University of Trieste

International School for Advanced Studies

The Abdus Salam International Centre for Theoretical Physics

Probabilistic Machine Learning

Lectures Notes

Author:
Marco Sciorilli

Saturday 11th September, 2021

Abstract

This document contains my notes on the course of Probabilistic Machine Learning held by Prof. Luca Bortolussi for the Master Degree in Data Science and Scientific Computing at Trieste University in the year 2020/2021. As they are a work in progress, every correction and suggestion is welcomed. Please, write me at: marco.sciorilli@gmail.com .

Contents

Chapter 1

Introduction to the course

1.1 Introduction

A model is a kind of a step of abstraction: it describes a complex object with a simpler one understandable by us. More specifically a mathematical model is one that use the language of mathematics, describes systematically relations between objects. A stochastic model is a mathematical model which has probability distributions as objects. We use a set of parameters to build the model, and we try to find the best set of parameters to describe the observed object. We want to automatically chose the best model among all possibles, in a thought through way. A major difference with statistic is that we focus on the algorithm, rather than the data. This allows us to use model usually far more complicated then the statistical one. The course is about how to effectively find a way to describe data.

1.2 Kind of learning

Different kind of machine learning, explained in a probabilistic prospective:

- Supervised learning: the models are functions, from a input it gives an output. There can be categorical output. Most of nowadays machine learning is of this kind.
- Unsupervised learning: find pattern in data
- Reinforcement learning: making the best decision in a certain scenario.

In this course we will mostly stick to supervised learning.

1.3 Generative and discriminative models

Generative models aim at describing the full probability distribution of input using a joint probability distribution of a input and an output. Probabilities are a reasonable way to describe the world. Discriminative learning: wants to describe the conditional probability, in order to discriminate among inputs.

We will try to work mostly on joint probability distribution.

1.4 Machine learning

Supervised learning: learn the joint distribution.
Unsupervised: learn the input probability and its property p(x).
Data generation: learn how to sample. Or maybe create new input from the probability distribution of input.

1.5 Inference and estimation

Inference: starting with a complex distribution, and compute simpler conditional distributions. we want doing inference in an effective way.
Estimation: given a set of parameters, with wants to find the best value of such parameter in the model, even a very complex one, to best describe the input.
In Bayesian statistic, estimation became an operation of inference. Bayesian machine learning has the downside that the computation is way harder (and don't scale very well), but it usually get the most out of data. The asymptotic equalness with the frequentist approach depends on the complexity of the model, usually not suitable for the input data-set available.

Chapter 2

Empirical Risk Minimization

2.1 Basic definitions

Let's create a generalize framework to formulate machine learning problems.

Definition 1. *Supervised Learning: predicting a function linking an input to an output.*

Definition 2. *Input space $X \subseteq \mathbb{R}^n$: the features considered are **real** features (could also be 0 and 1, or categorical etc.)*

Definition 3. *Output space $Y \subseteq \mathbb{R}$: can be a real number (could also be 0 and 1, or categorical etc.)*

Empirical risk minimization framework is based on probability, what we have in general is that are trying to describe the world (or the part of the world considered in our problem) in terms of probability. The true system we are trying to capture is a joint probability distribution of the input and the output, which we call "the data distribution".

Definition 4. *Data distribution $p(x, y)$, where $x \in X$ is an element of the input space, $y \in Y$ is an element of the output space. Data distribution $p(x, y) \in Dist(X \times Y)$ a distribution over x time y.*

What we are trying to do is to describe a link, a function link, between input and output, which rather than being strictly deterministic, is allowed to be probabilistic in nature. E.g. given a certain input we may observe several outcomes depending on some random process which is out of the detailed description we want to capture (so it is represented as a probability distribution). Maybe there is a way to understand the details of the problem we are studying, but those are likely too difficult to describe mathematically.
Sometimes we have a functional relationship between input and output (true functional relationship) $f : X \to Y$.
In this case we typically have the joint probability distribution which we can factorize as $p(x, y) = p(x)p(y|x)$ and $p(y|x) = p(y|f(x))$ i.e. the conditional distribution of y depends only on the value of $f(x)$ rather then on x itself. This is typically the case of classification problems.

In machine learning we want to learn this world, and we want to learn it from observations.

Definition 5. *Dataset D, with a certain cardinality $|D| = N$, $D \sim p^n(x, y)$ i.e. D is sampled from a probability distribution. This means that $D = \{(x_i, y_i)|i = 1, ..., N\}$, $(x_i, y_i) \sim p(x, y)$ i.e. D is composed by n pairs of inputs and outputs, and each of this pair is sampled from $p(x, y)$ independently from the other pairs.*

This is the scenario in which we are working. We would like to recover at least, if it is there, the function f which map input to output.
Whenever we want to learn, we make hypothesis on which are good candidates functions of our model i.e. we choose a set of functions that for us should contain a good model, maybe even the true model, for our data. This set of functions is our hypothesis class.

Definition 6. *Hypothesis class $H = \{h : X \to Y\}$ is a set of functions from input to output. Typically this set of function is depending on a parameter $h = h_\theta$, $\theta \in \Theta \subseteq \mathbb{R}^k$.*

The assumptions we make on the hypothesis class are the crucial ones. This assumptions is what allows us to learn.
H **encodes our inductive bias**. I.e. in our induction process, we are biasing out induction itself in a certain direction, which is defined by the hypothesis class.
We need a way to determine how good our choice of hypothesis match the date we observe, and what we predict to observe.

Definition 7. *Loss function $l(x, y, h) \in \mathbb{R}_{\geq 0}$: takes an input x, an output y and a function h, and returns how much error we commit using h to predict y. The higher the value, the worst is our prediction.*

Example 1. *Here some examples of loss functions:*

- *0-1 loss: $l(x, y, h) = I(h(x) \neq y)$, for a classification problem ($y \in \{0, 1\}$), is the indicator function that tell us if $h(x) \neq y$. There is no error if we are correctly predicting the class.*
- *Square loss: $l(x, y, h) = (h(x) - y)^2$, for a regression problem ($y \in \mathbb{R}$), is the square difference between our prediction and the observed values. It is always positive and differentiable.*

2.2 Risk and Empirical risk

The loss function acts on a single input-output pair, but we want a function h which work well on any pair according to the joint probability distribution $p(x, y)$. We can encode it with the use of risk.

Definition 8. *Risk, or Generalization error, $R(h) = \mathbb{E}_{x,y \in p(x,y)}[l(x, y, h)]$: every function h comes with a risk, defined as the expectation over the pairs x, y sampled over the true data generating distribution of the loss of the pair, according to the hypothesis h. e.g. What is the fraction of pair we are going to misclassify adopting the hypothesis h?*

Generalization error depends on the true data generating distribution, which we usually do not know. So in practice, we can replace the risk with something computable, called the empirical risk.

Definition 9. *Empirical risk, also called training error on $D \sim p^N$, $\hat{R}_D(h)$:it is the empirical approximation according to the sample of the actual risk. $\hat{R}_D(h) = \frac{1}{N}\sum_{i=1} l(x_i, y_i, h)$, i.e. the average of the loss, sampling N time from the distribution.*

Definition 10. *Risk minimization principle, find $h^* \in H$ s.t. $h^* = \arg\min_{h\in H} R(h)$: we want to find a function h^* in our hypothesis set such that it is the one with the minimum risk among the ones of our hypothesis set.*

If l is the 0-1 loss and $\exists h \in H$ *s.t.* $p(h(x) = f(x)) = 1$, where $f(x)$ is the true class, than H has the **realizability** property, so $R(h^*) = 0$. This means that we can actually find the true model. This is typically not the case.

Definition 11. *Empirical risk minimization, find $h^* \in H$ s.t. $h^*_D = \arg\min_{h\in H} \hat{R}_D(h)$: the same but with empirical risk.*

2.3 Bias Variance Tradeoff

Sometime finding the minimum of the empirical risk is very calculus intensive, so we are just satisfied with a good approximation. Also, what is $R(h^*_D)$ associated with my solution? We can understand the relationship between the true and the empirical risk through the bias-variance trade-off, and we can see if we can give error bound the true generalization error (probably approximately correct learning).

Bias-Variance trade-off in a regression problem

Let's consider the case of a regression problem. We are therefore considering the square loss $h \in H$. So the explicit expression of the generalization error choosing an hypothesis h is:

$$R(h) = \mathbb{E}_p[l(x, y, h)] = \int\int (h(x) - y)^2 p(x, y)dxdy \tag{2.1}$$

We can define a minimizer of R ($g = \arg\min_h R(h)$, if $g \in H$) as:

$$g(x) \coloneqq \mathbb{E}[y|x] = \int yp(y|x)dy \tag{2.2}$$

Proof. Let's write the generalisation error as:

$$R(h) = \int\int (h(x) - y)^2 p(x, y)dxdy \tag{2.3}$$

We can rewrite the inner term of the integral to make $g(x)$ appear

$$\begin{aligned}(h(x) - y)^2 &= (h(x) - g(x) + g(x) - y)^2 \\ &= (h(x) - g(x))^2 + (g(x) - y)^2 + 2(h(x) - g(x))(g(x) - y)\end{aligned}$$

The lower grade term in the integral is going to cancel out, because in the moment we integrate in respect to y, as is not appearing in the first part, became

$$\int 2(h(x) - g(x))(g(x) - y)p(y, x)dydx = \int 2(h(x) - g(x)) \int (g(x) - y)p(y|x)dyp(x)dx \tag{2.4}$$

The second integral is encoding for the conditional expectation:

$$\int (g(x) - y)p(y|x)dy = g(x) - \int yp(y|x)dy = 0 \tag{2.5}$$

So the lower grade term really cancel out.

$$R(h) = \int (h(x) - g(x))^2 p(x)dx + \int \int (g(x) - y)^2 p(x, y)dxdy \tag{2.6}$$

The second term depends only on the conditional expectation, not on the function we are evaluating, so it is essentially a constant, equal for all hs. It tells us how much y is varying across its conditional expectation (the conditional variance of y integrated over all possible xs). This is a sort of term expressing the idea of how noisy is our regression problem, i.e. how much we can deviate from the conditional expectation when we actually observe our process.
The first term is 0 if and only if $h(x) = g(x)$, which shows that $g(x)$ is a minimizer.

□

Now we are going to evaluate our error for a specific h function, the one we obtain by empirical risk minimization principle when we observe dataset D of size N generated according to the distribution p ($D \sim p^n$):

$$R(h_D^*) = \int (h_d^*(x) - g(x))^2 p(x)dx + noise \tag{2.7}$$

We want to study now the expected value of the generalization error with respect to our data. It is the average error we are going to make by adopting the empirical risk minimization to replace the true error.

$$\mathbb{E}_D[R(_D^*)] = \int \mathbb{E}_D[(h_D^* - g(x))^2]p(x)dx \tag{2.8}$$

If we add and subtract in the square parenthesis the expectation with respect to D of our function ($\pm\mathbb{E}_D(h_D^*(x))$). We can work out the square as before, and observe that $\mathbb{E}_D(h_D^* - \mathbb{E}_D[h_D^*(x)]) = 0$ (the expected deviation of an element from its mean is 0). So:

$$\begin{aligned}\mathbb{E}_D[R(h_D^*)] = &\int \mathbb{E}_D(h_D^* - g(x))^2 p(x)dx \\ &+ \int \mathbb{E}_D[(h_D^*(x) - \mathbb{E}_D[h_D^*])^2 p(x)dx \\ &+ \int \int (g(x) - y)^2 p(x, y)dxdy\end{aligned}$$

The term on top captures the square differences between the average predictor across all dataset, obtained from empirical risk minimisation, and the optimal predictor, which is the conditional expectation. If the difference is small, on average across all datasets our empirical risk minimisation is going to work, but if this difference is large, it means that across all datasets our empirical risk minimisation is not going to work well. We call this term $bias^2$ (squared-bias) because it essentially captures the intrinsic distortion of our

empirical risk minimization predictor, and so, of our set of hypothesis, and how well we can reconstruct the true function.
The second term encodes an expectation across all datasets of the variance of our predictors, i.e. how far each single instance of the empirical risk minimisation framework differs from the average predictor. It is called *variance* term. The larger the variance, the more the noisy are going to be the single instances of our framework, i.e. how the intrinsic variability of our dataset is going to impact on what we can reconstruct. High variance means we are in a region of over-fitting, low variance the opposite. High bias at the opposite means we are in a region of under-fitting, where we are not able to capture the true dynamic of what is going on.
The last term is the noise. What we observe is that the sum of bias and variance in relation to the complexity of the model, has a kind of convex shape, with a minimum where there is an optimal trade-off between this two values.

Chapter 3

PAC learning

3.1 Definition of PAC learning

We focus from now on the 0-1 loss function, $y = \{0, 1\}$. The hypothesis set function has the reliazability property, that for our case can be written as

$$\exists \overline{h} \in H \ s.t. \ p_{x,y}(\overline{h}(x) = y) = 1 \tag{3.1}$$

Definition 12. *H is PAC learnable if and only if*

$$\begin{aligned}
&\forall \epsilon, \delta \in (0, 1), \ \forall p(x, y) \\
&\exists m_{\epsilon,\delta} \in \mathbb{N} \ s.t. \ \forall m \geq m_{\epsilon,\delta}, \ \forall D \sim p^m, \ |D| = m \\
&P_D(R(h^*_D) \leq \epsilon) \geq 1 - \delta
\end{aligned}$$

This means: take a scenario (a data-generating distribution), and take an ϵ and a δ (two parameters governing our precision). Once we fix this parameters, we find a number m as a function of ϵ and δ that is telling us that we can learn with error bounded by ϵ the true function (assuming that the true function, the closest to the real one, belong to the set H). If we have data which is more than $1 - \delta$ points, then this is going to happen with high probability.
PAC literally means: probably approximately correct.

Definition 13. *H, A (the learning algorithm, i.e. the mechanism which returns the optimal function) is agnostic PAC learnable if and only if*

$$\begin{aligned}
&\forall \epsilon, \delta \in (0, 1), \ \forall p(x, y) \\
&\exists m_{\epsilon,\delta} \in \mathbb{N} \ s.t. \ \forall m \geq m_{\epsilon,\delta}, \ \forall D \sim p^m, \ |D| = m \\
&R(h^A_D) \leq \min_{h \in H} R(h') + \epsilon \ \textit{with probability} \ 1 - \delta
\end{aligned}$$

Where h^A_D is the result of A on H, D. Observations:

- *$m_{\epsilon,\delta}$: typically we require it to depend polynomially on $\frac{1}{\epsilon}, \frac{1}{\delta}$. This is because we do not want a bound to depend on a huge number of observations: it has to increase moderately with the complexity.*
- *A should run in polynomial time. This is because we want to limit the complexity of the algorithm.*

3.2 Learning finite hypothesis sets

Definition 14. *Class of finite hypothesis: H s.t. $|H| < \infty$*

The true solution may or may not be contained inside. Because we have a finite set of hypothesis functions, there is always a way in which we can discriminate, or assign classes to points: our representation power is finite, and prior that we have enough samples, we should be able to cover up all possibilities.

Definition 15. *H s.t. $|H| < \infty$ is agnostic PAC learnable with:*

$$m_{\epsilon,\delta} \leq \left\lceil \frac{2\log\left(\frac{2|H|}{\delta}\right)}{\epsilon^2} \right\rceil \tag{3.2}$$

Where $\log|H|$ is the measure of complexity of H.

Remark. If H is described by d parameters of type `DOUBLE` (64 bits), then $|H| \leq 2^{d\cdot 64}$, then

$$m_{\epsilon,\delta} \leq \frac{128 \cdot d + 2\log\left(\frac{2}{\delta}\right)}{\epsilon^2} \tag{3.3}$$

3.3 VC dimension

In most of the cases, at least from a theoretical prospective, we would like to reason on a infinite classes of functions (e.g. lines on a plane, boxes in space etc.). We need to define a proper notion of complexity for a class of functions to be able to say something meaningful.

Example 2. *Let's consider the plane $\mathbb{R}^2$, and two class: axis-lined lines and boxes. How many configurations of data points are those classes able to separate (to classify them correctly)?*

Let's take n points, and see if we can correctly classify them. To do so, we have to consider that all possible assignment could happen. So we have to essentially relativize a set of hypothesis function to a finite set of points. Here is a formal definition:

Definition 16. *Given a class of hypothesis functions ($H = \{h : X \to \{0,1\}\}$), a subset C of X of finite cardinality ($C \subseteq X,\ |C| = m,\ C = \{c_1, ..., c_m\} \subseteq X$). We define H relativize to C (or H to C) as:*

$$H_C = \{(h(c_1), ..., h(c_m) | h \in H\} \tag{3.4}$$

i.e. takes this points and apply one of our functions to this points and check which are the tuples of $\{0,1\}$ *we can generate.*

Remark. We say that H **shatters** C if and only if $|H_C| = 2^m$.

So we can define

Definition 17. *VC dimension:*

$$VCdim(H) = \max\{m | \exists C \subseteq X, |C| = m \text{ s.t. } H \text{ shatters } C\} \tag{3.5}$$

Its possible that the VC dimension is infinite.

3.4 VC dimension and PAC learning

Proposition 1. *If H shatters C, $|C| \geq 2m$, then we cannot learn H with m samples. So if $VCdim(H) = \infty$, H in not PAC learnable.*

i.e. there will be a specific true function that assign classes to n points such as we commit an error with high probability. This allow to state that if a set has infinite VC dimension, is not PAC learnable.
Infinite VC dimension means that whenever we have n points, whatever their assignment of class, we will always find a function which will captures exactly this assignment. Basically, every effect of noise will be captured with empirical loss 0, so we can over-fit as much as we want.

Theorem 2. *H is (agnostic) PAC learnable if and only if the VC dimension (of H) is finite ($< \infty$). In this case $\exists c_1, c_2$:*

$$c_1 \frac{VCdim(H) + \log\frac{1}{\delta}}{\epsilon^2} \leq m_{\epsilon,\delta} \leq c_2 \frac{VCdim(H) + \log\frac{1}{\delta}}{\epsilon^2} \tag{3.6}$$

i.e. the VC dimension is ruling the bound.

3.5 Rademacher Complexity

Complexity of a set of functions, more general then the VC dimension.

Definition 18. *Given $p(x,y)$, $D \sim p^m$, $H = \{h : X \to \{-1,1\}\}$, and defined the rademacher distribution as*

$$\sigma = (\sigma_1, ..., \sigma_m),\ \sigma_1 \in \{-1,1\},\ p(\sigma_1 = +1) = 0.5 \tag{3.7}$$

The rademacher complexity for dataset D is defined as

$$\hat{\mathcal{R}}_D(H) = \mathbb{E}_\sigma\Big[\sup_{h \in H} \frac{1}{m} \sum_{i=1}^{m} \sigma_i \cdot h(x_i)\Big] \tag{3.8}$$

i.e. $\sigma_i \cdot h(x_i)$ is the scalar product of the rademacher distribution with $h(x)$ evaluated on our dataset ($\frac{1}{m}\sigma_i \cdot h(x_i) \in [-1, 1]$) and is a measure of correlation between h and the random noise. If this value is 1, for the specific sample of random noise considered, we can obtain a perfect correlation, perfectly representing what we see.
For a specific σ we are going to choose the best h that correlates with that noise, and then take the expectation over the possible values of σ.
This is a property of both the function and of the dataset observed. We can define the rademacher complexity independent from the data, both rather dependent on their size.

Definition 19. *Rademacher complexity data independent:*

$$\mathcal{R}_m(H) = \mathbb{E}_{D\sim p^m}[\hat{\mathcal{R}}_D(H)] \tag{3.9}$$

Still depends on H and on p.

Let's state the PAC learning bounds on the rademacher complexity.
We fix H and $p(x, y)$; given $\forall\delta > 0$, with probability $\geq 1 - \delta$, for any $D \sim p^m$, $|D| = m$ and $\forall h \in H$, with have the following bound on the generalization error

$$R(h) \leq \hat{R}_D(h) + \mathcal{R}_m(H) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{3.10}$$

Where m is the number of data points. The last two terms play the role of ϵ. And the bound on the generalization error where we fix the data on the data-dependent rademacher complexity

$$R(h) \leq \hat{R}_D(h) + \mathcal{R}_D(H) + 3\sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{3.11}$$

Where also the last two term play the role of ϵ.

3.6 Rademacher Complexity and VC dimension

The links with VC dimension comes through the so called growth function. This function is telling us how the complexity grows increasing the cardinality of the number of data points we have.

Definition 20. *Grow function:*

$$\Pi_H : \mathbb{N} \to \mathbb{N} \qquad \Pi_H(m) = \max_{C\subseteq X, |C|=m} |H_C| \tag{3.12}$$

where $H_C = \{(h(c_1), ..., h(c_m))|h \in H, C = \{c_1, ..., c_m\}\}$

We have that $VCdim(H) = \max\{m|\Pi_H(m) = 2^m\}$. If this is infinite, it means that the complexity of the function h allows us to find a set of an arbitrary large number of points that can be classified in an arbitrary way. The growth function is an intermediate step, we can bound growth function by an expression depending on the VC dimension, and than bound the rademacher complexity with an expression depending on the growth function. We can combine this bounds and see that there is a dependency depending on the relationship between the two.

Lemma 3. *The relationship between VC dimension and rademacher complexity is give by the Sauer Lemma:*

$$\Pi_H(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{e \cdot m}{d}\right)^d = O(m^d) \tag{3.13}$$

where $d = VCdim(H)$

i.e. when d is finite, the growth function grows exponentially until d, then it start grows polynomially with a degrees that depends on the VC dimension set.
It holds that

$$\mathcal{R}_m(H) \leq \sqrt{\frac{2 \log \Pi_H(m)}{m}} \tag{3.14}$$

This different ways of measuring complexity are roughly similar, they gives us consistent results.

$$\mathcal{R}_m(H) \approx VCdim(H) \tag{3.15}$$

3.7 Empirical Risk Minimization and Maximum Likelihood

Quick Max-Likelihood recap

Given $D = \{(x_i, y_i)\}_{i=1,\dots,m}$, $D \sim p^m$ and $p = p(x, y)$, in the maximum likelihood scenario what we do is consider the data generating distribution, factorize it, and we typically make some hypothesis on the conditional distribution function (i.e. try to express it in a parametric form depending on a certain set of parameters θ, $p(y|x) = p(y|x, \theta)$).
In maximum likelihood we write the log-likelihood of the probability of the data given θ(i.e. observing y_i given x_i and a certain θ):

$$\mathcal{L}(\theta; D) = \sum_{i=1}^{m} \log p(y_i|x_i, \theta) \tag{3.16}$$

And we choose the parameter so that

$$\theta_{ML} = \arg\max_{\theta} \mathcal{L}(\theta; D) = \arg\min_{\theta} -\mathcal{L}(\theta; D) \tag{3.17}$$

Playing around a bit

$$\begin{aligned} \arg\min -\mathcal{L}(\theta; D) &= \arg\min_{\theta} -\frac{1}{m} \sum_{i=1}^{m} \log p(y_i|x_i, \theta) \\ &\approx \arg\min_{\theta} \mathbb{E}_{p(x,y)}[-\log p(y|x, \theta)] \end{aligned}$$

The expression $-\frac{1}{m} \sum_{i=1}^{m} \log p(y_i|x_i, \theta)$ is called cross-entropy.

