STA414
Lecture Notes

Yuchen Wang

April 22, 2020

Contents

1 Introduction **3**

2 Introduction to Probabilistic Models **3**
2.1 Overview of probabilistic models 3
2.2 Sufficient statistics 4

3 Directed Graphical Models **6**
3.1 Decision Theory 6
3.2 Joint Distributions 6
3.2.1 Number of parameters in a joint distribution 7
3.2.2 Conditional Independence 7
3.3 Directed acyclic graphical models (DAGM) 7
3.3.1 Conditional Independence in DAGM 8
3.3.2 Example of a DAGM: Markov Chain 11
3.3.3 Plates 11
3.3.4 Unobserved Variables 12

4 Exact Inference **13**
4.1 Inference as Conditional Distribution 13
4.2 Variable elimination 13
4.2.1 Sum-Product Inference 14

5 Message passing, Hidden Markov Models, and Sampling **16**
5.1 Message Passing & Belief Propagation 17
5.2 Hidden Markov models 17
5.2.1 Sequential data 17
5.2.2 Hidden Markov Models 18
5.2.3 Inference in HMMs 19
5.3 Sampling 20
5.3.1 Ancestral Sampling 21

6 Stochastic Variational Inference **21**
6.1 Motivation . . . 21
6.2 the TrueSkill latent variable model . . . 21
6.3 Posterior Inference in Latent Variable Models . . . 22
6.3.1 Approximating the Posterior Inference with Variational Methods . . . 22
6.3.2 Kullback-Leibler Divergence . . . 23
6.4 Tutorial - Alternative forms of the ELBO . . . 25
6.5 Tutorial - Mean Field Variational Inference . . . 26

7 Sampling and Monte Carlo Methods **26**
7.1 Sampling . . . 26
7.2 Importance Sampling . . . 27
7.3 Rejection Sampling . . . 29
7.4 Markov Chain Monte Carlo (MCMC) . . . 30
7.4.1 Metropolis-Hastings method . . . 30

8 Amortized Inference and Variational Auto-Encoders **31**
8.1 the algorithm for amortized inference . . . 32
8.2 Optimizing model parameters . . . 32
8.3 Variational Autoencoder (VAE) . . . 32

9 Normalizing Flows **34**
9.1 Basics . . . 35
9.1.1 More formal construction . . . 36
9.2 Applications . . . 36
9.2.1 Density estimation and sampling . . . 36
9.3 Methods . . . 36
9.3.1 Elementwise bijections . . . 37
9.3.2 Linear Flows . . . 37
9.3.3 Planar and Radial Flows . . . 38
9.3.4 Coupling Flows . . . 39
9.3.5 Autoregressive Flows . . . 40
9.3.6 Residual Flows . . . 41
9.3.7 Infinitesimal(Continuous) Flows . . . 41

1 Introduction

2 Introduction to Probabilistic Models

2.1 Overview of probabilistic models

In general, we have random variables $X = (X_1, \ldots, X_N)$ that are either *observed* or *unobserved*. Need 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 $p_\theta(X_1, \ldots, X_N)$. We assume there is a true joint p_*, which we are trying to learn with a model p_θ.
Assume we have the joint probability $p(X, C, Y)$

Regression

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

Classification / Clustering

$$p(C|X) = \frac{p(X,C)}{\sum_C p(X,C)}$$

Now that we have a distribution over class labels, we have a choice for how to assign the class label:

1. $c^* = \arg\max_c p(C = c|X)$
2. Sample the class assignment from our distribution, $c^* \sim p(C|X)$
3. Output the class assignment (however we chose it) along with its density under our distribution $(c^*, p(C = c^*|X))$. Can inform us of the model's uncertainty or confidence of the prediction.

Latent/hidden Variables Variables which are never observed in the dataset.

Operations on Probabilistic Models

- **Generate Data** For this we will need to 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 all variables taking on whose specific values.
- **Inference**: Compute 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.

Goals of joint distributions

1. Facilitate **efficient computation** of marginal and conditional distributions
2. Have **compact representation** so the size of the parameterization scales well for joint distributions over many variables.

Joint Dimensionality Suppose n is the number of variables and k is the number of states of each variable. Then dimensionality of our parameters is k^n.

2.2 Sufficient statistics

Theorem 2.1 (Fisher-Neyman factorization theorem)**.** If $f_\theta(x)$ is a pdf, then T is sufficient for θ if and only if nonnegative functions g and h can be found such that

$$f_\theta(x) = h(x)g_\theta(T(x))$$

i.e. the density f can be factored into a product such that one factor h does not depend on θ and the other factor, which does depend on θ, depends on x only through $T(x)$.

Definition 2.1 (Statistic and Sufficient statistic)**.** A statistic is a (possibly vector valued) deterministic function of a (set of) random variable(s). A 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. Formally, we say that $T(X)$ is a sufficient statistic for X if

$$T(x^{(1)}) = T(x^{(2)}) \implies L(\theta; x^{(1)}) = L(\theta; x^{(2)}) \quad \forall\theta$$

where L is the likelihood function.
Alternatively,

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

Equivalently (by the Neyman factorization theorem) we can write

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

Example 2.1 (Bernoulli Trials)**.** We observe N iid coin flips.
Model: $p(H) = \theta, P(T) = 1 - \theta$
Likelihood: $l(\theta; D) = \log\theta \sum_n x^{(n)} + \log(1-\theta)\sum_n (1 - x^{(n)})$

$$\begin{aligned} l(\theta; D) &= \log\theta \sum_n x^{(n)} + \log(1-\theta)\sum_n (1 - x^{(n)}) \\ &= \log\theta N_H + \log(1-\theta)N_T \end{aligned}$$

Notice that our likelihood depends on $N_H = \sum_n x^{(n)}$ (and N_T).
$\implies$ If we know this summary statistic $T(x) = \sum_n x^{(n)}$, then we know everything that is useful from our sample to do inference.

$$l(\theta; D) = T(X)\log\theta + (N - T(X))\log(1-\theta)$$

Then we take the derivative and set it to 0 to find the maximum

$$\Rightarrow \frac{\partial \ell}{\partial \theta} = \frac{T(X)}{\theta} - \frac{N - T(X)}{1 - \theta}$$
$$\Rightarrow \hat{\theta} = \frac{T(X)}{N}$$

This is our maximum likelihood estimation of the parameters $\theta, \theta^{\star}_{MLE}$.
sufficient statistics: counts

Example 2.2 (Multinomial)**.** We observe M iid die rolls (K-sided).
Model: $p(k) = \theta_k, \sum_k \theta_k = 1$
Likelihood: $l(\theta; D) = \sum_k N_k \log \theta_k$
Take derivatives and set to zero (enforcing $\sum \theta_k = 1$):

$$\frac{\partial \ell}{\partial \theta_k} = \frac{N_k}{\theta_k} - M$$
$$\Rightarrow \theta^*_k = \frac{N_k}{M}$$

sufficient statistics: counts

Example 2.3 (exponential family of distributions)**.** The result of the previous example distributions show that the MLE are just normalized counts. However, the simplicity of the sufficient statistics and MLE are due to those being members of the exponential family. In general, exponential family members have simple sufficient statistics and MLE for the natural statistics η

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

where:

- η are the parameters
- $T(x)$ are the sufficient statistics
- $h(x)$ is the base measure
- $A(\eta)$ is the normalizing constant

with log-likelihood

$$\begin{aligned} l(\eta; D) &= \log p(D; \eta) \\ &= (\sum_n \log h(x_n)) - NA(\eta) + (\eta^T \sum_n T(x_n)) \end{aligned}$$

Finding the derivative and setting to zero, we get

$$\eta_{MLE} = \frac{1}{N} \sum_n T(x_n)$$

which is the normalized counts of the data.

Example 2.4. univariate normal We have N i.i.d. samples $\{x_i\}_1^N$
Model: $p(x|\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{1}{2\sigma^2}(x-\mu)^2$
Gaussian distribution is a member of the exponential family, so we can put it into a natural form

$$p(x|\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{1}{2\sigma^2}\mu^2\} \exp\{\begin{bmatrix}\frac{\mu}{\sigma^2} & \frac{-1}{2\sigma^2}\end{bmatrix}\begin{bmatrix}x\\x^2\end{bmatrix}\}$$

From here, it is clear that the natural parameters and the sufficient statistics are

- $\eta = \begin{bmatrix}\frac{\mu}{\sigma^2}\\ \frac{-1}{2\sigma^2}\end{bmatrix}$
- $T(x) = \begin{bmatrix}x\\x^2\end{bmatrix}$

re-writing in terms of η

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

noting that

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

3 Directed Graphical Models

3.1 Decision Theory

We care about probabilities because they help us make decisions.
Denote action by a, state by s, value function by $V(s)$, utility function by $u(a)$, then

$$a^* = \underset{a}{argmax} \underbrace{E_{p(s|a,knowledge)}[V(s)]}_{u(a)}$$

3.2 Joint Distributions

Theorem 3.1 (chain rule of probability)**.** The joint distribution of N random variables can be computed by the chain rule

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

This is true for any joint distribution over any random variables (assuming full dependence between variables). More formally, in probability the chain rule for two random variables is

$$p(x,y) = p(x|y)p(y)$$

and for N random variables

$$p(x_1, x_2, \ldots, x_N) = \prod_{j=1}^{N} p(x_j|x_1, x_2, \ldots, x_{j-1})$$

for any ordering of the variables.

