UniTs - University of Trieste

Faculty of Scientific and Data Intensive Computing

Department of mathematics informatics and geosciences

Probabilistic Machine Learning

Lecturer:
Prof. Luca Bortolussi

Authors:
Andrea Spinelli
Christian Faccio

May 20, 2025

github.com/Spina02 andreaspinelli2002@gmail.com

Preface

As a student of Scientific and Data Intensive Computing, I've created these notes while attending the **Probabilistic Machine Learning** course.

This collection introduces ideas and instruments of machine learning from a probabilistic perspective—an approach that continues to grow in importance and serves as the foundation for many recent successes in generative Artificial Intelligence. The notes begin with a discussion on the fundamental role of probability and mathematics in machine learning, providing a framework for understanding ML concepts through this powerful lens.

Throughout the course, we will focus on the following topics:

- Basics of probability and probabilistic inference
- Probabilistic formulation of learning (Empirical Risk Minimization and PAC Learning)
- Graphical Models
- Inference with graphical models: belief propagation
- Hidden Markov Models for sequential data
- Bayesian Linear Regression and Classification, Laplace approximation, Model Selection
- Kernel Regression and Kernel functions, Gaussian Processes for regression (hints)
- Monte Carlo sampling
- Expectation Maximization and Variational Inference
- Bayesian Neural Networks
- Generative Modelling: Variational Autoencoders and Diffusion Processes

The structure of these notes follows the natural progression from fundamental probabilistic concepts to advanced generative models, emphasizing both theoretical foundations and practical applications. While these notes were primarily created for my personal study, they may serve as a valuable resource for fellow students and professionals interested in probabilistic machine learning.

Contents

1 Introduction **1**
- 1.1 Models and Probability . . . 1
- 1.2 Probability basics . . . 2
  - 1.2.1 Random Variables . . . 2
  - 1.2.2 Notable Probability Distributions . . . 2

2 Empirical Risk Minimization and PAC Learning **4**
- 2.1 Empirical Risk Minimization . . . 4
  - 2.1.1 Risk and Empirical Risk . . . 5
  - 2.1.2 Bias Variance Trade-off . . . 5
- 2.2 ERM and Maximum Likelihood . . . 6
- 2.3 KL divergence . . . 6
- 2.4 PAC Learning . . . 8
- 2.5 VC Dimension (Vapnik-Chervonenkis) . . . 10
  - 2.5.1 VC dimension and PAC learning . . . 11
  - 2.5.2 Rademacher Complexity . . . 11
  - 2.5.3 Rademacher complexity and VC dimension . . . 12

3 Probabilistic Graphical Models **14**
- 3.1 Probabilitic Inference . . . 14
- 3.2 Bayesian Networks . . . 15
  - 3.2.1 Sampling and reasoning in BN . . . 16
  - 3.2.2 Conditional Independence in BN . . . 19
  - 3.2.3 Naive Bayes . . . 21
- 3.3 Random Markov Fields . . . 22

12 Exact Inference in Probabilistic Graphical Models **105**
- 12.1 Factor Graphs . . . 106
  - 12.1.1 From Bayesian Networks to Factor Graphs . . . 107
  - 12.1.2 From Markov Random Fields to Factor Graphs . . . 108
- 12.2 Sum Product Algotithm . . . 109
- 12.3 Max Plus algorithm . . . 113
- 12.4 Inference in general Probabilistic Graphical Models . . . 116

5 Hidden Markov Models **37**
- 5.1 Inference in HMM . . . 39

6 Bayesian Linear Regression **42**
- 6.1 Gaussian Distributions . . . 42
  - 6.1.1 Principal Components . . . 42
  - 6.1.2 Completing the square . . . 42
  - 6.1.3 Further properties . . . 43
- 6.2 Bayesian Estimation . . . 44
- 6.3 Linear Regression . . . 45

6.4 Bayesian Linear Regression . . . 47
6.4.1 Online Learning . . . 49
6.5 Predictive distribution . . . 50
6.6 Model evidence . . . 52
6.6.1 Fixed-point algorithm . . . 52
6.6.2 Effective number of parameters . . . 53
6.7 Model Comparison . . . 54

7 Bayesian Linear Classification 56
7.1 Logistic Regression . . . 56
7.2 Laplace Approximation . . . 58
7.2.1 Laplace approximation for model comparison . . . 59
7.3 Bayesian Logistic Regression . . . 60

8 Sampling-based Inference 62
8.1 Approximate Sampling . . . 63
8.2 Markov Chain . . . 65
8.3 Markov Chain Monte Carlo . . . 67
8.4 Hamiltonian Monte Carlo . . . 73

9 Expectation Maximization 75
9.1 Evidence lower bound . . . 76
9.2 Expectation Maximization . . . 77
9.3 Mixture of Gaussians . . . 79
9.4 EM for Bayesian Networks . . . 81
9.5 EM for Hidden Markov Models . . . 82

10 Variational Inference 84
10.1 Variational Linear Regression . . . 88
10.2 Black box Variational Inference . . . 89
10.3 Control variates . . . 92
10.4 Bayesian Neural Networks . . . 93

11 Generative Modelling 96
11.1 Variational Autoencoders . . . 96
11.2 Diffusion Models . . . 100

12 Exact Inference in Probabilistic Graphical Models 105
12.1 Factor Graphs . . . 106
12.1.1 From Bayesian Networks to Factor Graphs . . . 107
12.1.2 From Markov Random Fields to Factor Graphs . . . 108
12.2 Sum Product Algotithm . . . 109
12.3 Max Plus algorithm . . . 113
12.4 Inference in general Probabilistic Graphical Models . . . 116

1 Introduction

1.1 Models and Probability

Machine learning is a field of computer science about **learning models**.

Definition: *Machine learning*

Machine learning *explores the study and construction of algorithms that can learn from and make predictions on data.* *Wikipedia*

If we dig into this statement, we may wonder what precisely do ML algorithms learn.

The answer to this question is (apparently) simple: they learn models of the observed data. Models that can then be used to make predictions or to extract information from such data.

Definition: *Model*

- A ***Model*** is a hypothesis that certain features of a system of interest are well replicated in another, simpler syetem.
- ***Mathematical Model*** is a model where the simpler system consists of a set of mathematical relations between objects (equations, inequalities, etc).
- A ***Stochastic Model*** is a mathematical model where the objects are probability distributions.

All modelling usually starts by defining a family of models indexed by some parameters, which are tweaked to reflect how well the feature of interest are replicated.

Machine learning deals with algorithms for automatic selecrion of a model from observations of the system.

Generative and Discriminative Learning

- **Generative Learning** im at describing the full probability distribution of inputs x or input/output pairs (x,y).
$$p(x,y) = p(x)p(y|x)$$
- **Discriminative Learning** aims at describing the conditional probability of output given the input, or a statistics/function of such probability
$$p(y|x) \quad or \quad y = f(x)$$

[to fix:]

- **Supervised Learning**: The algorithm learns from labeled data by mapping inputs to outputs.
- **Unsupervised Learning**: The algorithm identifies patterns or structures in unlabeled data.
- **Data Generatiion**: The algorithm generates new data points.

Inference and Estimation

Two central concepts for probabilistic machine learning are:

- **Inference**: Compute marginals and contitionals probability distributions applying the laws of probability.
- **Estimation**: Given data and a family of models, find the best parameters/models that explains the data.

In the Bayesian world: estimation $\approx$ inference.

Probability

Probability is a mathematical theory that deals with **uncertainty**

When a certain problems has to face practical difficulties due to it's complexity, we can use probability to model the ***aleatorical uncertainty***, which is the uncertainty due to the randomness of the system.

More often, we have a limited knowledge of the system, and we can use probability to model the ***epistemic uncertainty***, which is the uncertainty due to the lack of knowledge.

Tip: ***Everything is a probability distribution***

In machine learning **everything is a probability distribution**, even if not explicitly stated.

1.2 Probability basics

1.2.1 Random Variables

Random Variables are functions mapping outcomes of an experiment to real numbers. They serve as abstract representations of the outcomes in randomized experiments. Note that what we observe are the *realizations* (values resulting from an observed outcome) of these random variables.

Example: ***Random Variable***

Consider the following example:

$$\{Head, Tail\}, \qquad \{0,1\}, \qquad \left\{\tfrac{1}{2}, \tfrac{1}{2}\right\}.$$

Only the second is the random variable itself; the third is its probability distribution, while the first is the sample space of potential outcomes.

We consider a ***Sample Space*** Ω, which is the set of all possible outcomes of a random experiment. A random variable X is a function:

$$X : \Omega \to E, \qquad \text{where } E \subseteq \mathbb{R} \quad (\text{or } E \subseteq \mathbb{N})$$

with the probability measure

$$P(X \in S) = P(\{\omega \in \Omega \mid X(\omega) \in S\}), \quad S \subseteq E.$$

A model for our random outcome is the probability distribution of X. In particular, if the sample space is finite or countable the **probability mass function (pmf)** is given by:

$$p(x) := P(X = x).$$

If the sample space is infinite, we use the **probability density function (pdf)** where

$$P(a \leq X \leq b) = \int_a^b p(x)dx \quad \text{and} \quad \int_{\mathbb{R}} p(x)dx = 1.$$

1.2.2 Notable Probability Distributions

Below are some of the most common probability distributions.

Discrete Distributions

Distribution	pmf	Mean	Variance
Binomial $\mathrm{Bin}(n,p)$	$\binom{n}{x}p^x(1-p)^{n-x}$	np	$np(1-p)$
Bernoulli $\mathrm{Bern}(p)$	$p \quad (x=1), \quad 1-p \quad (x=0)$	p	$p(1-p)$
Discrete Uniform $\mathscr{U}(a,b)$	$\dfrac{1}{b-a+1}$	$\dfrac{a+b}{2}$	$\dfrac{(b-a+1)^2-1}{12}$
Geometric $\mathrm{Geom}(p)$	$(1-p)^{x-1}p$	$\dfrac{1}{p}$	$\dfrac{1-p}{p^2}$
Poisson $\mathrm{Pois}(\lambda)$	$\dfrac{\lambda^x e^{-\lambda}}{x!}$	λ	λ

Continuous Distributions

Distribution	pdf	Mean	Variance
Continuous Uniform $\mathscr{U}(a,b)$	$\begin{cases}\frac{1}{b-a} & x\in[a,b]\\ 0 & \text{otherwise}\end{cases}$	$\dfrac{a+b}{2}$	$\dfrac{(b-a)^2}{12}$
Exponential $\mathrm{Exp}(\lambda)$	$\lambda e^{-\lambda x}$	$\dfrac{1}{\lambda}$	$\dfrac{1}{\lambda^2}$
Gaussian $\mathscr{N}(\mu,\sigma^2)$	$\dfrac{1}{\sigma\sqrt{2\pi}}\exp\left(-\dfrac{1}{2}\left(\dfrac{x-\mu}{\sigma}\right)^2\right)$	μ	σ^2
Beta $\mathrm{Beta}(\alpha,\beta)$	$\dfrac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$	$\dfrac{\alpha}{\alpha+\beta}$	$\dfrac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$
Gamma $\mathrm{Gamma}(\alpha,\beta)$	$\dfrac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x}$	$\dfrac{\alpha}{\beta}$	$\dfrac{\alpha}{\beta^2}$
Dirichlet $Dir(\alpha)$	$\dfrac{1}{B(\alpha)}\prod_{i=1}^{K}x_i^{\alpha_i-1}$	$\tilde{\alpha}_i$	$\dfrac{\tilde{\alpha}_i(1-\tilde{\alpha}_i)}{\alpha_0+1}$
Student's t $St(\nu)$	$\dfrac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)}\left(1+\dfrac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$	0	$\begin{cases}\frac{\nu}{\nu-2} & \nu>2\\ \infty & 1<\nu\le 2\end{cases}$

Notes:

- For discrete distributions, $n\in\{0,1,2,\dots\}$, $p\in[0,1]$, and x runs over the support.
- For continuous distributions, parameters such as λ, μ, σ, α, and β belong to $\mathbb{R}$ (with appropriate restrictions) and $x\in\mathbb{R}$.
- In the Dirichlet distribution, $\tilde{\alpha}_i=\frac{\alpha_i}{\sum_{h=1}^{K}\alpha_h}$ and $\alpha_0=\sum_{i=1}^{K}\alpha_i$.
- For Student's t-distribution, $\nu>1$.

2

Empirical Risk Minimization and PAC Learning

In this chapter, we will introduce the concept of **Empirical Risk Minimization** (ERM) in which to frame learning problems, the notion of inductive bias, and the main results of algorithmic learnability, encapsulated in the definition of **Probably Approximately Correct** (PAC) Learning and of complexity of a set of hypothesis, namely VC-dimension and Rademacher complexity.

2.1 Empirical Risk Minimization

We begin by considering a supervised learning setting in which the **input space** X is a subset of $\mathbb{R}^n$, and the **output space** Y can be real-valued (e.g., $Y = \mathbb{R}$), binary (e.g., $Y = \{0,1\}$), or a finite set of classes (e.g., $Y = \{0,1,\ldots,K\}$). In this probabilistic framework, each input-output pair (x,y) is drawn from a joint probability distribution

$$p(x,y) \in \text{Dist}(X \times Y),$$

often referred to as the ***data generating distribution***.

By definition, this distribution factors into the marginal $p(x)$ and the conditional $p(y \mid x)$, so that

$$p(x,y) = p(x)\,p(y \mid x).$$

Because $p(x)$ and $p(y \mid x)$ describe how inputs and outputs are related, it is helpful to write them explicitly. The marginal distribution of x is

$$p(x) = \int p(x,y)\,dy,$$

while the conditional distribution of y given x is

$$p(y \mid x) = \frac{p(x,y)}{p(x)}.$$

A typical dataset D in supervised learning consists of N input-output pairs drawn independently from $p(x,y)$. We denote this as

$$D \sim p^N(x,y),$$

which means

$$D = \{(x_i,y_i) \mid i = 1,\ldots,N\},$$

where each (x_i,y_i) is sampled according to the joint distribution $p(x,y)$.

In many cases, we assume that $p(y \mid x)$ depends on some unknown function of x. Formally, one might write

$$p(y \mid x) = p\big(y \mid f(x)\big),$$

where f is the function we aim to learn. The central objective in supervised learning—through methods such as empirical risk minimization—is to find or approximate this function f by using the observed data D.

2.1.1 Risk and Empirical Risk

$h \in \mathscr{H} \quad x,y \sim p(x,y)$

loss function $l(x,y,h) \in \mathbb{R}_{\geq}$,

- 0-1 loss: $l(x,y,h) = \mathbb{I}(h(x) \neq y)$, eith $y \in \{0,1\}$.
- squared loss: $l(x,y,h) = (h(x)-y)^2$, with $y \in \mathbb{R}$.

We have a probabilistic process, so we have some inputs that are more likely than others. If a model makes a mistake on a more likely input, it should be penalized more.

Definition: ***Risk***

The ***risk*** (or ***generalization error***) is defined as:

$$R(h) = E_{x,y\sim p(x,y)}[l(x,y,h)]$$

Risk minimization principle:

The goal is to find the hypothesis h that minimizes the risk.

$$\text{find } h^* \in \mathscr{H} \text{ such that } h^* = \arg\min_{h\in\mathscr{H}} R(h)$$

Definition: ***Empirical Risk***

The ***empirical risk*** (or ***training error***) is defined as:

$$\hat{R} = \frac{1}{N}\sum_{i=1}^{N} l(x_i,y_i,h)$$

Empirical risk minimization principle:

The goal is to find the hypothesis h that minimizes the empirical risk.

$$\text{find } h_D^* = \arg\min_{h\in\mathscr{H}} \hat{R}(h)$$

2.1.2 Bias Variance Trade-off

In this section, we want to analyze the generalization error and decom- pose it according to the sources of error that we are going to commit.

In what follows, we will use the squared loss (hence we will focus on regression problems). Considering $h \in \mathscr{H}$, an explicit expression of the generalization error committed when choosing hypothesis h is:

$$R(h) = E_p[l(x,y,h)] = \int\int (h(x)-y)^2 p(x,y)dxdy$$

Theorem 1. *The minimizer of the generalization error R is:*

$$\boxed{g(x) = E[y|x] = \int y p(y|x)dy}$$

so that $g = \arg\min_h R(h)$, *if* $g \in \mathscr{H}$

We can rewrite the risk as:

$$R(h) = \underbrace{\int (h(x)-g(x))^2 p(x)dx}_{=0 \quad iff \quad h(x)=g(x)} + \overbrace{\iint (g(x)-y)^2 p(x,y)dxdy}^{\text{independent of } h\text{: intrinsic noise}}$$

$$
\begin{aligned}
E_D[R(h_D^*)] &= \underbrace{\int (E_D[h_D^*(x) - g(x)])^2 p(x)dx}_{bias^2} \\
&+ \underbrace{\int E_D[(h_D^*(x) - E_D[h_D^*(x)])^2 p(x)dx]}_{variance} \\
&+ \underbrace{\iint (g(x) - y)^2 p(x,y)dxdy}_{noise}
\end{aligned}
$$

2.2 ERM and Maximum Likelihood

Given a dataset $D = \{(x_i, y_i)\}_{i=1,\dots,m}$ s.t. $D \sim p^m, p = p(x,y)$

We factorize the data generating distributions as: $p(x,y) = p(x)p(y|x)$ and we make an hypothesis on $p(y|x)$, trying to express this conditional probability in a parametric form:

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

[check recording for missing part]

We consider the log Likelihood:

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

Then we apply the maximum likelihood principle:

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

since the avarage is an empirical approximation of the expectation.

Observation: ***Empirical Risk***

$-\frac{1}{m} \sum_{i=1}^{m} \log p(y_i|x_i,\theta)$ is known as **empirical risk**

2.3 KL divergence

Consider a probability distribution $p(x)$, then $-\log p(x)$ is a measure of **self-information**. Indeed, if $p(x) = 1$ then $-\log p(x) = 0$ (no self-information), describing substantially out (lack of) surprise in observing the event. If instead $p(x) = 0$ then $-\log p(x) = \infty$. In general, the more rare the event is, i.e. the lower is $p(x)$, the more self-information it carries, i.e. the larger is $-\log p(x)$.

Definition: ***Entropy***

In an information-theoretic sense, the **entropy** is a measure of the inforamtion that is carries by a random phenomenon, expressed as the expected amount of self-information that is conveyed by a realization of the random phenomenon.
It is formally defined as:

$$H(p) = E_p[-\log p(x)] = -\int p(x) \log p(x) dx$$

for the continuous case, and:

$$H(p) = E_p[-\log p(x)] = -\sum p(x) \log p(x)$$

for the discrete case.

For the discrete case, the maximum entropy is achieved for te uniform distrivution and is equal to $\log n$, where n is the number of possible outcomes. In the continuous case, for a fixed variance, the distribution that maximizes the entropy is the Gaussian. The entropy is always 0 if we have a deterministic distribution.

Definition: ***Kullback-Leibler divergence***

The **Kullback-Leibler divergence** (or **relative entropy**) between two probability distributions $p(x)$ and $q(x)$ is a measure of how one distribution diverges from a second, expected probability distribution. It is formally defined as:

$$D_{KL}(p||q) = E_p\left[\log \frac{p(x)}{q(x)}\right] = \int p(x) \log \frac{p(x)}{q(x)} dx$$

for the continuous case, and:

$$D_{KL}(p||q) = E_p\left[\log \frac{p(x)}{q(x)}\right] = \sum p(x) \log \frac{p(x)}{q(x)}$$

for the discrete case.

Intuitively, we are taking a sort of expected difference between p and q, expressed in terms of a log odds ratio. It tells us how different two distributions are: the larger KL the more different are p and q.

Properties of KL:

- $KL[q||p]$ is a convex function of q and p and $KL[q||p] \geq 0$
- KL is non-symmetric, i.e. $KL[q||p] \neq KL[p||q]$
- $KL[q||p] = -H[q] - \mathbb{E}_q[\log p]$, where the first term is the entropy and the second term is known as cross-entropy between p and q.

Suppose moreover, that p is fixed but unknown, $q = q_0$ can vary: what we usually do is trying to find the best q_θ that approximates p.

The **mutual information** between x and y is defined as:

$$I(x,y) = KL[p(x,y)||p(x)p(y)] = \int\int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy$$

$KL[p(x,y)||p(x)p(y)] = 0$ iff x and y are independent.

Moreover, the more dependent they are, the more different is $p(x,y)$ from the product of the marginals, the more information x carries about y and viceversa.

In other words, the higher the mutual information is, the more knowing y will tell us about x, the less residual uncertainty on x we will have.

Consider a dataset: $\underline{x} : x_1, \ldots, x_N$:

Definition: ***Empirical distribution***

The **empirical distribution** of a dataset $\underline{x}$ is defined as:

$$\hat{p}(\underline{x}) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x_i)$$

where $\delta(x)$ is the Dirac delta function.

It is an approximation of the input data generating function$p(x)$. Practically, the more observations we have, the more the empirical distribution will look like $p(x)$.

Given a distribution q, we can compute:

$$KL[p_{emp}||q] = \mathbb{E}_{p_{emp}}\left[\log \frac{p_{emp}(x)}{q(x)}\right] = \frac{1}{N}\sum_{i=1}^{N} \log \frac{p(x_i)}{q(x_i)}$$

If $q = q_0$, this is $-\frac{1}{N}L(\Theta)$ plus a constant. Hence maximizing $L(\Theta)$ is essentially equivalent to minimizing the KL between p_{emp} and q_0. This means that we can always rephrase maximum likelihood in terms of cross-entropy.

2.4 PAC Learning

Our goal is to measure how much we can learn as a function of the model complexity. This results in the **PAC** (Probably Approximately Correct) **learning** framework, which encodes the notion of model complexity and gives also bounds on the error that we commit. Here we consider it in the context of (binary) classification, i.e. $y \in 0, 1$, using the 0-1 loss.

Definition: ***PAC Learning***

A realizable hypothesis set $\mathscr{H}$ is ***PAC learnable*** iff $\forall \varepsilon, \delta \in (0,1), \forall p(x,y), \exists m_{\varepsilon,\delta} \in \mathscr{N}\, s.t. \forall m \geq m_{\varepsilon,\delta}, \exists D\ p^m, |D| = m$ then $p_D(R(h_D^*) \leq \varepsilon) \geq 1 - \delta$

This means that, fixing two parameters $\varepsilon, \delta \in (0,1)$, governing our precision, and a data generating distribution $p(x,y)$, we can find a number of samples $m_{\varepsilon,\delta}$ such that, with probability at least $1-\delta$, the empirical risk of the best hypothesis h_D^* is less than ε. Note that the probability here is over the dataset D, meaning that our learning will succeed for a fraction $1-\delta$ of sampled datasets.

In a more general setting,

Definition:

Given an hypothesis set $\mathscr{H}$ (not necessarily realizable) and an algorithm A, $\mathscr{H}$ *is* **agnostic PAC-learnable** $if f \forall \varepsilon, \delta \in (0,1), \forall p(x,y), \exists m_{\varepsilon,\delta} \in \mathscr{N}\ p^m, |D| = m \geq m_{\varepsilon,\delta} \Rightarrow p_D(R(h_D^*) \leq R(h^*) + \varepsilon) \geq 1 - \delta$, being h_D^A the result of applying A to $\mathscr{H}$ and D.

This means that, fixing two parameters $\varepsilon, \delta \in (0,1)$, governing our precision, and a data generating distribution $p(x,y)$, we can find a number of samples $m_{\varepsilon,\delta}$ such that, with probability at least $1-\delta$, the empirical risk of the best hypothesis h_D^* is less than the risk of the best hypothesis h^* plus ε.

Finite hypothesis sets

An hypothesis set is said to be **finite** is $\mathscr{H}$ is s.t. $|\mathscr{H}| < \infty$.

Using combinatorial arguments, we can prove that finite hypothesis sets are agnostic PAC-learnable with:

$$m_{\varepsilon,\delta} \leq \lceil \frac{2\log(\frac{2\mathscr{H}}{\delta})}{\varepsilon^2} \rceil$$

hence with polynomial dependency on ε and δ. In this framework, $\log(|\mathscr{H}|)$ is a measure of the complexity of the set $\mathscr{H}$.

Warning:

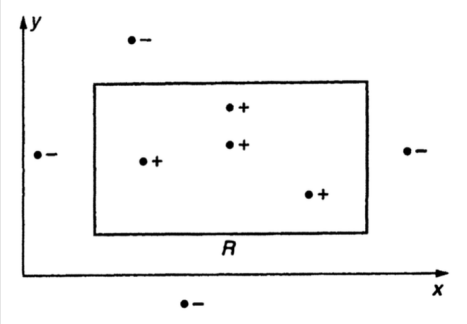
If $\mathscr{H}$ is described by d parameters of type double when represented in a computer (64 bits), it holds that $|\mathscr{H}| \leq 2^{d64}$, so we have a finite set of hypothesis, hence we can provide a bound on every implementable set of hypothesis functions.
In this case,

$$m_{\varepsilon,\delta} \leq \frac{128d + 2\lg(\frac{2}{\delta})}{\varepsilon^2}$$

i.e. we have linear dependency on the number of parameters.

Example: *Pac learning example*

We consider an example introduced by Kearns and Vazirani in the book "An Introduction to Computational Learning Theory". [2].
The goal is to learn a target axis-aligned rectangle R lying in $\mathbb{R}^2$ and $z \in -1, 1$ distributed according to a distribution $p(\mathbf{x}, z)$. The hypothesis set $\mathscr{H}$ is the set of all axis-aligned rectangles lying in $\mathbb{R}^2$. We assume there is a true rectangle of this king such that all positive points are inside it, and all negative points are outside.



We consider out hypothesis h to be the axis-aligned rectangle R' with the smallest area that includes all the positive examples and none of the negative ones.
What is the minimum number $m_{\varepsilon,\delta}$ of training examples so that, wiht probability at least $1-\delta$, h has an error at most ε with respect to the true rectangle and the distribution $p(\mathbf{x}, z)$?
Solution
We have to find an $m_{\varepsilon,\delta}$ such that $\forall m > m_{\varepsilon,\delta} P(error(h_m) > \varepsilon) \leq \delta$. First of all, let's notice that the error $error(h_m)$ is equal to the area of the target rectangle R minus the area of the internal rectangle $R' = h_m$ that we found.
Then, given an arbitrary error bound ε, we build within R 4 rectangles having each one an area of $\varepsilon/4$, on the sides of R.

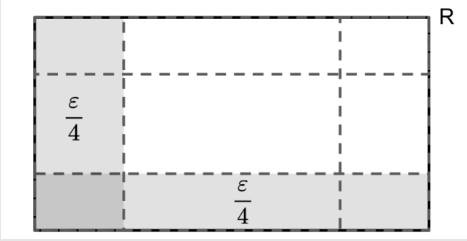


Figure 2.1: The 4 rectangles of area $\varepsilon/4$

Let's consider the m observations drawn from $p(\mathbf{x}, y)$. If each of the four rectangles defined above contains at least one point, we have $error(h_m) \leq \varepsilon$, because the area difference between R and R' would be fully covered by the 4 rectangles.

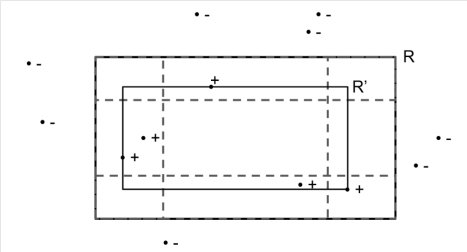


Figure 2.2: Configuration showing event

If we call this event B, we have:

$$B \rightarrow error(h_m) \leq \varepsilon$$

equivalently, by modus tollens:

$$error(h_m) > \varepsilon \rightarrow \neg B$$

This implies:

$$P(\neg B) \geq P(error(h_m) > \varepsilon)$$

$P(\neg B)$ is the probability that at least one of the 4 rectangles doesn't contain any of the m points. This probability is $(1-\varepsilon/4)^m$ for a single rectangle, hence we have $P(\neg B) \leq 4(1-\varepsilon/4)^m$. Thys the following chain of inequalities holds:

$$4(1-\varepsilon/4)^m \geq P(\neg B) \geq P(error(h_m) > \varepsilon)$$

Now, let $m_{\varepsilon,\delta}$ be such that:

$$\delta \geq 4(1-\varepsilon/4)^m_{\varepsilon,\delta} \geq P(\neg B) \geq P(error(h_m) > \varepsilon)$$

Finding $m_{\varepsilon,\delta}$ that satisfy this inequality would prove that $\forall m > m_{\varepsilon,\delta} P(error(h_m) > \varepsilon) \leq \delta$. We then require:

$$4(1-\varepsilon/4)^m_{\varepsilon,\delta} \leq \delta$$

and we use the inequality $(1-k) \leq e^{-k}$ to obtain:

$$m_{\varepsilon,\delta} \geq (4/\varepsilon) \cdot \ln(4/\delta)$$

In summary, provided a sample of at least $(4/\varepsilon) \cdot \ln(4/\delta)$ examples in order to choose an hypothesis rectangle R', we can assert that with probability at least $1 - delta, R'$ will misclassify a new point (drawn according to the same distribution from which the sample was chosen) with probability at most ε.
This proves that our hypothesis set $\mathscr{H}$ is PAC-learnable.

2.5 VC Dimension (Vapnik-Chervonenkis)

Consider a class of hypotheses functions $\mathscr{H} = h : X \rightarrow 0, 1$ and a subset $C = c_1, \ldots, c_m \subseteq X$ of input points.

Define $\mathscr{H}_c = (h(c_1), \ldots, h(c_m)) | h \in \mathscr{H}$, the set of all tuples of Booleans obtained by applying all possible hypothesis functions $h \in \mathscr{H}$ to all points in C. We say that $\mathscr{H}$ **shatters** the set C iff $|\mathscr{H}_C| = 2^m$.

Practically, this means that for any label assignment to points in C, we have a function in our hypothesis set which is able to match such an assignment. Namely, we can exactly describe every possible dataset with inputs in C.

Definition: *VC Dimension*

The **VC dimension of** $\mathscr{H}$ is defined as:

$$VC(\mathscr{H}) = \max\{m \in \mathbb{N} | \mathscr{H} \text{ shatters some } C \subseteq X, |C| = m\}$$

Remark: In calculating the VC dimension, it is enough that we find one set of m points that can be shattered, it is not necessary that we are able to shatter any m points.

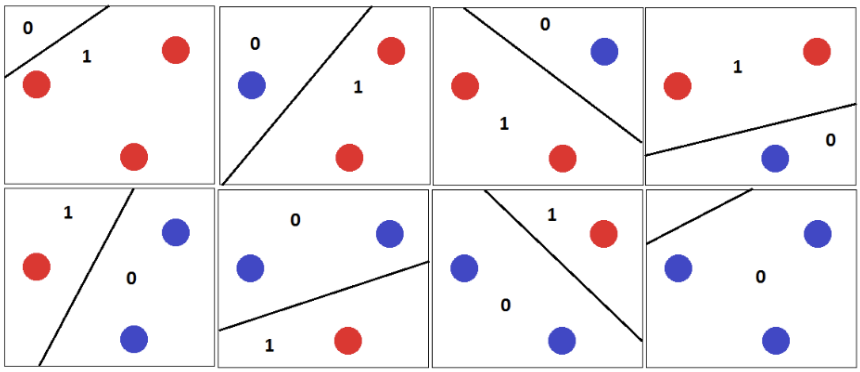


Figure 2.3: Proof that VCdim($\mathscr{H}_l \geq 3$)

2.5.1 VC dimension and PAC learning

In what follows, we will explore the reasons why VC dimension in crucial for PAC learnability.

Theorem 2. *If* $\mathscr{H}$ *shatters* $C, |C| \geq 2m$, *then we cannot learn* $\mathscr{H}$ *with* m *samples*

Hence, there will be an assignment of m samples to classes in which we are going to commit a large error.

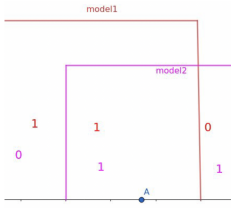


Figure 2.4: Visual interpretation of the theorem: it is impossible to train a model of type $\mathscr{H}_a+$ with only a point A with known classification, because the points different from A could have any classification.

If the $VCdim(\mathscr{H}) = \infty$, then $\mathscr{H}$ is not (agnostic) PAC learnable, indeed:

Theorem 3. $\mathscr{H}$ *is (agnostic) PAC learnable if* $VCdim(\mathscr{H}) < \infty$

In this case: $\exists c_1, c_2$ s.t.

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

Hence VC dimension gives us control on what we can or cannot learn.

2.5.2 Rademacher Complexity

Consider the data generating distribution $p(x,y)$, a dataset D p^m and an hypothesis class $\mathscr{H} = h : X \to -1, 1$.

Definition: *Rademacher Distribution*

A distribution $\sigma = (\sigma_1, \dots, \sigma_m) s.t. \sigma_i \in -1, 1 \forall i and p(\sigma_i = 1) = 0.5$ is called **Rademacher Distribution**

Definition: *Rademacher Complexity*

The **Rademacher Complexity** of $\mathscr{H}$ is defined as:

$$R_m(\mathscr{H}) = \mathbb{E}_\sigma \left[\sup_{h \in \mathscr{H}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i h(x_i) \right]$$

Remark: this is a property both of the function h and of the dataset D.

Observation:

$\sum_{i=1}^{m} \sigma_i h(x_i) = \sigma \cdot h(\underline{x})$ is the scalar product of the Rademacher distribution $\sigma$with the function h evaluated on our dataset, so that $\frac{1}{m} \sigma \cdot h(\underline{x}) \in [-1, 1]$, essentially is a measure of correlation of h with random noise σ.

Hence, for a specific choice of noise σ, we are going to look at the dataset and choose the best h that correlates with this noise; then we take the expectation w.r.t. σ.

Definition: *Rademacher Complexity*

Taking into account the data generating mechanism p, the **data-independent Rademacher complexity** is defined as:

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

Fix $\mathscr{H} and p(x,y)$, then $\forall \sigma > 0$ with probability at least $1 - \delta, \forall D\ p^m, |D| = m, \forall h \in \mathscr{H}$ we have:

$$R(h) \leq \hat{(}R_D)(h) + \underbrace{\mathscr{R}_m(\mathscr{H}) + \sqrt{\frac{\log(\frac{1}{\delta})}{2m}}}_{\varepsilon_1}$$

$$R(h) \leq \hat{(}R_D)(h) + \hat{\mathscr{R}}_D(\mathscr{H}) + 3\sqrt{\frac{\log(\frac{2}{\delta})}{sm}}_{\varepsilon_2}$$

Remark: computing the Rademacher complexity can be challenging, as it requires the solution of an optimization problem of any possible value of the Rademacher distribution.

2.5.3 Rademacher complexity and VC dimension

Definition: *Growth Function*

The **growth function** $\prod_{\mathscr{H}}$ is defined as:

$$\prod_{\mathscr{H}} : \mathbb{N} \to \mathbb{N}, \prod_{\mathscr{H}}(m) = \max_{x_1, \dots, x_m \in X} |\mathscr{H}_D|, |D| = m$$

with $\mathscr{H}_D = h(x_1), \dots, h(x_m) | h \in \mathscr{H}, D = x_1, \dots, x_m$

Hence the growth function describes how the complexity of what we can explain with out hypothesis set $\mathscr{H}$ increases with the cardinality of the data points that we have.

We can moreover define the $VCdim(\mathscr{H})$ in terms of the growth function:

$$VCdim(\mathscr{H}) = \max\{m \in \mathbb{N} | \prod_{\mathscr{H}}(m) = 2^m\}$$

Intuitively, this comes from the fact that if a set C is shattered by $\mathscr{H}$, then $\mathscr{H}_D = 2^m$.

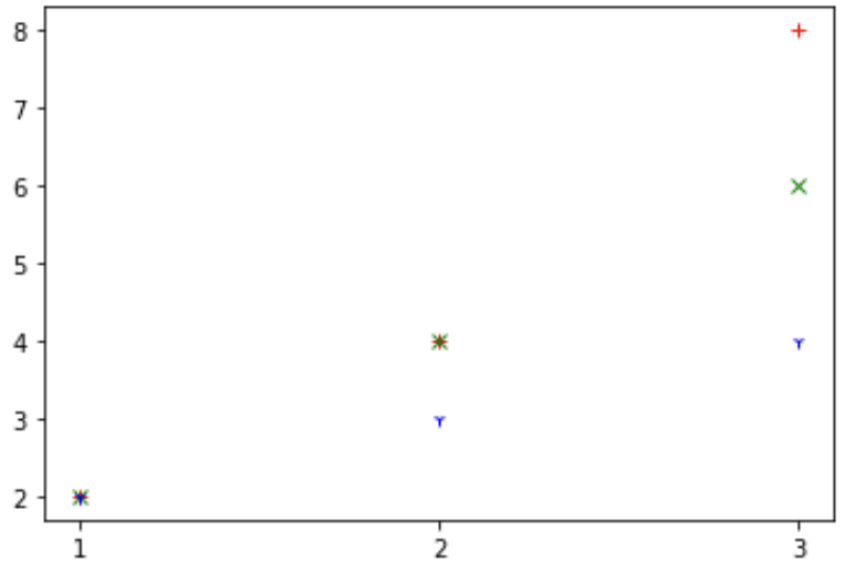


Figure 2.5: Plot of the growth functions of $\mathscr{H}_a$(blue), $\mathscr{H}_{a+}$(green), $\mathscr{H}_l$(red) for $1 \leq m \leq 3$

Theorem 4 (Sauer Lemma)**.**

$$\prod_{\mathscr{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{em}{d}\right)^d \leq O(m^d)$$

with $d = VCdim(\mathscr{H})$

Moreover it also holds that:

$$\mathscr{R}_m(\mathscr{H}) \leq \sqrt{\frac{2\log(\frac{2}{\delta})}{m}} + \sqrt{\frac{2\log(\frac{2}{\delta})}{m}} \leq O(\sqrt{\frac{d}{m}})$$

We can say that $\mathscr{R}_m(\mathscr{H})$ and $VCdim(\mathscr{H})$ are essentially equivalent, in the sense that when the VC dimension is ∞ then the upper bound on the Redemarcher complexity is independent of m and greater than 1 (if you substitute it into the PAC bound using Redemacher complexity, you get an error which remains always large, no matter how large is m). Otherwise, when the VC dimension is $< \infty$, the bound tends to go to 0 as m grows to infinity.

Summarizing, we have ways to measure the complexity of our hypotheses space: if this complexity is finite (in the sense of VC dimensionality), then what we have as a result is that we can constrain the error and provide bounds that tell us that this error is going towards 0, as we increase the data points (i.e. we will eventually learn). Instead if the VC dimension is infinite, no matter how many data points we have, we are not going to be able to learn, because the complexity of our model is too high, hence it can always overfit the data.

Hence, in order to be able to actually learn, we need to put some constraints in our hypothesis set, and this formalizes our inductive bias.

3
Probabilistic Graphical Models

3.1 Probabilitic Inference

Suppose having a set of random variates (either discrete or continuous) $x = (x_1, \ldots, x_n), z = (z_1, \ldots, z_m)$ with joint probability distribution $p(x,z)$.