Empirical Risk Minimization and Maximum Likelihood bridge

In the maximum likelihood framework we have a loss function

$$l(x, y, \theta) = -\log p(y|x, \theta) \tag{3.18}$$

But what is H?

- For regression $h(x, \theta) = \mathbb{E}_y[p(y|x, \theta)]$.
- For classification $h(x, \theta) = \arg\max_y p(y|x, \theta)$. This is the Bayes decision rule.

Choice of $p(y|x, \theta)$

- Regression $p(y|x, \theta) = \mathcal{N}(h_\theta(x), \sigma^2)$ (i.e. a normal distribution centered in some function, i.e. the expected value).
  This implies that
  $$-\log p(y|x, \theta) \propto (y - h_\theta(x))^2 \tag{3.19}$$
  Essentially the loss based on the sum of square comes out of a probabilistic assumption that there is a Gaussian noise that generates the observation, which mean is equal to the true value, with a certain standard deviation.

3.8 Introduction to Information Theory

Entropy: the idea is to use it to describe the average amount of information that is conveyed by something which has a probabilistic nature.

Definition 21. *Given a probability distribution $p(x)$, we define $-\log p(x)$ as a measure of self information.*

i.e. Let's imagine that $p(x) = 1$, so x is always happening. There is no information conveyed by $p(x)$ because we know that it is always happening. If $p(x) = 0$, then if x happens there should be an infinite quantity of information: something is very wrong in our model. When the probability distribution is very small, the self information carried is very large: the event is very unexpected. We want to attach a measure of information to the distribution itself: the expectation of the self information, the entropy.

Definition 22. *Entropy: $\mathcal{H}[p] = \mathbb{E}_p[-\log p(x)] = -\int p(x) \log p(x) dx$*
Entropy is maximum in the continuum case comes for a given fix minimum variance, the normal of Gaussian random variable.
In the discrete case: $\mathcal{H}[p] = -\sum_i p(x_i) \log p(x_i)$
Entropy is maximum in the discrete case for the uniform distribution, when $= \log K$, where K is the amount of different events that can happen

Definition 23. *Kullback Leibler divergence, or relative entropy: Given p,q*

$$KL[q\|p] = \int q(x) \log \frac{q(x)}{p(x)} dx \tag{3.20}$$

The discrete case is the same but with a sum.

i.e. it is a sort of expected difference between p and q. If they are the same p and q will always be equal, and the ration will always be equal to 1, so $KL = 0$:

$$KL[q\|p] = 0 \text{ iff } q = p \tag{3.21}$$

Also KL is a convex functional and $KL \geq 0$. The larger the KL divergence, the more the two distributions are different. The worst case scenario is when one of the two distribution is 0, so KL divergence explodes to infinity.
We can rewrite the KL divergence by splitting the logarithm in a difference.

$$KL[q\|p] = \int q(x) \log \frac{q(x)}{p(x)} dx = -\mathcal{H}[q] - \mathbb{E}_q[\log p] \tag{3.22}$$

KL divergence is important because it is essentially a measure of distance of the probability distributions. The best match of two probability distribution can be obtained by minimizing the KL divergence.
Given a fixed but known p, and a $q = q_\theta$ (depending on a set of parameters θ) that can vary, one could ask himself what is the best q_θ which could approximate p. And answer could be

$$\theta^* = \arg\min_\theta KL[q_\theta\|p] \tag{3.23}$$

This is the base of what is called **variational inference**.
If we have two random variables x,y and we compute the KL divergence between the joint distribution and the product of marginal distribution, than it will be 0 when the two variables are independent, and the more dependent the are (the more information x carries about y and vice versa), the great it will be. This is called the mutual information between x and y.

Definition 24. *Mutual information between x and y*

$$\mathcal{I}[x, y] = KL[p(x, y)\|p(x)p(y)] \tag{3.24}$$

Now let's consider a set of data points $x = x_1, ..., x_N$.

Definition 25. *Empirical distribution:*

$$p_{emp}(x) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(x = x_i) \tag{3.25}$$

i.e. give probability mass on the observation (kind of constructing an histogram). We want a nice q to approximate x. We can compute the KL divergence between the empirical distribution, and the distribution q

$$KL[p_{emp}\|q] = \overbrace{\mathbb{E}_{p_{emp}}[-\log q(x)]}^{-\frac{1}{N}\sum_i \log q(x_i)} - \mathcal{H}[p_{emp}]$$

If $q = q_\theta$

$$-\frac{1}{N} \sum_i \log q(x_i) = -\frac{1}{N} \mathcal{L}''(\theta) \tag{3.26}$$

So maximising $\mathcal{L}(\theta)$ is equivalent to minimizing the KL divergence between p_{emp} and q_θ

Chapter 4

Probabilistic Graphical Models

4.1 Introduction

Talking about generative model, we have a certain set of random variates (e.g. $x = (x_1, ..., x_n)$, $z = (z_1, ...z_m)$). Having a probabilistic model of the world means having a joint probability distribution (e.g.$p(x, z)$, known).
Typically in practical scenario, we want to reason on this joint distribution. We usually want to compute **marginal probability distribution**: $p(x) = \int p(x, z)dz$ if z are discrete $p(x) = \sum_z p(x, z)$. From a larger amount of information, we are interested in a specific subset of it.
Another typical operation we perform is **conditioning**: $p(z|x = \underline{x}) = \frac{p(\underline{x},z)}{p(\underline{x})}$, with $\underline{x}$ fixed values.
Or we can do **both** (marginalization and conditioning):

$$p(z_j|x_i = \underline{x}_i) = \frac{p(\underline{x}_i, z_j)}{p(\underline{x}_i)} = \frac{\int p(x, z)dx_{-i}dz_{-j}}{\int p(x, z)dx_{-i}dz}$$

This procedures are generalizing in a certain sense the concept of logical inference for boolean variables, to the probabilistic world of probability distributions.
What is the cost in terms of computation (considering discrete random variables, $z = z_1, ..., z_m$ with $z_i \in 0, 1$ boolean) of: $\int p(x, z)dz = p(x) = \sum_{z \in Z} p(x, z)$?
Let's remember that $z \in Z$ which has cardinality $|Z| = 2^m$, which is not an innocuous number (e.g. $m = 100 \leftarrow 2^m \approx 10^{30}$). We need then a more clever way to deal with data.
Let's introduce the concept of **factorization**, applying basic laws of probability:

$$\begin{aligned} p(x_1, .., x_n) &= p(x_2, ..., x_n|x_1)p(x_1) \\ &= p(x_3, ..., x_n|x_1, x_2) \cdot p(x_2|x_1)p(x_1) \\ &= p(x_4, ..., x_n|x_1, x_2, x_3)p(x_3|x_1, x_2)p(x_2|x_1)p(x_1) \\ &= p(x_n|x_1, .., x_{n-1})p(x_{n-1}|x_1, ...x_{n-1})...p(x_3|x_1, x_2)p(x_2|x_1)p(x_1) \end{aligned}$$

That is a factorization: representing a full joint probability distribution as the product of factors. It may happen that not all the terms in the conditioning set are relevant to

describe the conditional distribution we are interested in.
In some cases factorization can be simpler:

$$p(x_1, ..., x_n) = p(x_n|x_{n-1})p(x_{n-1}|x_{n-2})...p(x_3|x_2)p(x_2|x_1)p(x_1) \tag{4.1}$$

That for example is particularly lucky factorization.
Suppose our variables are categorical, and can assumes value in the range $x_i \in 1, ..., k$.
What is the cost of storing the values of a single probability distribution on x?
Using an array, storing the probability for each values of $p(x_i)$ (i.e. 1 with probability p_1, 2 with probability p_2, so on until $p_k = 1 - \sum_{i=1}^{k-1} p_i$, we need $k - 1 = O(k)$ doubles).
Suppose now we want to store $p(x_2|x_1)$. We are building this time a table, in which we have probability $p_{i,i}$ for each one of the categorical values. Each rows in this configuration costs $k - 1$, and in total we have k rows. So we need $k(k.1) = O(k^2)$ doubles.
So, the cost for every representation is:

- $p(x_1, .., x_n)$: cost $O(k^n)$
- $p(x_n|x_1, .., x_{n-1})p(x_{n-1}|x_1, ...x_{n-1})...p(x_3|x_1, x_2)p(x_2|x_1)p(x_1)$: each of the terms is of the order $O(k^n)$ where n is the number of variables considered for the evaluation, so again the total number of space required is of the order of $O(k^n)$
- $p(x_n|x_{n-1})p(x_{n-1}|x_{n-2})...p(x_3|x_2)p(x_2|x_1)p(x_1)$: we always have only two variates for all of the n conditional distributions, so the total number of space required is of the order $O(nk^2)$

The main kind of models that we are going to consider:

- Bayesian networks
- Markov random fields
- (Factor graph)

They are all graphs: they are composed of nodes and edges that can be directed or undirected. If all edges are directed and the graphic does not have cycle, we are talking about a Directed Acyclic Graphs used in Bayesian Network, if at the opposite the graph is undirected and can have cycle, we are talking about Random Markov fields. They are different ways to factorize a probability distribution function, based on different assumptions:Bayesian usually are more applicable, but usually inference between all of them are the same.
E.g. of a Bayesian Network:

$$p(x_1, x_2, x_3, x_4) = p(x_4|x_3)p(x_3|x_1, x_2)p(x_2)p(x_1) \tag{4.2}$$

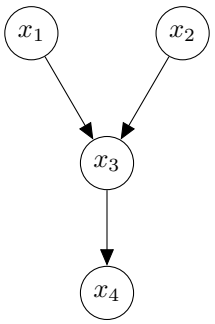


Figure 4.1: Bayesian network

4.2 Bayesian Networks

Definition 26. *pa_k: parents of x_k, i.e. $pa_k \subseteq x_n, ..., x_{k-1}$ are in the conditioning set of x_k, and $pa_k \Rightarrow p(x_k|pa_k)$ which is the factor of x_k*

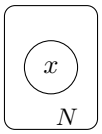
Definition 27. *Bayesian network is a DAG with:*

- *$\{x_1, ..., x_n\}$ vertices*
- *$(x_i, x_j), i < j$ and $x_i \in pa_j$ edge. We cannot have cycle.*

There are three conventions for the graphic representation of Bayesian Network:

- *Observed nodes are drawed colored.*

- *Plated node (or subgraph)*



  Where N represent the number of different instances of the node.

- *Deterministic quantities (or fixed) are solid circles (but I will use only the name of the variable).*

4.3 Sampling and Reasoning in Bayesian Networks

Definition 28. *Ancestral sampling: if the factorization is feasible, it means sampling follows the bayesian network from top to bottom (i.e. sample the ancestor, then progressively the descendants). It is possible only if we know how to sample effectively from the probability of x given its parents ($p(x|pa_x)$*

Example 3. *Let's consider a Bayesian network with the aim of predicting the grade of an exam, in which we define the node:*

- D = *difficulty*
- I = *intelligence*
- G = *grade*
- P = *passed a hard exam*

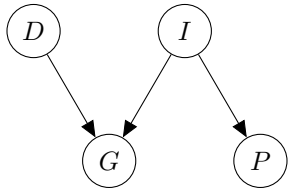


Figure 4.2: Bayesian network for grading prevision

Here are different kids of reasoning:

- *Casual reasoning: we observe the nodes on top, and want to compute the marginal probability of the variables which depends on what we observe. Information is flowing from top to bottom.*

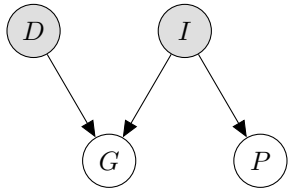


Figure 4.3: Bayesian network for grading prevision, causal reasoning.

- *Evidential reasoning: information flow from bottom to top. We observe the consequenses of something and try to infer the probability of the causes of the observation.*

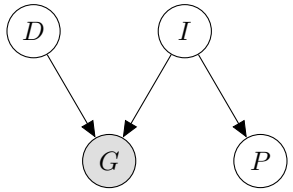


Figure 4.4: Bayesian network for grading prevision, evidential reasoning.

- *Intercausal reasoning: a sum of the two. The information goes both up to down and viceversa.*

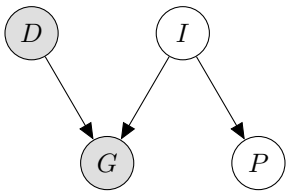


Figure 4.5: Bayesian network for grading prevision, intercausal reasoning

.

4.4 Naive Bayes

We have a certain set of features $x_1, ..., x_n$ that we observe. This observation are going to depend on class (the class of the observation we are going to have) $p(x_1, ..., x_n|c)$. This is a generative model which is **class conditional** (probability of observing certain data if it comes from a certain class. Solving a classification task, we want to compute $p(c|x_1, ..., x_n)$. Now $x_1, ..., x_n$ are no longer independent observations, but the features of our model.
Modelling this probability is potentially very difficult, so we make a simple assumption: $x_1, ..., x_n$ are independent among them given the class, which corresponds to the following network

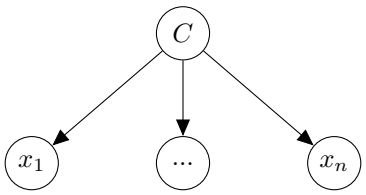


Figure 4.6: Naive Bayes structure (for all nodes of x)

Which can be written factorize as

$$p(x_1, ..., x_n|c) = \prod_{i=1}^{n} p(x_i|c) \tag{4.3}$$

This makes training the model much easier: we have observation for class C, and a certain number of observation in the train set. For each of this observation we focus only on the feature x_i, so given c, we fix x_i consider observations from the data we have on feature x_i for points of class c and fit a model parametric of $p(x_i|c, \theta)$ (i.e. it depends on a certain number of parameters), and we fit our parameter θ. Because we are fitting probabilistic models of individual variables, usually the fit is pretty easy.
We can compute now $p(c|x_1, ..., x_n) \propto p(c) \cdot p(x_1, ..., x_n|c) = p(c) \prod_i p(x_i|c)$, but not precisely (we still need some marginalization, and an estimate of $p(c)$).
Once we have fitted the model, and we have a certain observation, for example of two classes, we can compute the ration:

$$\frac{p(c = 0|x_1, ..., x_n)}{p(c = 1|x_1, ..., x_n)} \tag{4.4}$$

In the classification we choose the class which is dominating good for classification but not for estimating the probability of c given the observation.

4.5 Conditional Independence

Definition 29. *Let's consider a, b, c random variables. We say that a is conditional independent of b given c ($a \perp\!\!\!\perp b|c$) if and only if*

$$p(a|b,c) = p(a|c) \quad or \quad p(a,b|c) = p(a|c)p(b|c) \tag{4.5}$$

There are three scenario that we need to consider depending on how the arrows connect the three nodes (a, b and c).

First scenario: tail to tail

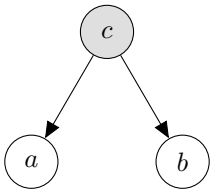


Figure 4.7: Bayesian network first scenario

In the path from a to b passing through c the arrows are tail to tail facing c. Let's compute

- $p(a,b) = \sum_c p(a|c)p(b|c)p(c) \neq p(a)p(b)$ so a and b are not conditional independent (if c is not observed).
- $p(a,b|c) = \frac{p(a,b,c)}{p(c)} = \frac{p(a|c)p(b|c)p(c)}{p(c)} = p(a|c)p(b|c)$ so a and b are conditional independent (if c is observed).

c is the root cause of a and b, so if we do not know c, there may be some relationship between them. Once we observe c, all the dependency is contained in c.

Second scenario: head to tail

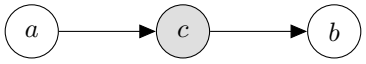


Figure 4.8: Bayesian network second scenario

So as before

- $p(a,b) = \sum_c p(b|c)p(c|a)p(a) = p(b|a)p(a) \neq p(a)p(b)$ so a and b are not conditional independent (if c is not observed).
- $p(a,b|c) = \frac{p(a,b,c)}{p(c)} = \frac{p(b|c)p(c|a)p(a)}{p(c)} = p(b|c)p(a|c)$ so a and b are conditional independent (if c is observed).

a causes c which causes b. Once we observe something in the middle, b is not going to be conditional independent on anything before c, because c already bring all the information necessary for b.

Third scenario: head to head

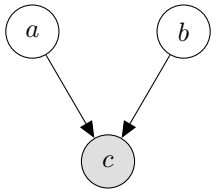


Figure 4.9: Bayesian network third scenario

So as before

- $p(a,b) = \sum_c p(a)p(b)p(c|a,b) = p(a)p(b)$ so a and b are conditional independent (if c is not observed).
- $p(a,b|c) = \frac{p(a,b,c)}{p(c)} = \frac{p(a)p(b)p(c|a,b)}{p(c)} \neq p(b|c)p(a|c)$ so a and b are not conditional independent (if c is observed). Also a and b are not conditional independent given d for d a descendent of c.

a and b are independent, but if we observe something downstream, that it is consequence of both a and b how may introduce some information which may be propagated back.

Exploring graph structure of the Bayesian network

Let's consider three node a, b and c. A path from a to b is blocked by c if and only if

- c is observed and the path is head to tail or tail to tail in c.
- c is not observed nor any descendent of c and the path is head to head in c.

Now let A, B and C subsets of nodes disjointed, if all path from a node in A to a node in B are blocked by a node in C than

$$A \perp\!\!\!\perp B|C \tag{4.6}$$

Definition 30. *Markov blanket: let's consider a node x_i, we will condition all the rest x_{-i}, which nodes remain in the conditional set (this nodes are the markov blanket)?*

$$p(x_i|x_{-i}) = \frac{p(x_1,...,x_i,...,x_n)}{p(x_1,...,x_{i-1},x_{i+1},...,n_n)} = \frac{\prod_i p(x_i|pa_j)}{\sum_{x_i}\prod_j p(x_j|pa_j)} \tag{4.7}$$

Summing over x_i (denominator), which of the terms will depend on x_i? Either x_i is x_j, or x_i belongs to the parents set $x_i \in pa_j$.
So the Markov blanket of x_i is made of:

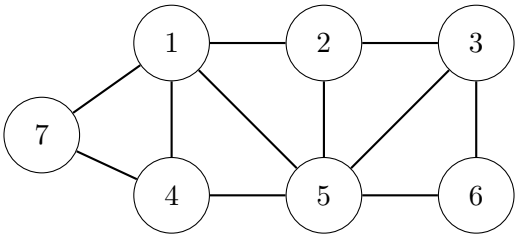
- pa_j

- *Children*
- *Co-parents*

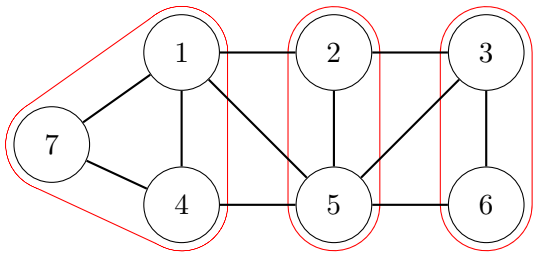
The Markov blanket "covers" x_i, and make it independent.

4.6 Markov Random Fields

Markov Random Fields are undirected graphs (i.e. graph in which edges has no arrow)



Where nodes are variables, and edges represent a sort of dependency (there is no direct correspondence between graph structure and the factorization implied). Let's see how conditional independence is represented in a markov random fields.



Let's considered three subset of nodes: $A = \{1, 4, 7\}$, $B = \{3, 4\}$ and $C = \{2, 5\}$. We can say that: $A \perp\!\!\!\perp B|C \Leftrightarrow$ all paths from a node $a \in A$ to any node $b \in B$ pass from C (are blocked by a node in C).
Markov blanket of x_i in this context is the set of neighbours of x_i.
When two nodes are conditionally independent given the rest of the nodes ($x_i \perp\!\!\!\perp x_j|x_{-\{i,j\}}$)?
When there is no path from x_i to x_j passing from any other node.
So $x_i \perp\!\!\!\perp x_j|x_{-\{i,j\}}$ if and only if there is no edge between them. From this we can deduce that

$$p(x_i, x_j|x_{-\{i,j\}}) = p(x_i|x_{-\{i,j\}})p(x_j|x_{-\{i,j\}}) \tag{4.8}$$

i.e. joint probability factorize as the product of marginal probability conditioning on all the other nodes.
But that means that x_i and x_j must belong to different factors. Nodes which are not connected by edges should belong to different factors.
From this derives that factors in a random markov field is equivalent to a maximal cliques in the graph.

Definition 31. *Cliques: subgraph fully connected.*
Maximal Cliques: cliques that cannot be extended.

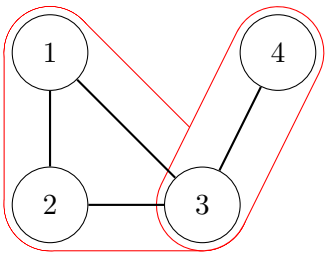


Figure 4.10: Markov Random Fields with two cliques overlapping on node 5

The reason why factors corresponds to cliques is that if we are connected on an edge, even if they are conditioning on all the rest, we remain independent.

