Data Science and Artificial Intelligence

Probabilistic Machine Learning

Lecture Notes

Luca Bortolussi

April 26, 2024

University of Trieste

Disclaimer
This book contains course notes for Probabilsitic Machine Learning course. Despite the efforts, it is likely that there are errors, imprecisions, unclear explanations. Plase report any of these to lbortolussi@units.it. You are also kindly requested not to distribute this document.

Acknowledgements
Thanks to Gaia Saveri, Federico Brand, Gugliemo Padula, Romina Doz, and Davide Scassola for helping writing, fixing, and proofreading these notes.

Uncertainty is the only certainty there is.
– John Allen Paulos

All models are wrong, but some are useful.

– George P. Box

Palla lunga. E pedalar.

– Nereo Rocco

Preface

These notes contain a short, but complete, presentation description of the content of the Probabilistic Machine Learning (PML) course held at the university of Trieste. This course is an introduction to probabilistic modelling in machine learning, with a focus on approximate inference techniques. The journey starts from graphical models and proceeds via Bayesian learning, Monte Carlo sampling, expectation maximization and variational inference, ending with an introduction to generative modelling techniques which are the backbone of the current successes of generative AI.

The notes are a continuous work in progress, and may not contain some side material - presented in notebooks - which are to be considered an appendix to these notes. Future versions will include more material, reflecting the rapid changes of the scientific landscape in this area, and possibly including some of the material of the notebooks.

In case something is not clear, or if you wish to expand your understanding of a certain topic (these notes are **not** complete or exhaustive of any topic) there are some reference books which will be helpful to read - and which contain much more material. The main reference book is the one of Bishop [1]. Most of the material in the lectures is heavily inspired by this book. Other good generalist reference are those of Murphy [2] and Friedman et al [3]. As for Bayesian learning, the book of Barber is very exhaustive [4], while a good reference for sampling methods is the book of Gelman [5].

Luca Bortolussi

Contents

Introduction 1

This course notes introduce ideas and instruments of machine learning form a probabilistic perspective. This approach is growing in importance and is the foundation of the main successes of generative Artificial Intelligence. After a brief discussion on the importance of probability and mathematics, and how to frame machine learning at a high level from this perspective, the structure of the notes will be introduced.

1.1 Models and Probability

What is Machine Learning all about?

According to Wikipedia, machine learning (ML) explores the study and construction of algorithms that can learn from and make predictions on data. 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.

But what is a model?

A model is a hypothesis that certain features of a system of interest are well replicated in another, simpler system. There many possible kinds of models, for instance mental models, verbal models, animal models. We are interested in mathematical ones.

A **mathematical model** is a model where the simpler system consists of a set of mathematical relations between objects (equations, inequalities, etc). We are mostly concerned with **stochastic models**, i.e. a mathematical models where the objects are probability distributions. This is a reasonable and effective choice, as probability is the most effective mathematical theory on the market to deal with uncertainty.

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 is captured. Machine learning deals with algorithms for automatic selection of one of such models from observations of the system.

In essence, probabilistic machine learning is an umbrella under which we can find several approaches to build probabilistic models of aspects of the world, and several strategies to learn these models from data.

Incorporating probability in the modelling process has the main effect of shifting the focus from point estimates to probability distributions, allowing us to capture uncertainty in the modelling process. It furthermore allows us to account in a more principled way extrinsic noise and intrinsic fluctuations in the data, possibly caused by unknown factors. Additionally, probabilistic modelling is naturally amenable of dealing with limited amount of data (still a crux in certain applications, i.e. in

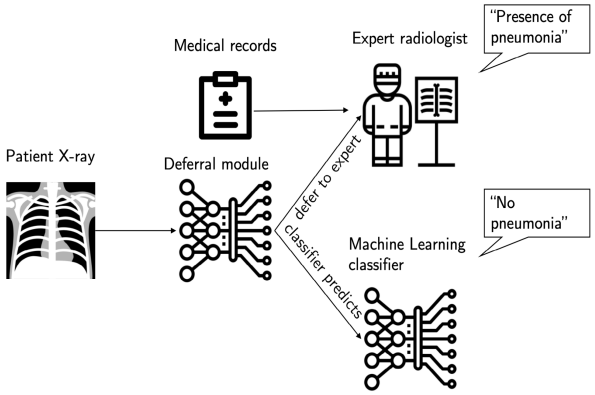


Figure 1.1: A medical scenario in which a network is trained to predict uncertainty of a classifier for medical images and to decide whether to defer decision to an expert radiologist. Taken from https://news.mit.edu/2020/machine-learning-health-care-system-understands-when-to-step-in-0731

medicine). Finally, a probabilistic approach enables us to leverage information and knowledge in a more principled way, either combining multiple models in an ensemble learner, or including prior knowledge in the data using appropriate prior distributions, leading to Bayesian machine learning.

1.1.1 The Importance of Uncertainty

Uncertainty is fundamental when we want to use the results of our models in making decisions. Uncertainty of a prediction is a measure of how much we can trust the prediction itself. Imagine an hypothetical scenario in which we describe uncertainty by Gaussian distribution, centered around the predicted value. Assume the value predicted is 5, but in one case the standard deviation is 0.1 and in the other case is 5. Which prediction would you trust more?

As an example of use of uncertainty to improve decisions, consider the scenario depicted in Figure1.1, in which a deep learning model is analysing x-ray images of patients. The system exploits an uncertainty measure to decide if the answer that will be computed by the model is trustable, and answers only in that case. Otherwise, the decision is deferred to an expert radiologist, that can take also the medical records into account.

Another example related to the safe control of cyber-physical systems is discussed briefly in Figure 1.2.

Uncertainty is broadly coming in two sauces: aleatoric and epistemic. **Aleatoric uncertainty** is essentially related to the randomness of the world around us, which can depend on the level of abstraction that we use in our description. For instance, we can consider a large set of gas molecules in a sealed room. We can in principle describe all their velocities and positions, but such a description will be too complex to obtain and manage. A much simpler and effective description is to assume that the molecules are randomly distributed in the room. This simple intuition gave rise to statistical mechanics in physics, and it is generally a good abstraction to describe complex systems.
Epistemic uncertainty, instead, corresponds to our lack of knowledge about a phenomenon. As an example, consider a pharmaceutical company developing a new drug. They have conducted several clinical trials to

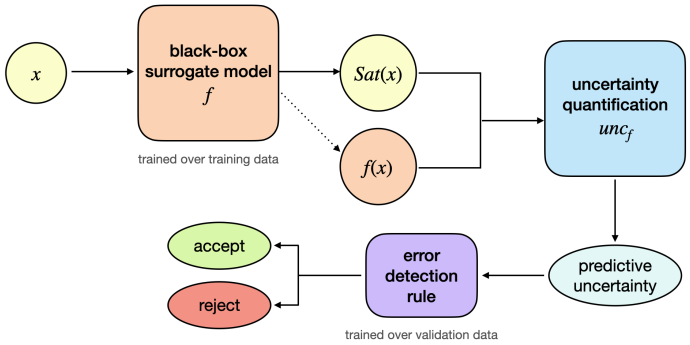


Figure 1.2: Neural predictive monitoring for cyber-physical systems. A neural network predicts is the system will be entering into an unsafe model of operation. The uncertainty of this prediction is then used to detect a potential error in the prediction, and to reject to make predictions that are likely to be wrong. This increases the trustability of the system and allows us to preemptively correct potential failures.
Example taken from L Bortolussi, F Cairoli et al. Neural predictive monitoring, Runtime Verification, 2019.

understand the drug's efficacy and safety profile. However, the drug's long-term effects remain unknown because it has only been tested over short periods. Here, the uncertainty arises from the lack of knowledge about how the drug will affect patients in the long term, not due to random effects. Note that as the drug is used over time in the general population, this epistemic uncertainty can decrease.

1.2 Generative Modelling

We now rephrase the fundamental classification of machine learning tasks in supervised and unsupervised learning using the language of probability. We can distinguish two classes of probabilistic models, which answer to different questions and contain different levels of information. In the following, the variables (features) of a system are divided in two groups: input x and output y. This subdivision is typically related to supervised learning tasks.

Discriminative models aim at describing the conditional probability of the output given the input:

$$p(y \mid x) \text{ or } y = f(x)$$

Instead of the conditional probability, we can restrict to a statistics or a function of $p(y \mid x)$ (for instance, the conditional average of y given x)

Generative models aim at describing the full probability distribution of inputs x or input/output pairs x, y:

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

Many machine learning tasks can be phrased in this setting:

- **Supervised learning**: learn $p(y \mid x)$ (or the best discriminant $y = f(x)$);
- **Unsupervised learning**: learn $p(x)$ or some properties of it (e.g. clusters);
- **Data generation**: learn a model of $p(x)$ and sample new elements from it (or from $p(x \mid y)$).

Notice that the last task is the one underneath Generative AI applications. For instance GPT-like models generate text in an autoregressive way (i.e. word by word).

1.3 Inference and Estimation

Two central concepts in probability and thus in probabilistic machine learning are the following:

- **Inference**: compute marginals and conditional probability distributions applying the laws of probability.
- **Estimation**: given data and a family of models, find the best parameters/models for the data.

The above distinction is typical in **frequentist** probability, i.e. a formulation of probability theory in which probabilities are seen as fraction of occurrences of events in repeated experiments (at least in principle, not all events are repeatable). The alternative probabilistic formulation is the **Bayesian** one, in which we treat everything as probabilistic, placing a prior distribution on our knowledge (i.e. model parameters) and computing a posterior distribution given the observations. In this framework, estimation boils down to inference.

Note that inference and estimation are typically hard computational tasks, and most of probabilistic machine learning aims at finding clever algorithms to perform efficiently these tasks.

1.4 Notes' Structure

In order to read the following notes and attend with profit classes, basic knowledge of probability, statistics and machine learning is required.

The notes are organized in chapters each dealing with a specific argument. The order of chapters reflects the order of presentation of argument in classes, but also a logical progression from description of models to efficient ways to perform inference and estimation.

Starred chapters and sections contain additional material with respect to the course. You do not have to study these sections, but they can give further insights to the world of probabilistic machine leaning.

Chapter two, however, is a bit anomalous, presenting a formulation of machine learning in terms of **empirical risk minimization** and connecting this formulation to the probabilistic perspective of maximum likelihood. In the chapter, we also discuss some basic idea of **algorithmic learning theory**, namely PAC learning and the related VC dimension and Rademacher complexity.

Chapter three introduces **probabilistic graphical models** (PGM), a graphical language to describe probability distributions (pd). Instances of this family are Bayesian networks and Markov Random Fields. A non-graphical variant of these ideas, using coding construct to model a pd as generative process, give rise to probabilistic programming languages.

One of such languages, Pyro, will be used throughout the course to code probabilistic models.

Chapter four shows how we can exploit the structure of (some) graphical models to perform **inference** tasks in an efficient way. It also discusses limitations of exact inference algorithms like **belief propagation**.

Chapter five discusses a special instance of PGM tailored to temporal series, namely **Hidden Markov Models**.

Chapter six introduces Bayesian machine learning, starting from the simplest model, i.e. **Bayesian linear regression**. We will work out the analytic formulation of regression, and discuss several aspects of Bayesian learning.

Chapter seven digs into the non-analytically tractable Bayesian classification, presenting **Bayesian logistic regression** and one first tool of approximate inference, namely **Laplace approximation**.

Chapter eight presents strategies for approximate inference based on sampling, in particular **Markov Chain Monte Carlo** methods, including some heavily used algorithms like **Hamiltonian Monte Carlo**. We will discuss also how to practically assess convergence of the method.

Chapter nine introduces a method to perform (approximate) parameter estimation for models in which some variables are not observed (i.e. latent). This approach, known as **Expectation Maximization**, is where we will first encounter the **Evidence Lower Bound**, a central tool in modern ML.

Chapter ten discusses the widely used **variational inference**, a method to approximate inference computations which is flexible and can be adapted to stochastic gradient descent algorithms as **Stochastic Variational Inference**. We will also discuss application of this approach for learning **Bayesian Neural Networks**.

Chapter eleven introduces generative modelling, presenting **variational auto-encoders**, one of the mostly used architectures. We will discuss them from the point of view of variational inference, explaining their training procedure in detail.

Chapter twelve continues in presenting a state of the art architecture of generative modelling, especially for images, namely **diffusion models**, in which data is slowly degraded into noise and we learn how to invert such a degradation process.

Chapter thirteen, finally, presents kernel methods and their application to Bayesian learning, namely **Gaussian Processes**, which are a non-parametric approach based on kernels.

Empirical Risk Minimization and PAC Learning

2

In this chapter, we discuss the theoretical foundations of machine learning, presenting the framework of empirical risk minimization (Sec. 2.1) in which to frame learning problems, the notion of inductive bias, and the main results of algorithmic learnability, encapsulated in the definition of PAC learning (Sec. 2.2) and of complexity of a set of hypothesis, namely VC-dimension and Rademacher complexity (Sec. 2.2.4 and 2.2.6). We will finish the chapter with an introduction to the information theoretical notion of Kullback-Leibler divergence (Sec. 2.3) and its link with maximum likelihood and empirical risk. The presentation of this chapter follows [6] and [7].

2.1 Empirical Risk Minimization

2.1.1 Notation and problem formalization

Consider the supervised learning scenario, in which there are an input and an output space. We will use the following notation:

- input space: $X \subseteq \mathbb{R}^n$ (real features or one-hot-encoding of categorical variables)
- output space: $Y = \mathbb{R}$, or $\{0, 1\}$, or $\{0, \dots, k\}$ depending on the problem at hand (i.e. regression, binary classification, multi-class classification).

We will work in a probabilistic framework. They key ingredient is the joint probability distribution between inputs and outputs $p(x, y) \in Dist(X \times Y)$ (with $x \in X, y \in Y$), called **data generating distribution**.

We assume that there is a functional relationship between inputs and outputs that we want to understand, given by a labelling function $f : X \to Y$. Re-writing the data generating distribution as $p(x, y) = p(x)p(y|x)$, it is sometimes the case that $p(y|x) = p(y|f(x))$ (e.g. in classification problems, or in regression problems in which an additive measurement error is assumed).

The input to our learning algorithm is a dataset D, where $|D| = N$, sampled from a probability distribution $D \sim p^N(x, y)$, i.e. it is a set of input-output pairs $D = \{(x_i, y_i)|i = 1, \dots, N\}$ such that $(x_i, y_i) \sim p(x, y)$. We assume these pairs to be independent (this is not true in general, however it is not a very restrictive assumption). Ideally we would like to recover f, starting from D.

A key point in Machine Learning is that, whenever we want to learn something, we need to make hypotheses on good candidate models for our task. So, since we are going to learn a function mapping inputs to

outputs (possibly phrased in probabilistic terms), we need to choose a set of functions that are likely to contain our true model, or at least to approximate it well.
This set is called **hypothesis class**, defined as $\mathcal{H} = \{h : X \to Y\}$. Typically these functions are chosen to be parametric, i.e. $h = h_\theta$ with $\theta \in \Theta \subset \mathbb{R}^k$ for some k.

Remark: restricting the set of models to consider is actually what enables us to learn (without any assumption, we cannot learn anything)!

$\mathcal{H}$ encodes our **inductive bias**, that is, the assumptions made to learn the target function and to generalize beyond training data. We stress once again that without inductive bias there is no learning.

2.1.2 Risk and empirical risk

Consider our set of hypotheses $\mathcal{H}$, a function $h \in \mathcal{H}$ and the joint probability distribution $p(x, y)$.

Definition 2.1.1 *The **loss function** $l(x, y, h) \in \mathbb{R}_{\geq 0}$, measures the error that we commit in using h to predict y from x. Intuitively, the higher its value, the worst our prediction.*

Examples:

- $0-1$ loss: $l(x, y, h) \equiv \mathbb{I}(h(x) \neq y)$, with $y \in \{0, 1\}$. That is, we have no error if we predict correctly the label of x, an error of 1 otherwise.
- squared loss $l(x, y, h) \equiv (h(x) - y)^2$, with $y \in \mathbb{R}$.

Remark: the loss function acts on a single input-output pair.

Definition 2.1.2 *The **risk** (or **generalization error**) is defined as*

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

The risk is therefore a property of the hypothesis function h, i.e. each h comes with an associated risk. Our goal is to find a function h that minimizes the risk.

Remark: the risk depends on the true data distribution (which is unknown).

Definition 2.1.3 *The **empirical risk** (or **training error**) is defined as :*

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

It is an empirical approximation, according to our sample, of the actual risk. This is what we can practically optimize.

Risk minimization principle: find $h^\star \in \mathcal{H}$ s.t. $h^\star = \operatorname{argmin}_{h \in \mathcal{H}} R(h)$. That is, we need to find the hypothesis that minimizes the risk.

Definition 2.1.4 *If l is the $0-1$ loss and $\exists h \in \mathcal{H}$ s.t. $p(h(x) = f(x)) = 1$ (with $f(x)$ true class), then $\mathcal{H}$ has the **realizability property** and $R(h^\star) = 0$.*

Hypothesis sets with the realizability property contain the true model (in general this is not the case in practice).

Empirical risk minimization principle: find $h_D^\star = \text{argmin}_{h \in \mathcal{H}} \hat{R}(h)$. This minimum can be combinatorially hard to achieve, so sometimes we relax this framework and we only find a *good* solution. In what follows, we are going to address the problem of computing $R(h_D^\star)$, i.e. risk associated to the optimal solution.

2.1.3 Bias-Variance Trade-off

In this section, we want to analyze the generalization error and decompose it according to the sources of error that we are going to commit, in order to shred some insight on the effect of a choice of the hypothesis set $\mathcal{H}$ on the generalization error.

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

$$R(h) = \mathbb{E}_p[l(x, y, h)] = \iint (h(x) - y)^2 p(x, y) dx dy$$

Theorem 2.1.1 *The minimizer of the generalization error R is:*

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

so that $g = \text{argmin}_h R(h)$, if $g \in \mathcal{H}$.

Proof. We can prove it by rewriting our risk as:

$$\begin{aligned} R(h) &= \iint (h(x) - y)^2 p(x, y) dx dy = \\ &= \iint (h(x) + g(x) - g(x) - y)^2 p(x, y) dx dy = \\ &= \iint [(h(x) - g(x))^2 + (g(x) - y)^2 + 2(h(x) - g(x))(g(x) - y)] p(x, y) dx dy \end{aligned}$$

Consider the term:

$$\begin{aligned} &\iint 2(h(x) - g(x))(g(x) - y) p(x, y) dx dy = \\ &= \int 2(h(x) - g(x)) p(x) \int (g(x) - y) p(y|x) dy dx = 0 \end{aligned}$$

since

$$\int (g(x) - y)p(y|x)dy = g(x) - \int yp(y|x)dy = 0$$

by definition of g. So that:

$$R(h) = \int (h(x) - g(x))^2 p(x)dx + \iint (g(x) - y)^2 p(x, y)dxdy$$

where the second term does not depend on h and expresses the idea of how noisy is our regression problem (this is something intrinsic to the problem).
The first term depends on h (actually it is the only thing that we can optimize when we minimize w.r.t. h), and it holds that:

$$\int (h(x) - g(x))^2 p(x)dx = 0 \leftrightarrow h(x) = g(x)$$

and so we proved that $g(x)$ is a minimizer. □

Remark: our goal is to to evaluate $R(h_D^\star)$, with $D \sim p^N(x, y)$.

By previous computations (evaluated at the solution of the empirical risk minimization problem), it holds that:

$$R(h_D^\star) = \int (h_D^\star(x) - g(x))^2 p(x)dx + noise$$

Consider now the average over all possible datasets:

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

Carrying on the same computations as before and observing that:

$$\mathbb{E}_D[h_D^\star(x) - \mathbb{E}_D[h_D^\star(x)]] = 0$$

we get that the expected generalization error of our empirical risk minimizer is:

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

The first term $\int (\mathbb{E}_D[h_D^\star(x) - g(x)])^2 p(x)dx$ captures the squared difference between the average predictor across all datasets (that we obtain from empirical risk minimization) and the optimal predictor. If this

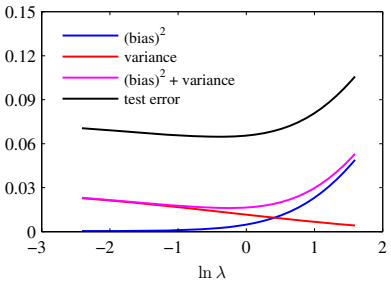


Figure 2.1: Bias-variance tradeoff. $\ln \lambda$ is a proxy for model complexity. Taken from Bishop.

difference is small, on average across all datasets our empirical risk minimization is going to work well; if it is large, across all datasets our empirical risk minimization is going to work bad. We call this term *squared bias* because it essentially captures the distortion of our empirical risk minimization predictor, hence of our set of hypothesis $\mathcal{H}$.

The second term $\int \mathbb{E}_D[(h_D^\star(x) - \mathbb{E}_D[h_D^\star(x)])^2]p(x)dx$ encodes the variance, across all datasets, of our predictor, that is, how far each single instance of the empirical risk minimizer differs from the average predictor. We call this the *variance term*. The larger the variance, the more the intrinsic variability of the dataset is going to impact on what we can reconstruct.

The third term $\iint (g(x)-y)^2 p(x,y)dxdy$ is the noise (as observed before).

Remark: high variance essentially means that we are in a region of overfitting, high bias means that we are in a region of underfitting.

Since our goal is to minimize the empirical risk, we face a trade-off, called the **bias-variance trade-off**. On one hand, choosing $\mathcal{H}$ to be a very rich class might lead to overfitting (increasing the variance). On the other hand, choosing $\mathcal{H}$ to be a very small set might lead to underfitting (increasing the bias). A visual depiction of the trade-off is shown in Figure 2.1.

2.2 PAC Learning

2.2.1 Basic definitions

In what follows, 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.

We are going to explore this framework in the context of (binary) classification, i.e. $y \in \{0,1\}$, using the $0-1$ loss.

Consider an hypothesis set $\mathcal{H}$ with the realizability property, i.e. $\exists \bar{h} \in H$ s.t. $p_{x,y}(\bar{h}(x) = y) = 1$, since $y \in \{0,1\}$ then $\exists f : X \to Y$ s.t. $p_{x,y}(\bar{h}(x) = f(x))$ (that is, our hypothesis set contains the true function).

Definition 2.2.1 *A realizable hypothesis set $\mathcal{H}$ is **PAC-learnable** iff* $\forall \varepsilon, \delta \in (0,1), \forall p(x,y), \exists m_{\varepsilon,\delta} \in \mathbb{N}$ *s.t.* $\forall m \geq m_{\varepsilon,\delta}, \forall D \sim p^m, |D| = m$ *then*

$$p_D(R(h_D^\star) \leq \varepsilon) \geq 1 - \delta.$$

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 $m_{\varepsilon,\delta}$ of observations (as a function of the parameters ε, δ), such that we are guaranteed to learn the true function with error bounded by ε (assuming that the true function is in $\mathcal{H}$) with high probability $(1-\delta)$, provided we have at least $m_{\varepsilon,\delta}$ data points in D. 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 the following, we consider a more general setting, in which we relax the realizability assumption, and furthermore we assume that we have at our disposal an algorithm A that takes D as input and returns a function h of $\mathcal{H}$ as output, ideally the minimizer of the empirical risk, but practically a good solution.

Definition 2.2.2 *Given an hypothesis set $\mathcal{H}$ (not necessarily realizable) and an algorithm A, $\mathcal{H}$ is **agnostic PAC-learnable** iff* $\forall \varepsilon, \delta \in (0,1), \forall p(x,y), \exists m_{\varepsilon,\delta} \in \mathbb{N}$ *s.t.* $\forall D \sim p^m,\ |D| = m \geq m_{\varepsilon,\delta}$

$$p_D\left(R(h_D^A) \leq min_{h \in H} R(h) + \varepsilon\right) \geq 1 - \delta$$

being h_D^A the result of applying A to $\mathcal{H}$ and D.

In other words, there exists a number $m_{\varepsilon,\delta}$ of data points such that the algorithm learns a function having error close ($\leq \varepsilon$) to the minimum with high probability $(1-\delta)$.

In both definitions, we have a bound on the generalization error in terms of ε and δ and, in order for this bound to hold, we need to have enough data points. Typically:

- $m_{\varepsilon,\delta}$ depends polynomially on $\frac{1}{\varepsilon}, \frac{1}{\delta}$ (since we want the number of observations to increase moderately with the complexity of the problem);
- A should run in polynomial time.

