Statistical Methods for Machine Learning II

🅭 by Xia, Tingfeng

2020 winter term

Preface

This document is consist of notes from lectures and the online course notes that I, personally, find interesting/important. You can find the online course notes on the course website here: https://probmlcourse.github.io/sta414/

Contents

1 Lecture 2 - Introduction to Probabilistic Models

1.1 Overview

We have a random vector in the form $X = (X_1, \ldots, X_m)$ which can be ***either observed or unobserved.*** To approach this in a generative way, we make the so called generative assumption. which is that $X \sim P_{true}(X)$, i.e. there is some true distribution that is behind the scene and our data is from such distribution.

Goal Model a parametric joint distribution $P_\theta(X)$ by learning the parameters. The learning here means we want to find a/the "close"/"best" estimation to our parameter θ. In this course we will investigate the following three problems,

- How to specify the joint, $P_\theta(X)$?
- What does "best"/"close" mean? In some sense we want to find $P_\theta \approx P_{true}$, however P_{true} might also be unknown.
- How to find the best θ? In this course we will generally rely on gradient methods, so ∇_θ...

1.2 Probabilistic Perspective on ML

With this perspective, we can think about common machine learning tasks differently, where random variables represent:

- X: (high dimensional) input data
- C: discrete label
- Y: continuous target

If we assume our knowledge of the joint of the above three, i.e. we know $P(X, C, Y)$, then we can write our familiar tasks in the following way

- ***Regression***:

$$p(Y|X) = \frac{p(X,Y)}{P(X)} = \frac{p(X,Y)}{\int p(X,Y)dY} \tag{1.1}$$

- ***Classification/Clustering***:

$$p(C|X) = \frac{p(X,C)}{\sum_{C'} p(X,C')} \tag{1.2}$$

1.2.1 (Example) Classification

Suppose we have data of the form $\mathcal{D} = \{(x,c)_i\}_i$. We assume that they came from a certain true distribution, i.e. $\{(x,c)_i\}_i \sim p(X,C)$. Then, the ultimate goal of the ML problem is converted into finding $p(C|X)$. Using Bayes Rule of total probability, we can expand the distribution of interest into

$$p(C|X) = \frac{p(X,C)}{P(X)} = \frac{p(X,C)}{\sum_{C'} p(X,C')} \tag{1.3}$$

Output Heuristics After we acquire $p(C|X)$ as above, we are one step away from our goal of output the actual prediction c^*. There are three ways that we can do this, namely

- ***MLE Estimate*** is the most intuitive one, we simply choose

$$c^* = \arg\max_c p(C = c|X) \tag{1.4}$$

- ***Sample Learnt Dist*** is another approach which produces non-deterministic results, i.e. we sample $c^* \sim p(C|X)$.

- ***Combined*** is usually a safe way of doing this. We output

$$(c^*, p(C = c^*|X)) \iff (\text{result, how sure?}) \tag{1.5}$$

  As an example, we have a ML algorithm drives a car. In this case, we might want to make decision only when the machine learning model has a certain level of confidence.

1.3 Observed vs Unobserved Random Variables

1.3.1 Supervised Dataset

$$\{x_i, c_i\}_{i=1}^N \sim p(X, C) \tag{1.6}$$

In such case, the class labels are observed and finding the conditional distribution $p(C|X)$ satisfies the supervised classification problem.

1.3.2 Unsupervised Dataset

$$\{x_i\}_{i=1}^N \sim p(X, C) \tag{1.7}$$

Still under the generative assumption, where we assume that there is some underlying distribution for our dataset. Further, we assume that the distribution of data is related to the class labels for the data points even though the class labels are never observed. **A common way to refer to an unobserved discrete class label is "cluster"**. However, in this case, our final goal of classification is still $p(C|X)$[1.1].

1.3.3 Latent Variables

> Further, like clusters, introducing assumptions about unobserved variables is a powerful modelling tool. We will make use of this by modelling variables which are never observed in the dataset, called latent or hidden variables. By introducing and modelling latent variables, we will be able to naturally describe and capture abstract features of our input data.

[1.1] Might be helpful to think of Gaussian Mixture Models

1.4 Operations on Probabilistic Models

- ***Generate Data***: Sample from the model.
- ***Estimate Likelihood***: When all variables are either observed or marginalized, we produce the result which is a single real number that describes the 'probability' of the all variables taking on those specific values.
- ***Inference***: Compute the expected value of some variables given others which are either observed or marginalized.
- ***Learning***: Set the parameters of the joint distribution given some observed data to maximize the probability of the observed data.

1.5 Desiderata of Probabilistic Models

We have two desires for the joint distribution to learn, namely

- The marginal and conditional distribution can be computed efficiently
- The representation of the joint distribution should be compact. This is especially important when we are dealing with joint distributions over many variables.

In general, total joint distribution are too large to specify and would require an insane amount of data to fit even we wanted to. Thus, we need modelling assumptions.

1.5.1 Fully Dependent Factorization (Chain Rule)

Suppose we have sample space defined such that $|T| = 2$, $|W| = 3$, and $|M| = 4$. Then the total joint distribution could be expanding using the chain rule as (*note: not unique*)

$$P_\theta(T, W, M) = P(T)P(W|T)P(M|T, W) \tag{1.8}$$

which requires[1.2] $|\theta| = (2 - 1) + (3 - 1) \times 2 + (4 - 1) \times 2 \times 3 = 23$ parameters in total to specify.

1.5.2 Assumptions (Independence)

Introducing assumptions results in

- a less expressive model
- $|\theta|$ (usually, much) smaller

which can be bad sometimes (since the model is less expressive) and is a trade-off that we as modellers have to deal with.

[1.2] Here the bars mean "cardinality" rather than vector norm

(Example) Fully Independent We assume that $T \perp W \perp M$, then by definition we know, for example, $P(W|T) = P(W)$. Then

$$\begin{aligned} P_\theta(T,W,M) &= P(T)P(W|T)P(M|T,W) && (1.9)\\ &= P(T)P(W)P(M) && (1.10) \end{aligned}$$

which requires only $|\theta| = (2-1) + (3-1) + (4-1) = 6$ to fit.

1.6 Likelihood Function

For some observed data X, the likelihood describes the likeliness of the data under then distribution with parameter θ.

$$L(\theta) = p(X|\theta) \tag{1.11}$$

In general, we prefer to deal with the log likelihood function, which is defined as

$$\ell(\theta; X) = \log L(\theta) = \log p(X|\theta) \tag{1.12}$$

1.7 Maximum Likelihood Estimation

The idea is to find

$$\hat{\theta}_{MLE} := \arg\max_\theta \ell(\theta; \mathcal{D}) \tag{1.13}$$

In the case of i.i.d, we can re-write as

$$\begin{aligned} \hat{\theta}_{MLE} &= \arg\max_\theta \ell(\theta; \mathcal{D}) && (1.14)\\ &= \arg\max_\theta \log \prod_m p\left(x^{(m)}|\theta\right) && (1.15)\\ &= \arg\max_\theta \sum_m \log p\left(x^{(m)}|\theta\right) && (1.16) \end{aligned}$$

1.8 Sufficient Statistics

(Definition) Statistic A statistic is a possibly vector valued *deterministic* function of a set of random variables.

(Definition) Sufficient Statistic is a statistic that conveys exactly the same information about the data generating process that created the data as the entire data itself.[1.3] In formal language, Sufficient Statistic ($T(X)$) for X could be defined as

$$T(X^{(1)}) = T(X^{(2)}) \implies L(\theta; X^{(1)}) = L(\theta; X^{(2)}), \quad \forall\theta \tag{1.17}$$

alternatively, we can define it as

$$P(\theta|T(X)) = P(\theta|X); \quad \text{i.e. data doesn't give further info} \tag{1.18}$$

[1.3] Could also be interpreted as "summarize the data with respect to the likelihood"

1.8.1 Fisher-Neyman Factorization Theorem

If the probability function is $f_\theta(x)$, then T is a sufficient statistic for θ if and only if non-negative functions g and h can be found such that

$$P(\theta|T(X)) = h(x, T(x))g(T(x), \theta) \tag{1.19}$$

1.9 Exponential Family

Factorizes as

$$\begin{aligned} p(x|\eta) &= h(x) \exp\left\{\eta' T(x) - g(\eta)\right\} && (1.20) \\ &= h(x)g(\eta) \exp\left\{\eta' T(x)\right\} && (1.21) \end{aligned}$$

1.9.1 (Example) 1-D Gaussian

$$\begin{aligned} p(x|\theta) = \mathcal{N}(x|\mu, \sigma) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) && (1.22) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x^2 - 2x\mu + \mu^2)\right) && (1.23) \\ &= \underbrace{\frac{1}{\sqrt{2\pi}\sigma}}_{h(x)} \underbrace{\exp\left(\frac{-\mu^2}{2\sigma^2}\right)}_{g(\eta)} \exp\left(\underbrace{\begin{bmatrix} \frac{\mu}{\sigma^2} & \frac{-1}{2\sigma^2} \end{bmatrix}}_{\eta} \underbrace{\begin{bmatrix} x \\ x^2 \end{bmatrix}}_{T(X)}\right) && (1.24) \end{aligned}$$

2 Lecture 3 - Directed Graphical Models

2.1 Decision Theory (Utility Theory)

Let a denote action, and a^* be the optimal one. Also use s to denote state and $V(\cdot)$ be the value function. We have, in general

Often, it is very hard to find a/the good/best value function that quantifies everything using a numerical value.

$$a^* = \arg\min_a / \arg\max_a \underbrace{\mathbb{E}_{p(s|a,knowledge)}\left[V(s)\right]}_{u(a) \triangleq \text{ utility of action } a} \tag{2.1}$$

2.2 Graphical Model Notation

2.2.1 Chain Rule Expansion

Given any joint probability of N random variables, we can expand it as follows

$$p\left(x_{1,\ldots,N}\right) = p\left(x_1\right) p\left(x_2|x_1\right) p\left(x_3|x_2, x_1\right) \ldots p\left(x_n|x_{n-1:1}\right) \tag{2.2}$$