Definition 32. *Defined* $C = \{$*set of maximal cliques*$\}$ *and* $x_c = \{x | x \in C \in \mathcal{C}\}$ *(where* $\mathcal{C}$ *is the set of maximal cliques). Given* $x = x_1, .., x_n$
Factorization in a MRF:

$$p(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(x_c) \tag{4.9}$$

i.e. the product over the maximal cliques of some function ψ_c *evaluated on the subset of the variables of* x *that belong the the clique; over a constant* Z *which guarantees the normalization into a probability density.*

$\psi_c(x_c) \geq 0$ *and have a finite integral* $\int \psi_c(x_c) dx_c < \infty$, ψ_c *is a* ***potential function****.*

$Z = \int \prod_c \psi_c(x_c) dx$ *is called* ***partition function*** *(but it is actually a constant).*

Z usually is hard to compute. If variables are discrete with k states $x_i \in \{1, ..., k\}$, then computing Z means:

$$Z = \sum_x \prod_c \psi_c(x_c) \tag{4.10}$$

with the sum on a subset of k^n states. Practically impossible. And in the case of the integral it is the same. But

- If we want to compute conditionals, then Z cancels out.
- If we want to compute marginal of few variables, we can work with potentials and normalize at the end (i.e. let's imagine that when we are working with potentials we may compute this one dimensional marginal. This one dimensional is an unnormalized marginal, so we have potentials, and numbers which does not sum up to 1, but we have to sum k numbers, which is easy).

Potentials can be defined in any way, they just have to be positive. A way to guarantee that something is positive, is to model it as the exponential of a real number. So by convention we usually model the potential function as

$$\psi_c(x_c) = exp(-E(x_c)) \tag{4.11}$$

which is known as the **Boltzmann distribution**, where $E(x_c)$ is the **energy**. States of low energy corresponds to states of high potential. It is convenient to work with Boltzmann models per the easiness of factorization

$$\begin{aligned}\psi_{c_1}(x_{c_1})\psi_{c_2}(x_{c_2}) &= exp(-E_1(x_{c_1}))exp(-E_2(x_{c_2})) \\ &= exp(-E_1(x_{c_1}) - E_2(x_{c_2}))\end{aligned}$$

Chapter 5

Inference in PGM

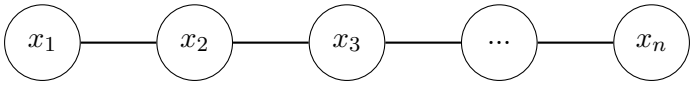
5.1 Introduction

Our task of inference is that whenever we have a joint distribution $p(x_1, ..., x_n) = p(x)$ which is a full joint distribution, the typical condition is that we have two subsets $y \subseteq x$ and $z \subseteq x$, with $y \cap z = \emptyset$, and usually $y \cup z \subsetneq x$.
We are observing y, so $y = \overline{y}$, what we want to compute is the conditional probability of z given $\overline{y}$

$$\begin{aligned} p(z|y = \overline{y}) &= \sum_{x'} p(z, x'|y = \overline{y}) \\ &= \int p(z, x'|y = \overline{y}) dx' \end{aligned}$$

It is a difficult task, because if $|x'| = m$ (x' has length m), its summation can be something of the order of $|x'| = m \in O(K^m)$. That can be challenging, if not numerically impossible. The problem is then how to do it efficiently.

Example 4. *Let's think about a markow random field (the same argument holds for Bayesian networks). We have variables which are arranged into a chain.*



We can write them down as a factorization

$$p(x) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)...\psi_{n-1,n}(x_{n-1}, x_n) \tag{5.1}$$

Now lets say, given $1 < k < n$, we want to compute $p(x_k)$ (a.k.a. marginalise our joint distribution over $x_1, ...x_{k-1}, x_{k+1}, ..., x_n$)

$$p(x_k) = \sum_{x_1,...x_{k-1},x_{k+1},...,x_n} p(x_1, ..., x_n) \tag{5.2}$$

which, in case of discrete sets, has cardinality of $O(K^{n-1})$ (pretty big!).
However, lets plug this sum in the form of the Random Markow Field

$$p(x_k) = \sum_{x_1,...x_{k-1},x_{k+1},...,x_n} p(x_1,...,x_n) = \sum_{x\setminus k} \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)...\psi_{n-1,n}(x_{n-1},x_n) \tag{5.3}$$

Now, lets consider the last term of the last equation. We are summing over all possible states of x_1. What if we first sum x_1 out, then deal with the rest. Doing so, we can recognise that x_1 only appear in the first term $\psi_{1,2}(x_1,x_2)$, and not in all the others. In this way x_1 disappears, and we have a problem which has 1 variable less. If we do it iteratively, we do $n-k$ times k sums, so we reduced the complexity to linear.

$$\begin{aligned}=&\frac{1}{Z}\sum_{x_{k-1}}\psi_{k-1,k}(x_{k-1},x_k)...\Big[\sum_{x_2}\psi_{2,3}(x_2,x_3)\Big[\sum_{x_1}\psi_{1,2}(x_1,x_2)\Big]\Big]\\ &\times\sum_{x_{k+1}}\psi_{k,k+1}(x_{k-1},x_k)...\Big[\sum_{x_{n-1}}\psi_{n-2,n-1}(x_{n-2},x_{n-1})\Big[\sum_{x_n}\psi_{n-1,n}(x_{n-1},x_n)\Big]\Big]\end{aligned}$$

We need a better representation, we use a dynamic programming algorithm, in the family of **massage passing algorithms**.
Lets define two messages:

- Forward messages

$$\mu_\alpha(x_k) = \sum_{x_{k-1}} \psi_{k-1,k}(x_{k-1},x_k)\mu_\alpha(x_{k-1}) \tag{5.4}$$

  with base case

$$\mu_\alpha(x_1) = 1 \tag{5.5}$$

- Backward messages

$$\mu_\beta(x_k) = \sum_{x_{k+1}} \psi_{k,k+1}(x_k,x_{k+1})\mu_\beta(x_{k+1}) \tag{5.6}$$

  with base case

$$\mu_\beta(x_n) = 1 \tag{5.7}$$

After the closing operation on the conditional distribution function (backward massages travelling from the end to k, forward messages from the start to k), we end up with

$$\begin{aligned}p(x_k) &\propto \mu_\beta(x_k)\mu_\alpha(x_k)\\ &= \frac{1}{Z_k}\mu_\beta(x_k)\mu_\alpha(x_k) \text{ with } Z_k = \sum_{x_k}\mu_\alpha(x_k)\mu_\beta(x_k)\end{aligned}$$

5.2 Factor Graphs

Definition 33. *Factor Graph: bipartite graph, i.e. a graph which has nodes divided in two different classes:*

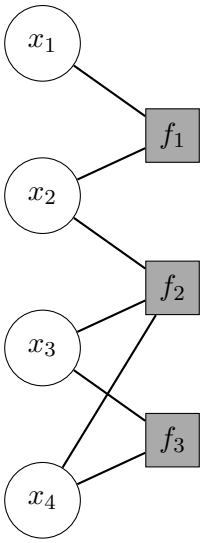
- *Variable nodes (**empty circle**)*
- *Factor nodes (**filled square**)*

The edges connect variables to factors, but not variables between themselves. We represent the factorize description of our joint probability density by having clearly worked out the factors, and knowing which variables influence a specific factor.

Lets assume we have a joint probability distribution that factorizes as such

$$p(x) = f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_3, x_4) \tag{5.8}$$

In the canonical way of representation, we separate node in two areas (i.e. left and right), in one area we have the variable nodes, in the other the factor nodes.



Bayesian network to factor graph

The following bayesian network

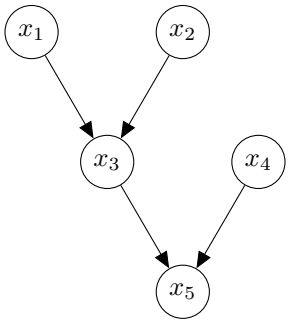
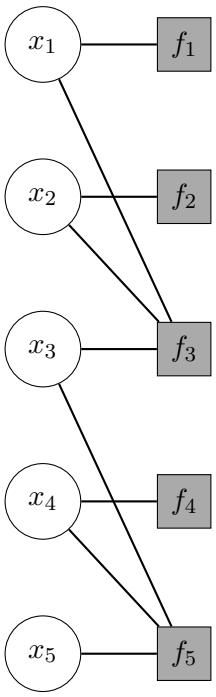


Figure 5.1: Bayesian network

corresponds to the factorization of the joint probability distribution as

$$p(x) = \underbrace{p(x_1)}_{f_1} \underbrace{p(x_2)}_{f_2} \underbrace{p(x_3|x_1, x_2)}_{f_3} \underbrace{p(x_4)}_{f_4} \underbrace{p(x_5|x_3, x_4)}_{f_5} \tag{5.9}$$



So

$$\forall p(x_j|p_{\alpha j}) \;\Rightarrow\; f_j \to x_j \cup p_{\alpha j} \tag{5.10}$$

For each term in the factorization of the bayesian network, we have a term f_j which is connected as represented.

Markow random field to factor graph

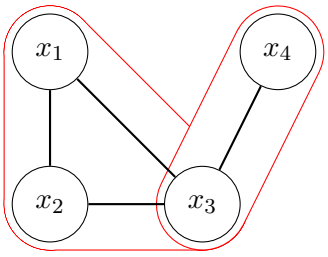
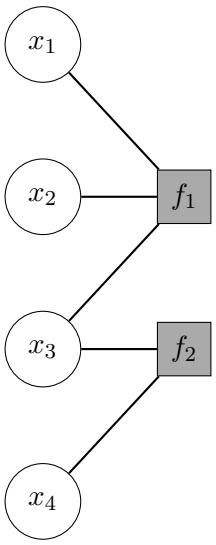


Figure 5.2: Markov Random Fields

Which is

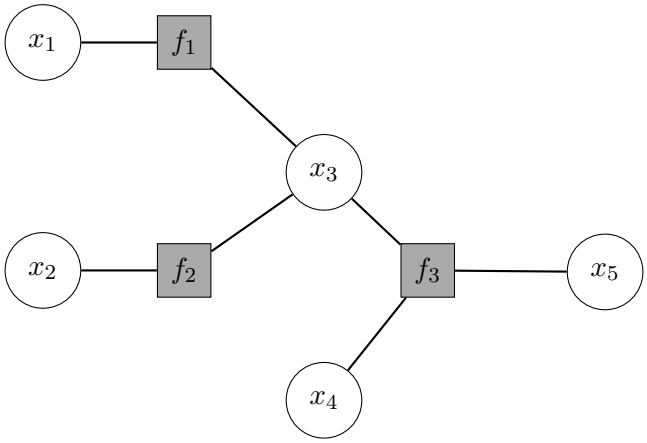
$$p(x) = \frac{1}{Z} \underbrace{\psi_1(x_1, x_2, x_3)}_{f_1} \underbrace{\psi_2(x_3, x_4)}_{f_2} \tag{5.11}$$



$$\forall c \in \mathcal{C} \;\Rightarrow\; f_c(x_c) = \psi_c(x_c) \tag{5.12}$$

5.3 Sum-Product algorithm

Lets consider the following factor graph



$$p(x) = f_1(x_1, x_3) f_2(x_2, x_3) f_3(x_3, x_4, x_5) \tag{5.13}$$

We want to illustrate the belief propagation algorithm. It is applicable to factor graphs which are trees (no loops). We can then identify a node which is the root, and nodes which are leaves (we "read" differently the graph depending on which node we pick as root). **The root is the node for which we want to compute the marginal distribution**. We work in two passages

- Forward pass: messages travel from the leaves to the root.
- Backward pass: messages travel from the root to the leaves.

After a forward and backward pass, we have the information to compute the marginal distribution in each node.

Forward pass

(Backward pass is the same, just in the opposite direction).
We take x_5 as the root. If we want to consider the marginal probability on x_5

$$\begin{aligned} p(x_5) &= \sum_{x_1,x_2,x_3,x_4} p(x_1,x_2,x_3,x_4,x_5) \\ &= \underbrace{\sum_{x_3,x_4} f_3(x_3,x_4,x_5) \overbrace{\Big[\sum_{x_1} f_1(x_1,x_3)\Big]}^{\mu_{f_1\to x_3}} \overbrace{\Big[\sum_{x_2} f_2(x_2,x_3)\Big]}^{\mu_{f_2\to x_3}}}_{\mu_{f_3\to x_5}(x_5)} \overbrace{\qquad\qquad}^{\mu_{x_3\to f_3}} \end{aligned}$$

The messages going from factors to variables include marginalisation and summation. Messages going from variables to factors collect all the incoming messages form the other factors and multiply them, sending them to the other factor.

Generalisation

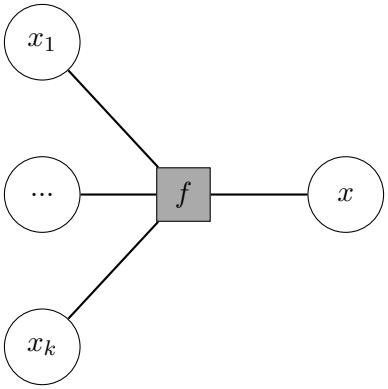
Some notations

Definition 34. *Given a variable node x or a factor node f, we call*

- $\mathcal{N}(x) = \{f_1, ..., f_k | f_i - x\}$, *i.e. the set of neighboring factors of a variable x: the list of factors which has an edge with x*
- $\mathcal{N}(f) = \{x_1, ..., x_k | x_i - f\}$, *i.e. the set of neighboring nodes of a factor f: the list of variables which has an edge with f.*

The notation $\mathcal{N}(f) \setminus x$ means: all the neighbours of f excluding x.

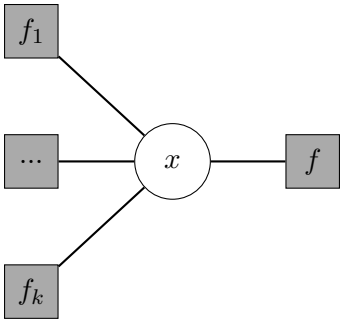
Lets now consider the following scenario



We are interested in the message from f to x

$$\mu_{f\to x}(x) = \sum_{x_1,...,x_k\in\mathcal{N}(f)\setminus x} f(x, x_1, ..., x_k) \cdot \prod_{x_i\in\mathcal{N}(f)\setminus x} \mu_{x_i\to f}(x_i) \tag{5.14}$$

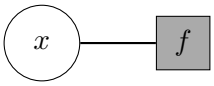
Alternatively, there is the situation



We are interested in the message from x to f

$$\mu_{x\to f}(x) = \prod_{f_i\in\mathcal{N}(x)\setminus f} \mu_{f_i\to x}(x) \tag{5.15}$$

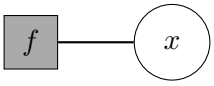
Then we remain only wih cases



For which

$$\mu_{x\to f}(x) = 1 \tag{5.16}$$

and



For which

$$\mu_{f\to x}(x) = f(x) \tag{5.17}$$

Which are the base cases.
After forward and backward passages, we want to compute:

- Marginal of a variable node x

$$p(x) = \prod_{f\in\mathcal{N}(x)} \mu_{f\to x}(x) \tag{5.18}$$

- Marginal of a factor node $f(\overline{x})$

$$p(\overline{x}) = f(\overline{x}) \cdot \prod_{x\in\mathcal{N}(f)} \mu_{x\to f}(x) \tag{5.19}$$

Now, how we condition on $y = \hat{y}$ with $y \subseteq \{x_1, ..., x_n\}$?

$$p(x|y = \hat{y}) \tag{5.20}$$

We compute the joint probability distribution fixing the variables y to $\hat{y}$ (this operation is called clamping $y \mapsto \hat{y}$). Once we have done that, we compute the joint probability distribution with a message passing algorithm as before, with the only difference that whenever we have a variable in y, instead of making a summation on this variable, or computing its value for all possible different states, we just consider the state corresponding to state $\hat{y}$.

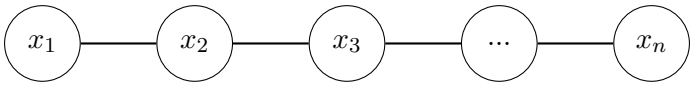
$$p(x|y = \hat{y}) = \sum_{x_1,...,x_k} p(x, x_1, ..., x_k, y = \hat{y})$$

Which is the sum product with y_j fixed to $\hat{y}_j$ for all y_j in y.

$$p(x|y = \hat{y}) = \frac{p(x, y = \hat{y})}{p(\hat{y})}$$
$$\text{where } \; p(\hat{y}) = \sum_x p(x, y = \hat{y})$$

5.4 Max-Plus algorithm

Max-plus algorithm solves the problem of finding the tuple in the states-space that maximises a probability.
Given a $p(x)$ we want to find $x^M = \arg\max_x p(x)$ (roughly we are going to find the maximum a posteriori).
Lets look at what happens in a chain structure



We know by definition that

$$p(x) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)...\psi_{n-1,n}(x_{n-1}, x_n) \tag{5.21}$$

So the maximum is

$$\begin{aligned}\max_x p(x) &= \max_{x_1} ... \max_{x_n} p(x) \\ &= \frac{1}{Z} \max_{x_1} \max_{x_2} \psi_{1,2}(x_1, x_2) ... \max_{x_{n-1}} \psi_{n-2,n-1}(x_{n-2}, x_{n-1}) \max_{x_n} \psi_{n-1,n}(x_{n-1}, x_n)\end{aligned}$$

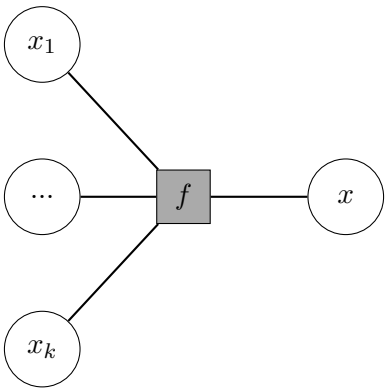
We can factorize as such because of the distributive properties of max

$$\begin{aligned}\text{if } a > 0 \qquad & \max(ab, ac) = a \max(b, c) \\ & \max(a + b, a + c) = a + \max(b, c)\end{aligned}$$

We maximise $\log p(x)$ instead of $p(x)$.

$$\log p(x) = \sum_i \log \psi_{i,i+1}(x_i, x_{i+1}) + \log(Z) \tag{5.22}$$

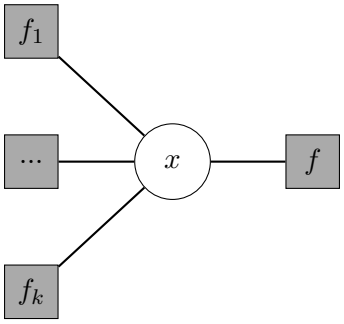
Essentially we have the same algorithm as the sum-product algorithm, where max replaces sums, and plus replaces the product. Lets now consider the following scenario



We are interested in the message from f to x

$$\hat{\mu}_{f\to x}(x) = \max_{x_1,...,x_k\in\mathcal{N}(f)\setminus x}\Big[\log f(x,x_1,...,x_k) + \sum_{x_i\in\mathcal{N}(f)\setminus x}\mu_{x_i\to f}(x_i)\Big] \tag{5.23}$$

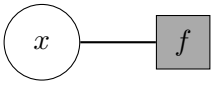
Alternatively, there is the situation



We are interested in the message from x to f

$$\hat{\mu}_{x\to f}(x) = \sum_{f_i\in\mathcal{N}(x)\setminus f}\mu_{f_i\to x}(x) \tag{5.24}$$

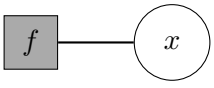
Then we remain only with cases



For which

$$\hat{\mu}_{x\to f}(x) = 0 \tag{5.25}$$

and



For which

$$\hat{\mu}_{f\to x}(x) = \log f(x) \tag{5.26}$$

Which are the base cases.
The algorithm consist in: fixing the root, doing the forward pass, where we have

$$\max_{x_{root}} \Big[\sum_{f \in \mathcal{N}(x_{root})} \mu_{f \to x_{root}}(x_{root}) \Big] = p_{\max} \tag{5.27}$$

also, we are going to find

$$x_{root}^{\max} = \arg\max_{x_{root}} \Big[\sum_{f \in \mathcal{N}(x_{root})} \mu_{f \to x_{root}}(x_{root}) \Big] \tag{5.28}$$

We than go backward with the maximisation until we reach the leaves. To do so, we need to store additional message

$$\phi_{f \to x}(x) = \arg \max_{x_1,...,x_k \in \mathcal{N}(f) \setminus x} \Big[\log f(x, x_1, ..., x_k) + \sum_{x_i \in \mathcal{N}(f) \setminus x} \mu_{x_1 \to f}(x_i) \Big] \tag{5.29}$$

So, to recap, we start from the root, we compute the $x_{root}^{\max}$, then we take all the factors which are sending a message to the root, and evaluate the function of the last equation of $x_{root}^{\max}$,

$$\phi_{f \to x_{root}}(x_{root}^{\max}) = \begin{cases} x_1 \to x_1^{\max} \\ ... \\ x_k \to x_k^{\max} \end{cases} \tag{5.30}$$

Now the max values are fixed. Now for any of those values, we will consider the massage for an incoming factor the the node of that value

$$\phi_{f \to x_i}(x_i^{\max}) \tag{5.31}$$

And we go on fixing all the variables on which the factor node of interest depends on, and we continue doing so backward from the root to the leaves, and at the end we have the tuple. In the contest of Hidden Markow Field, doing such process on a chain is called **Viterbi algorithm**, i.e. an instance of the max-plus algorithm, with the function ϕ, on a chain, in which every variable has the same number of states (corresponds to the different states we generate in a sequence; in this case, a markow process)

5.5 Inference for general PMGs

What happens if the factor graph has loops?
Junction tree algorithm: works on any factor graph by performing a transformation, i.e. it creates a bigger proper tree (over the cliques) on which we can apply the algorithm we want (inference is done over the cliques). It is exponential on the size of the largest clique (may be computationally heavy, so it is not really used in practice).
Alternatives are usually approximate: **montecarlo sampling**, i.e. sampling from the joint distribution, extracting information about the marginal, some expectations etc. It is very general and it is the most used.
Another method is **variational inference**: it approximate the posterior with a simpler form, belonging to a pre-specified class (the one that minimized the Kullback Leibler

divergence).
Another is the **Belief-loopy propagation**: run the sum-product algorithm, but instead of doing forward and backward, we iterate doing only forward with the information obtained in the previous step (until we reach a steady point, which does not guarantees to be the minimum).

