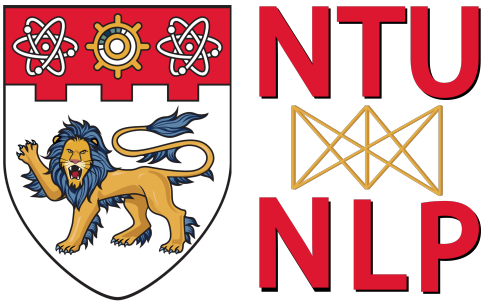
Deep Statistical Learning



My Note

Han Cheol Moon
School of Computer Science and Engineering
Nanyang Technological University
Singapore
hancheol001@edu.ntu.edu.sg

Contents

Chapter 1

Introduction

1.1 Probability

Definition 1 *Independence*

$$X \perp Y \leftrightarrow p(X,Y) = p(X)p(Y)$$

Definition 2 *Conditional independence*

$$X \perp Y|Z \leftrightarrow p(X,Y|Z) = p(X|Z)p(Y|Z)$$

All the dependencies between X and Y are mediated via Z. If X and Y are conditionally independent, then

$$\begin{aligned} p(X|Y,Z) &= \frac{p(X,Y|Z)}{p(Y|Z)} \\ &= \frac{p(X|Z)p(Y|Z)}{p(Y|Z)} \\ &= p(X|Z). \end{aligned}$$

1.2 Naive Bayes

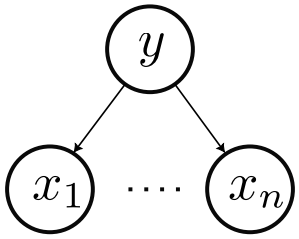
In this section, we discuss how to classify vectors of discrete-valued features $\mathbf{x}$. Recall that we discussed how to classify a feature vector $\mathbf{x}$ by applying Bayes rule to a generative classifier of the form

$$p(y = c|\mathbf{x}, \boldsymbol{\theta}) \propto p(\mathbf{x}|y = c, \boldsymbol{\theta})p(y = c|\boldsymbol{\theta})$$

The key to using such models is specifying a suitable form for the class-conditional density $p(\mathbf{x}|y = c, \boldsymbol{\theta})$, which defines what kind of data we expect to see in each class.

- $\mathbf{x} \in \{1, ..., K\}^D$,
  - K: the number of values for each feature.
  - D: the number of features.
- We will use a generative approach.

- Need to specify the class conditional distribution, $p(\mathbf{x}|y=c)$.
- A simple approach is to assume the features are **conditionally independence** given the class label.



- This allows us to write the class conditional density as a product of one dimensional densities:

$$p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_j|y=c,\boldsymbol{\theta}_{jc})$$

The resulting model is called a **naive Bayes classifier (NBC)**. The model is called "naive" since we assume the independence between the features, which is not true in practice. However, if often results in classifiers that work well.

The form of the class-conditional density depends on the type of each feature. We give some possibilities below:

- In the case of real-valued features, we can use the Gaussian distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} \mathcal{N}(x_j|\mu_{jc}^2)$, where μ_{jc} is the mean of feature j in objects of class c, and σ_{jc}^2 is its variance.
- In the case of binary features, we can use the Bernoulli distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Ber}(x_j|\mu_{jc})$, where μ_{jc} is the probability that feature j occurs in class c. This is sometimes called the **multivariate Bernoulli naive Bayes** model.
- In the case of categorical features, $x_j \in \{1,...,K\}$, we can model the multinomial distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Cat}(x_j|\mu_{jc})$, where $\boldsymbol{\mu}_{jc}$ is a histogram over the K possible values for x_j in class c.

The probability for a single data case is given by

$$p(\mathbf{x}_i, y_i|\boldsymbol{\theta}) = p(y_i|\boldsymbol{\pi}) \prod_j p(x_{ij}|\boldsymbol{\theta}_j) = \prod_c \pi_c^{\mathbb{I}(y_i=c)} \prod_j \prod_c p(x_{ij}|\boldsymbol{\theta}_{jc})^{\mathbb{I}(y_i=c)},$$

where $\boldsymbol{\pi}$ is a vector of class probability. Hence the log-likelihood is given by

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} \log p(x_{ij}|\boldsymbol{\theta}_{jc})$$

Algorithm 1: Fitting a naive Bayes classifier to binary features

Initialize $N_c = 0, N_{jc} = 0$;
for $i = 1 : N$ **do**
  $c = y_i$ //Class label of i-th example;
  $N_c := N_c + 1$;
  for $j = 1 : D$ **do**
    if $x_{ij} = 1$ **then**
      $N_{jc} := N_{jc} + 1$
  end
end
$\hat{\pi} = \frac{N_c}{N}, \hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$

1.3 Logistic Regression

Logistic regression corresponds to the following binary classification model:

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\mathbf{w}^T\mathbf{x}))$$

Logistic regression modles a logit (log odds) through a linear model. For binary data, the goal is to model the probability p that one of two outcomes occurs. The logit function is $\log\frac{p}{1-p}$, which varies between $-\infty$ and $+\infty$ as p varies between 0 and 1.

$$\log\frac{p}{1-p} = w_0x_0 + w_1x_1 + ... + w_nx_n$$

Note that the logistic regression model assumes that the log-odds (*logit*) of an observation y can be expressed as a linear function. In this context, the logit function is called the ***link function*** because it "links" the probability to the linear function of the predictor variables.

The negative log-likelihood for logistic regression is given by

$$\begin{aligned}\text{NLL}(\mathbf{w}) &= -\sum_{i=1}^{N}\log[\mu_i^{\mathbb{I}(y_i=1)} \times (1-\mu_i)^{\mathbb{I}(y_i=0)}] \\ &= -\sum_{i=1}^{N}[y_i\log\mu_i + (1-y_i)\log(1-\mu_i)], \ ^1\end{aligned}$$

where $\mu = \sigma(\mathbf{w}^T\mathbf{x})$. This is also called **cross-entropy** error function.

Another way to express NLL is as follows. Suppose $\hat{y}_i \in \{-1, +1\}$ instead of $y_i \in \{0, 1\}$. We have $p(y = 1) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$ and $p(y = -1) = \frac{1}{1+\exp(+\mathbf{w}^T\mathbf{x})}$. Hence

$$\begin{aligned}\text{NLL}(\mathbf{w}) &= -\frac{1}{N}\sum_{n=1}^{N}[\mathbb{I}(\hat{y}_n = 1)\log(\sigma(a_n)) + \mathbb{I}(\hat{y}_n = -1)\log(\sigma(-a_n))] \\ &= -\frac{1}{N}\sum_{n=1}^{N}\log(\sigma(\hat{y}_n a_n)) \\ &= \frac{1}{N}\sum_{i=1}^{N}\log(1+\exp(-\hat{y}_i\mathbf{w}^T\mathbf{x}_i).\end{aligned}$$

[1] $\mathbb{I}(y_i = 1) = y_i$, because $y_i \in \{0, 1\}$ is a binary variable

Note that the sigmoid is used for compressing the output into $[0, 1]$ and $\sigma(-a_n) = 1 - \sigma(a_n)$. Unlike the linear regression, there is no closed from solution for logistic regression, thus we need optimization algorithms for it. Typically, optimization process involves the gradient and Hessian.

$$\begin{aligned}
\mathbf{g} &= \frac{d}{d\mathbf{w}}\text{NLL}(\mathbf{w}) = \frac{d}{d\mu_i}\text{NLL}(\mathbf{w})\frac{d\mu_i}{d\mathbf{h}}\frac{d\mathbf{h}}{d\mathbf{w}} \\
&= \sum_{i=1}\left[-\frac{y_i}{\mu_i} + \frac{(1-y_i)}{(1-\mu_i)}\right]\frac{d\mu_i}{d\mathbf{h}}\frac{d\mathbf{h}}{d\mathbf{w}} = \sum_{i=1}\left[\frac{\mu_i - y_i}{\mu_i(1-\mu_i)}\right]\frac{d\mu_i}{d\mathbf{h}}\frac{d\mathbf{h}}{d\mathbf{w}} \\
&= \sum_i(\mu_i - y_i)\mathbf{x}_i = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}) \\
\frac{d\mu_i}{d\mathbf{h}} &= \mu_i(1-\mu_i) \\
\frac{d\mathbf{h}}{d\mathbf{w}} &= \mathbf{x}_i
\end{aligned}$$

where $\mathbf{h} = \mathbf{w}^T\mathbf{x}$.

We can also use the second-order method.

$$\begin{aligned}
\mathbf{H} &= \frac{d}{d\mathbf{w}}g(\mathbf{w})^T = \sum_i(\nabla_{\mathbf{w}}\mu_i)\mathbf{x}_i^T = \sum_i \mu_i(1-\mu_i)\mathbf{x}_i\mathbf{x}_i^T \\
&= \mathbf{X}^T\mathbf{S}\mathbf{X},
\end{aligned}$$

where $\mathbf{S} \triangleq \text{diag}(\mu_i(1-\mu_i))$. Note that $\mathbf{H}$ is positive definite, because the NLL is convex and has a global minimum.

Chapter 2

Training, Testing, and Regularization

2.1 Sources of Error in ML

$$E_{out} \leq E_{ml} + \Omega$$

- E_{out}: estimation of error.
- E_{ml}: error from a learning algorithm
- Ω: error caused by the variance from observations.

We also define

- f: target function
- g: learning function
- $g^{(D)}$: learned function based on D, or simply *hypothesis*.
- D: dataset drawn from the real world.
- $\bar{g}$: the average hypothesis of a given infinite number of Ds.

$$\bar{g}(x) = \mathbb{E}_D[g^{(D)}(x)].$$

Error of a single instance x from g learnt from D is given by

$$Err_{\text{out}}(g^{(D)}(x)) = \mathbb{E}_X[(g^{(D)}(x) - f(x))^2],$$

where X can be considered as test sets. Then, the expected error over the infinite number of datasets D sampled from a true data distribution is

$$\begin{aligned}\mathbb{E}_D[Err_{\text{out}}(g^{(D)}(x))] &= \mathbb{E}_D[\mathbb{E}_X[(g^{(D)}(x) - f(x))^2]] \\ &= \mathbb{E}_X[\mathbb{E}_D[(g^{(D)}(x) - f(x))^2]]\end{aligned}$$

Let's simplify the term inside with an average of hypothesis $\bar{g}(x)$:

$$\begin{aligned}\mathbb{E}_D[(g^{(D)}(x) - f(x))^2] &= \mathbb{E}_D[(g^{(D)}(x) - \bar{g}(x) + \bar{g}(x) - f(x))^2] \\ &= \mathbb{E}_D\big[(g^{(D)}(x) - \bar{g}(x))^2 + (\bar{g}(x) - f(x))^2 \\ &\quad + 2(g^{(D)}(x) - \bar{g}(x))(\bar{g}(x) - f(x))\big] \\ &= \mathbb{E}_D\big[(g^{(D)}(x) - \bar{g}(x))^2\big] + (\bar{g}(x) - f(x))^2 \\ &\quad + \mathbb{E}_D\big[2(g^{(D)}(x) - \bar{g}(x))(\bar{g}(x) - f(x))\big]\end{aligned}$$

Since, $\mathbb{E}_D\big[2(g^{(D)}(x) - \bar{g}(x))(\bar{g}(x) - f(x))\big]$ is 0, the expectation of the error becomes

$$\mathbb{E}_D[Err_{\text{out}}(g^{(D)}(x))] = \mathbb{E}_X\big[\mathbb{E}_D\big[(g^{(D)}(x) - \bar{g}(x))^2\big] + (\bar{g}(x) - f(x))^2\big].$$

Let's closely look at this formula. The errors are from two sources:

- **Variance**: $\mathbb{E}_D\big[(g^{(D)}(x) - \bar{g}(x))^2\big]$. Variance captures how much your classifier changes if you train on a different training set. We need to collect more data to reduce the variance.
- **Bias**: $(\bar{g}(x) - f(x))^2$. Bias is the inherent error that you obtain from your classifier even with infinite training data. We need to build a more complex model to reduce the bias.

However, if we reduce the bias, then the variance tends to increase.

2.1.1 Alternative Derivation

The derivation of the bias–variance decomposition for squared error proceeds as follows.[6][7] For notational convenience, abbreviate $f = f(x)$ and $\hat{f} = \hat{f}(x)$. First, recall that, by definition, for any random variable $\mathbf{X}$, we have

$$\operatorname{Var}\big[\hat{f}(x)\big] = \mathrm{E}[X^2] - \mathrm{E}[X]^2.$$

By rearranging, we get

$$\mathrm{E}[X^2] = \operatorname{Var}\big[\hat{f}(x)\big] + \mathrm{E}[X]^2.$$

Since f is deterministic

$$\mathrm{E}[f] = f$$

Thus, given $y = f + \varepsilon$ and $\mathrm{E}[\varepsilon] = 0$, implies $\mathrm{E}[y] = \mathrm{E}[f + \varepsilon] = \mathrm{E}[f] = f$

Also, since $\operatorname{Var}[\varepsilon] = \sigma^2$

$$\operatorname{Var}[y] = \mathrm{E}[(y - \mathrm{E}[y])^2] = \mathrm{E}[(y - f)^2] = \mathrm{E}[(f + \varepsilon - f)^2] = \mathrm{E}[\varepsilon^2] = \operatorname{Var}[\varepsilon] + \Big(\mathrm{E}[\varepsilon]\Big)^2 = \sigma^2$$

Thus, since ε and $\hat{f}$ are independent, we can write:

$$\begin{aligned}
\mathrm{E}\left[(y-\hat{f})^2\right] &= \mathrm{E}\left[(f+\varepsilon-\hat{f})^2\right] \\
&= \mathrm{E}\left[(f+\varepsilon-\hat{f}+\mathrm{E}[\hat{f}]-\mathrm{E}[\hat{f}])^2\right] \\
&= \mathrm{E}\left[(f-\mathrm{E}[\hat{f}])^2\right]+\mathrm{E}[\varepsilon^2]+\mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})^2\right]+2\,\mathrm{E}\left[(f-\mathrm{E}[\hat{f}])\varepsilon\right]+ \\
&\quad 2\,\mathrm{E}\left[\varepsilon(\mathrm{E}[\hat{f}]-\hat{f})\right]+2\,\mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})(f-\mathrm{E}[\hat{f}])\right] \\
&= (f-\mathrm{E}[\hat{f}])^2+\mathrm{E}[\varepsilon^2]+\mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})^2\right]+ \\
&\quad 2(f-\mathrm{E}[\hat{f}])\,\mathrm{E}[\varepsilon]+2\,\mathrm{E}[\varepsilon]\,\mathrm{E}\left[\mathrm{E}[\hat{f}]-\hat{f}\right]+2\,\mathrm{E}\left[\mathrm{E}[\hat{f}]-\hat{f}\right](f-\mathrm{E}[\hat{f}]) \\
&= (f-\mathrm{E}[\hat{f}])^2+\mathrm{E}[\varepsilon^2]+\mathrm{E}\left[(\mathrm{E}[\hat{f}]-\hat{f})^2\right] \\
&= (f-\mathrm{E}[\hat{f}])^2+\mathrm{Var}[y]+\mathrm{Var}\left[\hat{f}\right] \\
&= \mathrm{Bias}[\hat{f}]^2+\mathrm{Var}[y]+\mathrm{Var}\left[\hat{f}\right] \\
&= \mathrm{Bias}[\hat{f}]^2+\sigma^2+\mathrm{Var}\left[\hat{f}\right]
\end{aligned}$$

Part I

Kernel Methods

Chapter 3

Introduction to Kernel Methods

- The main idea is to use large set of fixed non-linear basis functions.
- The complexity depends on number of basis functions, but dual trick changes it to a size of dataset.

Kernel function: Let $\phi(\mathbf{x})$ be a set of basis functions that map inputs $\mathbf{x}$ to a feature space. In many algorithms, this feature space only appears in a dot product $\phi(\mathbf{x})^T\phi(\mathbf{x}')$ of input pairs $\mathbf{x}$ and $\mathbf{x}'$. Then, kernel function can be defined as $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$. Note that we only need to know $k(\mathbf{x}, \mathbf{x}')$, not $\phi(\mathbf{x})$.

- The kernel function is a measure of similarity between $\mathbf{x}_i$and $\mathbf{x}_j$

A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is said to be a positive semidefinite kernel if it is symmetric, (*i.e.,* $k(\mathbf{x}', \mathbf{x}) = k(\mathbf{x}, \mathbf{x}')$).

Recall that the linear regression can be modelee as follows:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w},$$

where $\mathbf{x} = [x_1, \ldots, x_m]^T$ and $\mathbf{w} = [w_1, \ldots, w_m]$. The ridge regression for $\mathbf{X} \in \mathbb{R}^{n\times m}$ matrix can be modeled as follows:

$$\begin{aligned}
J(\mathbf{w}) &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2 && (3.1)\\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w} && (3.2)\\
&= (\mathbf{y}^T - \mathbf{w}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w} && (3.3)\\
&= \mathbf{y}^T\mathbf{y} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\lambda\mathbf{I}\mathbf{w} && (3.4)\\
\frac{\partial J}{\partial \mathbf{w}} &= -\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w} + \mathbf{X}^T\mathbf{X}\mathbf{w} + 2\lambda\mathbf{I}\mathbf{w} = 0 && (3.5)\\
\mathbf{w} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} && (3.6)
\end{aligned}$$

Let's change it into a dual form:

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.7}$$
$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{y} \tag{3.8}$$
$$\mathbf{w} = \lambda^{-1}\mathbf{I}(\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w}) \tag{3.9}$$
$$= \mathbf{X}^T\lambda^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{3.10}$$
$$= \mathbf{X}^T\alpha \tag{3.11}$$
$$\lambda\alpha = (\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{3.12}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{X}^T\alpha) \tag{3.13}$$
$$\mathbf{y} = (\mathbf{X}\mathbf{X}^T\alpha + \lambda\alpha) \tag{3.14}$$
$$\alpha = (\mathbf{X}\mathbf{X}^T + \lambda)^{-1}\mathbf{y} \tag{3.15}$$
$$\alpha = (\mathbf{G} + \lambda)^{-1}\mathbf{y}, \tag{3.16}$$

where $\mathbf{G} = \mathbf{X}\mathbf{X}^T$ is a *Gram matrix*. Thus, we just have to solve $m \times m$ matrix.

Therefore, given a new input $\mathbf{x}' \in \mathbb{R}^m$, the prediction $f(\mathbf{x}') = (\mathbf{x}')^T\mathbf{w}$ can be expressed as follows:

$$f(\mathbf{x}') = (\mathbf{x}')^T\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y}.$$

Note that this formula depends only on the data via inner products.

$$(\mathbf{x}')^T\mathbf{X}^T = \begin{bmatrix} \langle \mathbf{x}', \mathbf{x}_1 \rangle \\ \vdots \\ \langle \mathbf{x}', \mathbf{x}_n \rangle \end{bmatrix}^T, \quad \mathbf{X}\mathbf{X}^T = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \dots & \langle \mathbf{x}_1, \mathbf{x}_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1 \rangle & \dots & \langle \mathbf{x}_n, \mathbf{x}_n \rangle \end{bmatrix}$$

Therefore, we can apply the kernel trick and consider the more general prediction function. In other words, we can leverage a very large number of basis functions.