To reason about such a model, we would perform:

- **Marginalization**: $p(x) = \sum_z p(x,z) = \int p(x,z)\, dz\, dx_{-i}$
- **Conditioning**: $p(x|z) = \frac{p(x,z)}{p(z)}$

These operations can be combined to perform **inference** on the model, i.e. to compute the probability of some variables given the values of others.

$$p(z_j|x_i = \bar{x}_i) = \frac{p(\bar{x}_i), z_j}{p(\bar{x}_i)} = \frac{\sum_{\bar{z}_j} p(\bar{x}_i, \bar{z}_j)}{p(\bar{x}_i)} = \frac{\int p(x,z)\, dx_{-i}\, dz_j}{\int P(x,z)\, dx_{-i}\, dz}$$

What is the cost of computation of performing these operations?

Well, focusing on discrete values for z such that $z_i \in 0,1$ and considering the marginalization over it:

$$p(x) = \sum_{z \in Z} p(x,z)$$

Here $z \in Z, with |Z| = 2^m$, resulting in an asympotically exponential cost in the number of variables.

Let's set up a strategy to perform inference and marginalizaton in a more efficient way.

Factorisation

Consider $p(x_1, \ldots, x_n)$. Applying the laws of probability we can write

$$\begin{aligned} p(x_1, \ldots, x_n) &= p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)\ldots p(x_n|x_1,\ldots,x_{n-1}) \\ &= \textstyle\prod_{i=1}^{n} p(x_i|x_1,\ldots,x_{i-1}) \\ &= p(x_n|x_1,\ldots,x_{n-1})p(x_{n-1}|x_1,\ldots,x_{n-2})\ldots p(x_1) \end{aligned}$$

What is then the cost in terms of computer memory to store these three representation of out distribution? Suppose $x_i \in 1, \ldots, k$, we have the following costs:

- $p(x_1)$ costs $k-1 = O(k)$ floating point numbers (single or double precision)
- $p(x_2|x_1)$ costs $k(k-1) = O(k^2)$ float numbers (we have different distributions for x_2, each costing $k-1$ floats)
- In general $p(x_n|x_1,\ldots,x_{n-1})$ costs $O(k^n)$ float numbers
- Factorization costs $O(k^n)$ float numbers, unfeasible

Probabilistic Graphical Models (PGM) are models of a probability distribution that describe/impose a certain factorization, hence allowing us to store and make inference effectively with joint distributions. There are three main kind of PGM:

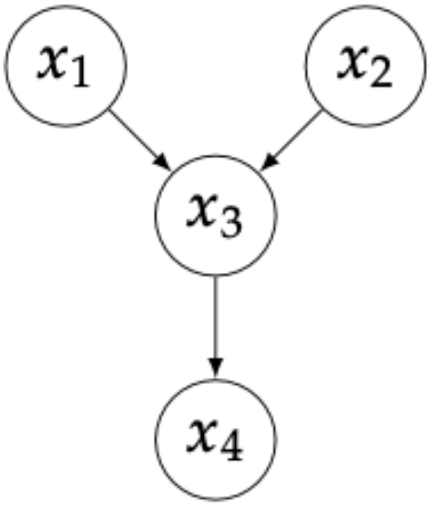
- Bayesian Networks
- Markov Random Fields
- Factor Graphs

3.2 Bayesian Networks

Definition: *Bayesian Network*

A Bayesian Network is a directed acyclic graph (DAG) $G = (V, E)$ where:
- $V = \{x_1, \ldots, x_n\}$ is a set of random variables
- $E \subseteq V \times V$ is a set of directed edges

such that each node x_i is associated with a conditional probability distribution $p(x_i|pa(x_i))$ where $pa(x_i)$ is the set of parents of x_i in G.



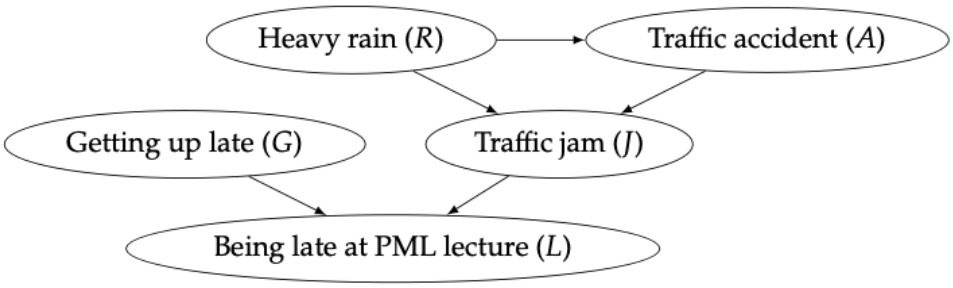


Figure 3.1: Example of a Bayesian Network

This BN corresponds to the following factorization:

$$p(L,G,J,R,A) = p(L|G,J)p(J|R,A)p(A|R)p(R)p(G)$$

Tip:

Directed edges does mean causality. BN **are not** a causal model.

Notational Conventions

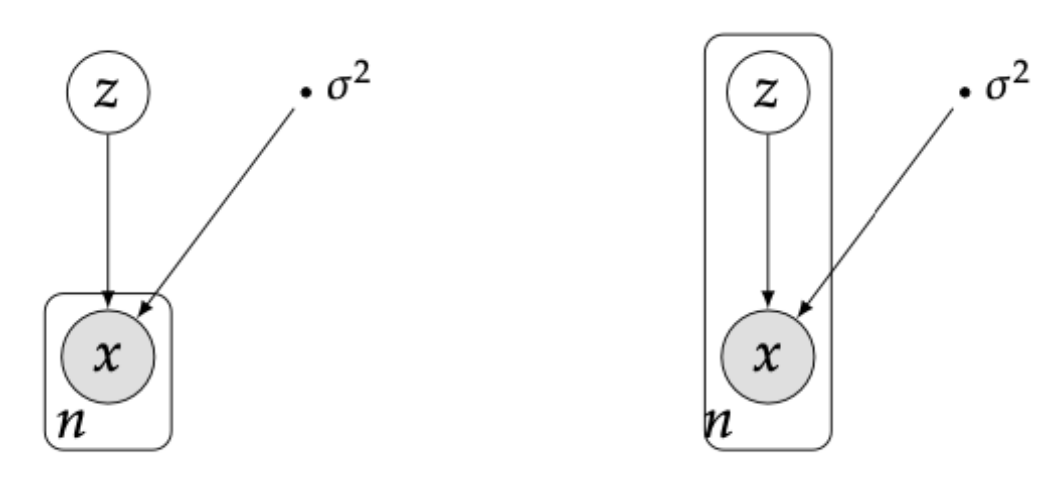
- Observed nodes (Random variables of which we know the value) are **coloured** or shadowed
- Plated nodes are a shorthand for a collection of nodes.
- Deterministic quantities represented as small solid circles. These are fixed parameters of the model (represented graphically as $\dot{\alpha}$)

Example: *Mixture of Gaussians*

As an example suppose to have a RV z that follows a discrete distribution and another RV x that is normally distributed with mean $\mu(z)$, depending on the value of z, and variance σ^2. This model is known as a **mixture of gaussians**.
We can write the joint distribution as:

$$p(x,z) = p(z)p(x|z) = p(z)\mathcal{N}(x|\mu(z),\sigma^2)$$

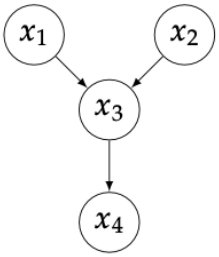
with $p(x|z) = \mathcal{N}(x|\mu_z,\sigma^2)$
Typically, in this scenario we have at our disposal n observations of the variable x, $\bar{x} = x_1,\ldots,x_n$, while the corresponding variable z is unobserved (hence z is a **latent variable**). We typically want to compute $p(z|\bar{x})$.
We can have two scenarios here: in the first $\bar{x}$ are sampled from a single realization of the variable z, while in the second one each x_i may have been drawn with a different z_i, hence we are interested in computing $p(z_i!x_i)$:



3.2.1 Sampling and reasoning in BN

Recalling the initial example

$$p(x_1,x_2,x_3,x_4) = p(x_4|x_3)p(x_3|x_2,x_1)p(x_2)p(x_1)$$

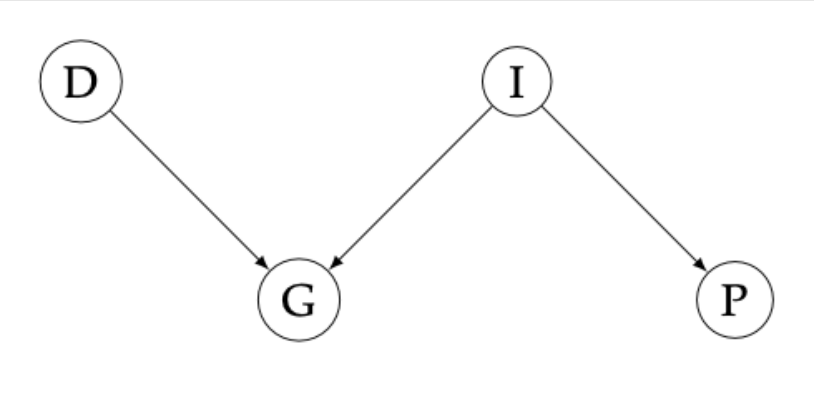


Ancestral sampling is a sampling technique that allows us to sample from a given distribution, if we know its factorization.

One can start to sample from the top of the network (our **ancestrals**), in the example x_1 and x_2, then move to their children and sample, in the example, x_3 from $p(x_3|x_2,x_1)$ using the values for x_1 and x_2 that have already been sampled, and so on. It is a simple and effective way to sample probability distribution, provided it is easy to sample from wach marginal and conditional.

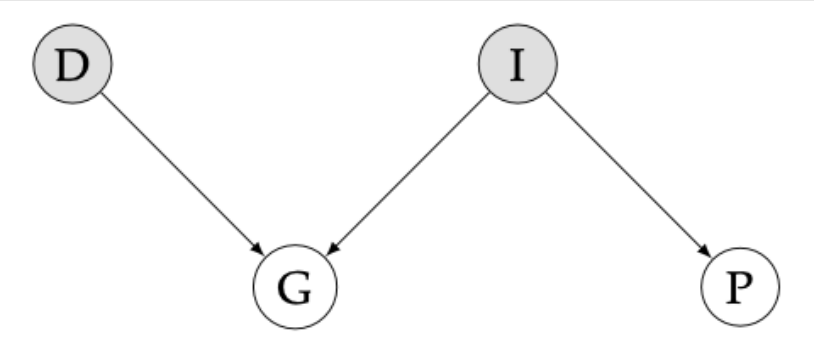
Example: *Reasoning with BN*

We consider here a model of students' grade in an exam.

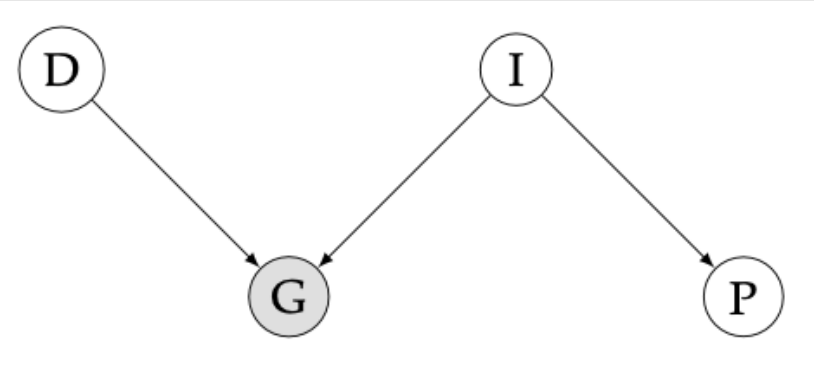


The difficulty of the exam (D) and the intelligence of the student (I) are independent, but the grade you get when you take an exam (G) is dependent on both (studying of course helps too). Having passed a hard exam (P) likely depends on the student's abilities but not on the difficulty and the grade of the current exam. We have three different forms of reasoning here:

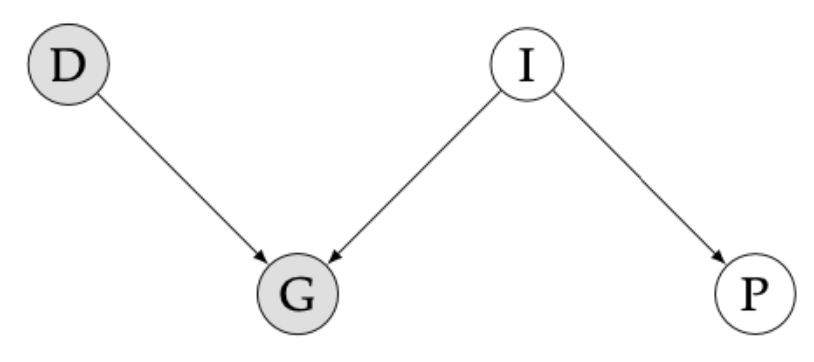
- **Causal Reasoning**: We observe something (the difficulty of the exams, how intelligent we are) and we infer the grade and if we have passed a hard exam. The reasoning goes from top to bottom.



- **Evidential Reasoning**: If we instead observe the grade, we might want to get information regarding the difficulty of the exams and how intelligent the student is. The reasoning goes from bottom to top.



- **Intercausal Reasoning**: If we observe the grade and the difficulty of the exams, we might want to infer the intelligence of the student. The reasoning goes from the middle to the top.



Bayesian networks of course can get very large and probabilistic inference becomes a complicated and important task to perform in the real world.
Remark: note that dependency in a BN can model both causality and correlation, and the two are not necessarily distinguishable within the model.

3.2.2 Conditional Independence in BN

Definition: *Conditional Independence*

Consider the RVs a, b, c. We say that a is **conditional independent** of b given c (written $a \perp\!\!\!\perp b|c$) if:

$$p(a,b|c) = p(a|c)p(b|c)$$

or equivalently:

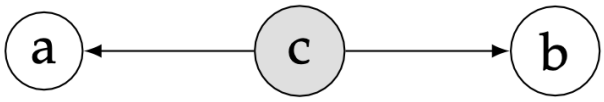
$$p(a|b,c) = p(a|c)$$

We have three scenarios to consider in Bayesian Networks to understand conditional independence.

Tail to Tail

Consider the RVs a, b, c. We say that a and b are tail to tail with c if:

$$p(a,b,c) = p(a|c)p(b|c)p(c)$$



We know that this Bayesian network implies the following factorization:

$$p(a,b) = \sum_c p(a|c)p(b|c)p(c) \neq p(a)p(b)$$

Also

$$p(a,b|c) = \frac{p(a,b,c)}{p(c)} = \frac{p(a|c)p(b|c)p(c)}{p(c)} = p(a|c)p(b|c)$$

Which means that $a \perp\!\!\!\perp b \not\perp\!\!\!\perp b|c$.

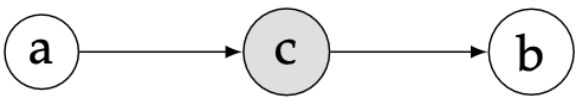
Head to Tail

Consider the RVs a, b, c. We say that a and b are head to tail with c if:

$$p(a,b,c) = p(a)p(c|a)p(b|c)$$

or

$$p(a,b,c) = p(b)p(c|b)p(a|c)$$



Again, we can use this factorization and study independence of our variables:

$$p(a,b) = \sum_c p(a)p(c|a)p(b|c) = p(a)\sum_c p(b|a)p(a)$$

Also

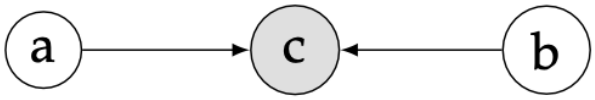
$$p(a,b|c) = \frac{p(b|c)p(c|a)p(a)}{p(c)} = p(b|c)p(a|c)$$

Which means, again, that $a \perp\!\!\!\perp b \not\perp\!\!\!\perp b|c$.

Head to Head

Consider the RVs a,b,c. We say that a and b are head to head with c if:

$$p(a,b,c) = p(a)p(b)p(c|a,b)$$



$$p(a,b) = \sum_c p(a)p(b)p(c|a,b) = p(a)p(b)$$

Also

$$p(a,b|c) = \frac{p(c|a,b)p(b)p(a)}{p(c)} \neq p(b|c)p(a|c)$$

Which means, this time, that $a \perp\!\!\!\perp banda \not\perp\!\!\!\perp b|c$.
Notice that also $a \perp\!\!\!\perp b|d, \forall d$ descendant of c.

Formalization

Formalizing the notion of conditional independence on Bayesian Network that we just saw,

Definition:

Given RVs a,b,c a path from a to b is **blocked** by c if:

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

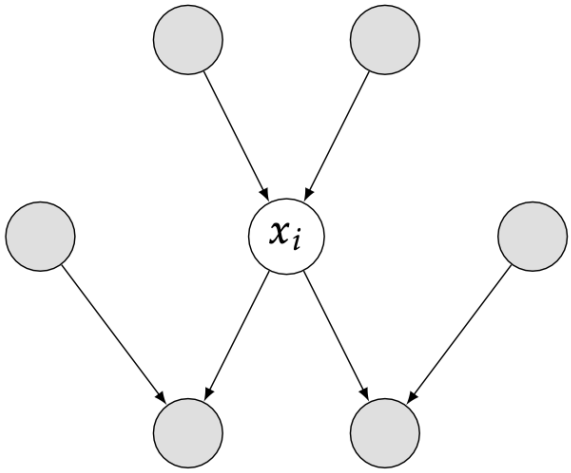
Proposition: Let A,B,C subsets, if all paths from a node in A to a node in B are blocked by a node in C then $A \perp\!\!\!\perp B|C$.

Markov Blanket

Consider a node x_i on the network and condition on everything else, i.e. on $x_{-i} = \{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n\}$. Which nodes will remain in the conditioning set? If we make the computation

$$p(x_i|x_{-i}) = \frac{p(x_1,\ldots,x_n)}{p(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)} = \frac{\prod_j p(x_j|pa_j)}{\sum_{x_i}\prod_j p(x_j|pa_j)}$$

In the denominator, each term in which x_i does not appear either as x_j or in $+pa_j$ will get out of the summation and cancel with the respective term in the numerator. So the only nodes that remain are thos belonging to pa_i, or those for which $x_i \in pa_j$ is known as the **Markov Blanket** of x_i (it contains parents, children and co-parents of x_i). Each node conditioned on its Markov blanket is independent of the rest of the network.



3.2.3 Naive Bayes

Naive Bayes is one of the simplest classification algorithms. Let's suppose to have a certain set of features $x_1,\dots,x_n$ dependent on a certain class c. We have the following generative model (conditioned on class c).

$$p(x_1,\dots,x_n|c)$$

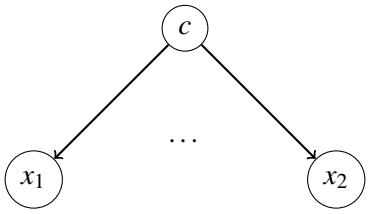
What we want to compute when we need to solve a classification task, is actually

$$p(c|x_1,\dots,x_n)$$

where $x_1,\dots,x_n$ are the features of one new data point, of which we would like to compute the probability of belonging to a certain class c. Modelling this can be very challenging.

Naive Bayes assumption We assume that out features are independent given the class c, i.e.

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



How do we train this model? Given that we have a set of data, we consider only the x_i feature of the points of the dataset belonging to class c and fit a parametric model of $p(x_i|c;\theta)$ by means of Maximum Likelihood or other algorithms. Since we are fitting probabilistic models with one single variable, such fit is in general pretty easy.

Then, using Bayes Theorem, we know that

$$p(c|x_1,\dots,x_n) \propto p(x_1,\dots,x_n|c)p(c) = p(c)\prod_{i=1}^{n} p(x_i|c)$$

Where we also estimate $p(c)$ by taking the fraction of points belonging to class c in our dataset.

Instead of using the estimate $p(c|x_1,\ldots,x_n)$, it is common practice in the context of Naive Bayes to look at the ratio

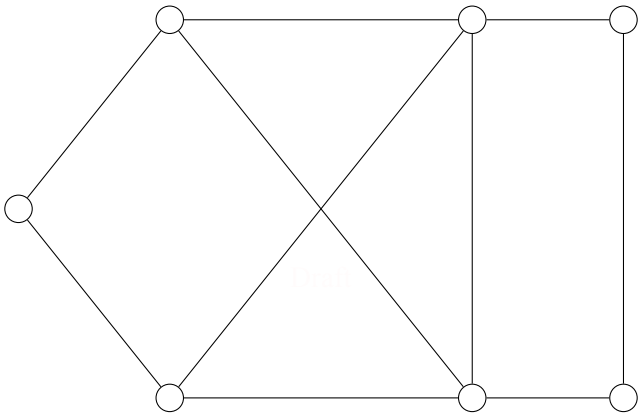
$$\frac{p(c=0|x_1,\ldots,x_n)}{p(c=1|x_1,\ldots,x_n)}$$

whose logarithm is known as **log-odd ratio**. Hence we can classify a certain point as belonging to class 0 if the ratio is greater than 1 (or greater than a certain classification threshold t) and belonging to class 1 otherwise. For multiclass problems, we can assign a point to the class of maximum conditional probability.

An example of appliation of Naive Bayes is the domain of document classification.

3.3 Random Markov Fields

Random Markov Fields are an undirected graph representation of a graphical model. Nodes correspond to RV and edges to dependency. Understanding conditional independence in Markov Random Fields is easy. The way that edges model dependency, however, is more complicated.



Conditional independence on Markov Random Fields

proposition 1. *Consider three subsets A,B,C. $A \perp B|C$ iff all paths from a node $a \in A$ to any node $b \in B$ pass from C (i.e. are blocked by a node in C).*

Definition: ***Markov Blanket***

A **Markov Blanket** in a Markov Random Field for a given x_i is the set of neighbors of x_i.

Factorization

Consider two nodes x_i and x_j. If there is no edge from x_i to x_j we know that $x_i \perp x_j|x_{-\{i,j\}}$. Therefore

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

and x_i,x_j belong to different factors.

This means that nodes connected by an edge should belong to the same factor(s). Extending the reasoning, if a subgraph is fully connected, then all its nodes should be in the same factor. Therefore, graphically speaking, factors correspond to **maximal cliques** in the graph.

Definition: *Clique*

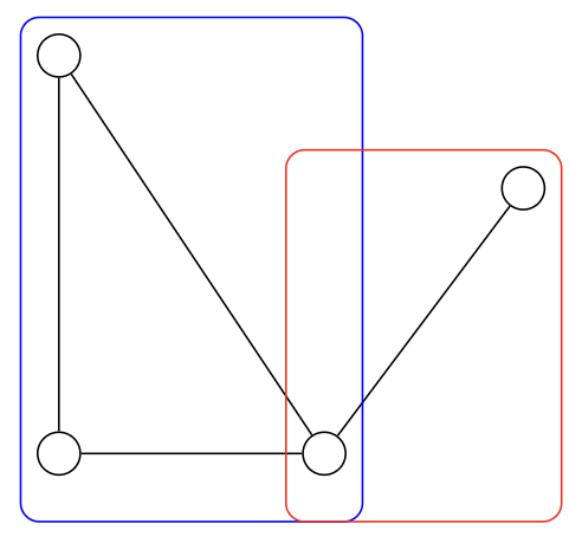
A **clique** is a fully connected subgraph.

Definition: *Maximal Clique*

A **maximal clique** is a clique that cannot be extended.

Example:

The following plot highlights the two maximal cliques of an undirected graph



Let's now consider $\mathscr{C} = \{\text{set of maximal cliques}\}$ and $x_C = \{x | x \in C, C \in \mathscr{C}\}$.
Then the factorization of a Markov Random Field is obtained as

$$p(x) = \frac{1}{Z} \prod_{c \in \mathscr{C}} \Psi_C(x_C)$$

Where Z is a normalization constant, and $\Psi_C(x_C)$ is some function evaluated on the variables that belong to the clique C.

Definition: *Potential Function*

$\Psi_C(x_C) \geq 0$ is called the **potential function** and it must have a finite integral, $\int \Psi_C(x_C)\, dx_C < \infty$.

Definition: *Partition Function*

$Z = \int \prod_C \Psi_C(x_C)\, dx_C < \infty$ is the **Partition Function**.

Being defined by a multidimensional integral, Z is typically hard to compute.

Therefore computing $p(x)$ might be too complicated, but notice that if we want to compute conditionals, which are ratios of marginal probabilities of $p(x)$, then Z cancels out, which makes them much easier to compute. Instead, if we want to compute marginals on few variables, we can work with the unnormalized potentials and normalize at the end. In this way we don't have to compute Z directly and we make our computations feasible.

One way to guarantee that $\Psi_C(x_C) \geq 0$ is to model $\Psi_C(x_C) = \exp(-E(x_C))$ where E is known as **energy**. This is called the **Boltzmann distribution**.

Remark. By convention, low energy corresponds to high potential.

The Boltzmann distribution is easy to work with because

$$\Psi_{C1}(x_{C1})\Psi_{C2}(x_{C2}) = \exp(-E_1(x_{C1}))\exp(-E_2(x_{C2})) = \exp(-E_1(x_{C1}) - E_2(x_{C2}))$$

Remark. There are things that one can model with Bayesian Networks that one cannot do with Markov Random Fields and viceversa.

Examples of Markov Random Fields are the Ising Model, which can also be used to denoise images, and Boltzmann Machines.

4 Exact Inference in Probabilistic Graphical Models

In the context of inference, out task is, given a joint distribution $p(X) = p(x_1, \ldots, x_n)$ to compute marginals or conditionals of the distribution.

We formalize this by considering two disjoint subsets of r.v.: $y \subseteq x, z \subseteq x$ s.t. $y \cap z = \emptyset$ and $y \cup z \subseteq x$.

Given that we observe y, we want to compute the conditional

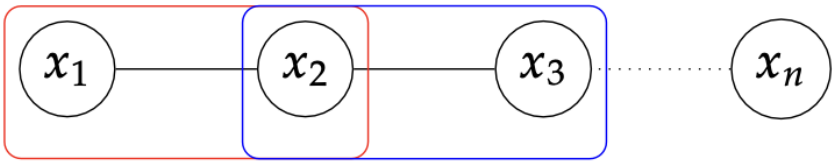
$$\begin{aligned} p(z|y=\bar{y}) &= \sum_x p(z,x'|y=\bar{y}) discreter.v. \\ &= p(z|y=\bar{y} = \int_x p(z,x'|y=\bar{y})) continuousr.v. \end{aligned}$$

being x' s.t. $x' \cup y \cup z = x$.

Remark: this summation can be of the order of $O(\mathscr{K}^m)(|x'| = m)$, hence obtaining this marginalization is computationally challenging!

Our goal is to carry out this computation efficiently, by leverage the factorization implied by the PGM.

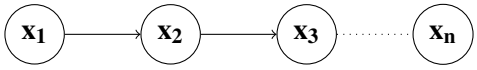
As an example, consider a Markov Random Field where variables are connected in a chain (colored rectangles represent the cliques):



The factorization implied by this PGM is:

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

Consider now the equivalent model in case of a Bayesian Network:



The factorization in this case reads as

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_2) \ldots p(x_n|x_{n-1})$$

Observation: ***Markov processes***

Chain structures like the previous are common because they repre- sent temporal series (more specifically, they are Markov processes).

Suppose that, given the model above, we want to compute the marginal

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

with $1 < k < n$. In principle this has complexity $O(\mathcal{K}^{n-1})$.

However, plugging the factorization implied by the Markov Random Field, we get(using the distributive laws of sum and product):

$$\begin{aligned} p(x_k) &= \sum_{x\_k} \frac{1}{Z} \psi_{1,2}(x_1,x_2) \dots \psi_{n-1,n}(x_{n-1},x_n) \\ &= \frac{1}{Z} \sum_{x_{k-1}} \psi_{k-1,k}(x_{k-1},x_k) \dots \left[\sum_{x_2} \psi_{2,3}(x_2,x_3) \left[\sum_{x_1} \psi(x_1,x_2)\right]\right] + \\ &+ \sum_{x_{k+1}} \psi_{k,k+1}(x_k,x_{k+1}) \dots \left[\sum_{x_{n-1}} \psi_{n-2,n-1}(x_{n-2},x_{n-1}) \left[\sum_{x_n} \psi_{n-1,n}(x_{n-1,x_n})\right]\right] \end{aligned}$$

Note that complexity is now $O(n \cdot k)$, i.e. linear in n.

We can define a dynamic programming algorithm, called **message-passing algorithm**, in which the information from the graph is summarized by local edge information.

We break the chain in two parts, the past and the future w.r.t. x_k, and we define the following:

- forward message: $\mu_\alpha(x_k) = \sum_{x_{k-1}} \psi_{k-1,k}(x_{k-1},x_k) \cdot \mu_\alpha(x_{k-1})$, with base case $\mu_\alpha(x_1) = 1$
- backward message: $\mu_\beta(x_k) = \sum_{x_{k+1}} \psi_{k,k-1}(x_k,x_{k+1}) \cdot \mu_\beta(x_{k+1})$, with base case $\mu_\beta(x_n) = 1$

In this algorithm, each node sends a message to its neighbors, and the messages are computed by the above formulas.

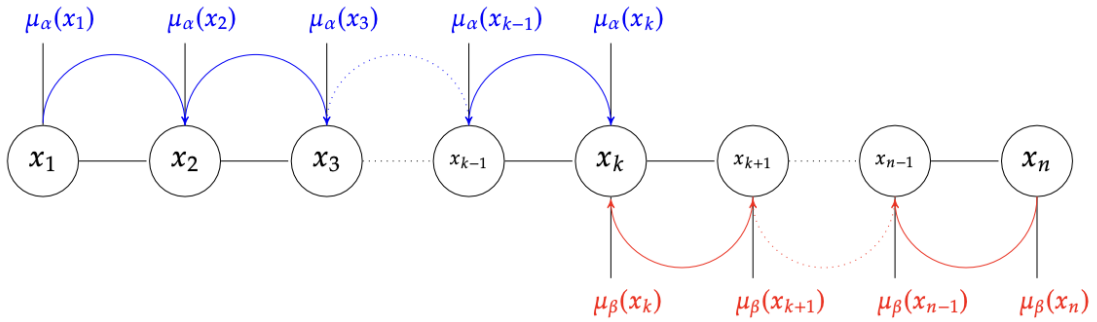


Figure 4.1: Message-passing algorithm

Once we have both $\mu_\alpha(x_k)$ and $\mu_\beta(x_k)$, we can observe that:

$$p(x_k) = \frac{1}{Z_k} \cdot \mu_\alpha(x_k)\mu_\beta(x_k) \propto \mu_\alpha(x_k)\mu_\beta(x_k)$$

with the normalization constant Z_k computed as $Z_k = \sum_{x_k} \mu_\alpha(x_k)\mu_\beta(x_k)$.

Summarizing, the idea behind this algorithm is to start from the extremes of the chain and pass messages (forward and backward) until the other end is reached. Once there, messages are combined to obtain an (un)normalized marginal distribution.

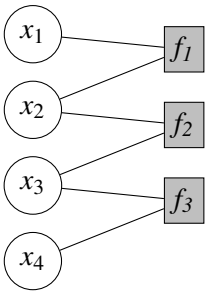
4.1 Factor Graphs

Our goal in what follows is to extend what we have done for chains to more general graph structures, i.e. trees and politrees. In order to do so, we need to introduce the formalism of **Factor Graphs**.

The idea behind them is to expose in a more clear way what are the factors and what are the variables. For this reason, Factor Graphs are bipartite graphs (i.e. nodes are divided in two classes), in which we have:

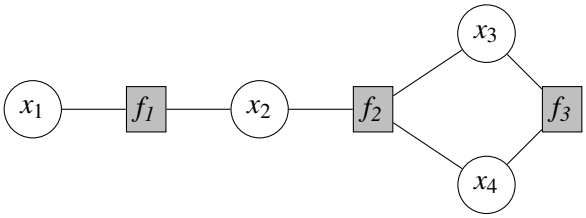
Variable nodes	Factor nodes
x	f

The canonical way of represent bipartite graphs is the following:



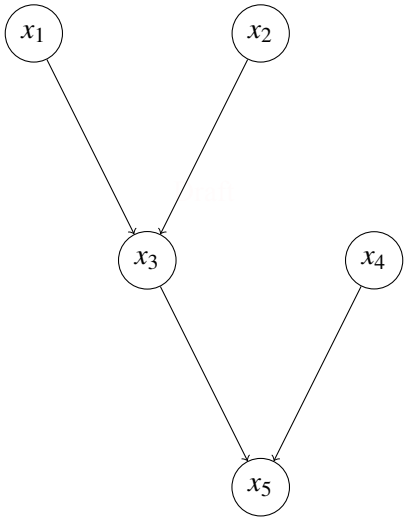
That is, all variable nodes are put on the left and all factor nodes on the right.

A more convenient, equivalent, representation could be:



4.1.1 From Bayesian Networks to Factor Graphs

Consider the following Bayesian Network:



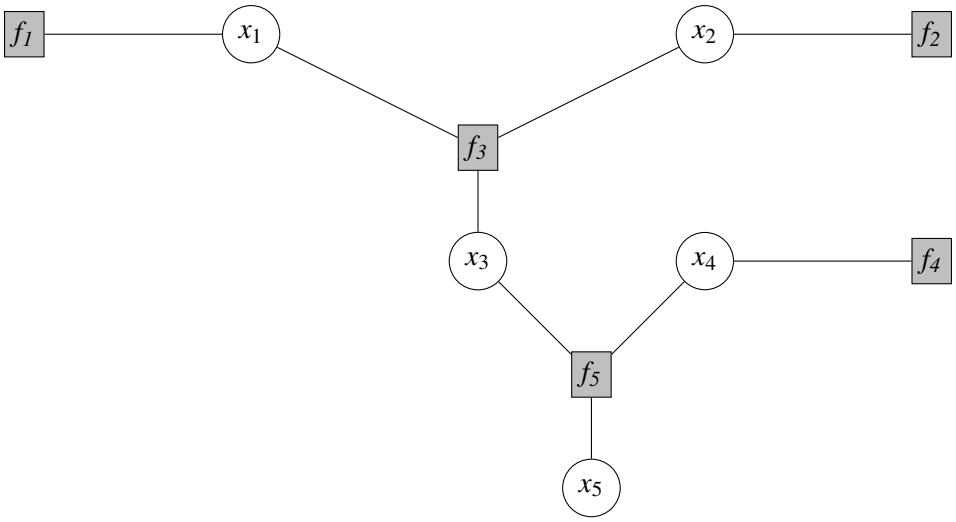
Which corresponds to the factorization:

$$\begin{aligned} p(x) &= p(x_1)p(x_2)p(x_3|x_1,x_2)p(x_4)p(x_5|x_3,x_4) \\ &= f_1(x_1)f_2(x_2)f_3(x_1,x_2,x_3)f_4(x_4)f_5(x_3,x_4,x_5) \end{aligned}$$

Fundamentally, the idea is to have a factor node (connected to the proper variable nodes) for each term of the factorization implied by the Bayesian Network.

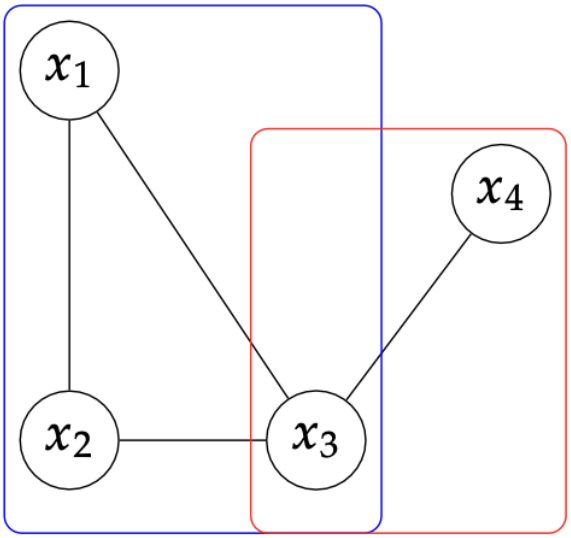
Formally: $\forall p(x_j|pa_j) \rightarrow f_j \curvearrowright x_j \cup pa_j$.

Hence the Factor Graph equivalent to the Bayesian Network above is:



4.1.2 From Markov Random Fields to Factor Graphs

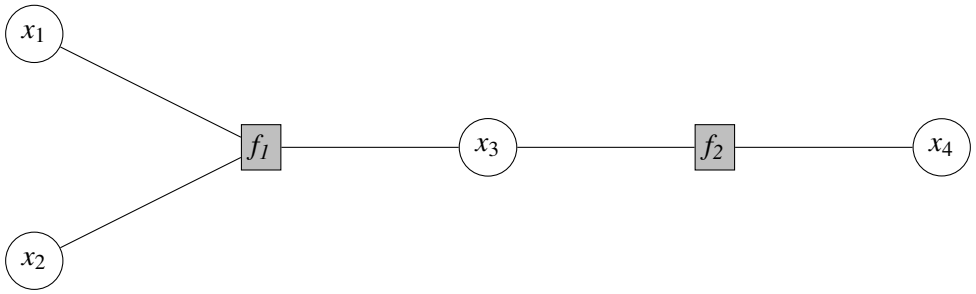
Consider the following Markov Random Field (rectangles correspond to the cliques):



Which implies the factorization:

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

This leads to the following Factor Graph:

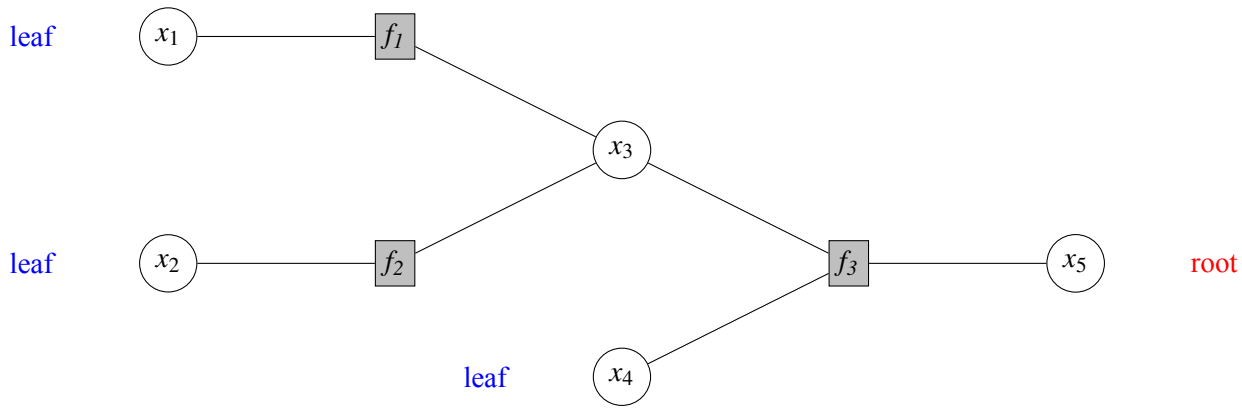


Hence, the conversion from Markov Random Fields to Factor Graphs works as: $\forall c \in \mathscr{C} \rightarrow f_c(x_c) \sim \psi_c(x_c)$ (equality does not hold because of the normalization constant).