3.2.1 Number of parameters in a joint distribution

$$p(x_1, \ldots, x_A | y_1, \ldots, y_B)$$

parameters = {(# possible states of $x_{1:A}$) - 1} $\times$ (# possible states of $y_{1:B}$)
binary: $(2^A - 1) \times 2^B$.

3.2.2 Conditional Independence

Definition 3.1 (conditionally independence)**.** Two random variables A, B are conditionally independent given a third variable C, denoted

$$X_A \perp X_B | X_C$$

if

$$\begin{aligned} &\iff p(X_A, X_B | X_C) = p(X_A | X_C) p(X_B | X_C) \\ &\iff p(X_A | X_B, X_C) = p(X_A | X_C) \\ &\iff p(X_B | X_A, X_C) = p(X_B | X_C) \end{aligned}$$

for all X_C.

3.3 Directed acyclic graphical models (DAGM)

Graphical models Probabilistic graphical models are a concise way to specify and reason about conditional independencies, without worrying about the detailed form of the distribution. There are three flavours:

- Undirected
- Factor graphs
- Directed

Definition 3.2 (directed acyclic graphical model)**.** A directed acyclic graphical model implies a restricted factorization of the joint distribution. In a DAG, variables are represented by nodes, and dependence are represented by edges.
Recall that for N random variables,

$$p(x_1, x_2, \ldots, x_N) = \prod_{j=1}^{N} p(x_j | x_1, x_2, \ldots, x_{j-1})$$

for any ordering of the variables.
In the context of DAG, we can write

$$p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i | parents(x_i))$$

where $parents(x_i)$ 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 nodes(s), which could be empty.

Remark 3.1. We are conditioning on parent nodes as opposed to every node. Therefore, the model that represents this distribution is exponential in the fan-in of each node (the number of nodes in the parent set), instead of in N.

Grouping variables We can always group variables together into one bigger variable:

$$p(x_i, x_{\pi_i}) = p(x_{\pi_i})p(x_i|x_{\pi_i})$$

3.3.1 Conditional Independence in DAGM

The simplest conditional independence relationship encoded in a Bayesian network can be stated as follows: a node is independent of its ancestors given its parents:

$$x_i \perp x_{\tilde{\pi}_i}|x_{\pi_i}$$

In general, missing edges imply conditional independence.

Definition 3.3 (D-Separation)**.** D-separation, or directed-separation is a notion of connectedness in DAGMs 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 that

$$x_A \perp x_B|x_C$$

if every variable in A is d-separated from every variable in B conditioned on all the variables in C.
To check if an independence is true, we can cycle through each node in A, do a depth-first search to read every node in B, and examine the path between them. If all of the paths are d-separated(i.e. conditionally independent), then

$$x_A \perp x_B|x_C$$

Example 3.1. Chain

Question: When we condition on y, are x and z independent?
Answer:
From the graph, we get

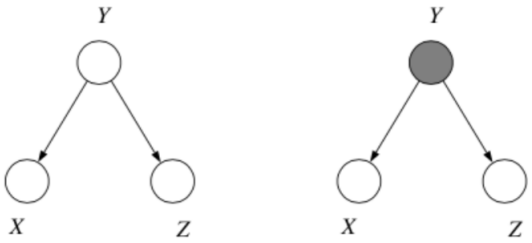
$$P(x, y, z) = P(x)P(y|x)P(z|y)$$

Then

$$\begin{aligned} P(z|x, y) &= \frac{P(x, y, z)}{P(x, y)} \\ &= \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)} \\ &= P(z|y) \end{aligned}$$

which implies $x \perp z|y$.

Example 3.2. Common Cause



Where we think of y as the "common cause" of the two independent effects x and z.
Question: When we condition on y, are x and z independent?
Answer: From the graph, we get

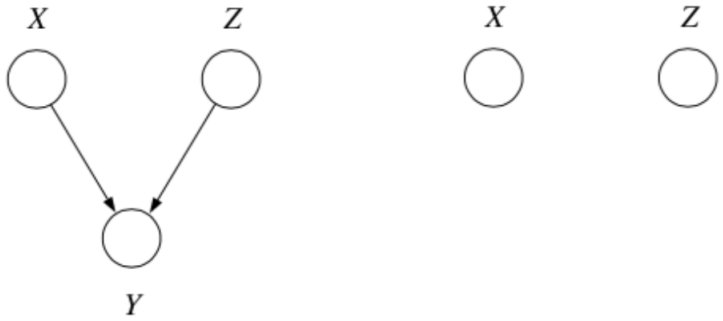
$$P(x, y, z) = P(y)P(x|y)P(z|y)$$

which implies

$$\begin{aligned} P(x, z|y) &= \frac{P(x, y, z)}{P(y)} \\ &= \frac{P(y)P(x|y)P(z|y)}{P(y)} \\ &= P(x|y)P(z|y) \end{aligned}$$

which implies $x \perp z|y$.

Example 3.3. Explaining Away



Question: $x \perp z$?
Answer:

$$\begin{aligned} P(x,z) &= \sum_y P(x,y,z) \\ &= \sum_y P(x)P(z)P(y|x,z) \\ &= P(x)P(z)\sum_y P(y|x,z) \\ &= P(x)P(z) \end{aligned}$$

which implies that $x \perp z$.
Question: $x \perp z|y$?
Answer:

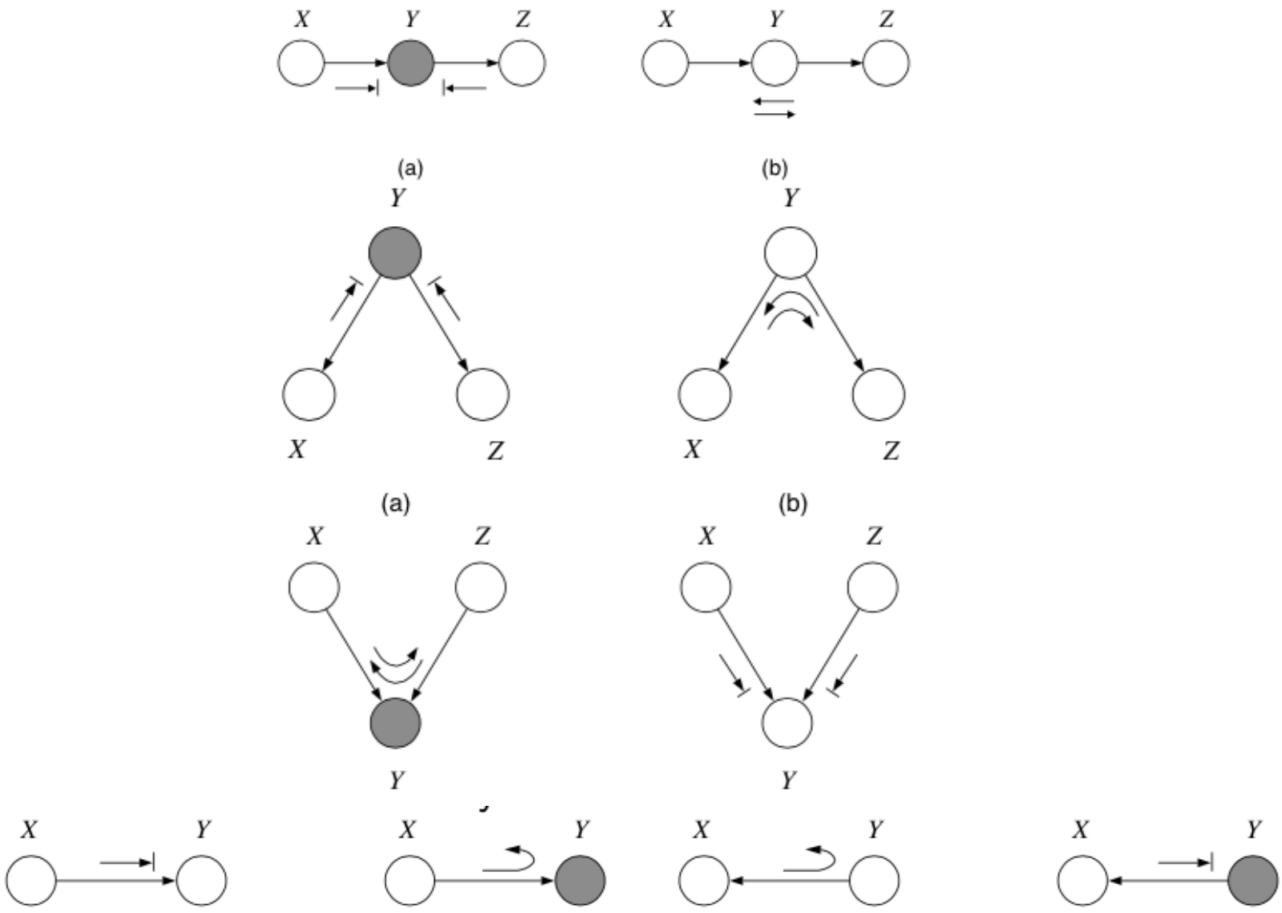
$$\begin{aligned} p(z|x,y) &= \frac{p(x)p(z)p(y|x,z)}{p(x)p(y|x)} \\ &= \frac{p(z)p(y|x,z)}{p(y|x)} \\ &= p(z|y) \end{aligned}$$

which implies $x \not\perp |y$.
In fact, x and z are marginally independent, but given y they are conditionally dependent. This important effect is called explaining away.