Chapter 4

Gaussian Process

4.1 Introduction

Gaussian processes are distributions over functions $f(x)$ of which the distribution is defined by a mean function $m(x)$ and positive definite covariance function $k(x, x')$:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

Let $f(x) = \phi(x)^T w$, with $w \sim \mathcal{N}(0, \alpha^{-1}\mathbf{I})$. Then, $m(x) = \mathbb{E}[f(x)] = \mathbb{E}[w]^T \phi(x) = 0$

$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] = \mathbb{E}[f(x)(f(x')]$

Chapter 5

Support Vector Machine

5.1 Introduction

5.1.1 Orthogonal Projection

Given two vectors x and y, we would like to find the orthogonal projection of x onto y.

By definition:

$$||z|| = ||x||cos(\theta).$$

Note that the dot product of x and y is

$$cos(\theta) = \frac{x \cdot y}{||x|| \cdot ||y||}.$$

So we can replace the cosine

$$||z|| = ||x|| \frac{x \cdot y}{||x|| \cdot ||y||}.$$

This results in

$$||z|| = u \cdot x,$$

where u is an unit vector of y, which has the same direction as z. Therefore we can express z as follows:

$$z = ||z|| \cdot u,$$

Then,

$$z = (u \cdot x)u.$$

Equivalently,

$$\text{Proj}_y x = \left(\frac{y \cdot x}{||y||^2} \right) y.$$

5.2 Decision Boundary with Margin

Support vectors are the data points that lie closest to the decision surface(or hyperplane). They are directly related to the optimal hyperplane. The goal of SVM is to find the optimal separating

hyperplane which maximizes the margin of the training data. The hyperplane can be written as the set of points $\mathbf{x}$, satisfying

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Note that the hyperplain is normal to the vector $\mathbf{w}$.

$$\mathbf{w}(\mathbf{x} - \mathbf{x}_0) = 0,$$

where $b = \mathbf{w} \cdot \mathbf{x}_0$. However, what is the optimal separating hyperplanes? The optimal hyperplane is the one which maximizes the margin of the training data. SVMs maximize the margin around the separating hyperplane. The decision function is fully specified by a subset of training samples, the support vectors. Let's consider a simple case, where training data is linearly separable, $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^p,\ y_i \in \{-1, 1\}\}_{i=1}^n$. Then, we can build two hyperplanes separating the data with no points between them:

- $H_1 : \mathbf{w} \cdot \mathbf{x} + b = 1$
- $H_2 : \mathbf{w} \cdot \mathbf{x} + b = -1$

There are two constraints:

1. $\mathbf{w} \cdot \mathbf{x} + b \geq 1$
2. $\mathbf{w} \cdot \mathbf{x} + b \leq -1$

These can be combined as follows:

$$y(\mathbf{w} \cdot \mathbf{x} + b) \geq 1.$$

To maximize the margin, we can consider a unit vector $\mathbf{u} = \frac{\mathbf{w}}{||\mathbf{w}||}$, which is perpendicular to the hyperplanes and a point x_0 on the hyperplane H_2. If we scale u from x_0, we get $z = x_0 + ru$. If we assume z is on H_1, then $\mathbf{w} \cdot z + b = 1$. This is equivalent to

$$\begin{aligned}
\mathbf{w} \cdot (x_0 + ru) + b &= 1 \\
\mathbf{w}x_0 + \mathbf{w}r\frac{\mathbf{w}}{||\mathbf{w}||} + b &= 1 \\
\mathbf{w}x_0 + r||\mathbf{w}|| + b &= 1 \\
\mathbf{w}x_0 + b &= 1 - r||\mathbf{w}||
\end{aligned}$$

x_0 is on H_2, so $\mathbf{w}x_0 + b = -1$

$$\begin{aligned}
-1 &= 1 - r||\mathbf{w}|| \\
r &= \frac{2}{||\mathbf{w}||}
\end{aligned}$$

Note that the scaled unit vector ru's magnitude is r. Thus, the maximization of margin is equivalent to maximize r. To maximize r, it is important to minimize the norm of $\mathbf{w}$. This is equivalent to an optimization problem.

$$\begin{aligned}
&\min ||w||, \quad \text{subject to} \\
&y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i.
\end{aligned}$$

This minimization problem gives the same result as the following:

$$\min \frac{1}{2}||w||^2, \quad \text{subject to}$$
$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i.$$

Now, we now have **convex quadratic optimization problem**. However, this hard margin cannot tolerate erroneous cases. There could be two solutions:

- Admits prediction errors.
- Use non-linearity (complex decision boundary).

5.3 Error Handling in SVM

Let's first try to solve the issue by allowing some prediction errors.

$$\min ||w|| + C \cdot N_e, \quad \text{subject to}$$
$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i.$$

where N_e is the number of errors. It means we consider all errors equally. This penalty approach is **0-1 loss**. This approach is not popular, since it is hard to solve. Another approach is to use a **slack variable** with **hinge loss**, instead of counting the number of errors.

$$\min ||w|| + C \sum_j \xi_j, \quad \text{subject to}$$
$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_j \quad \forall i, \xi_j \geq 0, \forall j.$$

Note that $\xi_j > 1$ when mis-classified by its definition:

$$\xi_j = (1 - (\mathbf{w}x_j + b)y_j)_+$$

Let's look at the new constraint. If some data points are mis-classified, then $\xi_j > 1$ and $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 0$. This approach is called **soft-margin SVM**. Lastly, how do we set C?

5.4 Kernel Trick

Applying the kernel trick simply means replacing the dot product of two examples in a dual form by a kernel function.

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \psi(\mathbf{x}_i) \cdot \psi(\mathbf{x}_j) \tag{5.1}$$

Equivalently,

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{5.2}$$

5.5 SVM Optimization: Lagrange Multipliers

$$\min_{\mathbf{x}} f(\mathbf{x})$$
$$\text{subject to} \quad g(\mathbf{x}) = 0.$$

The minimum of f is found when its gradient point in the same direction as the gradient of g. In other words, when:

$$\nabla f(\mathbf{x}) = \alpha \nabla g(\mathbf{x})$$

So if we want to find the minimum of f under the constraint g, we just need to solve the following function:

$$\mathcal{L}(\mathbf{x}, \alpha) = f(\mathbf{x}) - \alpha g(\mathbf{x})$$

Note that the α is called a *Lagrange multiplier*.

Recall that we want to solve the following convex quadratic optimization problem:

$$\min \frac{1}{2}||w||^2, \quad \text{subject to}$$
$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i.$$

We can reformulate the above problem as follows:

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}||w||^2 - \sum_i \alpha_i \Big[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \Big] \tag{5.3}$$

We could try to solve the optimization problem, but this problem can only be solved analytically when the number of examples is small. Thus, we will reformulate the problem in the duality principle.

To get the solution of the primal problem, we need to solve the following **Lagrangian problem**:

$$\max_{\mathbf{w},b} \min_{\alpha} \mathcal{L}(\mathbf{w}, b, \alpha)$$
$$\text{subject to} \quad \alpha_i \geq 0, \forall i.$$

You may have noticed that the method of Lagrange multipliers is used for solving problems with equality constraints, and here we are using them with inequality constraints. This is because the method still works for inequality constraints, provided some additional conditions (the **KKT conditions**) are met. We will discuss about this soon.

5.6 The Wolfe Dual Problem

The Lagrangian problem has m inequality constraints (where m is the number of training examples) and is typically solved using its *dual form*. The duality principle tells us that **an optimization problem can be viewed from two perspectives**: (*i*) The first one is the *primal problem*, a minimization problem in our case. (*ii*) The other one is the *dual problem*, which will be a maximization problem. What is interesting is that the maximum of the dual

problem will always be less than or equal to the minimum of the primal problem (we say it **provides a lower bound to the solution of the primal problem**).

In our case, we are trying to solve a convex optimization problem, and **Slater's condition** holds for affine constraints (Gretton, 2016), so Slater's theorem tells us that strong duality holds. Note that the strong duality denotes that the solutions from the dual and the primal are identical (the maximum of the dual problem is equal to the minimum of the primal problem).

Solving the minimization problem involves taking the partial derivatives of $\mathcal{L}$ with respect to $\mathbf{w}$ and b.

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}, b, \alpha) = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i$$
$$\nabla_b\mathcal{L}(\mathbf{w}, b, \alpha) = -\sum_i \alpha_i y_i$$

The first term gives

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

Let's substitute the objective function Eq. (5.3) with $\mathbf{w}$:

$$\begin{aligned}
\mathbf{W}(\alpha, b) &= \frac{1}{2}\Big(\sum_i \alpha_i y_i \mathbf{x}_i\Big) \cdot \Big(\sum_j \alpha_j y_j \mathbf{x}_j\Big) - \sum_i \alpha_i \left[y_i \left(\Big(\sum_j \alpha_j y_j \mathbf{x}_j\Big) \cdot \mathbf{x}_i + b \right) - 1 \right] \\
&= \frac{1}{2}\Big(\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j\Big) - \sum_i \alpha_i \left[y_i \left(\Big(\sum_j \alpha_j y_j \mathbf{x}_j\Big) \cdot \mathbf{x}_i + b \right) \right] + \sum_i \alpha_i \\
&= \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_i \alpha_i y_i b + \sum_i \alpha_i \\
&= \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_i \alpha_i y_i b
\end{aligned}$$

There is still b, but $b = 0$, so we can just remove it. Finally, we get

$$\mathbf{W}(\alpha, b) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{5.4}$$

This is the **Wolfe dual Lagrangian function**. Note that we transform the problem as a problem with regard to α. Also, this is again a quadratic programming problem. The optimization problem is also called the **Wolfe dual problem**:

$$\begin{aligned}
&\max_{\alpha} \mathbf{W}(\alpha, b) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\
&\text{subject to } \alpha_i \geq 0 \text{ for any } i = 1, \ldots, m \\
&\sum_{i=1}^{m} \alpha_i y_i = 0
\end{aligned}$$

Once we get the value of α, we can get the optimal $\mathbf{w}$ and b can be obtained by using $\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0$. Details will be covered in the following section.

5.7 Karush-Kuhn-Tucker conditions

Due to the inequality constraints, we need to set an additional requirement. The solution must also satisfy the **Karush-Kuhn-Tucker (KKT) conditions**.

The KKT conditions are first-order necessary conditions for a solution of an optimization problem to be optimal. Moreover, the problem should satisfy some regularity conditions. Luckily for us, one of the regularity conditions is Slater's condition, and we just saw that it holds for SVMs. Because the primal problem we are trying to solve is a convex problem, the KKT conditions are also sufficient for the point to be primal and dual optimal, and there is zero duality gap.

To sum up, if a solution satisfies the KKT conditions, we are guaranteed that **it is the optimal solution**. Note that solving the SVM problem is equivalent to finding a solution to the KKT conditions.

Part II

Generative Modeling

Chapter 6

Sampling Based Inference

6.1 Basic Sampling Methods

6.1.1 Inverse Transform Sampling

Inverse transform sampling is a basic method for pseudo-random number sampling, *i.e.,* for generating sample numbers at random from any probability distribution given its cumulative distribution function (CDF).

Assume that we already have a uniformly distributed random number generator, *e.g.,* np.random.randn()

1. Generate a random number $u \sim Unif[0, 1]$
2. Find the inverse of the desired CDF, $F_X^{-1}(x)$.
3. Compute $X = F_X^{-1}(u)$. The computed random variable X has distribution $F_X(x)$

However, it is **hard to compute the inverse of CDF** ($F_X(x)$)

- $F_X(x) : \mathbb{R} \mapsto [0, 1]$ is any CDF.
- CDF is a non-negative and non-decreasing (monotone) function that is continuous.
- Our objective is to simulate a random variable X distributed as F; that is, we want to simularte a X such that $P(X \leq x) = F(x)$.
- F is invertible since it is continuous and strictly increasing.

6.1.2 Ancestral Sampling

$$p(\mathbf{x}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2)\cdots$$

Sampling steps:

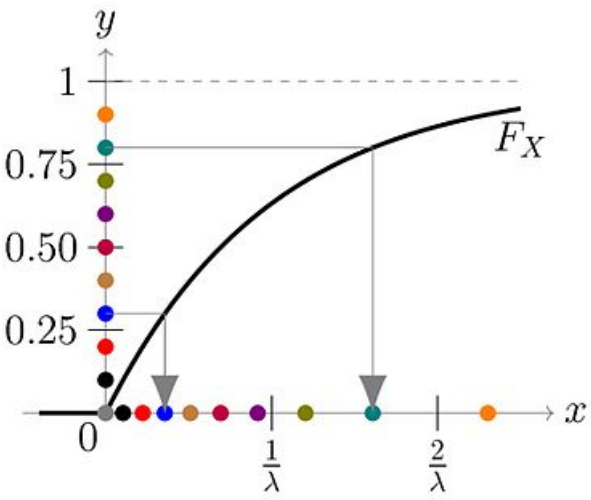


Figure 6.1: y-axis: Uniform distribution, x-axis: sample value

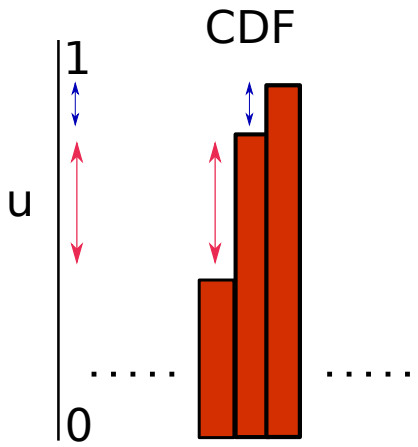


Figure 6.2: How can this sampling method recover the original distribution?

1. sample $\mathbf{x}_1$
2. sample $\mathbf{x}_2$ conditioned by $\mathbf{x}_1$
3. sample $\mathbf{x}_3$ conditioned by $\mathbf{x}_2$

6.1.3 Rejection Sampling

Rejection sampling is a simple method. It rejects samples violating a given condition (*e.g.*, conditions of conditional probability.). Let's see its theory.

Rejection sampling is a method for sampling from a distribution $p(x) = \frac{1}{Z}p'(x)$ that is difficulut to sample directly, but its unnormalized pdf $p'(x)$ is east to evaluate (Z is hard to compute). In rejection sampling, we need some simpler distribution $q(x)$, called a **proposal distribution**.

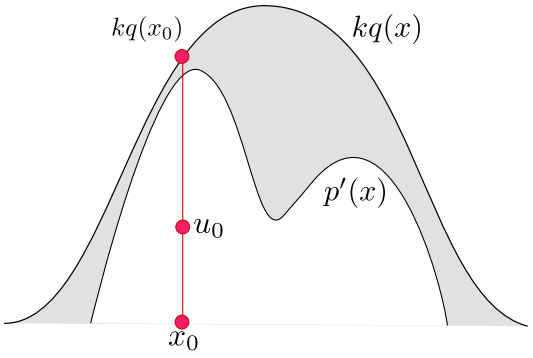
The intuition of rejection sampling is actually similar to Monte-Carlo estimation. By setting a large area (proposal distribution), we can sample points and take them that are inside the our

target distribution

To run the rejection sampling, introduce a constant k whose value is chosen such that $kq(x) \geq p'(x)$ for all values of x. The function $kq(x)$ is called a comparison function. Each step of the rejection sampler involves generating two random variables:

1. Sample $x_0 \sim q$
2. Sample $u_0 \sim U[0, kq(x_0)]$.

Finally, If $u_0 > p'(x_0)$, then the sample x_0 will be rejected, otherwise we add the sample x_0 to our set of samples $\{x^r\}$.



The original values of x are generated from the distribution $q(x)$ and these samples are then accepted with probability $p'(x)/kq(x)$ (see the figure above. The acceptance probability (*i.e.,* length) is the p' divided by kq). Then, the probability that a sample will be accepted is given by

$$\begin{aligned} p(accept) &= p\left(u \leq \frac{p'(x)}{kq(x)}\right) \\ &= \int p\left(u \leq \frac{p'(x)}{kq(x)}\middle| x\right) q(x)dx \\ &= \int \frac{p'(x)}{kq(x)} q(x)dx \\ &= \frac{1}{k}\int p'(x)dx \end{aligned}$$

Thus, the sampling will be more efficient if we choose small k to increase the change of acceptance.

6.1.4 Importance Sampling

We want to estimate an expectation of function $f(x), x \sim p(x)$, but it would be hard to estimate the distribution $p(x)$. Again, the importance sampling is not a method for generating samples from $p(\mathbf{x})$. In this case, we can use a simple distribution $q(x)$ by

$$\mathbb{E}(f) = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{N}\sum_{n=1}^{N}\frac{p(x_i)}{q(x_i)}f(x_i).$$

- Assume that $p(\mathbf{x})$ is known and too complicated to be sampled directly.

- Samples are independently drawn from a **proposal density** $Q(\mathbf{x})$, which is designed to be close to the true density $p(\mathbf{x})$ and **simpler**
- Generate R samples from $Q(\mathbf{x})$

6.2 Gibbs Sampling

The phrase "Markov chain Monte Carlo" encompasses a broad array of techniques that have in common a few key ideas. The setup for all the techniques that we will discuss in this book is as follows:

1. We want to sample from a some complicated density or probability mass function π. Often, this density is the result of a Bayesian computation so it can be interpreted as a posterior density. The presumption here is that we can evaluate π but we cannot sample from it.
2. We know that certain stochastic processes called Markov chains will converge to a stationary distribution (if it exists and if specific conditions are satisfied). Simulating from such a Markov chain for a long enough time will eventually give us a sample from the chain's stationary distribution.
3. Given the functional form of the density π, we want to construct a Markov chain that has π as its stationary distribution.
4. We want to sample values from the Markov chain such that the sequence of values $\{x_n\}$ generated by the chain converges in distribution to the density π.

In order for all these ideas to make sense, we need to first go through some background on Markov chains. The rest of this chapter will be spent defining all these terms, the conditions under which they make sense, and giving examples of how they can be implemented in practice.

6.2.1 Markov Chain

A Markov chain is a stochastic process that evolves over time by transitioning into different states. The sequence of states is denoted by the collection $\{X_i\}$ and the transition between states is random, following the rule

$$P(X_t|X_{t-1}, \ldots, X_0) = P(X_t|X_{t-1})$$

- Each node has a probability distribution of states.
- Each link represents a probability state transition.