So, whether we are starting from a Bayesian Network or from a Markov Random Field, we can convert our PGM into a Factor Graph and perform message passing on it, without loss of generality.

4.2 Sum Product Algotithm

Our goal is now to do inference in Factor Graphs, generalizing to more general graph structures what we did with chains previously. Consider the following Factor Graph:



The joint probabbility distribution implied by thie Factor Graph is:

$$p(x) = f_1(x_1,x_3)f_2(x_2,x_3)f_3(x_3,x_4,x_5)$$

Since the above graph is actually a tree, we can identify the **root** node (choice is arbitrary, here it is x_5) and consequently the **leaves** (in the example, x_1,x_2,x_4). Recall that there is a unique path from the root to any of the leaves and that following all those paths we visit all the nodes in the tree.

The message passing algorithm that we are going to detail is called **Belief Propagation** and works in Factor Graphs which are trees (i.e., without any loop).

Assume that the root is the node of which we want to compute the marginal (this is not necessarily the case, indeed after performing forward and backward pass we have sufficient information to compute the marginal w.r.t. each node in graph).

So, let's consider the marginal probability w.r.t. x_5:

$$\begin{aligned} p(x_5) &= \sum_{x_1,x_2,x_3,x_4} p(x_1,x_2,x_3,x_4,x_5) \\ &= \sum_{x_3,x_4} f_3(x_3,x_4,x_5) \left[\sum_{x_1} f_1(x_1,x_3)\right] \left[\sum_{x_2} f_2(x_2,x_3)\right] \end{aligned}$$

Take $\sum_{x_1} f_1(x_1,x_3)$: we can see this as a message going from factor f_1 to variable x_3. We denote it by $\mu_{f_1 \to x_3}(x_3)$.

Analogously, $\sum_{x_2} f_2(x_2,x_3) \doteq \sum_{x_1} f_1(x_1,x_3) \cdot \sum_{x_2} f_2(x_2,x_3)$.

Finally, the full summation can be seen as a message going from factor f_3 to variable x_5, i.e. $\mu_{f_3 \to x_5}(x_5) \doteq \sum_{x_3,x_4} f_3(x_3,x_4,x_5)[\sum_{x_1} f_1(x_1,x_3)][\sum_{x_2} f_2(x_2,x_3)]$.

We can observe the following:

- messages from factors to variables include a summation (an integral in the case of continuous r.v.) over all the variables on which the factor depends, except for the one we are sending the message to;
- messages from variables to fators collect via multiplication all the incoming messages from the other factors, except from the one we are sending the message to.

Formalizing, we call:

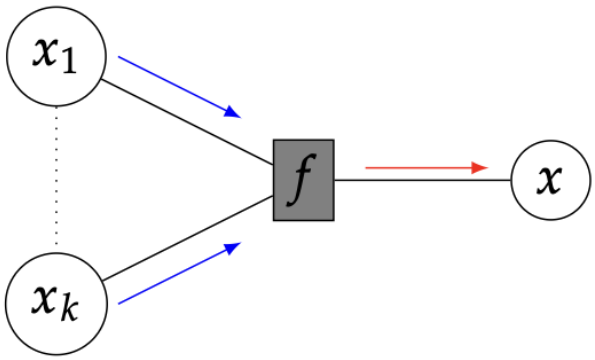
- set of neighboring nodes of variable x:

$$\mathcal{N}(x) = \{f_1, \ldots, f_k | f_i \text{ is connected to } x\}$$

- set of neighboring nodes of factor f:

$$\mathcal{N}(f) = \{x_1, \ldots, x_k | x_i \text{ is connected to } f\}$$

The scenario is the following:

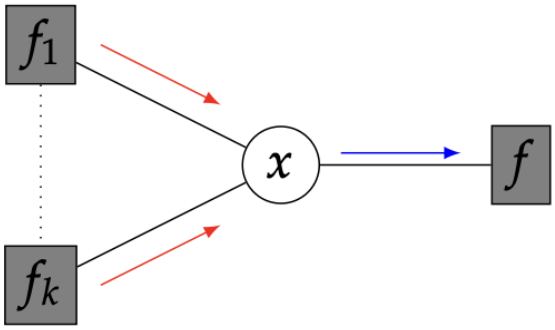


The red arrow in the figure above corresponds to the message $\mu_{f \to x}(x)$. Following what we have seen before, this is computed as:

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

($\mu_{x_i \to f}(x_i)$ are the blue arrows in the figure above).

Consider now the symmetric situation:



In this case:

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

Base cases are defined as:

- x leaf node: $\mu_{x \to f}(x) = 1$
- f leaf node: $\mu_{f \to x}(x) = f(x)$

The **Sum-product** algorithm works in two steps:

- forward pass: messages are sent from the leaves to the root;

- backward pass: messages are sent from the root back to the leaves.

In this way each edge will be traversed both by a forward and a backward message.

After these two passages, all possible messages are computed and we can obtain marginals and conditionals.

Consider the problem of computing the marginals, there are two cases in this scenario:

- computation of the marginal of a variable node x:

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

- computation of the marginal of a factor node $f(\bar{x})$ (i.e., $p(\bar{x})$ where $\bar{x} = \mathcal{N}(f)$):

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

Note that both these computations can be carried out once we have all the messages.

Moreover, the sum-product algorithm works also when the joint distribution is not normalized, in that case the obtained marginals have to be normalized.

Consider now the problem of computing conditionals on $y = \hat{y}, y \subseteq \{x_1, \ldots, x_n\}$, i.e. $p(x|y = \hat{y})$.

The first step is **clamping** y **to** $\hat{y}$, that is $p(x, x_1, \ldots, x_k, y = \hat{y})$ (i.e. we fix $y = \hat{y}$ in the joint).

Then we run the sum-product algorithm with y_j fixed to $\hat{y} \forall y_j \in y$ (practically, when running the message passing algorithm, whenever we have a variable in y we consider its corresponding state $\hat{y}$, instead of summing over it):

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

Finally, we compute the desired conditional as:

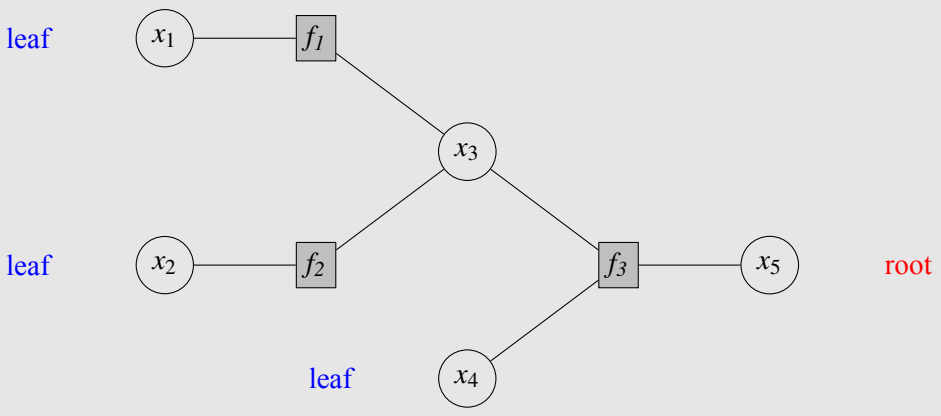
$$p(x|y = \hat{y}) = \frac{p(x, y = \hat{y})}{p(\hat{y})}$$

with $p(\hat{y}) = \sum_x p(x, y = \hat{y})$.

In this context, we can see $p(x, y = \hat{y})$ as an unnormalized conditional and $p(\hat{y})$ as its normalization constant.

Example:

consider the following factor graph:



that represents the unnormalized joint distribution

$$p(x_1,x_2,x_3,x_4,x_5) \propto f_1(x_1,x_3)f_2(x_2,x_3)f_3(x_3,x_4,x_5)$$

Variables are binary, i.e. $x_i \in \{0,1\}$ and factors are defined as:

$$f_1(x_1,x_3) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \text{ where } (x_1,x_3) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

$$f_2(x_2,x_3) = \begin{bmatrix} 0.1 \\ 0.4 \\ 0.2 \\ 0.3 \end{bmatrix} \text{ where } (x_2,x_3) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

$$f_3(x_3,x_4,x_5) = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix} \text{ where } (x_3,x_4,x_5) = \begin{matrix} (0,0,0) \\ (0,0,1) \\ (0,1,0) \\ (0,1,1) \\ (1,0,0) \\ (1,0,1) \\ (1,1,0) \\ (1,1,1) \end{matrix}$$

After arbitrarily choosing x_5 as root node, we can start by computing forward message edge by edge:

Forward pass

$$\begin{aligned}
\mu_{x_1 \to f_1}(x_1) &= 1 \\
\mu_{x_2 \to f_2}(x_2) &= 1 \\
\mu_{f_1 \to x_3}(x_3) &= \textstyle\sum_{x_1} f_1(x_1,x_3)\mu_{x_1 \to f_1}(x_1) = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix} \\
\mu_{f_2 \to x_3}(x_3) &= \textstyle\sum_{x_2} f_2(x_2,x_3)\mu_{x_2 \to f_2}(x_2) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \\
\mu_{x_3 \to f_3}(x_3) &= \mu_{f_1 \to x_3}(x_3)\mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix} \\
\mu_{x_4 \to f_3}(x_4) &= 1 \\
\mu_{f_3 \to x_5}(x_5) &= \textstyle\sum_{x_3,x_4} f_3(x_3,x_4,x_5)\mu_{x_3 \to f_3}(x_3)\mu_{x_4 \to f_3}(x_4) = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix}
\end{aligned}$$

Backward pass

$$\begin{aligned}
\mu_{x_5 \to f_3}(x_5) &= 1 \\
\mu_{f_3 \to x_4} &= \textstyle\sum_{x_3,x_5} f_3(x_3,x_4,x_5)\mu_{x_5 \to f_3}(x_5)\mu_{x_3 \to f_3}(x_3) = \begin{bmatrix} 0.054 \\ 0.276 \end{bmatrix} \\
\mu_{f_3 \to x_3} &= \textstyle\sum_{x_4,x_5} f_3(x_3,x_4,x_5)\mu_{x_5 \to f_3}(x_5)\mu_{x_4 \to f_3}(x_4) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \\
\mu_{x_3 \to f_1} &= \mu_{f_3 \to x_3}(x_3)\mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.09 \\ 0.49 \end{bmatrix} \\
\mu_{x_3 \to f_2}(x_3) &= \mu_{f_3 \to x_3}(x_3)\mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix} \\
\mu_{f_1 \to x_1}(x_1) &= \textstyle\sum_{x_3} f_1(x_1,x_3)\mu_{x_3 \to f_1}(x_3) = \begin{bmatrix} 0.125 \\ 0.205 \end{bmatrix} \\
\mu_{f_2 \to x_2}(x_2) &= \textstyle\sum_{x_3} f_2(x_2,x_3)\mu_{x_3 \to f_2}(x_3) = \begin{bmatrix} 0.222 \\ 0.108 \end{bmatrix}
\end{aligned}$$

Having computed all the messages, we can calculate the marginal for all the nodes, for example:

$$p(x_5) \propto \mu_{f_3 \to x_5}(x_5) = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix}$$

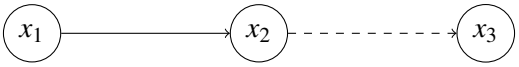
Notice that all marginals were normalized using the same constant $Z = 0.33$.

4.3 Max Plus algorithm

In what follows our task will be, given a joint density $p(x)$, to find the tuple $x^M = \arg\max_x p(x)$ (i.e. find a setting of the variables that has the largest probability and the value of that probability).

To do so, we will use the **max-plus algorithm**.

As an example, consider a chain structure described by a Markov Random Field:



whose factorization reads as

$$p(x) = \frac{1}{Z}\Psi_{1,2}(x_1,x_2)\dots\Psi_{n-1,n}(x_{n-1},x_n)$$

Our goal is to maximize $p(x)$ w.r.t. x_n.

It is possible to distribute the factorization so that only local computations are required:

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

This happens because of the distributive properties of the max, indeed it holds that, given $a > 0$:

- the max distributes over the product

$$\max(ab,ac) = a\cdot\max(b,c)$$

- the max distributes over the sum

$$\max(a+b,a+c) = a+\max(b,c)$$

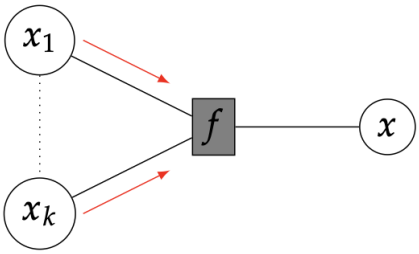
This allows us to take an approach similar to the one used in the sum-product algorithm.

Actually we will maximize $\log p(x)$. hence turning the product into sum of logarithms (and this justifies the name max-plus), i.e. our objective is to maximize

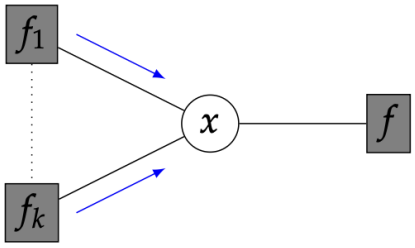
$$\log p(x) = \sum_i \log\Psi_{i,i+1}(x_i,x_{i+1}) - \log Z$$

This saves us from product of factors that are possibly very small, since they are probabilities.

Max-plus algorithm is very similar to sum-product: essentially max replaces sum and plus replaces product. This leads to the following scenarios:



$$\mu_{f\to x}(x) = \max_{x_1,\ldots,x_k}\left[\log f(x,x_1,\ldots,x_k) + \sum_{x_i\in\mathcal{N}(y)\setminus x}\mu_{x_i\to f}(x_i)\right]$$



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

With base cases:

- x leaf node: $\mu_{x\to f}(x) = 0$
- f leaf node: $\mu_{f\to x}(x) = \log f(x)$

In order to find the maximum of the joint distribution, that is

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

we just need to run the forward pass of the algorithm (i.e. we propagate messages from the leaves up to the root, as in the sum-product algorithm).

Remark. The result will be the same irrespective of which node is chosen as the root.

However, we want to find also the configuration of the variables for which the maximum is achieved. This can be done by applying a slight variation in the forward pass, meaning that we also need to keep track of which values of the variables gave rise to the maximum state of each variable. So during the forward pass we will also store:

$$\Phi_{f\to x}(x) = \arg\max_{x_1,\ldots,x_k\in\mathcal{N}(f)\setminus x}\left[\log f(x,x_1,\ldots,x_k) + \sum_{x_i\in\mathcal{N}(f)\setminus x}\mu_{x_i\to f}(x_i)\right]$$

Hence, since at the end of the forward pass we know the most probable value of the root node

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

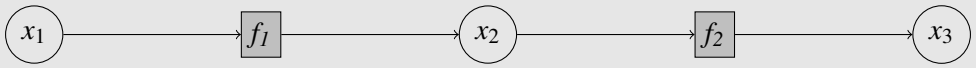
we can exploit the additional information that we stored to retreive the most probable state of the internal nodes by **back-tracking**, i.e. by following Φ. For example, assuming the variable nodes connected to f are $x_1,\ldots,x_k,x_{root}$, in the first backtracking step we will fix the value of $x_1,\ldots,x_k$ according to

$$\Phi_{f\to x_{root}}(x_{root}^{max}) = \left\{\begin{array}{c} x_1 \to x_1^{max} \\ \ldots \\ x_K \to x_k^{max} \end{array}\right\}$$

and then propagate backwards following Φ from the other factor nodes connected to $x_1,\ldots,x_k$.

Example:

Consider the following chain:



in which variables are binary, i.e. $x \equiv x_1, x_2, x_3 \in \{0,1\}$ and factors are defined as:

$$f_1(x_1,x_2) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \text{ where } (x_1,x_2) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

$$f_2(x_2,x_3) = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.2 \\ 0.2 \end{bmatrix} \text{ where } (x_2,x_3) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

We start by computing the forward messages edge by edge:

$$\mu_{x_1 \to f_1}(x_1) = 0$$

$$\mu_{f_1 \to x_2}(x_2) = \max_{x_1} [\log f_1(x_1,x_2) + \mu_{x_1 \to f_1}(x_1)] = \begin{bmatrix} \log 0.3 \\ \log 0.4 \end{bmatrix} \text{ where } x_2 = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$\mu_{x_2 \to f_2}(x_2) = \mu_{f_1 \to x_2}(x_2)$$

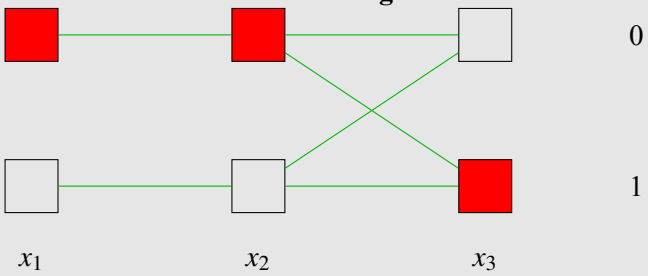
$$\mu_{f_2 \to x_3}(x_3) = \max_{x_2} [\log f_2(x_2,x_3) + \mu_{x_2 \to f_2}(x_2)] = \begin{bmatrix} \log 0.2 + \log 0.4 \\ \log 0.5 + \log 0.3 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

as well as backtracking functions:

$$\Phi_{f_1 \to x_2}(x_2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ where } x_2 = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$\Phi_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

At this point, we can write down the **lattice/trellis diagram**:



This kind of diagram shows explicitly the K (2 in this case) possible states (one per row of the diagram) for each of the variables x_i in the model. The two paths through the lattice correspond to configurations that give the global maximum of the joint probability distribution.
If, for example, we get that:

$$\max_{x_3} \mu_{f_2 \to x_3}(x_3) = \log 0.5 + \log 0.3$$
$$\arg\max_{x_3} = 1$$

then we can obtain the tuple that maximizes our joint density by following the path backward in the diagram above (we are guaranteed that this path is unique), starting from $x_3 = 1$, i.e. $(0,0,1)$ (red path in the figure).
Note: in the context of Hidden Markov Models, the max-plus algorithm is called *Viterbi algorithm*.

4.4 Inference in general Probabilistic Graphical Models

In what follows, we want to extend what we have seen so far to general probabilistic graphical models, meaning Factor Graphs which contain loops.

In these cases, we cannot identify the root and the leaves, hence we don't have well defined forward and backward directions.

There are several possibilities:

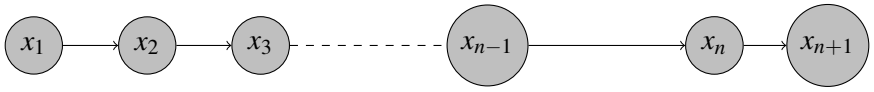
- **Junction Tree algorithm**: it roughly builds a tree over the cliques of the Factor Graph, then exact inference is done in this tree. The worst case complexity of this algorithm is exponential in the size of the largest clique (so possibly very heavy computationally);
- **Loopy Belief Propagation**: forward and backward pass of the sum-product algorithm are iterated several times, until a fix point is reached. Unfortunately there is no guarantee that the algorithm will converge. When it does, it provides an approximate answer to the inference problem;
- **Monte Carlo Sampling**: a general strategy for approximate inference based on sampling;
- **Variational Inference**: it approximates the posterior distribution with a simpler distribution belonging to a pre-specified (parametric) class, which is the closer one to the true posterior, minimizing the KL-divergence.

5 Hidden Markov Models

In what follows our goal is to model **time series data**. This kind of data are observed from a process evolving in time, typically at different time steps $x_1, x_2, \dots, x_N$ (i.e. assuming a discrete model of time). Since sequential data often arise through measurement of time series, there is correlation between observations at different time steps.

These data can come from very different domains: financial data, weather forecast data, speech data, epidemiological data.

Markov Chains are natural models for sequential data, the following is a Markov Chain of order 1:



Since all he points (up to a certain step N) are observed, the factorization by the model is:

$$p(x_1, x_2, \dots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2)\dots p(x_N|x_{N-1})$$

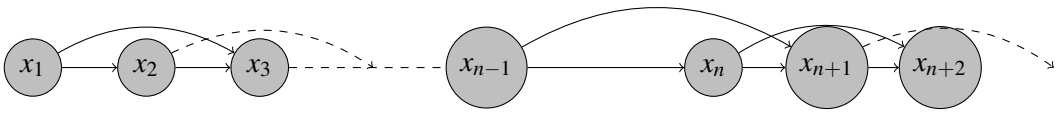
It holds that future observations are independent of all but the most recent observation:

$$x_{n+1} \perp x_1, x_2, \dots, x_n | x_n$$

A **time homogeneus** process is a process whose transiiton probability does not change in time, i.e. such that $p(x_1|x_{n-q}) = p(x_2|x_1)$.

These chains are not always the best model for describing sequential observations, indeed often there is a deeper dependency on the past, and first-order Markov chains suffer from too short memory.

In these cases, we can mode to **Markov models of order k**, where the dependency of x_n is on the previous k steps. The following is a second order Markov Chain:



In this case, the factorization of the model is:

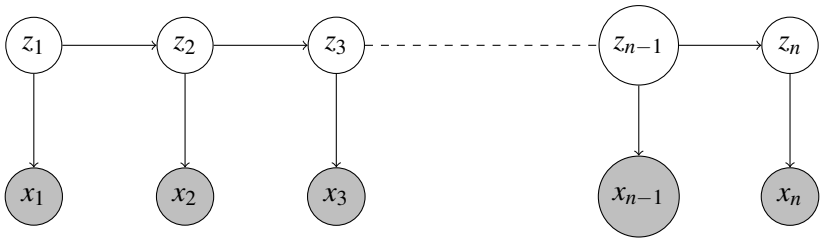
$$p(x_1, x_2, \dots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\dots p(x_{n+1}|x_n, x_{n-1})\dots$$

And it holds that:

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

Remark: if x_i are discrete, we talk about *Markov chains*; if x_i are continuous and $p(x_n|\dots)$ are Gaussian, we talk about *autoregressive models* (of order k).

If we want to build a model for sequential data that is not limited to the Markov assumption of any order, we can rely on **state space models**, which introduce **latent variables**. In terms of graphical model, we have that latent variables from a Markov chain, and each of them corresponds to an observation:



If we consider two observations x_i, x_j, then $x_i \not\perp x_j | \underline{x}$ (with $\underline{x} = x_1, \dots, x_n$) since there is not any obserbed node in the path from x_i to x_j.

In order to describe the space models like the one above, we need:

- **transition probabilities** $p(z_n|z_{n-1})$, which describe the evolution of the latent variables. They can be arranged in a matrix A s.t. $A_{ij} = p(z_n = j | z_{n-1} = i)$;
- **initial distribution** $p(z_1)$, specified by a vector π s.t. $\pi_i = p(z_1 = i)$,
- **emission probabilities** $p(x_n|z_n)$, parametrized by ψ. The emission probability for discrete x is a categorical distribution, for continuous x is typically a Gaussian or a mixture of Gaussians.

Definition: ***Hidden Markov Model***

The **Hidden Markov Model** is a specific instance of the state space model described before, in which the latent variables z_i are discrete. If instead the variables z_i are continuous and the transition probabilities $p(z_i|z_{i-1})$ are Gaussian, we talk about **Linear Dynamical System**.

Consider the parameters $\Theta = (A, \pi, \psi)$ and the variables

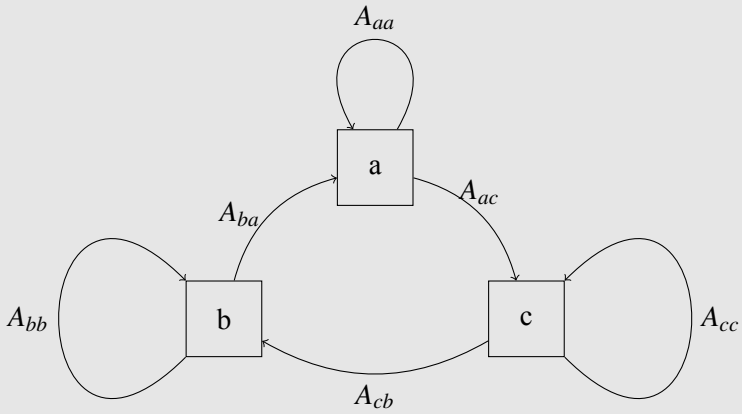
$$\underline{x}, \underline{z} = (x_1, \dots, x_n, z_1, \dots, z_n)$$

then

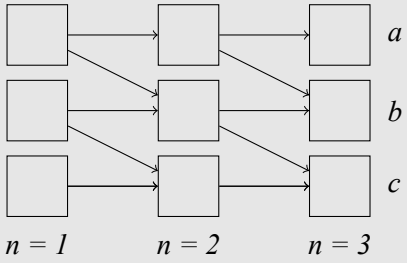
$$p(\underline{x}, \underline{z}|\Theta) = p(z_1|\pi) \left[\prod_{n=2}^{N} p(z_n|z_{n-1}, A) \right] \left[\prod_{n=1}^{N} p(x_n|z_n, \psi) \right]$$

Example:

We can graphically represent the states of $\underline{z}$ and the transition matrix as follows (note that this is not a PGM):



If we unfold the above graph over time, we obtain a sort of trellis diagram of the latent states:



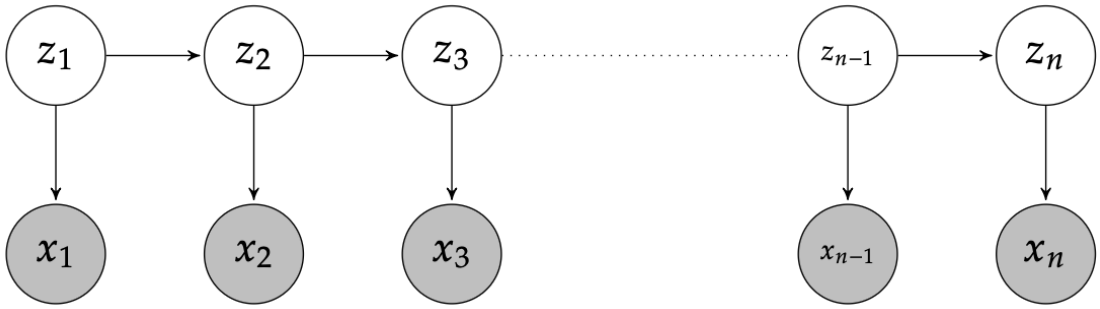
5.1 Inference in HMM

There are several class of inference problems that we may want to perform on HMMs:

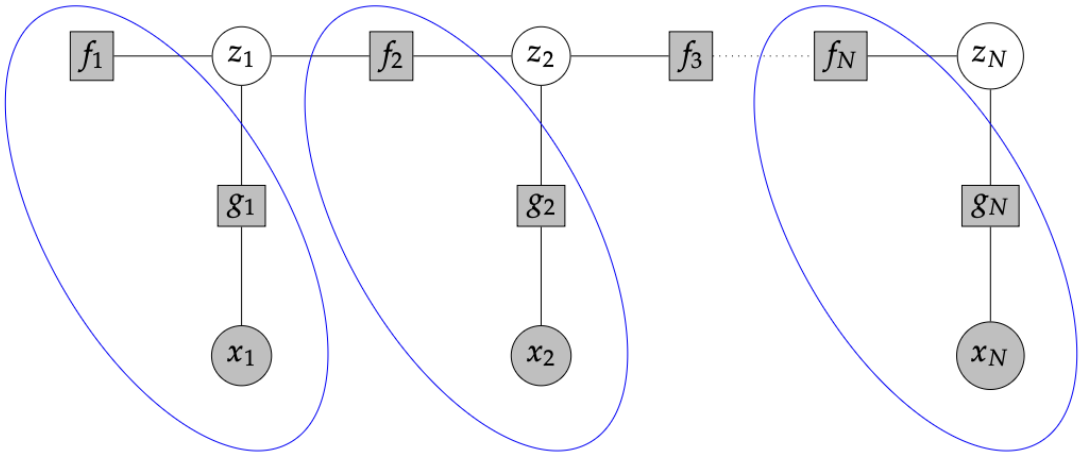
- **Filtering**: given the observations $\underline{x}$, we want to compute tue distribution over hidden states of the last latent variable at the end of the sequence, i.e. compute the probability $p(z_n|\underline{x})$.
- **Smoothing**: we want to compute the distribution of a latent variable somewhere in the middle of the sequence, at a certain time k, i.e. compute the probability of $p(z_k|\underline{x})$ with $k < N$.
- **Most likely explanation**: most likely sequence of latent variables that generated a particular sequence of observations, i.e. compute $z^* = \arg\max_{\underline{z}} p(\underline{z}|\underline{x})$ with $\underline{z} = z_1, \dots, z_n$.

Remark. We also need to take into account the *parameter learning* task, i.e. given an output sequence, find the transition and emission probabilities (usually solved via maximum likelihood).

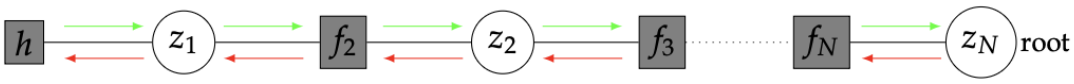
Our goal is now to recast the algorithms for exact inference in PGM (sum-product and max-plus) to the context of HMM. Since both work in Factor Graphs, we need to convert the following Bayesian Network (i.e. the representation we have used so far for HMM) into a Factor Graph:



In this case factor nodes essentially correspond to the edges of the above figure, with the addition of the factor node for the initial probability. Hence we get the following:



We can actually simplify the representation by collapsing into single factor nodes the three nodes inside each ellipsis in the diagram above (the rationale behind this is that observations, by definitions, are fixed). Hence we obtain a chain:



By construction it holds that:

$$h(z_1) = p(z_1)p(x_1|z_1) \text{ with } x_1 \text{ observed, hence clamped}$$
$$f_2(z_1,z_2) = p(z_2|z_1)p(x_2|z_2) \text{ with } x_2 \text{ observed, hence clamped}$$

until

$$f_N(z_{N-1},z_N) = p(z_N|z_{N-1})p(x_N|z_N) \text{ with } x_N \text{ observed, hence clamped}$$

Fixing z_N as the root, the sum-product algorithm reads as:

- Forward messages (green arrows in the figure above):

$$\mu_{z_{n-1}\to f_n}(z_{n-1}) = \mu_{f_{n-1}\to z_{n-1}}(z_{n-1})$$
$$\mu_{f_n\to z_n}(z_n) = \sum_{z_{n-1}} f_n(z_{n-1},z_n)\cdot\mu_{z_{n-1}\to f_n}(z_{n-1})$$

  We can eliminate $\mu_{z_{n-1}\to f_n}(z_{n-1})$ and obtain a recursion for the forward messages:

$$\alpha(z_n) := \mu_{f_n\to z_n}(z_n)$$
$$\alpha(z_n) = \sum_{z_{n-1}} f_n(z_{n-1},z_n)\cdot\alpha(z_{n-1})$$

- Backward messages (red arrows in the figure above):

$$\mu_{f_{n+1}\to z_n}(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n,z_{n+1})\cdot\mu_{f_{n+2}\to z_{n+1}}(z_{n+1})$$

  rewriting with the usual notation:

$$\beta(z_n) := \mu_{f_{n+1}\to z_n}(z_n)\beta(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n,z_{n+1})\cdot\beta(z_{n+1})$$

After computing all the messages, smoothing can be solved combining the forward and backward messages:

$$p(z_n, \underline{x}) = \alpha(z_n)\beta(z_n)$$

while filtering:

$$p(z_N, \underline{x}) = \alpha(z_N)$$

Moreover:

$$\begin{gathered} p(z_n, z_{n+1}, \underline{x}) = \alpha(z_n) f(z_n, z_{n+1}) \beta(z_{n+1}) \\ p(z_n|\underline{x}) = \frac{p(z_n, \underline{X})}{\sum_{z_n} p(z_n, \underline{X})} = \frac{p(z_n, \underline{X})}{p(\underline{X})} = \frac{p(z_n, \underline{X})}{\sum_{z_n} \alpha(z_n)} \end{gathered}$$

In order to find the most likely sequence, we need to plug the max-plus algorithm. This reads as:

$$\begin{aligned} \hat{\mu}_{f_n \to z_n} &= \max_{z_{n-1}} \{\log f_n(z_n, z_{n-1}) + \hat{y}_{f_{n-1} \to z_{n-1}}(z_{n-1})\} \\ \Phi(z_n) &= \arg\max_{z_{n-1}} \{\log f_n(z_n, z_{n-1}) + \hat{y}_{f_{n-1} \to z_{n-1}}(z_{n-1})\} \end{aligned}$$

Remark. In the context of HMM, the max-plus algorithm is known as *Viterbi algorithm*.

6 Bayesian Linear Regression

In this chapter we will introduce the simplest Bayesian machine learning approach, namely Bayesian linear regression. Under a Gaussian likeli- hood model for observations, the solution can be computer analytically, requiring a matrix inversion from a computational perspective. We will start by an introduction to Gaussian distributions, Bayesian inference and linear regression (to fix notation). Then we will dig into Bayesian regression, discuss the role of model evidence or marginal likelihood and briefly touch upon model comparison.

6.1 Gaussian Distributions

We are going to view in detail a number of useful properties of Multi- variate Gaussian Distribution, which will be very useful in the following sections and chapters.

Let's start by defining the probability density of the d-dimensional Multivariate Gaussian, denoted by $N(x|\mu,\Sigma)$, where μ is a d-dimensional vector and represents the mean of the Gaussian and Σ is a $d \times d$ positive definite matrix, called **Covariance matrix**:

$$N(x|\mu,\Sigma) = ((2\pi)^d det(\Sigma))^{-\frac{1}{2}} \cdot \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Sometimes $\Sigma^{-1} = A$, called the **Precision matrix**, is used instead of the covariance matrix in the definition of a Gaussian. We also refer to $(x-\mu)^T\Sigma^{-1}(x-\mu)$ as the **Mahalonobis distance**. Notice that having $\Sigma = I$ one obtains the **euclidean distance**.

6.1.1 Principal Components

Since Σ is positive definite, we can diagonalize it and decompose it in $\Sigma = E\Lambda E^T$ where $\Lambda = diag(\lambda_1,\ldots,\lambda_n)$ is a diagonal matrix composed of the eigenvalues of Σ and E is an orthogonal matrix (i.e. such that $EE^T = I$ holds) whose rows are eigenvectors of Σ.

We can do a change of coordinate in this way:

$$y = \Lambda^{-\frac{1}{2}E^T(x-\mu)}$$

Then we have that

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = (x-\mu)^T E\Lambda^{-\frac{1}{2}}\Lambda^{-\frac{1}{2}}E^T(x-\mu) = y^T y$$

Practically, we obtain a Gaussian distribution with mean zero and Covariance distribution equal to the identity, $N(x|0,1)$. Geometrically we are roto-translating the ellipsoids describing the level sets of a Gaussian Distribution into a sphere centered in the axis, such that points at one standard from the mean have distance 1 from the origin. This linear change of basis can always be performed.

6.1.2 Completing the square

Suppose that we have a probability density like

$$p(x) = c \cdot \exp\left(-\frac{1}{2}x^T Ax - b^T x\right)$$

This is actually a Gaussian distribution; to show it we need to do some algebra on $\log p(x)$. The following identity can be proved:

$$-\frac{1}{2}x^TAx - b^Tx = \frac{1}{2}(x-A^{-1}b)^TA(x-A^{-1}b) - \frac{1}{2}b^TA^{-1}b$$

Therefore we can use the properties of the exponential to get

$$x\cdot\exp(-\frac{1}{2}x^TAx - b^Tx) = N(x|A^{-1}b,A^{-1})\underbrace{\sqrt{(2\pi)^d det(A^{-1})}\cdot\exp(-\frac{1}{2}b^TA^{-1}b)\cdot c}_{=1}$$

which means that

$$p(x) = N(x|A^{-1}b,A^{-1})$$

Stated otherwise: **every distribution which is an exponential of a quadratic form is a Gaussian distribution**.

6.1.3 Further properties

The following properties will not be proved. You can find more details in the Bishop book:

- **Linear transformation**: Suppose to have $y = Mx+\eta$ where $x\sim\mathcal{N}(\mu_x,\Sigma_x),\eta\sim\mathcal{N}(\mu,\Sigma),x\perp\eta$. Then y is Gaussian, $y\sim\mathcal{N}(M\mu_x+\mu,M\Sigma_xM^T+\Sigma)$. This means that Gaussians are closed under linear transformations.
- **Marginals and conditionals**: Assume that

  $$z = \begin{bmatrix}x\\y\end{bmatrix},\quad z\sim\mathcal{N}(\mu,\Sigma)$$

  $$\mu = \begin{bmatrix}\mu_x\\\mu_y\end{bmatrix},\quad \Sigma = \begin{bmatrix}\Sigma_{xx} & \Sigma_{xy}\\\Sigma_{yx} & \Sigma_{yy}\end{bmatrix}$$

  The marginal distribution is pretty easy to obtain:

  $$x\sim\mathcal{N}(\mu_x,\Sigma_{xx})$$

  While the conditional distribution is just a little more complicated:

  $$p(x|y) = \mathcal{N}(x|\mu_x+\Sigma_{xy}\Sigma_{yy}^{-1}(y-\mu_y),\Sigma_{xx}+\Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx})$$

- **Product of Gaussians**: The product of two Gaussians densities is still a Gaussian

  $$\mathcal{N}(x|\mu_1,\Sigma_1)\mathcal{N}(x|\mu_2,\Sigma_2) = \mathcal{N}(x|\mu,\Sigma)\cdot K$$

  where

  $$\begin{gathered}S = \Sigma_1+\Sigma_2\\ \mu = S^{-1}(\Sigma_1\mu_1+\Sigma_2\mu_2)\\ \Sigma = \Sigma_1S^{-1}\Sigma_2\\ K = \frac{\exp(-\frac{1}{2}(\mu_1-\mu_2)^TS^{-1}(\mu_1-\mu_2))}{\sqrt{det(2\pi S)}}\end{gathered}$$

- **Bayesian Theorem**: Supposing to have

  $$x\sim\mathcal{N}(x|\mu,A^{-1}),\quad p(y|x) = \mathcal{N}(y|Mx+b,L^{-1})$$

  Then the joint distribution of x and y is still a Gaussian

  $$z = \begin{bmatrix}x\\y\end{bmatrix}\sim\mathcal{N}(z|\mu_z,\Sigma_z)$$

  In fact

  $$\begin{gathered}\ln p(z) = \ln p(x)+\ln p(y|x)\\ const-\tfrac{1}{2}(x-\mu)^TA(x-\mu)-\tfrac{1}{2}(y-Mx-b)^TL(y-Mx-b)\end{gathered}$$

By completing the square we obtain a Gaussian with the following mean and covariance

$$z \sim \mathcal{N}\left(\begin{bmatrix}\mu \\ M\mu + b\end{bmatrix}, R^{-1}\right)$$

Where

$$R = \begin{bmatrix}A + M^T LM & -M^T L \\ -LM & L\end{bmatrix}$$

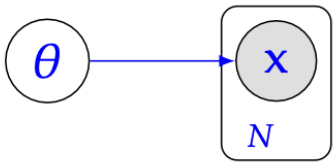
And

$$R^{-1} = \begin{bmatrix}A^{-1} + M^T L^{-1} M & -M^T L^{-1} \\ -LM & L^{-1}\end{bmatrix}$$

And then we can apply the marginalization and the conditional distribution formula that we have seen before to compute $p(y)$ and $p(x|y)$.