Theorem 3.2 (Bayes-Balls Algorithm)**.** In general, the algorithm works as follows:

1. Shades all nodes x_C
2. Place "balls" at each node in x_A (or x_B)
3. Let the "balls" "bounce" around according to some rules
4. If any of the balls reach any of the nodes in x_B from x_A (or x_A from x_B), then $x_A \not\perp x_B|x_C$; otherwise $x_A \perp x_B|x_C$.

The rules are as follows:



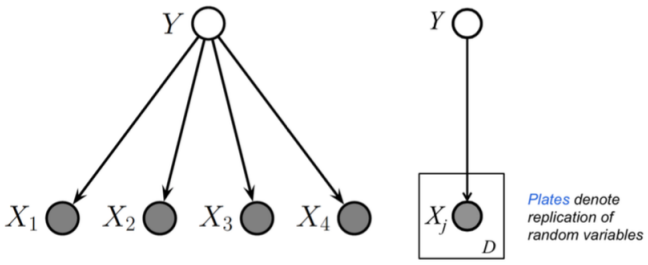
where arrows indicate paths the balls can travel, and arrows with bars indicate paths the balls cannot travel.

3.3.2 Example of a DAGM: Markov Chain

Markov chains are a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.
i.e. Conditional on the present state of the system, its future and past states are independent.

3.3.3 Plates

Because Bayesian methods treat parameters as random variables, we would like to include them in the graphical model. One way to do this is to repeat all the iid observations explicitly and show the parameter only once. A better way is to use plates, in which repeated quantities that are iid are put in a box.



The rules of plates:

1. 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. Duplicate every arrow going into the plate and every arrow leaving the plate by connecting the arrows to each copy of the structure.
2. Plates can be nested, in which case their arrows get duplicated also.
3. Plates can also 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.

3.3.4 Unobserved Variables

Certain variables in our models may be unobserved, either some of the time or always, at training time or at test time.

Partially unobserved variables If variables are **occasionally unobserved** then 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: $l(\theta; D) = \sum_{complete} \log p(x^c, y^c|\theta) + \sum_{missing} \log p(x^m|\theta)$

$$\begin{aligned} l(\theta; D) &= \sum_{complete} \log p(x^c, y^c|\theta) + \sum_{missing} \log p(x^m|\theta) \\ &= \sum_{complete} \log p(x^c, y^c|\theta) + \sum_{missing} \log \sum_y p(x^m, y|\theta) \end{aligned}$$

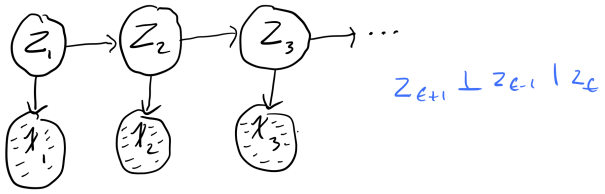
Latent variables What to do when a variable z is always unobserved?
Depends on where it appears in our model. If we never condition on it when computing the probability of the variables we do observe, then we can just forget about it and integrate it out.

Mixture models What if the class is unobserved? Then we sum it out

$$p(x; \theta) = \sum_{k=1}^{K} p(z = k; \theta_z) p(x|z = k; \theta_k)$$

We can use the Bayes' rule to compute the posterior probability of the mixture component given some data:

Hidden Markov Models (HMMs) Hidden Markov Model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unobserved (i.e. hidden) states. It is a very popular type of latent variable model.



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 joint probability represented by the graph factorizes according to

$$p(X_{1:T}, Z_{1:T}) = p(Z_{1:T})p(X_{1:T}|Z_{1:T}) = p(Z_1)\prod_{t=2}^{T} p(Z_t|Z_{t-1})\prod_{t=1}^{T} p(X_T|Z_t)$$

4 Exact Inference

4.1 Inference as Conditional Distribution

Notation 4.1.

$$\begin{aligned} X_E &= \text{ the observed evidence} \\ X_F &= \text{ the unobserved variable we want to infer} \\ X_R = X - \{X_F, X_E\} &= \text{ Remaining variables, extraneous to query} \end{aligned}$$

where X_R is the set of random variables in our model that are neither part of the query nor the evidence.

Definition 4.1 (exact inference)**.** The exact inference task is defined as: Given a fully parameterized DAG model over variables χ, $X_F \subseteq \chi, X_E \subseteq \chi$ s.t. $X_E \cap X_F = \emptyset$ and $\mathbf{e} \in val(X_E)$:

$$\text{compute } P(X_F|X_E = \mathbf{e})$$

X_E can be empty in which case we're after $P(X_F)$.

4.2 Variable elimination

Variable elimination is a simple and general exact inference algorithm in any probabilistic graphical model.

Simple Example: Chain

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

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

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

This 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:

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

Sum naively: $\mathcal{O}(k^n)$

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

Elimination ordering: $\mathcal{O}(nk^2)$

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

Remark 4.1. Computing the joint is NP-hard. We can catch up computations that are otherwise computed exponentially many times, but this depends on having a good variable elimination ordering.

4.2.1 Sum-Product Inference

Sum-product inference algorithm can be used to compute $P(Y)$ for directed and undirected models: $\forall Y$,

$$\tau(Y) = \sum_z \prod_{\phi \in \Phi} \phi(scope[\phi] \cap Z, scope[\phi] \cap Y)$$

where Φ is a set of potentials or factors.

We want to marginalize out variables in Z since they are extraneous

e.g. for $\phi(A, B, C)$, $\text{scope}[\phi] = \{A, B, C\}$

Directed models Φ is given by the conditional probability distributions for all variables

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

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

For DAG, we have $\phi = p$

Undirected models Φ is given by the set of unormalized potentials. Therefore, we must normalize the resulting $\tau(Y)$ by $\sum_Y \tau(y)$.

Example 4.1 (Directed Graph)**.**

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

We can write the conditional distributions as factors

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

If we are interested in inferring the probability of getting a job, $p(J)$, we can perform exact inference on the joint distribution by marginalizing according to a specific variable elimination

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

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

Fact 4.1 (Complexity of variable elimination ordering)**.** The complexity of the VE algorithm is

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

- m is the number of initial factors $= |\Phi|$
- 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 $\sum_i$
- $N_{max} = argmax_i N_i$ is the number of random variables inside the largest sum.

5 Message passing, Hidden Markov Models, and Sampling

Inference in Trees Tree is a general family of graphs for which the optimal elimination ordering is trivial to find, and which has linear cost in the number of nodes.

5.1 Message Passing & Belief Propagation

What if we want to compute the marginal of every variable in a graph: $p(x_i)\,\forall x_i \in X$? Run variable elimination separately for each variable x_i is computationally expensive.

Joint distribution for undirected graph models For an undirected graph $G = (V, E)$,

$$P(X_{1:n}) = \frac{1}{Z}\prod_i \phi(x_i) \prod_{(j,k)\in E} \phi_{j,k}(x_j, x_k)$$

Message-passing Belief propagation is based on message-passing of "message" between neighboring vertices of the graph. The message sent from variable j to $i \in N(j)$ is

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

Theorem 5.1 (Belief propagation algorithm)**.** As follows:

1. Choose root r arbitrarily
2. Pass messages from leaves to r
3. Pass messages from r to leaves
4. Compute $p(x_i) \propto \phi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i),\ \forall i$

5.2 Hidden Markov models

5.2.1 Sequential data

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

Recall the general joint factorization via the chain rule

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

First order Markov chain

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

This assumption greatly simplifies the factors in the joint distribution

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

Definition 5.1 (stationary/time-homogenous Markov chain)**.** As follows

- **Stationary Markov chain:** the distribution generating the data does not change through time:
$$p(x_t|x_{t-1}) = p(x_{t+k}|x_{t-1+k}) \quad \forall t, k$$
- **Non-stationary Markov chain:** the distribution generating the data is a function of time.