2.2.2 Finite hypotheses sets

An hypothesis set is said to be **finite** if $\mathcal{H}$ is s.t. $|\mathcal{H}| < \infty$.
Using combinatorial arguments, we can prove that finite hypothesis sets are agnostic PAC-learnable with:

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

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

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

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

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

2.2.3 Pac learning example

We consider an example introduced by Kearns and Vazirani in the book "An Introduction to Computational Learning Theory".

The goal is to learn a target axis-aligned rectangle R lying in $\mathbb{R}^2$ using a sample of m labeled training examples $(\mathbf{x}, z)$ with $\mathbf{x} \in \mathbb{R}^2$ and $z \in \{-1, 1\}$ distributed according to a distribution $p(\mathbf{x}, z)$. The hypothesis set $\mathcal{H}$ is the set of all axis-aligned rectangles lying in $\mathbb{R}^2$. We assume there is a true rectangle of this kind such that all positive points are inside it, and all negative points are outside.

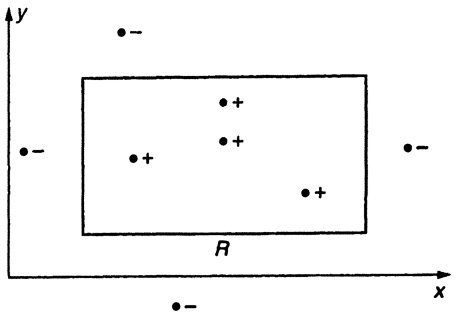


Figure 2.2: The target rectangle R with a sample of positive and negative examples

We consider our hypothesis h to be the axis-aligned rectangle R' with the smallest area that includes all of the positive examples and none of the negative ones.

We define the error of our hypothesis, error(h), as the probability that a positive point sampled from $p(\mathbf{x}, z)$ falls in the region between the true rectangle and R' (and so it is misclassified accordingly to h).

What is the minimum number $m_{\varepsilon,\delta}$ of training examples so that, with probability at least $1 - \delta$, h has en 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 $\forall m > m_{\varepsilon,\delta}\ P(\text{error}(h_m) > \varepsilon) \leq \delta$. First of all, let's notice that the error error(h_m) is proportional 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 four rectangles within R, each one such that the probability of falling inside it is $\varepsilon/4$, on the sides of R (see Figure 2.3).

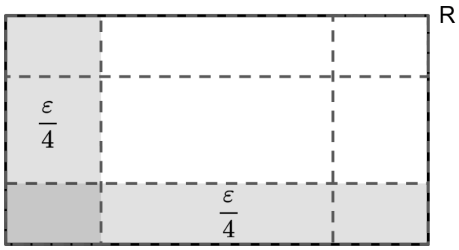


Figure 2.3: The 4 rectangles of area proportional to $\varepsilon/4$

Let us 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 difference between R and R' would be fully covered by the 4 rectangles (see Figure 2.4).

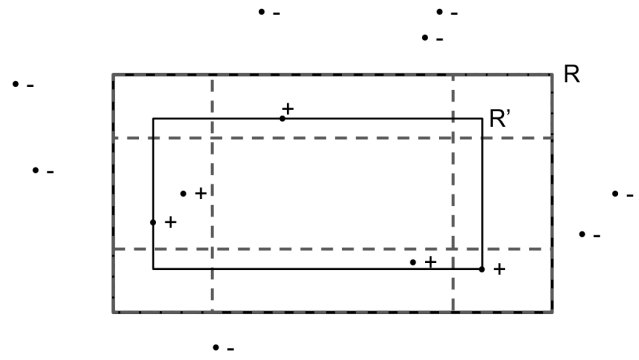


Figure 2.4: Configuration showing event B

If we call this event B, we have:

$$B \implies \text{error}(h_m) \leq \varepsilon$$

equivalently, by modus tollens:

$$\text{error}(h_m) > \varepsilon \implies \neg B$$

This implies:

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

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

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

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

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

Finding $m_{\varepsilon,\delta}$ that satisfy this inequality would prove that $\forall m > m_{\varepsilon,\delta}\, P(\text{error}(h_m) > \varepsilon) \leq \delta$. 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 $\mathcal{H}$ is PAC-learnable.

2.2.4 VC Dimension (Vapnik–Chervonenkis)

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

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

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

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

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.

Examples:

- $\mathcal{H}_a = \{h : \mathbb{R} \to \{0,1\} | h = \mathbb{1}_{\geq a}, a \in \mathbb{R}\}$ (threshold functions) has $VCdim(\mathcal{H}_a) = 1$
- $\mathcal{H}_{a+} = \{\mathbb{1}_{\geq a}, \mathbb{1}_{\leq a} | a \in \mathbb{R}\}$ has $VCdim(\mathcal{H}_{a+}) = 2$
- $\mathcal{H}_I = \{\mathbb{1}_{[a,b]} | a \leq b, a, b \in \mathbb{R}\}$ (intervals) has $VCdim(\mathcal{H}_I) = 2$
- $VCdim(\mathcal{H}_I \cup (1 - \mathcal{H}_I)) = 3$
- $\mathcal{H}_\ell = \{\mathbb{1}_{ax+by+c \geq 0}(x,y) | a, b, c \in \mathbb{R}\}$ (lines in $\mathbb{R}^2$) has $VCdim(\mathcal{H}_\ell) = 3$
- $\mathcal{H}_B = \{\mathbb{1}_B(x) \mid B \text{ is an axis-aligned box in } \mathbb{R}^2\}$ has $VCdim(\mathcal{H}_B) = 4$

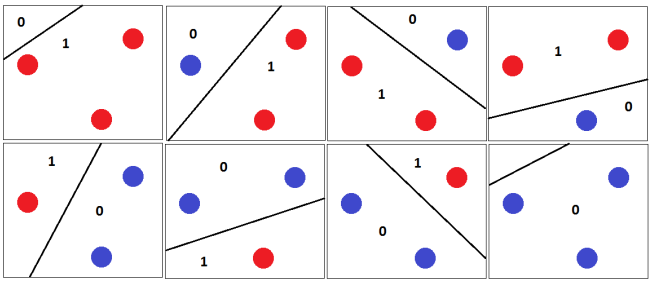


Figure 2.5: Proof that $VCdim(\mathcal{H}_\ell) \geq 3$

2.2.5 VC dimension and PAC learning

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

Proposition 2.2.1 *If $\mathcal{H}$ shatters C, $|C| \geq 2m$, then we cannot learn $\mathcal{H}$ with m samples.*

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

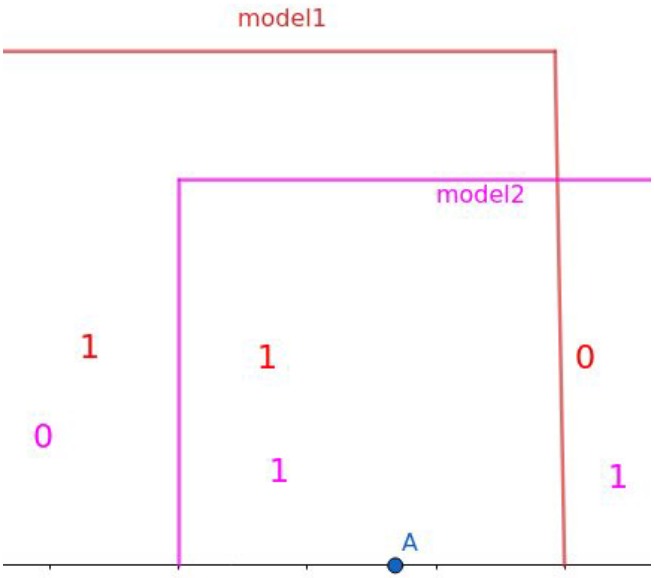


Figure 2.6: Visual interpretation of the theorem: it is impossible to train a model of type $\mathcal{H}_{a+}$ with only a point A with known classification (suppose 1) because the points different from A could have any classification.

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

Theorem 2.2.2 *$\mathcal{H}$ is (agnostic) PAC-learnable iff $VCdim(\mathcal{H}) < \infty$.*

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

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

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

2.2.6 Rademacher Complexity *

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

Definition 2.2.4 *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 2.2.5 *The **data-dependent Rademacher complexity** is defined as:*

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

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 σ 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 2.2.6 *The **data-independent Rademacher complexity** is defined as:*

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

Remark: this definition takes into account the data generating mechanism p.

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

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

$$R(h) \leq \hat{R}_D(h) + \underbrace{\hat{\mathcal{R}}_D(\mathcal{H}) + 3\sqrt{\frac{log(\frac{2}{\delta})}{2m}}}_{\varepsilon_2}$$

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

2.2.7 Rademacher complexity and VC dimension *

Definition 2.2.7 *The **growth function** $\Pi_{\mathcal{H}}$ is defined as:*

$$\Pi_{\mathcal{H}} : \mathbb{N} \to \mathbb{N},\ \Pi_{\mathcal{H}}(m) = \max_{C \subseteq X, |C|=m} |\mathcal{H}_C|$$

with $\mathcal{H}_C = \{(h(c_1), \ldots, h(c_m)) | h \in \mathcal{H}, C = \{c_1, \ldots, c_m\}\}$.

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

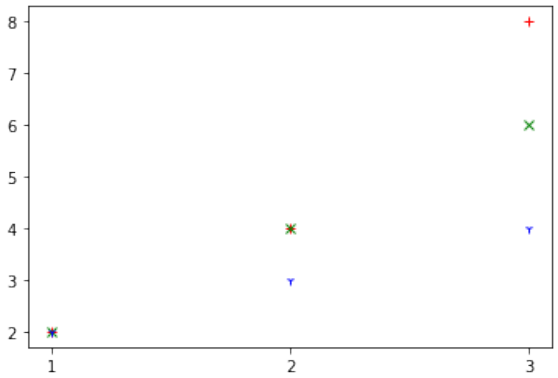


Figure 2.7: Plot of the growth functions of $\mathcal{H}_a$ (blue), $\mathcal{H}_{a+}$ (green) , $\mathcal{H}_\ell$ (red) for $1 \leq m \leq 3$

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

$$VCdim(\mathcal{H}) = \max\{m | \Pi_{\mathcal{H}}(m) = 2^m\}$$

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

Lemma 2.2.3 *Sauer Lemma:*

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{e \cdot m}{d}\right)^d \leq O(m^d)$$

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

Moreover it also holds that:

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

We can say that $\mathcal{R}_m(\mathcal{H})$ and $VCdim(\mathcal{H})$ are essentially equivalent, in the sense that when the VC dimension is ∞ then the upper bound on the Rademarcher complexity is independent of m and greater than 1 (if you substitute it into the PAC bound using Rademacher 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.

2.2.8 ERM and Maximum Likelihood

In the maximum likelihood framework:

- we have 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 distribution as: $p(x, y) = p(x)p(y|x)$ and we make hypothesis on $p(y|x)$, trying to express this conditional probability in a parametric form $p(y|x) = p(y|x, \theta)$
- 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, according to which:

$$\theta_{ML} = \operatorname*{argmax}_{\theta} L(\theta; D) = \operatorname*{argmin}_{\theta} -L(\theta; D)$$

It holds that:

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

since the average is an empirical approximation of the expectation.

Definition 2.2.8 $-\frac{1}{m} \sum_{i=1}^{m} \log p(y_i|x_i, \theta)$ *is known as* ***cross entropy****.*

Essentially, in the maximum likelihood framework, the loss function is $l(x, y, \theta) = -\log p(y|x, \theta)$, so we can recast the maximum likelihood principle as a minimization of the (empirical) risk.

For what concerns the space of hypotheses functions $\mathcal{H}$, typical choices are:

- for regression problems: $h(x, \theta) = \mathbb{E}_y[p(y|x, \theta)]$
- for classification problems: $h(x, \theta) = \operatorname{argmax}_y p(y|x, \theta)$ (i.e. the Bayes decision rule)

A typical choice of $p(y|x, \theta)$ in case of regression problems is: $\mathcal{N}(h_\theta(x), \sigma^2)$ so that $-\log(p(y|x, \theta)) \propto (y - \log(x))^2$, that is, the loss as sum of squares comes from the probabilistic assumption that the noise model on the data is Gaussian, centered around the true value.

2.3 KL divergence

Consider a probability distribution $p(x)$, than $-\log p(x)$ is a measure of **self-information**. Indeed, if $p(x) = 1$ then $-\log p(x) = 0$ (no self-information), describing substantially our (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 (the more surprise its occurrence brings), i.e. the larger is $-\log p(x)$.

In an information-theoretic sense, the **entropy** is a measure of the information that is carried by a random phenomenon, expressed as the expected amount of self-information that is conveyed by a realization of the random phenomenon.

Entropy is formally defined as:

$$H[p] = \mathbb{E}_p[-\log p(x)] = -\int p(x) \log p(x) dx$$

for the continuous case, and:

$$H[p] = -\sum_i p(x_i) \log p(x_i)$$

for the discrete case.

In the discrete case, the maximum entropy is achieved for the uniform distribution and it is equal to $\log K$, with K number of events that can happen. In the continuous case, for a fixed variance, the distribution that maximizes entropy is the Gaussian. The entropy is always 0 if we have a deterministic distribution.

Definition 2.3.1 *Consider two distribution p, q; the **Kullback-Leibler divergence** (also called **relative entropy**) is defined as:*

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

$KL[q||p] = 0$ iff $q = p$.

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.

Some properties of the KL divergence:

- $KL[q||p]$ 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 q and p.

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

We can do this by finding $\theta^\star = \text{argmin}_\theta \, KL[q_\theta||p]$. This is at the basis of variational inference techniques.

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

$$I[x, y] = KL[p(x, y)||p(x)p(y)]$$

$KL[p(x, y)||p(x)p(y)] = 0$ iff $p(x)$ and $p(y)$ 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 2.3.2 *The **empirical distribution** is defined as:*

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

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:

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

If $q = q_\theta$, 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_θ.
This means that we can always rephrase maximum likelihood in terms of cross-entropy.

Probabilistic Graphical Models 3

3.1 Probabilistic Inference

3.1.1 Introduction

Suppose to have the following set of random variates, which can be either discrete or continuous:

- $x = (x_1, \dots, x_n)$
- $z = (z_1, \dots, z_m)$

A **probabilistic model of the world** is expressed as a joint probability distribution, i.e. $p(x, z)$.

Typically, to reason about such a model, we would like to perform these operations:

- Marginalization: $p(x) = \int p(x, z)dz$ (or $\sum_z p(x, z)$ if z is a discrete R.V.)
- Conditioning: $p(z|x = \bar{x}) = \frac{p(\bar{x},z)}{p(\bar{x})}$, where $\bar{x}$ is a fixed value

These two operations may be combined, i.e. we may want to compute

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

What is the cost in terms of computation of performing these operations?

Let's focus on discrete values for z such that $z_i \in \{0, 1\}$ and consider the marginalization over z:

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

This seems like an easy task, but $z \in Z$, with $|Z| = 2^m$. If $m = 100$, then $|Z| \approx 10^{30}$, and we would have no chance to compute the summation, or even store our joint distribution in memory. Therefore knowing our model of the world might not be enough to get information regarding the world.

3.1.2 Factorization

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

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

This is known as a **factorization** of our joint distribution. In some cases, factorization can be simpler, because some variables in the conditioning set of a factor may be irrelevant to other variables. For example we could have

$$p(x_1, \cdots, x_n) = p(x_n|x_{n-1})p(x_{n-1}|x_{n-2}) \cdots p(x_2|x_1)p(x_1) \qquad \text{(c)}$$

What is the cost in terms of computer memory, to store these three representation of our distribution? Suppose $x_i \in \{1, \cdots, k\}$. Then we have the following storing costs:

- $p(x_1)$ costs $k - 1 = O(k)$ floating point numbers (either in single or double precision)
- $p(x_2|x_1)$ costs $k(k-1) = O(k^2)$ float numbers (we have k 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 (a) costs k^n float numbers. Unfeasible.
- Factorization (b) costs $O(k^n)$ because of the term $p(x_n|x_{n-1}, \cdots x_1)$
- Factorization (c) instead has just a total cost of $O(nk^2)$. This makes storing it in memory tractable.

Therefore, if we are able to exploit factorizations in some way, the problems of conditioning and marginalization become tractable.

Remark: given a joint distribution $p(x_1, \ldots, x_n)$, there are several ways (in fact, $n!$) of factorizing it, similarly to (b) above, depending on the order of variables. Any of these choices may correspond to a different factorization, i.e. to different ways of removing variables from conditioning set. The problem of identifying the most parsimonious factorization is known to be NP-hard.

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

Consider a joint distribution on $x_1, \ldots, x_4$, and assume the following factorization holds:

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

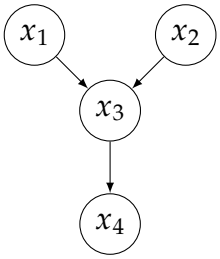
Definition 3.2.1 *We call pa_k the parents of x_k, i.e., the conditioning set of x_k*

This means that $p(x_k|pa_k)$ is the factor for x_k.

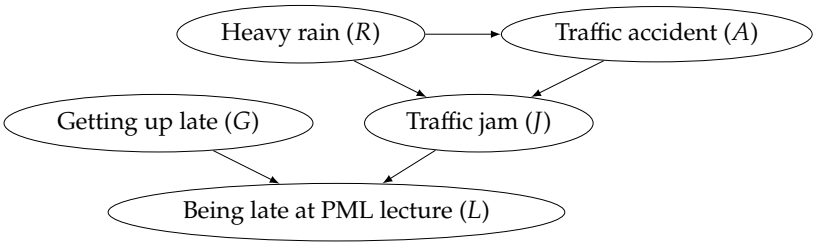
Example: In $p(x_1, x_2, x_3, x_4) = p(x_4|x_3)p(x_3|x_2, x_1)p(x_2)p(x_1)$ we have: $pa_4 = \{x_3\}$, $pa_3 = \{x_2, x_1\}$, $pa_2 = \{\}$, $pa_1 = \{\}$

Definition 3.2.2 *A **Bayesian Network (BN)** is a DAG (Direct Acyclic graph), having $\{x_1, \ldots, x_n\}$ as vertices and (x_i, x_j), $i < j$ and $x_i \in pa_j$ as edges.*

Example: The BN for $p(x_1, x_2, x_3, x_4) = p(x_4|x_3)p(x_3|x_2, x_1)p(x_2)p(x_1)$ is:



Example:



This BN corresponds to the following factorization:

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

3.2.1 Notational conventions

There are some conventions for writing the BN graph, here we list the most common ones:

- Observed nodes (Random variables of which we know the value) are **coloured** or shadowed 
- Plated nodes are a shorthand for a collection of nodes. The following represents the nodes $x_1, \ldots, x_n$ 
- Deterministic quantities represented as small solid circles. These are fixed parameters of the model (represented graphically as • α)

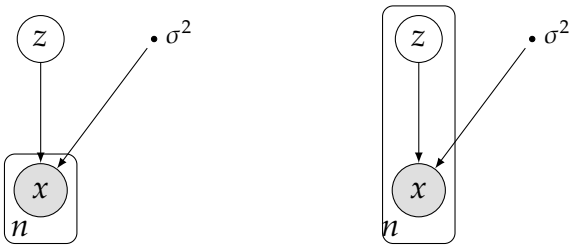
Example: Suppose to have now a RV z that follows a discrete distribution and another RV x that is normally distributed with mean $\mu(z)$, depending on the value of z, and variance σ^2 (shorthand for this is $x \sim \mathcal{N}(\mu_z, \sigma^2)$. This model is known as a **mixture of gaussians**. We can write

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

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, $\bar{x} = x_1, \cdots, x_n$, 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 one, $\bar{x}$ are sampled from a single realization of the variable z (on the left), while in the second one each x_i may have been drawn with a different z_i, hence we are interested in computing $p(z_i|x_i)$ (on the right):



The parameter σ^2 corresponds to the variance of the normal distribution associated with x.

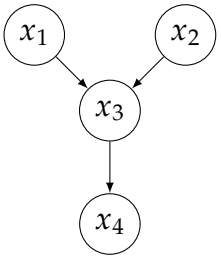
3.3 Sampling and reasoning in Bayesian Networks

We describe now a simple sampling algorithm for BN, and then turn to discuss several reasoning schemes.

3.3.1 Ancestral sampling

Let's recall our example

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

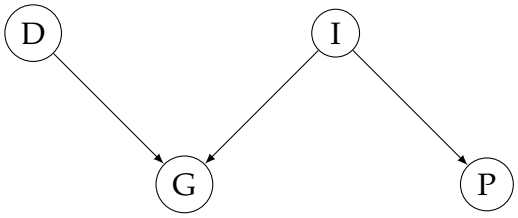


Ancestral sampling is a sampling technique that allows us to sample from a given distribution, if we know its factorization.

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 each marginal and conditional.

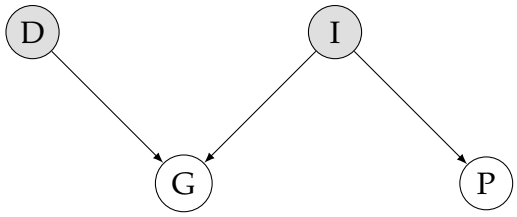
3.3.2 Reasoning with Bayesian Networks: an example

We'll move to another example of a Bayesian Network. We consider a model of students' grade in an exam.



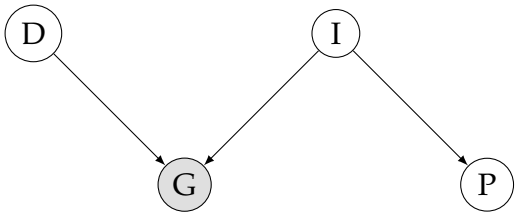
The difficulty of the exam (D) and the intelligence of the student (I) are independent, but the grade you get when you take an exam (G) is dependent on both (studying of course helps too). Having passed a hard exam (P) likely depends on the student's abilities but not on the difficulty and the grade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

- **Causal Reasoning**



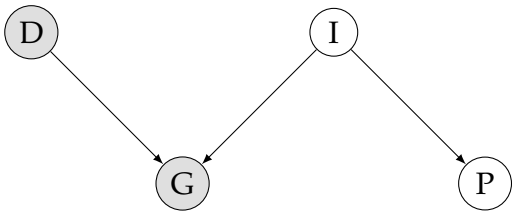
We observe something (the difficulty of the exams, how intelligent we are) and we *infer* the grade and if we have passed a hard exam. The reasoning goes from top to bottom.

- **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**



The information is propagated in both directions along the network.

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.

3.4 Conditional Independence in Bayesian Networks

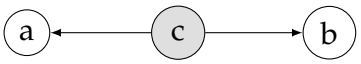
We define now in detail the notion of conditional independence, and study what BNs tell us in this respect.

Definition 3.4.1 *Consider the RVs a, b, c. We say that a is **conditional independent** of b given c (written $a \perp\!\!\!\perp b|c$) iff $p(a|b,c) = p(a|c)$ or equivalently $p(a,b|c) = p(a|c)p(b|c)$*

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

3.4.1 Tail to tail

Definition 3.4.2 *Consider the RVs a, b, c. We say that a and b are tail to tail with c iff $p(a,b,c) = p(a|c)p(b|c)p(c)$*



We know that this Bayesian network implies the following factorization

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

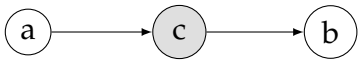
Also

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

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

3.4.2 Head to tail

Definition 3.4.3 *Consider the RVs a, b, c. We say that a and b are head to tail with c iff $p(a,b,c) = p(a)p(c|a)p(b|c)$ 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

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

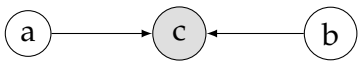
Also

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

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

3.4.3 Head to head

Definition 3.4.4 *Consider the RVs a, b, c. We say that a and b are head to head with c iff $p(a,b,c) = p(a)p(b)p(c|ab)$*



You know the drill:

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

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 $a \perp\!\!\!\perp b$ and $a \not\perp\!\!\!\perp b|c$.
Notice that also $a \not\perp\!\!\!\perp b|d, \forall d$ descendant of c

3.4.4 Conditional independence on the graph

Let's formalize the notion of conditional independence on Bayesian Network that we just saw:

Definition 3.4.5 *Given RVs a,b,c, a path from a to b is **blocked** by c iff:*

- *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.*

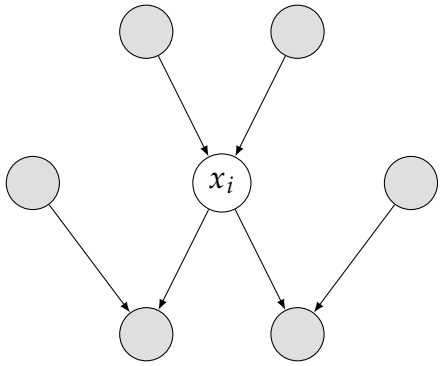
Proposition 3.4.1 *Let A, B, C subsets, if all paths from a node in A to a node in B are blocked by a node in C then $A \perp\!\!\!\perp B|C$*

3.4.5 Markov blanket

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

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

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 those belonging to pa_i, or those for which $x_i \in pa_j$. The union of pa_i with nodes in $\{x_j\} \cup pa_j$ 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.