Formally speaking, in the case of two random variables would simply

$$p(x, y) = p(x|y)p(y) \tag{2.3}$$

and the general case for N random variables could be written as [2.1]

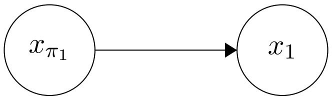
$$p\left(\bigcap_{i=1}^{N} x_i\right) = \prod_{j=1}^{N} p\left(x_j \middle| \bigcap_{k=1}^{j-1} x_k\right) \tag{2.4}$$

2.2.2 Graph Representation

(Example) Grouping Variables Consider the model

$$p\left(x_i, x_{\pi_i}\right) = p\left(x_{\pi_i}\right) p\left(x_i | x_{\pi_i}\right) \tag{2.5}$$

which we can use the following graph to represent:

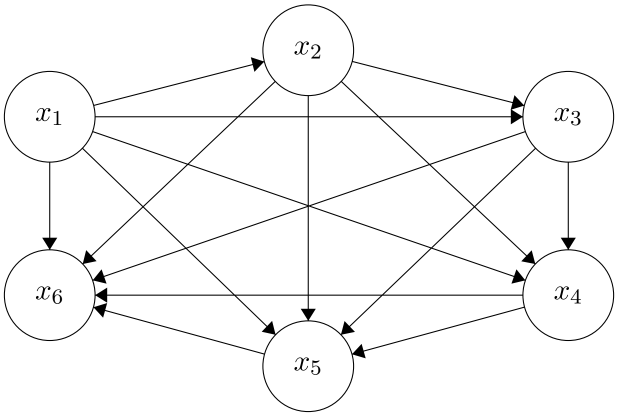


where

- ***nodes*** represent random variables
- ***arrows*** mean "conditioned on", e.g. "x_i is conditioned on x_{π_i}"

Notice that we can always group the variables together into one bigger variable, so in this example, x_{π_i} might represent a group of variables in stead of just one.

(Example) Fully Dependent 6 Nodes The total expansion of $p(x_{1:6})$ could be represented as



This is the resultant graphical model if we make ***absolutely no assumption*** on the independence. Notice that such model grows exponentially in complexity with respect to the number of parameters considered. We say such model "scales poorly".

[2.1] Note: when $k = 1$, $p\left(x_k | \cap_{j=1}^{k-1} x_j\right) = p(x_1)$

2.2.3 Conditional Independence

(Definition) Conditional Independence Let X be the set of nodes in our graph (the random variables of our model), then two sets of variables X_A, X_B are said to be conditionally independent given a third set of variables X_C if and only if either

$$p\left(X_A, X_B|X_C\right) = p\left(X_A|X_C\right) p\left(X_B|X_C\right) \tag{2.6}$$

or (***#! Important!***)

$$p\left(X_A|X_B, X_C\right) = p\left(X_A|X_C\right) \tag{2.7}$$

$$\iff p\left(X_B|X_A, X_C\right) = p\left(X_B|X_C\right) \tag{2.8}$$

and we denote the relation as $(X_A \perp X_B|X_C)$

2.2.4 Plates

In Bayesian methods, we treat parameters as random variables and hence we would like to include them in out graphical model. However, adding a node for each observation is quite cumbersome and thus we introduce plates, which denote replication of random variables.

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

Crossing Plates 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.

2.3 Directed Acyclic Graphical Models (DAGM)

A directed acyclic graphical model over N random variables look like

$$p\left(x_{1:N}\right) = \prod_{i}^{N} p\left(x_i|x_{\pi_i}\right) \tag{2.9}$$

where x_i is a random variable and x_{π_i} denotes the parents of the node (which could be an empty set). This notion is more general than the fully dependence model that looked at above. Notice that here each node is only dependent on its parents rather than all other nodes. Thus, the complexity of such model reduces to exponential in fan-in of each node, instead of the total N.

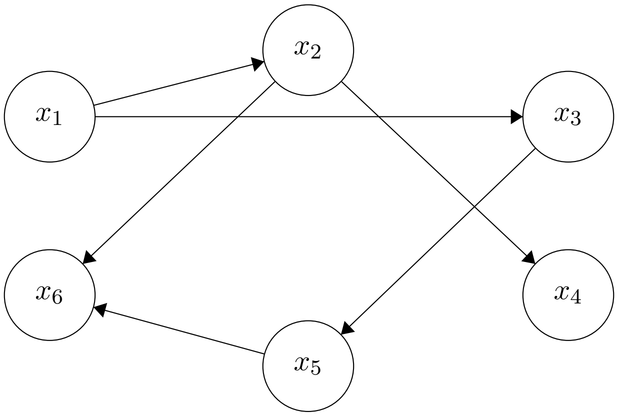
2.3.1 Independence Assumption on DAGMs

Then, we have the independence relationship of[2.2] $x_i \perp x_{Ancestor(\pi_i)}|x_{\pi_i}$ which expands into

$$p\left(x_{1,\ldots,6}\right) = p\left(x_1\right) p\left(x_2|x_1\right) p\left(x_3|x_1\right) p\left(x_4|x_2\right) p\left(x_5|x_3\right) p\left(x_6|x_2, x_5\right) \tag{2.10}$$

with the following respective graph

[2.2] Requires topological ordering, to be added later.



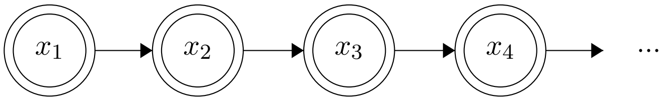
As we can see, the introduction of the assumption greatly reduced the complexity of the model.

2.3.2 (Example) Markov Chain

The following example has independence relationships that satisfies the Markov Property.

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_3)... \tag{2.11}$$

and could be represented, in graphical model, as



2.4 Directed - Separation

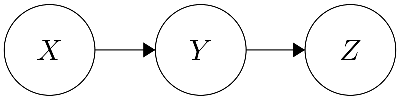
2.4.1 (Definition) D - Separation

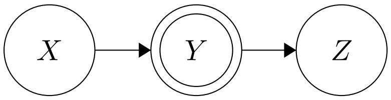
Directed-separation is a notion of connectedness in DAGs in which two (sets of) variables may or may not be connected conditioned on a third (set of) variable(s). D-connection implies conditional dependence and d-separation implies conditional independence.

2.4.2 Float Rules Derivation

Notation The double circles (or shaded circles) represent the notion of "conditioned-on".

Chain





In this case, we are interested in knowing whether or not

$$(X \perp Z)|Y \tag{2.12}$$

From the arrows between the nodes, we know that

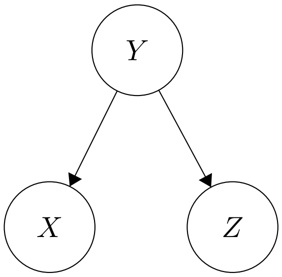
$$P(X,Y,Z) = P(X)P(Y|X)P(Z|Y) \tag{2.13}$$

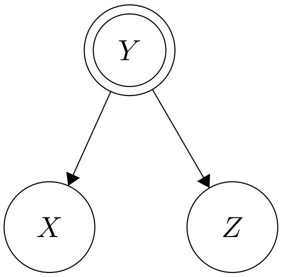
Then, we have

$$\begin{align}
P(X,Z|Y) &= \frac{P(X,Y,Z)}{P(Y)} \tag{2.14}\\
&= \frac{P(X)P(Y|X)P(Z|Y)}{P(Y)} \tag{2.15}\\
&= \frac{P(X,Y)P(Z|Y)}{P(Y)} \tag{2.16}\\
&= P(X|Y)P(Z|Y) \tag{2.17}
\end{align}$$

and this completes the proof. ■

Common Clause





As previous, we are interested in knowing whether or not $(X \perp Z)|Y$. From the graph, we can know that

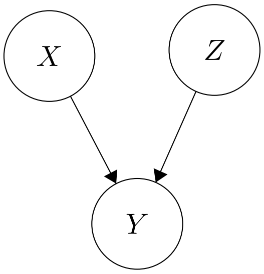
$$P(X,Y,Z) = P(Y)P(X|Y)P(Z|Y) \tag{2.18}$$

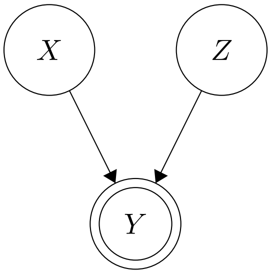
Then, we have

$$\begin{align}
P(X,Z|Y) &= \frac{P(X,Y,Z)}{P(Y)} \tag{2.19}\\
&= \frac{P(Y)P(X|Y)P(Z|Y)}{P(Y)} \tag{2.20}\\
&= P(X|Y)P(Z|Y) \tag{2.21}
\end{align}$$

and this completes the proof. ■

Explaining Away (Berkson's Paradox)





Notice from the graph that we have $P(X,Y,Z) = P(X)P(Z)P(Y|X,Z)$. First, I will prove that, in this case, $(X \not\perp Z)|Y$.

$$\begin{aligned}
P(Z|X,Y) &= \frac{P(X)P(Z)P(Y|X,Z)}{P(X)P(Y|X)} \tag{2.22}\\
&= \frac{P(Z)P(Y|X,Z)}{P(Y|X)} \tag{2.23}\\
&\neq P(Z|Y) \tag{2.24}
\end{aligned}$$

For marginal independence, i.e. $(X|Z)$, we want to show that $P(X,Z) = P(X)P(Z)$.

$$\begin{aligned}
P(X,Z) &= \sum_{Y'} P(X,Y',Z) \tag{2.25}\\
&= \sum_{Y'} P(X)P(Z)P(Y'|X,Z) \tag{2.26}\\
&= P(X)P(Z)\sum_{Y'} P(Y'|X,Z) \tag{2.27}\\
&= P(X)P(Z) \tag{2.28}
\end{aligned}$$