- $i \to j$: Accessible if a state j is accessible from i.
  - $i \leftrightarrow j$: Communicate between the two states.
- Reducibility: A Markov chain is ***irreducible*** if $i \leftrightarrow j, \forall i, j \in S$. Simply, if all states can visit other states, then it is irreducible.
- Periodic: State i has a period d (*i.e.,* periodically visit the state i) $\leftrightarrow$ aperiodic.
- Transience: A state is transient if, when we leave this state, there is a non-zero probability that we will never return to it. Conversely, a state is recurrent if we know that we will return to that state, in the future, with probability 1 after leaving it (if it is not transient).

  - Stationary Distribution: A probability of being in a state s at time-step t is equal to a probability of being in the state s at the next time-step. Then, it is a stationary probability distribution.

- Ergodicity: A state is ergodic if the state is recurrent, aperiodic. Markov chain is ergodic if all states are ergodic.

6.2.2 Stationary Distribution

Limit theorem of Markov chain For a Markov chain with a discrete state space and transition matrix P, let π_* be such that $\pi_* P = \pi_*$. Then π_* is a stationary distribution of the Markov chain and the chain is said to be stationary if it reaches this distribution.

The basic limit theorem for Markov chains says that, under a specific set of assumptions that we will detail below, we have

$$||\pi_* - \pi_n|| \to 0$$

as $n \to \infty$, where $|| \cdot ||$ is the total variation distance between the two densities. Therefore, no matter where we start the Markov chain (π_0), π_n will eventually approach the stationary distribution. Another way to think of this is that

$$\lim_{n\to\infty} \pi_n(i) = \pi_*(i).$$

for all states i in the state space. Note that π_0 is the probability distribution of the Markov chain at time 0. Also, π_n denote the distribution of the chain at time n.

`https://bookdown.org/rdpeng/advstatcomp/background.html`

Reversible MC Consider a stationary ergodic Markov chain with transition probability$p(i, j)$ and stationary distribution $\pi(i)$, if we reverse the process, we will get a reversed Markov chain with transition probability$q(i, j)$:

$$\begin{aligned} q(j,i) &= P(X_m = i | X_{m+1} = j) \\ &= \frac{P(X_m = i, X_{m+1} = j)}{P(X_{m+1} = j)} \\ &= \frac{P(X_m = i | X_{m+1} = j) P(X_{m+1} = j)}{P(X_{m+1} = j)} \\ &= \frac{\pi(i) p(i,j)}{\pi(j)} \\ \pi(i)p(i,j) &= \pi(j)q(j,i) \end{aligned}$$

If $p(i, j) = q(j, i)$, it is called time-reversible Markov chain.

6.3 Markov Chain Monte Carlo

The basic sampling methods we have learnt so far do not leverage past information, which assumes all samples are independent. In Markov chain based sampling, we will treat random variables as a sequence of sampling process.

In Markov Chain Monte Carlo(MCMC), we assume that a stationary distribution is already known. We are more interested in estimating a transition rule that describing the stationary distribution.

6.3.1 Metropolis-Hasting Algorithm

- Current value z^t
- Propose a candidate $z^* \sim q(z^*|z^t)$ where q_t is a proposal distribution (*i.e.,* Normal distribution).
- Same as importance and rejections samplings, yet the difference is the Markov property idea in the proposal distribution.
- With an acceptance probability (or moving criteria), α.

Ref: YouTube Lecture

Chapter 7

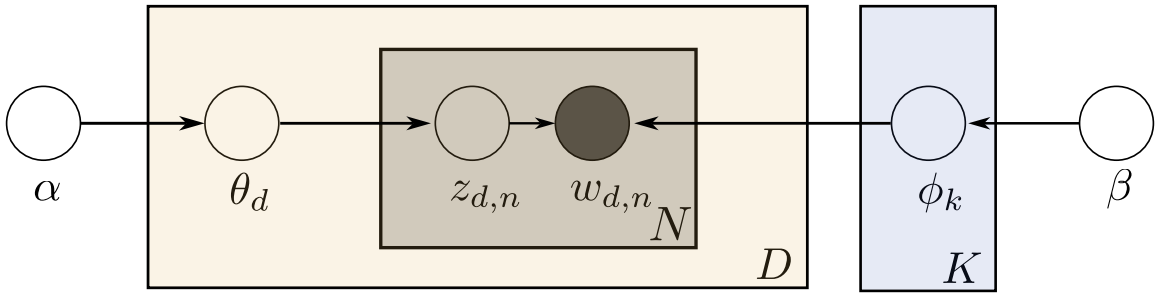
Topic Modeling

7.1 Latent Dirichlet Allocation

The assumptions of LDA:

- Each topic is a distribution over words.
- Each document is a mixture of corpus-wide topics.
- Each word is sampled from one of topics.

The LDA attempts to model the document generation process stochastically. However, we have to infer the latent structure (the distributions) of documents.



- $\theta_d \sim Dir(\alpha)$: For each document, draw topic distribution.
  - α: Dirichlet parameter
- $z_{d,n} \sim Mult(\theta_d)$: per-word topic assignment. The n-th word of document d is from which topic?
- $w_{d,n} \sim Mult(\phi_{z_{d,n}}, n)$: observed word. The n-th word in a document d is from a certain topic ($z_{d,n}$) distribution $\phi_{z_{d,n}}$.
- $\phi_k \sim Dir(\beta), i = \{1, \ldots, K\}$: topics.
  - β: topic hyperparameter (Dirichlet parameter).

The document generation process can be modelled as follows:

$$p(\phi_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\phi_i|\beta) \prod_{d=1}^{D} p(\theta_d|\alpha) \Big(\prod_{n=1}^{N} p(w_{d,n}|\phi_{1:K}, z_{d,n}) p(z_{d,n}|\theta_d) \Big).$$

7.1.1 LDA Inference

The posterior of the latent variables given the document is

$$p(\phi, \theta, \mathbf{z}|\mathbf{w}) = \frac{p(\phi, \theta, \mathbf{z}, \mathbf{w})}{\int_\phi \int_\theta \sum_{\mathbf{z}} p(\phi, \theta, \mathbf{z}, \mathbf{w})}$$

- The denominator is intractable

We want to estimate the topic distribution $\mathbf{z}$.

7.1.2 Dirichlet Distribution

The Dirichlet Distribution can be considered as a extension of the beta distribution.

$$p(P = \{p_i\}|\alpha_i) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i p_i^{\alpha_i - 1} \tag{7.1}$$

- $\sum_i p_i = 1$
- The posterior distribution of Dirichlet distribution is also Dirichlet distribution.

Chapter 8

Latent Variable Models

8.1 Introduction

8.1.1 Motivation of Latent Variable Models

If we knew a corresponding latent variable for each observation, then modelling might be easier. Imagine, how can we find $z^* =_z p(\mathbf{x}|\mathbf{z})$ for the data $\mathbf{x}$ as shown in Fig. 8.1(b)

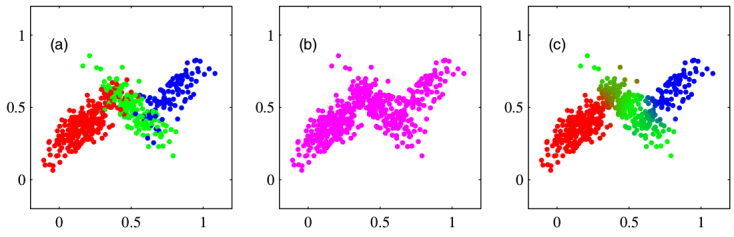


Figure 8.1: (a) Complete data set $p(\mathbf{x}|\mathbf{z})$. (b) Incomplete data set $p(\mathbf{x})$. (c) Inference result

For example, we want to model the complete data set $p(\mathbf{x}|\mathbf{z})$ under the i.i.d. assumption

$$p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) = \begin{cases} p(\mathcal{C}_1)p(\mathbf{x}_i|\mathcal{C}_1) \text{ if } z_i = 0 \\ p(\mathcal{C}_2)p(\mathbf{x}_i|\mathcal{C}_2) \text{ if } z_i = 1 \\ p(\mathcal{C}_3)p(\mathbf{x}_i|\mathcal{C}_3) \text{ if } z_i = 2 \end{cases}$$

$$p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N, \mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_N|\boldsymbol{\theta}) = \prod_{n=1}^{N}\prod_{k=1}^{K} \pi_k^{z_{nk}} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}$$

, where $\pi_k = p(\mathcal{C}_k)$ and $p(\mathbf{x}_i|\mathcal{C}_k) = \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. However, in many cases, it is not observable.

Chapter 9

Clustering

9.1 K-Means Clustering

Suppose we have a data set $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ consisting of N observations of a random D-dimensional variable $\mathbf{x} \in \mathbb{R}^D$. Our goal is to partition the data into some number K of clusters. Intuitively, we may think of a cluster as comprising a group of data points whose inter-point distances are small compared with the distances to points outside of the cluster.

This notion can be formalized by introducing a set of D-dimensional vectors $\boldsymbol{\mu}_k$, which represents the centers of the clusters. Our goal is to find an assignment of data points to clusters, as well as a set of vectors $\{\boldsymbol{\mu}_k\}$. Objective function of K-means clustering (*distortion measure*) can be defined as follows:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

, where $r_{nk} \in \{0, 1\}$ is a binary indicator variable which represents the **membership of data** $\mathbf{x}_n$. Our goal is to find values for the $\boldsymbol{\mu}_k$ and the r_{nk} so as to minimize J.

We can minimize J through an iterative procedure in which each iteration involves two successive steps corresponding to successive optimizations with respect to the $\boldsymbol{\mu}_k$ and the r_{nk} First we choose some initial values for the $\boldsymbol{\mu}_k$. Then in the first phase we minimize J with respect to the r_{nk}, keeping the $\boldsymbol{\mu}_k$ fixed. In the second phase we minimize J with respect to the $\boldsymbol{\mu}_k$, keeping r_{nk} fixed. This two-stage optimization is then repeated until convergence.

The r_{nk} can be optimized in a closed-form solution as follows:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \text{argmin}_j \, ||\boldsymbol{x}_n - \boldsymbol{\mu}_j||^2 \\ 0 & \text{otherwise} \end{cases}$$

Now consider the optimization of the $\boldsymbol{\mu}_k$ with the r_{nk} held fixed. The objective function J is a quadratic function of $\boldsymbol{\mu}_k$, and it can be minimized by setting its derivative with respect to $\boldsymbol{\mu}_k$ to zero giving

$$2 \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0.$$

We can arrange as

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\boldsymbol{x}_n}{\sum_n r_{nk}}.$$

The denominator of $\boldsymbol{\mu}_k$ is equal to the number of points assigned to cluster k. The mean of cluster k is essentially the same as the mean of data points $\mathbf{x}_n$ assigned to cluster k. For this reason, the procedure is known as the K-means clustering algorithm.

The two phases of re-assigning data points to clusters and re-computing the cluster means are repeated in turn until there is no further change in the assignments. These two phases reduce the value of the objective function J, so the convergence of the algorithm is assured. However, it may converge to a local rather than global minimum of J.

Some properties:

- Hard clustering ($\leftrightarrow$ Soft clusstering)
- Centroid initialization issue.
- The number of clusters is uncertain.
- Distance metric issue (*e.g.,* Euclidean?)

9.2 Gaussian Mixture Models

9.2.1 Multinomial Distribution

$$P(X|\boldsymbol{\mu}) = \prod_n \prod_k \mu_k^{x_{nk}} = \prod_k \mu_k^{\sum_n x_{nk}}.$$

How to determine the MLE solution of $\boldsymbol{\mu}$? *i.e.,* maximize$P(X|\boldsymbol{\mu})$ subject to $\mu_k \geq 0$ and $\sum_k \mu_k = 1$. We can use the Lagrange method.

$$\begin{aligned}
\mathcal{L} &= \sum_k \sum_n x_{nk} \ln \mu_k + \lambda(\sum_k \mu_k - 1) \\
\frac{\partial \mathcal{L}}{\partial \mu_k} &= \frac{\sum_n x_{nk}}{\mu_k} + \lambda. \\
\mu_k^{\mathrm{ML}} &= \frac{m_k}{N},
\end{aligned}$$

where $m_k = \sum_n x_{nk}$. We can consider the joint distribution of the quantities $m_1, \ldots, m_K$, conditioned on the parameters $\boldsymbol{\mu}$ and on the total number N observations:

$$\mathrm{Mult}(m_1, \ldots, m_K|\boldsymbol{\mu}, N) = \binom{N}{m_1, \ldots, m_K} = \frac{N!}{m_1!, \ldots, m_K!}.$$

Note that the variables m_k are subject to the constraint

$$\sum_k m_k = N.$$

9.2.2 Multivariate Gaussian Distribution

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

For a D-dimensional vector $\mathbf{x}$,

$$\begin{aligned}
\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \right) \\
\ln \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= -\frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) + C.
\end{aligned}$$

Note that the functional dependence of the Gaussian on $\mathbf{x}$ is through the quadratic form:

$$\Delta^2 = (\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}).$$

The quantity Δ is called the *Mahalanobis distance* from $\boldsymbol{\mu}$ to $\mathbf{x}$ and reduces to the Euclidean distance when $\boldsymbol{\Sigma}$ is the identity matrix.

Also,by using i.i.d. condition of a dataset, we can also express as follows:

$$\ln \mathcal{N}(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_n (\mathbf{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n-\boldsymbol{\mu}) + C$$

9.2.3 Gaussian Mixture Models

K-means clustering is a hard-clustering, but in some cases soft-clustering provides a better model in practice. Gaussian mixture model assumes a simple **linear superposition** of Gaussian components, aimed at providing a richer class of density models than the single Gaussian. Let's consider a single sample case and it can be expressed as follows:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma}_k)$$

Let us introduce a K-dimensional binary random variable $\mathbf{z}$ having a 1-of-K representation in which a particular element z_k is equal to 1 and all other elements are 0. I will explain more about $\mathbf{z}$ later. It satisfied the following properties:

- $z_k \in \{0, 1\}$
- $\sum_k z_k = 1$

The marginal distribution over $\mathbf{z}$ is specified in terms of the mixing coefficients π_k, such that

$$p(z_k = 1) = \pi_k$$

, where the mixing coefficients must satisfy

$$0 \leq \pi_k \leq 1$$

and

$$\sum_{k=1}^{K} \pi_k = 1$$

in order to be valid probabilities. We can also write pdf of $\mathbf{z}$ in a product of mixing coefficient because it is a 1-of-K representaion.

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k} = \pi_k \because z_k \in \{0, 1\}$$

Similarly, the conditional distribution of $\mathbf{x}$ given a particular $\mathbf{z}$ can be modeled to be a Gaussian distribution.

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Also can be represented in the form

$$\begin{aligned} p(\mathbf{x}|\mathbf{z}) &= \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k} \\ &= \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \because z_k \in \{0, 1\} \end{aligned}$$

Finally, marginal data distribution can be obtrained by summing the joint distribution over all possible states of $\mathbf{z}$ to give

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) \\ &= \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z}) = \sum_{z_1, ..., z_K} p(z_1, ..., z_K) p(\mathbf{x}|z_1, ..., z_K) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$

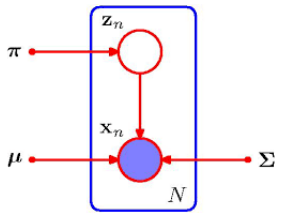


Figure 9.1: Graphical representation of GMM model. The GMM models a joint distribution $p(\mathbf{x}, \mathbf{z})$ in terms of a marginal distribution $p(\mathbf{z})$ and conditional distribution $p(\mathbf{x}|\mathbf{z})$ to model $p(\mathbf{x})$. Each $\mathbf{x}_n$ is coupled with $\mathbf{z}_n$

Note that for every observed data point $\mathbf{x}_n$ there is a corresponding latent variable $\mathbf{z}_n$, which **indicates the membership of** $\mathbf{x}_n$. This can be represented as in Fig. 9.1.

Now we can work with the joint distribution $p(\mathbf{x}, \mathbf{z})$ instead of the marginal distribution $p(\mathbf{x})$, which is hard to estimate directly as explained in §8.1.1.

Another quantity which plays a central role is the conditional proability of $\mathbf{z}$ given $\mathbf{x}$, $p(z_k = 1|\mathbf{x})$.

- $p(z_k = 1) = \pi_k$ can be viewed as a prior of $z_k = 1$
- $\gamma(z_k)$: assignment probability or responsibility. This represents the probability of assignment of a sample. This quantity will be updated through the Bayes Theorem.

  $\rightarrow$ A simple explanation is that **this is the classification result** of $\mathbf{x}_n$.

$$\begin{aligned}\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) &\equiv \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\mathbf{x}|z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}\end{aligned}$$

9.2.4 Maximum Likelihood

Suppose we have a data set of observations $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}^T \in \mathbb{R}^{N \times D}$ and we want to model the data distribution $p(\mathbf{X})$ using GMM. If we assume an i.i.d. data set, it can be expressed as follows:

$$p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

then its **loglikelihood function for GMM** is given by:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

How to solve this MLE? While a gradient-based optimization is possible, we consider the iterative *Expectation Maximization* algorithm.

Before, maximizing the likelihood, it is worth to emphasize two issues in GMM: (i) *singularities* and (ii) *identifiability*.

Singularity Before discussing how to maximize this function, it is worth emphasizing that there is a significant problem associated with the maximum likelihood framework applied to Gaussian mixture models, due to the presence of singularities. For simplicity, consider a Gaussian mixture whose components have covariance matrices given by $\Sigma_k = \sigma_k^2 I$, where I is the unit matrix, although the conclusions will hold for general covariance matrices. Suppose that one of the components of the mixture model, let us say the j-th component, has its mean $\boldsymbol{\mu}_j$ exactly equal to one of the data points so that $\boldsymbol{\mu}_j = \mathbf{x}_n$ for some value of n. This data point will then contribute a term in the likelihood function of the form

$$\mathcal{N}(\mathbf{x}_n|\mathbf{x}_n, \sigma_j^2 I) = \frac{1}{\sqrt{2\pi}\sigma_j}$$

If we consider the limit $\sigma_j \to 0$, then we see that this term goes to infinity and so the log likelihood function will also go to infinity. Thus the maximization of the log likelihood function is not a well posed problem because such singularities will always be present and will occur whenever one of the Gaussian components 'collapses' onto a specific data point. Recall that this problem did not arise in the case of a single Gaussian distribution as the variance can not be zero (recall the definition of variance).

Identifiability A further issue in finding MLE based solutions arises from the fact that for any given maximum likelihood solution, a K-component mixture will have a total ok $K!$ equivalent solutions corrsponding to the $K!$ ways of assigning K sets of parameters to K components. In other words, for any given point in the space of parameter values there will be a further $K! - 1$ additional points all of which give rise to exactly the same distribution.

9.2.5 Expectation Maximization for GMM

The goal of Expectation Maximization (EM) is to find maximum likelihood solutions for models having latent variables

- Suppose that it is hard to optimize $p(\mathbf{X}|\boldsymbol{\theta})$ directly.
- However, it is easier to optimize the complete-data likelihood function $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$
- In this case, we can use **EM algorithm**. EM algorithm is a general technique for finding maximum likelihood solutions for latent variable models.