3.5 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).

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

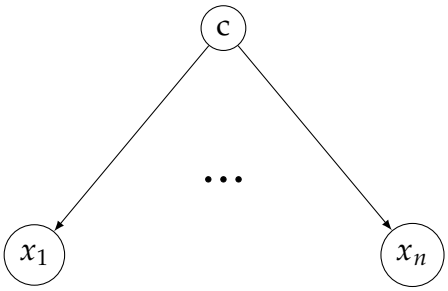
What we want to compute when we need to solve a classification task, is actually

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

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

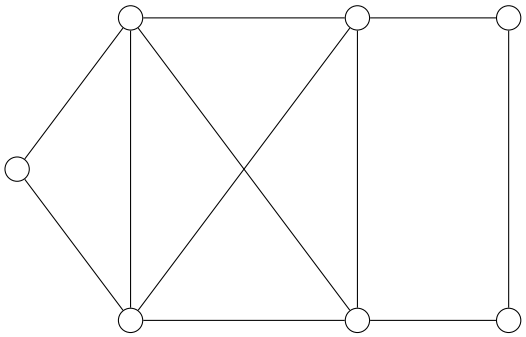
$$\frac{p(c=0|x_1,\dots,x_n)}{p(c=1|x_1,\dots,x_n)}$$

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 application of Naive Bayes is in the domain of document classification.

3.6 Random Markov Fields

Random Markov Fields are an undirected graph representation of a probabilistic 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.



3.6.1 Conditional independence on Markov Random Fields

Proposition 3.6.1 *Consider three subsets A, B, C. $A \perp\!\!\!\perp B|C$ 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 3.6.1 *A **Markov Blanket** in a Markov Random Field for a given x_i is the set of neighbours of x_i*

3.6.2 Factorization

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

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

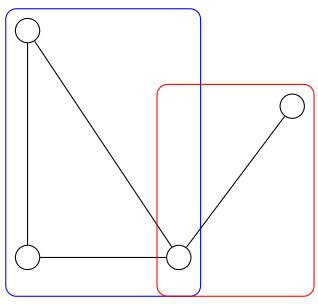
and x_i, x_j belong to different factors.

This means that nodes connected by an edge should belong to the same factor(s). Extending the reasoning, is 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 3.6.2 *A **clique** is a fully connected subgraph.*

Definition 3.6.3 *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 $\mathcal{C}$ = {set of maximal cliques} and $x_C = \{x | x \in C, C \in \mathcal{C}\}$

Then the factorization of a Markov Random Field is obtained as

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

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 3.6.4 $\Psi_C(x_C) \geq 0$ *is called the* ***potential function*** *and it must have a finite integral,* $\int \Psi_C(x_C) dx_C < \infty$.

Definition 3.6.5 $Z = \int \prod_C \Psi_C(x_C) dx_C < \infty$ *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.

Exact Inference in Probabilistic Graphical Models

4

4.1 Introduction

In the context of inference, our task is, given a joint distribution $p(x) = p(x_1, \dots, x_n)$, to compute marginals or conditionals of this 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 (this is denoted by $y = \bar{y}$), we want to compute the conditional

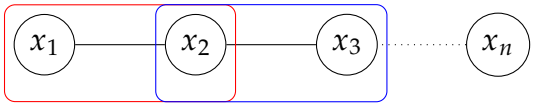
$$p(z|y = \bar{y}) = \sum_{x'} p(z, x'|y = \bar{y}) \text{ discrete r.v.}$$
$$= \int p(z, x'|y = \bar{y}) dx' \text{ continuous r.v.}$$

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

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

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

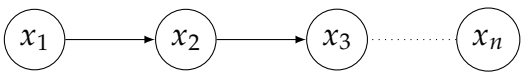
For example, consider a Markov Random Field where variables are connected in a chain (colored rectangles represent the cliques):



The factorization implied by this PGM is:

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

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



The factorization in this case reads as:

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

Note: chain structures like the previous are common because they represent temporal series (more specifically, they are Markov processes).

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

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

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):

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

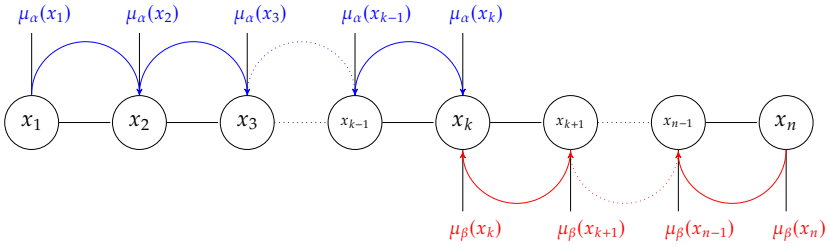
Note that complexity is now $O(n \cdot k)$, i.e. linear in n.

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 has a message:



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.

4.2 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 Factor Graphs 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:

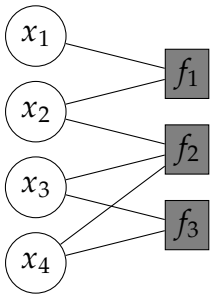
- variable nodes, denoted by 

- factor nodes, denoted by 

Consider the distribution (with f_i factors and x_i variables):

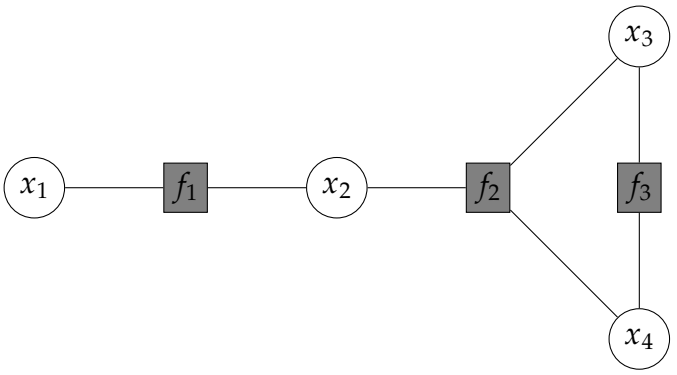
$$p(x) \propto f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_3, x_4)$$

The canonical way of represent bipartite graphs is the following:



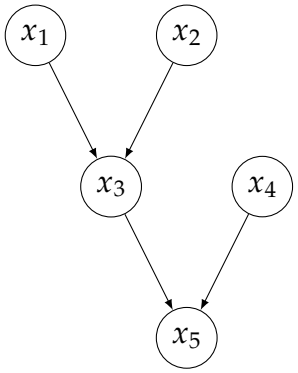
That is, all variable nodes are put on the left and all factor nodes on the right.

A more convenient, equivalent, representation could be:



4.2.1 From Bayesian Networks to Factor Graphs

Consider the following Bayesian network:

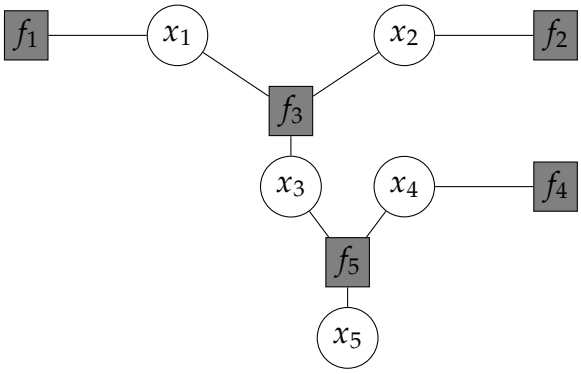


Which corresponds to the factorization:

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

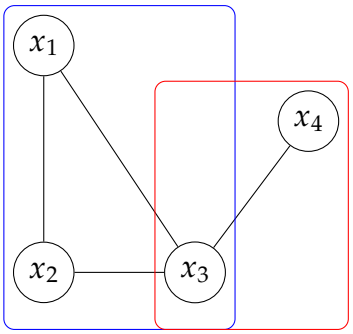
Fundamentally the idea is to have a factor node (connected to the proper variable nodes) for each term of the factorization implied by the Bayesian Network.
Formally: $\forall p(x_j|pa_j) \rightarrow f_j \rightsquigarrow \{x_j\} \cup pa_j$

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



4.2.2 From Markov Random Fields to Factor Graphs

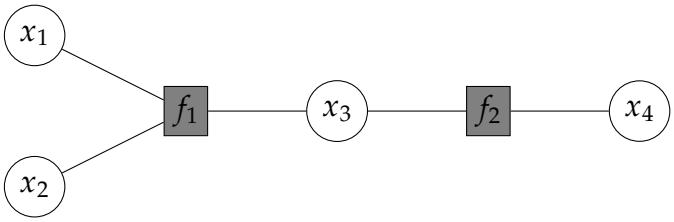
Consider the following Markov Random Field (rectangles correspond to the cliques):



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:

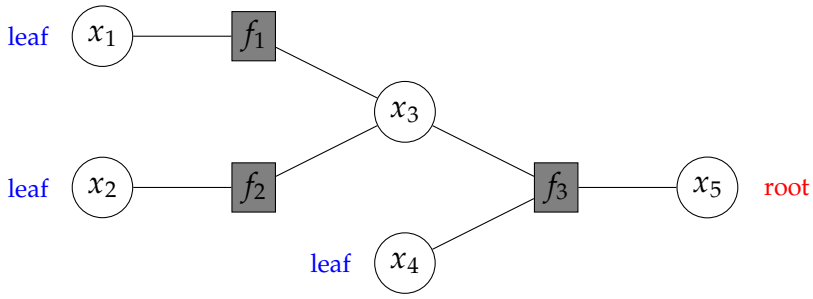


Hence, the conversion from Markov Random Fields to Factor Graphs works as: $\forall c \in \mathcal{C} \rightsquigarrow f_c(x_c) \approx \Psi_c(x_c)$ (equality does not hold because of the normalization constant).

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.

4.3 Sum Product algorithm

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 probability distribution implied by this 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, $\sum_{x_2} f_2(x_2, x_3) \doteq \mu_{f_2 \to x_3}(x_3)$ is a message from factor f_2 to variable x_3.

Their product is instead a message flowing from variable x_3 to factor f_3, denoted by $\mu_{x_3 \to f_3}(x_2, x_3) \doteq \sum_{x_1} f_1(x_1, x_3) \cdot \sum_{x_2} f_2(x_2, x_3)$

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 factors collect via multiplication all the incoming messages from the other factors, except from the one we are sending the message to.

Formalizing, we call:

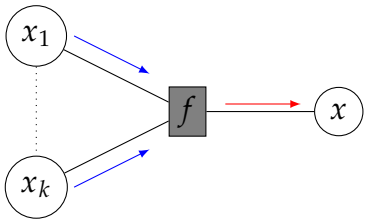
- set of neighboring nodes of variable x:

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

- 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:

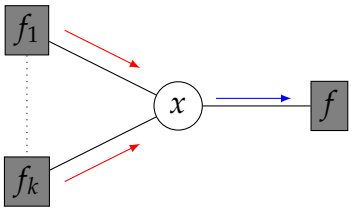


The red arrow in the figure above corresponds to the message $\mu_{f\to x}(x)$. Following what we have seen before, this is computed as:

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

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

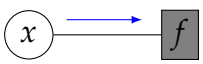
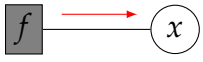
Consider now the symmetric situation:



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

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 $\hat{y}_j$ $\forall y_j \in y$ (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):

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

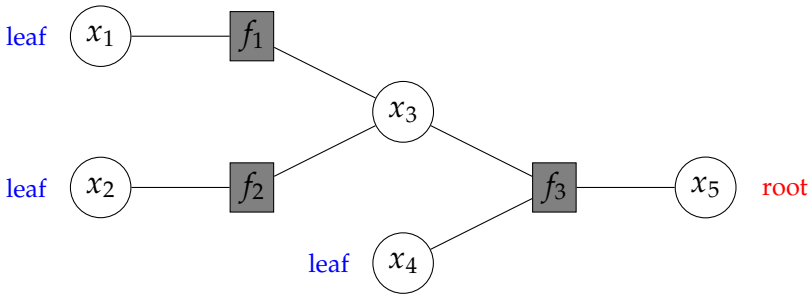
Finally, we compute the desired conditional as:

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

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:

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

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

$$f_3(x_3, x_4, x_5) = \begin{bmatrix} 0.1 \\ 0 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0 \\ 0.2 \\ 0.4 \end{bmatrix} \text{ where } (x_3, x_4, x_5) = \begin{matrix} (0,0,0) \\ (0,0,1) \\ (0,1,0) \\ (0,1,1) \\ (1,0,0) \\ (1,0,1) \\ (1,1,0) \\ (1,1,1) \end{matrix}$$

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

Forward pass

$\mu_{x_1 \to f_1}(x_1) = 1$

$\mu_{x_2 \to f_2}(x_2) = 1$

$\mu_{f_1 \to x_3}(x_3) = \sum_{x_1} f_1(x_1, x_3) \mu_{x_1 \to f_1}(x_1) = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$

$\mu_{f_2 \to x_3}(x_3) = \sum_{x_2} f_2(x_2, x_3) \mu_{x_2 \to f_2}(x_2) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$

$\mu_{x_3 \to f_3}(x_3) = \mu_{f_1 \to x_3}(x_3) \mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix}$

$$\mu_{x_4 \to f_3}(x_4) = 1$$

$$\mu_{f_3 \to x_5}(x_5) = \sum_{x_3, x_4} f_3(x_3, x_4, x_5) \mu_{x_3 \to f_3}(x_3) \mu_{x_4 \to f_3}(x_4) =$$

$$= \begin{bmatrix} 0.12 \cdot (0.1 + 0.1) + 0.42 \cdot (0.1 + 0.2) \\ 0.12 \cdot (0 + 0.1) + 0.42 \cdot (0 + 0.4) \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix}$$

Backward pass

$$\mu_{x_5 \to f_3}(x_5) = 1$$

$$\mu_{f_3 \to x_4}(x_4) = \sum_{x_3, x_5} f_3(x_3, x_4, x_5) \mu_{x_5 \to f_3}(x_5) \mu_{x_3 \to f_3}(x_3) =$$

$$= \begin{bmatrix} 0.12 \cdot (0.1 + 0) + 0.42 \cdot (0.1 + 0) \\ 0.12 \cdot (0.1 + 0.1) + 0.42 \cdot (0.2 + 0.4) \end{bmatrix} = \begin{bmatrix} 0.054 \\ 0.276 \end{bmatrix}$$

$$\mu_{f_3 \to x_3}(x_3) = \sum_{x_4, x_5} f_3(x_3, x_4, x_5) \mu_{x_5 \to f_3}(x_5) \mu_{x_4 \to f_3}(x_4) = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}$$

$$\mu_{x_3 \to f_1}(x_3) = \mu_{f_3 \to x_3}(x_3) \mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.3 \cdot 0.3 \\ 0.7 \cdot 0.7 \end{bmatrix} = \begin{bmatrix} 0.09 \\ 0.49 \end{bmatrix}$$

$$\mu_{x_3 \to f_2}(x_3) = \mu_{f_3 \to x_3}(x_3) \mu_{f_2 \to x_3}(x_3) = \begin{bmatrix} 0.3 \cdot 0.4 \\ 0.7 \cdot 0.6 \end{bmatrix} = \begin{bmatrix} 0.12 \\ 0.42 \end{bmatrix}$$

$$\mu_{f_1 \to x_1}(x_1) = \sum_{x_3} f_1(x_1, x_3) \mu_{x_3 \to f_1}(x_3) = \begin{bmatrix} 0.09 \cdot 0.3 + 0.49 \cdot 0.2 \\ 0.09 \cdot 0.1 + 0.49 \cdot 0.4 \end{bmatrix} = \begin{bmatrix} 0.125 \\ 0.205 \end{bmatrix}$$

$$\mu_{f_2 \to x_2}(x_2) = \sum_{x_3} f_2(x_2, x_3) \mu_{x_3 \to f_2}(x_3) == \begin{bmatrix} 0.12 \cdot 0.1 + 0.42 \cdot 0.5 \\ 0.12 \cdot 0.2 + 0.42 \cdot 0.2 \end{bmatrix} = \begin{bmatrix} 0.222 \\ 0.108 \end{bmatrix}$$

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

$$p(x_5) \propto \mu_{f_3 \to x_5}(x_5) = \begin{bmatrix} 0.15 \\ 0.18 \end{bmatrix} \propto \begin{bmatrix} 0.45 \\ 0.55 \end{bmatrix}$$

$$p(x_3) \propto \mu_{f_1 \to x_3}(x_3) \mu_{f_2 \to x_3}(x_3) \mu_{f_3 \to x_3}(x_3) = \begin{bmatrix} 0.4 \cdot 0.3 \cdot 0.3 \\ 0.6 \cdot 0.7 \cdot 0.7 \end{bmatrix} = \begin{bmatrix} 0.036 \\ 0.294 \end{bmatrix} \propto \begin{bmatrix} 0.11 \\ 0.89 \end{bmatrix}$$

$$p(x_2, x_3) \propto f_2(x_2, x_3) \mu_{x_2 \to f_2}(x_2) \mu_{x_3 \to f_2}(x_3) = \begin{bmatrix} 0.1 \cdot 0.12 \\ 0.5 \cdot 0.42 \\ 0.2 \cdot 0.12 \\ 0.2 \cdot 0.42 \end{bmatrix} = \begin{bmatrix} 0.012 \\ 0.21 \\ 0.024 \\ 0.084 \end{bmatrix} \propto \begin{bmatrix} 0.04 \\ 0.64 \\ 0.07 \\ 0.25 \end{bmatrix}$$

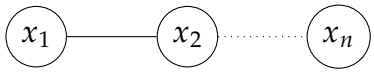
Notice that all marginals were normalized using the same constant $Z = 0.33$.

4.4 Max Plus algorithm

In what follows our task will be, given a joint density $p(x)$, to find the tuple $x^M = \text{argmax}_x\, p(x)$ (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

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

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:

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

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.

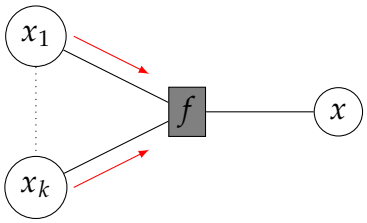
Actually we will maximize $\log p(x)$, hence turning the product into sum of logarithms (and this justifies the name max-plus), i.e. our objective is to maximize

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

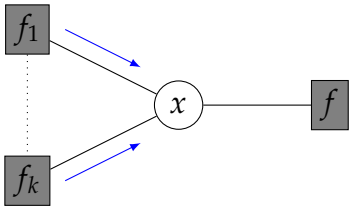
This saves us from product of factors that are possibly very small, since they are probabilities.

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

scenarios:



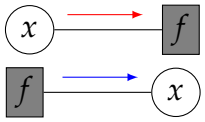
$$\mu_{f\to x}(x) = \max_{x_1,\dots,x_k} \left[\log f(x, x_1, \dots, x_k) + \sum_{x_i \in \mathcal{N}(y)\setminus x} \mu_{x_i \to f}(x_i) \right]$$



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

With base cases:

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



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 this 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:

$$\Phi_{f\to x}(x) = \operatorname*{argmax}_{x_1,\dots,x_k \in \mathcal{N}(f)\setminus x} \left[\log f(x, x_1, \dots, x_k) + \sum_{x_i \in \mathcal{N}(f)\setminus x} \mu_{x_i \to f}(x_i) \right]$$

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

$$x_{root}^{max} = \operatorname*{argmax}_{x_{root}} \left[\sum_{f \in \mathcal{N}(x_{root})} \mu_{f \to x_{root}}(x_{root}) \right]$$

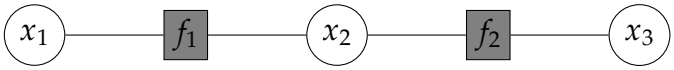
we can exploit the additional information that we stored to retrieve the most probable state of the internal nodes by **back-tracking**, i.e. by following Φ. For example, assuming the variable nodes connected to f

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

$$\Phi_{f \to x_{root}}(x_{root}^{max}) = \begin{Bmatrix} x_1 \to x_1^{max} \\ \dots \\ x_k \to x_k^{max} \end{Bmatrix},$$

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 forward messages edge by edge:

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

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

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

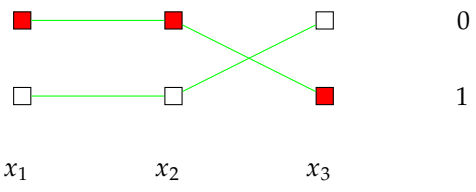
$$\mu_{f_2 \to x_3}(x_3) = \max_{x_2} \left[\log f_2(x_2, x_3) + \mu_{x_2 \to f_2}(x_2) \right] = \begin{bmatrix} \log 0.2 + \log 0.4 \\ \log 0.5 + \log 0.3 \end{bmatrix} \text{ where } x_3 = \begin{matrix} 0 \\ 1 \end{matrix}$$

as well as backtracking functions:

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

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

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:

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

$$\text{argmax}\, x_3 = 1$$

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*.

4.5 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.

Hidden Markov Models 5

5.1 Introduction

In what follows our goal is to model **sequential data** (i.e. **time series**). This kind of data are observed from a process evolving in time, typically at different time steps $x_1, x_2, \ldots, x_N$ (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 the points (up to a certain step N) are observed, the factorization implied by the model is:

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

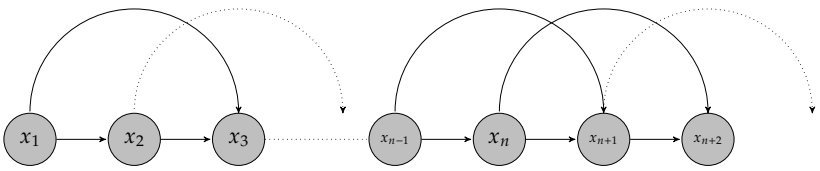
It holds that future observations are independent of all but the most recent observation:

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

A **time homogeneous** process is a process whose transition probability does not change in time, i.e. such that $p(x_n|x_{n-1}) = p(x_2|x_1)$.

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 move 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 joint probability distribution is:

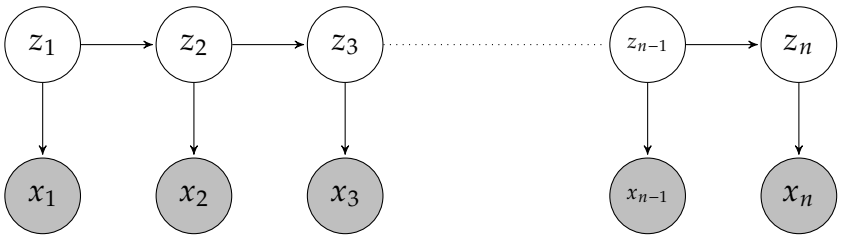
$$p(x_1, \dots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1) \dots p(x_{n+1}|x_n, x_{n-1}) \dots$$

And it holds that:

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

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 by 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 form a Markov chain, and each of them corresponds to an observation:



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

In order to describe state 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 probability** $p(x_n|z_n)$, 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.

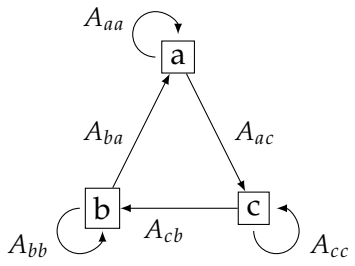
Definition 5.1.1 *The **Hidden Markov Model** is a specific instance of the state space model described before, in which the latent variables z_i are discrete. If instead the variables z_i are continuous and the transition probabilities $p(z_i|z_{i-1})$ are Gaussian, we talk about **Linear Dynamical System**.*

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

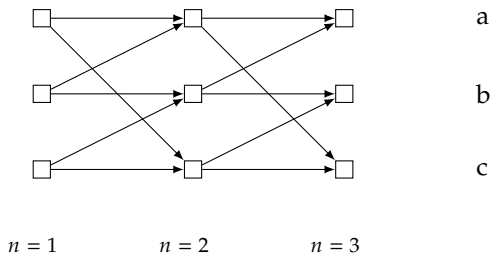
$\underline{x}, \underline{z} = (x_1, \dots, x_n, z_1, \dots, z_n)$, then:

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

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:

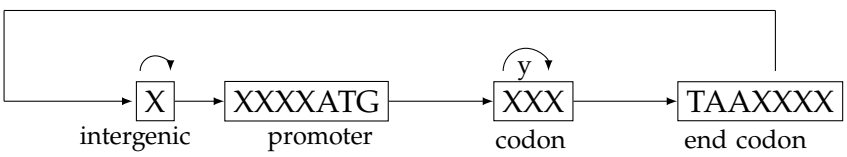


5.1.1 Application: HMM for gene finding

In the field of Computational Biology, a typical application of HMM is that of *Gene Finding*. This task consists in identifying the regions of genomic DNA that encodes genes (DNA belongs indeed to the class of sequential data).

