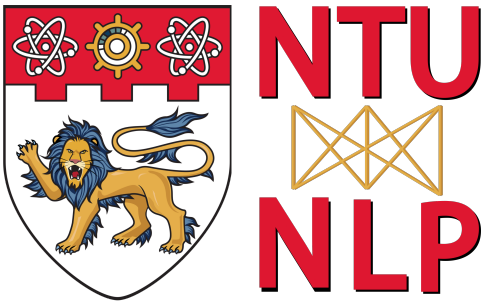
Deep Statistical Learning



My Note

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

Contents

Part I

Introduction

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 Transformations of Random Variable

1.2.1 Formal Definition

Suppose X is a continuous random variable with pdf $f(x)$. If we define $Y = g(X)$, where $g(\cdot)$ is a monotonically increasing function, then the pdf of Y can be obtained as follows:

$$\begin{aligned} p(Y \leq y) &= p(g(X) \leq y) \\ &= p(X \leq g^{-1}(y)) \end{aligned}$$

This can be re-written as by definition

$$F_Y(y) = F_X(g^{-1}(y))$$

By differentiating the CDFs on both sides w.r.t. y, we can get the pdf of Y. If the function $g(\cdot)$ is monotonically increasing, then the pdf of Y is given by

$$f_Y(y) = f_X(g^{-1}(y))\frac{d}{dy}g^{-1}(y)$$

On the other hand, if it is monotonically decreasing, then the pdf of Y is given by

$$f_Y(y) = -f_X(g^{-1}(y))\frac{d}{dy}g^{-1}(y)$$

Compactly, the above two equations can be combined into a following equation:

$$f_Y(y) = f_X(g^{-1}(y))\left|\frac{d}{dy}g^{-1}(y)\right|$$

1.2.2 Intuition

Given a random variable z and its known probability density function $z \sim \pi(z)$, we would like to construct a new random variable using a one-to-one mapping function $x = f(z)$. The function f is invertible, so $z = f^{-1}(x)$. Now the question is how to infer the unknown probability density function of the new variable, $p(x)$?

$$\int p(x)dx = \int \pi(z)dz = 1 \text{ ; Definition of probability distribution.}$$
$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|\frac{df^{-1}(x)}{dx}\right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

In multivariate case,

$$\mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x}) \tag{1.1}$$
$$p(\mathbf{x}) = \pi(\mathbf{z})\left|\det\frac{d\mathbf{z}}{d\mathbf{x}}\right| = \pi(f^{-1}(\mathbf{x}))\left|\det\frac{df^{-1}}{d\mathbf{x}}\right| \tag{1.2}$$

1.3 Gaussian Distribution

For a D-dimensional vector $\mathbf{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{1.3}$$
$$\tag{1.4}$$

1.3.1 Conditional Gaussian Distribution

Consider first the case of conditional distributions. Suppose $\mathbf{x}$ is a D-dimensional vector with Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and that we partition $\mathbf{x}$ into two disjoint subsets $\mathbf{x}_a$ and $\mathbf{x}_b$.

Thus, $\mathbf{x}_a$ has M components and $\mathbf{x}_b$ has $D - M$ components.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix}.$$

Similarly,

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix}$$

and the covariance matrix is given by

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{bmatrix}$$

Note that the symmetry $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$ implies that $\boldsymbol{\Sigma}_{ab}^T = \boldsymbol{\Sigma}_{ba}$. We can also define a *precision matrix* as follows:

$$\boldsymbol{\Lambda} \equiv \boldsymbol{\Sigma}^{-1}$$

We also introduce a partitioned form of the precision matrix:

$$\boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{bmatrix} \tag{1.5}$$

Because the inverse of a symmetric matrix is also symmetric, we see that $\boldsymbol{\Lambda}_{aa}$ and $\boldsymbol{\Lambda}_{bb}$ are symmetric and $\boldsymbol{\Lambda}_{ab}^T = \boldsymbol{\Lambda}_{ba}$. Note that, for instance, $\boldsymbol{\Lambda}_{aa}$ is not simply given by the inverse of $\boldsymbol{\Sigma}_{aa}$.

Now let's compute the conditional probability:

$$\begin{aligned} -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) &= \frac{1}{2}\left(\left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} \right)^T \begin{bmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{bmatrix} \left(\begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} \right) \right) \\ &= -\frac{1}{2}\big((\mathbf{x}_a - \boldsymbol{\mu}_a)^T \boldsymbol{\Lambda}_{aa} (\mathbf{x}_a - \boldsymbol{\mu}_a) + (\mathbf{x}_a - \boldsymbol{\mu}_a)^T \boldsymbol{\Lambda}_{ab} (\mathbf{x}_a - \boldsymbol{\mu}_a) \\ &\quad + (\mathbf{x}_a - \boldsymbol{\mu}_a)^T \boldsymbol{\Lambda}_{ba} (\mathbf{x}_a - \boldsymbol{\mu}_a) + (\mathbf{x}_a - \boldsymbol{\mu}_a)^T \boldsymbol{\Lambda}_{bb} (\mathbf{x}_a - \boldsymbol{\mu}_a) \big) \end{aligned}$$

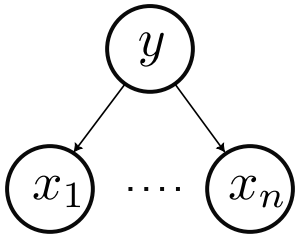
1.4 Naive Bayes

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
$\hat{\pi} = \frac{N_c}{N}, \hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$

1.5 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.

1.6 Decision Tree

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision making).

- Classification trees: Tree models where the target variable take a discrete set of values; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.
- Regression trees: Decision trees where the target variable can take continuous values (typically real numbers).

The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both them.

Some techniques, often called ensemble methods, construct more than one decision tree:

- Boosted tree: Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled. A typical example is AdaBoost.
- Bootstrap aggregated (or bagged): decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction, e.g., random forest.

- Rotation forest: in which every decision tree is trained by first applying principal component analysis (PCA) on a random subset of the input features

The basic algorithm used in decision trees is known as the ID3 (by Quinlan) algorithm. The ID3 algorithm builds decision trees using a top-down, greedy approach. Briefly, the steps to the algorithm are: Select the best attribute → Assign A as the decision attribute (test case) for the NODE. - For each value of A, create a new descendant of the NODE. - Sort the training examples to the appropriate descendant node leaf. - If examples are perfectly.

Now, the next big question is how to choose the best attribute. For ID3, we think of the best attribute in terms of which attribute has the most information gain, a measure that expresses how well an attribute splits that data into groups based on classification.

Pseudocode: ID3 is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples or until all attributes have been used.

The pseudocode assumes that the attributes are discrete and that the classification is binary. Examples are the training example. Target attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Finally, it returns a decision tree that correctly classifies the given Examples.

1.6.1 Information Gain

Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification. As you can see in the Fig. 1.3, attribute with low information gain (right) splits the data relatively evenly and as a result doesn't bring us any closer to a decision. Whereas, an attribute with high information gain (left) splits the data into groups with an uneven number of positives and negatives and as a result helps in separating the two from each other.

To define information gain precisely, we need to define a measure commonly used in information theory called entropy that measures the level of impurity in a group of examples. Mathematically, it is defined as:

$$Entropy = -\sum_i p_i \log p_i$$

, where i is the class index. Since, the basic version of the ID3 algorithm deal with the case where classification are either positive or negative, we can define entropy as:

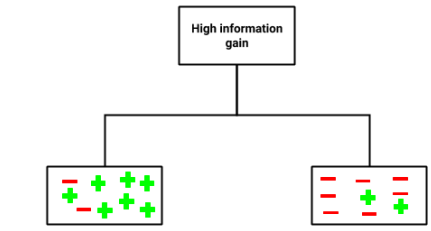
$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

, where

- S: training examples
- p_+: is the proportion of positive examples in S
- p_-: is the proportion of negative examples in S

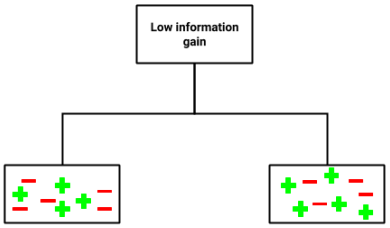
To illustrate, suppose S is a sample containing 14 boolean examples, with 9 positive and 5 negative examples. Then, the entropy of S relative to this boolean classification is:

$$Entropy([9+, 5-]) = -(9/14) \cdot \log_2(9/14) - (5/14) \cdot \log_2(5/14) = 0.940$$



.4

Figure 1.1: High information gain.



.4

Figure 1.2: Low information gain.

Figure 1.3: Information gain comparison.

Note that entropy is 0 if all the members of S belong to the same class. For example, if all members are positive (p_+=1), then $p_- = 0$, and $Entropy(S) = -1 \cdot \log_2(1) - 0 \cdot \log_2(0) = 0$. Entropy is 1 when the sample contains an equal number of positive and negative examples. If the sample contains unequal number of positive and negative examples, entropy is between 0 and 1. The following figure shows the form of the entropy function relative to a boolean classification as p_+ varies between 0 and 1.

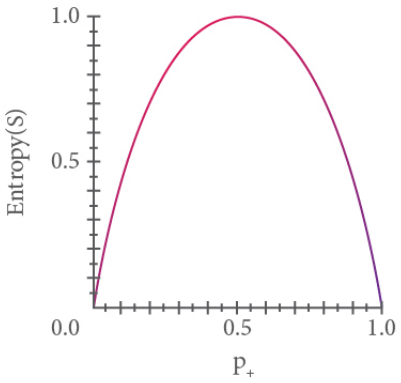


Figure 1.4: Entropy.

Now, given entropy as a measure of the impurity in a sample of training examples, we can now define information gain as a measure of the effectiveness of an attribute in classifying the training data. Information gain, $Gain(S, A)$ of an attribute A, relative to a sample of examples S, is defined as:

$$Gain(S, A) \equiv Entrpoy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

, where $|S_v|$ is a sample belongs to class v and $|S|$ is the number of training samples. In other

words,

Gain = Parent node of Entropy - [Average of Children Nodes Entropy]

Chapter 2

Bayesian Regression

Integrate over all θ

$$P(heads \mid D) = \int_\theta P(heads, \theta \mid D) d\theta \tag{2.1}$$

$$= \int_\theta P(heads \mid \theta, D) P(\theta \mid D) d\theta \quad \because \text{Chain rule, } P(A, B|C) = P(A|B, C)P(B|C) \tag{2.2}$$

$$= \int_\theta \theta P(\theta \mid D) d\theta \tag{2.3}$$

$$= E[\theta|D] \tag{2.4}$$

$$= \frac{n_H + \alpha}{n_H + \alpha + n_T + \beta} \tag{2.5}$$

2.1 Curve Fitting

We can assume that a target variable has a Gaussian distribution with a mean equal to the value $y(x, \mathbf{w})$ of the polynomial curven given by

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1}), \tag{2.6}$$

- t: target variable

  $$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

  where ϵ is a zero mean Gaussian noise with precision (inverse variance) β.

We not use the training data $\{\mathbf{x}, \mathbf{y}\}$ to determine the values of the unknown parameters $\mathbf{w}$ and β by maximum likelihood. If the data are assumed to be drawn independently from the distribution, then the likelihood function is given by

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}). \tag{2.7}$$

We can take a step towards a more Bayesian approach and introduce a prior distribution over the polynomial coefficients $\mathbf{w}$. For simplicity, we can use a Gaussian distribution from

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}), \tag{2.8}$$

where α is the precision of the distribution. Using Bayes' theorem, the posterior distribution for $\mathbf{w}$ is proportional to the product of the prior distribution and the likelihood function

$$p(\mathbf{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{w}|\mathbf{x}, \mathbf{t}, \beta)p(\mathbf{w}, \alpha). \tag{2.9}$$

We can now determined $\mathbf{w}$ by finding the most probable value of $\mathbf{w}$ given the data, in other words by maximizing the posterior distribution, MAP (maximum posterior). Taking a negative logarithm, then we can find that the maximum of the posterior is given by the minimum of

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}. \tag{2.10}$$

Thus we see that maximizing the posteiror distribution si equivalent to minimizing the regularized sum of squares error function encountered earler with a regularization parameter given by $\lambda = \alpha/\beta$.