Chapter 6

Markov Chain Monte Carlo

6.1 Introduction

As stated in the last paragraph of the previous chapter, Markov Chain Monte Carlo (MCMC) is a sampling method that we use in the cases in which the factor graph has loops.
The general problem is that we want to sample from $p(x|y = \hat{y})$, but that is difficult, because even the definition of such quantity, in the case of a loop, is not clear.
We can notice that our target distribution is definable up to a multiplicative constant, as something we can actually evaluate (i.e. fix variables x and y, than compute the probability density of a random tuple of x and a fixed y, we still have to divide by the probability of $\hat{y}$, this is the partition function, which we do not have; so we just have an unnormalized probability distribution function.)

$$P(x|y = \hat{y}) = \frac{1}{Z} p(x, y = \hat{y})$$
$$p(x) = \frac{1}{Z}\widetilde{p}(x) \qquad \text{with } \widetilde{p}(x) \geq 0 \text{ but not normalised}$$

Another typical scenario in which this happens is when our $p(x|y)$ is defined by the Bayes theorem

$$p(x|\hat{y}) = \frac{p(\hat{y}|x)p(x)}{p(\hat{y})} \tag{6.1}$$

in this case the probability $p(\hat{y})$ is difficult to compute.
Our sampling problem is the problem of generating $x_1, ..., x_n$ from $p(x)$, knowing only its unnormalized version $\widetilde{p}(x)$.
A typical application is to find the expectation of a function in x

$$\mathbb{E}_x\Big[f(x)\Big] = \int f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{6.2}$$

We want to generate $x_1, ..., x_n$ as independent sample.
We can:

- Sampling from $q(x) \approx p(x)$ (as close as possible) and correct.
- MCMC: allows us to generate sample from unnormalized posterior distribution.

6.2 Rejection Sampling

We have a distribution $p(x)$ which is complicate, and we do not know how to handle. We than have a simpler distribution $g(x)$ that we call the **proposal distribution**, and we assume

$$\exists M > 0 : Mg(x) \geq p(x), \forall x \in \mathcal{X} \tag{6.3}$$

So that $g(x)$ dominates $p(x)$ point wise. This imply that

$$\frac{p(x)}{Mg(x)} \leq 1 \tag{6.4}$$

What we do is:

1. Sampling $\hat{x}$ from $g(x)$.
2. Accept $\hat{x}$ with probability $\alpha = \frac{p(\hat{x})}{Mg(\hat{x})}$ (so only in the acceptance region below $p(x)$).
3. If rejected, we repeat until acceptance.

The expected number of samples from g per p sample is M. This method can be inefficient in high dimensions.

6.3 Importance Sampling

In this case, we apply a clever strategy to directly compute the expectation of our function f

$$\begin{aligned}\mathbb{E}_p[f] &= \int f(x)\frac{1}{Z}\widetilde{p}(x)dx \\ &= \frac{\int f(x)\widetilde{p}(x)dx}{\int \widetilde{p}(x)dx}\end{aligned}$$

We consider a proposal distribution g which we can sample from. We can multiply the two integrals (numerator and denominator) by $\frac{g(x)}{g(x)} = 1$. We can then rewrite the full integral as

$$\begin{aligned}\mathbb{E}_p[f] &= \int f(x)p(x)dx = \int f(x)\frac{p(x)}{g(x)}g(x)dx = \mathbb{E}_g[f\frac{p}{q}] \\ &= \frac{\int f(x)\frac{\widetilde{p}(x)}{g(x)}g(x)dx}{\int \frac{\widetilde{p}(x)}{g(x)}g(x)dx}\end{aligned}$$

We than sample $x_1, ..., x_N$ from $g(x)$ and get our estimate of $p(x)$ as

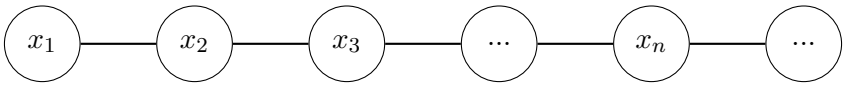
$$\frac{\frac{1}{N}\sum_{i=1}^{N} f(x_i)w(x_i)}{\frac{1}{N}\sum_{i=1}^{N} w(x_i)} \qquad \text{where } w(x_i) \coloneqq \frac{\widetilde{p}(x_i)}{g(x_i)}$$

We call $w(x_i)$ the importance weight.
If we have large importance weights, we have large variance in the estimate, so the algorithm is not going to work very well.
If we have very small importance weight, than we are going to have a stable and effective estimate: if $\frac{f\tilde{p}}{g}$ is approximately constant, then estimates can be very good.
Also, we can notice that

$$\frac{1}{N}\sum w(x_i) \approx Z \tag{6.5}$$

6.4 Markov Chains

A Markov chain is a stochastic process $(x_t)_{t\geq 0}$ with $t \in \mathbb{N}$, we are taking a sequence of variables $x_0, x_1, ...$ each of those variables belong to a certain space $\mathcal{X}$, $x_i \in \mathcal{X}$, which can be discrete or continuous. Our aim is to introduce a mechanism which governs the evolution of our state through the sequence. Markov chains satisfy the memoryless property, or Markov property, which can be shown graphically



There is a dependency that flow through the chain, following the property

$$p(x_n|x_{n-1}, ..., x_0) = p(x_n|x_{n-1}) \tag{6.6}$$

as well as the time homogeneity

$$p(x_n|x_{n-1}) = p(x_1|x_0) \qquad \forall n \geq 1 \tag{6.7}$$

Specifically,we are interested in markov chain which satisfies the property of ergodicity (or reducibility):

$$\forall x, y \in \mathcal{X}, \exists t > 0 : \; p(x_t = y|x_0 = x) > 0 \tag{6.8}$$

This means that we can reach any point in the states-space with non-0 probability, assuming that we do it in f moves.
We call the $p(y|x)$ one step transition probability, the **one step transition kernel**. Considering a discrete space, the transition kernel would actually be a transition matrix. In the case of continuous space, the kernel is a conditional probability density.
A **stationary distribution** of an ergodic markov chain (assuming their existence) is

$$\pi(y) = \int p(y|x)\pi(x)dx \tag{6.9}$$

So, $\pi(x)$ is invariant for the markov chain dynamics, and we can prove that (under ergodicity and some other restrains) if we can compute $p_n(x) = p(x_n \neq x_0) \xrightarrow[n\to\infty]{} \pi(x)$, and π is unique (it is the limiting distribution).

6.5 Detailed Balance

Enforcing the conditions of a stationary distribution is not always easy. The simple condition of reversibility guarantees that a distribution is stationary.

Definition 35. *A M.C. is reversible (satisfy the balance condition) if and only if* $\exists \pi$ *distribution so that*

$$p(x|y)\pi(y) = p(y|x)\pi(x) \tag{6.10}$$

than π *is stationary, because we can compute the condition per stationarity*

$$\int p(y|x)\pi(x)dx = \int p(x|y)\pi(y)dx = \pi(y)\int p(x|y)dx = \pi(y) \tag{6.11}$$

Reversibility is way easier to check than stationary, but still guarantees that we are going to work with a stationary distribution.

6.6 Metropolis-Hastings MCMC

We build a markov chain, such that transition kernel is easy to sample from, that is guaranteed to satisfy the detailed balance condition for the target distribution that we want to sample from.
We start from

$$p(x) = \frac{1}{Z}\widetilde{p}(x) \tag{6.12}$$

We fix $q(y|x)$ transition kernel of markov chain, easy to sample from, and makes the markov chain ergodic. A typical choice of the transition kernel, in particular for continuous space, is the gaussian distribution, possibly isotropic, with variance chosen in a clever way to explore the whole state-space.
However, this is not the final transition kernel we are going to use: we need to include in some way $p(x)$, otherwise it will be hard to have $p(x)$ as a stationary distribution.
The actual transition kernel from x to another y is

1. Sample y from $q(y|x)$
2. Set $x_{t+1} = \begin{cases} y & \text{with probability } \alpha(y|x) = \min\left\{1, \frac{\widetilde{p}(y)\cdot q(x|y)}{\widetilde{p}(x)\cdot q(y|x)}\right\} \\ x \end{cases}$

The use of the expression

$$\alpha(y|x) = \min\left\{1, \frac{\widetilde{p}(y)\cdot q(x|y)}{\widetilde{p}(x)\cdot q(y|x)}\right\} \tag{6.13}$$

is called **Metropolis-Hastings acceptance criterion** which, if $q(x|y) = q(y|x)$ we have a symmetric proposed distribution, becomes the Metropolis criterion.
Lets call the full transition kernel (which incorporate the two steps of the markov chain) $p(y|x)$.

If $y \neq x$

$$\begin{aligned} p(y|x)p(x) &= \alpha(y|x)q(y|x)p(x) \\ &= \min\left\{1, \frac{p(y)}{p(x)}\frac{q(x|y)}{q(y|x)}\right\} \cdot q(y|x)p(x) \\ &= \min\left\{q(y|x)p(x), q(x|y)p(y)\right\} \\ &= \min\left\{\frac{q(y|x)}{q(x|y)}\frac{p(x)}{p(y)}, 1\right\} \cdot q(x|y)p(y) \\ &= \alpha(x|y)q(x|y)p(y) \\ &= p(x|y)p(y) \end{aligned}$$

So, the detailed balanced condition holds.
Picking a bad q, the algorithm may still work, but may take a long number of steps before reaching the proper stationary distribution. As the dimensionality increase, it also increases the possibility of rejecting a step.

6.7 Gibbs sampling

First of all, we have a distribution $p(x) = p(x_1, ..., x_n)$ that we want to sample from.
We make an assumption: we know how to sample from the 1D conditionals $p(x_i|x_{\setminus i})$ where $x_{\setminus i} = (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$.
The process work as

1. Pick $k \in \{1, ..., n\}$.
2. Set $x_j^{(t+1)} = x_j^{(t)}$ for $j \neq k$.
3. Sample $x_k^{(t+1)} \sim p(x_k|x_{\setminus k}^{(t)})$

How we choose k?

- Classic way: Round-Robin strategy, i.e. fix and order, and loop in this order.
- Choose uniformly at random.

If we choose k as

$$q_k(y|x) = \begin{cases} p(y_k|x_{\setminus k}) & \text{when } \; y_{\setminus k} = x_{\setminus k} \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

That is the Metropolis-Hastings algorithm, with where the acceptance is always 1

$$\alpha_k(y|x) = 1 \tag{6.15}$$

Lets compute the ratio of the acceptance rate in the Metropolis-Hasting algorithm

$$\begin{aligned}\alpha_k &= \frac{p(y)q(x|y)}{p(x)q(y|x)} \\ &= \frac{p(y_k|y_{\setminus k})p(y_{\setminus k})}{p(x_k|x_{\setminus k})p(x_{\setminus k})} \cdot \frac{p(x_k|y_{\setminus k})}{p(y_k|x_{\setminus k})} \\ &\text{if the property } x_{\setminus k} = y_{\setminus k} \text{ holds, then everything cancel out} \\ &= 1\end{aligned}$$

Gibbs sampling can be generalized by

- Sampling blocks $x_j, ..., x_k \subseteq x$
- If $p(x_i|x_{\setminus i})$(we are not able to sample from the conditional) is not known, than
  - Rejection sampling to get the next point.
  - Metropolis within Gibbs.

There are some problems with Gibbs sampling:

- Ergodicity is not totally obvious.
- If variables are strongly correlated, then convergence is slow.

6.8 Inference in Bayesian Networks

We have a vector of variables x, y in a probabilistic graphical model p_{GM}, where $y = \hat{y}$ is observed, and x is not. We want to make inference on x (e.g. computing the marginal probability over certain variable x_y, conditioned on the observations: $p(x_y|y = \hat{y})$).
We want to sample from $p(x|y = \hat{y})$.
We know $\widetilde{p}(x) = p(x, y = \hat{y})$, but not $Z = p(y = \hat{y})$. There are different strategies

- Rejection sampling: sample from the full distribution $p(x, y)$ (ancestral sampling, really fast), rejecting when $y \neq \hat{y}$. It is very inefficient when working with a large set of variables.
- MCMC: sampling from the unnormalized $p(x, y = \hat{y})$, using it as a target distribution, building a proposal distribution reasonable enough (depending on the case). It is the state of the art, but not always the most efficient.
- If we can efficiently evaluate the one dimensional conditional distribution, conditional on all the rest $p(x_i|x_{\setminus x}, y) = p(x_i|mb_i)$ where mb is the markov blanket, then we use Gibbs sampler. All variables now, are going to be observed and fixed. We notice that, when using Gibbs sampler, all variables are going to be observed and fixed (which is relatively simple, depending on the cardinality of x).

6.9 Convergence diagnostics

How can we check if our chain has reached the stationary distribution?
The output we are interested in is a scalar function of the states $\psi : \mathcal{X} \to \mathbb{R}, \psi(x)$, we assume that ψ have values in $\mathbb{R}$, and transform it otherwise.
In our notation our sample is $x_1, ..., x_n$, and $\psi_j = \psi(x_j)$ (just notation), what we want is the expectation value $\overline{\psi} = \frac{1}{N}\sum_j \psi_j$ (i.e. estimation of $\mathbb{E}[\psi] = \int \psi(x)p(x)dx$). We need a Markov chain to be stationary (reach a stationary distribution), because we keep samples only after we have reached stationarity. What we do is

1. Sample $\frac{m}{2} \geq 1$ trajectories from over dispersed initial points.
2. Sample for $4n$ steps.
3. Throw away the first half: now we have only $2n$ points left (burning/warm up phase).
4. We split the remaining trajectories in two:
    - m sequences of n length each.

Lets introduce some other notation:

- x_{ij} with $1 \leq j \leq m$ (as it may vary across the sequences) , and $1 \leq i \leq n$ (as it vary across the sample).
- $\psi(x_{ij}) = \psi_{i,j}$
- $\overline{\psi_{ij}} = \frac{1}{m}\sum_{i=1}^{m} \psi_{ij}$,i.e. the expectation of over our sample of the samples corresponding to sequence j
- $\overline{\psi} = \frac{1}{m}\sum_{j=1}^{m} \psi_j$, i.e. it merges all those expectation together (the estimator that we want to use).

ψ is a random variable $\psi(x)$, we want to consider its variance $VAR(\psi)$. The variance of the estimator is $VAR(\overline{\psi})$. Within independent samples, the variance of $\overline{\psi}$ would be the one of ψ, divided by the number of samples. But our samples are not independent, there is correlation between samples along the chain (as we are simulating a trajectory in the space of samples, after a while we can be far away from the start, but after one step, we are surely near the previous one).
For the last argument, we define:

- $w = \frac{1}{m}\sum_{j=1}^{m} s_j^2$: within variance, the variance within each chain, where $s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\psi_{i,j} - \overline{\psi_j})^2$ is the estimated variance within the single chain. Because each of this chain does not explore the whole space, we can expect that each one is missing parts of the distribution , so the variance in our chain will always be a lower bound to the true variance of $\psi \to w \leq VAR(\psi)$
- $B = \frac{n}{m-1}\sum_{j=1}^{m}(\overline{\psi_j} - \overline{\psi})^2$ the between variance, i.e. we are trying to compute the variance between points in the different chains. We can show that

$$VAR(\psi) \leq VAR^{+}(\psi) = \frac{n-1}{n}w + \frac{1}{n}B \tag{6.16}$$

The two bounds converge to the true value, so the ratio between the two gives us an estimate of how close we are to a tight estimation.

$$\hat{R} = \sqrt{\frac{VAR(\psi)}{w}} \tag{6.17}$$

where $\hat{R} > 1$, $\hat{R} \xrightarrow[n\to\infty]{} 1$ when $\hat{R} \leq 1.1$ (converged as a rule of thumb).

6.10 Effective number of samples

If the $m \cdot n$ samples which we use to compute ψ would be independent, then we can estimate $VAR(\overline{\psi}) = \frac{VAR(\psi)}{n \cdot m}$. But this property does not hold for MC, for what we have said before.
What we see is

$$n \cdot mVAR(\overline{\psi}) \approx (1 + 2\sum_{k=1}^{\infty} \rho_k) VAR(\psi) \tag{6.18}$$

where ρ_k is the autocorrelation of lag k, defined as

$$\rho_k = CORR[\psi(x_i), \psi(x_{i+k})] \tag{6.19}$$

Comparing the case of dependency and independency, we can correct $n \cdot m$, and compute the effective number of samples as

$$n_{eff} = \frac{n \cdot m}{1 + 2\sum_{k=1}^{\infty} \rho_k} \tag{6.20}$$

and

$$VAR(\overline{\psi}) = \frac{VAR(\psi)}{n_{eff}} \tag{6.21}$$

We have a formula that says

$$E[(\psi_i - \psi_{i-k})^2] = 2(1 - \rho_k) VAR(\psi) \tag{6.22}$$

We can get an estimate using the variogram at lag k

$$V_k = \frac{1}{m(n-k)} \sum_j \sum_{i=k+1}^{n} (\psi_{i,j} - \psi_{i-k,j})^2 \tag{6.23}$$

with

$$\hat{\rho}_k = 1 - \frac{V_k}{2VAR^+(\psi)} \tag{6.24}$$

For large k we have few samples $\rightarrow$ very noisy estimates. We stop the sum, when we detect too much noise, so we compute K as

$$K = \min\{K | \text{K is odd}, \hat{\rho}_{k+1} + \hat{\rho}_{k+2} < 0\} \tag{6.25}$$

So the approximation is

$$\sum_{k=1}^{\infty} \rho_k \approx \sum_{k=1}^{T} \hat{\rho}_k \tag{6.26}$$

6.11 Hamiltonian Monte Carlo

We turn the problem into an hamiltonian one (movement in a landscape with a certain energy function, adding to the state space the momentum variables). It is the state of the art among the Montecarlo methods.
We want to sample from a certain probability distribution

$$p(x) = \frac{1}{Z_x}\widetilde{p}(x) = \frac{1}{Z_x}\exp(H_x(x)) \tag{6.27}$$

We now introduce the momentum variables y, where $|y| = |x|$

$$p(y) = \frac{1}{Z_y}\exp(H_y(y)) \qquad H_y(y) = -\frac{1}{2}y^T y \tag{6.28}$$

And $p(y)$ is a standard gaussian.

$$p(x,y) = p(x)p(y) = \frac{1}{Z_x Z_y}\exp[H_x(x) + H_y(y)] = \frac{1}{Z}\exp[H(x,y)] \tag{6.29}$$

Sample from $p(x,y)$ and forget y. We sample from x by moving along the force field defined by the hamiltonian (the lines which keep the hamiltonian constant).
Steps of the method:

1. We are in point x_i.
2. We sample $y \sim p(y)$.
3. We choose random direction in time (sample from set $\{1,-1\}$)(going back in time is always possible because we are working in an hamiltonian dynamics, there is not a privileged direction in the space, making the problem reversible).
4. We move according to hamiltonian dynamics from (x_i, y) to a candidate (x', y') doing L steps.
5. Metropolis-Hasting acceptance: accept if $H(x', y') > H(x, y)$, otherwise accept with probability $\exp(H(x', y') - H(x, y))$

We want to move from $(x, y) \to (x', y')$, in such a way that $H(x', y') = H(x_i, y)$. How we do it?
In each step we set $(x', y') = (x + \Delta x, y + \Delta y)$.
If the increment is small $H(x + \Delta x, y + \Delta y) \approx H(x, y) + \nabla_x H_x(x)^T \Delta x + \nabla_y H_y(y)^T \Delta y$ which is a first order Taylor expansion.
We can choose $\Delta x, \Delta y$ such that the imposed condition is held

$$\nabla_x H_x(x)^T \Delta x + \nabla_y H_y(y)^T \Delta y = 0 \tag{6.30}$$

So the condition for the hamiltonian dynamic are

$$\Delta x = \epsilon \nabla_y H_y(y)$$
$$\Delta y = -\epsilon \nabla_x H_x(x)$$

We decide at the beginning that $\epsilon = +\epsilon_0$ or $-\epsilon_0$ both with probability $\frac{1}{2}$. Doing L steps of this dynamics to get the final point (x', y') (there are better integration schemes that reduce the error, the most used is the **leap frog** integration).

Chapter 7

Hidden Markov Models

7.1 Introduction and definitions

We work with sequential date (also known as time series), i.e. we describe a process evolving in time. What we actually observe is the process at different steps (we cannot measure continuously, we have a set of subsequent measurement, an array) of $x_1, ..., x_n$ observations, where n is called "step" (a model of time, usually we call it t).
Using makov chains imply that correlation between steps has a really short memory (not always suitable for what we have to model) as it is typical of markov model.
We need more memory! A typical way is to switch from the classical chain we already saw in chapter 6, to one of **order k**: now the decency on the state x_n is not only on the one before, but on the k previous one.
If $k = 2$, the factorization became

$$p(x_1, ..., x_n) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)...p(x_{n+1}|x_n, x_{n-1}) \tag{7.1}$$

In this case

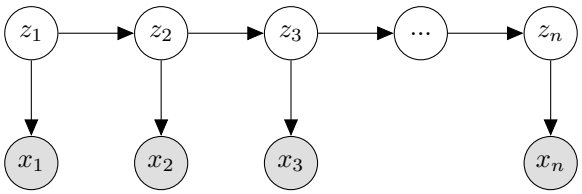
$$x_{n+2} \perp\!\!\!\perp x_{n-1}|x_n, x_{n+1} \tag{7.2}$$

i.e. the independency is pushed k nodes apart. This strategy gives us more memory, but also a more difficult factorization.
Lets gives some definitions:

- If x_i belongs to a descrite space, we are talking about markov chain.
- If x_i is continuous and the factors $p(x_n|...)$ are gaussian, than we are talking about autoregressive models (of order k).
- Stationary (or time homogeneous)

$$p(x_n|x_{n-1}) = p(x_2|x_1) \tag{7.3}$$

The idea is to rely on **state space models**: models that rely on latent variables (or state space)



The observations depends only on the current states. If we consider two observations, then the path between the two is not independent, neither conditional independent on any node z

$$x_i \not\perp\!\!\!\perp x_j | \underline{x} \tag{7.4}$$

where $\underline{x} : x_1, ..., x_n$ is observed.
Again, some nomenclature

- $p(z_n|z_{n-1})$ is the transition probability (between the latent states).
- $p(z_1)$ is the initial distribution.
- $p(x_n|z_n)$ is the emission probability (i.e. what we are going to observe in a certain state).

