# Mathematical Foundations of Machine Learning

Research Seminar and Report A (22/23)
Professor: Pedro Alexandre Santos

Master's in Applied Mathematics and Computation (MMAC)
Instituto Superior Técnico

Telmo Cunha (73487)[1]

January 31, 2023

[1]telmocunha@gmail.com

# Contents

# Introduction

This project arises in the context of a Seminar course, supervised by Professor Pedro Alexandre Santos[1], which is part of the Masters in Applied Mathematics and Computation at Instituto Superior Técnico (MMAC-IST). The main goal, content wise, was to study the formal foundations of learning and well established machine learning methods, focusing in particular on their mathematical formulation. This was done by following, for the most part, Chapters 1-7 of [Bis06] and Chapters 1-6 of [SSBD14].

The current version consists of independent study carried out by the author under the guidance of Professors Pedro Santos and João Costa[2]. I am very grateful to both for the several discussions and revisions of this document. Briefly, in each chapter we cover the following:

- **Chapter 1: Statistical Learning Theory -** This part is concerned with the formalization of the learning process. We explore in particular the notions of probably approximately correct (PAC) learning, Vapnik–Chervonenkis(VC)-dimension and their relation via the fundamental theorem of statistical learning.
- **Chapter 2: Linear Models for Supervised Learning -** Here we explore supervised learning methods with recourse to linearly parametrized models both in the context of regression and classification. This chapter is accompanied by several implementations of the methods in Python. In order of occurrence in the text we have the files: "polynomialcurvefitting.ipynb", "bayesianlinreg.ipynb", "bias-variance.ipynb", "perceptron.ipynb" and "logisticregression.ipynb".
- **Chapter 3: Neural Networks -** We introduce the simplest form of a neural network, the multilayer perceptron, and study its training process via gradient descent. We comment on some methods of regularization and end with a discussion of Bayesian neural networks. This chapter is also accompanied by several implementations of the methods in Python, in order of occurrence in the text we have the files: "polyfitsgd.ipynb", "nnet-reg.ipynb" and "nnet-class.ipynb".
- **Chapter 4: Kernel Methods -** Most methods we covered up to this point can be reformulated with recourse to something called a kernel function. We see in detail how this arises and explore some properties of kernels. As examples of kernel methods we introduce Support Vector Machines (SVMs) and Gaussian processes, and finalize by introducing the neural tangent kernel, which relates neural networks and kernel methods.
- **Chapter 5: Concluding remarks -** Because this seminar serves as a stepping stone for a prospective master's thesis we highlight in this chapter some considerations in this regard.
- **Chapter 6: Appendix -** The appendix consists of three parts: probability theory, information theory and Gaussian distributions. With the exception of the third part, where we cover some properties of Gaussian distributions in detail which are useful for Bayesian methods and Gaussian Processes, the other two are mostly a collection of definitions and basic properties.

[1] `https://math.tecnico.ulisboa.pt/professor?who=pasantos`
[2] `https://ciencia.iscte-iul.pt/authors/joao-lopes-costa/cv`

# Chapter 1

# Introduction to Statistical Learning Theory

In this first chapter we are concerned with formulating the learning problem, from a statistical point of view, and finding the conditions where we can "guarantee" that a learning algorithm is able to learn. What it means, for a learning algorithm to learn, will be defined formally via the **PAC (Probably Approximately Correct) learning** definition. Given the statistical nature of the problem, this "guarantee" is manifested in the ability to, given any probability that we desire over the success of a learning algorithm (technically in the probability of sampling a training dataset which guarantees a small risk), find the size of a dataset which is enough for the learner to have, at least, that probability of success. The main question we would like to answer in statistical learning theory is, how much data do I need to feed a learning algorithm to "guarantee" that it has a high chance of success? We will see that there is no easy answer to this question, in particular that there is no such thing as a general learning algorithm, one for which given a dataset of some fixed size is able to learn any task. This is the content of the **no-free-lunch theorem**, one of the main results to be found in this chapter. Ultimately, we will determine the characteristic property (of an hypothesis class) which guarantees (PAC) learnability. This is the content of the main result of the chapter, known as the **fundamental theorem in statistical learning**, which, informally, states that finite **VC(Vapnik-Chervonenkis)-dimension**, a measure of the flexibility within functions of the hypothesis class to approximate any possible underlying function on the learning task, is a necessary and sufficient condition to satisfy the (PAC) learning conditions.

## 1.1 Probably Approximately Correct (PAC) Learning Model

Before the machine has the opportunity to learn anything one must formalize the problem of learning and the elements at play. In this section we introduce a statistical learning framework for supervised learning, called the **probably approximately correct model** (PAC), based on Chapters 2 to 4 of [SSBD14].

### 1.1.1 PAC model for binary classification

We consider a **learner** to be any agent responsible for the learning process, an algorithm in general, and we shall start by looking at the specific **learning task** of binary classification. In this context, a learning task has the following components:

- **Information available to the learner:** This consists of **input data** from an arbitrary set $\mathcal{X}$ and a **label set** $\mathcal{Y}$, which contains the possible assignments to instances $x \in \mathcal{X}$. In the particular case of binary classification we have $\mathcal{Y} = \{0, 1\}$. The information available to the learner is a set $S = \{(x_n, y_n)\}_{n=1}^{N}$, called the **training data**, a finite sequence of pairs $(x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$.

- **What is learnt:** A map $h : \mathcal{X} \mapsto \mathcal{Y}$ which predicts an assignment $y \in \mathcal{Y}$ for any input $x \in \mathcal{X}$. We will refer to the function $h$ as the **predictor**.

- **Assumptions over the learning process:** We assume there exists a probability distribution over $\mathcal{X}$, denoted by $\mathcal{D}_{\mathcal{X}}$ and unknown to the learner, and also a correct labelling function $f : \mathcal{X} \mapsto \mathcal{Y}$, unknown as well, and for which $y_i = f(x_i)$, $\forall i = 1, ..., N$. Obtaining exactly this function $f$ would be the best possible outcome of the learner.

- **Measure of the learner's (in)competence:** This is defined as the measure of the set where the learner's prediction differs from $f$. Given a predictor $h : \mathcal{X} \mapsto \mathcal{Y}$ we define the **true error** of $h$ as:
$$L_{(\mathcal{D}_{\mathcal{X}},f)}(h) := \mathbb{P}_{x\sim\mathcal{D}_{\mathcal{X}}}[h(x) \neq f(x)] := \mathcal{D}_{\mathcal{X}}(\{x : h(x) \neq f(x)\}) \tag{1.1}$$
The error is the probability of randomly choosing $x \in \mathcal{X}$, according to distribution $\mathcal{D}_{\mathcal{X}}$, for which the prediction $h(x)$ is incorrect, i.e. differs from the correct label given by $f(x)$. In the best possible scenario we would have $L_{(\mathcal{D}_{\mathcal{X}},f)}(h) = 0$ while in the worst case scenario we would have $L_{(\mathcal{D}_{\mathcal{X}},f)}(h) = 1$.

Since both $\mathcal{D}_{\mathcal{X}}$ and $f$ are unknown we do not have access to the true error. With the available information, the training data, one can define the notion of **empirical risk** or **training error**.

**Definition 1.1.1.** ***(Empirical Risk/Training Error)***
*Given a training set $S$ and predictor $h : \mathcal{X} \mapsto \mathcal{Y}$, we define the empirical risk as:*
$$L_S(h) := \frac{|\{i \in \{1, ..., N\} : h(x_i) \neq y_i\}|}{N}, \tag{1.2}$$
*which is the fraction of incorrect classifications on the training data.*

Since the learner only has access to $S$ it makes sense to consider the learning paradigm where we minimize $L_S(h)$, which is called **empirical risk minimization** (ERM). However, a naive application of ERM will often lead to what is known as **overfitting**, i.e. the learner achieves training error $L_S(h) = 0$ while the true error $L_{(\mathcal{D}_{\mathcal{X}},f)}(h)$ may be too large. In particular, if we consider the predictor given by:
$$h'(x) = \begin{cases} y_i, \text{ if } \exists i \in \{1, ..., N\} \text{ s.t. } x_i = x, \\ 0, \text{ otherwise,} \end{cases} \tag{1.3}$$
which essentially memorizes the training data and labels everything else as zero, the learner always achieves $L_S(h') = 0$, zero training error. However, the true error, $L_{(\mathcal{D}_{\mathcal{X}},f)}(h')$, will depend on the relative measures of $\mathcal{D}_{\mathcal{X}}(\{x : f(x) = 1\})$ and $\mathcal{D}_{\mathcal{X}}(\{x : f(x) = 0\})$ which, in a continuous setting and depending on $\mathcal{D}_{\mathcal{X}}$ and $f$, can be arbitrarily close to 1 (a particular example, especially for fans of papayas, can be seen in Section 2.2.1 of [SSBD14]).

A way to overcome the overfitting problem is to use **inductive biases** or, equivalently, restricting the space of functions $\mathcal{H}$ where we search for a predictor, called the **hypothesis class**. The choice of the space $\mathcal{H}$ is done before seeing the data $S$ and typically reflects some prior knowledge (bias) about the problem. By restricting the hypothesis class we expect to reduce the chance of overfitting, since we are reducing the number of available functions. However, this inductive bias might result in a space which is not flexible enough to approximate the correct function $f$. This is a fundamental problem in learning theory known as the bias-complexity trade-off, we shall look into it in Section 1.3.

The ERM rule, restricted to the hypothesis class $\mathcal{H}$, solves the following optimization problem:
$$h_S := \text{ERM}_{\mathcal{H}}(S) \in \arg\min_{h\in\mathcal{H}} L_S(h). \tag{1.4}$$

A fundamental question in learning theory is to determine over which hypothesis classes $\mathcal{H}$ the predictor $h_S$ is guaranteed to not overfit. We finalize this section showing that, by imposing a bound on the number of predictors in $\mathcal{H}$, and thus rendering it finite, we can "guarantee" (in a probability sense) that $h_S$ will not overfit.

**Finite hypothesis class $\mathcal{H}$:** From now on we will assume that $|\mathcal{H}|$ is finite, i.e. the number of elements in the set $\mathcal{H}$. Assuming as well that $f : \mathcal{X} \mapsto \mathcal{Y}$ is the correct labelling function, by applying $\text{ERM}_{\mathcal{H}}$ to a given training data $S$ we obtain a predictor $h_S$ which satisfies:

$$h_S \in \arg\min_{h \in \mathcal{H}} L_S(h). \tag{1.5}$$

We make a further assumption, known as the **realizability assumption**, which is defined as:

**Definition 1.1.2.** *(**Realizability assumption**)*
*There exists $h^* \in \mathcal{H}$ such that $L_{(\mathcal{D}_\mathcal{X},f)}(h^*) = 0$.*

**Remark 1.1.1.** *Note that the realizability assumption depends on the triple $(\mathcal{H}, \mathcal{D}_\mathcal{X}, f)$, i.e. the hypothesis class $\mathcal{H}$ contains a predictor $h^*$ such that, for the distribution $\mathcal{D}_\mathcal{X}$ and labeling function $f$ we have $L_{(\mathcal{D}_\mathcal{X},f)}(h^*) = 0$.*

Under this assumption, with probability 1 over any sample $S$ with inputs sampled from $\mathcal{D}_\mathcal{X}$ and labeled by $f$, we have that:

$$h^* \in \arg\min_{h \in \mathcal{H}} L_S(h) \text{ and } L_S(h^*) = 0. \tag{1.6}$$

This implies that $h_S$, as given by equation (1.5), satisfies $L_S(h_S) = 0$. However, we are interested in estimating the true risk $L_{(\mathcal{D}_\mathcal{X},f)}(h_S)$. This quantity is dependent on the training sequence $S$ and is therefore a random variable. So, we are interested in evaluating the probability of sampling a sequence $S$ for which the true error $L_{(\mathcal{D}_\mathcal{X},f)}(h_S) < \epsilon$, i.e. we have a bound on the true error for some $\epsilon \in (0, 1)$. To quantify this we will make the further assumption that the data sequence $S$ is generated by sampling independently from $\mathcal{D}_\mathcal{X}$, i.e. the sample $S$ is independent and identically distributed (i.i.d.) by sampling inputs from $\mathcal{D}_\mathcal{X}$ and labelling them according to the function $f$, we write this as $S \sim \mathcal{D}_\mathcal{X}^N$.

**Definition 1.1.3.** *(**Accuracy ($\epsilon$) and confidence (1-$\delta$) parameters**)*
*We define $\delta \in (0, 1)$ to be the probability of sampling $S \sim {\mathcal{D}_\mathcal{X}}^{N(\epsilon,\delta)}$ for which $L_{(\mathcal{D}_\mathcal{X},f)}(h_S) > \epsilon$, for some $\epsilon \in (0, 1)$. We call $\epsilon$ the **accuracy parameter**. The **confidence parameter**, defined as $(1-\delta)$, is then the probability of sampling $S \sim {\mathcal{D}_\mathcal{X}}^{N(\epsilon,\delta)}$ for which $L_{(\mathcal{D}_\mathcal{X},f)}(h_S) \leq \epsilon$.*

**Remark 1.1.2.** *Note that both $\delta$ and $\epsilon$ are free parameters in the interval $(0, 1)$. They are related via the number of samples which is a function of the form $N(\epsilon, \delta)$, i.e. it depends on both $\epsilon$ and $\delta$. This will become clearer when we define the notion of PAC learning.*

We want to estimate an upper bound on the probability of having too large a risk according to $\epsilon$, i.e.:

$$\mathcal{D}_\mathcal{X}^N\big(\{S|_\mathcal{X} : L_{(\mathcal{D}_\mathcal{X},f)}(h_S) > \epsilon\}\big), \text{ with } S|_\mathcal{X} = (x_1, ..., x_N), \tag{1.7}$$

i.e. we are interested in bounding the probability of sampling a data sequence $S$ for which the learner fails to obtain a predictor $h_S$ which generalizes well, according to the accuracy parameter $\epsilon$.

Defining $\mathcal{H}_B = \{h \in \mathcal{H} : L_{(\mathcal{D}_\mathcal{X},f)}(h) > \epsilon\}$ and $M = \{S|_\mathcal{X} : \exists h \in \mathcal{H}_B, L_S(h) = 0\}$, via the realizability assumption, we have the following proposition:

**Proposition 1.1.1.** *We have the following set inclusion:*

$$\{S|_\mathcal{X} : L_{(\mathcal{D}_\mathcal{X},f)}(h_S) > \epsilon\} \subseteq M. \tag{1.8}$$

*Proof.* Let $S_x \in \{S|_{\mathcal{X}} : L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) > \epsilon\}$, by the realizability assumption $\exists h^* \in \arg\min_{h\in\mathcal{H}} L_S(h)$ with $L_S(h^*) = 0$ and, since $h_S \in \arg\min_{h\in\mathcal{H}} L_S(h)$, we must also have $L_S(h_S) = 0$. Thus, since $h_S \in \mathcal{H}_B$, because $L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) > \epsilon$, and $L_S(h_S) = 0$, we have that $S_x \in M$. ■

We can now prove the following :

**Theorem 1.1.2.** *(Corollary 2.3 of [SSBD14]) Consider a finite hypothesis class $\mathcal{H}$. Let $\delta, \epsilon \in (0,1)$ and $N \in \mathbb{N}$ satisfy:*

$$N \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}. \tag{1.9}$$

*Then, assuming the realizability assumption holds, for any labelling function $f \in \mathcal{H}$ and for any distribution $\mathcal{D}_{\mathcal{X}}$, with probability of at least $1-\delta$ over the choice of an i.i.d. sample $S$ of size $N$, for every ERM hypothesis $h_S$ the following holds:*

$$L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) \leq \epsilon. \tag{1.10}$$

*Proof.* Writing $M$ as follows:

$$M = \bigcup_{h\in\mathcal{H}_B} \{S|_{\mathcal{X}} : L_S(h) = 0\}, \tag{1.11}$$

where the set on the right side is the empty set whenever such an $h$ does not exist in $\mathcal{H}_B$ when choosing $S|_{\mathcal{X}} \in M$, by proposition 1.1.1, which assumes the realizability assumption, we have:

$$\mathcal{D}_{\mathcal{X}}^N\big(\{S|_{\mathcal{X}} : L_{(\mathcal{D},f)}(h_S) > \epsilon\}\big) \leq \mathcal{D}_{\mathcal{X}}^N(M) = \mathcal{D}_{\mathcal{X}}^N(\cup_{h\in\mathcal{H}_B}\{S|_{\mathcal{X}} : L_S(h) = 0\}). \tag{1.12}$$

By subadditivity of the measure, since $\mathcal{H}_B$ is finite and thus countable, we further have:

$$\mathcal{D}_{\mathcal{X}}^N\big(\{S|_{\mathcal{X}} : L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) > \epsilon\}\big) \leq \sum\nolimits_{h\in\mathcal{H}_B} \mathcal{D}_{\mathcal{X}}^N(\{S|_{\mathcal{X}} : L_S(h) = 0\}). \tag{1.13}$$

Fixing $h \in \mathcal{H}_B$, the event $L_S(h) = 0$ is equivalent to $h(x_i) = f(x_i)$ for all $i = 1, ..., N$. Using the i.i.d. assumption we can write:

$$\mathcal{D}_{\mathcal{X}}^N(\{S|_{\mathcal{X}} : L_S(h) = 0\}) = \mathcal{D}_{\mathcal{X}}^N(\{S|_{\mathcal{X}} : \forall i = 1, ..., N,\, h(x_i) = f(x_i)\}) = \prod_{i=1}^{N} \mathcal{D}_{\mathcal{X}}\big(\{x_i : h(x_i) = f(x_i)\}\big). \tag{1.14}$$

For each input sample $x_i$ we have:

$$\mathcal{D}_{\mathcal{X}}\big(\{x_i : h(x_i) = f(x_i)\}\big) = 1 - L_{(\mathcal{D}_{\mathcal{X}},f)}(h) \leq 1 - \epsilon, \tag{1.15}$$

by the definition in equation (1.1) and, since $h \in \mathcal{H}_B$, $L_{(\mathcal{D}_{\mathcal{X}},f)}(h) > \epsilon$.

Using the inequality $1 - \epsilon \leq e^{-\epsilon}$ (which can be checked via power series expansion), we have:

$$\mathcal{D}_{\mathcal{X}}^N(\{S|_{\mathcal{X}} : L_S(h) = 0\}) \leq (1-\epsilon)^N \leq e^{-N\epsilon} : \tag{1.16}$$

and, combining with (1.13), we obtain:

$$\mathcal{D}_{\mathcal{X}}^N\big(\{S|_{\mathcal{X}} : L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) > \epsilon\}\big) \leq \sum_{h\in\mathcal{H}_B} e^{-N\epsilon} = |\mathcal{H}_B|e^{-N\epsilon} \leq |\mathcal{H}|e^{-N\epsilon}. \tag{1.17}$$

Considering $\mathcal{D}_{\mathcal{X}}^N\big(\{S|_{\mathcal{X}} : L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) > \epsilon\}\big) \leq \delta$ and writing:

$$\delta = |\mathcal{H}|e^{-N\epsilon}. \tag{1.18}$$

We have equality for $N = \frac{1}{\epsilon}\log\big(\frac{|\mathcal{H}|}{\delta}\big)$ therefore, by picking $N \geq \frac{1}{\epsilon}\log\big(\frac{|\mathcal{H}|}{\delta}\big)$, we obtain:

$$\mathcal{D}_{\mathcal{X}}^N\big(\{S|_{\mathcal{X}} : L_{(\mathcal{D}_{\mathcal{X}},f)}(h_S) > \epsilon\}\big) \leq \delta, \tag{1.19}$$

which gives us equation (1.9). ■

In summary, if we have a training sequence $S$ with a large enough number of i.i.d. data points $N$, i.e. satisfying (1.9), over a finite hypothesis class $\mathcal{H}$, the $\text{ERM}_{\mathcal{H}}$ rule will *probably* (with confidence $1-\delta$) be *approximately* (up to true error $\epsilon$) correct when choosing a predictor $h_S$. Note that the bound on $N$ is independent of $\mathcal{D}_{\mathcal{X}}$ and of the function $f$. Intuitively, if we have a finite number of functions and one of those is a.e. equal to the actual labelling function then we can essentially distinguish between them. In particular we can find the one closest to $f$ by having enough datapoints sampled i.i.d. from the underlying distribution. If two distinct functions $f_1$ and $f_2$ are equal a.e. to $f$, except in a "small" set with non-zero measure $A$ then, if we sample a large enough number of points from the distribution such that we sample a "significant" amount of points from $A$ we will be able to, with a certain probability, pick the closest to $f$ via ERM. Since the realizability assumption holds, by taking $N \to \infty$, sampled i.i.d., we will hone in on the best possible function. Note that this is however never guaranteed since, for a finite sample size, it could be the case that we sampled the same input-output pair over and over.

Via the previous theorem we define the notion of a general hypothesis class $\mathcal{H}$ being PAC learnable.

**Definition 1.1.4.** ***(PAC learnability)***
*A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $N_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm which satisfies the following: for any $\epsilon, \delta \in (0,1)$, any distribution $\mathcal{D}_{\mathcal{X}}$, over input space $\mathcal{X}$, and any labelling function $f : \mathcal{X} \mapsto \{0,1\}$, provided that $\mathcal{H}$ satisfies the realizability assumption with respect to $\mathcal{D}_{\mathcal{X}}$ and $f$, the learning algorithm over $N \geq N_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated via $\mathcal{D}_{\mathcal{X}}$ and labeled by $f$, returns a predictor $h$ s.t., with probability at least $1-\delta$ (over the choice of $N$ training samples), the true error satisfies $L_{(\mathcal{D}_{\mathcal{X}},f)}(h) \leq \epsilon$.*

Since for any $N$ larger than the one given by equation (1.9) the PAC learning conditions are satisfied we introduce the notion of **sample complexity**, which is defined as follows:

**Definition 1.1.5.** ***(Sample complexity of $\mathcal{H}$)***
*Given $\epsilon, \delta \in (0,1)$ and a hypothesis class $\mathcal{H}$, the sample complexity is the smallest integer which satisfies the PAC learnability definition with respect to $\mathcal{H}$ , i.e.:*

$$\min\{N_{\mathcal{H}}(\epsilon,\delta) \in \mathbb{N} \text{ s.t. } \mathcal{H} \text{ is PAC learnable}\}. \tag{1.20}$$

With these definitions we can now write a corollary to Theorem 1.1.2 in the following way:

**Corollary 1.1.3.** *(Corollary 3.2 of [SSBD14]) Any finite hypothesis class $\mathcal{H}$ is PAC learnable with sample complexity satisfying:*

$$N_{\mathcal{H}}(\epsilon,\delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil, \tag{1.21}$$

*where, for $x \in \mathbb{R}$, $\lceil x \rceil$ represents the smallest integer larger than $x$.*

**Extension of the model:** We shall now consider an extension of this model in two ways. First we drop the realizability assumption, there is no guarantee that the true labelling function $f$ can be represented within the chosen hypothesis class $\mathcal{H}$ and, more importantly, such a function $f$ might not exist, i.e. over the features we have access to, it might be impossible to perfectly label the data. This requires the introduction also of a probability distribution over the set of labels $\mathcal{Y}$ and leads to the agnostic PAC model. The second extension is to generalize beyond the binary classification setting, to multiclass classification and regression problems.

### 1.1.2 Agnostic PAC model for binary classification

Instead of a distribution $\mathcal{D}_{\mathcal{X}}$ and a perfect labelling function $f$ we now consider a joint distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, and in particular over $\mathcal{X} \times \{0,1\}$ in the binary classification setting. The true error in this case is defined as:

$$L_{\mathcal{D}}(h) := \mathbb{P}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] = \mathcal{D}\big(\{(x,y) : h(x) \neq y\}\big). \tag{1.22}$$

The empirical risk, since we are still in the binary classification setting, remains unchanged and given by (1.2). The goal is to find a predictor $h$ which minimizes the true error in the PAC setting. The next proposition (exercise 3.7 of [SSBD14]) shows that the best possible predictor in this setting is given by the Bayes optimal predictor:

**Proposition 1.1.4.** ***(Bayes optimal predictor)***
*Given a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$, the best possible labelling function $f_{\mathcal{D}} : \mathcal{X} \mapsto \{0,1\}$, i.e. for which $L_{\mathcal{D}}$ is as small as possible, is given by:*

$$f_{\mathcal{D}}(x) = \begin{cases} 1, & \text{if } \mathbb{P}[Y = 1|X = x] \geq 1/2, \\ 0, & \text{otherwise.} \end{cases} \tag{1.23}$$

*Proof.* Let $g : \mathcal{X} \mapsto \{0,1\}$ be any other classifier, we show that $L_{\mathcal{D}}(g) \geq L_{\mathcal{D}}(f_{\mathcal{D}})$. We have:

$$L_{\mathcal{D}}(g) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathbb{1}_{[g(X)\neq Y]}] = \mathbb{E}_{x\sim\mathcal{D}_{\mathcal{X}}}[\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[g(X)\neq Y]}|X = x]], \tag{1.24}$$

by definition and the law of iterated (or total) expectation. The term inside the expectation over $X$ is equivalent to:

$$\begin{aligned} &\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[g(X)\neq Y]}|X = x] \\ &\quad = \mathbb{P}\big[\{g(X) \neq Y|X = x\}\big] \\ &\quad = \mathbb{P}\big[g(X) = 1, Y = 0|X = x\big] + \mathbb{P}\big[g(X) = 0, Y = 1|X = x\big] \\ &\quad = \mathbb{P}[g(X) = 1|X = x]\mathbb{P}[Y = 0|X = x] + \mathbb{P}[g(X) = 0|X = x]\mathbb{P}[Y = 1|X = x], \end{aligned} \tag{1.25}$$

since the labelling by $g$ is independent of the label probabilities coming from the distribution $\mathcal{D}$, i.e. it is any random classifier of the form $g : \mathcal{X} \mapsto \{0,1\}$. Because we have only two labels then $\mathbb{P}[Y = 0|X = x] = 1 - \mathbb{P}[Y = 1|X = x]$ and we obtain:

$$\begin{aligned} &\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[g(X)\neq Y]}|X = x] \\ &\quad = \mathbb{P}[g(X) = 1|X = x](1 - \mathbb{P}[Y = 1|X = x]) + \mathbb{P}[g(X) = 0|X = x]\mathbb{P}[Y = 1|X = x]. \end{aligned} \tag{1.26}$$

In the interest of comparing this term with its equivalent for $f_{\mathcal{D}}$ we compute:

$$\begin{aligned} &\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[f_{\mathcal{D}}(X)\neq Y]}|X = x] \\ &\quad = \mathbb{P}\big[\{f_{\mathcal{D}}(X) \neq Y|X = x\}\big] \\ &\quad = \mathbb{P}[f_{\mathcal{D}}(X) = 1|X = x](1 - \mathbb{P}[Y = 1|X = x]) + \mathbb{P}[f_{\mathcal{D}}(X) = 0|X = x]\mathbb{P}[Y = 1|X = x]. \end{aligned} \tag{1.27}$$

Now, if $\mathbb{P}[Y = 1|X = x] \geq 1/2$ then $\mathbb{P}[f_{\mathcal{D}}(X) = 1|X = x] = 1$ and $\mathbb{P}[f_{\mathcal{D}}(X) = 1|X = x] = 0$, similarly if $\mathbb{P}[Y = 1|X = x] < 1/2$ then $\mathbb{P}[f_{\mathcal{D}}(X) = 1|X = x] = 0$ and $\mathbb{P}[f_{\mathcal{D}}(X) = 1|X = x] = 1$. Therefore we can write:

$$\begin{aligned} &\mathbb{P}\big[\{f_{\mathcal{D}}(X) \neq Y|X = x\}\big] \\ &\quad = \mathbb{1}_{\mathbb{P}[Y=1|X=x]\geq 1/2}(1 - \mathbb{P}[Y = 1|X = x]) + \mathbb{1}_{\mathbb{P}[Y=1|X=x]<1/2}\mathbb{P}[Y = 1|X = x] \\ &\quad = \min\{1 - \mathbb{P}[Y = 1|X = x], \mathbb{P}[Y = 1|X = x]\}. \end{aligned} \tag{1.28}$$

From equation (1.26) we can write:

$$\begin{aligned} &\mathbb{P}[g(X) = 1|X = x](1 - \mathbb{P}[Y = 1|X = x]) + \mathbb{P}[g(X) = 0|X = x]\mathbb{P}[Y = 1|X = x] \\ &\quad \geq \big(\mathbb{P}[g(X) = 1|X = x] + \mathbb{P}[g(X) = 0|X = x]\big)\min\{1 - \mathbb{P}[Y = 1|X = x], \mathbb{P}[Y = 1|X = x]\} \\ &\quad = \min\{1 - \mathbb{P}[Y = 1|X = x], \mathbb{P}[Y = 1|X = x]\}. \end{aligned} \tag{1.29}$$

Thus, for every $x \in \mathcal{X}$ we have:

$$\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[g(X)\neq Y]}|X = x] \geq \mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[f_{\mathcal{D}}(X)\neq Y]}|X = x]. \tag{1.30}$$

Which implies that:

$$L_{\mathcal{D}}(g) = \mathbb{E}_{x\sim\mathcal{D}_{\mathcal{X}}}[\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[g(X)\neq Y]}|X=x]] \geq \mathbb{E}_{x\sim\mathcal{D}_{\mathcal{X}}}[\mathbb{E}_{y\sim\mathcal{D}_{\mathcal{Y}|x}}[\mathbb{1}_{[f_{\mathcal{D}}(X)\neq Y]}|X=x]] = L_{\mathcal{D}}(f_{\mathcal{D}}). \quad (1.31)$$

■

The distribution $\mathcal{D}$ is unknown to the learner, the predictor has to be determined with recourse only to the sample data $S$. Of course, this rules out the Bayes optimal predictor which requires knowledge of $\mathcal{D}$ and represents the minimal possible true error.