2.2 Bayesian Curve Fitting

Chapter 3

Training, Testing, and Regularization

3.1 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.

3.1.1 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:

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

Chapter 4

Optimization

4.1 Intuition of Gradient

Gradient is a vector function that represents a direction of steepest increase of a function to be differentiated at a certain point. For example, a convex function $z = ax^2 + by^2$ has a gradient $[2ax, 2by]$. Its steepest descent direction is $[-2ax, -2by]$. At the point x_0, y_0, the gradient direction for the function is $2ax_0(y - y_0) = 2by_0(x - x_0)$. But then the lowest point $(x, y) = (0, 0)$ does not lie on the line. Thus, we cannot find that minimum point in one step of "gradient descent". The steepeset direction does not lead to the bottom of the bowl.

4.1.1 Gradient Descent Proof

Most deep learning algorithms involve **optimization**.

- The derivative of the objective function $f(\mathbf{x})$ provides the slope of $f(\mathbf{x})$ at the point $f(\mathbf{x})$.
- It tells us how to change $\mathbf{x}$ in order to make a small improvement in our goal.

A function $f(\mathbf{x})$ can be approximated by its first-order Taylor expansion at $\bar{\mathbf{x}}$:

$$f(\mathbf{x}) \approx f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})^T (x - \bar{\mathbf{x}})$$

Now let $\mathbf{d} \neq 0, ||\mathbf{d}|| = 1$ be a direction, and in consideration of a new point $\mathbf{x} := \bar{\mathbf{x}} + \mathbf{d}$, we define:

$$f(\bar{\mathbf{x}} + \mathbf{d}) \approx f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})^T \mathbf{d}$$

We would like to choose $\mathbf{d}$ that minimizes the function f. From the Cauchy-Schwarz inequality [1], we know that

$$\nabla f(\bar{\mathbf{x}})^T \mathbf{d} \leq ||\nabla f(\bar{\mathbf{x}})||\ ||\mathbf{d}||$$

with equality when $\mathbf{d} = \lambda \nabla f(\bar{\mathbf{x}})$, where $\lambda \in R$.

[1] Cauchy-Schwarz Inequaility: $|\mathbf{a} \cdot \mathbf{b}| \leq ||\mathbf{a}||\ ||\mathbf{b}||$. Equality holds if and only if either $\mathbf{a}$ or $\mathbf{b}$ is a multiple of the other.

Since we want to minimize the function f, we negate the steepest direction $\mathbf{d}^*$ so the iterations of steepest descent becomes

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta \nabla f(\mathbf{x}^{(k)})$$

, where k is the index of iteration step and η is the learning rate parameter.

4.2 Normalized Gradient Descent

The underlying issue of the vanila gradient descent is the presence of saddle points in nonconvex functions; the gradient $\nabla f(x)$ vanishes near saddle points, which causes GD to "stall" in neighboring regions. This both slows the overall convergence rate and makes detection of local minima difficult. The detrimental effects of this issue become particularly severe in high-dimensional problems where the number of saddle points may proliferate.

However, in the normalized gradient descent

$$\frac{\nabla f(x)}{||\nabla f(x)||}$$

The normalized gradient preserves the direction of the gradient but ignores magnitude, because the normalization does not vanish near saddle points, the intuitive expectation is that NGD should not slow down in the neighborhood of saddle points and should therefore escape quickly.

4.3 Projected Gradient Descent

Gradient Descent (GD) is a standard way to solve unconstrained optimization problem. Starting from an initial point $x \in R^n$, GD itereates until a stopping criterion is met. Projected Gradient Descent (PGD) is a way to solve constrained optimization problem. Consider a constraint set $\mathcal{Q}$, starting from a initial point $x_0 \in \mathcal{Q}$, PGD iterates the following equation until a stopping condition is met:

$$x_{k+1} = P_{\mathcal{Q}}\Big(x_k - t_k \nabla f(x_k)\Big),$$

where $P_{\mathcal{Q}}$ is the projection operator

$$P_{\mathcal{Q}}(x_0) = \underset{x \in \mathcal{Q}}{\operatorname{argmin}} \frac{1}{2}||x - x_0||_2^2$$

In other words, given a point x_0, $P_{\mathcal{Q}}$ tries to to find a point $x \in \mathcal{Q}$ which is "closest"to x_0.

Note that a vector projection can be expressed as follows:

$$a_1 = ||a|| \cos\theta = \mathbf{a} \cdot \hat{\mathbf{b}} = \mathbf{a} \cdot \frac{\mathbf{b}}{||\mathbf{b}||}$$

Thus, a projection for unit L_2 ball is given by the solution of the equation as follows:

$$\mathbf{x} = \mathcal{P}_{||x||_2 \leq 1}(\mathbf{y})$$

The solution is

$$\mathbf{x} = \frac{\mathbf{y}}{\max\{1, ||\mathbf{y}||_2\}}$$

The "geometric" proof is given as follows: Let $\mathcal{S} = \{\mathbf{x} \in R^n : ||\mathbf{x}||_2 \leq 1\}$.

- If $\mathbf{y} \in \mathcal{S}$, then $||y||_2 \leq 1$ and $\mathbf{y}$ itself is the closest point to $\mathbf{y}$.
- If $\mathbf{y} \notin \mathcal{S}$, then $||y||_2 > 1$ and the closest point $\mathbf{x} \in \mathcal{S}$ to $\mathbf{y}$ will be simply $\frac{\mathbf{y}}{||\mathbf{y}||_2}$ as the norm of $\frac{\mathbf{y}}{||\mathbf{y}||_2} = 1$.

By combining the bost cases, we have

$$\mathbf{x} = \frac{\mathbf{y}}{\max\{1, ||\mathbf{y}||_2\}}$$

4.4 Exponentially Weighted Average

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$

Larger β value covers more longer history. EMA is exponentially weighted average the previous result.

4.5 Bias Correction

The initial values of v_t will be very low which need to be compensated, since the curve starts from 0, there are not many values to average on in the initial points. Thus, the curve is lower than the correct value initially and then moves in line with expected values. The β is the same as the averaging coefficient. As t becomes large, the impact of the bias correction will be decreased.

$$v_t = \frac{v_t}{1 - \beta^t}$$

4.6 Momentum

Momentum can reduce the oscillation in the gradients. Let's say w has a small value and b is in charge of oscillation. Then momentum can cancel out db by averaging them.

$$\begin{aligned}
v_{dw} &= \beta v_{dw} + (1 - \beta)dw \\
v_{db} &= \beta v_{db} + (1 - \beta)db \\
w &= w - \alpha v_{dw} \\
b &= b - \alpha v_{db}
\end{aligned}$$

4.7 Adagrad: Adaptive Gradient

$$\begin{aligned}
v_{dw} &= v_{dw} + dw \cdot dw \\
w &= w - \frac{\alpha}{\sqrt{v_{dw}} + \epsilon} v_{dw}
\end{aligned}$$

A con of Adagrad is learning rate will become very small

4.8 RMS Prop

$$
\begin{aligned}
s_{dw} &= \beta s_{dw} + (1-\beta)dw^2 \\
s_{db} &= \beta s_{db} + (1-\beta)db^2 \\
w &= w - \alpha \frac{dw}{\sqrt{s_{dw}}} \\
b &= b - \alpha \frac{db}{\sqrt{s_{db}}}
\end{aligned}
$$

4.9 ADAM

Its name is derived from adaptive moment estimation, and the reason it's called that is because Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network. N-th moment of a random variable is defined as the expected value of that variable to the power of n. More formally:

$$m_n = E[X^n]$$

To estimates the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch:

Since m and v are estimates of first and second moments, we want to have the following property:

$$E[m_t] = E[g_t] \tag{4.1}$$
$$E[v_t] = E[g_t^2] \tag{4.2}$$
$$\tag{4.3}$$

Unbiased estimators

ADAM uses both momentum style and RMS prop style averaging.

- $v_{dw} = \beta v_{dw} + (1-\beta)dw$
- $v_{db} = \beta v_{db} + (1-\beta)db$
- $s_{dw} = \beta s_{dw} + (1-\beta)dw^2$
- $s_{db} = \beta s_{db} + (1-\beta)db^2$

Using them,

- $v_{dw}^{\text{corr}} = \frac{v_{dw}}{1-\beta_1^t}$

- $v_{db}^{\text{corr}} = \frac{v_{db}}{1-\beta_1^t}$
- $s_{dw}^{\text{corr}} = \frac{s_{dw}}{1-\beta_2^t}$
- $s_{db}^{\text{corr}} = \frac{s_{db}}{1-\beta_2^t}$

Finally,

$$w = w - \alpha \frac{v_{dw}^{\text{corr}}}{\sqrt{s_{dw}^{\text{corr}}} + \varepsilon}$$
$$b = b - \alpha \frac{v_{db}^{\text{corr}}}{\sqrt{s_{db}^{\text{corr}}} + \varepsilon}$$

Chapter 5

Fourier Analysis

Reference: Y.D. Chong 2021, NTU Lecture Note

The Fourier transform is one of the most important mathematical tools used for analyzing functions. Given an arbitrary function $f(x)$, with a real domain ($x \in \mathbb{R}$), we can express it as a linear combination of complex waves

5.1 Fourier Series

We begin by discussing the Fourier series, which is used to analyze functions that are periodic in their inputs. A periodic function $f(x)$ is a function of a real variable x that repeats itself every time x changes by a. The constant a is called the *period*. We can write the periodicity condition as

$$f(x + a) = f(x), \forall x \in \mathbb{R}.$$

The value of $f(x)$ can be real or complex, but x should be real.

Let's consider what it means to specify a periodic function $f(x)$. One way to specify the function is to give an explicit mathematical formula for it. Another approach might be to specify the function values in $a/2 \leq x < a/2$. Since there's an uncountably infinite number of points in this domain, we can generally only achieve an approximate specification of f this way, by giving the values of f at a large but finite set x points.

There is another interesting approach to specifying f. We can express it as a linear combination of simpler periodic functions, consisting of sines and cosines:

$$f(x) = \sum_{n=1}^{\infty} \alpha_n \sin\left(\frac{2\pi n x}{a}\right) + \sum_{m=0}^{\infty} \beta_m \cos\left(\frac{2\pi m x}{a}\right).$$

Note that the index n does not include 0; since the sine term with $n = 0$ vanishes for all x, it's redundant. The above formula is called a *Fourier series*. Given the numbers $\{\alpha_n, \beta_m\}$, which are called the *Fourier coefficients*, $f(x)$ can be calculated for any x. The Fourier coefficients are real if $f(x)$ is a real function, or complex if $f(x)$ is complex.

5.1.1 Square-integrable functions

Can arbitrary periodic functions always be expressed as a Fourier series? It turns out that a certain class of periodic functions, commonly encountered in physical contexts, are guaranteed to always be expressible as Fourier series. These are called square-integrable functions such that the integral of the square of the absolute value is finite:

$$\int_{-a/2}^{a/2} |f(x)|^2 dx < \infty.$$

Unless otherwise stated, we will always assume that the functions we're dealing with are square-integrable.