For discrete z_i we have hidden markov models.
For continuous $z_i, p(z_i|z_{i-1})$ gaussian, we have linear dynamical systems.
In the case of hidden markov models, the transition probabilities can be represented by a discrete probability distribution

$$A_{i,j} = p(z_n = j|z_{n-1} = i) \tag{7.5}$$

that we call transition matrix.
We also write down the initial distribution as

$$\pi_i = p(z_1 = i) \tag{7.6}$$

And the emission probability as ψ parameters, which parameterize the emission probability (discrete x, or continuous x, i.e. gaussian, mixture of gaussian, other...).
Focusing on hidden markov models, we define the tuple θ of the parameters as

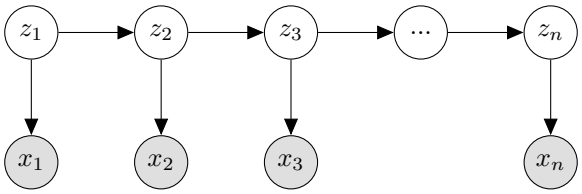
$$\theta = (A, \pi, \psi) \tag{7.7}$$

and define the joint probability distribution as

$$p(x, z|\theta) = p(z_1|\pi)\Big[\prod_{n=2}^{n} p(z_n|z_{n-1}, A)\Big]\Big[\prod_{n=1}^{n} p(x_n|x_n, \psi)\Big] \tag{7.8}$$

with $x, z = (x_1, ..., x_n, z_1, ..., z_n)$.
Typically the representation of the state of z is given by a diagram: a graphical representation for the states of z which can be unfolded as

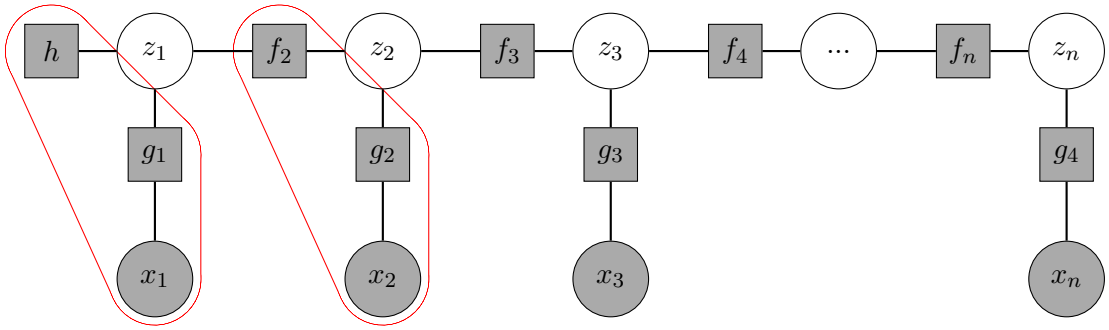
7.2 Inference in HMM



There are different kinds of problems we want to resolve using HMM.

- Filtering: given the observations we have, we want to do a prediction of the hidden state of the last variable; i.e. compute $p(z_n|\underline{x})$ (last time instant n).
- Smoothing (kind of a generalisation of filtering):compute the probability $p(z_n|\underline{x})$, for $k < n$.
- Most likely explanation: we want to find the most likely sequence of latent variables which generated the state. I.e. we want to compute $z^* \arg\max_z p(z|\underline{x})$, where $z = z_1, ...z_n$.
- (Learning: from a set of observations, we want to infer the parameters of the HMM.)

First thing to do is to translate the bayesian network representing the HMM into a factor graph. This means introducing a node for each factor, which in this case correspond to the component of the probability distribution.



We collapse the observation into the factor corresponding to the transition probability between internal states (in red in the image). We essentially eliminate operations on nodes for which messages are relatively simple to define and carry up.

$$h(z_1) = p(z_1)p(x_1|z_1)$$
$$f_2(z_1, z_2) = p(z_2|z_1)p(x_2|z_2)$$
$$...$$
$$f_n(z_{n-1}, z_n) = p(z_n|z_{n-1})p(x_n|z_n)$$

Lets focus on filtering and smoothing: can be solved by apply the sum-product algorithm. In algorithm, there are two kind of messages:

- $\mu_{z_{n-1}} \to f_n(z_{n-1}) = \mu_{f_{n-1}\to z_{n-1}}(z_{n-1})$: messages from variables to factor nodes are essentially the product of incoming factor messages on the corresponding variables.

- $\mu_{f_n \to z_n}(z_n) = \sum_{z_{n-1}} f_n(z_{n-1}, z_n) = \mu_{z_{n-1} \to f_n}(z_{n-1})$.
  We call it $\alpha(z_n) = \sum_{z_{n-1}} f_n(z_{n-1}, z_n)\alpha(z_{n-1})$

So smoothing is solved by computing $p(z_n, \underline{x}) = \alpha(z_n)\beta(z_n)$ (combining forward and backward messages).
Filtering is solved as $p(z_n, \underline{x}) = \alpha(z_n)$, and we can compute the joint probability of $p(z_n, z_{n+1}, \underline{x}) = \alpha(z_n)f(z_n, z_{n+1})\beta(z_{n+1})$.
If we want to solve one of this problems, once we have

$$p(z_n|\underline{x}) = \frac{p(z_n, \underline{x})}{\sum_{z_n} p(z_n, \underline{x}) = p(\underline{x})}$$
$$p(\underline{x}) = \sum_{z_n} \alpha(z_n)$$

The final result is hte most likely sequence (it is a maximisation problem), solved by the max-plus algorithm. The message look like

$$\hat{\mu}_{f_n \to z_n} = \max_{z_{n-1}} \Big\{ \log f_n(z_n, z_{n-1}) + \hat{\mu}_{f_{n-1} \to z_{n-1}}(z_{n-1}) \Big\}$$
$$\phi(z_n) = \arg\max_{z_{n-1}} \Big\{ \log f_n(z_n, z_{n-1}) + \hat{\mu}_{f_{n-1} \to z_{n-1}}(z_{n-1}) \Big\}$$

In the context of HMM the max-plus algorithm is often known as the Viterbi algorithm.

Chapter 8

Bayesian regression and classification

8.1 Bayesian Regression: Introduction to Regression

8.2 Bayesian Regression: Bayesian Linear Regression

Regularization is bias: a distortion that we introduce in our data. We want to bias our inference procedure towards 0.
Another way of doing so probabilistically is expressing our belief in the parameters, placing a prior distribution over those parameters, which encodes our bias (than doing everything in the bayesian way).

$$p(w|\alpha) = \mathcal{N}(w|0, \alpha I) \tag{8.1}$$

Define a probability on hyperparameters which will force the value of the parameters to be small. Given the prior distribution and the likelihood, we can compute the posterior

$$\underbrace{p(w|\underline{x}, \underline{y}, \alpha, \beta)}_{\text{Posterior likelihood}} = \frac{\overbrace{p(\underline{y}|\underline{x}, w, \alpha, \beta)}^{\text{Likelihood}}\overbrace{p(w|\alpha)}^{\text{Prior}}}{\underbrace{p(\underline{y}|\underline{x}, \alpha, \beta)}_{\text{Marginal likelihood}}} \tag{8.2}$$

Once we have the posterior, we compute its logarithm

$$\log p(w|\underline{x}, \underline{y}, \alpha, \beta) = \log p(\underline{y}|\underline{x}, w, \alpha, \beta) + \log p(w|\alpha) - \log p(\underline{y}|\underline{x}, \alpha, \beta) \tag{8.3}$$

But as the logarithm of the marginal does not depend on w, it is a constant.

$$\log p(w|\underline{x}, \underline{y}, \alpha, \beta) = -\frac{\beta}{2}\sum_{j=1}^{N}(y_j - w^T\phi(x_j))^2 - \alpha w^T w + constant \tag{8.4}$$

The first two terms are quadratic in w. Every time we have a quadratic function of w, we have a gaussian distribution. If we regroup the expression in the canonical form of the

gaussian distribution, we can extrapolate the mean and the variance.
The posterior distribution is then

$$p(w|\underline{x}, \underline{y}, \alpha, \beta) = \mathcal{N}(w|m_N, s_N) \tag{8.5}$$

where

$$m_N = \beta s_N \phi^+ \underline{y} \qquad s_N^{-1} = \alpha I + \beta \phi^T \phi \tag{8.6}$$

Which is very similar to the matrix we have to invert in regularized ridge regression. We can place a gaussian prior with a general mean and a general variance.

$$p(w|m_0, s_0) \tag{8.7}$$

If we use a general prior, we are going to have a posterior

$$p(w|\underline{x}, \underline{y}, \alpha, \beta) = \mathcal{N}(w, m_N, s_N) \tag{8.8}$$

computed again as before as

$$m_N = s_N[s_0^{-1} m_0 + \beta \phi^T \underline{y}] \qquad s_N^{-1} = s_0^{-1} + \beta \phi^T \phi \tag{8.9}$$

8.3 Bayesian Regression: Predictive distribution

The predictive distribution is, taking a new x, the probability

$$p(y|x, \underline{x}, \underline{y}, \alpha, \beta) = \underbrace{\int \underbrace{\underbrace{p(y|x, w, \alpha, \beta)}_{\text{Gaussian linear on } w} \underbrace{p(w|\underline{x}, \underline{y}, \alpha, \beta)}_{\text{Gaussian}}}_{\text{Gaussian}} dw}_{\text{Gaussian}} \tag{8.10}$$

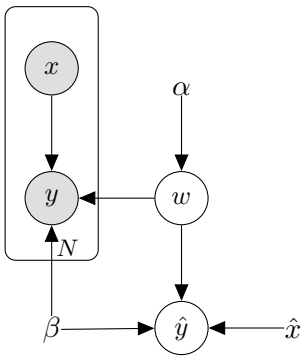
i.e. the probability of our y in a new point in space (giving a prediction in terms of full probability distribution).
So

$$p(y|x, \underline{x}, \underline{y}, \alpha, \beta) = \mathcal{N}(\frac{y}{m_N^T \phi(x)}, \sigma_N^2(x)) \tag{8.11}$$

With

$$\sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^T s_N \phi(x) \qquad \sigma_N^2(x) \geq \sigma_{N+1}^2(x), \sigma_N^2(x) \xrightarrow[N \to \infty]{} \frac{1}{\beta} \tag{8.12}$$

The variance has two terms: one is the intrinsic uncertainty in our model (noise from the observation), and the uncertainty we have caused by the finite amount of information at disposal (epistemic uncertainty).
Lets write a PGM of BLR.



8.4 Bayesian Regression: Evidence

Marginal likelihood is

$$p(\underline{y}|\underline{x},\alpha,\beta) = \int p(\underline{y}|\underline{x},w,\alpha,\beta)p(w|\alpha)dw \tag{8.13}$$

as the posterior

$$p(w|\underline{y},\underline{x},\alpha,\beta) = \frac{p(\underline{y}|\underline{x},w,\alpha,\beta)p(w|\alpha)}{p(y|\underline{x},\alpha,\beta)} \tag{8.14}$$

The bayesian way to deal with α and β is to place an hyperprior $p(\alpha,\beta)$, then compute the posterior distribution of α and β conditioned on the observation we have

$$p(\alpha,\beta|\underline{y},\underline{x}) \propto p(\underline{y}|\underline{x},\alpha,\beta) \qquad \underbrace{p(\alpha,\beta)}_{\text{Uninformative prior}} \tag{8.15}$$

This is in principle doable, but require the specification of the distributions of α and β. We want to approximate using the maximum a posteriori approximation of the posterior of α and β. So we ask for an uninformative prior of $p(\alpha,\beta)$. This means that $p(\alpha,\beta)$ is constant, does not really impact on the function definition, and the posterior is proportional to the likelihood.
Another approximation we impose is that our posterior is sharply picked around the maximum a posteriori (MAP)

$$p(\alpha,\beta|\underline{y},\underline{x}) \approx \delta_{MAP(\alpha,\beta)} \tag{8.16}$$

The two approximations imply that we can fix α,β to the maximum likelihood solution. We can then maximise the maximum likelihood.
How to compute the marginal likelihood? From the expression of the posterior (as we have everything we need).

$$\log p(\underline{y}|\underline{x},\alpha,\beta) = \frac{M}{2}\log(\alpha) + \frac{N}{2}\log\beta - E(m_N) - \frac{1}{2}\log|s_N|^{-1} - \frac{N}{2}\log 2\pi \tag{8.17}$$

Where

$$E(m_N) = \frac{\beta}{2}\|y - \phi m_N\|^2 + \frac{\alpha}{2}m_N^T m_N \tag{8.18}$$

and

$$m_N = \beta s_N \phi \underline{y} \qquad s_N^{-1} = \alpha I + \beta\phi^T\phi \tag{8.19}$$

8.5 Bayesian Regression: Evidence optimization via a fixpoint algorithm

The idea is to use the gradient as

$$\nabla \log p(\underline{y}|\underline{x}, \alpha, \beta) = 0 \tag{8.20}$$

And derive from tha fixpoint equations

$$\alpha = g_\alpha(\alpha, \beta)$$
$$\beta = g_\beta(\alpha, \beta)$$

Algorithm:

- Fix α_0, β_0
- Compute

$$\alpha_{N+1} = g_\alpha(\alpha_N, \beta_N)$$
$$\beta N + 1 = g_\beta(\alpha_N, \beta_N)$$

- Iterate until convergence

$$\|\alpha_{N+1} - \alpha_N\| + \|\beta_{N+1} - \beta_N\| < \epsilon$$

So we have

$$\log p(\underline{y}|\underline{x}, \alpha, \beta) = \frac{M}{2}\log\alpha + \frac{N}{2}\log\beta - E(m_N) - \frac{1}{2}\left|s_N^{-1}\right| - \frac{N}{2}\log 2\pi$$
$$E(m_N) = \frac{\beta}{2}\left\|\underline{y} - \phi m_N\right\|^2 + \frac{\alpha}{2}m_N^T m_N$$
$$s_N^{-1} = \alpha I + \beta\phi^T\phi$$

We have to compute the gradient of this function (the derivative of every single term). First of all, we need the determinant of $\left|s_N^{-1}\right|$. We need to compute $\beta\phi^T\phi$ and compute eigenvalues $\lambda_i > 0$

$$\left|s_N^{-1}\right| = \prod_{i=0}^{M-1}(\alpha + \lambda_i) \tag{8.21}$$

However, the eigenvalues $\beta\phi^T\phi$ do not depend on α. So, whenever

$$\frac{\partial}{\partial\alpha}\log\left|s_N^{-1}\right| = \frac{\partial}{\partial\alpha}\sum\log(\alpha + \lambda_i) = \sum_{i=1}^{M-1}\frac{1}{\alpha + \lambda_i}$$
$$\frac{\partial}{\partial\beta}\lambda_i = \frac{\lambda_i}{\beta}$$

So we get this equation for α

$$\alpha = \frac{\gamma}{m_N^T m_N} = g_\alpha(\alpha, \beta) \text{ ,when } \gamma = \sum_{i=0}^{M-1} \frac{\lambda_i}{\alpha + \lambda}$$

An for β

$$\frac{1}{\beta} = \frac{1}{N - \gamma} \sum_{n=1}^{N} y_n - m_N^T \phi(x_n) = \frac{1}{g_\beta(\alpha, \beta)}$$

8.6 Bayesian Regression: Effective number of parameters

We still have to discuss γ

$$\gamma = \sum_{i=0}^{M-1} \frac{\lambda_i}{\alpha + \lambda_i} \tag{8.22}$$

Where λ_i are the eigenvalues of $\beta \phi^T \phi$. The lambdas, after realignment and other adjustments, gives us information about the maximum likelihood solution in terms of the weights w. They give the curvature of the likelihood around the maximum likelihood solution.

- Small $\lambda_i \rightarrow$ large uncertainty on w_i:

$$\lambda_i << \alpha \Rightarrow \frac{\lambda_i}{\lambda_1 + \alpha} \approx 0 \tag{8.23}$$

- Large $\lambda_i \rightarrow$ small uncertainty on w_i:

$$\lambda_i >> \alpha \Rightarrow \frac{\lambda_i}{\lambda_1 + \alpha} \approx 1 \tag{8.24}$$

When we have a large uncertainty in a certain λ_i, it means that the parameter is not very sensitive. At the opposite, when the uncertainty is small, than the bayesian approach is likely to remain close to the maximum likelihood solution (not altered too much).
The effective number of parameters that should be used in the model, is the one such that the MAP and the ML do not change much.
So γ tells us how many effective parameters we have (significant for our function).
So for $N >> M \Rightarrow \gamma \approx M$. In this regime, also the equation for α and β simplify.
Marginal likelihood is also known as **evidence**.

8.7 Bayesian Regression: Bayesian model comparison

Suppose that we have two model2 M_1, M_2 (different choices of basis functions). Which one is the best to explain the data $\mathcal{D} = \{\underline{x}, \underline{y}\}$.
We want to use a bayesian strategy. We call the prior distributions on the models $p(M_j)$.

Once we have this, we want to compute the posterior on the model given the data $p(M_j|\mathcal{D})$, which is

$$p(M_j|\mathcal{D}) = \frac{\overbrace{p(\mathcal{D}|M_j)}^{\text{Marginal likelihood}} \quad p(M_j)}{\sum_j p(\mathcal{D}|M_j)p(M_j)} \tag{8.25}$$

We assumed that the hyperparameters are fixed (already optimized by strategy seen in previous lectures).
Which model should we choose between the two?

- Model averaging: rather that picking a model, we consider all of them, weighted according to their posterior distribution.

$$\underbrace{p(y|x,\mathcal{D})}_{\text{Predictive}} = \sum_j p(y|x,\mathcal{D},M_j) \cdot p(M_j|\mathcal{D}) \tag{8.26}$$

- Choose the best model

$$\frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_2)} = \text{Bayes factor} \tag{8.27}$$

  Which tell us the relative evidence between the model. We choose the model with the largest bayes factor. It can be proved that works, because this relationship holds (assuming that M_1 is the correct model)

$$\int p(\mathcal{D}|M_1) \log \frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_2)} d\mathcal{D} > 0 \tag{8.28}$$

8.8 Bayesian Regression: Equivalent Kernel

Lets consider the predictive distribution on a bayesian linear regression model

$$p(y|x,\underline{x},\underline{y},\alpha,\beta) = \mathcal{N}(y|m_N^T\phi(x), \sigma_N^2(x)) \tag{8.29}$$

Lets focus then on the mean predictor for y

$$y(x,m_N) = m_N^T\phi(x) = \beta\phi(x)^T s_N \phi\underline{y} = \sum_{n=1}^{N} \beta\phi^T(x)s_N\phi(x_n)y_n$$

$$= \sum_{n=1}^{N} k(x,x_n)y_n$$

the posterior predictive mean is essentially a linear combination of the observation that we have weighted by a term (the **equivalent kernel**) which is defined as

$$k(x,x') \coloneqq \beta\phi^T(x)s_N\phi(x') \tag{8.30}$$

Which does not depend on the observations, only on the input and our feature model.
When we take the predictive distribution, and evaluate the covariance at two different points of x

$$cov[y(x),y(x')] = \phi^T(x)s_N\phi(x') = \beta^{-1}K(x,x') \tag{8.31}$$

We can also write the kernel as a scalar product

$$K8x,x') = \psi^T(x)\psi(x') \qquad \psi(x) = \beta^{\frac{1}{2}}s_N^{\frac{1}{2}}\phi(x) \tag{8.32}$$

8.9 Bayesian Regression: Dual formulation of linear regression

In our problem we have our observations $\underline{xy} = (x_n, y_n)_{n=1,...,N}$, and a linear model $w^T\phi(x)$. Whenever we are optimising the weights, we are computing the values of our function on the basis function evaluated on the different input points $w^T\phi(x_n)$. This is telling us that, in the optimisation problem, the solution we are looking for w is contained in the subspace spanned by $< \phi(x_1), ..., \phi(x_N) >$. Essentially any component orthogonal to this space will be 0, and don't give any contribution.
We can than reparametrize w

$$w = \sum_{j=1}^{N} a_j \phi(x_j) \tag{8.33}$$

Practically, instead of using variables w, we can use variable a, known as the **dual variables**.
We can then introduce a kernel

$$k(x_i, x_j) \coloneqq \phi^T(x_i)\phi(x_j) \tag{8.34}$$

Once we have this kernel, we can define the **Gram matrix** (K^i is the i$^{\text{th}}$ column)

$$K = (k_{ij}) \qquad k_{ij} = k(x_i, x_j) \tag{8.35}$$

And with this definition we can write

$$w^T\phi(x_i) = a^T K^i \tag{8.36}$$

which is called "dual formulation".
Lets now write down the regularized regression problem that we want to solve

$$E_{\mathcal{D}}(a) + \lambda E_w(a) = \sum_{n=1}^{N} (y_n - a^T K^n)^2 + \lambda a^T K a \tag{8.37}$$

We still have a convex optimisation problem, so we can get the gradient of a with respect to 0, and obtain the maximum likelihood solution

$$\begin{aligned} \hat{a} &= (K + \lambda I)^{-1}\underline{y} \qquad \text{ML solution} \\ y(x^*) &= K_*^T(K + \lambda I)^{-1}\underline{y} \qquad \text{prediction} \end{aligned}$$

Where

$$K_* = (K(x_*, x_1), ..., K(x_*, x_N)) \tag{8.38}$$

This implicit dependency on the basis function, which allows us to rephrase the whole problem using only the scalar product evaluated between the inputs and the new prediction points, is known as the **kernel trick**. This allows us to work on function spaces that could even be infinite dimensional, by working implicitly.
Using the kernel trick, we are rephrasing the problem form the one of inverting the matrix which depends on the number of basis function that we have (complexity $O(M^3)$ to one that depends on the number of observations (complexity $O(N^3)$).

8.10 Bayesian Classification: Introduction and Logistic regression

Lets consider a dataset of input and output points $\mathcal{D} = (x_i, y_i), i = 1, ..., N$. The output point are categorical (belong to discrete class), typically we have a two class problem $y_n \in \{0, 1\}$. Otherwise we have a $k > 2$ class problem $y_n = (y_{nj})_{i=1,...,k}$, in this case we stick to the **one hot encoding**, where we represent our output over k classes by k boolean , with the constrain that only 1 is equal to 1, in the corresponding class

$$y_{nj} \in \{0, 1\} \qquad \sum_j y_{nj} = 1 \tag{8.39}$$

There are three main kind of approaches to the problem

- Discriminant function
$$f(x) \in \{1, ..., k\} \tag{8.40}$$
We try to learn a function which map every input into a class.

