\alpha G_0(A), \alpha(1 - G_0(A))\big) \tag{4.51}$$

According to mean and variance of Dirichlet distribution in Appendix B, we can get

$$\begin{aligned} E(G(A)) &= \frac{\alpha G_0(A)}{\alpha G_0(A) + \alpha G_0(A^c)} = G_0(A) \\ Var(G(A)) &= \frac{\alpha G_0(A) \alpha G_0(A^c)}{\alpha^2(\alpha + 1)} = \frac{G_0(A)(1 - G_0(A))}{\alpha + 1} \end{aligned} \tag{4.52}$$

Thus, α can be understood as an inverse variance (similar to the precision in Gaussian distribution).

Posterior distribution Let $G \sim DP(\alpha, G_0)$. Then we can draw samples from G. Let $\theta_1, ..., \theta_N$ be i.i.d. samples from G. Note that θ_i take values in Θ, since G is a distribution over Θ. Further, let $A_1, ..., A_K$ be a finite partition of Θ, and let $n_k = \#i : \theta_i \in A_k$ be the number of observed values in A_k, with $\sum_i n_i = n$. Then according to Eq. 4.50 and the conjugacy between Dirichlet distribution and the multinomial distribution, the posterior distribution over G is also a DP

$$\big(G(A_1), ..., G(A_K)\big) | \theta_1, ..., \theta_n \sim Dir(\alpha G_0(A_1) + n_1, ..., \alpha G_0(A_K) + n_K) \tag{4.53}$$

Note that $\theta_1, ..., \theta_N$ is sampled from the partitions $A_1, ..., A_K$, so it follows multinomial distribution. Further the partitions $A_1, ..., A_K$ is from Dirichlet prior, thus we can conclude the posterior over G is also a DP. Note that the posterior distribution via Eq.

4.53 is from Dirichlet, thus we can rewrite it with new parameters:

$$\alpha' = \sum_{i=1}^{K} \alpha G_0(A_i) + n_i = \alpha + n$$
$$\alpha G_0(A_i) + n_i = \alpha' G_0'(A_i) \implies G_0' = \frac{\alpha G_0 + \sum_{i=1}^{n} \delta_{\theta_i}}{\alpha + n}$$

where $\sum_{i=1}^{n} \delta_{\theta_i}$ is the point mass (or n_i) located at θ_i. Thus, we can rewrite Eq. 4.53 as

$$\big(G(A_1), ..., G(A_K)\big)|\theta_1, ..., \theta_n \sim Dir(\alpha + n, \frac{\alpha}{\alpha + n} G_0 + \frac{n}{\alpha + n} \frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n}) \tag{4.54}$$

Notice that the posterior base distribution is a weighted average between the prior base distribution G_0 and the empirical distribution $\frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n}$. In other words, as the number of observations grows, i.e. $n \gg \alpha$, the posterior is simply dominated by the empirical distribution.

Predictive distribution the predictive distribution θ_{n+1} after $\theta_1, ..., \theta_n$ can be estimated by marginalized out G

$$p(\theta_{n+1}) = \int_G p(\theta_{n+1}, G|\theta_1, ..., \theta_n) \tag{4.55}$$

Since θ_{n+1} draws from G, specifically with $\theta_{n+1}|G, \theta_1, ..., \theta_n \sim G$, for any measurable region $A \subset \Theta$, we have

$$p(\theta_{n+1} \in A|\theta_1, ..., \theta_n) = E[G(A)|\theta_1, ..., \theta_n] = \frac{\alpha G_0(A) + \sum_{i=1}^{n} \delta_{\theta_i}(A)}{\alpha + n} \tag{4.56}$$

which is exactly the same as the posterior base measure of G given the first n observations. Since Eq. 4.56 holds for any A, we have

$$\theta_{n+1}|\theta_1, ..., \theta_n \sim \frac{1}{\alpha + n}(\alpha G_0 + \sum_{i=1}^{n} \delta_{\theta_i}) \tag{4.57}$$

Notice data generated from this model can be partitioned according to the distinct values of the parameter. Thus, the variables $\{\theta_1, ..., \theta_n\}$ are randomly partitioned according to which variables are equal to the same value, with the distribution of the partition obtained from a Pólya urn scheme [26] or stick breaking prior [27]. Let

$\{\theta_1^*, ..., \theta_{|c|}^*\}$ denote the distinct values of $\{\theta_1, ..., \theta_n\}$, let $c = \{c_1, ..., c_n\}$ be assignment variables such that $\theta_i = \theta_{c_i}^*$, and let $|c|$ denote the number of cells in the partition. The distribution of θ_{n+1} follows the urn distribution:

$$\theta_{n+1} \sim \begin{cases} \theta_i^* \text{ with prob} \frac{|c|_i}{n+\alpha} & \text{if } i \in \{1, 2, ..., |c|\} \\ \theta \sim G_0, \text{ with prob} \frac{\alpha}{n+\alpha} & \text{otherwise new cluster} \end{cases}$$

Thus, The probability for assigning instance θ_{n+1} to either an existing component $|c|$ or to a new one cluster conditioned on the other component assignments (c_i) according to Eq. 4.56.

Chinese restaurant process: The sample process can also be modeled with Chinese restaurant process. In this metaphor we have a Chinese restaurant with an infinite number of tables, each of which can seat an infinite number of customers. The first customer enters the restaurant and sits at the first table. The second customer enters and decides either to sit with the first customer, or by herself at a new table. In general, the $n + 1$st customer either joins an already occupied table k with probability proportional to the number n_k of customers already sitting there, or sits at a new table with probability proportional to α. Fixing all but a single indicator z_i, we can obtain the conditional probability for each individual indicator

$$p(z_i = k|\mathbf{z}_{-i}, \alpha) = \int_{\boldsymbol{\pi}} p(z_i|\boldsymbol{\pi}) p(\boldsymbol{\pi}|\alpha) = \frac{n_{-i,k} + \alpha/K}{n - 1 + \alpha} \tag{4.58}$$

where the subscript $-i$ indicates all indices except for i, and $n_{-i,k}$ is the number of data points, excluding $\mathbf{x}_i$, that are associated with class k. Let K go to infinity, the conditional distribution of the indicator variables reaches the following limits [21]:

$$p(z_i|\mathbf{z}_{-i}, \alpha) = \begin{cases} p(z_i = k|\mathbf{z}_{-i}, \alpha) = \dfrac{n_{-i,k}}{n - 1 + \alpha} \\ p(z_i \neq z_{i'} \text{ for all } i \neq i'|\mathbf{z}_{-i}, \alpha) = \dfrac{\alpha}{n - 1 + \alpha} \end{cases} \tag{4.59}$$

In other words, the prior for assigning instance $\mathbf{x}_i$ to either an existing component k or to a new one cluster conditioned on the other component assignments (z_i) is given by Chinese restaurant process [18].

4.5.2 Dirichlet process mixture model

The most widely used nonparametric Bayesian method is the Dirichlet process mixture model (DPM) [15]. The DPM assumes that a probability distribution can be represented as an infinite mixture of parametric mixture model [17, 18, 19], where the parameters of those densities are generated from the Dirichlet process [25] parameterized by G_0 and α. As a generative model, we can generate $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$ with symmetric Dirichlet prior:

$$\begin{aligned} &\boldsymbol{\pi}|\alpha \sim Dir(\alpha/K, ..., \alpha/K) \\ &z_i|\boldsymbol{\pi} \sim Discrete(\pi_1, ..., \pi_K) \\ &\theta_k \sim G_0 \\ &\mathbf{x}_i|z_i, \{\theta_k\}_{k=1}^K \sim p(\mathbf{x}_i|\theta_{z_i}) \end{aligned} \tag{4.60}$$

where z_i is the cluster indicator for $\mathbf{x}_i$, $\boldsymbol{\pi}$ are mixing proportions for $\mathbf{x}_i$ and K is the total number of clusters. In this paper, we consider the most famous DPM with Gaussian distribution on each component. That means the mixture components of a DPM are specified by their mean and covariance $\{\mu_k, \Sigma_k\}_{k=1}^K$; and for each instance $\mathbf{x}_i$, its probability belongs to cluster k is Gaussian: $p(\mathbf{x}_i|\theta_k) \sim \mathcal{N}(\mu_k, \Sigma_k)$. G_0 specifies the prior on the joint distribution of (μ, Σ) common (or shared) among all components. If a multivariate Gaussians mean and covariance are both uncertain, we need to specify the priors for them. In general, conjugate prior is used, that means the covariance matrix is assigned an inverse-Wishart prior $\Sigma \sim \mathcal{W}(\nu, \Delta)$, and the mean $\mu \sim \mathcal{N}(\vartheta, \Sigma/\kappa)$, then normal-inverse-Wishart distribution [28] provides an appropriate conjugate prior, denoted by $\mathcal{NW}(\kappa, \vartheta, \nu, \Delta)$, with the following form:

$$p(\mu, \Sigma|\kappa, \vartheta, \nu, \Delta) \propto |\Sigma|^{-\left(\frac{\nu+d}{2}\right)} \exp\left\{-\frac{1}{2}\mathrm{tr}(\nu\Delta\Sigma^{-1}) - \frac{\kappa}{2}(\mu-\vartheta)\Sigma^{-1}(\mu-\vartheta)\right\} \tag{4.61}$$

where ν, κ, ϑ and Δ are hyperparameters common to all mixture components, expressing the belief where the component parameters should be similar, centered around some particular value. Here, ϑ is the expected mean, for which we have κ pseudo observations on the scale of observations $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$, for mathematical manipulation, refer to [28, 29] and Appendix B.

Likelihood and parameters estimation: Integrating over the parameters (μ, Σ) from the normal-inverse-Wishart posterior distribution, the predictive likelihood of a new observation $\mathbf{x}$ is multivariate Student-t distribution with $(\hat{\nu} - d + 1)$ degrees of freedom. Assuming $(\hat{\nu} > d + 1)$, the predictive likelihood can be approximated by a moment-matched Gaussian:

$$p(\mathbf{x}|\mathbf{x}_1, ..., \mathbf{x}_n, \kappa, \vartheta, \nu, \Delta) \propto \mathcal{N}\left(\mathbf{x}; \hat{\vartheta}, \frac{(\hat{\kappa}+1)\hat{\nu}}{\hat{\kappa}(\hat{\nu}-d-1)}\hat{\Delta}\right) \tag{4.62}$$

where all parameters are estimated with maximizing posterior for each component parameters and updating as following give n observations $\{\mathbf{x}_i\}_{i=1}^n$

$$\begin{aligned} &\hat{\kappa}\hat{\vartheta} = \kappa\vartheta + \textstyle\sum_{i=1}^n \mathbf{x}_i \quad \hat{\kappa} = \kappa + n; \\ &\hat{\nu}\hat{\Delta} = \nu\Delta + \textstyle\sum_{i=1}^n \mathbf{x}_i(\mathbf{x}_i)^T = \kappa\vartheta\vartheta^T - \hat{\kappa}\hat{\vartheta}\hat{\vartheta}^T \quad \hat{\nu} = \nu + n; \end{aligned} \tag{4.63}$$

Inference: Given the data points $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$ and its the cluster indicators $\mathcal{Z} = \{z_i\}_{i=1}^N$, the Gibbs sampling involves iterations that alternately draw samples from conditional probability while keeping other variables fixed. Recall that for each indicator variable z_i, we can derive its conditional posterior in DPM as follows:

$$\begin{aligned} & p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \mathbf{x}_i, \{\theta_k\}_{k=1}^K, \alpha, \beta) && (4.64) \\ = \;& p(z_i = k|\mathbf{x}_i, \mathbf{z}_{-\mathbf{i}}, \{\theta_k\}_{k=1}^K) && (4.65) \\ \propto \;& p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \{\theta_k\}_{k=1}^K) p(\mathbf{x}_i|z_i = k, \{\theta_k\}_{k=1}^K) && (4.66) \\ = \;& p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \alpha) p(\mathbf{x}_i|\theta_k) && (4.67) \end{aligned}$$

where the likelihood term $p(\mathbf{x}_i|\theta_k)$ according to Eq. (4.62), and $p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \alpha)$ can be modeled with Chinese restaurant process via Eq. 4.59, refer to [18] for more information.

4.5.3 Maximum margin Dirichlet process mixtures

Recall that the DPM model maximizes the joint model

$$P(\mathcal{X}, \mathbf{z}, \{\boldsymbol{\theta}_k\}_{k=1}^K) \propto p(\mathbf{z}) \prod_{i=1}^N p(\mathbf{x}_i|\boldsymbol{\theta}_{z_i}) \prod_{k=1}^K p(\boldsymbol{\theta}_k|\beta) \tag{4.68}$$

where $p(\boldsymbol{\theta}_k|\beta)$ is defined by the base measure $G_0(\beta)$, and $p(\mathbf{z}) = \frac{\Gamma(\alpha)\prod_{k=1}^{K}\Gamma(n_k+\alpha/K)}{\Gamma(n+\alpha)\Gamma(\alpha/K)^K}$ for the symmetric Dirichlet prior. As for our discriminative model, we maximize the following conditional likelihood:

$$P(\mathbf{z}, \{\boldsymbol{\theta}_k\}_{k=1}^{K}|\mathcal{X}) \propto p(\mathbf{z})\left[\prod_{i=1}^{N} p(\mathbf{x}_i|\boldsymbol{\theta}_{z_i})\right]\prod_{k=1}^{K} p(\boldsymbol{\theta}_k) \tag{4.69}$$

where $p(\mathbf{z})$ has the same definition as in DPM above, while we have no prior base measure restriction on $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$ in our discriminative model. $p(\boldsymbol{\theta}_k)$ for $k = \{1, ..., K\}$ can be thought as the Gaussian prior in SVM classifier in Eq. (4.79). The essential difference between our model and DPM is that our approach is a discriminative model, without modeling $p(\mathcal{X})$. And, we maximize a conditional probability for parameter estimation, instead of joint distribution as in DPM. Just as the conditional random fields model (CRF) [3], we propose a similar discriminative model, which do not need to tune prior assumption for $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$ constricted by $G_0(\beta)$ in DPM. Removing constraints reduces the statistical bias, and fit the model parameters well to the training data. In addition, the difficulty in modeling Eq. (4.68) is that it often contains many highly dependent features, which are difficult to model [4, 30].

4.5.3.1 Gibbs sampling

Given the data points $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{n}$ ($\mathbf{x}_i \in \mathbb{R}^d$) and their cluster indicators $\mathcal{Z} = \{z_i\}_{i=1}^{n}$, the Gibbs sampling involves iterations that alternately draws from conditional probability while keeping other variables fixed. Recall that for each indicator variable z_i, we can derive its conditional posterior in DPM as follows:

$$\begin{aligned}
& p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \mathbf{x}_i, \{\boldsymbol{\theta}_k\}_{k=1}^{K}, \alpha, \beta) && (4.70)\\
= \;& p(z_i = k|\mathbf{x}_i, \mathbf{z}_{-\mathbf{i}}, \{\boldsymbol{\theta}_k\}_{k=1}^{K}) && (4.71)\\
\propto \;& p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \{\boldsymbol{\theta}_k\}_{k=1}^{K})p(\mathbf{x}_i|z_i = k, \{\boldsymbol{\theta}_k\}_{k=1}^{K}) && (4.72)\\
= \;& p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \alpha)p(\mathbf{x}_i|\boldsymbol{\theta}_k) && (4.73)
\end{aligned}$$

where $p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \alpha)$ is determined by Eq. (4.59), and $p(\mathbf{x}_i|\boldsymbol{\theta}_k)$ is the likelihood for the current observation $\mathbf{x}_i$. To estimate $\boldsymbol{\theta}_k$, we need to maximize the conditional posterior, which depends on observations belonging to this cluster and prior $G_0(\beta)$.

If the current set for the cluster k, can be denoted as $\mathbf{x}_k$, with the number of elements $n_k = |\mathbf{x}_k|$, then DPM learns $\boldsymbol{\theta}_k$ by maximizing the posterior:

$$p(\boldsymbol{\theta}_k|\mathbf{x}_k, \beta) = p(\boldsymbol{\theta}_k|\beta) \prod_{i=1}^{|\mathbf{x_k}|} p(\mathbf{x}_{k_i}|\boldsymbol{\theta}_k) \tag{4.74}$$

For this model, a conjugate base distribution may exist, which can provide guarantee that the posterior probability can be computed in closed form and learn the model parameters explicitly. However, in general it is hard to choose an appropriate prior base distribution, i.e., often chosen based on mathematical and convenient concern. Moreover, it has the unappealing property of prior dependency, and cannot reflect the observed data distribution in real scenarios.

In our conditional likelihood model, we replace the generative model in DPM with our discriminative SVM classifier. More specifically, we relax the prior restriction $G_0(\beta)$ and learn the component parameters $\{\boldsymbol{\theta}_k\}_{k=1}^K$ in a discriminative manner. Thus, we define the following likelihood for instance $\mathbf{x}_i$ in Eq. (4.73):

$$p(\mathbf{x}_i|\boldsymbol{\theta}_k) \propto exp(\mathbf{x}_i^T\boldsymbol{\theta}_k - \lambda||\boldsymbol{\theta}_k||^2) \tag{4.75}$$

where λ is a regularization constant to control weights between the two terms above. By default, the prediction function should be proportional to $\arg\max_k(\mathbf{x}_i^T\boldsymbol{\theta}_k)$, for $k \in \{1, ..., K\}$. In our likelihood definition, we also minus $\lambda||\boldsymbol{\theta}_k||^2$ in Eq. (4.75), which can keep the maximum margin beneficial properties in the model to separate clusters as far away as possible. Moreover, it can get rid of trivial clustering results [31]. Note that the seminal work [32] basically fits a sigmoid function over SVM decision values (e.g. $\mathbf{x}_i^T\boldsymbol{\theta}_k$) to scale it to the range of $[0, 1]$, which can then be interpreted as a kind of probability. Compared to [32], we maximize $(\mathbf{x}_i^T\boldsymbol{\theta}_k)$ and minimize $\lambda||\boldsymbol{\theta}_k||^2$ simultaneously in Eq. (4.75), so our method can keep larger margin between clusters. Another understanding for the above likelihood is that Eq. (4.75) satisfies the general form of exponential families, which are functions solely of the chosen sufficient statistics [29]. Thus, such probability in Eq. (4.75) makes our model general enough to handle real applications.

Taking the similar form as in Eq. (4.73), we get the final Gibbs sampling strategy

for our MMDPM model:

$$\begin{aligned} & p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \mathbf{x}_i, \{\boldsymbol{\theta}_k\}_{k=1}^K, \alpha, \lambda) \\ & \propto p(z_i = k|\mathbf{z}_{-\mathbf{i}}, \alpha) exp(\mathbf{x}_i^T \boldsymbol{\theta}_k - \lambda ||\boldsymbol{\theta}_k||^2) \end{aligned} \tag{4.76}$$

we will introduce to learn component parameters $\{\boldsymbol{\theta}_k\}_{k=1}^K$ in the following Sec 4.5.3.2. For the new created cluster, we generate $\boldsymbol{\theta}_{K+1}$ that is perpendicular to all the previous $\boldsymbol{\theta}_k$, for $k \in \{1, ..., K\}$ [31, 33]. Basically, we random generate a vector $\boldsymbol{\theta} \in \mathbb{R}^d$, and then we project it on the weight vector $\boldsymbol{\theta}_k$, $k \in \{1, .., K\}$, and compute the residual vectors:

$$\boldsymbol{\theta}_{K+1} := \boldsymbol{\theta} - (\boldsymbol{\theta}_k^T \boldsymbol{\theta}) \boldsymbol{\theta}_k \tag{4.77}$$

The residual is the component of $\boldsymbol{\theta}_{K+1}$ that is perpendicular to $\boldsymbol{\theta}_k$, for $k \in \{1, ..., K\}$.

For the model we consider, we leverage MCMC algorithms for inference on the model discussed above by sampling each variable from posterior conditional probability given in Eq. (4.76) with others fixed in an alternative way. In addition, we update α using Adaptive Rejection Sampling (ARS) [24] as suggested in [17].

4.5.3.2 Maximum margin learning

Given the clustering label for each instance, we can use K-means to estimate the component parameters. Unfortunately, K-means cannot keep an larger margin properties between clusters. In our work, we estimate the component parameters under the maximum margin framework. More specifically, we use the variant of the passive aggressive algorithm (PA) [9] to learn component parameters. Basically, our online algorithm treats the labeling inference with Gibbs sampling as groundtruth. Then, for any instance in a sequential manner, it infers an outcome with the current model. If the prediction mismatches its feedback, then the online algorithm update its model under the maximum margin framework, presumably improving the chances of making an accurate prediction on subsequent rounds.

We denote the instance presented to the algorithm on round t by $\mathbf{x}_t \in \mathbb{R}^d$, which is associated with a unique label $z_t \in \{1, ..., K\}$. Note that the label z_t is determined by the above Gibbs sampling algorithm in Eq. (4.76). Let's define $\mathbf{w} = [\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_K]$ is a parameter vector with K clusters (by concatenating all the parameters $\{\boldsymbol{\theta}_k\}_{k=1}^K$ into

$\mathbf{w}$, that means $\mathbf{w}^{z_t}$ is z_t-th block in $\mathbf{w}$, or says $\mathbf{w}^{z_t} = \boldsymbol{\theta}_{z_t}$), and $\mathbf{\Phi}(\mathbf{x}_t, z_t)$ is a feature vector relating input $\mathbf{x}_t$ and output z_t, which is composed of K blocks, and all blocks but the z_t-th blocks of are set to be the zero vector while the z_t-th block is set to be $\mathbf{x}_t$. We denote by $\mathbf{w_t}$ the weight vector used by the algorithm on round t, and refer to the term $\gamma(\mathbf{w}_t; (\mathbf{x}_t, z_t)) = \mathbf{w}_t \cdot \mathbf{\Phi}(\mathbf{x}_t, z_t) - \mathbf{w}_t \cdot \mathbf{\Phi}(\mathbf{x}_t, \hat{z_t})$ as the (signed) margin attained on round t, where $\hat{z_t} = \max_{z \in [1,K]} \mathbf{w}_t \cdot \mathbf{\Phi}(\mathbf{x}_t, z)$. In our work, we use hinge-loss function, which is defined by the following,

$$\begin{aligned} & \ell(\mathbf{w}; (\mathbf{x}_t, z_t)) \\ = & \begin{cases} 0 & \text{if } \gamma(\mathbf{w}; (\mathbf{x}_t, z_t)) \geq 1 \\ 1 - \gamma(\mathbf{w}; (\mathbf{x}_t, z_t)) & \text{otherwise} \end{cases} \end{aligned} \tag{4.78}$$

Following the passive aggressive (PA) algorithm [9], we optimize the objective function:

$$\begin{aligned} \mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} & \frac{1}{2} ||\mathbf{w} - \mathbf{w}_t||^2 + C\xi \\ s.t. \; & \ell(\mathbf{w}; (\mathbf{x}_t, z_t)) \leq \xi \end{aligned} \tag{4.79}$$

where the L_2 norm of $\mathbf{w}$ on the right hand size can be thought as Gaussian prior in SVM classifier. If there's loss, then the updates of PA-1 has the following closed form

$$\begin{aligned} \mathbf{w}_{t+1}^{z_t} &= \mathbf{w}_t^{z_t} + \tau_t \mathbf{x}_t, \\ \mathbf{w}_{t+1}^{\hat{z_t}} &= \mathbf{w}_t^{\hat{z_t}} - \tau_t \mathbf{x}_t, \end{aligned} \tag{4.80}$$

where $\tau_t = \min\{C, \frac{\ell(\mathbf{w}_t; (\mathbf{x}_t, z_t))}{||\mathbf{x}_t||^2}\}$. Note that the Gibbs sampling step can decide the indicator variable z_t for $\mathbf{x}_t$, we think it is the ground truth assignment for $\mathbf{x}_t$, and then we update our parameter $\mathbf{w}$ using the above Eq. (4.80).

Parameter space analysis: if the data dimension is d, and the current cluster number is K, then $\mathbf{w}$ need $d \times K$ in our model. While for the DPM model, if we assume a Gaussian distribution, we need to update and store $d^2 \times K$ for covariance matrix, and that is computationally expensive for high dimensional data. Even for the diagonal covariance matrix, it still requires $2d \times K$ to store both mean and covariance.

Time complexity analysis: In the algorithm, we do both inference and learning, so it needs $O(n \times d^2 \times K)$ in each iteration. Note that K is changing in each iteration. While

for the DPM model, the component parameters updating requires Cholesky decomposition in most cases. Thus, our online updating on the component parameters is more efficient than the DPM model.

Model coherence: Given component parameters $\{\boldsymbol{\theta}_k\}_{k=1}^K$ estimated from maximum margin learning, we can predict the likelihood for each $\mathbf{x}_t$ as $p(\mathbf{z}_i|\mathbf{x}_t;\boldsymbol{\theta}_k) \propto exp(\mathbf{x}_i^T\boldsymbol{\theta}_k)$. If we want to keep a large margin between different clusters, then we can introduce a prior $\{\boldsymbol{\theta}_k\}_{k=1}^K$ and finally get the posterior probability in Eq. (4.75). Note that Eq. (4.75) is close to the probabilistic formula of K-means.

4.5.3.3 Algorithm

We list the pseudo code below in Algorithm 9 and implemented it in Matlab. Our method takes an EM-like algorithm to estimate model parameters: infer the label of the current instance with Gibbs sampling; and update model parameters given the label for that instance.

Data: Input: sequential training data $\mathcal{X}$, C, λ, iterations T
Result: Output: $\mathbf{w}$, and $\mathcal{Z}$
Initialize $\mathbf{w_1}$, labels $\mathcal{Z}$ for training data, and prior α;
for $i = 1; i < n; i++$ **do**
  $\mathbf{x}_i = \mathbf{x}_i - mean(\mathcal{X})$;
end
for $t = 1$ *to* T **do**
  Permutate $\mathcal{X}$;
  for $i = 1; i < n; i++$ **do**
    Select an instance $(\mathbf{x}_i, z_i)$, and update $n_{z_i} = n_{z_i} - 1$;
    Eliminate empty clusters (if there's empty cluster, then $K = K - 1$);
    Create a new cluster $\boldsymbol{\theta}_{K+1}$ using Eq. (4.77), and update the number of clusters $K = K + 1$;
    for $j = 1; j <= K; j++$ **do**
      Calculate posterior probability in Eq. (4.76) for each cluster;
    end
    Sample its assignment according to the posterior probability in Eq. (4.76);
    Update its assignment $\hat{z}_i$ and $n_{\hat{z}_i} = n_{\hat{z}_i} + 1$;
    Update $\mathbf{w}_t$ using maximum margin clustering algorithm in Eq. (4.80);
  end
  Update α with ARS algorithm;
end
Return $\mathbf{w}$ and $\mathcal{Z}$;

Algorithm 9: Maximum margin Dirichlet process mixtures model

Chapter 5

Semi-supervised Learning

Previously, we have discussed supervised and unsupervised learning. As for supervised learning, we need to specify the label for each instance. Moreover, we have lot of approaches to learn a mapping (linear or nonlinear) from input to output, with the purpose to make good prediction on the test data. However, labeled data is hard to get, especially considering that human annotation is boring and labeling may require experts/special devices. On the contrary, unlabeled data is cheap and unsupervised learning approach can learn a model, without target information available. However, in general, it is hard to achieve the same accuracy as supervised learning does. Thus, we hope to leverage labeled data as less as possible with a large amount of unlabeled data to improve prediction performance.

In this Chapter, we will introduce semi-supervised clustering and classification, which requires partial labels or side information to boost clustering or classification performance. In the semi-supervised classification, the training data consists of labeled $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$ and unlabeled data $\mathcal{U} = \{\mathbf{x}_j\}_{j=l+1}^{N}$, usually $(N > L)$, and we need to learn a mapping $f : X \mapsto Y$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, ..., K\}$. In general, we can further categorize it into two problems:

(1) Inductive semi-supervised learning: Given both labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$, the inductive learning is to learn a mapping $f : X \mapsto Y$, so that f can make good prediction on future data or testing data

(2) transductive learning: Given both labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$, the transductive learning learns a function $f : X \mapsto Y$, so that it can predict well on the unlabeled data $\mathcal{U}$. Note that f is not require to make predictions on the testing set.

As for semi-supervised classification, we need to learn a mapping from input $\mathbf{x}$ to y. Because for every input $\mathbf{x}$, we have the corresponding target y in the training set $\mathcal{D}$, so we learn the model in an supervised manner. Basically, given both labeled and unlabeled data, we will learn an initial model and then attempt to extend (adapt) it to handle unlabeled data. Most methods will extend approaches discussed in Chapter 4 to handle semi-supervised problems.

Another problem we are interested in is semi-supervised clustering. All previous clustering approaches need to define distance function to compute similarity between each pair $(\mathbf{x}_i, \mathbf{x}_j)$, and then partition the data in to different clusters. However, the predefined distance function might do not capture the intrinsic similarity information. Thus, we can provide side information to guide the clustering process. In general, we has constrained pairs such as cannot-link and must-link as the training data which can be used to learn the similarity measure, and the purpose is to cluster the data with high precision. Especially, we will talk about maximum margin semi-supervised clustering with pairwise constraints, and extend it with deep learning to learn hidden similarity metric.

5.1 Gaussian mixture models for semi-supervised classification

As we mentioned in Chapter 4, we discussed Gaussian mixture models (GMM) for clustering analysis. As for supervised classification problem (e.g. K classes), we can use GMM to model the data distribution, and use maximum likelihood to estimate model parameters. While for unsupervised clustering, we can use EM algorithm to estimate model parameters. In this part, we will extend GMM to handle semi-supervised clustering problem. Before we introduce it, we first talk the supervised case and use maximum likelihood to initialize the parameters of mixture models.

Problem statement: given the training data consists of labeled $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$ and unlabeled $\mathcal{U} = \{\mathbf{x}_{l+1}, ..., \mathbf{x}_N\}$ instances, we need to classify all unlabeled data. Formally, for any instance $\mathbf{x} \in \mathcal{D}$, we want to predict its class label $y \in [1, K]$. If we

write it in a conditional probability way, we maximize the following

$$\hat{y} = \arg\max_{y} p(y|\mathbf{x};\theta) \propto \arg\max_{y} p(\mathbf{x}|y;\theta)p(y) \tag{5.1}$$

where θ is the model parameters, $p(\mathbf{x}|y;\theta)$ is the conditional probability and $p(y)$ is the prior. The problem is how to learn the model parameters θ.

As said, we can apply GMM to model this problem, where we can make use of labeled data to initialize the model and then generalize it to all unlabeled data. In the following, we will introduce these two steps to label all data.

5.1.1 Model initialization

Because we have partially labeled data, we can maximize the joint likelihood $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$. In particular, given the training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$ with $y_i \in [1, K]$, we can use maximum likelihood estimation (MLE) to learn the parameters θ

$$\log p(\mathcal{D}|\theta) = \log \prod_{i=1}^{l} p(\mathbf{x}_i, y_i|\theta) = \sum_{i=1}^{l} \log[p(x_i|y_i,\theta)p(y_i|\theta)] \tag{5.2}$$

where the training data are statistically independent, so we can factorize the joint probability into the product of probability. If the training data obeys Gaussian distribution with parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$, then we can rewrite Eq. 5.2 as

$$\theta = \arg\max_{\theta} \log p(\mathcal{D}|\theta) \; s.t. \; \sum_{y_i=1}^{K} p(y_i|\theta) = 1 \tag{5.3}$$

Using the Lagrange multiplier, we can easily get the following sufficient statistics for $k = \{1, ..., K\}$:

$$\pi_k = \frac{\sum_{i=1}^{l} \delta(y_i, k)}{l} \tag{5.4}$$

$$\mu_k = \frac{\sum_{i=1}^{l} \delta(y_i, k)\mathbf{x}_i}{\sum_{i=1}^{l} \delta(y_i, k)} \tag{5.5}$$

$$\Sigma_k = \frac{\sum_{i=1}^{l} \delta(y_i, k)(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T}{\sum_{i=1}^{l} \delta(y_i, k)} \tag{5.6}$$

where $\delta(y,k)$ is the indictor function, which equals 1 if $y == k$, otherwise set to zero. All the statistics above is an estimator of a population parameter which is usually close to θ as the number of training data approaches to ∞.

5.1.2 Semi-supervised classification

Notice that the training data consists of both labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$ instances, the log likelihood function is defined as

$$\begin{aligned}
\log p(\mathcal{D},\mathcal{U}|\theta) &= \log\left[\left(\prod_{i=1}^{l} p(\mathbf{x}_i,y_i|\theta)\right)\left(\prod_{j=l+1}^{N} p(\mathbf{x}_j|\theta)\right)\right] \\
&= \sum_{i=1}^{l}\log p(\mathbf{x}_i|y_i,\theta)p(y_i|\theta) + \sum_{j=l+1}^{N}\log p(\mathbf{x}_j|\theta) \\
&= \sum_{i=1}^{l}\log p(\mathbf{x}_i|y_i,\theta)p(y_i|\theta) + \sum_{j=l+1}^{N}\log\sum_{y_j=1}^{K} p(\mathbf{x}_j|y_j,\theta)p(y_j|\theta) \qquad (5.7)
\end{aligned}$$

Note that for $j = \{l+1,...,N\}$, $\mathbf{x}_j$ has no label, so y_j is the latent variable in Eq. 5.7. Unfortunately, the latent labels $y_{l+1},...,y_N$ make the log likelihood in Eq. 5.7 non-concave and hard to optimize. Fortunately, we can use the Expectation Maximization (EM) algorithm (which has been extensively discussed in Chapter 4) to find a locally optimal θ. The EM algorithm to estimate the parameters is described in Eqs. 4.31-4.34 in Chapter 4.

There are two cases that we need to consider: (1) the number of the labeled data is larger than the unlabeled, i.e. $l > N - l + 1$. As we discussed, the labeled data can be used to learn model parameters with maximum likelihood estimation. (2) then number of the labeled data is less than the unlabeled ones, i.e. $l < N - l + 1$. In this case, we can use the labeled instances to initialize the model, and then use it to learn GMMs on the unlabeled data.

Data: the training set consists of labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$ and iterations T
Result: output $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$
Initialize $\theta_0 = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$ on the labeled data $\mathcal{D}$ via Eqs. 5.4 - 5.6;
Set iteration index t = 0;
while $t < T$ **do**
  E-step: infer the posterior
  for *Each* $\mathbf{x}_j \in \mathcal{U}$ **do**
    for *Each* $k \in [1, K]$ **do**
      $p_{jk} = p(y_j = k|\mathbf{x}_j, \theta_t) \propto p(y_j = k)p(\mathbf{x}_j|y_j = k, \theta_t)$;
      $\implies p_{jk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_j, \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_j, \mu_k, \Sigma_k)}$;
    end
  end
  M-step: estimate $\theta_{t+1} = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$
  for *Each* $k \in [1, K]$ **do**
    $\mu_k = \frac{\sum_{j=l+1}^{N} p_{jk}\mathbf{x}_j}{\sum_{j=l+1}^{N} p_{jk}}$;
    $\Sigma_k = \frac{\sum_{j=l+1}^{N} p_{jk}(\mathbf{x}_j - \mu_k)(\mathbf{x}_j - \mu_k)^T}{\sum_{j=l+1}^{N} p_{jk}}$;
    $\pi_k = \frac{\sum_{j=l+1}^{N} p_{jk}}{N-l+1}$;
  end
  t++;
end

Algorithm 10: GMM for semi-supervised learning

Note that we update $\pi_k = \frac{\sum_{j=l+1}^{N} p_{jk}}{N-l+1}$ in the above algorithm with denominator $N - l + 1$, which is the number of unlabeled instances. This algorithm uses the labeled data to initialize the Gaussian mixture models (GMM), and then use it to infer the unlabeled data until it converges. Another way to understand it is that we use a small bunch of labeled the instances to learn the model parameters, and use it as the seed to label the unseen data (kind of propagating the label to nearby instances).

5.2 Graph-based semi-supervised learning

In this part, we will introduce Graph-based semi-supervised learning, which extends the graph-based clustering (spectral clustering) approaches in Chapter 4. In the following sections, we introduce several different graph-based semi-supervised learning algorithms. They differ in the choice of the loss function and the regularizer.

Given the training data $\mathcal{D}$ and $\mathcal{U}$, we can construct a graph, where the vertices are the labeled and unlabeled instances $\{\mathbf{x}_i\}_{i=1}^N$. The edge between two vertices $\mathbf{x}_i$ and $\mathbf{x}_j$ represents how similarity the two instances are. Let S be the edge weights and L be the unnormalized Laplacian matrix. For the definition of similarity graph and Laplacian matrix, please refer spectral clustering in Chapter 4.

For the unlabeled data, we can minimize Eq. 4.38 to find f for clustering analysis:

$$\hat{f} = \underset{f}{\operatorname{argmin}} f^T L f = \underset{f}{\operatorname{argmin}} \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 \tag{5.8}$$

where S_{ij} is the similarity matrix defined in a similar way as spectral clustering. For example, we can use Gaussian kernel (or Radial Basis Function kernel) below

$$S_{ij} = \exp\left(- \frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2} \right) \tag{5.9}$$

where σ is the standard deviation. If $\mathbf{x}_i = \mathbf{x}_j$, then the weight $S_{ij} = 1$; when $||\mathbf{x}_i - \mathbf{x}_j|| \to \infty$, then $S_{ij} \to 0$. If the edge exists between $\mathbf{x}_i$ and $\mathbf{x}_j$, then to minimize the objective in Eq. 5.8, we need to set the same label $f_i = f_j$ to these two points. If $\mathbf{x}_i$ and $\mathbf{x}_j$ is not connected, we have $S_{ij} = 0$ by definition. If we do not have any constraints, we can minimize Eq. 5.8 to solve f (labeling), which has been discussed in spectral clustering. If we have partially labeled data available, it becomes a semi-supervised learning problem.

In the following part, we will use the labeled data as hard constraints and soft constraints respectively to define objective function, and then estimate model parameters in an recursive updating manner.