■

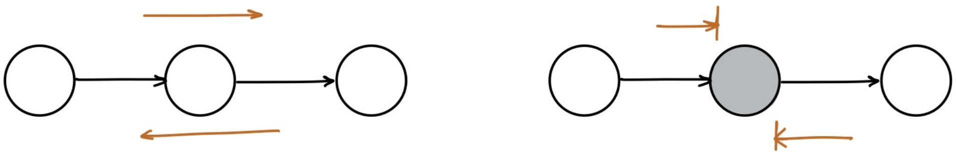
2.4.3 The Bayes-Ball Algorithm

To check if $x_A \perp x_B|x_C$, we will need

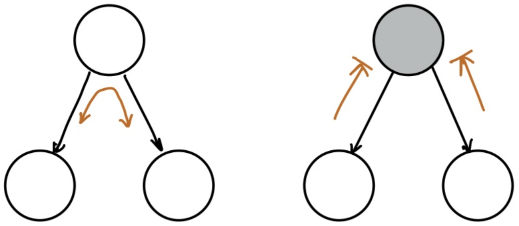
- Shade all nodes that were conditioned on, i.e. all nodes of x_C
- Place balls at each node in x_A (or x_B)
- Let the balls bounce around according to rules that are yet to be states below
  - If any of the balls reach any of the nodes in x_B from x_A (or reach x_A from x_B) then we declare $x_A \not\perp x_B|x_C$
  - Otherwise, $x_A \perp x_B|x_C$

2.4.4 Bayes Ball Rules

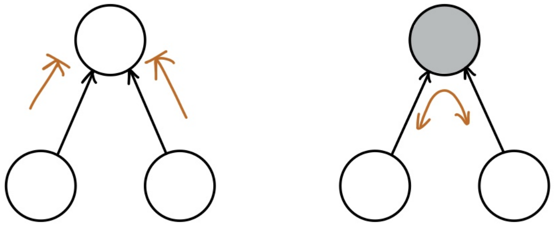
Chain



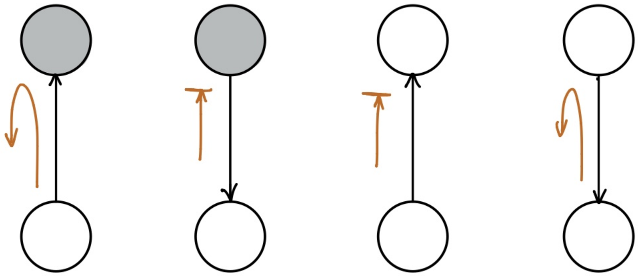
Common Cause



Explain Away



Linear



2.5 Unobserved Variables

Certain variables in our models may be unobserved , either some of the time or always, at training time or at test time. Graphically, we use shading to indicate observation.

2.5.1 Partially Unobserved Variables

[2.3] If variables are occasionally unobserved hen they are missing data, e.g., undefined inputs, missing class labels, erroneous target values. In this case, we can still model the joint distribution, but we marginalize the missing values

$$\ell(\theta;\mathcal{D}) = \sum_{\text{complete}} \log p\left(x^c, y^c|\theta\right) + \sum_{\text{missing}} \log p\left(x^m|\theta\right) \tag{2.29}$$

$$= \sum_{\text{complete}} \log p\left(x^c, y^c|\theta\right) + \sum_{\text{missing}} \log \sum_{y} p\left(x^m, y|\theta\right) \tag{2.30}$$

[2.3] An concrete example would be hospital data, where typically a large proportion of the data is missing.

2.5.2 Latent Variables

Above we discussed the case where some data are non-deterministically unobserved. Latent variables refers to those that ***never observed***. The handling of the latent variables depends on where it appears in our model,

- If we never condition on it when computing the probability of the variables we do observe, then we can just forget about it and integrate it out. For example, given y, x we fit the model
$$p(z, y|x) = p(z|y)p(y|x, w)p(w) \tag{2.31}$$
- If z is not a leaf node, marginalizing over it will induce dependencies between its children. For example, given y, x we can fit the model
$$p(y|x) = \sum_z p(y|x, z)p(z) \tag{2.32}$$

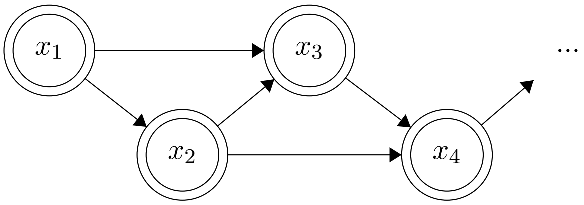
2.5.3 Mixture Models

In this case, we are looking at data that has no input on class information. We can sum the labels out,
$$p(x|\theta) = \sum_{k=1}^{K} p\left(z = k|\theta_z\right) p\left(x|z = k, \theta_k\right) \tag{2.33}$$
where bayes rule comes in handy for calculating the posterior (class responsibilities) of the mixture component given some data, i.e.,
$$p\left(z = k|x, \theta_z\right) = \frac{p\left(z = k|\theta_z\right) p_k\left(x|\theta_k\right)}{\sum_j p\left(z = j|\theta_z\right) p_j\left(x|\theta_j\right)} \tag{2.34}$$

2.6 Examples

2.6.1 Second-order Markov Chain

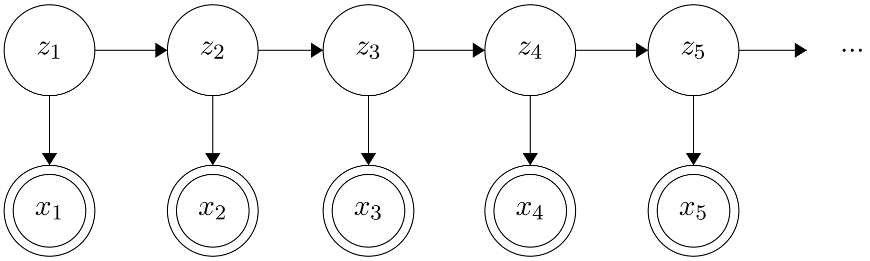
Consider the model
$$p\left(\mathbf{x}_{1:T}\right) = p\left(x_1, x_2\right) p\left(x_3|x_1, x_2\right) p\left(x_4|x_2, x_3\right) \cdots = p\left(x_1, x_2\right) \prod_{t=3}^{T} p\left(x_t|x_{t-1}, x_{t-2}\right) \tag{2.35}$$
which has the graphical representation (double circles mean "observed" here)



we notice that this model essentially assumes "the present depends on the past only through the current state as well as the last one."

2.6.2 Hidden Markov Models (HMMs)

HMM is a statistical model in which a system being modelled is assumed to be a Markov Process[2.4] with un-observed states. Here is the graphical model, where double circles mean "observed" here:



where

- z_t are hidden states taking on one of K discrete values
- x_t are observed variables taking on values in any space.

The above graph factorizes into

$$p\left(X_{1:T}, Z_{1:T}\right) = p\left(Z_{1:T}\right) p\left(X_{1:T} | Z_{1:T}\right) = p\left(Z_1\right) \prod_{t=2}^{T} p\left(Z_t | Z_{t-1}\right) \prod_{t=1}^{T} p\left(X_t | Z_t\right) \tag{2.36}$$

3 Lecture 4 - Exact Inference

3.1 Variable Elimination

3.1.1 (Simple Example) Chain

The example that we will consider is the simple chain

$$A \to B \to C \to D \tag{3.1}$$

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

$$X_F = \{D\}, X_E = \{\}, X_R = \{A, B, C\} \tag{3.2}$$

No observations for other variables means that we have no 'evidence' for other variables.

X_F is the set of variable that we are interested in, X_E is evidence, and X_R is the set of extraneous variables, i.e. what we marginalize out

The graphical model gives the factorization of

$$p(A, B, C, D) = p(A)p(B|A)p(C|B)p(D|C) \tag{3.3}$$

thus, if we want to find $p(D)$ we can marginalize over all other variables, i.e.

$$\begin{aligned} p(D) &= \sum_{A,B,C} p(A, B, C, D) && (3.4) \\ &= \sum_{C} \sum_{B} \sum_{A} p(A)p(B|A)p(C|B)p(D|C) && (3.5) \end{aligned}$$

If we marginalize the above the naïve way, then it cost exponentially $\mathcal{O}(k^n)$. Thus, we need to find a ***elimination ordering*** that gives us a smaller runtime complexity. We can reduce the complexity by first computing terms that appear across the other marginalization sums,

Here ϕ means factor, an non-normalized "pmf"

$$\phi(D) = \sum_C p(D|C) \sum_B p(C|B) \sum_A p(A)p(B|A) \tag{3.6}$$

$$= \sum_C p(D|C) \sum_B p(C|B)\phi(B) \tag{3.7}$$

$$= \sum_C p(D|C)\phi(C) \tag{3.8}$$

3.2 Sum-Product Inference

Let $X = Z \cup Y \wedge Z \cap Y = \emptyset$ be a set of random variables. . Then, to compute $P(Y)$ for a directed and undirected model could be achieved by the sum product inference algorithm

Here Y is the set that we want to do inference on and thus we are marginalizing over everything in Z

how to generalize to undirected graph?

$$\tau(Y) = \sum_z \prod_{\phi \in \Phi} \phi\left(\text{Scope}(\phi) \cap Z, \text{Scope}(\phi) \cap Y\right) \quad \forall Y \tag{3.9}$$

where the scope for each ϕ simply means the set of all random variables that have appeared in that specific ϕ, and Φ denotes the set of all ϕ's.

3.2.1 SP - Inference in Directed Models

In a directed model, the Φ is given by the conditional probability distributions for all variables, i.e.

$$\Phi = \{\phi_{x_i}\}_{i=1}^N = \{p\left(x_i | \text{ parents } (x_i)\right)\}_{i=1}^N \tag{3.10}$$

The resulting term $\tau(Y)$ will be automatically normalized.