- Discriminative approach
$$p(c_k|x) = \underbrace{f(\overbrace{w^T \phi(x)}^{h(x)})}_{\text{Activation function}} \tag{8.41}$$
We try to model explicitly the class conditional probability. In logistic regression, what we use is a linear function of x, than we pass it through an activation function f which returns a number between 0 and 1. f^{-1} is known as the link function.

- Generative approach: model the class conditional probabilities, by directly describing the generative distribution of input conditional class, and use the bayes theorem to compute the class conditional probability that we need for classification
$$p(c_k|x) = \frac{p(x|c_k)p(c_k)}{p(x)} \tag{8.42}$$

Logistic Regression

Once again we have a discriminative model: (x_n, y_n), $n = 1, ..., N$, $\phi(x) = (\phi_0(x), ..., \phi_{M-1}(x))$ is the basis functions. We explicitly model probability of class C_1 for a given input x as an activation function f which maps real number computed by a linear combination of basis function, to $\{0, 1\}$

$$p(c_1|x) = f(w^T \phi(x)) \tag{8.43}$$

This activation function can be (typical choices):

- Logit
$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{8.44}$$

- Probit
$$\psi(a) = \int_{-\infty}^{a} \mathcal{N}(\theta|0, 1)d\theta \tag{8.45}$$

Using logit, we define

$$s_n = \sigma(w^T \phi(x_n)) \tag{8.46}$$

For a fixed w, we use a bernoulli model for noise

$$p(y_n|x_n) = s_n^{y_n}(1 - s_n)^{1-y_n} \tag{8.47}$$

The likelihood is instead

$$p(\underline{y}|\underline{x}) = \prod_{n=1}^{N} s_n^{y_n}(1 - s_n)^{1-y_n} \tag{8.48}$$

And the cross-entropy

$$E(w) = -\frac{1}{N}\sum_{i=1}^{N} \log p(\underline{y}|\underline{x}) = -\frac{1}{N}\sum_{i=1}^{N} y_n \log s_n + (1 - y_n)\log(1 - s_n) \tag{8.49}$$

The dependency on the weights w is through the $\log s_n$, so the dependency is non-linear, surely not quadratic, so the whole expression is not gaussian. Its gradient

$$\nabla E(w) = \frac{1}{N}\sum_{i=1}^{N}(s_i - y_i)\phi(x_i) \tag{8.50}$$

($\nabla E(w) = 0$ has no analytical solution).
In the maximum likelihood we need to minimise the cross entropy (the function remain convex, so we can use numerical optimisation)

$$w_{ML} = \arg\min_w E(w) \tag{8.51}$$

- Gradient descent rule used

$$w_{n+1} = w_n - \eta_n \nabla E(w_n) \tag{8.52}$$

- The separating hyperplane is defined by the linear equation

$$w_{ML}^T \phi(x) = 0 \tag{8.53}$$

  whenever the data points are linearly inseparable (there are infinite hyperplane that could work). So the typical thing that one does is to introduce a penalty term in the function optimized. This keep the problem convex, so we can still use gradient descent (avoiding infinite solutions, and a possible exploding ML).

Once we have a multiclass problem (x_n, y_n), with $y_n = (y_{n1}, ..., y_{nk})$ with k classes.
Than we compute the conditional distribution (with W^T is a $K \times M$ matrix)

$$p(c_k|x) = \sigma_k(W^T \phi(x)) \tag{8.54}$$

obtained as k linear combinations, and passing them to a σ_k (a softmax function, i.e. takes a vector of real numbers, and turns them into probability distributions, by exponentiating them and normalizing)

$$\sigma_k(\underline{a}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{8.55}$$

Now

$$E(w) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{k} y_{nj}\log s_{nk} \qquad s_{nk} = \sigma_k(W^T) \tag{8.56}$$

and

$$\underbrace{\nabla_{Wj}E(w)}_{\text{Gradient for class j weight}} = \frac{1}{N}\sum_{n=1}^{N}(s_{nj} - y_{nj})\phi(x_n) \tag{8.57}$$

8.11 Bayesian Classification: Laplace Approximation

Laplace approximation: we get the mode of the distribution of interest, and send the gaussian in the mode, so it coincide with the one of the distribution. We need to identify the variance, identifying the hessian matrix (the second derivative), which gives us the curvature of the mode, and we match the one of the gaussian with the one of the distribution. We typical use this to approximate posterior distribution.
Lets start in a 1 dimensional case

$$p(z) = \frac{1}{Z}f(z) \qquad Z = \int f(z)dz \tag{8.58}$$

(we know f but not z). In the Laplace approximation we

- Find a mode of $f(z)$, say z_0
- Find a gaussian distribution that matches the curvature of f of z_0 with a normal distribution.

Let z_0 be a point of local maximum s.t. $\frac{d}{dz}f(z_0) = 0$. Now we Taylor expand $\log f(z)$ around z_0:

$$\log f(z) \approx \log f(z_0) - \frac{1}{2}A(z - z_0)^2 \tag{8.59}$$

Where

$$A = -\frac{d^2}{dz^2}\log f(z_0), A > 0 \tag{8.60}$$

If we exponentiate this Taylor expansion, we get an approximation of $f(x)$

$$f(x) \approx f(z_0)\exp\left(-\frac{1}{2}A(z - z_0)^2\right) \tag{8.61}$$

So we essentially have a gaussian where A plays the role of the precision of the gaussian distribution.
Now we want to approximate p by a gaussian $q(z) \sim \mathcal{N}(z|z_0, A^{-1}) = (\frac{A}{2\pi})^{\frac{1}{2}}\exp\big(-\frac{1}{2}A(z - z_0)^2\big)$, than we take the fact that

$$p(z) = \frac{1}{Z}f(z) \approx \frac{1}{Z}f(z_0)\exp\left(-\frac{1}{2}A(z - z_0)^2\right)$$
$$q(z) = \left(\frac{A}{2\pi}\right)^{\frac{1}{2}}\exp\left(-\frac{1}{2}A(z - z_0)^2\right)$$

And match the two expressions, obtaining as a consequence an estimation of the marginal likelihood

$$z \approx f(z_0)\Big(\frac{A}{2\pi}\Big)^{-\frac{1}{2}} \tag{8.62}$$

In the n-dimensional case, we now have a vector

$$p(z) = \frac{1}{Z} f(z) \tag{8.63}$$

taking the Taylor expansion, z_0 is a mode of f, and

$$\log f(z) \approx \log f(z_0) - \frac{1}{2}(z - z_0)^T A (z - z_0) \qquad A = -\nabla\nabla \log f(z_0) \tag{8.64}$$

So we take the guassian approximation as

$$q(z) = \mathcal{N}(z|z_0, A^{-1})$$
$$z = \frac{(2\pi)^{\frac{n}{2}}}{|A|^{\frac{1}{2}}} f(z_0)$$

so we have similar results carrying out the computation in this n dimensional cases.

8.12 Bayesian Classification: Laplace model selection

Suppose that we want to do some sort of bayesian model analysis of some data $\mathcal{D}$, and a model $\mathcal{M}$ which depends on parameters θ, and a prior distribution $p(\theta)$. We consider the posterior distribution

$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\mathcal{D}|\theta) \cdot p(\theta)}^{\text{Computable}}}{\underbrace{p(\mathcal{D})}_{\text{Hard to compute}}} \tag{8.65}$$

We remember that

$$p(\mathcal{D}) = \int p(\mathcal{D}|\theta) p(\theta) d\theta \tag{8.66}$$

This fits the previous setting if we set

$$f(\theta) = p(\mathcal{D}|\theta) \cdot p(\theta) \qquad z = p(\mathcal{D}) \tag{8.67}$$

Where we can use Laplace.

1. θ_{MP}: we need to identify the maximum a posteriori, the maximum of the function f (the normalized prior). We do Laplace approximation around maximum a posteriori. We get (with $M = |\theta|$ cardinality of parameters)

$$\log p(\mathcal{D}) \approx \log p(\mathcal{D}|\theta_{MP}) + \log p(\theta_{MP}) + \frac{M}{2}\log(2\pi) - \frac{1}{2}\log|A| \tag{8.68}$$

   Where

$$A = -\nabla\nabla[\log p(\mathcal{D}|\theta) + \log p(\theta_{MP})] \tag{8.69}$$

There is one index used for model comparison, that is a crude approximation of the Laplace approximation, but is faster to compute and does not require a prior distribution of the parameter, is know as the **bayesian information content (BIC)**

$$\log p(\mathcal{D}) \approx p(\mathcal{D}|\theta_{MP}) - \frac{1}{2} M \log N \tag{8.70}$$

The more parameters we have, the larger will be the penalty term. The more data we have, we need to use simpler model (not totally true, also the likelihood improves if we have a lot of data points).

8.13 Bayesian Classification: Bayesian Logistic Regression

Given observations $(x_n, y_n)_{n=1...N}$, we are going to consider a set of basis function $\phi(x) = \phi_0^{(x)}, ..., \phi_{M-1}^{(x)}$, and we are going to consider a linear model $\sigma(w^T \phi(x))$ with logit activation function.
The index of the model is the linear parameter w. In the bayesian strategy, we are going to place a prior distribution on w, then get the posterior (a gaussian distribution)

$$p(w) = \mathcal{N}(w|m_0, s_0) \tag{8.71}$$

The posterior is then, using bayes theorem

$$p(w|\underline{y}, \underline{x}) = \frac{p(\underline{y}|w, \underline{x}) \cdot p(w)}{p(\underline{y}|\underline{x})} \propto p(y|w, \underline{x}) p(w) \tag{8.72}$$

And

$$p(\underline{y}|w, \underline{x}) = \prod_{i=1}^{N} s_i^{y_i} (1 - s_i)^{1-y_i}$$
$$s_i = \sigma(w^T \phi(x_i))$$

Lets take the logarithm of this

$$\log p(w|\underline{y}, \underline{x}) = \underbrace{-\frac{1}{2}(w - m_0)^T s_0^{-1}}_{\propto \log p(w)} + \underbrace{\sum_{i=1} \Big[y_i s_i(w) + (1 - y_i)(1 - s_i(w)) \Big]}_{\log p(\underline{y}|w)} + c \tag{8.73}$$

Here $s_i(w)$ is not quadratic in w, but depend on it through a logit, so there is an exponential relation, not treatable analytically.
We instead plug in the Laplace approximation (Taylor expansion of the posterior around the mode,take a quadratic expansion of the second order term, then build the gaussian out of that). $p(w) = \frac{1}{Z} f(w)$ and $f(w) = p(w)p(\underline{y}|w)$

1. Find $w_{MAP} = \arg\max_w \log p(w|\underline{y}) = \arg\max_w \log p(w) + \log p(\underline{y}|w)$
2. Compute the hessian at $w_{MAP}$of the function f (the unnormalized posterior), which will gives us the gaussian

$$s_N^{-1} = s_0^{-1} + \sum_{n=1}^{N} s_n(w_{MAP})(1 - s_n(w_{MAP})) \phi(x_n) \phi^T(x_n) \tag{8.74}$$

So our posterior distribution is going to be approximated as

$$p(w|\underline{y}) \approx q(w) \qquad q(w) = \mathcal{N}(w|w_{MAP}, s_N) \tag{8.75}$$

With this posterior we need to do model averaging, i.e. compute the predictive distribution.

8.14 Bayesian Classification: Predictive Distribution

How we get to the predictive distribution?

$$\begin{aligned} p(c_1|x^*, \underline{y}, \underline{x}) &= \int p(c_1|x^*, w, \underline{x}, \underline{y}) p(w|\underline{y}, \underline{x}) dw \\ &\approx \int \sigma(w^T \phi(x^*)) q(w) dw \end{aligned}$$

But, even with the approximation, we still have a multidimensional integral (M-dimensional). We can notice that σ depends on w only through the projection of w in the lines spanned by the vector $\phi(x^*)$.
Because it is a scalar, it is a sort of 1-D thing. The scalar product is a linear combination of different coefficient of w, each multiplied by the corresponding basis function ϕ, evaluated at x^*.
So $a = w^T \phi(x^*)$ is gaussian, by the properties of the gaussian distribution. So practically this is $q(a) = \mathcal{N}(a|\mu_a, \sigma_a^2)$, with $\mu_a = w_{MAP}^T \phi(x^*)$ and $\sigma_a^2 = \phi^T(x^*) s_N \phi(x^*)$. We can than write

$$p(c_1|x^*, \underline{y}, \underline{x}) = \int \sigma(a) q(a) da \tag{8.76}$$

To calculate it, we use probit approximation

$$\sigma(a) \approx \psi(\lambda_a) \qquad \lambda^2 = \frac{\pi}{8} a \text{ matching } \sigma'(0) = \psi'(0) \tag{8.77}$$

Then we have the property

$$\int \psi(\lambda_a) q(a) da = \psi\left(\frac{\mu_a}{(\lambda^{-2} + \sigma^2)^{\frac{1}{2}}} \right) \tag{8.78}$$

So the wanted original expression can be further approximated as

$$p(c_1|x^*, \underline{y}, \underline{x}) \approx \sigma(k(\sigma_a^2), \mu_a) \qquad k(\sigma_a^2) = (1 + \pi \frac{\sigma_a^2}{8})^{\frac{1}{2}} \tag{8.79}$$

Chapter 9

Gaussian Processes

9.1 Introduction and Random Functions

Lets consider a vector $x \in \mathbb{R}^n$. We pick a function which is defined as

$$f(x) = w_0\phi_0(x) + ... + w_{M-1}\phi_{M-1}(x) \tag{9.1}$$

Were $\phi(x) = \phi_j(x)$ is our set of basis functions (they are as basis in a vector space, their directions in the spaces of function are what we use to construct more complicated functions by taking linear combination with weights $w_0, ..., w_{M-1}$).
The set of all functions defined as above, is a M-dimensional vector space

$$F = \{f | f(x) = w_0\phi_0(x) + ... + w_{M-1}\phi_{M-1}(x)\} \tag{9.2}$$

what is a distribution over the space F?
As my f is identified by the vector of coefficients $f \iff (w_0, ..., w_{M-1}) = w$, then the distribution over f is the distribution over the vector w.
For the moment, we take as distribution $w \sim \mathcal{N}(0, I)$.
Under some condition of regularity, our distribution over the space of functions is determined by the **finite dimensional projection**.
If we have our collection of functions F, and we pick a function $f \in F$, then the probability distribution over the space F is practically described by knowing the probability distribution of

$$\underbrace{p(f(x_1), ..., f(x_n))}_{\text{Finite dimensional projection}} \quad , \forall x_1, ..., x_N \in \mathbb{R}^n \tag{9.3}$$

How do the finite dimensional projection of distribution defined before looks like?
Lets fix a single point (look at the 1-D function), what is $p(f(x))$?
We know that $f(x) \in \mathbb{R}$ (is a real number), so its $p(f(x))$ is a distribution over reals.
We evaluate ϕ over x and we have M numbers, and we multiply this numbers by random coefficients which are gaussian distributed. So we have a linear combination of gaussian distributions.

$$f(x) = \sum_{j=0}^{M-1} \underbrace{w_j}_{\sim\mathcal{N}(0,1)} \underbrace{\phi_j(x)}_{\in\mathbb{R}} = \mathcal{N}(0, \underbrace{\phi^T(x) I \phi(x)}_{\phi^T(x)\phi(x)=k(x,x)}) \tag{9.4}$$

but what about 2-D marginals?
$f_1 \in F$ and fix $x_1, x_2 \in \mathbb{R}^n$, we want to consider $p(f(x_1), f(x_2))$ (the vector is in $\mathbb{R}^2$).

$$\begin{pmatrix} f(x_1) \\ f(x_2) \end{pmatrix} = w^T \begin{pmatrix} \phi(x_1) \\ \phi(x_2) \end{pmatrix} = \begin{pmatrix} \phi(x_1) & \phi(x_2) \end{pmatrix} w \tag{9.5}$$

where $w \sim \mathcal{N}(0, I)$ is still a set of gaussian distributions. So again

$$p(f(x_1), f(x_2)) = \mathcal{N}\left(0, \begin{pmatrix} \phi(x_1) \\ \phi(x_2) \end{pmatrix}^T I \begin{pmatrix} \phi(x_1) \\ \phi(x_2) \end{pmatrix}\right) \tag{9.6}$$

So

$$cov(f(x_1), f(x_2)) = \begin{pmatrix} \phi^T(x_1)\phi(x_1) & \phi^T(x_1)\phi(x_2) \\ \phi^T(x_2)\phi(x_1) & \phi^T(x_2)\phi(x_2) \end{pmatrix} \tag{9.7}$$

this matrix is symmetric, and is telling us that, if we want, we can rewrite it as

$$\begin{pmatrix} \phi^T(x_1)\phi(x_1) & \phi^T(x_1)\phi(x_2) \\ \phi^T(x_2)\phi(x_1) & \phi^T(x_2)\phi(x_2) \end{pmatrix} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) \\ k(x_1, x_2) & k(x_2, x_2) \end{pmatrix} \tag{9.8}$$

From which

$$p(f(x_1), ..., f(x_N)) = \mathcal{N}(0, K) \qquad K = (k_{ij}), k_{ij} = k(x_i, x_j) \tag{9.9}$$

Remembering that the kernel is $k(x_1, x_2) = \phi^T(x_1)\phi(x_2)$.
What we got is the property that finite dimension projections are gaussian distribution.

9.2 Definition of GP

Definition 36. ***Gaussian process (GP)****: a stochastic process indexed by a continuous variable $x \in \mathbb{R}^n$ such that all finite dimensional distributions are gaussians.*

For us, a GP is a random variable $f(x), x \in \mathbb{R}^n$ (we have an uncountable number of random variables). We are interested in the joint distribution of this variables. Stochastic process means random function. So, all together

$$f(x), x \in \mathbb{R}^n \; s.t. \; \forall x_1, ..., x_N \in \mathbb{R}^n, p(f(x_1), ..., f(x_N)) = \mathcal{N}(m_N, \Sigma_N) \tag{9.10}$$

So a GP is defined by $\mu : \mathbb{R}^n \to \mathbb{R}$, the mean at point $x(\mu(x))$ and another function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, the covariance function between two points, which means

$$f \sim GP(\mu, K) \; iff \; \forall x_1, ..., x_N \in \mathbb{R}^n p(\underline{f}) = p(f(x_1), ..., f(x_N)) = \mathcal{N}(\mu, K) \tag{9.11}$$

With

$$\mu = (\mu(x_1), ..., \mu(x_N)) \qquad K = (k_{ij})_{ij} \qquad k_{ij} = k(x_i, x_j) \tag{9.12}$$

K is not an arbitrary function, it has to satisfy some conditions:

$$K \text{ is such that } \forall x_1, ..., x_N, \; K \text{ is positive defined and symmetric} \tag{9.13}$$

9.3 Regression: noise free case

The function we are learning are not defined by any parameter. The method we have is not parametric, it heavily depends on the kernel, but apart from the restriction on the set of function we can pick from, we have no restrictions.

We start with a regression problem: we observe some $f(x)$ which is unknown. We observe the **true values** (no noise) at points $\underline{x} = x_1, ..., x_N$, so observations are $\underline{f} = f(x_1), ..., f(x_N)$.
We then consider test points $\underline{x^*} = x_1, ..., x_\nu$ where we want to actually test (or observe) the posterior values of our function, with observations $\underline{f^*} = f(x_1), ..., f(x_\nu)$
We want to calculate the joint prior distribution of $\underline{f}$ and $\underline{f^*}$, with prior $GP(0, K)$

$$p\begin{pmatrix}\underline{f}\\ \underline{f^*}\end{pmatrix} \sim \mathcal{N}(0, \begin{bmatrix}K(\underline{x},\underline{x}) & K(\underline{x},\underline{x^*})\\ K(\underline{x^*},\underline{x}) & K(\underline{x^*},\underline{x^*})\end{bmatrix} \tag{9.14}$$

where

$$K(\underline{x},\underline{x^*}) = (k_{ij})_{\substack{i=1,...,N\\ j=1,...,\nu}} \qquad k_{ij} = k(x_i, x_j^*) \tag{9.15}$$

If f is observed, then we have that we can easily compute the posterior distribution

$$p(\underline{f^*}|\underline{f}) \sim \mathcal{N}(K(\underline{x^*},x)K(\underline{x}\underline{x}^{-1}\underline{f}, K(\underline{x^*},\underline{x^*}) - K(\underline{x^*},\underline{x})K(x,x)^{-1}K(\underline{x},x^*)) \tag{9.16}$$

So the posterior distribution is obtained by inverting the Gram Matrix of the inputs points, then computing the kernel between the prediction point and observed points, get another matrix, and multiply the two and multiply by the observations, or rescale the covariance between prediction points by reducing it by a quantity which depends on inverse of the Gram matrix of the observations and the matrix between predictions and the kernel between prediction and input points.
This works for any tuple $\underline{x^*}$, this means that the posterior distribution, is the distribution of a function that has the following property: for any finite dimension projection on input points $\underline{x^*}$, the distribution is gaussian, with the declared mean and covariance. The posterior is then a Gaussian process. The mean of a posterior is not constant anymore.

9.4 Regression: noisy setting

What happens if we have noise?
We choose specifically gaussian noise, what we observe is not our function f, but $y(x) = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The observations that we have are, again, gaussian, have mean 0, but, considering the covariance matrix, we need to add σ^2 to each of the diagonal terms: noise is happening independently with variance σ^2 at each observation point.

$$p(\underline{y}) = \mathcal{N}(0, K(\underline{x},\underline{x} + \sigma^2 I) \tag{9.17}$$

At $\underline{x^*}$ and $\underline{f^*} = f(\underline{x^*})$ we want to make prediction. The joint prior is, again, gaussian: we have the convariance matrix, similar to before, but we have the $+\sigma^2 I$ at the prediction points.

$$p\begin{pmatrix}\underline{y}\\ \underline{f^*}\end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix}K(\underline{x},\underline{x}) + \sigma^2 I & K(\underline{x},\underline{x^*})\\ K(\underline{x^*},\underline{x}) & K(\underline{x^*},\underline{x^*})\end{bmatrix}\right) \tag{9.18}$$

Now the posterior is

$$p(\underline{f^*}|\underline{y}) \sim \mathcal{N}(K(\underline{x^*},\underline{x})[K(\underline{x},\underline{x}+\sigma^2 I]^{-1}\underline{y}, K(\underline{x^*},\underline{x^*}) - K(\underline{x^*},x)[k(\underline{x},\underline{x})+\sigma^2 I]^{-1}K(\underline{x},\underline{x^*}) \tag{9.19}$$

The posterior is again a GP, the "cost" is computing $(K(\underline{x},\underline{x}),\sigma^2 \underline{I}^{-1}$, which is $N \times N$ matrix (complexity $O(N^3)$).
Consider that we want to make a prediction on point $x, f(x)$

$$f(x) \sim \mathcal{N}(\underline{k}(K+\sigma^2 I)^{-1}\underline{y}, k(x,x) - \underline{k^T}(K+\sigma^2 I)^{-1}\underline{k} \tag{9.20}$$

with vector $\underline{k}$

$$\underline{k} = (k(x,\underline{x})) = (k(x,x_1),...,k(x,x_N)) \tag{9.21}$$

We notice that the mean can be written as

$$\mathbb{E}[f(x)|\underline{y}] = \sum_{i=1}^{N} \alpha_i k(x,x_i) \qquad \alpha = [k(\underline{x},\underline{x}+\sigma^2 I]^{-1}\underline{y} \tag{9.22}$$

So practically the mean can be a linear combination of kernel functions evaluated between the prediction points and all the other input points, weighted by some coefficients which depends on the observation of the input points.