5.2.1 Hard constraints

The hard constrained optimization means that all constraints of the labeled data must be satisfied when we minimize Eq. 5.8. Now, we have both labeled and unlabeled data, thus we can minimize the following constrained quadratic programming

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 \tag{5.10}$$
$$s.t.\ f(\mathbf{x}_i) = y_i$$

Take the derivative w.r.t. f_i in Eq. 5.10, and set it to zero

$$\frac{\partial \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2}{\partial f_i} = 0 \tag{5.11}$$
$$\implies \sum_j S_{ij}(f_i - f_j) = 0$$
$$\implies f_i = \frac{\sum_j S_{ij} f_j}{\sum_j S_{ij}} \tag{5.12}$$

where we can see that f_i is the weighted average of its neighbors. Eq. 5.12 is a harmonic function which has the same values as given labels on the labeled data, and satisfies the weighted average property on the unlabeled data.

Data: the training set consists of labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$ and iterations T
Result: output labeling: f
Initialize $f_i = y_i$ on the labeled data $\mathcal{D}$;
Construct the similarity matrix S ;
Set iteration index t = 0;
while $t < T$ **do**
  for *Each* $j = l + 1, ..., N$ **do**
    update f_j according to Eq. 5.12;
  end
  t++;
end

Algorithm 11: Graph-based semi-supervised learning (hard)

This algorithm is an iterative procedure to compute the harmonic function in Eq.

5.12. Initially, it sets $f(\mathbf{x}_i) = y_i$ for the labeled vertices $i = \{1, ..., l\}$, and some arbitrary value for the unlabeled vertices. Then it repeats to update all unlabeled vertices until convergence. This procedure is also called label propagation, because it propagates labels from the labeled vertices (which are fixed) gradually through the edges to all the unlabeled vertices.

5.2.2 Soft constraints

As discussed, the soft constrained approach here does not require the constraints of the labeled vertices in Eq. 5.10 is always satisfied. Then, we can add a weighed constraint to Eq. 5.8 as follows

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 + \frac{\lambda}{2} \sum_{i=1}^{l} (f(\mathbf{x}_i) - y_i)^2 \tag{5.13}$$

where λ is the coefficient to specify the weight of the labeled constraint $\sum_{i=1}^{l}(f(\mathbf{x}_i) - y_i)^2$. If $\lambda \to \infty$, then it has the same objective as in Eq. 5.10. Analogically, take the derivative w.r.t. f_i in Eq. 5.15, and set it to zero

$$\begin{aligned}
&\frac{\partial \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 + \frac{\lambda}{2} \sum_{i=1}^{l} (f(\mathbf{x}_i) - y_i)^2}{\partial f_i} = 0 \\
&\implies \sum_j S_{ij}(f_i - f_j) + \lambda(f_i - y_i) = 0 \\
&\implies (\sum_j S_{ij} + \lambda) f_i = \lambda y_i + \sum_{j=1}^{N} S_{ij} f_j \\
&\implies f_i = \frac{\sum_j S_{ij} f_j + \lambda y_i}{\sum_j S_{ij} + \lambda}
\end{aligned} \tag{5.14}$$

Note that for $i = \{1, ..., l\}$, the constraints exist, so we need to update f_i according to Eq. 5.14. While for $i = \{l + 1, ..., N\}$, there is no constraint, so we update f_i according to Eq. 5.12.

The algorithm 13 iteratively updates f_i for each vertice $\mathbf{x}_i$ until convergence. For the labeled vertices, we use Eq. 5.14, which is the solution with soft constraints. While for unlabeled vertices, there is no constraint, thus we still use the same formula in Eq. 5.12. When $\lambda \to \infty$, it become the hard constraint with $f(\mathbf{x}_i) = y_i$. Thus, the algo-

rithm with soft constraints is generally enough to handle different cases, by setting the weight λ.

Data: the training set consists of labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$ and iterations T
Result: output labeling: f
Initialize $f_i = y_i$ on the labeled data $\mathcal{D}$;
Construct the similarity matrix S ;
Set iteration index t = 0;
while $t < T$ **do**
  for *Each* $j = 1, ..., l$ **do**
    update f_j according to Eq. 5.14;
  end
  for *Each* $j = l + 1, ..., N$ **do**
    update f_j according to Eq. 5.12;
  end
  t++;
end

Algorithm 12: Graph-based semi-supervised learning (soft)

Manifold regularization: manifold regularization is to incorporate smooth constraint to f in Eq. 5.14. It is an inductive learning algorithm by defining f in the whole feature space: $f : X \mapsto R$. f is regularized to be smooth with respect to the graph by the graph Laplacian as in Eq. 5.14. However, this regularizer alone only controls f, namely the value of f on the $\mathcal{D} + \mathcal{U}$ training instances. To prevent f from being too wiggly (and thus having inferior generalization performance) outside the training samples, it is necessary to include a second regularization term, such as $||f||^2 = \int f(x)^2 dx$. Putting them together, the objective for manifold regularization becomes

$$\hat{f} = \operatorname*{argmin}_f \frac{1}{2} \sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 + \frac{\lambda_1}{2}||f||^2 + \frac{\lambda_2}{2} \sum_{i=1}^{l} (f(\mathbf{x}_i) - y_i)^2 \tag{5.15}$$

The so-called representer theorem guarantees that the optimal f admits a finite ($\mathcal{D} + \mathcal{U}$, to be exact) dimensional representation. There exist efficient algorithms to find the optimal f.

For unnormalized similarity matrix, we have discussed in spectral clustering in

Chapter 4,

$$\frac{1}{2}\sum_{i,j=1}^{N} S_{ij}(f_i - f_j)^2 = f^T \mathbf{L} f \tag{5.16}$$

Because the eigenvectors $\phi_1, ..., \phi_N$ of $\mathbf{L}$ form a basis in $\mathbb{R}^N$, thus f can be factorized as

$$f = \sum_i \alpha_i \phi_i \tag{5.17}$$

Thus, with simple algebra operations, we have

$$\begin{aligned} f^T \mathbf{L} f &= f^T \Phi^T \Lambda \Phi f \\ &= (\Phi f)^T \Lambda \Phi f \\ &= \sum_i \lambda_i \alpha_i^2 \end{aligned} \tag{5.18}$$

where $\Phi = [\phi_1, ..., \phi_N]$ and Λ is the diagonal matrix with eigenvectors $\{\lambda_1, ..., \lambda_N\}$. Notice that we plug Eq. 5.17 into Eq. 5.18 to get the final representation respect to α_i, for $i = [1, N]$.

Thus, to minimize Eq. 5.15, we also need to minimize the smooth term in Eq. 5.18. This indicates that the graph regularization term $f^T \mathbf{L} f$ prefers f only uses smooth basis with small λ_i.

5.3 Transductive support vector machines

Transductive support vector machines (TSVMs) [34] was first proposed by Joachims for text classification. Especially for very small training sets, TSVMs can substantially improve the already excellent performance of SVMs for text classification.

Before we talk about transductive support vector machines, let's revisit support vector machines (SVM) with hinge loss for the linear separable case. Given the labeled data $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^l$,

$$(\mathbf{w}, b) = \underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \tag{5.19}$$

$$s.t.\ y_i(\mathbf{w}^T\mathbf{x}+b) \geq 1-\xi_i \tag{5.20}$$

We can rewrite Eq. 5.19 as follows

$$(\mathbf{w},b) = \underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \{1-y_i(\mathbf{w}^T\mathbf{x}+b)\}_+ \tag{5.21}$$

where $\{x\}_+$ is x if $x > 0$, otherwise it is zero.

To extend SVM to handle both labeled and unlabeled data, we need to define a loss function, which minimizes the classification error. More specifically, for the unable training data $\mathcal{U}$, we can treat the label of each instance $\mathbf{x}$ as latent variable, and the purpose is to find a model fits the training data best. Given both the labeled and unlabeled data, the transductive learner aims to select a function h from a hypothesis space $\mathcal{H}$, so that the expected number of erroneous predictions on the unlabeled examples is minimized,

$$R = \int \frac{1}{N}\sum_{i=l+1}^{N} \ell(h(\mathbf{x}_i), y_i) dP(x_{l+1}, y_{l+1}) \cdots dP(x_N, y_N) \tag{5.22}$$

where $\ell(h(\mathbf{x}), y)$ is zero if $h(\mathbf{x}) = y$, otherwise it is one.

Specifically, for the linear separable case, the transductive SVM minimizes over

$$\begin{aligned}(y^*_{l+1}, ..., y^*_N, \mathbf{w}, b) = \arg \min_{y_{l+1},...,y_N,\mathbf{w},b} \frac{1}{2}||w||^2 \\ s.t.\ \forall^l_{i=1} : y_i[w^T x + b] \geq 1 \\ \forall^N_{j=l+1} : y^*_j[\mathbf{w}^T\mathbf{x}_j + b] \geq 1\end{aligned} \tag{5.23}$$

In other words, we attempt to find a labeling $y^*_{l+1}, ..., y^*_N$ of the unlabeled data $\mathbf{x}_{l+1}, ..., \mathbf{x}^*_N$ and the mapping model $\{\mathbf{w}, b\}$, so that this hyperplane can separates both labeled and unlabeled data with maximum margin. For nonlinear separable case, transductive SVMs takes a soft margin by minimizing over

$$\begin{aligned}(y^*_{l+1}, ..., y^*_N, \mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i + C^*\sum_{j=l+1}^{N}\xi^*_j \\ s.t.\ \forall^l_{i=1} : y_i[w^T x + b] \geq 1\end{aligned} \tag{5.24}$$

$$\forall_{j=l+1}^{N} : y_j^*[\mathbf{w}^T\mathbf{x}_j + b] \geq 1$$
$$\forall_{i=1}^{l} : \xi_i \geq 0$$
$$\forall_{j=l+1}^{N} : \xi_j^* \geq 0$$

where C and C^* are parameters set by the user, and y_j^* is the inferred label for $j = \{l+1, ..., N\}$ from all unlabeled data. Because we always assume the putative label y_j^*, so the unlabeled data $\mathbf{x}_j$ is always correctly classified. As for the hinge loss, the instance which are far away from decision boundary (i.e. $\mathbf{x}_j$ with $\mathbf{w}^T\mathbf{x}_j + b \geq 0$) incurs no loss. It only penalizes the unlabeled instances with $-1 < \mathbf{w}^T\mathbf{x}_j + b < 1$, which are within the margin.

Rewrite Eq. 5.24 into follows with hinge loss

$$\begin{aligned}(y_{l+1}^*, ..., y_N^*, \mathbf{w}, b) = \arg \min_{(y_{l+1}^*, ..., y_N^*, \mathbf{w}, b)} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\{1 - y_i(\mathbf{w}^T\mathbf{x} + b)\}_+ \\ & + C^*\sum_{j=l+1}^{N}\{1 - y_j^*(\mathbf{w}^T\mathbf{x} + b)\}_+ \end{aligned} \tag{5.25}$$

For the binary case,

$$y_j^*(\mathbf{w}^T\mathbf{x} + b) = sign(\mathbf{w}^T\mathbf{x} + b)(\mathbf{w}^T\mathbf{x} + b) = |\mathbf{w}^T\mathbf{x} + b| \tag{5.26}$$

Thus, plugging Eq. 5.26 back into the function 5.25, we can get

$$\begin{aligned}(y_{l+1}^*, ..., y_N^*, \mathbf{w}, b) = \arg \min_{(y_{l+1}^*, ..., y_N^*, \mathbf{w}, b)} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\{1 - y_i(\mathbf{w}^T\mathbf{x} + b)\}_+ \\ & + C^*\sum_{j=l+1}^{N}\{1 - |\mathbf{w}^T\mathbf{x} + b|\}_+ \end{aligned} \tag{5.27}$$

Learning: considering that we can compute the (sub)gradient w.r.t. $\mathbf{w}$ in the objective function 5.27, we can use gradient descent to estimate the model parameters.

$$\frac{\partial \mathcal{L}(y_{l+1}^*, ..., y_N^*, \mathbf{w}, b)}{\partial \mathbf{w}} = \mathbf{w} - C \sum_{\substack{i=1 \\ y_i(\mathbf{w}^T\mathbf{x}+b)<1}}^{l} y_i\mathbf{x}_i - C^* \sum_{\substack{j=l+1 \\ y_j^*(\mathbf{w}^T\mathbf{x}+b)<1}}^{N} y_j^*\mathbf{x}_j \tag{5.28}$$

Note that the derivative w.r.t. $\mathbf{w}$ only depends on the misclassified instances. Then we can update $\mathbf{w}$ with

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial \mathcal{L}(y^*_{l+1}, ..., y^*_N, \mathbf{w}, b)}{\partial \mathbf{w}} \tag{5.29}$$

where α is the learning rate.

There is one practical consideration. Empirically, it is sometimes observed that the solution to Eq. 5.24 is imbalanced. That is, the majority (or even all) of the unlabeled instances are predicted in only one of the classes. The reason for such behavior is not well-understood. To correct for the imbalance, one heuristic is to constrain the predicted class proportion on the unlabeled data, so that it is the same as the class proportion on the labeled data:

The user can specify the ratio or the number of unlabeled data to be assigned into positive class.

The heuristic approach [34] optimizes the following objective

$$(y^*_{l+1}, ..., y^*_N, \mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{l} \xi_i + C^*_- \sum_{j:y^*_j=-1}^{N} \xi^*_j + C^*_+ \sum_{j:y^*_j=1}^{N} \xi^*_j \tag{5.30}$$

$$\begin{aligned} s.t.\ & \forall^l_{i=1} : y_i[w^T x + b] \geq 1 \\ & \forall^N_{j=l+1} : y^*_j[\mathbf{w}^T \mathbf{x}_j + b] \geq 1 \\ & \forall^l_{i=1} : \xi_i \geq 0 \\ & \forall^N_{j=l+1} : \xi^*_j \geq 0 \end{aligned}$$

Data: the training set consists of labeled $\mathcal{D}$ and unlabeled $\mathcal{U}$
parameter: C, C^*, $ratio_+$: the ratio between the number of unlabeled instances to be assigned to positive class over to that of negative ones
output : the model $\{\mathbf{w}, b\}$ and labeling: $y^*_{l+1}, ..., y^*_N$
Initialize the model $(\mathbf{w}, b)$ with labeled training data $\mathcal{D}$;
Classify the unlabeled examples using the model $(\mathbf{w}, b)$, and select the top highest $(N - l + 1) * ratio_+$ instances to be in positive class, and the remaining into negative class;
$C^*_- = 10^{-5}$;
$C^*_+ = 10^{-5} * ratio$;
while $C^*_- < C^* || C^*_+ < C^*$ **do**
  $(\mathbf{w}, b, \xi, \xi^*)$ = optimize objective function 5.30;
  while $\exists_{m,n} : (y_m * y_n < 0) \& (\xi^*_m > 0) \& (\xi^*_n > 0) \& (\xi^*_m + \xi^*_n > 2)$ **do**
    $y^*_m = -y^*_m$;
    $y^*_n = -y^*_n$;
    $(\mathbf{w}, b, \xi, \xi^*)$ = optimize objective function 5.30;
  end
  $C^*_- = min(2C^*_-, C^*)$;
  $C^*_+ = min(2C^*_+, C^*)$;
end

Algorithm 13: TSVM: semi-supervised learning (soft)

It learns the model parameters with training an inductive SVM on the labeled data $\mathcal{D}$, and classify the unlabeled data accordingly. Then, it uniformly increase the influence of the test examples, by increasing the cost-factors C^*_- and C^*_+ upper to the user defined value of C^*. The algorithm uses the unbalanced cost C^*_- and C^*_+ to better accommodate the user defined ratio. For any two examples, which violates the condition of will be switched labels to decrease the objective function. For convergence, please refer to [34].

5.4 Semi-supervised clustering

Semi-supervised clustering is an very important topic in machine learning and data mining. The key challenge of this problem is how to learn a metric, such that the

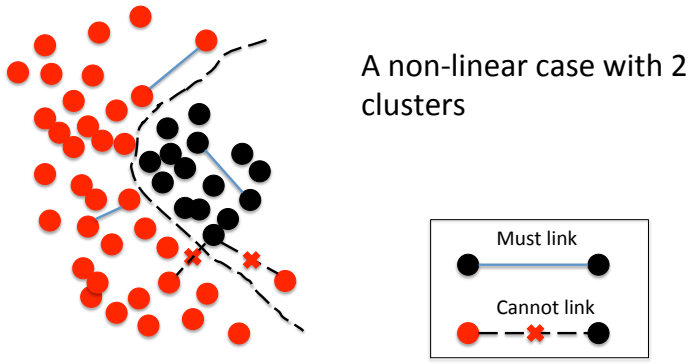


Figure 5.1. The graph shows data points belonging to two clusters. However, the clustering boundary is nonlinear and we will cannot improve clustering performance if we simply use GMMs to model it. Thus, we introduce the must-link and cannot-link, which can be used to guide the model to find the boundary.

instances sharing the same label are more likely close to each other on the embedded space. Especially for the nonlinear case, we need to find a mapping, which can best cluster the data with high precision. There is an emerging interest in semi-supervised clustering algorithms in the machine learning and data mining communities. The recent advances in the maximum margin clustering with pairwise constraints have attracted great attention in machine learning community. In addition to the data values, we assume there are a number of instance-level constraints on cluster assignment. More specially, we consider the following two types of pairwise relations in Fig. 5.1:

- Must-link constraints specify that two samples should be assigned into one cluster.
- Cannot-link constraints specify that two samples should be assigned into different clusters.

In this section, we will introduce the transductive semi-supervised maximum margin clustering, with deep features learned simultaneously in an unified framework.

5.4.1 Overview of the approach

Let $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$ ($\mathbf{x}_i \in \mathbb{R}^d$) be a set of N examples, which belongs to K clusters called $\mathcal{Z}$. In addition to the unlabeled data, there is additional partially labeled data in the form of pairwise constraints $C = \{(\mathbf{x}_i, \mathbf{x}_j, \delta(z_i = z_j)\}$, which is a kind of side information provided whether the two instances $(\mathbf{x}_i, \mathbf{x}_j)$ are from the same cluster or not (indicated by the δ function). The semi-supervised clustering attempts to learn weights $\mathbf{w}^k \in \mathbb{R}^D$, for each cluster $k = [1, K]$, to make these constraints satisfied as much as possible.

Suppose that we are interesting in learning a linear mapping (or Mahalanobis metric) [35, 36]. Then given the learned model $\mathbf{w}^k$, for $k = \{1, ..., K\}$, we use it to partition the data. Just like the multi-class classification problems [6], we use the joint feature representation $\Phi(\mathbf{x}, z)$ for each $(\mathbf{x}, z) \in \mathcal{X} \times \mathcal{Z}$

$$\Phi(\mathbf{x}, z) = \begin{bmatrix} \mathbf{x} \cdot \delta(z = 1) \\ \cdots \\ \mathbf{x} \cdot \delta(z = K) \end{bmatrix} \tag{5.31}$$

where δ is the indicator function (1 if the equation holds, otherwise 0). Correspondently, the hyperplanes for the K clusters can be parameterized by the weight vector $\mathbf{W} \in \mathbb{R}^{(K \times d) \times 1}$, which is the concatenation of weights $\mathbf{w}^k$, for $k = \{1, ..., K\}$. In other words, $\mathbf{W}[(k-1) \times d + 1 : k \times d] = \mathbf{w}^k$. The clustering of testing examples is done in the same manner as the multiclass SVM [6],

$$\max_{z \in [1,K]} \mathbf{W}^T \Phi(\mathbf{x}, z) \tag{5.32}$$

For inference, we first project data with joint feature mapping according to Eq. 5.31, then do clustering analysis via Eq. 5.32. In the following part, we will introduce how to learn the weight parameter $\mathbf{W}$ with maximum margin techniques.

5.4.2 Objective function

We propose the semi-supervised deep transductive machine for clustering, by leveraging transductive learning and deep learning under the maximum margin frame-

work.

In a similar manner as in [37, 38], we will incorporate the pairwise constraint information into our margin-based clustering framework. In addition, we leverage the unlabeled data to separate clusters with large margins, by following transductive learning. Specifically, given the pairwise constraint set $C = \{(\mathbf{x}_i, \mathbf{x}_j, \delta(z_i = z_j))\}$, we minimize the following transductive semi-supervised objective function

$$\min_{\mathbf{W}} \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{n^+}\sum_i \eta_i^+ + \frac{1}{n^-}\sum_j \eta_j^- + \frac{\beta}{UK}\sum_{i\in U}\xi_i \tag{5.33}$$

$s.t.$

$$\forall s_{i1}, s_{i2} \in \mathcal{Z}, s_{i1} \neq s_{i2}; \text{if } (\mathbf{x}_{i1}, \mathbf{x}_{i2}, \delta(z_{i1}, z_{i2})) \in C^+$$

$$\max_{z_{i1}=z_{i2}} \mathbf{W}^T\Phi(\mathbf{x}_{i1}, \mathbf{x}_{i2}, z_{i1}, z_{i2}) - \mathbf{W}^T\Phi(\mathbf{x}_{i1}, \mathbf{x}_{i2}, s_{i1}, s_{i2}) \geq 1 - \eta_i, \eta_i \geq 0 \tag{5.34}$$

$$\forall s_{j1}, s_{j2} \in \mathcal{Z}, s_{j1} = s_{j2}; \text{if } (\mathbf{x}_{j1}, \mathbf{x}_{j2}, \delta(z_{j1}, z_{j2})) \in C^-$$

$$\max_{z_{j1}\neq z_{j2}} \mathbf{W}^T\Phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, z_{j1}, z_{j2}) - \mathbf{W}^T\Phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, s_{j1}, s_{j2}) \geq 1 - \eta_j, \eta_j \geq 0 \tag{5.35}$$

$$\forall i \in U, \forall s_i \neq z_i \in \mathcal{Z}$$

$$\max_{z_i} \mathbf{W}^T\Phi(\mathbf{x}_i, z_i) - \mathbf{W}^T\Phi(\mathbf{x}_i, s_i) \geq 1 - \xi_i \tag{5.36}$$

where $\mathbf{W}$ is the clustering weight over the learned feature space, η_i^+ and η_j^- are non-negative slack variables, and $C^+ = \{(\mathbf{x}_i, \mathbf{x}_j, \delta(z_i = z_j))|z_i = z_j\}$ are the same label pairs, with the total number of pairwise constraints $n^+ = |C^+|$, $C^- = \{(\mathbf{x}_i, \mathbf{x}_j, \delta(z_i = z_j))|z_i \neq z_j\}$ are different-label pairs, with $n^- = |C^-|$. U is the number of the unlabeled data (instances), not belong to any pairwise constraints. For convenience, we define $\Phi(\mathbf{x}_i, \mathbf{x}_j, z_i, z_j) = \Phi(\mathbf{x}_i, z_i) + \Phi(\mathbf{x}_j, z_j)$, which means the mapping of a pairwise constraint as the sum of the individual example-label mappings.

Eqs. 5.34 and 5.35 specify the conditions that need to be satisfied, which means that the score for the most possible assigning scheme satisfying the constraints should be greater than that for any other assigning scheme with large margins. More specifically, for any pair $(\mathbf{x}_i, \mathbf{x}_j, 1) \in C^+$, it requires that the largest score for assigning $(\mathbf{x}_i, \mathbf{x}_j)$ into the same cluster should be greater than that for assigning the pair into

different clusters by at least 1 (soft margin can be applied here too). Analogously, for any dissimilar pair $(\mathbf{x}_i, \mathbf{x}_j, 0) \in C^-$, the score that they are assigned into the most two different clusters should be greater than that for partitioning them into the same cluster.

Eq. 5.36 is from the principles of transductive learning, which indicates that the score of the most assigned cluster label is greater by at least 1 than that of the runner up from the rest of the clusters.

The constrained optimization problem in Eq. 5.33 is hard to solve because the first inequality Eq. 5.34 and the second inequality Eq. 5.35 impose all the possible combinations of two clusters for each pairwise constraint. Thus, we transform it into the following equivalent unconstrained function which it is generally easier to solve

$$
\begin{aligned}
&\min_{\mathbf{W},\boldsymbol{\Theta}} \frac{\lambda}{2}||\mathbf{W}||^2 \\
&+ \frac{1}{n^+}\Bigg\{1 - \Bigg[\max_{\substack{z_{i1}=z_{i1} \\ (\mathbf{x}_{i1},\mathbf{x}_{i2},1)\in C^+}} \mathbf{W}^T\Phi(\mathbf{x}_{i1},\mathbf{x}_{i2},z_{i1},z_{i2}) - \\
&\qquad \max_{s_{i1}\neq s_{i1}} \mathbf{W}^T\Phi(\mathbf{x}_{i1},\mathbf{x}_{i2},s_{i1},s_{i2})\Bigg]\Bigg\}_+ \qquad (5.37a) \\
&+ \frac{1}{n^-}\Bigg\{1 - \Bigg[\max_{\substack{z_{j1}\neq z_{j2} \\ (\mathbf{x}_{j1},\mathbf{x}_{j2},0)\in C^-}} \mathbf{W}^T\Phi(\mathbf{x}_{j1},\mathbf{x}_{j2},z_{j1},z_{j2}) - \\
&\qquad \max_{s_{j1}=s_{j2}} \mathbf{W}^T\Phi(\mathbf{x}_{j1},\mathbf{x}_{j2},s_{j1},s_{j2})\Bigg]\Bigg\}_+ \qquad (5.37b) \\
&+ \frac{\beta}{UK}\sum_{i\in U}\left\{1 - [\max_{z_i} \mathbf{W}^T\Phi(\mathbf{x}_i,z_i) - \max_{s_i\neq z_i} \mathbf{W}^T\Phi(\mathbf{x}_i,s_i)]\right\}_+ \qquad (5.37c)
\end{aligned}
$$

where $\{x\}_+ = max(x, 0)$. The formula 5.37a specifies the condition that need to be satisfied for the same label pairwise constrains, while formula 5.37b denotes the conditions for different-label pairs. The last equation is corresponding to transductive constraints in Eq. 5.33.

In the objective function, we need to estimate the parameters, the clustering weight **W**.To minimize the objective function, we can compute the gradients w.r.t. **W**, and gradient-based methods can be used to optimize it. The objective function is not convex anymore, and we can only find a local minimum. In practice, we can use L-BFGS

or gradient descent to optimize the objective. In this section, we compute the gradients w.r.t. $\mathbf{W}$ and $\{\mathbf{w}_l\}_{l=1}^{L}$ respectively, and then use stochastic gradient descent to optimize the objective function.

5.4.3 Parameter learning

To minimize the objective function, we use the gradient descent to update all parameters. We learn the parameters in an alternating manner: (1) inference or label assignment, given the model parameters; (2) and then update model parameters with gradient descent. To compute the gradients of the parameters, we need to find the most violated constraints first. For the same label pairs, we have the following most violated set:

$$\begin{aligned} A^+ = \Big\{ & (\mathbf{x}_i, \mathbf{x}_j, \delta(z_{i1} = z_{i2})) \in C^+ | \\ & \max_{z_{i1}=z_{i1}} \mathbf{W}^T \Phi(\mathbf{x}_{i1}, \mathbf{x}_{i2}, z_{i1}, z_{i2}) \\ & - \max_{s_{i1} \neq s_{i1}} \mathbf{W}^T \Phi(\mathbf{x}_{i1}, \mathbf{x}_{i2}, s_{i1}, s_{i2}) < 1 \Big\} \end{aligned} \tag{5.38}$$

For the different-label pairs, we denote the most violated set as

$$\begin{aligned} A^- = \Big\{ & (\mathbf{x}_i, \mathbf{x}_j, \delta(z_{j1} = z_{j2})) \in C^- | \\ & \max_{z_{j1} \neq z_{j2}} \mathbf{W}^T \Phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, z_{j1}, z_{j2}) \\ & - \max_{s_{j1}=s_{j2}} \mathbf{W}^T \Phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, s_{j1}, s_{j2}) < 1 \Big\} \end{aligned} \tag{5.39}$$

Then, we compute the gradient w.r.t. $\mathbf{W}$

$$\begin{aligned} d\mathbf{W} = & \lambda \mathbf{W} + \\ & - \frac{1}{n^+} \sum_{(\mathbf{x}_{i1}, \mathbf{x}_{i2}, 1) \in A^+} \Big[\Phi(\mathbf{x}_{i1}, \mathbf{x}_{i2}, z_{i1}^+, z_{i2}^+) - \Phi(\mathbf{x}_{i1}, \mathbf{x}_{i2}, z_{i1}^-, z_{i2}^-) \Big] \\ & - \frac{1}{n^-} \sum_{(\mathbf{x}_{j1}, \mathbf{x}_{j2}, 0) \in A^-} \Big[\Phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, z_{j1}^-, z_{j2}^-) - \Phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, z_{j1}^+, z_{j2}^+) \Big] \end{aligned}$$

$$-\sum_{i\in U}\frac{\beta}{UK}\Big[\Phi(\mathbf{x}_i,z_i^+)-\Phi(\mathbf{x}_i,s_i^+)\Big], \tag{5.40}$$

where

$$(z_{i1}^+,z_{i2}^+)=\max_{z_{i1}=z_{i2}}\mathbf{W}^T\Phi(\mathbf{x}_{i1},\mathbf{x}_{i2},z_{i1},z_{i2}),$$
$$(z_{i1}^-,z_{i2}^-)=\max_{z_{i1}\neq z_{i1}}\mathbf{W}^T\Phi(\mathbf{x}_{i1},\mathbf{x}_{i2},z_{i1},z_{i2});$$

and for the unlabeled set $z_i^+=\max_{z_i}\Phi(\mathbf{x}_i,z_i)$ and $s_i^+=\max_{s_i\neq z_i^+}\Phi(\mathbf{x}_i,s_i)$

Note that for each pair $(\mathbf{x}_i,\mathbf{x}_j)$, if it violates the constraints in Eqs. 5.34 and 5.35, then we can compute the gradient w.r.t. $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively, which will be used to calculate the gradients $\mathbf{W}$. We use $\mathbf{x}=[\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_N]$ as the concatenation of all the instances, where $\mathbf{x}\in\mathcal{R}^{d\times N}$, with each column $\mathbf{x}(:,i)=\mathbf{x}_i$.

Data: the training data $\mathcal{X}$, pairwise constraints C, the number of clusters K, the number of iterations T, λ, and β;
Result: model parameters $\mathbf{W}$
Initialize $\mathbf{W}$ in the latent space;
for $i=1;i<=T;i++$ **do**
  if the objective in Eq. 5.37 has no significant changes, break;
  find the most violated constrains according to Eqs. 5.38 and 5.39;
  compute the gradient w.r.t. $\mathbf{W}$ via Eq. 5.40;
  update the parameters with gradient descent via Eq. 5.41;
end
Return model weight $\mathbf{W}$, as well as average accuracy;

Algorithm 14: maximum margin clustering with pairwise constraints

As for the clustering weight $\mathbf{W}$, we take a similar strategy as in [38] to initialize it. Then we compute the sub-gradient w.r.t. model parameters from the objective function in Eq. 5.37, and optimize the whole network with gradient descent.

Parameter updating: After pre-training, we take a similar strategy as in [38] to initialize the clustering weight $\mathbf{W}$. To optimize our objective function, we first infer the most violated constraints from pairwise and transductive information. With the learned features as inputs in the maximum margin framework, then we can derive the gradient w.r.t. the top layer clustering weight. Meanwhile, we can compute the gradient w.r.t. to the features, as well as the gradients w.r.t. the low level weights

through backpropagation. Finally, given the (sub)gradients w.r.t. all parameters in Eq. 5.37, we can optimize the whole network with gradient descent.

We also tried L-BFGS [39, 40] to update model parameters, but it does not perform well. In our model, we can update the model parameters as follows,

$$\mathbf{W} \leftarrow \mathbf{W} - \gamma_{\mathbf{W}} d\mathbf{W} \tag{5.41}$$

where $\gamma_{\mathbf{W}}$ is the learning rate for the clustering weight $\mathbf{W}$. For more details, refer to algorithm 15. After we learned the model parameters, we can do cluster analysis according to Eq. 5.32.

5.5 Semi-supervised deep metric learning

Previously, we have talked about linear metric learning via Eq. 5.33. In this section, we will consider to leverage deep learning to learn a nonlinear metric for semi-supervised clustering. Instead of learning a linear mapping or Mahalanobis metric [35, 36], we are interested in a non-linear mapping function. To make it easy to understand, suppose we have learned a nonlinear mapping function $f : \mathbb{R}^D \rightarrow \mathbb{R}^d$. Then, for each instance $\mathbf{x} \in \mathcal{X}$, we can get its embedding code $\mathbf{h} = f(\mathbf{x}) \in \mathcal{H}$ (note that the pairwise constrains also are kept in the coding space). Then given the learned features $\mathbf{h}$, we leverage semi-supervised maximum margin clustering to partition the data. The clustering of testing examples is done in the same manner as the multiclass SVM [6],

$$\max_{z \in [1,K]} \mathbf{W}^T \Phi(f(\mathbf{x}), z) \tag{5.42}$$

where $\mathbf{W} \in \mathbb{R}^{(K \times d) \times 1}$, which is the concatenation of weights $\mathbf{w}^k$, for $k = \{1, ..., K\}$. In other words, $\mathbf{W}[(k-1) \times d + 1 : k \times d] = \mathbf{w}^k$.

5.5.1 Overall approach

Instead of operating over the raw data, we first project the input $\mathcal{X}$ into embedded space (note that constraints are kept) and then apply the clustering. Specifically, we will use deep neural network (DNN) to pretrain the deep structure (with stacked RBMs). As mentioned before, $\mathbf{h} \in \mathbb{R}^d$ is the mapping code with function f, which is

non-linear mapping defined with L-layers neural network, s.t.

$$\mathbf{h}_i = f(\mathbf{x}_i) = \underbrace{f_L \circ f_{L-1} \circ \cdots \circ f_1}_{L \text{ times}}(\mathbf{x}_i) \tag{5.43}$$

where $\circ$ indicates the function composition, and f_l is logistic function or other non-linear mapping with the weight parameter $\boldsymbol{\theta}_l$ respectively for each layer $l = \{1, .., L\}$, refer further to Sec. 5.5.2 for more details. With a little abuse of symbols, for any input $\mathbf{x}$, If we denote the output of the l-th layer as $f_{1\to l}(\mathbf{x})$, then we can get $\mathbf{h} = f_{1\to L}(\mathbf{x})$.

In a similar manner as in Eq. 5.33, we minimize the following transductive semi-supervised objective function

$$\begin{aligned}
&\min_{\mathbf{W},\boldsymbol{\Theta}} \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{n^+}\sum_i \eta_i^+ + \frac{1}{n^-}\sum_j \eta_j^- + \frac{\beta}{UK}\sum_{i\in U}\xi_i \qquad (5.44)\\
&s.t.\\
&\forall s_{i1}, s_{i2} \in \mathcal{Z}, s_{i1} \neq s_{i2}; \text{if } (\mathbf{h}_{i1}, \mathbf{h}_{i2}, \delta(z_{i1}, z_{i2})) \in C^+\\
&\max_{z_{i1}=z_{i2}} \mathbf{W}^T\Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, z_{i1}, z_{i2}) -\\
&\qquad \mathbf{W}^T\Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, s_{i1}, s_{i2}) \geq 1 - \eta_i, \eta_i \geq 0\\
&\forall s_{j1}, s_{j2} \in \mathcal{Z}, s_{j1} = s_{j2}; \text{if } (\mathbf{h}_{j1}, \mathbf{h}_{j2}, \delta(z_{j1}, z_{j2})) \in C^-\\
&\max_{z_{j1}\neq z_{j2}} \mathbf{W}^T\Phi(\mathbf{h}_{j1}, \mathbf{h}_{j2}, z_{j1}, z_{j2}) -\\
&\qquad \mathbf{W}^T\Phi(\mathbf{h}_{j1}, \mathbf{h}_{j2}, s_{j1}, s_{j2}) \geq 1 - \eta_j, \eta_j \geq 0\\
&\forall i \in U, \forall s_i \neq z_i \in \mathcal{Z}\\
&\max_{z_i} \mathbf{W}^T\Phi(\mathbf{h}_i, z_i) - \mathbf{W}^T\Phi(\mathbf{h}_i, s_i) \geq 1 - \xi_i
\end{aligned}$$

where $\mathbf{W}$ is the clustering weight over the learned feature space, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_l\}_{l=1}^L$ are the weights for each layer in the deep architecture, η_i^+ and η_j^- are non-negative slack variables, and $\mathbf{h}_i$ is the mapping code from $\mathbf{x}_i$ via Eq. 5.43; $C^+ = \{(\mathbf{h}_i, \mathbf{h}_j, \delta(z_i = z_j))|z_i = z_j\}$ are the same label pairs, with the total number of pairwise constraints $n^+ = |C^+|$, $C^- = \{(\mathbf{h}_i, \mathbf{h}_j, \delta(z_i = z_j))|z_i \neq z_j\}$ are different-label pairs, with $n^- = |C^-|$. U is the set of unlabeled data (instances), not belong to any pairwise constraints. For convenience, we define $\Phi(\mathbf{h}_i, \mathbf{h}_j, z_i, z_j) = \Phi(\mathbf{h}_i, z_i) + \Phi(\mathbf{h}_j, z_j)$, which means the mapping of a pairwise constraint as the sum of the individual example-label mappings. The

multi-layers non-linear mapping function f projects $\mathbf{x}_i$ into $\mathbf{h}_i$, for $i \in [1, N]$. Instead of a linear mapping, the advantage of using a deep network to parametrize the function f is that a multi-layer network is better at learning a non-linear function that is presumably required to collapse classes in the latent space, in particular when the data consists of very complex non-linear structures.