3.2.2 SP - Inference in Undirected Models

For undirected models, Φ is given by the set of un-normalized potentials, and we ***must*** normalize the resulting $\tau(Y)$ by $\sum_Y \tau(y)$

3.2.3 (Example) Directed Graph

We have the following factorization of the joint distribution

$$p(C,D,I,G,S,L,H,J) = p(C)p(D|C)p(I)p(G|D,I)p(L|G)P(S|I)p(J|S,L)p(H|J,G) \tag{3.11}$$

and we will need the following potentials

$$\Phi = \{\phi(C), \phi(C,D), \phi(I), \phi(G,D,I), \phi(L,G), \phi(S,I), \phi(J,S,L), \phi(H,J,G)\} \tag{3.12}$$

[2.4] In continuous time, the Markov Chain model is known as Markov Process consider the problem where we want to infer $P(J)$ with elimination ordering $\prec_{\{C,D,I,H,G,S,L\}}$.[3.1] Then,

$$p(J) = \sum_L \sum_S \phi(J,L,S) \sum_G \phi(L,G) \sum_H \phi(H,G,J) \sum_I \phi(S,I)\phi(I) \sum_D \phi(G,D,I) \sum_C \phi(C)\phi(C,D) \tag{3.13}$$

$$= \ldots \underbrace{\sum_C \phi(C)\phi(C,D)}_{\tau(D)} \tag{3.14}$$

$$= \ldots \underbrace{\sum_D \phi(G,D,I)\tau(D)}_{\tau(G,I)} \tag{3.15}$$

$$= \ldots \underbrace{\sum_I \phi(S,I)\phi(I)\tau(G,I)}_{\tau(S,G)} \tag{3.16}$$

$$= \ldots \underbrace{\tau(S,G) \sum_H \phi(H,G,J)}_{\tau(S,G)\tau(G,J)} \qquad \text{// Note that } \tau(S,G) \text{ doesn't contain } H \tag{3.17}$$

$$= \ldots \underbrace{\sum_G \phi(L,G)\tau(S,G)\tau(G,J)}_{\tau(J,L,S)} \tag{3.18}$$

$$= \ldots \underbrace{\sum_S \phi(J,L,S)\tau(J,L,S)}_{\tau(J,L)} \tag{3.19}$$

$$= \ldots \underbrace{\sum_L \tau(J,L)}_{\tau(J)} \tag{3.20}$$

$$= \tau(J) \tag{3.21}$$

3.2.4 Complexity of VE

The complexity of VE is

$$\mathcal{O}\left(mk^{N_{\max}}\right) \tag{3.22}$$

where

- $m = |\Phi|$ is the number of initial factors, i.e. the number of terms in the original factorization of the joint probability (given the graphical model)
- 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 $\sum_i$ *at the time of marginalization*

[3.1] Note that what comes first in the ordering is the what we want to sum over in the inner most summation

- $N_{max} \triangleq \max_i N_i$ is the number of random variables inside the largest sum. (Inner sums are counted as one term all together)

4 Lecture 5 - Message passing, Hidden Markov Models, and Sampling

4.1 Message Passing

4.1.1 Belief Propagation: Motivation and Definitions

Our goal is to compute the marginal of every variable in graph $p(x_i), \forall x_i \in X$. Notice that in a tree, we have

$$P\left(X_{1:n}\right) = \frac{1}{Z} \prod_{k \in \{1,\ldots,n\}} \phi\left(x_k\right) \prod_{(i,j) \in T} \phi_{i,j}\left(x_i, x_j\right) \tag{4.1}$$

Below, T means the set of edges in the tree

and thus, if we want to compute $P(X_1)$ we can marginalize out all other variables

$$\begin{aligned} P(X_1) = \sum_{x_2,\ldots,x_n} P(X_{1:n}) &= \sum_{x_2,\ldots,x_n} \frac{1}{Z} \prod_k \phi\left(x_k\right) \prod_{(i,j)\in T} \phi_{i,j}\left(x_i, x_j\right) && (4.2) \\ &\propto \sum_{x_2,\ldots,x_n} \prod_k \phi\left(x_k\right) \prod_{(i,j)\in T} \phi_{i,j}\left(x_i, x_j\right) && (4.3) \end{aligned}$$

Now that we have the goal, we can group what we want to compute into a more structured representation, where we define ***message-passing*** from variable j to $i \in N(j)$ as

$$m_{j\to i}\left(x_i\right) = \sum_{x_j} \phi_j\left(x_j\right) \phi_{ij}\left(x_i, x_j\right) \prod_{k \in N(j) \neq i} m_{k \to j}\left(x_j\right) \tag{4.4}$$

4.1.2 Belief Propagation Algorithm

1. Choose an root r arbitrarily,

2. Pass messages from leaves to r

3. Pass messages from r to leaves

4. Compute
$$p\left(x_i\right) \propto \phi_i\left(x_i\right) \prod_{j \in \mathcal{N}(i)} m_{j\to i}\left(x_i\right), \forall_i \tag{4.5}$$

Step 2 and 3 here so that we will compute all the $m_{i\to j}$, $\forall(i,j)$, $(j,i) \in T$, after which we can use them to calculate the marginals using formula in step 4

Notice that the idea is that we want to calculate ***all*** of the marginals

4.2 Inference in Hidden Markov Models

There are the following four kinds of inference that we can do on a HMM, namely

- ***Filtering:*** Compute the belief state $p\left(z_t|\mathbf{x}_{1:t}\right)$ online. Notice that we call this inference operation filtering because this produces a result smoother than simply computing $p(z_t|\mathbf{x}_t)$.

- ***Smoothing:*** Compute $p\left(z_t|\mathbf{x}_{1:T}\right)$ offline. Notice that this computation happens after all $\mathbf{x}_{1:T}$ observations have been collected and hence is offline.[4.1]

Definition in KPM is different from course note

- ***Fixed lag smoothing:*** Compute $p\left(z_{t-\ell}|\mathbf{x}_{1:t}\right)$, where $\ell > 0$ is called the lag. Essentially, we want to do inference on the hidden state z_α and we wait until we observe all of $\mathbf{x}_{1:\alpha+\ell}$ before we carry out the computation.

- ***Prediction:*** In this case, we want to compute $p\left(z_{t+h}|\mathbf{x}_{1:t}\right)$ for some $h > 0$. (h is called the prediction horizon) Let's first take a look at the example where $h = 2$, then

$$\begin{aligned} p(z_{t+2}|\mathbf{x}_{1:t}) &= \sum_{z_{t+1}}\sum_{z_t} p(z_t, z_{t+1}, z_{t+2}|\mathbf{x}_{1:t}) && (4.6)\\ &= \sum_{z_{t+1}}\sum_{z_t} p(z_t|\mathbf{x}_{1:t})p(z_{t+1}|z_t, \mathbf{x}_{1:t})p(z_{t+2}|z_{t+1}, z_t, \mathbf{x}_{1:t}) && (4.7)\\ &= \sum_{z_{t+1}}\sum_{z_t} p\left(z_{t+2}|z_{t+1}\right)p\left(z_{t+1}|z_t\right)p\left(z_t|\mathbf{x}_{1:t}\right) && (4.8) \end{aligned}$$

where the last step of simplification could be easily justified using Bayes Ball rules. Above, we computed the prediction about the future hidden states, and it can be converted into prediction about the future observations using

$$p\left(\mathbf{x}_{t+h}|\mathbf{x}_{1:t}\right) = \sum_{z_{t+h}} p\left(\mathbf{x}_{t+h}|z_{t+h}\right)p\left(z_{t+h}|\mathbf{x}_{1:t}\right) \qquad (4.9)$$

which is called ***Posterior Predictive Density***.

4.3 The Forward - Backward Algorithm

Suppose that we want to find $p(z_t|x_{1:T}), \forall t$, then it can be broken down into two pieces

- ***Forward Filtering:*** Compute $p\left(z_t, x_{1:t}\right) \quad \forall t$, which effectively is the probability of the hidden state z_t given all past observations, up to including x_t.

- ***Backward Filtering:*** Compute $p\left(x_{1+t:T}|z_t\right) \quad \forall t$, which computes the probability of *all* future observations from x_{t+1} up to including x_T given the current hidden state, i.e. z_t.

Then, the complete smoothing $p(z_t|x_{1:T})$ could be computed as

This is called **Latent sequence given observations in HMM** in the course notes

$$\begin{aligned} p\left(z_t|x_{1:T}\right) &= p\left(x_{1:T}\right)p\left(z_t, x_{1:T}\right) && (4.10)\\ &\propto p\left(z_t, x_{1:T}\right) && (4.11)\\ &= p\left(z_t, x_{1:t}\right)p\left(x_{t+1:T}|z_t, x_{1:t}\right) && (4.12)\\ &= p\left(z_t, x_{1:t}\right)p\left(x_{t+1:T}|z_t\right) \qquad \text{// By Bayes Ball} && (4.13)\\ &= (\text{Forward Recursion})(\text{Backward Recursion}) && (4.14) \end{aligned}$$

[4.1] A very good intuitive example from KPM is that we have a detective investigating a crime scene, and as we gather more observation, the prediction that we make tends to be more and more certain and less 'stochastic'.

