Probabilistic Machine Learning

Derek Li

Contents

1 Introduction to Probabilistic Model

1.1 Overview of Probabilistic Model

We have random variables $X = (X_1, \cdots, X_N)$, and we want a model that captures the relationship between these variables. The approach of probabilistic generative models is to relate all variables by a learned joint probability distribution $p_\theta(X_1, \cdots, X_n)$.

Now let X be input data, C be discrete outputs (labels), Y be continuous outputs. We can use it for machine learning tasks:

- Regression:
$$p(Y|X) = \frac{p(X,Y)}{p(X)} = \frac{p(X,Y)}{\int p(X,Y)\mathrm{d}Y}.$$
- Classification/Clustering:
$$p(C|X) = \frac{p(X,C)}{\sum_C p(X,C)}.$$

The distinction between classification and clustering, or in general supervised and unsupervised learning, in probabilistic perspective is given by whether a random variable is observed or unobserved.

- Supervised dataset: $\{x_i, c_i\}_{i=1}^N \sim p(X,C)$.
- Unsupervised dataset: $\{x_i\}_{i=1}^N \sim p(X,C)$.
- Semi-Supervised learning: Variables are observed for some, but not all.

Like clusters, introducing assumptions about unobserved variables is a powerful modeling tool. We say a latent variable is never observed in the dataset. By introducing and modeling latent variables, we will be able to naturally describe and capture abstract features of are input data.

1.1.1 Operations on Probabilistic Model

The fundamental operations we will perform are:

- Generate data: We need know how to sample from the model.
- Estimate likelihood: When all variables are either observed or marginalized the result is a single real number which is the probability of the all variables taking on those specific values.
- Inference.
- Learning.

1.1.2 Desiderata of Probabilistic Model

We have two main goals for our joint distributions:

- Operations are efficient.
- p_θ is compact to represent.

1.2 Likelihood Function

We define log likelihood function as

$$l(\theta; x) = \ln(p(x|\theta)),$$

where x is fixed.

The process of learning is choosing θ to minimize some cost or loss function, $L(\theta)$ which includes $l(\theta)$. This can be done in may ways, including:

- Maximum likelihood estimation (MLE): $L(\theta) = l(\theta; \mathcal{D})$.
- Maximum a posteriori (MAP): $L(\theta) = l(\theta; \mathcal{D}) + r(\theta)$.

1.2.1 Maximum Likelihood Estimation

MLE is to pick values for our parameters:

$$\widehat{\theta}_{\text{MLE}} = \arg\max_{\theta} l(\theta; \mathcal{D}).$$

For i.i.d. data,

$$p(\mathcal{D}|\theta) = \prod_m p(x^{(m)}|\theta),$$

and

$$l(\theta; \mathcal{D}) = \sum_m \ln(p(x^{(m)}|\theta)).$$

Example 1.1 (Univariate Normal)**.** The model is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and the log likelihood function is

$$l(\theta; \mathcal{D}) = \ln(p(\mathcal{D}|\theta)) = -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2}\sum_i \frac{(x_i - \mu)^2}{\sigma^2},$$

then

$$\frac{\partial l}{\partial \mu} = \frac{1}{\sigma^2}\sum_i (x_i - \mu).$$

We set $\frac{\partial l}{\partial \mu} = 0$, and have

$$\mu^*_{\text{MLE}} = \frac{\sum_i x_i}{N}.$$

1.3 Sufficient Statistics

A sufficient statistic is a statistic that conveys exactly the same information about the data generating process that created that data as the entire data itself. In other words, once we know the sufficient statistic $T(x)$, then our inferences are the same as would be obtained from our entire data. Mathematically, we say $T(X)$ is a sufficient statistic for X if

$$T(x^{(1)}) = T(x^{(2)}) \Rightarrow L(\theta; x^{(1)}) = L(\theta; x^{(2)}), \forall\theta.$$

Put another way,

$$P(\theta|T(X)) = P(\theta|X).$$

By the Neyman factorization theorem, we have

$$P(\theta|T(X)) = h(x, T(x))g(T(x), \theta).$$

Example 1.2 (Exponential Family)**.**

$$p(x|\eta) = h(x)e^{\eta^T T(x) - A(\eta)}$$

or equivalently

$$p(x|\eta) = h(x)A(\eta)e^{\eta^T T(x)}.$$

For univariate normal distribution,

$$\begin{aligned} p(x) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x^2 - 2x\mu + \mu^2)\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\mu^2}{2\sigma^2}\right) \exp\left(\begin{pmatrix} \frac{\mu}{\sigma^2} & \frac{-1}{2\sigma^2} \end{pmatrix} \begin{pmatrix} x \\ x^2 \end{pmatrix}\right). \end{aligned}$$

Thus, $\eta^T = \begin{pmatrix} \frac{\mu}{\sigma^2} & \frac{-1}{2\sigma^2} \end{pmatrix}, T(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix}$.

We can rewrite $p(x)$ in terms of η :

$$p(x|\eta) = (\sqrt{2\pi})^{-\frac{1}{2}} \cdot (-2\eta_2)^{\frac{1}{2}} \cdot \exp\left(\frac{\eta_1^2}{4\eta_2}\right) \cdot \exp(\eta^T T(x)),$$

where $h(x) = (\sqrt{2\pi})^{\frac{1}{2}}, A(\eta) = (-2\eta_2)^{\frac{1}{2}} \cdot \exp\left(\frac{\eta_1^2}{4\eta_2}\right)$.

Example 1.3 (Bernoulli Trial)**.** Suppose $X \sim$ Bernoulli(θ). The likelihood is

$$L = \prod_{i=1}^{N} \theta^{x_i}(1-\theta)^{1-x_i} = \theta^{\sum_{i=1}^N x_i}(1-\theta)^{N - \sum_{i=1}^N x_i}.$$

Note that the likelihood depends on $\sum_{i=1}^N x_i$, i.e., if we know this summary statistic, which we will call $T(x) = \sum_{i=1}^N x_i$, then essentially we know everything that is useful from our sample to do inference.

We have

$$l(\theta; X) = \ln(p(X|\theta)) = T(X)\ln(\theta) + (N - T(X))\ln(1-\theta),$$

then

$$\frac{\partial l}{\partial \theta} = \frac{T(X)}{\theta} - \frac{N - T(X)}{1-\theta}.$$

We set $\frac{\partial l}{\partial \theta} = 0$, and have

$$\theta^*_{\text{MLE}} = \frac{T(X)}{N}.$$

2 Directed Graphical Model

2.1 Decision Theory

When faced with a choice of actions, we should:

- Determine the value of all possible outcomes.
- Find the probability of each outcome under each action.
- Multiply the two to get expected value, summing over all outcomes.
- Choose the action with highest expected value.

2.2 Joint Distribution

The joint distribution of N random variables can be decomposed by the chain rule of probability:

$$p(x_1, \cdots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\cdots p(x_N|x_{N-1:1}) = \prod_{i=1}^{N} p(x_i|x_1, \cdots, x_{N-1}).$$

2.2.1 Conditional Independence

Definition 2.1. Two random variables A, B are ***conditionally independent*** given a third variable C, denoted $X_A \perp X_B|X_C$ iff

$p(X_A, X_B|X_C) = p(X_A|X_C)p(X_B|X_C) \Leftrightarrow p(X_A|X_B, X_C) = p(X_A|X_C) \Leftrightarrow p(X_B|X_A, X_C) = p(X_B|X_C)$, for all X_C.

2.3 Directed Acyclic Graphical Model

Definition 2.2. A ***directed graphical model*** implies a restricted factorization of the joint distribution. In a DAG, variables are represented by nodes, and edges represent dependence.

The meaning of any particular directed acyclic graphical model D is that

$$p(x_1, \cdots, x_N) = \prod_{i=1}^{N} p(x_i|\text{Parents}_M(x_i)),$$

where $\text{Parents}_M(x_i)$ is the set of nodes with edges pointing to x_i. In other words, the joint distribution of a DAGM factors into a product of local conditional distributions, where each node (a random variable) is conditionally dependent on its parent node(s), which could be empty.

2.3.1 Markov Chain

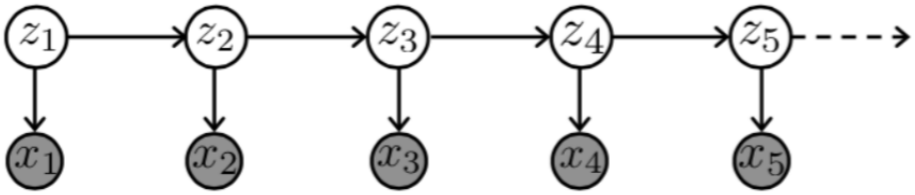
Definition 2.3. ***Markov chains*** are a stochastic model describing a sequence if possible events in which the probability of each event depends only on the state attained in the previous event.

The joint probability is

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2)\cdots.$$

We say the model satisfies the Markov property, i.e., conditional on the present state of the system, its future and past states are independent.

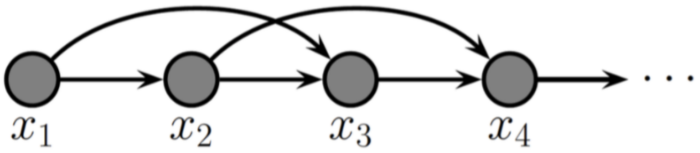
Definition 2.4. ***Hidden Markov chain*** is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) state.



z_t are hidden state taking on one of discrete values and x_t are observed variables taking on values in any space. The joint probability is

$$p(x_{1:T}, z_{1:T}) = p(z_{1:T})p(x_{1:T}|z_{1:T}) = p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\prod_{t=1}^{T} p(x_t|z_t).$$

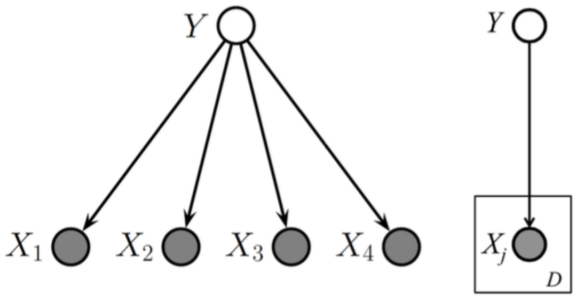
Definition 2.5. ***Second-Order Markov chain*** is a statistical Markov model in which the next state depends on two preceding ones.