Similarly, we transform it into the following equivalent unconstrained function which it is generally easier to solve

$$\begin{aligned}
&\min_{\mathbf{W},\Theta} \frac{\lambda}{2}||\mathbf{W}||^2 \\
&+ \frac{1}{n^+}\sum_i \left\{1 - \left[\max_{\substack{z_{i1}=z_{i1} \\ (\mathbf{h}_{i1},\mathbf{h}_{i2},1)\in C^+}} \mathbf{W}^T\Phi(\mathbf{h}_{i1},\mathbf{h}_{i2},z_{i1},z_{i2}) \right.\right. \\
&\qquad \left.\left. - \max_{s_{i1}\neq s_{i1}} \mathbf{W}^T\Phi(\mathbf{h}_{i1},\mathbf{h}_{i2},s_{i1},s_{i2})\right]\right\}_+ \qquad (5.45a) \\
&+ \frac{1}{n^-}\sum_j \left\{1 - \left[\max_{\substack{z_{j1}\neq z_{j2} \\ (\mathbf{h}_{j1},\mathbf{h}_{j2},0)\in C^-}} \mathbf{W}^T\Phi(\mathbf{h}_{j1},\mathbf{h}_{j2},z_{j1},z_{j2}) \right.\right. \\
&\qquad \left.\left. - \max_{s_{j1}=s_{j2}} \mathbf{W}^T\Phi(\mathbf{h}_{j1},\mathbf{h}_{j2},s_{j1},s_{j2})\right]\right\}_+ \qquad (5.45b) \\
&+ \frac{\beta}{UK}\sum_{i\in U}\left\{1 - [\max_{z_i} \mathbf{W}^T\Phi(\mathbf{h}_i,z_i) - \max_{s_i\neq z_i} \mathbf{W}^T\Phi(\mathbf{h}_i,s_i)]\right\}_+ \qquad (5.45c)
\end{aligned}$$

where $\{x\}_+ = max(x, 0)$ and $\mathbf{h}_i$ is the projected code of $\mathbf{x}_i$ using Eq. 5.43. The formula 5.45a specifies the condition that need to be satisfied for the same label pairwise constrains, while formula 5.45b denotes the conditions for different-label pairs. The last equation is corresponding to transductive constraints in Eq. 5.44.

5.5.2 Parameter learning

To minimize the objective function, we use the gradient descent to update all parameters. We learn the parameters in an alternating manner: (1) data projection, given the model parameters; (2) and then update model parameters with gradient descent. To compute the gradients of the parameters, we need to find the most violated con-

straints first. For the same label pairs, we have the following most violated set:

$$A^+ = \Big\{ (\mathbf{h}_i, \mathbf{h}_j, \delta(z_{i1} = z_{i2})) \in C^+ | \\ \max_{z_{i1}=z_{i1}} \mathbf{W}^T \Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, z_{i1}, z_{i2}) \\ - \max_{s_{i1} \neq s_{i1}} \mathbf{W}^T \Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, s_{i1}, s_{i2}) < 1 \Big\} \tag{5.46}$$

For the different-label pairs, we denote the most violated set as

$$A^- = \Big\{ (\mathbf{h}_i, \mathbf{h}_j, \delta(z_{j1} = z_{j2})) \in C^- | \\ \max_{z_{j1} \neq z_{j2}} \mathbf{W}^T \Phi(\mathbf{h}_{j1}, \mathbf{h}_{j2}, z_{j1}, z_{j2}) \\ - \max_{s_{j1}=s_{j2}} \mathbf{W}^T \Phi(\mathbf{h}_{j1}, \mathbf{h}_{j2}, s_{j1}, s_{j2}) < 1 \Big\} \tag{5.47}$$

Analogously, we can compute the most violated set on the unlabeled instances. Finally, we can calculate the gradient w.r.t. $\mathbf{W}$ as follows

$$\begin{aligned} d\mathbf{W} &= \lambda \mathbf{W} + \\ &- \frac{1}{n^+} \sum_{(\mathbf{h}_{i1}, \mathbf{h}_{i2}, 1) \in A^+} \Big[\Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, z_{i1}^+, z_{i2}^+) - \Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, z_{i1}^-, z_{i2}^-) \Big] \\ &- \frac{1}{n^-} \sum_{(\mathbf{h}_{j1}, \mathbf{h}_{j2}, 0) \in A^-} \Big[\Phi(\mathbf{h}_{j1}, \mathbf{h}_{j2}, z_{j1}^-, z_{j2}^-) - \Phi(\mathbf{h}_{j1}, \mathbf{h}_{j2}, z_{j1}^+, z_{j2}^+) \Big] \\ &- \sum_{i \in U} \frac{\beta}{UK} \Big[\Phi(\mathbf{h}_i, z_i^+) - \Phi(\mathbf{h}_i, s_i^+) \Big], \end{aligned} \tag{5.48}$$

where $(z_{i1}^+, z_{i2}^+) = \max_{z_{i1}=z_{i2}} \mathbf{W}^T \Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, z_{i1}, z_{i2})$,
$(z_{i1}^-, z_{i2}^-) = \max_{z_{i1} \neq z_{i1}} \mathbf{W}^T \Phi(\mathbf{h}_{i1}, \mathbf{h}_{i2}, z_{i1}, z_{i2})$; and for the unlabeled set $z_i^+ = \max_{z_i} \Phi(\mathbf{h}_i, z_i)$ and $s_i^+ = \max_{s_i \neq z_i^+} \Phi(\mathbf{h}_i, s_i)$

In order to learn discriminative features, we also need to estimate the weights in the multi-layer network. Note that for each pair $(\mathbf{h}_i, \mathbf{h}_j)$, if it violates the constraints in Eqs. 5.45a and 5.45b, then we can compute the gradient w.r.t. $\mathbf{h}_i$ and $\mathbf{h}_j$ respectively, which will be used to calculate the gradients of $\boldsymbol{\theta}_l$ for $l \in [1, L]$ in the deep network.

We use $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_N]$ as the concatenation of all the hidden codes, where $\mathbf{H} \in \mathcal{R}^{d \times N}$, with each column $\mathbf{H}(:, i) = \mathbf{h}_i$.

For the positive pairs, we have

$$d\mathbf{h}_{i1} = -\frac{1}{n^+} \sum_{i1} [\mathbf{W}_{z_{i1}^+} - \mathbf{W}_{z_{i1}^-}] \tag{5.49a}$$

$$d\mathbf{h}_{i2} = -\frac{1}{n^+} \sum_{i2} [\mathbf{W}_{z_{i1}^+} - \mathbf{W}_{z_{i2}^-}] \tag{5.49b}$$

where $\mathbf{W}_{z_{i1}^+}$ indicates the weight vector corresponding to the cluster label z_{i1}^+ in the whole weight matrix $\mathbf{W}$. More specifically, $\mathbf{W}_{z_{i1}^+} = \mathbf{W}[(z_{i1}^+ - 1) \times d + 1 : z_{i1}^+ \times d]$

For the negative pairs, we can get

$$d\mathbf{h}_{j1} = -\frac{1}{n^-} \sum_{j1} [\mathbf{W}_{z_{j1}^-} - \mathbf{W}_{z_{j1}^+}] \tag{5.50a}$$

$$d\mathbf{h}_{j2} = -\frac{1}{n^-} \sum_{j2} [\mathbf{W}_{z_{j2}^-} - \mathbf{W}_{z_{j1}^+}] \tag{5.50b}$$

For the unlabeled instances, we have

$$d\mathbf{h}_i = -\frac{\beta}{UK} \sum_{i} [\mathbf{W}_{z_i^+} - \mathbf{W}_{s_i^+}] \tag{5.51}$$

Given the gradient of $d\mathbf{h}_i$ for each hidden code, we can get the gradient w.r.t. $\mathbf{H}$ as

$$d\mathbf{H}(:, i) = d\mathbf{h}_i \tag{5.52}$$

where $d\mathbf{h}_i$ can be calculated according to Eqs. 5.49 and 5.50. Then, we can calculate the gradients w.r.t. lower level weights with back-propagation. For example $d\boldsymbol{\theta}_L = d\mathbf{H} \times \big(f_{1 \to L}(\mathcal{X}) \cdot (1 - f_{1 \to L}(\mathcal{X}))\big)$, where $\times$ represents matrix multiplication, and $\cdot$ indicates pointwise product.

Initialization: We used stacked RBMs to initialize the weights layer by layer greedily in the deep network, with contrastive divergence [41] (we used CD-1 in our experiments). Note that we used gaussian RBMs for the continuous data in the first layer, otherwise we used binary RBMs. Thus, our deep network can learn parametric nonlinear mapping from input $\mathbf{x}$ to output $\mathbf{h}$, $f : \mathbf{x} \to \mathbf{h}$.

Data: the training data $\mathcal{X}$, pairwise constraints C, the number of clusters K, the number of iterations T, λ, and β
Result: model parameters $\mathbf{W}$ and $\{\boldsymbol{\theta}_l\}_{l=1}^{L}$ in the deep neural network
Initialize $\boldsymbol{\theta}_l$ for $l = \{1, ..., L\}$ layer-by-layer greedily;
Initialize $\mathbf{W}$ in the latent space;
for $i = 1; i <= T; i++$ **do**
  if the objective in Eq. 5.45 has no significant changes, break;
  project all training data $\mathcal{X}$ into latent space via Eq. 5.43;
  find the most violated constrains according to Eqs. 5.38 and 5.47;
  compute the gradient w.r.t. $\mathbf{W}$ via Eq. 5.48;
  compute the gradient w.r.t. $\mathbf{H}$ via Eq. 5.52;
  compute the gradient w.r.t. $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_l\}_{l=1}^{L}$ with backpropagation;
  update the parameters with gradient descent via Eq. 5.53;
end
Return model parameters $\mathbf{W}$ and $\{\boldsymbol{\theta}_l\}_{l=1}^{L}$, as well as average accuracy;
Algorithm 15: Deep nonlinear metric learning for semi-supervised clustering

Parameter updating: After pre-training, we take a similar strategy as in [38] to initialize the clustering weight $\mathbf{W}$. To optimize our objective function, we first infer the most violated constraints from pairwise and transductive information. With the learned features as inputs in the maximum margin framework, then we can derive the gradient w.r.t. the top layer clustering weight. Meanwhile, we can compute the gradient w.r.t. to the features, as well as the gradients w.r.t. the low level weights through backpropagation. Finally, given the (sub)gradients w.r.t. all parameters in Eq. 5.45, we can optimize the whole network with gradient descent.

We also tried L-BFGS [39, 40] to update model parameters, but it does not perform well. In our model, we can update the model parameters as follows,

$$\begin{aligned} &\mathbf{W} \leftarrow \mathbf{W} - \gamma_{\mathbf{W}} d\mathbf{W}, \\ &\boldsymbol{\theta}_l \leftarrow \boldsymbol{\theta}_l - \gamma_{\boldsymbol{\theta}_l} d\boldsymbol{\theta}_l, l \in \{1, ..., L\} \end{aligned} \tag{5.53}$$

where $\gamma_{\mathbf{W}}$ is the learning rate for the clustering weight $\mathbf{W}$, and $\gamma_{\boldsymbol{\theta}_l}$ is the learning rate for weights $\boldsymbol{\theta}_l$ in the deep neural network. Thus, our method alternates between data projection and parameter optimization. For more details, refer to algorithm 15. After we learned the model parameters, we can do cluster analysis according to Eq. 5.42.

Chapter 6

Sequential Labeling

6.1 Introduction

Sequential data is common in a wide variety of domains including natural language processing, speech recognition and computational biology. In general, it is divided into time series and ordered data structures. As for the time-series data, it changes over time and keeps consistent in the adjacent clips. For example the time frames for speech or video analysis, daily prices of stocks or the rainfall measurements on successive days. There are also ordered data in the sequence, such as text and sentence for handwriting recognition, and genes. For example, successfully predicting protein-protein interactions requires knowledge of the secondary structures of the proteins and semantic analysis might involve annotating tokens with parts of speech tags.

The goal of this work is for sequence labeling, i.e. classify all items in a sequence. For example, in handwritten word recognition we wish to label a sequence of characters given features of the characters; in speech recognition we wish to label a sequence of phonemes; in weather analysis we wish to label a sequence of days as cloudy, rainy, sunny, etc from readings such as barometric pressure, temperature and humidity, and in gesture recognition we label a sequence of hand positions in the video stream. Compared to the static data, the sequential data provides more information, but also requires more complex model to handle it. The static data makes the assumption that training examples are independent and identically distributed

(i.i.d.). While for sequential data, this assumption does not necessarily hold because items arranged in a sequence often have interactions between the observations and the labels and between the labels. To classify the sequential data, we model it from the following two aspects: (1) we can transfer the model in the static data to handle the sequential data by labeling each item in the sequence separately. Thus, to design a better classifier is definitely helpful in the sequential labeling. (2) we can assume the sequence satisfies the Markov properties. In other words, there is correlation between the current item and the previous ones. For example, to model the context information in the sequential data, hidden Markov model (HMM) [42] was proposed for sequential classification, such as speech recognition. Recently, it shows that discriminative model, such as conditional random fields [3, 4, 43] in general is better than generative models on sequence labeling task.

On the one hand, we need a better classifier or feature selection strategy from the sequential data. On the other hand, how to discover the sequential patterns is important because they can be exploited to improve the classification performance. Hence, given a sequential data, the classification accuracy depends highly on the following factors: (1) how the original data is distributed and how the data representation is; (2) the classifier model and its accuracy on each static instance; (3) how to exploit the context information in the sequence. For instance, in the handwritten recognition, how to extract better representations from the data, improve the character level accuracy with better classifier and leverage the context information will significantly affect the final word level accuracy.

6.2 Markov properties

Given an observation sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$, the joint distribution $p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ uses the conditional distribution to factorize the joint distribution

$$\begin{aligned} p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) &= p(\mathbf{x}_T|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{T-1})p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{T-1}) \\ &= \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{t-1}) \end{aligned} \tag{6.1}$$

Because the high order conditional distribution is hard to compute when the fu-

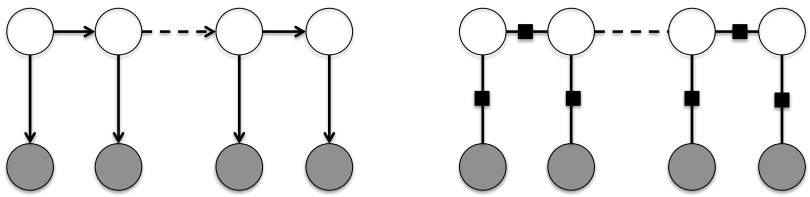


Figure 6.1. It shows two sequential models: hidden Markov model (the left figure) and conditional random fields (CRFs in the right), where white node is hidden variable and gray node indicates observation. (a) HMM is a generative model, where the observation depends on the hidden states, while (b) CRFs is a discriminative model where labels depends on observations.

ture T go to infinity. Thus it is more practical to consider that the prediction of future values is more related to the current states, compared to historical observations. Markov models assume the current states depends on the most recent values.

First order Markov model: assume that the current state at time t only depends on the state at $t-1$

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{t-1}) &= p(\mathbf{x}_t|\mathbf{x}_{t-1}) \\ \Longrightarrow p(\mathbf{x}_T|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{T-1}) &= p(\mathbf{x}_1)\prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}) \end{aligned} \tag{6.2}$$

Similarly, the second order

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{t-1}) &= p(\mathbf{x}_t|\mathbf{x}_{t-2}, \mathbf{x}_{t-1}) \\ \Longrightarrow p(\mathbf{x}_T|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{T-1}) &= p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)\prod_{t=3}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}) \end{aligned} \tag{6.3}$$

6.3 Hidden Markov model

Hidden Markov model (abbr. HMM), as a generative model, has been widely used to model sequential data. For the linear HMM, it assumes the Markov properties between nearby hidden states and each observation only depends on its corresponding hidden state, see Fig. 6.1(a). In the following part, we will introduce HMM to handle sequential prediction.

Given an observation sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$, we assume that there is an corresponding hidden states $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T$, where $\mathbf{x}_t$ is generated from the hidden state $\mathbf{h}_t$ and the hidden states are correlated with Markov properties. More specifically, the observations can be generated by the following process: (1) sample $\mathbf{h}_1$ from the distribution $p(\mathbf{h}_1)$, and also $\mathbf{x}_1$ from conditional distribution $p(\mathbf{x}_1|\mathbf{h}_1)$; (2) for $t = 2, ..., T$, choose $\mathbf{h}_t$ from the distribution $p(\mathbf{h}_t|\mathbf{h}_{t-1})$ for the first order dependency, and sample $\mathbf{x}_t$ from the conditional distribution $p(\mathbf{x}_t|\mathbf{h}_t)$.

Thus, we can get joint model with factorization (first order assumption)

$$p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) = \sum_{\mathbf{h}_t, t=1}^{T} p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T, \mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T) \tag{6.4}$$

$$= \sum_{\mathbf{h}_t, t=1}^{T} p(\mathbf{h}_1) \prod_{t=2}^{T} p(\mathbf{h}_t|\mathbf{h}_{t-1}) \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{h}_t) \tag{6.5}$$

Let K be the number of hidden states and M be the number of distinct symbols for each state in HMM, we need to learn the model parameters $\boldsymbol{\theta} = \{\{\pi_i\}_{i=1}^K, A, B\}$

(1) the initial state probability $p(\mathbf{h}_1 = i) = \pi_i$, for $i = 1, ..., K$;

(2) the transition probability (matrix) $p(\mathbf{h}_t = j|\mathbf{h}_t = i) = A_{ij}$, and $\sum_{j=1}^K A_{ij} = 1$;

(3) the emission probability $p(\mathbf{x}_t = m|\mathbf{h}_t = i) = B_{im}$, where each $\mathbf{x}_t$ has M distinct observations and $\mathbf{h}_t$ has K options with $\sum_m B_{im} = 1$

In general, given the training data, we have two main problems in machine learning: parameter estimation and prediction. Similarly, HMM has the same problems:

(1) Evaluation problem: determine the joint probability of a sequence $\mathbf{x}_1, ..., \mathbf{x}_T$, given the model,

$$p(\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta}) \tag{6.6}$$

(2) Decoding problem, given the observation $\mathbf{x}_1, ..., \mathbf{x}_T$, determine the most likely hidden states

$$\mathbf{h}_1^*, ..., \mathbf{h}_T^* = \arg\max_{\mathbf{h}_1, ..., \mathbf{h}_T} p(\mathbf{h}_1, ..., \mathbf{h}_T|\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta}) \tag{6.7}$$

(3) Learning problem: given the observation and the number of hidden states,

estimate the model parameters $\boldsymbol{\theta}$

$$\max_{\boldsymbol{\theta}} p(\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta}) \tag{6.8}$$

Solutions: in the following, we will introduce solutions to the basic three problems above.

6.3.1 Inference problem

The inference problem of HMM is to find the joint likelihood $p(\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta})$ given the model. Let's first talk about forward-backward algorithm, and then use it to handle inference problem.

The forward-backward algorithm is used in HMM to infer the posterior marginals of all hidden states given the observation sequence. More specifically, it computes

$$p(\mathbf{h}_t = i | \mathbf{x}_1, ..., \mathbf{x}_T) = \frac{p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_T)}{p(\mathbf{x}_1, ..., \mathbf{x}_T)} \tag{6.9}$$

The inference task can compute the posterior probability efficiently with two passes. The first pass uses dynamic programming in a forward manner in time, while the second goes backward.

Before we continue, we define the forward message:

$$\alpha_t^i = p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t) \tag{6.10}$$

where α_t^i or $\alpha_i(t)$ (which we will interchange explicitly) accounts for the partial observation sequence $\mathbf{x}_1, ..., \mathbf{x}_t$ and state $h_t = i$ at time t. And the backward message

$$\beta_t^i = p(\mathbf{x}_{t+1}, .., \mathbf{x}_T | \mathbf{h}_t = i) \tag{6.11}$$

where β_t^i or $\beta_i(t)$ represents for the remainder of the observation sequence $\mathbf{x}_{t+1}, .., \mathbf{x}_T$, given state $h_t = i$ at t.

After finishing the two passes, we can compute the marginal posterior probability

$$p(\mathbf{h}_t = i | \mathbf{x}_1, ..., \mathbf{x}_T) = \frac{p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_T)}{p(\mathbf{x}_1, ..., \mathbf{x}_T)}$$

$$
\begin{aligned}
&= \frac{p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{x}_{t+1}, ..., \mathbf{x}_T)}{p(\mathbf{x}_1, ..., \mathbf{x}_T)} \\
&= \frac{p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t) p(\mathbf{x}_{t+1}, ..., \mathbf{x}_T | \mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t)}{p(\mathbf{x}_1, ..., \mathbf{x}_T)} \\
&= \frac{p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t) p(\mathbf{x}_{t+1}, ..., \mathbf{x}_T | \mathbf{h}_t = i)}{p(\mathbf{x}_1, ..., \mathbf{x}_T)} \\
&= \frac{\alpha_t^i \beta_t^i}{p(\mathbf{x}_1, ..., \mathbf{x}_T)}
\end{aligned} \tag{6.12}
$$

As outlined above, the algorithm needs the following three steps to compute the posterior probability of any hidden state at time t:

- Compute the forward probabilities via Eq. 6.10;
- Compute the backward probabilities via Eq. 6.11
- Compute the posterior probabilities via Eq. 6.12

The problem is how to compute α_t^i efficiently. Notice that we can include the hidden status $\mathbf{h}_{i-1}$ into α_t^i to yield a recursive representation. More specifically, the forward algorithm computes the joint likelihood α_t^i at step t as

$$
\begin{aligned}
\alpha_t^i &= p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t) = p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_{t-1}, \mathbf{x}_t) \\
&= \sum_k p(\mathbf{h}_{t-1} = k, \mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_{t-1}, \mathbf{x}_t) \\
&= \sum_k p(\mathbf{h}_{t-1} = k, \mathbf{x}_1, ..., \mathbf{x}_{t-1}) p(\mathbf{h}_t = i, \mathbf{x}_t | \mathbf{h}_{t-1} = k, \mathbf{x}_1, ..., \mathbf{x}_{t-1}) \\
&= \sum_k p(\mathbf{h}_{t-1} = k, \mathbf{x}_1, ..., \mathbf{x}_{t-1}) p(\mathbf{h}_t = i | \mathbf{h}_{t-1} = k) p(\mathbf{x}_t | \mathbf{h}_t = i) \\
&= p(\mathbf{x}_t | \mathbf{h}_t = i) \sum_k \alpha_{t-1}^k A_{ki}
\end{aligned} \tag{6.13}
$$

Similarly, the backward algorithm to compute β_t^i is

$$
\begin{aligned}
\beta_t^i &= \sum_k p(\mathbf{x}_{t+1}, .., \mathbf{x}_T, \mathbf{h}_{t+1} = k | \mathbf{h}_t = i) \\
&= \sum_k p(\mathbf{x}_{t+2}, .., \mathbf{x}_T | \mathbf{x}_{t+1}, \mathbf{h}_{t+1} = k, \mathbf{h}_t = i) p(\mathbf{x}_{t+1} | \mathbf{h}_{t+1} = k, \mathbf{h}_t = i) p(\mathbf{h}_{t+1} = k | \mathbf{h}_t = i) \\
&= \sum_k p(\mathbf{x}_{t+2}, .., \mathbf{x}_T | \mathbf{h}_{t+1} = k) p(\mathbf{x}_{t+1} | \mathbf{h}_{t+1} = k) p(\mathbf{h}_{t+1} = k | \mathbf{h}_t = i)
\end{aligned}
$$

$$= \sum_k \beta_{t+1}^k p(\mathbf{x}_{t+1}|\mathbf{h}_{t+1} = k) A_{i,k} \tag{6.14}$$

Now let us go back to the inference problem to compute the probability of the sequence $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$:

$$p(\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta}) = \sum_{\mathbf{h}_T} p(\mathbf{x}_1, ..., \mathbf{x}_T, \mathbf{h}_T) \tag{6.15}$$

$$= \sum_{i=1}^K p(\mathbf{x}_1, ..., \mathbf{x}_T, \mathbf{h}_T = i) = \sum_{i=1}^K \alpha_T^i \tag{6.16}$$

6.3.2 Decoding problem

The decoding problem is to find the most likely hidden states given the observation sequence. We can use the same forward algorithm above, but we need to find the most likely hidden state in each step.

According to maximum a posterior probability:

$$\begin{aligned} &\arg\max_{i \in [1,..,K]} p(\mathbf{h}_t = i|\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta}) = \arg\max_i \frac{\alpha_t^i \beta_t^i}{p(\mathbf{x}_1, ..., \mathbf{x}_T)} \\ &= \arg\max_i \frac{\alpha_t^i \beta_t^i}{\sum_{i=1}^K \alpha_t^i \beta_t^i} \end{aligned} \tag{6.17}$$

The problem in decoding problem is that we need to maximize the globally (joint) posterior problem

$$\begin{aligned} &\arg\max_{\mathbf{h}_1,...,\mathbf{h}_T} p(\mathbf{h}_1, ..., \mathbf{h}_T|\mathbf{x}_1, ..., \mathbf{x}_T) = \arg\max_{\mathbf{h}_1,...,\mathbf{h}_T} \frac{p(\mathbf{h}_1, ..., \mathbf{h}_T, \mathbf{x}_1, ..., \mathbf{x}_T)}{p(\mathbf{x}_1, ..., \mathbf{x}_T))} \\ &\propto \arg\max_{\mathbf{h}_1,...,\mathbf{h}_T} p(\mathbf{h}_1, ..., \mathbf{h}_T, \mathbf{x}_1, ..., \mathbf{x}_T) \end{aligned} \tag{6.18}$$

Define the maximal probability of ending in state k at time t when we are maximizing over $\mathbf{h}_1, ..., \mathbf{h}_{t-1}$ as

$$v_t^k = \max_{\mathbf{h}_1,...,\mathbf{h}_{t-1}} p(\mathbf{x}_1, ..., \mathbf{x}_{t-1}, \mathbf{h}_1, ..., \mathbf{h}_{t-1}, \mathbf{x}_t, \mathbf{h}_t = k) \tag{6.19}$$

Then, we can get the forward rule for v_t^{k+1}

$$\begin{aligned}
v_{t+1}^k &= \max_{\mathbf{h}_1,...,\mathbf{h}_t} p(\mathbf{x}_1,...,\mathbf{x}_t,\mathbf{h}_1,...,\mathbf{h}_t,\mathbf{x}_{t+1},\mathbf{h}_{t+1}=k) \\
&= \max_{\mathbf{h}_1,...,\mathbf{h}_t} p(\mathbf{x}_1,...,\mathbf{x}_t,\mathbf{h}_1,...,\mathbf{h}_t)p(\mathbf{x}_{t+1},\mathbf{h}_{t+1}=k|\mathbf{x}_1,...,\mathbf{x}_t,\mathbf{h}_1,...,\mathbf{h}_t) \\
&= \max_{\mathbf{h}_1,...,\mathbf{h}_t} p(\mathbf{x}_{t+1},\mathbf{h}_{t+1}=k|\mathbf{h}_t)p(\mathbf{x}_1,...,\mathbf{x}_{t-1},\mathbf{h}_1,...,\mathbf{h}_{t-1},\mathbf{x}_t,\mathbf{h}_t) \\
&= \max_i \left\{ p(\mathbf{x}_{t+1},\mathbf{h}_{t+1}=k|\mathbf{h}_t=i) \max_{\mathbf{h}_1,...,\mathbf{h}_{t-1}} p(\mathbf{x}_1,...,\mathbf{x}_{t-1},\mathbf{h}_1,...,\mathbf{h}_{t-1},\mathbf{x}_t,\mathbf{h}_t=i) \right\} \\
&= \max_i p(\mathbf{x}_{t+1}|\mathbf{h}_{t+1}=k)A_{i,k}v_t^i
\end{aligned} \tag{6.20}$$

Thus, we can effectively compute v_T^k, and further we have

$$\arg\max_{\mathbf{h}_1,...,\mathbf{h}_T} p(\mathbf{h}_1,...,\mathbf{h}_T|\mathbf{x}_1,...,\mathbf{x}_T) = \arg\max_k v_T^k \tag{6.21}$$

After we find the best hidden state $\mathbf{h}_k$ at T according to Eq. 6.21, we can backtrace to get all the hidden states from $T-1$ to 1.

6.3.3 Learning problem

The learning problem of HMM is to estimate model parameters. Apparently, we can use maximum likelihood estimation. According to Eqs. 6.12 and 6.17, we define $\gamma_t(i)$ as the probability of being in state i at time t, given the observation sequence and the model

$$\gamma_t(i) = p(\mathbf{h}_t = i|\mathbf{x}_1,...,\mathbf{x}_T;\boldsymbol{\theta}) = \frac{\alpha_t^i\beta_t^i}{\sum_{i=1}^K \alpha_t^i\beta_t^i} \tag{6.22}$$

Considering all states at time t, we have

$$\sum_{i=1}^K \gamma_t(i) = 1 \tag{6.23}$$

Similarly, we can define $\xi_t(i,j)$ the probability of being in state i at time t and state

j at time $t+1$, given the observation and the model

$$\xi_t(i,j) = p(\mathbf{h}_t = i, \mathbf{h}_{t+1} = j | \mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta}) = \frac{p(\mathbf{h}_t = i, \mathbf{h}_{t+1} = j, \mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta})}{p(\mathbf{x}_1, ..., \mathbf{x}_T; \boldsymbol{\theta})}$$
$$= p(\mathbf{h}_t = i, \mathbf{x}_1, ..., \mathbf{x}_t) p(\mathbf{h}_{t+1} = j | \mathbf{h}_t = i) p(\mathbf{x}_{t+1} | \mathbf{h}_{t+1} = j) p(\mathbf{x}_{t+2}, ..., \mathbf{x}_T | \mathbf{h}_{t+1} = j)$$
$$= \frac{\alpha_t(i) A_{ij} B_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{K} \sum_{j=1}^{K} \alpha_t(i) A_{ij} B_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)} \tag{6.24}$$

After we calculate Eqs. 6.22 and 6.24, we can get the expected counts of being in state i, as well as the expected transitions from i to j

$$\sum_{t=1}^{T-1} \gamma_t(i) = expected\ number\ of\ being\ at\ state\ i,$$
$$\sum_{t=1}^{T-1} \xi_t(i,j) = expected\ number\ of\ transitions\ from\ state\ i\ to\ j$$

Thus, we estimate parameter $\boldsymbol{\theta}$ for an HMM as follows

$$\begin{aligned} \pi_i &= \gamma_1(i) \\ A_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ B_{im} &= \frac{\sum_{t=1, s.t. \mathbf{x}_t = m}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)} \end{aligned} \tag{6.25}$$

6.3.4 EM algorithm

Note that we only give parameter learning in the previous subsection, but do not tell how we yield the formulations for parameters A and B. In the following, we will show that how can we apply expectation maximization (EM) to HMM for parameter estimation. Given the observations $X = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$, what are the values of transition probability matrix A and emission matrix B? As a generative model, we can use EM algorithm to learn model parameters in Algorithm 16.

Data: the training set $\mathcal{D}$ and iterations T

Result: output A and B

set iteration index t = 0;

while $t < T$ **do**

E-step:

for *Each* $X \in \mathcal{D}$ **do**

calculate $q(H) = p(H|X; A, B)$;

end

M-step:

$A, B = \arg\max_{A,B} \sum_H q(H) \log \frac{p(H,X;A,B)}{q(H)}$ via Eq. 6.26 ;

s.t. $A_{i,j} \geq 0; \sum_{j=1}^{K} A_{i,j} = 1, \forall i, j;$

$B_{j,m} \geq 0; \sum_{m=1}^{M} B_{j,m} = 1, \forall j, m;$

if (converged) break;

end

Algorithm 16: EM algorithm

In order to derive Eq. 6.25, we will follow the EM algorithm above and give mathematical operations in details. Consider the maximum likelihood:

$$\begin{aligned} A, B &= \arg\max_{A,B} \log p(X; A, B) = \arg\max_{A,B} \log \sum_H p(H, X; A, B) \\ &= \arg\max_{A,B} \log \sum_H \frac{q(H) p(H, X; A, B)}{q(H)} \\ &\geq \arg\max_{A,B} \sum_H q(H) \log \frac{p(H, X; A, B)}{q(H)} \end{aligned} \tag{6.26}$$

To maximize $p(X; A, B)$, we need to maximize the low bound in Eq. 6.26. Thus, we have

$$\begin{aligned} &\sum_H q(H) \log \frac{p(H, X; A, B)}{q(H)} \\ =&\sum_H q(H) \log p(H, X; A, B) \\ =&\sum_H q(H) \log \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{h}_t; B) \prod_{t=1}^{T} p(\mathbf{h}_t|\mathbf{h}_{t-1}, A) \end{aligned}$$

$$= \sum_{H} q(H) \sum_{t=1}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{m=1}^{M} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j} \log B_{j,k} + \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} \log A_{i,j} \tag{6.27}$$

In addition, A and B also need to satisfy the following constraints, s.t.

$$\begin{aligned} \sum_{j=1}^{K} A_{i,j} = 1,\ i = 1, ..., K;\ A_{i,j} \geq 0 \\ \sum_{m=1}^{M} B_{j,m} = 1,\ i = 1, ..., K;\ B_{j,m} \geq 0 \end{aligned} \tag{6.28}$$

By combining Eqs. 6.27 and 6.28, we can construct the following Lagrangian with multipliers λ and ξ:

$$\begin{aligned} \mathcal{L}(A, B, \lambda, \xi) = \sum_{H} q(H) \sum_{t=1}^{T} \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{m=1}^{M} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j} \log B_{j,k} + \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} \log A_{i,j} \\ + \sum_{i=1}^{K} \lambda_i (1 - \sum_{j=1}^{K} A_{i,j}) + \sum_{j=1}^{K} \xi_j (1 - \sum_{m=1}^{M} B_{j,m}) \end{aligned} \tag{6.29}$$

Taking the partial derivatives and set them to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}(A, B, \lambda, \xi)}{\partial A_{i,j}} &= \sum_{H} q(H) \sum_{t=1}^{T} \frac{1}{A_{i,j}} \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} - \lambda_i = 0 \\ A_{i,j} &= \frac{1}{\lambda_i} \sum_{H} q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} \end{aligned} \tag{6.30}$$

$$\begin{aligned} \frac{\partial \mathcal{L}(A, B, \lambda, \xi)}{\partial B_{j,m}} &= \sum_{H} q(H) \sum_{t=1}^{T} \frac{1}{B_{j,m}} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j} - \xi_j = 0 \\ B_{j,m} &= \frac{1}{\xi_j} \sum_{H} q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j} \end{aligned} \tag{6.31}$$

Notice that $\sum_j A_{i,j} = 1$, we have the following normalization constraints

$$\begin{aligned} \sum_{j=1}^{K} A_{i,j} = \sum_{j=1}^{K} \frac{1}{\lambda_i} \sum_{H} q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} = 1 \\ \lambda_i = \sum_{j=1}^{K} \sum_{H} q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} = \sum_{H} q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i} \end{aligned} \tag{6.32}$$

where we use Eq. 6.30 in the above derivation.

Similarly, according to the constraint $\sum_m B_{j,m} = 1$, we have

$$\sum_m B_{j,m} = \sum_m \frac{1}{\xi_j} \sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j} = 1$$
$$\Longrightarrow \xi_j = \sum_m \sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j} = \sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_t=j} \tag{6.33}$$

Substituting Eq. 6.32 back to Eq. 6.30, and Eq. 6.33 back to Eq. 6.31 respectively, we can get the parameters A and B, which maximizes the likelihood w.r.t. the training data

$$A_{i,j} = \frac{\sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j}}{\sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i}} \tag{6.34}$$

$$B_{j,m} = \frac{\sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{x}_t=m \wedge \mathbf{h}_t=j}}{\sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_t=j}} \tag{6.35}$$

However, both Eqs. 6.34 and 6.35 sum over all possible hidden sequence $H = \{h_1, ..., h_T\}$. By defining $q(H) = p(H|X; A, B)$ via EM, we have

$$\sum_H q(H) \sum_{t=1}^{T} \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} \tag{6.36}$$
$$= \sum_{t=1}^{T} \sum_H \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} p(H|X; A, B) \tag{6.37}$$
$$= \frac{1}{p(X; A, B)} \sum_{t=1}^{T} \sum_H \mathbb{1}_{\mathbf{h}_{t-1}=i \wedge \mathbf{h}_t=j} p(H, X; A, B) \tag{6.38}$$
$$= \frac{1}{p(X; A, B)} \sum_{t=1}^{T} \alpha_i(t) A_{i,j} B_{\{j,\mathbf{x}_t\}} \beta_j(t+1) \tag{6.39}$$

then we can represent $A_{i,j}$ as

$$A_{i,j} = \frac{\sum_{t=1}^{T} \alpha_i(t) A_{i,j} B_{\{j,\mathbf{x}_t\}} \beta_j(t+1)}{sum_{j=1}^{K} \sum_{t=1}^{T} \alpha_i(t) A_{i,j} B_{\{j,\mathbf{x}_t\}} \beta_j(t+1)} \tag{6.40}$$

Similarly, we can have the representation for $B_{j,k}$ as

$$B_{j,k} = \frac{\sum_{i=1}^{K}\sum_{t=1}^{T}\mathbb{1}_{\mathbf{x}_t=m}\alpha_i(t)A_{i,j}B_{j,m}\beta_j(t+1)}{\sum_{i=1}^{K}\sum_{t=1}^{T}\alpha_i(t)A_{i,j}B_{j,m}\beta_j(t+1)} \tag{6.41}$$

6.4 Conditional random fields

In this part, we will introduce conditional random fields (CRFs), with a focus on the first order linear CRFs. And then we will extend it into deep CRFs, which leverage deep neural network to learn representations before forward into linear CRFs.