Our goal is, given an observable sequence of bases, find the most likely sequence of internal states that generated it, where the internal state essentially describes in which part of the DNA we are.

In prokaryotic cells, the latent variable state space is roughly the following:



with $x \in \{A, C, G, T\}$, $y \in \{[A, C, G, T]\}^3$.

Without going into full detail, the function of the internal states is:

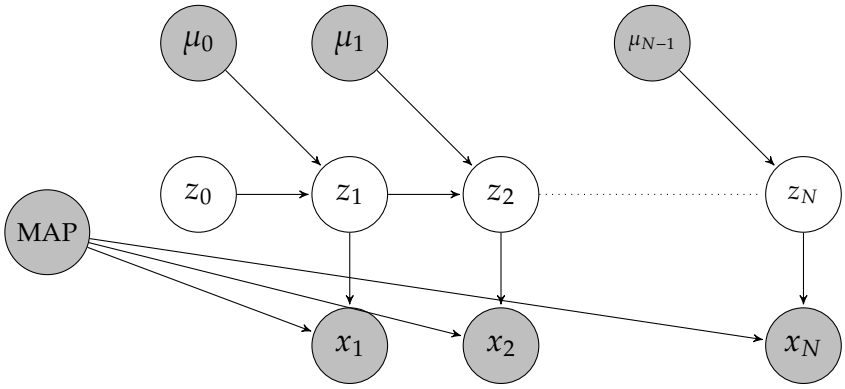
- Intergenic region: stretch of DNA sequences located between genes. It is a subset of noncoding DNA.
- Promoter: region of DNA where the transcription of gene is initiated.

- Codon: trinucleotide sequence of DNA or RNA that corresponds to a specific amino acid. It describes the relationship between the sequence of DNA bases (A, C, G and T) in a gene and the corresponding protein sequence that it encodes.
- End codon: nucleotide triplet that signals the termination of the translation process of the current protein.

5.1.2 Application: HMM for robot localization

Robot Localization is the process of determining where a mobile robot is located w.r.t. its environment. In a typical robot localization scenario, a map of the environment is available and the robot is equipped with sensors that observe the environment as well as monitor its own motion.

Graphically the situation is the following:



where μ_i are the controls we give to the robot and x_i are the readings from the sensors.

Note that this is a tree-like graphical model, so we can convert it into a factor graph and apply sum-product and max-plus algorithms to do inference.

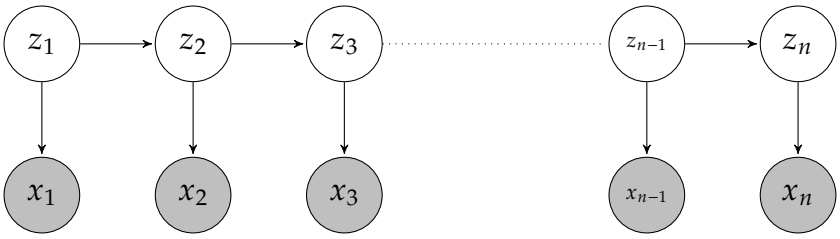
5.2 Inference in HMM

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

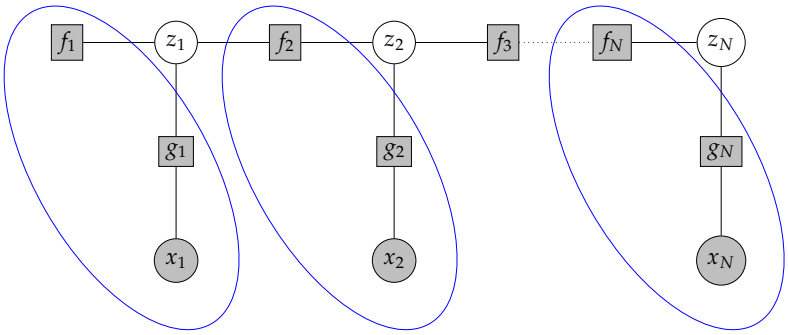
- **Filtering**: given the observations $\underline{x}$, we want to compute the distribution over hidden states of the last latent variable at the end of the sequence, i.e. compute the probability $p(z_n|\underline{x})$.
- **Smoothing**: we want to compute the distribution of a latent variable somewhere 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 $z^\star = \text{argmax}_{\underline{z}}\, p(\underline{z}|\underline{x})$ with $\underline{z} = z_1, \ldots, z_n$.

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).

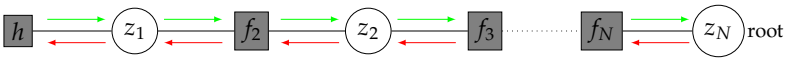
Our goal is now to recast the algorithms for exact inference in PGM (sum-product and max-plus) to the context of HMM. Since both work in Factor Graphs, we need to convert the following Bayesian Network (i.e. the representation we have used so far for HMM) into a Factor Graph:



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 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:

$$p(z_n, z_{n+1}, \underline{x}) = \alpha(z_n) f(z_n, z_{n+1})\beta(z_{n+1})$$

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

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

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

$$\Phi(z_n) = \operatorname*{argmax}_{z_{n-1}}\{\log f_n(z_n, z_{n-1}) + \hat{\mu}_{f_{n-1}\to z_{n-1}}(z_{n-1})\}$$

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

Bayesian Linear Regression 6

In this chapter we will introduce the simplest Bayesian machine learning approach, namely Bayesian linear regression. Under a Gaussian likelihood 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 Distribution

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

6.1.1 Definition

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**

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

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 **Mahalanobis distance**. Notice that having $\Sigma = I$ one obtains the **euclidean distance**.

6.1.2 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, $\mathcal{N}(x|0, I)$. 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.

6.1.3 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:

$$-\frac{1}{2}x^T Ax - b^T x = \frac{1}{2}(x - A^{-1}b)^T A(x - A^{-1}b) - \frac{1}{2}b^T A^{-1}b$$

Therefore we can use the properties of the exponential to get

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

which means that

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

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

6.1.4 Further closure 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 $x \sim \mathcal{N}(\mu_x, \Sigma_x)$, $\eta \sim \mathcal{N}(\mu, \Sigma)$, $x \perp\!\!\!\perp \eta$. Then y is Gaussian, $y \sim \mathcal{N}(M\mu_x + \mu, M\Sigma_x M^T + \Sigma)$. This means that Gaussians are closed under linear transformations.
- **Marginals and conditionals** Assume that

$$z = \begin{bmatrix} x \\ y \end{bmatrix} \qquad z \sim \mathcal{N}(\mu, \Sigma)$$
$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}$$

  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

$$\begin{aligned} S &= \Sigma_1 + \Sigma_2 \\ \mu &= \Sigma_1 S^{-1}\mu_2 + \Sigma_2 S^{-1}\mu_1 \\ \Sigma &= \Sigma_1 S^{-1}\Sigma_2 \\ K &= \frac{\exp\left(-\frac{1}{2}(\mu_1 - \mu_2)^T S^{-1}(\mu_1 - \mu_2)\right)}{\sqrt{\det(2\pi S)}} \end{aligned}$$

- **Bayesian Theorem** Supposing to have

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

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

$$z = \begin{bmatrix} x \\ y \end{bmatrix}$$

In fact

$$\ln p(z) = \ln p(x) + \ln p(y|x) =$$
$$\text{const } -\frac{1}{2}(x - \mu)^T A(x - \mu) - \frac{1}{2}(y - Mx - b)^T L(y - Mx - b)$$

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

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

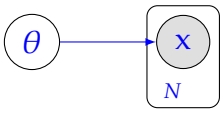
And

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

And then we can apply the marginalization and the conditional distributions 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 $\underline{x} = x_1, \ldots, x_n$ and a family of models $p(x|\theta)$, corresponding to our likelihood distribution, in which we are looking for the model that best fits our data.



In the Bayesian context, we also have a **prior** 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 $p(\underline{x}) = \int p(\underline{x}|\theta)p(\theta)d\theta$ is a hard integral to approximate in a high dimensional setting.

Now, we could compute the maximum a-posteriori, i.e. $\theta_{\text{map}} = \max_\theta p(\theta|\underline{x})$, but in this way, from the estimated parameters, we would 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 $x_1,\dots,x_n \in \{0,1\}$ so that $p(\underline{x}|\theta) = \text{Bernoulli}(\theta)$, and that we observed (1) k times and (0) $n-k$ times. A good choice for our prior in this scenario is

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

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

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

By using the prior 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

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

The predictive distribution is then defined through the following expectation:

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

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

Definition 6.2.1 *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 Introduction to Linear Regression

We will start with an introduction to linear regression. This will serve as a recap of notions that will be expanded in a 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 $i = 1, \dots N$, 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

$$\theta_{ML} = \text{argmax } p(\underline{y}|\underline{x}, \theta)$$

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

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

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

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

In particular, in linear regression, we will have that our function f is linear with respect to our weights w, that is

$$f(x, w) = w_0\phi_0(x) + \dots + w_{M-1}\phi_{M-1}(x)$$

Notice that ϕ 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, ...

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 6.3.1 *$\Phi_{ij} = \phi_j(x_i)$ 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

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

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.

6.3.1 Example

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

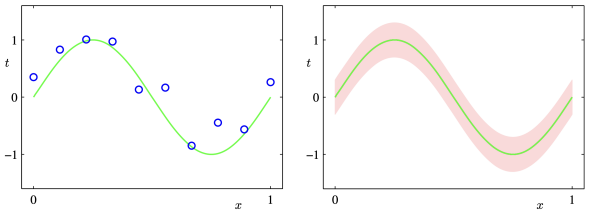


Figure 6.1: On the left, the sinusoidal function and the generated data points. On the right, the true conditional distribution $p(t|x)$ in which the green curve denotes the mean and the shaded region spans one standard deviation on each side of the mean (Bishop)

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.

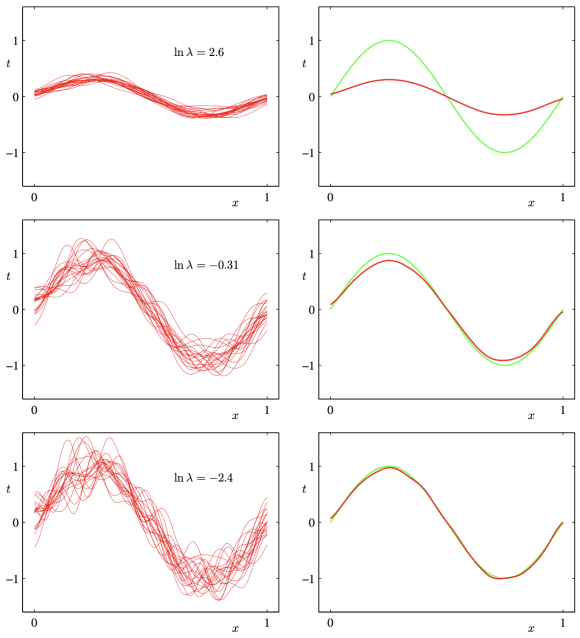


Figure 6.2: On the left, the result of fitting the model to the different data sets varying $ln(\lambda)$. On the right, the average of the fits on the 100 datasets (Bishop)

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?

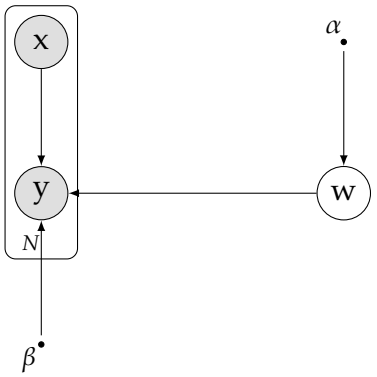
The key point to observe in order to do this step is that the regularization term is, in fact, a **bias** that we introduce in our data. What we can do instead of adding a penalty term in the loss, is to encode the bias in a **prior** distribution of our parameters, $p(w|\alpha)$ and then treat our problem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

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

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 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

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

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

Let's consider the logarithm of the posterior first

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

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.

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

where

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

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 $p(w|m_0, S_0) = \mathcal{N}(w|m_0, S_0)$, then our posterior becomes

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

with

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

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.

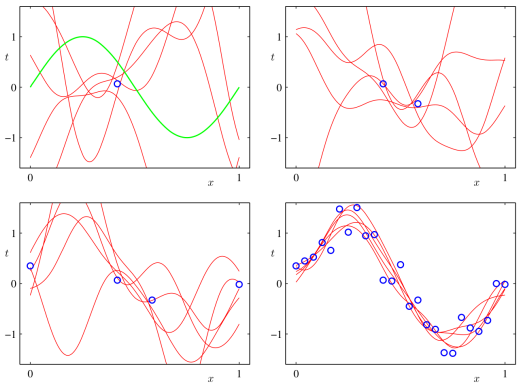


Figure 6.3: Samples from the posterior distribution of a Bayesian regression model on the dataset shown in section 6.3.1. Increasing the size of the dataset, the predictive distribution approximates better the true data distribution (Bishop)

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

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 7.1 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

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 centred on the true parameter values.

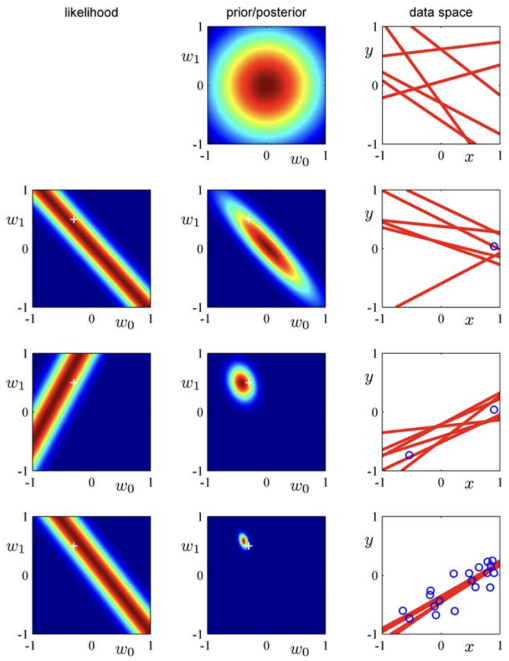


Figure 6.4: Illustration of sequential Bayesian learning for a simple linear model of the form $f(x, \mathbf{w}) = w_0 + w_1 \cdot x$ (Bishop)

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

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

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 probability above is

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

$$\sigma_N^2(x) = \frac{1}{\beta} + \phi^T(x) S_N \phi(x)$$

$$\sigma_N^2(x) \geq \sigma_{N+1}^2(x), \qquad \sigma_N^2 \to \frac{1}{\beta}, N \to \infty$$

The prediction is a Gaussian centered on an average prediction and having a variance which has two terms: one takes into account the

noise of 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.

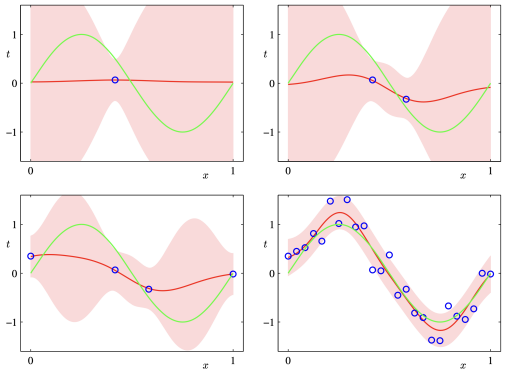
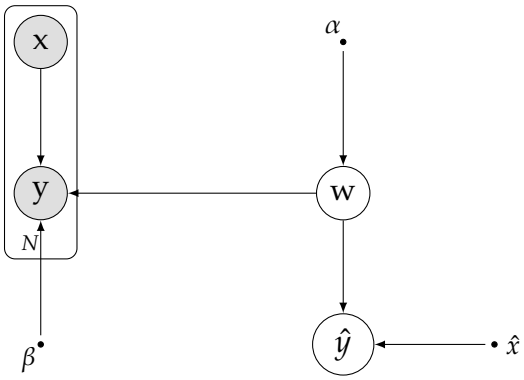


Figure 6.5: Example of the predictive distribution for a Bayesian linear regression model on the dataset shown in section 6.3.1. Increasing the number of data points, the red curve (which represents the mean of the predictive distribution) approximates better the sinusoidal function and the standard deviation (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

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

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 into 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 $p(\alpha, \beta) = \text{const}$
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 approximations means that we can fix α and β with Maximum Likelihood, which means that we can find our best hyper-parameters 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

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

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

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

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

$$||\alpha_{n+1} - \alpha_n|| + ||\beta_{n+1} - \beta n|| < \epsilon$$

Therefore we just need to compute

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

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

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

Moreover

$$\frac{\partial}{\partial\beta}\lambda_i = \frac{\lambda_i}{\beta}$$

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

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

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

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

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

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

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

$$\frac{p(D|\mathbb{M}_1)}{p(D|\mathbb{M}_2)}$$

which is known as the **Bayes Factor** (of $\mathbb{M}_1$ versus $\mathbb{M}_2$). It is basically a ratio 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. $D \sim p(D|M_1)$), 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 distribution $p(D|\mathbb{M}_2)$ 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.

Bayesian Linear Classification 7

7.1 Introduction

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 $y_{nj} \in \{0, 1\}$, $\sum_j y_{nj} = 1$. 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 explicitly the 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 $p(C_k|x)$.

7.2 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:

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

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:

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

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

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

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

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 $E(w) + \alpha w^T w$.

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:

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

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

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

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

Explicitly, this corresponds to:

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

and

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

In this case the cross-entropy is:

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

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

Hence the gradient for class j weights is:

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

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

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

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.

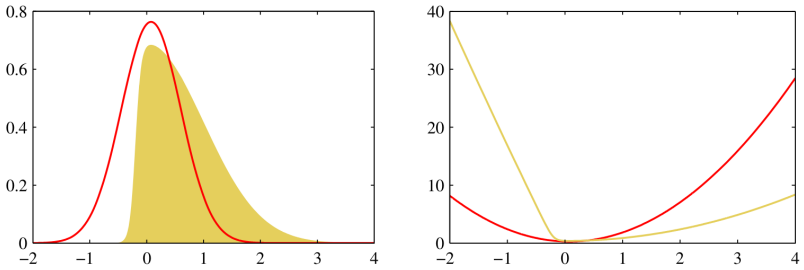


Figure 7.1: Illustration of the Laplace approximation applied to the distribution $p(z) \propto \exp(-z^2/2)\sigma(20z+4)$ where $\sigma(z)$ is the logistic sigmoid function. The left plot shows the normalized distribution $p(z)$ in yellow, together with the Laplace approximation centred on the mode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

7.3.1 Laplace approximation for model comparison

It is possible to use Laplace approximation for the marginal likelihood in 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 model 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.

7.4 Bayesian Logistic Regression

Given observations (x_n, y_n) with $n = 1, \dots, N$, consider a set of basis functions $\phi(x) = \phi_0(x), \dots, \phi_{M-1}(x)$ and the logit activation function $\sigma(w^T \phi(x))$.

To recast logistic regression in a Bayesian 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 $p(\underline{y}|w, \underline{x}) = \prod_{i=1}^{N} s_i^{y_i}(1 - s_i)^{1-y_i}$, being $s_i = s_i(w) = \sigma(w^T \phi(x_i))$.

Hence the log-posterior is now:

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

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 $w_{MAP} = \text{argmax}_w \log p(w|\underline{y}) = \text{argmax}_w \log p(w) + \log p(\underline{y}|w)$ 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):

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

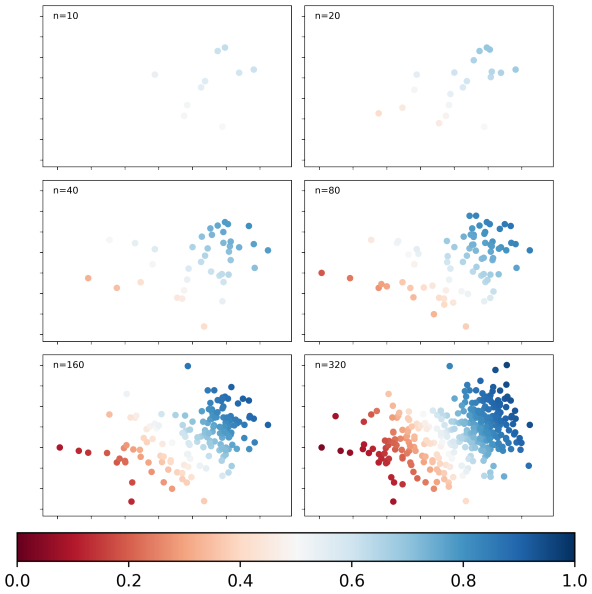


Figure 7.2: These plots show the predictive distribution for an increasing number of points in the training set. The more intense the colour, the more confident is the prediction (close to 0 or 1). As a consequence of using a Bayesian approach, the confidence of predictions increases with the number of observations.

This is in principle an M-dimensional integral. However, σ depends on w only through the 1-dimensional projection $a = w^T\phi(x^\star)$, 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 $\mu_a = w_{MAP}^T\phi(x^\star)$ and $\sigma_a^2 = \phi^T(x^\star)S_N\phi(x^\star)$, so that:

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

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:

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

being $\kappa(\sigma_a^2) = \left(1 + \pi\frac{\sigma_a^2}{8}\right)^{-\frac{1}{2}}$

In this way, the predictive distribution depends from μ_a but it is rescaled by the variance:

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

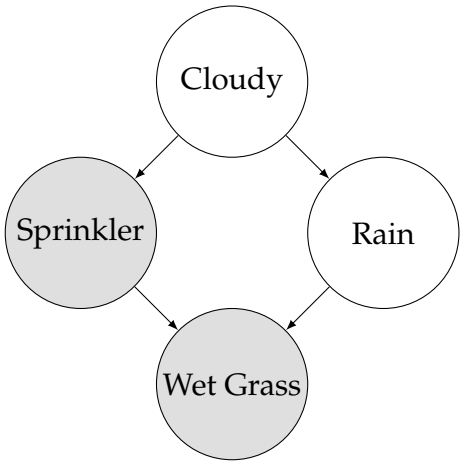
Remark: the dominating cost of this procedure is identifying the MAP.

Sampling-based Inference 8

8.1 Introduction

Our focus in this chapter is on solving inference 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 a simple example like



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

Therefore we need to change our strategy. One possibility is to rely on sampling to perform approximate inference of 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 distribution $\tilde{p}(x)$ such that

$$p(x) = \frac{1}{Z}\tilde{p}(x) \tag{8.1}$$

The typical scenario where this happens is given by Bayes Theorem

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{8.2}$$

where $p(y)$ is hard to compute (can be a 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 this set of samples we are also able to estimate quantities like

$$\mathbb{E}_x[f(x)] = \int f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{8.3}$$

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 tasks and vanilla methods are not scalable to high dimensional probability distributions.

8.2 Approximate sampling

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

8.2.1 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 $\exists M > 0 : Mg(x) \geq p(x), \forall x$, 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 samples $\hat{x}$, 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.