9.5 Kernel functions and Hilbert Spaces

Definition 37. ***Kernel****: an integral operator over a certain space $\mathcal{X}$ (or $\mathcal{X}^n$, but we will consider $\mathcal{X}$ as the compact subset of $\mathcal{X}^n$), coupled with a measure μ. A kernel defines an integral operator i.e. a kernel is a function k of pairs $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which defines an integral operator which takes as input a function f, and returns a new function $T_k(f)$, defined as*

$$(T_K f)(\boldsymbol{x}) = \int_{\mathcal{X}} k(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) d_\mu(\boldsymbol{y}) \tag{9.23}$$

A kernel is positive semidefinite if, for all $f \in L_2(\mathcal{X},\mu)$

$$\int_{\mathcal{X}\times\mathcal{X}} k(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{x}) f(\boldsymbol{y}) d_\mu(\boldsymbol{x}) d_\mu(\boldsymbol{y}) \geq 0 \tag{9.24}$$

The Gram matrix of a symmetric kernel is symmetric $k(\boldsymbol{x},\boldsymbol{y}) = k(\boldsymbol{y},\boldsymbol{x})$

They inherit properties from matrices: they have eigenvalues and eigenfunction.
An eigenfunction ϕ with eigenvalue λ of the kernel k satisfies

$$\int k(\mathbf{x},\mathbf{y})\phi(\mathbf{x}) d_\mu(\mathbf{x}) = \lambda\phi(\mathbf{y}) \tag{9.25}$$

There are possible infinite eigenfunctions, and we can order them by their eigenvector, and choose them so that they are orthonormal between them. They allow us to decompose the kernel into a series of functions, multiplied by their eigenvector (Mercer's theorem guarantees it under certain conditions).

Definition 38. ***Reproducing kernel hilbert space****: an Hilbert space with the property of having a special function inside, the kernel, that belongs to a set (the kernel is symmetric, we fix an argument and variate the other, the choice is indifferent), and have the reproducing property, i.e. when we take the inner product of the Hilbert space between a function f and the kernel k (with one fixed argument value $\boldsymbol{x}$), what we get is the function f evaluated on $\boldsymbol{x}$.*

For each kernel there is a Hilbert space which has this property.
If we take a RKHS, get its kernel, compute the eigenfunctions of our kernel (which can be infinity-dimensional), that we can use this eigenfunction to represent the elements of functions in our RKHS (so the series of sum representing the function using the eigenfunctions converges).

9.6 Examples of kernel functions

There are three main classes of kernels:

- Stationary kernel: function which are invariant to translations (depends on $\mathbf{x} - \mathbf{y}$)
- Isotropic kernel: invariant to rigid motions (depends on $\|\mathbf{x} - \mathbf{y}\|$)
- Dot-product kernel: invariant to rotation w.r.t the origin (depends on $\mathbf{x}^T\mathbf{x}$)

Another classification of kernel is according to how behave the function which are in the support of the GP in relation to continuity

- Continuity in mean square of a process f at $\mathbf{x}$: when we take $f(\mathbf{x})$ and we approach it, than $f(\mathbf{x}_k$ will converge to $f(\mathbf{x}))$

$$\mathbb{E}[\|f(\mathbf{x}_k) - f(\mathbf{x})\|^2] \xrightarrow[\mathbf{x}_k \to \mathbf{x}]{} 0 \tag{9.26}$$

  It is a strong notion of convergence, and is the one we are going to use.

- If a kernel is stationary, or isotropic, this corresponds to continuity in 0 (so we only have to check continuity there).
- If k is $2kth$ differentiable, f is kth differentiable.

The **Gaussian kernel** is defined as

$$k(\mathbf{x}, \mathbf{y}) = \alpha \exp\Big[- \frac{\|\mathbf{x} - \mathbf{y}\|^2}{\lambda^2} \Big] \tag{9.27}$$

If λ is small, we are going to give a value sensibly far from 0 only at points which are close to the one we are evaluating. So λ is a sort of radius in our space where the input points are significantly impacting our prediction of point x.
The gaussian kernel is isotropic, and not only gives smooth functions, but in compact sets enjoy the property of universality: we can approximate arbitarly well any continuous or measurable function.
The **Automatic-Relevance Detection Gaussian Kernel** drop the isotropicity of the kernel: instead of having 1 λ, we have a single $\langle$ for every coordinates, and what we do is

rescale each coordinate (using then a sort of rescaled norm). So discarding some λ's, we are keeping only the once which are relevant to our learning problem.

$$k(\mathbf{x}, \mathbf{y}) = \alpha \exp\Big[- \sum_j \frac{|x_j - y_j|^2}{\lambda_j^2} \Big] \tag{9.28}$$

Matern Kernel is defined as

$$k_{\text{Matern}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \Big(\frac{\sqrt{2\nu} r}{l} \Big)^{\nu} K_\nu \Big(\frac{\sqrt{2\nu} r}{l} \Big) \tag{9.29}$$

Has the property that the parameter ν regulates the differentiability of the kernel.
The **Polynomial kernel** is a dot-product kernel defined, for p integer

$$k(\mathbf{x}, \mathbf{x'}) = (\mathbf{x}^T \mathbf{x'}) p \tag{9.30}$$

Practically corresponds to a finite dimension RKHS, spanned by all the monomies of degree p in our variable. This allows us to consider all of them without actually listing them (a lot of work!)

9.7 Hyperparameter optimization

We want to determine the hyperparameter of the kernels. The standard way in a bayesian approach to fundamentally find the hyperparameter is to compute the marginal likelihood, and optimise it (the same prerequisites of bayesian regression have to hold). It works well if we can identify an analytical expression for the marginal likelihood (in this case, we can)

$$\mathcal{L} = \log p(\mathbf{y}|x) = \log \int p(\mathbf{f}|x) p(\mathbf{y}|\mathbf{f}, x) d\mathbf{f} \tag{9.31}$$

We can easily write down because we already know $p(\mathbf{y}—\mathbf{f, x})$, which is a Gaussian distribution with mean 0, and standard deviation given by $K + \sigma^2 I$.
We just need to get the logarithm of the density of this gaussian distribution,

$$\mathcal{L} = \underbrace{-\frac{1}{2} \mathbf{y}^T (K + \sigma^2 I)^{-1} \mathbf{y}}_{\text{Data fit}} \underbrace{-\frac{1}{2} \log \big| (K + \sigma^2 I) \big|}_{\text{Complexity penalty}} \underbrace{-\frac{N}{2} \log 2\pi}_{\text{A constant}} \tag{9.32}$$

we see it is composed by three terms.
To optimise, we can compute the gradient, which is a pretty simple expression

$$\frac{\partial}{\partial \theta_j} \log p(\mathbf{y}|x, \boldsymbol{\theta}) = \frac{1}{2} tr\Big((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - K^{-1}) \frac{\partial K}{\partial \theta_j} \Big) \qquad \boldsymbol{\alpha} = K^{-1} \mathbf{y} \tag{9.33}$$

We can take a different root for hyperparameter optimization, which is essentially going fully bayesian: place a prior over hyperparameters, hyperprior the hyperparameters, then try to get a posterior. In this way however we immediately loose the analytic tractability of GP.
Adding an extra element can be used to model a non-constant prior mean. Enforcing

the mean to 0 is typically done manipulating the observed data (subtracting the mean from the data). It is like using a linear model for the prior mean. The strategy is model the function we are targeting as a function describing linearly the mean, and a term accounting for the residuals, and we model the residuals as a GP

$$g(\mathbf{x}) = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^T \boldsymbol{\beta} \qquad f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x'})) \tag{9.34}$$

We can be bayesian by trying to place a prior of the weights β, and $h(\mathbf{x})$ is the basis functions evaluated on the input points. So we can place a gaussian prior over β and we have a GP prior over the residual, combine the two together, and what we get is that we are going to have still a gaussian process for the prior $g(\mathbf{x})$

$$g(\mathbf{x}) \sim \mathcal{GP}(\mathbf{h}(\mathbf{x})^T \mathbf{b}, k(\mathbf{x}, \mathbf{x'}) + \mathbf{h}(\mathbf{x})^T B \mathbf{h}(\mathbf{x'})) \tag{9.35}$$

where b is the mean for β.
The posterior is once again a GP.

9.8 GP classification

Lets say we are dealing with a 2-class classification problem. We need to model the probability of

$$p(c_1|x) = \sigma(f(x)) = \pi(x) \qquad f \sim \mathcal{GP}(\mu, k) \tag{9.36}$$

Observations are (x_n, y_n) with $y_n \in \{0, 1\}$, and $p(y|f(x)) = Bernoulli(\sigma(f(x))$. Lastly, $(\underline{x}, \underline{y})$ is our dataset and $f(\underline{x}) = \underline{f}$ is the vector $f(x_1), ..., f(x_N)$ where N is the number of observations.
Function $f(x)$ is known as **latent** (or nuisance) function. We notice that $\pi(x) = \sigma(f(x))$ is a random function.
Our aim is to find the posterior distribution, then make prediction with it.
The first thing we want to evaluate is

$$p(f(\underline{x}|\underline{y}) = \frac{p(\underline{y}|f(\underline{x})p(f(\underline{x}))}{p(y)} \tag{9.37}$$

That we have to make a prediction, so we choose the point $x^*, f(x^*)$, and we want to know the value of

$$p(f(x^*)|\underline{y}) = \int p(f(x^*)|f(\underline{x}))p(f(\underline{x})|\underline{y})df(\underline{x}) \tag{9.38}$$

Once we know it, we know the model as a random function of

$$p(\pi(x^*)|\underline{y}) = p(\sigma f(x^*)|\underline{y}) \tag{9.39}$$

And we can compute the expectation by integrating

$$\pi^* = \int \sigma(f(x^*))p(f(x^*)|\underline{y})df(x^*) \tag{9.40}$$

The problem is that, because of the likelihood $p(\underline{y}|f(\underline{x}))$, which is a product of Bernoulli, there is no more conjugally, so the integrals are no more algorithmically tractable.

What we do is to use Laplace approximation on the posterior of the observations ($p(f(x)|\underline{x}, \underline{y})$. First of all we have to compute the maximum a posteriori (optimising its logarithm) and we get an expression for log-posterior unnormalized, compute the first and second order derivatives (using Newton-Rapson scheme). The concept is: write down the unnormalised log posterior, and find the maximum.
The Laplace approximation of the predictive distribution is going to be gaussian, and we can practically get directly this Laplace approximation of the MAP and obtain an expression for the mean and the variance.

For GP we usually take a different kind of approximation, which belongs to the class of variational inference (or approximation), which roughly speaking works as: we take the Kullback Leibler divergence between the true function and an approximating function, and optimise the parameters that bring us the minimize Kullback Leibler divergence. We construct iteratively a gaussian approximation of the posterior, we iterate over observations the following operations: the posterior depends proportionally on the product of the likelihood for all the observations so, once we have our gaussian approximation of the posterior, we pick one of the observations, we factor out the term for $i - th$ likelihood, so we have now a sort distribution for all observation but i, than we multiply by the exact likelihood of observation i, we get something which is non-gaussian (the likelihood of observation i is non-gaussian), but we do a gaussian approximation by matching the sufficient statistic (matching mean and variance) of this distribution we obtained.
This method converges faster than the Laplace optimisation.

Chapter 10

Expectation Maximisation

10.1 Introduction

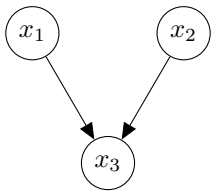
Expectation Maximisation is a methodology to perform maximum likelihood inference in scenarios where computing the likelihood explicitly may not be possible because of the presence of latent (unobserved) variables.
Lets focus for a moment on the learning of bayesian networks.
Bayesian Network is a description of a way to factorize a probability distribution

$$p(x) = \prod_i p(x_i|pa(x_i)) \tag{10.1}$$

where $pa(x_i)$ is the probability of its parents. We are still missing how to learn bayesian network from data: whenever we do inference of some sort, we are assuming that the distributions of the factor are known. But this is typically not the case, usually we do not have the parameters of the distributions, but instead we want to assess them at posteriori by running some estimations from data.
Each factor usually depends also on some parameter θ, and the goal is to estimate using maximum likelihood θ from data

$$p(x_i|pa(x_i), \theta) \tag{10.2}$$

In case in which all variables are observed, it is relatively easy: the structure of bayesian network already provides a factorization, and we can treat each of this factor independently. In particular we can learn a model of the distribution of x_i for each possible values of the tuple of its parents. If the variables are discrete, is like learning each row of a table independently, if the variables are continuous, there could already be a supposed dependency on the parameters that governs the continuous distribution of x_i on the value of its parent node, so we only need to learn the additional parameters that governs this dependency.
Lets make an example



$$p(x_1, x_2, x_3) = p(x_3|x_1, x_2)p(x_1)p(x_2) \qquad x_i \in \{0, 1\} \tag{10.3}$$

We want to learn the factor $p(x_3|x_1, x_2)$. We could consider all the possible values of the variables in the conditional set. So

$$\begin{aligned} p(x_3 = 1|x_1 = 0, x_2 = 0) &= \theta_{00} \\ p(x_3 = 1|x_1 = 0, x_2 = 1) &= \theta_{01} \\ p(x_3 = 1|x_1 = 1, x_2 = 0) &= \theta_{10} \\ p(x_3 = 1|x_1 = 1, x_2 = 1) &= \theta_{11} \end{aligned}$$

This are the four parameter we want to learn by maximum likelihood. We can do it by counting the instance, i.e. for the first case, the maximum likelihood estimate would be

$$\frac{\#(x_1 = 0, x_2 = 0, x_3 = 1)}{\#(x_1 = 0, x_2 = 0)} \tag{10.4}$$

This methodology is fine, but what happens if we are not observing one of the variables? We cannot do the maximum likelihood estimation!
Lets switch to a more general scheme. Consider that we have a probability distribution $p(x, z|\theta)$ on some number of variables x observed, and some variables z which are latent. We want to estimate θ by maximum likelihood. The only thing we can evaluate is a sort of marginal likelihood

$$p(x|\theta) = \sum_z p(x, z|\theta) \tag{10.5}$$

But this summation is likely to became intractable very quickly.
Even more problematic is when we have $x_1, ..., x_N$ observations and $z_1, ..., z_N$ latent states of observations, than the typical likelihood became

$$p(\underline{x}, \underline{z}|\theta) = \prod_n p(x_n, z_n|\theta) \tag{10.6}$$

What we typically do is to work with the logarithm

$$\log p(\underline{x}, \underline{z}) = \sum_n p(x_n, z_n) \tag{10.7}$$

But in this case

$$\log p(\underline{x}|\theta) \neq \sum \log p(x_n|\theta) \tag{10.8}$$

10.2 Evidence Lower Bound

Once again, we have a model of probability $p(x, z)$, with variables x observed $(x_1, ..., x_N)$, and variables z latent $(z_1, ..., z_N)$. Our goal is to compute

$$p(x|\theta) = \sum_z p(x, z|\theta) \tag{10.9}$$

Now we are going to consider the maximum likelihood problem of finding $\arg\max_\theta p(\underline{x}|\theta)$, which is the target function to optimise (usually intractable). Whenever we consider the joined distribution

$$p(x, z|\theta) = p(x|\theta)p(z|x, \theta) \tag{10.10}$$

we can always factorize it. The conditional distribution $p(z|x, \theta)$ play an important role. We are going to approximate this distribution, with what it is known as **variational approximation** $q(z)$ of $p(z|x, \theta)$. Now $p(z|x, \theta)$ is the best guess on the latent variable, given the observation that we have, and a certain value of the parameter θ.
We focus our attention on the notion of Kullback Leibler divergence between a distribution q, a the true distribution p

$$\begin{aligned} KL[q\|p] =& KL[q(z)\|p(z|x, \theta)] = \mathbb{E}_{q(z)}\Big[-\log \frac{p(z|x, \theta)}{q(z)}\Big] \\ =& -\sum_z q(z)\Big[\log \frac{p(z|x, \theta)}{q(z)} + \log p(x|\theta) - \log p(x|\theta)\Big] \\ =& \underbrace{-\sum_z q(z) \log \frac{p(x, z|\theta)}{q(z)}}_{\mathcal{L}(q|\theta)} + \log p(x|\theta) \end{aligned}$$

So, to summarise,

$$\log p(x|\theta) = \mathcal{L}(q, \theta) + KL[q(z)\|p(z|x, \theta)] \tag{10.11}$$

where $\mathcal{L}(q, \theta)$ is the **evidence lower bound** (ELBO).
Because the KL divergence of any two distribution is always positive, than

$$\mathcal{L}(q, \theta) \leq \log(x|\theta) \tag{10.12}$$

so it is a lower bound to the likelihood that we want to optimize.
We have introduced the distribution q (our variational distribution) and still have the parameter θ, which rule the distribution over full joint model. This decomposition make sense whenever we assume that the joint probability distribution of $p(x, z|\theta)$ is known and intractable.

10.3 Expectation Maximization Algorithm

To recap

$$\mathcal{L}(q, \theta) = \underbrace{\mathbb{E}_q[\log p(\underline{x}, z|\theta)]}_{\text{Energy term}} + \underbrace{\mathbb{E}_q[-\log q(z)]}_{\text{Entropy term } H(q)} = \mathbb{E}_q\Big[\log \frac{p(x, z|\theta)}{q(z)}\Big] \tag{10.13}$$

Our goal is to compute

$$\theta_{ML} = \arg\max_{\theta} \log p(\underline{x}|\theta) \tag{10.14}$$

but this is intractable, so we maximise $\mathcal{L}(q, \theta)$.
E-step: maximise $\mathcal{L}(q, \theta)$ with respect to q, with θ fixed to a value θ_{old}.
We see that q is maximised iff $KL[q\|p(z|x, \theta_{old}] = 0$. This happens iff $q_{new}(z) = q(z) = p(z|\underline{x}, \theta_{old})$.
Practically we compute this conditional distribution of q for θ fixed then, once we fix the z, we need also to evaluate the energy term with respect to q.
We then compute $\mathbb{E}_{q_{new}}[\log p(x, z|\theta)]$. We can easily notice, iff $x_1, ..., x_N$ are i.i.d.

$$\begin{aligned} p(\underline{z}|\underline{x}, \theta) =& \prod_{i=1}^{N} p(z_i|x_i, \theta) \\ =& \frac{p(\underline{x}, \underline{z}|\theta)}{\sum_z p(\underline{x}, \underline{z}|\theta)} = \frac{\prod p(z_i, x_i|\theta)}{\sum_z \prod p(z_i, x_i|\theta)} \\ =& \frac{\prod p(z_i, x_i|\theta)}{\prod \sum_{z_i} p(z_i, x_i|\theta)} \end{aligned}$$

M-step: maximise $\mathcal{L}(q, \theta)$ keeping q fixed to $q_{new} = p(z|\underline{x}, \theta_{old})$, which is equal to maximise $\mathbb{E}_{q_{new}}[\log p(z, \underline{x}|\theta)]$.
Then we have $\theta_{new} = \arg\max_\theta \mathbb{E}_{q_{new}}[\log p(\underline{x}, z|\theta)]$
Iterate E and M steps until convergence (when log-like or $\|\theta_{old} - \theta_{new}\| < \epsilon$). EM converges to a local optimum of $\log p(\underline{x}|\theta)$.