6.2 Bayesian Estimation

Consider n observations of i.i.d. random variables $\underline{x} = x_1, \ldots, x_n$ ad a family of models $p(x|\theta)$, corresponding to out likelihood distribution, in which we are looking for the model that best fits our data.



In the Bayesian context, we also have a **priori** distribution $p(\theta)$ and we can compute the posterior distribution

$$p(\theta|\underline{x}) = \frac{p(\underline{x}|\theta)p(\theta)}{p(\underline{x})}$$

The problem in this scenario, as usual, is computing the **marginal likelihood**, because $p(x) = \int p(\underline{x}|\theta)p(\theta)\,d\theta$ is a hard integral to approximate in a high dimensional setting.

Now, we could compute the maximum a-posteriori, i.e. $\theta_{map} = max_\theta p(\theta|\underline{x})$, but in this way, from the estimated parameters, we could obtain only point estimates as output. Instead, we want to get the entire distribution in order to have a complete representation of uncertainty. So, it's more convenient to compute the **predictive distribution** of x by using all the information contained in the posterior

$$p(x|\underline{x}) = \int p(x|\theta)p(\theta|\underline{x})\,d\theta$$

Which is obtained by the usual factorization

$$\int p(x,\theta|\underline{x})\,d\theta = \int p(x|\theta,\underline{x})p(\theta|\underline{x})\,d\theta = \int p(x|\theta)p(\theta|\underline{x})\,d\theta$$

Let's study an example: suppose to have $x_1, \ldots, x_n \in \{0,1\}$ so that $p(\underline{x}|\theta) = Bernoulli(\theta)$, and that we observed $(1)k$ times and $(0)n-k$ times. A good choice for our prior in this scenario is

$$p(\theta) = Beta(\theta|\alpha,\beta) = \frac{1}{B(\alpha,\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

Where B is the **Beta function**. It representa a family of distributions having a domain in $[0,1]$, which can be skewed more toward 0 or 1 by playing with the parameters. It can be showed that

$$\mathbb{E}_{Beta(\alpha,\beta)}[\theta] = \frac{\alpha}{\alpha+\beta}$$

By using the priori and the likelihood we can compute the logarithm of the posterior

$$\begin{aligned}\log p(\theta|\underline{x}) &= L(\theta) + \log B(\theta|\alpha,\beta) + \log p(\underline{x}) \\ &= k\log\theta + (n-k)\log(1-\theta) + (\alpha-1)\log\theta + (\beta-1)\log(1-\theta) + C \\ &= C + (k+\alpha-1)\log\theta + (N-k+\beta-1)\log(1-\theta)\end{aligned}$$

where the term C incorporates all the terms that are independent of θ. By recognising the functional form of a Beta distribution we arrive at the last equality

$$\log p(\theta|\underline{x}) = \log Beta(\theta|k+\alpha, N-k+\beta)$$

The predictive distribution is then defined through the following expectation:

$$p(x=1|\underline{x}) = \int p(x=1|\theta)p(\theta|\underline{x})\,d\theta = \int_0^1 \theta Beta(\theta|k+\alpha, N-k+\beta)\,d\theta = \frac{k+\alpha}{N+\alpha+\beta}$$

The Bernoulli and the Beta distribution are an example of **conjugate priors**.

Definition: *Conjugate Priors*

We say that the prior distribution and the likelihood distribution are **conjugate priors** if the corresponding posterior distribution has the the same functional form of the prior.

6.3 Linear Regression

We will start with an introduction to linear regression. This will serve as a recap of notions that will be expanded in Bayesian setting later on.

We are moving from the problem of describing probabilistic models and performing inference on them, to the problem of **supervised learning**.

We have data, in the form of $\underline{x}, \underline{y} : (x_i, y_i)$ where $i = 1, \ldots, N$, i.e. we have pairs of inputs and outputs. Assume that $p(y|x) = p(y|x,\theta)$ is a parametric model of our random variables. At first, we are going to choose θ_{ML} with a maximum likelihood approach

$$\theta_{ML} = argmaxp(\underline{y}|\underline{x},\theta)$$

Therefore what we need to do is to identify our parametric model, which in linear regression is just

$$p(y|x,\theta) = N(y|f(x,w),\beta^{-1})$$

Notice that in this case $\theta = (w,\beta)$. We can equivalently rewrite this with the (perhaps more familiar) notation

$$y = f(x,w) + \varepsilon \quad \varepsilon \sim N(0,\beta^{-1})$$

In particular, in linear regression, we will have that our function f is linear w.r.t. our weights w, that is

$$f(x,w) = w_0\psi_0(x) + \cdots + w_{M-1}\psi_{M-1}(x)$$

Notice that ψ can be, and usually are, non-linear functions of the input data. They are the **basis function** for our regression model. They can be monomials, Gaussian RBF, sigmoids (NN), ...

This means that the log likelihood of our model is, in fact

$$\log p(y|x,\theta) \propto -E_D(w) = -\frac{1}{2}\sum_{i=1}^{N}\left(y_i - w^T\phi(x_i)\right)^2$$

Minimizing the sum of squares E_D means maximizing the likelihood, hence, taking the gradient of the function we get

$$\nabla_w E_D(w) = \sum_{i=1}^{N}\left(y_i + w^T\phi(x_i)\right)\phi^T(x_i) = 0$$

which yields the following close form for our weights

$$w_m = \left(\Phi^T \Phi\right)^{-1} \Phi^T \underline{y}$$

Definition: *Defign Matrix*

$\Phi_{ij} = \phi_j(x_i)$ is called the **design matrix**, it is the j-th feature (basis function) evaluated on the i-th datum.

If M is large, solving the direct problem might be computationally difficult, but we can rely on optimization algorithms, such as gradient descent, to actually find the minimum of our negative log-likelihood, exploiting the fact that we are dealing with a quadratic form here.

In order to avoid overfitting, especially if the chosen basis functions are enough complex and expressive, we seldom introduce further regularization terms in the loss function, such as

$$f(x) = \begin{cases} \frac{1}{2}\|w\|_2^2, & Ridge \\ \frac{1}{2}\|w\|_1, & Lasso \end{cases}$$

Therefore we will minimize the quadratic loss plus one of the two penalty terms above:

$$E_D(w) + \lambda E_W(w)$$

where λ is the regularization coefficient.

Example:

As an example, we can generate synthetic datasets by adding Gaussian noise to a set of points belonging to the curve $y = \sin(2\pi x)$

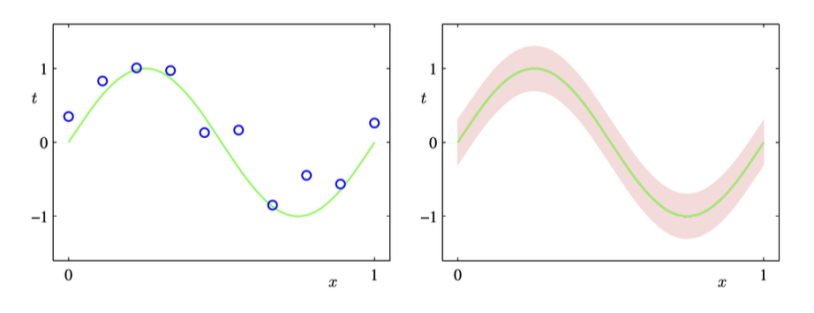


Figure 6.1: On the left, the sinusoidal function and the generated data points. On the right, the true conditional distri- bution $p(t|x)$ in which the green curve denotes the mean and the shaded region spans one standard deviation on each side of the mean (Bishop)

We build 100 data sets, each having 25 data points, and we perform Linear Regression using 24 Gaussian basis functions on the datasets varying the regularization coefficient λ.

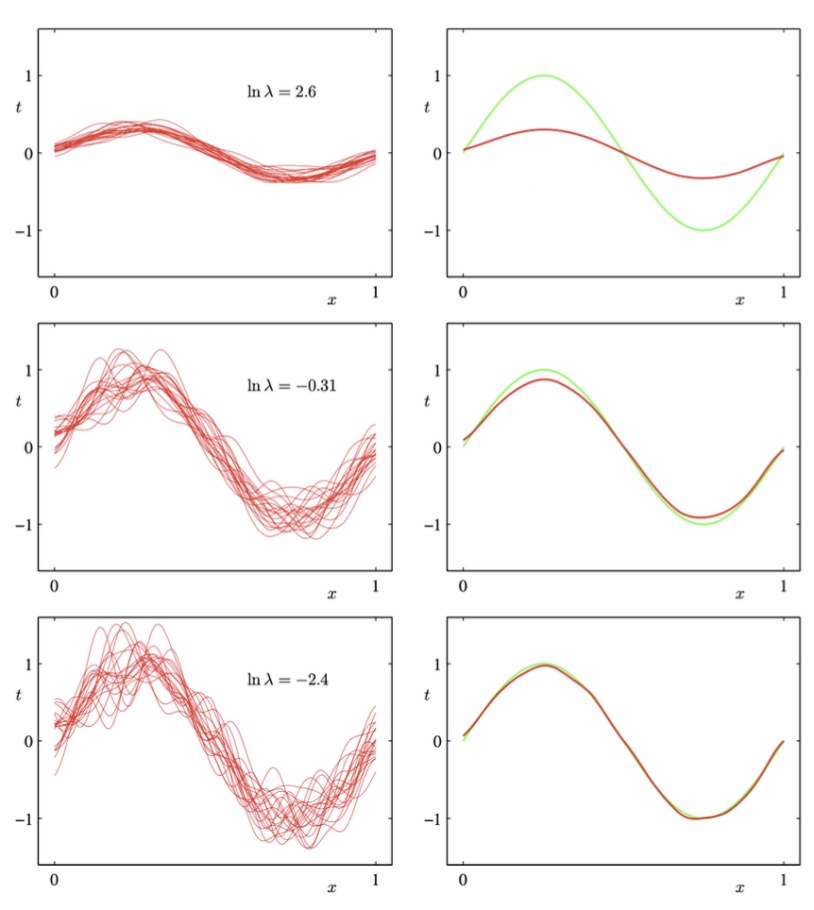


Figure 6.2: On the left, the result of fit- ting the model to the di□erent data sets varying $\ln(\lambda)$. On the right, the average of the fits on the □□□ datasets (Bishop)

6.4 Bayesian Linear Regression

By adding the regularization term, we are modifying the loss function in order to obtain better results in terms of overfitting, but one of the drawbacks is that we lose a nice probabilistic interpretation of our model: the object we are minimizing is not a negative likelihood anymore. How can we go back towards a more rigorous probabilistic setting?

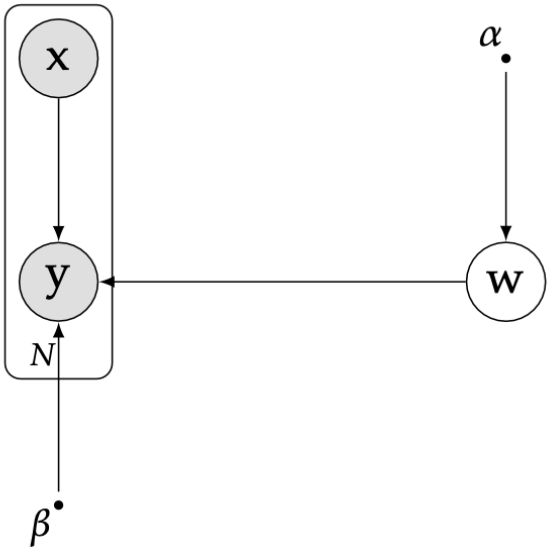
The key point to observe in order to do this step is that the regularization term is, in fact, a **bias** that we introduce in our data. What we can do instead of adding a penalty term in the loss, is to encode the bias in a **prior** distribution of our parameters. $p(w|\alpha)$ and then treat our problem in a Bayesian way. As an example:

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

where α is a **hyper-parameter** of our distribution (we will see some methods to choose it). This is a typical choice for our bias since then our weights will be forced to be small (which is the goal of the regularization).

For the moment let's suppose we have fixed our α. Since we have a likelihood of our observation, we can compute the posterior. Let's also introduce a Gaussian noise in the observations with precision β.

We can visualize the model in graphical terms as:



Applying Bayes theorem, the posterior is

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

In this scenario, choosing a Gaussian prior and having the Gaussian likelihoodof the linear regression problem, we have an analytical form for our posterios distribution, as we are about to see.

Let's consider the logarithm of the posterior first

$$\log p(w|\underline{\mathrm{x}},\underline{\mathrm{y}},\alpha,\beta) = -\frac{\beta}{2}\sum_{j=1}^{N}\left(y_i - w^T\phi(x_i)\right)^2 - \alpha w^T w + const$$

The logarithm of the marginal likelihood does not depend on w and is treated as a constant. The trick is to notice that we have a quadratic function of w, and, as we have seen in the first paragraph, if the logarithm of a distribution is a quadratic function then that distribution is a Gaussian.

$$p(w|\underline{\mathrm{x}},\underline{\mathrm{y}},\alpha,\beta) = N(w|m_N,S_N)$$

where

$$m_N = \beta S_N \Phi^T \underline{\mathrm{y}}$$
$$S_N^{-1} = \alpha I + \beta\Phi^T\Phi$$

which is very similar (but not equal) to what we get as a solution in Ridge Regression.

If we use a general Gaussian prior instead, of the form $p(w|m_0,S_0) = N(w|m_0,S_0)$, then our posterior becomes

$$p(w|\underline{\mathrm{x}},\underline{\mathrm{y}},\alpha,\beta) = N(w|m_n,S_N)$$

with

$$m_N = S_N\left[S_0^{-1}m_0 + \beta\Phi^T\underline{\mathrm{y}}\right]$$
$$S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$$

So, in Bayesian regression, we treat our parameters probabilistically, placing a prior distribution over them, computing the posterior given the observation that we have and we using it to make predictions. We have seen that for a Gaussian prior we have a Gaussian posterior, and we also have an analytically computable posterior, given that we know how to invert matrices.

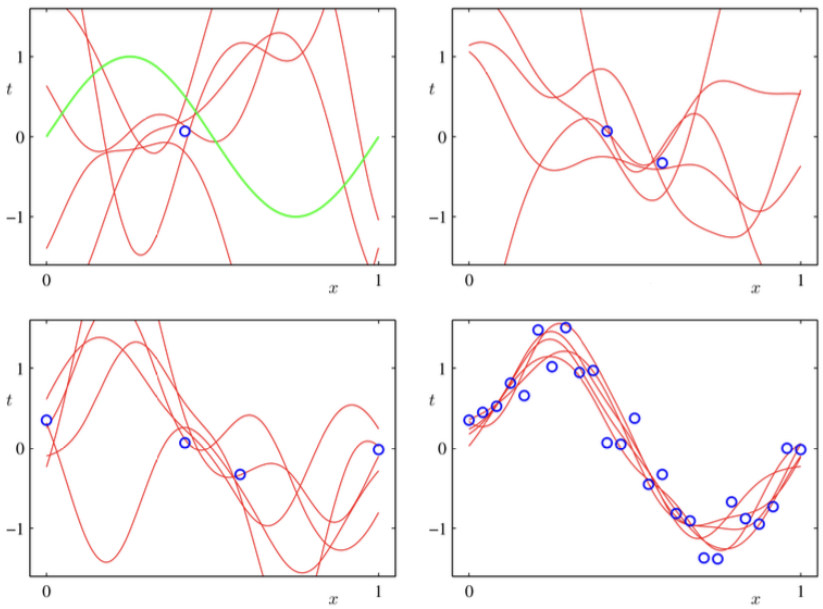


Figure 6.3: Samples from the posterior distribution of a Bayesian regression model on the dataset shown in section 6.3. Increasing the size of the dataset, the predictive distribution approximates better the true data distribution (Bishop)

6.4.1 Online Learning

There is an extra feature which is typical of Bayesian learning. When we first pick a prior distribution we are basically having random weights, corresponding to random lines in the data space (remember what the weights actually represent!). Once we compute the posterior, we are reshaping the Gaussian distribution from which we sample our weights, up until it becomes highly centered towards a point once we get a lot of observations. In this process, nothing forbids us to start from a prior that already takes into account some observations! This is a general principle of Bayesian learning: when we observe new data points, we use as new prior the posterior relative to the previous observations. Hence, Bayesian linear regression, is, *naturally*, an **online method**, which means that every time we observe new data we can easily incorporate them into our model without retraining the model from the start!

Example: ***Online learning***

In order to visualize how the posterior distribution is updated when including new training data, we consider the simple example reported in the Bishop's book. We want to fit a linear model of the form $f(x, \mathbf{w}) = w_0 + w_1 \cdot x$, assuming α and β known. The columns of figure 6.4 show:

- First column: the likelihood of the last data point $(\underline{x}, \underline{y})$ added to the training set as a function of $\mathbf{W}$, i.e. $p(\underline{y}|\underline{x}, \mathbf{w})$
- Second column: the posterior distribution obtained multiplying the prior (which is the posterior of the previous row) by the likelihood reported in the same row
- Third column: some samples of the regression function obtained by drawing samples of $\mathbf{w}$ from the posterior distribution

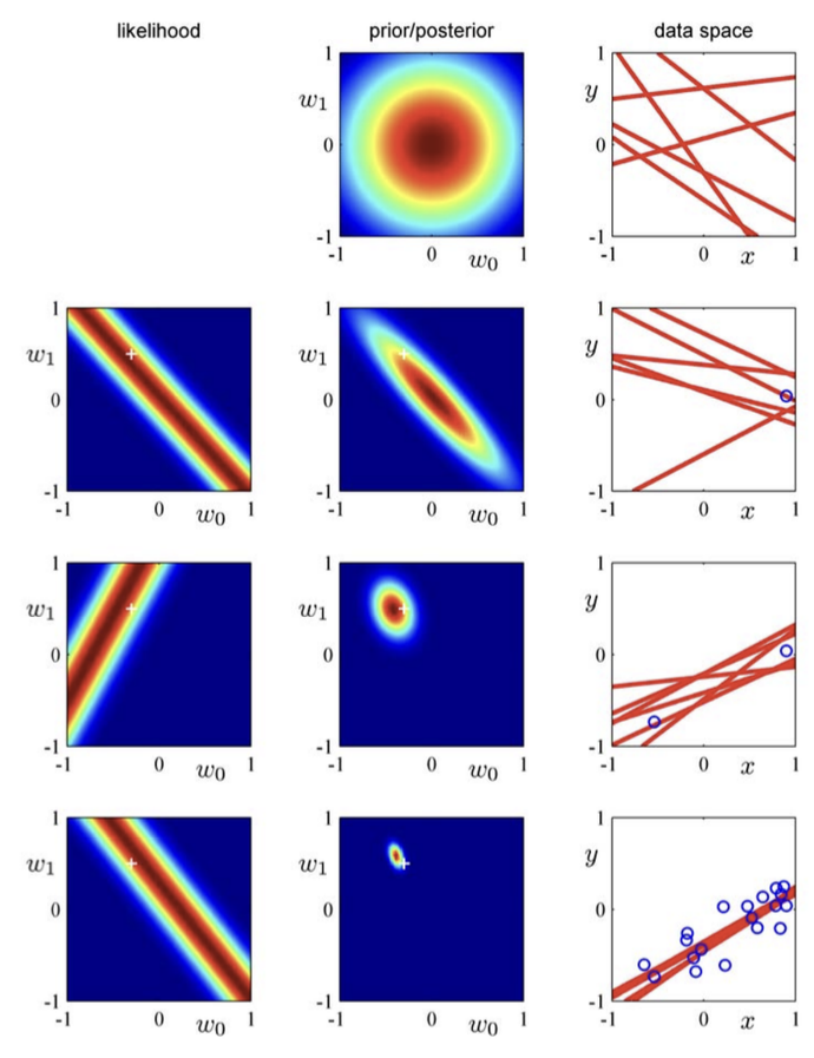


Figure 6.4: Illustration of sequentila Bayesian learning for a sample linear model of the form $f(x, \mathbf{w}) = w_0 + w_1 \cdot x$ (Bishop).

The first row corresponds to the situation before any data points are observed: the prior distribution of w_0, w_1 is a multivariate standard normal distribution. In the next rows this distribution is reshaped by the information contained in the dataset and the posterior distribution becomes sharper and centered on the true parameter values.

6.5 Predictive distribution

Remember that the probability distribution is just the prediction of a new point given our observations.

$$p(y|x, \underline{x}, \underline{y}, \alpha, \beta)$$

In Bayesian learning we average over all possible models

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

Since in the linear regression setting we have that both these distributions are Gaussians, we know that the product of Gaussians densities is a Gaussian, and also the marginal of a Gaussian is a Gaussian. Therefore it

can be shown that the probabilitiy above is

$$p(y|x,\underline{x},\underline{y},\alpha,\beta) = \mathcal{N}(y|m_N^T\phi(x),\sigma_N^2(x))$$
$$\sigma_N^2(x) = \frac{1}{\beta} + \phi^T(x)S_N\phi(x)$$
$$\sigma_N^2(x) \geq \sigma_{N+1}^2(x), \quad \sigma_N^2 \to \frac{1}{\beta} \to \infty$$

The prediction is a Gaussian centered on an average prediction and having a variance which has two terms: one takes into account the noise observations, while the second one takes into account the epistemic uncertainty of our model. As we increase our knowledge, the epistemic uncertainty goes to zero and we are left with just the aleatoric uncertainty.

As such, we can see, graphically, that the credibility interval of our model shrinks the more we add observations.

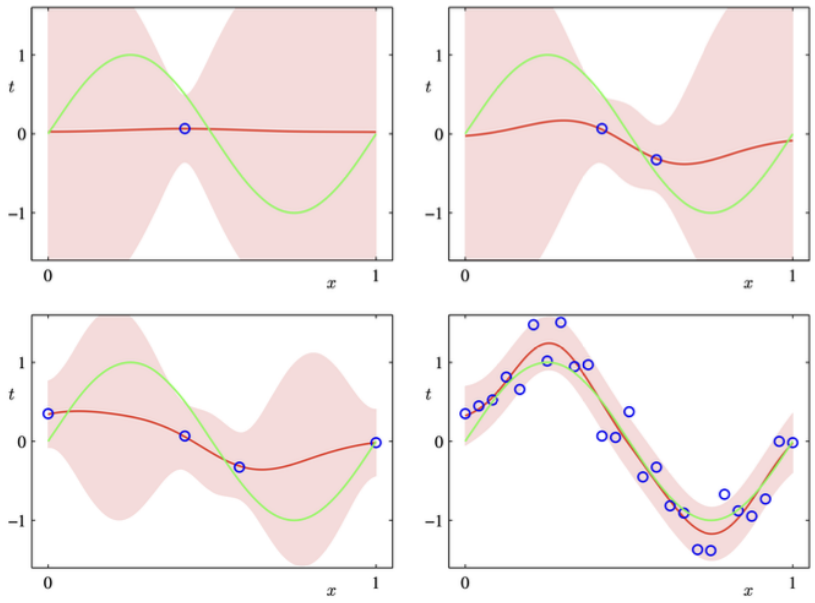
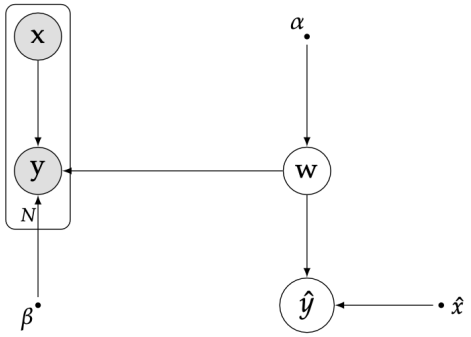


Figure 6.5: Example of the predictive distribution for a Bayesian linear regres- sion model on the dataset shown in section 6.3. Increasing the number of data points, the red curve (which rep- resents the mean of the predictive dis- tribution) approximates better the sinu- soidal function and the standard devia- tion (the shaded region) decreases. Note that the predictive uncertainty is smallest in the neighbourhood of the data points (Bishop)

We can represent the Bayesian linear regression model including the predictive with the following probabilistic graphical model:



Potentially, we could treat this problem also as an inference in a PGM (even though this does not make much

sense in practice since we have an analytical solution for the problem).

6.6 Model evidence

How can we deal with the hyper-parameters α and β? Remember that α^{-1} represents the variance of the prior distribution whereas β^{-1} is the noise of the observations. The tool to estimate them is to use the marginal likelihood

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

as the posterior is

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

If we would treat the hyper-parameters in a Bayesian way then we would need to place a prior over them, and this would mean having a hyperprior $p(\alpha,\beta)$ and then computing the posterior distribution $p(\alpha,\beta|\underline{y},\underline{x}) \propto p(\underline{y}|\underline{x},\alpha,\beta)p(\alpha,\beta)$.

This is doable, but then we would need to introduce other hyper-hyper parameters and this would lead us to a hierarchy of hyper-parameters.

An alternative instead is to make an approximation at this level. We need two approximations in fact:

1. First of all, we ask for an **uninformative prior** $p(\alpha,\beta)$, it can be a uniform distribution over an interval, or a Gaussian with a very broad variance. Let's suppose then that $p(\alpha,\beta) = const$
2. The posterior should be sharply peaked around the Maximum a Posteriori (MAP) of α and β. Hence $p(\alpha,\beta|\underline{y},\underline{x}) \approx \delta_{MAP(\alpha,\beta)}$

In fact, these two approximation means that we can fix α and β with Maximum Likelihood, which means that we can find our best hyperparameters by maximizing the Marginal Likelihood.

How to compute the marginal likelihood? Again, we rely on the closure properties of the Gaussian distribution. Since we have computed the posterior, and we already know the likelihood and the prior, we can just take the logarithm of the left and right term of Bayes' Theorem and solve the equation, else we can compute it directly (it is an integral of a Gaussian).

Therefore it can be proved that the marginal likelihood has the form

$$\log p(\underline{y}|\underline{x},\alpha,\beta) = \frac{M}{2}\log\alpha + \frac{N}{2}\log\beta - \mathbb{E}(m_N) - \frac{1}{2}\log|S_N^{-1}| - \frac{N}{2}\log 2\pi$$
$$\mathbb{E}(m_N) = \frac{\beta}{2}\|y - \Phi m_N\|^2 + \frac{\alpha}{2}m_N^T m_n$$
$$m_N = \beta S_N \Phi \cdot \underline{y}$$
$$S_N^{-1} = \alpha I + \beta\Phi^T\Phi$$

where N is the size of the dataset and M is the number of parameters. In order to maximize this we can of course compute the gradient with respect to α and β and do gradient ascent on the expression, for example.

6.6.1 Fixed-point algorithm

Here we will consider an alternative approach via a fixed-point algorithm. The general idea is to take $\nabla \log p(\underline{y}|\underline{x},\alpha,\beta) = 0$ and to derive fix point equations

$$\alpha = g_\alpha(\alpha,\beta)$$
$$\beta 0 = g_\beta(\alpha,\beta)$$

The algorithm works like this:

1. Fix α_0 and β_0

2. Compute

$$\alpha_{n+1} = g_\alpha(\alpha_n, \beta_n)$$
$$\beta_{n+1} = g_\beta(\alpha_n, \beta_n)$$

3. Iterate step 2 until convergence, i.e. up until

$$\|\alpha_{n+1} - \alpha_n\| + \|\beta_{n+1} - \beta_n\| < \varepsilon$$

Therefore we just need to compute

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

Let's start from the term

$$\log|S_N^{-1}|$$
$$S_N^{-1} = \alpha I + \beta\Phi^T\Phi$$

In order to compute this determinant we first need to compute the eigenvalues λ_i of $\beta\Phi^T\Phi$. Then

$$|S_N^{-1}| = \prod_{i=0}^{m-1}(\alpha + \lambda_i)$$

Notice that λ_i does not depend on α. Which means that

$$\frac{d}{d\alpha}\log|S_N^{-1}| = \frac{d}{d\alpha}\sum\log(\alpha + \lambda_i) = \sum_{i=1}^{m-1}\frac{1}{\alpha + \lambda_i}$$

Moreover

$$\frac{d}{d\beta}\lambda_i = \frac{\lambda_i}{\alpha + \lambda_i}$$

In the end, bu also deriving all the other terms we get

$$\alpha = \frac{\gamma}{m_N^T m_N} = g_\alpha(\alpha, \beta), \quad \gamma = \sum_{i=0}^{m-1}\frac{\lambda_i}{\alpha + \lambda_i}$$
$$\frac{1}{\beta} = \frac{1}{N-\gamma}\sum_{n=1}^{N}\left(y_n - m_N^T\Phi(x_n)\right)^2 = \frac{1}{g_\beta(\alpha,\beta)}$$

6.6.2 Effective number of parameters

Let's focus on the parameter

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

where λ_i are the eigenvalues of $\beta\Phi^T\Phi$ and they give us information about the maximum likelihood solution for w. In fact, they give us the **curvature** of the likelihood function (they represent the Hessian of the likelihood). Small λ_i means a large curvature of the likelihood function which implies a large uncertainty on w_i and vice versa. When we have a large uncertainty on w_i, it means that taking the Maximum Likelihood solution of that particular weight is not very sensible. That's because introducing a prior on the parameter would likely change this value a lot (the Bayesian estimation would be different than the maximum likelihood estimation). The effective number of parameters gives us the effective number of parameters which Maximum Likelihood estimation is close to their Maximum a Posteriori estimation.

In fact, we have that

$$\lambda_i << \alpha \implies \frac{\lambda_i}{\alpha+\lambda_i} \approx 0$$
$$\lambda_i >> \alpha \implies \frac{\lambda_i}{\alpha+\lambda_i} \approx 1$$

and by definition of γ we have the meaning that we have been introducing before.

Notice also that in the regime of large data $N >> M$, then $\gamma \approx M$. Here the equation for α and β are also simplified.

In this scenario, the theorem of Bernstein-von Mises implies that the prior has no asymptotic influence on the posterior and that posterior inference is consistent with the frequentist approach (i.e. Maximum Likelihood estimation). Of course, there are some assumptions for this theorem to hold: the key assumption is that the "true" value of the parameter is interior to the parameters space.

Thus, the effective number of parameters in Bayesian estimation is adaptive: parameters will be "included" (in the sense that their uncertainty is low) in the model only if there is enough evidence in the data to justify their use. In a certain sense, a Bayesian model can say "I don't know" when needed. This has the effect of avoiding overfitting and giving a more correct estimation of the error when doing predictions.

6.7 Model Comparison

Imagine that we are doing linear regression and we pick two different sets of basis functions. Which of the two models should we choose?

To answer this question, we can leverage the marginal likelihood.

Suppose that we have two models $\mathbb{M}_1$ and $\mathbb{M}_2$ which are different (in the linear regression context, this means having two different sets of basis functions). Which one is the best to explain the data $D = \underline{x}, \underline{y}$?

Since we want to be Bayesian, let's place a prior distribution on the models, $p(\mathbb{M}_j)$. The posterior distribution, by Bayes' theorem, is

$$p(\mathbb{M}_j|D) = \frac{p(D|\mathbb{M}_j)p(\mathbb{M}_j)}{\Sigma_j p(D|\mathbb{M}_j)p(\mathbb{M}_j)}$$

Notice that $p(D|\mathbb{M}_j) = \int p_{\mathbb{M}_j}(D, \theta_{\mathbb{M}_j}|\mathbb{M}_j)\, d\theta_{\mathbb{M}_j}$ is the **marginal likelihood** with respect to the parameters of $\mathbb{M}_j$. In fact, since we are not looking at a specific configuration of the parameters of the model $\mathbb{M}_j$, we have to marginalize them, obtaining the marginal likelihood. We also assume that hyper-parameters are fixed in this scenario.

How to perform model selection? We have two choices

1. Model averaging (a fully Bayesian approach): instead of choosing one model we consider both of them, weighted according to the posterior distribution. The predictive distribution then is
$$p(y|x, D) = \sum_j p(y|x, D, \mathbb{M}_j) \cdot p(\mathbb{M}_j|D)$$

2. Choose the best model by computing
$$\frac{p(D|\mathbb{M}_1)}{D|\mathbb{M}_2}$$
which is known as the **Bayes Factor** (of $\mathbb{M}_1$ versus $\mathbb{M}_2$). It is basically a ration of the evidences of the two models. The model $\mathbb{M}_j$ to choose is the one with the largest Bayes factor. Given that
$$\int p(D|\mathbb{M}_1) \log \frac{p(D|\mathbb{M}_1)}{p(D|\mathbb{M}_2)}\, dD > 0$$
since this is a Kullback-Leibler divergence, we observe that if $\mathbb{M}_1$ is the true model (i.e. $D \sim p(D|\mathbb{M}_1)$), the expectation of the logarithm of the Bayes Factor $\log \frac{p(D|\mathbb{M}_1)}{p(D|\mathbb{M}_2)}$ will be positive. Hence, on average, the correct model will have the largest Bayes factor.

Example:

Let $\mathbb{M}_1$ and $\mathbb{M}_2$ be two models s.t. $\mathbb{M}_1$ is nested in $\mathbb{M}_2$, i.e., the set of parameters of $\mathbb{M}_1$ is a subset of the parameters of $\mathbb{M}_2$ (for example, linear models where the set of basis functions of $\mathbb{M}_1$ is contained in the one of $\mathbb{M}_2$). This implies $\mathbb{M}_2$ is a more complex model than $\mathbb{M}_1$, and that the distributino $p(D|\mathbb{M}_2)$ is more spread than $p(D|\mathbb{M}_1)$ since the model can explain more data instances. Nevertheless, if $\mathbb{M}_1$ generated the data, then the Bayes factor will be in favor of $\mathbb{M}_1$, since $p(D|\mathbb{M}_1)$ is more concentrated on the few data instances that it can explain. Hence we can see the Occam's Razor principle emerging from the use of the Bayes factor, since the simpler model will be favored in absence of enough evidence to accept the more complex one.

7 Bayesian Linear Classification

The goal in (Bayesian) Linear Classification is (as the name suggests) to learn linear models for classification, meaning models in which the decision boundaries of the input space are linear functions of the input points.

This scenario is somehow more complicated than Bayesian Linear Regression, because, due to a different model for the noise in the observations, the posterior distribution is not analytically tractable. Hence the challenge is to find a good approximation of the posterior of interest.

Consider a dataset $D = (x_n, y_n)$, with $n = 1, \dots, N$, where y_n are categorical. There are various ways of representing class labels y_n, depending on the problem at hand:

- 2-class problems: $y_n \in \{0, 1\}$.
- K-class problem (K > 2): $y_n = (y_{nj})_{j=1,\dots,K}$ such that $y_{nj} \in \{0,1\}, \Sigma_j y_{nj} = 1$. This is called **one-hot-encoding** convention, essentially y_n is represented as a boolean vector of K numbers, with the constraint that only one entry is 1 and all the others are 0.

We have three possible approaches to classification:

- **Discriminant function** $f(x) \in \{1, \dots, K\}$: the goal is to learn a function that maps each input to a specific class (e.g. random forest classification is based on this approach).
- **Discriminative approach** $p(C_k|x) = f(h(x))$: the goal is to model explicitlythe class posterior (e.g. logistic regression, where $p(C_k|x) = f(w^T \phi(x))$). In this context f is called *activation function*, f^{-1} is called *link function*.
- **Generative approach** $p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$: the goal is to model the class conditional probability $p(x|C_k)$ from data, then, considering a prior over classes $p(C_k)$, plug the Bayes' theorem to compute class posterior $p(C_k|x)$.

7.1 Logistic Regression

As already mentioned, **Logistic Regression** is a discriminative model.

Given data $(x_n, y_n), n = 1, \dots, N$, we want to learn $p(C_K|x) = f(w^T \phi(x))$, where $\phi(x) = (\phi_0(x), \dots, \phi_{M-1}(x))$ are the *basis functions*.

The activation function is usually chosen to be either the **Logit** (Logistic sigmoid) function

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

or the **Probit** function (i.e. the cumulative distribution function of the standard Gaussian distribution):

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

Consider the binary classification scenario, i.e. $y_n \in \{0, 1\}$, with logit activation function. Call $s_n = \sigma(w^T \phi(x_n)) = p(C_1|x_n)$ (i.e. probability of assigning input x to class 1). For a fixed w, we use a Bernoulli model of noise:

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

hence the likelihood is

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

Once we have the likelihood, we can compute the cross-entropy error function as

$$\mathbb{E}(w) = -\frac{1}{N}\log p(\underline{y}|\underline{x}) = -\frac{1}{N} y_n \log s_n + (1-y_n)\log(1-s_n)$$

Remark: in this case the dependency on the weights w is highly non-linear, indeed it is through $\log s_n$, being s_n the sigmoid function.

The gradient of the cross-entropy reads as:

$$\nabla\mathbb{E}(w) = \frac{1}{N}\sum_{n=1}^{N}(s_n - y_n)\phi(x_n)$$

The equation $\nabla\mathbb{E}(w) = 0$ has no analytical solution, hence we need to resort to a numerical optimization method in order to find the maximum likelihood solution $w_{ML} = arg\min_w \mathbb{E}(w)$ (note that $\mathbb{E}(w)$ is a convex function).

One possibility is to use stochastic gradient descent for online training using the updated rule for w:

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

where η_n is called *learning rate*.

Remark: if the data are linearly separable in the feature space, then any separating hyperplane $w_{ML}^T\phi(x) = 0$ is a solution, hence we have ∞-many solutions and the optimization problem is ill-defined. To overcome this issue, we typically introduce a penalty term in the function that should be optimized (forcing the weights to be as small as possible), such that the problem remains convex, e.g. we might minimize $\mathbb{E}(w) + \alpha w^T w$.

The same ideas described so far can be recasted to the case of multi-class classification. In this scenario data are (x_n, y_n) with $y_n = (y_{n1}, \ldots, y_{nK})$, i.e. one-hot-encoding over K classes, and class-conditional distributions are:

$$p(C_k|x) = \sigma_j(W^T\phi(x))$$

where W^T is a $K \times M$ matrix, and

$$\sigma_j(\underline{a}) = \frac{\exp(a_k)}{\Sigma_j \exp(a_j)}$$

is called **softmax** function (intuitively it turns a vector of real numbers into a vector of probabilities).

Explicitly, this corresponds to:

$$\begin{cases} a_1 = w_1^T\phi(x) \\ \cdots \\ a_k = w_k^T\phi(x) \end{cases}$$

and

$$p(C_k|x) = \sigma_k(W^T\phi(x)) = \sigma_k(a_1, \ldots, a_k)$$

In this case the cross-entropy is:

$$\mathbb{E}(w) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{K} y_{nj}\log s_{nj}$$

with $s_{nj} = \sigma_j(W^T\phi(x_n))$.