4.3.1 Forward Filtering Recursion

$$\begin{aligned}
\alpha_t(z_t) = p\left(z_t, x_{1:t}\right) &= \sum_{z_{t-1}} p\left(z_{t-1}, z_t, x_{1:t}\right) \qquad (4.15)\\
&= \sum_{z_{t-1}} p\left(x_t|z_{t-1}, z_t, x_{1:t-1}\right) p\left(z_t|z_{t-1}, x_{1:t-1}\right) \underbrace{p\left(z_{t-1}, x_{1:t-1}\right)}_{=\alpha_{t-1}(z_{t-1})} \qquad (4.16)\\
\implies \alpha_t\left(z_t\right) &= p\left(x_t|z_t\right) \sum_{z_{t-1}} p\left(z_t|z_{t-1}\right) \alpha_{t-1}\left(z_{t-1}\right) \qquad (4.17)
\end{aligned}$$

Notice that our forward recursion contains our emission, $p(x_t|z_t)$ and transition $p(z_t|z_{t-1})$. The base case of the recursion could be unwinded into

$$\alpha_1\left(z_1\right) = p\left(z_1, x_1\right) = p\left(z_1\right) p\left(x_1|z_1\right) \tag{4.18}$$

4.3.2 Backward Filtering Recursion

$$\begin{aligned}
p\left(x_{t+1:T}|z_t\right) &= \sum_{z_{t+1}} p\left(z_{t+1}, x_{t+1:T}|z_t\right) \qquad (4.19)\\
&= \sum_{z_{t+1}} p\left(x_{t+2:T}|z_{t+1}, z_t, x_{t+1}\right) p\left(x_{t+1}|z_{t+1}, z_t\right) p\left(z_{t+1}|z_t\right) \qquad (4.20)\\
\implies \beta_t\left(z_t\right) &= \sum_{z_{t+1}} \underbrace{p\left(x_{t+2:T}|z_{t+1}\right)}_{=\beta_{t+1}(z_{t+1})} p\left(x_{t+1}|z_{t+1}\right) p\left(z_{t+1}|z_t\right) \qquad (4.21)
\end{aligned}$$

and hence if we recurse the above relationship, we will unwind to the base case

$$\beta_1\left(z_1\right) = p\left(x_{3:T}|z_2\right) p\left(x_2|z_2\right) p\left(z_2|z_1\right) \tag{4.22}$$

4.4 Sampling

The word sample here refers to a single realization from a distribution rather than a set.

The goal is to,

- Generate sample $\left\{x^{(r)}\right\}_{r=1}^{R}$ from a probability distribution $p(x)$, and/or
- Estimate expectations of functions $f(x)$ under some distribution $p(x)$, usually we want to estimate moments for a distribution

$$E = \mathop{\mathbb{E}}_{x\sim p(x)}[f(x)] = \int f(x)p(x)dx \tag{4.23}$$

4.4.1 Ancestral Sampling

Generating marginal samples If you are only interested in sampling a particular set of nodes, you can simply sample from all the nodes jointly, then ignore the nodes you don't need.

Generating conditional samples If you want to sample conditional on a node with no parents, that's also easy - you can simple do ancestral sampling starting from the nodes you have.

(#! Important:) However, to sample from a DAG conditional on leaf nodes is hard in the same way that inference is hard in general. E.g. sampling the unknown key in a crypto-system given the cypher-text but not knowing the plaintext. Finding ways to do this approximately is what a lot of the rest of the course will be about.

4.4.2 Simple Monte Carlo

(Definition) - Simple MC Given $\left\{x^{(r)}\right\}_{r=1}^{R} \sim p(x)$, we want to estimate the expectation $\underset{x\sim p(x)}{\mathbb{E}}[f(x)]$.

$$E = \underset{x\sim p(x)}{\mathbb{E}}[f(x)] \approx \frac{1}{R}\sum_{r=1}^{R} f\left(x^{(r)}\right) = \hat{E} \tag{4.24}$$

Unbiasedness of MC

$$\mathbb{E}_{x\sim p\left(\left\{x^{(i)}\right\}_{r=1}^{R}\right)}[\hat{E}] = \mathbb{E}\left[\frac{1}{R}\sum_{r=1}^{R} f\left(x^{(r)}\right)\right] \tag{4.25}$$

$$= \frac{1}{R}\sum_{r=1}^{R} \mathbb{E}\left[f\left(x^{(r)}\right)\right] \tag{4.26}$$

$$= \frac{1}{R}\sum_{r=1}^{R} \underset{x\sim p(x)}{\mathbb{E}}[f(x)] \tag{4.27}$$

$$= \frac{R}{R}\underset{x\sim p(x)}{\mathbb{E}}[f(x)] \tag{4.28}$$

$$= E \tag{4.29}$$

■

Variance of MC

$$\text{var}[\hat{E}] = \text{var}\left[\frac{1}{R}\sum_{r=1}^{R} f\left(x^{(r)}\right)\right] \tag{4.30}$$

$$= \frac{1}{R^2}\,\text{var}\left[\sum_{r=1}^{R} f\left(x^{(r)}\right)\right] \tag{4.31}$$

$$= \frac{1}{R^2}\sum_{r=1}^{R} \text{var}\left[f\left(x^{(r)}\right)\right] \tag{4.32}$$

$$= \frac{1}{R}\,\text{var}\left[f(x)\right] \tag{4.33}$$

We notice that the accuracy of MC estimates only depends on the variance of f, not on the dimension of x. Also, we the number of samples, R, increases, the variance $\hat{E}$ will decrease at a rate proportional to $\frac{1}{R}$. ■

5 Lecture 7 - Stochastic Variational Inference (SVI/ADVI)

5.1 Motivation - Approximating Posterior Inference

Notice that if we want to calculate the *exact* posterior distribution, for example,

$$p(z|x) = \frac{p(x|z)}{p(x)} = \frac{p(x,z)p(z)}{\int p(x,z)dz} \tag{5.1}$$

the bottom integral over all possible states soon becomes intractable, making the computation of the entire posterior distribution intractable. Hence, instead, we will approximate the posterior inference with ***variational methods***. Generally speaking, it works as follows

1. We introduce a family of distributions, with variational parameters ϕ, denoted as q_ϕ.
2. We want to encode a "distance metric" between $p(z|x)$ and $q_\phi(z)$.
3. Try to minimize the proposed "distance metric".

There are a few things to note

- First, as we will soon see, we use a measure of closeness that is ***not*** a distance metric. Also, using naïve distance metric could fail spectacularly (arbitrarily bad examples can be found if we use MSE).
- This formulation turns the Bayesian Inference into an optimization problem. If enough parts of the model is differentiable and could be well-approximated with Monte Carlo, then we can use gradient based optimization methods to solve this problem scalably.

5.2 The Kullback-Leibler Divergence

Suppose q_ϕ denotes a family of distributions with its own **variational parameters** ϕ, and p is some distribution. Then, the Kullback-Leibler divergence is defined as

$$\begin{aligned} D_{\mathrm{KL}}\left(q_\phi(z|x) \| p(z|x)\right) &= \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z|x)} dz && (5.2) \\ &= \mathop{\mathbb{E}}_{z \sim q_\phi}\left[\log \frac{q_\phi(z|x)}{p(z|x)}\right] && (5.3) \end{aligned}$$

Properties of KL Divergence

1. $D_{\mathrm{KL}}(q_\phi || p) \geq 0$, non-negativeness
2. $D_{\mathrm{KL}}(q_\phi || p) = 0 \iff q_\phi = p$
3. $D_{\mathrm{KL}}(q_\phi || p) \neq D_{\mathrm{KL}}(p || q_\phi)$, i.e. KL Divergence is ***not*** a metric distance.

5.2.1 Expectation Propagation

If we try to reverse the arguments, it leads to a different kind of variational inference that is called "expectation propagation". Typically, this leads to an algorithm that is more computationally expensive.

5.3 The Evidence Lower BOund (ELBO)

5.3.1 Definition and Equivalence Property

We cannot minimize the KL divergence exactly, thus we introduce a function, ELBO, and try to minimize that. Notice that ELBO is equal to the actual KL divergence up to a constant.

$D_{\mathrm{KL}}(q_\phi \| p)$ is intractable because it contains $p(z|x)$, which is intractable.

$$
\begin{aligned}
D_{\mathrm{KL}}\left(q_\phi(z|x) \| p(z|x)\right) &= \underset{z\sim q_\phi}{\mathbb{E}} \log \frac{q_\phi(z|x)}{p(z|x)} && (5.4)\\
&= \underset{z\sim q_\phi}{\mathbb{E}} \left[\log\left(q_\phi(z|x) \cdot \frac{p(x)}{p(z,x)}\right)\right] && (5.5)\\
&= \underset{z\sim q_\phi}{\mathbb{E}} \log \frac{q_\phi(z|x)}{p(z,x)} + \underset{z\sim q_\phi}{\mathbb{E}} \log p(x) && (5.6)\\
&= -\mathcal{L}(\phi; x) + \log p(x) && (5.7)
\end{aligned}
$$

where $\mathcal{L}(\phi; x) = -\underset{z\sim q_\phi}{\mathbb{E}} \log \frac{q_\phi(z|x)}{p(z,x)}$ is defined as the ELBO. If we re-arrange what we have above,

$$
\log p(x) = \mathcal{L}(\phi; x) + D_{\mathrm{KL}}\left(q_\phi(z|x) \| p(z|x)\right) \tag{5.8}
$$

and we notice that $D_{\mathrm{KL}}\left(q_\phi(z|x) \| p(z|x)\right) \geq 0$, it then follows that

$$
\mathcal{L}(\phi; x) \leq \log p(x) \tag{5.9}
$$

Hence, to minimize the KL divergence we need to maximize the ELBO, and they are equivalent.

5.3.2 Optimization goal of ELBO

We have, by definition of ELBO, that

$$
\begin{aligned}
\mathcal{L}(\phi; x) &= -\underset{z\sim q_\phi}{\mathbb{E}} \log \frac{q_\phi(z|x)}{p(x,z)} && (5.10)\\
&= \underset{z\sim q_\phi}{\mathbb{E}} \left[\log p(x,z) - \log q_\phi(z|x)\right] && (5.11)
\end{aligned}
$$

Recall, that our goal is to use gradient based methods to optimize this objective, meaning that we have to somehow compute, estimate to be precise in this case, the gradient of ELBO $\mathcal{L}(\phi; x)$

$$
\nabla_\phi \mathcal{L}(\phi) = \nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p(x,z) - \log q_\phi(z|x)\right] \tag{5.12}
$$

and we know from a previous lecture that one 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.