It is proven later (no-free-lunch, Theorem 1.2.2) that, without any prior assumption on $\mathcal{D}$, no algorithm can be guaranteed to find a predictor with the same performance as the Bayes optimal predictor. The requirement will be that the error of the predictor found should not be much larger than the best possible predictor within the hypothesis class being searched on. The notion of PAC learnability in this context is now defined as:

**Definition 1.1.6.** ***(Agnostic PAC learnability)***
*A hypothesis class $\mathcal{H}$ is agnostic PAC learnable if there exists $N_{\mathcal{H}} : (0,1)^2 \mapsto \mathbb{N}$ and a learning algorithm which satisfies: for every $\epsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, running the algorithm on $N \geq N_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples from $\mathcal{D}$, returns a predictor $h$ s.t., with probability at least $(1-\delta)$ (over the choice of the $N$ examples) we have:*

$$L_{\mathcal{D}}(h) \leq \min_{h'\in\mathcal{H}} L_{\mathcal{D}}(h') + \epsilon. \quad (1.32)$$

Under the realizability assumption, which implies that $\min_{h'\in\mathcal{H}} L_{\mathcal{D}}(h') = 0$, we recover the PAC learning model. So, agnostic PAC learning is a generalization of the previous definition of PAC learning.

### 1.1.3 Agnostic PAC model extension to multiclass classification and regression

The extension of agnostic PAC learning beyond binary classification is obtained by defining general notions of **loss function**, **risk function** and **empirical risk** as follows:

**Definition 1.1.7.** ***(Loss function)***
*Given a hypothesis class $\mathcal{H}$ and the set $\mathcal{X} \times \mathcal{Y}$, of input/target pairs, we define the loss function as a map of the form:*

$$\ell : \mathcal{H} \times \big(\mathcal{X} \times \mathcal{Y}\big) \mapsto \mathbb{R}_+. \quad (1.33)$$

**Definition 1.1.8.** ***(Risk function)***
*Given a predictor $h \in \mathcal{H}$ and a distribution $\mathcal{D}$, over $\mathcal{X} \times \mathcal{Y}$, the risk function $L_{\mathcal{D}}(h)$ is defined as the expectation of a loss function $\ell$ with respect to $\mathcal{D}$, i.e.:*

$$L_{\mathcal{D}}(h) := \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\ell\big(h,(x,y)\big)\big]. \quad (1.34)$$

**Remark 1.1.3.** *The function $\ell(h,\cdot) : \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}_+$, for $(x,y) \sim \mathcal{D}$, is a random variable so we must require further that $\ell$ be a measurable function over some $\sigma-$algebra defined on $\mathcal{X} \times \mathcal{Y}$.*

**Definition 1.1.9.** ***(Empirical risk)***
*Given a predictor $h \in \mathcal{H}$, a dataset $S = \{(x_n, y_n)\}_{n=1}^N$ generated from $\mathcal{D}$ and a loss function $\ell$, we define the empirical risk as:*

$$L_S(h) := \frac{1}{N}\sum_{n=1}^{N} \ell(h,(x_n,y_n)). \quad (1.35)$$

In binary and multiclass classification problems one typically uses the **0-1 loss** function which is defined as:

**Definition 1.1.10.** *(0-1 loss)*
*Given a predictor $h \in \mathcal{H}$, an input set $\mathcal{X}$ and $\mathcal{Y} = \{1, 2, ..., K\}$, for $K \in \mathbb{N}$, over a pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ we define:*

$$\ell_{0-1}(h, (x, y)) := \begin{cases} 1, & \text{if } h(x) \neq y, \\ 0, & \text{if } h(x) = y. \end{cases} \tag{1.36}$$

While, in the case of regression, one usually considers the square loss defined as:

**Definition 1.1.11.** ***(Square loss)***
*Given a predictor $h \in \mathcal{H}$, an input set $\mathcal{X}$ and $\mathcal{Y} = \mathbb{R}$, over a pair $(x, y) \in \mathcal{X} \times \mathbb{R}$ we define:*

$$\ell_{sq}(h, (x, y)) := (h(x) - y)^2. \tag{1.37}$$

With these generalizations, the notion of agnostic PAC learnability becomes:

**Definition 1.1.12.** ***(Agnostic PAC learnability for general loss functions)***
*A hypothesis class $\mathcal{H}$ is agnostic PAC learnable with respect to $\mathcal{X} \times \mathcal{Y}$ and loss function $\ell$, if there exists $N_{\mathcal{H}} : (0, 1)^2 \mapsto \mathbb{N}$ and a learning algorithm which satisfies: for every $\epsilon, \delta \in (0, 1)$ and every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, running the algorithm on $N \geq N_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples from $\mathcal{D}$, returns a predictor $h$ s.t., with probability at least $(1 - \delta)$ (over the choice of the $N$ examples) we have:*

$$L_{\mathcal{D}}(h) := \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\ell\big(h, (x, y)\big)\big] \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon. \tag{1.38}$$

We have showed, in Theorem 1.1.2, that any finite hypothesis class $\mathcal{H}$ is PAC learnable, under the realizability assumption and in the particular case of binary classification. We will now show that this remains true in the case of the agnostic PAC model for a general loss function. This requires the introduction of a property of the hypothesis class $\mathcal{H}$ known as **uniform convergence**.

### 1.1.4 Uniform convergence criteria

As we have seen, the ERM rule determines a predictor $h \in \mathcal{H}$ which minimizes the empirical risk over a sample $S$ obtained from $\mathcal{D}$. The goal is to know how accurate the empirical risk $L_S(h_S)$ is when compared to the true risk $L_{\mathcal{D}}(h_S)$. In particular, we will require an appropriate notion of distance between the empirical risk and the true risk. This is captured in the following definition:

**Definition 1.1.13.** *($\epsilon$-representative sample)*
*A training set $S$ is called $\epsilon$-representative, w.r.t. to a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, loss function $\ell$ and hypothesis class $\mathcal{H}$, if:*

$$\forall h \in \mathcal{H}, \ \ |L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon. \tag{1.39}$$

Given an accuracy parameter $\epsilon$ the next lemma shows that we require a $\epsilon/2$-representative sample $S$ in order to guarantee that we obtain a good predictor $h$ following ERM.

**Lemma 1.1.5.** *Consider the accuracy parameter $\epsilon$ and assume that a training set $S$ is $\epsilon/2$-representative (w.r.t. $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$, $\ell$ and $\mathcal{H}$). Then, any $h_S \in \arg\min_{h\in\mathcal{H}} L_S(h)$, i.e. the output of $ERM_{\mathcal{H}}(S)$, satisfies:*

$$L_{\mathcal{D}}(h_S) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon. \tag{1.40}$$

*Proof.* Since $S$ is $\epsilon/2$-representative we have that:

$$L_{\mathcal{D}}(h_S) \leq L_S(h_S) + \frac{\epsilon}{2} \leq L_S(h) + \frac{\epsilon}{2}, \tag{1.41}$$

for any $h \in \mathcal{H}$, since $h_S \in \arg\min_{h\in\mathcal{H}} L_S(h)$. Using again the $\epsilon/2$-representative condition we get:

$$L_S(h) + \frac{\epsilon}{2} \leq L_{\mathcal{D}}(h) + \frac{\epsilon}{2} + \frac{\epsilon}{2} = L_{\mathcal{D}}(h) + \epsilon. \tag{1.42}$$

Since this is true for any $h \in \mathcal{H}$ we obtain in particular:

$$L_{\mathcal{D}}(h_S) \leq \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon. \tag{1.43}$$

■

The following definition connects the notion of an $\epsilon-$representative sample with the requirements for PAC learnability.

**Definition 1.1.14.** ***(Uniform convergence property for $\mathcal{H}$)***
*A hypothesis class $\mathcal{H}$ has the uniform convergence property (w.r.t. to $\mathcal{X}\times\mathcal{Y}$ and loss function $\ell$) if there exists a function $N^{UC}_{\mathcal{H}} : (0,1)^2 \mapsto \mathbb{N}$ s.t. for every $\epsilon, \delta \in (0,1)$ and any distribution $\mathcal{D}$ over $\mathcal{X}\times\mathcal{Y}$, if $S$ is a sample of $N \geq N^{UC}_{\mathcal{H}}(\epsilon,\delta)$ examples drawn i.i.d. from $\mathcal{D}$ then, with probability at least $(1-\delta)$, $S$ is $\epsilon-$representative.*

With this property, and the previous lemma, we immediately get the following corollary:

**Corollary 1.1.6.** *(Corollary 4.4 of [SSBD14]) If a hypothesis class $\mathcal{H}$ has the uniform convergence property, with function $N^{UC}_{\mathcal{H}}$, then $\mathcal{H}$ is agnostically PAC learnable with sample complexity $N_{\mathcal{H}}(\epsilon,\delta) \leq N^{UC}_{\mathcal{H}}(\epsilon/2,\delta)$. In particular, the ERM rule is an agnostic PAC learner for the class $\mathcal{H}$.*

We now prove that finite hypothesis classes are agnostic PAC learnable, via this last corollary, by showing that they possess the uniform convergence property.

**Theorem 1.1.7.** *(Corollary 4.6 of [SSBD14]) Consider a finite hypothesis class $\mathcal{H}$, a domain $\mathcal{X}\times\mathcal{Y}$ and a loss function $\ell : \mathcal{H}\times\big(\mathcal{X}\times\mathcal{Y}\big) \mapsto [0,1]$. Then, $\mathcal{H}$ has the uniform convergence property with sample complexity:*

$$N^{UC}_{\mathcal{H}}(\epsilon,\delta) \leq \left\lceil \frac{\log(|2\mathcal{H}|/\delta)}{2\epsilon^2} \right\rceil. \tag{1.44}$$

*In particular, the $ERM_{\mathcal{H}}$ rule is agnostically PAC learnable with sample complexity:*

$$N_{\mathcal{H}}(\epsilon,\delta) \leq N^{UC}_{\mathcal{H}}(\epsilon/2,\delta) \leq \left\lceil \frac{2\log(|2\mathcal{H}|/\delta)}{\epsilon^2} \right\rceil. \tag{1.45}$$

*Proof.* Due to the previous corollary we need only show that $\mathcal{H}$ has the uniform convergence property, w.r.t . $\mathcal{X}\times\mathcal{Y}$ and $\ell$.

Let $(\epsilon,\delta) \in (0,1)$, we want to find $N(\epsilon,\delta)$ s.t., with probability $(1-\delta)$ over the choice of $N(\epsilon,\delta)$ i.i.d. samples from any $\mathcal{D}$, defined on $\mathcal{X}\times\mathcal{Y}$, we can guarantee that, for any $h \in \mathcal{H}$ we have $|L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon$, i.e. we want:

$$\mathcal{D}^{N(\epsilon,\delta)}\big(\{S : \forall h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon\}\big) \geq 1-\delta, \tag{1.46}$$

which is equivalent to finding $N(\epsilon,\delta)$ such that:

$$\mathcal{D}^{N(\epsilon,\delta)}\big(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\big) < \delta. \tag{1.47}$$

Similarly to the proof of Theorem 1.1.2 we can write:

$$\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\} = \cup_{h\in\mathcal{H}}\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}, \tag{1.48}$$

which, by subadditivity of the measure since $\mathcal{H}$ is finite, is equivalent to:

$$\mathcal{D}^{N(\epsilon,\delta)}\big(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\big) \leq \sum_{h\in\mathcal{H}} \mathcal{D}^{N(\epsilon,\delta)}\big(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\big). \tag{1.49}$$

The risk and empirical risk are given, respectively, by:

$$\begin{cases} L_{\mathcal{D}}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(h,(x,y)], \\ L_S(h) = \frac{1}{N}\sum_{i=1}^{N} \ell(h,(x_i,y_i)). \end{cases} \tag{1.50}$$

Because the $(x_i, y_i)$ are sampled i.i.d. from $\mathcal{D}$ we have that $\mathbb{E}[\ell(h,(x_i,y_i))] = L_{\mathcal{D}}(h)$ and thus $\mathbb{E}[L_S(h)] = L_{\mathcal{D}}(h)$. The quantity $|L_S(h) - L_{\mathcal{D}}(h)|$ is then the deviation of the random variable $L_S(h)$ with respect to its mean. The goal is then to show that the measure of $L_S(h)$ is concentrated around its mean.

**Remark 1.1.4.** *By the law of large numbers, i.e. when $N \to \infty$, the empirical average $L_S(h)$ converges to its expectation given by $L_{\mathcal{D}}(h)$. However, since we are interested in the gap between the empirical average over a finite sample $N$ and the true risk this asymptotic result is of no use here.*

We use instead the following measure concentration inequality known as Hoeffding's inequality:

**Lemma 1.1.8.** ***(Hoeffding's inequality)***
*Let $X_1, ..., X_N$ be a sequence of i.i.d. random variables such that $\mathbb{E}[X_i] = \mu$ and $\mathbb{P}[a \leq X_i \leq b] = 1$, for some $a, b \in \mathbb{R}$. Then, for any $\epsilon > 0$ we have:*

$$\mathbb{P}\left[\left|\frac{1}{N}\sum_{i=1}^{N} X_i - \mu\right| > \epsilon\right] \leq 2\exp\left(-\frac{2N\epsilon^2}{(b-a)^2}\right). \tag{1.51}$$

*Proof.* See the proof of lemma B.6 in the appendix of [SSBD14]. ■

Let $X_i = \ell(h,(x_i,y_i))$, because $h$ is fixed and the $(x_i, y_i)$ are sampled i.i.d. the $X_i$ are i.i.d. random variables with $\mathbb{E}[X_i] = L_{\mathcal{D}}(h) := \mu$. Then, we have:

$$|L_S(h) - L_{\mathcal{D}}(h)| = \left|\frac{1}{N(\epsilon,\delta)}\sum_{i=1}^{N(\epsilon,\delta)} X_i - \mu\right|. \tag{1.52}$$

Assuming that the range of $\ell \in [0,1]$ and thus that $X_i \in [0,1]$, via Hoeffding's inequality we obtain:

$$\mathcal{D}^{N}\big(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\big) = \mathbb{P}\left[\left|\frac{1}{N}\sum_{i=1}^{N} X_i - \mu\right| > \epsilon\right] \leq 2\exp\big(-2N\epsilon^2\big). \tag{1.53}$$

Back to equation (1.49) we obtain:

$$\mathcal{D}^{N(\epsilon,\delta)}\big(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\big) \leq \sum_{h\in\mathcal{H}} 2\exp\big(-2N\epsilon^2\big) = 2|\mathcal{H}|\exp\big(-2N\epsilon^2\big). \tag{1.54}$$

Then, to have:

$$\mathcal{D}^{N(\epsilon,\delta)}\big(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\big) \leq \delta, \tag{1.55}$$

we require that:

$$N \geq \frac{\log(2|\mathcal{H}|/\delta)}{2\epsilon^2}. \tag{1.56}$$

Therefore, the sample complexity will be smaller or equal to the smallest integer larger than this value. ■

**Remark 1.1.5.** *Perhaps it warrants a remark at this point on why this concern over finite hypothesis classes. If we choose as our hypothesis class $\mathcal{H}$ a parametric family of 1-parameter $\theta \in \mathbb{R}$ then $\mathcal{H}$ is infinite. However, in a computational setting using a 64 bit floating point representation,the hypothesis class has size $2^{64}$, i.e. there are $2^{64}$ possible values for $\theta$ and the hypothesis class is in fact finite. With $d$ parameters our hypothesis class would have a size of $2^{64d}$.*

Due to the overfitting problem we saw that it was advantageous to limit the space of predictors $\mathcal{H}$ to obtain the PAC learnable conditions. Over the next section we shall see why some form of inductive bias is required for successful learning. In particular, by the no-free-lunch theorem, there is no such thing as a general learner, i.e. a learner which given an i.i.d. dataset $S$, of a fixed size $N$, distributed according to any distribution $\mathcal{D}$ and labeled according to any function $f$, will have a high chance of obtaining a predictor with low risk. This in turn leads to the bias-complexity tradeoff, we will see how the error of an $\text{ERM}_{\mathcal{H}}$ learner splits into two competing terms over the size of the hypothesis class.

## 1.2 No-Free-Lunch Theorem

The no-free-lunch Theorem, mentioned quite a number of times at this point, shows there is no universal learner by associating to any learner $A$ a learning task where $A$ fails but a different learner succeeds. For the proof of the theorem, we require the following lemma:

**Lemma 1.2.1.** ***(Lemma B.1 from [SSBD14])***
*Let $X$ be a random variable that takes values in $[0,1]$. Assuming that $\mathbb{E}[X] = \mu$ then, for any $a \in (0,1)$ we have:*

$$\mathbb{P}[X \geq a] \geq \frac{\mu - a}{1 - a}. \tag{1.57}$$

*Proof.* The result follows from Markov's inequality, see the proof of both in appendix B of [SSBD14]. ■

**Theorem 1.2.2.** ***(No-free-lunch Theorem)***
*Let $A$ be any learning algorithm and consider a binary classification task using the 0-1 loss function over input domain $\mathcal{X}$. Let $N$, the size of the training set, be any number smaller than $|\mathcal{X}|/2$. Then, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ such that:*

- *There exists a function $f : \mathcal{X} \mapsto \{0,1\}$ with $L_{\mathcal{D}}(f) = 0$.*
- *With probability of at least 1/7 over the choice of $S \sim \mathcal{D}^N$ we have $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

*Proof.* The idea of the proof is to pick a finite subset $C \subset \mathcal{X}$ of size $2N$ s.t. any learner that obtains only half of the elements from $C$ is susceptible to miss-classifying the labels over the other $N$ unseen instances. This is due to the possibility of existing an "adversarial" labelling function $f$, i.e. one which contradicts the labels predicted by the learner on the, at least, $N$ unseen instances.

The set of functions $T = \{f \mid f : C \mapsto \{0,1\}\}$ has size $|T| = 2^{2N}$. Denote each of these functions by $f_i$, $i = 1, ..., |T|$, and for each construct a distribution over $C \times \{0,1\}$ of the form:

$$\mathcal{D}_i(\{(x,y)\}) = \begin{cases} 1/|C|, & \text{if } y = f_i(x), \\ 0, & \text{otherwise.} \end{cases} \tag{1.58}$$

This means that $f_i$ is a perfect labelling function for $\mathcal{D}_i$ with a uniform distribution over the $x$ instances in $C$, we have thus $L_{\mathcal{D}_i}(f_i) = 0$. The remainder of the proof is to show that, for any learning algorithm

$A$ which receives a training set $S$ of size $N$ from $C \times \{0,1\}$ and returns a predictor $A(S) : C \mapsto \{0,1\}$, the following holds:

$$\max_{i \in \{1,...,|T|\}} \mathbb{E}_{S \sim \mathcal{D}_i^N}[L_{\mathcal{D}_i}(A(S))] \geq 1/4. \tag{1.59}$$

To prove (1.59) denote all possible sequences of $N$ elements from $C$, of which there are $k = (2N)^N$, by $S_1, ..., S_k$. For each $S_j = (x_1, ..., x_N)$ denote by $S_j^i = ((x_1, f_i(x_1)), ..., (x_N, f_i(x_N)))$, i.e. the labelling of $S_j$ by the function $f_i$. If the underlying distribution is $\mathcal{D}_i$ the possible training sets that $A$ can receive are $S_1^i, ..., S_k^i$ with equal probability of being sampled since $\mathcal{D}_i$ is uniform over instances $x$ in $C$ for all $i = 1, ..., |T|$. We have then:

$$\mathbb{E}_{S \sim \mathcal{D}_i^N}[L_{\mathcal{D}_i}(A(S))] = \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)). \tag{1.60}$$

It is certainly true that:

$$\begin{aligned} \max_{i \in \{1,...,|T|\}} \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)) &\geq \frac{1}{|T|} \sum_{i=1}^{|T|} \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)) = \\ &= \frac{1}{k} \sum_{j=1}^{k} \frac{1}{|T|} \sum_{i=1}^{|T|} L_{\mathcal{D}_i}(A(S_j^i)) \geq \\ &\geq \min_{j \in \{1,...,k\}} \frac{1}{|T|} \sum_{i=1}^{|T|} L_{\mathcal{D}_i}(A(S_j^i)), \end{aligned} \tag{1.61}$$

since the maximum is larger or equal to the average which is larger or equal to the minimum. Fix now some $j \in \{1, ..., k\}$, denote $S_j = (x_1, ..., x_N)$ and let $v_1, ..., v_p$ be the examples in $C$ which do not appear in $S_j$, where clearly $p \geq N$. Then, for any $h : C \mapsto \{0,1\}$ and any $i = 1, ..., |T|$ we have:

$$L_{\mathcal{D}_i}(h) = \frac{1}{2N} \sum_{x \in C} \mathbb{1}_{[h(x) \neq f_i(x)]} \geq \frac{1}{2N} \sum_{r=1}^{p} \mathbb{1}_{[h(v_r) \neq f_i(v_r)]} \geq \frac{1}{2p} \sum_{r=1}^{p} \mathbb{1}_{[h(v_r) \neq f_i(v_r)]}. \tag{1.62}$$

Therefore we have:

$$\begin{aligned} \frac{1}{|T|} \sum_{i=1}^{|T|} L_{\mathcal{D}_i}(A(S_j^i)) &\geq \frac{1}{|T|} \sum_{i=1}^{|T|} \frac{1}{2p} \sum_{r=1}^{p} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} = \\ &= \frac{1}{2p} \sum_{r=1}^{p} \frac{1}{|T|} \sum_{i=1}^{|T|} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} \geq \\ &\geq \frac{1}{2} \min_{r \in \{1,...,p\}} \frac{1}{|T|} \sum_{i=1}^{|T|} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]}. \end{aligned} \tag{1.63}$$

Fixing some $r \in \{1, ..., p\}$ we can partition the functions $f_1, ..., f_{|T|}$ into $|T|/2$ disjoint pairs s.t. for each pair $(f_i, f_{i'})$ and any $c \in C$, $f_i(c) \neq f_{i'}(c)$ if and only if $c = v_r$. Notice that $|T|$ is always even, then, for any $f_i$ which labels $C \setminus \{v_r\}$ there is $f_{i'}$ which labels $C \setminus \{v_r\}$ in the exact same way but which labels $v_r$ differently, thus we can split them in the pairs as mentioned. Since each pair $(f_i, f_{i'})$ differs only at $c = v_r$ we have $S_j^i = S_j^{i'}$, because $v_r$ does not belong to $S_j$. This implies that, over the pair $(f_i, f_{i'})$:

$$\mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} + \mathbb{1}_{[A(S_j^{i'})(v_r) \neq f_{i'}(v_r)]} = 1, \tag{1.64}$$

and in particular:

$$\sum_{i=1}^{|T|} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} = \frac{|T|}{2}. \tag{1.65}$$

Since the result is the same for any $r \in \{1, ..., p\}$ we obtain:

$$\frac{1}{|T|} \sum_{i=1}^{|T|} L_{\mathcal{D}_i}(A(S_j^i)) \geq \frac{1}{4}. \tag{1.66}$$

Combining with equation (1.61), and since what we saw was valid for any $j \in \{1, ..., k\}$, we get:

$$\max_{i \in \{1,...,|T|\}} \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)) \geq \min_{j \in \{1,...,k\}} \frac{1}{|T|} \sum_{i=1}^{|T|} L_{\mathcal{D}_i}(A(S_j^i)) \geq \frac{1}{4}. \tag{1.67}$$

And finally, using equation (1.60) we obtain:

$$\max_{i \in \{1,...,|T|\}} \mathbb{E}_{S \sim \mathcal{D}_i^N}[L_{\mathcal{D}_i}(A(S))] \geq \frac{1}{4}. \tag{1.68}$$

Which implies that, for any learner $A$ there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0, 1\}$ and labelling function $f : \mathcal{X} \mapsto \{0, 1\}$ s.t., given $N$ i.i.d. samples from $\mathcal{D}$ as a training set we have $L_{\mathcal{D}}(f) = 0$ and:

$$\mathbb{E}_{S \sim \mathcal{D}^N}[L_{\mathcal{D}}(A(S))] \geq \frac{1}{4}. \tag{1.69}$$

Since $L_{\mathcal{D}}(A(S))$ is a random variable in $[0, 1]$ and using $\mu = 1/4$, i.e. the smallest possible value for its expectation, considering $a = 1/8$ we get immediately from Lemma 1.2.1:

$$\mathbb{P}_{S \sim \mathcal{D}^N}[L_{\mathcal{D}}(A(S)) \geq 1/8] \geq 1/7. \tag{1.70}$$

■

**Remark 1.2.1.** *Notice that the proof restricts its attention to a finite set $C$ which automatically defines a finite hypothesis class $\mathcal{H}$ of all functions from $C$ to $\{0, 1\}$ and is thus PAC learnable. However, we are limiting the training set size to $|C|/2$ so there is no contradiction.*

Thus, in the absence of any inductive bias, for any learner there exists a distribution where it fails, in the sense of having too large a risk, where another learner would succeed. A trivial successful learner in this case could be one where we take $\mathcal{H} = \{f\}$, where $f$ is the actual labelling function. Similarly, ERM with respect to a finite hypothesis class $\mathcal{H}$, and for which $N$ satisfies $N \geq 8 \log(7|\mathcal{H}|/6)$ (i.e. as appears in the theorem, $\epsilon = 1/8$ and $\delta = 6/7$), would also succeed, in the sense of being PAC learnable.

Over an infinite input domain $\mathcal{X}$, if we consider an ERM predictor over the hypothesis class of all functions $f : \mathcal{X} \mapsto \{0, 1\}$, which represents a lack of any inductive bias, the no-free-lunch theorem immediately implies the following corollary:

**Corollary 1.2.3.** *Let $\mathcal{X}$ be an infinite domain and let $\mathcal{H} = \{f \mid f : \mathcal{X} \mapsto \{0, 1\}\}$. Then $\mathcal{H}$ is not PAC learnable.*

*Proof.* Suppose the hypothesis class $\mathcal{H}$ is PAC learnable. Then, choosing $\epsilon < 1/8$ and $\delta < 1/7$ there exists a learning algorithm $A$ and an integer $N(\epsilon, \delta)$ s.t., for any distribution $\mathcal{D}$ over $\mathcal{X} \times \{0, 1\}$ and for some labelling function $f : \mathcal{X} \mapsto \{0, 1\}$, s.t. $f \in \mathcal{H}$ and $L_{\mathcal{D}}(f) = 0$ (realizability assumption), with probability greater than $1-\delta$ of sampling $S$, with $N(\epsilon, \delta)$ i.i.d. elements from $\mathcal{D}$, we have $L_{\mathcal{D}}(A(S)) \leq \epsilon$.

By the no-free-lunch theorem however, because $|\mathcal{X}|$ is infinite ($|\mathcal{X}| > 2N(\epsilon, \delta)$), there exists a distribution $\mathcal{D}'$ over $\mathcal{X} \times \{0, 1\}$ where, with probability $1/7 > \delta$ when sampling $S$ from $\mathcal{D}'$, with $N(\epsilon, \delta)$ i.i.d. elements, we have $L_{\mathcal{D}'}(A(S)) > 1/8 > \epsilon$. Since PAC learnability is valid for any distribution, and in particular for $\mathcal{D}'$ we reach a contradiction which proves the result. ■

## 1.3 Bias-complexity trade-off

In order to prevent this failure of the learner (in the sense of not being PAC learnable over such a hypothesis class) we need to introduce an inductive bias. This is reflected in the choice of hypothesis class $\mathcal{H}$ the learner has access to. Contrastingly, we would like that $\mathcal{H}$ be rich enough to contain the predictor which has no generalization error (in the PAC setting) or (in the agnostic PAC setting) that the smallest possible generalization error of predictors in class $\mathcal{H}$ is small enough, while simultaneously, $\mathcal{H}$ should not be too large in order to prevent overfitting. In particular, by the corollary above, the largest possible class, over an infinite domain, is not PAC learnable. This conflicting requirements are known as the bias-complexity tradeoff as we have mentioned before.

To better understand the problem we can split the error of an $\text{ERM}_{\mathcal{H}}$ learner in the following two terms:

$$L_{\mathcal{D}}(h_S) = \epsilon_{app} + \epsilon_{est}, \text{ with } \epsilon_{app} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h), \;\; \epsilon_{est} = L_{\mathcal{D}}(h_S) - \epsilon_{app}. \tag{1.71}$$

- **Approximation error ($\epsilon_{app}$):** This is the minimum possible true risk by restricting ourselves to the predictor class $\mathcal{H}$ and is a measure of the inductive bias. Picking a larger class $\mathcal{H}$ will either reduce this error or keep it the same.
- **Estimation error ($\epsilon_{est}$):** Since our predictor $h_S$ is obtained by ERM, i.e. it minimizes the empirical risk and not the true risk, there will be a difference between the risk of predictor $h_S$ and the risk of the best possible predictor within $\mathcal{H}$. This error depends on the size of the training set $S$ and on the size of the hypothesis class $\mathcal{H}$. In particular, for a finite hypothesis class, via equation (1.17) the bound on the probability of having a large risk $\delta$ increases with $|\mathcal{H}|$ and decreases with $N$.

We see then that these two error terms are competing in the sense that the approximation error benefits (i.e. is possibly reduced) by increasing the hypothesis class $\mathcal{H}$ while the estimation error benefits by having a smaller hypothesis class $\mathcal{H}$ and a larger sample size. Intuitively, if $\mathcal{H}$ has a "large" size, over a "small" dataset we might pick a predictor which does not accurately represent the true labelling function, despite having a low empirical risk (overfitting). On the other hand, if $\mathcal{H}$ has a "small" size, we might be unable to find a predictor which accurately represents the actual labelling function, this results in high values for both the risk and empirical risk (underfitting).

## 1.4 The VC-Dimension

In this section we introduce the notion of VC-dimension (where VC stands for Vapnik-Chervonenkis, the authors of the theory) and its relation to the fundamental Theorem in statistical learning theory. Previously we have seen that finite hypothesis classes are PAC learnable while the set of all possible functions, due to the no-free-lunch Theorem and within a specified (infinite) domain, is not. This new tool will allow us to understand which hypothesis classes $\mathcal{H}$ are PAC learnable and what is the associated sample complexity, in particular extending the previous results to infinite hypothesis classes.