Let us begin by writing down the conditions that must be satisfied at a maximum of the likelihood function. Setting the derivatives of $\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to the means $\boldsymbol{\mu}_k$ of the Gaussian components to zero, we obtain

$$0 = -\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

Multiplying by $\boldsymbol{\Sigma}_k^{-1}$ (which we assume to be non-singular) and rearranging we obtain

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n,$$

where we have defined

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}).$$

We can interpret N_k as the effective number of points assigned to cluster k. We can obtain the MLE solutions for other variables similarly.

Algorithm 2: EM algorithm for GMM

Initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k and evaluate the initial value of the log likelihood.

for n **do**

E step: evaluate the responsibilities of $\mathbf{x}_n$ based on the current parameter values with the given parameters

$$\gamma(z_{nk}) = p(z_k = 1|\mathbf{x}_n) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

where z_{nk} denote the k-th component of $\mathbf{z}_n$

M step: maximize expectation

- $\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n$
- $\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T$
- $\pi_k^{\text{new}} = p(z_k = 1) = \frac{N_k}{N}$

Evaluate the log likelihood to check for convergence of parameters

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

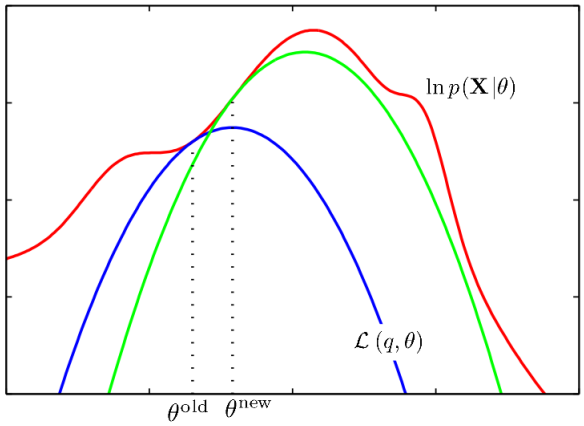


Figure 9.2: M-step of EM algorithm

9.3 Alternative View of EM

The goal of the EM algorithm is to find maximum likelihood (loglikelihood) solutions for models having latent variables.

$$\ln p(X|\theta) = ln \sum_Z p(X, Z|\theta).$$

We are not given the complete data set X, Z, but only the incomplete data X. Our state of knowledge of the values of the latent variables in Z is given only by the posterior distribution $p(Z|X, \theta)$. Because we cannot use the complete-data log likelihood, we consider instead its expected value under the posterior distribution of the latent variable, which corresponds (as we shall see) to the E step of the EM algorithm.

In the subsequent M step, we maximize this expectation. If the current estimate for the parameters is denoted θ_{old}, then a pair of successive E and M steps gives rise to a revised estimate θ^{new}.

The algorithm is initialized by choosing some starting value for the parameters θ_0. The use of the expectation may seem somewhat arbitrary.

In the E step, we use the current parameter values θ^{old} to find the posterior distribution of the latent variables given by $p(Z|X, \theta^{old})$. We then use this posterior distribution to find the expectation of the complete-data log likelihood evaluated for some general parameter value θ. This expectation, denoted $Q(\theta, \theta^{old})$, is given by

$$Q(\theta, \theta^{old}) = \sum_Z p(Z|X, \theta^{old}) \ln p(X, Z|\theta).$$

In the M step, we determine the revised parameter estimate θ^{new} by maximizing this function

$$\theta^{new} = \operatorname{argmax} Q(\theta, \theta^{old}).$$

Algorithm 3: General EM algorithm

The goal is to maximize the likelihood function $p(X|\theta)$ with respect to θ given a joint distribution $p(X, Z|\theta)$.

1. Init θ^{old}
2. E-Step: evaluate $p(Z|X, \theta^{old})$
3. M-Step: evaluate θ^{new} given by

$$\theta^{new} = \operatorname{argmax} Q(\theta, \theta^{old}),$$

where

$$Q(\theta, \theta^{old}) = \sum_Z p(Z|X, \theta^{old}) \ln p(X, Z|\theta).$$

4. Check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, then let

$$\theta^{old} \leftarrow \theta^{new}.$$

Return to the step 2.

9.4 Latent Variable Modeling

For each object x_i, we establish additional latent variable z_i which denotes the index of gaussian from which i-th object was generated. Then our model is

$$p(X, Z|\theta) = \prod_{i=1}^{n} p(x_i, z_i|\theta) = \prod_{i=1}^{n} p(x_i|z_i, \theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|\mu_{z_i}, \sigma_{z_i}^2)\pi_{z_i},$$

where $\pi_j = p(z_i = j)$ are prior probability of j-th gaussian and $\theta = \{\mu_j, \sigma_j, \pi_j\}_{j=1}^{K}$. If we know both X and Z then we can obtain explicit ML-solution:

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, p(X, Z|\theta) = \underset{\theta}{\operatorname{argmax}} \log p(X, Z|\theta).$$

However, in practice, we don't know Z, but only know X. Thus, we need to maximize w.r.t. θ the log of incomplete likelihood

$$\begin{aligned}
\log p(X|\theta) &= \ln \int p(X, Z|\theta) dZ && (9.1)\\
&= \ln \int q(Z|X) \frac{p(X, Z|\theta)}{q(Z|X)} dZ && (9.2)\\
&\geq \underbrace{\int q(Z|X) \ln \frac{p(X, Z|\theta)}{q(Z|X)} dZ}_{\text{ELBO, } \mathcal{L}(q,\theta)} \quad \text{by Jensen's Inequality.} && (9.3)\\
&= \int q(Z|X) \ln p(X, Z|\theta) - q(Z|X) \ln q(Z|X) dZ && (9.4)\\
&= \int q(Z|X)[\ln p(X|Z, \theta) + \ln p(Z|\theta)] - q(Z|X) \ln q(Z|X) dZ && (9.5)\\
&= \int q(Z|X) \ln p(X|Z, \theta) - q(Z|X) \ln \frac{q(Z|X)}{p(Z|\theta)} dZ && (9.6)\\
&= \mathbb{E}_{q(Z|X)} \ln p(X|Z, \theta) - KL(q(Z|X)||p(Z|\theta)) && (9.7)
\end{aligned}$$

To maximize the above equation, we need to minimize KL divergence.

9.4.1 Evidence Lower Bound (ELBO)

For any choice of inference model $q_\phi(z|x)$, we can represent the marginal probability of data (or model evidence) distribution, since the z is not related to x, so the integration does not affect x. Thus, we can also derive ELBO as follows:

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)]\\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x, z)}{p_\theta(z|x)}\right]\\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x, z) q_\phi(z|x)}{q_\phi(z|x) p_\theta(z|x)}\right]\\
&= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x, z)}{q_\phi(z|x)}\right]}_{=\mathcal{L}(\phi,\theta)(x)} + \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]}_{=D_{KL}(q_\phi(z|x)||p_\theta(z|x))}
\end{aligned}$$

To get more intuition about ELBO, we can express ELBO as follows:

$$\begin{aligned}
\mathcal{L}(\phi,\theta) &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(x,z) - \log q_\phi(z|x)\Big] \\
&= \mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(x) + \log p_\theta(z|x) - \log q_\phi(z|x)\Big] \\
&= \log p_\theta(x) - D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z|x)) \\
&\leq \log p_\theta(x)
\end{aligned}$$

ELBO can be also written as follows:

$$\begin{aligned}
\mathcal{L}(\phi,\theta) &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(x,z) - \log q_\phi(z|x)\Big] \\
&= \mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(z) + \log p_\theta(x|z) - \log q_\phi(z|x)\Big] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z))
\end{aligned}$$

We can get a conclusion that maximizing ELBO is equivalent to minimizing the KL divergence through the above equation. Fianlly, the log-likelihood can be rewritten as follows:

$$\log p_\theta(x) = \mathcal{L}(\phi,\theta) + D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z|x))$$

9.4.2 Expectation Maximization

We want to maximize ELBO, $\mathcal{L}(q,\theta)$ to minimize KL divergence between $q(Z)$ and $\log p(Z|X,\theta)$.

$$\max_{q,\theta} \mathcal{L}(q,\theta) = \max_{q,\theta} \int q(Z) \log \frac{p(X,Z|\theta)}{q(Z)} dZ.$$

We start from initial point θ_0 and iteratively repeat (*i*) E-step and (*ii*) M-step, iteratively:

- E-Step: θ_0 is fixed.

$$q(Z) = \operatorname*{argmax}_q \mathcal{L}(q,\theta) = \operatorname*{argmin}_q \mathrm{KL}(q(Z)|p(Z|X,\theta)) = p(Z|X,\theta_0).$$

  - This is because, maximizing ELBO is equal to minimizing KL divergence and the minimum q can be achieved when q is equal to $p(Z|X,\theta_0)$.
  - Now, we just have to evaluate $p(Z|X,\theta_0)$.

- M-Step: q is fixed.

$$\theta_* = \operatorname*{argmax}_\theta \mathcal{L}(q,\theta) = \operatorname*{argmax}_\theta \mathbb{E}_{q(Z)}[\log p(X,Z|\theta)]$$

  - Can be accomplished by taking derivatives
  - Set $\theta_0 = \theta_*$ and go to the E-Step until convergence

9.4.3 Categorical Latent Variables

$z_i \in \{1, ..., K\}$

$$p(x_i|\theta) = \sum_{k=1}^{K} p(x_i|k, \theta)p(z_i = k|\theta)$$

is simply a finite mixture of distributions.

E-Step:

$$q(z_i = k) = p(z_i = k|x_i, \theta) = \frac{p(x_k|z_i = k, \theta)p(z_i = k|\theta)}{\sum_{l=1}^{K} p(x_i|z_i = l, \theta)p(z_i = l|\theta)}$$

M-Step:

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{q(Z)}[\log p(X, Z|\theta)] = \sum_{i=1}^{n} \mathbb{E}_{q(z_i)}[\log p(x_i, z_i|\theta)] = \sum_{i=1}^{n}\sum_{k=1}^{K} q(z_i = k) \log p(x_i, k|\theta)$$

For GMM, we model $p(x|z)$ as Gaussian.

Chapter 10

Hidden Markov Models

10.1 Introduction

The HMM is based on the Markov chain assumption. A Markov chain is a model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather.

There are two important assumptions:

- Markov assumption
- Output independence: $p(x_i|z_1, \ldots, z_i, \ldots, z_T, x_1, \ldots, x_i, \ldots, x_T) = p(x_i|z_i)$

10.1.1 Conditional Independence

If two events A and B are **conditionally independent** given an event C then,

- $P(A \cap B|C) = P(A|C)P(B|C)$.
- $P(A|B, C) = P(A|C)$

10.1.2 Notation

- $X = (x_i, x_2, \ldots, x_T)$
- Initial state probabilities: $p(z_1) \sim \text{Multinomial}(\pi_1, ..., \pi_k)$, need to learn π
- Transition probability:

  $$p(z_t|z_{t-1} = i) \sim \text{Multinomial}(a_{i,1}, ..., a_{i,k})$$

  , where $a_{i,j} = p(z_t = j|z_{t-1} = i)$ and i and j denote clusters or states, respectively.
- Emission probability:

  $$p(x_t|z_t = i) \sim \text{Multinomial}(b_{i,1}, ..., b_{i,m})$$

  , where $b_{i,j} = p(x_t = j|z_t = i)$

10.2 Bayesian Network

10.2.1 Bayes Ball

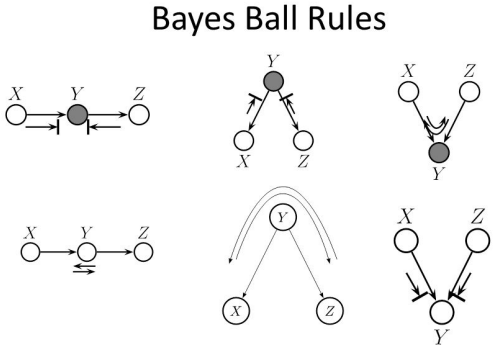


Figure 10.1: Bayes ball

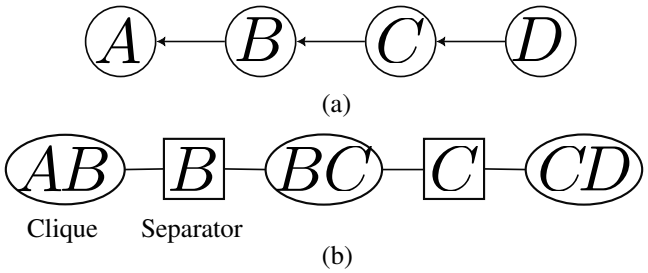
- Cascading: $P(Z|Y,X) = P(Z|Y)$. The information of Y decouples X and Z.
- Common parent: $P(X,Z|Y) = P(X|Y)P(Z|Y)$. The information of Y decouples X and Z.
- V-Structure (common child): Unlike the above two cases, the information of Y couples X and Z.

$$P(X,Y,Z) = P(X)P(Y)P(Y|X,Z).$$

10.2.2 Potential Function

Potential function is a function which is not a probability function, but it can become a probability function by normalizing it.

$$P(A,B,C,D) = P(A|B)P(B|C)P(C|D)P(D)$$



- Cliques: $\Psi(a,b), \Psi(b,c), \Psi(c,d)$
- Separators $\phi(b), \phi(c)$

Given a clique tree with cliques and separators, the joint probability distribution is defined as follows:

$$P(A,B,C,D) = P(U) = \frac{\prod_N \Psi(N)}{\prod_L \phi(L)} = \frac{\Psi(a,b)\Psi(b,c)\Psi(c,d)}{\phi(b)\phi(c)}$$

An effect of an observation propagates through the clique graph $\rightarrow$ **Belief propagation**. How to propagate the belief? **Absorption rule**!

Let's say we have some new observations about A, then it affects the clique $\Psi(a,b)$. The updated clique is now $\Psi^*(a,b)$. Similarly, $\phi^*(b) = \sum_A \Psi^*(a,b)$. Subsequently, $\Psi^*(b,c) = \Psi(b,c)\frac{\phi^*(b)}{\phi(b)}$.

10.3 Hidden Markov Models

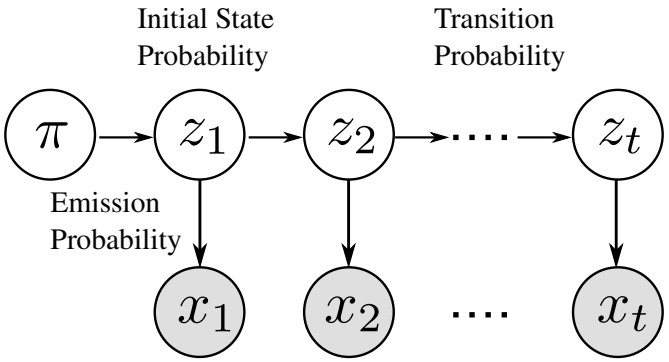


Figure 10.2: HMM Structure

The observation can be discrete or continuous. If the latent factors are continuous, then HMM is often referred as **Kalman filter**.

- Initial state probability: $P(z_1) \sim \text{Mult}(\pi_1, \ldots, \pi_k)$
- Transition probability: $P(z_t|z_{t-1}^i = 1) \sim \text{Mult}(a_{i,1}, \ldots, a_{i,k})$,
  where $P(z_t^j = 1|z_{t-1}^i = 1) = a_{i,j}$
- Emission probability: $P(x_t|z_t^i = 1) \sim \text{Mult}(b_{i,1}, \ldots, b_{i,m}) \sim f(x_t|\theta_i)$,
  where $P(x_t^j = 1|z_t^i = 1) = b_{i,j}$. The probability of observing x_j at the i-th cluster.

Note that i and j are indices of clusters.

There are three main problems in HMM:

1. Evaluation Questions (likelihood):
   - Given $\boldsymbol{\pi}, \mathbf{a}, \mathbf{b}, X$
   - Find $p(X|M, \boldsymbol{\pi}, \mathbf{a}, \mathbf{b})$
   - How much are X likely to be observed by a model M?
2. Decoding Questions:
   - Given $\boldsymbol{\pi}, \mathbf{a}, \mathbf{b}, X$
   - Find $\text{argmax}_Z\, p(Z|X, M, \boldsymbol{\pi}, \mathbf{a}, \mathbf{b})$
   - What is the most probable sequence of Z (latent states)?
3. Learning Questions: Forward-Backward (Baum-Welch)
   - Given X
   - Find $\text{argmax}_{\boldsymbol{\pi},\mathbf{a},\mathbf{b}}\, p(X|M, \boldsymbol{\pi}, \mathbf{a}, \mathbf{b})$
   - What would be the optimal model parameters?

10.4 Evaluation: Forward-Backward Probability

10.4.1 Joint Probability

We can factorize the joint distribution of HMM in Fig. 10.2 by using a Bayesian approach as follows:.

$$p(X,Z) = p(x_1,\ldots,x_t,z_1,\ldots,z_t) = p(z_1)p(x_1|z_1), p(z_2|z_1),\ldots,p(x_t|z_t), p(z_t|z_{t-1}) \tag{10.1}$$

As the number of latent factor increases, it is getting harder to decode the latent factors.

10.4.2 Marginal Probability

We want to compute the likelihood of sequence X which is given by

$$p(X|\boldsymbol{\pi},\mathbf{a},\mathbf{b}) = \sum_Z p(X,Z|\boldsymbol{\pi},\mathbf{a},\mathbf{b})$$

The computation can be done as follows:

$$\begin{aligned} p(X) &= \sum_Z p(X,Z) \\ &= \sum_{z_1}\cdots\sum_{z_t} p(x_1,\ldots,x_t,z_1,\ldots,z_t) \\ &= \sum_{z_1}\cdots\sum_{z_t} \pi_{z_1}\prod_{t=2}^{T} a_{z_{t-1},z_t}\prod_{t=1}^{T} b_{z_t,x_t} \end{aligned}$$

The last step is done by using Eq. (10.1)). The computation of this equation requires lots of computations, so we will change it into a **recursive form** by using the factorization rule $p(a,b,c) = p(a)p(b|a)p(c|a,b)$.

$$p(x_1,\ldots,x_t,z_t^k=1) = \sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},x_t,z_{t-1},z_t^k=1) \tag{10.2}$$

$$= \sum_{z_{t-1}} p(\underbrace{x_1,\ldots,x_{t-1},z_{t-1}}_{a},\underbrace{x_t}_{c},\underbrace{z_t^k=1}_{b}) \tag{10.3}$$

$$= \sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})p(z_t^k=1|x_1,\ldots,x_{t-1},z_{t-1})p(x_t|z_t^k=1,x_1,\ldots,x_{t-1},z_{t-1}) \tag{10.4}$$

$\because p(a,b,c) = p(a)p(b|a)p(c|a,b)$ or by the structure of HMM

$$= \sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})p(z_t^k=1|z_{t-1})p(x_t|z_t^k=1) \tag{10.5}$$

$$= p(x_t|z_t^k=1)\sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})p(z_t^k=1|z_{t-1}) \tag{10.6}$$

$$= b_{z_t^k,x_t}\sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})a_{z_{t-1},z_t^k} \tag{10.7}$$

- In the second line, the x_{t-1} and z_{t-1} are grouped together.
- Then, we can find the HMM structure by factorizing the equation.
- In the fourth line, x terms are removed, since z_t only relies on z_{t-1} by the Markov assumption. Similarly, x_t only depends on z_t. We can interpret this by using Bayes ball too.