The joint probability is

$$p(x_{1:T}) = p(x_1, x_2)p(x_3|x_1, x_2)\cdots = p(x_1, x_2)\prod_{t=3}^{T} p(x_t|x_{t-1}, x_{t-2}).$$

2.3.2 Plates



We can use plates where repeated quantities that are i.i.d. are put in a box. The rules of plates are simple: repeat every structure in a box a number of times given by the integer in the corner of the box, updating the plate index variable as you go, and then duplicate every arrow going into the plate and every arrow leaving the plate by connecting the arrows to each copy of the structure. We can write

$$p(x_1, \cdots, x_N|A) = \prod_{i=1}^{N} p(x_i|A).$$

Plates can be nested, in which case their arrows get duplicates also, according to the rule: draw an arrow from every copy of the source node to every copy of the destination node.

Plates can also cross (intersect), in which case the nodes at the intersection have multiple indices and get duplicated a number of times equal to the product of the duplication numbers on all the plates containing them.

2.4 D-Separation

Definition 2.6. ***D-separation*** or ***directed-separation*** is a notion of connectedness in DAGs in which two (sets of) variables may or may not be connected conditioned on a third (set of) variable(s).

D-connection implies conditional dependence and d-separation implies conditional independence.

In particular, we say $x_A \perp x_B | x_C$ if every variable in A is d-separated from every variable in B conditioned on all the variables in C.

2.4.1 Depth-First Search Algorithm

To check if an independence is true, we can cycle through each node in A, do a depth-first search to reach every node in B, and examine the path between them. If all of the paths are d-separated, then $x_A \perp x_B | x_C$.

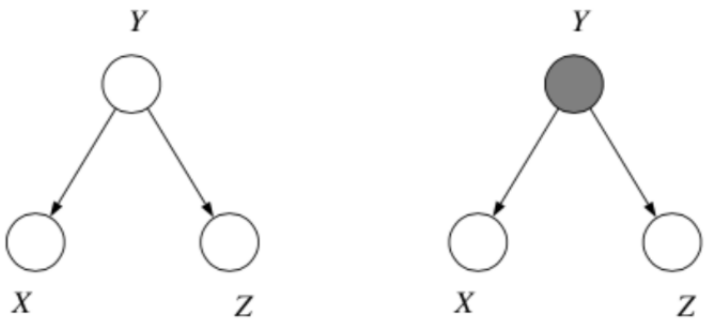
- Chain:

  * We have $p(X, Y, Z) = p(X)p(Y|X)p(Z|Y)$, which implies

$$p(X, Z|Y) = \frac{p(X, Y, Z)}{p(Y)} = \frac{p(X)p(Y|X)p(Z|Y)}{p(Y)} = p(X|Y)p(Z|Y),$$

  and thus $X \perp Z|Y$.

- Common cause:



  * We have

$$p(X|Z) = \frac{p(X, Z)}{p(Z)} = \frac{\sum_Y p(X, Y, Z)}{\sum_X \sum_Y p(X, Y, Z)} \neq p(X),$$

and thus $X \not\perp Z$.

* We have $p(X, Y, Z) = p(X|Y)p(Y)p(Z|Y)$, which implies

$$p(X, Z|Y) = \frac{p(X, Y, Z)}{p(Y)} = \frac{p(X|Y)p(Y)p(Z|Y)}{p(Y)} = p(X|Y)p(Z|Y),$$

and thus $X \perp Z|Y$.

- Explaining away:
  * We have

$$p(X|Z) = \frac{p(X, Z)}{p(Z)} = \frac{\sum_Y p(X, Y, Z)}{p(Z)} = \frac{\sum_Y p(X)p(Y|X, Z)p(Z)}{p(Z)} = p(X),$$

and thus $X \perp Z$.

* We have $p(X, Y, Z) = p(X)p(Y|X, Z)p(Z)$, which implies

$$p(X, Z|Y) = \frac{p(X, Y, Z)}{p(Y)} = \frac{p(X)p(Y|X, Z)p(Z)}{p(Y)} \neq p(X|Y)p(Z|Y),$$

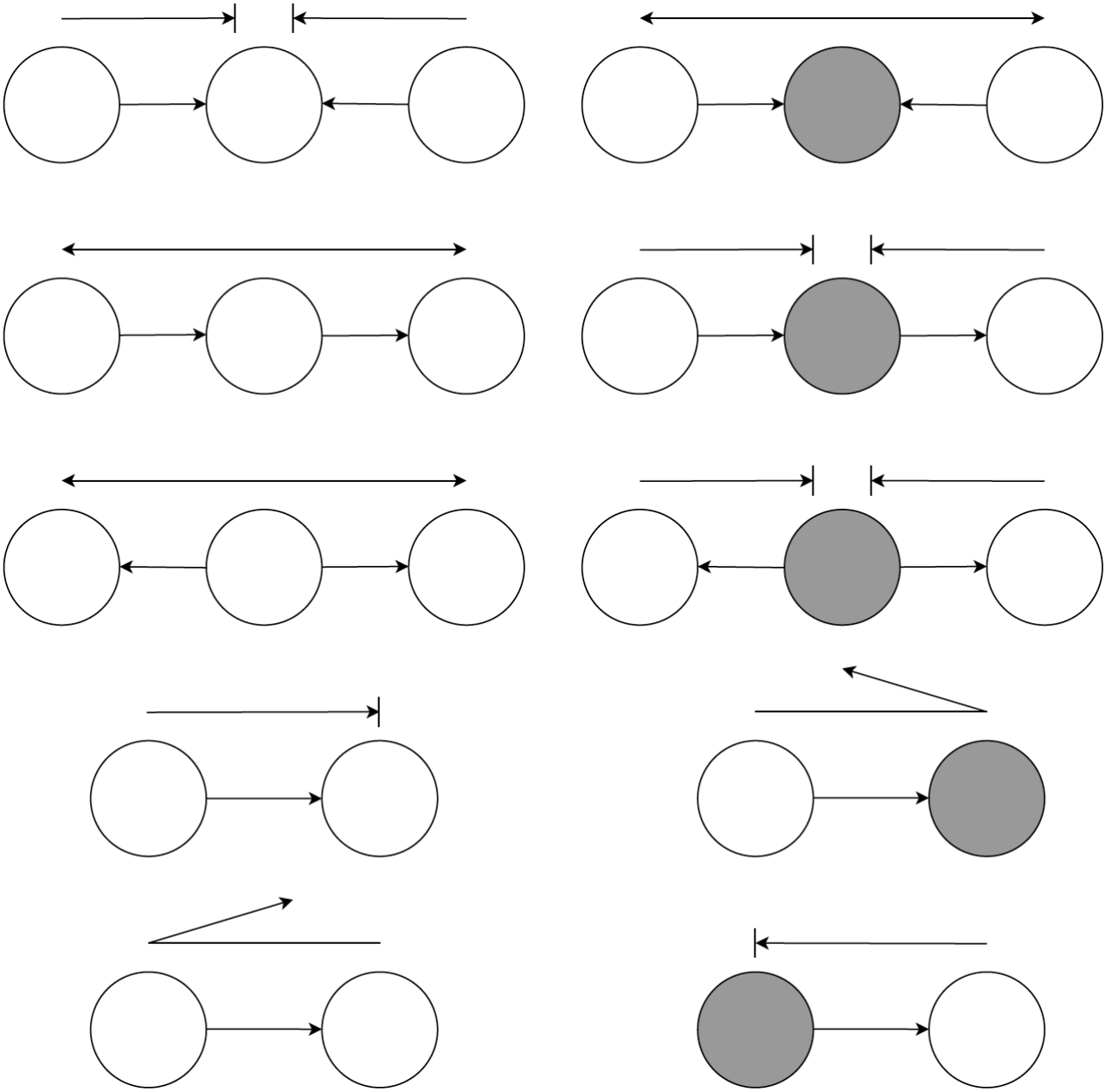
and thus $X \not\perp Z|Y$.

2.4.2 Bayes-Balls Algorithm

In general, the algorithm works as follows:

- Shade all nodes x_C (condition).
- Place "balls" at each node in x_A or x_B.
- Let the "balls" bounce around according to some rules.
  * If any of the balls reach any of the nodes in x_B from x_A or x_A form x_B, then $x_A \not\perp x_B|x_C$. Otherwise, $x_A \perp x_B|x_C$.

There are 10 basic rules:



3 Undirected Graphical Model

Definition 3.1. ***Undirected graphical model*** (UGM), also called ***Markov random field*** (MRF) or Markov network, is a set of random variables described by an undirected graph. As in DAGM, the nodes in the graph represent random variables and the edges represent probabilistic interactions between neighboring variables.

Definition 3.2. ***Global Markov property***: $X_A \perp X_B | X_C$ iff X_C separates X_A from X_B/.

Definition 3.3. A ***clique*** is an undirected graph is a subset of its vertices s.t. every two vertices in the subset are connected by an edge. A ***maximal clique*** is a clique that cannot be extended by including one more adjacent vertex. A ***maximum clique*** is a clique of the largest possible size in a given graph.

3.1 Parameterization of an UGM

Let $x = (x_1, \cdots, x_m)$ be the set of all random variables. Unlike in DGM, there is no topological ordering associated with an undirected graph, and so we cannot use the chain rule to represent the joint distribution $p(x)$.