6.4.1 Linear CRFs

Let $D = \{\langle \mathbf{x}_i, \mathbf{y}_i\rangle\}_{i=1}^{N}$ be a set of N training examples. Each example is a pair of a time series $\langle \mathbf{x}_i, \mathbf{y}_i\rangle$, with $\mathbf{x}_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, ..., \mathbf{x}_{i,T_i}\}$ and $\mathbf{y}_i = \{y_{i,1}, y_{i,2}, ..., y_{i,T_i}\}$, where $\mathbf{x}_{i,t} \in \mathbb{R}^d$ is the i-th observation at time t and $y_{i,t}$ is the corresponding label (we indicate its encoded vector as $\mathbf{y}_{i,t}$ that uses a so-called 1-of-K encoding). Linear first-order CRFs [3] is a conditional discriminative model over the label sequence given the data, refer to Fig. 6.1(b). For linear CRFs, we maximize the following conditional probability:

$$p(\mathbf{y}_i|\mathbf{x}_i) = \frac{\exp\{-E(\mathbf{x}_i, \mathbf{y}_i)\}}{Z(\mathbf{x}_i)} \tag{6.42}$$

where $Z(\mathbf{x}_i)$ is the partition function and $E(\mathbf{x}_i, \mathbf{y}_i)$ is the energy function given by

$$\begin{aligned} -E(\mathbf{x}_i, \mathbf{y}_i) &= \mathbf{y}_{i,1}^T\boldsymbol{\pi} + \mathbf{y}_{i,T_i}^T\boldsymbol{\tau} + \\ &\sum_{t=1}^{T_i}(\mathbf{x}_{i,t}^T\mathbf{W}\mathbf{y}_{i,t} + \mathbf{b}^T\mathbf{y}_{i,t}) + \sum_{t=2}^{T_i}\mathbf{y}_{i,t-1}^T\mathbf{A}\mathbf{y}_{i,t} \end{aligned} \tag{6.43}$$

where $\mathbf{y}_{i,1}^T\boldsymbol{\pi}$ and $\mathbf{y}_{i,T_i}^T\boldsymbol{\tau}$ are the initial-state and final-state factors respectively, $\mathbf{b}^T\mathbf{y}_{i,t}$ is the bias term for labels, $\mathbf{A} \in \mathbb{R}^{K\times K}$ represents the state transition (correlation) parameters and $\mathbf{W} \in \mathbb{R}^{d\times K}$ represents the parameters of the data-dependent term. Notice that $\mathbf{x}_{i,t}^T\mathbf{W}\mathbf{y}_{i,t} + \mathbf{b}^T\mathbf{y}_{i,t} = (\mathbf{x}_{i,t}^T\mathbf{W} + \mathbf{b}^T)\mathbf{y}_{i,t}$ models the compatibility between the prediction and the label at time step t. In other words, we hope the prediction $\mathbf{x}_{i,t}^T\mathbf{W} + \mathbf{b}^T$ matches the groundtruth $\mathbf{y}_{i,t}$, which can further boost the likelihood in Eq. 6.42. Re-

call that linear SVM uses the same form $\mathbf{x}_{i,t}^T\mathbf{W} + \mathbf{b}^T$, which also attempts to make it matches the groundtruth $\mathbf{y}_{i,t}$.

Compared to SVM, CRFs incorporates an additional term $\mathbf{y}_{i,t-1}^T\mathbf{A}\mathbf{y}_{i,t}$ to model the correlation in the sequence. Thus, CRFs extends SVM to handle sequential classification.

Objective function: we can add an regularization term to Eq. 6.42 as follows

$$\mathcal{L}(\mathbf{x}, y; \boldsymbol{\omega}) = p(\mathbf{y}_i|\mathbf{x}_i) + \lambda_1||\mathbf{W}||^2 + \lambda_2||\mathbf{A}||^2 \tag{6.44}$$

where we use $\boldsymbol{\omega} = \{\mathbf{A}, \mathbf{W}, \boldsymbol{\pi}, \boldsymbol{\tau}, \mathbf{b}\}$ to represent all model parameters.

Learning: we can estimate model parameters with maximum likelihood to Eq. 6.42. Firstly, we can calculate the (sub)gradients for all parameters:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \sum_{i=1}^{N}\sum_{t=2}^{T_i}\mathbf{y}_{i,t-1}(\mathbf{y}_{i,t})^T - \boldsymbol{\gamma}_{i,t-1}(\boldsymbol{\gamma}_{i,t})^T + \lambda_2\mathbf{A};$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\pi}} = \sum_{i=1}^{N}(\mathbf{y}_{i,1} - \boldsymbol{\gamma}_{i,1});$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\tau}} = \sum_{i=1}^{N}(\mathbf{y}_{i,T_i} - \boldsymbol{\gamma}_{i,T_i});$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \sum_{i=1}^{N}\big(\sum_{t=1}^{T_i}(\mathbf{y}_{i,t} - \boldsymbol{\gamma}_{i,t})\big);$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{i=1}^{N}\sum_{t=1}^{T_i}\big(\mathbf{x}_{i,t}(\mathbf{y}_{i,t} - \boldsymbol{\gamma}_{i,t})^T\big) + \lambda_1\mathbf{W} \tag{6.45a}$$

where $\boldsymbol{\gamma}_{i,t} \in \mathbb{R}^K$ is the prediction with the length K, which can be thought as the posterior probability for labels in the sequence and will be introduced in Sec. 6.4.4. Note that it is easy to derive the gradients of the ℓ_2 regularization term w.r.t. $\boldsymbol{\theta}$ in the objective in Eq. (6.47), which can be added to the gradients in Eq. (6.49).

Then, we can update model parameters with gradient based methods, such as

stochastic gradient descent and batch-based gradient method.

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \eta_{\omega} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\omega}} \tag{6.46}$$

where η_{ω} is the learning rate.

One of the main disadvantages of linear CRFs is the linear dependence on the raw input data term. Thus, we introduce our sequential labeling model with deep feature learning, which leverages both context information, as well as nonlinear representation learning [44]. In the following parts, we will introduce our model and our robust learning method. More details related to inference and learning will be introduced in the following deep CRFs.

6.4.2 Deep CRFs

Although it is possible to leverage the deep neural networks for structured prediction, its output space is explosively growing because of non-determined length of sequential data. Thus, we consider a compromised model, which combines CRFs and deep learning in an unified framework, refer Fig. 6.1(b). We propose an objective function with L layers neural network structure,

$$\mathcal{L}(D; \boldsymbol{\theta}, \boldsymbol{\omega}) = -\sum_{i=1}^{N} \log p(\mathbf{y}_{i,1}, ..., \mathbf{y}_{i,T_i} | \mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i}) + \lambda_2 ||\boldsymbol{\theta}||^2 + \lambda_3 ||\boldsymbol{\omega}|| \tag{6.47}$$

where $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ are the top layer parameters and lower layer ($l = \{1, ..., L-1\}$) parameters respectively, which will be explained later. The first row on the right side of the equation is from the linear CRFs in Eq. (6.42), but with latent features, which depends respectively on $\boldsymbol{\theta}$ and the latent non-linear features $\mathbf{h}_i = \{\mathbf{h}_{i,1}, .., \mathbf{h}_{i,T_i}\}$ in the coding space, with

$$\mathbf{h}_{i,t} = \underbrace{f_{L-1} \circ \cdots \circ f_1}_{L-1 \text{ times}}(\mathbf{x}_{i,t}) \tag{6.48}$$

where $\circ$ indicates the function composition, and f_i is logistic function with the weight parameter $\mathbf{W}_l$ respectively for $l = \{1, .., L-1\}$, refer more details in Sec. 5.5.2. With a bit abuse of notation, we denote $\mathbf{h}_{i,t} = f_{1 \rightarrow (L-1)}(\mathbf{x}_{i,t})$.

The last two terms in Eq. (6.47) are for regularization on the all parameters with $\boldsymbol{\theta} = \{\mathbf{A}, \mathbf{W}, \boldsymbol{\pi}, \boldsymbol{\tau}, \boldsymbol{b}, \boldsymbol{c}\}$, and $\boldsymbol{\omega} = \{\mathbf{W}_l | l \in [1, .., L-1]\}$. We add the ℓ_2 regularization to $\boldsymbol{\theta}$ as most linear CRFs does, while we have the ℓ_1-regularized term on weight parameters $\boldsymbol{\omega}$ in the deep neural network to avoid overfitting in the learning process.

6.4.3 Learning

In training the CRFs with deep feature learning, our aim is to minimize objective function $\mathcal{L}(D; \boldsymbol{\theta}, \boldsymbol{\omega})$ in Eq. (6.47). Because we introduce the deep neural network here for feature learning, the objective is not convex function anymore. However, we can find a local minimum in Eq. (6.47). In our learning framework, we optimize the objective function with an online learn algorithm, by mixing perceptron training and stochastic gradient descent. More specifically, we can update the top layer CRFs related parameters with Perceptron learning and the lower lever weights with stochastic gradient descent.

Firstly, we can calculate the (sub)gradients for all parameters. Considering different regularization methods for $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ respectively, we can calculate gradients w.r.t. them separately. As for the parameters in the negative log likelihood in Eq. 6.47, we can compute the gradients w.r.t. $\boldsymbol{\theta}$ as follows

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \sum_{i=1}^{N} \sum_{t=2}^{T_i} \mathbf{y}_{i,t-1} (\mathbf{y}_{i,t})^T - \boldsymbol{\gamma}_{i,t-1} (\boldsymbol{\gamma}_{i,t})^T; \tag{6.49a}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\pi}} = \sum_{i=1}^{N} (\mathbf{y}_{i,1} - \boldsymbol{\gamma}_{i,1}); \tag{6.49b}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\tau}} = \sum_{i=1}^{N} (\mathbf{y}_{i,T_i} - \boldsymbol{\gamma}_{i,T_i}); \tag{6.49c}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \sum_{i=1}^{N} \big(\sum_{t=1}^{T_i} (\mathbf{y}_{i,t} - \boldsymbol{\gamma}_{i,t}) \big); \tag{6.49d}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{i=1}^{N} \sum_{t=1}^{T_i} \big(\mathbf{h}_{i,t} (\mathbf{y}_{i,t} - \boldsymbol{\gamma}_{i,t})^T \tag{6.49e}$$

where $\gamma_{i,t} \in \mathbb{R}^K$ is the prediction with the length K, which can be thought as the posterior probability for labels in the sequence and will be introduced in Sec. 6.4.4. Note that it is easy to derive the gradients of the ℓ_2 regularization term w.r.t. $\boldsymbol{\theta}$ in the objective in Eq. (6.47), which can be added to the gradients in Eq. (6.49).

As for the gradients of weights $\boldsymbol{\omega} = \{\mathbf{W}_l | l \in [1, .., L-1]\}$, we first use backpropagation to get the partial gradient in the neural network, refer to [44] for more details. Then the gradient of the ℓ_1 term in Eq. (6.47) can be attached to get the final gradients w.r.t. $\mathbf{W}_l$ for $l = \{1, .., L-1\}$.

To update the CRF related parameters with perceptron learning, we need to find the most violated constraints for each example. Basically, given a training example $\langle \mathbf{x}_i, \mathbf{y}_i \rangle$, we infer its most violated labeling $\mathbf{y}_i^*$. If the frame is misclassified, then it directly performs a type of stochastic gradient descent on the energy gap between the observed label sequence and the predicted label sequence. Otherwise, we do not need to update the model parameters. Thus, for the parameters $\boldsymbol{\theta}$ from the negative log likelihood in Eq. (6.47), we first project $\mathbf{x}_i$ into the code $\mathbf{h}_i$ according to Eq. 6.48. Then, the updating rule takes the form below

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \frac{\partial}{\partial \boldsymbol{\theta}} \big(E(\mathbf{h}_i, \mathbf{y}_i) - E(\mathbf{h}_i, \mathbf{y}_i^*) \big) \tag{6.50}$$

where $\mathbf{y}_i^*$ is the most violated constraint in the misclassificated case, and $\eta_{\boldsymbol{\theta}}$ is a parameter step size. Note that the posterior probability $\gamma_{i,t} \in \mathbb{R}^K$ in Eq. (6.49) should be changed into the hard label assignment $\mathbf{y}_{i,t}^*$ in the inference stage.

While for the weights $\boldsymbol{\omega}$ in the deep neural network, we first use backpropagation to compute the gradients, and then update it as follows

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta_{\boldsymbol{\omega}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\omega}} \tag{6.51}$$

where $\eta_{\boldsymbol{\omega}}$ is the step size for the parameters.

6.4.4 Inference

Given the observation $\mathbf{x}_i = \{\mathbf{x}_{i,1}, ..., \mathbf{x}_{i,T_i}\}$, we first use Eq. 6.48 to compute the nonlinear code $\mathbf{h}_i = \{\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i}\}$. To simplify the problem, we assume the first-order CRFs here. To estimate the parameters $\boldsymbol{\theta}$, there are two main inferential problems

that need to be solved during learning: (1) the posterior probability (or the marginal distribution of a label given the codes) $\gamma_{i,t}(k) = p(y_{i,t} = k|\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i})$; (2) the distribution over a label edge $\xi_{i,t}(j,k) = p(y_{i,t} = j, y_{i,t+1} = k|\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i})$. The inference problem can be solved efficiently with Viterbi algorithm [42, 45].

For the given hidden sequence $\mathbf{h}_i = \{\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i}\}$, we assume the corresponding states $\{q_{i,1}, ..., q_{i,T_i}\}$. Furthermore, we define the forward messages:

$$\alpha_{i,t}(k) \propto p(y_{i,1}, .., y_{i,t}, q_{i,t} = k|\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i}) \tag{6.52}$$

, and the backward messages

$$\beta_{i,t}(k) \propto p(y_{i,t+1}, .., y_{i,T_i}|q_{i,t} = k, \mathbf{h}_{i,1}, ..., \mathbf{h}_{i,T_i}) \tag{6.53}$$

Furthermore, we have the following recursive expression

$$\alpha_{i,t+1}(j) = \left[\sum_{k=1}^{K} \alpha_{i,t}(k) A_{kj}\right] B(j, y_{i,t+1}); \tag{6.54}$$

$$\beta_{i,t}(j) = \sum_{k=1}^{K} A_{jk} B(k, y_{i,t+1}) \beta_{i,t+1}(k); \tag{6.55}$$

where $B(k, y_{i,t})$ is the probability to emit $y_{i,t}$ at the state k. We can compute it as follows

$$B(:, y_{i,t}) = exp\{\mathbf{h}_{i,t}^T \mathbf{W} + \mathbf{b}^T + \lambda_1 f_L(\mathbf{h}_{i,t})\} \tag{6.56}$$

After calculate $\alpha_{i,t+1}(j)$ and $\beta_{i,t}(j)$, we can compute the marginal probability for $\gamma_{i,t}$ and $\xi_{i,t}$ respectively

$$\gamma_{i,t}(k) \propto \alpha_{i,t}(k) \beta_{i,t}(k), \tag{6.57}$$

$$\xi_{i,t}(k,j) \propto \alpha_{i,t}(k) A_{kj} B(j, y_{i,t+1}) \beta_{i,t+1}(j); \tag{6.58}$$

Then, we can compute $\boldsymbol{\gamma}_{i,t}$ in Eq. (6.58), which is the concatenation: $[\gamma_{i,t}(1), ..., \gamma_{i,t}(K)]$.

In the testing stage, the main inferential problem is to compute the most likely label sequence $\mathbf{y}^*_{1,...,T}$ given the data $\mathbf{x}_{1,...,T}$ by $\arg\max_{\mathbf{y}\prime_{1,...,T}} p(\mathbf{y}\prime_{1,...,T}|\mathbf{x}_{1,...,T})$, which can be addressed similarly using the Viterbi algorithm mentioned above.

Chapter 7

Deep Learning

Supervised learning is to learn a model from the training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ with K classes, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, ..., K\}$. In other words, we need to learn a mapping f from input $\mathbf{x}$ to y. Because for every input $\mathbf{x}$, we have the corresponding target y in the training set $\mathcal{D}$, so we learn the model in an supervised manner. Similarly, as for unsupervised learning, the training target is not given. But we still need to learn a mapping function for clustering, dimension reduction, feature learning, etc.

Thus, no matter supervised or unsupervised learning, the key challenge is how to learn a good mapping function f, so that we can do classification/clustering in an effective and efficient way. Previously, we have talked many linear models, such as perceptron training, linear support vector machines (SVM), logistic regression, etc. However, these models cannot handle nonlinear cases. Fortunately, there is kernel SVMs, which can learn a nonlinear mapping via kernel trick. But one key weakness of kernel SVMs is that it is not scalable for large scale dataset because it needs to calculate Gram matrix.

Recent advances in deep learning [41, 46, 47] have sparked great interest in dimension reduction [44, 48] and classification problems [41, 49, 50]. In a sense, the success of deep learning lies on learned features via multiple layers of nonlinear mapping, which are useful for supervised/unsupervised tasks [47, 51]. For example, the deep autoencoders [44] pre-trained with stacked restricted Boltzmann machines (RBMs), learn low-dimensional manifold by minimizing the reconstruction error to facilitate the classification and visualization of data. Refer to Fig. (7.1) for visual understanding of the model. Convolutional neural network (CNN) yields significant results on

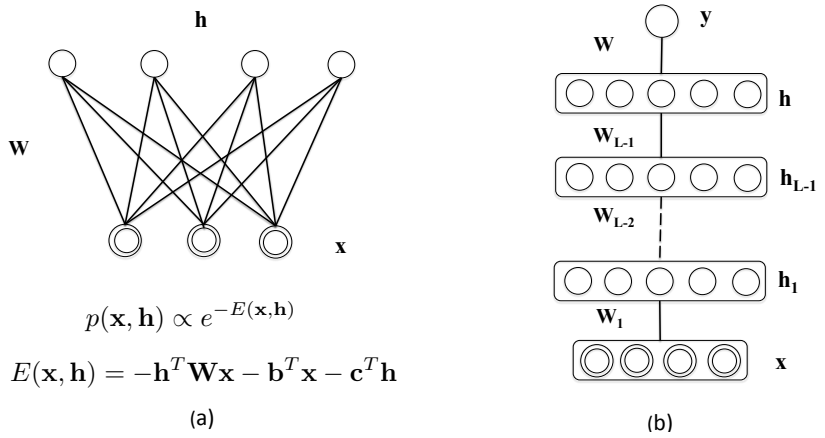


Figure 7.1. Restricted Boltzmann machines (RBMs) and deep neural networks. (a) Restricted Boltzmann machines (RBMs); (b) deep neural networks with full connections for classification.

image classification and object detection [52, 53]. Moreover, there are enough training data available for complex deep learning models in the era of data growing exponentially. In addition, deep learning is scalable for large dataset. For example, we can use stochastic or mini-batch gradient descent to update model parameters via backpropagation.

In this chapter, we will focus on different deep learning models and how to do learning and inference with mathematical formulas. We will first introduce Restricted Boltzmann machines (RBMs). Then, we will discuss deep belief networks and convolutional neural network. Further, more complex models, such as recurrent neural networks and long short term memory will be introduced to handle time series data. And also the learning approach, especially back-propagation will be discussed for these deep learning models.

7.1 Restricted Boltzmann machines

Restricted Boltzmann Machines (RBM) [44] are a particular form of Markov random field (undirected generative model), which are constructed with hidden nodes and visible nodes, and each connection in an RBM must link between a visible node and a hidden node (a bipartite graph, neither connection among visible nodes nor connection among hidden nodes). Recent advances in RBM and deep learning [54] have

attracted significant attention on many machine learning problems, such as dimension reduction [44], text categorization [49] and collaborative filtering [55]. In this part, we will focus on the building block, a.k.a. RBM.

7.1.1 Joint likelihood

An RBM with n hidden units is a parametric model of the joint distribution between a layer of hidden variables $\mathbf{h} = (h_1, ..., h_n)$ and the observations $\mathbf{v} = (v_1, ..., v_d)$, where there are no intra-layer connections. Please refer to Fig. 7.1 (a) for details. An RBM joint likelihood takes the following form:

$$p(\mathbf{v}, \mathbf{h}) \propto \exp(-E(\mathbf{v}, \mathbf{h})) \tag{7.1}$$

where the energy function is

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T \mathbf{W}\mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{c}^T \mathbf{h} \tag{7.2}$$

Note that an RBM is a generative model, which is the joint distribution of visible variables and hidden units $(\mathbf{v}, \mathbf{h})$. To generate data from an RBM, we can use Gibbs sampling to alternate between two layers (hidden units and visible units, i.e. sample one layer given the other layer fixed) until reach to an equilibrium distribution. Because an RBM has a single layer of hidden units, we can marginalize out them to get the distribution w.r.t. $\mathbf{v}$. In addition, an RBM is full connected bipartite graph, we can leverage the independence between visible variables and hidden units to yield very meaningful results.

By marginalizing out hidden variable $\mathbf{h}$, we can induce the following distribution w.r.t. $\mathbf{v}$:

$$\begin{aligned}
p(\mathbf{v}) &= \sum_{\mathbf{h}} \frac{1}{Z} \exp(\mathbf{v}^T \mathbf{W}\mathbf{h} + \mathbf{b}^T \mathbf{v} + \mathbf{c}^T \mathbf{h}) \\
&= \frac{1}{Z} \exp(\mathbf{b}^T \mathbf{v}) \prod_j \sum_{h_j \in \{0,1\}} \exp(\mathbf{v}^T W(:, j) h_j + c_j h_j) \\
&= \frac{1}{Z} \exp(\mathbf{b}^T \mathbf{v}) \exp\Big(\sum_j \log \sum_{h_j \in \{0,1\}} \exp\big(\mathbf{v}^T W(:, j) h_j + c_j h_j\big) \Big)
\end{aligned}$$

$$= \frac{1}{Z}\exp\left(\mathbf{b}^T\mathbf{v} + \sum_j \log\left(1+\exp(\mathbf{v}^T W(:,j) + c_j)\right)\right)$$
$$= \frac{1}{Z}\exp(-F(\mathbf{v})) \tag{7.3}$$

where the normalizing factor Z is called the partition function, $W_{.j}$ or $W_{(:,j)}$ is j-th column vector of the weight $\mathbf{W}$ and $F(\mathbf{v})$ is the free energy of an RBM with binary units with the following formula:

$$F(\mathbf{v}) = -\mathbf{b}^T\mathbf{v} - \sum_j \log\left(1+\exp(\mathbf{v}^T W(.,j) + c_j)\right) \tag{7.4}$$

To learn RBM parameters, we need to minimize the negative log likelihood $-\log p(\mathbf{v})$

$$-\frac{\partial \log p(\mathbf{v})}{\partial \boldsymbol{\theta}} = \frac{\partial F(\mathbf{v})}{\partial \boldsymbol{\theta}} - \sum_{\hat{\mathbf{v}}} p(\hat{\mathbf{v}})\frac{\partial F(\hat{\mathbf{v}})}{\partial \boldsymbol{\theta}}$$
$$= \frac{\partial F(\mathbf{v})}{\partial \boldsymbol{\theta}} - \langle \frac{\partial F(\hat{\mathbf{v}})}{\partial \boldsymbol{\theta}} \rangle \tag{7.5}$$

where $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$, $\hat{\mathbf{v}}$ can be sampled according to Gibbs sampling, i.e. Monte-Carlo Markov chain (MCMC) from $p(\mathbf{v})$, and the angle brackets are used to denote expectations under the distribution specified from the model. Considering MCMC sampling, we can sample $\mathbf{h}$ via Eq. 7.6c, and then sample $\hat{\mathbf{v}}$ from Eq. 7.6b. Finally, the sampled data will converge to the model's distribution.

7.1.2 Parameters learning

To learn model parameters, we can use gradient based approaches. However, the expectation in Eq. 7.5 is not explicitly given. Fortunately, we can use Gibbs sampling to get the expected value. The independence between the variables in one layer makes Gibbs sampling especially easy: instead of sampling new values for all variables separately, the states of all variables in one layer can be sampled jointly. Thus, Gibbs sampling can be performed in just two sub steps: sampling a new state $\mathbf{h}$ for the hidden neurons based on $p(h|v)$ and sampling a state $\mathbf{v}$ for the visible layer based on $p(v|h)$. This is also referred to as block Gibbs sampling.

And we can compute the following conditional likelihood:

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) \tag{7.6a}$$

$$p(v_i = 1|\mathbf{h}) = \text{logistic}(b_i + \sum_j W(i,j)h_j) \tag{7.6b}$$

$$p(h_i = 1|\mathbf{v}) = \text{logistic}(c_i + \sum_j W(j,i)v_j) \tag{7.6c}$$

where $\text{logistic}(x) = 1/(1+e^{-x})$. The conditional probability of a single variable being one can be interpreted as the firing rate of a (stochastic) neuron with sigmoid activation function. Before we continue, we first discuss how to derive the logistic distribution in Eq. 7.6.

$$p(v_i = 1|\mathbf{h}) = \frac{p(v_i = 1, \mathbf{h})}{p(v_i = 1, \mathbf{h}) + p(v_i = 0, \mathbf{h})} \tag{7.7}$$

$$= \frac{\exp(\sum_{-i,j} v_{-i} W_{-i,j} h_j) * \exp(\sum_j W_{i,j} h_j)}{\exp(\sum_{-i,j} v_{-i} W_{-i,j} h_j) * exp(\sum_j W_{i,j} h_j) + \exp(\sum_{-i,j} v_{-i} W_{-i,j} h_j)} \tag{7.8}$$

$$= \frac{\exp(\sum_j W_{i,j} h_j)}{\exp(\sum_j W_{i,j} h_j) + 1} \tag{7.9}$$

$$= \frac{1}{1 + \exp(-\sum_j W_{i,j} h_j)} \tag{7.10}$$

where $-i$ is any index in $\{1, ..., d\}$ except i. To make it easy to understand, we ignore bias $\mathbf{b}$ in the mathematical operation above.

To infer Eq. 7.5, we take the derivative of $p(\mathbf{v})$ w.r.t. $\boldsymbol{\theta}$,

$$\frac{\partial \ln p(\mathbf{v})}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left(\ln \frac{\sum_{\mathbf{h}} \exp\{-E(\mathbf{v}, \mathbf{h})\}}{Z} \right) \tag{7.11}$$

$$= \frac{\partial}{\boldsymbol{\theta}} \left(\ln \sum_{\mathbf{h}} \exp\{-E(\mathbf{v}, \mathbf{h})\} - \ln \sum_{\mathbf{h},\mathbf{v}} \exp\{-E(\mathbf{v}, \mathbf{h})\} \right) \tag{7.12}$$

$$= \frac{1}{\sum_{\mathbf{h}} \exp\{-E(\mathbf{v}, \mathbf{h})\}} \sum_{\mathbf{h}} \exp\{-E(\mathbf{v}, \mathbf{h})\} \frac{\partial -E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} \tag{7.13}$$

$$- \frac{1}{\sum_{\mathbf{h},\mathbf{v}} \exp\{-E(\mathbf{v}, \mathbf{h})\}} \sum_{\mathbf{h},\mathbf{v}} \exp\{-E(\mathbf{v}, \mathbf{h})\} \frac{\partial -E(\mathbf{v}, \mathbf{h})}{\partial \boldsymbol{\theta}} \tag{7.14}$$

$$= \sum_{\mathbf{h}} \frac{\exp\{-E(\mathbf{v},\mathbf{h})\}}{\sum_{\mathbf{h}} \exp\{-E(\mathbf{v},\mathbf{h})\}} \frac{\partial -E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} - \sum_{\mathbf{h},\mathbf{v}} \frac{\exp\{-E(\mathbf{v},\mathbf{h})\}}{\sum_{\mathbf{h},\mathbf{v}} \exp\{-E(\mathbf{v},\mathbf{h})\}} \frac{\partial -E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \tag{7.15}$$

$$= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} + \sum_{\mathbf{h},\mathbf{v}} p(\mathbf{h},\mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \tag{7.16}$$

To learn RBM parameters, we can use maximum likelihood estimation. In other words, we need to minimize the negative log likelihood $-\log p(\mathbf{v})$ given training data. Fortunately, the parameters updating can be calculated with an efficient stochastic descent method, namely contrastive divergence (CD) [41]. Thus, we get the following stochastic gradient updates for $\mathbf{W}$ from CD,

$$\frac{\partial \log p(\mathbf{v})}{\partial W_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{7.17}$$

where we ignore the biases of both hidden and observation layers. By taking the derivative w.r.t. W_{ij}, it yields

$$\frac{\partial \ln p(\mathbf{v})}{\partial W_{ij}} = \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) v_i h_j - \sum_{\mathbf{h},\mathbf{v}} p(\mathbf{h},\mathbf{v}) v_i h_j \tag{7.18}$$

$$= p(h_j = 1|\mathbf{v}) v_i - \sum_{\mathbf{v}} p(\mathbf{v}) p(h_j = 1|\mathbf{v}) v_i \tag{7.19}$$

And update $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ until convergence with gradient descent

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \frac{\partial \log p(\mathbf{v})}{\partial \boldsymbol{\theta}} \tag{7.20}$$

where $\boldsymbol{\theta}$ is the weight and biases, and η is the learning rate. Given the training data $\mathcal{D}$, we need to learn the weight $\mathbf{W}$ in Eq. 7.18. The CD learning algorithm with K steps is listed below:

Data: the training set $\mathcal{D}$, iterations T and learning rate α
Result: output the weight $\mathbf{W}$ and bias $\mathbf{b}$, $\mathbf{c}$
initialize the weights $\mathbf{W}$ and $\mathbf{b}, \mathbf{c}$;
set iteration index $t = 0$;
while $t < T$ **do**
  for *Each* $\mathbf{x}_i \in \mathcal{D}$ **do**
    $\mathbf{v}^0 = \mathbf{x}_i$;
    for *k=0,...,K-1* **do**
      sample $h_j^t \sim p(h_j|\mathbf{v}^k),\ \forall j \in \{1, 2, ..., n\}$;
      sample $v_i^t \sim p(v_i|\mathbf{h}^k),\ \forall i \in \{1, 2, ..., d\}$;
    end
    $\Delta W_{ij} = v_i^0 h_j^0 - v_i^K h_j^K$;
    $W_{ij} = W_{ij} + \alpha \Delta W_{ij}$;
  end
end

Algorithm 17: RBMs learning algorithm

Notice that momentum learning can also be used to make the algorithm stable. In the above Algorithm 17, we use the current model to predict the hidden variables (via Gibbs sampling). Then, given the hidden units (predictions), we use it to predict the ground truth (observations). If it does not match well, then we need to update model $(\mathbf{W}, \mathbf{b})$. The true expectation in Eq. 7.18 needs sampling steps $K \to \infty$.

Since computing the average over the true model distribution is intractable (because its complexity is still exponential in the size of the smallest layer), the parameters updating can be calculated with an efficient stochastic descent method with a few sampling steps via CD [41] to approximate that. Recent study shows that RBM is a Universal Approximators [56]. More specifically, if we increase the number of hidden units of an RBM, there are weight values for the new units that guarantee improvement in the training log-likelihood. In other words, an RBM can represent any distribution with enough hidden variables. In addition, we can extend the binary $\mathbf{v}$ to handle continuous values, which can be modeled with Gaussian distribution (or Gaussian RBMs).

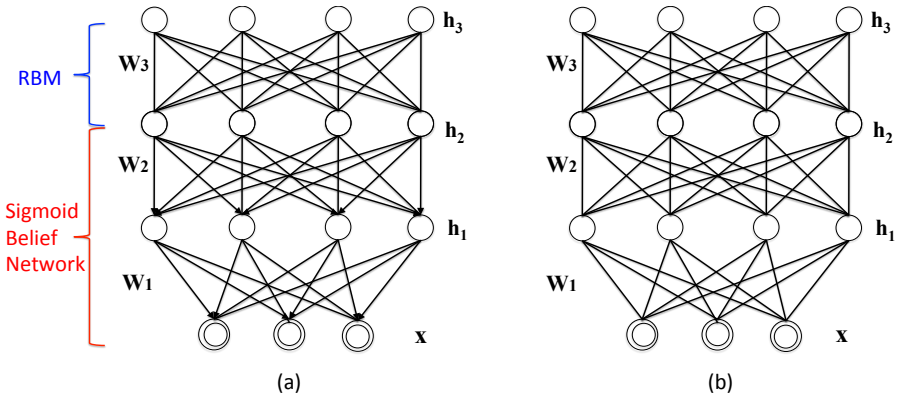


Figure 7.2. It shows two deep learning structure models: (a) deep belief networks, where the top layer is an RBM and the low level layer is sigmoid belief networks (there is downward arrows representing generative model); (b) deep Boltzmann machines (composed by multi-layer RBMs), with no hidden-to-hidden or visible-to-visible connections.

7.2 Deep belief networks

In this part, we will introduce another popular deep structure model for feature learning, namely deep belief network [41](or DBN), where the top layer is an RBM and the lower deep structure is a multi-layer directed belief network, refer to Fig. 7.2(a).

A DBN with L layers models the joint distribution between observed variables $\mathbf{v}$ and L hidden layers $\mathbf{h}^{(l)}, l = \{1, ..., L\}$ made of binary units $\mathbf{h}^{(l)}$ (binary variables), as follows:

$$p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, ..., \mathbf{h}^{(L)}) = p(\mathbf{v}|\mathbf{h}^{(1)})p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}), ..., p(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)}) \tag{7.21}$$

Based on the joint likelihood in Eq. 7.21, we can think the top layer in DBN is interpreted as an RBM $p(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)})$ and the lower layers as a directed sigmoid belief network (top-down, generative model). The motivation for deep learning is that deep architectures [56] can represent functions much more efficiently (in terms of number of required parameters) than shallow ones. Moreover, the layer-by-layer stacked structure can learn high level abstract representations, which may capture intrinsic data distribution.

Suppose the DBN used here has L layers, and the weight for each layer is indicated as $\mathbf{W}_l$ for $l = \{1, .., \mathrm{L}\}$. Specifically, we think an RBM is a 2-layer DBN model, with weight $\mathbf{W}_1$, refer to Fig. 7.3. To learn DBN, a greedy learning approach is proposed

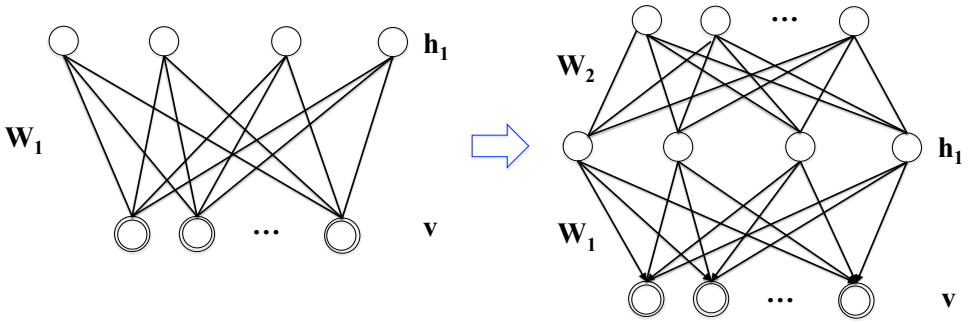


Figure 7.3. (a) 1-layer Restricted Boltzmann machine (RBM); (b) 2-layer DBN, which is equal to 1-layer RBM with tiled weight $\mathbf{W}_2 = \mathbf{W}_1^T$.

to estimate the weights layer by layer [41]. Basically, it trains an RBM model given the observations, and its output as the input to the next layer of the network at each stage in Algorithm 18.

In the following part, we will discuss that the greedy learning process of DBN will always improve the low bound to approximate the maximum likelihood.

$$
\begin{aligned}
S(\theta) &= \log \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}; \theta) \\
&= \log \sum_{\mathbf{h}} q(\mathbf{h}) \frac{p(\mathbf{v}, \mathbf{h}; \theta)}{q(\mathbf{h})} \\
&\geq \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{v}, \mathbf{h}; \theta)}{q(\mathbf{h})} \qquad (7.22) \\
&= \sum_{\mathbf{h}} q(\mathbf{h}) \log p(\mathbf{v}, \mathbf{h}; \theta) - \sum_{\mathbf{h}} q(\mathbf{h}) \log q(\mathbf{h}) \\
&= \sum_{\mathbf{h}} q(\mathbf{h}) \log [p(\mathbf{h}|\mathbf{v}; \theta) p(\mathbf{v}; \theta)] - \sum_{\mathbf{h}} q(\mathbf{h}) \log q(\mathbf{h}) \qquad (7.23) \\
&= \sum_{\mathbf{h}} q(\mathbf{h}) p(\mathbf{v}; \theta) - \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{q(\mathbf{h})}{p(\mathbf{h}|\mathbf{v}; \theta)} \\
&= \log p(\mathbf{v}; \theta) - KL(q || p_{\mathbf{h}|\mathbf{v}}) \\
&= L(q, \theta)
\end{aligned}
$$

where we use Jensen inequality in Eq. 7.22. The bound is tight when $q(\mathbf{h})$ is the true posterior probability $p(\mathbf{h}|\mathbf{v}; \theta)$. Also, it indicates that maximizing the log probability of the data is exactly the same as minimizing the Kullback-Leibler (KL) divergence.

According to Eq. 7.23, we have

$$\begin{aligned} \log p(\mathbf{v};\theta) &\geq \sum_{\mathbf{h}} q(\mathbf{h}) \log[p(\mathbf{h}|\mathbf{v};\theta)p(\mathbf{v};\theta)] - \sum_{\mathbf{h}} q(\mathbf{h}) \log q(\mathbf{h}) \\ &= \sum_{\mathbf{h}} q(\mathbf{h}) \log[p(\mathbf{h}|\mathbf{v};\mathbf{W}_2)p(\mathbf{v};\mathbf{W}_1)] + H(q) \end{aligned} \tag{7.24}$$

where $H(q)$ is the entropy of $q(\mathbf{h})$. Suppose we freeze $q(\mathbf{h}) = p(\mathbf{h}|\mathbf{v})$, which will keep the bound tight in Eq. 7.22. Then we want to learn the second layer of DBN with weight $\mathbf{W}_2$. The greedy learning algorithm for DBN is to fix $\mathbf{W}_1$ and attempt to learn weight $\mathbf{W}_2$ from $\mathbf{h}_1$ by maximizing the low bound in Eq. 7.24. If we have learnt $\mathbf{W}_1$, then to maximize Eq. 7.24 is the same to maximize the following

$$\sum_{\mathbf{h}} q(\mathbf{h}) \log p(\mathbf{h}|\mathbf{v};\mathbf{W}_2) \tag{7.25}$$

In other word, it is to maximize the likelihood of the second layer of RBM given the hidden layer $\mathbf{h}_1$ drawn from $q(\mathbf{h}|\mathbf{v})$ as the input. Thus, the greedy learning algorithm for DBN model can always improve the low bound and further maximize the likelihood over the training data.