5.1.2 Complex Fourier series and inverse relations

We have written the Fourier series as a sum of sine and cosine functions. However, sines and cosines can be expressed by exponential functions by using *Euler's formula.*

$$e^{ix} = \cos x + i \sin x \tag{5.1}$$

- $\cos x = \frac{e^{ix}+e^{-ix}}{2}$
- $\sin x = \frac{e^{ix}-e^{-ix}}{2i}$

Thus, Fourier series can be expressed as follows:

$$f(x) = \sum_{n=-\infty}^{\infty} e^{2\pi i n x/a} f_n$$

- i: complex number
- n: integer
- a: period
- f_n: Fourier coefficient

Part II

Kernel Methods

Chapter 6

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:

$$\begin{aligned}
J(\mathbf{w}) &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2 && (6.1)\\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w} && (6.2)\\
&= (\mathbf{y}^T - \mathbf{w}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w} && (6.3)\\
&= \mathbf{y}^T\mathbf{y} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\lambda\mathbf{I}\mathbf{w} && (6.4)\\
\frac{\partial J}{\partial \mathbf{w}} &= -\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w} + \mathbf{X}^T\mathbf{X}\mathbf{w} + 2\lambda\mathbf{I}\mathbf{w} = 0 && (6.5)\\
\mathbf{w} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} && (6.6)
\end{aligned}$$

Let's change it into a dual form:

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{6.7}$$
$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{y} \tag{6.8}$$
$$\mathbf{w} = \lambda^{-1}\mathbf{I}(\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w}) \tag{6.9}$$
$$= \mathbf{X}^T\lambda^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{6.10}$$
$$= \mathbf{X}^T\alpha \tag{6.11}$$
$$\lambda\alpha = (\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{6.12}$$
$$= (\mathbf{y} - \mathbf{X}\mathbf{X}^T\alpha) \tag{6.13}$$
$$\mathbf{y} = (\mathbf{X}\mathbf{X}^T\alpha + \lambda\alpha) \tag{6.14}$$
$$\alpha = (\mathbf{X}\mathbf{X}^T + \lambda)^{-1}\mathbf{y} \tag{6.15}$$
$$\alpha = (\mathbf{G} + \lambda)^{-1}\mathbf{y}, \tag{6.16}$$

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 7

Gaussian Process

Reference

7.1 Introduction

A Gaussian process (GP) is a probability distribution over possible functions that fit a set of points. More formally, GPs are distributions over functions $f(x)$ of which the distribution is defined by a mean function $m(x)$ and positive definite covariance function $k(x, x')$ (*i.e., kernel*). Thus, it is a distribution over functions whose shape is defined by the kernel:

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

Since we have the probability distribution over all possible functions, we can calculate the mean and the variance of the function to determine how confident in our predictions.

Let's say we have observations, and we have estimated functions $\mathbf{f}$ with these observations. Now say we have some new points $\mathbf{X}_*$ where we want to predict $f(\mathbf{X}_*)$.

The joint distribution of $\mathbf{f}$ and $\mathbf{f}_*$ can be modeled as:

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} m(\mathbf{X}) \\ m(\mathbf{X}_*) \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix} \right),$$

where

However, what we want is $\mathbf{f}_*$, predictions for new observations, which is a conditional distribution over $\mathbf{f}_*$.

Chapter 8

Support Vector Machine

8.1 Introduction

8.1.1 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,

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

8.2 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).

8.3 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?

8.4 Kernel Trick

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

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \psi(\mathbf{x}_i) \cdot \psi(\mathbf{x}_j) \tag{8.1}$$

Equivalently,

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{8.2}$$

8.5 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:

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}||w||^2 - \sum_i \alpha_i \Big[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \Big] \tag{8.3}$$

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.

8.6 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. (8.3) 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

$$\mathbf{W}(\alpha, b) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{8.4}$$

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.

8.7 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 III

Generative Modeling

Chapter 9

Introduction

9.1 KL Divergence

The KL divergence can be defined as follows:

$$D_{KL}(P||Q) = E_{x\sim P}\left[\log \frac{P(X)}{Q(X)}\right]$$

9.1.1 Properties

- Non symmetric
- $D_{KL} \in [0, \infty]$
- In order for the KL divergence to be finite, the support of P needs to be in the support of Q.

9.1.2 Rewriting the Objective

$$\begin{aligned} D_{KL}(P||Q) &= E_{x\sim P}\left[\log \frac{P(X)}{Q(X)}\right] \\ &= E_{x\sim P}[-\log Q(X)] - \mathcal{H}(P(X)) \end{aligned}$$

- $E_{x\sim P}[-\log Q(X)]$: Cross entropy
- $\mathcal{H}(P(X))$: Entropy of P

9.1.3 Forward and Reverse KL

Let's say there is a true distribution $P(X)$ with two modes and our approximation $Q(X)$ has one mode. Then,

- Forward KL: $D_{KL}(P||Q)$
- Reverse KL: $D_{KL}(Q||P)$

Forward KL: Mean-Seeking Behavior

$$\begin{aligned}\arg\min_\theta D_{KL}(P\|Q) &= \arg\min_\theta E_{x\sim P}[-\log Q_\theta(X)] - \mathcal{H}(P(X)) \\ &= \arg\min_\theta E_{x\sim P}[-\log Q_\theta(X)] \\ &= \arg\max_\theta E_{x\sim P}[\log Q_\theta(X)]\end{aligned}$$

Intuition: x will be sampled from the distribution P, and its value will be estimated from Q. Thus, there will be higher chance that x will be sampled from a space with higher probablity in P. Therefore, Q_θ has to consider all modes, which have high probabilities.

To use the forward KL, we have to have an access to the true model $P(X)$ for sampling.

Reverse KL: Mode-Seeking Behavior

$$\begin{aligned}\arg\min_\theta D_{KL}(Q\|P) &= \arg\min_\theta E_{x\sim Q_\theta}[-\log P(X)] - \mathcal{H}(Q_\theta(X)) \\ &= \arg\max_\theta E_{x\sim Q_\theta}[\log P(X)] + \mathcal{H}(Q_\theta(X))\end{aligned}$$

Intuition: x will be sampled from the distribution Q, and its value will be estimated from P. Thus, there will be higher chance that x will be sampled from a space with higher probablity in Q. Therefore, to maximize the value, we need to focus on a single mode.

To use the reverse KL, we have to be able to evaluate the true model $P(X)$.

Chapter 10

Sampling Based Inference

10.1 Basic Sampling Methods

10.1.1 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.

10.1.2 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:

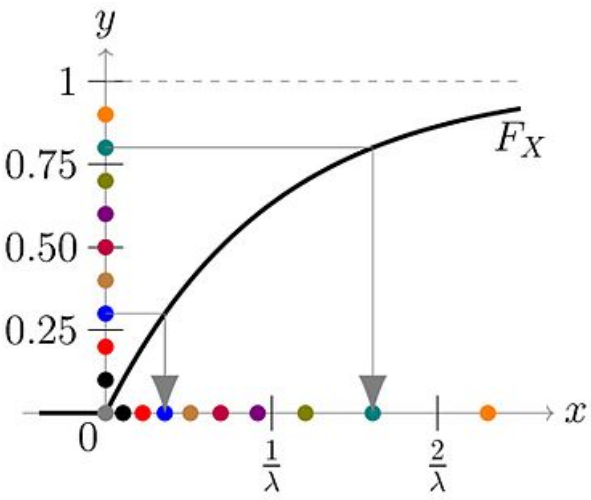


Figure 10.1: y-axis: Uniform distribution, x-axis: sample value

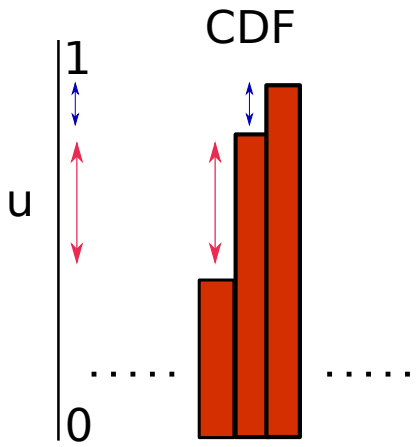


Figure 10.2: How can this sampling method recover the original distribution?

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

10.1.3 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**.

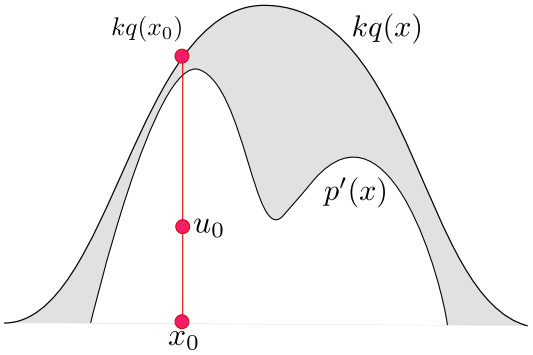
The intuition of rejection sampling is actually similar to Monte-Carlo estimation. By setting a large area (proposal distribution), we can sample points and take them that are inside the our

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.

10.1.4 Importance Sampling

We want to estimate an expectation of function $f(x)$, where $x \sim p(x)$, but it is 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

$$\begin{aligned}\mathbb{E}_p[f(\mathbf{x})] &= \int p(\mathbf{x})f(\mathbf{x})d\mathbf{x} \\ &= \int p(\mathbf{x})f(\mathbf{x})\frac{q(\mathbf{x})}{q(\mathbf{x})}d\mathbf{x} \\ &= \int q(\mathbf{x})\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right]d\mathbf{x} \\ &= \mathbb{E}_q\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right]\end{aligned}$$

- 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 $Q(\mathbf{x})$

By applying the Monte-Carlo method, we can get

$$\mathbb{E}_q\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}, \quad \mathbf{x}_i \sim p(\mathbf{x}) \tag{10.1}$$

- Unbiased estimation
- (Potentially) Smaller variance compared to the vanilla Monte-Carlo method above.
  - $Var_q\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right] < Var_p[f(\mathbf{x})]$
  - When $q(\mathbf{x})$ is high where $|p(\mathbf{x})f(\mathbf{x})|$ is high.

Chapter 11

Markov Chain Monte Carlo

11.1 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.

11.2 Markov Chain

Reference Link

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

Definition 3 *Let D be a finite set. A random process $X_1, X2, \ldots$ with values in D is called a Markov chain if*

$$P(X_t = x_{t+1}|X_t = x_t, \ldots, X_0 = x_0) = P(X_{t+1} = x_{t+1}|X_t = x_t)$$

We can think of X_t as a random state at time t, and the Markovian assumption is that the probability of transitioning from x_t to x_{t+1} only depends on x_t. In other words, the future state depends only on the present. Let p_{ij} be the probability of transitioning from state i to state j. A Markov chain can be defined by a transition probability matrix:

Definition 4 *The matrix $\mathbf{P} = (p_{ij})_{i,j} \in D$ is called the transition probability matrix.*

Thus, P is a $D \times D$ matrix, where D denotes the cardinality of D, and the cell value p_{ij} is the probability of transitioning from state i to state j, and the rows of P must sum to one. In this post, we will restrict ourselves to time *homogeneous Markov chains*:

Definition 5 *A Markov chain is called time homogeneous if*

$$\mathbb{P}\{X_{t+1} = j \,|\, X_n = i\} = p_{ij}, \forall n.$$

It state that *the transition probabilities are not changing as a function of time.* Finally, let's introduce some useful notation for the initial state of the Markov chain. Let

$$\mathbb{P}_{x_0}\{\cdot\} \triangleq \mathbb{P}\{\cdot|X_0 = x_0\}.$$

For example, I will write $\mathbb{P}_a\{X_1 = b\}$ rather than $\mathbb{P}\{X_1 = b|X_0 = a\}$. This is because all the conditional probabilities depend on the initial state, and it the usual notation is cumbersome.