5.3.3 Pathwise Gradient

Above, we saw that the objective is the gradient over an expectation and we now would like to make it an expectation over gradient. In general, such swapping is not allowed and we can only do so if the distribution we're taking the expectation over ***does not*** depend on the parameter, i.e. ϕ. Our goal is now to factor our the randomness of from q, and put it into a parameterless, fixed source of noice $p(\varepsilon)$. Formally, we need to find function $T(\phi, \varepsilon)$ such that

$$\left.\begin{aligned} \varepsilon &\sim p(\varepsilon) \\ z &= T(\phi, \varepsilon) \end{aligned}\right\} \implies z \sim q_\phi(z) \tag{5.13}$$

We usually start with $p(\varepsilon)$ being uniform or normal. Indeed it is not always easy to find these functions[5.1] and as a easy concrete example, we know

$$\left.\begin{aligned} \varepsilon &\sim \mathcal{N}(\varepsilon|0,1) \\ z &= \sigma\varepsilon + \mu \end{aligned}\right\} \implies z \sim \mathcal{N}(z|\mu, \sigma) \tag{5.14}$$

Then, by applying the trick we know,

$$\begin{aligned} \nabla_\phi \mathcal{L}(\phi) &= \nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p(x,z) - \log q_\phi(z|x)\right] && (5.15)\\ &= \nabla_\phi \mathbb{E}_{\varepsilon\sim p(\varepsilon)} \left[\log p(x, T(\phi,\varepsilon)) - \log q_\phi(T(\phi,\varepsilon)|x)\right] && (5.16)\\ &= \mathbb{E}_{\varepsilon\sim p(\varepsilon)} \nabla_\phi \left[\log p(x, T(\phi,\varepsilon)) - \log q_\phi(T(\phi,\varepsilon)|x)\right] && (5.17) \end{aligned}$$

5.4 Tutorial

5.4.1 Derivation Using Jensen's Inequality

Recall that by when Jensen's inequality is applied to probability distributions, we know that when f is concave,

$$f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)] \tag{5.18}$$

Then, for the log probability of the observations,

$$\begin{aligned} \log p(x) &= \log \int p(x,z)dz && (5.19)\\ &= \log \int p(x,z)\frac{q_\phi(z|x)}{q_\phi(z|x)}dz && (5.20)\\ &= \log \mathbb{E}_{z\sim q_\phi} \frac{p(x,z)}{q_\phi(z|x)} && (5.21) \end{aligned}$$

where we apply Jensen's Inequality

$$\begin{aligned} \implies \mathop{\mathbb{E}}_{z\sim q_\phi} \frac{p(x,z)}{q_\phi(z|x)} &\geq \mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{p(x,z)}{q_\phi(z|x)} && (5.22)\\ &= -\mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{q_\phi(z|x)}{p(x,z)} && (5.23)\\ &= \mathcal{L}(\phi; x) && (5.24) \end{aligned}$$

[5.1] http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/

5.4.2 Interpretations of ELBO

For the ELBO defined as

$$\text{ELBO} \triangleq \mathcal{L}(\phi; x) = -\mathop{\mathbb{E}}_{z\sim q_\theta} \log \frac{q_\phi(z|x)}{p(x,z)} \tag{5.25}$$

we have

1. The most general interpretation is just breaking up the logarithm. We have

$$\begin{aligned} \mathcal{L}(\phi; x) &= -\mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{q_\phi(z|x)}{p(x,z)} && (5.26) \\ &= \mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{p(x,z)}{q_\phi(z|x)} && (5.27) \\ &= \mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{p(z)p(x|z)}{q_\phi(z|x)} && (5.28) \\ &= \mathop{\mathbb{E}}_{z\sim q_\phi} [\log p(x|z) + \log p(z) - \log q_\phi(z|x)] && (5.29) \end{aligned}$$

2. We can try to write this using entropy as

$$\mathcal{L}(\phi; x) = \mathop{\mathbb{E}}_{z\sim q_\phi} [\log p(x|z) + \log p(z)] \mathcal{H}\left(q_\phi(z|x)\right) \tag{5.30}$$

3. We can frame ELBO as a trade off

$$\mathcal{L}(\phi; x) = \underbrace{\mathop{\mathbb{E}}_{z\sim q_\phi} [\log p(x|z)]}_{(\dagger)} - \underbrace{D_{\text{KL}}\left(q_\phi(z|x) \| p(z)\right)}_{(\ddagger)} \tag{5.31}$$

   - (†) is "reconstruction likelihood", i.e. how probable is x given z, which encourages the model to choose the distribution which best reconstructs the data.
   - (‡) acts as regularization, enforcing the idea that our parametrization shouldn't move us too far from the true distribution.

6 Lecture 8 - Sampling and Monte Carlo Methods

6.1 Motivation: Sampling

Note: In this section, we refer to a "sample" as a single realization x that is from a probability distribution $p(x)$.[6.1]

Problems to be solved Monte Carlo methods are computational techniques that make use of random numbers, it aims to solve one or both of the following problems

1. To generate samples

$$\{x^{(r)}\}_{r=1}^R \sim p(x) \tag{6.1}$$

2. and to estimate expectations of functions ($\phi(x)$) under the distribution of $p(x)$, i.e.,

$$\Phi = \mathop{\mathbb{E}}_{x\sim p(x)} [\phi(x)] = \int \phi(x)p(x)dx \tag{6.2}$$

[6.1] This is different from the normal sense of sample, which means one batch of sample from a distribution.

Sampling $p(x)$ is like hornets' nest To sample from

$$p(x) = \frac{\tilde{p}(x)}{Z} \tag{6.3}$$

there are two main complications even if we know how to evaluate $\tilde{p}(x)$

1. We don't typically know the normalizing constant, Z.
2. Even if we did know Z, the problem of drawing samples from $p(x)$ is still a chanlenging one, especially in high-dimensional spaces, because there is no obvious way to sample from p without enumerating most or all of the possible states.

Should this be $\tilde{p}$??

Note: There are only a handful of distribution, for example gaussian distribution, that we can easily sample from in a high dimensional space.

6.2 Lattice Discretization

Imagine that we wish to draw samples from the density $p(x) = \frac{\tilde{p}(x)}{Z}$, then what we can do is discretize the variable x and sample from the discrete probability distribution over a finite set of uniformly spaced points $S = \{x_i\}_i$. Then for $x_i \in S$, we evaluate $\tilde{p}(x_i)$ after which we can compute the normalizer

$$Z = \sum_i \tilde{p}_i(x_i) \tag{6.4}$$

and

$$p_i = \frac{\tilde{p}_i}{Z} \tag{6.5}$$

The main problem with this method, however, comes from the accuracy and cost of evaluating at lattice. If we want to sample D uniformly spaced points in one dimension and the system has N dimensions in total. Then, the the complexity is asymptotically $\mathcal{O}(D^N)$ which grows exponentially as dimension grows and soon become intractable.

6.3 Importance Sampling

Note: Importance Sampling doesn't help us solve the problem of generating samples from $p(x)$, rather it is a method that helps us estimate expectation of a function $\phi(x)$, i.e. problem 2.

Assumptions

1. To do importance sampling, we still need to assume that we can evaluate $p(x)$ (density that we wish to draw samples from) within a multiplicative constant. To be precise, we can evaluate a function $\tilde{p}(x)$ such that

$$p(x) = \tilde{p}(x)/Z \tag{6.6}$$

2. We further assume that we have a simpler density, $q(x)$ from which it is easy to ***sample*** from, i.e., $x \sim q(x)$ and easy to ***evaluate***, i.e., $tildeq(x)$

$$q(x) = \frac{\tilde{q}(x)}{Z_q} \tag{6.7}$$

where the density $q(x)$ is referred to as the sampler density.

Sampling Procedure In importance sampling, we generate in total R samples from $q(x)$, i.e. $\left\{x^{(r)}\right\}_{r=1}^{R} \sim q(x)$. If these points were samples from $p(x)$ then we could estimate Φ by

$$\Phi = \mathop{\mathbb{E}}_{x \sim p(x)}[\phi(x)] \approx \frac{1}{R}\sum_{r=1}^{R} \phi\left(x^{(r)}\right) = \hat{\Phi} \tag{6.8}$$

i.e., we could use a *Simple Monte Carlo Estimator.* The problem is that these sample are from q rather than p which can be completely unrelated densities. To solve this over-represented/under-represented problem, we can take into account that we are sampling from a different distribution by introducing weights, (essentially corrections)

$$\tilde{w}_r = \frac{\tilde{p}\left(x^{(r)}\right)}{\tilde{q}\left(x^{(r)}\right)} \tag{6.9}$$

Finally, we re-write our estimator under q, i.e.

$$\begin{aligned}
\Phi &= \int \phi(x)p(x)dx && (6.10)\\
&= \int \phi(x)\frac{p(x)}{q(x)}q(x)dx && (6.11)\\
&\approx \frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\frac{p\left(x^{(r)}\right)}{q\left(x^{(r)}\right)} && (6.12)\\
&= \frac{Z_q}{Z_p}\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\frac{\tilde{p}(x^{(r)})}{\tilde{q}(x^{(r)})} && (6.13)\\
&= \frac{Z_q}{Z_p}\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\tilde{w}_r && \text{since } w_r = \frac{\tilde{w}_r}{\sum_{r=1}^{R}\tilde{w}_r} \quad (6.14)\\
&= \frac{\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\tilde{w}_r}{\frac{1}{R}\sum_{r=1}^{R}\tilde{w}_r} && \text{since } \frac{Z_p}{Z_q} = \frac{1}{R}\sum_{r=1}^{R}\tilde{w}_r \quad (6.15)\\
&= \frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot w_r && (6.16)\\
&= \hat{\Phi}_{iw} && (6.17)
\end{aligned}$$

The estimator $\hat{\Phi}_{iw}$ is biased, but is consistent.

Pitfalls As it is easy to see, if we choose q to be some short tailed distribution, we might not get a good coverage for samples on p, and thus making the estimation not accurate. Notice that we want to support the entire real line with this method[6.2] and hence we cannot use a naïve uniform over a certain range, and using uniform over the entire real line will cause uniform distribution pdf to be infinitely small everywhere.

[6.2] Or otherwise this will just be uniform sampling. Indeed, importance sampling is a generalization of uniform sampling.