In order to show that finiteness is a sufficient but not necessary condition for a class to be PAC learnable we have the following example:

**Example 1.4.1.** *Let $\mathcal{X} = \mathbb{R}$ and consider $\mathcal{H}$ to be the set of threshold functions of the form $\mathcal{H} = \{h_a,\, a \in \mathbb{R} \,|\, h_a(x) = \mathbb{1}_{[x<a]}\}$. This hypothesis class has infinite size and, via the following lemma, $\mathcal{H}$ is PAC learnable with the ERM algorithm.*

**Lemma 1.4.1.** *Let $\mathcal{H} = \{h_a,\, a \in \mathbb{R} \,|\, h_a(x) = \mathbb{1}_{[x<a]} \in \{0,1\}\}$. Then, $\mathcal{H}$ is PAC learnable, using the ERM algorithm, with sample complexity $N_{\mathcal{H}}(\epsilon, \delta) \leq \lceil \log(2/\delta)/\epsilon \rceil$.*

*Proof.* Let $\mathcal{X} \subset \mathbb{R}$ and $\mathcal{D}$ be a distribution over $\mathcal{X}$ labeled according to a threshold function $h^*(x) = \mathbb{1}_{[x<a^*]}$, s.t. $L_{\mathcal{D}}(h^*) = 0$ (realizability assumption), and let $a_0 < a^* < a_1$ be such that:

$$\mathbb{P}_{x\sim\mathcal{D}}[x \in (a_0, a^*)] = \mathbb{P}_{x\sim\mathcal{D}}[x \in (a^*, a_1)] = \epsilon, \tag{1.72}$$

where, if $\mathcal{D}(-\infty, a^*) \leq \epsilon$ we let $a_0 = -\infty$ and if $\mathcal{D}(a^*, \infty) \leq \epsilon$ we let $a_1 = \infty$.

Given a training set $S$, with $N$ datapoints sampled i.i.d. from $\mathcal{D}$, define $b_0 := \max\{x : (x, 1) \in S\}$ and $b_1 := \min\{x : (x, 0) \in S\}$ where, if there is no positive labeled datapoint we set $b_0 = -\infty$ and if there is no negative labeled datapoint we set $b_1 = \infty$. Naturally, by how we defined the labelling of $\mathcal{D}$, we must have $b_0 \leq a^* \leq b_1$.

Suppose that, via ERM, we obtain the hypothesis $h_S = \mathbb{1}_{[x<b_S]}$ then we must have $b_0 \leq b_S \leq b_1$ (any other value would result in higher empirical risk, consider square loss for example). Therefore, a sufficient condition for $L_{\mathcal{D}}(h_S) \leq \epsilon$ is that $b_S \in [a_0, a_1]$ or that $b_0 \geq a_0$ and $b_1 \leq a_1$. Equivalently we can write:

$$\mathbb{P}_{S\sim(\mathcal{D},h^*)}[L_{\mathcal{D}}(h_S) > \epsilon] \leq \mathbb{P}_{S\sim(\mathcal{D},h^*)}[b_0 < a_0 \vee b_1 > a_1]. \tag{1.73}$$

By subadditivity we have:

$$\mathbb{P}_{S\sim(\mathcal{D},h^*)}[L_{\mathcal{D}}(h_S) > \epsilon] \leq \mathbb{P}_{S\sim(\mathcal{D},h^*)}[b_0 < a_0] + \mathbb{P}_{S\sim(\mathcal{D},h^*)}[b_1 > a_1]. \tag{1.74}$$

The event $b_0 < a_0$ occurs if and only if there are no datapoints in $S$ in the interval $(a_0, a^*)$, i.e.:

$$\mathbb{P}_{S\sim(\mathcal{D},h^*)}[b_0 < a_0] = \mathbb{P}_{S\sim(\mathcal{D},h^*)}[\forall(x, y) \in S, x \notin (a_0, a^*)] = (1 - \epsilon)^N \leq e^{-\epsilon N}. \tag{1.75}$$

Similarly, $b_1 > a_1$ occurs if and only if there are no datapoints in $S$ in the interval $(a^*, a_1)$, i.e.:

$$\mathbb{P}_{S\sim(\mathcal{D},h^*)}[b_1 > a_1] = \mathbb{P}_{S\sim(\mathcal{D},h^*)}[\forall(x, y) \in S, x \notin (a^*, a_1)] = (1 - \epsilon)^N \leq e^{-\epsilon N}. \tag{1.76}$$

By taking $N \geq \log(2/\delta)/\epsilon$ each of the terms in (1.75) and (1.76) is smaller than $\delta/2$ and we conclude that:

$$\mathbb{P}_{S\sim(\mathcal{D},h^*)}[L_{\mathcal{D}}(h_S) > \epsilon] \leq \delta. \tag{1.77}$$

■

The notion of VC-dimension turns out to be the correct property to characterize (PAC) learnability. To introduce it we need to consider first the following definitions:

**Definition 1.4.1.** ***(Restriction of $\mathcal{H}$ to $C$)***
*Let $\mathcal{H} = \{h \,|\, h : \mathcal{X} \mapsto \{0, 1\}\}$ and let $C = \{c_1, ..., c_N\} \subset \mathcal{X}$. The restriction of $\mathcal{H}$ to $C$ is defined as the following set of functions:*

$$\mathcal{H}_C = \{h' : C \mapsto \{0, 1\} \,|\, h'(c_1, ..., c_N) = (h(c_1), ..., h(c_N)),\ h \in \mathcal{H}\}, \tag{1.78}$$

*where each function in $\mathcal{H}_C$ is represented by a vector in $\{0, 1\}^N$.*

**Definition 1.4.2.** ***(Shattering)***
*We say that a hypothesis class $\mathcal{H}$, defined on $\mathcal{X}$, shatters a finite set $C \subset \mathcal{X}$ if the restriction of $\mathcal{H}$ to $C$ is the set of all possible functions from $C$ to $\{0, 1\}$, and thus $|\mathcal{H}_C| = 2^{|C|}$.*

The usefulness of the shattering concept derives from the proof of the no-free-lunch Theorem. In the proof, we pick a finite set $C$ and construct an adversarial distribution over the hypothesis class $\mathcal{H}$, which contains any possible function defined on $C$. In particular, if $\mathcal{H}$ shatters $C$ then the adversarial function is not restricted and can be any function $f : C \mapsto \{0, 1\}$. It is this freedom in the choice of function that allows the adversarial to exist. With this new concept of shattering we have the following corollary to the no-free-lunch Theorem.

**Corollary 1.4.2.** *Let $\mathcal{H} \subseteq \{f \mid f : \mathcal{X} \mapsto \{0,1\}\}$ be a hypothesis class, $N$ be the size of a training set $S$ and suppose that $\exists\, C \subset \mathcal{X}$, of size $2N$, that is shattered by $\mathcal{H}$. Then, for any learning algorithm $A$, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ and a predictor $h \in \mathcal{H}$ s.t. $L_{\mathcal{D}}(h) = 0$ but, with probability of at least $1/7$ over the choice of $S \sim \mathcal{D}^N$, we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

Intuitively, if the hypothesis class $\mathcal{H}$ contains any possible function from a set $C$ to $\{0,1\}$, of size $2N$, i.e. $\mathcal{H}$ shatters $C$, if we have access to only $N$ i.i.d. samples from $C$ then, over the missing points, the class $\mathcal{H}$ is equally good at explaining all possible outcomes and as such may generalize badly for a specific distribution. With this discussion we can now introduce the concept of VC-dimension:

**Definition 1.4.3.** ***(VC dimension)***
*Given a hypothesis class $\mathcal{H}$, the VC-dimension of $\mathcal{H}$ is the maximal size of a set $C \subset \mathcal{X}$ that can be shattered by $\mathcal{H}$, we denote it by $VCdim(\mathcal{H})$. We set $VCdim(\mathcal{H}) = \infty$ if $\mathcal{H}$ can shatter sets of arbitrarily large size.*

**Remark 1.4.1.** *Any non-empty class vacuously shatters an empty set, of size zero, therefore $VCdim \geq 0$. Furthermore, it is easy to see that $VCdim(\mathcal{H}) = 0$ if and only if the hypothesis class $\mathcal{H}$ consists of a single constant function. In particular, if there are two distinct functions there exists at least one point where they differ.*

The concepts developed so far allows one to establish the following fundamental result.

**Theorem 1.4.3.** ***(The fundamental Theorem of statistical learning** - Theorem 6.7 of [SSBD14])*
*Let $\mathcal{H}$ be a hypothesis class comprised of functions defined on $\mathcal{X}$ with values in $\{0,1\}$ and consider the 0-1 loss function over a binary classification task. Then, the following are equivalent:*

1. *$\mathcal{H}$ has the uniform convergence property.*
2. *Any ERM rule is a successful agnostic PAC learner for $\mathcal{H}$.*
3. *$\mathcal{H}$ is agnostic PAC learnable.*
4. *$\mathcal{H}$ is PAC learnable.*
5. *Any ERM rule is a successful PAC learner for $\mathcal{H}$.*
6. *$\mathcal{H}$ has finite VC-dimension.*

Before looking at the proof we require a new definition and a few other results. The notion of a growth function measures the number of possible functions within $\mathcal{H}$ when restricted to sets of finite size:

**Definition 1.4.4.** ***(Growth function)***
*Let $\mathcal{H}$ be a hypothesis class. The growth function $\tau_{\mathcal{H}} : \mathbb{N} \mapsto \mathbb{N}$ is defined as:*

$$\tau_{\mathcal{H}}(N) = \max_{C \subset \mathcal{X} : |C| = N} |\mathcal{H}_C|. \tag{1.79}$$

If $\text{VCdim}(\mathcal{H}) = d$, because $\mathcal{H}$ shatters sets of size d, for any $N \leq d$ we have $\tau_{\mathcal{H}}(N) = 2^N$ The following lemma, due to Sauer, Shelah and Perles, shows that for $N > d + 1$ the growth function increases polynomially instead of exponentially:

**Lemma 1.4.4.** *Let $\mathcal{H}$ be a hypothesis class with $VCdim(\mathcal{H}) \leq d < \infty$. Then, for all $N$, $\tau_{\mathcal{H}}(N) \leq \sum_{i=0}^{d} \binom{N}{i}$. In particular, if $N > d + 1$ one has $\tau_{\mathcal{H}}(N) \leq \left(\frac{eN}{d}\right)^d$.*

*Proof.* See Lemma 6.10 in Section 6.5.1 of [SSBD14]. ■

The following theorem which gives a uniform bound on the difference between the true risk and empirical risk as a function of the growth function.

**Theorem 1.4.5.** *Let $\mathcal{H}$ be a hypothesis class and let $\tau_{\mathcal{H}}$ be its growth function. Then, for every distribution $\mathcal{D}$ and every $\delta \in (0,1)$, with probability at least $1-\delta$ over the choice of a training set $S \sim \mathcal{D}^N$, for any $h \in \mathcal{H}$ we have:*

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{\log(\tau_{\mathcal{H}}(2N))}}{\delta\sqrt{2N}} \tag{1.80}$$

*Proof.* See Theorem 6.11 at the end of Section 6.5.2 of [SSBD14]. ■

And we will also make use the following lemma:

**Lemma 1.4.6.** *Let $a \geq 1$ and $b > 0$. Then, $x \geq 4a\log(2a) + 2b \implies x \geq a\log(x) + b$.*

*Proof.* See lemma A.2 in the appendix of [SSBD14]. ■

*Proof.* (of Theorem 1.4.3) Let us look at each implication:

- (1.⇒ 2.) This follows from Corollary 1.1.6 which follows from Lemma 1.1.5.
- (2.⇒ 3.) Trivially since if ERM is a successful learner the class must be agnositc PAC learnable.
- (3.⇒ 4.) Trivially since if $\mathcal{H}$ is agnostic PAC learnable then, if the realizability assumption holds, this reduces to the class $\mathcal{H}$ being PAC learnable. Agnostic PAC learnability is a generalization as was mentioned before.
- (2.⇒ 5.) Trivially, for the same reasons as the previous implication.
- (4.⇒ 6.) Suppose that $\text{VCdim}(\mathcal{H}) = \infty$ then, given a training set of size $N$ there exists a shattered set of size $2N$ therefore, by corollary 1.4.2, $\mathcal{H}$ is not PAC learnable.
- (5.⇒ 6.) This implies item 4 which in turn implies item 6 by the previous implication.
- (6.⇒ 1.) This implication is the more involved part. The idea is that, even if $\mathcal{H}$ is infinite, over a finite set $C \subset \mathcal{X}$, $|\mathcal{H}_C|$ grows polynomially with $d$ ($O(|C|^d)$) and this turns out to be enough for the hypothesis class to have the uniform convergence property.

  Let $\mathcal{H}$ be a hypothesis class s.t. $\text{VCdim}(\mathcal{H}) = d$, we want to prove that $\mathcal{H}$ has the uniform convergence property. Since we have freedom in selecting $N$ (we just need to show one such $N(\delta,\epsilon)$ exists) from Lemma 1.4.4, taking $N > d+1$, we have $\tau_{\mathcal{H}}(2N) \leq \left(\frac{2eN}{d}\right)^d$. Using Theorem 1.4.5, for any $\mathcal{D}$ and $\delta \in (0,1)$, with probability at least $1-\delta$ over the choice of a training set $S \sim \mathcal{D}^N$ we have:

  $$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{d\log\left(\frac{2eN}{d}\right)}}{\delta\sqrt{2N}} \tag{1.81}$$

  For simplicity, since we are free to take $N$ as large as we want, consider $N$ large enough s.t. $\sqrt{d\log(2eN/d)} \geq 4$, then we have:

  $$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{8}{\delta\sqrt{2N}} \leq \frac{1}{\delta}\sqrt{\frac{32}{N}} \leq \frac{1}{\delta}\sqrt{\frac{2d\log\left(\frac{2eN}{d}\right)}{N}}. \tag{1.82}$$

  To bound this quantity by $\epsilon$, which is what we require to prove the result, we need to show that:

  $$N \geq \frac{2d}{(\delta\epsilon)^2}\bigg(\log(N) + \log(2e/d)\bigg). \tag{1.83}$$

If $d = 0$, by the remark made previously, $\mathcal{H}$ has a single constant function. Thus, we have no limitation on the size of the training sample $N$ (it needs to hold for a single function) and we can guarantee that, for any choice of $\epsilon > 0$, there is a large enough $N$ such that $|L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon$. A formal justification can be given using the law of large numbers since $L_S(h) \to L_{\mathcal{D}}(h)$ when $N \to \infty$.

If $d \geq 1$, taking $a = 2d/(\epsilon\delta)^2$ and $b = d\log(2e/d)/(\epsilon\delta)^2$, then $a \geq 1$ and, if $b > 0$, by Lemma 1.4.6 we obtain the condition in equation (1.83) to finally obtain:

$$\forall h \in \mathcal{H},\ |L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon. \tag{1.84}$$

Suppose $b < 0$, let $C := 2d/(\delta\epsilon)^2 \geq 1$, we require $N$ such that:

$$N \geq C\bigg(\log(N) + \log\bigg(\frac{4e}{C(\delta\epsilon)^2}\bigg)\bigg), \tag{1.85}$$

where $\log\left(\frac{4e}{C(\delta\epsilon)^2}\right) < 0$. We are sure to find $N'$ such that $N' \geq C\log(N')$ (linear vs logarithmic growth) therefore we will surely satisfy the equation above and the result follows in general.

■

The following theorem is an extension of the previous result which provides bounds for the sample complexities as a function of the VC-dimension of the hypothesis class.

**Theorem 1.4.7.** *(**Quantitative version** - Theorem 6.8 of [SSBD14])*
*Let $\mathcal{H}$ be a hypothesis class comprised of functions defined on $\mathcal{X}$ with values in $\{0, 1\}$ and consider the 0-1 loss function. Assuming that $VCdim(\mathcal{H}) = d < \infty$ then there are $C_1, C_2 \in \mathbb{R}$ s.t.:*

1. *$\mathcal{H}$ has the uniform convergence property with sample complexity:*

$$C_1\frac{d + \log(1/\delta)}{\epsilon^2} \leq N_{\mathcal{H}}^{UC}(\epsilon, \delta) \leq C_2\frac{d + \log(1/\delta)}{\epsilon^2} \tag{1.86}$$

2. *$\mathcal{H}$ is agnostic PAC learnable with sample complexity:*

$$C_1\frac{d + \log(1/\delta)}{\epsilon^2} \leq N_{\mathcal{H}}(\epsilon, \delta) \leq C_2\frac{d + \log(1/\delta)}{\epsilon^2} \tag{1.87}$$

3. *$\mathcal{H}$ is PAC learnable with sample complexity:*

$$C_1\frac{d + \log(1/\delta)}{\epsilon} \leq N_{\mathcal{H}}^{UC}(\epsilon, \delta) \leq C_2\frac{d + \log(1/\delta + \log(\delta))}{\epsilon} \tag{1.88}$$

*Proof.* The proof is the whole of Chapter 28 in [SSBD14]. ■

**Remark 1.4.2.** *Although being stated for the case of binary classification similar results hold in other cases, such as regression with absolute and squared loss. However the theorem does not hold for all possible learning tasks, there are examples where the uniform convergence property does not hold for a particular task but still it is possible to learn that particular task (see exercise 13.2 in [SSBD14]).*

**Example 1.4.2.** *(Calculating the VC dimension)*

- ***Threshold functions:*** *Let* $\mathcal{H} = \{h_a,\, a \in \mathbb{R} \,|\, h_a(x) = \mathbb{1}_{[x<a]} \in \{0,1\}\}$ *an consider a set with a single element* $C = \{c_1\}$*. Then, if we take* $a = c_1 + 1$ *we have* $h_a(c_1) = 1$ *while if* $a = c_1 - 1$ *we have* $h_a(c_1) = 0$*. Thus,* $\mathcal{H}$ *shatters the set* $C$ *since the restriction of* $\mathcal{H}$ *to* $C$ *contains all possible functions of the form* $h' : C \mapsto \{0,1\}$*. On the other hand if we consider a set with two elements* $C' = \{c_1, c_2\}$*, with* $c_1 < c_2$ *then* $C$ *is not shattered by* $\mathcal{H}$ *since if* $c_1$ *is labeled as* $0$ *this implies* $a < c_1 < c_2$ *and thus* $c_2$ *can not be labeled* $1$*. Thus,* $\mathcal{H}$ *does not shatter* $C'$ *and we have* $VCdim(\mathcal{H}) = 1$*.*

- ***Intervals over*** $\mathbb{R}$***:*** *Let* $\mathcal{H} = \{h_{a,b},\, a, b \in \mathbb{R} \,|\, h_{a,b}(x) = \mathbb{1}_{[a<x<b]} \in \{0,1\}\}$*. Consider a set with two elements* $C = \{c_1, c_2\}$ *s.t.* $c_1 < c_2$*. Then* $\mathcal{H}$ *shatter* $C$ *since we can take* $h_{a,b}$ *s.t.* $a < c_1 < b < c_2$ *leading to the label* $(1,0)$*. We can take* $a < c_1 < c_2 < b$ *leading to the labels* $(1,1)$*. We can also consider* $c_1 < a < c_2 < b$ *leading to the labels* $(0,1)$ *and finally, by taking* $a < b < c_1 < c_2$ *or* $c_1 < c_2 < a < b$ *we have the labels* $(0,0)$*. Over an arbitrary set of three elements* $C' = \{c_1, c_2, c_3\}$ *we can not have the labels* $(1,0,1)$ *since if the edges are labeled with* $1$ *this automatically forces* $h_{a,b}(c_2) = 1$*. Thus,* $\mathcal{H}$ *does not shatter* $C'$ *and we have* $VCdim(\mathcal{H}) = 2$*.*

- ***Finite classes:*** *Let* $\mathcal{H}$ *be a finite hypothesis class and consider an arbitrary finite set* $C \subset \mathcal{X}$*. In a binary classification setting, the number of all possible functions from* $C \mapsto \{0,1\}$ *is* $2^{|C|}$ *therefore, if* $|\mathcal{H}| < 2^{|C|}$*,* $\mathcal{H}$ *can not shatter* $C$*. This implies immediately that* $VCdim(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$*.*

  **Remark 1.4.3.** *The number of parameters defining the hypothesis class is not, necessarily, equal to the VCdim. For a particular example showing this see exercise 6.8 of [SSBD14].*

This concludes our brief introduction to statistical learning theory. We completely focused on the statistical part of learning but, an equally important factor in practice is the computational complexity involved in learning a particular task. A discussion of this topic can be seen in Chapter 8 of [SSBD14].

## 1.5 Summary

In this first chapter we formally introduce the statistical learning problem which, intuitively, consists on modelling some unknown phenomena via interpolation with recourse to a discrete and finite set of datapoints, the **training data**. This model, the **predictor**, is given as the output of some **learning algorithm** which has access to the training data but is unaware of the underlying phenomena generating the data (which is assumed to follow some unknown distribution).

Defining a measure of success of the learner, **true risk**, we consider a particular learning paradigm, **empirical risk minimization** (ERM) and try to answer the most fundamental question: How does the **empirical risk**, a measure of learning success over the training data, relates to the **true risk**, a measure of success over the (unknown) generating distribution?

We saw that it was possible, using ERM, to find a predictor with lowest possible empirical error but for which the true error could be made arbitrarily "bad", by essentially memorizing all data and constructing an "adversarial" distribution to the predicted labels on unseen data. This leads to the **overfitting** problem.

At this point, instead of abandoning the ERM paradigm, we introduce some constraints on the problem, called an **inductive bias**, which is reflected in the choice of the **hypothesis class**. Essentially, before seeing the data, we restrict the search for a predictor within a certain class of functions. This leads to one of the main concepts of the chapter:

- **PAC Learning**: This is a characterization of the hypothesis class. Informally we think of a certain hypothesis class as PAC learnable if there exists at least one learning algorithm such that we can make the true risk as small as we want, with as large confidence as we want over the choice of the training data, by providing our learning algorithm a sufficiently large number of samples.

With this notion we were able to show that any finite hypothesis class is PAC learnable. The PAC learnable definition however, requires that, within the hypothesis class, there exists a function which has the same true risk as the actual labelling function, which is known as the **realizability assumption**. Relaxing this condition, and introducing the concept of a general **loss function** and a general learning task (we had been considering only binary classification at this point), leads to a generalization of PAC learning known as **agnostic PAC learning**.

- **Agnostic PAC Learning**: This is the same idea as PAC learning with the introduction of a joint distribution over both input space and the label set. The true risk will now have a minimum value which is the minimum possible true risk over predictors found within the hypothesis class, with the minimal possible error being given by the Bayes optimal predictor.

Next, we introduce a sufficient (and necessary by the fundamental theorem of statistical learning) condition, for a hypothesis class to be agnostic PAC learnable. This is known as **uniform convergence** and is a property of the hypothesis class which, over the agnostic PAC learning conditions, gives a bound on the difference between the empirical risk and the true risk. Via this definition we then prove that a finite hypothesis class enjoys this uniform convergence property and is therefore agnostic PAC learnable, in particular using the ERM algorithm.

The next major result, in the context of binary classification, is known as the **no-free-lunch theorem**.

- **No-free-lunch theorem:** This theorem states that there is no learning algorithm which, for any possible distribution over the input and label sets, and having access to a fixed training set of size $N$, is capable of providing, with high probability, a predictor which generalizes well, i.e. has low true risk. The key point here is that the learner is working with a fixed set of a certain size $N$ and we have total freedom over the possible underlying distribution and labelling function over a set of $2N$ points, within the hypothesis class. Over a set of size $2N$ the learner does not know what happens in, at least, $N$ possible pairs over the input and label space. Thus, with some (lower bounded) probability over sampling a training set $S$ of size $N$, whatever the output of the learning algorithm, there will be a certain distribution and labelling function for which the predictor will have poor generalization performance.

This result shows that, in the absence of any inductive bias or prior knowledge, any learning algorithm that we use may fail. In particular we have seen that, over an infinite input domain, the hypothesis class of all functions in this domain is not PAC learnable. This leads us to what is known as the **bias-complexity tradeoff**.

- **Bias-complexity tradeoff:** We saw that restricting the hypothesis class reduces the chance of overfitting. In particular, with a finite hypothesis class and under the realizability assumption, we are "guaranteed" to not overfit. However, approximating the realizability assumption requires an increased flexibility in the hypothesis class, which implies increasing the size of $\mathcal{H}$. Having a "small" hypothesis class means a strong inductive bias, an increased belief that the real function behaves as those present in the hypothesis class. This reduces the overfitting problem but increases the chance of **underfitting**, since the real function might not be well represented by those found in the hypothesis class. By increasing the size of the hypothesis class we are, in general, increasing the complexity of the functions it can represent, this reduces the chance of underfitting but increases the chance of overfitting. Thus, in general, a larger bias implies a lower complexity and vice-versa.

By the results we have up to this point one might be tempted to associate learnability with finiteness of the hypothesis class. However, this is not the case. Even though finiteness is a sufficient condition, the last concept we introduce, that of **VC dimension**, is the property that determines whether a hypothesis class is in fact PAC learnable. Understanding the proof of the no-free-lunch theorem is paramount in order to understand the introduction of the VC-dimension concept and its role in defining learnability.

- **VC-dimension:** In the no-free-lunch theorem proof the key property in constructing an "adversarial" distribution and labelling function is the fact that the hypothesis class contains all possible functions defined within a chosen set. The VC dimension of a hypothesis class characterizes precisely the size of the largest possible set for which the hypothesis class contains all possible labelling functions.

Finally, the major result of the chapter, the **fundamental theorem of statistical learning**, relates the notion of a hypothesis class being PAC learnable, in particular via ERM, with the finiteness of the VC-dimension. Furthermore, the VC dimension provides bounds for the **sample complexity**, i.e. the minimum number of sampled datapoints required for PAC learnable conditions.

# Chapter 2

# Linear Models for Supervised Learning

In this chapter we consider the learning paradigm of empirical risk minimization (ERM) based on hypothesis classes which consist of families of parametrized functions that depend linearly on the parameters. To this end we will closely follow [Bis06].

In the first section we cover **regression problems** starting with an example of **curve fitting** using polynomial **basis functions**. This is a very pedagogical example which clearly shows the nuisances of overfitting and underfitting discussed in the previous chapter. It further allows for the introduction of several topics which will be discussed in greater detail later. Next, we generalize the previous discussion by considering a general set of basis functions and any possible input and output dimension. In this setting we show how to determine a predictor by maximizing the **likelihood function** which, as we shall see, is equivalent to **least squares**. To control the overfitting problem we will introduce the concept of **regularization** and, in the context of square loss, we inspect how the solution behaves as a function of the **regularizer**. A major part of the chapter introduces the **Bayesian approach** to regression where we see the appearance of an implicit form of regularization. We end this section with an extension of the bias-complexity trade-off, discussed in Chapter 1, which appears in this setting as the **bias-variance decomposition** by considering a frequentist approach via an ensemble of datasets. These last two topics contain each a detailed example (based on those found in [Bis06]) which are fully implemented in Python.

## 2.1 Linear Models for Regression

In a regression problem we are interested in predicting a continuous quantity which typically lives in a subset of $\mathbb{R}$. As a way to introduce several topics, which will be discussed in more detail later, we start by considering a particular example of curve fitting.

### 2.1.1 Example: Polynomial Curve fitting

Suppose we are given a training set with $N$ observations of a variable $x \in \mathbb{R}$, given by $\boldsymbol{x} = \{x_1, ..., x_N\}$, together with its target values $t \in \mathbb{R}$ given by $\boldsymbol{t} = \{t_1, ..., t_N\}$. This set is generated under some distribution over input space and is "labelled" by an underlying function with the possible addition of a noise term. The goal is to find the underlying function labelling the data, or at least a good approximation, and to do so, we fit the data to a polynomial function with the following form:

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j. \tag{2.1}$$

One way to determine the optimal $M + 1$ parameters $\boldsymbol{w} = (w_0, ..., w_M)$ is by following the paradigm of empirical risk minimization with square loss as a loss function. Denoting $E(\boldsymbol{w})$ as the empirical

risk as a function of the parameters we have:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2. \tag{2.2}$$

By minimizing this error we obtain a parameter vector $\boldsymbol{w}^*$ which defines the predictor $y(x, \boldsymbol{w}^*)$. However, there is still lurking around the problem of choosing the best possible $M$, the degree of the polynomial in (2.1). By choosing a large enough $M$ we increase the flexibility of our model and also the size of the hypothesis class. As we have discussed in Chapter 1, this may achieve exactly zero error on the training set but have poor performance on a test set, due to overfitting. On the other hand, if $M$ is too small, we may not be able to accurately represent the underlying labeling function and the model ends up underfitting. Note that the adjectives large and small are relative to the complexity of the underlying function and the amount of data we have at our disposal. Let us consider a specific example to illustrate this discussion.

Suppose we have a distribution over input space $\mathcal{X} = [0, 1]$, which follows $x \sim \text{Uni}[0, 1]$, and let the underlying labelling function be $g(x) = \sin(2\pi x)$. To the value of $g$ we add a Gaussian noise term $\epsilon \sim \mathcal{N}(0, 0.3^2)$ such that the target variable $t$ is given by $t(x) = g(x)+\epsilon$. Our dataset consists of $N = 20$ pairs of the form $\{(x_1, t_1), ..., (x_{20}, t_{20})\}$. The following figure (fig. 2.1) shows a fit, considering $M = 1$, $M = 3$ and $M = 19$ respectively, by determining the parameters $\boldsymbol{w}^*$ via minimization of equation (2.2):

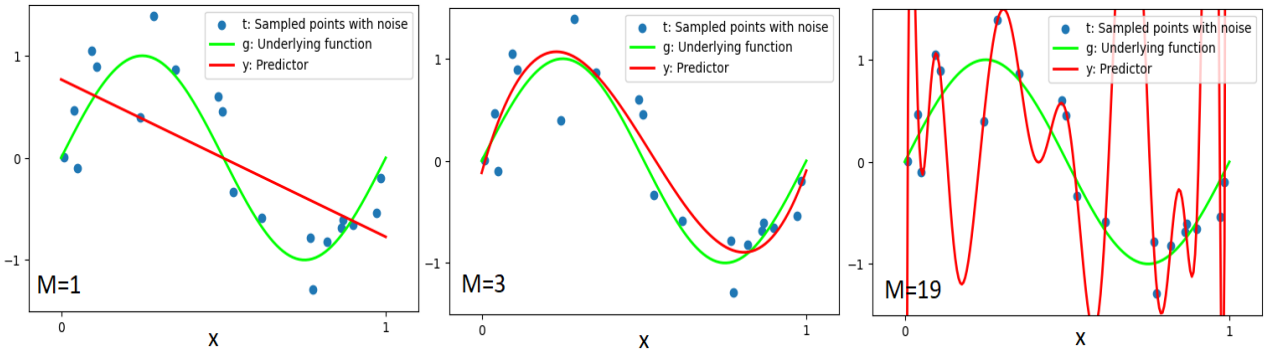


Figure 2.1: Polynomial fit obtained by minimizing the least squares error with $M = 1, 3, 19$.