Hence the gradient for class j weights is:

$$\nabla_{w_j}\mathbb{E}(w) = \frac{1}{N}\sum_{n=1}^{N}(s_{nj} - y_{nj})\phi(x_n)$$

7.2 Laplace Approximation

The idea behind **Laplace Approximation** is to approximate an (unknown) distribution with a Gaussian. Notice that it is a local approximation and does not capture the properties of the global distribution. Intuitively this technique consists in centering a Gaussian in a mode of the distribution and use information from the Hessian (i.e. we match the curvature) in order to identify the variance of the Gaussian.

This approximation is often used when we want to approximate some posterior distribution, which is known up to a normalization constant.

In the 1-dimensional case, the form of the distribution that we want to approximate is $p(x) = \frac{1}{Z}f(z)$, with $Z = \int f(z)\,dz$. The idea is to:

- find a mode z_0 of $f(z)$, i.e. a point such that $\frac{d}{dz}f(z_0) = 0$ and z_0 is a point of maximum;
- match the curvature of f at z_0 with a normal distribution.

We can rely on the fact that the logarithm of a Gaussian density is a quadratic function and Taylor expand $\log f(z)$ around z_0:

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

with $A = -\frac{d^2}{dz^2}\log f(z_0), A > 0$ (since z_0 is a mode).

Hence taking the exponential:

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

Since we seek to approximate $p(z)$ with a Gaussian $q(z)$, this is given by:

$$q(z) \sim \mathcal{N}(z|z_0, A^{-1})$$

i.e. A takes the role of the precision of the approximating Gaussian distribution. More explicitly:

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

Since $p(z) = \frac{1}{Z}f(z) \approx \frac{1}{Z}f(z_0)\exp(-frac12A(z - z_0)^2)$, we also have an approximation of the marginal likelihood:

$$Z \approx f(z_0)\left(\frac{A}{2\pi}\right)^{\frac{1}{2}}$$

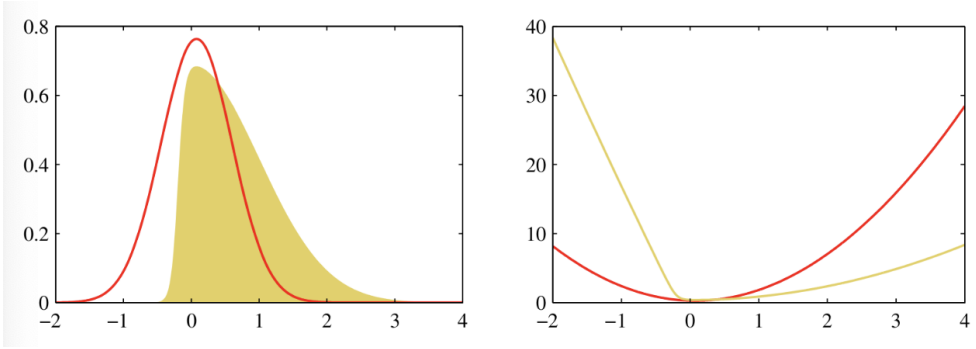
H 

Figure 7.1: Illustration of the Laplace approximation applied to the distribution $p(z) \propto \exp(\frac{-z^2}{2})\sigma(20z+4)$ where $\sigma(z)$ is the logistic sigmoid function. The left plot shows the normalized distribution $p(z)$ in yellow, together with the Laplace approximation centred on the mode z_0 of $p(z)$ in red. The right plot shows the negative logarithms of the corresponding curves.

In the n-dimensional case we proceed in the same way: given a density $p(z) = \frac{1}{Z}f(z)$, we find a mode z_0 (s.t. $\nabla \log f(z_0) = 0$) and approximate $\log f(z)$ around z_0 by Taylor expansion:

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

with $A = -\nabla\nabla \log f(z_0)$.

This gives a Gaussian approximation around z_0 by:

$$q(z) = \mathcal{N}(z|z_0, A^{-1})$$

Furthermore the normalization constant can be approximated as

$$Z = \frac{(2\pi)^{\frac{1}{2}}}{|A|^{\frac{1}{2}}} f(z_0)$$

and for the multivariate case:

$$Z = \frac{(2\pi)^{\frac{k}{2}}}{|A|^{\frac{1}{2}}} f(z_0)$$

Remark: if the distribution p is very skewed, the Laplace approximation is not very effective; if the distribution p is multimodal, we should take the dominant mode (if present) as mean of the approximating Gaussian.

7.2.1 Laplace approximation for model comparison

It is possible to use Laplace approximation for the marginal likelihood in t a model comparison framework.

Consider data D and a parametric model M depending on parameters θ. We fix a prior $p(\theta)$ over θ, and we plug the Bayes' theorem to get $p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$.

Typically the marginal likelihood $p(D) = \int p(D|\theta)p(\theta)\,d\theta$ is hard to compute. This fits the previous framework if we set $f(\theta) = p(D|\theta)p(\theta)$ and $Z = p(D)$.

By Laplace approximation around the maximum-a-posteriori (MAP) estimate θ_{MAP} we get:

$$p(D) \approx \frac{(2\pi)^{\frac{M}{2}}}{|A|^{\frac{1}{2}}} f(\theta_{MAP}) = \frac{(2\pi)^{\frac{M}{2}}}{|A|^{\frac{1}{2}}} p(D|\theta_{MAP})p(\theta_{MAP})$$

So,

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

with $M = |\theta|$ (number of parameters) and $A = -\nabla\nabla\left[\log p(D|\theta_{MAP}) + \log p(\theta_{MAP})\right]$.

Remark: the marginal likelihood is a trade off between model complexity and fit to the data (indeed the last three terms in the previous sum penalize the log-likelihood in terms of complexity).

The **Bayesian Information Content (BIC)** index is defined as

$$\log p(D) \approx \log p(D|\theta_{MAP}) - \frac{1}{2}M\log N$$

and it is a further approximation of the marginal likelihood. It can be used to penalize log-likelihood w.r.t. model complexity, to compare different models.

7.3 Bayesian Logistic Regression

Given observations (x_n, y_n) with $n = \ldots, N$, consider a set of basis functions $\phi(x), \ldots, \phi_{M-1}(x)$ and the logit activation function $\sigma(w^T\phi(x))$.

To recast logistic regression in a Bayesina framework, we place a Gaussian prior over w, $p(w) = \mathcal{N}(w|m_0, S_0)$, with m_0, S_0 fixed or computed via marginal likelihood optimization. The posterior is then:

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

Recall that, in the 2-class problem with a Bernoulli model of noise, the likelihood reads as $p(\underline{y}|w, \underline{x}) = \prod_{i=1}^{N} s_i^{y_i}$, being $s_i = s_i(w) = \sigma(w^T\phi(x_i))$.

Hence the log-posterior is now:

$$\begin{aligned}\log p(w|\underline{y}, \underline{x}) &= \log p(w) + \log p(\underline{y}|w) + C \\ &= -\tfrac{1}{2}(w - m_0)^T S_0^{-1}(w - m_0) + \textstyle\sum_{i=1}^{N}\left[y_i \log s_i(w) + (1 - y_i)\log(1 - s_i(w))\right] + C\end{aligned}$$

Remark: as already mentioned, $s_i(w)$ is not quadratic on w, it is actually an exponential dependency on w (hence not analytically tractable).

We can perform Laplace approximation of the posterior, the steps are the following:

1. find $w_{MAP} = arg\max_w \log p(w|\underline{y}) = arg\max_w \log p(w) + \log p(\underline{y}|w)$ by running a numerical optimization (we can ignore the constant which does not change the location of the maximum).
2. Compute the Hessian at w_{MAP} and invert it: this will give the precision matrix of the Gaussian

$$S_N^{-1} = S_0^{-1} + \sum_{n=1}^{N} s_n(w_{MAP})(1 - s_n(w_{MAP}))\phi(x_n)\phi^T(x_n)$$

Hence, the Laplace approximation of the posterior is $p(w|\underline{y}) \approx q(w)$ with

$$q(w) = \mathcal{N}(w|w_{MAP}, S_N)$$

Given this posterior, we need to marginalize it to compute the **predictive distribution** (which will allow us to do model averaging).

In the binary classification scenario, the predictive distribution for class C_1 is given by (plugging the previous approximation):

$$p(C_1|x^*,\underline{y},\underline{x}) = \int p(C_1|x^*,w,\underline{x},\underline{y})p(w|\underline{y},\underline{x})\,dw \approx \int \sigma(w^T\phi(x^*))q(w)\,dw$$

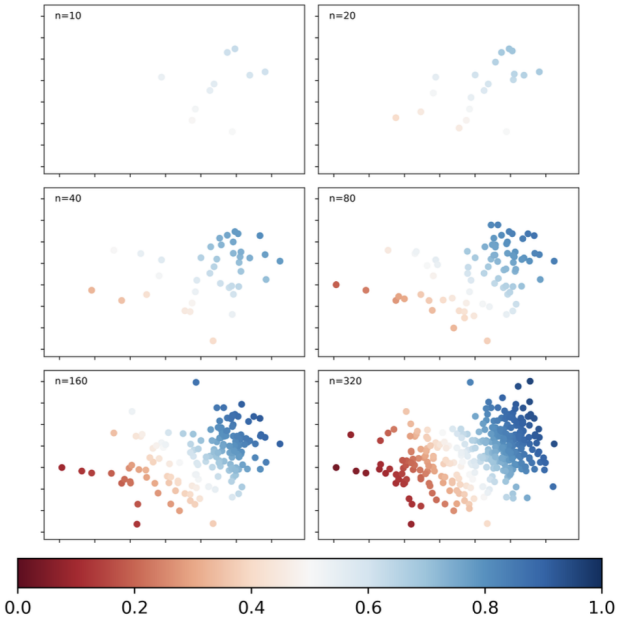


Figure 7.2: These plots show the predic- tive distribution for an increasing num- ber of points in the training set. The more intense the colour, the more confident is the prediction (close to □ or □). As a con- sequence of using a Bayesian approach, the confidence of predictions increases with the number of observations.

This is in principle and M-dimensional integral. However, σ depends on w only through the 1-dimensional projection $a = w^T\phi(x^*)$, which is a linear combination of Gaussians, because w are normally distributed and ϕ are fixed. Hence q restricted to the dimension identified by a is still a Gaussian distribution: $q(a) \sim \mathcal{N}(a|\mu_a,\sigma_a^2)$, with $\mu_a = w_{MAP}^T\phi(x^*)$ and $\sigma_a^2 = \phi^T(x^*)S_N\phi(x^*)$, so that

$$p(C_1|x^*,\underline{y},\underline{x}) = \int \sigma(a)q(a)\,da$$

At this point we can use the **probit approximation** trick, i.e. we can approximate the previous integral by approximating the logistic function with the probit $\sigma(a) \approx \psi(\lambda a)$, such that $\lambda^2 = \frac{\pi}{8}$ and $\sigma'(0) = \psi'(0)$.
Hence:

$$\int \psi(\lambda a)q(a)\,da = \int \psi(\lambda a)\mathcal{N}(a|\mu_a,\sigma_a^2)\,da = \Psi\left(\frac{\mu_a}{(\lambda^{-2}+\sigma_a^2)^{\frac{1}{2}}}\right)$$

so that, approximating back to the logistic, we get:

$$p(C_1|x^*,\underline{y},\underline{x}) \approx \sigma(k(\sigma_a^2)\mu_a)$$

being $k(\sigma_a^2) = \left(1+\pi\frac{\sigma_a^2}{8}\right)^{-\frac{1}{2}}$.
In this way, the predictive distribution depends from μ_a but it is rescaled by the variance:

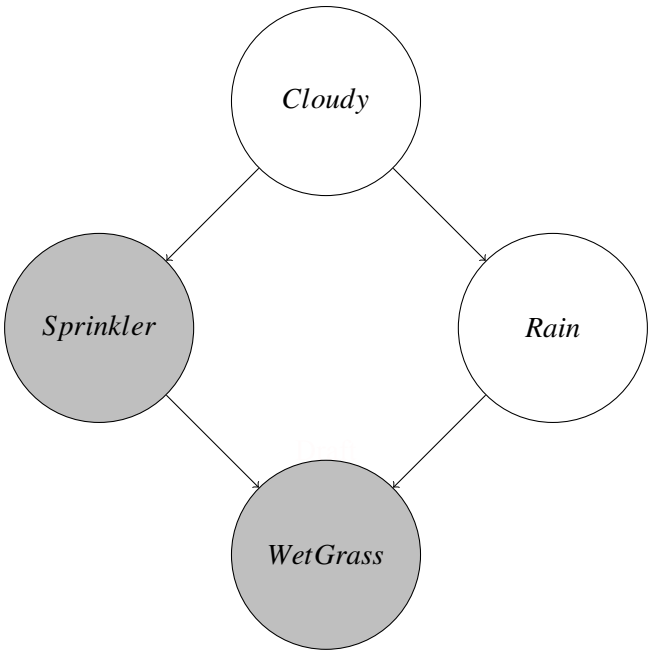
- if $\sigma_a^2 = 0$ then $p(C_1|x^*,\underline{y},\underline{x}) \approx \sigma(\mu_a)$
- if $\sigma_a^2 >> 0$ then $p(C_1|x^*,\underline{y},\underline{x}) \to \frac{1}{2}$ which represents the maximum level of uncertainty on the prediction

Remark: the dominating cost of this procedure is identifying the MAP.

8 Sampling-based Inference

Our focus in this chapter is on solving inference based problems by sampling strategies. This is needed because in the context of Bayesian Networks we are only able to perform inference if our network satisfies certain structural properties; in particular it has to be a tree or a poli-tree.

Even in a simple exampl like



does not satisfy the property that makes our belief propagation algorithm work.

Therefore we need to change our strategy. One possiblity is to rely on sampling to perform approximate inference on the probability distributions we are interested in.

The question then is how to sample from $p(x|y=\bar{y})$?

To do this we notice that we can generally compute the unnormalized probability distibution $\tilde{p}(x)$ such that

$$p(x) = \frac{1}{Z}\tilde{p}(x)$$

The typical scenario where this happens is given by the Bayes Theorem

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

where $p(y)$ is hard to compute (can be complicated high dimensional integral).

Our sampling problem than boils down to generate samples $x_1, \ldots, x_n$ from $p(x)$ knowing $\tilde{p}(x)$, as independent as possible among them. Once we have set of samples also able to estimate quantities like

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

We are going to see two main ways to perform this:

- Sample from $q(x) \approx p(x)$ and then correct. These are very common methods but they may suffer from efficiency issues.
- Markov Chain Monte Carlo (MCMC), which allows us to generate a set of samples from an unnormalized distribution.

Remark. Sampling is a computational intensive task and vanilla methods are not scalable to high dimensional probability distributions.

8.1 Approximate Sampling

We are going to explore methods that allow us to sample from a surrogate $q(x) \approx p(x)$.

Rejection Sampling

Suppose to have a distribution $p(x)$ and another distribution $g(x)$ from which it is easy to sample, called the **proposal distribution**, such that $\exists M > 0 : Mg(x) \geq p(x), \forall$, which of course implies that $\frac{p(x)}{Mg(x)} \leq 1$.

Rejection sampling consists in the following algorithm:

1. Sample $\hat{x}$ from $g(x)$
2. Accept $\hat{x}$ with probability $\frac{p(\hat{x})}{Mg(\hat{x})}$
3. If reject, then repeat the algorithm until acceptance

Graphically, once we sample $\hat{x}$, the acceptance mechanism corresponds to sample uniformly a number α between 0 and 1, multiply it by $Mg(x)$ and accept if and only if $\alpha Mg(x)$ falls below the original distribution at the point $\hat{x}$. We reject if the opposite happens.

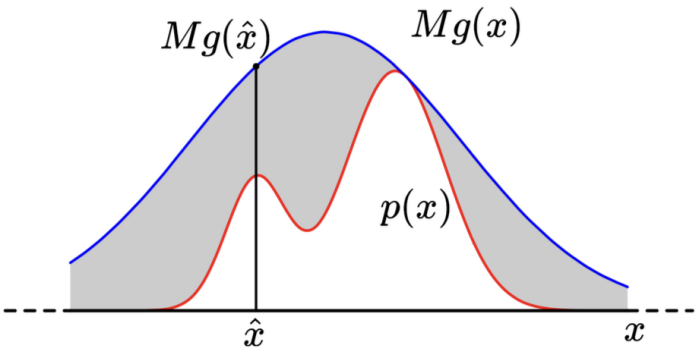


Figure 8.1: In the rejection sampling method, samples are drawn from a simple distribution $g(x)$ and rejected if they fall in the grey area between the distribution $p(x)$ and the scaled distribution $Mg(x)$. The resulting samples are distributed according to $p(x)$. [1]

Remark. We can actually compute the expected number of samples from g to accept a single sample on p, and this expectation is in fact M. Therefore, if M is large, this method becomes inefficient. Rejection sampling is especially inefficient in high dimension (you need a large M to "cover" p).

Remark. If we consider the unnormalized distribution $\tilde{p}(x)$ in place of $p(x)$ in the rejection sampling scheme, then we can use the fraction of rejected points to estimate the normalization constant Z. In fact,

$$Z = \int \tilde{p}(x)\,dx = M \int \frac{\tilde{p}(x)}{Mg(x)} g(x)\,dx = M \cdot p(accept)$$

Importance Sampling

The goal in importance sampling is to evaluate

$$\mathbb{E}_p[f(x)] = \int f(x)\frac{1}{Z}\tilde{p}(x)\,dx = \frac{\int f(x)\tilde{p}(x)\,dx}{\int \tilde{p}(x)\,dx}$$

where f if an integrable function.

We consider a proposal distribution $g(x)$ from which we know how to sample from, and we turn the expectation on p into an expectation on g.

$$\frac{\int f(x)\tilde{p}(x)\,dx}{\int \tilde{p}(x)} = \frac{\int f(x)\tilde{p}(x)\frac{g(x)}{g(x)}}{\int \tilde{p}(x)\frac{g(x)}{g(x)}} = \frac{\mathbb{E}_g\left[f\frac{\tilde{p}}{g}\right]}{\mathbb{E}_g\left[\frac{\tilde{p}}{g}\right]}$$

Then we sample $x_1, \ldots, x_n$ from $g(x)$ and replace the expectation with sums

$$\frac{\mathbb{E}_g\left[\frac{p}{g}\right]}{\mathbb{E}_g\left[\frac{p}{g}\right]} = \frac{\frac{1}{N}\sum_{i=1}^{N} f(x_i)w(x_i)}{\frac{1}{N}\sum_{i=1}^{N} w(x_i)}$$

where

$$w(x_i) := \frac{\tilde{p}(x_i)}{g(x_i)}$$

are known as the **importance weights**.

If $\frac{f\tilde{p}}{g}$ is approximately constant, then estimates can be very good. If weights vary a lot instead, we have a large variance and consequently a poor estimate. So, the quality of the result depends from the choice of $g(x)$.

Notice that $\frac{1}{N}\sum_{i=1}^{N} w(x_i) \approx Z$, so we also have an approximation of the partition fraction.

Remark. This technique can be used to estimate functions of rare events, increasing the probability that the event happens and then correcting it.

Example:

If we want to compute $\mathbb{E}_p[f(x)]$ where $p(x) = \mathcal{N}(x;0,1)$ and $f(x) = x^{20}$, using samples from $p(x)$ would be inefficient, as the expected value is mostly influenced by extreme values of x. One can then use importance sampling with a Gaussian with larger variance as proposal distribution.

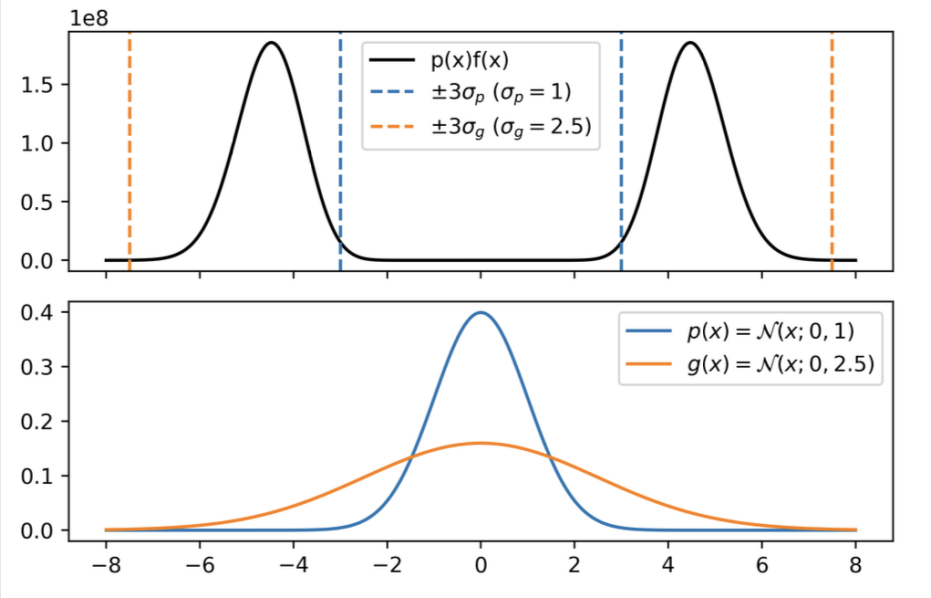


Figure 8.2: Importance sampling example. Sampling from $p(x) = \mathcal{N}(x;0,1)$ to compute $\mathbb{E}_p[f(x)]$ with $f(x) = x^{20}$ would be inefficient, as most of the contribution to the integral is given by values of x that are rare ($|x| > 3\sigma_p$). Using as proposal distribution a Gaussian with larger variance allows us to sample from the region that contributes to the expected value.

8.2 Markov Chain

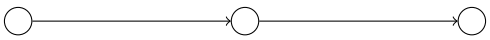
A Markov Chain is a stochastic process, denoted as

$$\{X_t\}_{t\geq 0}$$

with $t \in \mathcal{N}, X_0, X_1, \ldots, X_t \in \mathcal{X}$ where $\mathcal{X}$ can be discrete or continuous.

It can be described as a *dynamical system*, i.e. a system in which we start from a certain state x_0 with a given probability $p_0(x)$ and that changes state according to a dynamic described by the *transition kernel* $p(x_n|x_{n-1})$.

Remark. Markov Chains satisfy the *memoryless property*, which corresponds to the assumptions encoded into the following probabilistic graphical model:



Therefore we have that

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

which is known as the memoryless or Markov property.

We also require the **time homogeneity** property that states that

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

which means that the probability to jump from a certain state to another state stays the same for every time step.

Definition: ***Transition Kernel***

$p(x_n|x_{n-1})$ satisfying the time homogeneity property is called the (one step) **transition kernel**.

For a discrete state space we would have a *transition matrix*, while for a continuous state space, $p(x_n|x_{n-1})$ is a probability density on x_n depending continuously on x_{n-1}.

Since we are interested at the behaviour of the Markov Chain as the index n progresses, i.e. for large times, we need to define few more concepts.

Definition: ***Ergodic Markov Chain***

A *Markov Chain* is **ergodic** $iff \forall x,y \in \mathscr{X}, \exists t \geq 0 : p(x_t = y|x_0 = x) > 0$.

If a Markov Chain is ergodic, it means that there is always the possibility of going from a given state x to another given state y if we are patient enough. This means that the entire state space is reachable, no matter where we start exploring.

Definition: ***Stationary Distribution***

A **Stationary Distribution** $\Pi(y)$ is such that

1. $$\Pi(y) = \int p(y|x)\Pi(x)\,dx$$
   which means that Π is an **invariant measure** with respect to the Markov Chain dynamics defined bu the Transition Kernel $p(y|x)$
2. $p_n(x) = p(x_n|x_0) \xrightarrow[n\to\infty]{} \Pi(x)$
3. Π is unique

Detailed Balance

The following condition is sufficient to guarantee that $\Pi(x)$ is a stationary distribution.

Definition: ***Reversible Markov Chain***

We say that a Markov Chain is **reversible** (or it satisfies the **balance condition**) $iff \exists \Pi(x)$, probability distribution such that

$$p(x|y)\Pi(y) = p(y|x)\Pi(x)$$

proposition 2. *If an ergodic Markov Chain is reversible with respect to the distribution $\Pi(x)$, then $\Pi(x)$ is a stationary distribution.*

Proof. If the Markov Chain is reversible then we can write

$$\int p(y|x)\Pi(x)\,dx = \int p(x|y)\Pi(y)\,dx = \Pi(y)\int p(x|y)\,dx = \Pi(y)$$

□

Remark. It is not true that the existence of a stationary distribution implies that our Markov Chain is reversible.

We are now ready to tackle Markov Chain Monte Carlo: by requiring that the Markov Chain we are going to define satisfies the detailed balance condition for a given distribution $\Pi(x)$, we will be eventually ($t \to \infty$) able to sample from distribution of interest, i.e. $\Pi(x)$.

8.3 Markov Chain Monte Carlo

Metropolis Hastings

Intuitively, the idea is that we want to sample from a distributino and we build an ergidic Markov Chain with a transition kernel such that the stationary distribution coincides with the one we want to sample from.

Assume that we want to sample from $p(x) = \frac{1}{Z}\tilde{p}(x)$, where we know the unnormalized distirbution $\tilde{p}(x)$ but the normalization constant is too complicated to compute. We fix $q(x|y)$, the *proposal kernel* of our Markov Chain, such that

1. It is easy to sample from $q(x|y)$
2. It makes our Markov Chain ergodic

A typical choice for our proposal kernel is to use a Gaussian distribution, in which the variance is chosen in such a way that makes it quick to reach the stationary distribution.

Suppose that we start from a certain state x at the time step t.

The algorithm works as follows:

1. We sample y from $q(x|y)$
2. Borrowing from rejection sampling, we are going to accept/reject the new sampled point based on the following rule

$$x_{t+1} = \begin{cases} y & \text{with probability } \alpha(y|x) = \min\left\{1, \frac{\tilde{p}(y)q(x|y)}{\tilde{p}(x)q(y|x)}\right\} \\ x & \text{otherwise} \end{cases}$$

The criterion for defining

$$\alpha(y|x) = \min\left\{1, \frac{\tilde{p}(y)q(x|y)}{\tilde{p}(x)q(y|x)}\right\}$$

is called the **Metropolis-Hastings** criterion; for symmetric transition kernels $q(x|y) = q(y|x)$, it becomes the Metropolis criterion.

Observe that

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

and so the regions with higher probability $p(y) > p(x)$ are more likely to be visited.

Notice that we haven't defined in a full mathematical way the transition kernel of our Markov Chain here, but it can be derived bu the operational procedure given above.

Lemma 1. *The Metropolis-Hastings acceptance criterion satisfies the detailed balance condition.*

Proof. Let's start from the full transition kernel in an implicit form and suppose that $y \neq x$. Then we need to prove that

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

If $y \neq x$, the first term reduces to the product of the probability to sample y given x, which is given by $q(y|x)$, times the probability to accept y, which is given by $\alpha(y|x)$. Therefore

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

Notice that

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

Performing some calculations we obtain

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

□

There might still be issues: if we pick a bad q the algorithm may take a lot of steps before reaching the stationary distribution. We say that the **mixing time** is high. This time can be estimated by some statistics on the chain that is being sampled, that form a sort of *diagnostics tools*. However, there is not a way to determine the exact moment in which the steady state is reached. Moreover, another issue is that the samples x_n, x_{n+1} are not independent.

Another good property that we may want to have in our proposal kernel is to have a good balance in between exploration and exploitation, which becomes especially important in multimodal distribution.

Gibbs Sampling

This time, let's write the probability distribution from which we want to sample from as $p(x) = p(x_1, \ldots, x_n)$ and suppose that we know how to sample from the 1-dimensional conditionals, i.e. from $p(x_i|x_{-i})$ where $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i-1}, \ldots, x_n)$.

The main loop of Gibbs sampling algorithm is the following:

1. Pick $k \in 1, \ldots, n$
2. Set $x_j^{(t+1)} = x_j^{(t)}$ for $j \neq k$ (the j-th component of the t+1 state)
3. Sample $x_k^{(t+1)}$ from $\tilde{p}(x_k|x_{-k}^{(t)})$

Therefore we are choosing a coordinate and sample along that coordinate, keeping everything else fixed. We have different ways to choose k:

1. Round-Robin strategy, i.e. sample starting from 1, go up to k on the first k steps and then repeat.
2. Choose uniformly at random.

Lemma 2. *The transition kernel of Gibbs Sampling is exactly the same as Metropolis-Hastings algorithm.*

Proof. In fact, for Gibbs Sampling we have

$$q_k(y|x) = \begin{cases} p(x_k|x_{-k}) & \text{when } y_{-k} = x_{-k} \\ 0 & \text{otherwise} \end{cases}$$

Which means that $\alpha_k(y|x) = 1$ because

$$\alpha_k(y|x) = \frac{p(y)q(x|y)}{p(x)q(y|x)} = \frac{p(y_k|y_{-k})p(y_{-k})}{p(x_k|x_{-k})p(x_{-k})} \cdot \frac{p(x_k|y_{-k})}{p(y_k|x_{-k})}$$

Where the equality is given by the standard conditional probability expansion. Also notice that $x_{-k} = y_{-k}$, otherwise there would be no jump according to the definition of our proposal kernel. Then

$$\frac{p(y_k|y_{-k})p(y_{-k})}{p(x_k|x_{-k})p(x_{-k})} = \frac{p(y_k|y_{-k})}{p(x_k|x_{-k})} = 1$$

□

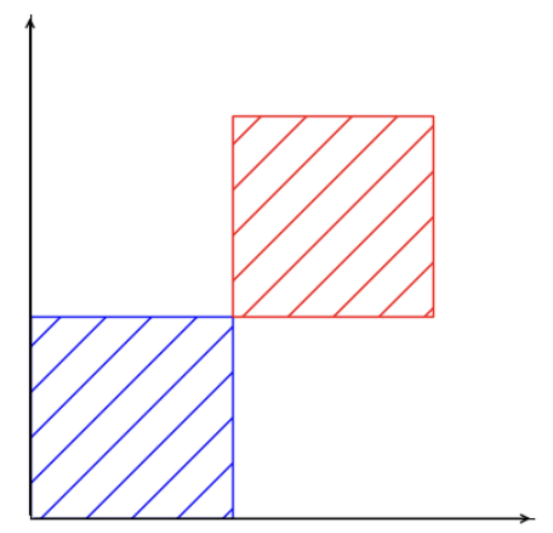
Gibbs Sampling algorithm can be generalized, for example, by sampling blocks instead of a single variable, i.e. we can sample $x_j, \dots, x_k \subseteq x$. Moreover, we could apply this even if we don't know explicitly how to sample from $p(x_i|x_{-i})$, for example by applying rejection sampling, or even a Metropolis-Hastings MCMC to smaple our conditional distribution (the latter strategy is called "Metropolis within Gibbs").

Remark.

- We may not satisfy ergodicity in Gibbs sampling, since it may not always be possible to find a path between two stated which is only made of "single component" steps.

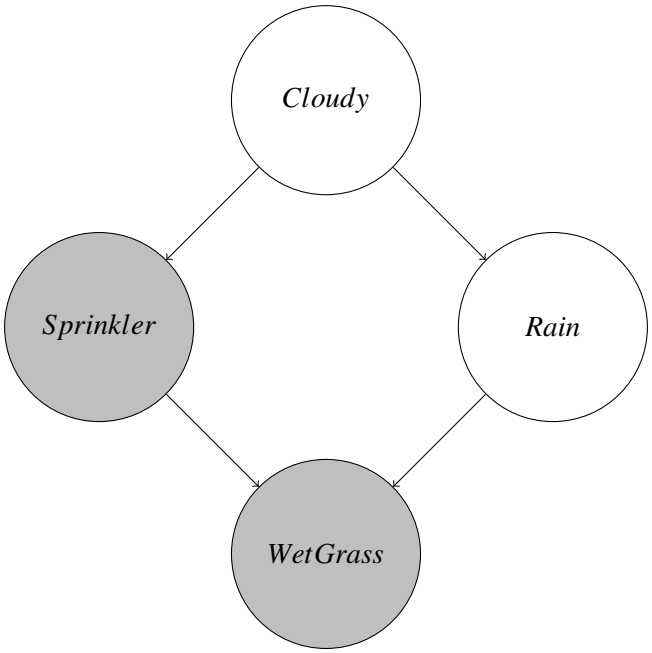
Example:

Consider the bivariate distribution $p(x,y) = 2$ if $(x \in [0,0.5] \wedge y \in [0,0.5]) \vee (x \in [0.5,1] \wedge y \in [0.5,1]), 0$ otherwise. In this case, we cannot sample from an "island" different from the one where we start sampling.



- If variables are strongly correlated, then our mixing time can be very high.

Sampling based inference in PGM



Suppose that we observe the variables "Sprinkler" and "Wet grass", but we don't observe "Cloudy" and "Rain".

More generally, suppose to have a set of variables (could be vector of random variables), x and y that are described by a PGM and we know that y is observed, $y = \hat{y}$. We want to make inference on x, which typically means computing $p(x|y = \hat{y})$. We aim at sampling from this probability, but we do not have access to such a conditional distribution. We only know $\tilde{p}(x) = p(x, y = \hat{y})$, but not the normalization constant $X = p(y = \hat{y})$.

Here are some strategies then to generate samples from this distribution:

- Rejection sampling: sample from the full joint $p(x,y)$ which can be done by ancestral sampling (which is fast) and then reject if $y \neq \hat{y}$. If we observe a lot of variables then this becomes very inefficient.
- A better strategy is to use MCMC. We can sample from $p(x, y = \hat{y})$ using some proposal distribution (which will depend on the variables that we are trying to sample). Although the state of the art of MCMC (Hamiltonian Monte Carlo, explained later on) is generally good, in the framework of Bayesian Networks we can do better.
- If we can compute efficiently the one dimensional conditional distribution, i.e. $p(x_i|x_{-i}, y) = p(x_i|MB_i)$, where MB_i is the Markov Blanket of x_i, we can use *Gibbs sampling*. This is especially efficient with discrete, and relatively small state spaces.

Convergence Diagnostics

How can we check that out Markov Chain has reached the steady state? We will put forward a set of tools that can monitor one or more trajectories and roughly tell us whether we reached it or not. Notice that since we are interested in sampling the stationary probability distributions, we shall start keeping the samples only when we actually reached the steady state.

Let's define some notation for the rest of the section. We are going to denote a general function over a state of the MCMC trajectory $(X_t)_{t\geq 0}$ as $\Psi : \mathscr{X} \to \mathbb{R}$. This function can be a lot of different things, depending on what we are interested in computing, e.g. a projection on single coordinate.

We will assume that Ψ has values in $\mathbb{R}$, and not just in a subset of it, and, if it does, we transform the function Ψ to make it compliant to this assumption (taking the logarithm of quantities in between $(0,\infty)$ for example). Let's fix some further notation:

- $x_1, \ldots, x_n$ is our sampled trajectory

- $\Psi_j := \Psi(x_j)$
- $\bar{\Psi} := \frac{1}{N}\sum_j \Psi_j$ is the estimate of $\mathbb{E}[\Psi] = \int \Psi(x)p(x)\,dx$

On a high level, the idea is to look at more than one chain and compare the distribution of the samples that we obtain and see if they look more or less the same.

Practically,

1. We sample $\frac{m}{2} \geq 1$ trajectories from overdispersed initial points. We try to start from different states that are far away in our state space.
2. Sample for $4n$ steps.
3. We throw away the first half of every trajectory so that we have only $2n$ points left. This phase is known as the **burn-in** or **warm-up** phase. We do this because it takes time to reach the steady state (notice this is an heuristic: convergence to the stationary distribution can happen faster or slower than $2n$ steps).
4. Then we split in 2 parts the remain trajectories, so that we are left with m different trajectories each of length n.

From now on we will denote each sample as x_{ij} where $i \in [1,n]$ and $j \in [1,m]$, where this notation describes the i_{th} sample of the j_{th} trajectory.

We will also denote $\Psi(x_{ij}) = \Psi_{ij}$ and define

$$\begin{cases} \bar{\Psi}_j := \frac{1}{n}\sum_{i=1}^{n} \Psi_{ij} \\ \bar{\Psi} := \frac{1}{m}\sum_{j=1}^{m} \bar{\Psi}_j \end{cases}$$

Hence, $\bar{\Psi}_j$ is the average within the trajectory j and $\bar{\Psi}$ the average over all the trajectories, respectively. We are also interested in the variance of $\bar{\Psi}$, but since our samples are not independent, we don't have that $VAR[\bar{\Psi}] = \frac{1}{n}VAR[\Psi]$, i.e. the variance of the estimator in this case is not just the variance of our random variable divided by the number of samples. If you think about two consecutive points in the chain, there is a high chance that the correlation between them is higher than that of two points which are sampled distantly in time from one another.

Let's define these two quantities:

$$W := \frac{1}{m}\sum_{j=1}^{m} s_j^2, \quad s_j^2 := \frac{1}{n-1}\sum_{i=1}^{n}(\Psi_{ij} - \bar{\Psi}_j)^2$$

$$B := \frac{n}{m-1}\sum_{j=1}^{m}(\bar{\Psi}_j - \bar{\Psi})^2$$

W is called the **within variance** while B is called the **between variance**.

We know by definition that

$$W \leq VAR[\Psi],$$

because when sampling single trajectories we have not necessarily explored and visited the full space. Increasing the number of samples we will converge the true variance. Moreover, as long as the initial states are overdispersed, it can be shown that

$$VAR[\Psi] \leq VAR^{+}[\Psi] := \frac{n-1}{m}W + \frac{1}{n}B$$

Then we have both a lower and an upper bound for our variance, both converging to the true variance. Therefore we can compute the statistics

$$\hat{R} := \sqrt{\frac{VAR^{+}[\Psi]}{W}}$$

but unfortunately this is not the case as we have already seen.

The correlations among nearby points (when positive as typically the case) are increasing the variance of the estimator $\bar{\Psi}$, in a way which can be expressed by this formula:

$$nmVAR[\bar{\Psi}] \approx \left(1 + 2\sum_{k=1}^{\infty} \rho_k\right) VAR[\Psi]$$

Where ρ_k is the autocorrelation of lag k, which is, by definition

$$\rho_k := Corr[\Psi(x_i), \Psi(x_{i+k})]$$

So it is the correlation in between two points in the same chain which are k steps apart.

If we compare the formula above with what we would have in case of independence, we can find the **effective number of samples** produced by our chain

$$n_{eff} = \frac{nm}{(1 + 2\sum_{k=1}^{\infty} \rho_k)} \leq nm$$

And we can also write

$$VAR[\bar{\Psi}] = \frac{VAR[\Psi]}{n_{eff}}$$

A typical rule of thumb here is to reach at least $n_{eff} = 100$ effective samples.

The question now is: how do we compute the autocorrelation? We know that this identity holds (no proof given)

$$\mathbb{E}[(\Psi_i - \Psi_{i-k}^2)] = 2(1 - \rho_k)VAR[\Psi]$$

and since we can estimate both the left hand side term and $VAR[\Psi]$ from our data, we can also estimate ρ_k by inverting the formula.