Higher-order Markov chains second order:

$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-1}, x_{t-2})$$

m-order:

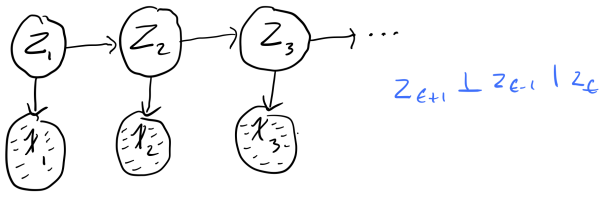
$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-m:t-1})$$

Parameterization How does the order of temporal dependence affect the number of parameters in our model?
Assume x is a discrete random variable with k states.

1. $x_t : k - 1$, as the last state is implicit.
2. first-order chain: $k(k-1)$, as we need k numbers of parameters for each parameter of x_t
3. m-order chain: $k^m(k-1)$, as we need k^m number of parameters for each parameter of x_t

5.2.2 Hidden Markov Models

Hidden Markov Model (HMM) hide the temporal dependence by keeping it in the unobserved state. For each observation x_t, we associate a corresponding unobserved hidden/latent variable z_t



The joint probability represented by the graph factorizes according to

$$p(X_{1:T}, Z_{1:T}) = p(Z_{1:T})p(X_{1:T}|Z_{1:T}) = p(Z_1)\prod_{t=2}^{T} p(Z_t|Z_{t-1})\prod_{t=1}^{T} p(X_T|Z_t)$$

Unlike simple Markov chains, the observations are not limited by a Markov assumption of any order, i.e. x_t isn't necessarily independent of any other observation, no matter how many other observations we make.

Definition 5.2 (Parameterization of a hidden Markov model)**.** Assuming we have a homogeneous model, we only have to learn three distributions

1. **Initial distribution:** $\pi(i) = p(z_1 = i)$. The probability of the first hidden variable being in state i (often denoted by π.)
2. **Transition distribution:** $T(i, j) = p(z_{t+1} = j|z_t = i)$, $i \in \{1, \ldots, k\}$. The probability of moving from hidden state i to hidden state j.
3. **Emission probability:** $\epsilon_i(x_t) = p(x_t|z_t = i)$. The probability of an observed random variable x_t given the state of the hidden variable that "emitted" it.

5.2.3 Inference in HMMs

HMMs are just tree-structured DAGs, meaning that inference in them is linear in the number of time steps. We can do exact inference in them.

Main tasks we perform with HMMs

1. Compute the probability of a latent sequence given an observation sequence. (e.g. compute $p(z_i|x_{1:t})\forall i$ with the Forward-Backward algorithm)
2. Compute the marginal likelihood $p(x_1, x_2, \ldots, x_T)$ in order to fit parameters
3. Infer the most likely sequence of hidden states (i.e. compute $Z^* = \underset{z_{1:T}}{argmax}\, p(z_{1:T}|x_{1:T})$ using the Viterbi algorithm)

Assuming that we know the initial $p(z_1)$, transition $p(z_t|z_{t-1})$, and emission $p(x_t|z_t)\forall t \in [1, T]$ This 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, $p(z_n|x_{1:t}) \quad 1 < n < t$

Prediction Let's first take a look at the example where $k = 2$, then

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

Theorem 5.2 (Forward-backward algorithm)**.** The Forward-backward algorithm is used to efficiently estimate the latent sequence given an observation sequence under a HMM. That is, we want to compute

$$p(z_t|x_{1:T}) \quad \forall t \in [1, T]$$

It is computed in two parts, and then multiplied together:

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

Note that

$$\begin{aligned} p(z_t|x_{1:T}) &\propto p(z_t, x_{1:T}) \\ &= p(z_t, x_{1:t}) p(x_{t+1:T}|z_t, x_{1:t}) \\ &= \underbrace{p(z_t, x_{1:t})}_{forward\, recursion} \underbrace{p(x_{t+1:T}|z_t)}_{backward\, recursion} \end{aligned}$$

Forward Filtering

$$\begin{aligned} p(z_t, x_{1:t}) &= \sum_{z_{t-1}=1}^{k} p(z_{t-1}, z_t, x_{1:t}) \\ &= \sum_{z_{t-1}=1}^{k} p(x_t|z_{t-1}, z_t, x_{1:t-1}) p(z_t|z_{t-1}, x_{1:t-1}) p(z_{t-1}, x_{1:t-1}) \end{aligned}$$

Define $\alpha_t(z_t) := p(z_t, x_{1:t})$, then

$$\alpha_t(z_t) = p(x_t|z_t) \sum_{z_{t-1}=1}^{k} p(z_t|z_{t-1}) \alpha_{t-1}(z_{t-1})$$

If we recurse all the way down to $\alpha_1(z_1)$, we get

$$\alpha_1(z_1) = p(z_1, x_1) = p(z_1) p(x_1|z_1)$$

Backward Filtering

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

Define $\beta_t(z_t) := p(x_{1+t:T}|z_t)$, then

$$\beta_t(z_t) = \sum_{z_{t+1}}^{k} \beta_{t+1}(z_{t+1}) p(x_{t+1}|z_{t+1}) p(z_{t+1}|z_t)$$

If we recurse all the way down to $\beta_1(z_1)$, we get

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

5.3 Sampling

A sample form a distribution $p(x)$ is a single realization x whose probability distribution is $p(x)$. This contrasts with the alternative usage in statistics, where sample refers to a collection of realization x.

The problems to be solved The aims of Monte Carlo methods are to solve one or both of the following problems

1. To generate samples $\{x^{(r)}\}_{r=1}^{R}$ from a given probability distribution $p(x)$.
2. To estimate expectations of functions, $f(x)$, under distribution $p(x)$:

$$E = E_{x \sim p(x)}[f(x)] = \int f(x) p(x)\, dx$$

5.3.1 Ancestral Sampling

"Sampling in a topological order"
i.e. at each step, sample from any conditional distribution that you haven't visited yet, whose parents have all been sampled. This procedure will always start with the nodes that have no parents.

Example 5.1. In a chain or HMM, you would always start with z_1 and move to the right. In a tree, you would always start from the root.

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 a variable conditional on a node with no parents, that is also easy - you can simple do ancestral sampling starting from the nodes you have.
However, to sample from a DAG conditional on leaf nodes is hard. Finding ways to do this approximately is what a lot of the rest of the course will be about.

6 Stochastic Variational Inference

6.1 Motivation

In modern machine learning is most often used to infer the conditional distribution over the latent variables given the observations (and parameters). (the posterior distribution) This can be written as

$$p(z|x,\alpha) = \frac{p(z,x|\alpha)}{\int_z p(z,x|\alpha)}$$

Since the integral cannot be easily computed analytically, we will use variational inference. The main idea behind this is to choose a family of distributions over the latent variables $z_{1:m}$ with its own set of variational parameters v, i.e., $q(z_{1:m}|v)$. Then, we find the setting of the parameters that makes our approximation closest to the posterior distribution (this is where optimization algorithms come in). Then we can use q with the fitted parameters in place of the posterior (e.g. to form predictions about future data, or to investigate the posterior distribution over the hidden variables, find modes, etc).

6.2 the TrueSkill latent variable model

A player ranking system for competitive games.
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. We initially don't know anything about anyone's skill, for simplicity we start with an independent Gaussian prior.

Remark 6.1. We never get to observe the player's skill directly, which make this a *latent variable model*. Instead, we observe the outcome of a series of matches between different players.

the model Each player has a fixed level of skill, denoted z_i.
For each game, the probability that player i beats player j is given by

$$\sigma(z_i - z_j)$$

where sigma is the logistic function $\sigma(y) = \frac{1}{1+\exp(-y)}$. So we have

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

Remark 6.2. *The exact form of the prior and likelihood aren't particularly important, as long as the higher the skill level, the higher the chance of winning each game.*
We can write the entire joint likelihood of set of players and games as:

$$p(z_1, z_2, \ldots, z_N, g_1, g_2, \ldots, g_T) = \left[\prod_{i=1}^{N} p(z_i)\right] \left[\prod_{player_i, player_j \in g_k} p(i \text{ beats } j | z_i, z_j)\right]$$

Computing the posterior over two player's skills requires integrating over all the other players' skills

$$p(z_1, z_2 | g_1, g_2, \ldots, g_T) = p(z_1, z_2 | x) = \int \ldots \int p(z_1, z_2, \ldots, z_N | x)\, dz_3 \ldots dz_N$$

where x is a $N \times 2$ matrix and stores (i, j) pairs that i beats j.

6.3 Posterior Inference in Latent Variable Models

Consider the probabilistic model $p(x, z)$ where

- $x_{1:T}$ are the observations
- $z_{1:N}$ are the unobserved latent variables

The conditional distribution of the unobserved variables given the observed variables (the posterior inference) is

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

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

6.3.1 Approximating the Posterior Inference with Variational Methods

Approximating the Posterior Inference with Variational Methods works as follows:

1. Introduce a variational family $q_\phi(z)$ with parameters ϕ.
2. Encode some notion of "distance" between $p(z|x)$ and $q_\phi(z)$.
3. Minimize this distance.

Remark 6.3. This turns Bayesian Inference into an optimization problem. If enough parts 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.