Now we can find a recursive structure of $p(x_1, \ldots, x_t, z_t^k = 1)$ as follows:

$$\alpha_t^k = p(x_1, \ldots, x_t, z_t^k = 1) = b_{k,x_t} \sum_i \alpha_{t-1}^i a_{i,k}$$

, where α_t^k **is the probabilities of being in state** k **after observing the first** t **observations.** Thus,

$$\begin{aligned} p(x_1, \ldots, x_t) &= \sum_{\mathbf{z}} p(x_1, \ldots, x_t, z) \\ &= \sum_k \alpha_t^k \end{aligned}$$

Note that α_t^k is also called **Forward probability**.

10.4.3 Forward Algorithm

Forward probability solves the evaluation problem. Essentially, this is a dynamic programming, so it calculates required values in a bottom-up manner.

- Forward probability: α_t^k, $Time \times States$

Algorithm 4: Forward Algorithm

Create a probability matrix $forward[M, T] = \alpha_t^k$
Initialization:
for *each state k=1,...,M* **do**
  $\alpha_1^k \leftarrow \pi_k b_{k,x_1}$
for *time step t=2,...,T* **do**
  for *each step k=1,...,M* **do**
    $\alpha_t^k = b_{k,x_t} \sum_i \alpha_{t-1}^i a_{i,k}$
Return $p(X) = \sum_i^M \alpha_T^i$

Note again that

$$p(X) = p(x_1, ..., x_T) = \sum_i \alpha_T^i = \sum_i p(x_1, ..., x_T, z_T^i = 1)$$

Note also that the forward-algorithm returns $p(X)$ and forward probability is the probability of being in state k after observing the first t observations without Z.

10.4.4 Backward Probability

The forward probability only considers an observation at t. To determine the z_t, we need to leverage the future observations. **The backward probability β is the probability of seeing the observations from time $t+i$ to the end, given that we are in state k at time t.**

$$\beta_t^k = p(x_{t+1}, \dots, x_T | z_t^k = 1)$$

We want to compute $p(z_t^k = 1|X)$ rather than $p(x_1, \dots, x_t, z_t^k = 1)$. In other words, we will leverage the whole observations X.

$$\begin{aligned} p(z_t^k = 1, X) &= p(x_1, \dots, x_t, z_t^k = 1, x_{t+1}, \dots, x_T) \\ &= p(x_1, \dots, x_t, z_t^k = 1) p(x_{t+1}, \dots, x_T | x_1, \dots, x_t, z_t^k = 1) \\ &= p(x_1, \dots, x_t, z_t^k = 1) p(x_{t+1}, \dots, x_T | z_t^k = 1) \\ &= \alpha_t^k \beta_t^k \end{aligned}$$

We already know that $p(x_1, \dots, x_t, z_t^k = 1) = \alpha_t^k$. We just need to compute backward probability as follows:

$$\begin{aligned} \beta_t^k &= p(x_{t+1}, \dots, x_T | z_t^k = 1) \\ &= \sum_{z_{t+1}} p(\underbrace{z_{t+1}}_{a}, \underbrace{x_{t+1}}_{b}, \underbrace{x_{t+2}, \dots, x_T}_{c} | z_t^k = 1) \\ &= \sum_i p(z_{t+1}^i = 1 | z_t^k = 1) p(x_{t+1} | z_{t+1}^i = 1, z_t^k = 1) p(x_{t+2}, \dots, x_T | x_{t+1}, z_{t+1}^i = 1, z_t^k = 1) \\ &\because p(a, b, c) = p(a) p(b|a) p(c|a, b) \\ &= \sum_i p(z_{t+1}^i = 1 | z_t^k = 1) p(x_{t+1} | z_{t+1}^i = 1) p(x_{t+2}, \dots, x_T | z_{t+1}^i = 1) \\ &= \sum_i a_{k,i} b_{i,x_{t+1}} \beta_{t+1}^i \end{aligned}$$

Another recursive structure:

$$\begin{aligned} p(z_t^k = 1, X) &= \alpha_t^k \beta_t^k \\ &= b_{k,x_t} \sum_i \alpha_{t-1}^i a_{i,k} \times \sum_i a_{k,i} b_{i,x_t} \beta_{t+1}^i \end{aligned}$$

This means at time t, the latent label is belong to some class k and this can be computed by using the forward probability and the backward probability. Now we can compute

$$p(z_t^k = 1|X) = \frac{p(z_t^k = 1, X)}{p(X)} = \frac{\alpha_t^k \beta_t^k}{p(X)}$$

Then,

$$k_t = \underset{k}{\operatorname{argmax}}\, p(z_t^k = 1|X)$$

Note that this is for a single latent variable at a single time step given the whole observation X, but we want to decode a sequence of latent variables. Thus, we need some decoding algorithm.

10.5 Decoding: Viterbi Algorithm

For any model, such as an HMM, that contains hidden variables, **the task of determining which sequence of variables is the underlying source of some sequence of observations is called the decoding task**.

We might propose to find the best sequence as follows:

1. For each possible hidden state sequence (HHH, HHC, HCH, etc.), we could run the forward algorithm and compute the likelihood of the observation sequence given that hidden state sequence.
2. Then, we could choose the hidden state sequence with the maximum observation likelihood.

However, this is not a feasible solution, because there are an exponentially large number of state sequences.

Instead, the most common decoding algorithms for HMMs is the **Viterbi algorithm**. Like the forward algorithm, **Viterbi** is a kind of **dynamic programming algorithm.**

Note that the Viterbi algorithm is identical to the forward algorithm except that it takes the **max** over the previous path probabilities whereas the forward algorithm takes the **sum**. This is because, we want to obtain **the most probable latent variable sequence**. Note also that the Viterbi algorithm has one component that the forward algorithm doesn't have: **backpointers**. The reason is that while the forward algorithm needs to produce an observation likelihood, the Viterbi algorithm must produce a probability and also the most likely state sequence. We compute this best state sequence by keeping track of the path of hidden states that led to each state and then at the end backtracing the best path to the beginning (the Viterbi backtrace).

We can leverage the forward-backward probabilities:

- $k^* = \text{argmax}_k \, p(z_t^k = 1|X) = \text{argmax}_k \, p(z_t^k = 1, X) = \text{argmax}_k \, \alpha_t^k \beta_t^k$

We will use a forward approach:

$$
\begin{aligned}
V_t^k &= \max_{z_1,\ldots,z_{t-1}} p(x_1, \ldots, x_{t-1}, z_1, \ldots, z_{t-1}, x_t, z_t^k = 1) && (10.8)\\
&= \max_{z_1,\ldots,z_{t-1}} p(x_t, z_t^k = 1|x_1, \ldots, x_{t-1}, z_1, \ldots, z_{t-1}) p(x_1, \ldots, x_{t-1}, z_1, \ldots, z_{t-1}) && (10.9)\\
&= \max_{z_1,\ldots,z_{t-1}} p(x_t, z_t^k = 1|z_{t-1}) p(x_1, \ldots, x_{t-2}, z_1, \ldots, z_{t-2}, x_{t-1}, z_{t-1}) && (10.10)\\
&= \max_{z_{t-1}} p(x_t, z_t^k = 1|z_{t-1}) \max_{z_1,\ldots,z_{t-2}} p(x_1, \ldots, x_{t-2}, z_1, \ldots, z_{t-2}, x_{t-1}, z_{t-1}) && (10.11)\\
&= \max_{i \in z_{t-1}} p(x_t, z_t^k = 1|z_{t-1}^i = 1) V_{t-1}^i && (10.12)\\
&= \max_{i \in z_{t-1}} p(x_t|z_t^k = 1) p(z_t^k = 1|z_{t-1}^i = 1) V_{t-1}^i && (10.13)\\
&= p(x_t|z_t^k = 1) \max_{i \in z_{t-1}} p(z_t^k = 1|z_{t-1}^i = 1) V_{t-1}^i && (10.14)\\
&= b_{k,x_t} \max_{i \in z_{t-1}} a_{i,k} V_{t-1}^i && (10.15)
\end{aligned}
$$

- V_t^k is Viterbi variable which denotes the probability that the HMM is in state k at t after observing the first t observations and $t-1$ latent variables. In another words, this is the probability of most likely sequence of states ending at state $z_t = k$.
- The first line assumes that the observation at time t and the latent variable are fixed and also the fourth line has the recursive structure.
- The third step, only z_{t-1} can affect the z_t, so we can remove all other unnecessary variables.
- The step six can be derived by the HMM structure.
- $i \in z_{t-1}$ simply denotes the index of potential cluster at $t-1$.
- We have already computed the backward and the forward probabilities. So we just need to apply the Viterbi algorithm.

Note that Also note that we present the most probable path by taking the maximum over all possible previous state sequences $\max_{z_1,\ldots,z_{t-1}}$. Like other DP-algorithm, Viterbi fills each cell recursively.

Algorithm 5: Viterbi Algorithm

$V_t^k = viterbi[M, T]$, where M is the number states
for *k=1,...,M* **do**
  $V_1^k \leftarrow \pi_{z_k} b_{k,x_1}$
  $backpointer[k, 1] \leftarrow 0$
for *t=2,...,T* **do**
  for *k=1,...,M* **do**
    $V_t^k \leftarrow b_{k,x_t} \max_{k'} V_t^{k'} a_{k',k}$, where k' is the previous state.
    $backpointer[k, t] \leftarrow b_{k,x_t} \operatorname{argmax}_{k'} V_t^{k'} a_{k',k}$
$bestpathprob \leftarrow \max_k V_T^k$ //termination step
$bestpathpointer \leftarrow \operatorname{argmax}_k V_T^k$ //termination step
$bestpath \leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
Return *bestpathpointer*, *bestpathprob*

Viterbi algorithm typically shows some technical issues:

- Underflow problems $\rightarrow \log V$.

10.6 Learning: Baum-Welch Algorithm

We have to learn HMM parameters with only X. Baum-Welch algorithm or Forward-Backward Algorithm is a standard training algorithm for HMM. The algorithm let us train both the transition and the emission probabilities of the HMM. If we do not have the information about Z, then we can assign the most probable Z given X.

- Given X, estimate parameters π, a, b.
- Then, find the most probable Z given the parameters.

We will use EM algorithm!

10.6.1 EM Algorithm

$$P(X|\theta) = \sum_Z P(X, Z|\theta) \rightarrow \ln P(X|\theta) = \ln \sum_Z P(X, Z|\theta).$$

We cannot directly estimate the log-likelihood function, so we will estimate the expectation of it.

$$\begin{aligned} Q(\theta, \theta^{old}) &= \mathbb{E}_Z \ln P(X, Z|\theta) \\ &= \sum_Z p(Z|X, \theta^{old}) \ln P(X, Z|\theta) \\ &= \sum_Z p(Z|X, \pi^t, a^t, b^t) \ln P(X, Z|\pi, a, b). \end{aligned}$$

Note that $p(X, Z) = \pi_{z_1} \prod_{t=2}^T a_{z_{t-1}, z_t} \prod_{t=2}^T b_{z_t, x_t}$. Thus, $\ln p(X, Z) = \ln \pi_{z_1} + \sum_{t=2}^T \ln a_{z_{t-1}, z_t} + \sum_{t=1}^T \ln b_{z_t, x_t}$. Therefore

$$Q(\theta, \theta^{old}) = \sum_Z p(Z|X, \theta^{old}) \bigg(\ln \pi_{z_1} + \sum_{t=2}^T \ln a_{z_{t-1}, z_t} + \sum_{t=1}^T \ln b_{z_t, x_t} \bigg).$$

To optimize the above function we will use the Lagrange method as follows:

$$\mathcal{L}(\pi, a, b) = Q(\theta, \theta^{old}) - \lambda_\pi \bigg(\sum_{i=1}^K \pi_i - 1 \bigg) - \sum_i^K \lambda_{a_i} \bigg(\sum_{j=1}^K a_{i,j} - 1 \bigg) - \sum_i^K \lambda_{b_i} \bigg(\sum_{j=1}^K b_{i,j} - 1 \bigg).$$

The constraints are for forcing the sum of each probability is equal to 1.

Now, take a partial derivative for each parameter. Let's take a derivative with regard to π_i first. Then,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \pi_i} &= \frac{\partial Q(\theta, \theta^{old})}{\partial \pi_i} - \lambda_\pi \\ &= \frac{\partial}{\partial \pi_i} \sum_Z p(Z|X, \theta^{old}) \ln \pi_{z_1} - \lambda_\pi \\ &= \frac{p(z_1^i = 1|X, \theta^{old})}{\pi_i} - \lambda_\pi \\ \frac{\partial \mathcal{L}}{\partial \lambda_{\pi_i}} &= \sum_{i=1}^K \pi_i - 1 = 0 \rightarrow \sum_{i=1}^K \pi_i = 1. \end{aligned}$$

By setting the derivative is equal to zero,

$$\pi_i = \frac{p(z_1^i = 1|X, \theta^{old})}{\lambda_\pi}.$$

By using the constraint of π, the Lagrange multiplier λ_π must be a normalizer.

$$\pi_i = \frac{p(z_1^i = 1|X, \theta^{old})}{\sum_{j=1}^{K} p(z_1^j = 1|X, \theta^{old})}.$$

Similarly, we can compute other parameters too.

$$a_{i,j}^{t+1} = \frac{\sum_{t=2}^{T} p(z_{t-1}^i = 1, z_t^j = 1|X, \theta^{old})}{\sum_{t=2}^{T} p(z_{t-1}^i = 1|X, \theta^{old})}.$$
$$b_{i,j}^{t+1} = \frac{\sum_{t=1}^{T} p(z_{t1}^i = 1|X, \theta^{old}) I(x_t = j)}{\sum_{t=1}^{T} p(z_t^i = 1|X, \theta^{old})},$$

where $I(x)$ is an indicator function which returns 1 if x is true and 0, otherwise.

10.7 Summary

- Forward-probability: probability of being in state k after observing the first t observations.

$$\alpha_t^k = p(x_1, ..., x_t, z_t^k = 1)$$

- Backward-probability: probability of observations from time $t+1$ to the end, given that we are in state k

$$\beta_t^k = p(x_{t+1}, ..., x_T | z_t^k = 1)$$

- These two sets of probability distributions can then be combined to obtain the distribution over states at any specific point in time given the entire observation sequence

$$\begin{aligned} p(z_t^k = 1, X) &= p(x_1, ..., x_t, z_t^k = 1, x_{t+1}, ..., x_T) \\ &= p(x_1, ..., x_t, z_t^k = 1) p(x_{t+1}, ..., x_T | x_1, ..., x_t, z_t^k = 1) \\ &= p(x_1, ..., x_t, z_t^k = 1) p(x_{t+1}, ..., x_T | z_t^k = 1) \\ &= \alpha_t^k \beta_t^k \end{aligned}$$

  In short, if we know the forward and backward probability, we could know the cluster of state at time t given our observations.

- Forward-algorithm: return a marginal likelihood of the observed sequence
- Forward-backward: predict a single hidden state
- Viterbi: predict an entire sequence of hidden states
- Baum-Welch: unsupervised training (EM)

There are two shortcomings of HMM:

- HMM models capture dependences between each state and only its corresponding observation: Most NLP cases, many tasks needs not only local but also global feature (sentence level).
- Mismatch between learning objective function and prediction objective function: HMM learns a joint distribution of states and observations $p(Y, X)$, but we are more interested in $p(Y|X)$

Chapter 11

Explicit Generative Models

11.1 Variational Autoencoder

Our goal is to find the data distribution $p(X)$. Fig. 11.1 represents a general structure of deep generative model. As you can see, we first sample $z \sim p(z)$ and feed it into a deep neural network $f(z)$ and output x.

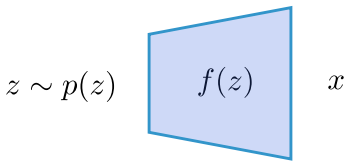


Figure 11.1: General structure of deep generative models. This model does not infer z from x.

VAE performs an inference by introducing a probabilistic encoder, called inference network. VAEs are generative model with a latent variable distributed according to some distribution $p(z_i)$. The observed variable is distributed according to a conditional distribution

$$p_\theta(x_i|z_i)$$

This conditioning means the latent variable values are the one most likely given the observations. We also create a distribution $q_\phi(z_i|x_i)$. We would like to be able to encode our data into the latent variable space. Let's model the distribution.

- $p_\theta(x_i|z_i) \sim \mathcal{N}(x_i|\mu(z_i), \sigma^2(z_i))$: A probabilistic decoder (or generative network, θ)
- $q_\phi(z_i|x_i)$: A probabilistic encoder (or inference network ϕ). We can choose a family of distributions for our conditional distribution q (*e.g.,* standard Gaussian distribution).

  $$q_\phi(z_i|x_i) = \mathcal{N}(z_i|\mu(x_i, W_1), \sigma^2(x_i, W_2)I),$$

  where W_1 and W_2 are network weights and collectively denoted as ϕ. We create a neural network to model the distribution q from our data in a non-linear manner. The outputs of the network are μ and σ.