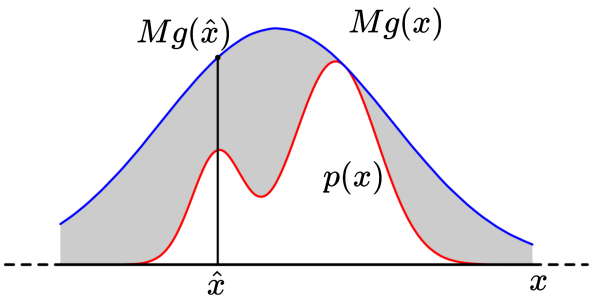


Figure 8.1: In the rejection sampling method, samples are drawn from a simple distribution $g(x)$ and rejected if they fall in the grey area between the distribution $p(x)$ and the scaled distribution $Mg(x)$. The resulting samples are distributed according to $p(x)$. (Bishop)

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,

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

8.2.2 Importance sampling

The goal in importance sampling is to evaluate

$$\mathbb{E}_p[f(x)] = \int f(x)\frac{1}{Z}\tilde{p}(x)dx = \frac{\int f(x)\tilde{p}(x)dx}{\int \tilde{p}(x)dx} \tag{8.4}$$

where f is 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.

$$\frac{\int f(x)\tilde{p}(x)dx}{\int \tilde{p}(x)} = \frac{\int f(x)\tilde{p}(x)\frac{g(x)}{g(x)}}{\int \tilde{p}(x)\frac{g(x)}{g(x)}} = \frac{\mathbb{E}_g[f\frac{\tilde{p}}{g}]}{\mathbb{E}_g[\frac{\tilde{p}}{g}]} \tag{8.5}$$

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

$$\frac{\mathbb{E}_g[f(x)\frac{p}{g}]}{\mathbb{E}_g[\frac{p}{g}]} = \frac{\frac{1}{N}\sum_{i=1}^N f(x_i)w(x_i)}{\frac{1}{N}\sum_{i=1}^N w(x_i)} \tag{8.6}$$

where

$$w(x_i) := \frac{\tilde{p}(x_i)}{g(x_i)} \tag{8.7}$$

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 function.

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.

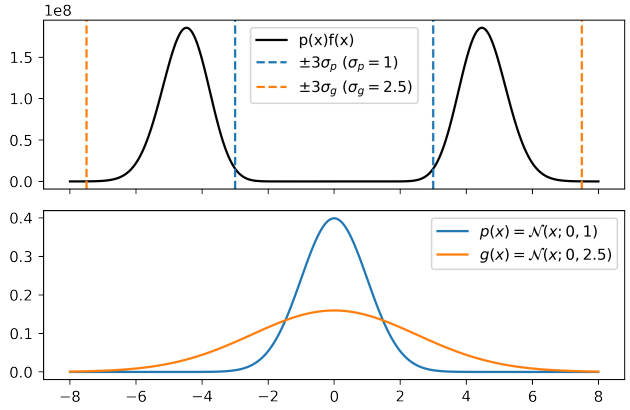


Figure 8.2: Importance sampling example. Sampling from $p(x) = \mathcal{N}(x; 0, 1)$ to compute $\mathbb{E}_p[f(x)]$ with $f(x) = x^{20}$ would be inefficient, as most of the contribution to the integral is given by values of x that are rare ($|x| > 3\sigma_p$). Using as proposal distribution a Gaussian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

8.3 Markov chain

8.3.1 Introduction

A Markov Chain is a stochastic process, denoted as

$$\{X_t\}_{t \geq 0} \tag{8.8}$$

with $t \in \mathbb{N}$, $X_0, X_1, \ldots, X_t \in \mathcal{X}$ 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

$$p(x_n|x_{n-1}, \ldots, x_0) = p(x_n|x_{n-1}) \tag{8.9}$$

which is known as the memoryless or Markov property.

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

$$p(x_n|x_{n-1}) = p(x_1|x_0), \ \forall n \geq 1 \tag{8.10}$$

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

Definition 8.3.1 *$p(x_n|x_{n-1})$ 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 8.3.2 *A Markov Chain is* ***ergodic*** *iff* $\forall x, y \in \mathcal{X}, \exists t \geq 0 : p(x_t = y|x_0 = x) > 0$.

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 8.3.3 *A* ***stationary distribution*** $\Pi(y)$ *is such that*

1. $$\Pi(y) = \int p(y|x)\Pi(x)dx \tag{8.11}$$
   which means that Π is an ***invariant measure*** *with respect to the Markov Chain dynamics defined by 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*

8.3.2 Detailed Balance

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

Definition 8.3.4 *We say that a Markov Chain is* ***reversible*** *(or it satisfies the* ***balance condition****) iff $\exists \Pi(x)$, probability distribution such that*

$$p(x|y)\Pi(y) = p(y|x)\Pi(x) \tag{8.12}$$

Proposition 8.3.1 *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

$$\int p(y|x)\Pi(x)dx = \int p(x|y)\Pi(y)dx = \Pi(y)\int p(x|y)dx = \Pi(y) \tag{8.13}$$

□

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. $\Pi(x)$.

8.4 Markov Chain Monte Carlo

8.4.1 Metropolis Hastings

Intuitively, the idea is that we want to sample from a distribution and we build an ergodic 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 distribution $\tilde{p}(x)$ 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 $q(y|x)$
2. Borrowing from rejection sampling, we are going to accept/reject the new sampled point based on the following rule

$$x_{t+1} = \begin{cases} y, \text{ with probability } \alpha(y|x) = min\left\{1, \frac{\tilde{p}(y)q(x|y)}{\tilde{p}(x)q(y|x)}\right\} \\ x, \text{ otherwise} \end{cases} \tag{8.14}$$

The criterion for defining

$$\alpha(y|x) = min\left\{1, \frac{\tilde{p}(y)q(x|y)}{\tilde{p}(x)q(y|x)}\right\} \tag{8.15}$$

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

Observe that

$$\frac{\tilde{p}(y)}{\tilde{p}(x)} = \frac{p(y)}{p(x)} \tag{8.16}$$

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 by the operational procedure given above.

Lemma 8.4.1 *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

$$p(y|x)p(x) = p(x|y)p(y) \tag{8.17}$$

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

$$p(y|x)p(x) = q(y|x)\alpha(y|x)p(x) = \min\left\{1, \frac{p(y)q(x|y)}{p(x)q(y|x)}\right\} \cdot q(y|x)p(x) \tag{8.18}$$

Notice that

$$\frac{p(y)}{p(x)} = \frac{\tilde{p}(y)}{\tilde{p}(x)} \tag{8.19}$$

Performing some calculations we obtain

$$\min\left\{1, \frac{p(y)q(x|y)}{p(x)q(y|x)}\right\} \cdot q(y|x)p(x) = \min\{q(y|x)p(x), p(y)q(x|y)\} = \tag{8.20}$$

$$\min\left\{\frac{p(x)q(y|x)}{p(y)q(x|y)}, 1\right\} \cdot q(x|y)p(y) = \alpha(x|y)q(x|y)p(y) = p(x|y)p(y) \tag{8.21}$$

□

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.

8.4.2 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 $k \in \{1, \dots, n\}$
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 8.4.2 *The transition kernel of Gibbs Sampling is exactly the same as Metropolis-Hastings algorithm*

Proof. In fact, for Gibbs Sampling we have

$$q_k(y|x) = \begin{cases} p(y_k|x_{-k}), & \text{when } y_{-k} = x_{-k} \\ 0 & \text{otherwise} \end{cases} \tag{8.22}$$

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

$$\alpha_k(y|x) = \frac{p(y)q(x|y)}{p(x)q(y|x)} = \frac{p(y_k|y_{-k})p(y_{-k})}{p(x_k|x_{-k})p(x_{-k})} \cdot \frac{p(x_k|y_{-k})}{p(y_k|x_{-k})} \tag{8.23}$$

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

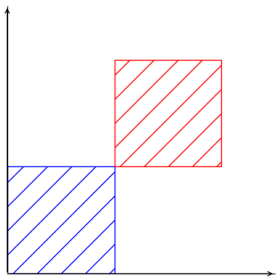
$$\frac{p(y_k|y_{-k})p(y_{-k})}{p(x_k|x_{-k})p(x_{-k})} \cdot \frac{p(x_k|y_{-k})}{p(y_k|x_{-k})} = 1 \tag{8.24}$$

□

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 sample our conditional distribution (the latter strategy is called "Metropolis within Gibbs").

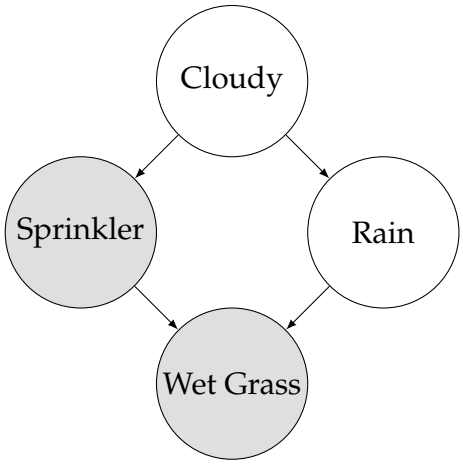
Remark.

- We may not satisfy ergodicity in Gibbs sampling, since it may not always be possible to find a path between two states 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.

8.4.3 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 $Z = p(y = \hat{y})$.

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.

8.4.4 Convergence Diagnostics

How can we check that our 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 $\Psi : \mathcal{X} \to \mathbb{R}$. 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 logarithms 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

$$\begin{cases} \bar{\Psi}_j := \frac{1}{n} \sum_{i=1}^{n} \Psi_{ij} \\ \bar{\Psi} := \frac{1}{m} \sum_{j=1}^{m} \bar{\Psi}_j \end{cases} \tag{8.25}$$

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

$\text{VAR}[\bar{\Psi}] = \frac{1}{n}\text{VAR}[\Psi]$, 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:

$$W \coloneqq \frac{1}{m}\sum_{j=1}^{m} s_j^2, \qquad s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\Psi_{ij} - \bar{\Psi}_j)^2 \tag{8.26}$$

$$B \coloneqq \frac{n}{m-1}\sum_{j=1}^{m}(\bar{\Psi}_j - \bar{\Psi})^2 \tag{8.27}$$

W is called the **within variance** while B is called the **between variance**.

We know by definition that

$$W \leq \text{VAR}[\Psi],$$

because when sampling single trajectories we have not necessarily explored and visited the full space. Increasing the number of samples we will converge to the true variance. Moreover, as long as the initial states are overdispersed, it can be shown that

$$\text{VAR}[\Psi] \leq \text{VAR}^+[\Psi] \coloneqq \frac{n-1}{m}W + \frac{1}{n}B$$

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

$$\hat{R} \coloneqq \sqrt{\frac{\text{VAR}^+[\Psi]}{W}} \tag{8.28}$$

which can be monitor while running the MCMC simulations. Notice that $\hat{R} > 1$ and that $\hat{R} \xrightarrow[n\to\infty]{} 1$. Heuristically, we can say that when $\hat{R} \leq 1.1$ we have converged to our stationary distribution.

8.4.5 Effective sample size

We may want to have some measure of efficiency of our statistics, i.e. we may want to compute $\text{VAR}[\bar{\Psi}]$. Notice the following: if $m \cdot n$ samples are independent then we know that

$$\text{VAR}[\bar{\Psi}] = \frac{\text{VAR}[\Psi]}{n \cdot m} \tag{8.29}$$

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:

$$nm\text{VAR}[\bar{\Psi}] \approx \left(1 + 2\sum_{k=1}^{\infty} \rho_k\right) \text{VAR}[\Psi] \tag{8.30}$$

Where ρ_k is the **autocorrelation** of lag k, which is, by definition

$$\rho_k \coloneqq \text{Corr}[\Psi(x_i), \Psi(x_{i+k})] \tag{8.31}$$

So it is the correlation in between two points in the same chain which are k steps apart.

If we compare the formula above (8.31) with what we would have in case of independence (8.30), we can find the **effective number of samples** produced by our chain

$$n_{\text{eff}} = \frac{nm}{(1 + 2\sum_{k=1}^{\infty} \rho_k)} < nm \tag{8.32}$$

And we can also write

$$\text{VAR}[\bar{\Psi}] = \frac{\text{VAR}[\Psi]}{n_{\text{eff}}} \tag{8.33}$$

A typical rule of thumb here is to reach at least $n_{\text{eff}} = 100$ effective samples.

The question now is: how do we compute the autocorrelation? We know that this identity holds (no proof given)

$$\mathbb{E}[(\Psi_i - \Psi_{i-k}^2)] = 2(1 - \rho_k)\text{VAR}[\Psi] \tag{8.34}$$

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

$$V_k = \frac{1}{m(n-k)} \sum_{j=1}^{m} \sum_{i=k+1}^{n} (\Psi_{i,j} - \Psi_{i-k,j})^2 \tag{8.35}$$

So that

$$\hat{\rho}_k = 1 - \frac{V_k}{2\text{VAR}^+[\Psi]} \tag{8.36}$$

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 in (8.30) when the following condition is satisfied

$$T = \min\{k | k \text{ is odd}, \ \hat{\rho}_{k+1} + \hat{\rho}_{k+2} < 0\} \tag{8.37}$$

When this condition is satisfied we are in a regime where our summation is not relevant any more.

Therefore, we approximate the sum as

$$\sum_{k=1}^{\infty} \rho_k \approx \sum_{k=1}^{T} \hat{\rho}_k \tag{8.38}$$

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.

8.5 Hamiltonian Monte Carlo

Hamiltonian 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 variables* Monte Carlo methods.

The idea is to turn the problem into an Hamiltonian, augmenting the state space with momentum variables 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 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

$$\frac{1}{Z}\tilde{p}(x) = \frac{1}{Z_x}\exp(H_x(x)) \tag{8.39}$$

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 to them the probability distribution $p(y) = \frac{1}{Z_y}exp(H_y(y))$, where is typical to make the assumption

$$H_y(y) = \frac{1}{2}y^T y \tag{8.40}$$

i.e. to consider $p(y)$ a standard Gaussian.

We are going to sample from the joint distribution, exploiting the independence of our variables

$$p(x,y) = p(x)p(y) = \frac{1}{Z_x Z_y}\exp(H_x(x) + H_y(y)) = \frac{1}{Z}\exp(H(x,y)) \tag{8.41}$$

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 trajectory on the probability distribution space accordingly to the equations 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. we sample from $\{-1, 1\}$ uniformly. This makes our problem reversible and provides ergodicity.
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

$$H(x + \Delta x, y + \Delta y) \approx H(x, y) + \nabla_x H_x(x)^T \Delta x + \nabla_y H_y(y)^T \Delta y \quad (8.42)$$

Then, enforcing the condition $H(x', y') = H(x, y)$, we can derive the equations for our movement:

$$\Delta x = \epsilon \nabla_y H_y(y) \quad (8.43)$$

$$\Delta y = -\epsilon \nabla_x H_x(x) \quad (8.44)$$

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 $exp(H(x', y') - H(x, y)) = exp(0) = 1$), the 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.

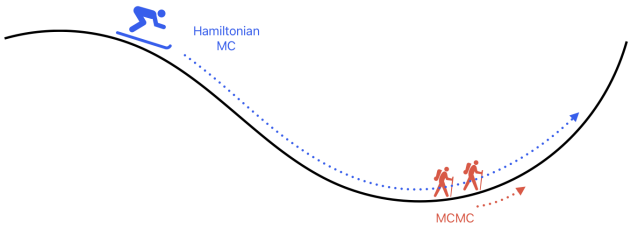
Remark Since we are required to compute gradients, Hamiltonian Monte Carlo works only for continuous variables. The advantage of exploiting gradient information is similar to the one in gradient based optimizers (gradient descent).

Remark Our acceptance criterion is, in fact, Metropolis acceptance criterion

$$\alpha = \min\left\{1, \frac{p(x', y')}{p(x, y)}\right\} = \min\{1, \exp(H(x', y') - H(x, y))\} \quad (8.45)$$

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


Hamiltonian
MC
MCMC

9 Expectation Maximization

9.1 Introduction

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**.

9.1.1 Learning Bayesian Networks

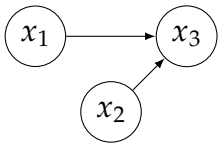
We will start by focusing on a simpler problem to set the scene. Remember that a BN is a way to factorize a joint distribution according to:

$$p(x) = \prod_i p(x_i|pa(x_i)).$$

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 values 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!

9.1.2 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 need to sum over exponentially many possible states, which makes our problem intractable.

The problem becomes even more complex when we have $\underline{x} = x_1, \ldots, x_n$ observations and $\underline{z} = z_1, \ldots, z_n$ latent states of observations since we would like to work with the logarithm of our distribution for numerical stability

$$p(\underline{x}, \underline{z}|\theta) = \Pi_n p(x_n, z_n|\theta) \qquad \log p(\underline{x}, \underline{z}|\theta) = \sum_n \log p(x_n, z_n|\theta)$$

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 $p(x|\theta)$.

9.2 Evidence lower bound

Again, consider a joint model $p(x, z)$ with x observed and z unobserved with corresponding observations $x_1, \ldots, x_n$ and latent states $z_1, \ldots, z_n$. Our goal is to compute $p(x|\theta) = \sum_z p(x, z|\theta)$ and we want to learn θ_{ML}, i.e. θ such that

$$\text{argmax}_\theta \; p(\underline{x}|\theta)$$

This problem is typically intractable for the reasons described before.

We will 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

$$-\sum_z q(z) \log \frac{p(z|x, \theta)}{q(z)} = -\sum_z q(z)\left[\log \frac{p(z|x, \theta)}{q(z)} + \log p(x|\theta) - \log p(x|\theta)\right]$$

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 9.2.1

$$\mathcal{L}(q, \theta) = \sum_z q(z) \log \frac{p(x, z|\theta)}{q(z)} \tag{9.1}$$

*is the **Evidence Lower Bound** (also called **ELBO**).*

From the expression above we can rewrite our log likelihood as

$$\log p(x|\theta) = \mathcal{L}(q, \theta) + KL[q(z)||p(z|x, \theta)] \tag{9.2}$$

Remark Since $KL[q|p] \geq 0$ then $\mathcal{L}(q, \theta) \leq \log(p(x|\theta))$ 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).

9.3 Expectation Maximization

Let's start from this expression for the ELBO

$$\mathcal{L}(q, \theta) = \mathbb{E}_q\left[\log \frac{p(x, z|\theta)}{q(z)}\right] = \mathbb{E}_q[\log p(x, z|\theta)] + \mathbb{E}_q[-\log q(z)]$$

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 $\theta_{ML} = \text{argmax}_\theta \log p(\underline{x}|\theta)$ 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** step (or the e-step), the maximization with respect to θ is known as the **maximization** step (or the m-step).

9.3.1 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 $\mathcal{L}(q, \theta)$ is maximum whenever $KL[q(z)||p(z|\underline{x}, \theta)] = 0$, since $\log p(\underline{x}|\theta)$ does not depend on q and

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

Therefore

$$q_{max} = p(\underline{z}|\underline{x}, \theta_{OLD})$$

where

$$p(\underline{z}|\underline{x}, \theta) = \prod_{i=1}^{N} p(z_i|x_i, \theta) \qquad \text{if } x_1, \dots, x_n \text{ are i.i.d}$$

Then we compute $\mathbb{E}_{q_{new}}\left[\log p(\underline{x}, z|\theta)\right]$ 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 $p(\underline{z}|\underline{x}, \theta_{OLD})$.

9.3.2 M-step

Now we need to maximize $\mathcal{L}(q, \theta)$ keeping q fixed to $q = q_{new} = p(\underline{z}|\underline{x}, \theta_{OLD})$ which is tantamount to maximize w.r.t θ

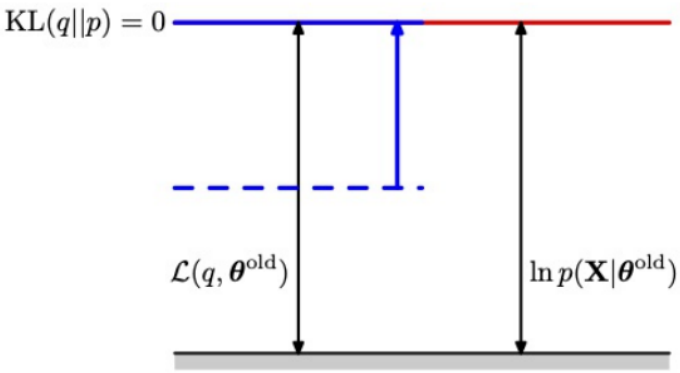
$$\mathbb{E}_{q_{new}}[\log p(z, \underline{x}|\theta)]$$

i.e. the energy term.

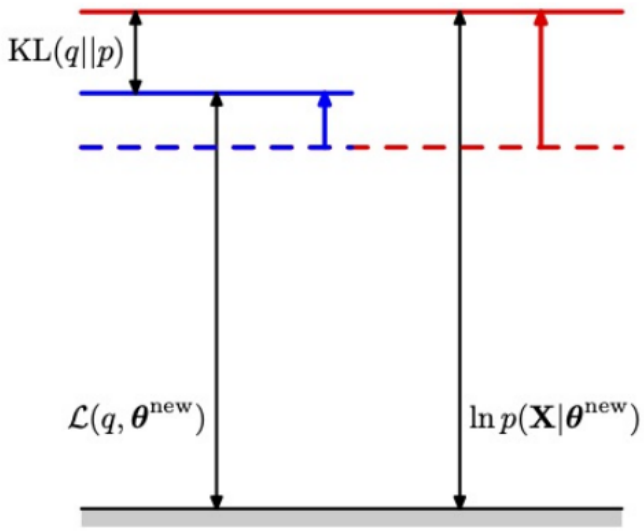
This maximization really depends on the problem at hand, but usually is just about explicitly computing the gradient, setting it to zero, and find

$$\theta_{new} = \text{argmax}_\theta \, \mathbb{E}_{q_{new}}[\log p(\underline{x}, z|\theta)]$$

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: $\mathcal{L}(q_{NEW}, \theta_{NEW}) - \mathcal{L}(q_{OLD}, \theta_{OLD}) < \delta$, where δ is a chosen constant.

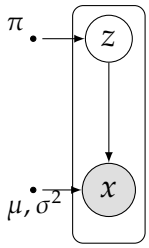
Remark In general the EM algorithm converges to a **local** optimum of $\log p(\underline{x}|\theta)$.

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 iterations will reduce $\text{KL}(q||p)$ but it will not be set to zero, hence convergence is not guaranteed.

9.4 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 for each Gaussian distribution.

Moreover we have N observations of x which means that we also have have N corresponding z unobserved realizations of the variable $\underline{z} = (z_{ij}), \quad i = 1, \dots, N, j = 1, \dots, k$.

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 Gaussian 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.

$$p(z|\theta) = \prod_j \pi_j^{z_j}$$

$$p(z|\underline{x}, \theta) \propto \prod_{n=1}^{N} \prod_{j=1}^{K} \pi_j^{z_{nj}} \mathcal{N}(x_n|\mu_j, \sigma_j^2)^{z_{nj}}$$

$$p(z = j|x, \theta) = \frac{\pi_j \mathcal{N}(x|\mu_j, \sigma_j^2)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x|\mu_i, \sigma_i^2)}$$

where the last probability distribution is considered since we have seen that, given i.i.d. r.v., our conditional distribution factorizes as $p(\underline{z}|\underline{x}, \theta) = \Pi_{i=1}^{N} p(z_i|x_i, \theta)$

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 compute

$$\mathbb{E}_{p(z|x)}[z_{nj}] = p(z = j|x_n, \theta) = \frac{\pi_j \mathcal{N}(x_n|\mu_j, \sigma_j^2)}{\sum_i \pi_i \mathcal{N}(x_n|\mu_i, \sigma_i^2)} := \gamma(z_{nj})$$

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 their maxima. Performing the calculations we find

$$\mu_j^{new} = \frac{1}{N_j} \sum_n \gamma(z_{nj}) x_n$$

$$\Sigma_j^{new} = \frac{1}{N_j} \sum_n \gamma(z_{nj})(x_n - \mu_j^{new})^T (x_n - \mu_j^{new})$$

$$\pi_j^{new} = \frac{N_j}{N}, \qquad N_j = \sum_{n=1}^{N} \gamma(z_{nj})$$

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 running 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 spherical.

9.5 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

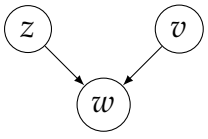
$$\sum_n \mathbb{E}_{q^n} [\log p(v_n, z_n|\theta)] = \sum_n \sum_i \mathbb{E}_{q^n} \left[\log p\left(x_i^n|pa(x_i^n)|\theta_i\right)\right]$$

Then we can optimize

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

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

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

Assume that we have observations $(v_1, w_1), \ldots, (v_n, w_n)$. The e-step for each observations 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 the idea of how to compute it and how to optimize with respect to it

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

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 (slightly more complicated) example is the energy term with respect to w, i.e.