We associate potential functions or factors with each maximal clique in the graph. For a given clique c, we define the potential function or factor $\psi_c(x_c|\theta_c)$ to be any non-negative function, where x_c is some subset of variables in x involved in a unique and maximal clique. The joint distribution is proportional to the product of clique potentials

$$p(x) \propto \prod_{c \in \mathcal{C}} \psi_c(x_c|\theta_c).$$

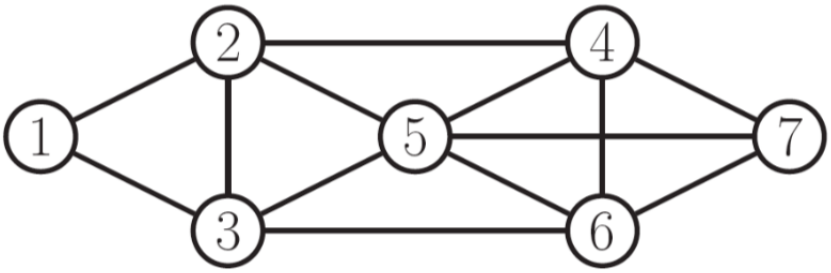
More formally, a positive distribution $p(x) > 0$ satisfies the conditional independence properties of an undirected graph G iff p can be represented as a product of factors, one per maximal clique, i.e.,

$$p(x|\theta) = \frac{1}{Z(\theta)} \prod_{c \in \mathcal{C}} \psi_c(x_c|\theta_c),$$

where $\mathcal{C}$ is the set of all maximal cliques of G and $Z(\theta)$ is the partition function defined as

$$Z(\theta) = \sum_x \prod_{c \in \mathcal{C}} \psi_c(x_c|\theta_c).$$

Example 3.1. $p(x) \propto \psi_{1,2,3}(x_1, x_2, x_3)\psi_{2,3,5}(x_2, x_3, x_5)\psi_{2,4,5}(x_2, x_4, x_5)\psi_{3,5,6}(x_3, x_5, x_6)\psi_{4,5,6,7}(x_4, x_5, x_6, x_7)$.



4 Exact Inference

4.1 Inference as Conditional Distribution

Let X_E be the observed evidence, X_F be the unobserved variable we want to infer and $X_R = X \backslash \{X_F, X_E\}$ be the remaining variables, extraneous to query. For inference, we will marginalize out these extraneous variables, focusing on the joint distribution over evidence and subject of inference:

$$p(X_F, X_E) = \sum_{X_R} p(X_F, X_E, X_R).$$

In particular, inference will focus computing the conditional probability distribution

$$p(X_F|X_E) = \frac{p(X_F, X_E)}{p(X_E)} = \frac{p(X_F, X_E)}{\sum_{X_F} p(X_F, X_E)} = \frac{p(X_F, X_E)}{\sum_{X_F, X_R} p(X_F, X_E, X_R)}.$$

4.2 Variable Elimination (VE)

Variable elimination

- Is a simple and general exact inference algorithm in any probabilistic graphical model.
- Has computational complexity that depends on the graph structure of the model.
- Can use dynamic programming to avoid enumerating all variable assignments.

Example 4.1 (Chain)**.** Suppose a simple chain

$$A \to B \to C \to D,$$

where we want to compute $P(D)$ with no observations for other variables. We have

$$X_F = \{D\}, X_E = \{\}, X_R = \{A, B, C\}.$$

The graphical model describes the factorization of the joint distribution as

$$p(A, B, C, D) = p(A)p(B|A)p(C|B)p(D|C).$$

If the goal is to compute the marginal distribution $p(D)$ with no observed variables then we marginalize over all variables but D :

$$p(D) = \sum_{A,B,C} p(A, B, C, D).$$

However, if we sum naively, it will be exponential $\mathcal{O}(k^4)$, where k is the number states:

$$p(D) = \sum_C \sum_B \sum_A p(A)p(B|A)p(C|B)p(D|C).$$

If we choose elimination ordering:

$$\begin{aligned} p(D) &= \sum_C p(D|C) \sum_B p(C|B) \sum_A p(A)p(B|A) \\ &= \sum_C p(D|C) \sum_B p(C|B)p(B) \\ &= \sum_C p(D|C)p(C), \end{aligned}$$

we reduce the complexity by first computing terms that appear across the other marginalization sums. So by dynamic programming to do the computation inside out instead of outside in, we have done inference over the joint distribution represented by the chain without generating it explicitly. The cost is $\mathcal{O}(4k^2)$.

4.2.1 Intermediate Factor

In the above example, each time we eliminated a variable it resulted in a new conditional or marginal distribution. However, in general eliminating does not produce a valid conditional or marginal distribution of the graphical model. For example, we have

$$p(X, A, B, C) = p(X)p(A|X)p(B|A)p(C|B, X)$$

and we want to marginalize over X :

$$p(A, B, C) = p(B|A) \sum_X p(X)p(A|X)p(C|B, X),$$

However, $\sum_X p(X)p(A|X)p(C|B, X)$ does not correspond to a valid conditional or marginal distribution since it is unnormalized.

Hence, we introduce factors ϕ, which are not necessarily normalized distributions, but which describe the local relationship between random variables.

In the above example:

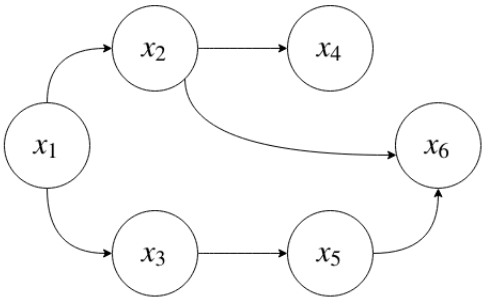
$$\begin{aligned} p(A, B, C) &= \sum_X p(X)p(A|X)p(B|A)p(C|B, X) \\ &= \sum_X \phi(X)\phi(A, X)\phi(A, B)\phi(X, B, C) \\ &= \phi(A, B) \sum_X \phi(X)\phi(A, X)\phi(X, B, C) \\ &= \phi(A, B)\tau(A, B, C). \end{aligned}$$

The original conditional distributions are represented by factors over all variables involved, which obfuscates the dependence relationship between the variables encoded by the conditional distribution. Following marginalizing over X we introduce a new factor τ over the remaining variables.

Note that for directed acyclic graphical models, who are defined by factorizing the joint into conditional distributions, we introduce intermediate factors to only be careful about notation.

However, there are other kinds of graphical models (e.g. undirected graphical models, and factor graphs) that are not represented by factorizing the joint into a product of conditional distributions. Instead, they factorize into a product of local factors, which will need to be normalized.

Example 4.2 (DGM)**.** Observing the state of a random variable x_6, find $p(x_1|x_6)$.



First recall that

$$p(x_1, \cdots, x_6) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5),$$

where

$$X_F = \{x_1\}, X_E = \{x_6\}, X_R = \{x_2, x_3, x_4, x_5\},$$

and

$$p(x_1|x_6) = \frac{p(x_1, x_6)}{\sum_{x_1} p(x_1, x_6)}.$$

We use variable elimination to compute $p(x_1, x_6)$:

$$\begin{aligned}
p(x_1, x_6) &= p(x_1) \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5) \\
&= p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x_3} p(x_3|x_1) \sum_{x_4} p(x_4|x_2) \sum_{x_5} p(x_5|x_3)p(x_6|x_2, x_5) \\
&= p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x_3} p(x_3|x_1) \sum_{x_4} p(x_4|x_2)p(x_6|x_2, x_3) \\
&= p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x_3} p(x_3|x_1)p(x_6|x_2, x_3) \sum_{x_4} p(x_4|x_2) \\
&= p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x_3} p(x_3|x_1)p(x_6|x_2, x_3) \\
&= p(x_1) \sum_{x_2} p(x_2|x_1)p(x_6|x_1, x_2) \\
&= p(x_1)p(x_6|x_1).
\end{aligned}$$

The complexity of variable elimination is related to the elimination ordering. Unfortunately, finding the best elimination ordering is NP-hard.

4.2.2 Sum-Product Inference Algorithm

Computing $p(Y)$ for directed and undirected models is given by sum-product inference algorithm

$$\tau(Y) = \sum_z \prod_{\phi \in \Phi} \phi(z_{\text{Scope}[\phi] \cap Z}, y_{\text{Scope}[\phi] \cap Y}), \forall Y,$$

where Φ is a set of potentials or factors.

For directed models, Φ is given by the conditional probability distributions for all variables

$$\Phi = \{\phi_{x_i}\}_{i=1}^N = \{p(x_i|\text{Parents}(x_i))\}_{i=1}^N,$$

where the sum is over the set $Z = X - X_F$. The resulting term $\tau(Y)$ will automatically be normalized.

For undirected models, Φ is given by the set of unnormalized potentials. Therefore, we must normalize the resulting $\tau(Y)$ by $\sum_Y \tau(y)$, i.e.,

$$p(Y) = \frac{\tau(Y)}{\sum_Y \tau(t)}.$$

Example 4.3 (Directed Graph)**.** The graph describes a factorization of the joint distribution

$$p(C, D, I, G, S, L, H, J) = p(C)p(D|C)p(I)p(G|D, I)p(L|G)p(S|I)p(J|S, L)p(H|J, G).$$

For notation convenience, we can write the conditional distributions as factors:

$$\Phi = \{\phi(C).\phi(D, C), \phi(I), \phi(G, D, I), \phi(L, G), \phi(S, I), \phi(J, S, L), \phi(H, J, G)\}.$$

Suppose now we are interested in inferring $p(J)$.