6.3.2 Kullback-Leibler Divergence

We will measure the distance between q_ϕ and p using Kullback-Leibler divergence:

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

Property 6.1. Properties of the KL Divergence:

1. $D_{KL}(q_\phi||p) \geq 0$
2. $D_{KL}(q_\phi||p) = 0 \iff q_\phi = p$
3. $D_{KL}(q_\phi||p) \neq D_{KL}(p||q_\phi)$

Remark 6.4. The significance of the last property is that D_{KL} is not a true distance measure.

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

$$q_\phi \approx p \implies D_{KL}(q_\phi||p) = 0$$

In other words, we want to find ϕ^* s.t.

$$\phi^* = \arg\min_\phi D_{KL}[q_\phi||p]$$

Remark 6.5. The computation of $D_{KL}(q_\phi||p)$ is intractable, because it contains the term $p(z|x)$.

Evidence Lower Bound (ELBO) To do variational inference, we want to minimize the KL divergence between our approximation q and our posterior p. However, we cannot actually minimize this quantity directly, but a function that is equal to it up to a constant.

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

Claim: Maximizing the ELBO $\implies$ minimizing $D_{KL}(q_\phi||p)$.

Proof. Rearranging, we get

$$\begin{aligned} D_{KL}(q_\phi(z|x)||p(z|x)) &= -\mathcal{L}(\phi;x) + \log p(x) \\ \implies \mathcal{L}(\phi;x) + D_{KL}(q_\phi(z|x)||p(z|x)) &= \log p(x) \end{aligned}$$

Since $D_{KL}(q_\phi(z|x)||p(z|x)) \geq 0$, we have

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

Therefore, maximizing the ELBO $\implies$ minimizing $D_{KL}(q_\phi||p)$. ■

why this proof?

Optimizing the ELBO We have

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

If we want to optimize this with gradient methods, we will need to compute $\nabla_\phi \mathcal{L}(\phi)$:

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

Pathwise Gradient In general, we cannot switch the derivative and the expectation (an integral), if the distribution we are taking the expectation over ($q_\phi(z|x)$) depends on the parameter (ϕ).
So we need to factor out the randomness from q and put it into a parameterless, fixed source of noise $p(\epsilon)$. To do this, we need to find a function $T(\epsilon, \phi)$ such that:

$$\epsilon \sim p(\epsilon),\ z = T(\epsilon,\phi) \implies z \sim q_\phi(z)$$

Remark 6.6. *Usually we start with $p(\epsilon)$ being uniform or Normal. It's not always easy to find these functions.*

Example 6.1.

$$\epsilon \sim \mathcal{N}(0,1),\ z = \sigma\epsilon + \mu \implies z \sim \mathcal{N}(\mu,\sigma)$$

Using this trick, we can write our expectation:

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

Now we have a differentiable function that we can estimate with simple Monte Carlo.

Remark 6.7. Some notable extensions:

1. We can make q_ϕ arbitrarily expressive, for instance using a mixture of Gaussians, or a normalizing flow.

2. We can fit both ϕ and parameters of $p(z|x)$ at the same time.

6.4 Tutorial - Alternative forms of the ELBO

We have that

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

Form 1 The most general interpretation of the ELBO is given by

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

Form 2 *Recall that Entropy is a measure of expected "surprise": How uncertain are we of the value of a draw from this distribution?*

$$H(X) := - \mathop{\mathbb{E}}_{X \sim p} [\log p(X)] = - \sum_{x \in X} p(x) \log p(x)$$

Rewrite Form 1 using entropy

$$\begin{aligned} L(\phi; x) &= - \mathop{\mathbb{E}}_{z \sim q_\phi} \log \frac{q_\phi(z|x)}{p(x,z)} \\ &= \mathop{\mathbb{E}}_{z \sim q_\phi} \log \frac{p(x,z)}{q_\phi(z|x)} \\ &= \mathop{\mathbb{E}}_{z \sim q_\phi} \log \frac{p(z)p(x|z)}{q_\phi(z|x)} \\ &= \mathop{\mathbb{E}}_{z \sim q_\phi} [\log p(x|z) + \log p(z) - \log q_\phi(z|x)] \\ &= \mathop{\mathbb{E}}_{z \sim q_\phi} [\log p(x|z) + \log p(z)] \, H\left[q_\phi(z|x)\right] \end{aligned}$$

Form 3

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

This frames the ELBO as a tradeoff. The negative of this function is the loss function we use for training VAEs. The first term can be thought of 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. The second term acts as regularization, by enforcing the idea that our parameterization shouldn't move us too far from the true distribution.

6.5 Tutorial - Mean Field Variational Inference

so tired...

7 Sampling and Monte Carlo Methods

7.1 Sampling

Definition 7.1 (sample)**.** A sample from a distribution $p(x)$ is a single realization x whose probability distribution is $p(x)$.

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

1. To generate samples $\{x^{(r)}\}_{r=1}^{R}$ from a given probability distribution $p(x)$.
2. To estimate expectations of functions, $\phi(x)$, under the distribution $p(x)$.

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

Definition 7.2 (simple Monte Carlo)**.** Given $\{x^{(r)}\}_{r=1}^{R} \sim p(x)$, we estimate the expectation $\mathbb{E}_{x\sim p(x)}[f(x)]$ using the average sum, and call it estimator $\hat{\Phi}$:

$$\Phi = \mathbb{E}_{x\sim p(x)}[f(x)] \sim \frac{1}{R}\sum_{r=1}^{R} f(x^{(r)}) = \hat{\Phi}$$

Property 7.1. If the vectors $\{x^{(r)}\}_{r=1}^{R}$ are generated from $p(x)$ then the expectation of $\hat{\Phi}$ is Φ. In fact, $\hat{\Phi}$ is an unbiased estimator of Φ.

Proof.

$$\begin{aligned}
\mathbb{E}[\hat{\Phi}]_{x\sim p(\{x^{(r)}\}_{r=1}^{R})} &= \mathbb{E}[\frac{1}{R}\sum_{r=1}^{R} f(x^{(r)})] \\
&= \frac{1}{R}\sum_{r=1}^{R} \mathbb{E}[f(x^{(r)})] \\
&= \frac{1}{R}\sum_{r=1}^{R} \mathop{\mathbb{E}}_{x\sim p(x)}[f(x)] \\
&= \frac{R}{R}\mathop{\mathbb{E}}_{x\sim p(x)}[f(x)] \\
&= \Phi
\end{aligned}$$

■

Property 7.2. As the number of samples of R increases, the variance of $\hat{\Phi}$ will decrease proportional to $\frac{1}{R}$.

Proof.

$$\begin{aligned} Var[\hat{\Phi}] &= Var\left[\frac{1}{R}\sum_{r=1}^{R} f(x^{(r)})\right] \\ &= \frac{1}{R^2} Var\left[\sum_{r=1}^{R} f(x^{(r)})\right] \\ &= \frac{1}{R^2}\sum_{r=1}^{R} Var\left[f(x^{(r)})\right] && \text{(by i.i.d. assumption)} \\ &= \frac{R}{R^2} Var[f(x)] \\ &= \frac{1}{R} Var[f(x)] \end{aligned}$$

■

Remark 7.1. The accuracy of the Monte Carlo estimate depends only on the variance of ϕ, not on the dimensionality of the x. So regardless of the dimensionality of x, it may be that as few as a dozen independent samples suffice to estimate Φ satisfactorily.

7.2 Importance Sampling

A method for estimating the expectation of a function. It can be viewed as a generalization of the uniform sampling method.

Assume the density from which we wish to draw samples, $p(x)$, can be evaluated within a multiplicative constant. That is, we can evaluate a function $\tilde{p}(x)$ such that

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

We further assume we have a simpler density, $q(x)$ from which it is easy to sample from and easy to evaluate

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

we call such a density $q(x)$ the sampler density.

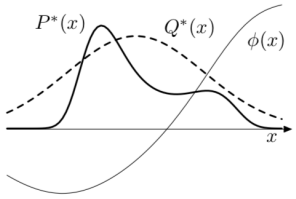


Figure 29.5. Functions involved in importance sampling. We wish to estimate the expectation of $\phi(x)$ under $P(x) \propto P^*(x)$. We can generate samples from the simpler distribution $Q(x) \propto Q^*(x)$. We can evaluate Q^* and P^* at any point.