Data: the training set $\mathcal{D}$, the number of laeyrs L and learning rate α
Result: output the weight $\{\mathbf{W}_l\}_{l=1}^{L}$ and bias $\mathbf{b}$, $\mathbf{c}$
initialize the weights $\mathbf{W}_1$ and $\mathbf{b}, \mathbf{c}$;
set iteration index l= 0;
while $l < L$ **do**
  if $l == 0$ **then**
    the input data is from observations $\mathbf{v}$;
  else
    the input data is $\mathbf{h}_l$;
  end
  Learn the l-th weight $\mathbf{W}_l$, given the input from the above step;
  Freeze $\mathbf{W}_l$, and use the samples from $q(\mathbf{h}_{l+1}|\mathbf{W}_l)$ as the training data to learn the next layer of binary features with an RBM ;
end

Algorithm 18: Greedy layer-wise learning algorithm

A further approach has been proposed to stack RBMs into a deep Boltzmann ma-

chine (DBM) [57], which is the fully generative model and can be trained by maximizing the joint likelihood. We will introduce this model in the next part.

7.3 Deep Boltzmann machines

A Boltzmann machine is a fully connected stochastic and generative model. Compared to an RBM, there is no restrictions between visible-to-visible or hidden-to-hidden connections. A Deep Boltzmann machine basically stacks multiple layers of Boltzmann machines together to construct a hierarchal structure, in order to learn meaningful high level representations. In this chapter, we will consider a more general case, namely k-fan DBM. A k-fan DBM can be viewed as a composition of k unimodal undirected pathways. Each path-way can be a DBM and can be pretrained separately in a completely unsupervised fashion, which make it possible to leverage a large supply of unlabeled data. Then, we can fine-tune the whole model by maximizing the the joint likelihood. The right graph in Fig. 7.4 is a 3-fan DBM, where each branch is an DBM.

Suppose we have a set of visible inputs $\mathbf{v}^x \in \{0,1\}^D$ and a sequence of layers of hidden units for each input. For clarity, we start by modeling each input using a separate two-layer case. For input $\mathbf{v}^x$, and its two layers of hidden units $\mathbf{h}^{(x1)} \in \{0,1\}^{n_{x1}}$ and $\mathbf{h}^{(x2)} \in \{0,1\}^{n_{x2}}$, the probability for the visible vector $\mathbf{v}^x$ is given by

$$P(\mathbf{v}^x;\theta_x) = \sum_{\mathbf{h}^{(x1)},\mathbf{h}^{(x2)}} P(\mathbf{v}^x,\mathbf{h}^{(x1)},\mathbf{h}^{(x2)};\theta_x) \tag{7.26}$$

$$= \frac{1}{\mathcal{Z}(\theta_x)} \sum_{\mathbf{h}^{(x1)},\mathbf{h}^{(x2)}} \exp\left(\sum_{k,j} W_{k,j}^{(x1)} v_k^x h_j^{(x1)} + \sum_{j,l} W_{j,l}^{(x2)} h_j^{(x1)} h_l^{(x2)} \right) \tag{7.27}$$

where $\theta_x = \{\mathbf{W}^{(x1)}, \mathbf{W}^{(x2)}\}$. Note that we only consider the binary observation (can be easily extended into real value case with Gaussian RBMs) and ignore biases for both visible and hidden units for clarity. Analogously, we can get the likelihoods for $\mathbf{v}^y$ and $\mathbf{v}^z$ respectively in the same formulas, but with different subscripts.

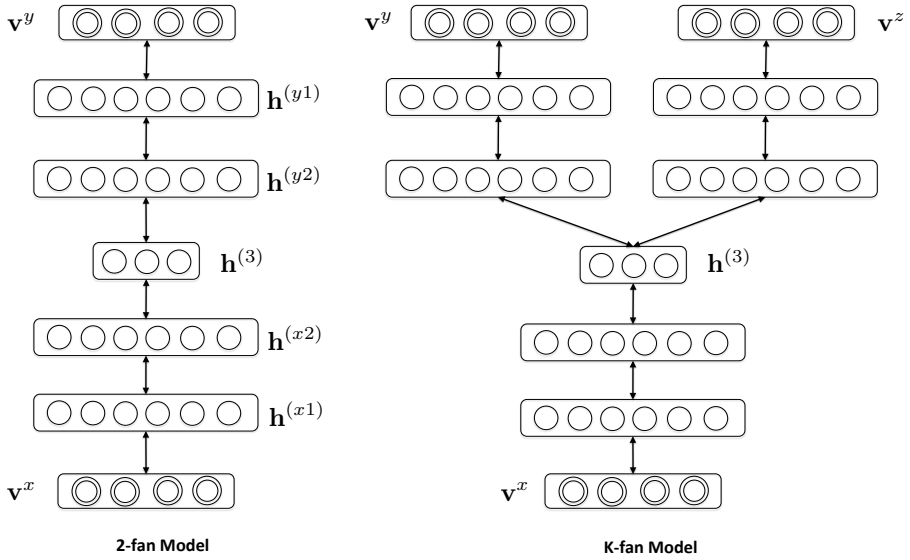


Figure 7.4. The left is a 2-fan deep DBM, and the right shows a 3-fan deep DBM, where each branch is a DBM with bi-directional connections.

7.3.1 Joint likelihood

To form our k-fan deep model (considering k=3), we combine the three models from $\mathbf{v}^x$, $\mathbf{v}^y$ and $\mathbf{v}^z$, by adding an additional layer of binary hidden units on top of them. The resulting graphical model is shown in the right panel of Fig. 7.4. The joint distribution over the multi-modal inputs can be written as:

$$\begin{aligned} P(\mathbf{v}^x, \mathbf{v}^y, \mathbf{v}^z; \theta) = & \sum_{\mathbf{h}^{(x2)}, \mathbf{h}^{(y2)}, \mathbf{h}^{(z2)}, \mathbf{h}^{(3)}} P(\mathbf{h}^{(x2)}, \mathbf{h}^{(y2)}, \mathbf{h}^{(z2)}, \mathbf{h}^{(3)}) \\ & \Big(\sum_{\mathbf{h}^{(x1)}} P(\mathbf{v}^x, \mathbf{h}^{(x1)}, \mathbf{h}^{(x2)}) \Big) \Big(\sum_{\mathbf{h}^{(y1)}} P(\mathbf{v}^y, \mathbf{h}^{(y1)}, \mathbf{h}^{(y2)}) \Big) \\ & \Big(\sum_{\mathbf{h}^{(z1)}} P(\mathbf{v}^z, \mathbf{h}^{(z1)}, \mathbf{h}^{(z2)}) \Big) \end{aligned} \tag{7.28}$$

where $\mathbf{W}^{(x3)}$, $\mathbf{W}^{(y3)}$ and $\mathbf{W}^{(z3)}$ are respectively the top layer weights which connected to the top shared layer $\mathbf{h}^{(3)}$ for each input. And we need to estimate the parameters $\theta = \{\theta_x, \theta_y, \theta_z, \mathbf{W}^{(x3)}, \mathbf{W}^{(y3)}, \mathbf{W}^{(z3)}\}$, where we ignore the biases.

Notice that we have observations from k branches, and we can use maximum likelihood to estimate all parameters. However, fully connected Boltzmann machine is hard to learn especially considering the intra layer connections. In this part, we

only consider an easy case with inter layer connections. Moreover, unlike DBN, we have fully observations across all branches, thus we can incorporate an top-down fine-tuning step in addition to an initial bottom-up pass.

7.3.2 Parameter learning

In learning stage, all inputs are available. Thus, we can use CD to learn the shared presentation and model parameter effectively. In practice, we divide the parameter estimation into two stages: parameter initialization and fine-tuning. The parameter initialization stage initializes each branch parameters separately (e.g. contrastive divergence) and the fine-tuning focuses on learning the joint representations shared by different modalities. More specifically, because of the joint multimodal deep structure, we can estimate the model parameters by maximizing the joint likelihood in Eq. 7.28.

Parameter initialization: we first use pretraining to initialize the weights of each branch separately. Basically, it is to learn a stack of RBMs greedily layer-by-layer. To put it simply, the learned features of the current layer RBM are treated as the "data" to train the next RBM in the stack. Assume that we have a training set $\mathcal{D} = \langle \mathbf{v}_i^x, \mathbf{v}_i^y, \mathbf{v}_i^z \rangle_{i=1}^N$. Then, for each branch $\mathbf{v}^x$, we can use RBMs to initialize the weights $\{\mathbf{W}^{(x1)}, \mathbf{W}^{(x2)}, \mathbf{W}^{(x3)}\}$ in the layer-wise manner mentioned above.

$$
\begin{aligned}
p(v_i^x = 1|\mathbf{h}^{(x1)}) &= \sigma(\sum_j W_{ij}^{(x1)} h_j^{(x1)}) \\
p(h_j^{(x1)}|\mathbf{v}^x) &= \sigma(\sum_i W_{ij}^{(x1)} v_i^x + \sum_i W_{ij}^{(x1)} v_i^x)
\end{aligned} \tag{7.29}
$$

Then, we update the model parameters with CD. Basically, we use Eq. 7.29 to infer the visible and hidden variables and then calculate the gradient w.r.t. $\mathbf{W}^{(x1)}$ via Eq. 7.17. Further, we can update model parameters with gradient descent. After we learn the current RBM, we can use its output as the input to train the next RBM and so on. In Eq. 7.29, we double count $\sum_i W_{ij}^{(x1)} v_i^x$, please refer to [58] for more details.

Parameter fine-tuning: because this model is intractable, we use an efficient approximate learning and inference, such as mean-field method, to estimate data dependent expectations, and an MCMC based stochastic approximation procedure to approximate the model's expected sufficient statistics. In variational learning [58, 59, 60]

the true posterior distribution over latent variables $p(\mathbf{h}|\mathbf{v};\theta)$ for each training vector $\mathbf{v}$, is replaced by an approximate posterior $q(\mathbf{h}|\mathbf{v};\mu)$ and the parameters are updated by following the gradient of a lower bound on the log-likelihood:

$$\ln P(\mathbf{v};\theta) \geq \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v};\mu)\ln p(\mathbf{v},\mathbf{h};\theta) + \mathcal{H}(q) \tag{7.30}$$

$$= \ln p(\mathbf{v};\theta) - KL[p(\mathbf{h}|\mathbf{v};\mu)||q(\mathbf{h}|\mathbf{v};\mu)] \tag{7.31}$$

where $\mathbf{h} = \{\mathbf{h}^{(x1)}, \mathbf{h}^{(x2)}, \mathbf{h}^{(y1)}, \mathbf{h}^{(y2)}, \mathbf{h}^{(z1)}, \mathbf{h}^{(z2)}, \mathbf{h}^{(3)}\}$, $\mathbf{v} = \{\mathbf{v}^x, \mathbf{v}^y, \mathbf{v}^z\}$, and μ is the mean-field approximation to hidden variable (see further), and $\mathcal{H}(\cdot)$ is the entropy functional. To maximize the log-likelihood of the training data, is to find parameters that minimize the Kullback–Leibler divergences between the approximating and true posteriors.

Similar to [61], we use the naive mean-field approach, with fully factorized distribution to approximate the true posterior:

$$q(\mathbf{h}|\mathbf{v};\mu) = \Big(\prod_i q(h_i^{(3)}|\mathbf{v})\Big(\prod_k q(h_k^{(x1)}|\mathbf{v})\prod_j q(h_j^{(x2)}|\mathbf{v})\Big)$$
$$\Big(\prod_k q(h_j^{(y1)}|\mathbf{v})\prod_j q(h_j^{(y2)}|\mathbf{v})\Big)\Big(\prod_k q(h_k^{(z1)}|\mathbf{v})\prod_j q(h_j^{(z2)}|\mathbf{v})\Big) \tag{7.32}$$

where $\mu = \{\mu_x^{(1)}, \mu_x^{(2)}, \mu_y^{(1)}, \mu_y^{(2)}, \mu_z^{(1)}, \mu_z^{(2)}, \mu^{(3)}\}$ are the mean-field parameters with

$$q(h^{(xl)} = 1) = \mu_x^{(l)}, \text{for } l = 1, 2. \tag{7.33}$$

$$q(h^{(yl)} = 1) = \mu_y^{(l)}, \text{for } l = 1, 2. \tag{7.34}$$

$$q(h^{(zl)} = 1) = \mu_z^{(l)}, \text{for } l = 1, 2. \tag{7.35}$$

$$q(h^{(3)} = 1) = \mu^{(3)}. \tag{7.36}$$

Then μ can be used to update the hidden variables in the data dependent item in Eq. 7.17. And the model dependent hidden variables in Eq. 7.17 can be sampled with MCMC. Then, the model parameter can be updated with CD algorithm according to Eq. 7.20.

7.4 Deep neural network

In this section, will will discuss deep neural network, which consists of fully connected multiple hidden layers and the top layer is usually used for classification. We will first introduce one hidden layer discriminative RBMs for classification and then discuss about multi-layers deep neural network.

7.4.1 One hidden layer model

Given the training pair $(\mathbf{x}_i,\ y_i)$, the linear classification model (such as perceptron learning and SVM) has the general form $f(x) = \mathbf{w}^T\mathbf{x}_i + b$ for target prediction. However, the linear mapping function f cannot handle nonlinear hyperplane when the data is not linear separable. Thus, it is meaningful to exploit nonlinear mapping, in order to capture the intrinsic geometric structure of the data. In this part, we will introduce discriminative RBM to learn non-linear mapping for classification.

Considering the generative RBM model in Eq. 7.1, it can learn the hidden representations $\mathbf{h}$ which can effectively capture the latent data structure. Thus, we can use the latent representation as the input to any classifiers, such as SVM and logistic regression to make prediction. More specifically, it can be thought as an discriminative RBM, where the hidden units will be used as the input to predict the label $\hat{y}$. If the predict $\hat{y}$ does not match y, then we can update the whole model with backpropagation (we use back propagation to calculate the gradients in each layer, and then update it with gradient based methods, such as stochastic gradient descent and L-BFGS). In the following part, we will discuss the model and mathematical operations in details.

Given the training data $\mathcal{D} = (\mathbf{x}_i,\ y_i)_{i=1}^N$, we can learn a discriminative RBM classifier, where the full connection weight between the visible and the hidden is W_1 and the weight between the hidden layer and the output label is $\mathbf{w}$. Note the visible to the hidden layer is nonlinear mapping, while the hidden layer to the label output is linear mapping. Thus, given the visible $\mathbf{x}$, we have the following equations:

$$\mathbf{h} = f(\mathbf{x}) = \sigma(W_1^T\mathbf{x} + b_1); \tag{7.37}$$

$$\hat{y} = \mathbf{w}^T\mathbf{h} + b \tag{7.38}$$

where $\sigma(x) = 1/(1+\exp(-x))$. If we use the square error as the objective function, then we need to minimize:

$$\mathcal{L}(W_1, \mathbf{w}; \mathcal{D}) = \frac{1}{2}(y - \hat{y})^2 \tag{7.39}$$

Notice that except least square, we can use other loss functions too, such as hinge loss and cross entropy.

Take the derivative w.r.t. $\hat{y}$ in Eq.7.39, we have

$$\frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \hat{y}} = -(y - \hat{y}) \tag{7.40}$$

If we use the linear prediction model in Eq. 7.38, then we get

$$\frac{\partial \hat{y}}{\partial \mathbf{w}} = \mathbf{h} \tag{7.41}$$

$$\frac{\partial \hat{y}}{\partial \mathbf{h}} = \mathbf{w} \tag{7.42}$$

And through the chain rule, we have

$$\frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \hat{y}} \frac{\partial \hat{y}}{\mathbf{w}} = -\mathbf{h}(y - \hat{y})^T \tag{7.43}$$

$$\frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \mathbf{h}} = \frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}} = -\mathbf{w}(y - \hat{y}) \tag{7.44}$$

Similarly, through simple linear algebra, we can get the following gradient w.r.t. W_1

$$\frac{\partial \mathbf{h}}{\partial W_1} = \mathbf{x}[\mathbf{h}(1 - \mathbf{h})]^T \tag{7.45}$$

And further,

$$\begin{aligned}
\frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial W_1} &= \frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial W_1} \\
&= \frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial W_1} \\
&= -\mathbf{x}[\mathbf{h}(1 - \mathbf{h})]^T [\mathbf{w}(y - \hat{y})]^T \\
&= -\mathbf{x}[\mathbf{w}(y - \hat{y})\mathbf{h}(1 - \mathbf{h})]^T
\end{aligned} \tag{7.46}$$

Notice that the gradient of sigmoid function $\sigma(x) = 1/(1 + exp(-x))$ is

$$\begin{aligned}\frac{\partial \sigma(x)}{\partial x} &= -(1 + exp(-x))^{-2} exp(-x)(-1) \\ &= \frac{exp(-x)}{1 + exp(-x)} \frac{1}{1 + exp(-x)} \\ &= \sigma(x)(1 - \sigma(x))\end{aligned} \tag{7.47}$$

Thus, Eq. 7.45 can be easily induced according Eq. 7.47

In the following, we will introduce to estimate model parameters based on two widely used learning algorithms: stochastic gradient descent (SGD) and mini-batch based learning. Both methods are gradient based approaches to update the model parameters. The only difference is that stochastic gradient descent updates the model based on each instance, while the mini-batch learning updates the model based the gradient of a batch (which contains multiple instances, for example 100 instances).

Stochastic gradient descent:

Data: the training set $\mathcal{D}$, iterations T and learning rate α

Result: output the weight W_1, $\mathbf{w}$ and bias $b,\ b_1$

initialize the weights W_1, $\mathbf{w}$ and $b,\ b_1$;

set iteration index t = 0;

while $t < T$ **do**

  for *Each* $(\mathbf{x}_i, y_i) \in \mathcal{D}$ **do**

    $\mathbf{h}_i = \sigma(W_1^T \mathbf{x}_i + b_1)$ via Eq. 7.37;

    $\hat{y}_i = \mathbf{w}^T \mathbf{h}_i + b$ via Eq. 7.38;

    $\nabla \mathbf{w} = -\mathbf{h}_i (y_i - \hat{y}_i)$ via Eq. 7.43;

    $\nabla W_1 = -\mathbf{x}_i [\mathbf{w}(y_i - \hat{y}_i)\mathbf{h}(1 - \mathbf{h})]^T$ via Eq. 7.46;

    $\mathbf{w} = \mathbf{w} - \alpha \nabla \mathbf{w}$;

    $W_1 = W_1 - \alpha \nabla W_1$;

  end

end

Algorithm 19: Learning discriminative RBMs with stochastic gradient descent

Batched based learning: the basic idea is to divide the whole training set into batches,

and for each batch we need to minimize

$$\mathcal{L}(W_1, \mathbf{w}) = \frac{1}{2}\sum_j (y_j - \hat{y}_j)^2 \tag{7.48}$$

Then, the gradients w.r.t. W_1 and $\mathbf{w}$ can be written as

$$\frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial \mathbf{w}} = -\sum_j \mathbf{h}_j (y_j - \hat{y}_j) \tag{7.49}$$

$$\frac{\partial \mathcal{L}(W_1, \mathbf{w})}{\partial W_1} = -\sum_j \mathbf{x}_j [\mathbf{w}(y_j - \hat{y}_j)\mathbf{h}_j(1 - \mathbf{h})_j]^T \tag{7.50}$$

Data: the training set $\mathcal{D}$, the batch size m, iterations T and learning rate α
Result: output the weight W_1, $\mathbf{w}$ and bias $b,\ b_1$
initialize the weights W_1, $\mathbf{w}$ and $b_1,\ b$;
set iteration index t = 0;
divide the training set into n batches;
while $t < T$ **do**
  // for each batch ;
  for $i = 1, ..., n$ **do**
    for $j = 1, ..., m$ **do**
      $\mathbf{h}_j = \sigma(W_1^T \mathbf{x}_j + b_1)$ via Eq. 7.37;
      $\hat{y}_j = \mathbf{w}^T \mathbf{h}_j + b$ via Eq. 7.38;
    end
    $\nabla \mathbf{w} = -\sum_j \mathbf{h}_j (y_j - \hat{y}_j)$ via Eq. 7.49;
    $\nabla W_1 = -\sum_j \mathbf{x}_j [\mathbf{w}(y_j - \hat{y}_j)\mathbf{h}_j(1 - \mathbf{h}_j)]^T$ via Eq. 7.50;
    $\mathbf{w} = \mathbf{w} - \alpha \nabla \mathbf{w}/m$;
    $W_1 = W_1 - \alpha \nabla W_1/m$;
  end
end

Algorithm 20: Mini-batch learning algorithm

In general, SGD works well when the objective function has a lot of local minimum/maximum. Apparently, the gradient of a single data is much noisier compared to the average gradient of batch. In this case, a noisy data will have a large step to move the model out of local minima into a region that might be more optimal. While

minibatch learning takes the average gradient over all instances in the batch, which intentionally makes small enough step (to be computationally tractable). However, it might be easily trapped into local minimum and cannot jerk out to better solution.

On the other hand, the minibatch learning in general more computationally efficient when we exploit parallelism across processors/machines. More specifically, we can exploit matrix multiplication and parallelism across multiple CPU/GPUs to get much better throughput.

7.4.2 Multi-layer deep neural network

We can generalize the one hidden layer case to multi-layer deep neural network for classification. As we stack more layers, we can get the intermediate layer which consist of lower level layers with shared representations. And we can also learn high level abstract representations which are meaningful to capture discriminative features or parts for classifications. In this part, we will discuss the fully connected deep neural network (DNN), which forwards the input with matrix multiplication and generates output via activation function (i.e. sigmoid function) and further the output can be used as input to construct a very deep structure.

Recall that a deep belief network (DBN) is composed of stacked RBMs [41] learned layer by layer greedily, where the top layer is an RBM and the lower layers can be interpreted as a directed sigmoid belief network [47]. Thus we can use DBN to initialize the fully connected deep neural networks. After we learn the representation for the data, we use the output (or learned representations) as the input of any classifier, such as logistic regression, SVM or softmax function. In other words, we add another layer with weight parameters $\mathbf{w}$ to predict the label in the supervised case, shown in Fig. 7.1. More specifically, we need to compute the conditional probability $p(y|\mathbf{h})$ for classification problems. Then, we can maximize the conditional likelihood as follows:

$$\begin{aligned}\mathcal{L}(\mathcal{D};\Theta) =& max_{\Theta} \log \prod_i (p(\hat{y}_i|\mathbf{h}_i;\Theta))^{y_i} \\ =& min_{\Theta} - \sum_i y_i \log p(\hat{y}_i|\mathbf{h}_i;\Theta)\end{aligned} \tag{7.51}$$

where $\Theta = \{\mathbf{W}_i\}_{i=1}^{\mathrm{L}}$ (ignore biases), and non-linear mappings $\mathbf{h}_i$ is the output with L

layers neural network, s.t.

$$\mathbf{h}_i = \underbrace{f_{\mathrm{L}} \circ f_{\mathrm{L}-1} \circ \cdots \circ f_1}_{\text{L times}}(\mathbf{x}_i) \tag{7.52}$$

where $\circ$ indicates the function composition, and f_l is logistic function or other non-linear mapping with the weight parameter $\mathbf{W}_l$ respectively for $l = \{1, .., \mathrm{L}\}$. With a bit abuse of notation, we denote $\mathbf{h}_i = f_{1\to\mathrm{L}}(\mathbf{x}_i)$.

$p(\hat{\mathbf{y}}_i|\mathbf{h}_i;\Theta)$ in Eq. 7.51 is the softmax function below

$$p(\hat{\mathbf{y}}_i|\mathbf{h}_i;\Theta) = \frac{\exp(\mathbf{h}_i^T\mathbf{w}(:,i))}{\sum_k \exp(\mathbf{h}_i^T\mathbf{w}(:,k))} \tag{7.53}$$

where $\mathbf{w} \in \mathbb{R}^{n\times K}$ is the weight in the top layer for classification (bias is ignored for clarity).

In the top layer, we have the top layer weight $\mathbf{w}$ and the hidden representation $\mathbf{h}_i$. We can compute the gradients w.r.t. $\mathbf{w}$ first, and then back propagate to the low level layers. To compute the derivative of $\mathbf{w}$, we have the following

$$\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{w}} = \sum_i \mathbf{h}_i(p(\hat{\mathbf{y}}_i|\mathbf{x}_i) - \mathbf{y}_i) \tag{7.54}$$

where $\mathbf{y}_i$ is 1 of K encoding of y_i and $\mathbf{h}_i$ is the output of L layers of neural network. In the following, we will discuss in details on how to calculate the derivative of softmax function w.r.t. $\mathbf{w}$ in Eq. 7.54.

Given $p(\hat{\mathbf{y}}) = [p(\hat{y}_1), ..., p(\hat{y}_K)]$, we will first compute the gradient w.r.t. $\alpha_j(\Theta) = (\mathbf{h}_i^T\mathbf{w}(:,j))$, and then generalize it to $k \neq j$. To make the derivative easy to understand, we assume that

$$p(\hat{y}_j|\mathbf{h}_i;\Theta) = \frac{\exp(\alpha_j(\Theta))}{\sum_k \exp(\alpha_k(\Theta))} \tag{7.55}$$

Then take the derivative w.r.t. $\alpha_j(\Theta)$

$$\frac{\partial y_j \log p(\hat{y}_j|\mathbf{h}_i;\Theta)}{\partial \alpha_j}$$

$$= \frac{y_j}{p(\hat{y}_j)} \frac{\exp(\alpha_j(\Theta)) \sum_k \exp(\alpha_k(\Theta)) - \exp(\alpha_j(\Theta))\exp(\alpha_j(\Theta))}{[\sum_k \exp(\alpha_k(\Theta))]^2}$$
$$= y_j(1 - p(\hat{y}_j)) \tag{7.56}$$

Similarly, $\forall k \neq j$ and its prediction $p(\hat{y}_k)$, we take the derivative w.r.t. $\alpha_j(\Theta)$,

$$\frac{\partial y_k \log p(\hat{y}_k|\mathbf{h}_i;\Theta)}{\partial \alpha_j}$$
$$= \frac{y_k}{p(\hat{y}_k)} \frac{-\exp(\alpha_k(\Theta))\exp(\alpha_j(\Theta))}{[\sum_s \exp(\alpha_s(\Theta))]^2}$$
$$= -y_k p(\hat{y}_j) \tag{7.57}$$

Finally, we can yield the following gradient w.r.t. $\alpha_j(\Theta)$

$$\frac{\partial p(\hat{\mathbf{y}})}{\partial \alpha_j} = \sum_j \frac{\partial y_j \log p(\hat{y}_j|\mathbf{h}_i;\Theta)}{\partial \alpha_j}$$
$$= \frac{\partial \log p(\hat{y}_j|\mathbf{h}_i;\Theta)}{\partial \alpha_j} + \sum_{k \neq j} \frac{\partial \log p(\hat{y}_k|\mathbf{h}_i;\Theta)}{\partial \alpha_j}$$
$$= y_j - y_j p(\hat{y}_j) - \sum_{k \neq j} y_k p(\hat{y}_j)$$
$$= y_j - p(\hat{y}_j)(y_j + \sum_{k \neq j} y_k) = y_j - p(\hat{y}_j) \tag{7.58}$$

where we use the results in Eqs. 7.56 and 7.57. Thus, we can infer $\frac{\partial p(\hat{\mathbf{y}})}{\partial \boldsymbol{\alpha}} = \mathbf{y} - p(\hat{\mathbf{y}})$, and also get the gradient w.r.t. the top layer weight in Eq. 7.54.

Similarly, we can take the derivative w.r.t. $\mathbf{h}_i$ and get

$$\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_i} = \sum_i \mathbf{w}(p(\hat{\mathbf{y}}_i|\mathbf{x}_i) - \mathbf{y}_i) \tag{7.59}$$

Then, we can use backpropagation to calculate gradients w.r.t. parameters in the lower level layers and update them with gradient based methods. In the following part, we will discuss backpropagation, which provides a very intuitive explanation and it is easy to implement in practice.

7.4.3 Understanding backpropagation

The purpose is to find Θ, which can minimize the objective function $\mathcal{L}(\mathcal{D};\Theta)$ in Eq. 7.51. We will use the example from the i-th layer to the $i+1$-th layer as the study case, and the corresponding function mapping is $\mathbf{h}_{i+1} = f(\mathbf{h}_i)$ with the weight W_{i+1}. Note that $\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}}$ has been determined in the backpropagation stage, and we want to use it to infer the gradient information in the i-th layer. Then, according to chain rule, we can compute the gradients w.r.t. W_{i+1} and $\mathbf{h}_i$ respectively

$$\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial W_{i+1}} = \frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}} \frac{\partial \mathbf{h}_{i+1}}{\partial W_{i+1}} \tag{7.60}$$

If we use sigmoid function, Eq. 7.60 can be further written as

$$\begin{aligned} &\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial W_{i+1}} = \frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}} f(\mathbf{h}_i)(1 - f(\mathbf{h}_i))\mathbf{h}_i^T \\ \Longrightarrow\ &\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial W_{i+1}} = \frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}} \mathbf{h}_{i+1}(1 - \mathbf{h}_{i+1})\mathbf{h}_i^T \end{aligned} \tag{7.61}$$

where we use the derivative of sigmoid function in Eq. 7.47. We can also get the following result w.r.t. $\mathbf{h}_i$

$$\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_i} = \frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i} = \frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}} \mathbf{h}_{i+1}(1 - \mathbf{h}_{i+1})W_{i+1}^T \tag{7.62}$$

Apparently, given the $\frac{\partial \mathcal{L}(\mathcal{D};\Theta)}{\partial \mathbf{h}_{i+1}}$, we can use the chain rule to get the gradients w.r.t. W_{i+1} and $\mathbf{h}_i$ via Eqs. 7.61 and 7.62 effectively. Through the same process in a top-down manner, we can get all gradients w.r.t. to weights in all layers and then use any gradient-based methods to optimize the objective Eq. 7.51.

Error backpropagation: assume that the difference between the groundtruth and prediction can be indicated as $\Delta\mathbf{y} = \mathbf{y} - \hat{\mathbf{y}}$. Then we can propagate the error back with an efficient matrix multiplication. To make it easy to understand, we use the example from the i-th layer to the $i+1$-th layer as the study case, and the corresponding function mapping $\mathbf{h}_{i+1} = f(\mathbf{h}_i)$.

Let's first consider the simple linear mapping, such as $\mathbf{h}_{i+1} = f(\mathbf{h}_i) = W_{i+1}^T\mathbf{h}_i$. In the forward case, we propagate $\mathbf{h}_i$ into $\mathbf{h}_{i+1}$ with the weight W_{i+1}. If there is any

change w.r.t. $\mathbf{h}_i$, s.t. for any input $\Delta\mathbf{h}_i$, we have the forward equation

$$\Delta\mathbf{h}_{i+1} = W_{i+1}^T \Delta\mathbf{h}_i \tag{7.63}$$

Thus, we can propagate the error through matrix multiplication. Considering the symmetric property of the linear model, any change w.r.t. $\mathbf{h}_{i+1}$ can be backpropagated to $\mathbf{h}_i$ in the same way with matrix multiplication. To back propagate error, we assume further that the error in the layer $i+1$ is $\Delta\mathbf{h}_{i+1}$, then we can get the error at i-th layer as $\Delta\mathbf{h}_i = \Delta\mathbf{h}_{i+1}^T W_{i+1}$.

Note that we just consider a linear mapping $f(\mathbf{x}) = W^T\mathbf{x}$. If the mapping f is nonlinear, then we need to consider the following calculus,

$$\Delta\mathbf{h}_{i+1} = \Delta f(\mathbf{h}_i) = \frac{\partial f(\mathbf{h}_i)}{\partial \mathbf{h}_i}\Delta\mathbf{h}_i \tag{7.64}$$

If f is the linear mapping, we can get the same matrix multiplication in Eq. 7.63. Otherwise, we can consider the nonlinear scenario. i.e. If f is the sigmoid function, then

$$\Delta\mathbf{h}_{i+1} = \Delta f(\mathbf{h}_i) = f(\mathbf{h}_i)(1 - f(\mathbf{h}_i))W_{i+1}\Delta\mathbf{h}_i \tag{7.65}$$

Note we use the first order (linear) to approximate the nonlinear case, we think $f(\mathbf{h}_i)(1 - f(\mathbf{h}_i))W_{i+1}$ as the multiplication matrix in the linear case, thus we can get the following formula in the backpropagation step

$$\Delta\mathbf{h}_i = \Delta\mathbf{h}_{i+1} W_{i+1}^T f(\mathbf{h}_i)(1 - f(\mathbf{h}_i)) \tag{7.66}$$

The basic idea is to linearize $f(\mathbf{h}_i)$. More specifically, we can approximate $f(\mathbf{h}_i)$ with the linear function $f(\mathbf{h}_i)(1 - f(\mathbf{h}_i))W_{i+1}$, and then use it to do backpropagation. After we get the error $\Delta\mathbf{h}_i$, we can use the same linear function to backpropagate the error to W_i.

Similarly, consider the change ΔW_i w.r.t. W_i, then we have the forward error

$$\begin{aligned} \Delta\mathbf{h}_{i+1} = \Delta f(\mathbf{h}_i) &= \frac{\partial f(\mathbf{h}_i)}{\partial W_i}\Delta W_i \\ \implies \Delta\mathbf{h}_{i+1} &= f(\mathbf{h}_i)(1 - f(\mathbf{h}_i))\mathbf{h}_i \Delta W_i \end{aligned} \tag{7.67}$$

If we think the matrix $f(\mathbf{h}_i)(1 - f(\mathbf{h}_i))\mathbf{h}_i$ is the linear approximation in Eq. 7.67, then we can get the derivative w.r.t. W_i

$$dW_i = \mathbf{h}_i^T \Delta \mathbf{h}_{i+1} f(\mathbf{h}_i)(1 - f(\mathbf{h}_i)) \tag{7.68}$$

where we use dW_i as the gradient w.r.t. W_i. Analogically, given the error $\Delta \mathbf{h}_i$ in the i-th layer, we can backpropagate it into the $i-1$ layer in the same way as mentioned above. After we calculate the gradients w.r.t. all parameters, we can use gradient based methods, such as stochastic gradient descent or L-BFGS to update model parameters.

7.5 Convolutional neural networks

The convolutional neural networks (CNN) [62] is proposed by Yann Lecun et al., which was initially used for handwritten recognition. The basic idea is that we have multiple fitters to do convolution over the input image and the output can be used as the input to another layer of convolution and repeat this process to construct the hierarchical deep structure. The intuitive understanding is that the first convolutional operation over the input image will output edges information, which will be combined into more complex objects in the following layer. More specifically, the intermediate layers can be shared and reused in the following layer to handle multi-class recognition.

In this part, we will focus on how to compute the gradients via backpropagation and discuss corresponding mathematical operations in details. Suppose we have objective function $\mathcal{L}(\mathcal{D}; \Theta)$, which could be negative loglikelihood, cross error entropy and so on. And the purpose is to learn model parameters $\Theta = \{\mathbf{W}_{i,j}^l\}, l = [1, 2, .., L]$ given the training data $\mathcal{D}$. As we mentioned before, DNN uses simple matrix multiplication to forward information to the next layer. If the input is from high dimensional data (e.g. images), then it is not scalable to use the full connected DNN to learn model parameters. Unlike DNN, CNN leverages convolution over input to forward signals, which can significantly reduce the size of model parameters and is very effective when the input data are images. Notice that the main difference between CNN and DNN is that CNN replaces multiplication with convolution. Thus, we will specif-

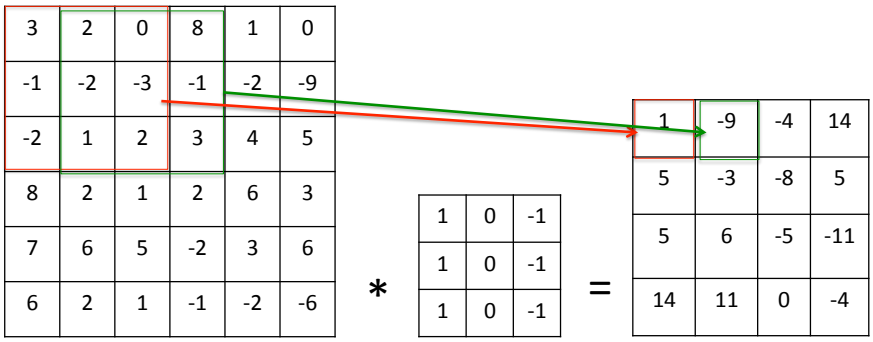


Figure 7.5. It is a convolution example between an image (left figure with size 6×6) and a filter (3×3). Basically, we can use the filter to scan the whole image from left to right and top to down to compute the convolutional values.

ically discuss forward and backward operations via convolution in the following.