Eliminating ordering $\prec \{C, D, I, H, G, S, L\}$

$$\begin{aligned}
p(J) &= \cdots \underbrace{\sum_C \phi(C)\phi(D, C)}_{\tau(D)} \\
&= \cdots \underbrace{\sum_D \phi(G, D, I)\tau(D)}_{\tau(G,I)} \\
&= \cdots \underbrace{\sum_I \phi(I)\phi(S, I)\tau(G, I)}_{\tau(S,G)} \\
&= \cdots \underbrace{\left[\sum_H \phi(H, J, G)\right]}_{\tau(J,G)} \tau(S, G) \\
&= \cdots \underbrace{\sum_G \phi(L, G)\tau(J, G)\tau(S, G)}_{\tau(L,J,S)} \\
&= \cdots \underbrace{\sum_S \phi(J, S, L)\tau(L, J, S)}_{\tau(J,L)} \\
&= \underbrace{\sum_L \tau(J, L)}_{\tau(J)} = \tau(J).
\end{aligned}$$

Note that since our original factors correspond to conditional and marginal distributions we do not need to renormalize the final factor $\tau(J)$. However, if we started with potential factors not from a conditional distribution, we would have to normalize $\frac{\tau(J)}{\sum_J \tau(j)}$.

4.3 Complexity of VE

The complexity of the VE algorithm is

$$\mathcal{O}(mk^{N_{\max}}),$$

where m is the number of potential functions, $|\Phi|, k$ is the number of states each random variable takes (assumed to be equal here), N_i is the number of random variables inside each sum $\sum_i$, and $N_{\max} = \arg\max_i N_i$.

Example 4.4 (Complexity of Eliminating Ordering $\prec \{C, D, I, H, G, S, L\}$)**.** We have $|\Phi| = 9, N_{\max} = N_G = 4$ and thus the complexity of the variable elimination under this ordering is $\mathcal{O}(8k^4)$.

5 Message Passing

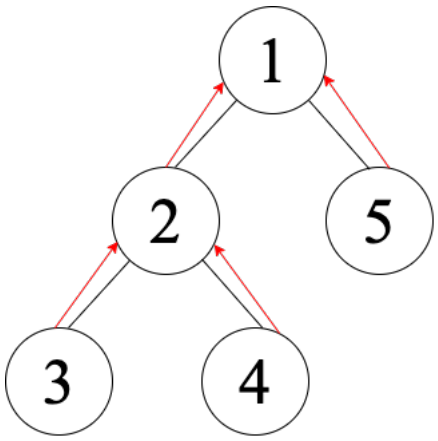
Belief propagation is based on message-passing of messages between neighboring vertices of the graph. The message sent from variable j to i is

$$m_{j\to i}(x_i) = \sum_{x_j} \phi_j(x_j)\phi_{ij}(x_i, x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j),$$

where each message $m_{j\to i}(x_i)$ is a vector with one value for each state of x_i. In order to compute $m_j \to m_i$, we must have computed $m_k \to j(x_j)$ for $k \in N(j) \neq i$. Wherefore, we need a specific ordering of the messages.

Example 5.1. Consider a tree:

$$p(x_{1:n}) = \frac{1}{Z}\prod_{i=1}^{N} \phi_i(x_i) \prod_{i,j\in T} \phi_{i,j}(x_i, x_j).$$



Now we want to compute $p(x_1)$. Choosing an ordering is equivalent to choosing a root. We choose x_1 to be the root, then

$$\begin{aligned}
m_{5\to 1}(x_1) &= \sum_{x_5} \phi_5(x_5)\phi_{15}(x_1, x_5) \\
m_{3\to 2}(x_2) &= \sum_{x_3} \phi_3(x_3)\phi_{23}(x_2, x_3) \\
m_{4\to 2}(x_2) &= \sum_{x_4} \phi_4(x_4)\phi_{24}(x_2, x_4) \\
m_{2\to 1}(x_1) &= \sum_{x_2} \phi_2(x_2)\phi_{12}(x_1, x_2)m_{3\to 2}(x_2)m_{4\to 2}(x_2).
\end{aligned}$$

Thus

$$p(x_1) \propto \phi_1(x_1)m_{2\to 1}(x_1)m_{5\to 1}(x_1),$$

where $Z = \sum_{x_1} p(x_1)$.

Algorithm 1: Belief Propagation Sampling

choose root r arbitrarily;
pass messages from leafs to r;
pass messages from r to leafs;
compute

$$p(x_i) \propto \phi_i(x_i) \prod_{j \in N(i)} m_{j \to i}(x_i), \forall i.$$

Note that the message coming back down the tree from the root each only depend on the information from the other branches and there is no double-counting of information.

6 Sampling

We assume the density from which we wish to draw samples, $p(x)$ can be evaluated to within a multiplicative constant, i.e., we can evaluate a function $\widetilde{p}(x)$ s.t.

$$p(x) = \frac{\widetilde{p}(x)}{Z}.$$

6.1 Ancestral Sampling

Given a DAG and the ability to sample from each of its factors given its parents, we can sample from the joint distribution over all the nodes by ***ancestral sampling***, which simply means sampling in a topological order, i.e., at each step, sample from any conditional distribution that we have not visited yet, whose parents have all been sampled. This procedure will always start with the nodes that have no parents.

Algorithm 2: Ancestral Sampling

choose a node who has all parents already sampled;
sample it loop until all sampled.

Example 6.1. In a chain or HMM, you will always start with z_1 and move to the right. In a tree, you would always start from the root.

6.1.1 Generating Marginal Samples

If we are only interested in sampling a particular set of nodes, we can simply sample from all the nodes jointly, then ignore the nodes we do not need.

6.1.2 Generating Conditional Samples

If we want to sample conditional on a node with no parents, it is easy to do ancestral sampling starting from the nodes we have. However, to sample from a DAG conditional on leaf nodes is hard in the same way that inference is hard in general. For example, sampling the unknown key in a cryptosystem given the cyphertext but not knowing the plaintext.

6.2 Simple Monte Carlo

Definition 6.1. ***Simple Monte Carlo*** is given $\{x^{(i)}\}_{i=1}^N \sim p(x)$ we estimate the expectation $\mathbb{E}_{x\sim p(x)}[\phi(x)]$ to be the estimator $\widehat{\Phi}$

$$\Phi = \mathbb{E}_{x\sim p(x)}[\phi(x)] \approx \frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)}) = \widehat{\Phi}.$$

Property 6.1. If the vectors $\{x^{(i)}\}_{i=1}^N$ are generated from $p(x)$ then the expectation of $\widehat{\Phi}$ is Φ, i.e., $\widehat{\Phi}$ is an unbiased estimator of Φ.

Proof.

$$\begin{aligned}\mathbb{E}_{x\sim p(\{x^{(i)}\}_{i=1}^N)}[\widehat{\Phi}] &= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[\phi(x^{(i)})] \\ &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{x\sim p(x)}[\phi(x)] = \frac{N}{N}\mathbb{E}_{x\sim p(x)}[\phi(x)] = \Phi.\end{aligned}$$

□

Property 6.2. As the number of samples of N increases, the variance of $\widehat{\Phi}$ will decrease proportional to $\frac{1}{N}$.

Proof.

$$\mathrm{Var}(\widehat{\Phi}) = \mathrm{Var}\left[\frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})\right] = \frac{1}{N^2}\sum_{i=1}^{N}\mathrm{Var}[\phi(x^{(i)})] = \frac{1}{N}\mathrm{Var}[\phi(x)].$$

□

The accuracy of the Monte Carlo estimate depends only on the variance of ϕ not on the dimensionality of the x. So regardless or the dimensionality of x, it may be that as few as a dozen independent samples $\{x^{(i)}\}$ suffice to estimate Φ satisfactorily.

Suppose we can evaluate $\widetilde{p}(x)$, it is still hard to sample $p(x)$:

- We do not typically know the normalizing constant Z.
- Even we knew Z, the problem of drawing samples from $p(x)$ is still a challenge, especially in high-dimensional spaces, since there is no obvious way to sample from p without enumerating most or all of the possible states.

Example 6.2 (Lattice Discretization)**.** A (bad) idea is ***lattice discretization***. We could discretize the variable x and sample from the discrete probability distribution over a finite set of uniformly spaced points $\{x_i\}$. If we evaluate $\widetilde{p}(x_i)$ at each point x_i, we could compute

$$Z = \sum_i \widetilde{p}_i(x_i)$$

and

$$p_i = \frac{\widetilde{p}_i}{Z}$$

and could sample from the probability distribution $\{p_i\}_{i=1}^N$ using various methods based on a source of random bits. Unfortunately, the cost of the procedure is intractable. To evaluate Z, we must visit every point in the space. Suppose there are 50 uniformly spaced points in one dimension. If our system had $N = 1000$ dimensions, then the corresponding number of points would be $50^N = 50^{1000}$. Even if each component x_n took only two discrete values, the number of evaluations of $\widetilde{p}$ needed to evaluate Z would take many times. The cost of the lattice discretization method of sampling from $p(x)$ is exponential in the dimension of data, i.e., D^N where D is the number of data points and N is the dimension.

6.3 Monte Carlo Method

6.3.1 Importance Sampling

Importance sampling is not a method for generating samples from $p(x)$ but a method for estimating the expectation of a function $\phi(x)$. It can be viewed as a generalization of the uniform sampling method.

The density from which we wish to draw samples $p(x)$ can be evaluated to within a multiplicative constant, i.e., we can evaluate a function $\widetilde{p}(x)$ s.t. $p(x) = \frac{\widetilde{p}(x)}{Z}$ as usual. Wu further assume we have a simpler density $q(x)$ from which it is easy to sample from and easy to evaluate

$$q(x) = \frac{\widetilde{q}(x)}{Z_q}$$

where a density $q(x)$ is the ***sampler density***.

In importance sampling, we generate N samples from $q(x) : \{x^{(i)}\}_{i=1}^{N} \sim q(x)$.

If these points were samples from $p(x)$, then we could estimate Φ by

$$\Phi = \mathbb{E}_{x\sim p(x)}[\phi(x)] \approx \frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)}) = \widehat{\Phi},$$

where we could use a simple Monte Carlo estimator.