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

let's restrict to the case $z=0, v=1 \quad (\theta_{w01})$

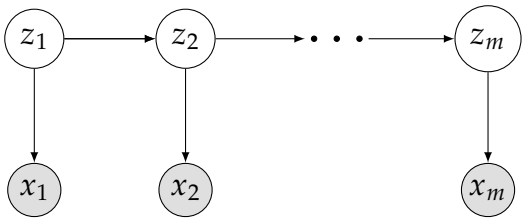
$$\sum_{n:w_n=1, v_n=1} q^n(z=0)\log\theta_{w01} + \sum_{n:w_n=0, v_n=1} q^n(z=0)\log(1-\theta_{w01})$$

Maximizing over this yields

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

9.6 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

$$p(z_1 = i) = \pi_i$$
$$p(z_i = j | z_{i-1} = k) = A_{kj}$$
$$p(x_i | z_i = k) = p(x_i | \phi_k)$$
$$\theta = (\pi, A, \phi)$$

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

$$q^n(z) = p(z | x^n, \theta_{OLD}) \; \forall n$$

and this is done just by running our message passing algorithm for Bayesian Networks. For the m-step instead, we want

$$E(\theta) = \sum_{n=1}^{N} \left[\sum_{k=1}^{K} q^n(z_{1k}) \ln \pi_k + \sum_{i=2}^{M} \sum_{j,k=1}^{K} q^n(z_{(i-1)j}, z_{ik}) \ln A_{jk} \right.$$
$$\left. + \sum_{i=1}^{M} \sum k = 1^K q^n(z_{ik}) \ln p(x_i^n | \phi_k) \right]$$

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

$$\pi_k = \frac{\sum_n q^n(z_{1k})}{\sum_i \sum_n q^n(z_{1i})}.$$

Similarly, we can obtain expressions for the other parameters.

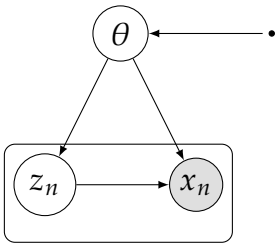
Variational Inference 10

10.1 Introduction

Variational Inference is a deterministic approximation 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, where the 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

$$\log p(\underline{x}) = \mathcal{L}(q) + KL[q(z)||p(z|\underline{x})]$$

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

$$KL[q(z)||p(z|\underline{x})] = -\int q(z) \left[\log p(z|\underline{x}) - \log q(z)\right] dz$$

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 minimizes the KL-divergence is the same as the one maximizing the ELBO $\mathcal{L}(q)$
- $\mathcal{L}(q)$ 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 $\mathcal{L}(q)$ 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. $Q = \{q(z|\lambda), \lambda \in \mathcal{R}^k\}$ (might be for example Gaussian distributions, with λ representing the average and the covariance matrix of our Gaussian). The optimization problem is then on λ, hence we need to find $\text{argmax}_\lambda \mathcal{L}(q(\lambda)) = \text{argmax}_\lambda \mathcal{L}(\lambda)$, but the caveat is that this would typically be a highly non-linear and non-convex optimization. Lastly, notice that λ would typically be composed of parameters for local latent variables and global latent variables, i.e. $q(z|\lambda) = q(\theta|\lambda_\theta) \prod_i q(z_i|\lambda_i, \theta)$

10.2 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 $z = (z_1, \dots, z_M)$ 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 name comes from physics and stochastic processes).

Let's start by writing the lower bound for the mean field approximation

$$\begin{aligned}
\mathcal{L}(q) &= \mathbb{E}\left[\log p(\underline{x}, z) - \log q(z)\right] = \\
&\int \prod_i q_i [\log p(\underline{x}, z) - \sum_i \log q_i] dz = \\
&= \int q_j \left[\int \log p(\underline{x}, z) \prod_{i \neq j} q_i dz_i \right] dz_j - \int q_j \log q_j dz_j + \text{const} \\
&= \int q_j \mathbb{E}_{i \neq j}[\log p(\underline{x}, z)] dz_j - \int q_j \log q_j dz_j + \text{const}
\end{aligned}$$

where on the second row we factored out 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

$$\begin{aligned}
\mathcal{L}(q_j(z_j)) &= \mathbb{E}_{q_j}[\log \tilde{p}(\underline{x}, z_j) - \log q_j] + \text{ const}, \\
\text{hence } \mathcal{L}(q_j(z_j)) &= -KL[q_j || \tilde{p}(\underline{x}, z_j)] + \text{ const}
\end{aligned}$$

Now we have M lower bounds $\mathcal{L}(q_j)$, 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

$$q_j^\star(z_j) = \tilde{p}(\underline{x}, z_j) \tag{10.1}$$

Hence $\log q_j^\star(z_j) = \mathbb{E}_{i \neq j}[\log p(\underline{x}|z)] + \text{ const}$ which implies that

$$q_j^\star(z_j) = \frac{\exp(\mathbb{E}_{i \neq j}[\log p(\underline{x}, z)])}{\int \exp\left(\mathbb{E}_{i \neq j}[\log p(\underline{x}, z)]\right) dz_j}$$

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 expectation 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.

10.2.1 Example on Gaussian distribution

We will use mean field variational inference to go from a Gaussian to a factorized Gaussian. We have a joint distribution $p(z) = \mathcal{N}(z|\mu, \Lambda^{-1})$ 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.

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

This has a nice form, because, since $\log q_1^\star(z_1)$ is a quadratic form, we know that $q_1^\star(z_1)$ is Gaussian $\mathcal{N}(z_1|m_1, \Lambda_{11}^{-1})$ where $m_1 = \mu_1 - \Lambda_{11}^{-1}\Lambda_{12}(\mathbb{E}[z_2] - \mu_2)$.

By symmetry we have that

$$q_2^\star(z_2) = \mathcal{N}(z_2|m_2, \Lambda_{22}^{-1}), \qquad m_2 = \mu_2 - \Lambda_{22}^{-1}\Lambda_{21}(\mathbb{E}[z_1] - \mu_1)$$

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)

10.2.2 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 problem, 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:

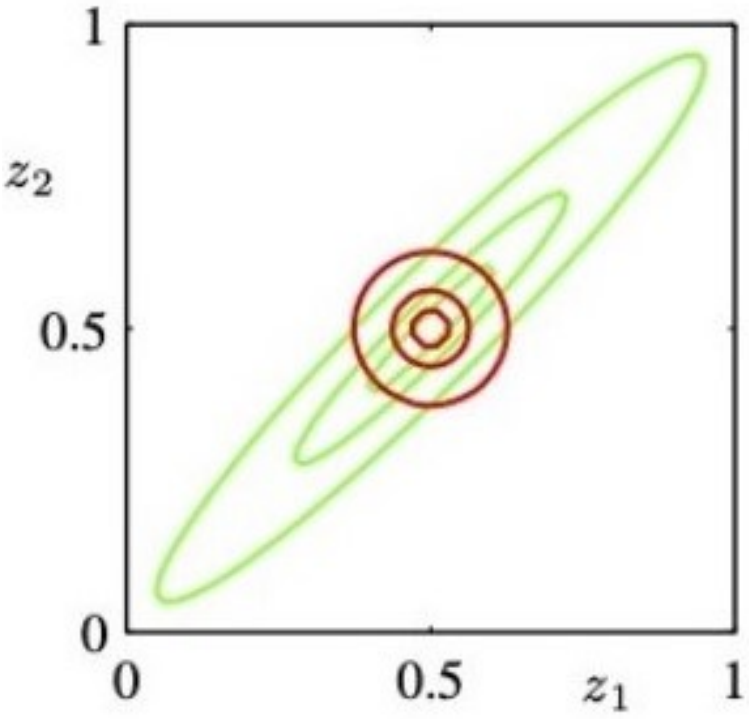
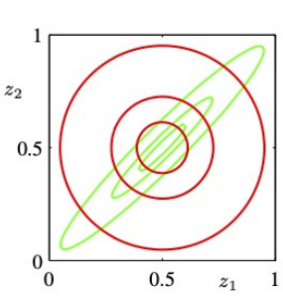
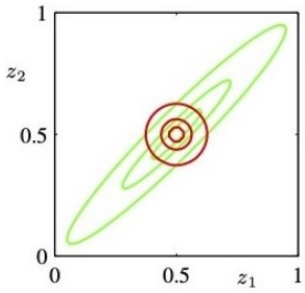


Figure 10.1: Original distribution (green) and its mean field approximation (Red)





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 behaviour of these two approximations. The approximation of the direct KL-divergence 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.

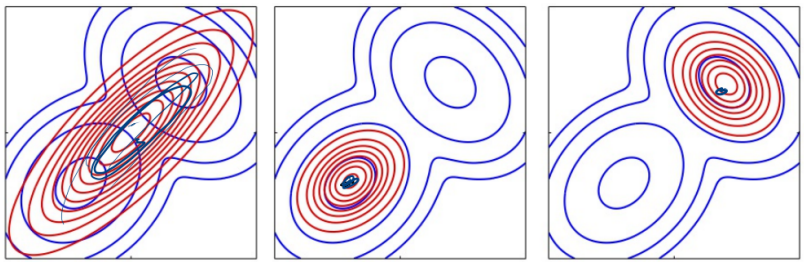
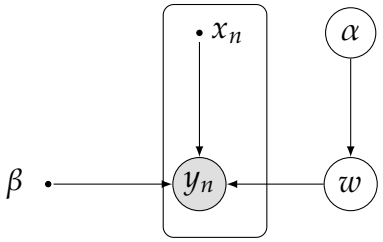


Figure 10.4: On the left, the inverse KL divergence, on the right, the direct KL divergence

10.3 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

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

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

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

Therefore what we have is, in fact, another Gamma distribution

$$
\begin{aligned}
q^\star(\alpha) &= \text{Gamma}(\alpha|a_N, b_N) \\
a_N &= a_0 + \frac{M}{2} \\
b_N &= b_0 + \frac{1}{2}\mathbb{E}_q[w^T w]
\end{aligned}
$$

We can also workout what happens for the other term of the variational distribution

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

Again, notice that here we have a quadratic form, hence completing the square gives us a Gaussian distribution

$$
\begin{aligned}
q^\star(w) &= \mathcal{N}(w|m_N, S_N) \\
m_N &= \beta S_N \Phi^T y \\
S_N &= (\mathbb{E}[\alpha]I + \beta\Phi^T\Phi)^{-1}
\end{aligned}
$$

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:

$$
\begin{aligned}
\mathbb{E}[\alpha] &= \frac{a_N}{b_N} \\
\mathbb{E}[w^T w] &= m_N^T m_N + \text{Trace}(S_N)
\end{aligned}
$$

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 can then be used for Bayesian model comparison.

10.4 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 tipically 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.

10.4.1 Reparameterization trick

The first solution is called "**Reparameterization trick**". It can be used if we can write $z = g_\nu(\varepsilon)$ 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 $\hat{\lambda} = (\lambda, \nu)$ where ν are the parameters of the function g. Then we can write

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

and our expectation does not depend on λ anymore. Hence we can sample from it and compute the gradient:

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

Let's also define the following function for easier readability

$$G(\varepsilon) = [\nabla_{\hat{\lambda}} \log p(x, g_\nu(\varepsilon)) - \nabla_{\hat{\lambda}} q(g_\nu(\varepsilon)|\lambda)]$$

In practice, we sample $\varepsilon_j \sim \hat{q}(\varepsilon)$ and the sampled gradient is just

$$\nabla_{\hat{\lambda}} \mathcal{L}(\hat{\lambda}) = \frac{1}{S} \sum_{s=1}^{S} G(\varepsilon_s)$$

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 reparametrization 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:

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

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:

$$z = g_\nu(\varepsilon) = \mu + \sigma \cdot \varepsilon$$
$$\varepsilon \sim \hat{q}(\varepsilon) = \mathcal{N}(\varepsilon|0,1)$$

10.4.2 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

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

Using the dominated convergence theorem to get the gradient inside the integral and then applying the product rule for derivatives we get

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

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

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

Then we can sample $z_s \sim q(z|\lambda)$ and have an estimate of our gradient

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

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.

10.4.3 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

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

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 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

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

So essentially now we are taking an expectation over a *smaller* set of variables. This is playing the role of $\hat{J}(x)$ 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 Markov blanket of z_i.

10.5 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 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 compute $\mathbb{E}_q[\hat{f}]$. 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 $a^\star$ that minimizes the variance, which by deriving the expression before w.r.t a and setting it to zero turns out to be

$$a^\star = \frac{COV(f, h)}{VAR(h)}$$

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 $a^\star$ for a is hard to compute, because we would need to know the covariance precisely, but we can estimate it as $\hat{a}^\star$ reusing the same samples that we used to estimate the gradient.

Then, our estimation of the gradient using control variates becomes

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

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

A scheme summarizing the different kinds and applications of Variational Inference is shown in Figure 10.5.

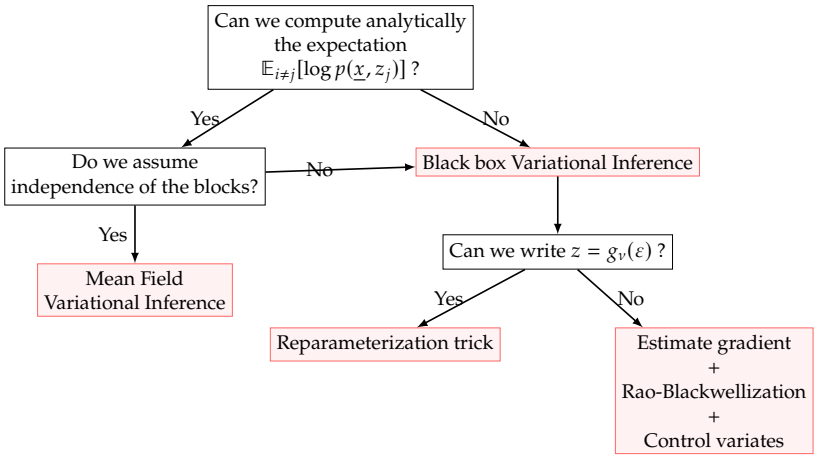


Figure 10.5: A schematic decision diagram to identify the suitable Variational Inference scheme to use in the application at hand.

10.6 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 networks under a Bayesian lens, which for its nature behaves in a probabilistic way. The model uncertainty should be introduced in the parameters $\boldsymbol{w}$. 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 predictions. 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 $p(\boldsymbol{w}$ on weights and then learning the posterior distribution $p(\boldsymbol{w}|\bar{x}, \bar{y})$ with a suitable learning algorithm, using it to compute the predictive distribution:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int_W p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) p(\boldsymbol{w}|\bar{x}, \bar{y}) d\boldsymbol{w}$$

where W is the space of the possible parameters.

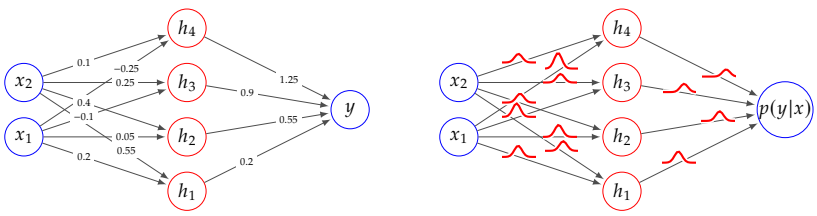


Figure 10.6: Neural network (left) VS Bayesian neural network (right). We consider that the BNN outputs a probability distribution, which is obtained by computing the predictive distribution

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 the 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 $L1$ regularization.

Computing directly $p(\boldsymbol{w}|\bar{x}, \bar{y})$, 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 $q(\boldsymbol{w}|\theta)$ which approximates $p(\boldsymbol{w}|\bar{x}, \bar{y})$. At this point, we write the ELBO:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\log p(\boldsymbol{w}, \bar{x}, \bar{y}) - \log q(\boldsymbol{w}|\theta)] \\ &= \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\log p(\bar{y}|\boldsymbol{w}, \bar{x}) + \log p(\boldsymbol{w}) - \log q(\boldsymbol{w}|\theta)]\end{aligned}$$

and use its opposite as the loss function of the neural network:

$$\text{Loss}(\theta) = \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\log q(\boldsymbol{w}|\theta) - \log p(\bar{y}|\boldsymbol{w}, \bar{x}) - \log p(\boldsymbol{w})]$$

10.6.1 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 reparametrization 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 [8] .

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 $\boldsymbol{\sigma}$ to be non-negative and so we parametrize it as:

$$\boldsymbol{\sigma} = \log(1 + \exp(\boldsymbol{\rho}))$$

In this case,

$$\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\rho})$$

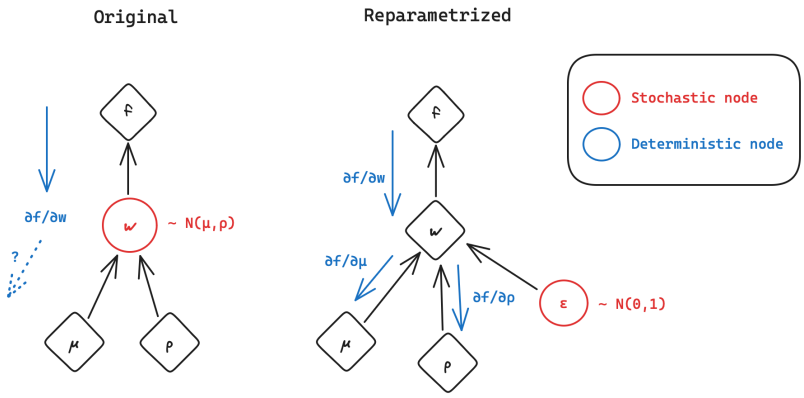
and we can reparametrize $\boldsymbol{w}$ as:

$$\boldsymbol{w} = \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \cdot \varepsilon$$

with $\varepsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

In this way, we are now able to sample $\boldsymbol{w}$ 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

$$\mathbb{E}_{\varepsilon}[f(\boldsymbol{w}, \boldsymbol{\theta})] = \mathbb{E}_{\varepsilon}[\log p(\bar{y}|\bar{x}, \boldsymbol{w}) + \log p(\boldsymbol{w}) - \log q(\boldsymbol{w}|\theta)]$$



We can then perform the optimization step by step by repeating the following procedure:

1. Sample $\varepsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
2. Compute the weights as $\boldsymbol{w} = \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \cdot \varepsilon$
3. Execute the forward pass
4. Compute the unbiased Monte Carlo gradients with respect to the parameters and calculate the update steps (backpropagation):

$$\begin{aligned}
\Delta_{\mu} &= \nabla_{w} f(\boldsymbol{w}, \boldsymbol{\theta}) \cdot \nabla_{\mu} \boldsymbol{w} + \nabla_{\mu} f(\boldsymbol{w}, \boldsymbol{\theta}) \\
&= \nabla_{w} f(\boldsymbol{w}, \boldsymbol{\theta}) + \nabla_{\mu} f(\boldsymbol{w}, \boldsymbol{\theta}) \\
\Delta_{\rho} &= \nabla_{w} f(\boldsymbol{w}, \boldsymbol{\theta}) \cdot \nabla_{\rho} \boldsymbol{w} + \nabla_{\rho} f(\boldsymbol{w}, \boldsymbol{\theta}) \\
&= \nabla_{w} f(\boldsymbol{w}, \boldsymbol{\theta}) \frac{\varepsilon}{1 + \exp(-\boldsymbol{\rho})} + \nabla_{\rho} f(\boldsymbol{w}, \boldsymbol{\theta})
\end{aligned}$$

5. Update the variational parameters:

$$\begin{aligned}
\boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} - \alpha \Delta_{\mu} \\
\boldsymbol{\rho} &\leftarrow \boldsymbol{\rho} - \alpha \Delta_{\sigma}
\end{aligned}$$

where α is the learning rate.

Remark. Notice that $\nabla_{w} f(\boldsymbol{w}, \boldsymbol{\theta})$, 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.

Generative Modelling 11

11.1 Variational Autoencoders

11.1.1 Introduction

Variational Autoencoders (VAE) [9, 10] 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 $p(\boldsymbol{y}|\boldsymbol{x})$, generative techniques aim to obtain the whole distribution of the data $\underline{\boldsymbol{x}} = \boldsymbol{x}_1, ..., \boldsymbol{x}_n$, i.e. estimate $p(\boldsymbol{x})$ with the parametric distribution $p(\boldsymbol{x}, \theta)$, 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 or 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 placed throughout the space.

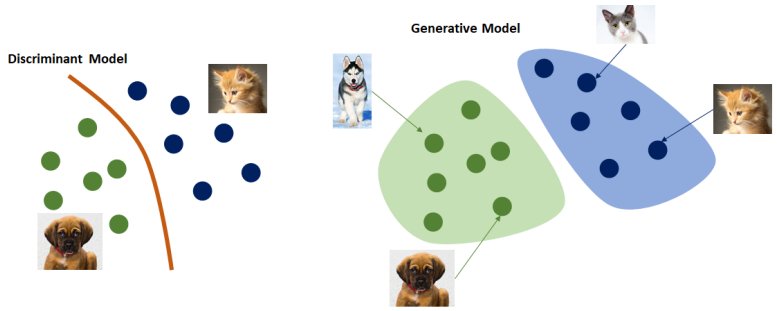


Figure 11.1: Image from *https://medium.com/@jordi299/about-generative-and-discriminative-models-d8958b67ad32*

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:

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

We assume that the posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ 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 goals that we would like to achieve are:

- Learning the parameters θ
- Approximating the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ in order to understand which are the latent features extracted from a given instance $\boldsymbol{x}$
- Performing approximate inference on $\boldsymbol{x}$ 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 us to use Expectation Maximization. But we must remember that the E-step requires being able to evaluate the conditional distribution $p(\boldsymbol{z}|\boldsymbol{x}, \theta_{OLD})$ 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.

11.1.2 Autoencoding Variational Bayes (AEVB [11])

Following the idea of black-box variational inference, we start by considering a parametric variational distribution for the latent variables $q(\boldsymbol{z}|\phi)$. We would like to optimize the ELBO jointly on θ and ϕ by stochastic gradient ascent and so we need to compute $\nabla_{\theta,\phi}\mathcal{L}(\phi, \theta)$. We fix the prior distribution of $\boldsymbol{z}$ as a standard Gaussian:

$$p(\boldsymbol{z}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

and we start by rewriting the lower bound as:

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

As we said before, we are not able to work with the entire dataset and thus we have to consider a mini-batch $\boldsymbol{x}_1, ..., \boldsymbol{x}_m$ and approximate:

$$\mathcal{L}(\phi, \theta) \approx \frac{1}{m}\sum_{j=1}^{m} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x}_j,\phi)}[\log p(\boldsymbol{x}_j|\boldsymbol{z}, \theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x}_j, \phi)||p(\boldsymbol{z})]$$

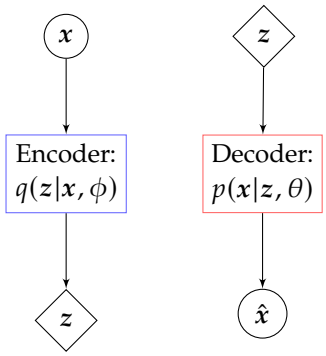
We can try to understand the meaning of this expression:

- The first term is the reconstruction error. It considers how well we are reconstructing $\boldsymbol{x}$ given $\boldsymbol{z}$ sampled from q. This is maximized when $p(\boldsymbol{x}|\boldsymbol{z}, \theta)$ assigns the highest probability to the original instance $\boldsymbol{x}$. We call $p(\boldsymbol{x}|\boldsymbol{z}, \theta)$ **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 $q(\boldsymbol{z}|\boldsymbol{x}, \phi)$ **encoder**.

We can represent the architecture in the following way:



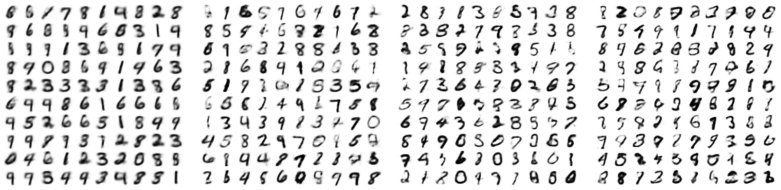
where the input $\boldsymbol{x}$ is mapped through the encoder into a useful latent space $\boldsymbol{z}$ from which we can reconstruct $\hat{\boldsymbol{x}}$ via the decoder. We would like $\hat{\boldsymbol{x}}$ to be as similar as possible to $\boldsymbol{x}$.

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 $q(\boldsymbol{z}|\boldsymbol{x}, \phi)$ through a deterministic transformation $g(\boldsymbol{\varepsilon}, \boldsymbol{x}, \phi)$ which maps a noise variable $\boldsymbol{\varepsilon}$ (distributed accordingly to a distribution we can easily sample from, as $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$) to the distribution of $\boldsymbol{z}$:

$$\boldsymbol{z} = g(\boldsymbol{\varepsilon}, \boldsymbol{x}, \phi)$$



(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 [11].