Forward: at a convolution layer, the feature maps from previous layers are convolved with learnable weights (or kernels) and put through the activation function to form the output feature map, shown in Fig. 7.5. Each output map may combine convolutions with multiple input maps. Suppose we have the feature maps $\mathbf{h}_i^l$ at the l-th layer, $i = 1, 2, ...$, and we want to aggregate all feature maps and forward them into the next layer with filter $\mathbf{W}_{j,i}^l$. In general, we have that

$$\mathbf{h}_j^{l+1} = \sum_i \mathbf{W}_{j,i}^l * \mathbf{h}_i^l \tag{7.69}$$

where $*$ indicates convolutional operations. For the 2-D case, we have

$$\mathbf{h}_j^{l+1}(p,q) = \sum_{i,u,v} \mathbf{W}_{j,i}^l(u,v)\mathbf{h}_i^l(p-u, q-v) \tag{7.70}$$

$$= \sum_{i,u,v} \mathbf{W}_{j,i}^l(p-u, q-v)\mathbf{h}_i^l(u,v) \tag{7.71}$$

Backward: suppose we have get the gradient w.r.t. $\mathbf{h}_j^{l+1}$ in $(l+1)$-th layer, and we want to backpropagate it to l-th layer

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{j,i}^l(u,v)} = \sum_{p,q} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_j^{l+1}(p,q)} \frac{\partial \mathbf{h}_j^{l+1}(p,q)}{\partial \mathbf{W}_{j,i}^l(u,v)} \tag{7.72}$$

$$= \sum_{p,q} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_j^{l+1}(p,q)} \mathbf{h}_i^l(p-u, q-v) \tag{7.73}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_j^{l+1}(p,q)} * \mathbf{h}_i^l(-u, -v) \tag{7.74}$$

Therefore, by using spatial inversion of filters i.e., flip the filter about each axis, we can use convolution again to compute gradients for each layer via backpropagation. After we calculate gradients, we can use the same algorithms from DNN to learn model parameters (or filters).

7.6 Deep learning for dimension reduction

Previously, we have introduced deep learning to learn representations, which can be used for classification. In this part, we will discuss Autoencoder [44] for dimension reduction and also its application on image denoising problems.

7.6.1 Deep autoencoder

We first introduce the autoencoder, and then we generalize it into deep autoencoder. The traditional Autoencoder (AE) [44] relies on the deterministic mapping function f_θ that transforms an input vector $\mathbf{v} \in \mathbb{R}^d$ into hidden representation $\mathbf{h}$ with n neurons, whose typical form is an affine transformation followed by a linear or nonlinear mapping

$$f_\theta(\mathbf{v}) = s(\mathbf{W}\mathbf{v} + \mathbf{b}) \tag{7.75}$$

where the parameter $\theta = \{\mathbf{W}, \mathbf{b}\}$, with $\mathbf{W} \in \mathbb{R}^{n \times d}$ and $\mathbf{b} \in \mathbb{R}^n$. The resulting hidden representation $\mathbf{h}$ is then mapped back to reconstruct the input in the original space $\hat{\mathbf{v}} = g_{\theta'}(\mathbf{h})$. More specifically, the mapping function is

$$g_{\theta'}(\mathbf{h}) = s(\mathbf{W}'\mathbf{h} + \mathbf{b}') \tag{7.76}$$

with $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. The function s in Eqs. (7.75) and (7.76) can be linear or nonlinear mapping, such as logistic function. If it is a tied weight autoencoder, then $\theta' = \theta^T$. AE can be trained by minimizing the reconstruction error or cross entropy with var-

ious optimization methods [44, 46]. For example, given the original data $\mathbf{v}_i$, it can minimize the following reconstruction error:

$$\theta, \theta' = \text{argmin}_{\theta,\theta'} \sum_{i=1}^{N} (\mathbf{v}_i - \hat{\mathbf{v}}_i)^2 = \text{argmin}_{\theta,\theta'} \sum_{i=1}^{N} (\mathbf{v}_i - g \circ f(\mathbf{v}_i))^2 \tag{7.77}$$

where we ignore the underscript parameters in the mapping functions f and g for convenience, and $\circ$ indicates function composition.

To generalize Eq. 7.77 into deep Autoencoder, we can add more layers to encode and decode signals. Given the input $\mathbf{v}$, we can add the encoder and the decoder with L layers respectively, and minimize the cross entropy below

$$\{\theta_i\}_{i=1}^{L}, \{\theta_i'\}_{i=1}^{L} =$$

$$\text{argmin}_{\theta,\theta'} - \sum_{i=1}^{N} \mathbf{v}_i \text{log} \hat{\mathbf{v}}_i + (1 - \mathbf{v}_i) \text{log}(1 - \hat{\mathbf{v}}_i) \tag{7.78}$$

$$\text{with } \hat{\mathbf{v}}_i = \underbrace{g_1 \circ g_2 \circ \cdots \circ g_L}_{L \text{ times}} \circ \underbrace{f_L \circ f_{L-1} \circ \cdots \circ f_1}_{L \text{ times}} (\mathbf{v}_i) \tag{7.79}$$

where $\circ$ indicates function composition, f_i to encode signal at the i-th layer and g_i to decode signal at the i-th layer. The objective function in Eq. 7.78 hopes the decoded signal $\hat{\mathbf{v}}$ matches the input $\mathbf{v}$ as much as possible.

To learn model parameters, we take the same strategy mentioned before, i.e. back-propagation to compute gradients and learn the model with gradient based methods.

7.6.2 Deep denosing autoencoder

A denoising autoencoder (DAE) generalizes the AE with the clean and corrupted pairs $\langle \mathbf{v}, \tilde{\mathbf{v}} \rangle$, and it can be trained to reconstruct a clean "repaired" input from a corrupted version of it. The key difference between AE and DAE is that f is now a deterministic function of $\tilde{\mathbf{v}}$ rather than $\mathbf{v}$. Note that the purpose of the denoising auto-encoder is to learn more robust features to handle the random noise setting.

The deep denoising autoencoder (DDAE) here is similar to the deep autoencoder, both of which are consisted of an encoder and a symmetric decoder [44]. The difference between our model and the deep autoencoder is that our model needs noisy and

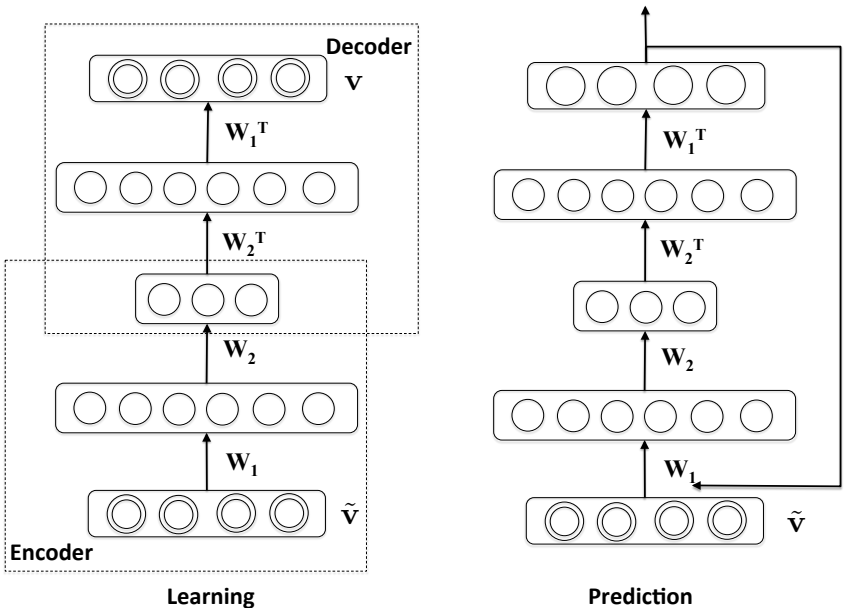


Figure 7.6. An example of a 2-layer deep denoising autoencoder with tied weights. The left graph depicts the encoder and decoder, which can be learned with pretraining (RBM) and fine-tuning (backpropagation) process. The right graph shows the reconstruction strategy, an option to add the feedback loop into the deep denoising autoencoder.

clean data pairs to learn a neural network for denoising problems, while the deep autoencoder requires the input and the output are the same for representation learning. More specifically, given the original data $\mathbf{v}_i$ and its corrupted $\tilde{\mathbf{v}}_i$ for $i = \{1, 2, ..., N\}$, for a DDAE model with L layers, we use the following cross entropy loss:

$$\{\theta_i\}_{i=1}^{L}, \{\theta_i'\}_{i=1}^{L} =$$

$$\text{argmin}_{\theta,\theta'} - \sum_{i=1}^{N} \mathbf{v}_i \log \hat{\mathbf{v}}_i + (1 - \mathbf{v}_i) \log(1 - \hat{\mathbf{v}}_i) \tag{7.80}$$

$$\text{with } \hat{\mathbf{v}}_i = \underbrace{g_1 \circ g_2 \circ \cdots \circ g_L}_{L \text{ times}} \circ \underbrace{f_L \circ f_{L-1} \circ \cdots \circ f_1}_{L \text{ times}}(\tilde{\mathbf{v}}_i) \tag{7.81}$$

where we ignore the underscript for parameters in functions f_i and g_i, for $i = \{1, ..., L\}$. In our model, we use the logistic (or sigmoid) function in each layer, and we can learn the parameters by minimizing Eq. (7.80). An example of DDAE with $L = 2$ is shown in Fig. (7.6).

7.6.2.1 Parameter Learning

We first use pretraining to initialize the weights of deep denoising autoencoder. Basically, it is to learn a stack of RBMs greedily layer-by-layer. To put it simply, the learned features of the current layer RBM are treated as the "data" to train the next RBM in the stack. Assume that we have a training set $\mathcal{D} = \langle \tilde{\mathbf{v}}_i, \mathbf{v}_i \rangle_{i=1}^{N}$, where $\mathbf{v}_i \in \mathbb{R}^d$ is the clean data, and $\tilde{\mathbf{v}}_i$ is a noisy version of it. For each layer, we can use RBM for pretraining to learn the encoder. An RBM with n hidden units is a parametric model of the joint distribution between a layer of hidden variables $\mathbf{h} = (h_1, ..., h_n)$ and the noisy observations $\tilde{\mathbf{v}} = (\tilde{v}_1, ..., \tilde{v}_d)$. The RBM joint likelihood takes the form:

$$p(\tilde{\mathbf{v}}, \mathbf{h}) \propto e^{-E(\tilde{\mathbf{v}}, \mathbf{h})} \tag{7.82}$$

where the energy function is

$$E(\tilde{\mathbf{v}}, \mathbf{h}) = -\mathbf{h}^T \mathbf{W}_1 \tilde{\mathbf{v}} - \mathbf{b}^T \tilde{\mathbf{v}} - \mathbf{c}^T \mathbf{h} \tag{7.83}$$

And we can compute the following conditional likelihood:

$$p(\tilde{\mathbf{v}}|\mathbf{h}) = \prod_i p(\tilde{v}_i|\mathbf{h}) \tag{7.84a}$$

$$p(v_i = 1|\mathbf{h}) = \text{logistic}(b_i + \sum_j W_1(i,j) h_j) \tag{7.84b}$$

$$p(h_i = 1|\tilde{\mathbf{v}}) = \text{logistic}(c_i + \sum_j W_1(j,i) \tilde{v}_j) \tag{7.84c}$$

where $\text{logistic}(x) = 1/(1 + e^{-x})$. To learn RBM parameters, we need to minimize the negative log likelihood $-\log p(\tilde{\mathbf{v}})$ on training data $\mathcal{D}$, the parameters updating can be calculated with an efficient stochastic descent method, namely contrastive divergence (CD) [41].

The weights in the deep denoising autoencoder can be initialized layer by layer greedily. For the L-layer DDAE, the weight for each layer is indicated as $\mathbf{W}_i$ for $i = \{1, .., \text{L}\}$. Specifically, we think RBM is a 1-layer encoder of DDAE, with weight $\mathbf{W}_1$. Thus, DDAE can learn parametric nonlinear mapping from input $\tilde{\mathbf{v}}$ to output $\mathbf{h}$, $f : \tilde{\mathbf{v}} \rightarrow \mathbf{h}$. For example, for 1-layer DDAE, we have $\mathbf{h} = \text{logistic}(\mathbf{W_1}^T \tilde{\mathbf{v}} + \mathbf{c})$, where the weight $\mathbf{W_1}$ in the encoder is pretrained with RBM and can be mirrored to initialize

the weight of the decoder.

After the pretraining for the encoder, the RBMs are "unrolled" to create a deep autoencoder by initially mirroring the weights between the encoder and the decoder, refer to an example DDAE with 2-layers in Fig. 7.6. Then, we minimize the reconstruction error in Eq. (7.80) with the global fine-tuning stage, which uses backpropagation through the whole deep denoising auto-encoder to fine-tune the weights for optimal reconstruction. The difference between our DDAE and the deep autoencoder [44] is that DDAE learns a mapping from a noisy observation to the clean of it, while the deep autoencoder tries to find a projection to reconstruct the original data.

7.6.2.2 Prediction

In this section, we will argue that the feedback loop added in the DDAE in Fig. 7.6 will enhance the "reconstructed" regions in the input.

After training the DDAE model with the global fine-tuning, we can use it to reconstruct the corresponding clean image given any noisy observation. For any input $\tilde{\mathbf{v}}$, we infer the output $\hat{\mathbf{v}}$ with feed-forward algorithm according to Eq. (7.81). In order to effectively enhance the "repaired" image well in the reconstruction process, we propose to add the feedback loop in the reconstruction stage.

To simplify the problem, we assume the 1-layer DDAE here, which can be easily generalized into multi-layers DDAE model. And in general, DDAE employs the logistic function for nonlinear mapping. Given any input image $\tilde{\mathbf{v}} \in \mathbb{R}^d$, the reconstructed output is $\hat{\mathbf{v}} = f_\theta \circ g_{\theta'}(\tilde{\mathbf{v}})$. As we known, if $f(v)$ and $g(v)$ are monotonically increasing functions, then so $f(v) + g(v)$ and $f \circ g(v)$ are. Note that the logistic function is monotonically increasing function, and thus its composition functions are also monotonically increasing. The deep denoising autoencoder here is the compositions of monotonically increasing functions, thus the feedback loop in the inference stage will enhance the positive feedback and ignore the negative feedback. To simplify the problem, we use an example to explain our idea. Without loss of generality, for any given pixel $\tilde{\mathbf{v}}_i$, we consider two main cases for an ideal denoising autoencoder here: (1) if the given pixel $\tilde{\mathbf{v}}_i$ is corrupted, and the deep denoising autoencoder can remove the noise of it; (2) suppose the pixel is from the original clean image, and the deep denoising autoencoder keep it in the output. For the first case, $\tilde{\mathbf{v}}_i$ =1 and its reconstruction $\hat{\mathbf{v}}_i = 0$, we have $f_\theta \circ g_{\theta'}(\tilde{\mathbf{v}}_i + \hat{\mathbf{v}}_i) = f_\theta \circ g_{\theta'}(\tilde{\mathbf{v}}_i) = 0$. Thus, the denoising

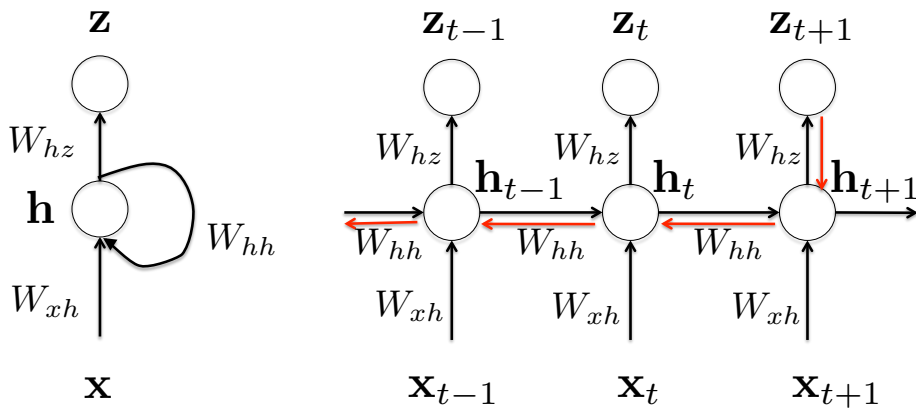


Figure 7.7. It is a RNN example: the left recursive description for RNNs, and the right is the corresponding extended RNN model in a time sequential manner.

network with loop can still predict correctly. For the second case, $\tilde{\mathbf{v}}_i$ =1 and $\hat{\mathbf{v}}_i = 1$, we have $f_\theta \circ g_{\theta'}(\tilde{\mathbf{v}}_i + \hat{\mathbf{v}}_i) > f_\theta \circ g_{\theta'}(\tilde{\mathbf{v}}_i) = \hat{\mathbf{v}}_i$. In other words, the feedback loop enhance the clean regions of the input in this situation. Hence, the feedback loop in the denoising autoencoder is helpful on the denosing task.

There may be other situations, but they can be ignored if the denoising autoencoder did well on the reconstruction process. As we known, the influence of the output to the feedback (recurrent) network is decreasing in the loop. Thus, even if the inference is wrong, the loop network can reduce its effect in the denoising network. We will show the advantage of the enhanced denoising autoencoder in the experiments.

7.7 Recurrent neural networks

Recall that we have talked about HMM, which is a generative model and the output $\mathbf{x}_t$ depends on its corresponding hidden state $\mathbf{h}_t$ in Chapter 3. Although it is effective model for sequential data, it cannot capture the intrinsic information from the data (i.e. it does not exploit the multiple layers of nonlinear mapping), and also it is expensive to handle high order correlation. In this chapter, we will discuss recurrent neural networks (RNNs), shown in Fig. 7.7. A RNN is a kind of neural network, which can send feedback signals (to form a directed cycle), such as Hopfield net [63] and long-short term memory (LSTM) [64].

RNN models a dynamic system, where the hidden state $\mathbf{h}_t$ is not only dependent

on the current observation $\mathbf{x}_t$, but also relies on the previous hidden state $\mathbf{h}_{t-1}$. More specifically, we can represent $\mathbf{h}_t$ as

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{7.85}$$

where f is a nonlinear mapping. Thus, $\mathbf{h}_t$ contains information about the whole sequence, which can be inferred from the recursive definition in Eq. 7.85. In other words, RNN can use the hidden variables as a memory to capture long term information from a sequence.

Suppose that we have the following RNN model, such that

$$\mathbf{h}_t = tanh(W_{\mathbf{hh}}\mathbf{h}_{t-1} + W_{\mathbf{xh}}\mathbf{x}_t + \mathbf{b}_{\mathbf{h}}) \tag{7.86}$$

$$z_t = softmax(W_{hz}\mathbf{h}_t + \mathbf{b}_z) \tag{7.87}$$

where $tanh(x)$ is defined as

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Take the derivative of $tanh(x)$ w.r.t. x

$$\begin{aligned}
&\frac{\partial tanh(x)}{\partial(x)} = \frac{\partial \frac{sinh(x)}{cosh(x)}}{\partial x} \\
=&\frac{\frac{\partial sinh(x)}{\partial x} cosh(x) - sinh(x)\frac{\partial cosh(x)}{\partial x}}{\big(cosh(x)\big)^2} \\
=&\frac{[cosh(x)]^2 - [sinh(x)]^2}{\big(cosh(x)\big)^2} \\
=&1 - [tanh(x)]^2
\end{aligned} \tag{7.88}$$

More specifically, the RNN model above has one hidden layer as depicted in Fig. 7.7. Notice that it is very easy to extend the one hidden case into multiple layers, which has been discussed in deep neural network before. Considering the varying length for each sequential data, we also assume the parameters in each time step are the same across the whole sequential analysis. Otherwise it will be hard to compute

the gradients. In addition, sharing the weights for any sequential length can generalize the model well. As for sequential labeling, we can use the maximum likelihood to estimate model parameters. In other words, we can minimize the negative log likelihood the objective function

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\sum_t y_t log z_t \tag{7.89}$$

In the following, we will use notation $\mathcal{L}$ as the objective function for simplicity. And further we will use $\mathcal{L}(t+1)$ to indicate the output at the time step $t+1$, s.t. $\mathcal{L}(t+1) = -y_{t+1} log z_{t+1}$. Let's first take the derivative with respect to z_t

$$\frac{\partial \mathcal{L}}{\partial z_t} = -(y_t - z_t) \tag{7.90}$$

Note the weight W_{hz} is shared across all time sequence, thus we can differentiate to it at each time step and sum all together

$$\frac{\partial \mathcal{L}}{\partial W_{hz}} = \sum_t \frac{\partial \mathcal{L}}{\partial z_t} \frac{\partial z_t}{\partial W_{hz}} \tag{7.91}$$

Similarly, we can get the gradient w.r.t. bias b_z

$$\frac{\partial \mathcal{L}}{\partial b_z} = \sum_t \frac{\partial \mathcal{L}}{\partial z_t} \frac{\partial z_t}{\partial b_z} \tag{7.92}$$

Now let's go through the details to derive the gradient w.r.t. W_{hh}. Considering at the time step $t \rightarrow t+1$ in Fig. 7.7,

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{hh}} = \frac{\partial \mathcal{L}(t+1)}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial W_{hh}} \tag{7.93}$$

where we only consider one step $t \rightarrow (t+1)$. And because the hidden state $\mathbf{h}_{t+1}$ partially dependents on $\mathbf{h}_t$, so we can use backpropagation to compute the above partial derivative. Think further W_{hh} is shared cross the whole time sequence, according to the recursive definition in Eq. 7.86. Thus, at the time step $(t-1) \rightarrow t$, we can further

get the partial derivative w.r.t. W_{hh} as follows

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{hh}} = \frac{\partial \mathcal{L}(t+1)}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial W_{hh}} \tag{7.94}$$

Thus, at the time step $t+1$, we can compute gradient w.r.t. z_{t+1} and further use backpropagation through time (BPTT) from t to 0 to calculate gradient w.r.t. W_{hh}, shown as the red chain in Fig. 7.7. Thus, if we only consider the output z_{t+1} at the time step $t+1$, we can yield the following gradient w.r.t. W_{hh}

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial \mathcal{L}(t+1)}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial W_{hh}} \tag{7.95}$$

Aggregate the gradients w.r.t. W_{hh} over the whole time sequence with back propagation, we can finally yield the following gradient w.r.t. W_{hh}

$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \sum_{t} \sum_{k=1}^{t+1} \frac{\partial \mathcal{L}(t+1)}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial W_{hh}} \tag{7.96}$$

Now we turn to derive the gradient w.r.t. W_{xh}. Similarly, we consider the time step $t+1$ (only contribution from $\mathbf{x}_{t+1}$) and calculate the gradient w.r.t. to W_{xh} as follows

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{xh}} = \frac{\partial \mathcal{L}(t+1)}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial W_{xh}}$$

Because $\mathbf{h}_t$ and $\mathbf{x}_{t+1}$ both make contribution to $\mathbf{h}_{t+1}$, we need to backpropagte to $\mathbf{h}_t$ as well. If we consider the contribution from the time step t, we can further get

$$\begin{aligned} \frac{\partial \mathcal{L}(t+1)}{\partial W_{xh}} &= \frac{\partial \mathcal{L}(t+1)}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial W_{xh}} + \frac{\partial \mathcal{L}(t+1)}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial W_{xh}} \\ &= \frac{\partial \mathcal{L}(t+1)}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial W_{xh}} + \frac{\partial \mathcal{L}(t+1)}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial W_{xh}} \end{aligned} \tag{7.97}$$

Thus, summing up all contributions from t to 0 via backpropagation, we can yield

the gradient at the time step $t+1$

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{xh}} = \sum_{k=1}^{t+1} \frac{\partial \mathcal{L}(t+1)}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial W_{xh}} \tag{7.98}$$

Further, we can take derivative w.r.t. W_{xh} over the whole sequence as

$$\frac{\partial \mathcal{L}}{\partial W_{xh}} = \sum_{t} \sum_{k=1}^{t+1} \frac{\partial \mathcal{L}(t+1)}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial W_{xh}} \tag{7.99}$$

However, there are gradient vanishing or exploding problems to RNNs. Notice that $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k}$ in Eq. 7.99 indicates matrix multiplication over the sequence. Because RNNs need to backpropagate gradients over a long sequence (with small values in the matrix multiplication), gradient value will shrink layer over layer, and eventually vanish after a few time steps. Thus, the states that are far away from the current time step does not contribute to the parameters' gradient computing (or parameters that RNNs is learning). Another direction is the gradient exploding, which attributed to large values in matrix multiplication.

Considering the weakness of RNNs, long short term memory (LSTM) was proposed to handle gradient vanishing problem. Notice that RNNs makes use of a simple *tanh* function to incorporate the correlation between $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ and $\mathbf{h}_t$, while LSTM model such correlation with a memory unit. And LSTM has attracted great attention for time series data recently and yielded significant improvement over RNNs. For example, LSTM has demonstrated very promising results on handwriting recognition task [65]. In the following part, we will introduce LSTM, which introduces memory cells for the hidden states.

7.7.1 Long short term memory (LSTM)

The architecture of RNNs have cycles incorporating the activations from previous trim steps as input to the network to make a decision for the current input, which makes RNNs better suited for sequential labeling tasks. However, one vital problem of RNNs is the gradient vanishing problem. LSTM [64] extends the RNNs model with two advantages: (1) introduce the memory information (or cell) (2) handle long sequential better, considering the gradient vanishing problem, refer to Fig. 7.8 for the

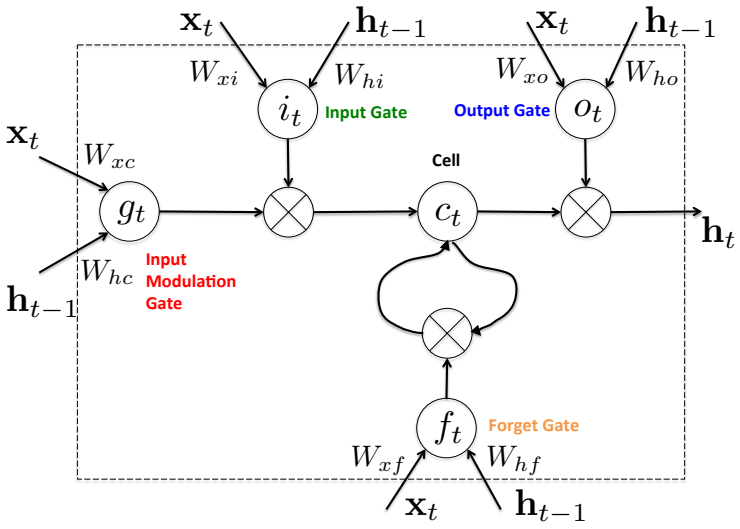


Figure 7.8. It is an unit structure of LSTM, including 4 gates: input modulation gate, input gate, forget gate and output gate.

unit structure. In this part, we will introduce how to forward and backward LSTM neural network, and we will derive gradients and backpropagate error in details.

The core of LSTM is a memory unit (or cell) $\mathbf{c}_t$ in Fig. 7.8, which encodes the information of the inputs that have been observed up to that step. The memory cell $\mathbf{c}_t$ has the same inputs ($\mathbf{h}_{t-1}$ and $\mathbf{x}_t$) and outputs $\mathbf{h}_t$ as a normal recurrent network, but has more gating units which control the information flow. The input gate and output gate respectively control the information input to the memory unit and the information output from the unit. More specifically, the output $\mathbf{h}_t$ of the LSTM cell can be shut off via the output gate.

As to the memory cell itself, it is also controlled with a forget gate, which can reset the memory unit with a sigmoid function. More specifically, given a sequence data $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$ we have the gate definition as follows:

$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + b_f) \tag{7.100}$$

$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + b_i) \tag{7.101}$$

$$\mathbf{o}_t = tanh(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + b_o) \tag{7.102}$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{g}_t \tag{7.103}$$

$$\mathbf{g}_t = \sigma(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + b_c) \tag{7.104}$$

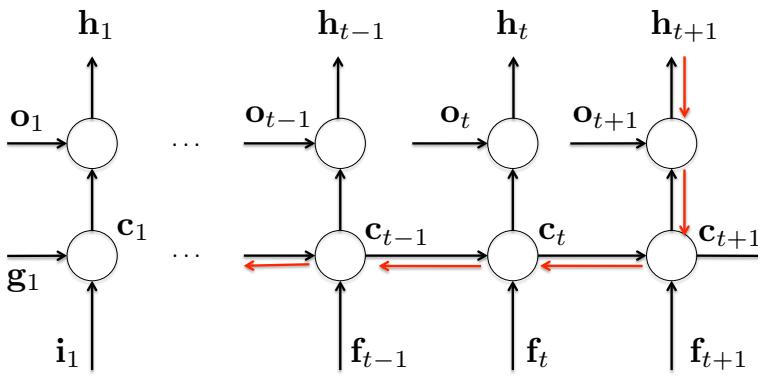


Figure 7.9. This graph unfolds the memory unit of LSTM, with the purpose to make it easy to understand error backpropagation.

$$\mathbf{h}_t = \mathbf{o}_t \circ tanh(\mathbf{c}_t),\ z_t = softmax(W_{hz}\mathbf{h}_t + b_z) \tag{7.105}$$

where $\mathbf{f}_t$ indicates forget gate, $\mathbf{i}_t$ input gate, $\mathbf{o}_t$ output gate and $\mathbf{g}_t$ input modulation gate. Note that the memory unit models much more information than RNNs, except Eq. 7.102.

Same as RNNs to make prediction, we can add a linear model over the hidden state $\mathbf{h}_t$, and output the likelihood with softmax function

$$z_t = softmax(W_{hz}\mathbf{h}_t + b_z)$$

If the groundtruth at time t is y_t, we can consider minimizing least square $\frac{1}{2}(y_t - z_t)^2$ or cross entroy to estimate model parameters. Thus, for the top layer classification with weight W_{hz}, we can take derivative w.r.t. z_t and W_{hz} respectively

$$dz_t = y_t - z_t \tag{7.106}$$

$$dW_{hz} = \sum_t \mathbf{h}_t dz_t \tag{7.107}$$

$$d\mathbf{h}_T = W_{hz} dz_T \tag{7.108}$$

where we only consider the gradient w.r.t. $\mathbf{h}_T$ at the last time step T. For any time step t, its gradient will be a little different, which will be introduced later (refer to Eq. 7.118).

then backpropagate the LSTM at the current time step t

$$d\mathbf{o}_t = tanh(\mathbf{c}_t)d\mathbf{h}_t \tag{7.109}$$
$$d\mathbf{c}_t = \left(1 - tanh(\mathbf{c}_t)^2\right)\mathbf{o}_t d\mathbf{h}_t \tag{7.110}$$
$$d\mathbf{f}_t = \mathbf{c}_{t-1}d\mathbf{c}_t \tag{7.111}$$
$$d\mathbf{c}_{t-1} += \mathbf{f}_t \circ d\mathbf{c}_t \tag{7.112}$$
$$d\mathbf{i}_t = \mathbf{g}_t d\mathbf{c}_t \tag{7.113}$$
$$d\mathbf{g}_t = \mathbf{i}_t d\mathbf{c}_t \tag{7.114}$$

where the gradient w.r.t. $tanh(\mathbf{c}_t)$ in Eq. 7.110 can be derived according to Eq. 7.88.

further, back-propagate activation functions over the whole sequence

$$\begin{aligned} dW_{xo} &= \sum_t (1 - \mathbf{o}_t^2)\mathbf{x}_t d\mathbf{o}_t \\ dW_{xi} &= \sum_t \mathbf{i}_t(1 - \mathbf{i}_t)\mathbf{x}_t d\mathbf{i}_t \\ dW_{xf} &= \sum_t \mathbf{f}_t(1 - \mathbf{f}_t)\mathbf{x}_t d\mathbf{f}_t \\ dW_{xc} &= \sum_t \mathbf{g}_t(1 - \mathbf{g}_t)\mathbf{x}_t d\mathbf{g}_t \end{aligned} \tag{7.115}$$

Note that the weights W_{xo}, W_{xi}, W_{xf} and W_{xc} are shared across the whole sequence, thus we need to take the same summation over t as RNNs in Eq. 7.91. Similarly, we have

$$\begin{aligned} dW_{ho} &= \sum_t (1 - \mathbf{o}_t^2)\mathbf{h}_{t-1} d\mathbf{o}_t \\ dW_{hi} &= \sum_t \mathbf{i}_t(1 - \mathbf{i}_t)\mathbf{h}_{t-1} d\mathbf{i}_t \\ dW_{hf} &= \sum_t \mathbf{f}_t(1 - \mathbf{f}_t)\mathbf{h}_{t-1} d\mathbf{f}_t \\ dW_{hc} &= \sum_t \mathbf{g}_t(1 - \mathbf{g}_t)\mathbf{h}_{t-1} d\mathbf{g}_t \end{aligned} \tag{7.116}$$

and corresponding hiddens at the current time step $t - 1$

$$d\mathbf{h}_{t-1} = (1-\mathbf{o}_t^2)W_{ho}d\mathbf{o}_t + \mathbf{i}_t(1-\mathbf{i}_t)W_{hi}d\mathbf{i}_t + \mathbf{f}_t(1-\mathbf{f}_t)W_{hf}d\mathbf{f}_t + \mathbf{g}_t(1-\mathbf{g}_t)W_{hc}d\mathbf{g}_t \quad (7.117)$$

$$d\mathbf{h}_{t-1} = d\mathbf{h}_{k-1} + W_{hz}dz_{t-1} \quad (7.118)$$

where we consider two sources to derive $d\mathbf{h}_{t-1}$, one is from activation function in Eq. 7.117 and the other is from the objective function at the time step t in Eq. 7.108.

7.7.2 Error backpropagation

In this part, we will give detail information on how to derive gradients, especially Eq. 7.112. As we talked about RNNs before, we can take the same strategy to unfold the memory unit, shown in Fig. 7.9. Suppose we have the least square objective function

$$\mathcal{L}(\mathbf{x},\theta) = \min \sum_t \frac{1}{2}(y_t - z_t)^2 \quad (7.119)$$

where $\theta = \{W_{hz}, W_{xo}, W_{xi}, W_{xf}, W_{xc}, W_{ho}, W_{hi}, W_{hf}, W_{hc}\}$ with biases ignored. To make it easy to understand in the following, we use $\mathcal{L}(t) = \frac{1}{2}(y_t - z_t)^2$.

At the time step T, we take derivative w.r.t. $\mathbf{c}_T$

$$\frac{\partial \mathcal{L}(T)}{\partial \mathbf{c}_T} = \frac{\partial \mathcal{L}(T)}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{c}_T} \quad (7.120)$$

At the time step $T-1$, we take derivative of $\mathcal{L}(T-1)$ w.r.t. $\mathbf{c}_{T-1}$ as

$$\frac{\partial \mathcal{L}(T-1)}{\partial \mathbf{c}_{T-1}} = \frac{\partial \mathcal{L}(T-1)}{\partial \mathbf{h}_{T-1}}\frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{c}_{T-1}} \quad (7.121)$$

However, according to Fig. 7.9, the error is not only backpropagated via $\mathcal{L}(T-1)$, but also from $\mathbf{c}_T$, thus the final gradient w.r.t. $\mathbf{c}_{T-1}$

$$\begin{aligned}\frac{\partial \mathcal{L}(T-1)}{\partial \mathbf{c}_{T-1}} &= \frac{\partial \mathcal{L}(T-1)}{\partial \mathbf{c}_{T-1}} + \frac{\partial \mathcal{L}(T)}{\partial \mathbf{c}_{T-1}} \\ \frac{\partial \mathcal{L}(T-1)}{\partial \mathbf{c}_{T-1}} &= \frac{\partial \mathcal{L}(T-1)}{\partial \mathbf{h}_{T-1}}\frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{c}_{T-1}} + \frac{\partial \mathcal{L}(T)}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{c}_T}\frac{\partial \mathbf{c}_T}{\partial \mathbf{c}_{T-1}}\end{aligned} \quad (7.122)$$

where we use the chain rule in Eq. 7.122. Further, we can rewrite Eq. 7.122 as

$$d\mathbf{c}_{T-1} = d\mathbf{c}_{T-1} + \mathbf{f}_T \circ d\mathbf{c}_T \tag{7.123}$$

In a similar manner, we can derive Eq. 7.112 at any time step.

7.8 Generalized k-fan deep model

Deep learning architectures, such as autoencoder, convolutional neural network and deep belief network, can only handle one input and one output. However, when we have different types of data available, how to leverage multi-source information for data analysis (e.g. classification and regression) is a challenge. In this paper, we propose a K-fan deep model, which can handle the multi-input and multi-output learning problems effectively. In particular, the deep structure has K-branch for different inputs where each branch can be composed of a multi-layer deep model, and a shared representation is learned in an discriminative manner to tackle multimodal tasks. Given a deep structure, objective function depends on task at hand. In experiments, we handle matrix factorization, joint visual restoration and labeling, and multi-view multi-class object recognition. To estimate the model parameters, we initialize the deep model parameters with contrastive divergence (CD) to maximize the joint distribution, and then we use backpropagation to update the model according to specific objective function.

In other words, our modal is a multi-path feed-forward neural network with a shared representation, which can be optimized in an discriminative manner to handle different tasks.

We test our model MovieLens dataset, MNIST dataset and multi-view and multi-class dataset, and the experimental results demonstrate that the model can effectively leverages multi-source information and predict multiple tasks well over competitive baselines.

7.8.1 Introduction

We are exploring deep learning in a joint framework when we have multiple forms of data available in the information age, such as images, labels, texts and videos. Each modality is characterized by very distinct statistical properties, but it also reflects one or two facets of the data even though they come from different input channels. Thus, it is possible to leverage different inputs to learn a shared representation in the prediction tasks, such as data restoration and classification. Recent advances in deep learning [44] and multi-modality learning [66] shed light on joint representation learning which captures the real-world concept that the data corresponds to. The deep learning methods [41, 47], such as deep belief networks (DBNs) [41], convolutional neural network (CNN) [67, 68] and long short term memory (LSTM) [64], can learn an abstract and expressive representations, which can capture a huge number of possible input configurations. Hence, the representation learned is useful for classification and information retrieval. The multimodal learning model [61] extends the deep learning framework, such as deep autoencoder or deep Boltzmann machines (DBMs), to handle different modalities. However, these previous deep learning models [41, 61, 66, 68] are kind of 2-fan deep model (a special case of our model), and can only handle or predict one task. Often, the joint representation learned [61] is not robust enough when the data is typically very noisy and there may be missing. Furthermore, how to leverage multi-source information from raw data is also an interesting topic for classification and information retrieval.