Remedy As a way to (semi-) solve the above pitfall, we can choose q to be a heavy-tailed distribution, for example Cauchy. Of course, the best of all solution is to find a q that looks like p, but this is in a lot of cases an un achievable goal.

6.4 Rejection Sampling

In rejection sampling we assume again a one-dimensional density $p(x) = \tilde{p}(x)/Z$ that is too complicated a function for us to be able to sample from it directly. We assume that we have a simpler ***proposal*** density $q(x)$ which we can evaluate (within a multiplicative factor Z_q, as before), and from which we can generate samples. We further assume that we know the value of a constant c such that

$$c\tilde{q}(x) > \tilde{p}(x) \quad \text{for some } c \text{ constant}\,, \forall x \tag{6.18}$$

It is somewhat important to note that in rejection sampling we care only about the proportion, and we need a q, or $\tilde{q}$ actually, that "covers" p, or $\tilde{p}$ actually. This means that we don't care, and thus don't need to know the normalizing constant to perform the sampling procedure. Naïvely, we can just scale q to a huge $\tilde{q}$, and as long as it covers $\tilde{p}$, we are in a good shape.

Sampling Procedure The procedure works as follows

1. Generate two random numbers, the first x is such that $x \sim q(x)$, and the second u is generated such that $u \sim Unif[0, c\tilde{q}(x)]$
2. Evaluate $\tilde{p}(x)$ and accept or reject the sample x by comparing the value of u with the value of $\tilde{p}(x)$, to be more specific
   - If $u > \tilde{p}(x)$, then we reject x,
   - otherwise we accept x, effectively adding x to our set of samples. (value of u is discarded, and in the next iteration u will be sampled again from, possibly, a different uniform distribution.)

Pitfalls

- Rejection sampling works the best if we are able to find q that is a good approximation to p. If q is very different from p then, for cq to exceed p everywhere, c will almost always necessarily be large. This will cause us a very high rejection ratio, and making the algorithm super inefficient.
- In high-dimensional space, the problem is even worse and it is very likely that the requirement that $c\tilde{q} > \tilde{p}$ will force c to be so huge that acceptances will be very rare indeed.
- Besides, finding such a value of c may be difficult too, since in many problems we know neither where the modes of $\tilde{p}$ 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, since

$$\text{acceptance rate } = \frac{\text{area under } \tilde{p}}{\text{area under } c\tilde{q}} = \frac{1}{Z} \tag{6.19}$$

vas ist das?

6.5 Metropolis-Hasting Method: Example of MCMC

The proposed importance sampling and rejection sampling work well only if the proposal density $q(x)$ is similar to $p(x)$. In general, in a high dimensional case, such q is very hard to find. In Metropolis-Hasting, in contrast to importance and rejection sampling methods, it is not necessary for $q(x'|x^{(t)})$ to lock similar to $p(x)$ in order for the algorithm to be useful. The density $q(x'|x^{(t)})$ might be a simple distribution such as a gaussian centred at $x^{(t)}$, but can in general be any *fixed* density from which we can draw samples.

Sampling Procedure

- A tentative new state x' is generated from the proposal density $q(x'|x^{(t)})$, then we compute
$$a = \frac{\tilde{p}\left(x'\right) q\left(x^{(t)}|x'\right)}{\tilde{p}\left(x^{(t)}\right) q\left(x'|x^{(t)}\right)} \tag{6.20}$$
- If $a \geq 1$, then the new state is accepted.
- Otherwise, the new state is accepted with probability a. Then, we perform update as follows,
$$x^{(t+1)} \leftarrow \text{accepted ? } x' \ : \ x^{(t)} \tag{6.21}$$

Note on Difference Notice that in rejection sampling, rejected points are discarded and have no influence on the list of samples $x_r^{(r)}$ that we collected. Here, a rejection causes the current state to be written again onto the list.

Convergence and MCMC Metropolis-Hastings converges to $p(x)$ for any $q\left(x'|x^{(t)}\right) \geq 0 \quad \forall x', x^{(t)}$ as $t \to \infty$. That is, out list of samples is such that
$$\{x^{(r)}\}_{r=1}^{R} \to p(x) \tag{6.22}$$

The Metropolis-Hastings method is an example of a Markov Chain Monte Carlo method. In contrast to rejection sampling, where the accepted points are independent samples from the desired distribution, MCMC methods involve a Markov process in which a sequence of states 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 run for a considerable time in order to generate samples that are effectively independent samples from p.

7 Lecture 9 - Amortized Inference and Variational Auto Encoders

7.1 Motivations

7.1.1 Efficient Use of (Accumulative) Partial Evidence

With enough experience, doctors, plumbers, detectives, etc. can very quickly tell what is going on and what they still need more information about, if they've seen enough similar situations. Hence, perhaps we could somehow train a neural network to look at the data for a person x_i, and then output an approximate posterior $q_{\phi_i}(z_i|x_i)$

7.2 Amortized Inference

Here "amortized" essentially means to spread out over time. Previously, we do SVI from scratch every time we see a new datapoint. Now, instead, we will gradually learn a function that can look at the data for a person x_i, and then output an approximate posterior $q_\phi(z_i|x_i)$. This is known as a ***recognition model***, instead of a separate ϕ_i for each data example, we will just have a single global ϕ that specifies the parameter of the recognition model. Because the relationship between data and posteriors is complex and hard to specify by hand, we will do this through a neural net. (we make the neural net learn the function!)

7.2.1 Simple Example (Parametrizing Gaussian)

We have seen one way to specify a probability distribution given an input with neural networks in assignment 2: we can simply have a network take in x_i, and output the mean and variance vector for a Gaussian, parametrized as

$$q_\phi(z_i|x_i) = \mathcal{N}(z_i|\mu_\phi(x_i), \Sigma_\phi(x_i)) \tag{7.1}$$

The graphical model for such model specification has ϕ ***outside of*** the plate, which means we only have one global ϕ to update and optimize.

7.2.2 Amortized Inference - Algorithmic Procedure

1. Sample a datapoint
2. compute parameters of approximate posterior (recognition)
3. compute gradient of Monte Carlo estimate of ELBO with respect to ϕ, i.e. $\nabla_\phi \overset{\sim MC}{ELBO}$
4. update according to gradient descent

Then, when we want to make predictions about a new datapoint, we can use out fast recognition model if we want. Of course, we might also want to stop and do something slower but more accurate, for example like per-sample SVI, and/or MCMC.

7.2.3 Optimizing Model Parameters

Now that $p_\theta(x)$ depends on parameters θ, then the ELBO is a function of both ϕ and θ. We can optimize them together, still, using the re-parametrize trick we saw earlier.

$$\nabla_{\theta,\phi}\mathcal{L}(\phi) = \nabla_{\theta,\phi}\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x,z) - \log q_\phi(z|x)\right] \tag{7.2}$$

$$= \mathbb{E}_{\epsilon\sim p(\epsilon)}\nabla_{\theta,\phi}\left[\log p_\theta(x, T(\phi,\epsilon)) - \log q_\phi(T(\phi,\epsilon)|x)\right] \tag{7.3}$$

This allows us to jointly fit the model parameters and the recognition network, by subsampling training examples and using simple Monte Carlo with gradient descent optimizers. Such formulation is referred to as a variational auto encoder, VAE (See Figure 7.1 for layout of architecture).

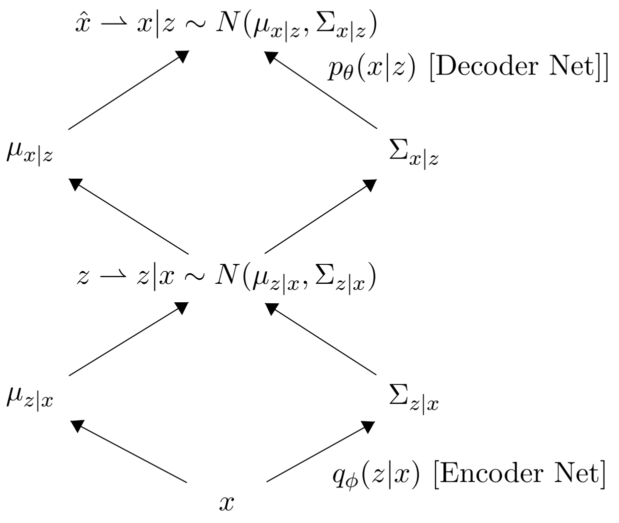

$\hat{x} \rightharpoonup x|z \sim N(\mu_{x|z}, \Sigma_{x|z})$
$p_\theta(x|z)$ [Decoder Net]]
$\mu_{x|z}$
$\Sigma_{x|z}$
$z \rightharpoonup z|x \sim N(\mu_{z|x}, \Sigma_{z|x})$
$\mu_{z|x}$
$\Sigma_{z|x}$
$q_\phi(z|x)$ [Encoder Net]
x

Figure 7.1: Variational Autoencoder Architecture

7.3 Variational Auto Encoders (VAEs)

7.3.1 Autoencoders: Definition and Problems

An autoencoder takes an input x, and encodes it into a code vector z[7.1], after which the code vector z is decoded into $\hat{x}$ that is "similar" to x, our original data. Essentially, what we want to learn is two mappings, namely the encoder $g(x) \to z$ and the decoder $f(z) \to \hat{x}$. In general, g and f could be arbitrarily complex, and thus we model them neural nets. The optimization goal of such problem is to minimize the reconstruction error, i.e.

$$\mathcal{J} = \texttt{reconstruction\_error}(x, \hat{x}) \tag{7.4}$$

Unsupervised Nature of Autoencoders Notice that here the cost defined in equation 7.4 doesn't rely on *any* explicit label input. Rather, it is a form of unsupervised learning, where the learning goal is to learn the mappings to reconstruct the inputs, i.e. $\tilde{x} = f \circ g(x) \approxeq x$.