The expectation above is named **Variogram at lag k** and can be estimated as

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

So that

$$\hat{\rho}_k = 1 - \frac{V_k}{2VAR^{+}[\Psi]}$$

We still have problems in the limit of large k because we would have few samples and very noisy estimates. Therefore we will stop the sum over k when the following condition is satisfied

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

When this condition is satisfied we are in a regime where our summation is now relevant anymore.

Therefore, we approximate the sum as

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

In conclusion, we have two different diagnostics tool to detect convergence: either we look at an estimate of the variance and compute the factor $\hat{R}$ or we look at the number of effective samples and have an estimate of how decent our approximation is going to be.

8.4 Hamiltonian Monte Carlo

Hemiltonian Monte Carlo can be considered as the state of the art method for doing Markov Chain Monte Carlo. It falls into the category of *augmented varibles* Monte Carlo methods.

The idea it to turn the problem into an Hemiltonian, augmenting the state space with momentum variavbles that provide a sort of "kinetic energy" that allows the algorithm to move along the surface of the energy corresponding to our probability distribution. This scheme improves a lot of the mixing time, hence the efficiency of MCMC algorithms. Moreover, it can be used in the case of high-dimensional multimodal distributions because it does not remain stuck in a single mode.

As usual, we start in a situation in which we want to sample from $p(x) = \frac{1}{Z}\tilde{p}(x)$, and now we express our distribution as

$$\frac{1}{Z}\tilde{p}(x) = \frac{1}{Z_x}\exp(H_x(x))$$

This procedure is called the *Boltzmann Trick* and it is always possible (just take the logarithm of the distribution) and turns our probability distribution in the form of an energy.

Then, we introduce momentum variables y, as many as the number of x variables that we have, and we assign them the probability distribution $p(y) = \frac{1}{Z_y}\exp(H_y(y))$, where is typical to make the assumption

$$H_y(y) = \frac{1}{2}y^t y$$

i.e. to consider $p(y)$ a standard Gaussian.

We are going to sample from the joint distribution, exploiting the independence of our variables

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

The idea of the algorithm is to sample from $p(x,y)$ and then forget about y. But how do we sample? We are going to sample according to the force field defined by the Hamiltonian, i.e. along lines which keep the energy constant. This means that, given some velocity, we follow a trakectory on the probability distribution space accordingly to the equation of motion in order to explore the space without losing energy. In fact, we would like to preserve energy because we want to move between regions with high probability.

Hence we have the following algorithm:

1. We start from point x_i
2. We sample $y \sim p(y)$, i.e. we randomize the momentum
3. We choose a random direction in time, i.e. sample from {-1,1}
4. We move according to Hamiltonian dynamics from (x_i, y) to a candidate (x', y') doing L steps
5. We introduce the following rejection criterion: we accept if $H(x',y') > H(x,y)$, otherwise we accept with probability $\exp(H(x',y')) - H(x,y)$

Moving accordingly to Hamiltonian dynamics means to move from (x,y) to (x',y') keeping $H(x',y') = H(x,y)$. At each step we set $(x',y') = (x+\Delta x, y+\Delta y)$ and Taylor expand the Hamiltonian

$$H(x+\Delta x, y+\Delta y) \approx H(x,y) + \nabla_x H_x(x)^T \Delta x + \nabla_y H_y(y)^T \Delta y$$

Then, enforcing the condition $H(x',y') = H(x,y)$, we can derive the equations for our movement:

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

and we do L steps of this dynamics to find (x',y').

Notice that, even if in principle we require the Hamiltonian to stay constant (and in that case we would always accept because $\exp(H(x',y') - H(x,y)) = \exp(0) = 1$), there are still numerical integration errors introduced by the approximation above which actually let us change the value of the Hamiltonian, hence we need to define the acceptance criterion as above to ensure that the dynamics compensate this error.

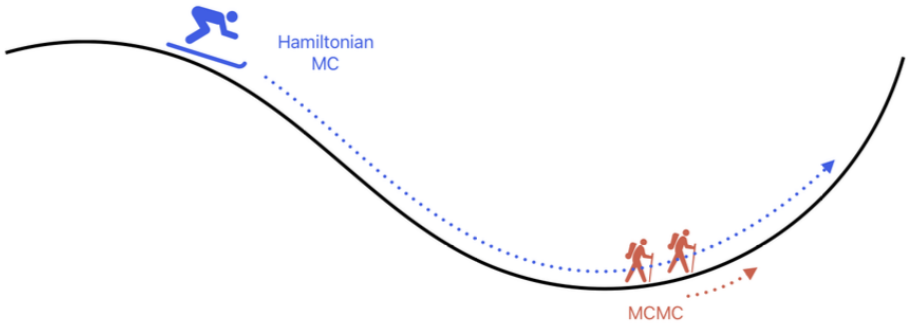
Remark Since we are required to compute gradients, Hamiltonian Monte Carlo works only for continuous variables. The advantage of exploiting gradient information is similar to the one in gradient based optimizers (gradient descent).

Remark Our acceptance criterion is, in fact, Metropolis acceptance criterion

$$\alpha = \min\left\{1, \frac{p(x',y')}{p(x,y)}\right\} = \min 1, \exp(H(x',y') - H(x,y))$$

Additional heuristics, like the *no-u-turn* Hamiltonian Monte Carlo, are the state of the art of Monte Carlo methods.

Remark The number of effective samples is higher than standard MCMC, because with Hamiltonian Monte Carlo we take larger steps.



9 Expectation Maximization

Expectation maximization is a method that allows us to perform *maximum likelihood* inference in scenarios where computing the likelihood explicitly may not be possible because of the presence of **latent variables**.

Learning Bayesian Networks

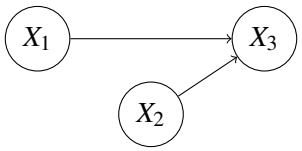
We will start by focusing on a simpler problem to set the scene. Remember that a BN is a way to factorize a joint distribution according to:

$$p(x) = \prod_i p(x_i|pa(x_i))$$

We know how to compute inference, marginals, etc... but how can we learn Bayesian Networks from data? Here we focus on the problem of learn parametric models of conditional distributions, for a fixed structure of the BN.

We typically have that each factor $p(x_i|pa(x_i),\theta_i)$ depends also on some parameters θ_i which can be estimated by Maximum likelihood (ML) from data.

Let's start with the following example



$$p(X_1,X_2,X_3) = p(X_3|X_1,X_2)p(X_1)p(X_2)$$

and suppose that $x_i \in \{0,1\}$. Focus on learning the factor $p(x_3|x_1,x_2)$ and consider all the possible value in the conditioning set. Therefore we want to learn the following values:

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

Learning by maximum likelihood these parameters just means computing

$$\theta_{ij} = \frac{\#(x_1 = i,x_2 = j,x_3 = 1)}{\#(x_1 = i,x_2 = j)},$$

where the notation # returns the cardinality of the set of observations satisfying the constraint in brackets.

Remark. In a continuous setting you may want to model $p(x_3|x_1,x_2)$ as a combination of values of x_1,x_2. The specific choice of the parametric model depends on what is the phenomena you are trying to model.

In this context, as long as the number of parents of each x_i is relatively small, computations are feasible. What happens though if we don't observe, for example, x_1? We cannot employ this model for latent variables!

Problem formulation

Let's consider a more general scheme like the following: we have a certain number of variables x which are observed and a set of variables z which is unobserved. Generally speaking, we will have a parametric model $p(x,z|\theta)$ and we would like to compute θ_{ML}. In order to identify θ_{ML} we need to optimize the marginalized likelihood on x:

$$p(x|\theta) = \sum_z p(x,z|\theta)$$

If z is high-dimensional, we would to sum over exponentially many possible states, which makes our problem intractable.

The problem becomes even more complex when we have $\underline{x} = x_1,\dots,x_n$ observations and $\underline{z} = z_1,\dots,z_n$ latent states of observations since we would like to work with the logarithm of our distribution for numerical stability

$$p(\underline{x},\underline{z}|\theta) = \prod_n p(x_n,z_n|\theta) \quad \log p(\underline{x},\underline{z}|\theta) = \sum_n \log p(x_n,z_n|\theta)$$

but then

$$\log p(\underline{x}|\theta) \neq \sum_n \log p(x_n|\theta)$$

because of the summation over z in the definition of $p(x|\theta)$.

9.1 Evidence lower bound

Again, consider a joint model $p(x,z)$ with x observed and z unobserved with corresponding observations $x_1,\dots,x_n$ and latent states $z_1,\dots,z_n$. Our goal is to compute $p(x|\theta) = \sum_n p(x,z|\theta)$ and we want to learn θ_{ML}, i.e. θ such that

$$\arg\max_\theta p(\underline{x}|\theta)$$

This problem is typically intractable for the reasons described before.

We still introduce a key approximation which will also be useful for variational inference. Consider

$$p(x,z|\theta) = p(x|\theta)p(z|x,\theta)$$

The idea is to approximate $p(z|x,\theta)$ with the so called **variational approximation** $q(z)$ (which can be any distribution over z).

Let's consider the **Kullback-Leibler divergence** of q and p

$$KL[q||p] = KL[q(z)||p(z|x,\theta)] = \mathbb{E}_{q(z)}\left[-\log\frac{p(z|x,\theta)}{q(z)}\right] = \sum_z q(z)\log\frac{p(z|x,\theta)}{q(z)}$$

The trick now is to add and subtract the term $|\log p(x|\theta)$ in the logarithm

$$-\sum_z q(z)\log\frac{p(z|x,\theta)}{q(z)} = -\sum_z q(z)\left[\log\frac{p(z|x,\theta)}{q(z)}\right] + \log p(x|\theta) - \log p(x|\theta)$$

And then we can start to aggregate these factors obtaining

$$KL[q(z)||p(z|x,\theta)] = -\sum_z q(z)\log\frac{p(x,z|\theta)}{q(z)} + \log p(x|\theta)$$

Definition: *ELBO*

$$\mathscr{L}(q,\theta) = \sum_z q(z)\log\frac{p(x,z|\theta)}{q(z)}$$

is the **Evidence Lower Bound** (also called **ELBO**).

From the expression above we can rewrite our log likelihood as

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

Remark. Since $KL[q|p] \geq 0$ then $\mathscr{L}(q,\theta) \leq \log(p(x|\theta))$ i.e. it is a lower bound for the log likelihood.

Whenever we have a tractable form for $p(x,z|\theta)$ then the ELBO is also tractable. Expectation Maximization will actually optimize the ELBO with respect to both θ and q (since q is a distribution we are using the tools of variational calculus to perform this optimization) in an "alternated" fashion (optimize with respect to one and the other argument in two different steps).

9.2 Expectation Maximization

Let's start from this expression for the ELBO

$$\mathscr{L}(q,\theta) = \mathbb{E}_q\left[\log\frac{p(x,z|\theta)}{q(z)}\right] = \mathbb{E}_q[\log p(x,z|\theta)] + \mathbb{E}_q[-\log q(z)]$$

The first term is called the **Energy term**, while the second term is actually the entropy of the q distribution (written $H(q)$).

The goal of the EM algorithm is to find $\theta_{ML} = \arg\max_\theta \log p(\underline{x}|\theta)$ which, since it is analitically intractable as we have seen, will lead us to maximize the ELBO, in both q and θ.

The way in which we proceed is to maximize for each variable on different steps. The maximization with respect to q is known as the **expectation maximization** step (or the m-step).

E-step

Let's fix θ to θ_{OLD}. We need to solve an optimization problem in a function space. This is easy though if we notice that $\mathscr{L}(q,\theta)$ is maximum whenever $KL[q(z)||p(z|\underline{x},\theta)] = 0$, since $\log p(\underline{x}|\theta)$ does not depend on q and

$$\mathscr{L}(q,\theta) = \log p(\underline{x}|\theta) - KL[q(z)||p(z|\underline{x},\theta)]$$

Therefore

$$q_{max} = p(\underline{z}|\underline{x},\theta_{OLD})$$

where

$$p(\underline{z}|\underline{x},\theta) = \prod_{i=1}^{N} p(z_i|x_i,\theta) \quad \text{if } x_1,\ldots,x_n \text{ are i.i.d}$$

Then we compute $\mathbb{E}_{q_{new}}[\log p(\underline{x}, z|\theta)]$ as a function of θ and the e-step is completed.

Remark. In order for the EM algorithm to work effectively, we need to be able to evaluate analytically or numerically the conditional distribution $p(\underline{z}|\underline{x}, \theta_{OLD})$.

M-step

Now we need to maximize $\mathscr{L}(q, \theta)$ keeping q fixed to $q = q_{new} = p(\underline{z}|\underline{x}, \theta_{OLD})$ which is tantamount to maximize w.r.t. θ

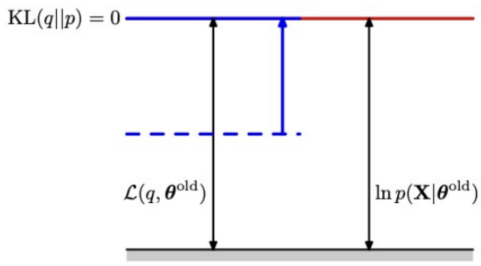
$$\mathbb{E}_{q_{new}}[\log p(z, \underline{x}|\theta)]$$

i.e. the energy term.

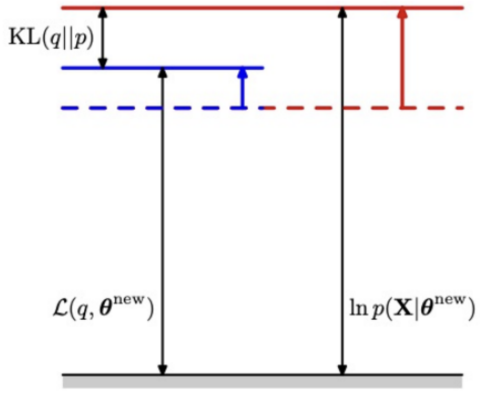
This maximization really depends on the problem at hand, but usually is just about explicitly computing the gradient, setting it to zero, and find

$$\theta_{new} = \arg\max_{\theta} \mathbb{E}_{q_{new}}[\log p(\underline{x}, z|\theta)]$$

The algorithm works by repeating these two steps until convergence (i.e. whenever $||\theta_{OLD} - \theta NEW|| < \varepsilon$). The reason why this works is easily explained in a graphical way.



After setting the Kullback-Leibler divergence to zero, we close the gap vertically and we make the log-likelihood equal to the lower bound.



In the M-step, since we are maximizing the lower bound, we are pushing it up, but then also the log-likelihood will be pushed up, possibly of a larger quantity than the lower bound, creating a new gap.

After both the E and M steps are completed, the log-likelihood is always increasing, until it gets really close to a local maximum and eventually reaches it. So we can consider as the termination condition of the EM algorithm the following: $\mathscr{L}(q_{NEW}, \theta_{NEW}) - \mathscr{L}(q_{OLD}, \theta_{OLD}) < \delta$, where δ is a chosen constant.

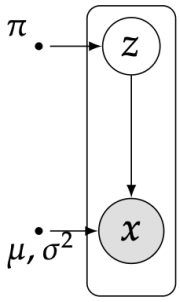
Remark. In general the EM algorithm converges to a **local** optimum of $\log p(\underline{x}|\theta)$.

Remark. If $p(\underline{z}|\underline{x},\theta)$ is not easily computable, we can use a q that is an approximation. In this case EM interactions will reduce $KL(q||p)$ but it will not be set to zero, hence convergence is not guaranteed.

9.3 Mixture of Gaussians

We will see an application of the EM algorithm in a typical scenario.

Let's start with a graphical representation of a Gaussian Mixture model:



where z is a discrete R.V. $z_1,\dots,z_k,z_j \in \{0,1\},\sum_j z_j = 1$ which means that we have a **one-hot-encoding** representation for z. π is a discrete probability distribution that defines the probability of being assigned to the j-th component of the Gaussian mixture, while μ,σ^2 are the mean and the variance of each Gaussian distribution.

Moreover we have N observations of x which means that we also have N corresponding z unobserved realizations of the variable $\underline{z} = (z_{ij}), i = 1,\dots,N, j = 1,\dots,k$.

The joint distribution for the GMM model, using the one-hot encoding representation for our variates is

$$p(x,z|\theta) = \prod_{j=1}^{K} \pi_j^{z_j} \cdot \mathcal{N}(x|\mu_j,\sigma_j^2)^{z_j}$$

where notice that z_j is 1 only for one j-th component and zero otherwise, which means that in the product we only really get one factor (the others are all 1).

This is not the typical thing that we write for a mixture of Gaussians though: since we don't observe z, we need to write the marginal distribution

$$p(x|\theta) = \sum_{j=1}^{K} \pi_j \mathcal{N}(x|\mu_j,\sigma_j^2)$$

The problem is that the direct optimization of this is intractable. Then we also need to consider the other terms that we need in our algorithm, i.e.

$$\begin{gathered} p(z|\theta) = \textstyle\prod_j \pi_j^{z_j} \\ p(z|\underline{x},\theta) \propto \textstyle\prod_{n=1}^{N}\prod_{j=1}^{K} \pi_j^{z_{nj}} \mathcal{N}(x_n|\mu_j,\sigma_j^2)^{z_{nj}} \\ p(z=j|x,\theta) = \frac{\pi_j \mathcal{N}(x|\mu_j,\sigma_j^2)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x|\mu_i,\sigma_i^2)} \end{gathered}$$

where the last probability distribution is considered since we have seen that, given i.i.d. r.v., our conditional distribution factorizes as $p(\underline{z}|\underline{x},\theta) = \prod_{i=1}^{N} p(z_i|x_i,\theta)$.

In this model we can easily evaluate the conditional distribution of z given x, which is what makes the computation of the EM algorithm effective.

Then it is relatively simple to comptute

$$\mathbb{E}_{p(z|x)}[z_{nj}] = p(z=j|x_n,\theta) = \frac{\pi_j \mathcal{N}(x_n|\mu_j,\sigma_j^2)}{\sum_i \pi_i \mathcal{N}(x_n|\mu_i,\sigma_i^2)} := \gamma(z_{nj})$$

which is called the **responsibility** in the gaussian mixture model scenario, and it is needed to compute what we actually care about for the e-step of the EM, which is the expectation of

$$\log p(\underline{x},\underline{z}|\theta) = \sum_{n=1}^{N}\sum_{j=1}^{K} z_{nj}[\log \pi_j + \log \mathcal{N}(x_n|\mu_j,\sigma_j^2)]$$

that is

$$\mathbb{E}_{p(\underline{Z}|\underline{X},\theta)}[\log p(\underline{x},\underline{z}|\theta)] = \sum_{n=1}^{N}\sum_{j=1}^{K} \mathbb{E}[z_{nj}][\log \pi_j + \log \mathcal{N}(x_n|\mu_j,\sigma_j^2)]$$

Now we can solve in close form the optimization problem, which means that we can compute analytically the derivatives with respect to our parameters in order to determine the maxima. Performing calculations we find

$$\begin{gathered}
\mu_j^{new} = \tfrac{1}{N_j}\textstyle\sum_n \gamma(z_{nj}) x_n \\
\Sigma_j^{new} = \tfrac{1}{N_j}\textstyle\sum_n \gamma(z_{nj})(x_n-\mu_j^{new})^T(x_n-\mu_j^{new}) \\
\pi_j^{new} = \tfrac{N_j}{N}, \quad N_j = \textstyle\sum_{n=1}^{N}\gamma(z_{nj})
\end{gathered}$$

where π_j^{new} is the count of the probability that each observation comes from the j-th component over all the observations; μ_j^{new} is a weighted mean of the variables x for all the observations that comes from component j; Σ_j^{new} is again a weighted average of all the covariances.

Remark. A good initial guess for the parameters for our algorithm is given by runing a k-means clustering and considering the averages and covariances of each cluster as a starting value for μ_j and Σ_j. In fact, k-means could be thought of as an approximation of EM, where covariances are assumed to be identical and sperical.

9.4 EM for Bayesian Networks

We are going back to the motivating example that we have seen in the introduction to explain how to solve the problem we posed.

Let's consider a BN on a certain set of variables such that their joint distribution is such that

$$p(x) = \prod_i p(x_i|pa(x_i), \theta_i)$$

And also we have our usual assumption that the set of variables x is divided into a set of visible (observable) variables v, and a set of hidden variables z. We can alternatively write then

$$p(x) = p(v, z|\theta)$$

In general, we need to consider how to compute

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

for fixed θ, which we need to compute the expectation step. This conditional, in the Bayesian settings scenario, is very easily computable using belief propagation.

Let us denote with $\underline{v} = (v_1, \dots, v_n)$ the set of observations of v, then our goal is to do maximum likelihood (using the expectation maximization algorithm) to compute the best θ for our distribution. Define

$$q^n(z) := p(z|v_n, \theta)$$

which is our conditional distribution for each possible observation of v. We can also extend it to all variables x, by means of the delta distribution

$$q^n(x) = p(z|v, \theta)\delta(v, v_n)$$

The e-step then just boils down to computing this $q^n(x)$. For the m-step we need the energy

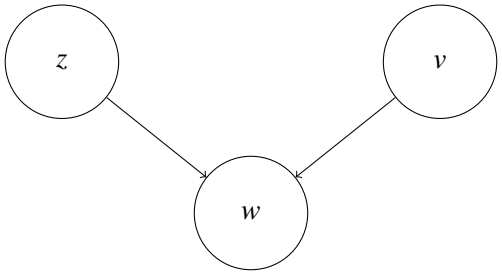
$$\sum_n \mathbb{E}_{q^n}[\log p(v_n, z_n|\theta)] = \sum_n \sum_i \mathbb{E}_{q^n}[\log p(x_i^n|pa(x_i^n)|\theta_i)]$$

Then we can optimize

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

over θ_i for each i.

Let's introduce an example to make it clearer. Consider this simple structure



and each of the variables is boolean. The probability distributions are defined by the parameters

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

Assume that we have observations $(v_1,w_1),\ldots,(v_n,w_n)$. The e-step for each observation just corresponds to find the distribution

$$q^n(z) = p(z|v=v_n, w=w_n, \theta)$$

or, as a function of x

$$q^n(x) = p(z|v=v_n, w=w_n, \theta)\delta(v,v_n)\delta(w,w_n)$$

Let's write the energy term for a couple of factors to get an idea of how to compute it and how to optimize with respect to it

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

If we maximize this with the constraint $\theta_z \in [0,1]$ we obtain

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

A second (slighly more complicated) example is the energy term with respect to w, i.e.

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

let's restric to the case $z=0, v=1 \quad (\theta_{w01})$

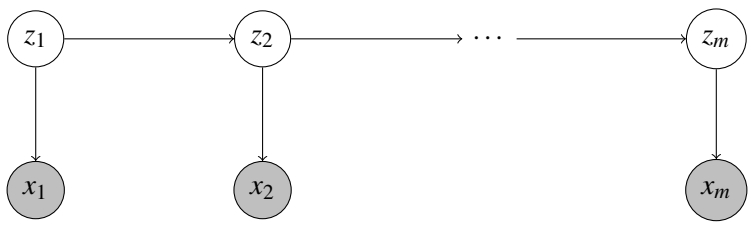
$$\sum_{n:w_n=1 v_n=1} q^n(z=0)\log\theta_{w01} + \sum_{n:w_n=0,v_n=1} q^n(z=0)\log(1-\theta_{w01})$$

Maximizing over this yields

$$\theta_{w01} = \frac{\sum_n \mathscr{F}(w_n=1)\mathscr{F}(v_n=1)q^n(z=0)}{\sum_n \mathscr{F}(w_n=1)\mathscr{F}(v_n=1)q^n(z=0) + \sum_n \mathscr{F}(w_n=0)\mathscr{F}(v_n=1)q^n(z=0)}$$

9.5 EM for Hidden Markov Models

We are going to address how to use the Expectation Maximization algorithm in the context of Hidden Markov Models, which, remember, are just a special case of Bayesian Networks. Let's consider the following HMM:



with each $z_i \in \{1,\dots,k\}$ being a categorical variable, we have the following probability distributions

$$\begin{gathered} p(z_1 = i)\pi_i \\ p(z_i = j|z_{i-1} = k) = A_{kj} \\ p(x_i|z_i = k) = p(x_i|\phi_k) \\ \theta = (\pi, A, \phi) \end{gathered}$$

The EM algorithm for Hidden Markov model is known as the **Baum-Welch algorithm**.

The observation set in this context corresponds to whole trajectories (all the x variables of the model are observed in each trajectory). We will write them as $\underline{x} = x^1,\dots,x^N$ where each $x^i = (x_1^i,\dots,x_M^i)$.

For the e-step, we need to compute

$$q^n(z) = p(z|x^n, \theta_{OLD}) \quad \forall n$$

and this is done just by running our message passing algorithm for Bayesian Networks. For the m-step instead, we want

$$E(\theta) = \sum_{n=1}^{N} \left[\sum_{k=1}^{K} q^n(z_{1k}) \ln \pi_k + \sum_{i=2}^{M} \sum_{j,k=1}^{K} q^n(z_{(i-1)}, z_{ik}) \ln A_{jk} + \sum_{i=1}^{M} \sum k = 1^K q^n(z_{ik}) \ln p(x_i^n|\phi_k) \right]$$

Then we just need to maximize over all the parameters belonging to theta, i.e. $\theta = (\pi, A, \phi)$.

Working on the calculations we get, for example for π_k

$$\pi_k = \frac{\Sigma_n q^n(z_{1k})}{\Sigma_i \Sigma_n q^n(z_{1i})}$$

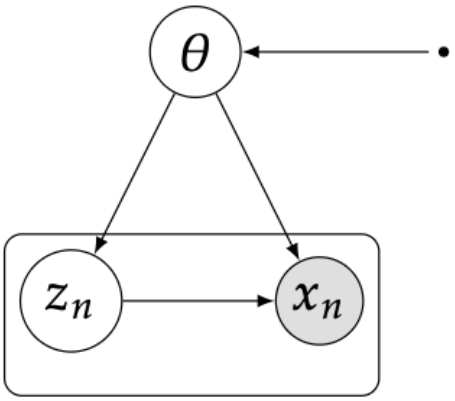
Similarly, we can obtain expressions for the other parameters.

10 Variational Inference

Variational Inference is a deterministic approximatino to perform inference. This differentiates it from other approximate inference techniques, such as Markov Chain Monte Carlo, as there is no sampling involved.

Since Variational Inference is a big topic, we are just going to introduce the basic ideas of the subject.

We start by considering a joint distribution $p(x,z)$, with x observable variables and z non observable (latent) variables. This scenario includes examples like the ones explored in the Expectation Maximization chapter [9], where latent variables are "coupled" with the observables even though we cannot observe them (we will call these **local latent variables** and denote them as z_n), but z this time can also be latent parameters of our distribution (**global latent variables** θ), thus encapsulating the possibility of doing inference on the parameters. Therefore z is represented as $z = (z_1, \dots, z_n, \theta)$ and our schema is represented by the following PGM:



We have observations $x_1, \dots, x_n = \underline{x}$, which will sometimes also be denoted simply as x, and we would like to compute the posterior distribution

$$p(z|\underline{x})$$

and also the model evidence

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

This was also the context in Expectation Maximization, where we were considering the ELBO, and we discussed that we can decompose the evidence as

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

where q is the so called variational distribution.

In EM, we were concerned about optimizing the ELBO by a two-step optimization over q and θ. Now θ are not explicitly present, but rather trated probabilistically and included in z. Hence, we only have the variational distribution and the goal in Variational Inference is to find the best q that approximates $p(z|\underline{x})$.

Notice this: $p(\underline{x})$ is fixed because observations are fixed, it's a number. The two terms in which we decompose it are the ELBO and the KL divergence, therefore:

- the q that maximizes the KL-divergence is the same as the one maximizing the ELBO $\mathscr{L}(q)$
- $\mathscr{L}$(q) is maximum when $KL[q||p] = 0$, i.e. $q = p(z|\underline{x})$

However, in most of the cases the computation of $p(z|\underline{x})$ is intractable.

The solution to this problem is to restrict q to a tractable family of distributions. Therefore the optimal $q(z)$ is likely to be such that $KL[q||p] > 0$. The goal of Variational Inference is then to maximize $\mathscr{L}(q)$ in a suitably restricted space of variational distributions q (we will call this space Q).

Let's make a first example to see how to choose this space. Think of Q as a set of parametric distributions, i.e. $Q = \{q(z|\lambda), \lambda \in \mathscr{R}^k\}$ (might be for example Gaussian distributions, with λ representing the average and the covariance matrix of our Gaussian). The optimizatino problem is then on λ, hence we need to find $\arg\max_{\lambda}(q(\lambda)) = \arg\max_{\lambda}\mathscr{L}(\lambda)$, but the caveat is that this would typically be a highly non-linear and non-convex optimization. Lastly, notice that λ would typically be compose of parameters for local latent variables and global latent variables, i.e. $q(z|\lambda) = q(\theta|\lambda_\theta)\prod_i q(z_i|\lambda_i, \theta)$.

Mean Field Variational Inference

Rather than choosing a parametric model for our variational distribution, we can instead opt for a different strategy.

Given that we are interested in approximating $p(z|\underline{x})$ by $q(z)$, we assume that z can be decomposed in M different blocks of variables $z = (z_1, \ldots, z_M)$ and we further assume the **independence of the blocks**, i.e. that

$$q(z) = \prod_{i=1}^{M} q_i(z_i)$$

Sometimes we will denote $q_i(z_i) = q_i$ for brevity.

This is known as the **mean field** assumption (the same name comes from physics and stochastic processes).

Let's start by writing the lower bound for the mean field approximation:

$$\begin{aligned}
\mathscr{L}(q) &= \mathbb{E}[\log p(\underline{x}, z) - \log q(z)] \\
&= \int \textstyle\prod_i q_i [\log p(\underline{x}, z) - \sum_i \log q_i]\, dz \\
&= \int q_j \left[\int log p(\underline{x}, z) \textstyle\prod_{i \neq j} q_i\, dz_i\right] dz_j - \int q_j \log q_j\, dz_j + \text{const} \\
&= \int q_j \mathbb{E}_{i \neq j}[\log p(\underline{x}, z)]\, dz_j - \int q_j \log q_j\, dz_j + \text{const}
\end{aligned}$$

where on the second row we factored our the terms depending on one factor q_j, hiding all the terms not depending on q_j in the *const* term.

Next we will define the function

$$\log \tilde{p}(\underline{x}, z_j) = \mathbb{E}_{i \neq j}[\log p(\underline{x}, z)] + \text{const}$$

Which means, wrapping up

$$\begin{aligned}
&\mathscr{L}(q_j(z_j)) = \mathbb{E}_{q_j}[\log \tilde{p}(\underline{x}, z) - \log q_j] + \text{const}, \\
&\text{hence } \mathscr{L}(q_j(z_j)) = -KL[q_j||\tilde{p}(\underline{x}, z_j)] + \text{const}
\end{aligned}$$

Now we have M lower bounds $\mathscr{L}(q_j)$, which have to be maximized for q_j, with $q_i, i \neq j$ fixed. Since the ELBO is maximized when $KL[q||p] = 0$, we know that our best approximation is indeed

$$q_j^*(z_j) = \tilde{p}(\underline{x}, z_j)$$

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

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

If we can compute analytically the expectation $\mathbb{E}_{i \neq j}[\log p(\underline{x}, z)]$ then we are in a scenario in which we can actually perform the mean field approximation. We cannot compute directly the expectaion because we do not know q_i. What we do then, is that we initialize q_i to some initial distribution, then cycle through q_j, optimizing the ELBO with respect to the coordinate j and fixing all the others $q_{\neg j}$. We repeat for all the coordinates in turn (**coordinate ascent**) until convergence, which is guaranteed since the bound is convex on q_j.

Therefore, given that we know how to compute these integrals over the logarithm of the joint distribution, mean field approximation gives us a relatively easy method to approximate our posterior distribution.

However, being able to compute the integral depends on the model, which means that this approximation is not suitable for every scenario.

Example: ***Gaussian distribution***

We will use mean field variational interence to go from a Gaussian to a factorized Gaussian. We have a joint distribution $p(z) = \mathscr{N}(z|\mu, \Lambda^{-1})$ where $\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$ and Λ is the precision matrix, so we are dealing with a 2-dimensional probability distribution. The mean field approximation implies that $q(z) = q(z_1)q(z_2)$. Therefore we need to perform the expectations.

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

This is a nice form, because, since $\log q_1^*(z_1)$ is a quadratic form, we know that $q_1^*(z_1)$ is Gaussian $\mathscr{N}(z_1|m_1, \Lambda_{11}^{-1})$ where $m_1 = \mu_1 - \Lambda_{11}^{-1}\Lambda_{12}(\mathbb{E}[z_2] - \mu_2)$.
By symmetry we have that

$$q_2^*(z_2) = \mathscr{N}(z_2|m_2, \Lambda_{22}^{-1}), \quad m_2 = \mu_2 - \Lambda_{22}^{-1}\Lambda_{21}(\mathbb{E}[z_1] - \mu_1)$$

In our case we can solve directly these equations by noticing that $\mathbb{E}[z_i] = \mu_i$ which means that $m_1 = \mu_1$ and $m_2 = \mu_2$.
Notice that these are different than the marginals of a Gaussian. The variance of each component is different than just the diagonal component of the precision matrix (when we compute the covariance in the marginalization process we need to invert the full matrix).

Variational Inference with direct and inverse KL

We may ask ourselves what would happen if instead of trying find the best variational distribution q such that

$$kl[q||p] \quad \text{is minimum}$$

we tried to solve the inverse proble, i.e. finding the q such that

$$KL[p||q] \quad \text{is minimum}$$

Looking at the example of the Gaussian that we introduced before we have the following qualitative results:

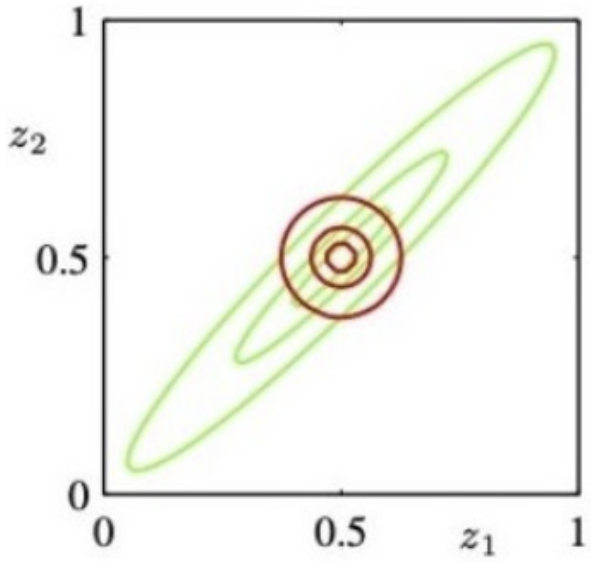


Figure 10.1: Original distribution (green) and its mean field approximation (Red)

Generally speaking, the inverse problem is intractable, as it requires to evaluate an expectation with respect to the unknown distribution p, but in our scenario, we can make it tractable by using the mean field approximation so that $q(z) = q(z_1)q(z_2)$.

In particular, the variational distribution found by minimizing $KL[p||q]$ is such that $q(z_i)$ is exactly the i_{th} marginal of the Gaussian distribution, hence it encompasses all the "original range" of our distribution (the border of the picture in red is the same of the picture in green).

This is a very common behavious of these two approximations. The approximation of the direct KL-divergence is known as the **direct KL** approximation, while the one of the inverse KL is known as the **zero forcing** approximation scheme. If $p(z) \approx 0$ then $q(z) \approx 0$ around the mode. That's because if you take a small p and a large q you get a large $KL[q||p]$.

The approximation of the inverse KL-divergence instead is known as the **zero avoiding** approximation scheme. Which means that if $q(z)$ is non zero then $p(z)$ is non zero, for the same (but inverse) reason as before. In multidimensional distributions the variational distribution will end up overlapping different modes of our system.

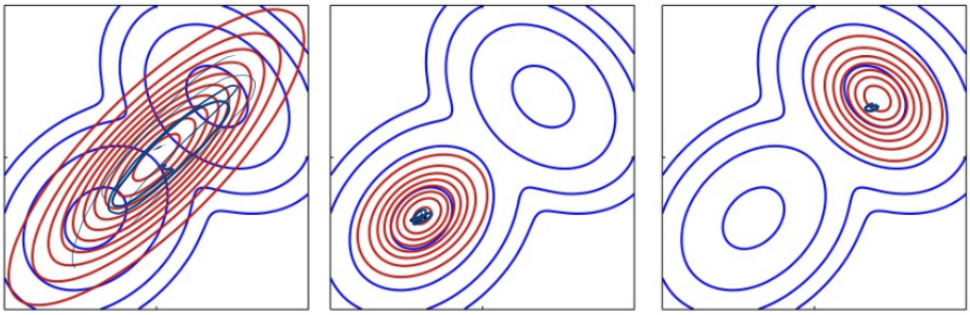
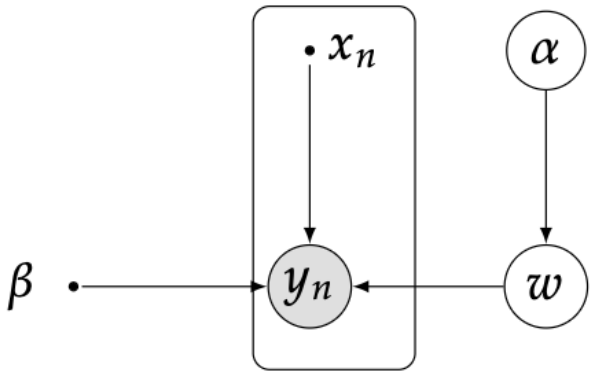


Figure 10.2: On the left, the inverse KL divergence, on the right, the direct KL divergence.

10.1 Variational Linear Regression

Mean Field variational inference can be used also to make approximate inference in the context of Linear regression. In particular, we will consider the case in which we would like to put a hyperprior over the parameter α, which regulates the variance of the prior for our weights.



Our joint distribution is

$$p(y,w,\alpha) = p(y|w)p(w|\alpha)p(\alpha)$$

with each term, since we are dealing with Linear regression, defined by

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

and our goal becomes to compute the posterior distributions for w and α using mean field variational inference, i.e. to compute $p(w,\alpha|y)$.

We will use the mean field variational distribution $q(w,\alpha) = q(w)q(\alpha)$.

Let's start with

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

Therefore what we have is, in fact, another Gamma distribution

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

We can also workout what happens for the other term of the variational distribution

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

Again, notice that here we have a quadratic form, hence completing the square gives us a Gaussian distribution

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

The solution for w is very similar to what we have in linear regression keeping α fixed, but now instead of just α we have its expectation in the equation for the covariance matrix.

Notice that we know what are these expectations:

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

Then we can just initialize the expectation for α and then we start iterating by computing the expectation of $w^T w$ and so on.

We can also in principle compute $\mathcal{L}(q) \approx p(y)$ that approximates the model evidence, which cna then be used for Bayesian model comparison.

10.2 Black box Variational Inference

As we have seen, the mean field approximation efficacy is dependent on our ability to compute the expectations that arise in the equations determining the components of our variational distribution. If we can't compute the expectations, we are not able to use the approximation at all. Here black-box (or stochastic) Variational Inference enters the picture as a viable alternative.