But when we generate samples from q, values of x where $q(x)$ is greater than $p(x)$ will be over-represented in this estimator, and points where $q(x)$ is less than $p(x)$ will be under-represented. To take into account the fact that we have sampled from the wrong distribution, we introduce ***weight***:

$$\widetilde{w}_r = \frac{\widetilde{p}(x^{(i)})}{\widetilde{q}(x^{(i)})}.$$

We rewrite the estimator under q :

$$\begin{aligned}
\Phi &= \int \phi(x)p(x)\mathrm{d}x = \int \phi(x)\cdot\frac{p(x)}{q(x)}\cdot q(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})\frac{p(x^{(i)})}{q(x^{(i)})} \\
&= \frac{Z_q}{Z_p}\frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})\frac{\widetilde{p}(x^{(i)})}{\widetilde{q}(x^{(i)})} = \frac{Z_q}{Z_p}\frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})\widetilde{w}_i = \frac{\frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})\widetilde{w}_i}{\frac{1}{N}\sum_{i=1}^{N}\widetilde{w}_i} \\
&= \frac{1}{N}\sum_{i=1}^{N}\phi(x^{(i)})w_i = \widehat{\Phi}_{\mathrm{iw}}
\end{aligned}$$

where $\frac{Z_p}{Z_q} = \frac{1}{N}\sum_{i=1}^{N}\widetilde{w}_i, w_i = \frac{\widetilde{w}_i}{\sum_{i=1}^{N}\widetilde{w}_i}$, and $\widehat{\Phi}_{\mathrm{iw}}$ is the importance weighted estimator (biased and consistent).

6.3.2 Rejection Sampling

We assume again a one-dimensional density $p(x) = \frac{\widetilde{p}(x)}{Z}$ that is too complicated a function for us to sample from it directly. We assume that we have simpler ***proposal density*** $q(x)$ which we can evaluate (within a multiplicative factor Z_q), and from which we can generate samples. We further assume that we know the value of a constant c s.t.

$$c\widetilde{q}(x) > \widetilde{p}(x), \forall x.$$

The procedure is as follows:

- Generate two random numbers: x is generated from the proposal density $q(x)$, and u is generated uniformly from the interval $[0, c\widetilde{q}(x)]$.

- Evaluate $\widetilde{p}(x)$ and accept or reject the sample x by comparing the value of u with the value of $\widetilde{p}(x)$: If $u > \widetilde{p}(x)$, then x is rejected; otherwise x is accepted and x is added to set of samples $\{x^{(i)}\}$ and the value of u discarded.

Note that rejection sampling will work best if q is a good approximation to p. If q is very different from p, then for cq to exceed p everywhere, c will necessarily have to be large and the frequency of rejection will be large.

6.3.2.1 Rejection Sampling in High Dimension

In a high-dimensional problem, it is likely that the requirement that $c\widetilde{q}$ be an upper bound for $\widetilde{p}$ will force c to be so huge that acceptances will be very rare. Besides, finding such a value of c may be difficult since in many problems we know neither where the modes of $\widetilde{p}$ are located nor how high they are.

In general c grows exponentially with the dimensionality N so the acceptance rate is expected to be exponentially small in N

$$\text{Acceptance rate} = \frac{\text{Area under } \widetilde{p}}{\text{Area under } c\widetilde{q}} = \frac{1}{Z}.$$

6.3.3 Metropolis-Hastings Method

Importance sampling and rejection sampling work well only if the proposal density $q(x)$ is similar to $p(x)$. In high dimensions, it is hard to find such q.

The Metropolis-Hastings algorithm instead makes use of a proposal density q which depends on the current state $x^{(t)}$. The density $q(x'|x^{(t)})$ might be a simple distribution such as a Gaussian centered on the current $x^{(t)}$, but in general can be any fixed density from which we can draw samples. It is not necessary that $q(x'|x^{(t)})$ look at all similar to $p(x)$ in order for the algorithm to be practically useful.

The procedure is as follows:

- A tentative new state x' is generated from the proposal density $q(x'|x^{(t)})$. To decide whether to accept the new state, we compute

$$a = \frac{\widetilde{p}(x')q(x^{(t)}|x')}{\widetilde{p}(x^{(t)})q(x'|x^{(t)})}.$$

- If $a \geqslant 1$ then the new state is accepted; otherwise, the new state is accepted w.p. a. If accepted, set $x^{(t+1)} = x'$; otherwise, set $x^{(t+1)} = x^{(t)}$.

Note that in rejection sampling, rejected points are discarded and have no influence on the list of samples $\{x^{(i)}\}$ but in Metropolis-Hastings method, a rejection causes the current state to be written again onto the list.

Metropolis-Hastings converges to $p(x)$ for any $q(x'|x^{(t)}) \geqslant 0, \forall x', x^{(t)}$ as $t \to \infty$, i.e., the list of samples converges towards the true distribution $\{x^{(i)}\}_{i=1}^{N} \to p(x)$.

There are however, no guarantees on convergence. The Metropolis-Hastings method is an example of a Markov chain Monte Carlo method (MCMC). In contrast to rejection sampling where the

accepted points $\{x^{(t)}\}$ are independent samples from the desired distribution, Markov chain Monte Carlo methods involve a Markov process in which a sequence of states $\{x^{(t)}\}$ is generated, each sample $x^{(t)}$ having a probability distribution that depends on the previous value $x^{(t-1)}$. Since successive samples are dependent, the Markov chain may have to be run for a considerable time in order to generate samples that are effectively independent samples from p.

As it is difficult to estimate the variance of an importance sampling estimator, it is difficult to assess whether a Markov chain Monte Carlo method has converged, and to quantify how long one has to wait to obtain samples that are effectively independent samples from p.

7 Hidden Markov Model

7.1 Markov Chain

Consider a sequential data

$$x_{1:T} = \{x_1, \cdots, x_T\},$$

where sequential data can be time-series (e.g., stock prices, speech) or ordered (e.g., textual data). The assumption that data is i.i.d. is a poor assumption for sequential data, as there is often clear dependencies between data points. But if we use general joint factorization with chain rule, then it is intractable for high-dimensional data - each factor requires exponentially many parameters to specify as a function of T.

Therefore, we assume now that each observation is independent of all previous observations except most recent, then data can be modeled as a first-order Markov chain

$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-1}).$$

The assumption simplifies the factors in the joint distribution

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t|x_{t-1}, \cdots, x_1) = \prod_{t=1}^{T} p(x_t|x_{t-1}).$$

In this model, then number of parameters is $(K-1) + (K-1)K(T-1)$, suppose every X is with K states.

Note that there is stationary and non-stationary distributions that generate data:

- ***Stationary Markov chain/Homogeneous Markov chain***: The distribution generating the data does not change through time.

$$p(x_t|x_{t-1}) = p(x_{t+k}|x_{t-1+j}), \forall t, k.$$

- ***Non-Stationary Markov chain***: The distribution generating the data is a function of time. We can always convert a non-stationary chain into a stationary one by appending time to the state.

We can also generalize to high-order Markov chain:

$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-m:t-1}).$$

7.2 Hidden Markov Model

Assuming we have a homogeneous model, the joint distribution of the model is

$$p(x_{1:T}, z_{1:T}) = \underbrace{p(z_1)}_{\text{Initial distribution}} \prod_{t=2}^{T} \underbrace{p(z_t|z_{t-1})}_{\text{Transition distribution}} \prod_{t=1}^{T} \underbrace{p(x_t|z_t)}_{\text{Emission probability}}.$$

- ***Initial distribution***: $\pi(i) = P(z_i = i)$, the probability of the first hidden variable.
- ***Transition distribution***: $T(i,j) = P(z_t = j|z_{t-1} = i)$, the probability of moving from hidden state i to hidden state j.

- ***Emission probability***: $\varepsilon_i(x) = P(x|z_t = i)$, the probability of an observed random variable x given the state of the hidden variable that emitted it.

Example 7.1. Suppose $x_t \in \{N, Z, A\}, z_t \in \{H, S\}$. Our observed states are whether or not we are watching Netflix (N), sleeping (Z), or working on the assignment (A) and our hidden states are whether we are happy (H) or sad (S). Say further that we are given the initial (π), transition (T), and emission probabilities (ε):

π	
H	0.70
S	0.30

T	H	S
H	0.80	0.20
S	0.10	0.90

ε	N	Z	A
H	0.40	0.50	0.10
S	0.10	0.30	0.60

From the tables, we can perform inference with the model. For example

$$\begin{aligned}
p(z_3 = H|z_1 = S) &= \sum_{z_2} p(z_3 = H, z_2|z_1 = S) \\
&= \sum_{z_2} p(z_3 = H|z_2, z_1 = S)p(z_2|z_1 = S) \\
&= \sum_{z_2} p(z_3 = H|z_2)p(z_2|z_1 = S) \\
&= p(z_3 = H|z_2 = H)p(z_2 = H|z_1 = S) + p(z_3 = H|z_2 = S)p(z_2 = S|z_1 = S) \\
&= 0.8 \times 0.1 + 0.1 \times 0.9 = 0.17.
\end{aligned}$$

7.2.1 Inference in HMM

HMM is tree-structured DAG, meaning that inference in them is always cheap (linear). We can use standard variable elimination and message passing to do exact inference. The main tasks we perform with HMM are as follows:

- Compute the probability of a latent sequence given an observation sequence (posterior marginal): $p(z_t|x_{1:T}), \forall t \in [1, T]$, which can be achieved with the forward-backward algorithm - regular message passing.
- Compute the marginal likelihood $p(x_1, \cdots, x_T)$ in order to fit parameters: The parameters of a HMM are sometimes learned with the Baum-Welch algorithm, a special case of the expectation-maximization algorithm.
- Infer the most likely sequence of hidden states: Compute $Z^* = \arg\max_{z_{1:T}} p(z_{1:T}|x_{1:T})$, which can be achieved with the Viterbi algorithm.