In this part, we will introduce a framework, the K-fan deep model, where we generalize the previous 2-fan deep learning structures [41, 64, 66] to handle multiple inputs and outputs. Our model is composed of K-branch deep models for different inputs respectively, and a shared representation is learnt to tackle multimodal tasks in an discriminative manner. Our model is powerful because each branch can be a multi-layer deep model, for example, we can use DBN, CNN or LSTM in each branch to handle different modalities, such as images, texts, labels and videos. Refer to the right panel in Fig. 7.10 with DBNs used in each branch for a 3-fan deep structure case. Most similar to our work are the bi-modal deep models [61, 66] to handle image-text data or speech-vision data. The multimodal DBM proposed by Ngiam et al. used a deep autoencoder for speech and vision fusion, while the other method [61] lever-

aged DBMs to learn hidden representations for bi-modal image-text data. There are, however, several crucial differences between our model and other methods. First, our model is a discriminative model, which can handle the K-fan different modalities, instead of bi-modal inputs. Moreover, each branch can be a deep model, such as DBN, CNN and LSTM to handle different forms of data. Thus, our model is more powerful than 2-fan modality models. Secondly, our deep structure can jointly learn from multiple inputs to predict multi-output with shared representations. Lastly, given the deep K-fan structure, it is very flexible to design an objective function and learn the model parameters in an discriminative manner by fusing multiple inputs.

7.8.2 Joint multi-modal deep model

Our K-fan deep model is a deep neural network with K different kinds of inputs coupled stochastic binary hidden units in a hierarchical structure. The inputs can be binary or real values, and they share a hidden layer via multi-layers non-linear transform of RBMs for each input. For clarity, we will use 3-way deep structure to explain our model, shown in Fig. 7.10.

7.8.3 Learning and Inference

In learning stage, all inputs are available. Thus, we can use CD to learn the shared presentation and model parameter effectively. In practice, we divide the parameter estimation into two stages: parameter initialization and fine-tuning. And the parameter initialization stage focuses on learning the joint representations shared by different modalities, while the fine-tuning stage emphasizes on the discriminative learning according to the properties of tasks in our hand. More specifically, because of the joint multimodal deep structure, we first initialize the model parameters by maximizing the joint likelihood in Eq. 7.28. Then we update our model parameters by optimizing different objective functions according to different tasks. Note that for different functions, we have the same parameter initialization step via CD because we use the same multimodal deep structure.

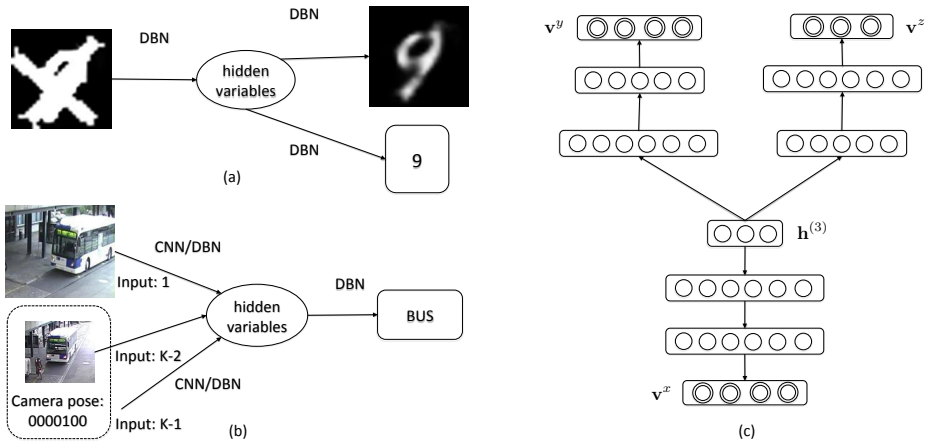


Figure 7.10. The multi-view k-fan deep model for image classification. For each image, we have a deep model, such as CNN or DBN. At the same time, we have another deep branch DBN to model camera pose. The final prediction is the joint feed forward from both image and camera view.

7.8.4 K-fan deep model examples

Matrix completion is to factorize a matrix into a product of (in general) two matrices [69, 70]. Given N users and M items, their sparse preference matrix $R \in \mathbb{R}^{N\times M}$ can be approximated by the product of hidden factors: matrix $V \in \mathbb{R}^{M\times d}$ for items and $U \in \mathbb{R}^{N\times d}$ for users, where d is the length of hidden dimension. The basic idea behind such model is that the user's preferences are determined by a small of number of latent variables. More specifically, the rating R_{ij} of user i and item j can be approximated by the inner product of the latent user-specific and item-specific vectors U_i and V_j. The joint distribution over ratings, users and movies is

$$
\begin{aligned}
&P(R,U,V|\sigma^2) = P(R|U,V,\sigma^2)P(U,V)\\
=&P(R|U,V)P(U)P(V) &(7.124)\\
=&\mathcal{N}(R|UV^T,\sigma^2)\mathcal{N}(U|{\sigma_U}^2)\mathcal{N}(V|{\sigma_V}^2) &(7.125)
\end{aligned}
$$

where the three likelihoods above can be modeled with Gaussian distributions with variance σ, σ_U and σ_V respectively. Furthermore, the objective function can be equiv-

alently written with quadratic regularization terms:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M}(R_{ij} - U_i V_j^T)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N}||U_i||^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M}||V_j||^2 \tag{7.126}$$

We can find a local minimum of the objective function above by performing gradient descent in U and V. The key is how to learn a better hidden factors to approximate the target matrix R.

Our approach here for matrix factorization is to use deep learning to learn features, which can map user i and item j into a latent space with multiple hidden layers. However, this approach may be trapped into overfitting. Moreover, no correlation between users or items is considered in this case. Inspired by memory networks [71, 72], we consider the memory information (similarity between users or items) into our matrix factorization model. Thus, we define the correlation matrix $S \in \mathbb{R}^{M\times T}$, which describes the top T similar coefficients for each item. More specifically, give a item j, we initialize S with the following cosine similarity

$$S_{jt} = \frac{\sum_{n=1}^{N} R_{nj}R_{nt}}{\sqrt{\sum_{n=1}^{N} R_{ni}^2}\sqrt{\sum_{n=1}^{N} R_{nt}^2}} \tag{7.127}$$

In practice, if $T = M$, it means $S \in \mathbb{R}^{M\times M}$ is the whole correlation matrix on all items, which will take a lot of memory to store. If we do not consider correlation between items $(T = 0)$, it is reduced to the probabilistic matrix factorization model (PMF) [69]. Thus, given the item j, we rank its correlation with all the rest items, and select the top T correlated items (indices) to fulfill the matrix G_j (which stores the indices). And we use S_j to store the correlated coefficients related to the items in G_j. Finally, we define our memory-based deep model with one hidden layer as

$$\begin{aligned}\min_{U,V,H} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M}(R_{ij} - \sum_{t\in G_j} E_{it}S_{jt})^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N}||U_i||^2 \\ + \frac{\lambda_V}{2}\sum_{j=1}^{M}||V_j||^2 + \frac{\lambda_S}{2}\sum_{j=1}^{M}||S_j||^2\end{aligned} \tag{7.128}$$

where $E_{it} = U_i V_t^T, s.t.\ t \in G_j$. Fig. 7.11 is the example with one hidden layer for

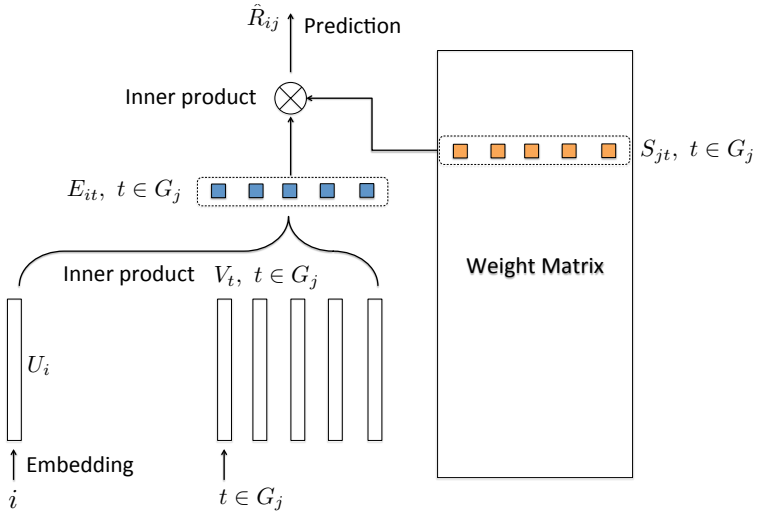


Figure 7.11. The k-fan model for matrix completion, shown with the top 5 nearest neighbors. The whole model has two inputs (the user i and the item t) and one output ($\hat{R}_{ij}$). Note that we can easily extend it into multiple layers to model complex representations for both users and items.

matrix completion. Note that U_i is the hidden representation for user i, and V_t is the hidden representation for item t (which is in the top T nearest neighbors of the movie j). For each user i, we can use the 1 of N encoding to get its binary code, and then use the weight U_i to get its mapping code (which is U_i too). Analogously, we can use multiple layers to get the representation for each user. Thus, it is easy to extend our model into multiple layers to handle possible non-linear regression problems.

To learn the model weights U, V and S, we basically initialize $U$and V with Gaussian distribution, and S via Eq. 7.127. Then, we compute the gradients w.r.t. U, V and S respectively, and update them with gradient descent. The final prediction in the test case is

$$\begin{aligned} & E_{it} = U_i V_t^T, s.t.\ t \in G_j \\ & \Rightarrow \hat{R}_{ij} = \sum_{t \in G_j} E_{it} S_{jt} \end{aligned} \tag{7.129}$$

Joint visual restoration and labeling: For the given corrupted input $\mathbf{v}^x$, we need to restore its clear image $\mathbf{v}^y$ as well as predict its label $\mathbf{v}^z$. Thus, we propose the

following objective function for the binary case

$$\begin{aligned}\theta &= \text{argmin}_\theta \mathcal{J}(\mathbf{v}^x, \mathbf{v}^y, \mathbf{v}^z; \theta) \\ &= \text{argmin}_\theta - \sum_{i=1}^{N} \mathbf{v}_i^y \log \hat{\mathbf{v}}_i^y + (1 - \mathbf{v}_i^y) \log(1 - \hat{\mathbf{v}}_i^y) \\ &- \lambda \sum_{i=1}^{N} \mathbf{v}_i^z \log \hat{\mathbf{v}}_i^z + (1 - \mathbf{v}_i^z) \log(1 - \hat{\mathbf{v}}_i^z) \end{aligned} \tag{7.130}$$

where θ is the set of weights in the 3-way deep architecture respectively (we ignore the subscripts for clarity), and λ is the constant to balance the two losses in Eq. 7.130. And $\hat{\mathbf{v}}_i^y$ and $\hat{\mathbf{v}}_i^z$ are the predictions from the noise input $\mathbf{v}_i^x$, specified as follows

$$\mathbf{h}_i = \underbrace{f_L \circ f_{L-1} \circ \cdots \circ f_1}_{L \text{ times}}(\mathbf{v}_i^x) \tag{7.131}$$

$$\hat{\mathbf{v}}_i^y = \underbrace{g_1 \circ g_2 \circ \cdots \circ g_L}_{L \text{ times}}(\mathbf{h}_i) \tag{7.132}$$

$$\hat{\mathbf{v}}_i^z = \underbrace{\phi_1 \circ \phi_2 \circ \cdots \circ \phi_L}_{L \text{ times}}(\mathbf{h}_i) \tag{7.133}$$

where $\circ$ indicates function composition, $\mathbf{h}_i$ is the shared hidden representation from the triplet $\langle \mathbf{v}^x, \mathbf{v}^y, \mathbf{v}^z \rangle$, and the functions f_l, g_l, and ϕ_l are non-linear projections, such as sigmoid function. We ignore the underscript for parameters in mapping functions f_l, g_l, and ϕ_l, for $l = \{1, ..., L\}$ in the above equations. In our case, we use the same number of layers L to all branches for simplicity and clarity. Note that each branch in the deep structure network can have different number of layers and nodes, only if they keep the same dimetionality of the shared representation.

Multi-view multi-class object recognition: Assume that $\mathbf{v}^x$ is the input image contains different objects, and $\mathbf{v}^y$ is the vector to describe the views to catch the object with cameras, and $\mathbf{v}^z$ indicates which class the object belongs to. The purpose is to answer whether these additional views with the low level image features are helpful to improve the recognition accuracy. Thus, we propose the following objective

$$\theta = \text{argmin}_\theta \mathcal{L}(\mathbf{v}^x, \mathbf{v}^y, \mathbf{v}^z; \theta)$$

$$= \text{argmin}_\theta - \sum_{i=1}^{N} \mathbf{v}_i^z \log \hat{\mathbf{v}}_i^z + (1 - \mathbf{v}_i^z) \log(1 - \hat{\mathbf{v}}_i^z) \tag{7.134}$$

where θ is the set of the weights in the 3-way deep architecture as before. And $\hat{\mathbf{v}}_i^z$ is the prediction from the image $\mathbf{v}_i^x$ and its view $\mathbf{v}_i^y$, specified as follows

$$\mathbf{h}^{(x2)} = \underbrace{f_{L-1} \circ \cdots \circ f_1}_{L-1 \text{ times}}(\mathbf{v}_i^x) \tag{7.135}$$

$$\mathbf{h}^{(y2)} = \underbrace{g_{L-1} \circ \cdots \circ g_1}_{L-1 \text{ times}}(\mathbf{v}_i^y) \tag{7.136}$$

$$\mathbf{h}_i = \text{sigmoid}({\mathbf{h}^{(x2)}}^T \mathbf{W}^{(x3)} + {\mathbf{h}^{(y2)}}^T \mathbf{W}^{(y3)}) \tag{7.137}$$

$$\hat{\mathbf{v}}_i^z = \underbrace{\phi_1 \circ \phi_2 \circ \cdots \circ \phi_L}_{L \text{ times}}(\mathbf{h}_i) \tag{7.138}$$

where we ignore the bias term for $\mathbf{h}_i$ for clarity. The application case is shown in Fig. 7.10.

Given the parameter initialization with CD algorithm, we can minimize the objective function in Eq. 7.130 or 7.134 respectively to estimate the model parameters. To fine-tune the model, we compute the gradients w.r.t. weights via backpropagation in each layer in the objective function, and then we use any gradient-based methods to update the model parameters, such as L-BFGS [39].

Note that the joint visual restoration and labeling task is different from the multi-view multi-class recognition task. In fact, we can see the differences between the two objective functions in Eqs. 7.134 and 7.130, even though they used the same multimodal deep structure and initialized the model parameters with the same CD algorithm. The multi-view multi-class object recognition leverage multiple inputs to improve the classification performance, while the joint visual restoration and labeling learns a joint model for multiple outputs from only one input channel. The former has one input and two outputs, thus it is related to multi-task learning. While the latter has two inputs and one output prediction, by leveraging multi-source information for multi-classification problem.

7.8.5 Relationship to other models

We analyzed the differences between our model and other deep structures, such as deep autoencoder and deep Boltzmann machines.

Deep autoencoder The deep autoendcoder [44] can be thought as a bi-modal deep model with feed-forward networks. It consists of encoder and decoder in order to recover the data itself by learning the shared hidden representation. On the contrary, our model is a K-way deep feed-forward neural network with multiple channels and our model can adjust to different optimization problems given the same architecture, for example the joint visual restoration and labeling. Although these two models can use the same CD algorithm to initialize model parameters, our model is more powerful to handle multiple output predictions, instead of just recovering the data in the deep autoencoder.

Deep Boltzmann machines A multimodal DBM can be viewed as a composition of unimodal undirected pathways. Each path-way can be pretrained separately in a completely unsupervised fashion, which make it possible to leverage a large supply of unlabeled data. The middle graph in Fig. 7.10 is a 3-fan multimodal DBM. In our framework, we define a K-fan deep structure, where each branch can be a deep learning model, such as DBN, LSTM and CNN for different multimodal inputs. Thus, our model is more powerful. Moreover, the multimodal DBM is a generative model, while our model is a discriminative model. In other words, our modal is a multi-path feed-forward neural network with a shared representation, which can be optimized in an discriminative manner to handle different tasks.

Multi-task learning Multi-task learning is an approach to learn a problem together with other related problems at the same time, using a shared representation. Our multiple deep neural network can predict multiple outputs by learning a shared representation for multi-task. Thus, our model can be used to handle multi-task learning problems, such as the joint visual restoration and labeling. In addition, our model can leverage multiple inputs or resources to improve the prediction or classification, such as the multi-view multi-class object recognition in our case. Thus, our deep K-fan structure is flexible and powerful, and can be optimized according to different tasks.

Appendix A

Quadratic Programming

A.1 Introduction

Many machine learning problems needs mathematical background, such as algebra, statistics and optimization. For example, the dual form of SVM requires to solve a second order optimization problem. In this appendix, we will introduce quadratic programming (QP), which includes quadratic nonlinearities into the objective functions. We will introduce a few methods to solve convex quadratic programming, especially active set methods and sequential quadratic programming methods.

A.2 Quadratic Programming

A quadratic programming (QP) is to optimize a quadratic objective function (either minimize or maximize) of a finite number of variables, subject to a finite number of linear inequality and (or) equality constraints. Let us write the quadratic programming in the general form

$$\begin{aligned} \min \quad & f(x) \\ & s.t.\ g_i(x) \leq 0, i \in [1, m] \quad h_j(x) = 0, j \in [1, l] \end{aligned} \tag{A.1}$$

where x is the variable that we want to optimize, $f(x) : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $g_i(x) : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, ..., m$ and $h_j(x) : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, ..., l$ are respectively

the inequality and equality constraints. To make the optimization problem easier, we assume $f(x)$, $g_i(x)$ and $h_j(x)$ are all differentiable.

To find the local minima (or maxima) of $f(x)$ subject to the above constraints, we can incorporate Lagrange multipliers and construct the following Lagrange function

$$\begin{aligned} &\min f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{l} \mu_j h_j(x) \\ &s.t.\ \lambda_i \geq 0,\ \lambda_i g_i(x) = 0, \forall i = 1, ..., m \end{aligned} \tag{A.2}$$

If there is local minima in the objective A.2, then the following necessary conditions hold (by taking the derivative w.r.t. x and setting to zero)

$$\begin{aligned} &\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) + \sum_{j=1}^{l} \mu_j \nabla h_j(x) = 0 \\ &s.t.\ \lambda_i \geq 0, \forall i = 1, ..., m \text{ (dual feasible condition)} \\ &\quad \lambda_i g_i(x) = 0, \forall i = 1, ..., m \text{ (complementary slackness)} \end{aligned} \tag{A.3}$$

The above conditions are called the Kuhn-Karush-Tucker (KKT) conditions. For x^* to be optimal, some of the inequalities may be tight and some not. These that are tight constraints $(g_i(x) = 0)$ always satisfy the complementary slackness, while for these that are not tight can be ignored with $\lambda_i = 0$.

Example: Let us consider the following QP case

$$\begin{aligned} \min \quad & f(x) = \frac{1}{2}\mathbf{x}^T H \mathbf{x} - \mathbf{d}^T \mathbf{x} \\ & s.t.\ \mathbf{b} - A\mathbf{x} \geq 0 \end{aligned} \tag{A.4}$$

We can yield the following conclusion:

(1) If there are only equality constraints in the objective function, then the KKT conditions yield a linear system of equations. There are simple conditions that guarantee a unique solution of this linear system. The obtained critical point is a minimum point if an additional condition is satisfied. More specifically, if the constraint matrix A has a full row rank, and H is a symmetric matrix satisfying $\forall \mathbf{x} \in \mathbb{R}^n$, H is

symmetric matrix satisfying (positive definite)

$$\mathbf{x}^T H\mathbf{x} > 0 \tag{A.5}$$

then there is a unique minimum which is obtained as a solution for a linear system of equations. This case is further considered in Section A.3.

(2) In the more general case where are also inequality constraints, if H is positive semi-definite then the QP is convex. We derive an active set method for this case in Section A.4.

(3) A different method, which works particularly well for the case of nonnegativity constraints, is considered in Section A.5.

A.3 Equality constraints with direct solution

Assume that we have the following QP problem with equality constraints

$$\begin{aligned} \min \quad & f(x) = \frac{1}{2}\mathbf{x}^T H\mathbf{x} - \mathbf{d}^T\mathbf{x} \\ & s.t.\ \mathbf{b} - A\mathbf{x} = 0 \end{aligned} \tag{A.6}$$

where A is a $m \times n$ matrix with a full row rank m, and $H \in \mathbb{R}^{n\times n}$ is a symmetric matrix satisfying A.5. The KKT conditions for the solution $\mathbf{x}^* \in \mathbb{R}^n$ of the QP A.6 give rise to the following linear system

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} d \\ b \end{bmatrix} \tag{A.7}$$

where $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ is the corresponding Lagrange multiplier.

Then, the KKT matrix

$$K = \begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \tag{A.8}$$

is symmetric indefinite. Assume that $A \in \mathbb{R}^{m\times n}$ has full row rank $m \leq n$ and that

the reduced Hessian H is positive definite. Then, the KKT matrix K is nonsingular. Hence, the KKT system A.7 has a unique solution $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$.

Proof: suppose the KKT matrix is singular, then there exists x, z, not both zero, satisfies the following

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = 0 \tag{A.9}$$

Then we have:

$$Hx + A^T z = 0 \tag{A.10}$$

$$Ax = 0 \tag{A.11}$$

According to A.10, we have $x^T Hx + x^T A^T z = 0$. Because $Ax = 0$ via A.11, we have $x^T Hx = 0$. Because H is positive definite, we can further get $Hx = 0$. Plugging $Hx = 0$ into A.10, we have $A^T z = 0$.

Now we considering two assumptions:

(1) If $x \neq 0$ and $z = 0$, because H is positive definite, $x^T Hx > 0$ holds, which violates the assumptions.

(2) If $x = 0$ and $z \neq 0$, because A is full rank matrix $m \leq n$, we have $A^T z \neq 0$. However it also violates $A^T z = 0$ we derived.

Thus, we have $x = 0$ and $z = 0$. And further we conclude the KKT matrix is non-singular.

Theorem: let (x^*, λ^*) be the unique solution of the KKT system, then x^* is the unique global solution of the QP.

Proof: Let $x \in F$ be a feasible point, i.e., $A\mathbf{x} = b$, and $p = \mathbf{x}^* - \mathbf{x}$. Then $Ap = 0$. Substituting $x = x^* - p$ into the objective functional, we get

$$\begin{aligned} f(x) &= \frac{1}{2}(x^* - p)H(x^* - p) - (x^* - p)^T d \\ &= \frac{1}{2}p^T Hp - p^T Hx^* + p^T d + f(x^*) \end{aligned} \tag{A.12}$$

Now, $Hx^* = d - A^T\lambda^*$ according to Eq. A.7. Observing $Ap = 0$, we have

$$p^T Hx^* = p^T(d - A^T\lambda^*) = p^T d - (Ap)^T\lambda^* \tag{A.13}$$

Plugging it back to A.12, we can get $f(x) = \frac{1}{2}p^T Hp + f(x^*) > f(x^*)$. This inequality holds because H is positive definite, thus $\forall p \in \mathbb{R}^n$, $p^T Hp > 0$ always holds.

A.3.1 Symmetric indefinite factorization

A possible way to solve the KKT system A.7 is to provide a symmetric factorization of the KKT matrix according to

$$P^T KP = LDL^T \tag{A.14}$$

where P is an appropriately chosen permutation matrix, L is lower triangular with diag(L) = I, and D is block diagonal. Based on A.14, the KKT system A.7 is solved as follows:

$$K \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = LDL^T \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} d \\ b \end{bmatrix} \tag{A.15}$$

define $y = DL^T$, then we have

$$Ly = \begin{bmatrix} d \\ b \end{bmatrix} \tag{A.16}$$

then we can solve y, after that we define $\hat{y} = L^T$, then we have to solve

$$D\hat{y} = y \tag{A.17}$$

so we have $\hat{y} = D^{-1}y$. Finally, we have

$$L^T \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \hat{y} \tag{A.18}$$

then we get **x**

A.3.2 Range-space approach

The range-space approach applies, if $H \in \mathbb{R}^{n\times n}$ is symmetric positive definite. Block Gauss elimination of the primal variable $\mathbf{x}^*$ leads to the Schur complement system

$$AH^{-1}A^T\lambda^* = AH^{-1}\mathbf{d} - \mathbf{b} \tag{A.19}$$

with the Schur complement $S \in \mathbb{R}^{m\times m}$ given by $S := AH^{-1}A^T$. The range- space approach is particularly effective, if H is well conditioned and easily invertible (e.g., B is diagonal or block- diagonal), H^{-1} is known explicitly (e.g., by means of a quasi-Newton updating formula), the number m of equality constraints is small.

In practice one needs H to be well-conditioned in order for this algorithm to be numerically accurate.

In electrical engineering this is often referred to as node elimination or Kron reduction.

$H\mathbf{x} + A^T\lambda^* = \mathbf{d} \implies \mathbf{x} = H^{-1}(\mathbf{d} - A^T\lambda^*)$. Then the bottom block equations yield

$$(AH^{-1}A^T)\lambda = AH^{-1}\mathbf{d} - \mathbf{b} \tag{A.20}$$

Eliminating λ by Eq. A.20, we obtain

$$\begin{aligned}\mathbf{x} &= H^{-1}(\mathbf{d} - A^T\lambda) \\ &= (I - H^{-1}A^T(AH^{-1}A^T)^{-1}A)H^{-1}\mathbf{d} + H^{-1}A^T(AH^{-1}A^T)^{-1}\mathbf{b}\end{aligned} \tag{A.21}$$

And we can rewrite this as follows

$$x = (I - TA)H^{-1}\mathbf{d} + T\mathbf{b}\ , T = H^{-1}A^T(AH^{-1}A^T)^{-1} \tag{A.22}$$

Notice that T is a projection matrix, satisfying $AT = I$. This is particularly useful when $m \ll n$, because the matrix $AH^{-1}A^T$ is small in dimension.

A.4 Active set methods

For simplicity of notation, let us assume that there are only inequality constraints in A.4 and write them as

$$\mathbf{b} - A\mathbf{x} \geq 0 \tag{A.23}$$

The KKT conditions read

$$H\mathbf{x}^* + A^T\boldsymbol{\lambda}^* = d \tag{A.24a}$$

$$\mathbf{b} - A\mathbf{x}^* \geq 0 \tag{A.24b}$$

$$(A\mathbf{x}^* - \mathbf{b})^T\boldsymbol{\lambda}^* = 0 \tag{A.24c}$$

$$\boldsymbol{\lambda}^* \geq 0 \tag{A.24d}$$

If we know which constraints are active at the critical point, i.e. if we know $\mathcal{A}(\mathbf{x}^*)$, then we can gather the active constraints into a system of equalities, denoted $\hat{A}\mathbf{x}^* = \hat{b}$. For the other rows, $\lambda^* = 0$. Denoting the components of $\boldsymbol{\lambda}^*$ corresponding to the active rows by $\hat{\boldsymbol{\lambda}}^*$, we get from A.24a and A.24b the subsystem with equality constraints

$$H\mathbf{x}^* + A^T\hat{\boldsymbol{\lambda}}^* = d \tag{A.25}$$

$$\hat{A}\mathbf{x}^* = \hat{b} \tag{A.26}$$

This can be solved by the techniques discussed in the previous section, and we are done: at the optimum we should have A.24a and A.24b holding.

But, of course we do not know the final active set $\mathcal{A}(\mathbf{x}^*)$, in general. So, we develop an active set method. This method can be viewed as a generalization of the Simplex method for linear programming. But it is more complex (and less successful in general).

We start with a feasible $\mathbf{x}_0$, which may be obtained for instance from a corresponding LP solver. At the feasible iterate $\mathbf{x}_k$ we maintain a working set $\mathcal{W}_k \in \mathcal{A}(\mathbf{x}_k)$ of constraints which are active at $\mathbf{x}_k$ such that their gradients are linearly independent.

1. At first, we check whether $\mathbf{x}_k$ is optimal in the subspace defined by $\mathcal{W}_k$. Note that, as a function of $\mathbf{p}$, with the first order approximation (high order will approxi-

mate to zero)

$$f(\mathbf{x}_k + \mathbf{p}) = \frac{1}{2}(\mathbf{x}_k + \mathbf{p})^T H(\mathbf{x}_k + \mathbf{p}) - d^T(\mathbf{x}_k + \mathbf{p}) \tag{A.27}$$

$$= \frac{1}{2}\mathbf{p}^T H\mathbf{p} + \mathbf{p}^T(H\mathbf{x}_k - \mathbf{d}) + const \tag{A.28}$$

Thus, we ask if we can take a small step in the direction $\mathbf{p}$ defined by the quadratic program with equalities

$$\min_{\mathbf{p}} \frac{1}{2}\mathbf{p}^T H\mathbf{p} + \mathbf{g}_k^T\mathbf{p}$$
$$s.t.\ A_k^T\mathbf{p} = 0 \tag{A.29}$$

where $\mathbf{g}_k = (H\mathbf{x}_k - \mathbf{d})$ and A_k is composed of those rows of A which are in the working set.

The solution of the subproblem A.29 is of course given by the solution of the KKT system

$$\begin{bmatrix} H & A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \hat{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} d - H\mathbf{x}_k \\ 0 \end{bmatrix} \tag{A.30}$$

(a) If the solution of A.29 is $\mathbf{p} = 0$ then $\mathbf{x}_k$ is optimal in the current working subspace. Proceed to stage 2 below

(b) Otherwise, consider setting

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{p},\ 0 < \alpha \leq 1 \tag{A.31}$$

The step size α must be chosen to maintain feasibility. For any $i \in \mathcal{W}_k$, it will maintain feasibility because of A.30, but for the other rows $i \notin \mathcal{W}_k$ we must ensure $b_i - \mathbf{a}_i^T(\mathbf{x}_k + \alpha\mathbf{p}) \geq 0$. So define (as in the Simplex method for LP)

$$\alpha = \min(1, \min_{\alpha_i^T\mathbf{p}>0} \frac{b_i - \mathbf{a}_i^T\mathbf{x}_k}{\alpha_i^T\mathbf{p}}) \tag{A.32}$$

A constraint l which would yield $\alpha < 1$ in A.32 is a blocking constraint. Add the blocking constraint to the working set $\mathcal{W}_k$ to form $\mathcal{W}_{k+1}$ and update the iterate by

A.31. Otherwise, $\alpha = 1$ in A.32. We update by A.31 but do not change the working set, $\mathcal{W}_{k+1} = \mathcal{W}_k$.

2. Now that an optimal $\mathbf{x}_k$ has been found for $\mathcal{W}_k$, we check optimality: We set $\lambda_i^k = 0$ for $i \notin \mathcal{W}_k$; the other Lagrange multipliers at this point are known from having solved A.30.

(a) If $\hat{\boldsymbol{\lambda}} \geq 0$ then all KKT conditions hold and the optimum point has been found.

(b) Otherwise, there is a component $\lambda_q < 0$. Then we can decrease f($\mathbf{x}$) further by dropping q from the working set. Thus, $\mathcal{W}_{k+1}$ is formed by dropping q from $\mathcal{W}_k$, and the iteration is repeated.

A.5 Sequential programming methods

As in the previous chapter, we have considered methods for solving a general constrained optimization problem

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &s.t.\ \Omega = \{\mathbf{x} \in \mathbb{R}^n | h_i(\mathbf{x}) = 0, i \in \boldsymbol{\xi}, g_j(\mathbf{x}) \geq 0, j \in \mathcal{I}\} \end{aligned} \tag{A.33}$$

where $\boldsymbol{\xi}$ indicates the set of equality constraints and $\mathcal{I}$ represents the inequity set. In this part, we will discuss sequential quadratic programming (SQP) methods. As the name implies, SQP methods are iterative methods which solve at each iteration a subquadratic programming problem (QP). More specifically, it constructs a subquadratic QP at the iteration k and solves each QP subproblems w.r.t. the current solution $\mathbf{x}_k$ (reiterates until convergence). Therefore, the basic idea is to approximate $f(\mathbf{x})$ by its local quadratic approximation with Taylor series expansion

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \tag{A.34}$$

Similarly, we can approximate the constraints

$$\begin{aligned} h_i(\mathbf{x}) &\approx h_i(\mathbf{x}_k) + \nabla h_i(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \\ g_i(\mathbf{x}) &\approx g_i(\mathbf{x}_k) + \nabla g_i(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \end{aligned}$$

For the problem A.33 at the k-th iteration (where we start from the current iterate $\mathbf{x}_k$) one solves for the next search direction

$$\begin{aligned} &\min_{\mathbf{p}} \frac{1}{2}\mathbf{p}^T H_k \mathbf{p} + \nabla f_k^T \mathbf{p} \\ &s.t.\ \nabla h_i(\mathbf{x}_k)^T \mathbf{p} + h_i(\mathbf{x}_k) = 0,\ i \in \boldsymbol{\xi} \\ &\nabla g_j(\mathbf{x}_k)^T \mathbf{p} + g_j(\mathbf{x}_k) \geq 0,\ j \in \mathcal{I}\} \end{aligned} \tag{A.35}$$

where H_k is usually a positive semi-definite approximation of the Hessian matrix of $f(\mathbf{x})$ and $\mathbf{p} = \mathbf{x} - \mathbf{x}_k$. Most modern general-purpose software is based on SQP techniques.

Let us restrict attention to the equality constrained case, i.e. assume for the moment that $\mathcal{I}$ is empty. Recall that our development of methods for unconstrained optimization has depended heavily on modeling the objective function f by a quadratic and solving nonlinear equations by linearizing them. This here is a direct extension of the same principles. Line search and trust region approaches can be devised. Indeed, almost everything that we have learned so far in this course comes into play in SQP methods!

A.5.1 Basic SQP and Newton-Lagrange

We can incorporate the Lagrange multiplier λ_i for the equality constraint $h_i(\mathbf{x}) = 0$, and get the following Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i h_i(\mathbf{x}) \tag{A.36}$$

Then the KKT conditions are given by

$$\begin{bmatrix} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \\ \nabla_{\boldsymbol{\lambda}_i} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \end{bmatrix} = \begin{bmatrix} \nabla f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i \nabla h_i(\mathbf{x}) \\ h_i(\mathbf{x}),\ i \in [1, l] \end{bmatrix} = 0 \tag{A.37}$$

Suppose that in the current iteration k, we have solution $\mathbf{x}_k$. Then at the step $k+1$, we need to find direction $\delta\mathbf{x}$, s.t. $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\mathbf{x}$. The main idea behind SQP is to model problem at the current point $\mathbf{x}_k$ by a quadratic programming subrpoblem and

then use the solution to this problem to construct a more accurate approximation $\mathbf{x}_{k+1}$.

Assume that $f(\mathbf{x})$ is twice differentiable objective function, then we perform a Taylor series expansion to A.37,

$$\begin{bmatrix} \nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) \\ \nabla_{\boldsymbol{\lambda}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) \end{bmatrix} + \begin{bmatrix} \nabla^2_{\mathbf{x}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) & \nabla h(\mathbf{x}) \\ \nabla h(\mathbf{x}) & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = 0 \tag{A.38}$$

where $\delta\mathbf{x} = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\delta\boldsymbol{\lambda} = \lambda_{k+1} - \lambda_k$. And we can rewrite it as follows

$$\begin{bmatrix} \nabla^2_{\mathbf{x}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) & \nabla h(\mathbf{x}) \\ \nabla h(\mathbf{x}) & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) \\ -\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) \end{bmatrix} = \begin{bmatrix} -\nabla_{\mathbf{x}} f(\mathbf{x}_k) - \nabla h(\mathbf{x}_k)\boldsymbol{\lambda}_k \\ -h(\mathbf{x}_k) \end{bmatrix} \tag{A.39}$$

Reminding that $\boldsymbol{\lambda}_{k+1} = \delta\boldsymbol{\lambda} + \boldsymbol{\lambda}_k$, we yield

$$\begin{bmatrix} \nabla^2_{\mathbf{x}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) & \nabla h(\mathbf{x}) \\ \nabla h(\mathbf{x}) & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \boldsymbol{\lambda}_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla_{\mathbf{x}} f(\mathbf{x}_k) \\ -h(\mathbf{x}_k) \end{bmatrix} \tag{A.40}$$

The Algorithm 21 below summarizes the Newton-Lagrange method for solving SQP problem A.36.

Initialize $(\mathbf{x}_0, \boldsymbol{\lambda}_0)$;
Set iteration index k = 0;
while $k < K$ **do**
  Solve $(\delta\mathbf{x}, \boldsymbol{\lambda}_{k+1})$ by solving the Lagrange-Newton system via A.5.1;
  Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\mathbf{x}$;
  Update $k = k + 1$;
  if (converged) break;
end

Algorithm 21: Lagrange-Newton method

Suppose that there exists a solution $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ satisfies A.36. A standard analysis of Newton-Lagrange method can show that if the KKT matrix is nonsingular, and $|(\mathbf{x}^*, \boldsymbol{\lambda}^*) - (\mathbf{x}_0, \boldsymbol{\lambda}_0)|$ is sufficiently small (neighborhood), where $f(\mathbf{x})$ and $h(\mathbf{x})$ are twice-continuously differentiable, then the SQP iterates $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ will converge to $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ at a Q-superlinear rate.

A.5.2 SQP with inequality constraints

As noticed before, if we consider inequality constraints according to A.35, we can construct Lagrangian function

$$\min \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^{l} \mu_j h_j(\mathbf{x}) \tag{A.41}$$

Taking the second derivatives, we yield

$$\begin{aligned} \min \nabla \mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\mathbf{p} + \frac{1}{2}\mathbf{p}\nabla^2 \mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\mathbf{p} \\ s.t.\ h(\mathbf{x}_k) + \nabla h_i(\mathbf{x}_k)\mathbf{p} = 0 \\ g(\mathbf{x}_k) + \nabla g_i(\mathbf{x}_k)\mathbf{p} \leq 0 \end{aligned} \tag{A.42}$$

where $\boldsymbol{\lambda}_k$ and $\boldsymbol{\mu}_k$ are the Lagrangian multipliers.