Following up on the comment above, clearly $M = 1$ is not sufficient to capture the behaviour of the function, we are too biased by forcing a linear model on the variable $x$. On the other hand, for $M = 19$ we perfectly fit the $N = 20$ points resulting in zero empirical risk, since this gives rise to a linear system of equations:

$$t_i = \sum_{j=0}^{M} w_j {x_i}^j, \text{ for } i = 1, ..., 20, \tag{2.3}$$

which allows, in general, to determine a unique vector of parameters $\boldsymbol{w} = (w_0, ..., w_{19})$ satisfying the equations. In this case however the function varies wildly, as seen in panel 3 of figure 2.1, and is clearly overfitting. A way to keep flexibility in the model, but reduce the overfitting problem, is to introduce a **regularization** term. Denoting the regularized version of the empirical risk as $\tilde{E}(\boldsymbol{w})$, with a particular form for the regularizer, we have:

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2, \quad \lambda > 0, \tag{2.4}$$

where $\lambda$ mediates the relative importance between reducing least squares error versus reducing the norm of $\boldsymbol{w}$, i.e. for large values of $\lambda$ we are discouraging the model of obtaining large values for this norm. The motivation for the regularization term comes from the following observation: If we look at the coefficients for each value of $M$, table 2.1, we realize that the increasingly large norms for $\boldsymbol{w}$ are responsible for the rapid fluctuations we saw in fig. 2.1.

| $\boldsymbol{w}$ | M=1 | M=3 | M=10 | M=19 |
|---|---|---|---|---|
| ${w_0}^*$ | -1.54 | 20.41 | 8.57e+04 | -2.19e+09 |
| ${w_1}^*$ | 0.77 | -31.72 | -4.37e+05 | 2.27e+10 |
| ${w_2}^*$ | | 11.39 | 9.56e+05 | -9.89e+10 |
| ${w_3}^*$ | | -0.12 | -1.17e+06 | 2.38e+11 |
| ${w_5}^*$ | | | -4.03e+05 | 2.45e+11 |
| ${w_{10}}^*$ | | | 2.98e-01 | 1.50e+11 |
| ${w_{15}}^*$ | | | | -4.64e+07 |
| ${w_{19}}^*$ | | | | -1.181e+01 |

Table 2.1: Coefficient values of the polynomial fit for $M = 1, 3, 10, 19$.

For larger values of $M$, the coefficients become extremely large in absolute value, which is a result of the function matching precisely the datapoints by over-tuning to the noise within the data. Figure 2.2 shows the results obtained by using the regularization term of equation (2.4) with $M = 19$. Of course, if $\lambda = 0$, we just obtain the third panel in figure 2.1.

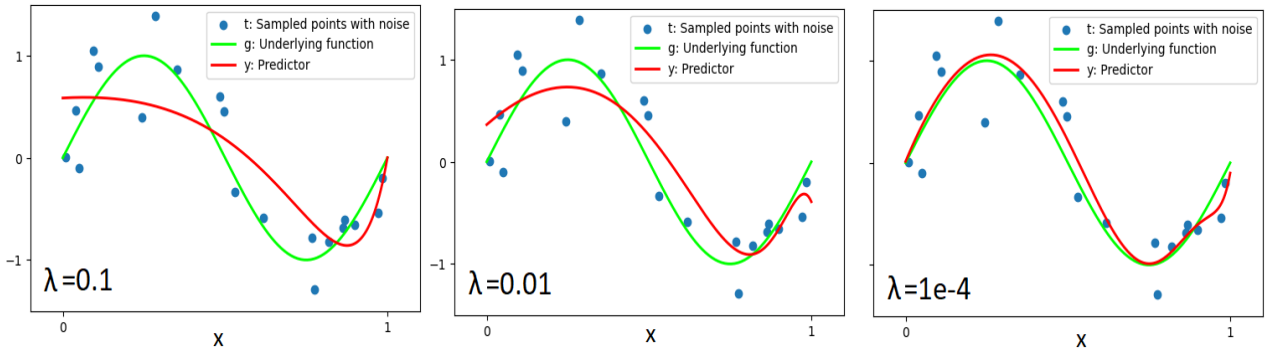


Figure 2.2: Regularized fit with $M = 19$ for $\lambda = 0.1, 0.01, 0.0001$.

Compared to the third panel in figure 2.1 the overfitting problem is evidently reduced. In particular, for $\lambda = 0.0001$, we obtain a really close fit to the underlying labelling function.

The approach used so far returns an optimal value for the parameter $\boldsymbol{w}$, however, with this point estimate of $\boldsymbol{w}$ we have no idea about the uncertainty over the prediction. We consider now a parametrization which allows to estimate this uncertainty.

**Introducing Uncertainty:** In the interest of quantifying the uncertainty over the predicted values we introduce a probabilistic approach. Suppose that, around each target value $t$, we consider a Gaussian distribution with mean $y(x, \boldsymbol{w})$, as given by (2.1), and variance $\beta^{-1}$ (where $\beta$ is know as the precision), i.e.:

$$p(t|x, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(x, \boldsymbol{w}), \beta^{-1}). \tag{2.5}$$

To determine the parameters, $\boldsymbol{w}$ and $\beta$, we maximize the probability of observing the training data as a function of the parameters. This is known as **maximum likelihood**, where the **likelihood**

**function** is given by:

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t|y(x_n, \boldsymbol{w}), \beta^{-1}). \tag{2.6}$$

When taking the product, we assume that the data is independent and identically distributed (i.i.d.). Maximizing the likelihood is equivalent to maximizing the log likelihood, since the logarithm is a strictly increasing function. Thus, applying the logarithm to the expression above we obtain:

$$\begin{aligned} \ln p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) &= \sum_{n=1}^{N} \ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} \exp \left( -\frac{\beta}{2} \big(t_n - y(x_n, \boldsymbol{w})\big)^2 \right) \right) \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln (2\pi) - \frac{\beta}{2} \sum_{n=1}^{N} \big(t_n - y(x_n, \boldsymbol{w})\big)^2. \end{aligned} \tag{2.7}$$

From this expression we see that maximization with respect to $\boldsymbol{w}$ is equivalent to minimizing the following expression:

$$\frac{1}{2} \sum_{n=1}^{N} \big(t_n - y(x_n, \boldsymbol{w})\big)^2. \tag{2.8}$$

This is the same minimization problem as before, we are again minimizing the sum of squares error. So, under the assumption of a Gaussian noise, minimizing the sum of squares error is equivalent to maximizing the likelihood function, with respect to $\boldsymbol{w}$.

Maximizing the likelihood with respect to $\boldsymbol{\beta}$ we obtain:

$$\begin{aligned} \frac{d}{d\beta} \ln p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = 0 \Leftrightarrow \frac{N}{2\beta^*} &= \frac{1}{2} \sum_{n=1}^{N} \big(t_n - y(x_n, \boldsymbol{w})\big)^2 \Leftrightarrow \\ \Leftrightarrow \frac{1}{\beta^*} &= \frac{1}{N} \sum_{n=1}^{N} \big(t_n - y(x_n, \boldsymbol{w})\big)^2. \end{aligned} \tag{2.9}$$

Thus, the variance parameter $\beta$ of the normal distribution becomes simply the sample variance. Our predictor is now a Gaussian distribution over the target value $t$ for each new value of $x$ and is given by:

$$p(t|x, \boldsymbol{w}^*, \beta^*) = \mathcal{N}(t|y(x, \boldsymbol{w}^*), \beta^{*-1}) \tag{2.10}$$

Applying this on our example above using $M = 3$, $\lambda = 0$ (no regularization) and considering the standard deviation, $\sigma = \sqrt{\beta^{*-1}}$, we obtain the following plot, fig. 2.3.

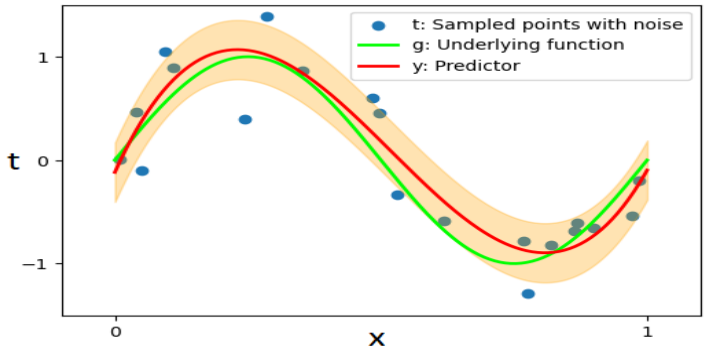


Figure 2.3: Fit with $M = 3$ and $\lambda = 0$, the uncertainty region is over $[y(x, \boldsymbol{w}^*) - \sigma, y(x, \boldsymbol{w}^*) + \sigma]$.

**Source code:** For the source code with the implementation of this example see the file "polynomialcurvefitting.ipynb".

Over the next sections we will look more formally on some of the ideas seen in this example. In particular, we will see next an extension of maximum likelihood and least squares to any input-output dimension and general basis functions. We fill further look briefly at the geometric intuition behind the solution to the least squares problem.

### 2.1.2 Maximum likelihood and Least Squares

The general problem in linear regression is the following: Given a training set of $N$ datapoints $\{\boldsymbol{x}_n\}$ with target values $\{\boldsymbol{t}_n\}$, $n = 1, ..., N$, we want to be able to predict a new value of $t$, for any given $\boldsymbol{x}$, in a way that generalizes as well as possible.

**Remark 2.1.1.** *In general, $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{t}_i \in \mathbb{R}^K$, for $i = 1, ..., N$, with $N, d, K \in \mathbb{N}$. We shall see in a while (in the **Multi-dimensional Outputs** paragraph) that there is no loss in generality by treating solely the case $t_i \in \mathbb{R}$, we shall do so henceforth.*

Consider the parametric model with $M$ parameters:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}), \tag{2.11}$$

where $\boldsymbol{x} = [x_1 \, ... \, x_d]^T$, $\boldsymbol{w} = [w_0 \, ... \, w_{M-1}]^T$, $\boldsymbol{\phi}(\boldsymbol{x}) = [1 \, ... \, \phi_{M-1}(\boldsymbol{x})]^T$ and where the $\phi_{j\geq 1}(\boldsymbol{x})$ are possibly non-linear functions of the input known as **basis functions**. Notice that the model is linear in the parameters $\boldsymbol{w}$ and possibly non-linear in the inputs $\boldsymbol{x}$.

Typical functions used as basis functions are polynomials, as seen in the curve-fitting example, Gaussian basis functions of the form:

$$\phi_j(\boldsymbol{x}) = \exp\left( - \frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|_2^2}{2s^2} \right), \tag{2.12}$$

and sigmoidal functions of the form:

$$\phi_j(\boldsymbol{x}) = \left[ 1 + \exp\left( \frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|_2}{s} \right) \right]^{-1}. \tag{2.13}$$

Similarly to the curve-fitting example we **assume** that the target variable $t$ is given by some underlying deterministic function $g$ with additive Gaussian noise $\epsilon \sim \mathcal{N}(0, \beta^{-1})$, i.e.

$$t(\boldsymbol{x}) = g(\boldsymbol{x}) + \epsilon. \tag{2.14}$$

Considering the parametric model defined above we can write:

$$p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1}), \tag{2.15}$$

which means that the probability of observing a certain value $t$ for input $\boldsymbol{x}$ will be normally distributed with mean given by $y(\boldsymbol{x}, \boldsymbol{w})$ and precision $\beta$. This is exactly the same we did in the curve-fitting example when we introduced uncertainty on the model but now the predictor is defined for a general input dimension and for general basis functions.

Considering now a dataset $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with target values $\boldsymbol{t} = \{t_1, ..., t_N\}$, and under an i.i.d. assumption, we can write the likelihood as:

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}), \tag{2.16}$$

where the mean is replace via (2.11). We want to determine the parameters of our model $(\boldsymbol{w}, \beta)$ which better agree with the observed data, i.e. the parameters which maximize the probability of seeing the data we have access to. To do this, we solve the following optimization problem:

$$\underset{\boldsymbol{w}\in\mathbb{R}^M, \beta\in\mathbb{R}}{\arg\max} \quad p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta). \tag{2.17}$$

This is maximum likelihood which, as seen in the curve fitting example, is equivalent to least squares. We can equivalently solve:

$$\underset{\boldsymbol{w}\in\mathbb{R}^M, \beta\in\mathbb{R}}{\arg\max} \ln(p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta)) = \underset{\boldsymbol{w}\in\mathbb{R}^M, \beta\in\mathbb{R}}{\arg\max} \frac{N}{2}\ln\beta - \frac{\beta}{2}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n))^2. \tag{2.18}$$

Minimizing the above expression with respect to $\boldsymbol{w}$ we obtain:

$$\begin{aligned}
&\nabla_{\boldsymbol{w}} \ln(p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta)) = 0 \\
&\Leftrightarrow \beta \sum_{j=0}^{M-1}\sum_{n=1}^{N} \big(t_n - \boldsymbol{w}^{*T}\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\phi_j(x_n){w_j}^* = 0 \\
&\Leftrightarrow \sum_{n=1}^{N} \big(t_n - \boldsymbol{w}^{*T}\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\boldsymbol{\phi}(\boldsymbol{x}_n)^T = 0 \\
&\Leftrightarrow \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\boldsymbol{x}_n)^T = \boldsymbol{w}^{*T}\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T \\
&\Leftrightarrow \boldsymbol{\Phi}^T\boldsymbol{t} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w}^*.
\end{aligned} \tag{2.19}$$

Where we define the matrix $\boldsymbol{\Phi}$ as:

$$\boldsymbol{\Phi} := \begin{bmatrix} \phi_0(x_1) & ... & \phi_{M-1}(x_1) \\ \vdots & \ddots & \vdots \\ \phi_0(x_N) & ... & \phi_{M-1}(x_N) \end{bmatrix}. \tag{2.20}$$

Considering $N \geq M$ we can solve for a unique $\boldsymbol{w}^*$ using:

$$\boldsymbol{w}^* = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}, \tag{2.21}$$

where $\boldsymbol{\Phi}^\dagger := (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T$ is the pseudo-inverse of $\boldsymbol{\Phi}$. In the overparametrized regime, $(M > N)$, there are infinite solutions. A particular one, which corresponds to the one with minimal Euclidean norm (can be found via Lagrange multipliers, see [Str19]), is given by:

$$\boldsymbol{w}^* = \boldsymbol{t}\boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1}. \tag{2.22}$$

By maximizing with respect to $\beta$ we obtain:

$$\frac{1}{\beta^*} = \frac{1}{N}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^{*T}\boldsymbol{\phi}(\boldsymbol{x}_n))^2, \tag{2.23}$$

which is the sample variance of the target values $t$ compared to the model function $y(\boldsymbol{x}, \boldsymbol{w}^*)$, as we have seen before. Finally, the maximum likelihood solution prediction on an unseen input $\boldsymbol{x}$ is given by:

$$p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}^*), {\beta^*}^{-1}), \tag{2.24}$$

with $\boldsymbol{w}^*$ given by equation (2.21) and $\beta^*$ given by equation (2.23).

**Geometric Interpretation of Least Squares:** Consider the vector $\boldsymbol{t} = (t_1, ..., t_N)^T$ in $\mathbb{R}^N$, where each axis corresponds to the target variable $t_i$, $i = 1, ..., N$. The basis function $\phi_j$, for each $j = 0, ..., M-1$, can be evaluated at the $N$ datapoints $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N)^T$. Define:

$$\varphi_j(\boldsymbol{x}) = \begin{bmatrix} \phi_j(\boldsymbol{x}_1) \\ \vdots \\ \phi_j(\boldsymbol{x}_N) \end{bmatrix} \in \mathbb{R}^N, \tag{2.25}$$

these are the column vectors of $\boldsymbol{\Phi}$. If $M < N$ , i.e. the number of basis functions is less than the number of datapoints, and the $\varphi_j(\boldsymbol{x})$, for each $j = 0, ..., M-1$, are independent, then these span a linear subspace $\mathcal{S}$ of $\mathbb{R}^N$ of dimension $M$. Defining $\boldsymbol{y} = (y(\boldsymbol{x}_1, \boldsymbol{w}), ..., y(\boldsymbol{x}_N, \boldsymbol{w}))^T$, where each $y(\boldsymbol{x}_i, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i)$, then $\boldsymbol{y}$ lies in the subspace $\mathcal{S}$ since we can write it as the linear combination:

$$\boldsymbol{y} = \begin{bmatrix} y(\boldsymbol{x}_1, \boldsymbol{w}) \\ \vdots \\ y(\boldsymbol{x}_N, \boldsymbol{w}) \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}_1) \\ \vdots \\ \sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}_N) \end{bmatrix} = \sum_{j=0}^{M-1} w_j \begin{bmatrix} \phi_j(\boldsymbol{x}_1) \\ \vdots \\ \phi_j(\boldsymbol{x}_N) \end{bmatrix} = \sum_{j=0}^{M-1} w_j \varphi_j(\boldsymbol{x}). \tag{2.26}$$

The sum-of-squares error, without the $\frac{1}{2}$ factor which is there only for convenience when taking derivatives, is just the square of the Euclidean distance between $\boldsymbol{y}$ and $\boldsymbol{t}$, i.e.:

$$\sum_{j=1}^{N} (t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n))^2 = \|\boldsymbol{t} - \boldsymbol{y}\|_2^2. \tag{2.27}$$

So, the solution to least-squares is the vector $\boldsymbol{y}$ which is closest to $\boldsymbol{t}$ and lies within the subspace $\mathcal{S}$. This is given by the projection of $\boldsymbol{t}$ on the subspace $\mathcal{S}$ which is the content of equation (2.21), see figure 2.4 for a picture of the idea. For a more detailed view of this geometric properties see Section II.2 of [Str19].

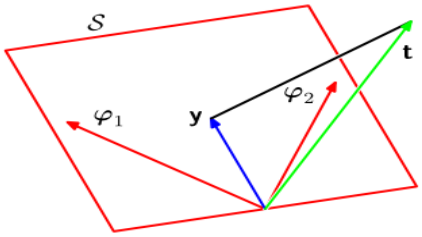


Figure 2.4: Geometry of least-squares. Source: Figure 3.2 of [Bis06].

**Multi-dimensional Outputs:** We return to the initial comment of no loss in generality when treating the outputs in $\mathbb{R}$ as opposed to $\mathbb{R}^K$. Instead of a target variable $t \in \mathbb{R}$ we now have a target variable $\boldsymbol{t} \in \mathbb{R}^K$, $K > 1$, such that, on each dimension, we consider the same $M$ basis functions. Each dimension will have its corresponding weights, i.e. we can write:

$$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x}), \tag{2.28}$$

where $\boldsymbol{W}$ is a $M \times K$ matrix and thus $\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) \in \mathbb{R}^K$. As before we can consider uncertainty over each target value with a multivariate Gaussian distribution. Using a general covariance matrix $\boldsymbol{\Sigma}$ this means:

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{W}, \beta) = \mathcal{N}(\boldsymbol{t}|\boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\Sigma}). \tag{2.29}$$

If we have datapoints $\boldsymbol{t}_1, ..., \boldsymbol{t}_N$, define the matrix $\boldsymbol{T}_{N\times K}$ by:

$$\boldsymbol{T} := \begin{bmatrix} \text{——} & \boldsymbol{t}_1^T & \text{——} \\ \vdots & \vdots & \vdots \\ \text{——} & \boldsymbol{t}_N^T & \text{——} \end{bmatrix}. \tag{2.30}$$

Under an i.i.d. assumption the log likelihood is given by:

$$\begin{aligned} &\ln\big(p(\boldsymbol{T}|\boldsymbol{X}, \boldsymbol{W}, \beta)\big) \\ &= \ln\Big(\prod_{n=1}^{N} \mathcal{N}(\boldsymbol{t}|\boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\Sigma})\Big) \\ &= \ln\Big(\prod_{n=1}^{N} \frac{1}{(2\pi)^{K/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\Big(-\frac{1}{2}\big(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)^T\boldsymbol{\Sigma}^{-1}\big(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\Big)\Big) \\ &= \ln\Big(\prod_{n=1}^{N} \frac{1}{(2\pi)^{K/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\Big) + \ln\Big(\prod_{n=1}^{N} \exp\Big(-\frac{1}{2}\big(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)^T\boldsymbol{\Sigma}^{-1}\big(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\Big)\Big) \\ &= -\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{NK}{2}\ln(2\pi) - \frac{1}{2}\sum_{n=1}^{N}\big(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)^T\boldsymbol{\Sigma}^{-1}\big(\boldsymbol{t}_n - \boldsymbol{W}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big). \end{aligned} \tag{2.31}$$

Taking the derivative with respect to $\boldsymbol{W}$ we obtain:

$$\nabla_{\boldsymbol{W}} \ln\big(p(\boldsymbol{T}|\boldsymbol{X}, \boldsymbol{W}, \beta)\big) = 0 \Leftrightarrow -\sum_{n=1}^{N} \boldsymbol{\Sigma}^{-1}(\boldsymbol{t}_n - \boldsymbol{W}^{T^*}\boldsymbol{\phi}(\boldsymbol{x}_n))\boldsymbol{\phi}(\boldsymbol{x}_n)^T = 0. \tag{2.32}$$

Multiplying by $\boldsymbol{\Sigma}$ and introducing $\boldsymbol{\Phi}$ as before we obtain, in general:

$$\boldsymbol{W}^* = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{T}. \tag{2.33}$$

The vector $\boldsymbol{w}_k$ of coefficients associated to $\boldsymbol{t}_k$ is given by:

$$\boldsymbol{w}_k^* = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}_k, \tag{2.34}$$

where $\boldsymbol{t}_k$ corresponds to column $k$ of $\boldsymbol{T}$. We see then that each output dimension $k = 1, ..., K$ is obtained by just computing the pseudo-inverse $\boldsymbol{\Phi}^\dagger$. Thus, solving the problem with multiple outputs $K$ is equivalent to solving $K$ single output problems and there is no loss in generality by considering the single output case as mentioned at the beginning.

### 2.1.3 Regularized Least Squares

As seen in the curve fitting example, in order to reduce the overfitting problem we can consider the addition of a regularization term to the error function. In general we can write a regularized error $\tilde{E}(\boldsymbol{w})$ as:

$$\tilde{E}(\boldsymbol{w}) = E_D(\boldsymbol{w}) + \lambda E_W(\boldsymbol{w}), \tag{2.35}$$

where the first term, $E_D$, is the data dependent error and the second, $E_W$, is the regularization term with regularization coefficient $\lambda > 0$ (which indirectly depends on the data via how $\boldsymbol{w}$ is obtained).

The regularizer used in the polynomial curve fitting example, $E_W(\boldsymbol{w}) = \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$, is known as **weight decay**, since it encourages the reduction in the square of the norm of the weight vector, $\|\boldsymbol{w}\|_2^2$. If we consider $E_D(\boldsymbol{w})$ to be least squares, the main advantage of weight decay is that the total objective

function remains convex in $\boldsymbol{w}$, so we can easily compute its minimizer via a similar calculation done to obtain (2.21). The solution is given by:

$$\boldsymbol{w}^* = (\lambda \boldsymbol{I} + \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}. \tag{2.36}$$

A more general regularizer can be given by:

$$E_W(\boldsymbol{w}) = \frac{\lambda}{2} \sum_{j=1}^{M} |w_j|^q, \;\; q \in \mathbb{N} \tag{2.37}$$

which for $q = 2$ corresponds to weight decay. For $q = 1$ this known as the **lasso**. This case is particularly interesting since, for larger values of $\lambda$, this leads to sparse solutions. To see this, we first show the equivalence between the following two optimization problems (exercise 3.5 of [Bis06]):

$$\min_{\boldsymbol{w}} \; \frac{1}{2} \sum_{n=1}^{N} \big(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\big)^2 + \frac{\lambda}{2} \sum_{j=1}^{M} |w_j|^q, \tag{2.38}$$

and

$$\begin{aligned} &\min_{\boldsymbol{w}} \; \frac{1}{2} \sum_{n=1}^{N} \big(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\big)^2, \\ &\text{s.t.} \;\; \sum_{j=1}^{M} |w_j|^q \leq \eta, \end{aligned} \tag{2.39}$$

for a certain value of $\eta \in \mathbb{R}$. Let us look at the second problem, by rewriting the constraint:

$$\sum_{j=1}^{M} |w_j|^q \leq \eta \Leftrightarrow \frac{1}{2} \bigg( \sum_{j=1}^{M} |w_j|^q - \eta \bigg) \leq 0, \tag{2.40}$$

the dual problem is given by:

$$\begin{aligned} &\max_{\lambda} \inf_{\boldsymbol{w}} \; \mathcal{L}(\boldsymbol{w}, \lambda) := \frac{1}{2} \sum_{n=1}^{N} \big(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\big)^2 + \frac{\lambda}{2} \bigg( \sum_{j=1}^{M} |w_j|^q - \eta \bigg), \\ &\qquad \text{s.t.} \;\; \lambda \geq 0, \end{aligned} \tag{2.41}$$

where $\mathcal{L}(\boldsymbol{w}, \lambda)$ is the Lagrangian function. Because both the target function (quadratic) and the constraint (a norm) are convex, the primal is a convex problem. Slater's point condition holds trivially, since there are no equality constraints, and we conclude that strong duality holds. In particular, for an optimal $\lambda^*$ and minimizer of the Lagrangian function $\boldsymbol{w}^*(\lambda^*)$, this implies the following KKT (Karush-Kuhn-Tucker) condition known as complementary slackness (see Sections 5.5.2 and 5.5.3 of [BV04]):

$$\frac{\lambda'}{2} \bigg( \sum_{j=1}^{M} |w_j^*|^q - \eta \bigg) = 0 \implies \eta = \sum_{j=1}^{M} |w_j^*|^q. \tag{2.42}$$

Therefore, at the optimal value, the minimizer occurs at the boundary of the constraint region. Considering $E_D(\boldsymbol{w}) := \frac{1}{2} \sum_{n=1}^{N} \big(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\big)^2$ and the region $\sum_{j=1}^{M} |w_j|^q \leq \eta$ we can plot level curves to understand the origin of the sparseness of the minimizer $\boldsymbol{w}^*$. See figure 2.5 for a comparison between $q = 2$ and $q = 1$.

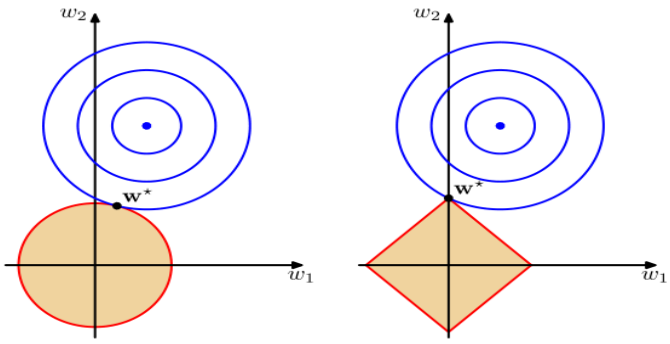


Figure 2.5: Level curves of $E_D(\boldsymbol{w})$ (in blue) together with the constraint region for $q = 2$ (left) and $q = 1$ (right). For $q = 2$ we have $\boldsymbol{w}^* = (0, w_2^*)$. Source: Figure 3.4 of [Bis06].

We see that, for $q = 1$, the boundary of the constraint region leads to sparse solutions for $\boldsymbol{w}$.

Although regularization reduces the overfitting problem in more flexible models it somewhat shifts the problem to the determination of an appropriate $\lambda$. This can be dealt with by using model comparison techniques. A typically used approach is the method of **cross-validation**. This consists of dividing the training data in $S$ subsets, typically of equal size, then training the model with $S - 1$ subsets and evaluating the performance on the held out subset. This is done for all $S$ possible combinations and the average performance is considered.

### 2.1.4 Bayesian Linear Regression

Since many events are not repeatable, such as the event of whether it rains tomorrow for example, it helps to have a way to quantify the belief over events such as this. The Bayesian interpretation views probability as assigning a degree of belief over a certain event which we are then able to update as new evidence arises, via Bayes' Theorem, which supports or not the previously held belief.

In a Bayesian setting one considers a degree of belief on the parameters $\boldsymbol{w}$ that govern our predictor function which is updated by the action of observing the data, $D = \{(x_i, t_i)\}_{i=1}^N$, via Bayes' Theorem:

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)}. \tag{2.43}$$

As an *a priori* belief (called the **prior**), i.e. our "belief" on the distribution of $\boldsymbol{w}$ before seeing the data, we can consider, for example, an isotropic Gaussian with a specified variance $\alpha^{-1}\boldsymbol{I}$, which is given by:

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left(-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right), \tag{2.44}$$

where $\alpha$ is an hyperparameter and $M$ is the dimension of the vector $\boldsymbol{w}$, i.e. the number of basis functions plus a bias term in general. The **posterior**, i.e. the distribution in $\boldsymbol{w}$ after observing data $D$, is proportional to the **likelihood** times the prior:

$$p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) \propto p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha), \tag{2.45}$$

where the missing factor is just a normalization constant. The most probable value of the parameters $\boldsymbol{w}$ given the data is found by **maximizing the posterior (MAP)** which is equivalent to minimizing

the negative log. Considering the likelihood function as given by (2.6) we can write:

$$p(\boldsymbol{w}|\boldsymbol{x},\boldsymbol{t},\alpha,\beta) \propto \underbrace{\prod_{n=1}^{N}\left(\frac{\beta}{2\pi}\right)^{1/2}\exp\Big(-\frac{\beta}{2}\big(t_n-y(x_n,\boldsymbol{w})\big)^2\Big)}_{=p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w},\beta)\text{ (likelihood)}}\underbrace{\left(\frac{\alpha}{2\pi}\right)^{(M+1)/2}\exp\Big(-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\Big)}_{=p(\boldsymbol{w}|\alpha)\text{ (prior)}} \tag{2.46}$$

$$= \gamma\prod_{n=1}^{N}\exp\Big(-\frac{\beta}{2}\big(t_n-y(x_n,\boldsymbol{w})\big)^2-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\Big),$$

where:

$$\gamma := \left(\frac{\beta}{2\pi}\right)^{1/2}\left(\frac{\alpha}{2\pi}\right)^{(M+1)/2}. \tag{2.47}$$

Taking the negative logarithm of this expression we obtain:

$$-\log(\gamma)-\sum_{n=1}^{N}\Big(-\frac{\beta}{2}\big(t_n-y(x_n,\boldsymbol{w})\big)^2-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\Big). \tag{2.48}$$

Minimization of this expression with respect to $\boldsymbol{w}$ is then equivalent to minimizing the following:

$$\frac{1}{2}\sum_{n=1}^{N}\big(t_n-y(x_n,\boldsymbol{w})\big)^2+\frac{\alpha}{2\beta}\boldsymbol{w}^T\boldsymbol{w}. \tag{2.49}$$

Thus, with a Bayesian approach, the MAP paradigm is equivalent to minimizing the regularized sum-of-squares error we saw before in equation (2.4) by considering $\lambda=\frac{\alpha}{\beta}$. We see then that the Bayesian method has an implicit regularization term and will not overfit in general, for an appropriate choice of the hyperparameters.