7.2.2 Forward-Backward Algorithm

The forward-backward algorithm is used to efficiently estimate the latent sequence given an observation sequence under HMM.

Assume that we know the initial $p(z_1)$, transition $p(z_t|z_{t-1})$ and emission $p(x_t|z_t)$ probabilities $\forall t \in [1, T]$. The task of hidden state inference breaks down into the following:

- ***Filtering***: Compute posterior over current hidden state, $p(z_t|x_{1:t})$.
- ***Prediction***: Compute posterior over future hidden state, $p(z_{t+k}|x_{1:t})$.
- ***Smoothing***: Compute posterior over past hidden state, $p(z_n|x_{1:t}), 1 < n < t$.

$p(z_t|x_{1:T}), \forall t$ are computed in two parts that are then multiplied together:

- ***Forward Filtering***: Compute $p(z_t|x_{1:t}), \forall t$.
- ***Backward Filtering***: Compute $p(x_{1+t:T}|z_t), \forall t$.

For forward filtering,

$$\begin{aligned} p(z_t|x_{1:t}) \propto p(z_t, x_{1:t}) &:= \alpha(t) \\ &= \sum_{z_{t-1}} p(z_t, z_{t-1}, x_t, x_{1:t-1}) \\ &= \sum_{z_{t-1}} p(x_t|z_{t-1}, z_t, x_{1:t-1}) p(z_t|z_{t-1}, x_{1:t-1}) p(z_{t-1}, x_{1:t-1}) \\ &= \sum_{z_{t-1}} p(x_t|z_t) p(z_t|z_{t-1}) \underbrace{p(z_{t-1}, x_{1:t-1})}_{\alpha(t-1)}. \end{aligned}$$

The forward recursion contains emission $p(x_t|z_t)$ and transition $p(z_t|z_{t-1})$ probabilities and if we recur all the way down to $\alpha(1)$, we get

$$\alpha(1) = p(z_1, x_1) = p(z_1)p(x_1|z_1).$$

Each step is $\mathcal{O}(K^2)$ and the total complexity is $\mathcal{O}(tK^2)$.

For backward filtering,

$$\begin{aligned} p(x_{t+1:T}|z_t) := \beta(t) &= \sum_{z_{t+1}} p(z_{t+1}, x_{t+1:T}|z_t) = \sum_{z_{t+1}} p(x_{t+2:T}|z_{t+1}, z_t, x_{t+1}) p(x_{t+1}|z_{t+1}, z_t) p(z_{t+1}|z_t) \\ &= \sum_{z_{t+1}} \underbrace{p(x_{t+2:T}|z_{t+1})}_{\beta(t+1)} p(x_{t+1}|z_{t+1}) p(z_{t+1}|z_t). \end{aligned}$$

8 Kullback-Leibler Divergence

8.1 Approximate Inference

Given the joint distribution

$$p(X) = p(X_1, \cdots, X_N) = p(X_E, X_F, X_R),$$

wi want to perform inference involving the distribution $p(X_F|X_E)$. Exact methods for marginalizing the required variables X_R and X_E is practically difficult. Instead we approximate either by sampling or variation methods.

8.1.1 Goal of Approximate Inference

We approximate

$$p(X) = \frac{1}{Z}\widetilde{p}(X)$$

with simpler distribution $q(X;\phi)$ and adjust ϕ so the distributions are close, $p \approx q$. Then

$$\sum_x f(x)p(x) \approx \sum_x f(x)q(x;\phi)$$

$$\mathbb{E}_{x\sim p(x)}[f(x)] \approx \mathbb{E}_{x\sim q(x;\phi)}[f(x)]$$

Note that to measure close between distributions, Euclidean distance in parameter space.

8.1.2 Kullback-Leibler (KL) Divergence

A way to compare two distributions defined

$$D_{\mathrm{KL}}(q\|p) = \mathbb{E}_{x\sim q(x)}\left[\ln\left(\frac{q(x)}{p(x)}\right)\right] = \sum_x q(x)\ln\left(\frac{q(x)}{p(x)}\right)$$

with the properties: (1) $D_{\mathrm{KL}}(q\|p) \geqslant 0$. (2) $D_{\mathrm{KL}}(q\|p) = 0 \Leftrightarrow q = p$. (3) $D_{\mathrm{KL}}(q\|p) \neq D_{\mathrm{KL}}(p\|q)$. Therefore, KL Divergence is not a distance metric.

8.1.2.1 $D_{\mathrm{KL}}(q\|p)$ v.s. $D_{\mathrm{KL}}(p\|q)$

We consider the difference between $\ln\left(\frac{q(x)}{p(x)}\right)$ and $\ln\left(\frac{p(x)}{q(x)}\right)$.

- ***Reverse-KL Information Projection*** - $D_{\mathrm{KL}}(q\|p) = \mathbb{E}_{x\sim q(x)}\left[\ln\left(\frac{q(x)}{p(x)}\right)\right]$:
  - $p \sim q \Rightarrow D_{\mathrm{KL}}$ is small.
  - p ls large, q is small $\Rightarrow D_{\mathrm{KL}}$ is small.
  - p is small, q is large $\Rightarrow D_{\mathrm{KL}}$ is large.
  - Hence, $D_{\mathrm{KL}}(q\|p)$ penalizes q having mass where p has none.
- ***Forward-KL Moment Projection*** - $D_{\mathrm{KL}}(p\|q) = \mathbb{E}_{x\sim p(x)}\left[\ln\left(\frac{p(x)}{q(x)}\right)\right]$:
  - $p \sim q \Rightarrow D_{\mathrm{KL}}$ is small.
  - p is large, q is small $\Rightarrow D_{\mathrm{KL}}$ is large.
  - p is small, q is large $\Rightarrow D_{\mathrm{KL}}$ is small.
  - Hence, $D_{\mathrm{KL}}(p\|q)$ penalizes q missing mass where p has some.

The choice of direction in practice does not appeal to any notion of whether information or moment projection is a better idea of closeness between distributions.

9 Stochastic Variational Inference

9.1 TrueSkill Latent Variable Model

The problem that this model is supposed to solve is inferring the skill of a set of players in a competitive game, based only on observing who beats who when they play against each other. In the TrueSkill model, each player has a fixed level of skill, denoted z_i. We initially do not know anything about anyone's skill, but for simplicity we assume every skill is independent.

We never get to observe the players' skills directly, which makes it a latent variable model. Instead, we observe the outcome of a series of matches between different players. For each game, the probability that player i beats player j is given by $\sigma(z_i - z_j)$ where $\sigma(\cdot)$ is the logistic function $\sigma(y) = \frac{1}{1+\exp(-y)}$. Hence,

$$p(i \text{ beats } j|z_i, z_j) = \frac{1}{1 + \exp(-(z_i - z_j))}.$$

The exact form of the prior and likelihood are not particularly important as long as the win probability for player i is non-decreasing in z_i and non-increasing for z_j. So the higher the skill level, the higher the chance of wining each game. We can write the joint likelihood of a set of players and games as

$$p(z_1, \cdots, z_N, \text{Game } 1, \cdots, \text{Game } T) = \left[\prod_{i=1}^{N} p(z_i)\right]\left[\prod_{i,j\in\text{Games}} p(i \text{ beats } j|z_i, z_j)\right].$$

Computing even the posterior over two players' skills requires integrating over all the other players' skills:

$$p(z_1, z_2|\text{Game } 1, \cdots, \text{Game } T) := p(z_1, z_2|x) = \int \cdots \int p(z_1, \cdots, z_N|x)\mathrm{d}z_3 \cdots \mathrm{d}z_N.$$

9.2 Posterior Inference in Latent Variable Model

Consider the probabilistic model $p(x, z)$ where $x_{1:T}$ are the observations, and $z_{1:N}$ are the unobserved latent variables. The conditional distribution of the unobserved variables given the observed variables (posterior inference) is

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x, z)\mathrm{d}z}$$

which we denote as $p_\theta(z|x)$. Whenever the number of values that z can take is large, the computation $\int p(x, z)\mathrm{d}z$ is intractable, making the computation of the conditional distribution itself intractable. Thus we have to use approximate inference.

9.2.1 Approximating the Posterior Inference with Variational Method

Approximation of the posterior inference with variational method works as follows:

- Introduce a variational family $q_\phi(z)$ with parameters ϕ.
- Encode some notion of distance between $p(z|x)$ and $q_\phi(z)$.
- Minimize the distance.

It turns Bayesian inference into an optimization problem, and if enough parts of our model are differentiable and well-approximated by simple Monte Carlo, we can use stochastic gradient descent to solve this optimization problem scalably.

Note that whatever parametric family of functions we choose for $q_\phi(z)$, it is unlikely that this variational family will have the true posterior $p(z|x)$ in it.

Once we have approximate posterior, we can find approximate answers to questions about the true posterior. For example, find

$$p(z_A > z_B|x) = \mathbb{E}_{p(z_A,z_B|x)}[I(z_A > z_B)]$$

which we could easily approximate with simple Monte Carlo if we could sample from $p(z_A, z_B|x)$ when we had a good approximate posterior. For example, we could sample from $q(z|x)$ and

$$p(z_A > z_B|x) \approxeq \mathbb{E}_{q(z_A,z_B|x)}[I(z_A > z_B)].$$

We will measure the distance between q_ϕ and p using the Kullback-Leibler divergence. Recall that

$$D_{\mathrm{KL}}(q_\phi(z|x)\|p(z|x)) = \int q_\phi(z|x) \ln\left(\frac{q_\phi(z|x)}{p(z|x)}\right) \mathrm{d}z = \mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{q_\phi(z|x)}{p(z|x)}\right)\right].$$

9.3 Variational Objective

We want to approximate p by finding a q_ϕ s.t.

$$q_\phi \approx p \Rightarrow D_{\mathrm{KL}}(q_\phi\|p) = 0.$$

But the computation of $D_{\mathrm{KL}}(q_\phi\|p)$ is intractable because it contains the term $p(z|x)$, which is intractable. To circumvent the issue of intractability, we can use ***evidence lower bound*** (***ELBO***).