In importance sampling, we generate R samples from $q(x)$

$$\{x^{(r)}\}_{r=1}^{R} \sim q(x)$$

If these points were samples from $p(x)$, then we could estimate Φ by a simple Monte Carlo estimator:

$$\Phi = \mathbb{E}_{x \sim p(x)}[f(x)] \sim \frac{1}{R} \sum_{r=1}^{R} f(x^{(r)}) = \hat{\Phi}$$

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 weights:

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

Finally, we rewrite our estimator under q

$$\Phi = \int \phi(x)p(x)dx \tag{1}$$
$$= \int \phi(x) \cdot \frac{p(x)}{q(x)} \cdot q(x)dx \tag{2}$$
$$\approx \frac{1}{R}\sum_{r=1}^{R} \phi\left(x^{(r)}\right) \frac{p\left(x^{(r)}\right)}{q\left(x^{(r)}\right)} \tag{3}$$
$$= \frac{Z_q}{Z_p}\frac{1}{R}\sum_{r=1}^{R} \phi(x^{(r)}) \cdot \frac{\tilde{p}(x^{(r)})}{\tilde{q}(x^{(r)})} \tag{4}$$
$$= \frac{Z_q}{Z_p}\frac{1}{R}\sum_{r=1}^{R} \phi(x^{(r)}) \cdot \tilde{w}_r \tag{5}$$
$$= \frac{\frac{1}{R}\sum_{r=1}^{R} \phi(x^{(r)}) \cdot \tilde{w}_r}{\frac{1}{R}\sum_{r=1}^{R} \tilde{w}_r} \tag{6}$$
$$= \frac{1}{R}\sum_{r=1}^{R} \phi(x^{(r)}) \cdot w_r \tag{7}$$
$$= \hat{\Phi}_{iw} \tag{8}$$

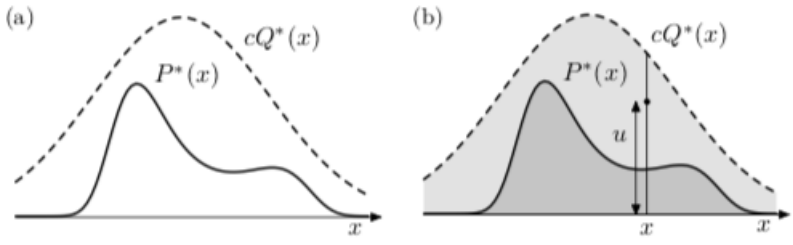
where $\frac{Z_p}{Z_q} = \frac{1}{R}\sum_{r=1}^{R} \tilde{w}_r, w_r = \frac{\tilde{w}_r}{\sum_{r=1}^{R} \tilde{w}_r}$, and $\hat{\Phi}_{iw}$ is our importance weighted estimator.

Remark 7.2. $\hat{\Phi}_{iw}$ is biased.

7.3 Rejection Sampling

We assume that a one-dimensional density $p(x) = \frac{\tilde{p}(x)}{Z}$ is too complicated a function for us to be able to sample from it directly. Also, we have a 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 such that

$$c\tilde{q}(x) > \tilde{p}(x) \quad x$$



The procedure is as follows:

1. Generate two random numbers

- x is generated from the proposal density $q(x)$
- u is generated uniformly from the interval $[0, c\tilde{q}(x)]$

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

Rejection sampling in high dimensions In a high-dimensional problem, it is very likely that the requirement that $c\tilde{q}$ be an upper bound for $\tilde{p}$ will force c to be so huge that acceptances will be very rare indeed. 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.

Remark 7.3. 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 one such q.

7.4 Markov Chain Monte Carlo (MCMC)

Construct a Markov chain over the assignments to a probability function p; the chain will have a stationary distribution equal to p itself; by running the chain for some number of time, we will thus sample from p.

7.4.1 Metropolis-Hastings method

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 centred on the current $x^{(t)}$, but in general can be any fixed density from which we can draw samples.

Remark 7.4. *In contrast to importance sampling and rejection sampling, it is not necessary $q(x'|x^{(t)})$ look at all similar to $p(x)$ in order for the algorithm to be practically useful. An example of a proposal density with two different states $(x^{(1)}, x^{(2)})$.*

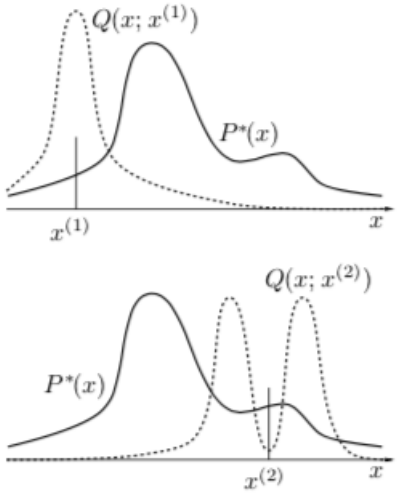


Figure 29.10. Metropolis–Hastings method in one dimension. The proposal distribution $Q(x'; x)$ is here shown as having a shape that changes as x changes, though this is not typical of the proposal densities used in practice.

We assume we can evaluate $\tilde{p}(x)$ for any x. 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

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

If $a \geq 1$¡ then the new state is accepted; Otherwise, the new state is accepted with probability a.
If accepted, set $x^{(t+1)} = x'$. Otherwise, set $x^{(t+1)} = x^{(t)}$.

Theorem 7.1. As $t \to \infty$, $\{x^{(r)}\}_{r=1}^{R} \to p(x)$ for any $q(x'|x^{(t)}) \geq 0$.

Remark 7.5. *Just as it was difficult to estimate the variance of an importance sampling estimator, so 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.*

8 Amortized Inference and Variational Auto-Encoders

"Amortize" just means "spread out a cost over time". Instead of doing SVI from scratch every time we see a new datapoint, we're going to try to gradually learn a function that can look at the data for a person x_i, and then output an approximate posterior $q_\phi(z_i|x_i)$. We'll call this a "recognition model".
Instead of a separate ϕ_i for each data example, we'll just have a single global ϕ that specifies

the parameters of the recognition model. Because the relationship between data and posteriors is complex and hard to specify by hand, we'll do this with a neural network.

8.1 the algorithm for amortized inference

1. Sample a datapoint
2. Compute parameters ϕ of approximate posterior
3. Compute gradient of Monte Carlo estimate of ELBO wrt ϕ
4. Update ϕ

8.2 Optimizing model parameters

Didn't really understand this section

8.3 Variational Autoencoder (VAE)

An autoencoder takes some data as input and discovers some latent state representation of the data. The encoder network takes in the input data (such as an image) and outputs a single value for each encoding dimension. The decoder takes this encoding and attempts to recreate the original input.

Example 8.1. MNIST Let's give an explicit model for MNIST images of handwritten digits. We will choose our prior on z to be the standard Gaussian

step 1: insert a prior of each latent variable; step 2: estimate the distribution of the observed variable; step 3: calculate the posterior

$$\mathcal{N}(0, I)$$

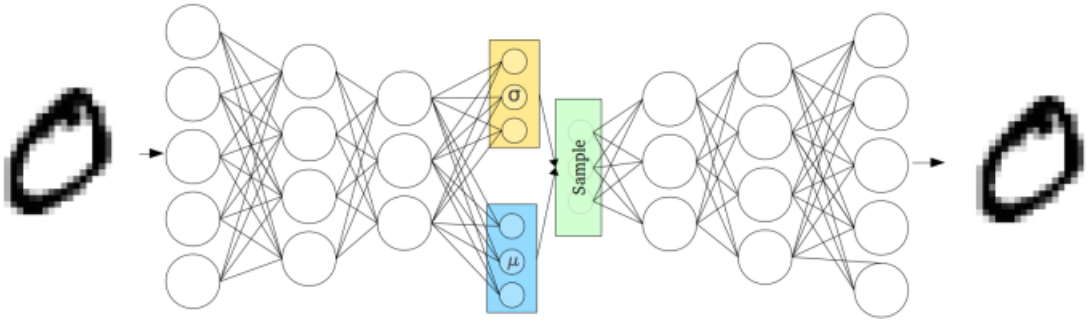
Our likelihood function is

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

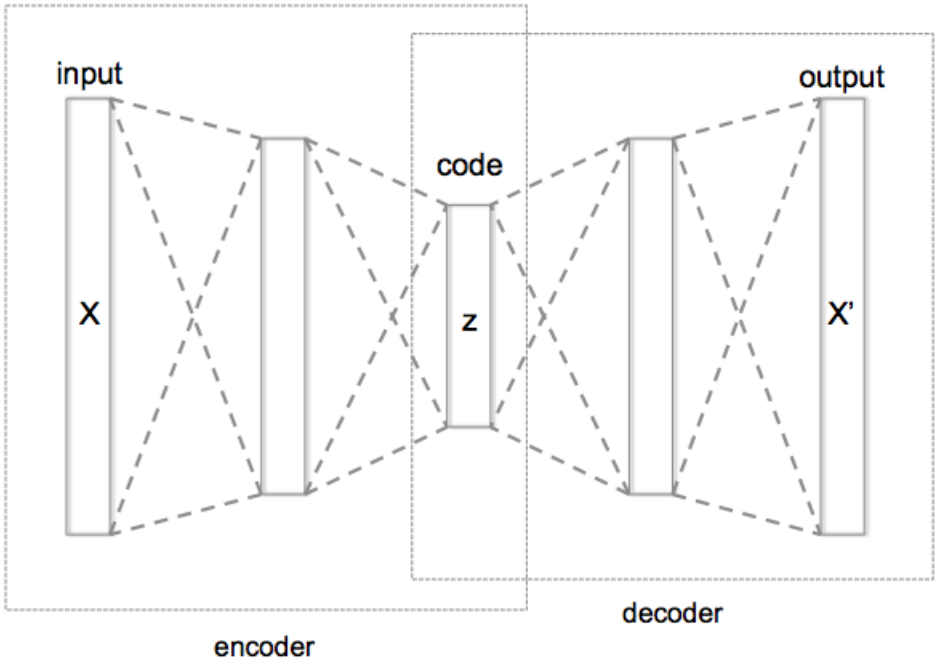
Our approximate posterior is

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

Define our encoder and decoder to be
Encoder: $g_\phi(x_i) = \phi_i = (\mu_i, \log \sigma_i)$ (learning the distribution of z)
Inputs x_is are encoded to vectors μ and $\log \sigma_i$, which parameterize $q_\phi(z|x)$. Before decoding, we draw a sample $z \sim q_\phi(z|x) = \mathcal{N}(\mu(x).\sigma(x)I)$.
Decoder: $f_\theta(z_i) = \theta_i$ (reconstructing the distribution of x)
Then evaluate $p_\theta(x_i|z_i)$. We compute the loos function (negative ELBO) and propagate its derivative with respect to θ and ϕ, through the networks during training.