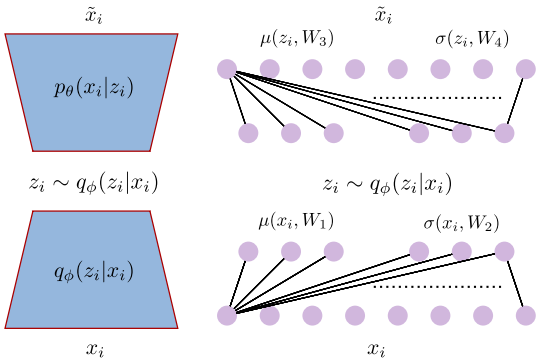


Figure 11.2: Overview of variational autoencoder.

$$
\begin{aligned}
p(X, Z|\theta) &= \prod_{i=1}^{n} \underbrace{p(x_i|z_i, \theta)}_{\text{Likelihood, Generator}} \underbrace{p(z_i|\theta)}_{\text{Prior on latent variable}} \\
&= \prod_{i=1}^{n} \mathcal{N}(x_i| \underbrace{\mu(z_i), \sigma^2(z_i)}_{\text{Non-linear}}) \mathcal{N}(z_i|0, I)
\end{aligned}
$$

Subsequently, marginal distributions can be expressed as follows under i.i.d. assumption:

$$
\begin{aligned}
p(X|\theta) &= \prod_{i=1}^{n} p(x_i|\theta) \\
&= \prod_{i=1}^{n} \int p(x_i, z_j|\theta) dz_j \\
&= \prod_{i=1}^{n} \int p(x_i|z_i, \theta) p(z_i|\theta) dz_i \\
&= \prod_{i=1}^{n} \int \mathcal{N}(x_i|\mu(z_i), \sigma^2(z_i)) \underbrace{\mathcal{N}(z_i|0, I)}_{\text{Mixture weight}} dz_i
\end{aligned}
$$

- As you can see, the marginal distribution $p(X|\theta)$ becomes a mixture of Gaussian (infinite mixture of Gaussian).
- Even though $p(x|z)$ and $p(z)$ are normal, $p(x)$ is not normal, because it is a mixture distribution.
- The non-linearity of Gaussian parameters (modeled by a neural network), conjugacy between the prior and the likelihood does not hold anymore.
- Again, μ and σ is non-linear function of z modeled by some non-linear neural network. The neural network works as a powerful non-linear parameter approximator (based on universal approximation theorem).
- Simple prior is used. Let's consider the data x is an image of 100×100 pixels. Then the covariance matrix has to be 10000×10000. Thus, it is common to set a simple prior such as the standard Gaussian (covariance matrix is diagonal matrix). However, even if we set a simple distribution, with the infinite mixture of Gaussian, we can model any distribution.

- VAE uses a global parametric model to predict the local variational parameters for each data point (**amortized inference**).
- It allows to convert complicated large-dimensional data distributions into simple lower-dimensional latent variable representations.

11.1.1 VAE Optimization

We can train VAE using variational inference with the following objective function, ELBO:

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z))$$

Let's closely look at this objective function:

- In $q_\phi(z|x)$, x is a given data, so it is not stochatic. How to sample z?
- q has to be deterministic and differentiable.

  $\rightarrow$ **Reparameterization trick**!

  $$\tilde{z} \sim q_\phi(z|x) \rightarrow \tilde{z} \sim g_\phi(\epsilon, x)$$

  , where $\epsilon \sim p(\epsilon)$.
- Estimated by using Monte-Carlo estimation

  $$\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \approx \frac{1}{N} \sum_j \log p_\theta(x_i|z_j).$$

11.1.2 Conditional VAE

If we have label information about data, then it would provide a better optimization of VAE model. Recall that the following objective function is the objective of the original VAE:

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z))$$

In conditional VAE,

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(z|x,y)}[\log p_\theta(x|y, z)] - D_{\mathrm{KL}}(q_\phi(z|x, y) \;||\; p_\theta(z|y))$$

$$\log p(X|Y) = \ln \int q(Z|X, Y) \frac{p(X, Z|Y)}{q(Z|X, Y)} dZ \tag{11.1}$$

$$\geq \underbrace{\int q(Z|X, Y) \ln \frac{p(X, Z|Y)}{q(Z|X, Y)} dZ}_{\text{ELBO, } \mathcal{L}(q,\theta)} \quad \text{by Jensen's Inequality.} \tag{11.2}$$

$$\ldots \tag{11.3}$$

$$\ldots \tag{11.4}$$

$$= \mathbb{E}_{q(Z|X,Y)}[\ln p(X|Z, Y)] - KL(q(Z|X, Y)||p(Z|Y)) \tag{11.5}$$

Note that not we have a prior $p_\theta(z|y)$. However, we have no idea about latent variable z, so we simply assume that we cannot impact the z by y. Thus, we typically set it as a standard normal distribution. Also, we can simply concatenate the input X with Y.

11.1.3 Variational Deep Embedding (VaDE)

The generative process of VADE $p(x, z, c) = p(x|z)p(z|c)p(c)$:

- Choose a cluster $c \sim Cat(\pi)$
- Choose a latent vector $z \sim \mathcal{N}(\mu_c, \sigma_c^2 I)$
- Choose a sample x:

$$x \sim \begin{cases} Ber(\mu_x) & \text{If x is binary} \\ \mathcal{N}(\mu_x, \sigma_x^2 I) & \text{else} \end{cases}$$

ELBO of VaDE:

$$\log p(X) = \ln \int \sum_c p(X, Z, C) dz \tag{11.6}$$

$$\geq \underbrace{\int q(Z, C|X) \ln \frac{p(X, Z, C)}{q(Z, C|X)} dZ}_{\text{ELBO}} \tag{11.7}$$

The ELBO can be decomposed as follows:

$$\begin{aligned} \mathcal{L}_{ELBO} &= \mathbb{E}_q(z, c|x) \left[\ln \frac{p(x, z, c)}{q(z, c|x)} \right] \\ &= \mathbb{E}_q(z, c|x)[\ln p(x, z, c) - \ln q(z, c|x)] \\ &= \mathbb{E}_q(z, c|x)[\ln p(x|z) + \ln p(z|c) + \ln p(c) - \ln q(z|x) - \ln q(c|x)] \end{aligned}$$

By using two factorizations:

- $p(x, z, c) = p(x|z)p(z|c)p(c)$
- $q(z, c|x) \approx q(z|x)q(c|x)$ (Mean-field assumption)
  - $q(z|x) \sim \mathcal{N}$: encoder, estimate mean and variance.
  - $q(c|x)$: assignment probability of Gaussian mixture model

11.1.4 Importance Weighted VAE

Chapter 12

Implicit Generative Models

12.1 Generative Adversarial Networks

- Generator's distribution: p_g
- Prior on input noise: $p_z(z)$
- Mapping to data space: $z \to x$ through $G(z; \theta_g)$

  a differentiable multilayer perceptron with parameter θ_g
- $D(x; \theta_d)$: a differentiable multilayer perceptron with parameter θ_d. It outputs a single scalar
- $D(x)$: probability that x (real) came from the data rather than p_g (fake)

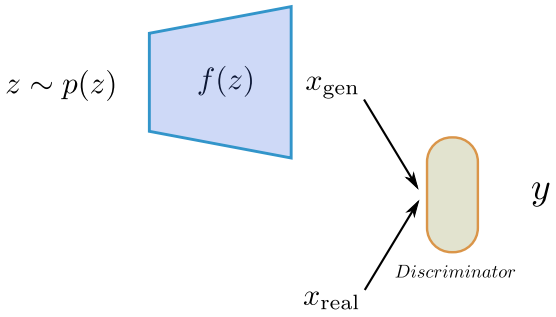


Figure 12.1: GAN structure

12.1.1 Discriminator

The discriminator's goal is to maximize the following equation given G

$$\mathbb{E}_{x \sim p_{data}(x)} \log(D(x)) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z)))$$

The optimal discriminator given G can be denoted as D^*_G. To get the optimal discriminator, define a value function

$$V(G, D) := \mathbb{E}_{x \sim p_{data}(x)} \log(D(x)) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z))).$$

Then, $D_G^* = \text{argmax}_D V(G, D)$

However, the generator G wants to minimize the value function given $D = D_G^*$.

$$G^* = \text{argmin}_G V(G, D_G^*).$$

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- $\min_G \rightarrow$ try to generate fake data that is similar to real data
- $\max_D \rightarrow$ try to assign correct label [1]

At this point, we must show that this optimization problem has a unique solution G^* and that this solution satisfies $p_G = p_{data}$.

One big idea from the GAN paper–, which is different from other approaches is that G **need not be invertible**. Many pieces of notes online miss this fact when they try to replicate the proof and incorrectly use the change of variables formula from calculus (which would depend on G being invertible). Rather, the whole proof relies on this equality:

$$\mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z))) = \mathbb{E}_{x \sim p_G(x)} \log(1 - D(x)).$$

With the above equality,

$$\begin{aligned}
&\mathbb{E}_{x \sim p_{data}(x)} \log(D(x)) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z))) \\
&= \int_x p_{data}(x) \log D(x) \, \mathrm{d}x + \int_z p(z) \log(1 - D(G(z))) \, \mathrm{d}z \\
&= \int_x p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)) \, \mathrm{d}x
\end{aligned}$$

Additionally, we will use the following property:

$$f(y) = a \log y + b \log(1 - y).$$

To find a critical point,

$$f'(y) = 0 \Rightarrow \frac{a}{y} - \frac{b}{1 - y} = 0 \Rightarrow y = \frac{a}{a + b}$$

If $a + b \neq 0$, do the second derivative test:

$$f''\big(\frac{a}{a+b}\big) = -\frac{a}{(\frac{a}{a+b})^2} - \frac{b}{(1 - \frac{a}{a+b})^2} < 0$$

If $a, b \in (0, 1)$, $\frac{a}{a+b}$ is a maximum.

By rewriting the equation,

$$\begin{aligned}
V(G, D) &= \int_x p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)) \, \mathrm{d}x \\
&\leq \int_x \max_y p_{data}(x) \log y + p_G(x) \log(1 - y) \, \mathrm{d}x
\end{aligned}$$

Thus, if $D(x) = \frac{p_{data}}{p_{data} + p_G}$, then we can achieve the maximum $V(G, D)$.

[1] The above equation is trained separately at the same time, don't get confused

12.1.2 Generator

If we achieve the optimal G (i.e., $p_G = p_{data}$), then D would be completely confused and $D^*_G(x) = \frac{p_{data}}{p_{data}+p_G} = \frac{1}{2}$ (it means that D cannot make a clear decision.).

The global minimum of the virtual training criterion $C(G) = \max_D V(G, D)$ is acheived if and only if $p_G = p_{data}$. Let's plug $D^*_G(x)$ into the criterion then,

$$C(G) = \int_x p_{data}(x) \log \big(\frac{p_{data}(x)}{p_G(x) + p_{data}(x)}\big) + p_G(x) \log \big(\frac{p_G(x)}{p_G(x) + p_{data}(x)}\big)\, \mathrm{d}x.$$

To get the minimum $C(G)$, we can use the Jansen-Shannon divergence:

$$\begin{aligned}
D_{JS}(p_{data}||p_G) &= \frac{1}{2}\left[D_{KL}\Big(p_{data}\Big|\Big|\frac{p_{data}+p_G}{2}\Big) + D_{KL}\Big(p_G\Big|\Big|\frac{p_{data}+p_G}{2}\Big)\right] \\
&= \frac{1}{2}\left[\left(\int_x p_{data}(x)\log\left(\frac{2p_{data}(x)}{p_{data}(x)+p_G(x)}\right)dx\right) + \left(\int_x p_G(x)\log\left(\frac{2p_G(x)}{p_{data}(x)+p_G(x)}\right)dx\right)\right] \\
&= \frac{1}{2}\left[\left(\int_x p_{data}(x)\log 2 + p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_G(x)}\right)dx\right) + \right. \\
&\qquad \left.\left(\int_x p_G(x)\log 2 + p_G(x)\log\left(\frac{p_G(x)}{p_{data}(x)+p_G(x)}\right)dx\right)\right] \\
&= \frac{1}{2}\left[\left(\log 2 + \int_x p_{data}(x)\log\left(\frac{2p_{data}(x)}{p_{data}(x)+p_G(x)}\right)dx\right) + \right. \\
&\qquad \left.\left(\log 2 + \int_x p_G(x)\log\left(\frac{2p_G(x)}{p_{data}(x)+p_G(x)}\right)dx\right)\right] \\
&= \frac{1}{2}(\log 4 + C(G))
\end{aligned}$$

Thus,

$$C(G) = -\log 4 + 2D_{JS}(p_{data}||p_G)$$

Since the Jensen-Shannon divergence between two distributions is always non-negative and zero only when they are equal, we have shown that $C^* = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_G = p_{data}$, i.e., the generative model perfectly replicating the data generating process.

12.2 Some notes

What would be the optimal discriminator that separates the two different distributions $p(x)$ and $q(x)$? It turns out that it is

$$f(x) = \frac{q(x)}{p(x) + q(x)}$$

Actually, there are many choices for classifiers e.g., KL-divergence

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 12.2: Training GAN

1. What do we need to learn a classifier?
   - Only samples from $p(x)$ and $q(x)$
2. How do we parameterize $q(x)$?
   - Parametric density function (Gaussian)
   - Define implicitly (GANs approach): define mapping from one (noise) to another (data or image)

The orignal GAN does not learn the data distributions.

12.3 Wasserstein Generative Adversarial Networks

12.3.1 KL Divergence

Definition:

$$D_{\mathrm{KL}}(q(x)||p(x)) = \int q(x) \log \frac{q(x)}{p(x)} dx$$

- Forward KL:
  - If $q(z) \to 0$, Forward KL $\to \infty$
  - Zero avoiding for $q(z)$
- Reverse KL:
  - If $p(z) \to 0$, Reverse KL $\to \infty$
  - Zero forcing: $q(z) \to 0$

Typically, $p(x)$ and $q(x)$ are far apart at the initial state.

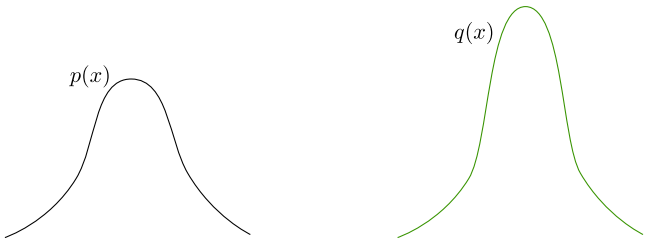


Figure 12.3: Two distributions: $p(x)$ and $q(x)$

Thus, both the forward KL and the reverse KL suffers an unstability issue. Specifically, in each case, if the denominator goes to zero, then the divergence goes to infinity.

12.3.2 Jensen-Shannon Divergence

Definition:

$$D_{JS}(p_{data}||p_G) = \frac{1}{2}\left[D_{KL}\left(p_{data}\middle|\middle|\frac{p_{data}+p_G}{2}\right) + D_{KL}\left(p_G\middle|\middle|\frac{p_{data}+p_G}{2}\right)\right]$$

The KL divergence's issue can be alleviated by JS-divergence. Consider a simple example in Fig. 12.46

$$\forall (x,y) \in P, x = 0 \text{ and } y \sim U(0,1)$$
$$\forall (x,y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0,1)$$

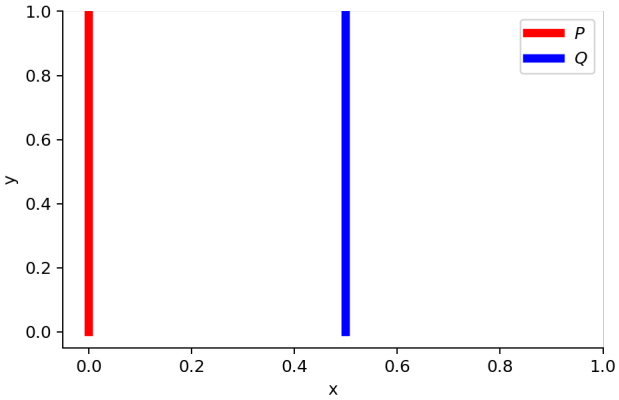


Figure 12.4: Two distributions: $p(x)$ and $q(x)$

$$
\begin{aligned}
D_{\mathrm{KL}}(q(x)||p(x)) &= \infty \\
D_{\mathrm{KL}}(p(x)||q(x)) &= \infty \\
D_{JS}(p_{data}||p_G) &= \frac{1}{2}\left[D_{KL}\left(p_{data}\middle|\middle|\frac{p_{data}+p_G}{2}\right) + D_{KL}\left(p_G\middle|\middle|\frac{p_{data}+p_G}{2}\right)\right] \\
&= \frac{1}{2}\left[D_{KL}\left(p_{data}\middle|\middle|\frac{p_{data}}{2}\right) + +D_{KL}\left(p_G\middle|\middle|\frac{p_G}{2}\right)\right] \\
&= \frac{1}{2}[\log 2 + \log 2] = \log 2 \\
W(p,q) &= |\theta|
\end{aligned}
$$

Therefore, Jensen-Shannon divergence is more stabler than KL divergece. This is one of the reasons why GAN, which uses JS divergence works better than VAE, which uses KL divergence.

However, JS divergence also has some problem. If the value is close to $\frac{1}{2}\log 2$, then the gradient will be very small or close to zero, because the divergence is close to constant. It means that a training speed is very slow. Thus, we need a better metric.

12.3.3 Wasserstein Distance

Wasserstein Distance is a measure of the distance between two probability distributions. It is also called Earth Mover's distance, short for EM distance, because informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution.

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|]$$

- Π: is the transportation plan and the set of all possible joint probability distributions between p_r and p_g. One joint distribution $\gamma \sim \Pi(p_r, p_g)$ describes one transport plan.
- $\mathbb{E}_{x,y\sim\gamma}\|x-y\| = \sum_{x,y}\gamma(x,y)\|x-y\|$

- Finally, we take the minimum one among the costs of all dirt moving solutions as the EM distance (by infimum).

12.4 WGAN

However, consider all possible joint distribution is intractable, so dual solution can be used.

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

So to calculate the Wasserstein distance, we just need to find a 1-Lipschitz function. To enforce the constraint, WGAN applies a very simple clipping to restrict the maximum weight value in f, i.e. the weights of the discriminator

Suppose this function f comes from a family of K-Lipschitz continuous functions, $\{f_w\}_{w \in W}$, parameterized by w. In the modified Wasserstein-GAN, the "discriminator" model is used to learn w to find a good f_w and the loss function is configured as measuring the Wasserstein distance between p_r and p_g.