Property 9.1. Maximizing the ELBO is equivalent to minimizing $D_{\mathrm{KL}}(q_\phi\|p)$.

Proof. We have

$$\begin{aligned} D_{\mathrm{KL}}(q_\phi(z|x)\|p(z|x)) &= \mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{q_\phi(z|x)}{p(z|x)}\right)\right] = \mathbb{E}_{z\sim q_\phi}\left[\ln\left(q_\phi(z|x)\cdot\frac{p(x)}{p(z,x)}\right)\right] \\ &= \mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{q_\phi(z|x)}{p(z,x)}\right)\right] + \mathbb{E}_{z\sim q_\phi}[\ln(p(x))] \\ &= -\mathcal{L}(\phi;x) + \ln(p(x)) \end{aligned}$$

where $\mathcal{L}(\phi;x)$ is the ELBO. Also note that $\ln(p(x))$ is not dependent on z. Rearranging, we have

$$\mathcal{L}(\phi;x) + D_{\mathrm{KL}}(q_\phi(z|x)\|p(z|x)) = \ln(p(x)).$$

Since $D_{\mathrm{KL}}(q_\phi(z|x)\|p(z|x)) \geqslant 0$, then

$$\mathcal{L}(\phi;x) \leqslant \ln(p(x))$$

and thus maximizing the ELBO is equivalent to minimizing $D_{\mathrm{KL}}(q_\phi(z|x)\|p(z|x))$. □

We have alternative derivation: Starting with Jensen's inequality

$$f(\mathbb{E}[X]) \leqslant \mathbb{E}[f(x)]$$

if X is a r.v. and f is a convex function. Given that $\ln(\cdot)$ is a concave function, we have

$$\ln(p(x)) = \ln\left(\int p(x,z)\mathrm{d}z\right) = \ln\left(\int p(x,z)\frac{q_\phi(z|x)}{q_\phi(z|x)}\mathrm{d}z\right) = \ln\left(\mathbb{E}_{z\sim q_\phi}\left[\frac{p(x,z)}{q_\phi(z|x)}\right]\right)$$

and thus

$$\ln\left(\mathbb{E}_{z\sim q_\phi}\left[\frac{p(x,z)}{q_\phi(z|x)}\right]\right) \geqslant \mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{p(x,z)}{q_\phi(z|x)}\right)\right] = -\mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{q_\phi(z|x)}{p(x,z)}\right)\right] = \mathcal{L}(\phi;x)$$

where $\mathcal{L}(\phi;x)$ is the ELBO.

The most general interpretation of the ELBO is given by

$$\mathcal{L}(\phi;x) = \mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{p(z)p(x|z)}{q_\phi(z|x)}\right)\right] = \mathbb{E}_{z\sim q_\phi}[\ln(p(x|z)) + \ln(p(z)) - \ln(q_\phi(z|x))],$$

which can be rewritten using entropy

$$\mathbb{E}_{z\sim q_\phi}[\ln(p(x|z)) + \ln(p(z))]\mathbb{H}[q_\phi(z|x)]$$

or

$$\mathbb{E}_{z\sim q_\phi}[\ln(p(x|z))] - D_{\mathrm{KL}}(q_\phi(z|x)\|p(z))$$

as a trade-off where the first term can be thought as a reconstruction likelihood, i.e., how probable is x given z, which encourages the model to choose the distribution which best reconstructs the data; where the second term acts as regularization - our parametrization should not move us too far from the true distribution.

9.3.1 Optimizing the ELBO

We have

$$\mathcal{L}(\phi) = -\mathbb{E}_{z\sim q_\phi}\left[\ln\left(\frac{q_\phi(z|x)}{p(x,z)}\right)\right] = \mathbb{E}_{z\sim q_\phi}[\ln(p(x,z)) - \ln(q_\phi(z|x))].$$

We can optimize with gradient methods, and we have

$$\nabla_\phi\mathcal{L}(\phi) = \nabla_\phi\mathbb{E}_{z\sim q_\phi(z|x)}[\ln(p(x,z)) - \ln(q_\phi(z|x))].$$

Note that we can get an unbiased estimator of any expectation as long as we can sample from the distribution and evaluate the function using simple Monte Carlo.

9.3.1.1 Pathwise Gradient

We have the gradient of an expectation and we can turn it into an expectation by switching the derivative and expectation operators. However, in general, we cannot switch the derivative and expectation if the distribution we are taking the expectation over depends on the parameter.

Hence, we need to factor out the randomness from q and put it unto a parameterless, fixed source

of noise $p(\varepsilon)$. We need to find a function $T(\varepsilon, \phi)$ s.t. if $\varepsilon \sim p(\varepsilon)$ and $z = T(\varepsilon, \phi)$, then $z \sim q_\phi(z)$. For example, transforming standard normal into Gaussian with a given mean and variance:

$$\varepsilon \sim \mathcal{N}(\varepsilon|0,1), z = \sigma\varepsilon + \mu \Rightarrow z \sim \mathcal{N}(z|\mu,\sigma).$$

With the trick, we can write:

$$\begin{aligned}\nabla_\phi \mathcal{L}(\phi) &= \nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}[\ln(p(x,z)) - \ln(q_\phi(z|x))] \\ &= \nabla_\phi \mathbb{E}_{\varepsilon\sim p(\varepsilon)}[\ln(p(x,T(\phi,\varepsilon))) - \ln(q_\phi(T(\phi,\varepsilon)|x))] \\ &= \mathbb{E}_{\varepsilon\sim p(\varepsilon)} \nabla_\phi[\ln(p(x,T(\phi,\varepsilon))) - \ln(q_\phi(T(\phi,\varepsilon)|x))].\end{aligned}$$

Now we have a differentiable function that we can estimate with simple Monte Carlo. Additionally, if $p(x|z)$ factorizes over the points in the dataset, we can speed up computation by randomly subsampling the data at each iteration as well. Hence, SVI can be used to fit models with millions of parameters on millions of datapoints.

There are some notable extensions: We can make q_ϕ arbitrarily expressive, for instance using a mixture of Gaussian, or a normalizing flow; we can fit both ϕ and parameters of $p(z|x)$ at the same time, for instance we might want to learn the shape of the function that determines the probability of a win given both players' skills.

9.3.1.2 Score Function Gradient Estimator

Score function gradient estimator, also called the likelihood ratio or reinforce, is given by

$$\nabla_\phi \mathbb{E}_{z\sim q_\phi(z)} f(z) = \nabla_\phi \int f(z) q_\phi(z) \mathrm{d}z.$$

If we assume $q_\phi(z)$ is a continuous function of ϕ, then

$$\nabla_\phi \mathbb{E}_{z\sim q_\phi(z)} f(z) = \int \nabla_\phi f(z) q_\phi(z) \mathrm{d}z = \int f(z) \nabla_\phi q_\phi(z) \mathrm{d}z.$$

Using the log-derivative trick, i.e., $\nabla_\phi \ln(q_\phi) = \frac{\nabla_\phi q_\phi}{q_\phi}$,

$$\nabla_\phi \mathbb{E}_{z\sim q_\phi(z)} f(z) = \int f(z) q_\phi(z|x) \nabla_\phi[\ln(q_\phi(z|x))] \mathrm{d}z = \mathbb{E}_{z\sim q_\phi(z)}[f(z)\nabla_\phi[\ln(q_\phi(z|x))]]$$

where $q_\phi(z|x)$ is the score function. Finally, we have

$$\nabla_\phi \mathcal{L}(\phi; x) = \mathbb{E}_{z\sim q_\phi(z)}[(\ln(p(x,z)) - \ln(q_\phi(z|x)))\nabla_\phi[\ln(q_\phi(z|x))]]$$

which is unbiased as the pathwise gradient, but usually have higher variance.

9.4 Mean Field Variational Inference

In mean field variation inference, we restrict to variational families q that full factorizes to $q_\phi(z)$ (but not x), i.e., we approximate $p(z|x)$ with $q_\phi(z)$:

$$q_\phi(z) = \prod_{i=1}^{N} q(z_i|\phi_i).$$

Note that if q's are in the same family as p's, we can optimize by coordinate ascent: Fix all other variables to optimize local, aggregate local to optimize global, and repeat until KL divergence.

10 Amortized Inference and Variational Autoencoder

10.1 Amortized Inference

Suppose a network take in x_i and output the mean and variance vector for a Gaussian:

$$q_\phi(z_i|x_i) = \mathcal{N}(z_i|\mu_\phi(x_i), \Sigma_\phi(x_i)).$$

The graphical model for the recognition model has the ϕ outside the plate. The algorithm for amortized inference is: 1. Sample a data point. 2. Compute parameters of approximate posterior (recognition). 3. Compute gradient of Monte Carlo estimate of ELBO w.r.t. ϕ. 4. Update ϕ.

If $\ln(p_\theta(x))$ depends on parameters θ, then the ELBO is a function of both, and we can optimize them together:

$$\begin{aligned}\nabla_{\theta,\phi}\mathcal{L}(\phi) &= \nabla_{\theta,\phi}\mathbb{E}_{z\sim q_\phi(z|x)}[\ln(p_\theta(x,z)) - \ln(q_\phi(z|x))] \\ &= \mathbb{E}_{\varepsilon\sim p(\varepsilon)}\nabla_{\theta,\phi}[\ln(p_\theta(x, T(\phi,\varepsilon))) - \ln(q_\phi(T(\phi,\varepsilon)|x))].\end{aligned}$$

In the TrueSkill model, it would let us learn the shape of the likelihood function. For example,

$$p(i \text{ beats } j|z_i, z_j) = \frac{1}{1+\exp(-\theta(z_i, z_j))}.$$

Thus we can jointly fit the model parameters and the recognition network by subsampling training examples and using simple Monte Carlo with stochastic gradient descent, which is called a ***variational autoencoder*** (VAE).