7.3.2 Problems with Deterministic Autoencoders

Proximity in data space doesn't mean proximity in feature space The codes that we learn by the model is deterministic, i.e.

$$\begin{aligned} g\left(x_1\right) = z_1 \Rightarrow f\left(z_1\right) = \tilde{x}_1 \\ g\left(x_2\right) = z_2 \Rightarrow f\left(z_2\right) = \tilde{x}_2 \end{aligned} \tag{7.5}$$

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

$$z_1 \approx z_2 \not\Longrightarrow x_1 \approx x_2 \tag{7.6}$$

[7.1] Notice that for autoencoders to actually "encode" the inputs, we necessarily have to choose the dimension of z so that it forms a bottleneck. If we allow z to have the same (or even larger) dimensionally with/than x and $\hat{x}$, then likely the model will learn two identity mappings!

If the space has regions where no data gets encoded to, and you sample/generate a variation from there, the decoder will generate an unrealistic output, because the decoder has no idea how to deal with that region of the latent space. During training, it never saw encoded vectors coming from that region of latent space

Adding Noise By Hand is Hard We said that we wish to embed x's into a lower dimension space, and code it as z. However, in practice it is very hard to determine what the low dimension z should be. Similar to this,

1. We can add noise to data before encoding, after which we try to reconstruct the original data. But how much noise?

2. We can add noise to the latent variable, code, z after encoding, after which we try to reconstruct the original data. Again, how much noise should we add?

7.3.3 Variational Auto Encoders

In an hope to address the problems above, we introduce stochasticity into out model: even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling. Figure 7.1 illustrates the architecture of a variational autoencoder.

VAE Solves Problems in Section 7.3.2, WHY? The VAE generation model learns to reconstruct its inputs not only from the encoded points but also from the area around them. This allows the generation model to generate new data by sampling from an "area/volume" instead of only being able to generate already seen data corresponding to the particular fixed encoded points.

7.3.4 Utilization of Latent Variable Models

The question is: once we have obtained the model, what do we do with it?

If a latent variable model has a compact prior and vector valued code z's, we can

1. Sample new data to check the model,

2. Encode (sample from the approximate posterior) of two data points, and interpolate between them in the latent space, then decode along the path

3. Learn a function to predict some property of the examples from the low-dimensional z space instead of directly rom x. (semi-supervised learning, essentially use unsupervised learning to cleanup the data, and to learn a "better" representation.)

8 Lecture 10 - Normalizing Flows

Normalizing Flows, just like VAEs (Section 7.3) and GANs (Section 9), is a generative modelling method.

Definition A normalizing flow is a learnt transformation from a simple distribution (base[8.1]) to a complex distribution (target). The goal is to make the complex distribution look like our data. The learnt transformation is a composition of a sequence of mappings that are

- invertible, and
- differentiable

Let's now discuss this idea more formally and explain why we would need the mappings to be invertible and differentiable.

More Formal Objective Suppose that you have a random variable $\mathbb{R}^D \ni Z \sim p_Z$ at hand. The distribution p_Z is called a ***base distribution***, which has to satisfy the following

- known,
- tractable to sample, and
- tractable to evaluate density $p_Z(\cdot)$

A typical candidate for the base distribution would be a simple gaussian. The ***target distribution*** $\mathbb{R}^D \ni X \sim p_X$ is a complex distribution transformed from p_Z in a tractable way.

8.1 The Transformation - Generative and Normalizing Directions

Generative Direction is the direction where we compute $X = f(Z)$, where f is (1) invertible and (2) differentiable with Jacobian

$$\partial f(\overline{Z}) = \left.\frac{\mathrm{d}f}{\mathrm{d}Z}\right|_{Z=\overline{Z}} \tag{8.1}$$

Normalizing Direction is the reverse of the above operation, i.e.

$$Z = g(X) = f^{-1}(X) \tag{8.2}$$

where g is (1) trivially invertible[8.2] and (2) differentiable with Jacobian

$$\partial g(\overline{X}) = \left.\frac{\mathrm{d}g}{\mathrm{d}X}\right|_{X=\overline{X}} \tag{8.3}$$

8.2 Arbitrary Expressiveness

It has been shown that if f, the generative direction mapping, is arbitrarily expressive, then p_Z can be transformed into ***any*** p_X.

[8.1] Although we call this simple distribution a base distribution, we should really think of it as a gaussian. (Aside: this is where the name of "normalizing" comes from.)

[8.2] since it is defined as inverse of f

8.3 Generative Goals

- ***Sampling/Generating:*** should be easy to do, since $Z \sim p_Z$ is by design easy to sample from, and from there we can compute the tractable transformation $X = f(Z)$.
- ***Evaluating Density:*** suppose that we have $X \sim p_{data}$ and then we know $Z = g(X)$. Our goal is to find evaluations of $p_X(\cdot)$. The naïve pitfall is summarized below. In general, we have
$$p_X(X) = p_Z(g(X))\,|\partial g(X)| \tag{8.4}$$
and this is called the ***change of variables formula for R.V.s***

Pitfall in Evaluating Density in Generative Goals The pitfall is that we might naïvely think
$$P_X(X) \overset{?}{=} p_Z(g(X)) \tag{8.5}$$
Suppose that $Z \sim \text{Unif}[0,1]$, and $X = f(Z) = 2Z$. Then, $X \sim \text{Unif}[0,2]$. If we follow the naïve idea above, we will get
$$p_X(X=2) \overset{?}{=} p_Z(2Z=2) = p_Z(Z=1) = 1 \tag{8.6}$$
but this must not be the case! Since $X \sim \text{Unif}[0,2]$ and thus $p_X(X=2) = 1/2 \neq 1$. ***Why did this happen?*** This is because the mapping also alters the "lateral volume" of the random variable, expanding the original 1 into 2. (It might also help to think that the mapping is communicating two spaces with different densities, Unif[0, 1] is more dense than Unif[0, 2].) To account for this, we must take into consider the change in volume, which is equal to
$$|\partial f(X)| = \det \left.\frac{\mathrm{d}f}{\mathrm{d}X}\right|_{X=2} = \frac{1}{2} \tag{8.7}$$
where the general form of the formula is presented in Equation 8.4.

8.4 Learning Flows

8.4.1 Learning Directly and Problems Associated

To learn the flows, we use maximum likelihood, i.e.
$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_X\left[-\log p_X(X|\boldsymbol{\theta})\right] &(8.8)\\ &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_X\left[-\log p_Z(g(X)) - \underbrace{\log\det\left[\partial g(X)\right]}_{(\dagger)}\right] &(8.9)\end{aligned}$$
where we expect the (†) Jacobian determinant term to be tractable. However, in the general case, the determinant of a arbitrary $M_{D\times D}(\mathbb{R})$ matrix, and hence the Jacobian determinant of interest is order of $\mathcal{O}(D^3)$, which would be too slow and intractable to be inside a training loop. In Section 8.2, we saw that if we permit f to be arbitrarily expressive, then p_Z can be transformed into any p_X. The challenge lies within the fact that constructing arbitrary bijections is difficult, and this is where Normalizing Flows is to the rescue! We can think of it this way: Normalizing Flows $\equiv$ Transformations that have Jacobian determinant tractable such that

- it is easy to compute $|\partial f(Z)|$, and
- it is tractable to invert $|\partial f(g(X))|^{-1} = |\partial g(x)|$

8.4.2 Constructing Tractable Flows - General Idea

To do so, we will use the property that the composition of flows is itself a flow. To be more precise, suppose that

$$f = f_N \circ f_{N-1} \circ \cdots \circ f_1 \circ f_0 \tag{8.10}$$

has inverse

$$g = g_0 \circ g_1 \circ \cdots \circ g_{N-1} \circ g_N \tag{8.11}$$

with Jacobian Determinant

$$|\partial g(X)| = \prod_{i=0}^{n} |\partial g_i(X)| \tag{8.12}$$

As a consequence,

$$\log p_X(X) = \log p_Z(g(X)) + \sum_{i=1}^{N} \log |\partial g_i(X_i)| \tag{8.13}$$

where $X_i = g_{i+1} \circ \cdots \circ g_N(X)$ and in particular $X_N = X$.

8.4.3 Element-wise Flow Construction

The first method that we propose is element-wise operation of f. Suppose that we have

$$X = \begin{bmatrix} x_1 & x_2 & \cdots x_D \end{bmatrix}^\top \in \mathbb{R}^D \tag{8.14}$$

then we construct the mapping as

$$f(X) = \begin{bmatrix} f_1(x_1) & f_2(x_2) & \cdots & f_D(x_D) \end{bmatrix}^\top \tag{8.15}$$

Using such construction, our Jacobian will be diagonal, i.e.

$$\partial f = \operatorname{diag} \begin{bmatrix} \partial f_1(x_1) & \partial f_2(x_2) & \cdots & \partial f_D(x_D) \end{bmatrix}^\top \tag{8.16}$$

and thus the determinant computation is simply

$$\det \partial f = \prod_i \partial f_i(x_i) \tag{8.17}$$

which is easy to compute. This method comes with a cost that it can not model any dependence between dimensions of the data, which might be too restrictive considering that we want to model complex data such as image.

8.4.4 Affine/Linear Flow Construction

This method uses f of the format

$$f(X) = \mathbf{A}X + \mathbf{b} \tag{8.18}$$

and the normalizing direction would be

$$g(Z) = \mathbf{A}^{-1}(Z - \mathbf{b}) \tag{8.19}$$

The Jacobian in this case could be expressed in terms of $\mathbf{A}$, to be more explicit, we have

$$\partial f = \mathbf{A} \qquad \text{and} \qquad \partial g = \mathbf{A}^{-1} \tag{8.20}$$

and we notice that the determinant computation is still expensive unless we restrict A:

- ***Diagonal* $\mathbf{A}$*:*** then this is just the element-wise construction we saw in Section 8.4.3,
- ***Triangular* $\mathbf{A}$*:*** then the determinant computation cost is $\mathcal{O}(D)$ and is much cheaper than the general case. [8.3]

9 Lecture 11 - Generative Adversarial Networks (GANs)

[8.3] In the case that the base distribution is a gaussian of D dimensions, then this construction just describes a multi-variate gaussian distribution with covariance matrix $\mathbf{A}$.