Deterministic Autoencoders An autoencoder takes an input, encodes it into a vector, then decodes to produce something similar to the original data. Or: autoencoders reconstruct their own input using an encoder and a decoder.
Encoder: $g(x) \to z$
Decoder: $f(z) \to \hat{x}$
The encoder, $g(x)$, takes in the input data (such as an image) and outputs a single value for each encoding dimension while the The decoder, $f(z)$ takes this encoding and attempts to recreate the original input.
Our 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, into. It is important that the encoder reduces the dimension, for example by learning how to ignore noise (otherwise, we would just learn the identify function). The big idea is that the code contains only the most important features of the input, such that we can reconstruct the input from the code reliably

$$\tilde{x} = f(g(x)) \approx x$$

Problem 1 - Proximity in data space does not mean proximity in feature space

$$x_1 \approx x_2 \implies z_1 \approx z_2$$

Problem 2: "White" latent space region If the space has regions where no data gets encoded to, and you sample/generate a variation from there, the decoder will simply 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.

Solution: Adding noise to autoencoders

- Can add noise to data before encoding, reconstruct original data. But how much noise?
- Can add noise to latents after encoding, reconstruct original data. But how much noise?

Solution - Variational Autoencoders (VAEs) This stochastic generation means, that 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.

Why does a VAE solve the problems of a deterministic autoencoder?

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" instead of only being able to generate already seen data corresponding to the particular fixed encoded points.

9 Normalizing Flows

Normalizing Flows(NF)[1] are a family of generative models with tractable distributions where both sampling and density evaluation can be efficient and exact.

Applications

- Image generation
- Video generation
- Audio generation
- Graph generation

[1] reference: `https://arxiv.org/pdf/1908.09257.pdf`

- Reinforcement learning

Definition 9.1. Normalizing Flows is a learned transformation of a simple distribution (base) to a complex distribution (target) by a sequence of invertible and differentiable mappings.

9.1 Basics

Let $Z \in \mathbb{R}^D$ be a random variable with a known and tractable probability density function $p_Z : \mathbb{R}^D \to \mathbb{R}$. Let f be an invertible function and $Y = f(Z)$.

Definition 9.2 (generative direction)**.** Then using the change of variables formula, one can compute the probability density function of the random variable Y:

$$p_Y(y) = p_Z(g(y))|\det Dg(y)| = p_Z(g(y))|\det Df(g(y))|^{-1} \tag{9}$$

where g is the inverse of f, $Dg(u) = \frac{\partial g}{\partial y}$ is the Jacobian of G and $Df(z) = \frac{\partial f}{\partial z}$ is the Jacobian of f. This new density function $p_Y(y)$ is called a pushforward of the density p_Z by the function f and denoted by $f_* p_Z$.
In the context of generative models, the above function f (a generator) "pushes forward" the base density p_Z to a more complex density. This movement from base density to final complicated density is the generative direction

Remark 9.1. *To generate a data point y, one can sample z from the base distribution, and then apply the generator $y = f(z)$.*

Definition 9.3 (normalizing direction)**.** The inverse function g moves in the opposite, normalizing direction: from a complicated and irregular data distribution towards the simpler, more regular or "normal" form, of the base measure p_Z.

Fact 9.1. If the transformation f can be arbitrarily complex, one can generate any distribution p_Y from any base distribution p_Z.

Constructing flows Constructing arbitrarily complicated non-linear invertible functions(bijections) can be difficult. Since the composition of invertible functions is itself invertible, and the determinant of its Jacobian has a specific form, we can use a sequence of simple bijections which are convenient to compute, invert, and calculate the determinant of their Jacobian.
In particular, let $f_1, \ldots, f_N$ be a set of N bijective functions and define $f = f_N \circ f_{N-1} \circ \ldots \circ f_1$ to be a composition of the functions. It can be shown that f is also bijective, with inverse

$$g = g_1 \circ \ldots \circ g_{N-1} \circ g_N$$

and the determinant of the Jacobian is

$$\det Dg(y) = \prod_{i=1}^{N} \det Dg_i(x_i)$$

where $x_i = f_i \circ \ldots \circ f_1(z) = g_{i+1} \circ \ldots \circ g_N(y)$ and so $x_N = y$.

9.1.1 More formal construction

Definition 9.4. If $(Z, \Sigma_Z), (Y, \Sigma_Y)$ are measurable spaces, f is a measurable mapping between them, and μ is a measure on Z, then one can define a measure on Y (called the pushforward measure and denoted by $f_*\mu$ by

$$f_*\mu(U) = \mu(f^{-1}(U)), \quad \text{for all } U \in \Sigma_Y$$

Data can be understood as a sample from a measured "data" space (Y, Σ_Y, v) which we want to learn. To do that one can introduce a simpler measured space $(Z, \Sigma_Z, \mu$ and find a function $f : Z \to Y$ s.t. $v = f_*\mu$. This function f can be interpreted as a "generator", and Z as a latent space.
Assume $Z = \mathbb{R}^D$, all sigma-algebras are Bore,, and all measures are absolutely continuous with respect to Lebesgue measure(i.e. $\mu = p_Z dz$)

Definition 9.5. A function $f : \mathbb{R}^D \to \mathbb{R}^D$ is called a diffeomorphism, if it is bijective, differentiable, and its inverse is differentiable as well.
The pushforward of an absolutely continuous measure $p_Z dz$ by a diffeomorphism f is also absolutely continuous with a density function given by (9).

9.2 Applications

9.2.1 Density estimation and sampling

For simplicity assume that only a single flow, f, is used and it is parameterized by the vector θ. Further assume that the base measure p_Z is given and is parameterized by the vector ϕ. Given a set of data observed from some complicated distribution, $D = \{y_i\}_{i=1}^M$, we can then perform likelihood-based estimation of the parameters $\Theta = (\theta, \phi)$. The data likelihood in this case simply becomes

$$\begin{aligned} \log p(D|\Theta) &= \sum_{i=1}^{M} \log p_Y(y_i|\Theta) \\ &= \sum_{i=1}^{M} \log p_Z(g(y_i|\theta)|\phi) + \log|\det Dg(y_i|\theta)| \end{aligned}$$

During training, the parameters of the flow (θ) and of the base distribution (ϕ) are adjusted to maximize the log-likelihood.

9.3 Methods

Normalizing Flows should satisfy several conditions in order to be practical. They should

- Be invertible (for estimating the density we need to know their inverse)
- Be expressive enough to model real distributions
- Be computationally efficient: calculation of the Jacobian determinant, sampling from the base distribution, and application of the forward and inverse functions should be tractable.

In this section, we describe different types of flows and comment on the above properties.

9.3.1 Elementwise bijections

Let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. Then, if $x = (x_1, x_2, \ldots, x_D)^T$,

$$f(x) = (h(x_1), h(x_2), \ldots, h(x_D))^T$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h (since the Jacobian matrix is diagonal). This can also be generalized by allowing each element to have its own distinct bijective function.

Problem Elementwise operations cannot express any form of correlation between dimensions.

9.3.2 Linear Flows

Linear mappings can express correlation between dimensions:

$$f(x) = Ax + b$$

where $A \in \mathbb{R}^{D \times D}$ and $b \in \mathbb{R}^D$ are parameters. If A is an invertible matrix, the function is invertible.

Problem Limited expressiveness.

Example 9.1 (Gaussian base distribution)**.**

$$p_Z(z) = \mathcal{N}(z, \mu, \Sigma)$$

After transformation by a linear flow, the distribution remains Gaussian:

$$p_Y = \mathcal{N}(y, A\mu + b, A^T \Sigma A)$$

More generally, a linear flow of a distribution from the exponential family remains in the exponential family.