Since we are actually interested in predicting the target value $t$ for any input $x$ we want to compute the average over the parameter distribution:

$$p(t|x,D)=\int p(t|x,\boldsymbol{w})p(\boldsymbol{w}|D)d\boldsymbol{w}, \tag{2.50}$$

where $p(t|x,\boldsymbol{w})$ is given by equation (2.5) and $p(\boldsymbol{w}|D)$ is the posterior distribution obtained by normalizing equation (2.46).

To outline the above, in a Bayesian setting we assume that our model parameters $\boldsymbol{w}$ follow a certain probability distribution $p(\boldsymbol{w})$, called the prior distribution. The probability of observing data $D=\{(x_1,t_1),...,(x_N,t_N)\}$, for a particular value of $\boldsymbol{w}$, is quantified via the conditional $p(D|\boldsymbol{w})$ which, viewed as a function of $\boldsymbol{w}$, is called the likelihood function. It measures the probability of observing data $D$ under some model which depends on the parameters $\boldsymbol{w}$. By using Bayes' Theorem:

$$p(\boldsymbol{w}|D)=\frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)}, \tag{2.51}$$

we can compute the distribution over $\boldsymbol{w}$ conditioned on having observed data $D$, to which we call the posterior distribution, i.e. $p(\boldsymbol{w}|D)$. Then, via (2.51), we compute the posterior $p(\boldsymbol{w}|D)$ having observed dataset $D$ which allows to determine the predictor given by (2.50).

We will now consider a generalization of this procedure by taking a prior with a general covariance matrix, this will require a few results which can be seen in appendix, Section 6.3.1. Since the likelihood function $p(\boldsymbol{t}|\boldsymbol{w})$ (eq. 2.16) is the exponential of a quadratic term, and assuming the precision $\beta$ is

known, the conjugate prior is also Gaussian (see Section 6.3.8 in appendix). Thus, if we introduce a prior probability over the parameters of the form:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_0, \boldsymbol{S}_0), \tag{2.52}$$

the posterior distribution is Gaussian as well. If we have access to a dataset of $N$ samples, obtained i.i.d., the posterior is given by:

$$p(\boldsymbol{w}|\boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, \boldsymbol{S}_N), \tag{2.53}$$

with:

$$\begin{aligned} \boldsymbol{m}_N &= \boldsymbol{S}_N\big(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\Phi^T\boldsymbol{t}\big), \\ \boldsymbol{S}_N^{-1} &= \boldsymbol{S}_0^{-1} + \beta\Phi^T\Phi. \end{aligned} \tag{2.54}$$

To see this, we just need to complete the square of the product and read the exponential terms as a function of $\boldsymbol{w}$ (exercise 3.7 of [Bis06]). The exponential term of the product $p(\boldsymbol{t}|\boldsymbol{w})p(\boldsymbol{w})$ is given by:

$$\begin{aligned} &\exp\bigg[-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{m}_0)^T\boldsymbol{S}_0^{-1}(\boldsymbol{w}-\boldsymbol{m}_0) - \frac{\beta}{2}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(x_n))^2\bigg] \\ &= \exp\bigg[-\frac{1}{2}\boldsymbol{w}^T\boldsymbol{S}_0^{-1}\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 - \frac{\beta}{2}\boldsymbol{w}^T\big(\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T\big)\boldsymbol{w} + \beta\boldsymbol{w}^T\sum_{n=1}^{N}\boldsymbol{\phi}(x_n) + c\bigg] \\ &= \exp\bigg[-\frac{1}{2}\boldsymbol{w}^T\bigg(\boldsymbol{S}_0^{-1} + \beta\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T\bigg)\boldsymbol{w} + \boldsymbol{w}^T\bigg(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n)\bigg) + c\bigg], \end{aligned} \tag{2.55}$$

where $c \in \mathbb{R}$ is a constant term. By comparing with the expansion of a general multivariate Gaussian exponent we obtain:

$$\boldsymbol{S}_N^{-1} = \boldsymbol{S}_0^{-1} + \beta\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T = \boldsymbol{S}_0^{-1} + \beta\Phi^T\Phi. \tag{2.56}$$

While the mean is given by:

$$\begin{aligned} \boldsymbol{S}_N^{-1}\boldsymbol{m}_N &= \boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n) \\ \Leftrightarrow \boldsymbol{m}_N &= \boldsymbol{S}_N\bigg(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\sum_{n=1}^{N}t_n\boldsymbol{\phi}(x_n)\bigg) = \boldsymbol{S}_N\big(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta\boldsymbol{\Phi}^T\boldsymbol{t}\big), \end{aligned} \tag{2.57}$$

with $\boldsymbol{\Phi}$ as defined in (2.20). Since the distribution is Gaussian, the maximum of the posterior distribution (MAP) is just the mean $\boldsymbol{m}_N$ and, when we have no data i.e. $N = 0$, the posterior reverts to the original prior. We will see now how evaluate the posterior $p(\boldsymbol{w}|\boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_{N+1}, \boldsymbol{S}_{N+1})$ when we receive a new datapoint and consider our prior to be $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, \boldsymbol{S}_N)$, i.e. the posterior after evaluating the original dataset of size $N$.

**Sequential data updates:** Suppose that the data arrives sequentially (exercise 3.8, 3.9 of [Bis06]). After computing the posterior for $N$ observations, when receiving a new data point $(x_{N+1}, t_{N+1})$ the prior is taken to be:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_N, \boldsymbol{S}_N), \tag{2.58}$$

with the likelihood function being given by:

$$p(t_{N+1}|\boldsymbol{w}) = \mathcal{N}(t_{N+1}|\boldsymbol{\phi}(x_{N+1})^T\boldsymbol{w}, \beta^{-1}). \tag{2.59}$$

Using the result from Proposition 6.3.14 we obtain the new posterior:

$$\begin{aligned} p(\boldsymbol{w}|t_{N+1}) &= \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_{N+1}, \boldsymbol{S}_{N+1}) \\ &= \mathcal{N}\Big(\boldsymbol{w}\big|\big(\boldsymbol{S}_{N+1}(\beta t_{N+1}\boldsymbol{\phi}(x_{N+1}) + \boldsymbol{S}_N^{-1}\boldsymbol{m}_N)\big), \big(\boldsymbol{S}_N^{-1} + \beta\boldsymbol{\phi}(x_{N+1})\boldsymbol{\phi}(x_{N+1})^T\big)^{-1}\Big). \end{aligned} \tag{2.60}$$

This result shows how we can update the posterior by using data which arrives sequentially.

**The predictor:** So far, we have a probability distribution over parameters given the data and hyper-parameters, however in general we are actually interested in estimating the probability distribution over the target $t$ for a new value of $x$, i.e. we want the actual predictor. To obtain it we should evaluate:

$$p(t|D, \alpha, \beta) = \int p(t|\boldsymbol{w}, \beta)p(\boldsymbol{w}|D, \alpha, \beta)d\boldsymbol{w}, \tag{2.61}$$

where $D$ is the training data $\{(x_i, t_i)\}_{i=1}^N$. The distribution $p(t|\boldsymbol{w}, \beta)$ follows equation (2.15) while the posterior $p(\boldsymbol{w}|D, \alpha, \beta)$ is given by equation (2.53).

Using again Proposition 6.3.14, where the term:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_N, \boldsymbol{S}_N), \tag{2.62}$$

corresponds to $p(\boldsymbol{x})$, the term:

$$p(t|\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\phi}(x)^T\boldsymbol{w}, \beta^{-1}), \tag{2.63}$$

corresponds to $p(\boldsymbol{y}|\boldsymbol{x})$ and $p(t)$ corresponds to $p(\boldsymbol{y})$, using the expression for $p(\boldsymbol{y})$ given in the proposition we obtain (exercise 3.10 in [Bis06]):

$$\begin{aligned} p(t|D, \alpha, \beta) &= \mathcal{N}\big(t|\boldsymbol{m}_N^T\boldsymbol{\phi}(\boldsymbol{x}), \sigma_N^2(\boldsymbol{x})\big) \\ \sigma_N^2(\boldsymbol{x}) &= \frac{1}{\beta} + \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}). \end{aligned} \tag{2.64}$$

It can be shown that $\sigma_{N+1}^2(\boldsymbol{x}) \leq \sigma_N^2(\boldsymbol{x})$ (exercise 3.11 in [Bis06] - not solved) and that, in the limit as $N \to \infty$, the second term, which represents the uncertainty over the parameters $\boldsymbol{w}$, goes to zero and the variance comes solely from the noise in the data, governed by $\beta$.

Let us consider now a specific example of what we have seen so far.

**Example 2.1.1.** *Suppose we have an underlying function $f(x, \boldsymbol{a}) = a_1x + a_0$ with $a_0 = -0.3$ and $a_1 = 0.5$ and let us consider the hypothesis class of straight lines with two parameters $w_0, w_1$, i.e. the predictor takes the form:*

$$y(x, \boldsymbol{w}) = w_0 + w_1x. \tag{2.65}$$

*We generate data by sampling $x \sim Uni[-1, 1]$ with target values given by $t(x) = f(x, \boldsymbol{a}) + \epsilon$, where $\epsilon$ is a noise term with standard deviation $\sigma = 0.2$ and thus $\epsilon \sim \mathcal{N}\big(0, 0.04\big)$.*

*The goal is to find $a_0$ and $a_1$ based on the training set $D$, under a Bayesian model, and see how the performance depends on the size of the training set, $|D| := N$. We start by defining the likelihood function which takes the following form:*

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}\big(t_n|y(x_n, \boldsymbol{w}), \beta^{-1}\big), \tag{2.66}$$

*under an i.i.d. assumption, and where we take the precision to be a known parameter given by* $\beta = (1/0.2)^2 = 25$*, i.e.* $\beta^{-1}$ *is precisely the variance of* $\epsilon$*. We consider a prior on* $\boldsymbol{w}$ *which is given by:*

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}\big(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}\big), \tag{2.67}$$

*where* $\alpha$ *is also known and takes value* $\alpha = 2$*. We shall now observe the evolution of the posterior as we do sequential draws from the training data:*

- ***Step 0 - No data received:*** *At this point we have not seen any data so the parameters simply come from the initial prior which is given by:*

$$\boldsymbol{w} \sim \mathcal{N}\big(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}\big) = \frac{\sqrt{\alpha}}{2\pi}\exp\bigg(-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\bigg). \tag{2.68}$$

*The following figure (fig. 2.6) shows the prior over* $\boldsymbol{w}$ *on the left panel (with the actual parameters marked with a red dot) and six draws of the predictor according to this prior on the right panel.*

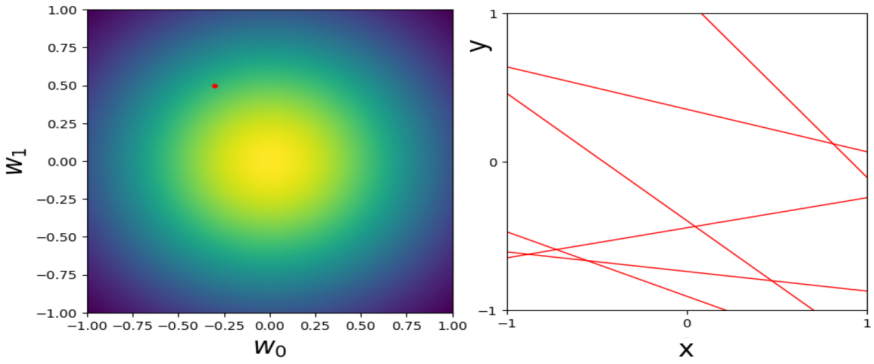


Figure 2.6: i) Prior probability density; ii) Sampled predictors from the prior; (No datapoints available).

- ***Step 1 - One datapoint received:*** *Consider that we now receive a first datapoint* $(x_1, t_1)$*. The likelihood function becomes:*

$$p(t_1|x_1, \boldsymbol{w}, \beta) = \mathcal{N}\big(t_1|\, y(x_1, \boldsymbol{w}), \beta^{-1}\big) = \frac{\sqrt{\beta}}{\sqrt{2\pi}}\exp\bigg(-\frac{\beta}{2}\big(t_1 - y(x_1, \boldsymbol{w})\big)^2\bigg), \tag{2.69}$$

*with normalization constant given by:*

$$\begin{aligned} p(t_1|x_1) &= \int p(t_1|x_1, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha)d\boldsymbol{w} \\ &= \frac{\sqrt{\alpha\beta}}{(2\pi)^{3/2}}\int \exp\bigg(-\frac{\beta}{2}\big(t - y(x_1, \boldsymbol{w})\big)^2 - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\bigg)d\boldsymbol{w}. \end{aligned} \tag{2.70}$$

*The posterior, which is given in general by:*

$$\begin{aligned} p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) &= \frac{p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha)}{p(\boldsymbol{t})}, \\ p(\boldsymbol{t}) &= \int p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha)d\boldsymbol{w}, \end{aligned} \tag{2.71}$$

*takes the form:*

$$p(\boldsymbol{w}|x_1, t_1, \alpha, \beta) = \frac{p(t_1|x_1, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha)}{p(t_1|x_1)} = \frac{\exp\Big(-\frac{\beta}{2}\big(t_1 - y(x_1, \boldsymbol{w})\big)^2 - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\Big)}{\int \exp\Big(-\frac{\beta}{2}\big(t_1 - y(x_1, \boldsymbol{w})\big)^2 - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\Big)d\boldsymbol{w}}. \tag{2.72}$$

*The following figure (fig. 2.7) shows the likelihood function as a function of $\boldsymbol{w}$ on the left panel (i), the posterior over the parameters $\boldsymbol{w}$ on the middle panel (ii) and six draws of the predictor according to this posterior on the right panel (iii), where the blue dot is the datapoint received.*

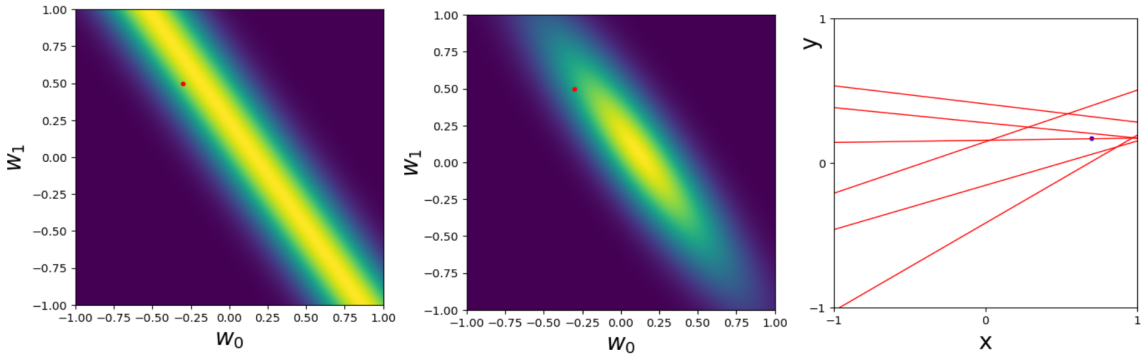


Figure 2.7: i) Likelihood function; ii) Posterior probability density; iii) Sampled predictors; (With one datapoint available).

- ***Step 2 - Two datapoints received:*** *Consider now the second update with datapoint $(x^{(2)}, t^{(2)})$, where we write $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{t} = (t_1, t_2)$. Under and i.i.d. assumption the likelihood function becomes:*

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{n=1}^{2} \mathcal{N}\big(t_n|\, y(x_n, \boldsymbol{w}), \beta^{-1}\big) = \prod_{n=1}^{2} \frac{\sqrt{\beta}}{\sqrt{2\pi}} \exp\left( -\frac{\beta}{2}\big(t_n - y(x_n, \boldsymbol{w})\big)^2 \right). \tag{2.73}$$

*The prior is now the posterior we obtained on the previous step and the normalization constant becomes:*

$$p(\boldsymbol{t}|\boldsymbol{x}) = \int p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) p(\boldsymbol{w}|x_1, t_1, \alpha, \beta) d\boldsymbol{w}. \tag{2.74}$$

*The new posterior is given by:*

$$p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) = \frac{p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) p(\boldsymbol{w}|x_1, t_1, \alpha, \beta)}{p(\boldsymbol{t}|\boldsymbol{x})}. \tag{2.75}$$

*The following figure (fig. 2.8) shows the likelihood function over $\boldsymbol{w}$ on the left panel (i), the posterior over $\boldsymbol{w}$ on the middle panel (ii) and six draws of the predictor according to this posterior on the right panel (iii).*

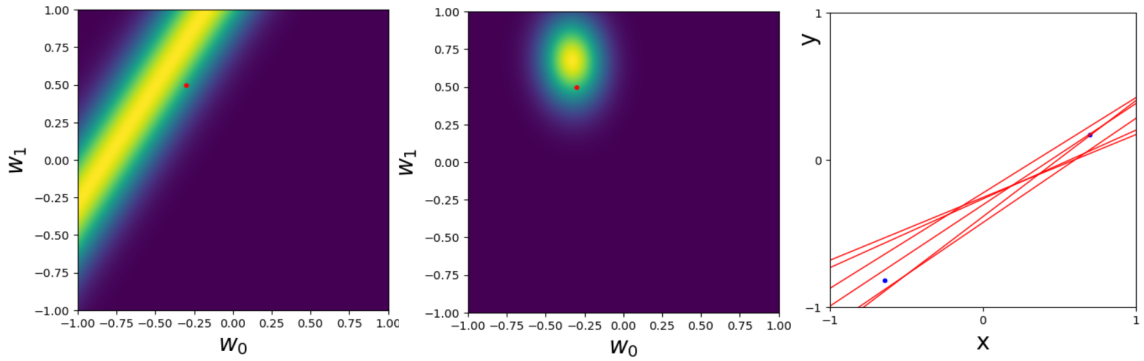


Figure 2.8: i) Likelihood function; ii) Posterior probability density; iii) Sampled predictors; (With two datapoints available).

- ***Entire dataset with 50 datapoints:*** *Suppose we have 50 datapoints $\{(x_1, t_1), ..., (x_{50}, t_{50})\}$, performing the sequential updates as shown in the previous steps we obtain the posterior, and six draws from this posterior, given in the following figure (fig. 2.9).*

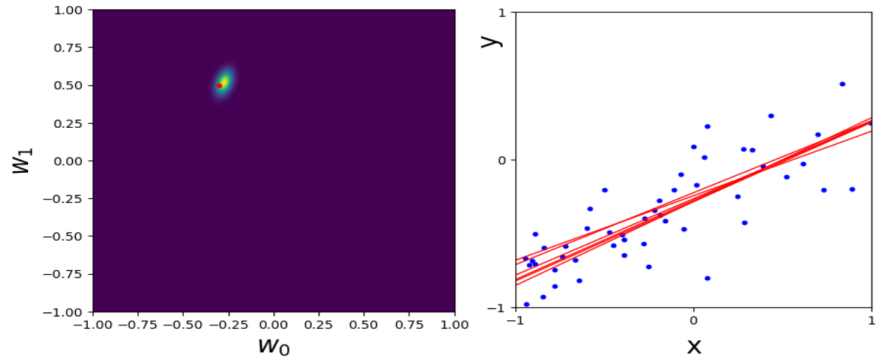


Figure 2.9: i) Likelihood function; ii) Posterior probability density; iii) Sampled predictors; (With fifty datapoints available).

- ***Entire dataset with 2000 datapoints:*** *In the interest of seeing how the posterior behaves when we have a large number of datapoints consider the dataset* $\{(x_1, t_1), ..., (x_{2000}, t_{2000})\}$. *Performing the sequential updates as before we obtain the posterior given in the following figure (fig. 2.10).*

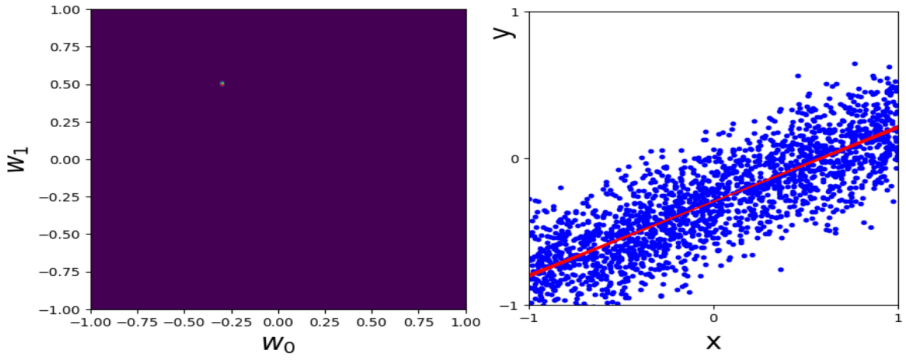


Figure 2.10: i) Likelihood function; ii) Posterior probability density; iii) Sampled predictors; (With two-thousand datapoints available).

*We see that increasing the data narrows the posterior towards the real values of the parameters, i.e. it tends to a Dirac delta centred on the true parameters.*

***Source code:*** *For the source code with this implementation see the file "bayesian-linreg.ipynb".*

In a Bayesian setting, as considered in this section, besides the prior there is an assumption over the distribution of the target variable a priori, equation (2.15). If this choice of model distribution is not representative of the actual underlying distribution our predictor will be misleading. We now look briefly how to select the best model among a finite set.

**Bayesian model comparison:** Suppose we have $L$ models $\mathcal{M}_i$, $i = 1, ..., L$, where each model corresponds to a distribution over the data $D$, for example a normal distribution over the target variable $t$ as we have been considering. We assume that, among our models, one of them is correct (realizability assumption) and we want to find which one. We consider a uniform prior probability over the models $p(\mathcal{M}_i)$, i.e. they are all equally likely. To compute the posterior probability over the models given the observed data $D$ we use Baye's Theorem:

$$p(\mathcal{M}_i|D) \propto p(D|\mathcal{M}_i)p(\mathcal{M}_i). \tag{2.76}$$

The likelihood function $p(D|\mathcal{M}_i)$ expresses the preference of the data given the model and is referred to as **model evidence**. Knowing the posterior we can compute the predictor on a new value $\boldsymbol{x}$ by:

$$p(t|\boldsymbol{x}, D) = \sum_{i=1}^{L} p(t|\boldsymbol{x}, \mathcal{M}_i, D)p(\mathcal{M}_i|D). \tag{2.77}$$

The final predictor is a weighted average of the predictors of each model, $p(t|\boldsymbol{x}, \mathcal{M}_i, D)$, by the posterior probabilities of the models $p(\mathcal{M}_i|D)$. For further insights on model evidence see Section 3.4 of [Bis06].

If each model is governed by a set of parameters $\boldsymbol{w}_i$ then the model evidence is given by:

$$p(D|\mathcal{M}_i) = \int p(D|\boldsymbol{w}_i, \mathcal{M}_i)p(\boldsymbol{w}_i|\mathcal{M}_i)d\boldsymbol{w}_i, \tag{2.78}$$

using a prior distribution on the parameters $p(\boldsymbol{w}_i|\mathcal{M}_i)$.

As discussed in Chapter 1, the bias-complexity tradeoff is a crucial consideration when choosing the hypothesis class. We will see now, in the context of linear regression, how this arises as the bias-variance trade-off.

### 2.1.5 The Bias-Variance Decomposition

When solving a learning problem we want our resulting predictor to generalize well, i.e. it should be accurate on unseen data. As seen before, if the model complexity is too large (relative to dataset size and underlying distribution), we might end up with a predictor which is highly tuned to the noise within the data and ends up with poor generalization performance, despite great performance within the training set (overfitting). However, if we are under very restrictive assumptions, the model might be so simplistic as to not be able to capture the actual trends in the data (underfitting). Let us see, formally, how this concepts arise in the context of linear regression.

Let $(\boldsymbol{x}, t) \sim \mathcal{D}$, with joint probability density $p(\boldsymbol{x}, t)$, where $\boldsymbol{x}$ is the input variable and $t$ the target variable. Consider a random loss function $\ell(t, y(\boldsymbol{x}))$ where $y$ is the predictor. The expected loss of $\ell$ according to distribution $\mathcal{D}$ is given by:

$$\mathbb{E}[\ell] = \int\int \ell(t, y(\boldsymbol{x}))p(\boldsymbol{x}, t)d\boldsymbol{x}dt. \tag{2.79}$$

For the specific case of square loss this becomes:

$$\mathbb{E}[\ell] = \int\int (y(\boldsymbol{x}) - t)^2 p(\boldsymbol{x}, t)d\boldsymbol{x}dt. \tag{2.80}$$

Using calculus of variations (see appendix D of [Bis06]) we can minimize $\mathbb{E}[\ell]$ with respect to $y(\boldsymbol{x})$ to obtain:

$$\frac{\delta\mathbb{E}[\ell]}{\delta y(\boldsymbol{x})} = 2\int (y(\boldsymbol{x}) - t)p(\boldsymbol{x}, t)dt = 0. \tag{2.81}$$

Solving for $y(\boldsymbol{x})$ we obtain:

$$\begin{aligned} y(\boldsymbol{x})\int p(\boldsymbol{x}, t)dt &= \int tp(\boldsymbol{x}, t)dt \\ \Leftrightarrow y(\boldsymbol{x}) = \frac{\int tp(\boldsymbol{x}, t)dt}{p(\boldsymbol{x})} &= \int tp(t|\boldsymbol{x})dt = \mathbb{E}_t[t|\boldsymbol{x}]. \end{aligned} \tag{2.82}$$

Thus, the optimal solution occurs when the predictor function $y(\boldsymbol{x})$ is the expectation of $t$ conditioned on $\boldsymbol{x}$, which makes sense intuitively. Knowing this, we can write:

$$\begin{aligned}(y(\boldsymbol{x})-t)^2 &= (y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}]+\mathbb{E}[t|\boldsymbol{x}]-t)^2 \\ &= (y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])^2+2(y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])(\mathbb{E}[t|\boldsymbol{x}]-t)+(\mathbb{E}[t|\boldsymbol{x}]-t)^2.\end{aligned} \tag{2.83}$$

Replacing in equation (2.80), and keeping in mind that $\mathbb{E}_t[t|\boldsymbol{x}]$ is a function of $\boldsymbol{x}$, we obtain:

$$\begin{aligned}\mathbb{E}[\ell] &= \int\int \Big((y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])^2+2(y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])(\mathbb{E}[t|\boldsymbol{x}]-t)+(\mathbb{E}[t|\boldsymbol{x}]-t)^2\Big)p(\boldsymbol{x},t)d\boldsymbol{x}dt \\ &= \int (y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])^2\int p(\boldsymbol{x},t)dtd\boldsymbol{x}+2\Big(\int y(\boldsymbol{x})\mathbb{E}[t|\boldsymbol{x}]\int p(\boldsymbol{x},t)dtd\boldsymbol{x} \\ &\quad -\int h(\boldsymbol{x})\int tp(\boldsymbol{x},t)dtd\boldsymbol{x}+\int \mathbb{E}[t|\boldsymbol{x}]^2\int p(\boldsymbol{x},t)dtd\boldsymbol{x}-\int \mathbb{E}[t|\boldsymbol{x}]\int tp(\boldsymbol{x},t)dtd\boldsymbol{x}\Big) \\ &\quad +\int\int (\mathbb{E}[t|\boldsymbol{x}]-t)^2p(\boldsymbol{x},t)dtd\boldsymbol{x}.\end{aligned} \tag{2.84}$$

From the following results:

$$\begin{aligned}\int p(\boldsymbol{x},t)dtd\boldsymbol{x} &= p(\boldsymbol{x})d\boldsymbol{x}, \\ \int tp(\boldsymbol{x},t)dtd\boldsymbol{x} &= \int tp(t|\boldsymbol{x})p(\boldsymbol{x})dtd\boldsymbol{x} = \mathbb{E}[t|\boldsymbol{x}]d\boldsymbol{x},\end{aligned} \tag{2.85}$$

the terms in the inner brackets of expression (2.84) cancel and we obtain:

$$\begin{aligned}&\int (y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])^2p(\boldsymbol{x})d\boldsymbol{x}+\int\int (\mathbb{E}[t|\boldsymbol{x}]-t)^2p(t|\boldsymbol{x})p(\boldsymbol{x})dtd\boldsymbol{x} \\ =&\int (y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])^2p(\boldsymbol{x})d\boldsymbol{x}+\int \mathbb{E}\big[(t-\mathbb{E}[t|\boldsymbol{x}])^2\,|\,\boldsymbol{x}\big]p(\boldsymbol{x})d\boldsymbol{x}.\end{aligned} \tag{2.86}$$

Which can finally be written as:

$$\mathbb{E}[\ell] = \int (y(\boldsymbol{x})-\mathbb{E}[t|\boldsymbol{x}])^2p(\boldsymbol{x})d\boldsymbol{x}+\int \mathrm{Var}[t|\boldsymbol{x}]p(\boldsymbol{x})d\boldsymbol{x}. \tag{2.87}$$

From this expression we notice that the predictor appears only on the first term, which is zero if $y(\boldsymbol{x}) = \mathbb{E}_t[t|\boldsymbol{x}]$, as determined before by minimizing with respect to $y(\boldsymbol{x})$. The second term is the expectation over $\boldsymbol{x}$ of the variance $\mathrm{Var}[t|\boldsymbol{x}]$, it is independent of $y(\boldsymbol{x})$ and represents an irreducible minimum value on the expectation of the loss function, it arises as intrinsic noise in the data.