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

There are two ways to satisfy the Lipschitz continuity:

- Weight clipping
- Gradient Penalty

12.4.1 Lipschitz continuity

The function f in the new form of Wasserstein metric is demanded to satisfy $\|f\|_L \leq K$, meaning it should be K-Lipschitz continuous. Lil2017

A real-valued function $f : \mathbb{R} \to \mathbb{R}$ is called K-Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in \mathbb{R}$

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

Here K is known as a Lipschitz constant for function $f(\cdot)$. Functions that are everywhere continuously differentiable is Lipschitz continuous, because the derivative, estimated as $\frac{|f(x_1)-f(x_2)|}{|x_1-x_2|}$, has bounds. However, a Lipschitz continuous function may not be everywhere differentiable, such as $f(x) = |x|$

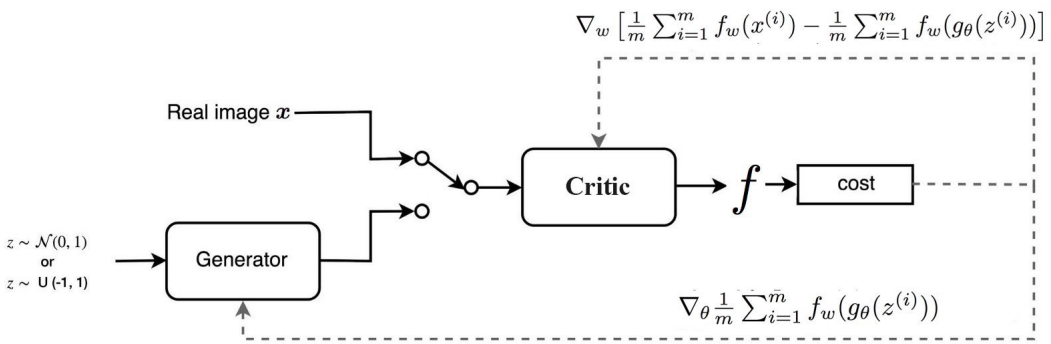


Figure 12.5: WGAN

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^m \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$	$\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^m \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m}\sum_{i=1}^m \left[f(x^{(i)}) - f(G\left(z^{(i)}\right))\right]$	$\nabla_\theta \frac{1}{m}\sum_{i=1}^m f(G\left(z^{(i)}\right))$

Figure 12.6: WGAN

12.5 InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets

12.5.1 Joint Entropy

$$H(X,Y) = \mathbb{E}_{X,Y}[-\log p(x,y)] = -\sum_{x,y} p(x,y)\log p(x,y)$$

12.5.2 Conditional Entropy

$$\begin{aligned}
H(X|Y) &= \mathbb{E}_Y[H(X,Y)] \\
&= -\sum_{y\sim p_Y(y)} p(y) \sum_{x\sim p_X(x)} p(x|y)\log p(x|y) \\
&= -\sum_{y\sim p_Y(y)} \sum_{x\sim p_X(x)} p(y)p(x|y)\log p(x|y) \\
&= -\sum_{y\sim p_Y(y)} \sum_{x\sim p_X(x)} p(x,y)\log p(x|y) = -\mathbb{E}_{x,y}[\log p(x|y)] \\
&= -\sum_{y\sim p_Y(y)} \sum_{x\sim p_X(x)} p(x,y)\log \frac{p(x,y)}{p(y)} \\
&= -\sum_{y\sim p_Y(y)} \sum_{x\sim p_X(x)} p(x,y)\log p(x,y) + \sum_{y\sim p_Y(y)} \sum_{x\sim p_X(x)} p(x,y)\log p(y) \\
&= H(X,Y) - H(Y)
\end{aligned}$$

12.5.3 Variational Mutual Information Maximization

$$\begin{aligned}
I(c;G(z,c)) &= H(c) - H(c|G(z,c)) \\
&= H(c) + \int\int p(c=c', x=G(z,c))\log p(c=c'|x=G(z,c))dc'dz \\
&= H(c) + \mathbb{E}_{x\sim G(z,c),c'\sim p(c|x)}[\log p(c'|x)] \\
&= H(c) + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}[\log p(c'|x)] \\
&= H(c) + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\left[\log \frac{p(c'|x)Q(c'|x)}{Q(c'|x)}\right] \\
&= H(c) + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\left[\log \frac{p(c'|x)}{Q(c'|x)}\right] + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\Big[\log Q(c'|x)\Big] \\
&= H(c) + \mathbb{E}_{x\sim G(z,c)}\Big[D_{KL}(p(c'|x)||Q(c'|x))\Big] + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\Big[\log Q(c'|x)\Big] \\
&\geq H(c) + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\Big[\log Q(c'|x)\Big]^2
\end{aligned}$$

Thus we get a lower bound for the mutual information as follows:

$$I(c; G(z,c)) \geq H(c) + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\Big[\log Q(c'|x)\Big]$$

However, we still have a problem. We need to sample c from $p(c|x)$. Thus, we need to replace it with a known distribution. Firstly, with the reasoning that $x \sim G(z,c)$ means sample c from $p(c)$ then sample x from $G(z,c)$. So we can express $\mathbb{E}_{x\sim G(z,c)}$ with $\mathbb{E}_{c\sim p(c)}\mathbb{E}_{x\sim G(z,c)}$. and by the Lemma 12.5.3,

$$\begin{aligned}
I(c; G(z,c)) &\geq H(c) + \mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\Big[\log Q(c'|x)\Big] \\
&= H(c) + \mathbb{E}_{c\sim p(c)}\mathbb{E}_{x\sim G(z,c)}\mathbb{E}_{c'\sim p(c|x)}\Big[\log Q(c'|x)\Big] \\
&= H(c) + \mathbb{E}_{c\sim p(c)}\mathbb{E}_{x\sim G(z,c)}\Big[\log Q(c'|x)\Big]^3
\end{aligned}$$

Thus, we can directly sample c from the known distribution instead of $p(c|x)$.

For random variables X, Y and function $f(x,y)$ under suitable regularity conditions:

$$\mathbb{E}_{x\sim X, y\sim Y|x}[f(x,y)] = \mathbb{E}_{x\sim X, y\sim Y|x, x'\sim X|y}[f(x',y)]$$

$$\begin{aligned}
\mathbb{E}_{x\sim X, y\sim Y|x}[f(x,y)] &= \int_x P(x)\int_y P(y|x)f(x,y)dydx \\
&= \int_x\int_y P(x,y)f(x,y)dydx \\
&= \int_{x'}\int_y P(x',y)f(x',y)dydx' \\
&= \int_{x'}\int_y P(y)P(x'|y)f(x',y)dydx' \\
&= \int_{x'}\int_y\int_x P(x,y)P(x'|y)f(x',y)dxdydx' \\
&= \int_x P(x)\int_y P(x|y)\int_{x'} P(x'|y)f(x',y)dxdydx' \\
&= \mathbb{E}_{x\sim X, y\sim Y|x, x'\sim X|y}[f(x',y)]
\end{aligned}$$

Chapter 13

Diffusion Model

13.1 Introduction

(Denoising) Diffusion models have emerged as the new SOTA family of deep generative models.

- First proposed in 2015.
- Outperform GANs on image synthesis (DDPM) $\sim$ 2020.
- Stable training dynamics.
- Image synthesis, super resolution, text-to-image, and so on.

The overall idea is to construct a Markov chain of progressively less noisy samples. Each transition denoises a noisy sample. Diffusion models consist of two Markov chains:

1. Forward: A Markov chain of diffusion steps to **slowly add random noise** to data.

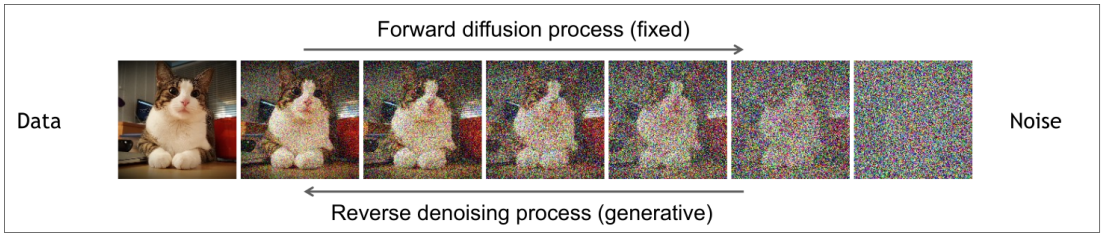
$$\mathbf{x}_0 \to \mathbf{x}_1 \cdots \to \mathbf{x}_T$$

2. Backward (Reverse): Learn to **reverse the diffusion process** to construct desired data samples from the noise.

$$\mathbf{x}_T \to \mathbf{x}_{T-1} \cdots \to \mathbf{x}_0$$

Some properties of diffusion models:

1. Diffusion model has a pre-defined sampling equation.
   - The equation relies on a random noise.
   - Noise is all we need $\to$ Predict noise at a time step t.
2. Fit a model via forward and backward processes.
3. **Iterative transform** of one distribution into another via **Makov Chain**.
   - $\mathcal{D}_{data} \to \mathcal{N}$.



- $\mathcal{N} \to \mathcal{D}_{data}$.
- Diffusion model≈Generative Markov Chain.

4. Learn a transition model:

$$p_\theta(\mathbf{x}_{t-1}|x_t) = \mathcal{N}(\mathbf{x}_{t-1}|\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)).$$

5. Base case: $p(\mathbf{x}_T) = \mathcal{N}(0, I)$
6. Marginal distribution over $\mathbf{x}_0$:

$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_T) d\mathbf{x}_1, \ldots, \mathbf{x}_T$$

7. We want to learn the parameters so that

$$p(\mathbf{x}_0) \approx p_\theta(\mathbf{x}_0)$$

13.2 Forward Diffusion

- We want to model a forward trajectory (by the Markov property):

$$q(\mathbf{x}_{0:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} \underbrace{q(\mathbf{x}_t|\mathbf{x}_{t-1})}_{\text{Transition kernel}}$$

- Slow transform with a large T: $\mathbf{x}_0 \to \mathbf{x}_1 \cdots \to \mathbf{x}_T$
  - Imagine someone said he is from Germany.
  - We can't exactly track his journey without more information.
  - We need to add more steps!
- How to model $q(\mathbf{x}_t|\mathbf{x}_{t-1})$?

Forward Diffusion: $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ In a continuous case (*e.g.,* image), each transition can be parameterized as follows:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{13.1}$$

- $\beta_t \in (0, 1)$ is a variance at time t.
- $\sqrt{1-\beta_t}$ downscales $\mathbf{x}_{t-1}$ to be 0, $\beta_1 < \cdots < \beta_t$. Thus, $\mathbf{x}_t$ will become more noisier. $\mathbf{x}_t$ can be sampled as:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t} \odot \epsilon$$

1. Sample $\mathbf{x}_t \sim q(\mathbf{x}_t)$ and scale it by $\sqrt{1-\beta_t}$
2. Adds noise $\epsilon \sim \mathcal{N}(0, I)$ with variance β_t.

- The above process is autoregressive (*i.e.,* ancestral sampling), but we can sample $\mathbf{x}_t$ directly from $q(\mathbf{x}_t|\mathbf{x}_0)$ in an analytic form:

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t} \odot \epsilon_{t-1} && (13.2)\\
&= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t} \odot \epsilon_{t-1} && (13.3)\\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}} \odot \epsilon_{t-2}\right) + \sqrt{1-\alpha_t} \odot \epsilon_{t-1} && (13.4)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1}} \odot \epsilon_{t-2} + \sqrt{1-\alpha_t} \odot \epsilon_{t-1} && (13.5)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1} + 1 - \alpha_t} \odot \epsilon_{t-2} && (13.6)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1 - \alpha_t\alpha_{t-1}} \odot \epsilon_{t-2} && (13.7)\\
&= \dots && (13.8)\\
&= \sqrt{\prod_t \alpha_t}\mathbf{x}_0 + \sqrt{1 - \prod_t \alpha_t} \odot \epsilon_{t_0} && (13.9)\\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \odot \epsilon && (13.10)\\
&\sim \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}), && (13.11)
\end{aligned}
$$

  where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$. Thus, $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \odot \epsilon$. Note that the fifth step is done by using the property of sum of two Gaussian distributions (*e.g.,* $\mathcal{N}(0, \sigma_1^2 I) + \mathcal{N}(0, \sigma_2^2 I) = \mathcal{N}(0, (\sigma_1^2 + \sigma_2^2) I)$).

We can get some intuitions:

- The original input $\mathbf{x}_0$ **gradually loses all info** during the forward diffusion process.
- This Markov chain has a **stationary distribution**: As $t \to \infty$, $q(\mathbf{x}_t) \approx \mathcal{N}(0, I)$.
  - In practice, T is a very high number *e.g.,* 1,000.
  - Minimize info loss for each step.
  - Allow a smooth training.

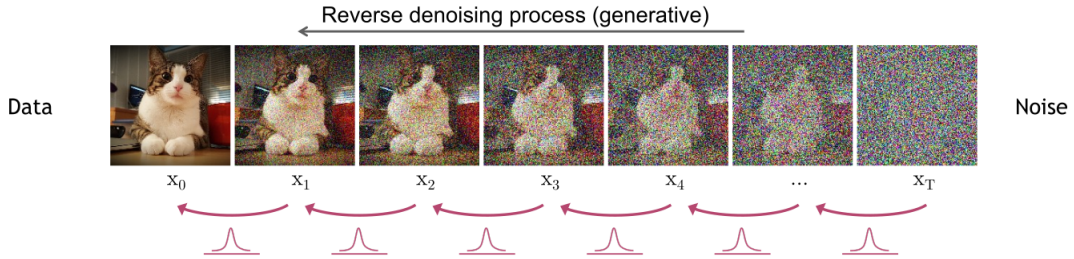
13.3 Backward Process

Generative Learning by Denoising:

- Now we know how to model the forward process (diffusion process).
- However, a noisy image is not what we want.
- We want to generate a new image with high quality.

How to generate data? If we can reverse the forward process, then we can draw a true sample. We call it **Backward process**!

- Start from $q(\mathbf{x}_T) \approx \mathcal{N}(0, I)$.
- Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T|0, I)$
  - Sample a noise vector from a prior distribution.
- Iteratively sample $\mathbf{x}_{t-1} \sim q(\mathbf{x}_{t-1}|\mathbf{x}_t)$.
  - $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$: **true denoising distribution** (we don't know).
- In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is intractable.
- We can **approximate** $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as **Normal distribution** if β_t is small in each forward diffusion step.
  - *e.g.*, $\mathbf{x}_3$ to $\mathbf{x}_2$



CVPR 2022 Tutorial: Denoising Diffusion-based Generative Modeling: Foundations and Applications

Backward Process

- Approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ using a neural network, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$.
- Backward process: $p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$.
  - $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; 0, I)$.
  - $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$.
  - We can model the denoising distribution as Normal distribution like above (*c.f.*, Eq. (13.45)).
  - Note that the reverse conditional probability $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is tractable when it is conditioned on $\mathbf{x}_0$ as shown in Eq. (13.45). This allows us to train a neural network to model this denoising distribution.
- Key to the success of this sampling process is training the reverse Markov chain to match the actual time reversal of the forward Markov chain.
- After optimizing the backward process, the sampling procedure is that just sample Gaussian noise from $p(\mathbf{x}_T)$ and then iteratively running the denoising transitions (backward process) for T steps to generate a novel $\mathbf{x}_0$.

13.4 Distribution Modeling

What we want to learn (or model) is $p_\theta(\mathbf{x}_0) \approx p(\mathbf{x}_0)$ (approximate data distribution).

- $p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$
- It is intractable to compute all trajectories.

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\mathbf{x}_0 \sim p}[\log p_\theta(\mathbf{x}_0)] = \mathbb{E}_{\mathbf{x}_0 \sim p}\left[\log \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}\right].$$

- Thus, we will use variational lower bound with KL-Div:

$$\log p_\theta(\mathbf{x}_0) = \log \int p(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \tag{13.12}$$

$$= \log \int p(\mathbf{x}_{0:T}) \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{13.13}$$

$$= \log \int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{13.14}$$

$$\geq \int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{13.15}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \quad \rightarrow \text{ELBO} \tag{13.16}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.17}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) p(\mathbf{x}_0|\mathbf{x}_1) \prod_{t=1}^{T-1} p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_T|\mathbf{x}_{T-1}) \prod_{t=1}^{T-1} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.18}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \prod_{t=1}^{T-1} \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.19}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} [\log p(\mathbf{x}_0|\mathbf{x}_1)] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \tag{13.20}$$

$$+ \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\sum_{t=1}^{T-1} \log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.21}$$

$$= \cdots + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.22}$$

$$= \cdots + \int_{x_1} \cdots \int_{x_T} q(\mathbf{x}_1, \ldots, \mathbf{x}_T) \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] dx_1 \ldots dx_T + \ldots \tag{13.23}$$

$$= \cdots + \int_{x_T} \int_{x_{T-1}} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \cdots \int_{x_1} q(\mathbf{x}_1, \ldots, \mathbf{x}_T) dx_1 \ldots dx_T + \ldots \tag{13.24}$$

$$= \cdots + \int_{x_T} \int_{x_{T-1}} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] q(\mathbf{x}_t, \mathbf{x}_{t-1}) dx_{T-1} dx_T + \ldots \tag{13.25}$$

$$= \cdots + \mathbb{E}_{q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.26}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} [\log p(\mathbf{x}_0|\mathbf{x}_1)] + \mathbb{E}_{q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \tag{13.27}$$

$$+ \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{13.28}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} [\log p(\mathbf{x}_0|\mathbf{x}_1)] - \mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} [D_{KL}(q(\mathbf{x}_T|\mathbf{x}_{T-1}) \| p(\mathbf{x}_T))] \tag{13.29}$$

$$- \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_{t+1}|\mathbf{x}_0)} D_{KL}[q(\mathbf{x}_t|\mathbf{x}_{t-1}) \| p(\mathbf{x}_t|\mathbf{x}_{t+1})] \tag{13.30}$$

$$\because q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0) = \frac{q(\mathbf{x}_t, \mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_0)} = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) q(\mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_0)} = q(\mathbf{x}_t|\mathbf{x}_{t-1}) q(\mathbf{x}_{t-1}|\mathbf{x}_0) \tag{13.31}$$

- The sixth step is done by Markov property.
- The first term is a *reconstruction term.*
- The second term is a *prior matching term.* This term requires no optimization, as it has no trainable parameters; furthermore, as we have assumed a large enough T such that the final distribution is Gaussian, this term effectively becomes zero.
- The last term is a *consistency term.* It endeavors to make the distribution at xt consistent, from both forward and backward processes. That is, a denoising step from a noisier image should match the corresponding noising step from a cleaner image, for every intermediate timestep. This term is minimized when we train $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})$ to match the Gaussian distribution $q(\mathbf{x}_t|\mathbf{x}_{t-1})$, which is defined in Eq. (13.1).
- Under this derivation, all terms of the ELBO are computed as expectations, and can therefore be approximated using Monte Carlo estimates.
- However, actually optimizing the ELBO using the terms we just derived might be suboptimal, because the consistency term is computed as an expectation over two random variables $\{x_{t-1}, x_{t+1}\}$ for every time step, the variance of its Monte Carlo estimate could potentially be higher than a term that is estimated using only one random variable per time step. As it is computed by summing up $T-1$ consistency terms, the final estimated value of the ELBO may have high variance for large T values.

Let us instead try to derive a form for our ELBO where each term is computed as an expectation over **only one random variable at a time**. The key insight is that we can rewrite encoder transitions as $q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$, where the extra conditioning term is superfluous due to the Markov property. Then, according to Bayes rule, we can rewrite each transition as:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}$$

Armed with this equation, we can factorize the ELBO again as follows:

$$L = \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \quad \rightarrow \text{ELBO} \tag{13.32}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \tag{13.33}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)\prod_{t=2}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_1|\mathbf{x}_0)\prod_{t=2}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \tag{13.34}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)\prod_{t=2}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_1|\mathbf{x}_0)\prod_{t=2}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}\right] \quad \text{by Markov Property} \tag{13.35}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}\right] \tag{13.36}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}}\right] \tag{13.37}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{q(\mathbf{x}_T|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \tag{13.38}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_T|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \tag{13.39}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p(\mathbf{x}_0|\mathbf{x}_1)] + \mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \tag{13.40}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p(\mathbf{x}_0|\mathbf{x}_1)] - D_{KL(}q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T)) - \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p(\mathbf{x}_{t-1}|\mathbf{x}_t))] \tag{13.41}$$

Let's closely look at the last three terms:

- $\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p(\mathbf{x}_0|\mathbf{x}_1)]$: reconstruction term.
- $D_{KL(}q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))$: Prior matching term.
  - No trainable parameters
- $\sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p(\mathbf{x}_{t-1}|\mathbf{x}_t))]$: Denoising matching term.
  - $p_\theta(\mathbf{x}_{t1}|\mathbf{x}_t)$ as an approximation to tractable, ground-truth denoising transition step $q(\mathbf{x}_{t1}|\mathbf{x}_t, \mathbf{x}_0)$. The $q(\mathbf{x}_{t1}|\mathbf{x}_t, \mathbf{x}_0)$ transition step can act as a ground-truth signal, since it defines how to denoise a noisy image with access to what the final, completely denoised image $\mathbf{x}_0$ should be. This term is therefore minimized when the two denoising steps match as closely as possible, as measured by their KL Divergence.