We can transform inequality constraints A.42 into equality constraints with slack variables, which can be easily solved

$$\begin{aligned} \min \nabla \mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\mathbf{p} + \frac{1}{2}\mathbf{p}\nabla^2 \mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\mathbf{p} \\ s.t.\ h(\mathbf{x}_k) + \nabla h_i(\mathbf{x}_k)\mathbf{p} = 0 \\ g(\mathbf{x}_k) + \nabla g_i(\mathbf{x}_k)\mathbf{p} + \mathbf{z} = 0 \end{aligned} \tag{A.43}$$

where $\mathbf{z} \geq 0$. Then we can treat it as SQP with equality constraints to solve it.

A.6 Other nonlinear optimization techniques

Exception the quadratic programming mentioned above, there are other QP methods, such as interior point and gradient projection methods. Except those approaches, there are another two widely used optimization methods: penalty and augmented Lagrangian methods. In the following part, we will give an brief introduction to these two methods.

Suppose we want to optimize the objective function with equality constraints:

$$\begin{aligned} \min \quad & f(x) \\ & s.t.\ h_i(x) = 0,\ i = [1, l] \end{aligned} \tag{A.44}$$

Penalty method: the basic idea of penalty method is to define a quadratic penalty function

$$\min f(x) + \frac{\mu}{2} \sum_i h_i^2(x) \tag{A.45}$$

where $\mu > 0$ is the penalty parameter. Specifically, it becomes a single unconstrained optimization problem, by including a nonnegative constraints into A.44. We can minimize this unconstrained function by gradually increasing the value of μ, until the objective function above converge to sufficient accuracy.

Augmented Lagrangian method: it combines the properties of the Lagrangian function and the quadratic penalty function together

$$\mathcal{L}(\mathbf{x}; \lambda, \mu) = f(\mathbf{x}) + \sum_i \lambda_i h_i(\mathbf{x}) + \frac{\mu}{2} \sum_i h_i^2(\mathbf{x}) \tag{A.46}$$

The augmented Methods basically optimize the above function in an alternative manner: fix μ to estimate the optimal Lagrange multiplier vector and fix λ to some positive value, then find a value of $\mathbf{x}$ that approximately minimizes $\mathcal{L}(\mathbf{x}; \lambda, \mu)$.

A.7 Newtons method for nonlinear equations

As mentioned in previous sections, we can convert constrained problem into unconstrained problem, which can make QP problem much easy. In this section, we will introduce numerical methods (Newton) method to optimize unconstrained problems.

Optimization problem 1: assume that $f(\mathbf{x})$ is a first order differential function, and we want to find solution for the objective function below

$$f(\mathbf{x}) = 0, \tag{A.47}$$

As in Newtons method for one variable, we need to start with the current approximation $\mathbf{x}_k$. Then we can derive the formula for a better approximation $\mathbf{x}_{k+1}$. According to Taylor series expansion, we can use the first order to approximate $f(\mathbf{x})$, with

$$f(\mathbf{x}) \approx f(\mathbf{x}) + \frac{\partial f(\mathbf{x_k})}{\partial x}(\mathbf{x} - \mathbf{x_k}) = \mathbf{f}(\mathbf{x}) + \frac{\partial \mathbf{f}(\mathbf{x_k})}{\partial \mathbf{x}}\mathbf{\Delta x} \tag{A.48}$$

We wish to find $\mathbf{x}$ that makes f equal to the zero vectors, so lets choose $\mathbf{x}_{k+1}$ so that

$$f(\mathbf{x}) \approx f(\mathbf{x}) + \frac{\partial f(\mathbf{x_k})}{\partial x}(\mathbf{x} - \mathbf{x_k}) = \mathbf{0} \tag{A.49}$$

then we can get

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\frac{\partial f(\mathbf{x_k})}{\partial x})^{-1} f(\mathbf{x}_0) \tag{A.50}$$

provided that the inverse exists. The formula is the vector equivalent of the Newtons method formula we learned before. However, in practice we never use the inverse of a matrix for computations, so we cannot use this formula directly. Rather, we can do the following. First solve the equation

$$\frac{\partial f(\mathbf{x_k})}{\partial x}\Delta \mathbf{x} = -f(\mathbf{x}_k) \tag{A.51}$$

Since $\nabla f(\mathbf{x}_0)$ is a known matrix and $f(\mathbf{x}_0)$ is a known vector, this equation is just a system of linear equations, which can be solved efficiently and accurately (Even when the system of equations is singular, you can find a least-squares solution by solving the system $A^T A\mathbf{x} = A^T b$). Once we have the solution vector x, we can obtain our improved estimate $\mathbf{x}_1$ by

$$\mathbf{x}_1 = \mathbf{x}_0 + \Delta \mathbf{x} \tag{A.52}$$

Optimization problem 2: assume that $f(\mathbf{x})$ is a second order differential function, and we want to minimize objective function below

$$\min f(\mathbf{x}) \tag{A.53}$$

According to Taylor series expansion, we can approximate $f(\mathbf{x})$ at $\mathbf{x}_k$ with the

following second order approximation:

$$f(\mathbf{x}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \tag{A.54}$$

Taking the derivative w.r.t. $\mathbf{x}$ and set to zero, we have

$$\nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0 \tag{A.55}$$

Then, we can update $\mathbf{x}$ as follows

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k) \tag{A.56}$$

Appendix B

Sets and Probabilities

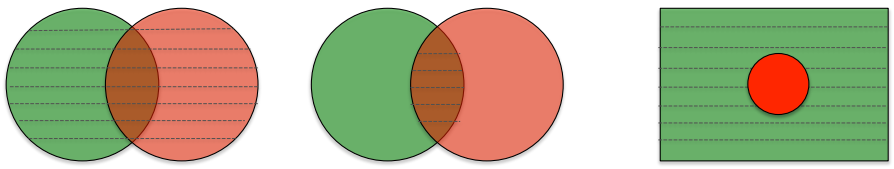
B.1 Introduction

this section will talk about probabilistic concepts here.

B.2 basic concepts

Let A and B be events in the same sample space. The Union of A and B (denoted by $A \cup B$) as the set consisting of those points belonging to either A or B. The intersection of A and B, written as $A \cap B$ as the set of points that belongs to both A and B. Define the complement of A, written A_c, as the points which do not belong to A. The visual understanding about these operations over sets is shown in Fig. B.1.

Analogously, the union of n events A_1, A_2, ..., A_n is defined as $\cap_{i=1}^{n} A_i = A_1 \cap$



(a). Union operation (b). Intersection (c). Complement

Figure B.1. The graph shows the operations between two sets A and B, and the results are marked with dash lines. (a) the union of A and B; (b) the intersection of A and B; (c) the complement of A.

$A_2 ... \cap A_n$, which indicates the set consisting of those points, which belong to at least one one the events A_1, A_2, ..., A_n. Similarly, the union definition of an infinite sequence of events A_1, A_2,... as the set of points belonging to at least one of the events A_1, A_2, ...(with notation $\cap_{i=1}^{\infty} A_i$).

Similarly, the intersection of an infinite sequence of events as the set of points belonging to all the events in the sequence and is denoted as $\cap_{i=1}^{\infty} A_i = A_1 \cap A_2 \cap ....$

For any two events or sets A and B in the same event, we call it mutually exclusive or disjoint if A and B have no points in common, denote as $A \cap B = \emptyset$.

Analogously, given a finite (or infinite) event sequence A_1, ..., A_n, we can them mutually exclusive if no two of the events share a common subset or point. More specifically, this condition can be written

$$A_i \cap A_j = \emptyset, \ \forall i, j, \ i \neq j \tag{B.1}$$

For any two events or sets A and B, if A and B are disjoint, then

$$P(A \cup B) = P(A) + P(B) \tag{B.2}$$

Example1.

properties: the union and intersection operations are commutative and associative

(1) commutative laws: For any two finite sets A and B: $A \cup B = B \cup A$, $A \cap B = B \cap A$

(2) associative laws: For any three finite sets A, B and C:

$$(A \cup B) \cup C = A \cup (B \cup C) \tag{B.3}$$

$$(A \cap B) \cap C = A \cap (B \cap C) \tag{B.4}$$

(3) Idempotent laws: the union or intersection of any finite set itself is itself.

$$A \cup A = A, \ A \cap A = A \tag{B.5}$$

(4) Distributive laws:

$$A \cup (B \cap C) = (A \cup C) \cap (A \cup C) \tag{B.6}$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{B.7}$$

(5) De Morgan's laws:

$$A - (B \cup C) = (A - B) \cap (A - C) \tag{B.8}$$

$$A - (B \cap C) = (A - B) \cup (A - C) \tag{B.9}$$

$$(A \cup B)^c = A^c \cap B^c \tag{B.10}$$

$$(A \cap B)^c = A^c \cup B^c \tag{B.11}$$

Intuitively, the probability of an event is a measure of how likely the event is to occur when we run the experiment. Mathematically, probability is a function on the collection of events that satisfies certain axioms.

Definition: A probability measure (or probability distribution) P for a random experiment is a real-valued function, defined on the collection of events, that satisfies the following conditions: (1) $P(A) \geq 0$, $A \in \mathcal{F}$; (2) $P(\Omega) = 1$; (3) If $\{A_i, i\ I\}$ is a countable, pairwise disjoint collection of events then

$$P(\cup A_{ii=1 \in I}) = \sum_{i \in I} P(A_i) \tag{B.12}$$

where $\mathcal{F}$ is the collection of all subsets of Ω, which forms a sigma field.

Definition: A probability space is a triple $(\Omega, \mathcal{F}, P)$, where Ω is a set, $\mathcal{F}$ a sigma field of subsets of Ω, and P is a probability measure on $\mathcal{F}$.

1. if $\emptyset$ denotes the empty set, then

$$P(\emptyset) = 0,\ P(\Omega) = 1 \tag{B.13}$$

2. for any set A and B, we have

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{B.14}$$

3. If $B \subset A$, then $P(B) \leq P(A)$; in fact

$$P(A - B) = P(A) - P(B) \tag{B.15}$$

4. Given a finite sets A_1, A_2, ..., we have

$$P(A_1 \cup A_2, ...) \leq P(A_1) + P(A_2) + ... \tag{B.16}$$

B.3 Conditional probability

1. Definition of event independency: two events A and B are independent iff $P(A \cap B) = P(AB) = P(A)P(B)$. In other words, the occurrence or nonoccurrence of A has not relationship to the event B, and vice versa.

2. The joint likelihood: given two events A and B, the joint likelihood is defined as $P(A \cap B) = P(AB)$. If A and B are independent, then we have the equation in 1.

3. Conditional probability: the conditional probability of A given B is defined as

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} \tag{B.17}$$

where $P(B) > 0$. Furthermore, we have $P(AB) = P(A|B)P(B) = P(B|A)P(A)$.

4. Theorem of total probability. Let A_1, A_2, ... be a finite or countable partitions of the space Ω (mutually exclusive and exhaustive events, such as $P(A_i \cap A_j) = 0$ and $\cup_i A_i = \Omega$), then for any event B, we have

$$P(B) = \sum_i P(A_i \cap B) = \sum_i P(A_i B) = \sum_i P(B|A_i)P(A_i) \tag{B.18}$$

Further, we can derive

$$P(A_k|B) = \frac{P(A_k B)}{P(B)} = \frac{P(A_k)P(B|A_k)}{\sum_i P(B|A_i)P(A_i)} \tag{B.19}$$

5. product decomposition and chain rule

$$P(AB) = P(B)P(A|B) = P(A)P(B|A) \tag{B.20}$$

$$P(X_1, X_2, ..., X_N) = P(X_1)P(X_2|X_1) \cdots P(X_N|X_1, ..., X_{N-1}) \tag{B.21}$$

6. Marginal likelihood. We just consider two random variable A and B, then we

can compute the marginal distribution

$$P(A) = \sum_B P(A, B) \text{ discrete case} \tag{B.22}$$

$$P(A) = \int_B P(A, B) \text{ continuous case} \tag{B.23}$$

B.4 Expectation and Variance

Let X be a numerically-valued random variable in the sample space Ω and the probability f(x), then the expected value is defined as

$$\mu = E(X) = \sum_i x_i P_i(x_i) = \int_{x \in \Omega} x f(x) dx \tag{B.24}$$

where $f(x)$ is the density function. some properties: Let X and Y be random variables with finite expected values, then

$$E(X + Y) = E(X) + E(Y) \tag{B.25}$$

And for any constant a, we have $E(aX) = aE(X)$.

If X and Y are two independent random variables, then we have

$$E(XY) = E(X)E(Y) \tag{B.26}$$

Similarly, we can define Variance

$$\Sigma(X) = E\big((X - \mu)^2\big) = \int (x - \mu)^2 f(x) dx \tag{B.27}$$

And the standard deviance is $\sigma(X) = \sqrt{\Sigma(X)}$

Some properties:

$$\Sigma(X) = E(X^2) - \mu^2,\ \Sigma(aX) = a^2 V(X),\ \Sigma(a + X) = \Sigma(X) \tag{B.28}$$

where a is any constant.

For any two independent random variables X and Y, we have

$$\Sigma(X+Y) = \Sigma(X) + \Sigma(Y) \tag{B.29}$$

B.5 Posterior probability

As mentioned before, we can compute the conditional probability $P(A|B)$ for the event A given the event B, and vice versa. If we know $P(A|B)$ and we want to infer $P(B|A)$, we need to know the prior distribution of the event B (or $P(B)$) according to Eq. Given conditional probability $P(A|B)$, we define the posterior probability $P(B|A)$ with

$$P(B|A) \propto P(A|B)P(B) \tag{B.30}$$

where $P(B)$ is the prior distribution of the event B.

Conjugate distributions: if the posterior distributions $p(\theta|x)$ are in the same family as the prior probability $p(\theta)$ the prior and posterior are then called conjugate distributions, and the prior is call a conjugate prior (which gives a closed form expression for the posterior). In other words, by assuming the conjugate prior, we can derive all the results (posterior) in closed form.— wiki

Example, suppose the data $D = x$ is generated from the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, and its likelihood is

$$p(D|\mu) \propto \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2 \right) \tag{B.31}$$

And further the parameter μ have Gaussian prior

$$p(\mu|\mu_0, \sigma_0) \propto \exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2 \right) \tag{B.32}$$

Then we can compute the posterior for μ as

$$p(\mu|D) \propto p(D|\mu, \sigma)p(\mu|\mu_0, \sigma_0)$$

$$
\begin{aligned}
&=\exp\Big(-\frac{1}{2\sigma^2}(x-\mu)^2\Big)\exp\Big(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\Big)\\
&=\exp\Big(-\frac{1}{2\sigma^2}(x-\mu)^2-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\Big)\\
&=\exp\Big(-\frac{1}{2\sigma^2}(x^2-2x\mu+\mu^2)-\frac{1}{2\sigma_0^2}(\mu^2-2\mu\mu_0+\mu_0^2)\Big)\\
&=\exp\Big(-\big(\mu^2(\frac{1}{2\sigma^2}+\frac{1}{2\sigma_0^2})-2\mu(\frac{x}{2\sigma^2}+\frac{\mu_0}{2\sigma_0^2})+(\frac{x^2}{2\sigma^2}+\frac{\mu_0^2}{2\sigma_0^2})\big)\Big) \qquad \text{(B.33)}
\end{aligned}
$$

Thus, we can see from Eq. B.33 that the posterior is still Gaussian. More specifically, the conjugate distribution of Gaussian is still Gaussian. By using maximum likelihood estimation, we can get the following posterior mean

$$
\begin{aligned}
\mu_1 &= \frac{\frac{x}{2\sigma^2}+\frac{\mu_0}{2\sigma_0^2}}{\frac{1}{2\sigma^2}+\frac{1}{2\sigma_0^2}} = \frac{x\sigma_0^2+\mu\sigma^2}{\sigma_0^2+\sigma^2}\\
&= \frac{\sigma^2}{\sigma_0^2+\sigma^2}\mu_0+\frac{\sigma_0^2}{\sigma_0^2+\sigma^2}x \qquad \text{(B.34)}\\
&= \mu_0+(x-\mu_0)\frac{\sigma_0^2}{\sigma_0^2+\sigma^2} \qquad \text{(B.35)}
\end{aligned}
$$

B.34 shows the average mean according two observations μ_0 and x, while B.35 in the last equation is the prior mean adjusted towards the data x. When there is more and more data observations, the likelihood will be more and more important. And correspondingly, the weight from prior μ_0 will become smaller and smaller. Thus B.34 expresses the tradeoff between prior and likelihood.

If we generate N points $D=\{x_i\}_{i=1}^N$ (i.i.d.), then the likelihood is

$$
p(D|\mu,\sigma)=\prod_{i=1}^N p(x_i|\mu,\sigma)\propto\prod_{i=1}^N\exp\Big(-\frac{1}{2\sigma^2}(x_i-\mu)^2\Big) \qquad \text{(B.36)}
$$

Further, we can easily extend Eq. B.33 to get the following posterior

$$
\begin{aligned}
&p(\mu|D)\propto p(D|\mu)p(\mu)\\
&=\exp\Big(-\big(\mu^2(\frac{1}{2\sigma^2}+\frac{N}{2\sigma_0^2})-2\mu(\frac{\sum_{i=1}^N x_i}{2\sigma^2}+\frac{\mu_0}{2\sigma_0^2})+(\frac{\sum_{i=1}^N x_i^2}{2\sigma^2}+\frac{\mu_0^2}{2\sigma_0^2})\big)\Big) \qquad \text{(B.37)}
\end{aligned}
$$

$$=\exp\left(-\frac{1}{2\sigma_n^2}(\mu-\mu_n)^2\right) \tag{B.38}$$

Apparently Eq. B.37 is still from Gaussian distribution. By matching the corresponding coefficients of μ^2, μ and constant between B.37 and B.38, we can estimate σ_n and μ_n

$$\frac{1}{2\sigma_n^2}=\frac{1}{2\sigma_0^2}+\frac{N}{2\sigma^2} \implies \frac{1}{\sigma_n^2}=\frac{1}{\sigma_0^2}+\frac{N}{\sigma^2} \tag{B.39}$$

$$\frac{\mu_n}{\sigma_n^2}=\frac{\sum_i x_i}{\sigma^2}+\frac{\mu_0}{\sigma_0^2} \implies \mu_n=\sigma_n^2(\frac{\sum_i x_i}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}) \tag{B.40}$$

Thus, we can see that μ_n in B.40 above can be further expressed as

$$\mu_n=\frac{\sigma^2}{N\sigma_0^2+\sigma^2}\mu_0+\frac{\sigma_0^2}{N\sigma_0^2+\sigma^2}\sum_{i=1}^{N}x_i=\frac{\sigma^2}{N\sigma_0^2+\sigma^2}\mu_0+\frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\bar{x} \tag{B.41}$$

Another way to understand these results is if we work with the precision of a Gaussian, which is $1/\sigma^2$. Higher precision, low variance. According to B.40 and B.39 then we have

$$\lambda_N=\lambda_0+N\lambda$$
$$\mu_n=\frac{\mu_0\lambda_0+N\bar{x}\lambda}{\lambda_n}$$

B.6 Common distributions

B.6.1 Binomial distribution

Assume the successful probability is $p\in[0,1]$. Then the probability that we exactly k successes in n trials is

$$prob(X=k)=\binom{n}{k}p^k(1-p)^{n-k} \tag{B.42}$$

where $\binom{n}{k}=\frac{n!}{(n-k)!\,k!}$ indicates all the different ways of k times successes in n trials. In addition, in n trials, we have k successes with probability p^k and $n-k$ failures with

probability $(1-p)^{n-k}$. After considering the all the combinations of k successes, we can easily get the above probability mass function.

B.6.2 Multinomial distribution

The multinomial distribution is the generalization of binomial distribution. Assume that for each trial, we have k possible outcomes, and its corresponding successful probability is $p_1, ..., p_k$, where $p_i \in [0,1]$ for $i = 1, ..., k$ and $\sum_{i=1}^{k} p_i = 1$. Then the probability that we exactly $n_1, ..., n_k$ successes in n trials is

$$prob(X_1 = n_1, ..., X_k = n_k) = \frac{n!}{n_1 \cdots n_k!} p_1^{n_1} \cdots p_k^{n_k} \tag{B.43}$$

where $\frac{n!}{n_1 \cdots n_k!} p_1^{n_1} \cdots p_k^{n_k}$ indicates the probability for all the different ways of $n_1, ..., n_k$ successes in n trials, and $\sum_{i=1}^{k} n_i = n$. After considering the all the combinations of $n_1, ..., n_k$ successes, we can easily get the above probability mass function.

B.6.3 Dirichlet distribution

Suppose that we have k-dimensional support vectors $x_1, ..., x_k$, with corresponding pseudo counts parameters $\alpha_1, ..., \alpha_k$, then we have the probability density function

$$p(x_1, ..., x_k) = \frac{1}{\mathcal{B}(\boldsymbol{\alpha})} \prod_{i=1}^{k} x_i^{\alpha_i - 1} \tag{B.44}$$

where $x_i > 0$ for $i = 1, ..., k$ and $\sum_{i=1}^{k} x_i = 1$. In general, it is denoted as $x \sim Dir(\alpha_1, ..., \alpha_k)$ or simply $x \sim Dir(\alpha)$. And the normalization constant is the multivariate Beta function

$$\mathcal{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k} \alpha_i)}, \ \Gamma(n) = (n-1)! \tag{B.45}$$

Dirichlet distribution is often used as the prior distribution (or conjugate prior) of multinomial distribution.

Some properties related to mean and covariance:

$$
\begin{aligned}
&mean:\ E\big((x_1,...,x_k)\big) = \left(\frac{\alpha_1}{\alpha},...,\frac{\alpha_k}{\alpha}\right),\ where\ \alpha = \sum_i \alpha_i \\
&covariance:\ Cov(x_i,x_j) = \frac{-\alpha_i\alpha_j}{\alpha^2(\alpha+1)} \\
&marginal\ distribution:\ x_i \sim Dir(\alpha_i, \sum_{j\neq i}\alpha_j)
\end{aligned}
\tag{B.46}
$$

B.6.4 Normal distribution

Given a real valued random variable x, the normal distribution $\mathcal{N}(\mu,\sigma)$ is defined

$$
p(x) = \mathcal{N}(x|\mu,\sigma) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\big\{-\frac{1}{2\sigma^2}(x-\mu)^2\big\} \tag{B.47}
$$

where μ is the mean and σ is the standard variance. Similarly, we generate it into the d dimensional case

$$
p(X) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\big\{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\big\} \tag{B.48}
$$

where Σ is the covariance matrix and $|\Sigma|$ is its corresponding determinant.

B.6.5 Gamma distribution

The gamma distribution is a flexible distribution for positive real valued random variable, $x > 0$. It is defined in terms of two parameters.

$$
p(x) = G(x|\alpha,\beta) \propto x^{\alpha-1}e^{-x\beta} \tag{B.49}
$$

where $\alpha > 0$ controls the shape of the distribution, and $\beta > 0$ controls the scale along the horizontal axis. In general, the Gamma distribution is widely used to model the prior over precision (related to variance in Gaussian distribution).

B.6.6 Wishart distribution

If we generate Gamma distribution into multidimension in $\mathbb{R}^d$, we can get the Wishart distribution. Let X is a d dimensional symmetric positive definite matrix, then the Wishart distribution is defined as

$$W(X|\Lambda) = \frac{1}{Z}|X|^{(\nu-d-1)/2}\exp\{-\frac{1}{2}tr(\Lambda^{-1}X)\} \tag{B.50}$$

where ν indicates the degrees of freedom with $\nu \geq d$, Λ is the scale matrix (positive semi-definite covariance), and $Z = 2^{(\nu d)/2}\Gamma_d(\nu/2)|\Lambda|^{\nu/2}$

B.6.7 Inverse Wishart distribution

This is the multidimensional generalization of inverse Gamma distribution in $\mathbb{R}^d$. Let X is a d dimensional symmetric positive definite matrix and $\nu > d-1$ is the degrees of freedom, then the Wishart distribution is defined as

$$W(X|\Lambda) = \frac{1}{Z}|X|^{-(\nu+d+1)/2}\exp\{-\frac{1}{2}tr(\Lambda X^{-1})\} \tag{B.51}$$

where Λ is the covariance matrix (PSD), and

$$Z = \frac{|\Lambda|^{\nu/2}}{2^{(\nu d)/2}\Gamma_d(\nu/2)} \qquad \Gamma_d(\nu/2) = \pi^{d(d-1)/4}\prod_{i=1}^{d}\Gamma(\frac{\nu+1-i}{2}) \tag{B.52}$$

B.7 Gaussian with Normal-inverse-Wishart prior

Now assume the prior μ, Σ, we know μ, but we do not know Σ. The conjugate prior for the covariance matrix of a Gaussian distribution with know mean is the Inverse-Wishart distribution, a multivariate generalization of the scaled inverse-χ^2 density. The d-dimensional Inverse-Wishart density function, with covariance Λ_0 and ν_0 degree of freedom is

$$p(\Sigma|\nu_0, \Lambda_0) \propto |\Lambda_0|^{-\frac{\nu_0+d+1}{2}}\exp\{-\Lambda_0\Sigma^{-1}\} \tag{B.53}$$

where $\nu > (d-1)$

Gaussian prior over $\mu|\Sigma \sim N(\mu_0, \Sigma/\kappa_0)$, where κ_0 can be thought as the pseudo–observations on the scale of observations (μ_0, Σ).

$$p(\mu|\Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|^{1/2}}}\exp\{-\frac{\kappa_0}{2}(\mu-\mu_0)^T\Sigma^{-1}(\mu-\mu_0)\} \tag{B.54}$$

Then by combining Eqs. B.53 and B.54, we can get the normal-inverse-wishart prior $p(\mu, \Sigma)$

$$p(\mu,\Sigma) \stackrel{\text{def}}{=} \frac{1}{Z}|\Sigma|^{-(\frac{\nu_0+d}{2}+1)}\exp\{-\frac{1}{2}tr(\Lambda_0\Sigma^{-1}) - \frac{\kappa_0}{2}(\mu-\mu_0)^T\Sigma^{-1}(\mu-\mu_0)\} \tag{B.55}$$

where $tr(\cdot)$ indicates trace, and Z has the following form

$$Z = \frac{2^{\nu_0 d/2}\Gamma_d(\nu_0/2)(2\pi/\kappa_0)^{d/2}}{|\Lambda_0|^{\nu_0/2}} \tag{B.56}$$

Consider a set of N of observations $D = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ from a multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with normal-inverse-Wishart prior $p(\mu, \Sigma)$, shown in Eq. B.55. Since the normal-inverse-Wishart is the conjugate prior $p(\mu, \Sigma) \sim NIW(\mu_0, \kappa_0, \Lambda_0, \nu_0)$, the posterior distribution $p(\mu, \Sigma|D, \mu_0, \kappa_0, \Lambda_0, \nu_0)$ is also NIW. In the following part, we will derive the posterior probability and estimate the model parameters (μ, Σ) with MAP.

$$\begin{aligned}
&p(\mu,\Sigma|D,\mu_0,\kappa_0,\Lambda_0,\nu_0) = \prod_{i=1}^{N}\frac{1}{\sqrt{2\pi|\Sigma|^{1/2}}}\exp\{-\frac{1}{2}(\mathbf{x}_i-\mu)^T\Sigma^{-1}(\mathbf{x}_i-\mu)\} \\
&\frac{1}{Z}|\Sigma|^{-(\frac{\nu_0+d}{2}+1)}\exp\{-\frac{1}{2}tr(\Lambda_0\Sigma^{-1}) - \frac{\kappa_0}{2}(\mu-\mu_0)^T\Sigma^{-1}(\mu-\mu_0)\} \\
&\propto |\Sigma|^{-(\frac{\nu_0+N+d}{2}+1)}\exp\{-\frac{1}{2}tr(\Lambda_0\Sigma^{-1})\} \\
&\exp\Big\{-\frac{1}{2}\sum_{i=1}^{N}tr\big((\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^T\Sigma^{-1}\big) - \frac{\kappa_0}{2}tr\big((\mu-\mu_0)(\mu-\mu_0)^T\Sigma^{-1}\big)\Big\} \\
&\propto |\Sigma|^{-(\frac{\nu_0+N+d}{2}+1)}\exp\{-\frac{1}{2}tr(\Lambda_0\Sigma^{-1})\} \\
&\exp\Big\{-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{x}_i\mathbf{x}_i^T - 2\mathbf{x}_i\mu^T + \mu\mu^T) - \frac{1}{2}\kappa_0(\mu\mu^T - 2\mu_0\mu^T + \mu_0\mu_0^T)\} \\
&\propto |\Sigma|^{-(\frac{\nu_0+N+d}{2}+1)}\exp\{-\frac{1}{2}tr(\Lambda_0\Sigma^{-1})\}
\end{aligned}$$

$$\exp\Big\{ -\frac{1}{2}\big((N+\kappa_0)\mu\mu^T - 2(\sum_{i=1}^{N}\mathbf{x}_i + \kappa_0\mu_0)\mu^T + \sum_i \mathbf{x}_i\mathbf{x}_i^T + \kappa_0\mu_0\mu_0^T\big)\Big\} \tag{B.57}$$

According to maximum a posterior, we estimate μ and Λ as follows

$$\begin{aligned}
\kappa_n &= \kappa_0 + N \\
\mu_n &= \frac{\kappa_0\mu_0 + \sum_{i=1}\mathbf{x}_i}{\kappa_0 + N} = \frac{\kappa_0}{\kappa_0+N}\mu_0 + \frac{N}{\kappa_0+N}\bar{\mathbf{x}} \qquad \text{(B.58)} \\
\nu_n &= \nu_0 + N \\
S &= \sum_{i=1}^{N}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T \\
\Lambda_n &= \Lambda_0 + S + \frac{\kappa_0 N}{\kappa_0+N}(\bar{\mathbf{x}} - \mu_0)(\bar{\mathbf{x}} - \mu_0)^T \qquad \text{(B.59)}
\end{aligned}$$

Note that to infer Λ_n, we need to put μ back into Eq. B.57,

$$\begin{aligned}
p(\mu, \Sigma | D, \mu_0, \kappa_0, \Lambda_0, \nu_0) &= \prod_{i=1}^{N}\frac{1}{\sqrt{2\pi|\Sigma|^{1/2}}}\exp\{-\frac{1}{2}(\mathbf{x}_i - \mu)^T\Sigma^{-1}(\mathbf{x}_i - \mu)\} \\
&\propto |\Sigma|^{-(\frac{\nu_0+N+d}{2}+1)}\exp\Big\{ -\frac{1}{2}tr(\Lambda_0 + S + \kappa_0(\mu-\mu_0)(\mu-\mu_0)^T)\Sigma^{-1}\Big\} \qquad \text{(B.60)} \\
&\propto |\Sigma|^{-(\frac{\nu_0+N+d}{2}+1)}\exp\Big\{ -\frac{1}{2}tr\big(\Lambda_0 + S + \frac{\kappa_0 N}{\kappa_0+N}(\bar{\mathbf{x}} - \mu_0)(\bar{\mathbf{x}} - \mu_0)^T\big)\Sigma^{-1}\Big\} \qquad \text{(B.61)}
\end{aligned}$$

where we replace μ with Eq. B.58 in Eq. B.60.

Bibliography

[1] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[2] L. R. Rabiner, "Readings in speech recognition," A. Waibel and K.-F. Lee, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pp. 267–296.

[3] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *ICML*, 2001, pp. 282–289.

[4] C. Sutton and A. Mccallum, *Introduction to Conditional Random Fields for Relational Learning*. MIT Press, 2006.

[5] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[6] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *JMLR*, pp. 1453–1484, 2005.

[7] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

[8] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear svm," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 408–415. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390208

[9] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *JMLR*, pp. 551–585, 2006.

[10] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 133–142. [Online]. Available: http://doi.acm.org/10.1145/775047.775067

[11] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *In Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 681–687.

[12] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in Neural Information Processing Systems 7*. MIT Press, 1995, pp. 585–592.

[13] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 849–856.

[14] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007.

[15] C. E. Antoniak, "Mixtures of dirichlet processes with applications to bayesian nonparametric problems," *Annals of Statistics*, 1974.

[16] J. Sethuraman and R. C. Tiwari, "Convergence of Dirichlet measures and the interpretation of their parameter," Tech. Rep., 1981.

[17] C. E. Rasmussen, "The infinite gaussian mixture model," in *NIPS12*. MIT Press, 2000, pp. 554–560.

[18] D. M. Blei and M. I. Jordan, "Variational inference for dirichlet process mixtures," *Bayesian Analysis*, vol. 1, pp. 121–144, 2005.

[19] R. M. Neal, "Markov chain sampling methods for dirichlet process mixture models," *JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS*, pp. 249–265, 2000.

[20] A. Vlachos, Z. Ghahramani, and A. Korhonen, "Dirichlet process mixture models for verb clustering," in *ICML workshop*, 2008.

[21] D. Görür and C. E. Rasmussen, "Dirichlet process gaussian mixture models: Choice of the base distribution," *J. Comput. Sci. Technol.*, 2010.

[22] Y. W. Teh, "Dirichlet processes," in *Encyclopedia of Machine Learning*. Springer, 2010.

[23] K. Kurihara, M. Welling, and Y. Teh, "Collapsed variational Dirichlet process mixture models," in *Proc. Int. Jt. Conf. Artif. Intell.*, vol. 20, 2007, pp. 2796–2801.

[24] W. R. Gilks and P. Wild, "Adaptive rejection sampling for gibbs sampling," *Applied Statistics*, pp. 337–348, 1992.

[25] T. S. Ferguson, "A Bayesian analysis of some nonparametric problems," *The Annals of Statistics*, vol. 1, no. 2, pp. 209–230, 1973.

[26] D. Blackwell and J. MacQueen, "Ferguson distributions via Polya urn schemes," *The Annals of Statistics*, pp. 353–355, 1973.

[27] H. Ishwaran and L. F. James, "Gibbs sampling methods for stick-breaking priors," *Journal of the American Statistical Association*, pp. 161–173, 2001.

[28] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, "Bayesian data analysis," 2004.

[29] E. B. Sudderth, "Graphical models for visual object recognition and tracking," Ph.D. dissertation, MIT, 2006. [Online]. Available: http://www.cs.brown.edu/~sudderth/papers/sudderthPhD.pdf

[30] T. P. Minka, "A comparsion of numerical optimizers for logistic regression," Tech. Rep., 2003.

[31] M. Hoai and A. Zisserman, "Discriminative sub-categorization," in *CVPR*, 2013.

[32] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *ADVANCES IN LARGE MARGIN CLASSIFIERS*, 1999, pp. 61–74.

[33] L. Xiao, D. Zhou, and M. Wu, "Hierarchical classification via orthogonal transfer," in *ICML*, 2011.

[34] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 200–209.

[35] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng, "Online and batch learning of pseudo-metrics," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 94–.

[36] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *J. Mach. Learn. Res.*, vol. 10, pp. 207–244, Jun. 2009.

[37] N. Nguyen and R. Caruana, "Improving classification with pairwise constraints: A margin-based approach." in *ECML/PKDD (2)*, ser. Lecture Notes in Computer Science, W. Daelemans, B. Goethals, and K. Morik, Eds., vol. 5212. Springer, 2008, pp. 113–124.

[38] H. Zeng and Y. ming Cheung, "Semi-supervised maximum margin clustering with pairwise constraints," *IEEE Trans. Knowl. Data Eng.*, pp. 926–939, 2012.

[39] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, Sep. 1995.

[40] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[41] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, jul 2006.

[42] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *PROCEEDINGS OF THE IEEE*, 1989, pp. 257–286.

[43] A. McCallum, D. Freitag, and F. C. N. Pereira, "Maximum entropy markov models for information extraction and segmentation," in *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 591–598.

[44] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[45] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[46] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1953039

[47] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *PAMI*, pp. 1798–1828, 2013.

[48] J. Weston and F. Ratle, "Deep learning via semi-supervised embedding," in *International Conference on Machine Learning*, 2008.

[49] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, "Learning algorithms for the classification restricted boltzmann machine," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 643–669, Mar. 2012.

[50] Y. Tang, "Deep learning using support vector machines." in *Workshop on Representational Learning, ICML 2013*, vol. abs/1306.0239, 2013.

[51] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Mar. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1756025

[52] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, 2014, pp. 580–587.

[53] A. Karpathy and F. Li, "Deep visual-semantic alignments for generating image descriptions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 3128–3137.

[54] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.

[55] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *ICML*. New York, NY, USA: ACM, 2007, pp. 791–798.

[56] N. Le Roux and Y. Bengio, "Representational power of restricted boltzmann machines and deep belief networks," *Neural Comput.*, vol. 20, no. 6, pp. 1631–1649, Jun. 2008.

[57] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *AISTATS*, 2009, pp. 448–455.

[58] ——, "An efficient learning procedure for deep boltzmann machines," *Neural Computation*, vol. 24, no. 8, pp. 1967–2006, 2012.

[59] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy." in *NIPS*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan Kaufmann, 1993, pp. 3–10.

[60] R. M. Neal and G. E. Hinton, "Learning in graphical models," M. I. Jordan, Ed. Cambridge, MA, USA: MIT Press, 1999, ch. A View of the EM Algorithm That Justifies Incremental, Sparse, and Other Variants, pp. 355–368.

[61] N. Srivastava and R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," *Journal of Machine Learning Research*, vol. 15, pp. 2949–2980, 2014.

[62] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[63] J. J. Hopfield, "Neurocomputing: Foundations of research," J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Neural Networks and Physical Systems with Emergent Collective Computational Abilities, pp. 457–464.

[64] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[65] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks." in *NIPS*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2008, pp. 545–552.

[66] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning." in *ICML*, 2011, pp. 689–696.

[67] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[68] 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 Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[69] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Advances in Neural Information Processing Systems 20*, 2007, pp. 1257–1264.

[70] ——, "Bayesian probabilistic matrix factorization using markov chain monte carlo," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 880–887.

[71] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *ICLR*, 2015.

[72] S. Sukhbaatar, a. szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2440–2448.