Example 10.1 (MNIST)**.** We will choose our prior on z to be the standard Gaussian $\mathcal{N}(0, I)$. The likelihood function is

$$p_\theta(x_i|z_i) = \prod_{d=1}^{D} \text{Ber}(x_{id}|\mu_\theta(z_i))$$

and approximate posterior is

$$q_\phi(z|x) = \mathcal{N}(\mu(x), \sigma(x)I).$$

We use neural networks as encoder and decoder: The encoder is $g_\phi(x_i) = \phi_i = [\mu_i, \ln(\sigma_i)]$ and the decoder is $f_\theta(z_i) = \theta_i$ where μ_i is the Bernoulli mean for each pixel in the input, x_i is encoded to vectors μ and $\ln(\sigma_i)$ which parameterize $q_\phi(z|x)$. Before decoding, we draw sample $z \sim q_\phi(z|x)$ and evaluate its likelihood under the model with $p_\theta(x|z)$. We compute the loss function $L(\theta, \phi; x)$ and propagate its derivative w.r.t. θ and ϕ, $\nabla_\theta L, \nabla_\phi L$, through the network during training.

10.2 Autoencoder

Definition 10.1. An ***autoencoder*** takes an input, encodes it into a vector, then decodes to produce something similar to the original data, i.e., autoencoder reconstruct the own input using an ***encoder*** and a ***decoder***. The encoder $g(x) \to z$ takes in the input data and outputs a single value for each encoding dimension while the decoder $f(z) \to \widehat{x}$ takes the encoding and attempts to recreate the original input.

The goal is to learn g, f from unlabeled data and usually we specify f and g with neural networks and minimize squared reconstruction error. z is the code the model attempts to compress a representation of the input x. For deterministic autoencoder, it is important that this code is a bottleneck, i.e., $\dim F < \dim X$, as it forces the encoder to reduce the dimension.

There are two main problems with deterministic autoencoder. Proximity in data space does not mean proximity in feature space: The embeddings or codes learned by the model are deterministic, i.e.,

$$g(x_1) = z_1 \Rightarrow f(z_1) = \widehat{x}_1$$
$$g(x_2) = z_2 \Rightarrow f(z_2) = \widehat{x}_2$$

but proximity in feature space is not enforced for inputs in close proximity in data space, i.e.,

$$z_1 \approx z_2 \not\Rightarrow x_1 \approx x_2.$$

If the space has regions where no data gets encoded to and we sample a variation from there, the decoder will simply generate an unrealistic output, because the decoder cannot deal with the region of the latent space. During training, it never saw encoded vectors coming from the region of latent space.

10.3 Variational Autoencoder (VAE)

This stochastic generation means that even for the same input, while the mean and standard deviations remain the same, the actual encoding will vary on every single pass simply due to sampling.

The VAE generation model learns to reconstruct its inputs not only from the encoded points but also from the area around them, which allows the generation model to generate new data by sampling from an area instead of only being able to generate already seen data corresponding to the particular fixed encoded points.

11 Generative Adversarial Network (GAN)

11.1 Review: Generative Model

Generative models make the assumption that the data was generated from some distribution

$$\{x_i\}_{i=1}^{N} \sim p_{\text{data}}.$$

We want to learn a model p_{model} that represents an estimate of p_{data}. The density p_{model} can be explicitly defined or approximated by random sampling from model.

11.1.1 Prototypical Generative Model

The prototypical generative model follows the following procedure:

- Sample some noise, $z \sim p(z)$.
- Pass the noise through a model G.
- Get a sample x from $G(z)$.

In VAE, we see that $x \sim p_\theta(x|z)$.

11.1.2 Maximum Likelihood Estimate

The basic idea is to define a model which provides an estimate of a probability distribution parameterized by parameters θ, and define a likelihood function that represents the probability of the data and train the parameters to maximize the likelihood.

11.1.3 Explicit Density Model

Explicit density model defines an explicit density function $p_{\text{model}}(x;\theta)$ which is used to train the model, typically via MLE. Maximization of the likelihood function is straightforward, but the main difficulty present in explicit density model is designing a model that can capture all of the complexity of the data to be generated while still maintaining computational tractability. In the intractable case, we use variational (e.g., VAE) or MCMC approximations to get $p_{\text{model}}(x;\theta)$.

11.2 Implicit Density Model

Implicit density model is trained without explicitly defining a density function.

11.3 Generative Adversarial Network (GAN) Approach

In the generative adversarial approach, we do not have likelihoods. The basic idea of GAN is to set up a game between two players:

- One of them is called the ***generator*** (G_{θ_G}). The generator creates samples that are intended to come from the same distribution as the training data.
- The other player is the ***discriminator*** (D_{θ_D}). The discriminator examines samples to determine whether they are real or fake.

The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake). The generator is trained to fool the discriminator. Both are almost always modeled as neural networks and are therefore differentiable w.r.t. their outputs. Once training is complete, we throw away the discriminator.

Formally, GAN is structured as a probabilistic model containing latent variables z and observed variable x to sample from the model: First sample $z \sim p(z)$ where $p(z)$ is some prior distribution and then $x = G(z) \sim p_{\text{model}}$. Note that we never explicitly define a distribution $p_{\text{model}}(x; \theta)$.

Contrast with VAE, where to sample from the model: First sample $z \sim q_\phi(z|x)$ for some input x, then using the encoder to compute $\theta = f(z)$, and finally, using the decoder to sample $x \sim p_\theta(x|z)$. Note that we explicitly define a distribution $p_\theta(x|z)$ and sample from it.

11.3.1 Adversarial Game

In the adversarial game, both players have cost functions that are defined in terms of both players' parameters:

- The discriminator wants to minimize $J^{(D)}(\theta_D, \theta_G)$ and must do so while controlling only θ_D.
- The generator wants to minimize $J^{(G)}(\theta_D, \theta_G)$ and must do so while controlling only θ_G.

Since each player's cost depends on the other player's parameters, but each player cannot control the other player's parameters, it is straightforward to describe as a game rather than as an optimization problem.

11.3.2 Training Procedure

The training process consists of simultaneous stochastic gradient descent. On each step, two mini-batches are sampled: a mini-batch of x values from the dataset and a mini-batch of z values from the model's prior over latent variables. For instance, $x \sim p_{\text{data}}$ and $z \sim p(z) \Rightarrow x \sim P_{G(z)}$. Then two gradient steps are made simultaneously: one updating θ_D to reduce $J^{(D)}$ and on updating θ_G to reduce $J^{(G)}$.

11.3.3 Cost Function

11.3.3.1 The Discriminator's Cost

The cost used for the discriminator is almost always:

$$J^{(D)}(\theta_D, \theta_G) = -\mathbb{E}_{x \sim p_{\text{data}}}[\ln(D(x))] - \mathbb{E}_{z \sim p(z)}[\ln(1 - D(G(z)))]$$

which is the standard cross-entropy cost that is minimized when training a standard binary classifier with a sigmoid output. The only difference is that the classifier is trained on two mini-batches of data: one coming from the dataset where the label is 1 for all examples, and one coming from the generator where the label is 0 for all examples.

11.3.3.2 Optimal Discriminator Strategy

The goal is to minimize $J^{(D)}(\theta_D, \theta_G)$ in function space, specifying $D(x)$ directly. We begin by assuming that both p_{data} and p_{model} are nonzero everywhere.

To minimize $J^{(D)}$ w.r.t. D, we can write the functional derivatives w.r.t. a single entry $D^{(x)}$ and set them equal to zero:

$$\frac{\delta}{\delta D(x)} J^{(D)} = 0.$$

By solving the equation, we have

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}.$$

The discriminator estimates the ratio between the data density and the sum of the data and model densities. Wherever the output of the discriminator is large, the model density is too low, and wherever the output of the discriminator is small, the model density is too high. Estimating the ratio is the key approximation mechanism used by GAN.

11.3.3.3 Optimal Generator Strategy

The simplest version of the game is a ***zero-sum game***, in which the sum of all player's costs is always zero. In this version of the game,

$$J^{(G)} = -J^{(D)}.$$

Since $J^{(G)}$ is tied directly to $J^{(D)}$, we can summarize the entire game with a value function specifying the discriminator's payoff:

$$V(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)}).$$

Zero-sum game is also called ***minimax game*** because the solution involves minimization in an outer loop and maximization in an inner loop:

$$\theta^{(G)*} = \arg\min_{\theta^{(G)}} \arg\max_{\theta^{(D)}} V(\theta^{(D)}, \theta^{(G)})$$

where we maximize θ_D s.t. $D_{\theta_D}(x) = 1$ and $D_{\theta_D}(G(z)) = 0$ and minimize θ_G s.t. $D_{\theta_D}(G_{\theta_G}(z)) \to 1$.

11.3.3.4 Heuristic, Non-Saturating Game

In the minimax game, the discriminator minimizes a cross-entropy, but the generator maximizes the same cross-entropy. This is unfortunate for the generator, because when the discriminator successfully rejects generator samples with high confidence, e.g., $D(G(z)) = 0$, the generator's gradient vanishes, which is known as the ***saturation problem***.

To solve the problem, we introduce the ***non-saturating game***, in which we continue to use cross-entropy minimization for the generator, but let

$$J^{(G)} = -\mathbb{E}_{z \sim p(z)}[\ln(D(G(z)))],$$

which leads to a strong gradient signal at $D(G(z)) = 0$. In the minimax game, the generator minimizes the log-probability of the discriminator being correct and the generator maximizes the log-probability of the discriminator being mistaken.