Consider a simple Markov chain modeling the weather. The weather has two states: rainy and sunny. Thus, $D = \{r, s\}$ and X_n is the "weather on day n". The Markov chain model is as follows. If today is rainy (r), tomorrow it is sunny (s) with probability p. If today it is sunny, tomorrow it is rainy with probability q. Then, transition matrix is given by

$$\mathbf{P} = \begin{bmatrix} 1-p & p \\ q & 1-q \end{bmatrix}.$$

The state diagram of the chain can be represented as follows: As a consequence of the Markovian

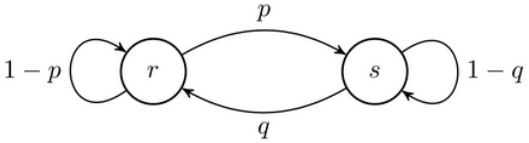


Figure 11.1: A sample Markov chain.

assumptions, the probability of any path on a Markov chain is just the multiplication of the

numbers along the path. For example, for some Markov chain with states $D = \{a, b, c, d\}$, the probability of a particular path, say $a \to b \to b \to d \to c$ factorize as

$$p_{ab} \cdots p_{dc}$$

Let's try to formulate this by

$$\mathbb{P}_i\{X_n = j\}.$$

For instance, let's say $n = 2$. Then, we have

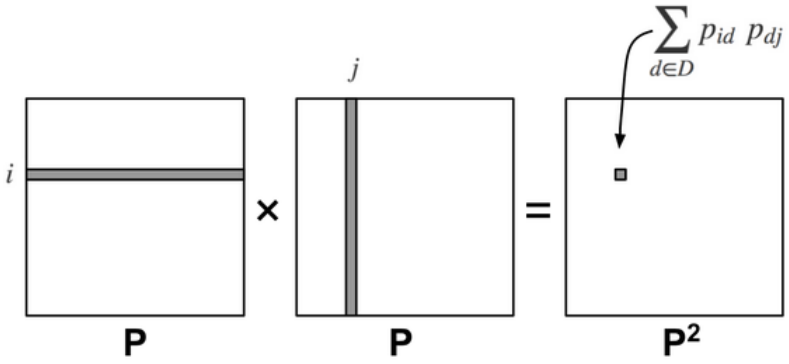
$$\begin{aligned}\mathbb{P}_i\{X_2 = j\} &= \sum_{d \in D} \mathbb{P}_i\{X_2 = j, X_1 = d\} \\ &= \sum_{d \in D} \mathbb{P}_i\{X_2 = j | X_1 = d\}\mathbb{P}_i\{X_1 = d\} \\ &= \sum_{d \in D} p_{id}p_{dj}\end{aligned}$$

This is equivalent to a dot product of the i-th row to j-th column of the transition matrix $\mathbb{P}$. This can be expressed as follows:

$$\mathbb{P}\{X_2 = j | X_0 = i\} = \sum_{d \in D} p_{id}p_{dj} = \left(\mathbf{P}^2_{ij}\right).$$

This can be generalized to

$$\mathbb{P}_i\{X_n = j\} = \left(\mathbf{P}^n_{ij}\right).$$



In sum, the dot product performs marginalization, and everything works out nicely thanks to the Markovian assumption. If the D-dimensional vector v represents a discrete distribution over initial states X_0, then $\mathbf{v^T P}$ is a D-dimensional vector representing the probability distribution over X_1.

11.2.1 Ergodicity

Let's discuss about *ergodicity*, which is a property of a random process in which its time average is the same as its probability space average. It can be defined as follows:

Definition 6 *A Markov chain $\{X_n\}$ is called ergodic if the limit*

$$\boldsymbol{\pi}(j) = \lim_{n \to \infty} \mathbb{P}_i\{X_n = j\}$$

exists for every state j and does not depend on the initial state i. The D-dimensional vector $\boldsymbol{\pi}(j)$ is called the stationary probability.

In other words,

- The probability $\boldsymbol{\pi}(j)$ of reaching at state j (*i.e.,* $\mathbb{P}_i\{X_n = j\}$)
- After a long time (*i.e.,* $\lim_{n\to\infty}$)
- Regardless of the initial state i (*i.e.,* $\mathbb{P}_i\{X_n = j\}$).

Equivalently, it can be expressed as

$$\boldsymbol{\pi}(j) = \lim_{n\to\infty} \left(\mathbf{P}^n\right)_{ij}.$$

The ergodicity gives the following property:

$$\begin{aligned}
\boldsymbol{\pi}(j) &= \lim_{n\to\infty} \left(\mathbf{P}^n\right)_{ij} \\
&\overset{\star}{=} \lim_{n\to\infty} \left(\mathbf{P}^{n+1}\right)_{ij} \\
&= \lim_{n\to\infty} \left(\mathbf{P}^n\mathbf{P}\right)_{ij} \\
&= \lim_{n\to\infty} \sum_{d\in D} \left(\mathbf{P}^n\right)_{id}\mathbf{P}_{dj} \\
&= \sum_{d\in D} \boldsymbol{\pi}(d)\mathbf{P}_{dj}
\end{aligned}$$

- The step $\star$ holds since the step n and $n+1$ does not matter under the limit.

Finally, we can write this as

$$\boldsymbol{\pi}^T = \boldsymbol{\pi}^T\mathbf{P},$$

where $\boldsymbol{\pi}$ is a column vector. Hence, the name "stationary probability distribution" denotes that it is a distribution that does not change over time.

Since we are interested in ergodicity, let's now introduce some properties related to reachability and long-term behavior.

Definition 7 *We say that there is a path from i to j ($i \rightsquigarrow j$) if there is a nonzero probability that starting at i, we can reach j at some point in the future.*

Definition 8 *A state i is called **transient** if there exists a state j such that $i \rightsquigarrow j$ but $j \not\rightsquigarrow i$.*

Definition 9 *A state i is called **recurrent** if for all states j there exist $i \rightsquigarrow j$ and $j \rightsquigarrow i$.*

Definition 10 *A Markov chain is **irreducible** if $i \leftrightarrow j, \forall i, j \in S$. Simply, if all states are able to visit other states, it is irreducible.*

Definition 11 *State i has a **period** d (*i.e., *periodically visit the state i) $\leftrightarrow$ aperiodic.*

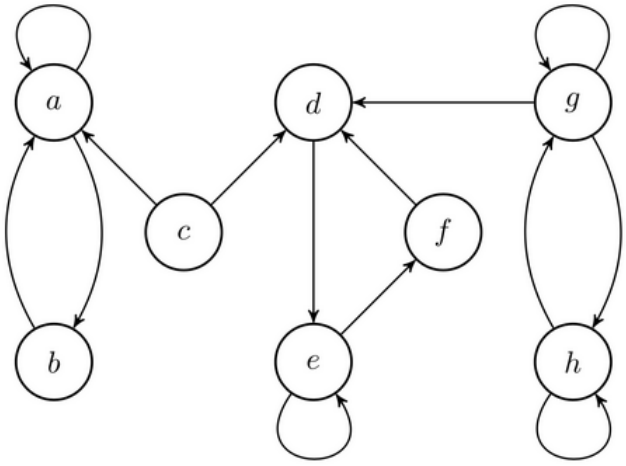


Figure 11.2: A Markov chain with $D = \{a, b, c, d, e, f, g, h\}$. The state c is a transient state. Note that g and h are also transient as there exist states that they cannot return. There are two recurrent classes: $\{a, b\}$ and $\{d, e, f\}$.

Intuitively, a *transient* state is a state that cannot return to itself; while a *recurrent* state can return. In other words, these two conditions are opposite. A transient state is not recurrent, and a recurrent state is not transient. Note that if a state is recurrent, every reachable state is also recurrent. We think of a set of recurrent states as a "class" or a "recurrent class". We can notice that Fig. 11.1 is ergodic but Fig. 11.2 is not since some states are not recurrent.

In sum a state is ergodic if the state is recurrent and aperiodic. Markov chain is ergodic if all states are ergodic.

11.2.2 Limit Theorem of Markov chain

Reference Link

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.

11.2.3 Time Reversibility

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.

11.3 Markov Chain Monte Carlo

MCMC aims to generate samples from some complex probability distribution $p(x)$ that is difficult to directly sample from.

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.

Ground rules for MCMCs:

- MCMCs stochastically explore the parameter space in such a way that the histogram of their samples produces the target distribution.
- Markovian: Evolution of the chain (*i.e.,* collections of samples from one iteration to the other) only depends on the current position and some transition probability distribution (*i.e.,* how we move from one point in parameter space to another). This means that the chain has no memory and past samples cannot be used to determine new positions in parameter space.
- The chain will converge to the target distribution if the transition probability is:
  - Irreducible: From any point in parameter space, we must be able to reach any other point in the space in a finite number of steps.
  - Positive recurrent: For any point in parameter space, the expected number of steps for the chain to return to that point is finite. This means that the chain must be able to re-visit previously explored areas of parameter space.
  - Aperiodic: The number of steps to return to a given point must not be a multiple of some value k. This means that the chain cannot get caught in cycles.

11.3.1 Metropolis-Hasting Algorithm

Suppose we have a target posterior distribution $\pi(x)$, where x here can be any collection of parameters (not a single parameter). In order to move around this parameter space we must formulate some proposal distribution:

$$q(x_{i+1} \mid x_i),$$

that specifies the probability of moving to a point in parameter space, x_{i+1}, given that we are currently at x_i. The Metropolis Hastings algorithm accepts a "jump" to x_{i+1} with the following probability

$$\kappa(x_{i+1} \mid x_i) = \min\left(1, \frac{\pi(x_{i+1})q(x_i \mid x_{i+1})}{\pi(x_i)q(x_{i+1} \mid x_i)}\right) = \min(1, H),$$

where the fraction above is called the Hastings ratio, H. The above expression represents that the probability of transitioning from point x_{i+1} given the current position x_i is a function of the ratio of the value of the posterior at the new point to the old point (*i.e.*, $\pi(x_{i+1})/\pi(x_i)$) and the ratio of the transition probabilities at the new point to the old point (*i.e.*, $q(x_i \mid x_{i+1})/q(x_{i+1} \mid x_i)$). Firstly, it is clear that if the ratio is bigger than 1 then the jump will be accepted. Secondly, the ratio of the target posteriors ensures that the chain will gradually move to high probability regions. Lastly, the ratio of the transition probabilities ensures that the chain is not "favored" toward certain locations by the proposal distribution function. Note that many proposal distributions are symmetric (*i.e.*, $q(x_{i+1} \mid x_i) = q(x_i \mid x_{i+1})$).

The Metropolis-Hasting algorithm is then:

```
def mh_sampler(x0, lnprob_fn, prop_fn, prop_fn_kwargs={}, iterations=100000):
    """Simple metropolis hastings sampler.

    :param x0: Initial array of parameters.
    :param lnprob_fn: Function to compute log-posterior.
    :param prop_fn: Function to perform jumps.
    :param prop_fn_kwargs: Keyword arguments for proposal function
    :param iterations: Number of iterations to run sampler. Default=100000

    :returns:
        (chain, acceptance, lnprob) tuple of parameter chain , acceptance rate
        and log-posterior chain.
    """

    # number of dimensions
    ndim = len(x0)

    # initialize chain, acceptance rate and lnprob
    chain = np.zeros((iterations, ndim))
    lnprob = np.zeros(iterations)
    accept_rate = np.zeros(iterations)

    # first samples
    chain[0] = x0
    lnprob0 = lnprob_fn(x0)
    lnprob[0] = lnprob0

    # start loop
    naccept = 0
    for ii in range(1, iterations):

        # propose
        x_star, factor = prop_fn(x0, **prop_fn_kwargs)

        # draw random uniform number
        u = np.random.uniform(0, 1)

        # compute hastings ratio
        lnprob_star = lnprob_fn(x_star)
        H = np.exp(lnprob_star - lnprob0) * factor

        # accept/reject step (update acceptance counter)
        if u < H:
            x0 = x_star
            lnprob0 = lnprob_star
            naccept += 1

        # update chain
        chain[ii] = x0
        lnprob[ii] = lnprob0
```

```
51          accept_rate[ii] = naccept / ii
52
53      return chain, accept_rate, lnprob
```

Chapter 12

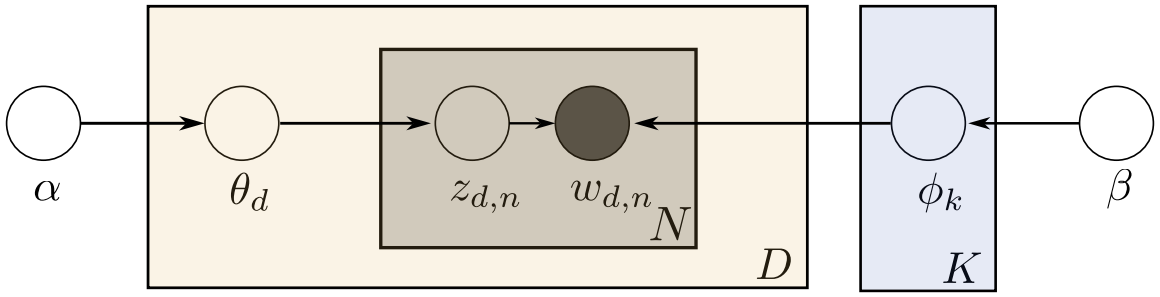
Topic Modeling

12.1 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:

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

12.1.1 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 $\mathbf{z}$.