Since we only have access to a finite dataset D we can not determine $y(\boldsymbol{x})$ with perfect accuracy. If we consider a parametric model for $y(\boldsymbol{x})$, of the form $y(\boldsymbol{x},\boldsymbol{w})$, and consider a frequentist approach, where we have only a point estimate for the parameters, in order to obtain an uncertainty over this estimate we can consider the following **ensemble** process:

**Ensemble process:** Suppose we have a large number of datasets $D_I$, for some index set $I$, each of size $N$. Over each dataset D we can run our model to determine a predictor $y_D(\boldsymbol{x},\boldsymbol{w})$, emphasizing that the predictor depends on the particular dataset D. We define the performance of the model by its average performance over the whole ensemble $D_I$. Formalizing this discussion, we insert this data dependence on equation (2.87), where we define $h^*(\boldsymbol{x}) := \mathbb{E}_t[t|\boldsymbol{x}]$ to be the optimal predictor:

$$\mathbb{E}[\ell] = \int \big(y_D(\boldsymbol{x},\boldsymbol{w})-h^*(\boldsymbol{x})\big)^2p(\boldsymbol{x})d\boldsymbol{x}+\int\int \big(t-h^*(\boldsymbol{x})\big)^2p(\boldsymbol{x},t)d\boldsymbol{x}dt. \tag{2.88}$$

We want to compute the expectation over the ensemble, i.e. $\mathbb{E}_D[\mathbb{E}[\ell]]$. We apply the same trick as before and write the first term in parenthesis as:

$$\begin{aligned}\big(y_D(\boldsymbol{x},\boldsymbol{w}) - h^*(\boldsymbol{x})\big)^2 &= \big(y_D(\boldsymbol{x},\boldsymbol{w}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] + \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] - h^*(\boldsymbol{x})\big)^2 \\ &= \big(y_D(\boldsymbol{x},\boldsymbol{w}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2 + \big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] - h^*(\boldsymbol{x})\big)^2 \\ &\quad + 2\bigg(\big(y_D(\boldsymbol{x},\boldsymbol{w}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)\big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] - h^*(\boldsymbol{x})\big)\bigg).\end{aligned} \tag{2.89}$$

By taking the expectation of the expression above with respect to D, for the last term we obtain:

$$\begin{aligned}&\mathbb{E}_D\bigg[\big(y_D(\boldsymbol{x},\boldsymbol{w})\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big) - \big(y_D(\boldsymbol{x},\boldsymbol{w})h^*(\boldsymbol{x})\big) + \big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]h^*(\boldsymbol{x})\big) - \big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2\bigg] \\ &= \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2 - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)h^*(\boldsymbol{x}) + \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)h^*(\boldsymbol{x}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2 \\ &= 0.\end{aligned} \tag{2.90}$$

Thus, the expectation with respect to D of the expression in (2.89) becomes:

$$\begin{aligned}&\mathbb{E}_D\bigg(\big(y_D(\boldsymbol{x},\boldsymbol{w}) - h^*(\boldsymbol{x})\big)^2\bigg) \\ &= \mathbb{E}_D\bigg(\big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] - h^*(\boldsymbol{x})\big)^2\bigg) + \mathbb{E}_D\bigg(\big(y_D(\boldsymbol{x},\boldsymbol{w}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2\bigg) \\ &= \underbrace{\big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] - h^*(\boldsymbol{x})\big)^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_D\bigg(\big(y_D(\boldsymbol{x},\boldsymbol{w}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2\bigg)}_{\text{variance}}.\end{aligned} \tag{2.91}$$

The term denoted $(\text{bias})^2$ reflects the squared difference between the expected predictor over all datasets, $\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]$, and the best possible target function $h^*(\boldsymbol{x})$. The second term, denoted variance since it is in fact a variance, reflects the average of how much each predictor, defined over a certain dataset D, differs from the average predictor over all datasets $\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]$. These calculations, together with equation (2.88), give us:

$$\begin{aligned}\mathbb{E}_D[\mathbb{E}[\ell]] = &\underbrace{\int\bigg(\big(\mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})] - h^*(\boldsymbol{x})\big)^2\bigg)p(\boldsymbol{x})d\boldsymbol{x}}_{(\text{bias})^2} \\ &+ \underbrace{\int\mathbb{E}_D\bigg(\big(y_D(\boldsymbol{x},\boldsymbol{w}) - \mathbb{E}_D[y_D(\boldsymbol{x},\boldsymbol{w})]\big)^2\bigg)p(\boldsymbol{x})d\boldsymbol{x}}_{\text{variance}} + \underbrace{\int\int\big(t - h^*(\boldsymbol{x})\big)^2 p(\boldsymbol{x},t)d\boldsymbol{x}dt}_{\text{noise (irreducible error)}},\end{aligned} \tag{2.92}$$

where these terms refer now to the integrated quantities over $\boldsymbol{x}$.

Comparing to the error splitting seen in the bias-complexity trade-off of Chapter 1, we see that the approximation error corresponds to the bias term, it is the comparison to the best possible predictor. The variance term, like the estimation error, is the sensitivity to the particular dataset being considered. If the model is more complex, i.e. has higher flexibility in function space, it will be more sensitive to the dataset and will thus result in higher variance. To better understand the trade-off aspect of bias and variance let us look at a particular example in a similar setting to the curve fitting scenario we have considered at the start of the chapter.

**Example 2.1.2.** *Suppose we have again a distribution $x \sim Uni[0,1]$, over the input space, a labeling function $g(x) = \sin(2\pi x)$ and a Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.3)$ s.t. the target values are of the form*

*$t(x) = g(x) + \epsilon$. We consider an ensemble $\{D_i\}_{i=1}^{L}$ of $L = 100$ datasets, each of size $|D_i| = 25$. Our hypothesis class is a linear combination of 24 Gaussian basis functions with a bias term of the form:*

$$h(x, \boldsymbol{w}) = w_0 + \sum_{j=1}^{24} w_j \exp\left( - \frac{(x - \mu_j)^2}{2s^2} \right), \tag{2.93}$$

*where the variance is fixed for all 24 basis functions with value $s^2 = 0.01$ and the mean is given by $\mu_j = \frac{(j-1)}{23}$ for $j = 1, ..., 24$.*

*To determine the parameters we use empirical risk minimization via regularized least squares with weight decay, i.e. equation (2.37) with $q = 2$. The solution is given by equation (2.36).*

*The following figure, fig. 2.11, shows the fit over 20 of the 100 datasets for different values of the regularization parameter $\lambda$, which controls the complexity of the functions in the hypothesis class by limiting the norm of $\boldsymbol{w}$ (higher $\lambda$ implies less flexibility, i.e. higher bias, since the norm of $\boldsymbol{w}$ must be kept smaller). It also shows (in blue) the average predictor over the entire ensemble of 100 datasets.*

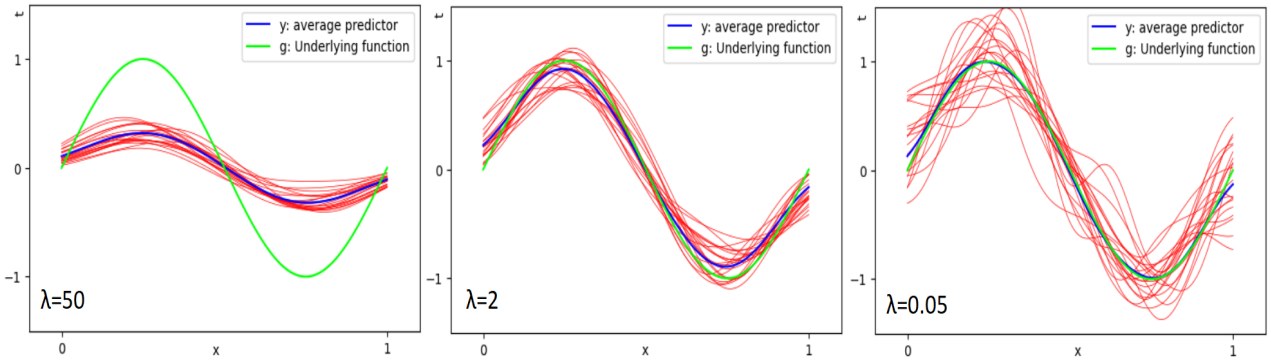


Figure 2.11: Fits for 20 datasets of the ensemble (in red); the average predictor over the entire ensemble (in blue); and the underlying labeling function $g$ (in green), for three different values of $\lambda$.

*Denoting the predictor obtained on dataset $D_i$, via regularized least squares, as $h_i^*$, the average over the ensemble is given by:*

$$\bar{h}^*(x) = \frac{1}{L} \sum_{i=1}^{L} h_i^*(x). \tag{2.94}$$

*To compute the bias$^2$ and the variance terms from equation (2.92) we approximate the integrals via a finite sum by sampling a large number of points $M$ from the distribution, we have:*

$$\begin{aligned} bias^2 &\approx \frac{1}{M} \sum_{n=1}^{M} (\bar{h}(x_n) - g(x_n))^2, \\ variance &\approx \frac{1}{M} \sum_{n=1}^{M} \frac{1}{L} \sum_{l=1}^{L} (h^{(l)}(x_n) - \bar{h}(x_n))^2. \end{aligned} \tag{2.95}$$

*We consider $M = 1000$ and use a test set of the same size to compute the test error of the averaged predictor. The next figure, fig. 2.12 shows a plot of the bias$^2$, the variance, a sum of these two terms, and the test error of the averaged predictor for values of the regularization parameter in the interval $\lambda \in [0.08, 4.5]$.*

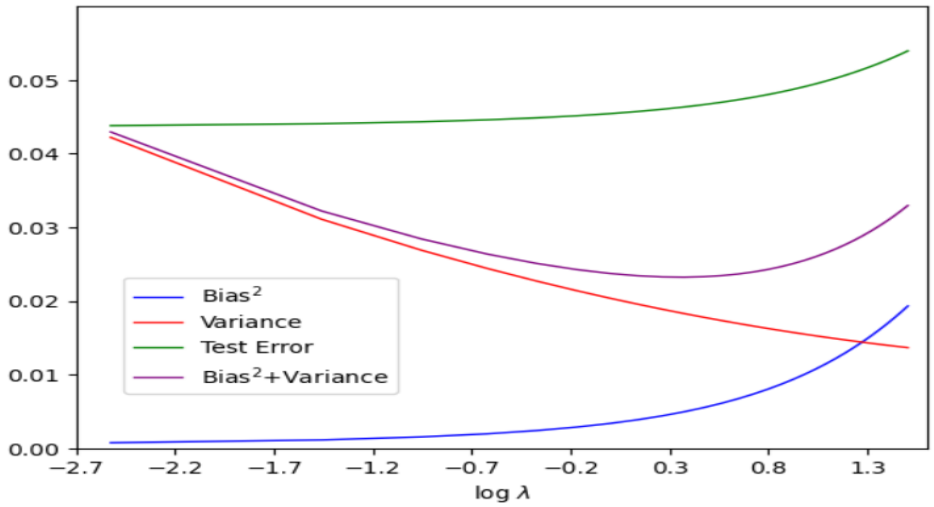


Figure 2.12: Plot of the bias$^2$, variance, the sum of both and the test error as a function of $\log(\lambda)$.

*Theoretically we would expect that, on average, and because the noise term is independent of $\lambda$, the minimum of the test error would be close to the minimum of the term (bias$^2$+variance), which is what is shown in figure 3.6 of [Bis06]. However, performing several runs that is not what we obtain. In figure 3.5 of [Bis06] the last panel is also, apparently, a better fit than the supposed best at approximately $\lambda = exp(-0.31) \sim 0.73$. We notice also that, to obtain a similar looking curve to the first panel of fig. 3.5 of [Bis06], we need to use a much higher $\lambda = 50$ than $\lambda = exp(2.6) \sim 13.46$, which was used for that plot. Because the test error depends on $\lambda$ this might be the source of the discrepancy although we could not find an error in the implementation.*

***Source code:*** *For the source code with this implementation see the file "bias-variance.ipynb".*

We end this section on linear regression with the following observation regarding the limitations imposed by considering fixed basis functions.

**Limitations of fixed basis functions and the manifold hypothesis:** Suppose that, in the example of polynomial curve fitting, we take the predictor $y(\boldsymbol{x}, \boldsymbol{w})$ to be a polynomial of order 3. Over an input $\boldsymbol{x} \in \mathbb{R}^n$ we have the following:

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{i=1}^{n} w_1 x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} w_{ijk} x_i x_j x_k. \tag{2.96}$$

Due to permutation symmetries some of the coefficients are the same but, despite this, the number of them still grows with $n^3$. If we consider more complex models with large $M$, the degree of the polynomial, if $n \gg M$, the growth in parameters follows $n^M$ while in the case $M \gg n$ it follows $M^n$ (exercise 1.16 in [Bis06] - not solved). This quickly becomes intractable from a computational point of view.

Real data however typically follows the **manifold hypothesis**, i.e. the input parameters, suppose $n$ of them, are actually approximately restricted to a submanifold of the higher dimensional space $\mathbb{R}^n$. This could be exploited by using basis functions localized within regions of this submanifold which, with the approaches we have been using so far, requires handcrafted basis functions. Neural networks, which we are going to consider in Chapter 3, have basis functions which themselves depend on adaptable parameters. Thus, during the training process of the neural network, these parameters adapt in such a way that automatically captures these submanifolds, supposing of course that we have enough data and network is specified properly.

### 2.1.6 Summary

In this first section on **linear models** we explored algorithms for **regression tasks** and saw detailed examples of their application. The first example, on **curve fitting**, allowed us to put some context to the discussion of Chapter 1, by clearly seeing the problems of overftting and underfitting and how the flexibility of the hypothesis class relates to these issues in a more concrete setting. Via **least squares**, we were able to find a parametrized predictor by having a point estimate on the parameters which govern the model. To improve on this point estimate, and obtain some measure of uncertainty on our prediction, we followed two distinct paradigms:

- **Maximum Likelihood:** We assume that our predictor is modelled by some probability distribution, whose mean is given by a linearly parametrized family of functions $y(\cdot, \boldsymbol{w})$ and whose variance is also given by some parameter. Assuming our dataset is obtained i.i.d. from the underlying distribution our goal was to maximize the **likelihood function**, i.e. to find which parameters maximize the probability of observing the dataset we have access to. We saw, when considering a Gaussian distribution as our model, the equivalence of maximizing the likelihood function and minimizing least squares, in order to determine $\boldsymbol{w}$. By minimization with respect to the variance parameter as well, we were able to finally define our predictor with associated uncertainty. These results we were able to define in a more general setting, in particular for any input-output dimension and any **basis function**. However, as we saw in the polynomial curve fitting problem, this procedure may overfit. This motivated the introduction of the **regularization** term which constraints the flexibility of the model and reduces the overfitting problem. Regularization however, requires the determination of another hyperparameter, via **cross-validation** for example.

- **Bayesian Regression**: The main idea here is to use the **Bayesian interpretation of probability**, as opposed to a **frequentist** interpretation, i.e. a degree of belief over the occurrence of a certain event. This allows for a sequential update on our belief over the parameters (the distribution over parameters) that govern the model, which we define a priori. The process begins precisely by first defining a parametrized model over the target, i.e. a distribution of the target given the input and then defining a **prior** belief, i.e. a distribution over the parameters, which reflects our belief before having access to any data. Given a datapoint we can compute the probability of seeing this particular datapoint, under the assumed parametrized model, as a function of the parameters, which is the meaning of the **likelihood function**. Then, by Baye's Theorem, we are able to update the belief over parameters by using this observed datapoint which results in what is called the **posterior**. Applying this procedure to our dataset gives us a final distribution over parameters based on the observation of the entire dataset, which we can then use to compute the final predictor for any new input by averaging over the posterior parameter distribution. In this setting we saw that if we consider simply the maximum of the posterior distribution **(MAP)** this gives precisely the same optimization problem as least squares with a weight decay regularizer. Thus, besides estimating all hyperparameters via the training data, the Bayesian approach includes an implicit counter to overfitting.

Finally, we extended the bias-complexity trade-off discussion to the regression setting via the **bias-variance trade-off** calculation.

- **Bias-variance trade-off**: Given a distribution over input-output space we determined the best possible predictor to be given by the expectation of the target variable given the input under this distribution. This allows us to determine the expected loss for a specific predictor, i.e. one which is determined with recourse to a specific dataset. However, because the predictor depends on the dataset, we need to consider the expectation of the loss together with an expectation over the possible datasets. After some calculations we determined that this expectation is divided in

three terms, to which we called bias, variance and noise. The bias term reflects how far we are from the best possible predictor, the variance reflects how sensible each predictor is to the used dataset while the noise reflects an irreducible error since the target function is not deterministic, as associated to it some noise distribution.

We finalized the section with an important remark regarding computational complexity. It is possible to see that the fixed basis function approach leads quickly to computationally intractable problems due to an exponential growth either in the number of basis functions or in the input dimension. This motivates the introduction of other models, in particular neural networks, which have the ability to, under the **manifold hypothesis** assumption and via adaptable parameters, "automatically" determine the submanifold and the possible relevant directions within it.

## 2.2 Linear Models for Classification

In a classification problem we want to predict the labels associated to some input $\boldsymbol{x} \in \mathcal{X}$. This could be a binary classification problem, as we saw frequently in Chapter 1 where we considered $\mathcal{Y} = \{0, 1\}$, or a multiclass classification problem where $\mathcal{Y} = \{1, 2, ..., K\}$ for some $K \in \mathbb{N}$. We start by introducing the perceptron algorithm, an historically relevant part of machine learning.

### 2.2.1 Perceptron

The perceptron, invented in 1943 by McCulloch and Pitts, is a linear discriminant model for binary classification motivated by the behaviour of a biological neuron. Suppose that our target variable is such that $t \in \{-1, +1\}$ and $x \in \mathbb{R}^d$. A simple way to predict the label is to define:

$$\hat{y} = \text{sign}(\boldsymbol{w}^T \boldsymbol{\phi}(x) + b) = \begin{cases} +1, & \text{if } \boldsymbol{w}^T \boldsymbol{\phi}(x) + b \geq 0, \\ -1, & \text{if } \boldsymbol{w}^T \boldsymbol{\phi}(x) + b < 0. \end{cases} \tag{2.97}$$

We think of $\boldsymbol{w}^T \boldsymbol{\phi}(x) + b$ as the score of the positive class, i.e. the class labeled as $\{+1\}$, where $\boldsymbol{\phi}$ is some feature transformation of the input data. The decision boundary corresponds to an hyperplane, in the embedded space via $\boldsymbol{\phi}$, and is given by:

$$\boldsymbol{w}^T \boldsymbol{\phi}(x) + b = 0. \tag{2.98}$$

The following figure, 2.13, shows an example of a dataset with $x \in \mathbb{R}^2$ and binary labels together with a separating hyperplane defining the labeled regions.

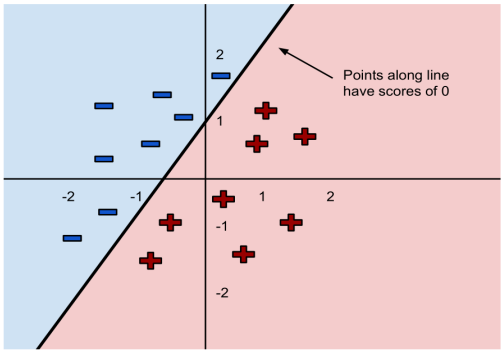


Figure 2.13: Dataset and separating hyperplane. Source: DL-Course (IST) Slides.

We shall consider from now on that the bias is implicit in $\boldsymbol{\phi}$, i.e. $\hat{y} = \text{sign}(\boldsymbol{w}^T \boldsymbol{\phi}(x))$ where $\boldsymbol{\phi}_0(x) = 1$ and $\boldsymbol{w}_0$ corresponds to the bias term. The figure above shows a particular property of the data, it can be perfectly separated with an hyperplane. Rigorously this leads to the following definition.

**Definition 2.2.1.** ***(Linearly separable data)***
*A dataset* $D = \{(x_1, y_1), ..., (x_N, y_N)\}_{n=1}^N$ *is* ***linearly separable*** *with margin* $\gamma > 0$ *if and only if there exists a weight vector* $\boldsymbol{u}$*, with* $\|\boldsymbol{u}\|_2 = 1$*, such that:*

$$y_n \boldsymbol{u}^T \boldsymbol{\phi}(x_n) \geq \gamma, \ \ \forall\, n = 1, ..., N. \tag{2.99}$$

The motivation for this definition comes from the following consideration. The shortest distance from a point $\boldsymbol{\phi}(x)$ to the decision boundary (the hyperplane) is given by a vector perpendicular to the plane, and thus parallel to $\boldsymbol{w}$, since $\boldsymbol{w}$ is orthogonal to the hyperplane. Then, a vector between them is of the form $r\boldsymbol{w}/\|\boldsymbol{w}\|_2$, where $r$ is the Euclidean distance. If we label the point lying on the hyperplane closest to $\boldsymbol{\phi}(x)$ as $\boldsymbol{\phi}^*(x)$ we have then:

$$\boldsymbol{\phi}^*(x) = \boldsymbol{\phi}(x) - yr\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}, \tag{2.100}$$

where $y$ gives the correct sign depending on whether the point is above or below the hyperplane. Since $\boldsymbol{\phi}^*(x)$ lies on the decision boundary it satisfies $\boldsymbol{w}^T\boldsymbol{\phi}^*(x) = 0$ and we can write:

$$\begin{aligned} &\boldsymbol{w}^T\boldsymbol{\phi}^*(x) = \boldsymbol{w}^T\left(\boldsymbol{\phi}(x) - yr\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) = 0 \\ \Leftrightarrow &\frac{1}{y}\boldsymbol{w}^T\boldsymbol{\phi}(x) = r\frac{\boldsymbol{w}^T\boldsymbol{w}}{\sqrt{\boldsymbol{w}^T\boldsymbol{w}}} \\ \Leftrightarrow &r = y\frac{\boldsymbol{w}^T\boldsymbol{\phi}(x)}{\|\boldsymbol{w}\|_2}, \quad \text{since } y^{-1} = y. \end{aligned} \tag{2.101}$$

Therefore we can write $r = y\boldsymbol{u}^T\boldsymbol{\phi}(x)$ with $\|\boldsymbol{u}\|_2 = 1$. The perceptron algorithm is the following:

**Algorithm 1** Perceptron algorithm

**inputs:**
  $D = \{(x_n, y_n)\}_{n=1}^N$
**initialize:**
  $\boldsymbol{w}^{(0)} = 0$ (model weights)
  $k = 0$ (number of mistakes)
**repeat**
  get training example $(x_n, y_n)$
  compute: $\hat{y}_n = \text{sign}(\boldsymbol{w}^T\boldsymbol{\phi}(x) + b)$
  **if** $y_n \neq \hat{y}_n$ **then**
    $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_n\boldsymbol{\phi}(x_n)$
    increment $k$
  **end if**
**until** maximum number of epochs
**outputs:**
  $\boldsymbol{w}^{(k)}$

The following theorem, due to Novikoff (1962), shows that the perceptron algorithm is guaranteed to converge in a finite number of steps provided that the data is linearly separable.

**Theorem 2.2.1.** ***(Perceptron's convergence)***
*Let* $R = \max_n \|\boldsymbol{\phi}(x_n)\|_2$ *and suppose we have linearly separable data* $D$*. The perceptron algorithm is guaranteed to converge to a separating hyperplane after, at most,* $\frac{R^2}{\gamma^2}$ *updates.*

*Proof.* The update in the algorithm above is given by:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_n\boldsymbol{\phi}(x_n). \tag{2.102}$$

We can compute a lower bound on $\|\boldsymbol{w}^{(k)}\|_2$ as follows, consider a weight vector $\boldsymbol{u}$, with $\|\boldsymbol{u}\|_2 = 1$ then we have:

$$\begin{aligned}
\boldsymbol{u}^T\boldsymbol{w}^{(k)} &= \boldsymbol{u}^T(\boldsymbol{w}^{(k-1)} + y_n\boldsymbol{u}^T\boldsymbol{\phi}(x_n)) \\
&\geq \boldsymbol{u}^T\boldsymbol{w}^{(k-1)} + \gamma \qquad \text{(by definition of linearly separable data)} \\
&\geq \boldsymbol{u}^T\boldsymbol{w}^{(k-2)} + 2\gamma \\
&\geq k\gamma \qquad \text{(applying the previous step } k \text{ times and since } \boldsymbol{w}^{(0)} = 0).
\end{aligned}$$

Then, by Cauchy-Schwarz inequality, we get:

$$\|\boldsymbol{w}^{(k)}\|_2 = \|u\|_2\|\boldsymbol{w}^{(k)}\|_2 \geq \langle \boldsymbol{u}, \boldsymbol{w}\rangle = \boldsymbol{u}^T\boldsymbol{w}^{(k)} \geq k\gamma. \tag{2.103}$$

We can compute an upper bound on $\|\boldsymbol{w}^{(k)}\|_2$ by:

$$\begin{aligned}
\|\boldsymbol{w}^{(k)}\|_2^2 &= \|\boldsymbol{w}^{(k-1)}\|_2^2 + \|\boldsymbol{\phi}(x_n)\|_2^2 + 2y_n{\boldsymbol{w}^{(k)}}^T\boldsymbol{\phi}(x_n) \\
&\leq \|\boldsymbol{w}^{(k-1)}\|_2^2 + R^2 \\
&\leq kR^2.
\end{aligned}$$

This holds because the weights are updated whenever $\hat{y}_n \neq y_n$, which implies that $2y_n{\boldsymbol{w}^{(k)}}^T\boldsymbol{\phi}(x_n) < 0$. Together we have:

$$k^2\gamma^2 \leq kR^2 \implies k \leq \frac{R^2}{\gamma^2}. \tag{2.104}$$

■

One critique of the perceptron algorithm, which is often considered to have some responsibility in the "AI winter" of the 1970s, is the following problem known as the "XOR problem".

**XOR problem:** The perceptron algorithm can solve linearly separable problems such as those defined by the logical functions **AND, OR**, see figure 2.14 (for a computational implementation of this).

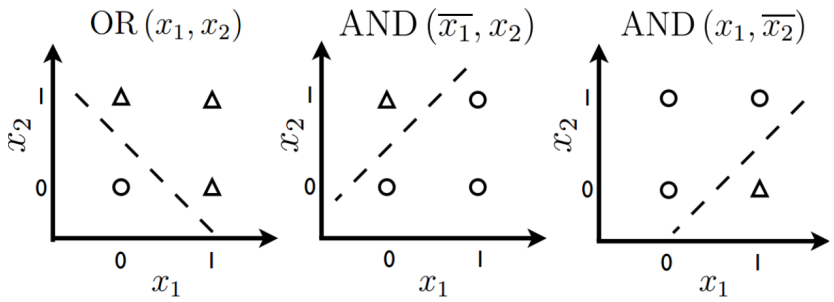


Figure 2.14: AND/OR Functions. Source: DL-Course (IST) Slides.

The following figure, fig. 2.15 shows an implementation of the perceptron algorithm on the logical functions **NOT, AND** and **OR** (see the file "perceptron.ipynb") and in particular shows the obtained separating hyperplanes.

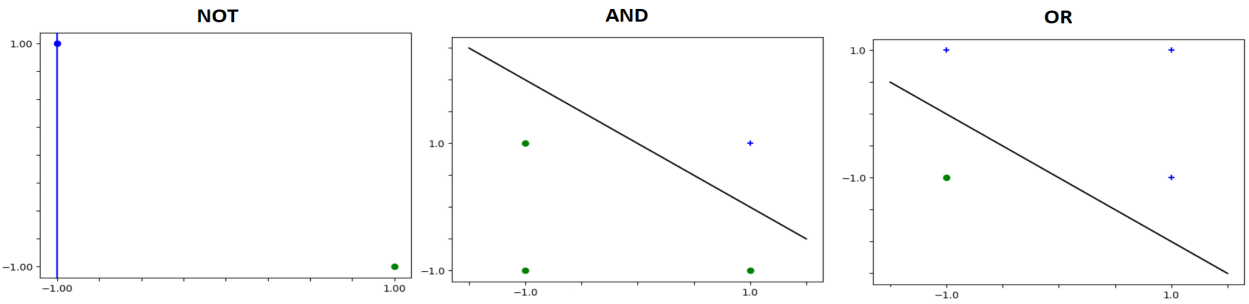


Figure 2.15: Learning NOT/AND/OR via the perceptron algorithm.

However, without a feature transformation, it can not learn the **XOR** logical function, see figure 2.16 on the left. By the transformation of the features shown on the right however the XOR problem is solvable.

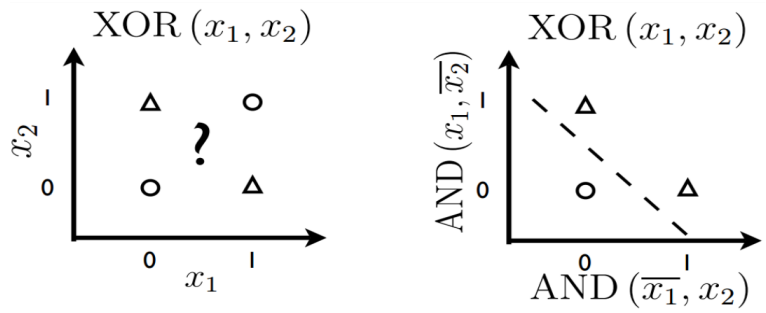


Figure 2.16: XOR Problem. Source: DL-Course (IST) Slides.

Note also that the perceptron algorithm does not find the best margin for the separating hyperplane, which can be achieved via support vector machines and will be covered later in Chapter 4.

The idea behind the perceptron can be extended to a multiclass classification setting by constructing a vector of weights associated to each label and defining the boundaries as the half-spaces resultant from the intersection of the different hyperplanes.

**Multiclass Classification:** Suppose we have $|\mathcal{C}|$ classes with $x \in \mathbb{R}^D$ and $\boldsymbol{\phi}(x) \in \mathbb{R}^{D+1}$, we define the predictor as:

$$\hat{y}(x) = \underset{y \in \mathcal{C}}{\arg\max}\ \boldsymbol{w}_y^T \boldsymbol{\phi}(x) = \underset{y \in \mathcal{C}}{\arg\max}\ \boldsymbol{W}^T \boldsymbol{\phi}(x), \tag{2.105}$$

where:

$$\boldsymbol{W}_{|\mathcal{C}| \times (D+1)} := \begin{bmatrix} \text{——} & w_1^T & \text{——} \\ \vdots & \ddots & \vdots \\ \text{——} & w_{|\mathcal{C}|}^T & \text{——} \end{bmatrix}. \tag{2.106}$$

This will divide the space in regions which are delimited by hyperplanes. When $\mathcal{C} = \{-1, +1\}$, this reduces to the binary classification we had above:

$$\hat{y} = \underset{y \in \{-1,+1\}}{\arg\max}\ \boldsymbol{w}_y^T \boldsymbol{\phi}(x) = \begin{cases} +1, & \text{if } {\boldsymbol{w}_{+1}}^T \boldsymbol{\phi}(x) > {\boldsymbol{w}_{-1}}^T \boldsymbol{\phi}(x) \\ -1, & \text{otherwise} \end{cases} = \text{sign}\big(({\boldsymbol{w}_{+1}}^T - {\boldsymbol{w}_{-1}}^T)\boldsymbol{\phi}(x)\big), \tag{2.107}$$

where we can define $\boldsymbol{w} := {\boldsymbol{w}_{+1}}^T - {\boldsymbol{w}_{-1}}^T$, to recover the original version with a single vector of parameters.