Computing the determinant

$$|\det Df(x)| = \det(A)$$

which can be computed in $\mathcal{O}(D^3)$, which is expensive for large D. By restricting the form of A (at the expense of expressive power) we can avoid the computational costs:

Diagonal If A is diagonal with nonzero diagonal entries, then complexity of computing inverse or the determinant is $\mathcal{O}(D)$. However, the result is an elementwise transformation and cannot express correlation between dimensions.

Remark 9.2. A diagonal linear flow is useful for representing normalization layers, which have become a ubiquitous part of modern neural networks.

Triangular More expressive form of linear transformation. Taking the determinant costs $\mathcal{O}(D)$, inversion costs $\mathcal{O}(D^2)$ (a single pass of back-substitution).

Permutation and Orthogonal The expressiveness of triangular transformations is sensitive to the ordering of dimensions. Reordering the dimensions can be done easily using a permutation matrix (absolute determinant = 1).
However, the permutations cannot be directly optimized.
A more general alternative is orthogonal transformations, whose inverse and absolute determinant are both trivial to compute.

Factorizations Instead of limiting the form of A, one can also use the LU factorization:

$$f(x) = PLUx + b$$

where L is lower triangular with ones on the diagonal, U is upper triangular with non-zero diagonal entries, and P is a permutation matrix.
The determinant is the product of the diagonal entries of U which can be computed in $\mathcal{O}(D)$.
The inverse of f can be computed using two passes of backward substitution in $\mathcal{O}(D^2)$.
However, the discrete permutation P cannot be easily optimized.

Convolution Easy to compute, but it can be difficult to efficiently calculate the determinant or unsure invertibility.

9.3.3 Planar and Radial Flows

Relatively simple but inverses aren't easily computed $\implies$ not widely used in practice.

Planar flows Expand and contract the distribution along certain specific directions:

$$f(x) = x + uh(w^T x + b)$$

where $u, w \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are parameters and $h : \mathbb{R} \to \mathbb{R}$ is a smooth non-linearity. The Jacobian determinant for this transformation is

$$\begin{aligned} \det\left(\frac{\partial f}{\partial x}\right) &= \det(\mathbb{1}\{D\} + uh'(w^T x + b)w^T) \\ &= 1 + h'(w^T x + b)u^T w \end{aligned}$$

This can be computed in $\mathcal{O}(D)$ time.
The inversion of this flow isn't possible in closed form and may not exist for certain choices of $h(\cdot)$ and certain parameter settings.

Sylvester flows A more expressive form of planar flows:

$$f(x) = x + Uh(W^T x + b)$$

where $U, W \in \mathbb{R}^{D \times M}, b \in \mathbb{R}^M, h : \mathbb{R}^M \to \mathbb{R}^M$ is an elementwise smooth nonlinearity, where $M \leq D$ is a hyperparameter to choose and which can be interpreted as the dimension of a hidden layer. In this case the Jacobian determinant is

$$\begin{aligned} \det\left(\frac{\partial f}{\partial x}\right) &= \det(\mathbb{1}\{D\} + Udiag(h'(W^T x + b))W^T) \\ &= \det(\mathbb{1}\{M\} + diag(h'(W^T x + b))WU^T) \end{aligned}$$

This can be computationally efficient if M is small.

Radial flows Radial flows instead modify the distribution around a specific point so that

$$f(x) = x + \frac{\beta}{\alpha + ||x - x_0||}(x - x_0)$$

where $x_0 \in \mathbb{R}^D$ is the point around which the distribution is distorted, and $\alpha, \beta \in \mathbb{R}$ are parameters, $\alpha > 0$.
The Jacobian determinant can be computed relatively efficiently, but the inverse cannot be given in closed form but does exist under suitable constraints on the parameters.

9.3.4 Coupling Flows

One of the most widely used flow architectures.

Coupling flows Consider a (disjoint) partition of the input $x \in \mathbb{R}^D$ into two subspaces: $(x^A, x^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$ and a bijection $\hat{f}(\cdot; \theta) : \mathbb{R}^d \to \mathbb{R}^d$, parametrized by θ. Then one can define a function $f : \mathbb{R}^D \to \mathbb{R}^D$ by the formula:

$$\begin{aligned} y^A &= \hat{f}(x^A; \Theta(x^B)) \\ y^B &= x^B \end{aligned}$$

where the parameters θ are defined by arbitrary function $\Theta(x^B)$ which only uses x^B as input. This function is called a conditioner. The function $\hat{f}$ is called a coupling layer, and the resulting function f is called a coupling flow.
A coupling flow is invertible if and only if $\hat{f}$ is invertible and has inverse:

$$\begin{aligned} x^A &= \hat{f}^{-1}(y^A; \Theta(x^B)) \\ x^B &= y^B \end{aligned}$$

The Jacobian of f is a block triangular matrix where the diagonal blocks are $D\hat{f}$ and the identity matrix respectively. Hence

$$\det Df = \det D\hat{f}$$

Most coupling layers are applied to x^A element-wise:

$$\hat{f}(x^A;\theta) = (\hat{f}_1(x_1^A;\theta_1),\dots,\hat{f}_d(x_d^A;\theta_d))$$

where each $\hat{f}_i(\cdot;\theta_i) : \mathbb{R} \to \mathbb{R}$ is a scalar bijection. In this case a coupling flow is a triangular transformation (i.e., has a triangular Jacobian matrix).
The power of a coupling flow resides in the ability if a conditioner $\Theta(x^B)$ to be arbitrarily complex. In practice it is usually modelled as a neural network (e.g. a shallow ResNet).

9.3.5 Autoregressive Flows

One of the most widely used flow architectures.

Direct autoregressive flows Let $\hat{f}(\cdot;\theta) : \mathbb{R} \to \mathbb{R}$ be a bijection parameterized by θ. Then an autoregressive model is a function $f : \mathbb{R}^D \to \mathbb{R}^D$, which outputs each entry of $y = f(x)$ conditioned on the previous entries of the input:

$$y_t = \hat{f}(x_t;\Theta_t(x_{1:t-1}))$$

For each $t = 2,\dots,D$ we choose arbitrary functions $\Theta_t(\cdot)$ mapping $\mathbb{R}^{t-1}$ to the set of all parameters, and Θ_1 is a constant. The functions $\Theta_t(\cdot)$ are called conditioners.
The Jacobian matrix of the autoregressive transformation f is triangular. Each output y_t only depends on $x_{1:t}$, and so the determinant is just a product of its diagonal entries:

$$\det(Df) = \prod_{t=1}^{D} \frac{\partial \hat{f}}{\partial x_t}$$

In practice, it is possible to compute all the entires of the direct flow in one pass using a single network with appropriate masks.
However, the computation of the inverse is more challenging (involves recursion, cannot be parallelized)

Inverse autoregressive flows (IAF) Outputs each entry of y conditioned the previous entries of y (w.r.t. the fixed ordering)

$$y_t = \hat{f}(x_t;\theta_t(y_{1:t-1}))$$

Computation of the IAF is sequential and expensive, but the inverse of IAD (which is a direct autoregressive flow) can be computed relatively efficiently.

Remark 9.3. One should use IAFs if fast sampling is needed (e.g. for stochastic variational inference), and DAFs if fast density estimation is desirable. However, the two methods are theoretically equivalent and can learn the same distribution.

9.3.6 Residual Flows

Residual networks are compositions of the functions of the form

$$f(x) = x + F(x)$$

Such a function is called a residual connection, and here the residual block $F(\cdot)$ is a feed-forward neural network of any kind.
The central idea of a reversible network architecture based on residual connections is a variation of additive coupling layers: consider a disjoint partition of $\mathbb{R}^D = \mathbb{R}^d \times \mathbb{R}^{D-d}$ denoted by $x = (x^A, x^B)$ for the input and $y = (y^A, y^B)$ for the output, and define a function:

$$y^A = x^A + F(x^B)$$

$$y^B = x^B + G(y^A)$$

where $F : \mathbb{R}^{D-d} \to \mathbb{R}^d$ and $G : \mathbb{R}^d \to \mathbb{R}^{D-d}$ are residual blocks.
This network is invertible but computation of the Jacobian is inefficient.

9.3.7 Infinitesimal(Continuous) Flows

The residual connections discussed in the previous section can be viewed as discretizations of a first order ordinary differential equation (ODE):

$$\frac{d}{dt}x(t) = F(x(t), \theta(t))$$

where $F : \mathbb{R}^D \times \Theta \to \mathbb{R}^D$ is a function which determines the dynamic (the evolution function), Θ is a set of parameters and $\theta : \mathbb{R} \to \Theta$ is a parameterization. The discretization of this equation (Euler's method) is

$$x_{n+1} - x_n = \epsilon F(x_n, \theta_n)$$

and this is equivalent to a residual connection with a residual block $\epsilon F(\cdot, \theta_n)$.
We can consider the case where we do not discretize but try to learn the continuous dynamical system instead. Such flows are called infinitesimal or continuous. There are two distinct types: **ODE-based methods** and **SDE-based methods (Langevin flows)**.