12.1.2 Dirichlet Distribution

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

$$p(P = \{p_i\}|\alpha_i) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i p_i^{\alpha_i - 1} \tag{12.1}$$

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

Chapter 13

Latent Variable Models

13.1 Introduction

13.1.1 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 $z^* = \text{argmax}_z \, p(\mathbf{x}|\mathbf{z})$ for the data $\mathbf{x}$ as shown in Fig. 13.1(b)

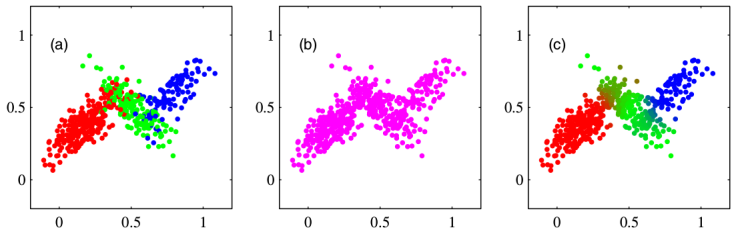


Figure 13.1: (a) Complete data set $p(\mathbf{x}|\mathbf{z})$. (b) Incomplete data set $p(\mathbf{x})$. (c) Inference result

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 14

Clustering

14.1 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?)

14.2 Gaussian Mixture Models

14.2.1 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

$$\sum_k m_k = N.$$

14.2.2 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:

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

14.2.3 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

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

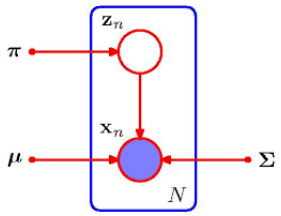


Figure 14.1: Graphical representation of GMM model. The GMM models a joint distribution $p(\mathbf{x}, \mathbf{z})$ in terms of a marginal distribution $p(\mathbf{z})$ and conditional distribution $p(\mathbf{x}|\mathbf{z})$ to model $p(\mathbf{x})$. Each $\mathbf{x}_n$ is coupled with $\mathbf{z}_n$

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. 14.1.

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 §13.1.1.

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 $\mathbf{x}_n$.

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

14.2.4 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.

14.2.5 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

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

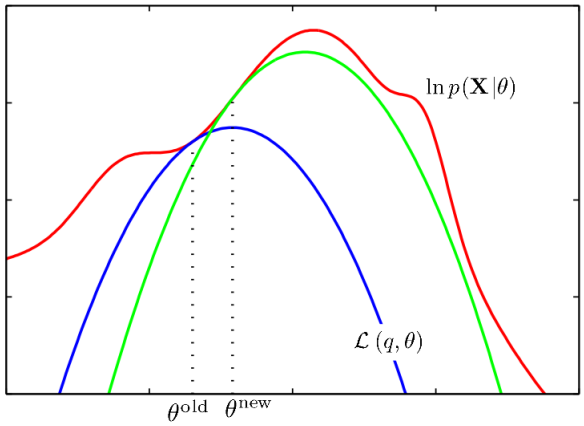


Figure 14.2: M-step of EM algorithm

14.3 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

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

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

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

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.

14.4 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

$$\begin{aligned}
\log p(X|\theta) &= \ln \int p(X, Z|\theta) dZ && (14.1)\\
&= \ln \int q(Z|X) \frac{p(X, Z|\theta)}{q(Z|X)} dZ && (14.2)\\
&\geq \underbrace{\int q(Z|X) \ln \frac{p(X, Z|\theta)}{q(Z|X)} dZ}_{\text{ELBO, } \mathcal{L}(q,\theta)} \quad \text{by Jensen's Inequality.} && (14.3)\\
&= \int q(Z|X) \ln p(X, Z|\theta) - q(Z|X) \ln q(Z|X) dZ && (14.4)\\
&= \int q(Z|X)[\ln p(X|Z, \theta) + \ln p(Z|\theta)] - q(Z|X) \ln q(Z|X) dZ && (14.5)\\
&= \int q(Z|X) \ln p(X|Z, \theta) - q(Z|X) \ln \frac{q(Z|X)}{p(Z|\theta)} dZ && (14.6)\\
&= \mathbb{E}_{q(Z|X)} \ln p(X|Z, \theta) - KL(q(Z|X)||p(Z|\theta)) && (14.7)
\end{aligned}$$

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

14.4.1 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:

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

14.4.2 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

14.4.3 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 15

Hidden Markov Models

15.1 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: $p(x_i|z_1, \ldots, z_i, \ldots, z_T, x_1, \ldots, x_i, \ldots, x_T) = p(x_i|z_i)$

15.1.1 Conditional Independence

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

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

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

15.2 Bayesian Network

15.2.1 Bayes Ball

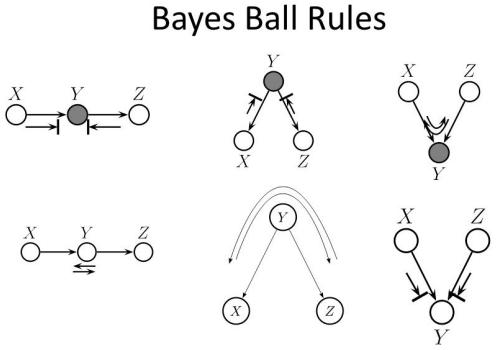


Figure 15.1: Bayes ball

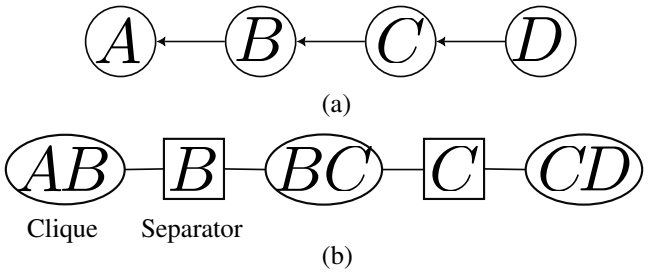
- Cascading: $P(Z|Y,X) = P(Z|Y)$. The information of Y decouples X and Z.
- Common parent: $P(X,Z|Y) = P(X|Y)P(Z|Y)$. The information of Y decouples X and Z.
- V-Structure (common child): Unlike the above two cases, the information of Y couples X and Z.

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

15.2.2 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, $\Psi^*(b,c) = \Psi(b,c)\frac{\phi^*(b)}{\phi(b)}$.

15.3 Hidden Markov Models

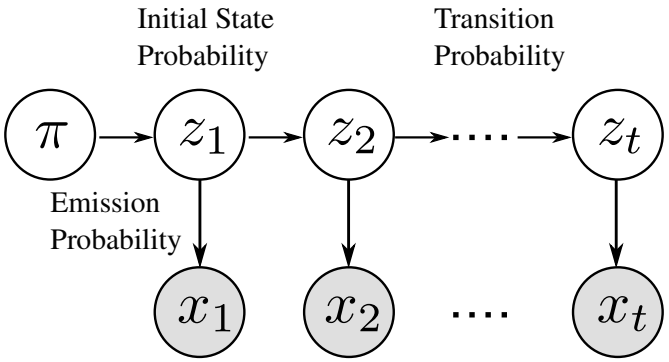


Figure 15.2: HMM Structure

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?

15.4 Evaluation: Forward-Backward Probability

15.4.1 Joint Probability

We can factorize the joint distribution of HMM in Fig. 15.2 by using a Bayesian approach as follows:.

$$p(X,Z) = p(x_1,\ldots,x_t,z_1,\ldots,z_t) = p(z_1)p(x_1|z_1), p(z_2|z_1),\ldots,p(x_t|z_t), p(z_t|z_{t-1}) \tag{15.1}$$

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

15.4.2 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. (15.1)). The computation of this equation requires lots of computations, so we will change it into a **recursive form** by using the factorization rule $p(a,b,c) = p(a)p(b|a)p(c|a,b)$.

$$p(x_1,\ldots,x_t,z_t^k=1) = \sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},x_t,z_{t-1},z_t^k=1) \tag{15.2}$$

$$= \sum_{z_{t-1}} p(\underbrace{x_1,\ldots,x_{t-1},z_{t-1}}_{a},\underbrace{x_t}_{c},\underbrace{z_t^k=1}_{b}) \tag{15.3}$$

$$= \sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})p(z_t^k=1|x_1,\ldots,x_{t-1},z_{t-1})p(x_t|z_t^k=1,x_1,\ldots,x_{t-1},z_{t-1}) \tag{15.4}$$

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

$$= \sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})p(z_t^k=1|z_{t-1})p(x_t|z_t^k=1) \tag{15.5}$$

$$= p(x_t|z_t^k=1)\sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})p(z_t^k=1|z_{t-1}) \tag{15.6}$$

$$= b_{z_t^k,x_t}\sum_{z_{t-1}} p(x_1,\ldots,x_{t-1},z_{t-1})a_{z_{t-1},z_t^k} \tag{15.7}$$

- 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**.

15.4.3 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.

15.4.4 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.

15.5 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:

$$
\begin{aligned}
V_t^k &= \max_{z_1,\ldots,z_{t-1}} p(x_1, \ldots, x_{t-1}, z_1, \ldots, z_{t-1}, x_t, z_t^k = 1) && (15.8)\\
&= \max_{z_1,\ldots,z_{t-1}} p(x_t, z_t^k = 1|x_1, \ldots, x_{t-1}, z_1, \ldots, z_{t-1}) p(x_1, \ldots, x_{t-1}, z_1, \ldots, z_{t-1}) && (15.9)\\
&= \max_{z_1,\ldots,z_{t-1}} p(x_t, z_t^k = 1|z_{t-1}) p(x_1, \ldots, x_{t-2}, z_1, \ldots, z_{t-2}, x_{t-1}, z_{t-1}) && (15.10)\\
&= \max_{z_{t-1}} p(x_t, z_t^k = 1|z_{t-1}) \max_{z_1,\ldots,z_{t-2}} p(x_1, \ldots, x_{t-2}, z_1, \ldots, z_{t-2}, x_{t-1}, z_{t-1}) && (15.11)\\
&= \max_{i \in z_{t-1}} p(x_t, z_t^k = 1|z_{t-1}^i = 1) V_{t-1}^i && (15.12)\\
&= \max_{i \in z_{t-1}} p(x_t|z_t^k = 1) p(z_t^k = 1|z_{t-1}^i = 1) V_{t-1}^i && (15.13)\\
&= p(x_t|z_t^k = 1) \max_{i \in z_{t-1}} p(z_t^k = 1|z_{t-1}^i = 1) V_{t-1}^i && (15.14)\\
&= b_{k,x_t} \max_{i \in z_{t-1}} a_{i,k} V_{t-1}^i && (15.15)
\end{aligned}
$$

- 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 $\rightarrow \log V$.