Typically, the variational distribution is chosen to be a Gaussian:

$$q(\boldsymbol{z}|\boldsymbol{x},\phi) = \mathcal{N}(\boldsymbol{\mu}_z(\boldsymbol{x},\phi), {\sigma_z}^2(\boldsymbol{x},\phi)\cdot\boldsymbol{I})$$

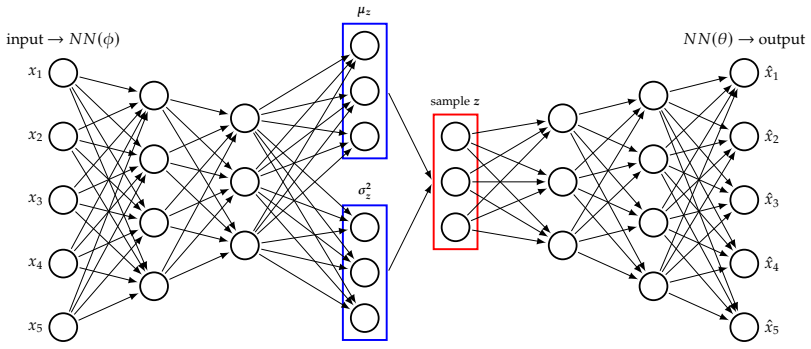
so that we have an analytical expression for $KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$ and we can rewrite $\boldsymbol{z}$ as

$$\boldsymbol{z} = \boldsymbol{\mu}_z(\boldsymbol{x},\phi) + {\sigma_z}^2(\boldsymbol{x},\phi)\cdot\boldsymbol{\varepsilon}$$

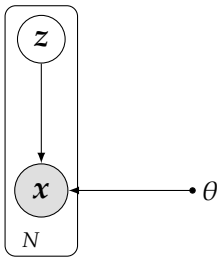
At this point, we can evaluate the gradient of the ELBO as:

$$\nabla_{\theta,\phi}\mathcal{L}(\phi,\theta) = \mathbb{E}_{\boldsymbol{\varepsilon}}[\nabla_{\phi,\theta}\log p(\boldsymbol{x}|g(\boldsymbol{\varepsilon},\boldsymbol{x},\phi),\theta)] - \nabla_\phi KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$$

We still have to choose the functions $\boldsymbol{\mu}_z(\boldsymbol{x},\phi)$ and ${\sigma_z}^2(\boldsymbol{x},\phi)$. 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.

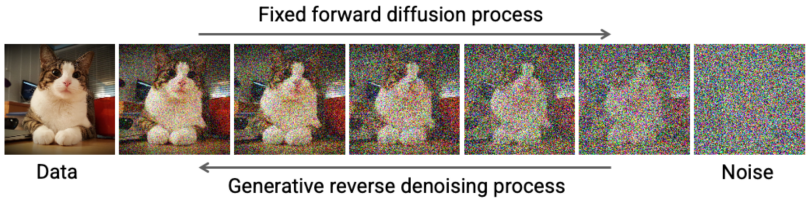
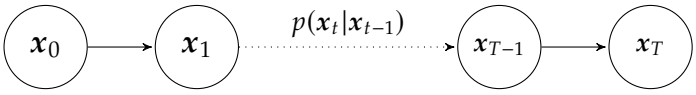


Figure 11.3: Image from [12], *Denoising Diffusion-based Generative Modeling: Foundations and Applications*

11.2.1 Forward Diffusion Process

Given a data point $\boldsymbol{x}_0 \in \mathcal{X}$ sampled from the data distribution $p(\boldsymbol{x}_0)$, consider a Markov chain $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ with $t \in 1, \ldots, T$ satisfying the following properties:

- It is easy to sample from $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$
- For T large enough, $p(\boldsymbol{x}_T)$ is approximately a known distribution from which it is easy to sample and that does not depend on $\boldsymbol{x}_0$, i.e, $p(\boldsymbol{x}_T) \approx p(\boldsymbol{x}_T|\boldsymbol{x}_0)$. This distribution is referred as the **prior**.



Such Markov chain is referred as the **forward diffusion process**, and $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ 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 $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T$ into:

$$p(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T) = p(\boldsymbol{x}_0) \prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$$

In practice, the most used forward process in continuous space is based on the injection of Gaussian noise:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) := \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I})$$

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 $\alpha_t \coloneqq 1 - \beta_t$, $\bar{\alpha}_t \coloneqq \prod_{i=1}^{t} \alpha_i$, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ we have

$$\begin{aligned}
\boldsymbol{x}_t &= \sqrt{\alpha_t}\boldsymbol{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-2}\right) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\hat{\boldsymbol{\epsilon}}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\hat{\boldsymbol{\epsilon}}
\end{aligned}$$

where $\boldsymbol{\epsilon}_{t-2}$ and $\boldsymbol{\epsilon}_{t-1}$ have been merged into $\hat{\boldsymbol{\epsilon}}_{t-2}$.

Hence, we have

$$p(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, 1 - \bar{\alpha}_t)$$

This is also useful to show, as $\bar{\alpha}_t \to 0$, that

$$\lim_{t\to\infty} p(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \mathbf{0}, \boldsymbol{I})$$

So for a sufficiently large T, $\boldsymbol{x}_T \dot{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. In other words, this process ends up in corrupting the data into white noise.

11.2.2 Backward Process

Now that we are able to gradually corrupt data points $\boldsymbol{x}_0$ into noise, for generating new data samples we have to revert this process. We have seen that we can easily sample from $p(\boldsymbol{x}_T)$, but sampling from

$$p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T) = p(\boldsymbol{x}_T)\prod_{t=1}^{T} p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$$

is non-trivial, since we do not have access to the reverse transition kernels $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$* that would allow us to perform ancestral sampling as in the forward process. Therefore, the reverse transition kernels have to be learned. In order to do this, we fit parametric distributions $q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, from which it easy to sample and that can be easily computed, for example:

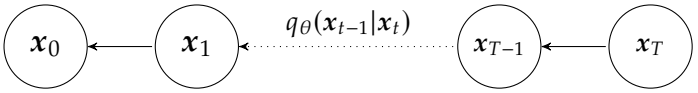
$$q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \coloneqq \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t))$$

where the mean and covariance functions are neural networks.

The objective is then to find parameters θ such that

$$q_\theta(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T) \coloneqq p(\boldsymbol{x}_T)\prod_{t=1}^{T} q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \approx p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)$$

* It is easy to prove that the reverse of a Markov chain is also a Markov chain.



The parametric kernels are learned by minimizing the Kullback-Leibler divergence between the forward and backward processes:

$$\begin{aligned} D_{KL}(p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)||q_\theta(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)) &= \mathbb{E}_{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\left[\log \frac{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}{q_\theta(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\right] \\ &= -\mathbb{E}_{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\left[\log q_\theta(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)\right] + \text{const} \\ &= -\mathbb{E}_{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\left[\sum_{t=0}^{T} \log q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\right] + \text{const} \end{aligned}$$

Remark. The forward process $p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)$ 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 samples 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.

11.2.3 Denoising parametrization

There is empirical evidence that learning directly the backward transition kernels $q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ is not the most effective strategy. Alternatively, it is possible to write the backward transition kernel in the following way:

$$p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \int_{\boldsymbol{x}_0\in\mathcal{X}} p(\boldsymbol{x}_{t-1},\boldsymbol{x}_0|\boldsymbol{x}_t)d\boldsymbol{x}_0 = \int_{\boldsymbol{x}_0\in\mathcal{X}} p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)p(\boldsymbol{x}_0|\boldsymbol{x}_t)d\boldsymbol{x}_0$$

This means that we can sample $\boldsymbol{x}_{t-1}$ if we can sample $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0|\boldsymbol{x}_t)$ and plug it into $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)$. It is possible to show that for the Gaussian diffusion process introduced above the distribution $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)$ is also a Gaussian and can be computed analytically:

$$p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_{t-1};\mu_t(\boldsymbol{x}_t,\boldsymbol{x}_0),\sigma_t\boldsymbol{I})$$

$$\mu_t(\boldsymbol{x}_t,\boldsymbol{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0$$

$$\sigma_t = \frac{(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\beta_t$$

Instead $p(\boldsymbol{x}_0|\boldsymbol{x}_t)$ (the denoising kernel) has to be learned, again by fitting a parametric distribution $q_\theta(\boldsymbol{x}_0|\boldsymbol{x}_t)$. Interestingly, this works despite $p(\boldsymbol{x}_0|\boldsymbol{x}_t)$ being highly multimodal and non-Gaussian, so only a "mean" approximation is possible.

However, it is common in DDPMs to avoid predicting directly $\boldsymbol{x}_0$ given $\boldsymbol{x}_t$. In fact, a model is trained to predict the noise that was added to $\boldsymbol{x}_0$ in order to obtain $\boldsymbol{x}_t$. Recall the property following from $p(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, 1-\bar{\alpha}_t)$:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t \text{ with } \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

We can rewrite the mean of $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$, or $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{\epsilon}_t)$, as

$$\mu_t(\boldsymbol{x}_t, \boldsymbol{\epsilon}_t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$$

Now we can fit a parametric model $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ to approximate $p(\boldsymbol{\epsilon}_t|\boldsymbol{x}_t)$. This is usually done in a simplified way by minimizing the squared error:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\|^2\right] \\ &= \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t\right)\right\|^2\right]\end{aligned}$$

where the time step t is sampled uniformly in $\{1, \ldots, T\}$, $\boldsymbol{x}_0$ is randomly sampled from the dataset, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. 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: The training and sampling algorithms in DDPMs (image from [13]).

Remark. The empirical advantage of predicting the noise $\boldsymbol{\epsilon}_t$ instead of $\boldsymbol{x}_0$ 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.

11.2.4 Diffusion in discrete space

Despite being originally thought for data in continuous space, it is also possible to define a diffusion process in discrete space [14], even though its usefulness is questionable and subject of current research. Assuming without loss of generality data $\boldsymbol{x} \in \{1, 2, \dots, c\}^d$, there are two main kinds of discrete diffusion processes:

- **Uniform**: At each time step t, every component x_t^i of $\boldsymbol{x}_t$ 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 $\boldsymbol{x}_t$ switches to a particular value called the "mask", often referred as the [MASK] token. The prior distribution of such diffusion process is the single point $[\text{MASK}]^d$, 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.

11.2.5 Score-based diffusion models

Given the objective to sample from the distribution $p(\boldsymbol{x})$ that generated the data, score-based models models aim at estimating the so called *score* of $p(\boldsymbol{x})$, defined as $\nabla_x \log p(\boldsymbol{x})$ and then use sampling techniques that exploit the knowledge of the score of the distribution.

An example of score-based model is diffusion with stochastic differential equations (SDE), a generalization in continuous time of the Markov chain diffusion process discussed above, where the transition kernel $p(\boldsymbol{x}_{t+\epsilon}|\boldsymbol{x}_t)$ is implicitly defined through the following SDE:

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}$$

where $\mathrm{d}\boldsymbol{w} \sim \mathcal{N}(0, \boldsymbol{I}\mathrm{d}t)$ is a Wiener process.

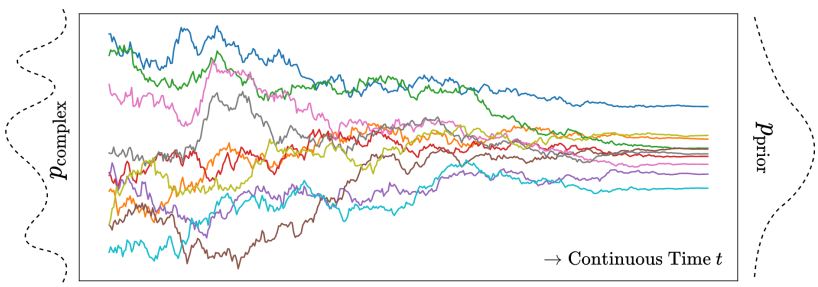


Figure 11.5: Image from [16], *An introduction to Diffusion Probabilistic Models*.

Generation of new data points reduces to solving the inverse-time SDE, that it can be proven to be:

$$\mathrm{d}\boldsymbol{x} = \left[\boldsymbol{f}(\boldsymbol{x}, t) - g^2(t)\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})\right] \mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}$$

For this we need to estimate the time dependent score $\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$, for example using a neural network $s_\theta(\boldsymbol{x}, t)$. The most popular method is *denoising score matching*, where the following loss in minimized:

$$\mathbb{E}_{t\sim u(0,1),\boldsymbol{x}_0\sim p(\boldsymbol{x}_0),\boldsymbol{x}_t\sim p(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left\|\boldsymbol{s}_\theta(\boldsymbol{x}_t, t) - \nabla_{\boldsymbol{x}_t} \ln p(\boldsymbol{x}_t \mid \boldsymbol{x}_0)\right\|$$

that is deeply connected with the denoising loss of DDPMs.

Kernels and Gaussian Processes* 12

We will introduce here kernels and apply the concept of duality to Bayesian regression. In this setting we will use kernels inside probabilistic models for regression and classification tasks, and this will lead us to the framework of Gaussian Processes, which are one of the most flexible approaches for supervised learning, especially when we have a limited number of observations.

12.1 Equivalent Kernel

Consider a predictive distribution on a Bayesian Linear Regression model

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

Let's focus our attention on the mean predictor, by rewriting the expression as

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

Here we can observe that the predictive mean is a linear combination of the observations that we have weighted by a term, which is defined, in our case, as:

$$k(x, x') = \beta\phi^T(x)S_N\phi(x')$$

and it is called the **equivalent kernel**.

It acts as a sort of smoother that depends on the distance between x and x_n.

If we evaluate the covariance in between

$$cov[y(x), y(x')] = \phi^T(x)S_N\phi(x') = \beta^{-1}k(x, x')$$

Notice also that we can rewrite the equivalent kernel as

$$k(x, x') = \Psi^T(x)\Psi(x) \qquad \Psi(x) = \beta^{\frac{1}{2}}S_N^{\frac{1}{2}}\phi(x)$$

I.e. we can see the equivalent kernel as a rescaled scalar product in between our basis functions.

12.2 Dual Formulation of Linear Regression

Let's restate our problem. We have observations $\underline{x}, \underline{y} = (x_n, y_N)_{n\in 1,\dots,N}$ and a linear model $w^T\phi(X)$. Whenever we optimize the weights, we are evaluating our basis function on the input points. Which means that the

solution that we are searching is contained in the subspace spanned by $< \phi(x_1), \ldots, \phi(x_n) >$. Which means that we can rewrite our vector of weight as

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

So basically we can use variables a instead of variables w. Variables a are known as **dual variables**. Now let's introduce a kernel in between two observations points $k(x_i, x_j) = \phi^T(x_i)\phi(x_j)$ and the corresponding **Gram Matrix**, i.e.

$$K = (k_{ij}) \qquad k_{ij} = k(x_i, x_j) \qquad K^i \, i^{th} \text{column}$$

which means that we have

$$w^T \phi(x_i) = a^T K^i$$

If we write the regularized problem that we are trying to solve, then we can write

$$E_D(a) + \lambda E_w(a) = \sum_{n=1}^{N} (y_n - a^T K^n)^2 + \lambda a^T K a$$

We still have a convex optimization problem, which means that we can compute the maximum likelihood solution, which is

$$\hat{a} = (k + \lambda I)^{-1} \underline{y} \qquad \text{ML solution}$$

whereas the prediction on a new point $x*$ would be

$$y(x*) = K_*^T (K + \lambda I)^{-1} \underline{y}$$
$$K_* = (k(x_*, x_1), \ldots, k(x_*, x_n))$$

This new formulation is what is known as the **Kernel trick**: we don't need to know the basis function, we just need their scalar product (which can allow us also to work with infinite dimensional function spaces). Moreover, the kernel trick formulation has a complexity of the order $O(N^3)$, therefore it depends on the number of observations, whilst the basis function formulation depends on the number of basis functions considered $O(M^3)$.

12.3 Random Functions

A **Random Function** can be thought of as a probability distribution over functions. More formally:

Definition 12.3.1 *A random function (or stochastic process) is an infinite collection of random variables, indexed by the argument of the function.*

In order to have an intuition of what this means, consider a vector $\mathbf{x} \in \mathbb{R}^n$ and pick a function $f(\mathbf{x}) = w_0\phi_0(\mathbf{x}) + \ldots + w_{M-1}\phi_{M-1}(\mathbf{x})$ (all $\phi_i(\mathbf{x})$ are fixed).

Consider the set $\mathcal{F} = \{f(\mathbf{x}) = w_0\phi_0(\mathbf{x}) + \ldots + w_{M-1}\phi_{M-1}(\mathbf{x})\}$, if all ϕ_i $i \in 0, \ldots, M-1$ are orthogonal, it can be easily proved that $\mathcal{F}$ is a M-dimensional vector space, where $\Phi(\mathbf{x}) = (\phi_j(\mathbf{x}))$ is the set of basis functions.

We want to consider a distribution over the function space $\mathcal{F}$: observing that each function in $\mathcal{F}$ is identified by its vector of coefficients $\mathbf{w}$, then a distribution over $\mathcal{F}$ is equivalent to a distribution over $\mathbf{w}$.

Under certain conditions (i.e. when $\mathcal{F}$ is a Polish space) the distribution over $\mathcal{F}$ is determined by **finite dimensional projections**.

That is, a distribution is practically described by knowing: $p(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N))$, $\forall \mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^n$.

Consider a Gaussian prior over the coefficients $\mathbf{w} \sim \mathcal{N}(0, \mathbb{I})$.

If we fix a single point $\mathbf{x} \in \mathbb{R}^n$, i.e. we look at the 1-dim marginal $p(f(\mathbf{x}))$, since $f(\mathbf{x}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}) \in \mathbb{R}$ with $w_j \sim \mathcal{N}(0,1)$, then $f(\mathbf{x}) \sim \mathcal{N}(0, \phi^T\phi)$ by the properties of the Gaussian distribution.

Consider now 2-dim marginals, that is take $f \in \mathcal{F}$ and fix two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$, our goal is to study $p(f(\mathbf{x}_1), f(\mathbf{x}_2))$, i.e. find its mean and covariance matrix (notice that $f(\mathbf{x}_1), f(\mathbf{x}_2)$ are linear combinations of Gaussians, so this is again a Gaussian distribution). It turns out that:

$$p(f(\mathbf{x}_1), f(\mathbf{x}_2)) = \mathcal{N}\left(0, \begin{pmatrix}\Phi(\mathbf{x}_1)\\ \Phi(\mathbf{x}_2)\end{pmatrix}^T \mathbb{I} \begin{pmatrix}\Phi(\mathbf{x}_1)\\ \Phi(\mathbf{x}_2)\end{pmatrix}\right)$$

That is:

$$cov(f(\mathbf{x}_1), f(\mathbf{x}_2)) = \begin{pmatrix}\Phi^T(\mathbf{x}_1)\Phi(\mathbf{x}_1) & \Phi^T(\mathbf{x}_1)\Phi(\mathbf{x}_2)\\ \Phi^T(\mathbf{x}_2)\Phi(\mathbf{x}_1) & \Phi^T(\mathbf{x}_2)\Phi(\mathbf{x}_2)\end{pmatrix} := \begin{pmatrix}k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2)\\ k(\mathbf{x}_1,\mathbf{x}_2) & k(\mathbf{x}_2,\mathbf{x}_2)\end{pmatrix}$$

again using properties of the multivariate Gaussian distribution. Note that entries of the covariance matrix are the evaluations of the kernel in each pair of points.

Analogously, the n-dimensional distribution $p(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$ is going to be $\mathcal{N}(0, K)$, with $K = (K_{ij})$ and $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ (i.e. the kernel is the scalar product of the basis functions).

Thus, in the example that we have considered, all finite dimensional projections are Gaussian distributions, and this will be a key point in the definition of Gaussian Processes.

12.4 Gaussian Processes - basic definitions

Definition 12.4.1 *A **Gaussian Process** (GP) is a stochastic process indexed by a continuous variable $\boldsymbol{x} \in \mathbb{R}^n$, such that all finite dimensional distributions*

are multivariate Gaussian.

So a GP f is essentially a collection of random variables $f(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^n$ (i.e. uncountably many r.v.), and we are interested in the joint distribution of these random variables.

By definition: $\forall \mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^n, p(f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)) = \mathcal{N}(m_N, \Sigma_N)$

Hence a GP is uniquely defined by a function $\mu : \mathbb{R}^n \to \mathbb{R}$ which computes the mean at every point $\mathbf{x}$, and by a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ which computes the covariance between any two points.

Thus, exploiting the definition: $f \sim GP(\mu, K)$ if and only if:

$$\forall \mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^n, \; p(\mathbf{f}) = p(f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)) = \mathcal{N}(\underline{\mu}, K)$$

being the mean $\underline{\mu} = (\mu(\mathbf{x}_1), \dots, \mu(\mathbf{x}_n))$, and the covariance matrix (also called **Gram matrix**) $K = (K_{ij})$ (with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$).

Of course K has to be symmetric and positive definite $\forall \mathbf{x}_1, \dots, \mathbf{x}_N$.

In what follows our goal will be to do inference. It is important to stress from the beginning that GP are non-parametric models (only fixed hyperparameters enter in the computations), and this makes them a very powerful method.

12.5 GP Regression

12.5.1 Noise-free case

We start by considering regression in the noise-free case, i.e. we assume to observe exactly the values we want to predict.

Suppose to observe the exact value of the GP at points $\mathbf{x} = x_1, \dots, x_N$, with observations $\mathbf{f} = f(x_1), \dots, f(x_N)$.

Consider also test points $\mathbf{x}^\star = x_1^\star, \dots, x_v^\star$, with function values $\mathbf{f}^\star = f(x_1^\star), \dots, f(x_v^\star)$ unobserved, i.e. to be estimated.

The prior is a $f \sim GP(0, K)$. Inference in this case is particularly simple: it all boils down to (multivariate) Gaussian properties.

First of all we can compute the joint prior distribution of $\mathbf{f}$ and $\mathbf{f}^\star$, that is:

$$p\begin{pmatrix}\mathbf{f} \\ \mathbf{f}^\star\end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(\mathbf{x}, \mathbf{x}) & K(\mathbf{x}, \mathbf{x}^\star) \\ K(\mathbf{x}^\star, \mathbf{x}) & K(\mathbf{x}^\star, \mathbf{x}^\star)\end{bmatrix}\right)$$

where the covariance is a block matrix, where each block is a Gram matrix, with $K(\mathbf{x}, \mathbf{x}^\star) = (K_{ij})_{i \in 1,\dots,N} j \in 1, \dots, \mu$ with $K_{ij} = k(x_i, x_j^\star)$.

If f is observed then we can compute the posterior (note that we are conditioning a Gaussian distribution):

$$p(\mathbf{f}^\star|\mathbf{f}) \sim \mathcal{N}\left(K(\mathbf{x}^\star,\mathbf{x})K(\mathbf{x},\mathbf{x})^{-1}\mathbf{f}, K(\mathbf{x}^\star,\mathbf{x}^\star) - K(\mathbf{x}^\star,\mathbf{x})K(\mathbf{x},\mathbf{x})^{-1}K(\mathbf{x},\mathbf{x}^\star)\right)$$

Since this works for any tuple $\mathbf{x}^\star$, the posterior is a GP.

Remark: the mean of the posterior is not constant anymore.

12.5.2 Noisy setting

If we introduce noise, then we observe $\mathbf{y}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ (i.e. we have a component depending on the process and a noise term). Hence observations are Gaussian with 0 mean and covariance $cov(\mathbf{y}) = K(\mathbf{x},\mathbf{x}) + \sigma^2\mathbb{I}$.

So the joint prior between observations $\mathbf{y}$ and test points is:

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

Conditioning on observations $\mathbf{y}$ we get the posterior (which is again GP):

$$p(\mathbf{f}^\star|\mathbf{y}) \sim \mathcal{N}\left(K(\mathbf{x}^\star,\mathbf{x})\left[K(\mathbf{x},\mathbf{x})+\sigma^2\mathbb{I}\right]^{-1}\mathbf{y}, K(\mathbf{x}^\star,\mathbf{x}^\star)-K(\mathbf{x}^\star,\mathbf{x})\left[K(\mathbf{x},\mathbf{x})+\sigma^2\mathbb{I}\right]^{-1}K(\mathbf{x},\mathbf{x}^\star)\right)$$

The dominant cost of this procedure is computing the inverse $\left[K(\mathbf{x}^\star,\mathbf{x}) + \sigma^2\mathbb{I}\right]^{-1}$, which is an $N \times N$ matrix, hence complexity is $O(N^3)$ (it depends on the amount of observations).