Similarly to the binary case, if the data is linearly separable, the multiclass perceptron algorithm we present next is guaranteed to find the hyperplanes which separate the data.

**Algorithm 2** Multiclass perceptron algorithm

**inputs:**
$D = \{(x_n, y_n)\}_{n=1}^{N}$
**initialize:**
$\boldsymbol{W}^{(0)} = 0$ (model weights)
$k = 0$ (number of mistakes)
**repeat**
get training example $(x_n, y_n)$
compute: $\hat{y}_n = \arg\max_{y \in \mathcal{C}} \boldsymbol{w}_y^{(k)^T} \boldsymbol{\phi}(x_n)$
**if** $y_n \neq \hat{y}_n$ **then**
$\boldsymbol{w}_{y_n}^{(k+1)} = \boldsymbol{w}_{y_n}^{(k)} + \boldsymbol{\phi}(x_n)$ (increase weights of correct class)
$\boldsymbol{w}_{\hat{y}_n}^{(k+1)} = \boldsymbol{w}_{\hat{y}_n}^{(k)} - \boldsymbol{\phi}(x_n)$ (decrease weights of incorrect class)
increment $k$
**end if**
**until** maximum number of epochs
**outputs:**
$\boldsymbol{W}^{(k)}$

Let us now look at two specific implementations of these methods, a binary classification setting for data in $\mathbb{R}^2$ and the other case for multiclass classification with 10 labels and data in $\mathbb{R}^{64}$, corresponding to images of handwritten digits.

**Example 2.2.1.** ***(Binary classification with input space*** $\mathbb{R}^2$***)***
*Suppose we have input* $\boldsymbol{x} \in \mathbb{R}^2$ *with labels* $y \in \{-1, 1\}$ *and we have access to the dataset* $D = \{((-1, 0), -1), ((0, 0.25), 1), ((1, 1), 1), ((1, -1), -1)\}$. *We take* $\phi$ *to be the identity and initialize the parameters as zero. The following figure, fig. 2.17, shows the update of the perceptron algorithm for each epoch until convergence. The blue points marked with a "+" sign are positive labels while the green dots correspond to the negative labels.*

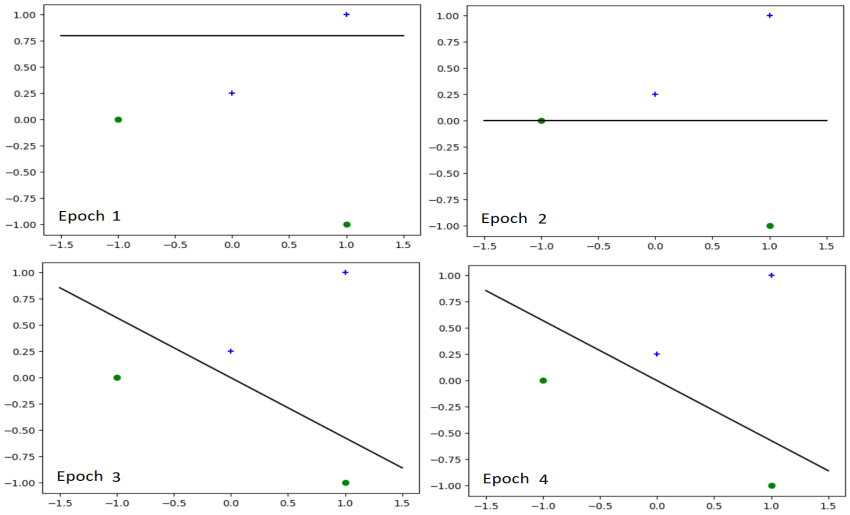


Figure 2.17: Hyperplane update of the perceptron algorithm over each epoch.

**Example 2.2.2.** ***(Multiclass classification with input space*** $\mathbb{R}^{64}$***)***

*In this example we consider our data to be the test set of the UCI handwritten digits dataset*[1] *which contains* $N = 1797$ *datapoints. Each datapoint consists of a handwritten digit image which corresponds to an* $8 \times 8$ *matrix of integer values in the range* $0, ..., 16$*. The input data lives in heavily discretized version* $\mathbb{R}^{64}$ *due to only 17 possible values on each pixel. We separate our dataset in a training set with* $80\%$ *of the data and a test set with the remaining* $20\%$*. Running an implementation of the multiclass perceptron algorithm shown above we obtain the following figure (fig. 2.18) which shows the train and test accuracies as a function of the epoch.*

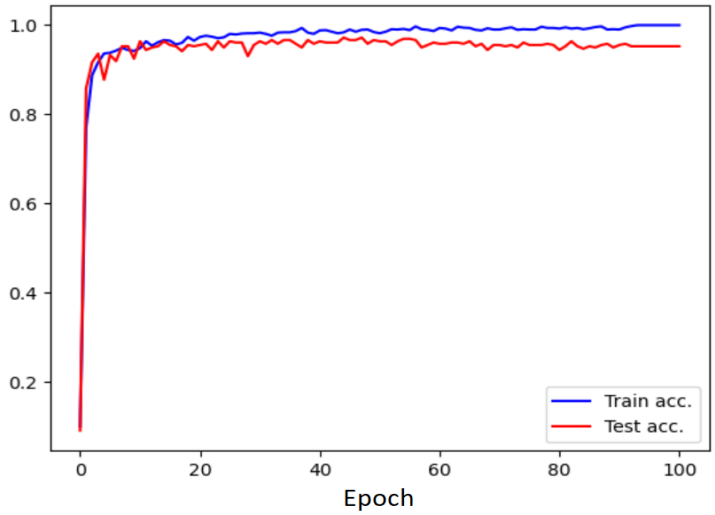


Figure 2.18: Accuracies of the multiclass perceptron algorithm on the UCI handwritten digits dataset.

*Over a short number of epochs the method already gives quite an accurate prediction on the test set with a final value of* $95\%$ *accuracy at the end of the 100 epochs. Over the training set the network has an accuracy of* $100\%$ *which shows that the data is linearly separable.*

**Source code:** For the source code with the implementation of these example see the file "perceptron.ipynb".

### 2.2.2 Probabilistic Generative Models

The perceptron method is a linear non-probabilistic discriminant method, to each possible input it assigns a definite label, as opposed to a probability distribution, and does not provide any estimate for the probability of sampling a particular point over the input space. We now introduce a generative approach to classification, where we consider distributions which model both the output and the inputs. This allows for example, after determining the joint distribution, to generate synthetic data and thus the generative name.

Suppose we are in the case of binary classification with two classes $\mathcal{C}_1$ and $\mathcal{C}_2$. Given a new input $\boldsymbol{x}$ we can write:

$$p(\mathcal{C}_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \frac{1}{1 + \exp(-a)} := \sigma(a), \tag{2.108}$$

where:

$$a = \log \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)}. \tag{2.109}$$

[1] `https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`

The function $\sigma : \mathbb{R} \to (0,1)$ is known as the **logistic sigmoid**. If we have $|\mathcal{C}| > 2$ classes we can similarly write:

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_{j=1}^{|\mathcal{C}|} p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_j)} = \frac{\exp(a_k)}{\sum_{j=1}^{|\mathcal{C}|}\exp(a_j)}, \tag{2.110}$$

where:

$$a_k = \log(p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)). \tag{2.111}$$

This is a multiclass generalization of the sigmoid function known as **softmax**. Suppose we model $p(\boldsymbol{x}|\mathcal{C}_k) \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, i.e. the input over each class is normally distributed, centered around $\boldsymbol{\mu}_k$, and they all share the covariance matrix $\boldsymbol{\Sigma}$.

For $|\mathcal{C}| = 2$, we can compute:

$$\begin{aligned} a &= \log\left(\frac{\exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)\big)p(\mathcal{C}_1)}{\exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)\big)p(\mathcal{C}_2)}\right) \\ &= -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2) + \log\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \\ &= \frac{1}{2}\boldsymbol{x}^T\big(\boldsymbol{\Sigma}^{-1}-\boldsymbol{\Sigma}^{-1}\big)\boldsymbol{x} + (\boldsymbol{\mu}_1^T-\boldsymbol{\mu}_2^T)\boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \left(-\frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \log\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}\right) \\ &=: \boldsymbol{w}^T\boldsymbol{x} + w_0, \end{aligned} \tag{2.112}$$

where:

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \tag{2.113}$$

$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \log\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}. \tag{2.114}$$

We can see, by (2.112), that due to the assumption of equal covariance $a$ has a linear dependence on $\boldsymbol{x}$. Thus, the decision boundary, i.e. when:

$$p(\mathcal{C}_1|\boldsymbol{x}) = p(\mathcal{C}_2|\boldsymbol{x}) = 1 - p(\mathcal{C}_1|\boldsymbol{x}) \implies p(\mathcal{C}_1|\boldsymbol{x}) = \frac{1}{2}, \tag{2.115}$$

is a linear function of $\boldsymbol{x}$ and thus a separating hyperplane. For the general case $|\mathcal{C}| = K$ one can compute:

$$a_k(\boldsymbol{x}) = \boldsymbol{w}_k^T\boldsymbol{x} + w_{k0}, \tag{2.116}$$

where:

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k, \tag{2.117}$$

$$w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log p(\mathcal{C}_k). \tag{2.118}$$

Where again the decision boundaries are linear functions of $\boldsymbol{x}$. Note that here we could be considering a general feature transformation via $\boldsymbol{phi}$ but it makes calculations too dense in notation. If one considers different covariance matrices $\boldsymbol{\Sigma}_k$ the quadratic terms in (2.112) no longer cancel, this leads to quadratic instead of linear decision boundaries.

**Determining parameters:** Still in the binary classification setting suppose we have dataset $D$, where we take $t_n = 1$ if it belongs to class $\mathcal{C}_1$ and $t_n = 0$ if it belongs to class $\mathcal{C}_2$. We can determine the parameters of the parametric model $p(\boldsymbol{x}|\mathcal{C}_k)$ and the priors $p(\mathcal{C}_k)$ using maximum likelihood.

Let $\alpha \in [0,1]$ and consider the prior probabilities over the classes $p(\mathcal{C}_1) = \alpha$ and $p(\mathcal{C}_2) = 1 - \alpha$. Since we have:

$$p(\boldsymbol{x}_n, \mathcal{C}_1) = p(\boldsymbol{x}_n|\mathcal{C}_1)p(\mathcal{C}_1) = \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\alpha, \tag{2.119}$$

$$p(\boldsymbol{x}_n, \mathcal{C}_2) = p(\boldsymbol{x}_n|\mathcal{C}_2)p(\mathcal{C}_2) = \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})(1 - \alpha). \tag{2.120}$$

The likelihood function is given by:

$$p(\boldsymbol{t}, \boldsymbol{X}|\alpha, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \Big(\alpha\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\Big)^{t_n} \Big((1-\alpha)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\Big)^{1-t_n}, \tag{2.121}$$

where $\boldsymbol{t}$ and $\boldsymbol{X}$ are the vectors containing all targets and inputs respectively. Considering the log likelihood we have:

$$\log p(\boldsymbol{t}, \boldsymbol{X}|\alpha, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} t_n \log\Big(\alpha\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\Big) + (1-t_n)t_n \log\Big((1-\alpha)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\Big). \tag{2.122}$$

- **Maximizing with respect to $\alpha$:** This is equivalent to maximizing:

$$\sum_{n=1}^{N} t_n \log(\alpha) + (1 - t_n)\log(1 - \alpha). \tag{2.123}$$

  Computing the derivative with respect to $\alpha$ and setting the expression equal to zero we obtain:

$$\frac{1}{\alpha}\sum_{n=1}^{N} t_n - \frac{1}{1-\alpha}\sum_{n=1}^{N}(1 - t_n) = 0 \tag{2.124}$$

$$\Leftrightarrow \frac{1}{\alpha}N_1 = \frac{1}{1-\alpha}N_2 \tag{2.125}$$

$$\Leftrightarrow \alpha = \frac{N_1}{N_1 + N_2}, \tag{2.126}$$

  where $N_1$ is the number of datapoints labeled as $\mathcal{C}_1$ and $N_2$ the number of datapoints labeled as $\mathcal{C}_2$. The prior probability is then just the ratio of datapoints in each class, which makes intuitive sense.

- **Maximizing with respect to $\boldsymbol{\mu}_k$:** For $\boldsymbol{\mu}_1$ this is equivalent to maximizing:

$$\sum_{n=1}^{N} t_n\Big(-\frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_1)\Big). \tag{2.127}$$

  Again, taking the derivative and equating to zero one obtains:

$$\frac{d}{d\boldsymbol{\mu}_1}\Big(-\frac{1}{2}(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1)\sum_{n=1}^{N} t_n + \sum_{n=1}^{N} t_n\boldsymbol{x}_n^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1\Big) = 0 \tag{2.128}$$

$$\Leftrightarrow -\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 N_1 + \boldsymbol{\Sigma}^{-1}\Big(\sum_{n=1}^{N} t_n\boldsymbol{x}_n\Big) = 0 \tag{2.129}$$

$$\Leftrightarrow \boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n\boldsymbol{x}_n. \tag{2.130}$$

Which is just the mean of all $N_1$ points labeled in $\mathcal{C}_1$. Similarly we have:

$$\boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)\boldsymbol{x}_n. \tag{2.131}$$

These correspond to the averages over input space of points with the corresponding label, again quite an expected result.

- **Maximizing with respect to $\boldsymbol{\Sigma}$:** The terms depending on $\boldsymbol{\Sigma}$ from equation (2.122) are given by:

$$\begin{aligned}&\sum_{n=1}^{N} t_n \log\left(\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\right) - \frac{1}{2}\sum_{n=1}^{N} t_n\Big((\boldsymbol{x}_n-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_1)\Big)\\ &+\sum_{n=1}^{N}(1-t_n)\log\left(\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\right) - \frac{1}{2}\sum_{n=1}^{N}(1-t_n)\Big((\boldsymbol{x}_n-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_2)\Big).\end{aligned} \tag{2.132}$$

This can be written as $-\frac{N}{2}\log(|\boldsymbol{\Sigma}|) - \frac{N}{2}\mathrm{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{S})$ where:

$$\begin{aligned}\boldsymbol{S} &= \frac{N_1}{N}\boldsymbol{S}_1 + \frac{N_2}{N}\boldsymbol{S}_2,\\ \boldsymbol{S}_1 &= \frac{1}{N_1}\sum_{n\in\mathcal{C}_1}(\boldsymbol{x}_n-\boldsymbol{\mu_1}))(\boldsymbol{x}_n-\boldsymbol{\mu_1}))^T,\\ \boldsymbol{S}_2 &= \frac{1}{N_2}\sum_{n\in\mathcal{C}_2}(\boldsymbol{x}_n-\boldsymbol{\mu_2}))(\boldsymbol{x}_n-\boldsymbol{\mu_2}))^T.\end{aligned} \tag{2.133}$$

And, according to [Bis06], Section 4.2.2, the maximum occurs precisely when $\boldsymbol{\Sigma} = \boldsymbol{S}$.

These results can naturally be generalized to $K$ classes, (see exercise 4.10 of [Bis06]). To summarize, by modeling the input distribution, conditioned on the labels, via a parameterized Gaussian, i.e. $p(\boldsymbol{x}|\mathcal{C}_k) \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, and with some constant priors on the labels, the regions of classification will be separated by hyperplanes. We can estimate all the required parameters $(\alpha, \boldsymbol{\mu_k}, \boldsymbol{\Sigma})$ via maximum likelihood which allows for the construction of a joint distribution over input space and the labels space. This joint then permits the generation of synthetic data, i.e. new input/label pairs which are not part of the dataset.

**Probabilistic Discriminative Models:** In a discriminative model we directly maximize the likelihood via the conditional $p(\mathcal{C}_k|\boldsymbol{x})$, in a probabilistic approach, which has the advantage of requiring less parameters to learn and may lead to a better performance than the generative models, when the assumed behaviour of the conditional $p(\boldsymbol{x}|\mathcal{C}_k)$ is a bad approximation to the true distribution. We have already seen the perceptron as an example of a non-probabilistic linear discriminant model. We now introduce another famous discriminative, model known as logistic regression, a probabilistic discriminant model.

### 2.2.3 Logistic Regression

Motivated by the generative models computation we can consider:

$$p(\mathcal{C}_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})), \tag{2.134}$$

with an arbitrary feature map $\boldsymbol{\phi}$ s.t. $\boldsymbol{\phi}_0(x) = 1$. We shall determine the parameters via maximum likelihood.

Consider a dataset $D = \{(\boldsymbol{\phi}(\boldsymbol{x}_n), t_n)\}_{n=1}^N$, already feature transformed, with $t_n \in \{0, 1\}$ and class $\mathcal{C}_1$ corresponding to $t = 1$. The maximum likelihood function, under an i.i.d. assumption, is given by:

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} \Big(\sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}))\Big)^{t_n} \Big(1 - \sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}))\Big)^{1-t_n}. \tag{2.135}$$

Defining the error function $E(\boldsymbol{w})$ as the negative log we obtain:

$$E(\boldsymbol{w}) = -\log p(\boldsymbol{t}|\boldsymbol{w}) = -\sum_{n=1}^{N} \big(t_n \log y_n + (1 - t_n)\log(1 - y_n)\big), \tag{2.136}$$

where $y_n = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n))$. Before computing $\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$ we show the following:

$$\frac{d\sigma(a)}{da} = \sigma(a)(1 - \sigma(a)). \tag{2.137}$$

The computation is simply:

$$\begin{aligned} \frac{d}{da}\left(\frac{1}{1 + \exp(-a)}\right) &= \frac{\exp(-a)}{(1 + \exp(-a))^2} \\ &= \frac{1 + \exp(-a) - 1}{(1 + \exp(-a))^2} \\ &= \frac{\sigma(a)^2}{\sigma(a)} - \sigma(a)^2 \\ &= \sigma(a)(1 - \sigma(a)). \end{aligned} \tag{2.138}$$

Using this result we compute (Exercise 4.13 [Bis06]):

$$\begin{aligned} \nabla_{\boldsymbol{w}} E(\boldsymbol{w}) &= -\Big(\sum_{n=1}^{N} t_n \frac{\sigma'\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)}{\sigma\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)}\boldsymbol{\phi}(\boldsymbol{x}_n) + (t_n - 1)\frac{\sigma'\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)}{\big(1 - \sigma\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\big)}\boldsymbol{\phi}(\boldsymbol{x}_n)\Big) \\ &= -\sum_{n=1}^{N} t_n\big(1 - \sigma\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\big)\boldsymbol{\phi}(\boldsymbol{x}_n) + (1 - t_n)\sigma\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big)\boldsymbol{\phi}(\boldsymbol{x}_n) \\ &= \sum_{n=1}^{N} \big(\sigma\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\big) - t_n\big)\boldsymbol{\phi}(\boldsymbol{x}_n) \\ &= \sum_{n=1}^{N} \big(y_n - t_n\big)\boldsymbol{\phi}(\boldsymbol{x}_n). \end{aligned} \tag{2.139}$$

This no longer has a closed-form solution due to the nonlinearity in $\sigma$. However, because the error function is strictly convex (which we prove next), we have a unique minimizer which can be determined with recourse to optimization algorithms such as gradient descent.

The computation of the Hessian follows (Exercise 4.15 [Bis06]):

$$\begin{aligned} \boldsymbol{H} = \nabla\nabla E(\boldsymbol{w}) = \nabla_{\boldsymbol{w}}\Big(\sum_{n=1}^{N} \big(y_n - t_n\big)\boldsymbol{\phi}(\boldsymbol{x}_n)\Big) = \sum_{n=1}^{N} \nabla_{\boldsymbol{w}}\big(\sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n))\boldsymbol{\phi}(\boldsymbol{x}_n)\big) = \\ = \sum_{n=1}^{N} y_n(1 - y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T. \end{aligned} \tag{2.140}$$

Given a vector $\boldsymbol{v} \in \mathbb{R}^M$, $\boldsymbol{v} \neq \mathbf{0}$, where $M$ is the dimension of the embedding given by $\boldsymbol{\phi}$, we have:

$$\begin{aligned} &\boldsymbol{v}^T \Bigg( \sum_{n=1}^{N} y_n(1-y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T \Bigg) \boldsymbol{v} = \sum_{n=1}^{N} \lambda_n \boldsymbol{v}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{v} = \\ &= \sum_{n=1}^{N} \lambda_n \|\boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{v}\|_2^2 > 0, \end{aligned} \tag{2.141}$$

where $\lambda_n := y_n(1-y_n) > 0$ since $y_n \in (0,1)$. Thus, we conclude that $\boldsymbol{H}$ is positive definite and $E(\boldsymbol{w})$ is strictly convex.

**Multiclass Classification:** As seen before, when talked about generative models, we can write:

$$p(\mathcal{C}_k|\boldsymbol{\phi}(\boldsymbol{x})) = y_k(\boldsymbol{\phi}(\boldsymbol{x})) = \frac{\exp(a_k)}{\sum_{j=1}^{|\mathcal{C}|} \exp(a_j)}, \tag{2.142}$$

with linear terms $a_k = \boldsymbol{w}_k^T \boldsymbol{\phi}(\boldsymbol{x})$. Using a 1-of-$K$ coding scheme, i.e. each class is represented by a $K-$vector of binary variables with a non-zero entry in the respective class, the likelihood function under and i.i.d. assumption can be written as:

$$p(\boldsymbol{T}|\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\boldsymbol{\phi}(\boldsymbol{x}_n))^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_k\big(\boldsymbol{\phi}(\boldsymbol{x}_n)\big)^{t_{nk}}, \tag{2.143}$$

where $\boldsymbol{T}_{N\times K}$ is a matrix with elements $t_{nk}$. Again, we define the error function:

$$E(\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = -\log p(\boldsymbol{T}|\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \log y_{nk}. \tag{2.144}$$

We are now interested in computing $\nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = 0$ which means computing the terms (Exercise 4.17 [Bis06]):

$$\frac{\partial y_k}{\partial \boldsymbol{w}_j} = \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial \boldsymbol{w}_j} \tag{2.145}$$

We have two situations, either $j = k$ or $j \neq k$. For the case $j = k$ we obtain:

$$\frac{\partial y_k}{\partial a_k} = \frac{\exp(a_k) \sum_{j=1}^{|\mathcal{C}|} \exp(a_j) - \exp(2a_k)}{\Big( \sum_{j=1}^{|\mathcal{C}|} \exp(a_j) \Big)^2} = y_k - y_k^2 = y_k(1-y_k). \tag{2.146}$$

For $j \neq k$ we have:

$$\frac{\partial y_k}{\partial a_j} = -\frac{\exp(a_j)\exp(a_k)}{\Big( \sum_{j=1}^{|\mathcal{C}|} \exp(a_j) \Big)^2} = -y_j y_k. \tag{2.147}$$

Which we can write together as:

$$\frac{\partial y_k}{\partial a_j} = y_k(\boldsymbol{I}_{kj} - y_j), \tag{2.148}$$

with $\boldsymbol{I}$ the identity matrix.

Using this result we have (Exercise 4.18 [Bis06]):

$$\begin{aligned}
\nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, ..., \boldsymbol{w}_K) &= -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\frac{y_{nk}(\boldsymbol{I}_{kj} - y_{nj})}{y_{nk}}\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}(\boldsymbol{I}_{kj} - y_{nj})\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\boldsymbol{I}_{kj}\boldsymbol{\phi}(\boldsymbol{x}_n) + \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}y_{nj}\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= -\sum_{n=1}^{N} t_{nj}\boldsymbol{\phi}(\boldsymbol{x}_n) + \sum_{n=1}^{N} y_{nj}\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \sum_{n=1}^{N}(y_{nj} - t_{nj})\boldsymbol{\phi}(\boldsymbol{x}_n),
\end{aligned} \tag{2.149}$$

using the fact that $\sum_{k=1}^{K} t_{nk} = 1$. As in the binary case, it can be shown that the Hessian is positive definite (Exercise 4.20 [Bis06] - not solved) and one can obtain a minimizer with recourse to gradient descent methods.

We now show two implementations of logistic regression using the same datasets as in the perceptron examples.

**Example 2.2.3.** *(**Binary classification with input space** $\mathbb{R}^2$)*
*Suppose we have input* $\boldsymbol{x} \in \mathbb{R}^2$ *with label* $y \in \{-1, 1\}$ *and we have access to the dataset* $D = \{((-1, 0), 0), ((0, 0.25), 1), ((1, 1), 1), ((1, -1), 0)\}$. *We take* $\phi$ *to be the identity and initialize the parameters as zero. The following figure, fig. 2.19, shows the resulting hyperplane of the logistic regression algorithm after 1 epoch of full gradient descent (which will be explained thoroughly in Chapter 3).*

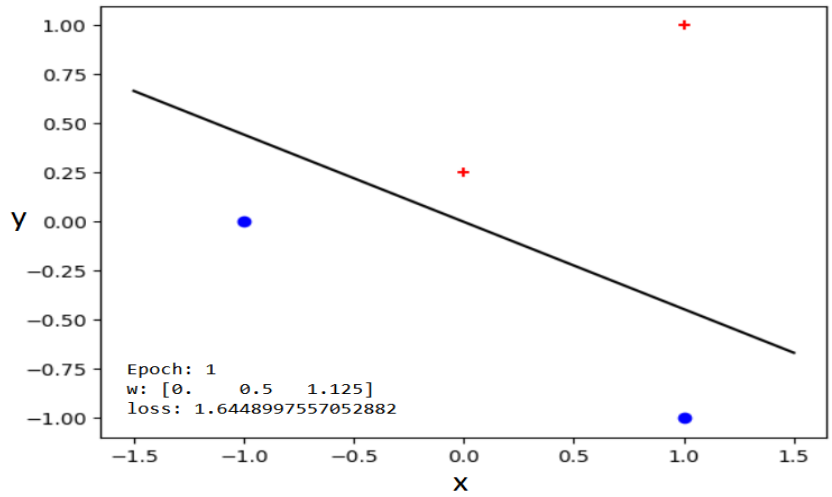


Figure 2.19: Hyperplane update of the logistic regression algorithm over 1 epoch.

*Further updates optimize the separating region and reduce the loss.*

**Example 2.2.4.** *(**Multiclass classification with input space** $\mathbb{R}^{64}$)*
*In this example we consider the UCI handwritten digits dataset as in the perceptron example. Running an implementation of the multiclass logistic regression algorithm, shown above, we obtain the following*

*figure (fig. 2.18) which shows the error function and the train and test accuracies as a function of the epoch.*

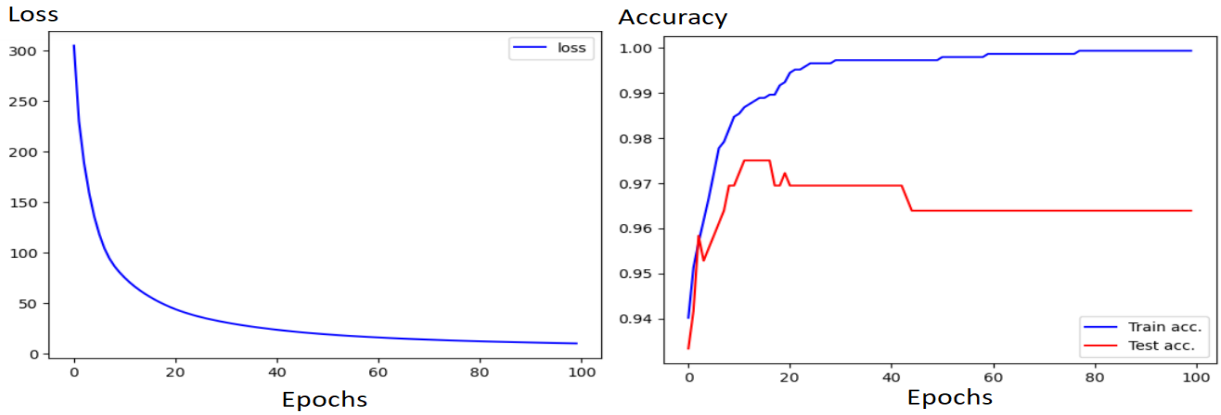


Figure 2.20: Accuracies of the multiclass perceptron algorithm on the UCI handwritten digits dataset. The accuracy is shown only from the first epoch onwards (does not include the loss at initialization).

*Again this method achieves quite good results, for this particular dataset, with a test accuracy of* 96% *after 100 epochs of training using stochastic gradient descent (which we will cover in the following chapter). Similarly to the multiclass perceptron example, over the training set the network has an accuracy of* 100% *which shows that the data is linearly separable.*

**Source code:** For the source code with the implementation of these example see the file "logisticregression.ipynb".

### 2.2.4 Bayesian Logistic Regression

Given a dataset $D = \{(\boldsymbol{x}_i, t_i)\}_{i=1}^N$, in the context of binary classification, i.e. $t_i \in \{0, 1\}$, we want to consider a Bayesian treatment of logistic regression. Given a prior distribution $p(\boldsymbol{w})$ we want to evaluate the posterior given by:

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)}. \tag{2.150}$$

We have seen that the likelihood function in this setting is given by:

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} \Big(\sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}))\Big)^{t_n} \Big(1 - \sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}))\Big)^{1-t_n}. \tag{2.151}$$

If we take a Gaussian prior as we have been doing, or any other that quickly comes to mind, its clear that computing $p(D) = \int p(D|\boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}$ is analytically intractable. Thus, in order to keep exploring the conjugate properties of a Gaussian prior and a Gaussian likelihood function, which results in a Gaussian posterior, we will make use of a Gaussian approximation method on the likelihood function, called the **Laplace Approximation** which we introduce next.