15.6 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!

15.6.1 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.

15.7 Python Implementation

15.7.1 Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm used to determine the most probable sequence of hidden states in a Hidden Markov Model (HMM) based on a sequence of observations.

The algorithm works by recursively computing the probability of the most likely sequence of hidden states that ends in each state for each observation.

At each time step, the algorithm computes the probability of being in each state and emits the current observation based on the probabilities of being in the previous states and making a transition to the current state.

Assuming we have an HMM with N hidden states and T observations, the Viterbi algorithm can be summarized as follows:

Initialization: At time t=1, we set the probability of the most likely path ending in state i for each state i to the product of the initial state probability pi and the emission probability of the first observation given state i. This is denoted by: delta[1,i] = pi * b[i,1]. Recursion: For each time step t from 2 to T, and for each state i, we compute the probability of the most likely path ending in state i at time t by considering all possible paths that could have led to state i. This probability is given by:

delta[t,i] = $\max_j(delta[t-1,j] * a[j,i] * b[i,t])$

Here, a[j,i] is the probability of transitioning from state j to state i, and b[i,t] is the probability of observing the t-th observation given state I.

We also keep track of the most likely previous state that led to the current state i, which is given by:

psi[t,i] = $\text{argmax}_j(delta[t-1,j] * a[j,i])$

- Termination: The probability of the most likely path overall is given by the maximum of the probabilities of the most likely paths ending in each state at time T. That is, P* = $\max_i(delta[T,i]).Backtracking : Starting from the state i * that gave the maximum probability at time T, we recurs$ $1 to obtain the most likely path of hidden states.$

The Viterbi algorithm is an efficient and powerful tool that can handle long sequences of observations using dynamic programming.

15.8 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 16

Explicit Generative Models

16.1 Variational Autoencoder

Our goal is to find the data distribution $p(X)$. Fig. 16.1 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.

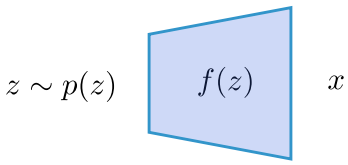


Figure 16.1: General structure of deep generative models. This model does not infer z from x.

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.

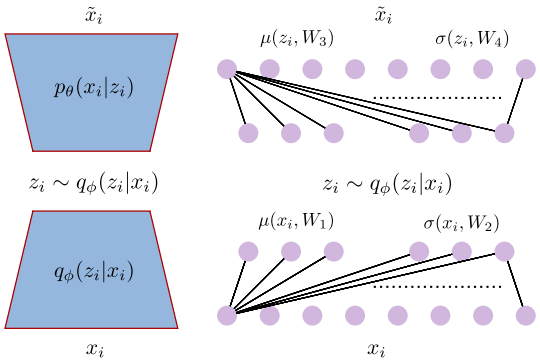


Figure 16.2: Overview of variational autoencoder.

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

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.

16.1.1 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

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

16.1.2 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,

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

$$\log p(X|Y) = \ln \int q(Z|X, Y) \frac{p(X, Z|Y)}{q(Z|X, Y)} dZ \tag{16.1}$$

$$\geq \underbrace{\int q(Z|X, Y) \ln \frac{p(X, Z|Y)}{q(Z|X, Y)} dZ}_{\text{ELBO, } \mathcal{L}(q,\theta)} \quad \text{by Jensen's Inequality.} \tag{16.2}$$

$$\ldots \tag{16.3}$$

$$\ldots \tag{16.4}$$

$$= \mathbb{E}_{q(Z|X,Y)}[\ln p(X|Z, Y)] - KL(q(Z|X, Y)||p(Z|Y)) \tag{16.5}$$

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.

16.1.3 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:

$$\log p(X) = \ln \int \sum_c p(X, Z, C) dz \tag{16.6}$$

$$\geq \underbrace{\int q(Z, C|X) \ln \frac{p(X, Z, C)}{q(Z, C|X)} dZ}_{\text{ELBO}} \tag{16.7}$$

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

16.1.4 Importance Weighted VAE

Chapter 17

Implicit Generative Models

17.1 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)

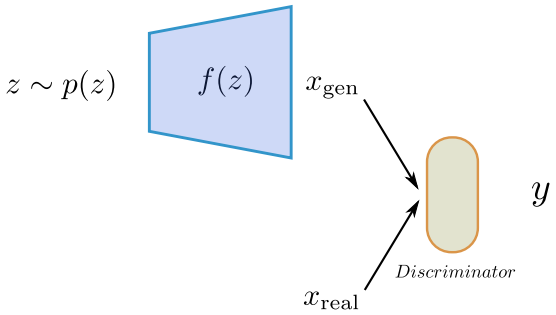


Figure 17.1: GAN structure

17.1.1 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

17.1.2 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.

17.2 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 17.2: 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.

17.3 Wasserstein Generative Adversarial Networks

17.3.1 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.

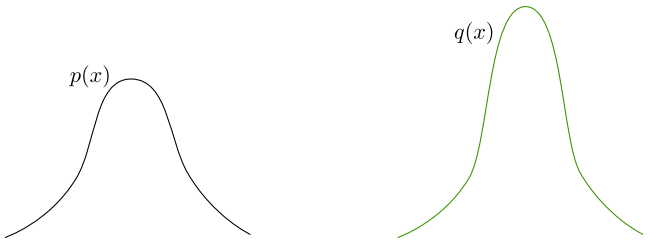


Figure 17.3: Two distributions: $p(x)$ and $q(x)$

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.

17.3.2 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. 17.46

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

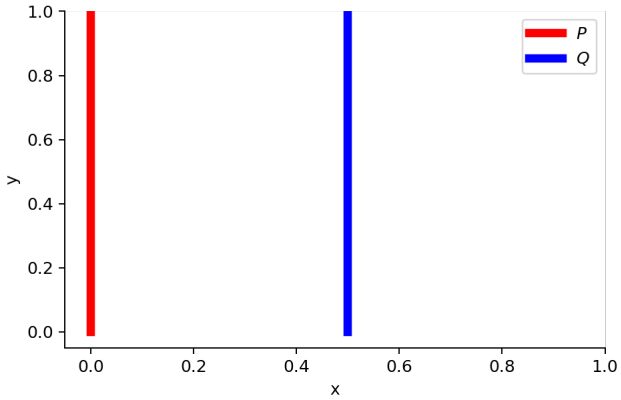


Figure 17.4: Two distributions: $p(x)$ and $q(x)$

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

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.

17.3.3 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).

17.4 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

17.4.1 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.

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 $f(x) = |x|$

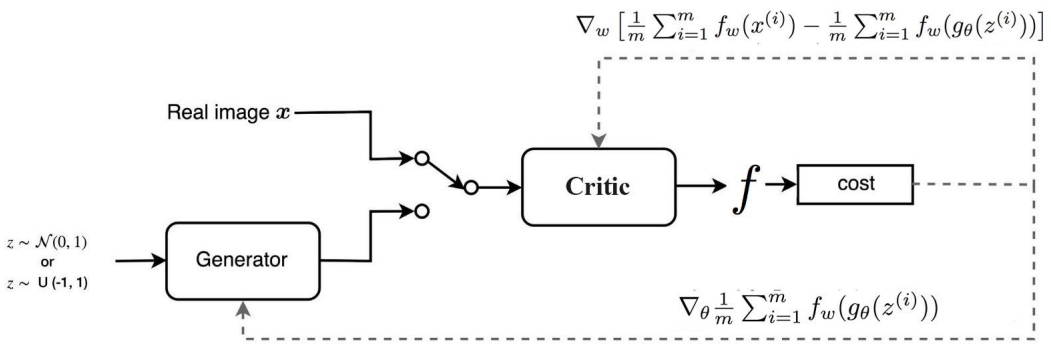


Figure 17.5: WGAN

	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 17.6: WGAN

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

17.5.1 Joint Entropy

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

17.5.2 Conditional Entropy

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

17.5.3 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 1,

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

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

lemma 1 *For random variables X, Y and function $f(x,y)$ under suitable regularity conditions:*

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

proof 1

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

Chapter 18

Diffusion Model

18.1 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.

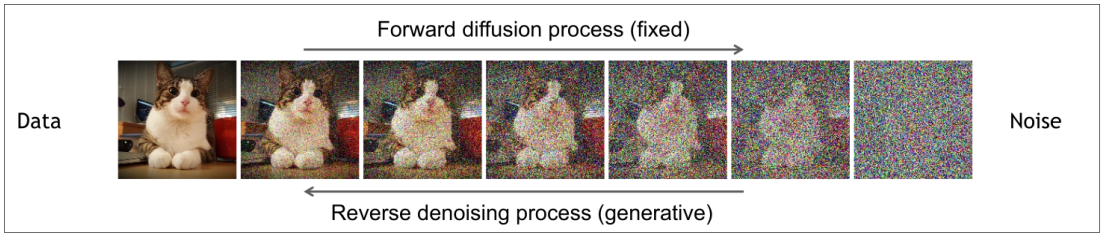
$$\mathbf{x}_0 \to \mathbf{x}_1 \cdots \to \mathbf{x}_T$$

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

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

18.2 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:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{18.1}$$

- $\beta_t \in (0, 1)$ 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:

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t} \odot \epsilon_{t-1} && (18.2)\\
&= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t} \odot \epsilon_{t-1} && (18.3)\\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}} \odot \epsilon_{t-2}\right) + \sqrt{1-\alpha_t} \odot \epsilon_{t-1} && (18.4)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1}} \odot \epsilon_{t-2} + \sqrt{1-\alpha_t} \odot \epsilon_{t-1} && (18.5)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1} + 1 - \alpha_t} \odot \epsilon_{t-2} && (18.6)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1 - \alpha_t\alpha_{t-1}} \odot \epsilon_{t-2} && (18.7)\\
&= \dots && (18.8)\\
&= \sqrt{\prod_t \alpha_t}\mathbf{x}_0 + \sqrt{1 - \prod_t \alpha_t} \odot \epsilon_{t_0} && (18.9)\\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \odot \epsilon && (18.10)\\
&\sim \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}), && (18.11)
\end{aligned}
$$

  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.

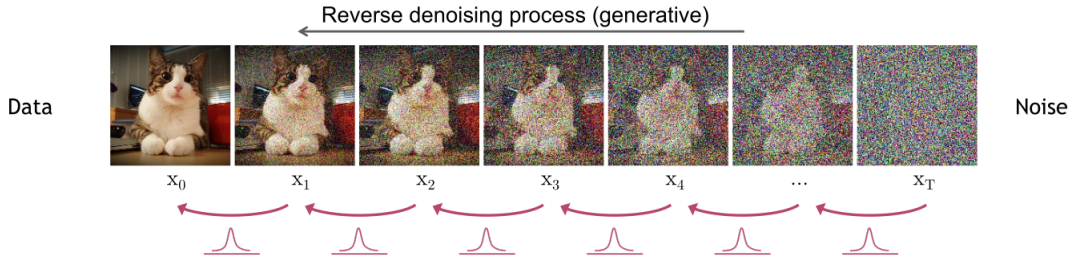
18.3 Backward Process

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. (18.45)).
  - 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. (18.45). 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 $\mathbf{x}_0$.