If we want to predict $f(\mathbf{x}^\star)$ for a single point $\mathbf{x}^\star$, it holds that:

$$f(\mathbf{x}^\star) \sim \mathcal{N}(\mathbf{K}(K + \sigma^2\mathbb{I})^{-1}\mathbf{y}, K(\mathbf{x},\mathbf{x}) - \mathbf{K}^T(K + \sigma^2\mathbb{I})^{-1}\mathbf{K})$$

being $\mathbf{K} = (K(\mathbf{x}^\star,\mathbf{x})) = (k(\mathbf{x}^\star, x_1), \ldots, k(\mathbf{x}^\star, x_N))$.

Note that the mean can be rewritten as $\mathbb{E}[f(\mathbf{x}^\star)|\mathbf{y}] = \sum_{i=1}^N \alpha_i k(\mathbf{x}^\star, x_i)$ with $\alpha_i = [K(\mathbf{x},\mathbf{x}) + \sigma^2\mathbb{I}]^{-1}\mathbf{y}$, i.e. as a linear combination of the kernels evaluated on the input points.

12.6 Kernel functions and Hilbert spaces

We can see a kernel as defining an integral operator T_k on a space $\mathcal{X}$ with measure μ:

$$(T_k f)(\mathbf{x}) = \int_{\mathbf{x}} k(\mathbf{x},\mathbf{y}) f(\mathbf{y}) d\mu(\mathbf{y})$$

Note that this is a function of $\mathbf{x}$.

Kernels have the property of being positive semidefinite if they satisfy the following:

$$\int_{\mathfrak{X}\times\mathfrak{X}} k(\mathbf{x},\mathbf{y})f(\mathbf{x})f(\mathbf{y})d\mu(\mathbf{x})d\mu(\mathbf{y}) \geq 0 \; \forall f \in L_2(\mathfrak{X},\mu)$$

or equivalently if the Gram matrix $K = (K_{ij})$ evaluated on any finite collection of points $\{\mathbf{x}_i, i \in 1,\dots,n\}$ is positive (semi)definite.

An eigenfunction ϕ for kernel k with eigenvalue λ is defined as:

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

There can be ∞-many eigenfunctions, they can be ordered w.r.t. decreasing eigenvalues (in analogy with what happens when working with matrices) and they can be chosen orthogonal (i.e. such that $\int \phi_i(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = \delta_{ij}$).

It holds the following theorem, that describes how kernels can be decomposed according to eigenfunctions:

Theorem 12.6.1 *(Mercer's theorem)*
Let $(\mathfrak{X},\mu)$ be a finite measure space and $k \in L_\infty(\mathfrak{X}^2,\mu^2)$ be a kernel such that $T_k : L_2(\mathfrak{X},\mu) \to L_2(\mathfrak{X},\mu)$ is positive definite. Let $\phi_i \in L_2(\mathfrak{X},\mu)$ be the normalized eigenfunctions of T_k associated with the eigenvalues $\lambda_i > 0$. Then:

1. *the eigenvalues $\{\lambda_i\}_{i=1}^\infty$ are absolutely summable;*
2. $$k(\boldsymbol{x},\boldsymbol{x}') = \sum_{i=1}^{\infty} \lambda_i\phi_i(\boldsymbol{x})\phi_i^\star(\boldsymbol{x}')$$
   holds μ^2 almost everywhere, where the series converges absolutely and uniformly μ^2 almost everywhere.

This brings us to the concept of **Reproducing Kernel Hilbert Space** (RKHS):

Definition 12.6.1 *Let $\mathcal{H}$ be a Hilbert space of real functions f defined on an index set $\mathfrak{X}$. Then $\mathfrak{X}$ is called a Reproducing Kernel Hilbert Space endowed with an inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ if there exists a function $k : \mathfrak{X}\times\mathfrak{X} \to \mathbb{R}$ with the following properties:*

1. *for every $\boldsymbol{x}$, $k(\boldsymbol{x},\boldsymbol{x}')$ as a function of $\boldsymbol{x}'$ belongs to $\mathcal{H}$;*
2. *k has the reproducing property $\langle f(\cdot), k(\cdot,\boldsymbol{x})\rangle_{\mathcal{H}} = f(\boldsymbol{x})$*

Note that from the previous definition it follows that as $k(\mathbf{x},\cdot)$ and $k(\mathbf{x}',\cdot)$ are in $\mathcal{H}$, then $\langle k(\mathbf{x},\cdot), k(\mathbf{x}',\cdot)\rangle_{\mathcal{H}} = k(\mathbf{x},\mathbf{x}')$.

Moreover, the following theorem states that the RKHS uniquely determines k and vice versa:

Theorem 12.6.2 *(Moore-Aronszajn theorem)*
Let $\mathcal{X}$ be an index set. Then for every positive definite function $k(\cdot,\cdot)$ on $\mathcal{X}\times\mathcal{X}$ there exists a unique RKHS, and vice versa.

If we take a RKHS and compute the eigenfunctions of a kernel, then we can use them to represent the elements of the RKHS as a linear combination of the eigenfunctions ϕ_j of k : $f(\mathbf{x}) = \sum_j f_j\phi_j(\mathbf{x})$, with coefficients satisfying the smoothness constraint $\sum_j \frac{f_j^2}{\lambda_j} < \infty$.

Moreover it is possible to show that such functions define an Hilbert space $\mathcal{H}$ with inner product $\langle f, g\rangle_{\mathcal{H}} = \sum_j \frac{f_j g_j}{\lambda_j}$.

This Hilbert space is the RKHS corresponding to the kernel k in which:

$$\langle f(\cdot), k(\cdot,\mathbf{x})\rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{f_i\lambda_i\phi_i(\mathbf{x})}{\lambda_i} = f(\mathbf{x})$$
$$\langle k(\mathbf{x},\cdot), k(\mathbf{x}',\cdot)\rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{f_i\lambda_i\phi_i(\mathbf{x})\lambda_i\phi_i(\mathbf{x}')}{\lambda_i} = k(\mathbf{x},\mathbf{x}')$$

Furthermore, since $||\cdot||_{\mathcal{H}} = \sqrt{\langle\cdot,\cdot\rangle_{\mathcal{H}}}$, the norm of $k(\mathbf{x},\cdot)$ is $k(\mathbf{x},\mathbf{x}) < \infty$ and it belongs to $\mathcal{H}$.

12.6.1 Some examples of kernel functions

There are three important classes of kernel functions, which can be distinguished by looking at the dependence of the kernel $k(\mathbf{x},\mathbf{y})$ on $\mathbf{x}$ and $\mathbf{y}$:

- **Stationary kernel**: it is a function of $\mathbf{x}-\mathbf{y}$ (i.e. invariant to translations);
- **Isotropic kernel**: it is a function of $||\mathbf{x}-\mathbf{y}||$ (i.e. invariant to rigid motions);
- **Dot-product kernel**: it is a function of $\mathbf{x}^T\mathbf{y}$ (i.e. invariant to rotations w.r.t. the origin)

Moreover we can define the notion of **continuity in mean square** of a GP f at $\mathbf{x}$:

Definition 12.6.2 *A GP f is continuous in mean square at $\mathbf{x}$ if, for each $\mathbf{x}_k \to \mathbf{x}$, it holds that $\mathbb{E}[|f(\mathbf{x}_k) - f(\mathbf{x})||^2] \to 0$ (i.e. the convergence of the sequence of inputs is preserved in mean).*

It can be proved that a process is continuous in mean square at $\mathbf{x}$ if and only if the kernel k is continuous at $k(\mathbf{x},\mathbf{x})$. For stationary and isotropic kernels, k must be continuous at 0 (i.e. we only need to check continuity in a single point).

It holds also that if k is $2h$-th differentiable, then f is h-th differentiable in mean square (in particular, if the kernel is smooth, then also the process

is smooth).

The **Gaussian kernel** (or Squared Exponential kernel) is an isotropic kernel defined as:

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

where α is called the *amplitude* and λ is the characteristic *length-scale*. λ is a crucial parameters, which regulates the speed of decay of the correlation between points. The RKHS of this kernel is enjoys the universality property (i.e. it is dense) in the space of continuous functions over a compact set in $\mathbb{R}^n$.

A generalization of this kernel is the **Automatic Relevance Detection Gaussian Kernel**:

$$k(\mathbf{x}, \mathbf{y}) = \alpha \exp\left[- \sum_i \frac{|x_i - y_i|^2}{\lambda_i^2} \right]$$

where there is a different λ_i for each coordinate of the input points.

The **Matérn kernel** is another isotropic kernel defined as:

$$k_{matern}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu} r}{\ell} \right)$$

with $\nu, \ell > 0$ are parameters and K_ν is a modified Bessel function.

The parameter μ regulates the differentiability of the kernel: if $\nu > h$ then the GP with Matérn kernel is h times differentiable (in mean square). For $\nu \to \infty$, the Matérn kernel becomes the Gaussian kernel.

A typical choice for MK is $\nu = p + \frac{1}{2}$:

$$k_{\nu=p+\frac{1}{2}}(r) = \exp\left(- \frac{\sqrt{2\nu} r}{\ell} \right) \frac{\Gamma(p+1)}{\Gamma(2p+1)} \sum_{i=0}^{p} \frac{(p+i)!}{i!(p-i)!} \left(\frac{\sqrt{8\nu} r}{\ell} \right)^{p-i}$$

For $\nu = \frac{1}{2}$ we get the Exponential Kernel:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left[\frac{||\mathbf{x} - \mathbf{y}||}{\lambda} \right]$$

which is continuous but nowhere differentiable.

For what concerns dot-product kernels, we have the **Polynomial kernel**, with $p \in \mathbb{Z}$:

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

which corresponds to a kernel obtained by a set of polynomial basis

functions:

$$k(\mathbf{x},\mathbf{x}') = (\mathbf{x}\cdot\mathbf{x}')^p = \left(\sum_{d=1}^{D} x_d x'_d\right)^p = \left(\sum_{d_1=1}^{D} x_{d_1} x'_{d_1}\right)\ldots\left(\sum_{d_p=1}^{D} x_{d_p} x'_{d_p}\right) =$$
$$= \sum_{d_1=1}^{D}\cdots\sum_{d_p=1}^{D}(x_{d_1}\ldots x_{d_p})(x'_{d_1}\ldots x'_{d_p}) := \phi(\mathbf{x})\phi(\mathbf{x}')$$

Where the basis functions ϕ_m are given by all the monomials of degree p:

$$\phi_m(\mathbf{x}) = \sqrt{\frac{p!}{m_1!\ldots m_D!}} x_1^{m_1}\ldots x_D^{m_D}$$

with $\sum_j m_j = p$.

The **Rational Quadratic kernel** is defined by:

$$k_{QR}(r) = \left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha}$$

with $\alpha, \ell > 0$, which essentially is a scale mixture (an infinite sum) of squared exponential functions with different characteristic length-scales.

The **Periodic Kernel** (which allows us to embed periodic effects in the description of our process) is given by:

$$k(x,x') = \exp\left(-\frac{2\sin^2(\frac{x-x'}{2})}{\ell^2}\right)$$

where the 1-dimensional input variable x is mapped to the 2-dimensional $\mathbf{u}(x) = (\cos(x), \sin(x))$ to give rise to a periodic random function of x.

There are rules for constructing complex kernels starting from simpler ones. Given valid kernels $k_1(\mathbf{x},\mathbf{x}'), k_2(\mathbf{x},\mathbf{x}')$ the following kernels will also be valid:

$k(\mathbf{x},\mathbf{x}') = c\cdot k_1(\mathbf{x},\mathbf{x}')$ with $c > 0$
$k(\mathbf{x},\mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}')$ with $f(\cdot)$ any function
$k(\mathbf{x},\mathbf{x}') = q(k_1(\mathbf{x},\mathbf{x}'))$ with $q(\cdot)$ polynomial with non-negative coefficients
$k(\mathbf{x},\mathbf{x}') = \exp(k_1(\mathbf{x},\mathbf{x}'))$
$k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}')$
$k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}')k_2(\mathbf{x},\mathbf{x}')$
$k(\mathbf{x},\mathbf{x}') = k_3(\phi(\mathbf{x}),\phi(\mathbf{x}'))$ with $\phi(\mathbf{x}) \in \mathbb{R}^M$, k_3 valid kernel
$k(\mathbf{x},\mathbf{x}') = \mathbf{x}^T A\mathbf{x}'$ with A symmetric poisitve semidefinite
$k(\mathbf{x},\mathbf{x}') = k_a(\mathbf{x}_a,\mathbf{x}'_a) + k_b(\mathbf{x}_b,\mathbf{x}'_b)$ with $\mathbf{x}_a, \mathbf{x}_b$ vars. s.t. $\mathbf{x} = (\mathbf{x}_a,\mathbf{x}_b)$ and k_a, k_b valid kernels
$k(\mathbf{x},\mathbf{x}') = k_a(\mathbf{x}_a,\mathbf{x}'_a)k_b(\mathbf{x}_b,\mathbf{x}'_b)$ as above

Combining different kernels allows us to combine multiple effects to describe our process at hand.

12.7 Hyperparameters optimization

Hyperparametrs of the kernel function have an impact on the covariance of the process. In practice, rather than fixing the covariance function, we may prefer to use a parametric family of functions and then infer the parameter values from the data.

The standard way to do so in a Bayesian approach is optimizing the (log) marginal likelihood. This is a data-driven method and works well if we can identify an analytical form for the log marginal likelihood. It turns out that this is our case:

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

We can rewrite it observing that $\mathbf{y} \sim \mathcal{N}(0,(K+\sigma^2\mathbb{I}))$:

$$\mathcal{L} = -\frac{1}{2}\mathbf{y}^T(K+\sigma^2\mathbb{I})^{-1}\mathbf{y} - \frac{1}{2}\log|(K+\sigma^2\mathbb{I})| - \frac{N}{2}\log 2\pi$$

The previous equation decomposes as:

- $-\frac{1}{2}\mathbf{y}^T(K+\sigma^2\mathbb{I})^{-1}\mathbf{y}$ which is the data fit;
- $-\frac{1}{2}\log|(K+\sigma^2\mathbb{I})|$ which is a complexity penalty term;
- $-\frac{N}{2}\log 2\pi$ which is a constant term.

Hence the marginal likelihood balances fit to the data and complexity penalty.

It is worth noting that the log marginal likelihood may be multimodal, hence not straightforward to optimize.

In order to maximize the marginal likelihood we can compute its derivative w.r.t. an hyperparameter θ, which is:

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|\mathbf{x},\theta) = \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}y - \frac{1}{2}tr\left(K^{-1}\frac{\partial K}{\partial\theta_j}\right) = \\ = \frac{1}{2}tr\left((\alpha\alpha^T - K^{-1})\frac{\partial K}{\partial\theta_j}\right)$$

where $\alpha = K^{-1}\mathbf{y}$. Once we have evaluated K^{-1} (which we need also for the inference task), then it is easy to optimize $\mathcal{L}$.

We can also be fully Bayesian and place a prior on hyperparameters and find their posterior. If we do this, then we lose the analytical form. Hence with GP we typically work with marginal likelihood optimization to find hyperparameters.

As already mentioned, the typical choice for the prior mean is the zero function. In this case we usually realign the mean of the GP and the mean of the data by subtracting the sample mean from the data.

As an alternative, we can use a generalized linear model for the prior mean:

$$g(\mathbf{x}) = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^T \beta$$

where $f(\mathbf{x}) \sim GP(0, k(\mathbf{x}, \mathbf{x}'))$.

We can also be Bayesian and place a Gaussian prior on the weights $\beta \sim \mathcal{N}(\mathbf{b}, B)$ and get a GP:

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

In this way we obtain the a predictive distribution at a point $\mathbf{x}^\star$ having as mean and covariance respectively:

$$\begin{aligned} \bar{\mathbf{g}}(\mathbf{x}^\star) &= H_\star^T \bar{\beta} + K_\star^T K_y^{-1}(\mathbf{y} - H^T \bar{\beta}) = \bar{\mathbf{f}}(\mathbf{x}^\star) + R^T \bar{\beta} \\ cov(\mathbf{g}^\star) &= cov(\mathbf{f}^\star) + R^T (B^{-1} + H K_y^{-1} H^T)^{-1} R \end{aligned}$$

where the H matrix collects the $\mathbf{h}(\mathbf{x})$ for all training (and $H_\star$ for all test) points, $\bar{\beta} = (B^{-1} + H K_y^{-1} H^T)^{-1}(H K_y^{-1} \mathbf{y} + B^{-1} \mathbf{b})$ and $R = H_\star - H K_y^{-1} K_\star$.

Hence the mean of the new predictive distribution is $H_\star^T \bar{\beta}$ (from the linear model), plus a term coming from the GP model of residuals.

12.8 GP classification

In what follows we will consider the 2-class classification problem (everything can be easily extended to the multiclass classification).

Our goal is to model $p(C_1|x)$: we can take the Sigmoid function, so that the model is $p(C_1|x) = \sigma(f(x)) = \pi(x)$ with $f \sim GP(\mu, K)$.

We have observations of the form (x_n, y_n) with $y_n \in \{0, 1\}$ (hence our dataset is $(\mathbf{x}, \mathbf{y})$, $\mathbf{f} = f(\mathbf{x}) = f(x_1), \ldots f(x_N)$) and likelihood model $p(\mathbf{y}|f(\mathbf{x})) = \text{Bernoulli}(\sigma(f(\mathbf{x})))$ (as in logistic regression).

In this setting the function f is called *latent* (or nuisance) function (we don't directly observe it, but it governs the distribution of our data).

Remark: $\pi(\mathbf{x}) = \sigma(f(\mathbf{x}))$ is a random function.

The first thing that we want to evaluate is the probability of the latent function given the observations, by Bayes' theorem:

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

If we want to make a prediction at point $\mathbf{x}^\star$, first we need to compute the posterior at prediction points $\mathbf{x}^\star$:

$$p(f(\mathbf{x}^\star)|\mathbf{y}) = \int p(f(\mathbf{x}^\star)|f(\mathbf{x}))p(f(\mathbf{x})|\mathbf{y})df(\mathbf{x})$$

Once we know this, we can compute the predictive distribution at $\mathbf{x}^\star$:

$$\pi^\star = \int \sigma(f(\mathbf{x}^\star))p(f(\mathbf{x}^\star)|\mathbf{y})df(\mathbf{x}^\star)$$

The problem is that, since the likelihood is a product of Bernoulli, there is no conjugacy (as in the case of Bayesian logistic regression), hence the posterior is not analytically tractable. The simplest solution is to rely on the Laplace approximation of the posterior $p(\mathbf{f}|\mathbf{y})$ around the MAP $\hat{f}$.

In order to do so we need to optimize the unnormalized log posterior, which reads as:

$$\begin{aligned}\Psi(\mathbf{f}) &= \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|\mathbf{x}) = \\ &= \log p(\mathbf{y}|\mathbf{f}) - \frac{1}{2}\mathbf{f}^T K^{-1}\mathbf{f} - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi\end{aligned}$$

Differentiating w.r.t. $\mathbf{f}$ we get:

$$\begin{aligned}\nabla\Psi(\mathbf{f}) &= \nabla \log p(\mathbf{y}|\mathbf{f}) - K^{-1}\mathbf{f} \\ \nabla\nabla\Psi(\mathbf{f}) &= \nabla\nabla \log p(\mathbf{y}|\mathbf{f}) - K^{-1} = -W - K^{-1}\end{aligned}$$

where W is diagonal, since observations are i.i.d.

We can optimize the log posterior via the Newton-Rapson scheme:

$$\begin{aligned}\mathbf{f}^{\text{new}} = \mathbf{f} - (\nabla\nabla\Psi)^{-1}\nabla\Psi &= \mathbf{f} + (K^{-1} + W)^{-1}(\nabla \log p(\mathbf{y}|\mathbf{f}) - K^{-1}\mathbf{f}) = \\ &= (K^{-1} + W)^{-1}(W\mathbf{f} + \nabla \log p(\mathbf{y}|\mathbf{f}))\end{aligned}$$

Plugging the definition of Laplace approximation around the MAP $\hat{f}$, we obtain a Gaussian q with mean:

$$\mathbb{E}_q[\mathbf{f}^\star|\mathbf{x}, \mathbf{y}, \mathbf{x}^\star] = k(\mathbf{x}^\star)^T K^{-1}\hat{f} = k(\mathbf{x}^\star)^T \nabla \log p(\mathbf{y}|\hat{f})$$

and variance:

$$\begin{aligned}\mathbb{V}_q[\mathbf{f}^\star|\mathbf{x}, \mathbf{y}, \mathbf{x}^\star] &= k(\mathbf{x}^\star, \mathbf{x}^\star) - k_\star^T K^{-1} k_\star + k_\star^T K^{-1}(K^{-1} + W)^{-1}K^{-1}k_\star = \\ &= k(\mathbf{x}^\star, \mathbf{x}^\star) - k_\star^T (K + W^{-1})^{-1} k_\star\end{aligned}$$

So that the prediction $\pi^\star$ can be computed by the integral:

$$\pi^\star \simeq \mathbb{E}_q[\pi^\star|\mathbf{x}, \mathbf{y}, \mathbf{x}^\star] = \int \sigma(\mathbf{f}^\star)q(\mathbf{f}^\star|\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)d\mathbf{f}^\star$$

which can be approximated with the same logit-probit-logit trick used for Bayesian logistic regression.

However for GP typically we use different methods of approximation, belonging to the family of Variational Inference methods.

A first option is to approximate the posterior by a Gaussian q, which minimizes the KL divergence between q and the posterior distribution, i.e. $KL[q(\mathbf{f}|\mathbf{x}, \mathbf{y})||p(\mathbf{f}|\mathbf{x}, \mathbf{y})]$.

Alternatively we can use **Expectation Propagation**, which constructs iteratively a Gaussian approximation of the posterior, iterating over observations the following operations (observe that the posterior depends proportionally on the product of the likelihoods):

- take the current Gaussian approximation and factor out the term for the i-th likelihood $p(y_i|f_i)$, obtaining a distribution for all observations but the i-th one (called the cavity);
- multiply the cavity by the exact likelihood of the i-th observation, and find a Gaussian approximation by moment matching of such a (non-Gaussian) distribution.

This method converges faster than Laplace approximation typically (there is no optimization to run) and provides also an approximation of the marginal likelihood.

As a final observation, note that the main problems of GP prediction are:

- it is vulnerable to outliers, since adding a new observation always reduce the uncertainty at all points;
- optimization of hyperparameters is tricky, especially if σ^2 is unknown;
- GP predictions rely on a matrix inversion which scales cubically with the number of points (i.e. GP don't scale well with big data).

Bibliography

Here are the references in citation order.

[1] Christopher M Bishop. *Pattern recognition and machine learning*. English. 00000. New York, NY: Springer, 2009 (cited on page v).

[2] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. Cambridge, MA: MIT Press, 2012 (cited on page v).

[3] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *Elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001. (Visited on 06/15/2016) (cited on page v).

[4] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012. (Visited on 07/27/2015) (cited on page v).

[5] Andrew Gelman et al. *Bayesian data analysis*. English. OCLC: 1051325129. 2014. (Visited on 01/29/2020) (cited on page v).

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning*. en. Cambridge University Press, 2014 (cited on page 7).

[7] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. en. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2018 (cited on page 7).

[8] Charles Blundell et al. *Weight Uncertainty in Neural Networks*. 2015 (cited on page 117).

[9] Stefano Ermon Volodymyr Kuleshov. *The variational auto-encoder*. en-us. URL: https://ermongroup.github.io/cs228-notes/extras/vae/ (visited on 02/26/2024) (cited on page 119).

[10] Jaan Altosaar. *Tutorial - What is a Variational Autoencoder?* Jan. 2021. DOI: 10.5281/zenodo.4462916 (cited on page 119).

[11] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022 (cited on pages 120, 122).

[12] Karsten Kreis, Ruiqi Gao, and Arash Vahdat. *Denoising Diffusion-based Generative Modeling: Foundations and Applications*. URL: https://cvpr2022-tutorial-diffusion-models.github.io (visited on 02/26/2024) (cited on page 123).

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 'Denoising diffusion probabilistic models'. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851 (cited on page 126).

[14] Jacob Austin et al. 'Structured denoising diffusion models in discrete state-spaces'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 17981–17993 (cited on page 127).

[15] Yang Song et al. 'Score-based generative modeling through stochastic differential equations'. In: *arXiv preprint arXiv:2011.13456* (2020).

[16] Ayan Das. *An introduction to Diffusion Probabilistic Models*. en-us. Dec. 2021. URL: https://ayandas.me/blog-tut/2021/12/04/diffusion-prob-models.html (visited on 02/26/2024) (cited on page 127).