10.4 Mixtures of Gaussians

10.5 EM for Bayesian Networks

The idea is that doing EM on bayesian networks is not that complicated, and the factorization of the joint probability distribution will essentially allows us to split the maximisation step into small tractable sub-problem.
Lets consider a bayesian network on a certain set of variables $x = (v, z)$ (v for visible variables, z for latent ones), and the set of parameters $\theta = (\theta_i)_{i=1,...,m}$such that

$$p(x) = \prod_i p(x_i|pa(x_i), \theta_i) \tag{10.15}$$

That we can alternatively write as

$$p(x) = p(v, z|\theta) \tag{10.16}$$

Now we have to consider the computation of (for a fixed θ)

$$p(z|v = \hat{v}, \theta) \tag{10.17}$$

This is a conditional distribution of our network, and it is something typically computable within bayesian network in a reasonable way (particularly for special case in which we

can use belief propagation algorithms). The expectation step reduce to the computation of this distribution, for all possible observations of v.

$$\underline{v} = (v_1, ..., v_N) \qquad \text{observations of variables } v \tag{10.18}$$

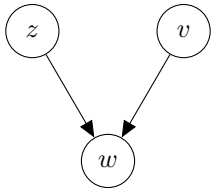
We can essentially define the distribution $q^n(z) = p(z|v_n, \theta)$ for observation n.
We can also extend it to a distribution over all variables x, $q^n(x) = p(z|v_n, \theta)\delta(v, v_n)$.
For the M-step we need the energy, in this case is (and using the factorization of the bayesian network)

$$\sum_n \mathbb{E}_{q^n}[\log p(v_n, z_n|\theta)] = \sum_n \sum_i \mathbb{E}_{q^n}[\log p(x_i^n|pa(x_i^n|\theta_i)]$$

we can optimize

$$\sum_n \mathbb{E}_{q^n}[\log p(x_i|pa(x_i|\theta_i)] \tag{10.19}$$

over θ_i of each i. The factorization allows us to perform the maximisation step of the expectation maximisation on the single parameter governing the single factors, and using only "local" information around the factor.
Lets see an example



With $z, v, w \in \{0, 1\}$. We would like to know

$$\begin{aligned} &p(z = 1) = \theta_z \\ &p(v = 1) = \theta_v \\ &p(w = 1|z = a, v = b) = \theta_{wab,\ a,b\in\{0,1\}} \end{aligned}$$

The E-step for each observation corresponds to finding

$$q^n(z) = p(z|v = v_n, w = w_n, \theta) \qquad q^n(x) = p(z|v = v_n, w = w_n, \theta)\delta(v, v_n)\delta(w, w_n) \tag{10.20}$$

One just has to compute using standard algorithm for bayesian network the probability for z in the needed state, for all possible values of v_n, w_n.
Now the energy term, we have one for each factor. Lets write down some of them

$$\sum_n \mathbb{E}_{q^n}[\log \underbrace{p(z^n|\theta_z)}_{\theta_z \text{ if } z^n=1}] = \sum_n \log \theta_z q^n(z = 1) + \log(1 - \theta_z) \cdot q^n(z = 0)$$

If we perform maximum likelihood estimation, what we are going to obtain is

$$\theta_z = \frac{\sum_n q^n(z = 1)}{\sum_n q^n(z = 1) + \sum_n q^n(z = 0)} = \frac{1}{N} \sum_n q^n(z = 1) \tag{10.21}$$

And the energy term corresponding to the factor w

$$\sum_n \mathbb{E}_{q^n}[\log p(w_n|z, v_n, \theta_w)]$$

we can sort out the case $z = 0, v = 1$ (θ_{w01}:

$$\sum_{n:w_n=1,v_n=1} q^n(z=0)\log\theta_{w01} + \sum_{n:w_n=0,v_n=1} q^n(z=0)\log(1-\theta_{w01})$$

If we find the maximum, we get the estimate

$$\theta_{w01} = \frac{\sum_n \mathbb{I}(w_n=1)\mathbb{I}(v_n=1)q^n(z=0)}{\sum_n \mathbb{I}(w_n=1)\mathbb{I}(v_n=1)q^n(z=0) + \sum_n \mathbb{I}(w_n=0)\mathbb{I}(v_n=1)q^n(z=0)} \tag{10.22}$$

10.6 EM for Hidden Markov Models

The algorithm for expectation maximisation for hidden markov model is know as the **Baum-Welch algorithm**. We give as defined the bayesian network described at 7.1.
We have our observation set $\underline{x} = x^1, ..., x^N$.
E-step: we need to compute

$$q^n(z) = \underbrace{p(z|x^n, \theta)}_{\text{Message passing}} \quad \forall n \tag{10.23}$$

Which can be done by the standard inference of bayesian network.
M-step: we want the energy function

$$\mathbb{E}(\theta) = \sum_{n=1}^{N}\Big[\sum_{k=1}^{K} q^n(z_{1k})\ln\pi_k + \sum_{i=2}^{M}\sum_{j,k=1}^{K} q^n(z_{i-1\,j}, z_{ik})\ln Ajk + \sum_{i=1}^{M}\sum_{k=1}^{K} q^n(z_{ik})\ln p(x_1^n|\phi_k)\Big] \tag{10.24}$$

We can show that the solution on the M step for π_k is

$$\pi_k = \frac{\sum_n q^n(z_{1k}}{\sum_j \sum_n q^n(z_{1j})} \tag{10.25}$$

Chapter 11

Variational Inference

11.1 Introduction

Lets consider a model $p(x, z)$ which represent a joint distribution between x and z, with x observable variables, and z latent variables.
Latent variables are devided between:

- Local latent variables z_n.
- Global latent variables θ

So z is

$$z = (z_1, ..., z_N, \theta) \tag{11.1}$$

We have observation $\underline{x} = x_1, ...x_N$, and we are interested in

$$p(z|\underline{x}) \qquad \text{Posterior}$$
$$p(\underline{x}) = \int p(\underline{x}, z) \qquad \text{Model evidence}$$

As in expectation maximisation, here we are also interested in ELBO

$$p(\underline{x}) = \mathcal{L}(q) + \underbrace{KL[q\|p]}_{KL[q(z)\|p(z|\underline{x})]} \tag{11.2}$$

We can write explicitly

$$\mathcal{L}(q) = \int q(z)[\log p(\underline{x}, z) - \log q(z)]dz = \mathbb{E}_q[\log p(\underline{x}, z) - \log q(z)]$$
$$KL[q(z)\|p(z|\underline{x})] = -\int q(z)[\log p(z|\underline{x}) - \log q(z)]dz$$

where q is the variational distribution. We are going to use the distribution q to approximate the posterior distribution. The goal is to find the best q that approximate $p(z|\underline{x})$. Lets focus on the decomposition: as the observations are fixed, so also is $p(\underline{x})$, which is essentially a number. This number decompose in two terms depending on q: the lower

bound, and the Kullback Leibler divergence (which is always positive). So the lower bound is always smaller or equal to $p(\underline{x})$, and its maximised when the Kullback Leibler divergence is equal to 0

$$\text{Minimize } KL[q\|p] \iff \text{Maximise ELBO } \mathcal{L}(q) \tag{11.3}$$

$\mathcal{L}(q)$ is maximum when $KL[\|p] = 0$ of and only if $q(z) = p(z|\underline{x})$. But in most of the cases $p(z|\underline{x})$ is intractable.
The solution is to restrict q to a tractable family of distributions. The optimal $q(z)$ likely is such that its $KL[q\|p] > 0$.
Variational inference is to solve an optimization problem to maximize $\mathcal{L}(q)$ in a suitably restricted space of variational distribution $q \in Q$.
Scenario: Q is a set of parametric distributions $q(z|\lambda)$ for some parameters $\lambda \in \mathbb{R}^k$. Then we need to find $\arg\max_\lambda \mathcal{L}(\lambda)$. This is a highly non-linear non-convex optimization.

$$q(z|\lambda) = q(\theta|\lambda_\theta) \prod_i q(z_i|\lambda_i, \theta) \qquad \lambda = (\lambda_\theta, \lambda_1, ..., \lambda_N) \tag{11.4}$$

11.2 Mean Field Variational Inference

We need to compute or approximate the conditional distribution by a member of the variational distribution q, i.e. $p(z|\underline{x}) \log q(z)$, and we make the assumption that $z = (z_1, ..., z_M)$ can be decomposed in M different blocks.
This assumption is known as the **Mean field assumption**

$$q(z) = \prod_{i=1}^{M} \underbrace{q(z_i)}_{q_i \text{ for simplicity}} \tag{11.5}$$

Lets start writing down the lower bound

$$\begin{aligned}
\mathcal{L}(q) =& \mathbb{E}_q[\log p(x,z) - \log q(z)] \\
=& \int \prod_i q_i [\log p(x,z) - \sum_i \log q_i] dz \qquad \text{Factor out the terms depending on } q_j \\
=& \int q_j \underbrace{\Big[\int \log p(x,z) \prod_{i \neq j} q_i dz_i \Big]}_{\mathbb{E}_{i\neq j}[\log p(x,z)]} dz_j - \int q_j \log q_j dz_j + const
\end{aligned}$$

We are going to use this expectation to find a function $\widetilde{p}$, defined by

$$\log \widetilde{p}(x, z_j) = \mathbb{E}_{i\neq j}[p(x,z)] \tag{11.6}$$

So the above expression of the ELBO becomes

$$= \mathbb{E}_{i\neq j}[\log \widetilde{p}(x, z_j) - \log q_j] + const \tag{11.7}$$

So, to recap

$$\mathcal{L}(q_j) = KL[q_j \| \widetilde{p}(x, z_j)] + const \tag{11.8}$$

Which is maximised for q_j, with $i+j$ fixed, by assuming

$$q_j^*(z_j) = \widetilde{p}(x, z_j) \tag{11.9}$$

Hence $\log q_j^*(z_j) = \mathbb{E}_{i\neq j}[\log p(x,z)] + const$, which imply that

$$q_j^*(z_j) = \frac{\exp(\mathbb{E}_{i\neq j}[\log p(x,z)])}{\int \exp(\mathbb{E}_{i\neq j}[\log p(x,z_j)])dz_j} \tag{11.10}$$

If we can compute analytically this expectation $\mathbb{E}_{i\neq j}[\log p(x,z)]$. We initialize q_i, then cycle through q_j, optimized them in turn (coordinate gradient descend), and we do it until convergence. This is guaranteed because the kernel is convex on the q_j.

11.3 Example: Gaussian distribution

We are going to use variational inference to go from a gaussian, to a factorized gaussian. We start from

$$p(z) = \mathcal{N}(z|\mu, \Lambda^{-1}), \qquad \mu = \begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}, \Lambda \text{ precision matrix} \tag{11.11}$$

It is a two dimensional system $z = (z_1, z_2)$, our. We are going to assume that $q(z) = q(z_1)q(z_2)$.
We start from

$$\begin{aligned}\log q_1^*(z_1) =& \mathbb{E}_{z_2}[\log p(z)] + const\\ =& \mathbb{E}_{z_2}\Big[-\frac{1}{2}(z_1-\mu_1)^2\Lambda_{11} - (z_1-\mu_1)\Lambda_{12}(z_2-\mu_2)\Big] + const\\ =& -\frac{1}{2}(z_1-\mu_1)^2\Lambda_{11} - (z_1-\mu_1)\Lambda_{12}(\mathbb{E}[z_2]-\mu_2) + const\end{aligned}$$

so we end up in $q_1^*(z_1)$ is gaussian $\mathcal{N}(z_1|m_1, \Lambda 11^{-1}$ with $m_1 = \mu_1 - \Lambda_{11}^{-1}\Lambda_{12}(\mathbb{E}[z_2]-\mu_2)$. And by symmetry $q_2^*(z_2) = \mathcal{N}(z_2|m_2, \Lambda_{22}^{-1})$ with $m_2 = \mu_2 - \Lambda_{22}^{-1}\Lambda_{21}$. If we couple m_1 and m_2, one can obtain that

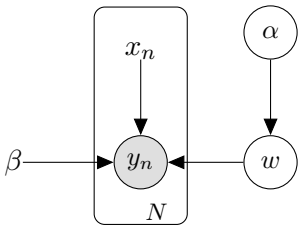
$$m_1 = \mathbb{E}[z_1] \Rightarrow m_1 = \mu_1 \quad m_2 = \mu_2 \tag{11.12}$$

11.4 VI with Direct and Reverse KL divergence

In this section we are going to compare what happens in the two dimensional gaussian we just explore when we take the problem of finding the variational distribution which is optimal respect to the direct Kullback Leibler from the actual condition distribution z given all the rest, to what happens when we try to optimize the reverse Kullback Leibler (i.e. the distribution from p to the conditional distribution q).
We are working with $q(z) = q(z_1)q(z_2)$: in the first approach we minimize $KL[q\|p]$, in the second $KL[p\|q]$. The second on is usually intractable, but in the specific case of gaussians, it is actually easier.
The approximation of the direct KL approx is called **zero forcing approximation scheme**: if $p(z) \approx 0$, than $q(z) \approx 0$, around the mode.
The approximation of the reverse KL approx is called **zero avoiding approximation scheme**: $p(z)$ is non-zero where $p(z)$ is non-zero. So we may have a situation in which the variational distribution overlaps different modes of the system.

11.5 Variational Linear Regression

Another application of mean field variational inference is the scenario of linear regression when we want to place a hyperprior distribution over the parameter α which regulate the variance of our prior.
Lets consider a problem represented in a bayesian network as



we are going to treat α as a parameter, but as a proper random variable, on which we are going to place an hyperprior on α, then consider what happens in term of posterior distribution and predictive distribution.
It is factorized as

$$p(y, w, \alpha) = p(y|w)p(w|\alpha)p(\alpha) \tag{11.13}$$

Lets write down what are this distribution

$$\begin{aligned}
p(y|w) &= \prod_{n=1}^{N} \mathcal{N}(y_n|w^T\phi(x_n), \beta^{-1}) \\
p(w|\alpha) &= \mathcal{N}(w|0, \alpha^{-1}I) \\
p(\alpha) &= Gamma(\alpha|a_0, b_0) = \frac{1}{\Gamma(a_0)} b_0^{a_0} \alpha^{a_0-1} e^{-b_0\alpha}
\end{aligned}$$

Our goal is to get $p(w, \alpha|y)$, and we are going to use a variational distribution $q(w, \alpha) = q(w)q(\alpha)$.
Lets start with

$$\begin{aligned}
q^*(\alpha) =& \mathbb{E}_w[\log p(y, w, \alpha)] + const \\
=& \log p(\alpha) + \mathbb{E}_w[\log p(w|\alpha)] + const \\
=& (a_0 - 1)\log\alpha - b_0\alpha + \frac{M}{2}\log\alpha - \frac{\alpha}{2}\mathbb{E}[w^T w] + const
\end{aligned}$$

So

$$q^*(\alpha) = gamma(\alpha|a_N, b_N) \qquad a_N = a_0 + \frac{M}{2} \quad b_N = b_0 + \frac{1}{2}\mathbb{E}_q[w^T w] \tag{11.14}$$

So we can get a close form for the distribution of $q^*(\alpha)$. Now we notice that

$$\begin{aligned}
\log q^*(w) =& \log p(y|w) + \mathbb{E}_\alpha[\log p(w|\alpha)] + const \\
=& -\frac{\beta}{2}\sum_{n=1}^{N} w^T\phi(x_n) - y_n]^2 - \frac{1}{2}\mathbb{E}[\alpha]w^T w + const
\end{aligned}$$

compleating the square, we get that

$$q^*(w) = \mathcal{N}(w|m_N, s_N) \qquad m_N = \beta s_N \phi^T y \quad s_N = (\mathbb{E}[\alpha] I + \beta \phi^T \phi)^{-1}$$

So we can again solve the problem, because we to compute the optimal factor for α we need to know the expectation of $w^T w$, and to compute the optimal factor for w, we need the expectation for α. One takes the $Gamma$, the other the gaussian distribution, so we can easily work out the solution in a iteratively way.
We know the expectation is

$$\mathbb{E}[\alpha] = \frac{a_N}{b_N} \quad \mathbb{E}[w^T w] = m_N^T m_N + Trace(s_N) \tag{11.15}$$

We can also in principle compute $\mathcal{L}(q) \approx p(y)$ for model comparison (approximate model evidence).

11.6 Black Box Variational Inference

Discuss a method which can be applied more generally than Mean Field Variational Inference. With the MFVI, if we cannot compute analytically the expectation on all factors (apart from the one considered at the moment), the method is not usable.
In the Black Box Variational Inference instead what we are doing is doing stochastic gradient descent on the ELBO by suitably sampling the gradient of the ELBO with respect to the variational parameters.
In this case we are typically working on a variational distribution which is parametric $q(z|\lambda)$.Its lower bound is

$$\mathcal{L}(\lambda) = \mathbb{E}_{q(z|\lambda)}[\log p(x, z) - \log q(z|\lambda)] \tag{11.16}$$

We want to estimate $\nabla_\lambda \mathcal{L}(\lambda)$. This is the gradient of an expectation. The only problem is that the distribution we are averaging is actually a distribution that depends on the same parameters on which we are trying to compute the gradient.
Solution 1: Reparameterization trick

$$if \quad z = g_\nu(\epsilon), \quad \epsilon \sim \hat{q}(\epsilon) \text{independent of } \lambda \text{ and fixed} \tag{11.17}$$

We can write

$$\mathcal{L}(\overline{\lambda}) = \mathbb{E}_{\hat{q}(\epsilon)}[\log p(x, g_\nu(\epsilon) - \log q(g_\nu(\epsilon)|\lambda)] \tag{11.18}$$

essentially replaced z with $g_\nu(\epsilon)$. But no the expectation does not depend on λ and can be sampled easily.
In particular

$$\nabla_{\overline{\lambda}} \mathcal{L}(\overline{\lambda}) = \mathbb{E}_{\hat{q}(\epsilon)} \underbrace{[\nabla_{\overline{\lambda}} \log p(x, g_\nu(\epsilon)) - \nabla_{\overline{\lambda}} q(g_\nu(\epsilon)|\lambda)]}_{G(\epsilon)} \tag{11.19}$$

so we sample $\epsilon_j \sim \hat{q}(\epsilon)$, and our estimate of the gradient is

$$\nabla_{\overline{\lambda}} \mathcal{L}(\lambda) = \frac{1}{S} \sum_{s=1}^{S} G(\epsilon_i) \tag{11.20}$$

on which we use stochastic gradient descent.I q is not gaussian, this solution can be complicated, or even not possible.
Solution 2: non reparameterizable $q(z|\lambda)$. What we do is

$$\begin{aligned}\nabla_\lambda \mathcal{L}(\lambda) =& \nabla_\lambda \mathbb{E}_{q(z|\lambda)}[\log p(x,z) - \log q(z|\lambda)] \\ =& \nabla_\lambda \int q(z|\lambda)[\log p(x,z) - \log q(z|\lambda)]dz \\ =& \int \nabla_\lambda [\underbrace{\log p(x,z)}_{\nabla_\lambda p(x,z)=0} - \log q(z,\lambda)]q(z|\lambda)dz + \int \nabla_\lambda q(z|\lambda)[\log p(x,z) - \log q(z|\lambda)] \\ =& -\underbrace{\mathbb{E}_q[\nabla_\lambda \log q(z,\lambda)]}_{1} + \underbrace{\int \nabla_\lambda q(z|\lambda)[\log p(x,z) - \log q(z|\lambda)]dz}_{2}\end{aligned}$$

And we calculate the terms as

$$\begin{aligned}1 &: \mathbb{E}_q\Big[\frac{\nabla_\lambda q(z|\lambda)}{q(z|\lambda)}\Big] = \int \nabla_\lambda q(z|\lambda)dz = \nabla_\lambda \int q(z,\lambda)dz = \nabla_\lambda 1 = 0 \\ 2 &: \nabla_\lambda \log q(z|\lambda) = \frac{\nabla_\lambda q(z|\lambda)}{q(z|\lambda)} \Rightarrow \nabla_\lambda \log q(z|\lambda) \cdot q(z)\end{aligned}$$

This means that we can write down the gradient as

$$\begin{aligned}\nabla_\lambda \mathcal{L}(\lambda) =& \int q(z|\lambda)\nabla_\lambda \log q(z|\lambda)[\log p(x,z) - \log q(z|\lambda)] \\ =& \mathbb{E}_q[\nabla_\lambda \log q(z|\lambda)[\log p(x,z) - \log q(z|\lambda)]]\end{aligned}$$

Now we can sample $z_s \sim q(z|\lambda)$, and we estimate

$$\nabla_\lambda \mathcal{L}(\lambda) \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log(z_s|\lambda)[\log p(x,z_s) - \log q(z_s|\lambda)] \tag{11.21}$$

but the variance of this estimation is very high. A control of it can be done using two strategies, that we will see in the next sections.

11.7 Variance Reduction: Rao-Blackwellization

Lets consider x, y random variates, and some function $J(x,y)$, so $\mathbb{E}[J(x,y)]$ is our target. Define

$$\hat{J}(x) = \mathbb{E}_y[J(x,y)|x] \tag{11.22}$$

we are essentially marginalizing y, for any given value of x. We know that

$$\mathbb{E}_x[\hat{J}(x)] = \mathbb{E}[J(x,y)] \tag{11.23}$$

also

$$var[\hat{J}(x)] = var[J(x.y)] - \mathbb{E}[(J(x,y) - \hat{J}(x))^2] < var[J(x,y)] \tag{11.24}$$

we are going to consider a mean field factorization of $q(z|\lambda) = \prod_{i=1}^{N} q(z_i|\lambda_i)$.
Because a single z_i depends only on λ_i, we are going to consider the gradient with respect

only to λ_i.
We are going to consider $q_{(i)}$ which is the marginal of $q(z|\lambda)$ on the the markov blanket $z_{(i)}$ of z_i (in p).
And $p_{(i)}(x, z_{(i)}$ the product to factors $p(x, z)$ depending on $z_{(i)}$.
When we are tring to take the marginalization

$$\mathbb{E}_{q\setminus i}[\nabla_{\lambda_i}\mathcal{L}(z_{(i)}] = \hat{\nabla}_{\lambda_i}\mathcal{L}z_{(i)} = \mathbb{E}[\nabla_{\lambda_i}\log q(z_i|\lambda_i)(\log p_i(x, z_i) - \log q(z_i|\lambda_i))] \tag{11.25}$$

We need now to consider samples from $z_s \sim q_{(i)}(z|\lambda)$

11.8 Variance Reduction: Control Variates

We have a certain function f and we want to estimate $\mathbb{E}_q[f]$. Instead we are going to estimate $\mathbb{E}_q[\hat{f}]$ such that $\mathbb{E}_q[\hat{f}] = \mathbb{E}[f]$ and $var_q[\hat{f}] < var[f]$.
We choose a function $h : \mathbb{E}[h] < \infty$, and define

$$\hat{f}_a(z) = f(z) - a(h(z) - \mathbb{E}[h(z)]) \tag{11.26}$$

The a^* that minimizes variace is

$$a^* = \frac{cov(f, h)}{var(h)} \tag{11.27}$$

The larger the covariance, the larger the reduction in variance.
In our scenario

$$\begin{aligned} f_i(z) &= \nabla_{\lambda_i}\log q(z_i|\lambda_i)[\log p_i(x, z_{(i)}) - \log(z_i\lambda_i)] \\ h_i(z) &= \underbrace{\nabla_{\lambda_i}\log q(z_i|\lambda_i)}_{\mathbb{E}[\cdot]=0} \end{aligned}$$

We want to estimate a

$$\hat{a}_i = \frac{\sum_{d=1}^{n_i} \hat{cov}(f_i^d, h_i^d)}{\sum_{d=1}^{n_i} \hat{var}(h_i^d)} \tag{11.28}$$

with $\#z_{(i)=n_i}$. And the final object we need to consider is

$$\hat{\nabla}_{\lambda_i} f = \frac{1}{S}\sum_{s=1}^{S} \nabla_{\lambda_i}\log q(z_s|\lambda_i)[\log p_i(x, z_s) - \log q_i(z_s|\lambda_i) - \hat{a}_i^*] \tag{11.29}$$

Sampling $z_s \sim q_{(i)}(z|\lambda)$.