18.4 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:

$$\log p_\theta(\mathbf{x}_0) = \log \int p(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \tag{18.12}$$

$$= \log \int p(\mathbf{x}_{0:T}) \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{18.13}$$

$$= \log \int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{18.14}$$

$$\geq \int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{18.15}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \quad \rightarrow \text{ELBO} \tag{18.16}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.17}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) p(\mathbf{x}_0|\mathbf{x}_1) \prod_{t=1}^{T-1} p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_T|\mathbf{x}_{T-1}) \prod_{t=1}^{T-1} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.18}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \prod_{t=1}^{T-1} \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.19}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} [\log p(\mathbf{x}_0|\mathbf{x}_1)] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \tag{18.20}$$

$$+ \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\sum_{t=1}^{T-1} \log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.21}$$

$$= \cdots + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.22}$$

$$= \cdots + \int_{x_1} \cdots \int_{x_T} q(\mathbf{x}_1, \ldots, \mathbf{x}_T) \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] dx_1 \ldots dx_T + \ldots \tag{18.23}$$

$$= \cdots + \int_{x_T} \int_{x_{T-1}} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \cdots \int_{x_1} q(\mathbf{x}_1, \ldots, \mathbf{x}_T) dx_1 \ldots dx_T + \ldots \tag{18.24}$$

$$= \cdots + \int_{x_T} \int_{x_{T-1}} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] q(\mathbf{x}_t, \mathbf{x}_{t-1}) dx_{T-1} dx_T + \ldots \tag{18.25}$$

$$= \cdots + \mathbb{E}_{q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.26}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} [\log p(\mathbf{x}_0|\mathbf{x}_1)] + \mathbb{E}_{q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \tag{18.27}$$

$$+ \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \tag{18.28}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} [\log p(\mathbf{x}_0|\mathbf{x}_1)] - \mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} [D_{KL}(q(\mathbf{x}_T|\mathbf{x}_{T-1}) \| p(\mathbf{x}_T))] \tag{18.29}$$

$$- \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_{t+1}|\mathbf{x}_0)} D_{KL}[q(\mathbf{x}_t|\mathbf{x}_{t-1}) \| p(\mathbf{x}_t|\mathbf{x}_{t+1})] \tag{18.30}$$

$$\because q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0) = \frac{q(\mathbf{x}_t, \mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_0)} = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) q(\mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_0)} = q(\mathbf{x}_t|\mathbf{x}_{t-1}) q(\mathbf{x}_{t-1}|\mathbf{x}_0) \tag{18.31}$$

- 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 x_t 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. (18.1).
- 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:

$$L = \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \quad \rightarrow \text{ELBO} \tag{18.32}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \tag{18.33}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)\prod_{t=2}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_1|\mathbf{x}_0)\prod_{t=2}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \tag{18.34}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)\prod_{t=2}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_1|\mathbf{x}_0)\prod_{t=2}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}\right] \quad \text{by Markov Property} \tag{18.35}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}\right] \tag{18.36}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}}\right] \tag{18.37}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{q(\mathbf{x}_T|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \tag{18.38}$$

$$= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)p(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_T|\mathbf{x}_0)} + \log \prod_{t=2}^{T} \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \tag{18.39}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p(\mathbf{x}_0|\mathbf{x}_1)] + \mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \tag{18.40}$$

$$= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p(\mathbf{x}_0|\mathbf{x}_1)] - D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T)) - \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p(\mathbf{x}_{t-1}|\mathbf{x}_t))] \tag{18.41}$$

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.
  - $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as an approximation to tractable, ground-truth denoising transition step $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$. The $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ 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.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \tag{18.42}$$

$$= \frac{\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_{t-1}, (1-\bar{\alpha}_t)\mathbf{I})\mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I})}{\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})} \tag{18.43}$$

$$\vdots \tag{18.44}$$

$$\propto \mathcal{N}\Bigg(\mathbf{x}_{t-1}; \underbrace{\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t}}_{\mu_q(\mathbf{x}_t, \mathbf{x}_0)}, \underbrace{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{I}}_{\Sigma_q(t)}\Bigg) \tag{18.45}$$

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. (18.45) 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. (18.11):

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

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

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

$$\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t} \tag{18.46}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\odot\epsilon}{\sqrt{\bar{\alpha}_t}}}{1-\bar{\alpha}_t} \tag{18.47}$$

$$\vdots \tag{18.48}$$

$$= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}_0 \tag{18.49}$$

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. (18.45) instead of estimating it from a network for simplicity. Finally, we can solve the KL-divergence term:

$$\mathcal{L} = \min_{\theta} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \tag{18.50}$$

$$\vdots \tag{18.51}$$

$$= \min_{\theta} \frac{1}{2\sigma_q^2(t)} \frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)\alpha_t} \left[\|\boldsymbol{\epsilon}_0 - \hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t)\|_2^2\right]. \tag{18.52}$$

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.

18.5 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

18.6 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.

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

18.6.1 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 $p(\mathbf{x})$

$$\mathcal{L}_\theta = \frac{1}{2}\mathbb{E}_{p(x)}\left[||\nabla_x \log p(x) - s_\theta(x)||_2^2\right] \tag{18.53}$$
$$= \frac{1}{2}\mathbb{E}_{p(x)}\left[\left(\nabla_x \log p(x) - s_\theta(x)\right)^2\right] \tag{18.54}$$
$$= \frac{1}{2}\int p(x)(\nabla_x \log p(x) - s_\theta(x))^2 dx \tag{18.55}$$
$$= \underbrace{\frac{1}{2}\int p(x)(\nabla_x \log p(x))^2 dx}_{\text{independent from theta}} + \frac{1}{2}\int p(x) s_\theta(x)^2 dx - \int p(x) s_\theta(x) \nabla_x \log p(x) dx \tag{18.56}$$
$$= \cdots - \int p(x) s_\theta(x) \nabla_x \log p(x) dx \tag{18.57}$$
$$= \cdots - \int p(x) s_\theta(x) \frac{\nabla_x p(\mathbf{x})}{p(x)} dx \tag{18.58}$$
$$= \cdots - \int \nabla_{\mathbf{x}} p(x) s_\theta(x) dx \tag{18.59}$$
$$= \cdots - \left[p(x) s_\theta(x)\right]_{x=-\infty}^{\infty} + \int p(x) \nabla_x s_\theta(x) dx \tag{18.60}$$
$$= \frac{1}{2}\int p(x) s_\theta(x)^2 dx + \int p(x) \nabla_x s_\theta(x) dx + \text{const} \tag{18.61}$$
$$= \frac{1}{2}\mathbb{E}_{p(x)}[s_\theta(x)^2] + \mathbb{E}_{p(x)}[\nabla_x s_\theta(x)] + \text{const.} \tag{18.62}$$

- 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.

18.6.2 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 IV

Natural Language Processing

Chapter 19

Introduction

19.1 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:

19.1.1 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:

$$\begin{aligned}
\mathcal{L}_{CE} &= -\mathbb{E}_{w\sim P}[P_\theta(w_i|w_{<i})] && (19.1)\\
&\approx -\frac{1}{N}\sum_{i=1}^{N}\log P_\theta(w_i|w_{<i}) && (19.2)\\
&= -\frac{1}{N}\log\prod_{i=1}^{N}P_\theta(w_i|w_{<i}) && (19.3)\\
&= \log\left(\prod_{i=1}^{N}P_\theta(w_i|w_{<i})\right)^{-\frac{1}{N}} && (19.4)\\
&= \log\sqrt[N]{\frac{1}{\prod_{i=1}^{N}P_\theta(w_i|w_{<i})}} && (19.5)\\
& && (19.6)
\end{aligned}$$

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

19.1.2 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 20

Classical NLP Techniques

20.1 Edit Distance

Edit distance is a method used in spell correction to determine how similar two words are by calculating the minimum number of operations (insertions, deletions, or substitutions) required to transform one word into another. The smaller the edit distance, the more similar the two words are.

For example, let's say we have the following dictionary of valid words: "cat", "car", "cart", "care", "cards", "cast". If the input word is "carr", we calculate the edit distance between "carr" and each word in the dictionary as follows:

- cat: 3 (insert "r" and "r", then delete "t")
- car: 1 (replace second "r" with "t")
- cart: 2 (insert "t" and delete second "r")

The smallest edit distance is 1, between "carr" and "car", so "car" would be selected as the corrected word.

The edit distance method can be improved by using techniques such as weighting the importance of each operation (insertion, deletion, substitution) or using a more sophisticated algorithm such as *Levenshtein distance*. Additionally, the method can be combined with other methods such as language modeling or phonetic analysis to further improve spell correction accuracy.

20.2 Point-wise Mutual Information

Point-wise Mutual Information (PMI) is a statistical measure to calculate the association between two words in a given corpus. PMI is calculated by comparing the probability of the co-occurrence of two words with their individual probabilities of occurrence.

Formally, it is a quantity which is closely related to the mutual information is the point-wise

mutual information. For two events (not random variables) x and y, this is defined as

$$\text{PMI}[x, y] \triangleq \log \frac{p(x, y)}{p(x)p(y)} \tag{20.1}$$

$$= \frac{p(x|y)}{p(x)} = \frac{p(y|x)}{p(y)} \tag{20.2}$$

This measures the discrepancy between these events occurring together compared to what would be expected by chance.

- x and y are two words being considered,
- $P(x)$ is the probability of the occurrence of word x in the corpus,
- $P(y)$ is the probability of the occurrence of word y in the corpus, and
- $P(x, y)$ is the probability of the co-occurrence of words x and y in the corpus. In other words, x and y are adjacent

For terms with three words, the formula becomes:

$$\text{PMI}[x, y, z] = \log \frac{p(x, y, z)}{p(x)p(y)p(z)}$$

PMI values can range from $-\infty$ to ∞. Positive PMI values indicate that the words have a strong association, while negative values indicate that the words are unlikely to appear together.

For example, consider a small corpus of text:

- "The cat sat on the mat. The dog sat on the mat."
- The matrix below is 6×6 considering all possible combination of the "forward" co-occurrences.

$$\begin{bmatrix} & the & cat & dog & sat & on & mat \\ the & 0 & 1 & 1 & 0 & 0 & 2 \\ cat & 0 & 0 & 0 & 1 & 0 & 0 \\ dog & 0 & 0 & 0 & 1 & 0 & 0 \\ sat & 0 & 0 & 0 & 0 & 2 & 0 \\ on & 2 & 0 & 0 & 0 & 0 & 0 \\ mat & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

20.2.1 Remove Stopwords prior to PMI

In the above example we have not removed stopwords, so some of you might be wondering if we need to remove stopwords prior to PMI. It depends on the problem statement but if your objective is to find the related words, you should remove stopwords prior to calculating PMI.

20.3 TF-IDF

The term TF stands for term frequency, and the term IDF stands for inverse document frequency.

The TF-IDF representation takes into account the importance of each word in a document. In the bag-of-words model, each word is assumed to be equally important, which is obviously a less accurate assumption.

The method to calculate the TF-IDF weights of a term in a document is given by the following formula:

- Term Frequency (TF), $tf(t,d)$, is the relative frequency of term t within document d,

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

  where $f_{t,d}$ is the raw count of a term in a document, *i.e.,*, the number of times that term t occurs in document d. Note the denominator is simply the total number of terms in document d (counting each occurrence of the same term separately). There are various other ways to define term frequency:

  - The raw count itself: $tf(t,d) = f_{t,d}$
  - Boolean "frequencies":

$$tf(t,d) = \begin{cases} 1, & \text{if } t \text{ occurs in } d \\ 0 & \text{otherwise} \end{cases}$$

  - Logarithmically scaled frequency: $tf(t,d) = \log(1 + ft, d)$
  - Augmented frequency, to prevent a bias towards longer documents, *e.g.,*. raw frequency divided by the raw frequency of the most frequently occurring term in the document:

$$tf(t,d) = 0.5 + 0.5\frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

- Inverse document frequency: The IDF is a measure of how much information the word provides, *i.e.,*, if it is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, which is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient:

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

  - N total number of documents in the corpus ($= |D|$).
  - $|\{d \in D : t \in d\}|$: number of documents where the term t appears. If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the numerator $1 + N$ and denominator to $1 + |\{d \in D : t \in d\}|$.
  -

20.3.1 Term frequency–inverse document frequency

TF-IDF is calculated as a multiplication of $tf(t, D)$ and $idf(t, D)$.