The general idea is to perform a **Monte-Carlo estimate** of the gradient of the ELBO with respect to the variational parameters and then perform gradient ascent with these estimates.

Our variational distribution will be typically parametric in this scenario $q(z(\lambda))$. The lower bound is

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

We want to compute $\nabla_\lambda \mathcal{L}(\lambda)$ which we cannot compute analytically. The idea, as previously stated, is to sample this gradient, but how can we do it? We need to turn the gradient of this estimation into the estimation of a gradient in order to exploit the capabilities of Monte Carlo methods.

We have two strategies, the first one works only for special cases, while the second one, albeit more complex, is usable in general.

Reparameterization trick

The first solution is called **"Reparameterization trick"**. It can be used if we can write $z = g_v(\varepsilon)$ as a certain function g of ε, with ε being some random variable coming from a distribution $\hat{q}(\varepsilon)$ which is independent of λ. We also define $\hat{\lambda} = (\lambda, v)$ where v are the parameters of the function g. Then we can write

$$\mathscr{L}(\hat{\lambda}) = \mathbb{E}_{\hat{q}(\varepsilon)}[\log p(x, g_v(\varepsilon)) - \log q(g_v(\varepsilon)|\lambda)]$$

and our expectation does not depend on λ anymore. Hence we can sampole from it and compute the gradient:

$$\nabla_{\hat{\lambda}} \hat{\geq} = \mathbb{E}_{\hat{q}(\varepsilon)}[\nabla_{\hat{\lambda}} \log p(x, g_v(\varepsilon)) - \nabla_{\hat{\lambda}} q(g_v(\varepsilon)|\varepsilon)]$$

Let's also define the following function for easier readability

$$G(\varepsilon) = [\nabla_{\hat{\lambda}} \log p(x, g_v(\varepsilon)) - \nabla_{\hat{\lambda}} q(g_v(\varepsilon)|\lambda)]$$

In practice, we sample $\varepsilon_j \sim \hat{q}(\varepsilon)$ and the sampled gradient is just

$$\nabla_{\hat{\lambda}} \mathscr{L}(\hat{\lambda}) = \frac{1}{S} \sum_{s=1}^{S} G(\varepsilon_s)$$

Then we use Stochastic Gradient Ascent (SGA), which is the ascending version of the same algorithm that we use in Neural Network backpropagation, and we just iterate these two steps, Monte Carlo approximation of the gradient and SGA up until convergence.

The caveat is being able to perform the reparameterization trick, which is easy for Gaussians, for example, and less easy for other distributions.

Example:

If we consider the parametric variational distribution to be a univariate Gaussian distribution, we can write it as:

$$q(z|\lambda) = \mathscr{N}(z|\mu, \sigma^2)$$

Now we would like to express $q(z|\lambda)$ by a distribution $\hat{q}(\varepsilon)$ independent from λ combined with a function $g_\lambda(\varepsilon)$ that depends from λ. In particular, we consider:

$$z = g_v(\varepsilon) = \mu + \sigma \cdot \varepsilon$$
$$\varepsilon \sim \hat{q}(\varepsilon) = \mathscr{N}(\varepsilon|0, 1)$$

Non-reparameterizable $q(z|\lambda)$

The goal is still to rewrite the gradient of the expectation as the expectation of the gradient; the general case is more complicated but an expression can be nonetheless found. Let us start from

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

Using the dominated convergence theorem to get the gradient inside the integral and then applying the product rule for derivatives we get

$$\nabla_\lambda \mathscr{L}(\lambda) = \int \nabla_\lambda [\log p(x,z) - \log q(z|\lambda)] q(z|\lambda)\, dz + \int \nabla_\lambda q(z|\lambda)[\log p(x,z) - \log q(z|\lambda)]\, dz$$

Taking a look at the first term, we see that $\nabla_\lambda [\log p(x,z)] = 0$ and we can write it as

$$
\begin{aligned}
\int \nabla_\lambda [\log p(x,z) - \log q(z|\lambda)] q(z|\lambda)\, dz &= -\mathbb{E}_q[\nabla_\lambda \log q(z|\lambda)] \\
&= -\mathbb{E}_q\left[\frac{\nabla_\lambda q(z|\lambda)}{q(z|\lambda)}\right] \\
&= \int \frac{\nabla_\lambda q(z|\lambda)}{q(z|\lambda)} q(z|\lambda)\, dz \\
&= \int \nabla_\lambda q(z|\lambda)\, dz \\
&= \nabla_\lambda \int q(z|\lambda)\, dz \\
&= \nabla_\lambda 1 \\
&= 0
\end{aligned}
$$

We still need to workout the second term. Again we use the fact that

$$
\nabla_\lambda [\log q(z|\lambda)] = \frac{\nabla_\lambda q(z|\lambda)}{q(z|\lambda)}
$$

which we rephrase as

$$
\nabla_\lambda q(z|\lambda) = \nabla_\lambda [\log q(z|\lambda)] q(z|\lambda)
$$

Recapping

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

Then we can sample $z_s \sim q(z|\lambda)$ and have an estimate of our gradient

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

Again we have an estimate of the gradient and we can use SGA to find convergence. Unfortunately this estimates of the gradient has a very high variance, which slows down the convergence of our algorithm, so in the next steps we would see strategies on how to control this variance.

Rao-Blackwellization

The first technique to control the variance of the stochastic estimation of the gradient of the ELBO that we have defined in black box variational inference, is known as **Rao-Blackwellization**.

Consider x, y random variates and some function $J(x,y)$ of which we want to compute the expectation.

First let's define

$$
\hat{J}(x) = \mathbb{E}_y[J(x,y)|x] \quad \text{hence } \mathbb{E}_x[\hat{J}(x)] = \mathbb{E}_{xy}[J(x,y)]
$$

Crucially, we know, by properties of the conditional expectation, that

$$
Var[\hat{J}(x)] = Var[J(x,y)] - \mathbb{E}[(J(x,y) - \hat{J}(x))^2] < Var[K(x,y)]
$$

Now, let's consider a mean field factorization of $q(z|\lambda) = \prod_{i=1}^{N} q(z_i|\lambda_i)$. Since every z_i depends only on λ_i we are going to consider the gradient with respect to the single λ_i and then exploit this factorization to simplify the expression.

We need to a couple more things:

- q_i: marginal of $q(z|\lambda)$ on the terms that form the Markov Blanket z_i in p
- $p_i(x, z_i)$ the product of factors of $p(x|z)$ depending on z_i

We are not going to perform the computation here (they are reported in the black-box variational inference paper), but we get that

$$\hat{\nabla_{\lambda_i}}[\mathscr{L}] := \mathbb{E}_{q(i)}[\nabla_{\lambda_i}\mathscr{L}(z_i)] = \mathbb{E}_{q(i)}[\nabla_{\lambda_i}\log q(z_i|\lambda_i)[\log p_i(x,z_i) - \log q(z_i|\lambda_i)]]$$

So essentially now we are taking an expectation over a *smaller* set of variables. This is playing the role of $\hat{f}(x)$ recasted on our variational inference problem, hence the variance of the estimation is reduced with respect to the original formulation of the problem.

More specifically, we need to consider samples $z_s \sim q_i(z|\lambda)$ and this distribution is just the product of the factors belonging to the Markiv blanket of z_i.

10.3 Control variates

Let's first introduce the general idea of control variates. Say we have a function f of which we want to know the expectation with respect to a certain distribution q. Instead of directly computating the expectation of f, we will define a new function $\hat{f}$ such that $\mathbb{E}_q[\hat{f}] = \mathbb{E}_q[f]$ and $VAR_q[\hat{f}] < VAR_q[f]$, and compite $\mathbb{E}_q[\hat{f}]$. How do we build such a $\hat{f}$?

We choose a function h such that $\mathbb{E}[h] < \infty$ and define

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

One can trivially see that indeed $\mathbb{E}_q[\hat{f}] = \mathbb{E}[f]$ and that

$$Var[\hat{f}] = Var[f] - 2aCov[f,h] + a^2 Var[h]$$

Additionally we have the freedom to choose a and we can fix it to the value a^* that minimizes the variance, which by deriving the expression before w.r.t. a and setting it to zero turns out to be

$$a^* = \frac{COV(f,h)}{VAR(h)}$$

So the larger the covariance, the larger the reduction in the estimation of the variance.

In our scenario, we start from Rao-Blackwellization

$$f_i(z) = \nabla_{\lambda_i}\log q(z_i|\lambda_i)[\log p_i(x,z_i) - \log(z_i|\lambda_i)]$$

and we will choose

$$h_i(z) = \nabla_{\lambda_i}\log q(z_i|\lambda_i)$$

since we have seen that $\mathbb{E}[h_i(z)] = 0$.

The optimal choice a^* for a is hard to compute, because we would need to know the covariance precisely, but we can estimate it as $\hat{a}^*$ reusing the same samples taht we used to estimate the gradient.

Then, our estimation of the gradient using control variates becomes

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

where $z_s \sim q_i(z|\lambda)$.

A scheme summarizing the different kinds and applications of Variational Inference is shown in Figure 10.3.

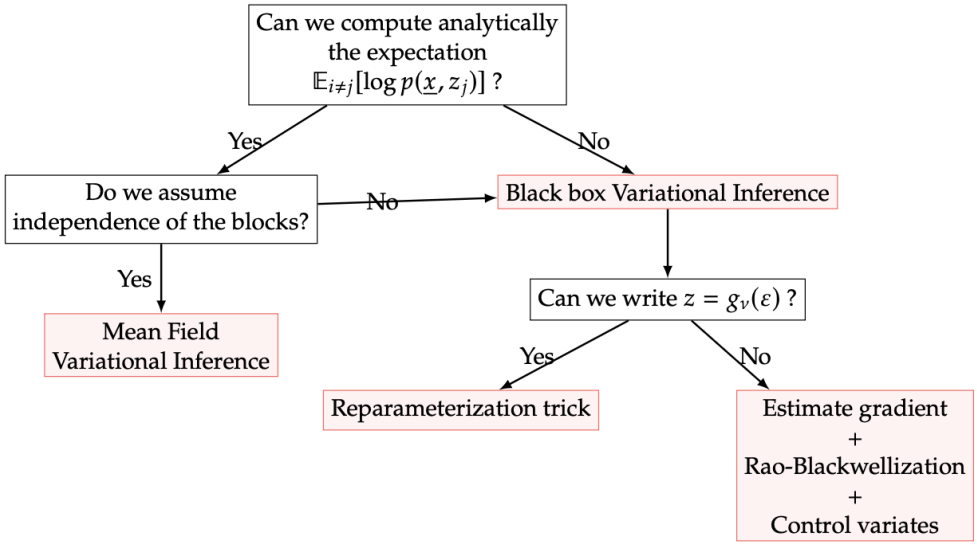


Figure 10.3: A schematic decision dia- gram to identify the suitable Variational Inference scheme to use in the applica- tion at hand.

10.4 Bayesian Neural Networks

Looking at the world of deep learning, we will now try to understand how the methods we have just described can prove useful when applied to deep neural networks. These architectures are powerful function approximators which can be trained using gradient-based optimization. Sometimes this flexibility, which is their main strength, can become a weakness, e.g. if it leads to overfitting or when the data available is limited. Moreover, deep neural networks lack the uncertainty estimation on the output.

To overcome these issues we can examine neural network under a Bayesian lens, which for its nature behaves in a probabilistic way. The model uncertainty should be introduced in the parameters w. Imagine to train many times the network on the same dataset with a stochastic optimization technique: the parameters would probably be different each time, as the predictors. This procedure, used in Neural Networks ensembles, would lead to a measure of uncertainty on the weights which reflects on the uncertainty on the output, though heavily dependent on the training mechanism and on the initialization of model parameters. However, there is a simpler and more grounded way to introduce uncertainty, i.e. switching from point-wise weights to probability distributions.

This means placing a prior distribution $p(w$ on weights and then learning the posterior distribution $p(w|\bar{x},\bar{y}))$ with a suitable learning algorithm, using it to compute the predictive distribution:

$$p(y|x) = \int_w p(y|x,w)p(w|\bar{x},\bar{y})\,dw$$

where W is the space of the possible parameters.

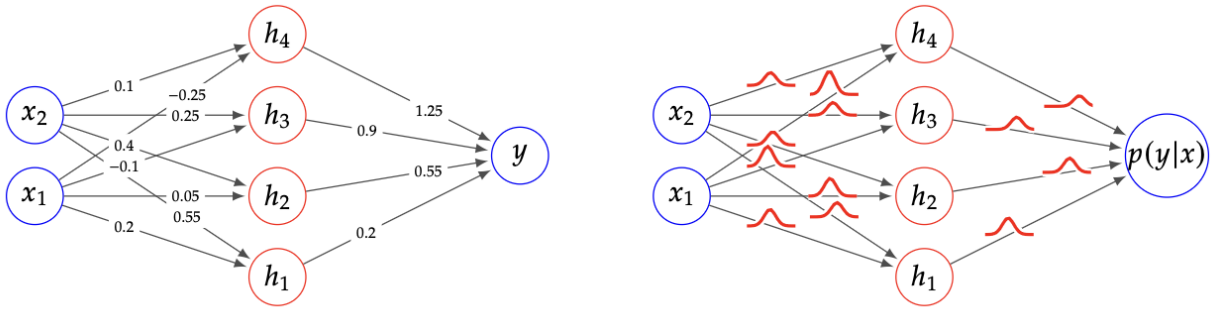


Figure 10.4: Neural network (left) VS Bayesian neural network (right). We consider tht the BNN outputs a probability distribution, which is obtained by computing the predictive distribution

Remark. Looking at the predictive distribution, we can consider it as a form of Bayesian model averaging: the first term in the integral, i.e. the likelihood, corresponds to the forward pass through a neural network with a specific set of weights, multiplied by the posterior probability of that set of weights over all possible weight values. This is equivalent to using an ensemble of an uncountably infinite number of neural networks.

Remark. The Bayesian approach allows us to regularize the learning by placing a suitable prior on the weights w: with a Gaussian prior we obtain a L2 regularization; with a Laplace prior a L1 regulatization.

Computing directly $p(w|\bar{x},\bar{y})$, and thus the predictive distribution, is intractable as it requires learning a very high-dimensional distribution (the number of weights in a neural network is typically very large). So, we consider a variational distribution $q(w|\theta)$ which approximates $p(w|\bar{x},\bar{y})$. At this point, we write the ELBO:

$$\begin{aligned}\mathscr{L}(\theta) &= \mathbb{E}_{q(w|\theta)}[\log p(w,\bar{x},\bar{y}) - \log q(w|\theta)] \\ &= \mathbb{E}_{q(w|\theta)}[\log p(\bar{y}|w,\bar{x}) + \log p(w) - q(w|\theta)]\end{aligned}$$

and use its opposite as the loss function of the neural network:

$$Loss(\theta) = \mathbb{E}_{q(w|theta)}[\log q(w|\theta) - \log p(\bar{y}|w,\bar{x}) - \log p(w)]$$

Bayes by Backprop

In order to optimize the loss function above, we need to compute the gradient with respect to the parameters θ. This is usually not tractable as we cannot interchange the gradient with the expectation, as they both act on the same parameters. So we use the reparameterization trick. In this way, we can perform the optimization by combining sampling with backpropagation, the milestone of deep learning. This algorithm is called Bayes by Backprop.

We start by assuming that the variational posterior distribution is a Gaussian distribution with diagonal covariance matrix (but in principle we could use any reparameterizable distribution). We notice that we need to require σ to be non-negative and so we parametrize it as:

$$\sigma = \log(1+\exp(\rho))$$

In this case,

$$\theta = (\mu,\rho)$$

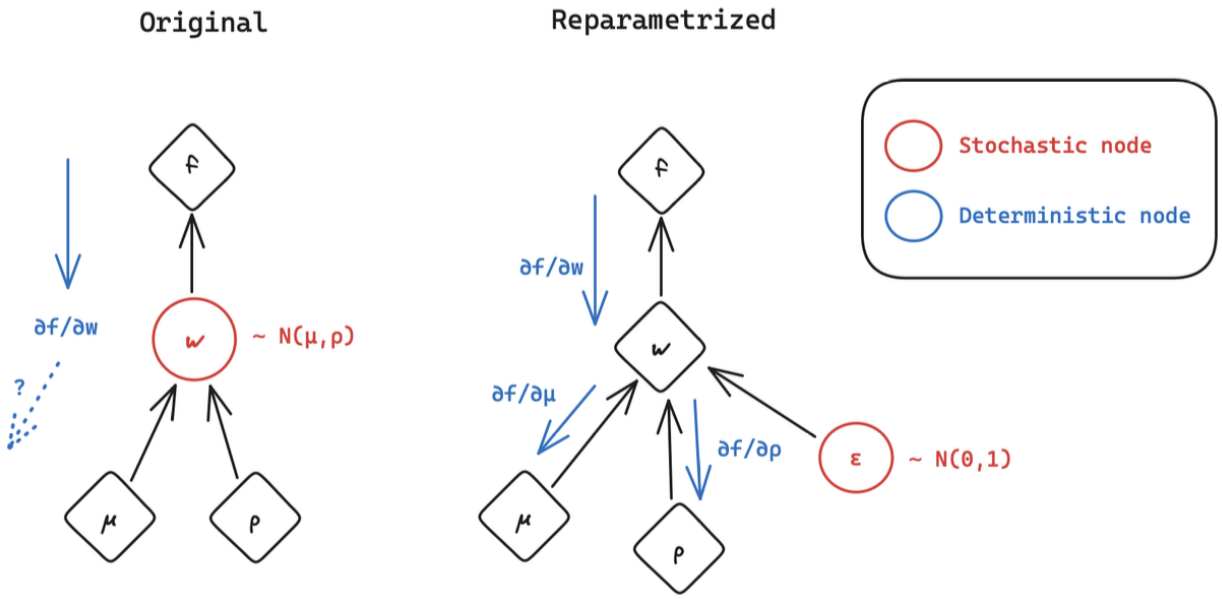
and we can reparametrize w as:

$$w = \mu + \log(1+\exp(\rho)) \cdot \varepsilon$$

with $\varepsilon \sim \mathcal{N}(0, I)$.

In this way, we are now able to sample w by sampling from a simple distribution which does not depend on the parameters that we are optimizing. So, after taking the expectation, we are interested in minimizing

$$\mathbb{E}_{\varepsilon}[f(w,\theta)] = \mathbb{E}_{\varepsilon}[\log p(\bar{y}|\bar{x},w) + \log p(w) - \log q(w|\theta)]$$



We can then perform the optimization step by step by repeating the following procedure:

1. Sample $\varepsilon \sim \mathcal{N}(0, I)$
2. Compute the weights as $w = \mu + \log(1 + \exp(\rho)) \cdot \varepsilon$
3. Execute the forward pass
4. Compute the unbiased Monte Carlo gradients with respect to the parameters and calculate the update steps (backpropagation):

$$\begin{aligned} \Delta_{\mu} &= \nabla_w f(w,\theta) \cdot \nabla_{\mu} w + \nabla_{\mu}(w,\theta) \\ &= \nabla_w f(w,\theta) + \nabla_{\mu}(w,\theta) \\ \Delta_{\rho} &= \nabla_w f(w,\theta) \cdot \nabla_{\rho} w + \nabla_{\rho} f(w,\theta) \\ &= \nabla_w f(w,\theta) \frac{\varepsilon}{1+\exp(-\rho)} + \nabla_{\rho} f(w,\theta) \end{aligned}$$

5. Update the variational parameters:

$$\begin{aligned} \mu &\leftarrow \mu + \alpha \Delta_{\mu} \\ \rho &\leftarrow \rho - \alpha \Delta_{\sigma} \end{aligned}$$

   where α is the learning rate.

Remark. Notice that $\nabla_w f(w,\theta)$, which is present in both Δ_{μ} and Δ_{ρ} is the usual gradient found by backpropagation: Bayes by Backprop just scales and shifts it.

Remark. As usual in deep learning, minibatches are used in the training process. In this case, the KL cost has to be re-weighted in a proper way.

11 Generative Modelling

11.1 Variational Autoencoders

Variational Autoencoders (VAEs) are one of the most commonly used and efficient approaches for the task of generative modelling. Unlike discriminative models, whose objective is to learn the conditional distribution of the data $\underline{x} = x_1, \ldots, x_n$, i.e. estimate $p(x)$ with the parametri distribution $p(x, \theta)$, typically differentiable with respect to θ.

In this way, it becomes possible to sample from such a probability distribution, thus generating new instances similar to the training dataset, or to assign to a given instance the probability of having been generated by the same process as the training dataset.

In general, generative modelling deals with a harder task with respect to discriminative models because in the former there is much more knowledge to learn. Intuitively, this is valid also for humans: by seeing many labelled images of dogs and cats it is much easier to distinguish dogs from cats than to draw a new image of a dog and a cat. In fact, in order to assign a label it is only required to find out some patterns which are different between the categories, while to generate a new instance it is necessary to capture correlations in the dataset, as for example the fact that dogs have two eyes, a tail, etc.. So, the discriminative model has to learn a decision boundary in the data space, while the generative one has to understand how data is planced throughout the space.

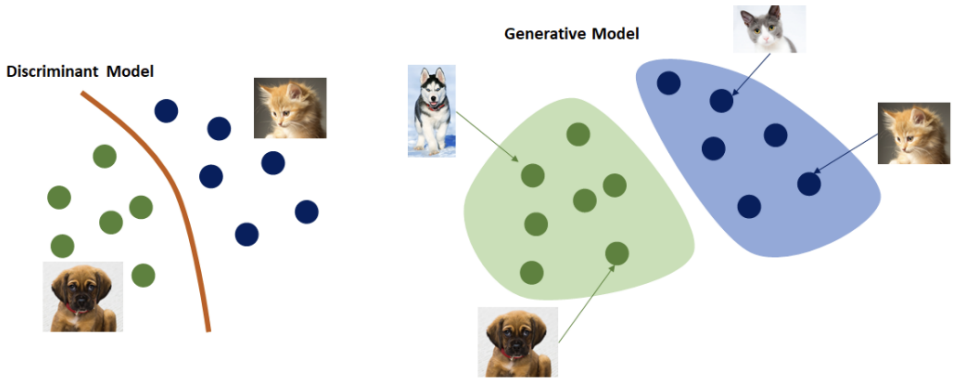


Figure 11.1: Image from *https://medium.com/@jordi299/about-generative-and-discriminative-models-d8958b67ad32*

It may be more convenient to avoid learning directly the distribution of the data (pixels in our example), but to introduce latent features z which can describe the data and rewrite our distribution as

$$p(x, z|\theta) = p(x|z, \theta)p(z)$$

We assume that the posterior distribution $p(z|x)$ is intractable and that we are working in the big data regime, so that we are forced to consider mini-batches during the training phase. The goal that we would like to achieve are:

- Learning the parameters θ

- Approximating the posterior $p(z|x)$ in order to understand which are the latent features extracted from a given instance x
- Performing approximate inference on x to be able to fill holes in an instance

In this scenario, where we have a parametric distribution with latent variables, it might occur to use to use Expectation Maximization. But we must remember that the E-step requires being able to evaluate the conditional distribution $p(z|x,\theta_{OLD})$ which we have assumed here to be intractable. Other techniques that we have already seen, such as mean field variational approximation or sampling-based methods, turn out to be too computationally expensive as they do not scale well with large datasets. We will now see how to solve the problem by applying stochastic variational inference together with an encoder-decoder approach.

Autoencoding Variatinoal Bayes (AEVB)

Following the idea of black-box variational inference, we start by considering a parametric variational distribution for the latent variables $q(z|\phi)$. We would like to optimize the ELBO jointly on θ and ϕ by stochastic gradient ascent and so we need to compute $\nabla_{\theta,\phi}\mathscr{L}(\phi,\theta)$. We fix the prior distribution of z as a standard Gaussian:

$$p(z) \sim \mathscr{N}(0,I)$$

and we start by rewriting the lower bound as:

$$\begin{aligned} \prec,\subseteq \quad &= \mathbb{E}_{q(z|x,\phi)}[\log p(x,z|\theta) - \log q(z|x,\phi)] \\ &= \mathbb{E}_{q(z|x,\phi)}[\log p(x|z,\theta) + \log p(z) - \log q(z|x,\phi)] \\ &= \mathbb{E}_{q(z|x,\phi)}[\log p(x|z,\theta)] - \mathbb{E}_{q(z|x,\theta)}\left[\log \tfrac{q(z|x,\phi)}{p(z)}\right] \\ &= \mathbb{E}_{q(z|x,\phi)}[\log p(x|z,\theta)] - KL[q(z|x,\phi)||p(z)] \end{aligned}$$

As we said before, we are not able to work with the entire dataset and thus we have to consider a mini-batch $x_1,\ldots,x_m$ and approximate:

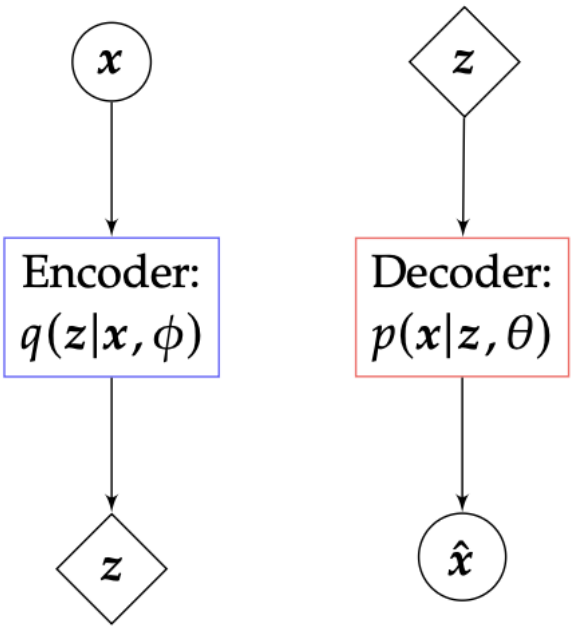
$$\mathscr{L}(\phi,\theta) \approx \frac{1}{m}\sum_{j=1}^{m} \mathbb{E}_{q(z|x_j,\phi)}[\log p(x_j|z,\theta)] - KL[q(z|x_j,\phi)||p(z)]$$

We can try to understand the meaning of this expression:

- The first term is the reconstruction error. It considers how well we are reconstructing x given z sampled from q. This is maximized when $p(x|z,\theta)$ assigns the highest probability to the original instance x. We call $p(x|z,\theta)$ **decoder**.
- The second term is a regularization term which encourages the variational distribution to look like a Gaussian distribution and not some kind of identity mapping. This avoids learning a mapping from inputs to latent features that just ”stores” the inputs in different regions of the latent space, hence it favours generalization.

Both the terms involve the expectation with respect to the variational distribution of the latent variables that we interpret as features describing the data. So, we call $q(z|x,\phi)$ **encoder**.

We can represent the architecture in the following way:



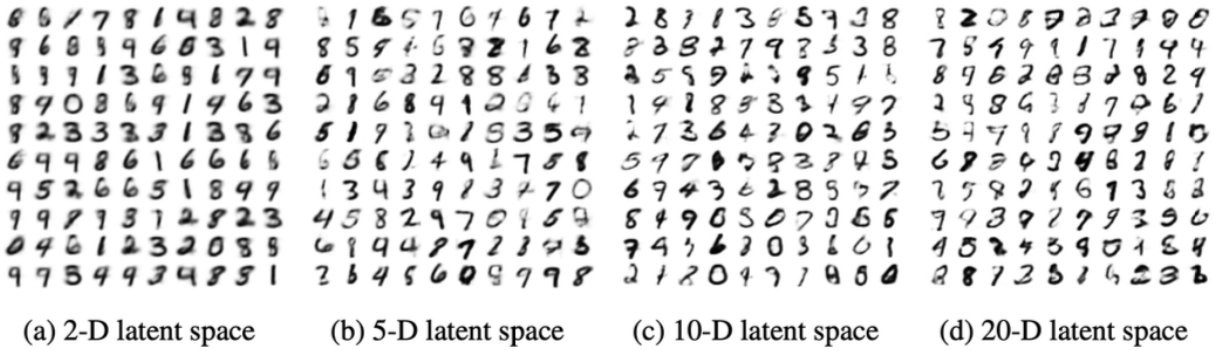
where the input x is mapped through the encoder into a useful latent space z from which we can reconstruct $\hat{x}$ via the decoder. We would like $\hat{x}$ to be as similar as possible to x.

Remark. Notice that the variational parameters are shared across all the datapoints: using global parameters allows us to "amortize" the cost of inference (**amortized inference**). Instead, in mean-field variational inference we had different parameters for each datapoint. This changes the behaviour of the model when we have a new datapoint: in mean-field VI we need to maximize the ELBO for each new point while here we can keep the global parameters fixed or run the variational inference again (maybe when many points are added).

So far we have obtained an expression for the ELBO, which is the objective function of our optimization. In order to maximize it, we need a good estimate of the gradient and we cannot obtain it directly from the previous expression because the gradient has to be computed on the same variables involved in the expectation. Therefore, it is not possible to swap the gradient and the expectation and this is where the **reparametrization trick** comes in.

We express the variational distribution $q(z|x,\phi)$ through a deterministic transformation $g(\varepsilon,x,\phi)$ which maps a noise variable ε (distributed accordingly to a distribution we can easily sample from, as $\varepsilon \sim \mathcal{N}(0,I)$) to the distribution of z.

$$z = g(\varepsilon,x,\phi)$$



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 11.2: Random samples from learned generative models (AEVB) of MNIST for different dimensionalities of latent space

Typically, the variational distribution is chosen to be a Gaussian:

$$q(z|x,\phi) = \mathcal{N}(\mu_z(x,\phi), \sigma_z^2(x,\phi) \cdot I)$$

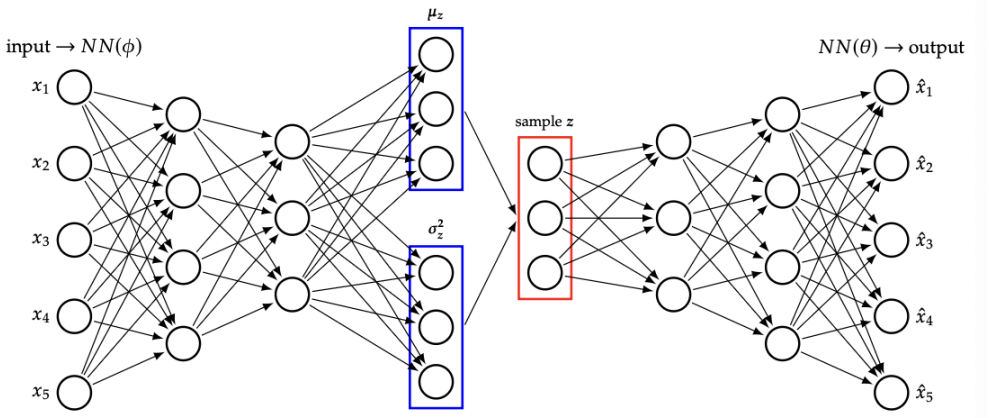
so that we have an analytical expression for $KL[q(z|x,\phi)||p(z)]$ and we can rewrite z as

$$z = \mu_z(x,\phi) + \sigma_z^2(x,\phi) \cdot \varepsilon$$

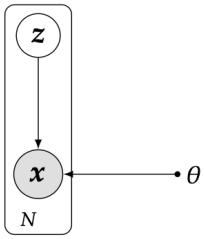
At this point, we can evaluate the gradient of the ELBO as

$$\nabla_{\theta,\phi}\mathscr{L}(\phi,\theta) = \mathbb{E}_\varepsilon[\nabla_{\phi,\theta} \log p(x|g(\varepsilon,x,\phi),\theta)] - \nabla_\phi KL[q(z|x,\phi)||p(z)]$$

We still have to choose the functions $\mu_z(x,\phi)$ and $\sigma_z^2(x,\phi)$. To have high expressiveness and efficient optimization, we can opt for neural networks: we just built a **variational autoencoder**.



Remark. The variational autoencoder can be interpreted as a directed probabilistic graphical model with latent variables.



11.2 Diffusion Models

Diffusion models are a family of probabilistic generative models that progressively destruct data by injecting noise, then learn to reverse this process for sample generation. In particular, we will discuss denoising diffusion probabilistic models (DDPMs), but many concepts are common to other formulations.

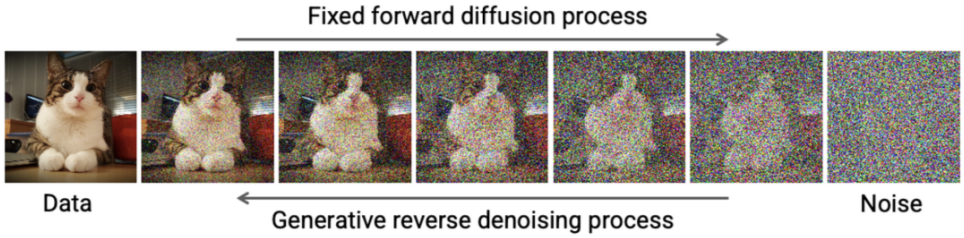
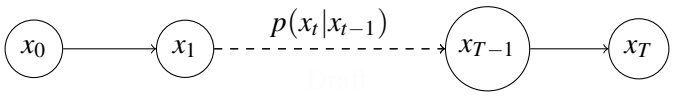


Figure 11.3: Denoising Diffusion-based Generative Modeling: Foundations and Applications

Forward Diffusino Process

Given a data point $x_0 \in \mathscr{X}$ sampled from the data distribution $p(x_0)$, consider a Markov chain $p(x_t|x_{t-1})$ with $t \in 1,\dots,T$ satisfying the following properties:

- It is easy to sample from $p(x_t|x_{t-1})$
- For T large enough, $p(x_T)$ is approximately a known distribution from which it is easy to sample and that does not depend on x_0, i.e., $p(x_T) \approx p(x_T|x_0)$. This distribution is referred as the **prior**.



Such Markov chain is referred as the **forward diffusion process**, and $p(x_t|x_{t-1})$ is often referred as the **forward transition kernel**. Using the chain rule of probability and the Markov property, we can factorize the joint distribution of $x_0,\dots,x_T$ into:

$$p(x_0,\dots,x_T) = p(x_0)\prod_{t=1}^{T} p(x_t|x_{t-1})$$

In practice, the most used forward process in continuous space is based on the injection of Gaussian noise:

$$p(x_t|x_{t-1}) := \mathscr{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

where $\beta_t \in (0,1)$ regulates the amount of noise that is injected at each step (the larger, the faster the convergence to the prior distribution). A useful property of the above process is that we can sample at any arbitrary time step in a closed form using the reparameterization trick. Given $\alpha_t := 1-\beta_t, \bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i$, and $\varepsilon_t \sim \mathscr{N}(0,I)$ we have

$$\begin{aligned} x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\varepsilon_{t-1} \\ &= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\varepsilon_{t-2}) + \sqrt{1-\alpha_t}\varepsilon_{t-1} \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\hat{\varepsilon}_{t-2} \\ &= \dots \\ &= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\hat{\varepsilon} \end{aligned}$$

where ε_{t-2} and ε_{t-1} have been merged into $\hat{\varepsilon}_{t-2}$.

Hence, we have

$$p(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, 1-\bar{\alpha}_t)$$

This is also useful to show, as $\bar{\alpha}_t \to 0$, that

$$\lim_{t\to\infty} p(x_t|x_0) = \mathcal{N}(x_t; 0, I)$$

So for a sufficiently large T, $x_T \sim \mathcal{N}(0,I)$. In other words, this process ends up in corrupting the data into white noise.

Backward Process

Now that we are able to gradually corrupt data points x_0 into noise, for generating new data samples we have to revert this process. We have seen that we can easily sample from $p(x_T)$, but sampling from

$$p(x_0,\dots,x_T) = p(x_T)\prod_{t=1}^{T} p(x_{t-1}|x_t)$$

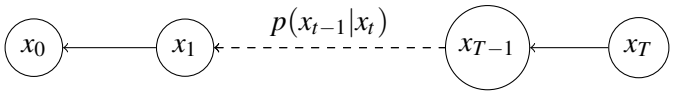
is non-trivial, since we do not have access to the reverse transition kernels $p(x_{t-1}|x_t)$* that would allow us to perform ancestral sampling as in the forward process. Therefore, the reverse transition kernels $q_\theta(x_{t-1}|x_t)$, from which it is easy to sample and that can be easily computed, for example:

$$q_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t,t), \Sigma_\theta(x_t,t))$$

where the mean and covariance functions are neural networks.

The objective is then to find parameters θ such that

$$q_\theta(x_0,\dots,x_T) := p(x_T)\prod_{t=1}^{T} q_\theta(x_{t-1}|x_t) \approx p(x_0,\dots,x_T)$$



The parametric kernels are learned by minimizing the Kullback-Leibler divergence between the forward and backward processes:

$$\begin{aligned} D_{KL}(p(x_0,\dots,x_T)||q_\theta(x_0,\dots,x_T)) &= \mathbb{E}_{p(x_0,\dots,x_T)}\left[\log \tfrac{p(x_0,\dots,x_T)}{q_\theta(x_0,\dots,x_T)}\right] \\ &= -\mathbb{E}_{p(x_0,\dots,x_T)}[\log q_\theta(x_0,\dots,x_T)] + \text{const} \\ &= -\mathbb{E}_{p(x_0,\dots,x_T)}\left[\textstyle\sum_{t=0}^{T} \log q_\theta(x_{t-1}|x_t)\right] + \text{const} \end{aligned}$$

Remark. The forward process $p(x_0,\dots,x_T)$ was considered constant with respect to the parameters θ. Although parameters of the forward process (as β_t) are usually considered fixed (hyperparameters), it is also possible to learn them. In that case we cannot consider the forward process as constant with respect to the parameters.

Minimizing this inverse KL divergence is equivalent to solving a maximum likelihood estimation problem: we can sample full trajectories starting from sample from the dataset and use them to fit parametric functions q_θ by stochastic optimization, as in a supervised learning problem.

Remark. It can be shown that, if forward transition kernels corrupts the data slowly enough (T is large), then the reverse transition kernels will be approximately Gaussian. This is what makes the approximation

of reverse transition kernels possible and approachable as a prediction problem. Conversely, if forward transition kernels add too much noise, the reverse transition probabilities can be multimodal, hence a parametric approximation would be intractable.

Denoising parametrization

There is empirical evidence that learning directly the backward transition kernels $q_\theta(x_{t-1}|x_t)$ is not the most effective strategy. Alternatively, it is possible to write the backward transition kernel in the following way:

$$p(x_{t-1}|x_t) = \int_{x_0 \in \mathscr{X}} p(x_{t-1}|x_0)\, dx_0 = \int_{x_0 \in \mathscr{X}} p(x_{t-1}|x_t, x_0) p(x_0|x_t)\, dx_0$$

This means that we can sample x_{t-1} if we can sample $x_0 \sim p(x_0|x_t)$ and plug it into $p(x_{t-1}|x_t, x_0)$. It is possible to show that for the Gaussian diffusion process introduced above the distribution $p(x_{t-1}|x_t, x_0)$ is also a Gaussian and can be computed analytically:

$$\begin{aligned} p(x_{t-1}|x_t, x_0) &= \mathscr{N}(x_{t-1}; \mu_t(x_t, x_0), \sigma_t I) \\ \mu_t(x_t, x_0) &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 \\ \sigma_t &= \frac{(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \beta_t \end{aligned}$$

Instead $p(x_0|x_t)$ (the denoising kernel) has to be learned, again by fitting a parametric distribution $q_\theta(x_0|x_t)$. Interestingly, this works despite $p(x_0|x_t)$ being highly multimodal and non-Gaussian, so only a "mean" approximation is possible.