In the last term, $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ can be further factorized as follows:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}.$$

Then, $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1})$ by Markov property. Now, let's compute the KL-divergence in the denoising matching term.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \tag{13.42}$$

$$= \frac{\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_{t-1}, (1-\bar{\alpha}_t)\mathbf{I})\mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I})}{\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})} \tag{13.43}$$

$$\vdots \tag{13.44}$$

$$\propto \mathcal{N}\Bigg(\mathbf{x}_{t-1}; \underbrace{\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t}}_{\mu_q(\mathbf{x}_t, \mathbf{x}_0)}, \underbrace{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{I}}_{\Sigma_q(t)}\Bigg) \tag{13.45}$$

We have therefore shown that at each step, $\mathbf{x}_{t-1} \sim q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is normally distributed, with mean $\mu_q(\mathbf{x}_t, \mathbf{x}_0)$ that is a function of $\mathbf{x}_t$ and $\mathbf{x}_0$, and variance $\Sigma_q(t)$ as a function of α coefficients.

We can use the Eq. (13.45) to compute the optimal θ by solving $\min_\theta D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$, but we can simplify the optimization problem by using Eq. (13.11):

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \odot \epsilon$$

$$\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t} \odot \epsilon}{\sqrt{\bar{\alpha}_t}}$$

By plugging the above equation into $\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$, we can exclude $\mathbf{x}_0$ term as follows:

$$\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t} \tag{13.46}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\odot\epsilon}{\sqrt{\bar{\alpha}_t}}}{1-\bar{\alpha}_t} \tag{13.47}$$

$$\vdots \tag{13.48}$$

$$= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}_0 \tag{13.49}$$

Thus, we can force $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$, which has no dependency on $\mathbf{x}_0$ term, to match the $\boldsymbol{\mu}_q$. Since the $\mathbf{x}_t$ is given at training time, we just need to predict the $\boldsymbol{\epsilon}_t$. Then, we can express $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ as follows:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$$

Thus, $\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ can be expressed as follows:

$$\mathcal{N}\left(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\right)$$

Note that we can use the variance derived in Eq. (13.45) instead of estimating it from a network for simplicity. Finally, we can solve the KL-divergence term:

$$\mathcal{L} = \min_{\theta} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \tag{13.50}$$

$$\vdots \tag{13.51}$$

$$= \min_{\theta} \frac{1}{2\sigma_q^2(t)} \frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)\alpha_t} \left[\|\boldsymbol{\epsilon}_0 - \hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t)\|_2^2\right]. \tag{13.52}$$

Here $\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t)$ is a is a neural network that learns to predict the source noise $\boldsymbol{\epsilon}_0 \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$ that determines $\mathbf{x}_t$ from $\mathbf{x}_0$. We have therefore shown that the overall learning objective is equivalent to learning to predict the noise.

13.5 Summary

- The loss function can be decomposed.

$$L_{\text{VLB}} = L_T + L_{T-1} + \cdots + L_0.$$

- $L_T = D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))$
  - Constant ≈ 0 since x_T is a Gaussian noise.
- $L_t = D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$ for $t > 1$
  - This is the main part.
- $L_0 = -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)$
  - Can be modeled by a separate decoder.
- $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)I)$
- $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$,

  We can sample by $\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon$
- $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$.
  - We need to learn mean and variance.
  - DDPM kept the variance fixed and let the neural network only learn the mean μ_θ.
  - $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) = \sigma_t^2\mathbf{I}$ and set $\sigma_t^2 = \beta_t$.
  - Improved DDPM model trains σ also.
- One can reparameterize the mean to make the nerual network learn the added noise via a network ϵ_θ.

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\underbrace{\epsilon_\theta(\mathbf{x}_t, t)}_{\text{Network}}\right)$$

- Final objective function L_t is

$$||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2 = ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon, t)||^2$$

  - $t \sim \text{Unif}[\{1, .., T\}]$
  - $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon \sim q(\mathbf{x}_t|\mathbf{x}_0)$
  - $\epsilon \sim \mathcal{N}(0, I)$
- $\mathbf{x}_t$ is perturbed by ϵ and the noise prediction network ϵ_θ predicts ϵ.

Algorithm 6: Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \text{Unif}[\{1, \cdots, T\}]$
  $\epsilon \sim \mathcal{N}(0, I)$
  Take gradient descent step on $\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon, t)||^2$
until *converged*;

The training process is given by

1. $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
2. Sample a noise level t between 1 and T (*i.e.,* random time step).
3. Sample a noise from a Gaussian distribution and perturb the input by the sampling equation.
4. NN is trained to predict this noise ϵ used for generating $\mathbf{x}_t$.
5. β is often scheduled linearly.
6. Σ is set equal to β.

The sampling process is given by

Algorithm 7: Sampling

$\mathbf{x}_T \sim \mathcal{N}(0, I)$
for $t = T, \cdots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(0, I)$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \Sigma_t \mathbf{z}$
return $\mathbf{x}_0$

- Ancestral sampling.
- T is typically around 1,000

13.6 Score Matching

- Suppose $\{\mathbf{x}_0, \cdots, \mathbf{x}_N\}$, where each data point (*e.g.,* image, video, or text)) is sampled independently from a data distribution $p(\mathbf{x})$.
- Given the dataset, the goal of generative modeling is to fit a model to the data distribution such that we can synthesize new data points at will by sampling from the model.
- One way is to directly model the distribution function as in likelihood-based models. Let $f_\theta(\mathbf{x}) \in \mathbb{R}^d$, then we can define a density function:

$$p_\theta(\mathbf{x}) = \frac{\exp^{-f_\theta(\mathbf{x})}}{Z_\theta}$$

.

- $f_\theta(\mathbf{x}) \in \mathbb{R}^d$ is often called unnormalized probabilistic model or energy-based model.
- Energy-based model originates from the Gibbs distribution in statistical physics.
- $p_\theta(\mathbf{x})$ can be trained by maximizing the log-likelihood of the data.

$$\max_\theta \sum_i^N \log p_\theta(\mathbf{x}_i).$$

- The gradient of the loglikelihood is given by

$$\begin{aligned}
\nabla_\theta \log p_\theta(\mathbf{x}) &= \nabla_\theta f_\theta(\mathbf{x}) - \nabla_\theta Z_\theta \\
\nabla_\theta Z_\theta &= \frac{\nabla_\theta Z_\theta}{Z_\theta} \\
&= \frac{1}{Z_\theta} \nabla_\theta \int \exp(f_\theta(\mathbf{x})) d\mathbf{x} \\
&= \frac{1}{Z_\theta} \int \exp(f_\theta(\mathbf{x})) \nabla_\theta f_\theta(\mathbf{x}) d\mathbf{x} \\
&= \int \frac{1}{Z_\theta} \exp(f_\theta(\mathbf{x})) \nabla_\theta f_\theta(\mathbf{x}) d\mathbf{x} \\
&= \int p_\theta(\mathbf{x}) \nabla_\theta f_\theta(\mathbf{x}) d\mathbf{x} \\
&= \mathbb{E}_{p_\theta(\mathbf{x})}[\nabla_\theta f_\theta(\mathbf{x})] \\
\nabla_\theta \log p_\theta(\mathbf{x}) &= \nabla_\theta f_\theta(\mathbf{x}) - \mathbb{E}_{p_\theta(\mathbf{x})}[\nabla_\theta f_\theta(\mathbf{x})]
\end{aligned}$$

- However, it is undesirable, since Z_θ is intractable.
  - For instance, a gray scale image of 100×100 has $256^{10,000}$ space.
- Thus, we have to sidestep the issue by using some solutions, for instance:
  - Approximate by using VAE or MCMC

Instead, we can leverage ***Stein Score***:

- By modeling a score function, instead of the density function, we can sidestep the difficulty of computing the intractable normalizaing constants.

- *Stein Score* function: $\nabla_{\mathbf{x}} \log p(\mathbf{x})$.
  - **Not a gradient w.r.t. model parameters.**
  - Gradient of the log probability density function.
  - Not same as the score in stat.
- It is a direction that maximizes a log data density.
- A model for approximating the score function is called a *score-based model* $s_\theta(\mathbf{a})$.
- Score-based models does not have to compute the intractable normalizing constant, Z_θ.

$$s_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_\theta}_{\text{Constant}}.$$

13.6.1 Fisher Divergence

We need to know about *Fisher Divergence*:

- Given *i.i.d.* samples $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\} \sim p_{data}(\mathbf{x}) = p(\mathbf{x})$.
- Estimating the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$.
- Score model $s_\theta(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^D$.
- Use score estimator $s_\theta(x)$:

$$\mathcal{L}_\theta = \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}\left[||\nabla_{\mathbf{x}} \log p(\mathbf{x}) - s_\theta(\mathbf{x})||_2^2\right].$$

- It is called *Fisher divergence*.
- Intuitively, the Fisher divergence compares the squared distance between the ground-truth data score and the score-based model.
  - It changes the problem into a *regression* problem.
- Direct computation of the divergence is **infeasible** due to the unknown data score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$.
  - Since we have no access to the true data distribution $p(\mathbf{x})$.

Fortunately, there exists a family of methods called ***score matching*** that minimize the Fisher divergence without knowledge of the ground-truth data score.

- Score matching objectives can directly be estimated on a dataset and optimized with stochastic gradient descent, analogous to the log-likelihood objective for training likelihood-based models (with known normalizing constants).
- We can train the score-based model by minimizing a score matching objective, without requiring adversarial optimization.

$$\mathcal{L}_\theta = \mathbb{E}_{p(\mathbf{x})}\left[\frac{1}{2}||s_\theta(x)||_2^2 + tr(\nabla_{\mathbf{x}} s_\theta(x))\right]$$
$$\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{2}||s_\theta(x)||_2^2 + tr(\nabla_{\mathbf{x}} s_\theta(x))\right]$$

- $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\} \sim p(\mathbf{x})$
- $\nabla_{\mathbf{x}} s_\theta(x)$: Jacobian
- Remove the dependency of $p(\mathbf{x})$

$$\mathcal{L}_\theta = \frac{1}{2}\mathbb{E}_{p(x)}\left[||\nabla_x \log p(x) - s_\theta(x)||_2^2\right] \tag{13.53}$$
$$= \frac{1}{2}\mathbb{E}_{p(x)}\left[\left(\nabla_x \log p(x) - s_\theta(x)\right)^2\right] \tag{13.54}$$
$$= \frac{1}{2}\int p(x)(\nabla_x \log p(x) - s_\theta(x))^2 dx \tag{13.55}$$
$$= \underbrace{\frac{1}{2}\int p(x)(\nabla_x \log p(x))^2 dx}_{\text{independent from theta}} + \frac{1}{2}\int p(x) s_\theta(x)^2 dx - \int p(x) s_\theta(x) \nabla_x \log p(x) dx \tag{13.56}$$
$$= \cdots - \int p(x) s_\theta(x) \nabla_x \log p(x) dx \tag{13.57}$$
$$= \cdots - \int p(x) s_\theta(x) \frac{\nabla_x p(\mathbf{x})}{p(x)} dx \tag{13.58}$$
$$= \cdots - \int \nabla_{\mathbf{x}} p(x) s_\theta(x) dx \tag{13.59}$$
$$= \cdots - \left[p(x) s_\theta(x)\right]_{x=-\infty}^{\infty} + \int p(x) \nabla_x s_\theta(x) dx \tag{13.60}$$
$$= \frac{1}{2}\int p(x) s_\theta(x)^2 dx + \int p(x) \nabla_x s_\theta(x) dx + \text{const} \tag{13.61}$$
$$= \frac{1}{2}\mathbb{E}_{p(x)}[s_\theta(x)^2] + \mathbb{E}_{p(x)}[\nabla_x s_\theta(x)] + \text{const.} \tag{13.62}$$

- The second last term used the integration by parts.
- The last step is done by a boundary condition assumption which makes score function to be zero (*c.f.,* Sliced score matching paper).
  - $p_{data}(x) \to 0$ as $|x| \to \infty$.
  - In other words, gradient vanishes on the boundary.

13.6.2 Langevin Dynamics

- Once we have trained a score-based model $s_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$, we can use an iterative procedure called *Langevin Dynamics* (LD) [**?**] to draw samples from it.
- LD provides an MCMC procedure to sample from a distribution $p(\mathbf{x})$ using only its score function.

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \frac{\epsilon}{2}\nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1}) + \sqrt{\epsilon}\mathbf{z}_t$$

- Specifically, it initializes the chain from an arbitrary prior distribution $\mathbf{x}_0 \sim \pi(\mathbf{x})$, and then iterates the following
- Sample from $p(x)$ using only the score $\nabla_x \log p(x)$.
- $\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
- ϵ is the step size.
- As $T \to \infty$ and $\epsilon \to 0$, $\mathbf{x}_T$ will become the true probability density $p(\mathbf{x})$.

Part III

Natural Language Processing

Chapter 14

Introduction

14.1 Evaluation Metrics

- Recall: TP/(TP+FN). Find all relevant cases whithin a dataset.
- Precision: TP/(TP+FP): While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant.
- The F1 score is the harmonic mean of precision and recall taking both metrics into account in the following equation:

14.1.1 Perplexity

Intuitively, perplexity can be understood as a *measure of uncertainty*. The perplexity of a language model can be seen as the level of perplexity. Consider a language model with an entropy of three bits, in which each bit encodes two possible outcomes of equal probability. This means that when predicting a symbol, that language model has to choose among $2^3 = 8$ possible options. Thus, we can argue that this language model has a perplexity of 8.

It can be modeled as $2^H(P, Q)$:

$$\begin{aligned} PPL(W) &= P(w_1, \cdots, w_N)^{-\frac{1}{N}} \\ &\approx \left(\prod_{i=1}^{N} P(w_i | w_{<i}) \right)^{-\frac{1}{N}} \\ &= \sqrt[n]{\frac{1}{\prod_{i=1}^{N} P(w_i | w_{<i})}} \end{aligned}$$

Let's derive it from a cross-entropy. We want to optimize P_θ instead of the true distribution P:

$$\begin{aligned}
\mathcal{L}_{CE} &= -\mathbb{E}_{w\sim P}[P_\theta(w_i|w_{<i})] && (14.1)\\
&\approx -\frac{1}{N}\sum_{i=1}^{N}\log P_\theta(w_i|w_{<i}) && (14.2)\\
&= -\frac{1}{N}\log\prod_{i=1}^{N}P_\theta(w_i|w_{<i}) && (14.3)\\
&= \log\left(\prod_{i=1}^{N}P_\theta(w_i|w_{<i})\right)^{-\frac{1}{N}} && (14.4)\\
&= \log\sqrt[N]{\frac{1}{\prod_{i=1}^{N}P_\theta(w_i|w_{<i})}} && (14.5)\\
& && (14.6)
\end{aligned}$$

Thus, $PPL(W) = \exp\Big(\mathcal{L}_{CE}\Big)$.

14.1.2 Cross-Entropy and Perplexity

$$\begin{aligned}
H(P,Q) &= -\sum_x P(x)\log Q(x)\\
&= -\sum_x P(x)[\log P(x) + \log Q(x) - \log P(x)]\\
&= -\sum_x P(x)\left[\log P(x) + \log\frac{Q(x)}{P(x)}\right]\\
&= H(P) + D_{KL}(P||Q)
\end{aligned}$$

It should be noted that since the empirical entropy $H(P)$ is unoptimizable, when we train a language model with the objective of minimizing the cross entropy loss, the true objective is to minimize the KL-divergence of the distribution, which was learned by our language model from the empirical distribution of the language.

Chapter 15

Transformer

15.1 Attention Mechanism

The attention mechanism mimics the retrieval of a value v_i for a query q based on a key k_i in database.

$$attn(q, k, v) = \sum_i sim(q, k_i) \times v_i$$

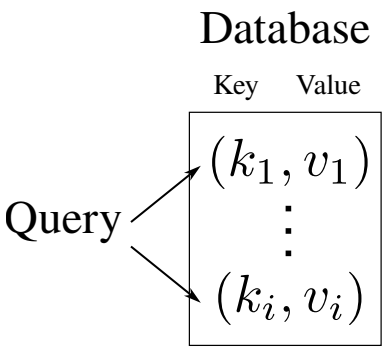


Figure 15.1: The most similar key will be selected by measuring a similarity between a query and a key.

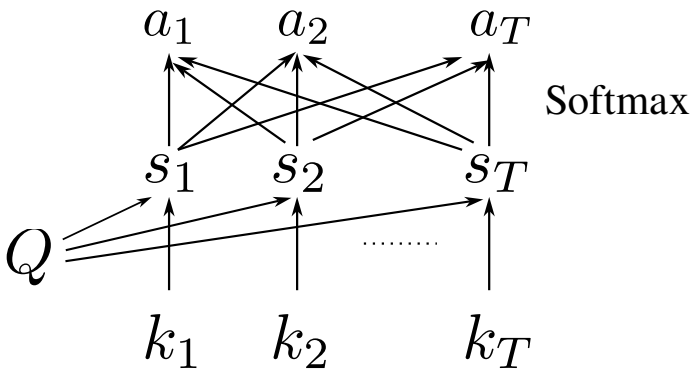


Figure 15.2: The similarity s_t is computed by a query and keys

There are several choices for a similarity function.

- $q^T k_i$: dot product.
- $\frac{q^T k_i}{\sqrt{d}}$: scaled dot product.
- $q^T W k_i$: general dot product.
- $w_q^T q + w_k^T k_i$: additive similarity.

Finally, the attention score can be computed by using a softmax:

$$a_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

15.2 Transformer

Attention:

$$attn(Q, K, V) = softmax(\frac{Q^T K}{\sqrt{d_k}})V$$

Masked attention:

$$\text{MA}(Q, K, V) = softmax\left(\frac{Q^T K + M}{\sqrt{d_k}}\right)V,$$

where M is a matrix of 0 and $-\infty$. Note that $-\infty$ will make *exp* term to be zero.

Bibliography

[1] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Adaptive computation and machine learning series. MIT, Cambridge, MA, 2012.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.