- A high weight in TF–IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms.
- Since the ratio inside the IDF's log function is always greater than or equal to 1, the value of IDF (and TF–IDF) is greater than or equal to 0.
- As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the IDF and TF–IDF closer to 0.

20.3.2 Python Implementation

20.3.3 Link with Information Theory

This expression shows that summing the TF–IDF of all possible terms and documents recovers the mutual information between documents and term taking into account all the specificities of their joint distribution. Each TF–IDF hence carries the "bit of information" attached to a term x document pair.

20.4 BM25

20.5 Label Smoothing

For each training example x, our model computes the probability of each label $k \in \{1, ..., K\}$, $p(k|x) = \frac{\exp(z_k)}{\sum_i \exp(z_i)}$. Here z_k are the logits.

Label smoothing is a mechanism to regularize the classifier layer by estimating the marginalized effect of label-dropout during training.

Vanila corss-entropy can cause two problem:

- First, it may result in over-fitting: if the model learns to assign full probability to the ground-truth label for each training example, it is not guaranteed to generalize.
- Second, it encourages the differences between the largest logit and all others to become large, and this, combined with the bounded gradient $\frac{\partial \ell}{\partial z_k}$, reduces the ability of the model to adapt. Intuitively, this happens because the model becomes too confident about its predictions.

We propose a mechanism for encouraging the model to be less confident. While this may not be desired if the goal is to maximize the log-likelihood of training labels, it does regularize the model and makes it more adaptable. The method is very simple. Consider a distribution over

labels $u(k)$, independent of the training example x, and a smoothing parameter ϵ. For a training example with ground-truth label y, we replace the label distribution $q(k|x) = \delta_{k,y}$ with

$$q'(k|x) = (1 - \epsilon)\delta_{k,y} + \epsilon u(k)$$

which is a mixture of the original ground-truth distribution $q(k|x)$ and the fixed distribution $u(k)$, with weights $1 - \epsilon$ and ϵ, respectively. This can be seen as the distribution of the label k obtained as follows:

$$q'(k|x) = (1 - \epsilon)\delta_{k,y} + \frac{\epsilon}{K}$$

We refer to this change in ground-truth label distribution as label-smoothing regularization, or LSR.

20.5.1 Another Interpretation

Instead of using one-hot encoded vector, we introduce noise distribution $u(y|x)$. Our new ground truth label for data (x_i, y_i) would be

$$\begin{aligned} p'(y|x_i) &= (1 - \varepsilon)p(y|x_i) + \varepsilon u(y|x_i) \\ &= \begin{cases} 1 - \varepsilon + \varepsilon u(y|x_i) & \text{if } y = y_i \\ \varepsilon u(y|x_i) & \text{otherwise} \end{cases} \end{aligned}$$

Where ε is a weight factor, $\varepsilon \in [0, 1]$, and note that $\sum_{y=1}^{K} p'(y|x_i) = 1$.

Chapter 21

POS Tagging

21.1 Introduction

Part-of-speech (POS) tagging is the process of labeling words in a text with their corresponding parts of speech in natural language processing (NLP). It helps algorithms understand the grammatical structure and meaning of a text.

Chapter 22

Transformer

22.1 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.

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

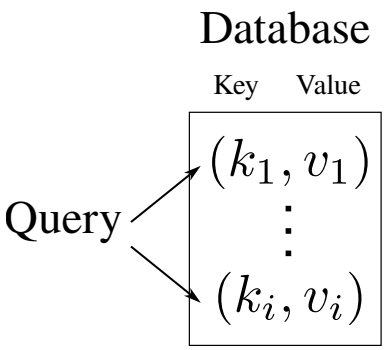


Figure 22.1: The most similar key will be selected by measuring a **similarity** between a query and a key.

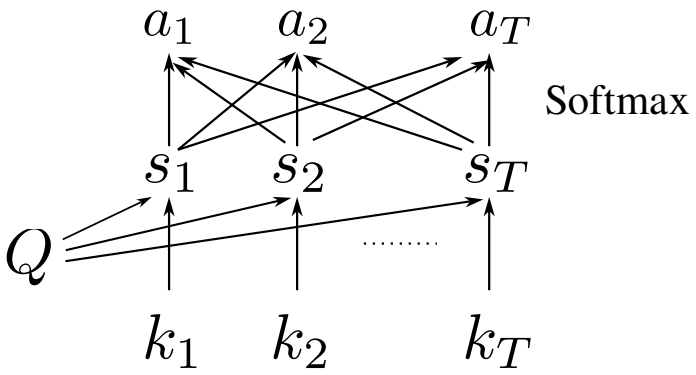


Figure 22.2: The similarity s_t is computed by a query and keys

There are several choices for a similarity function.

- $q^T k_i$: dot product.
- $\frac{q^T k_i}{\sqrt{d}}$: scaled dot product $\rightarrow$ reduce the variance of the final attention weights.
- $q^T W k_i$: 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:

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

22.2 Transformer

22.2.1 Self-Attention

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

22.2.2 Masked attention

The masked attention is often referred to cross-attention. This is just a self-attention in decoder.

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

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

22.2.3 Skip Connection

This is a regularization technique.

22.2.4 Positional Embedding

```
def position_encoding(seq_len: int, dim_model: int)->Tensor:
    pos = torch.arange(seq_len, dtype=torch.float).reshape(1, -1, 1)
    dim = torch.arange(dim_model, dtype=torch.float).reshape(1, 1, -1)
    phase = pos / (1e4 ** (dim // dim_model))
    return torch.where(dim.long() % 2 == 0, torch.sin(phase), torch.cos(phase))
```

22.2.5 Encoder

22.2.6 Decoder

The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. And just like we did with the encoder

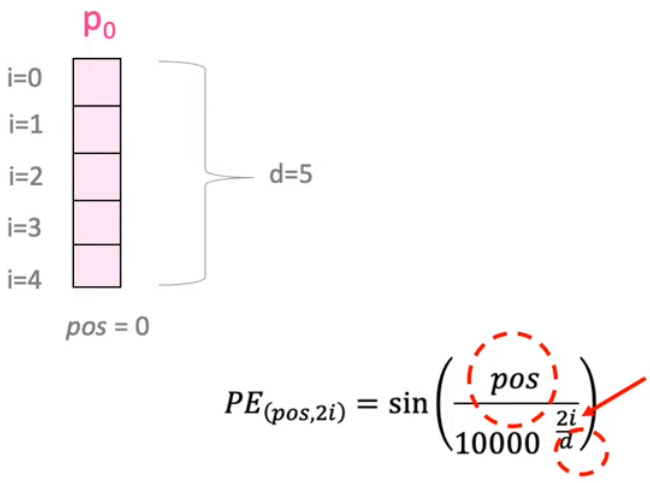


Figure 22.3: Positional embedding.

inputs, we embed and add positional encoding to those decoder inputs to indicate the position of each word.

```
def forward(self, tgt: Tensor, memory: Tensor) -> Tensor:
  seq_len, dimension = tgt.size(1), tgt.size(2)
  tgt += position_encoding(seq_len, dimension)
  for layer in self.layers:
    tgt = layer(tgt, memory)
  return torch.softmax(self.linear(tgt), dim=-1)
```

Part V

Advanced Topics

Chapter 23

Neural Ordinary Differential Equations

23.1 Neural Ordinary Differential Equations

Euler method can be expressed as follows:

$$y_n = y_{n-1} + h\frac{\partial y_{n-1}}{\partial x_{n-1}},$$

where h is the step size. The function can be expressed in an iterative form:

$$y_n = y_1 + h\frac{\partial y_1}{\partial x_1} + h\frac{\partial y_2}{\partial x_2} + \cdots + h\frac{\partial y_{n-1}}{\partial x_{n-1}}$$

Residual function is given by

$$h_{t+1} = h_t + f(h_t, \theta).$$

The h is iteratively updated as follows:

$$\begin{aligned} h_2 &= h_1 + f(h_1, \theta) \\ h_3 &= h_2 + f(h_2, \theta) = h_1 + f(h_1, \theta) + f(h_2, \theta) \\ &\vdots \end{aligned}$$

These iterative update can be seen as *discretization* of the *continuous* transformation of Euler method as the residual function is done by discrete steps. Note that this is the key contribution of this approach.

Chapter 24

State Space Model

24.1 Kalman Filter

Ref: Kalman Filter Tutorial

24.2 Introduction to State-Space Model

Reference: State-Space Models.

The state of a dynamic system is a set of physical quantities, the specification of which completely determines the evolution of the system. The state variables are minimum set of variables that fully describe (enough information) the system.

Many dynamic systems can be represented by using differential equations, since how the systems are changing is actually a function of current state.

$$\dot{x} = Ax + Bu,$$
$$y = Cx + Du.$$

The first and the second equations are known as *state equation* and *output equation*, respectively. The state equation tells us that how the state vector changes with the state vector and the (external) input.

- $x \in \mathbb{R}^n$: A state vector.
- $\dot{x} \in \mathbb{R}^n$: state derivative *i.e.,* $\left(\frac{dX}{dt}\right)$ represents the changes of the state vector. You can notice that this is a linear combination of state vector and the input vector.
- $u \in \mathbb{R}^m$: Input
- $y \in \mathbb{R}^p$: Output
- $A \in \mathbb{R}^{n \times n}$: This matrix describes how the state vector influences the changes of the state vector.
- $B \in \mathbb{R}^{n \times m}$: This matrix describes how the (external) input vector influences the changes of the state vector.

- $C \in \mathbb{R}^{p \times n}$: Typically, an identity matrix (I).
- $D \in \mathbb{R}^{p \times m}$: Typically, zeros

24.2.1 Stability

The linear state space model is stable if all eigenvalues of A are negative real numbers or have negative real parts to complex number eigenvalues. If all real parts of the eigenvalues are negative then the system is stable, meaning that any initial condition converges exponentially to a stable attracting point. If any real parts are zero then the system will not converge to a point and if the eigenvalues are positive the system is unstable and will exponentially diverge.

24.2.2 First Order System in State Space

Let's consider an example to transform a first order linear system (without time delay):

$$\tau_p \frac{dy}{dt} = -y + K_p u,$$

where τ_p and K_p are time constant and gain, respectively. Then, we can transform it into a state space form as follows:

$$\begin{aligned} \dot{x} &= \left[-\frac{1}{\tau_p} \right] x + \left[\frac{K_p}{\tau_p} \right] u, \\ y &= [1]x + [0]u \\ &= x. \end{aligned}$$

Here, $A = -\frac{1}{\tau_p}$, $B = \frac{K_p}{\tau_p}$, $C = 1$, and $D = 0$. We can solve such model by using various methods like Laplace transform.

24.3 Efficiently Modeling Long Sequences with Structured State-Spaces

The state space model (SSM) can be defined as follows:

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t), \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t). \end{aligned}$$

It maps a single dimensional input signal $u(t)$ to an N-dim latent state $x(t)$ before projecting to a one-dim output signal $y(t)$.

Our goal is to simply use the SSM as a black-box representation in a deep sequence model, where $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$ are parameters learned by gradient descent. We will omit the parameter $\mathbf{D}$ for exposition (or equivalently, assume $\mathbf{D} = 0$, because the term $\mathbf{D}u$ can be viewed as a skip connection and is easy to compute).

An SSM maps a input $u(t)$ to a state representation vector $x(t)$ and an output $y(t)$. For simplicity, we assume the input and output are one-dimensional, and the state representation is N-dimensional. The first equation defines the change in $x(t)$ over time.

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.