However, it is common in DDPMs to avoid predicting directly x_0 given x_t. In fact, a model is trained to predict the noise that was added to x_0 in order to obtain x_t. Recall the property following from $p(x_t|x_0) = \mathscr{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, 1 - \bar{\alpha}_t)$:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon_t \text{ with } \varepsilon_t \sim \mathscr{N}(0, I)$$

We can rewrite the mean of $p(x_{t-1}|x_t, x_0)$, or $p(x_{t-1}|x_t, \varepsilon_t)$, as

$$\mu_t(x_t, \varepsilon_t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_t \right)$$

Now we can fit a parametric model $\varepsilon_\theta(x_t, t)$ to approximate $p(\varepsilon_t|x_t)$. This is usually done in a simplified way by minimizing the squared error:

$$\begin{aligned} \mathscr{L}(\theta) &= \mathbb{E}_{,,}[||\varepsilon_t - \varepsilon_\theta(x_t, t)||^2] \\ &= \mathbb{E}_{t, x_0, \varepsilon_t}\left[||\varepsilon_t - \varepsilon_\theta\left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon_t, t\right)||^2\right] \end{aligned}$$

where the time step t is sampled uniformly in $\{1, \ldots, T\}$, x_0 is randomly sampled from the dataset, and $\varepsilon_t \sim \mathscr{N}(0, I)$. The training and sampling algorithm are then summarized as follows:

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 11.4: Training and sampling algorithm for DDPMs

Remark. The empirical advantage of predicting the noise ε_t instead of x_0 could be due to the fact that it is easier to train a neural network when the target output is (marginally) distributed according to a normal distribution.

Summary

A DDPM makes use of two Markov chains: a forward chain that perturbs data to noise, and a reverse chain that converts noise back to data. The forward Markov chain is designed with the goal to gradually transform any data distribution into a simple prior distribution (e.g., standard Gaussian). The backward Markov chain instead reverses the former starting from the known distribution and gradually converging to the data distribution. Since the backward Markov chain is unknown, it is learned by a deep neural network using examples generated with the forward process. New data points are subsequently generated by first sampling a random vector from the prior distribution, followed by ancestral sampling through the reverse Markov chain.

Diffusion in discrete space

Despite bein originally thought for data in continuous space, it is also possible to define a diffusion process in discrete space, even though its usefulness is questionable and subject of current research. Assuming without loss of generality data $x \in \{1,2,\dots,c\}^d$, there are two main kinds of discrete diffusion processes:

- **Uniform**: At each time step t, every component x_t^i of x_t switches to a random value with a given small probability ε_t. The prior distribution of such diffusion process is the uniform over $\{1,2,\dots,c\}^d$.
- **Absorbing**: At each time step t, every component x_t^i of x_t switches to a partiular values called the "mask", often referred as the [MASK] token. The prior distribution of such diffusion process is the single point $[MASK]^d$, that is the absorbing state of the Markov chain. A variant of the absorbing diffusion process is when at each time step t the t^{th} component x_t^t is masked. Modelling the corresponding reverse process is then equivalent to fitting an autoregressive model.

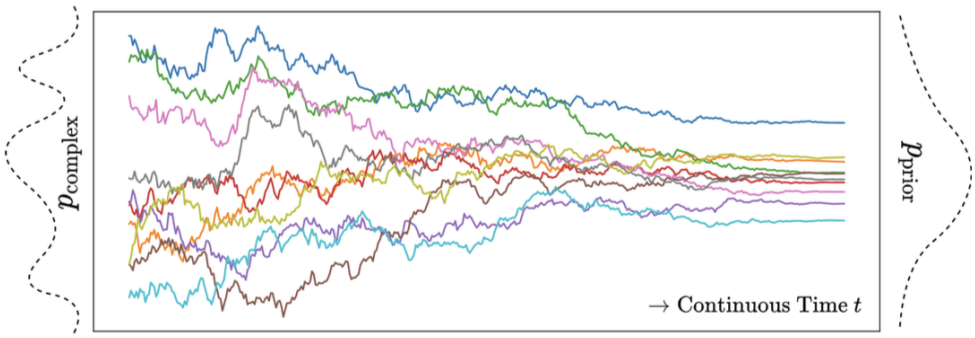
Score-based diffusion models

Given the objective to sample from the distribution $p(x)$ that generated the data score-based models aim at estimating the so called *score* of $p(x)$, defined as $\nabla_x \log p(x)$ and then use sampling techniques that exploit the knowledge of the score of the distribution.

An example of score-based model is diffusion with stochstic differential equations (SDE), a generalization in continuous time of the Markov chain diffusion process discussed above, where the transition kernel $p(x_{t+\varepsilon}|x_t)$ is implicitly defined through the following SDE:

$$dx = f(x,t)\,dt + g(t)\,dw$$

where $dw \sim \mathcal{N}(0, I\,dt)$ is a Wiener process.



Generation of new data points reduces to solving the inverse-time SDE, that it can be proven to be

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]\, dt + g(t)\, dw$$

FOr this we need to estimate the time dependent score $\nabla_x \log p_t(x)$, for example using a neural network $s_\theta(x,t)$. The most popular method is *denoising score matching*, where the following loss is minimized:

$$\mathbb{E}_{t\sim u(0,1), x_0 \sim p(x_0), x_t \sim p(x_t|x_0)} ||s_\theta(x_t,t) - \nabla_{x_t} \ln p(x_t|x_0)||$$

that is deeply connected with the denoising loss of DDPMs.

12 Exact Inference in Probabilistic Graphical Models

In the context of inference, out task is, given a joint distribution $p(X) = p(x_1, \ldots, x_n)$ to compute marginals or conditionals of the distribution.

We formalize this by considering two disjoint subsets of r.v.: $y \subseteq x, z \subseteq x$ s.t. $y \cap z = \emptyset$ and $y \cup z \subseteq x$.

Given that we observe y, we want to compute the conditional

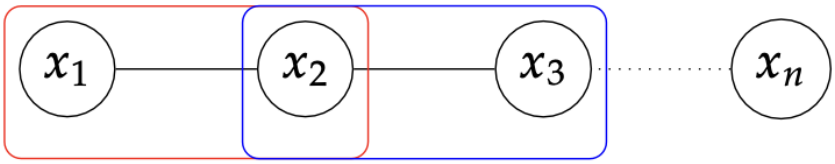
$$\begin{aligned} p(z|y=\bar{y}) &= \sum_x p(z,x'|y=\bar{y}) discreter.v. \\ &= p(z|y=\bar{y} = \int_x p(z,x'|y=\bar{y})) continuousr.v. \end{aligned}$$

being x' s.t. $x' \cup y \cup z = x$.

Remark: this summation can be of the order of $O(\mathscr{K}^m)(|x'| = m)$, hence obtaining this marginalization is computationally challenging!

Our goal is to carry out this computation efficiently, by leverage the factorization implied by the PGM.

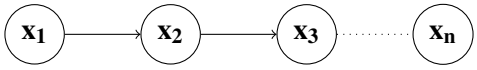
As an example, consider a Markov Random Field where variables are connected in a chain (colored rectangles represent the cliques):



The factorization implied by this PGM is:

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

Consider now the equivalent model in case of a Bayesian Network:



The factorization in this case reads as

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_2) \ldots p(x_n|x_{n-1})$$

Observation: ***Markov processes***

Chain structures like the previous are common because they repre- sent temporal series (more specifically, they are Markov processes).

Suppose that, given the model above, we want to compute the marginal

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

with $1 < k < n$. In principle this has complexity $O(\mathcal{K}^{n-1})$.

However, plugging the factorization implied by the Markov Random Field, we get(using the distributive laws of sum and product):

$$\begin{aligned} p(x_k) &= \sum_{x\_k} \frac{1}{Z} \psi_{1,2}(x_1,x_2) \dots \psi_{n-1,n}(x_{n-1},x_n) \\ &= \frac{1}{Z} \sum_{x_{k-1}} \psi_{k-1,k}(x_{k-1},x_k) \dots \left[\sum_{x_2} \psi_{2,3}(x_2,x_3) \left[\sum_{x_1} \psi(x_1,x_2)\right]\right] + \\ &+ \sum_{x_{k+1}} \psi_{k,k+1}(x_k,x_{k+1}) \dots \left[\sum_{x_{n-1}} \psi_{n-2,n-1}(x_{n-2},x_{n-1}) \left[\sum_{x_n} \psi_{n-1,n}(x_{n-1,x_n})\right]\right] \end{aligned}$$

Note that complexity is now $O(n \cdot k)$, i.e. linear in n.

We can define a dynamic programming algorithm, called **message-passing algorithm**, in which the information from the graph is summarized by local edge information.

We break the chain in two parts, the past and the future w.r.t. x_k, and we define the following:

- forward message: $\mu_\alpha(x_k) = \sum_{x_{k-1}} \psi_{k-1,k}(x_{k-1},x_k) \cdot \mu_\alpha(x_{k-1})$, with base case $\mu_\alpha(x_1) = 1$
- backward message: $\mu_\beta(x_k) = \sum_{x_{k+1}} \psi_{k,k-1}(x_k,x_{k+1}) \cdot \mu_\beta(x_{k+1})$, with base case $\mu_\beta(x_n) = 1$

In this algorithm, each node sends a message to its neighbors, and the messages are computed by the above formulas.

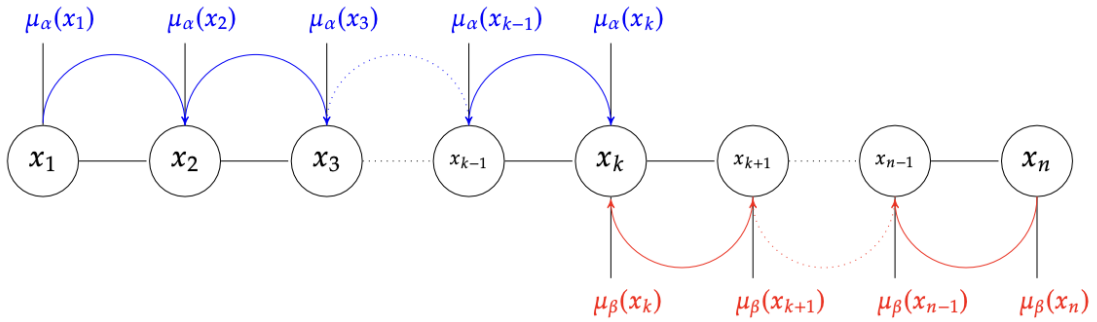


Figure 12.1: Message-passing algorithm

Once we have both $\mu_\alpha(x_k)$ and $\mu_\beta(x_k)$, we can observe that:

$$p(x_k) = \frac{1}{Z_k} \cdot \mu_\alpha(x_k)\mu_\beta(x_k) \propto \mu_\alpha(x_k)\mu_\beta(x_k)$$

with the normalization constant Z_k computed as $Z_k = \sum_{x_k} \mu_\alpha(x_k)\mu_\beta(x_k)$.

Summarizing, the idea behind this algorithm is to start from the extremes of the chain and pass messages (forward and backward) until the other end is reached. Once there, messages are combined to obtain an (un)normalized marginal distribution.

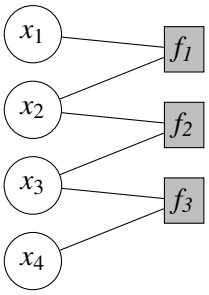
12.1 Factor Graphs

Our goal in what follows is to extend what we have done for chains to more general graph structures, i.e. trees and politrees. In order to do so, we need to introduce the formalism of **Factor Graphs**.

The idea behind them is to expose in a more clear way what are the factors and what are the variables. For this reason, Factor Graphs are bipartite graphs (i.e. nodes are divided in two classes), in which we have:

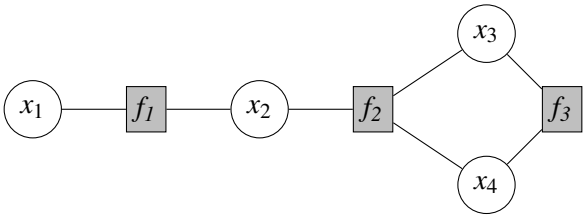
Variable nodes	**Factor nodes**
x	f

The canonical way of represent bipartite graphs is the following:



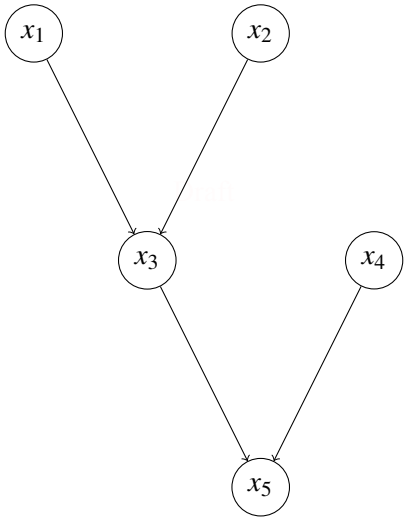
That is, all variable nodes are put on the left and all factor nodes on the right.

A more convenient, equivalent, representation could be:



12.1.1 From Bayesian Networks to Factor Graphs

Consider the following Bayesian Network:



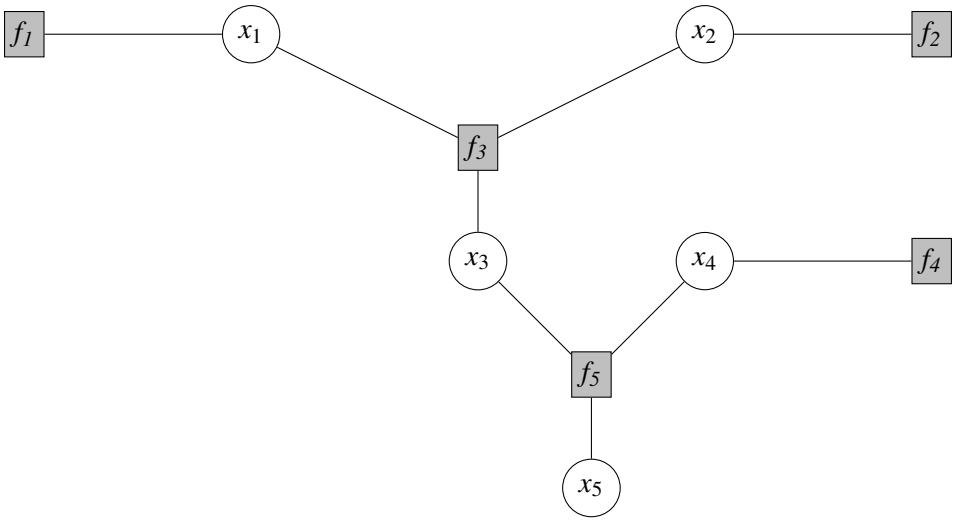
Which corresponds to the factorization:

$$\begin{aligned} p(x) &= p(x_1)p(x_2)p(x_3|x_1,x_2)p(x_4)p(x_5|x_3,x_4) \\ &= f_1(x_1)f_2(x_2)f_3(x_1,x_2,x_3)f_4(x_4)f_5(x_3,x_4,x_5) \end{aligned}$$

Fundamentally, the idea is to have a factor node (connected to the proper variable nodes) for each term of the factorization implied by the Bayesian Network.

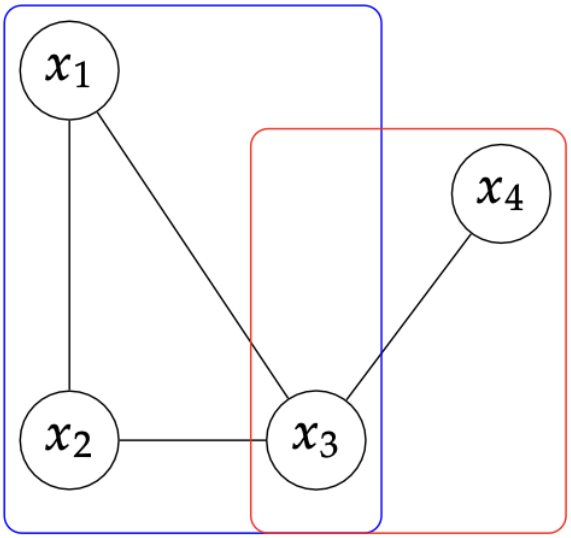
Formally: $\forall p(x_j|pa_j) \rightarrow f_j \curvearrowright x_j \cup pa_j$.

Hence the Factor Graph equivalent to the Bayesian Network above is:



12.1.2 From Markov Random Fields to Factor Graphs

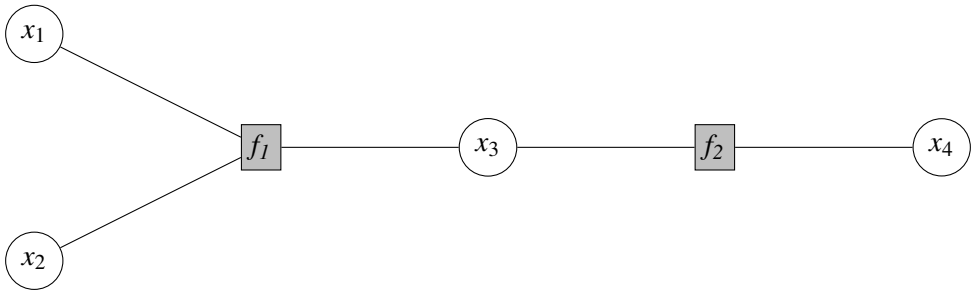
Consider the following Markov Random Field (rectangles correspond to the cliques):



Which implies the factorization:

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

This leads to the following Factor Graph:

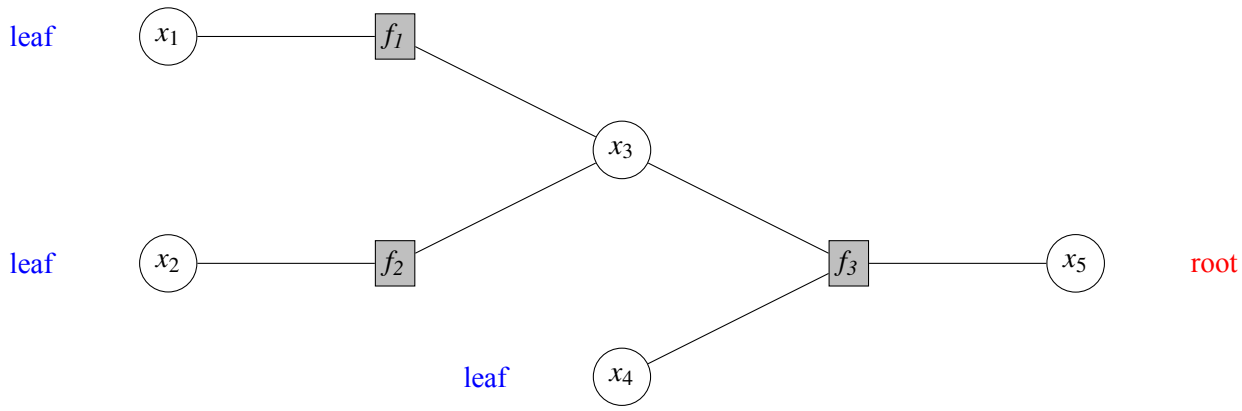


Hence, the conversion from Markov Random Fields to Factor Graphs works as: $\forall c \in \mathscr{C} \rightarrow f_c(x_c) \sim \psi_c(x_c)$ (equality does not hold because of the normalization constant).

So, whether we are starting from a Bayesian Network or from a Markov Random Field, we can convert our PGM into a Factor Graph and perform message passing on it, without loss of generality.

12.2 Sum Product Algotithm

Our goal is now to do inference in Factor Graphs, generalizing to more general graph structures what we did with chains previously. Consider the following Factor Graph:



The joint probabbility distribution implied by thie Factor Graph is:

$$p(x) = f_1(x_1,x_3)f_2(x_2,x_3)f_3(x_3,x_4,x_5)$$

Since the above graph is actually a tree, we can identify the **root** node (choice is arbitrary, here it is x_5) and consequently the **leaves** (in the example, x_1,x_2,x_4). Recall that there is a unique path from the root to any of the leaves and that following all those paths we visit all the nodes in the tree.

The message passing algorithm that we are going to detail is called **Belief Propagation** and works in Factor Graphs which are trees (i.e., without any loop).

Assume that the root is the node of which we want to compute the marginal (this is not necessarily the case, indeed after performing forward and backward pass we have sufficient information to compute the marginal w.r.t. each node in graph).

So, let's consider the marginal probability w.r.t. x_5:

$$\begin{aligned} p(x_5) &= \sum_{x_1,x_2,x_3,x_4} p(x_1,x_2,x_3,x_4,x_5) \\ &= \sum_{x_3,x_4} f_3(x_3,x_4,x_5) \left[\sum_{x_1} f_1(x_1,x_3)\right] \left[\sum_{x_2} f_2(x_2,x_3)\right] \end{aligned}$$

Take $\sum_{x_1} f_1(x_1,x_3)$: we can see this as a message going from factor f_1 to variable x_3. We denote it by $\mu_{f_1 \to x_3}(x_3)$.

Analogously, $\sum_{x_2} f_2(x_2,x_3) \doteq \sum_{x_1} f_1(x_1,x_3) \cdot \sum_{x_2} f_2(x_2,x_3)$.

Finally, the full summation can be seen as a message going from factor f_3 to variable x_5, i.e. $\mu_{f_3 \to x_5}(x_5) \doteq \sum_{x_3,x_4} f_3(x_3,x_4,x_5)[\sum_{x_1} f_1(x_1,x_3)][\sum_{x_2} f_2(x_2,x_3)]$.

We can observe the following:

- messages from factors to variables include a summation (an integral in the case of continuous r.v.) over all the variables on which the factor depends, except for the one we are sending the message to;
- messages from variables to fators collect via multiplication all the incoming messages from the other factors, except from the one we are sending the message to.

Formalizing, we call:

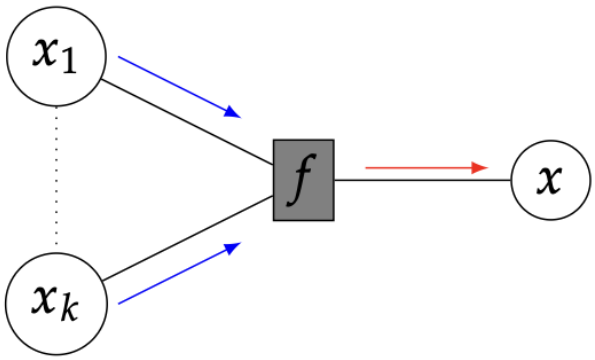
- set of neighboring nodes of variable x:

$$\mathcal{N}(x) = \{f_1, \ldots, f_k | f_i \text{ is connected to } x\}$$

- set of neighboring nodes of factor f:

$$\mathcal{N}(f) = \{x_1, \ldots, x_k | x_i \text{ is connected to } f\}$$

The scenario is the following:

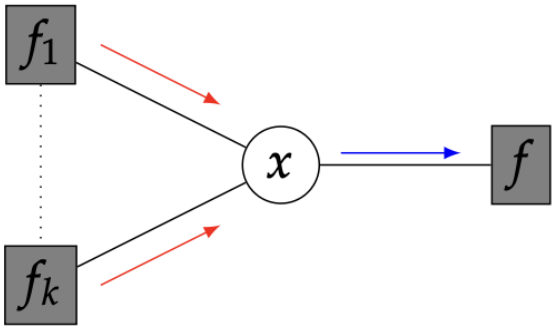


The red arrow in the figure above corresponds to the message $\mu_{f\to x}(x)$. Following what we have seen before, this is computed as:

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

($\mu_{x_i\to f}(x_i)$ are the blue arrows in the figure above).

Consider now the symmetric situation:



In this case:

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

Base cases are defined as:

- x leaf node: $\mu_{x\to f}(x) = 1$
- f leaf node: $\mu_{f\to x}(x) = f(x)$

The **Sum-product** algorithm works in two steps:

- forward pass: messages are sent from the leaves to the root;

- backward pass: messages are sent from the root back to the leaves.

In this way each edge will be traversed both by a forward and a backward message.

After these two passages, all possible messages are computed and we can obtain marginals and conditionals.

Consider the problem of computing the marginals, there are two cases in this scenario:

- computation of the marginal of a variable node x:

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

- computation of the marginal of a factor node $f(\bar{x})$ (i.e., $p(\bar{x})$ where $\bar{x} = \mathcal{N}(f)$):

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

Note that both these computations can be carried out once we have all the messages.

Moreover, the sum-product algorithm works also when the joint distribution is not normalized, in that case the obtained marginals have to be normalized.

Consider now the problem of computing conditionals on $y = \hat{y}, y \subseteq \{x_1, \ldots, x_n\}$, i.e. $p(x|y = \hat{y})$.

The first step is **clamping** y **to** $\hat{y}$, that is $p(x, x_1, \ldots, x_k, y = \hat{y})$ (i.e. we fix $y = \hat{y}$ in the joint).

Then we run the sum-product algorithm with y_j fixed to $\hat{y} \forall y_j \in y$ (practically, when running the message passing algorithm, whenever we have a variable in y we consider its corresponding state $\hat{y}$, instead of summing over it):

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

Finally, we compute the desired conditional as:

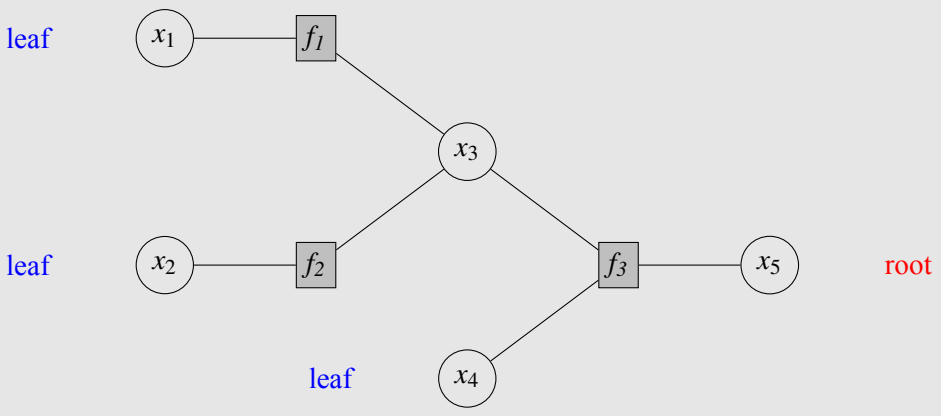
$$p(x|y = \hat{y}) = \frac{p(x, y = \hat{y})}{p(\hat{y})}$$

with $p(\hat{y}) = \sum_x p(x, y = \hat{y})$.

In this context, we can see $p(x, y = \hat{y})$ as an unnormalized conditional and $p(\hat{y})$ as its normalization constant.

Example:

consider the following factor graph:



that represents the unnormalized joint distribution

$$p(x_1,x_2,x_3,x_4,x_5) \propto f_1(x_1,x_3)f_2(x_2,x_3)f_3(x_3,x_4,x_5)$$

Variables are binary, i.e. $x_i \in \{0,1\}$ and factors are defined as:

$$f_1(x_1,x_3) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \text{ where } (x_1,x_3) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

$$f_2(x_2,x_3) = \begin{bmatrix} 0.1 \\ 0.4 \\ 0.2 \\ 0.3 \end{bmatrix} \text{ where } (x_2,x_3) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

$$f_3(x_3,x_4,x_5) = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix} \text{ where } (x_3,x_4,x_5) = \begin{matrix} (0,0,0) \\ (0,0,1) \\ (0,1,0) \\ (0,1,1) \\ (1,0,0) \\ (1,0,1) \\ (1,1,0) \\ (1,1,1) \end{matrix}$$

After arbitrarily choosing x_5 as root node, we can start by computing forward message edge by edge:

Forward pass

$$\begin{aligned}
\mu_{x_1 \to f_1}(x_1) &= 1 \\
\mu_{x_2 \to f_2}(x_2) &= 1 \\
\mu_{f_1 \to x_3}(x_3) &= \textstyle\sum_{x_1} f_1(x_1,x_3)\mu_{x_1 \to f_1}(x_1) = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix} \\
\mu_{f_2 \to x_3}(x_3) &= \textstyle\sum_{x_2} f_2(x_2,x_3)\mu_{x_2 \to f_2}(x_2) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \\
\mu_{x_3 \to f_3}(x_3) &= \mu_{f_1 \to x_3}(x_3)\mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix} \\
\mu_{x_4 \to f_3}(x_4) &= 1 \\
\mu_{f_3 \to x_5}(x_5) &= \textstyle\sum_{x_3,x_4} f_3(x_3,x_4,x_5)\mu_{x_3 \to f_3}(x_3)\mu_{x_4 \to f_3}(x_4) = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix}
\end{aligned}$$

Backward pass

$$\begin{aligned}
\mu_{x_5 \to f_3}(x_5) &= 1 \\
\mu_{f_3 \to x_4} &= \textstyle\sum_{x_3,x_5} f_3(x_3,x_4,x_5)\mu_{x_5 \to f_3}(x_5)\mu_{x_3 \to f_3}(x_3) = \begin{bmatrix} 0.054 \\ 0.276 \end{bmatrix} \\
\mu_{f_3 \to x_3} &= \textstyle\sum_{x_4,x_5} f_3(x_3,x_4,x_5)\mu_{x_5 \to f_3}(x_5)\mu_{x_4 \to f_3}(x_4) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \\
\mu_{x_3 \to f_1} &= \mu_{f_3 \to x_3}(x_3)\mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.09 \\ 0.49 \end{bmatrix} \\
\mu_{x_3 \to f_2}(x_3) &= \mu_{f_3 \to x_3}(x_3)\mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix} \\
\mu_{f_1 \to x_1}(x_1) &= \textstyle\sum_{x_3} f_1(x_1,x_3)\mu_{x_3 \to f_1}(x_3) = \begin{bmatrix} 0.125 \\ 0.205 \end{bmatrix} \\
\mu_{f_2 \to x_2}(x_2) &= \textstyle\sum_{x_3} f_2(x_2,x_3)\mu_{x_3 \to f_2}(x_3) = \begin{bmatrix} 0.222 \\ 0.108 \end{bmatrix}
\end{aligned}$$

Having computed all the messages, we can calculate the marginal for all the nodes, for example:

$$p(x_5) \propto \mu_{f_3 \to x_5}(x_5) = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix}$$

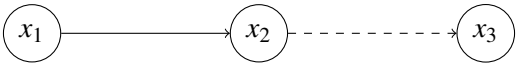
Notice that all marginals were normalized using the same constant $Z = 0.33$.

12.3 Max Plus algorithm

In what follows our task will be, given a joint density $p(x)$, to find the tuple $x^M = \arg\max_x p(x)$ (i.e. find a setting of the variables that has the largest probability and the value of that probability).

To do so, we will use the **max-plus algorithm**.

As an example, consider a chain structure described by a Markov Random Field:



whose factorization reads as

$$p(x) = \frac{1}{Z}\Psi_{1,2}(x_1,x_2)\dots\Psi_{n-1,n}(x_{n-1},x_n)$$

Our goal is to maximize $p(x)$ w.r.t. x_n.

It is possible to distribute the factorization so that only local computations are required:

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

This happens because of the distributive properties of the max, indeed it holds that, given $a > 0$:

- the max distributes over the product

$$\max(ab,ac) = a\cdot\max(b,c)$$

- the max distributes over the sum

$$\max(a+b,a+c) = a+\max(b,c)$$

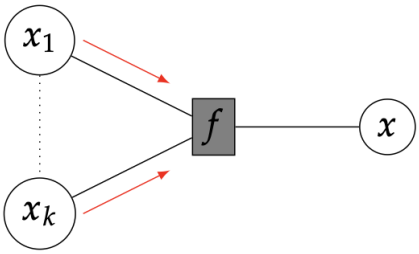
This allows us to take an approach similar to the one used in the sum-product algorithm.

Actually we will maximize $\log p(x)$. hence turning the product into sum of logarithms (and this justifies the name max-plus), i.e. our objective is to maximize

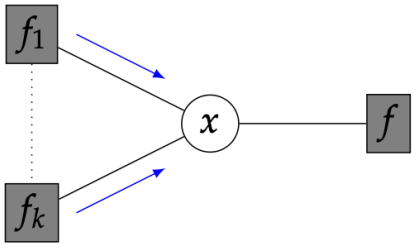
$$\log p(x) = \sum_i \log\Psi_{i,i+1}(x_i,x_{i+1}) - \log Z$$

This saves us from product of factors that are possibly very small, since they are probabilities.

Max-plus algorithm is very similar to sum-product: essentially max replaces sum and plus replaces product. This leads to the following scenarios:



$$\mu_{f\to x}(x) = \max_{x_1,\dots,x_k}\left[\log f(x,x_1,\dots,x_k) + \sum_{x_i\in\mathcal{N}(y)\setminus x}\mu_{x_i\to f}(x_i)\right]$$



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

With base cases:

- x leaf node: $\mu_{x\to f}(x) = 0$
- f leaf node: $\mu_{f\to x}(x) = \log f(x)$

In order to find the maximum of the joint distribution, that is

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

we just need to run the forward pass of the algorithm (i.e. we propagate messages from the leaves up to the root, as in the sum-product algorithm).

Remark. The result will be the same irrespective of which node is chosen as the root.

However, we want to find also the configuration of the variables for which the maximum is achieved. This can be done by applying a slight variation in the forward pass, meaning that we also need to keep track of which values of the variables gave rise to the maximum state of each variable. So during the forward pass we will also store:

$$\Phi_{f\to x}(x) = \operatorname*{arg\,max}_{x_1,\dots,x_k\in\mathcal{N}(f)\setminus x}\left[\log f(x,x_1,\dots,x_k) + \sum_{x_i\in\mathcal{N}(f)\setminus x}\mu_{x_i\to f}(x_i)\right]$$

Hence, since at the end of the forward pass we know the most probable value of the root node

$$x_{root}^{max} = \operatorname*{arg\,max}_{x_{root}}\left[\sum_{f\in\mathcal{N}(x_{root})}\mu_{f\to x_{root}}(x_{root})\right]$$

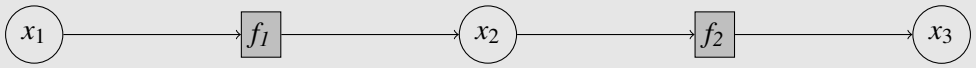
we can exploit the additional information that we stored to retreive the most probable state of the internal nodes by **back-tracking**, i.e. by following Φ. For example, assuming the variable nodes connected to f are $x_1,\dots,x_k,x_{root}$, in the first backtracking step we will fix the value of $x_1,\dots,x_k$ according to

$$\Phi_{f\to x_{root}}(x_{root}^{max}) = \left\{\begin{array}{c} x_1 \to x_1^{max} \\ \dots \\ x_K \to x_k^{max} \end{array}\right\}$$

and then propagate backwards following Φ from the other factor nodes connected to $x_1,\dots,x_k$.

Example:

Consider the following chain:



in which variables are binary, i.e. $x \equiv x_1, x_2, x_3 \in \{0,1\}$ and factors are defined as:

$$f_1(x_1,x_2) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \text{ where } (x_1,x_2) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

$$f_2(x_2,x_3) = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.2 \\ 0.2 \end{bmatrix} \text{ where } (x_2,x_3) = \begin{matrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{matrix}$$

We start by computing the forward messages edge by edge:

$$\mu_{x_1 \to f_1}(x_1) = 0$$

$$\mu_{f_1 \to x_2}(x_2) = \max_{x_1} [\log f_1(x_1,x_2) + \mu_{x_1 \to f_1}(x_1)] = \begin{bmatrix} \log 0.3 \\ \log 0.4 \end{bmatrix} \text{ where } x_2 = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$\mu_{x_2 \to f_2}(x_2) = \mu_{f_1 \to x_2}(x_2)$$

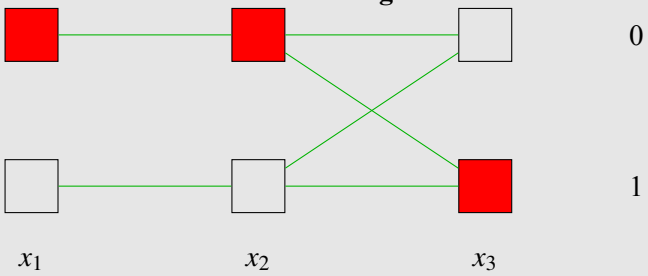
$$\mu_{f_2 \to x_3}(x_3) = \max_{x_2} [\log f_2(x_2,x_3) + \mu_{x_2 \to f_2}(x_2)] = \begin{bmatrix} \log 0.2 + \log 0.4 \\ \log 0.5 + \log 0.3 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

as well as backtracking functions:

$$\Phi_{f_1 \to x_2}(x_2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ where } x_2 = \begin{matrix} 0 \\ 1 \end{matrix}$$

$$\Phi_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

At this point, we can write down the **lattice/trellis diagram**:



This kind of diagram shows explicitly the K (2 in this case) possible states (one per row of the diagram) for each of the variables x_i in the model. The two paths through the lattice correspond to configurations that give the global maximum of the joint probability distribution.
If, for example, we get that:

$$\max_{x_3} \mu_{f_2 \to x_3}(x_3) = \log 0.5 + \log 0.3$$
$$\arg\max_{x_3} = 1$$

then we can obtain the tuple that maximizes our joint density by following the path backward in the diagram above (we are guaranteed that this path is unique), starting from $x_3 = 1$, i.e. $(0,0,1)$ (red path in the figure).
Note: in the context of Hidden Markov Models, the max-plus algorithm is called *Viterbi algorithm*.

12.4 Inference in general Probabilistic Graphical Models

In what follows, we want to extend what we have seen so far to general probabilistic graphical models, meaning Factor Graphs which contain loops.

In these cases, we cannot identify the root and the leaves, hence we don't have well defined forward and backward directions.

There are several possibilities:

- **Junction Tree algorithm**: it roughly builds a tree over the cliques of the Factor Graph, then exact inference is done in this tree. The worst case complexity of this algorithm is exponential in the size of the largest clique (so possibly very heavy computationally);
- **Loopy Belief Propagation**: forward and backward pass of the sum-product algorithm are iterated several times, until a fix point is reached. Unfortunately there is no guarantee that the algorithm will converge. When it does, it provides an approximate answer to the inference problem;
- **Monte Carlo Sampling**: a general strategy for approximate inference based on sampling;
- **Variational Inference**: it approximates the posterior distribution with a simpler distribution belonging to a pre-specified (parametric) class, which is the closer one to the true posterior, minimizing the KL-divergence.

Bibliography

[1] Christopher M Bishop et al. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.
[2] Michael J Kearns et al. *An introduction to computational learning theory*. MIT press, 1994.