**Laplace Approximation method:** Consider first the case of a single variable $z$ with distribution given by:

$$p(z) = \frac{f(z)}{Z}, \tag{2.152}$$

where $Z = \int f(z)dz$ is the normalization constant. The goal of the method is to determine a Gaussian distribution $q(z)$ centered on a mode of the distribution $p(z)$. Suppose we have determined such a

mode, $z = z_0$, which in general might be a local maximum of $p(z)$, we have then:

$$\left.\frac{df(z)}{dz}\right|_{z=z_0} = 0, \text{ and } \left.\frac{d^2 f(z)}{dz^2}\right|_{z=z_0} < 0. \tag{2.153}$$

Because the exponential term of a Gaussian distribution is quadratic in the variables its logarithm will be quadratic in the variables, thus we consider the Taylor expansion of $\log f(z)$ up to second order around the maximum $z = z_0$, i.e:

$$\log f(z)\big|_{z=z_0} \simeq \log f(z_0) - \frac{1}{2}A(z - z_0)^2, \tag{2.154}$$

where:

$$A = -\left.\frac{d^2}{dz^2}\log f(z)\right|_{z=z_0}. \tag{2.155}$$

Then, around $z = z_0$, we have:

$$f(z) \simeq f(z_0)\exp\bigg(-\frac{A}{2}(z - z_0)^2\bigg), \tag{2.156}$$

and the approximation Gaussian is given by:

$$q(z) = \left(\frac{A}{2\pi}\right)^{1/2}\exp\bigg(-\frac{A}{2}(z - z_0)^2\bigg) = \mathcal{N}(z|z_0, A^{-1}). \tag{2.157}$$

The extension to an $M-$dimensional distribution $p(\boldsymbol{z})$, by similar computations, is given by:

$$q(\boldsymbol{z}) = \frac{|\boldsymbol{A}|^{1/2}}{(2\pi)^{M/2}}\exp\bigg(-\frac{1}{2}(\boldsymbol{z} - \boldsymbol{z_0})^T\boldsymbol{A}(\boldsymbol{z} - \boldsymbol{z_0})\bigg) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{z}_0, \boldsymbol{A}^{-1}), \tag{2.158}$$

where the Hessian matrix $\boldsymbol{A}_{M\times M}$ is given by:

$$\boldsymbol{A} = -\nabla\nabla\log f(\boldsymbol{z})\big|_{\boldsymbol{z}=\boldsymbol{z}_0}. \tag{2.159}$$

Typically, the mode $\boldsymbol{z}_0$ is obtained numerically to which we then apply methods for approximating the Hessian $\boldsymbol{A}$. Note that the approximation will depend on the local maximum we obtain after this numerical procedure. It will of course also be severely limited by the fact that it approximates the true distribution at a single point and thus overlooks any global properties of the true distribution.

**Laplace approximation of the posterior:** Back to the starting problem, suppose we have a Gaussian prior given by:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m_0}, \boldsymbol{S_0}). \tag{2.160}$$

The term $f(z)$ of the Laplace approximation corresponds to the term in the posterior given by:

$$p(\boldsymbol{w}|\boldsymbol{t}) \propto p(\boldsymbol{t}|\boldsymbol{w})p(\boldsymbol{w}). \tag{2.161}$$

Suppose we have found, via some numerical method, a maximum of (2.161) around which we will expand the function, call it $\boldsymbol{w}_{MAP}$. This defines the mean of the Gaussian approximation. Replacing the likelihood with (2.151) and with the prior given by (2.160) we obtain:

$$\begin{aligned}\log p(\boldsymbol{w}|\boldsymbol{t}) = &-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{m_0})^T\boldsymbol{S_0^{-1}}(\boldsymbol{w} - \boldsymbol{m_0}) \\ &+\sum_{n=1}^{N}(t_n\log(y_n)) + (1 - t_n)\log(1 - y_n) + c,\end{aligned} \tag{2.162}$$

where $y_n = \sigma(\boldsymbol{w^T \phi(x_n)})$ and some constant $c \in \mathbb{R}$. To find the covariance matrix of the Gaussian approximation we need to take second derivatives of the negative of the expression above, as seen in (2.159). We obtain:

$$\boldsymbol{S_N^{-1}} = -\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} \log p(\boldsymbol{w}|\boldsymbol{t}) = \boldsymbol{S_0^{-1}} + \sum_{n=1}^{N} y_n(1 - y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T. \tag{2.163}$$

And the Laplace approximation of the posterior is given by:

$$q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}_{MAP}, \boldsymbol{S_N}). \tag{2.164}$$

**Predictive distribution:** The predictive distribution is given by (2.61), which we restate here for convenience:

$$p(t|D) = \int p(t|\boldsymbol{w})p(\boldsymbol{w}|D)d\boldsymbol{w}, \tag{2.165}$$

For the two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, given a new input $\boldsymbol{x}$ with feature vector $\boldsymbol{\phi}(\boldsymbol{x})$ we have then:

$$\begin{aligned} p(\mathcal{C}_1|\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{t}) &= \int p(\mathcal{C}_1|\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{t})d\boldsymbol{w} \simeq \int \sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})q(\boldsymbol{w})d\boldsymbol{w} \\ p(\mathcal{C}_2|\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{t}) &= \int p(\mathcal{C}_2|\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{t})d\boldsymbol{w} \simeq \int (1 - \sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}))q(\boldsymbol{w})d\boldsymbol{w} \end{aligned} \tag{2.166}$$

For several methods for computing these integrals see Section 4.5.2 of [Bis06].

### 2.2.5 Summary

In this section we explored linear methods of classification. In particular we looked at two distinct modeling paradigms, which model the probability of assigning a certain label given the input and **generative models** which construct joint distributions over both input and label space.

- **Perceptron:** The perceptron algorithm, a non-probabilistic linear discriminant model, essentially divides the input space via hyperplanes and assigns a label to each region. In particular we saw that, in the case of **linearly separable data**, by Novikoff's Theorem, the algorithm converges in a finite number of steps to a separating hyperplane. However, this hyperplane will not, in general, find the best separating margin. Another limitation of the perceptron, due to its linearity, was that it could not learn a simple function such as the XOR function.

- **Generative approach:** As an example of this approach we constructed a joint distribution over input and label space by modeling the inputs, conditioned on the labels, with a Gaussian distribution whose mean depends on the label and a covariance matrix which is the same for all labels. Considering a constant value for the prior over the labels, in the binary classification setting, we showed that, together with the assumption of equal covariance, the decision boundaries where linear. Then, via maximum likelihood, we determined all parameters, the prior probabilities, the means and the covariance matrix. This joint probability allows us to then generate synthetic data as opposed to discriminant models, since they do not model a distribution over the input space. However this requires fitting a lot more parameters than discriminant models which may still work better if our model assumption for the input distribution is not close to the underlying distribution.

- **Logistic Regression:** Motivated by the discussion on generative models we define the probability of a class given the input as the **sigmoid** of a linear model, in the parameters, over some feature space. Then, by considering the maximum likelihood approach, we show that, despite

not having a closed-form solution, the empirical risk is a convex function and we are guaranteed to find a unique minimizer by using an appropriate minimization method, such as gradient descent. The predictor partitions the transformed feature space via parallel hyperplanes, where each assigns a constant probability over a particular label. Again, due to the linearity, logistic regression can not solve the XOR problem without a transformation of features. We further saw how to extend logistic regression to the multiclass case via a **1-of-$K$ coding** scheme.

We finalized this section looking at a Bayesian approach to logistic regression.

- **Bayesian Logistic Regression:** The Bayesian paradigm, as we have considered in linear regression, leads to an analytically intractable posterior due to the complex form of the likelihood function. In order to obtain an analytical solution we can consider the **Laplace approximation** of the posterior. This approximation method essentially finds a mode of the posterior, which may only be a local maximum, and constructs a Gaussian centred on that mode by taking as exponential term the second order Taylor expansion around the mean. By inspection we then see the form of the covariance matrix as the Hessian matrix which allows us to completely define the distribution. The predictor can then be built out of the posterior in the usual way, where the tractability of the integrals can be explored further in Section 4.5.2 of [Bis06]. Of course, taking a Gaussian approximation may not be well suited for the posterior distribution which, in general, will not be unimodal. These sacrifices however are made in favour of an analytical result and in practice we should rely on numerical methods when determining the posterior.

This finalizes our brief overview of linear methods in machine learning following mainly Chapters 3 and 4 of [Bis06]. In the context of classification we clearly saw the limitation of linear methods, such as we presented them. This motivates the introduction of non-linear methods, in the next chapter, where we look in particular at neural networks.

# Chapter 3

# Neural Networks

In this chapter we introduce the concept of a **neural network**, another method for supervised learning via function approximation. We will use as guideline Chapter 5 of [Bis06] but we will often follow other sources, such as [Mur22] and [BGKP21]. We start by providing a rigorous definition of the most fundamental architecture of a neural network, known as a **feedfoward fully connected neural network**, following [BGKP21]. Next we will see how the parameters of a neural network adapt during the training process. In particular we will look at **gradient descent** (full and stochastic gradient descent) and the method of **backpropagation**. Due to the, usually, large number of parameters on a neural network, they are prone to overfitting phenomena and next we cover some methods of regularization which are typically used in this context. We finalize the chapter by looking at **Bayesian neural networks**, where, unlike the Bayesian Regression case with conjugate priors, the posterior is not analytically tractable. It can still be approximated however, and to this end we will use the **Laplace approximation** method as we have done for the case of Bayesian logistic regression.

## 3.1 Feedforward Fully Connected Neural Networks

So far, the models we have considered construct predictors by using a linear combination of a fixed number of, possibly, nonlinear basis functions, $\{\phi_j\}_{j=1}^M$, of the form:

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\bigg(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\bigg), \tag{3.1}$$

where we have taken $f$ to be the identity function, in the case of regression, and a nonlinear **activation function** (sigmoid/softmax) in the case of classification. Neural networks extend this paradigm by constructing adaptable basis functions by themselves depending on parameters, i.e. the $\phi_j(\boldsymbol{x})$ become $\phi_j(\boldsymbol{x}, \boldsymbol{\theta})$, with its own set of parameters $\boldsymbol{\theta}$. Deep neural networks are constructed by applying this idea recursively witch each added layer. Throughout the chapter we will consider the simplest example of neural network, known as feedforward neural network or multilayer perceptron (MLP), and we begin by precisely showing a formal definition of it following [BGKP21].

**Definition 3.1.1.** ***(FFFC Neural Network)***
*A feed-forward fully connected neural network is defined by its architecture $a = (\boldsymbol{N}, \sigma)$, where $\boldsymbol{N} \in \mathbb{N}^{L+1}$ is the number of neurons which, for each layer $\ell = 0, ..., L$, has $N_\ell$ neurons, and where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function. The number of parameters $P$ of the network is given by:*

$$P(\boldsymbol{N}) := \sum_{\ell=1}^{L} N_\ell N_{\ell-1} + N_\ell. \tag{3.2}$$

*The output at each layer $\ell$ is defined by $y^{(\ell)} : \mathbb{R}^{N_0} \times \mathbb{R}^{P(\boldsymbol{N})} \mapsto \mathbb{R}^{N_\ell}$, even though not all parameters are required, and the realization function of the network is defined as $y_a : \mathbb{R}^{N_0} \times \mathbb{R}^{P(\boldsymbol{N})} \to \mathbb{R}^{N_L}$, i.e. $y_a = y^{(L)}(x, \theta)$. For each input $x \in \mathbb{R}^{N_0}$ and the parameters:*

$$\theta = \{\theta^{(\ell)}\}_{\ell=1}^{L} = \{(W^{(\ell)}, b^{(\ell)})\}_{\ell=1}^{L} \in \mathbb{R}^{P(N)}, \tag{3.3}$$

*we have:*

$$\begin{aligned} y^{(1)}(x,\theta) &= W^{(1)}x + b^{(1)}, \\ z^{(\ell)}(x,\theta) &= \sigma(y^{(\ell)}(x,\theta)), \quad \ell = 1, ..., L-1, \quad (\sigma \text{ applied component wise}), \\ y^{(\ell+1)}(x,\theta) &= W^{(\ell+1)}z^{\ell}(x,\theta) + b^{(\ell+1)}, \quad \ell = 1, ..., L-1. \end{aligned} \tag{3.4}$$

*The $W^{(\ell)} \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ are called the weight matrices and the $b^{(l)} \in \mathbb{R}^{N_\ell}$ the bias vectors. The FFFC neural network is said to have width given by $\max\{N_\ell\}_{\ell=1}^{L}$ and is called a deep if $L > 2$ and shallow if $L = 2$.*

**Remark 3.1.1.** *The bias vector can always be implicitly introduced in the weights $W$ by the following transformation:*

$$x \to \begin{bmatrix} x \\ 1 \end{bmatrix}, \left(W^{(\ell)}, b^{(\ell)}\right) \to \begin{bmatrix} W^{(\ell)} & b^{(\ell)} \\ 0 & 1 \end{bmatrix}, \ \ell = 1, ..., L-1, \text{ and } \left(W^{(L)}, b^{(L)}\right) \to \begin{bmatrix} W^{(L)} & b^{(L)} \end{bmatrix}. \tag{3.5}$$

*We introduce an extra dimension on the input vector, with value one, and also on the output vector of every layer except the last layer.*

Typical activation functions used in the hidden layers are the rectified linear unit (ReLU), given by $\sigma(x) = \max\{0, x\}$ and sigmoidal functions, such as the logistic function $\sigma(x) = 1/(1 + e^{-x})$ or the hyperbolic tangent, $\sigma(x) = \tanh(x)$. On the output layer the activation depends on the type of problem, typically in regression one uses the identity while in multiclass problems one uses the softmax function, we shall discuss this ahead.

**Remark 3.1.2.** *Note that if we remove the nonlinearities, i.e. the activation functions, then the whole network reduces to a single linear transformation, since it becomes simply a composition of linear maps, which would be equivalent to a neural network without any hidden layers.*

**Example 3.1.1.** ***(Forward computation with 2 hidden-layers)***
*We consider now a specific example of a FFFC neural network with 2 hidden-layers. The architecture of the neural network can be seen in the following diagram, fig. (3.1).*

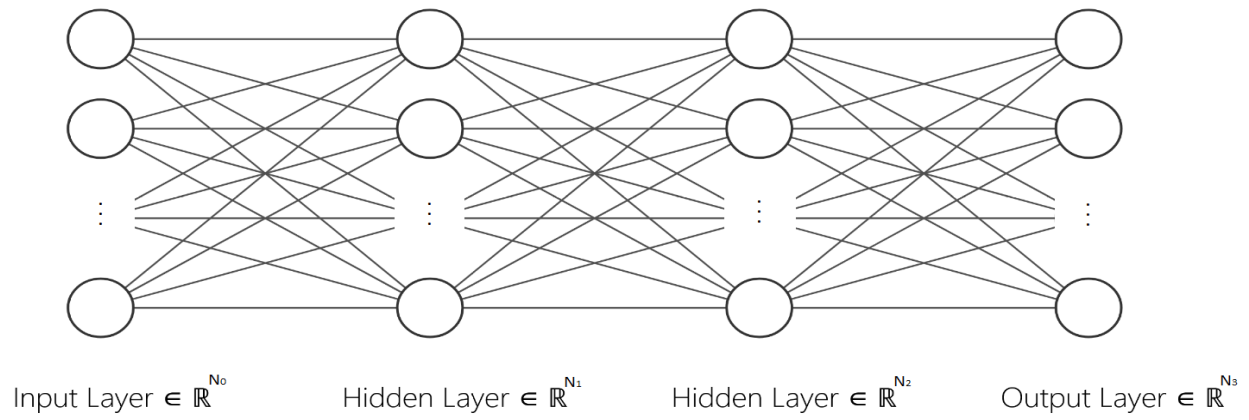


Figure 3.1: Schematic of a FFFC neural network with two hidden layers (drawn using NN SVG).

*The input variable $x \in \mathbb{R}^{N_0}$, $x = (x_1, ..., x_{N_0})$, goes in the neurons of the first hidden layer $\ell = 1$ to compute:*

$$y_j^{(1)}(x) = \sum_{i=1}^{N_0} w_{ji}^{(1)} x_i + b_i^{(1)}, \;\; j = 1, ..., N_1. \tag{3.6}$$

*In vector notation, the first hidden layer computes the pre-activation:*

$$y^{(1)}(x) = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \dots & w_{1N_0}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N_1 1}^{(1)} & w_{N_1 2}^{(1)} & \dots & w_{N_1 N_0}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{N_0} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_{N_1}^{(1)} \end{bmatrix} = W^{(1)} x + b^{(1)}, \tag{3.7}$$

*with the output of the first hidden layer being given by:*

$$z^{(1)}(x) = \sigma(y^{(1)}(x)) = \sigma(W^{(1)} x + b^{(1)}), \quad \sigma \text{ applied component wise.} \tag{3.8}$$

*The second layer receives the $z^{(1)}(x)$ as input and computes:*

$$\begin{aligned} y_k^{(2)}(x) &= \sum_{j=1}^{N_1} w_{kj}^{(2)} \sigma\Big( \sum_{i=1}^{N_0} w_{ji}^{(1)} x_i + b_i^{(1)} \Big) + b_k^{(2)}, \;\; p = 1, ..., N_2, \\ z_k^{(2)}(x) &= \sigma(y_k^{(2)}(x)). \end{aligned} \tag{3.9}$$

*In vector notation we can write:*

$$y^{(2)}(x) = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & \dots & w_{1N_1}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N_2 1}^{(2)} & w_{N_2 2}^{(2)} & \dots & w_{N_2 N_1}^{(2)} \end{bmatrix} \begin{bmatrix} z_1^{(1)} \\ \vdots \\ z_{N_1}^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ \vdots \\ b_{N_2}^{(2)} \end{bmatrix} = W^{(2)} z^{(1)}(x) + b^{(2)}, \tag{3.10}$$

*and $z^{(2)}(x) = \sigma(W^{(2)} z^{(1)}(x) + b^{(2)})$. Finally the output layer computes:*

$$y_p^{(3)}(x) = \sum_{k=1}^{N_2} w_{pk}^{(3)} \sigma\Big( \sum_{j=1}^{N_1} w_{kj}^{(2)} \sigma\Big( \sum_{i=1}^{N_0} w_{ji}^{(1)} x_i + b_i^{(1)} \Big) + b_k^{(2)} \Big) + b_p^{(3)}, \;\; p = 1, ..., N_3, \tag{3.11}$$

*which in vector notation is:*

$$y_a(x) = y^{(3)}(x) = \begin{bmatrix} w_{11}^{(3)} & w_{12}^{(3)} & \dots & w_{1N_2}^{(3)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N_3 1}^{(3)} & w_{N_3 2}^{(3)} & \dots & w_{N_3 N_2}^{(3)} \end{bmatrix} \begin{bmatrix} z_1^{(2)} \\ \vdots \\ z_{N_2}^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(3)} \\ \vdots \\ b_{N_3}^{(3)} \end{bmatrix} = W^{(3)} z^{(2)}(x) + b^{(3)}. \tag{3.12}$$

*These calculations, moving from the input to the output through the network, are called **forward propagation**. In this particular case we take the activation of the last layer to be the identity.*

In terms of approximation properties of neural networks we have the following fundamental result:

**Theorem 3.1.1.** ***(Universal Approximation Theorem)***
*Let $C(X, Y)$ be the space of continuous function from a set $X$ to a set $Y$ and consider an activation function $\sigma \in C(\mathbb{R}, \mathbb{R})$. Then, $\sigma$ is not a polynomial if and only if, for every $n \in N$, $m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m)$ and $\epsilon > 0$ there exist $k \in \mathbb{N}$, $W^{(1)} \in \mathbb{R}^{k \times n}$,$b \in \mathbb{R}^k$, $W^{(2)} \in \mathbb{R}^{m \times k}$ such that:*

$$\begin{aligned} &\sup_{x \in K} \|f(x) - g(x)\|_2 < \epsilon, \;\; \text{where,} \\ &\quad g(x) = W^{(2)}\big(\sigma(W^{(1)} x + b)\big). \end{aligned} \tag{3.13}$$

In other words, with arbitrary $n, m \in \mathbb{N}$, for a neural network with input $x \in \mathbb{R}^n$ and output $y(x) \in \mathbb{R}^m$, there exists $k \in \mathbb{N}$ such that, with a single hidden layer of width $k$, the realization function $y_a$ can approximate any continuous function $f$, with domain on a compact set $K$, provided that the activation function $\sigma$ is not a polynomial. This result appears in an earlier form by George Cybenko [Cyb89] for the case of sigmoid activation functions and was extended by Hornik et al. [HSW89] to arbitrary "squashing functions". Although this result shows that shallow neural networks can learn any continuous function, there are both experimental and theoretical results favouring deeper networks, see for example [SS16], [Tel16] and [PV17].

**Symmetries in Weight Space** The mapping $(a, \theta) \to y_a(\cdot, \theta)$ is highly non-injective since there are several symmetries in weight-space which leave the realization function $y_a(\cdot, \theta)$ unchanged. A trivial case which shows the non-injectivity, in the example above, would be to take $W^{(2)} = 0$ and have the final output be just $b^{(2)}$. Clearly the output is the same for whatever values of $W^{(1)}$ and $b^{(1)}$.

As a more interesting example, consider a single hidden-layer neural network witch activation function $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, because this is an odd function, $\tanh(-x) = -\tanh(x)$, at the output of neuron $j$ of the hidden layer and at output $k$ of the output layer we have the following term:

$$w_{kj}^{(2)}(\tanh({w_j^{(1)}}^T x)) = -w_{kj}^{(2)}(\tanh(-{w_j^{(1)}}^T x)). \tag{3.14}$$

Thus, for all $k = 1, ..., N_2$, we can redefine the weights $\hat{w}_{kj}^{(2)} = -w_{kj}^{(2)}$ and $\hat{w}_j^{(1)} = -w_j^{(1)}$, and these lead to the same realization function. If we have $M$ neurons, we can can envision performing this weight shift at each neuron which gives a total of $2^M$ possible different weights leading to a unique realization function.

A different symmetry arises when we interchange the weights of different units, for example:

$$\begin{aligned} \text{neuron } i: \quad & z_i^{(1)} = \tanh\left(w_{i0}^{(1)} + w_{i1}^{(1)} x_1 + ... + w_{iN_1}^{(1)} x_{N_1}\right), \\ \text{neuron } j: \quad & z_j^{(1)} = \tanh\left(w_{j0}^{(1)} + w_{j1}^{(1)} x_1 + ... + w_{jN_1}^{(1)} x_{N_1}\right). \end{aligned} \tag{3.15}$$

If we shift all weights of neuron $i$ with those of neuron $j$, which we represent after the shift as $z_{i\to j}^{(1)}, z_{j\to i}^{(1)}$, on output $k = 1, ..., N_2$ of the output layer we have:

$$\begin{aligned} y_k^{(2)} &= w_{k0}^{(2)} + w_{k1}^{(2)} z_1^{(1)} + ... + w_{ki}^{(2)} z_i^{(1)} + ... + w_{kj}^{(2)} z_j^{(1)} + ... + w_{kN_2}^{(2)} z_{N_1}^{(1)} \\ &= w_{k0}^{(2)} + w_{k1}^{(2)} z_1^{(1)} + ... + w_{kj}^{(2)} z_{i\to j}^{(1)} + ... + w_{ki}^{(2)} z_{j\to i}^{(1)} + ... + w_{kN_2}^{(2)} z_{N_1}^{(1)}. \end{aligned} \tag{3.16}$$

Thus, we see that the realization function remains unchanged by this permutation. This corresponds to a permutation of the $i$ and $j$ rows of the weight matrix $W^{(1)}$ and a permutation of columns $i$ and $j$ of matrix $W^{(2)}$, there are $M!$ such permutations.

In total, these amount to $2^M M!$ configurations of all the weights $\theta$ that lead to the same realization function for a FFFC neural network with one hidden layer. For a larger number of layers the possible configurations will be given by products of these factors. This result was in fact shown to account for all possible (non-accidental) symmetries and to be independent on the particular choice of the hyperbolic tangent as an activation function, see [KK94].

## 3.2 Neural Network Training

The training of the neural network is performed by following the empirical risk minimization paradigm we studied in Chapter 1. As we did before, in the case of linear regression, we can have a probabilistic

interpretation of the output by considering the following distribution:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}\big(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1}\big), \tag{3.17}$$

where the mean $y(\boldsymbol{x}, \boldsymbol{w})$ is now the output of the neural network $y_a(\boldsymbol{x}, \boldsymbol{w})$. Given a training dataset:

$$D = \{(\boldsymbol{x}_1, t_1), ..., (\boldsymbol{x}_N, t_N)\}, \tag{3.18}$$

and depending on the learning task, we will have the following empirical risks:

- **Regression Problem:** By the equivalence of maximum likelihood and least-squares, as we have seen in Chapter 2, to determine $\boldsymbol{w}$ we need to minimize the following:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\big(y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n\big)^2. \tag{3.19}$$

  However now, due to the non-linearity of $y(\boldsymbol{x}, \boldsymbol{w})$ as a function of $\boldsymbol{w}$, the error function $E(\boldsymbol{w})$ may be highly non-convex in $\boldsymbol{w}$. Thus, minimization with respect to $\boldsymbol{w}$ will not, in general, correspond to a global minimizer.

- **Binary Classification Problem:** If we denote $t = 1$ for class $\mathcal{C}_1$ and $t = 0$ for class $\mathcal{C}_2$, we can consider a sigmoidal activation function on the output of the neural network:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sigma\big(y^{(L)}(\boldsymbol{x}, \boldsymbol{w})\big) = \frac{1}{1 + \exp\left(-y^{(L)}(\boldsymbol{x}, \boldsymbol{w})\right)}, \tag{3.20}$$

  such that $y(\boldsymbol{x}, \boldsymbol{w}) \in [0, 1]$ can be interpreted as the conditional probability $p(\mathcal{C}_1|\boldsymbol{x})$ where $p(\mathcal{C}_2|\boldsymbol{x}) = 1 - y(\boldsymbol{x}, \boldsymbol{w})$. The conditional distribution of the target is a Bernoulli distribution given by:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = y(\boldsymbol{x}, \boldsymbol{w})^t(1 - y(\boldsymbol{x}, \boldsymbol{w}))^{1-t}. \tag{3.21}$$

  The likelihood function, under an i.i.d. assumption, is given by:

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{n=1}^{N} y(\boldsymbol{x}_n, \boldsymbol{w})^{t_n}(1 - y(\boldsymbol{x}_n, \boldsymbol{w}))^{1-t_n}. \tag{3.22}$$

  Maximizing the likelihood function is equivalent to minimizing the negative log likelihood which is easily seen to be:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} t_n \log\big(y(\boldsymbol{x}_n, \boldsymbol{w})\big) + (1 - t_n)\log\big(1 - y(\boldsymbol{x}_n, \boldsymbol{w})\big). \tag{3.23}$$

  This is known as the **cross-entropy error function**.

- **Multiclass Classification Problem:** Suppose we have $K$ mutually exclusive classes. Each class $k$ can be encoded as a binary vector of size $K$, where the class $k$ is represented by the vector:

$$(0, ..., 0, \underbrace{1}_{\text{entry } k}, 0, ..., 0). \tag{3.24}$$

  The network would then have $K$ outputs at the last layer where each entry $k$ is interpreted as the probability of the corresponding binary variable being 1, i.e. $y_k(\boldsymbol{x}, \boldsymbol{w}) = p(t_k = 1|\boldsymbol{x})$. The likelihood function under an i.i.d. assumption would then be:

$$p(\boldsymbol{T}|\boldsymbol{X}, \boldsymbol{w}_1, ..., \boldsymbol{w}_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} y_k(\boldsymbol{x}_n, \boldsymbol{w})^{t_{nk}} \tag{3.25}$$

and the negative log likelihood becomes:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln\big(y_k(\boldsymbol{x}_n, \boldsymbol{w})\big), \tag{3.26}$$

where the activation function in the output layer is given by the **softmax** function:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \frac{\exp(y_k^{(L)}(\boldsymbol{x}, \boldsymbol{w}))}{\sum_j \exp(y_j^{(L)}(\boldsymbol{x}, \boldsymbol{w}))}, \tag{3.27}$$

satisfying $y_k \in [0, 1]$ and $\sum_k y_k = 1$.

**Parameter Optimization:** The ideal goal after an optimization procedure would be to determine all parameters $\boldsymbol{w}$ for which $E(\boldsymbol{w})$ attains its global minimum (or a "good enough" local minimum). To do this we can compute $\nabla E(\boldsymbol{w}) = 0$, which identifies minimums but also maximums and saddle points of the function. Because the error function is highly nonconvex, even if we find a minimum it may only be a local minimum. Furthermore, based on the weight symmetries discussed above, there will be several equivalent such weights. As we have seen, in the particular case of a network with two layers and $M$ neurons, there would be at least $2^M M!$ such points.

Since an analytical solution is not possible, an alternative way is to optimize the error function $E$ by using an iterative approach. We start by describing a first-order method of optimization known as gradient descent.

**Gradient Descent:** The gradient of a differentiable function $E(\boldsymbol{w})$ at a point $\boldsymbol{w}'$ is a vector which points in the direction of greatest rate of increase of $E(\boldsymbol{w})$ around $\boldsymbol{w}'$. By changing parameters in the opposite direction of the gradient we are moving in the direction of greatest decrease in $E(\boldsymbol{w})$ around $\boldsymbol{w}'$, this is the main idea behind gradient descent. What remains is to iteratively apply this step until a minimum is reached. The update rule of gradient descent is given by:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla_{\boldsymbol{w}} E(\boldsymbol{w})\big|_{\boldsymbol{w}^{(\tau)}}, \;\; \eta > 0, \tag{3.28}$$

where $\eta$ is known as the learning rate and $\tau$ is the update epoch. The algorithm further requires the specification of a starting point $\boldsymbol{w}^{(0)}$, typically taken to be normally distributed or zero initialized.

The choice of $\eta$ has a significant impact on the method. If $\eta$ is chosen to be constant but too large, it may fail to converge, if constant but too small it may take too long to converge. This is the case even if $E(\boldsymbol{w})$ where convex. We can find an upper bound for the rate $\eta < 2/L$, where $L$ is the Lipschitz constant of the gradient [Mur22]. However this constant is usually not know so the stepsize needs to be adapted using for example line search (see [Mur22], Section 8.2.2.2).

**Momentum:** Another issue with gradient descent is the so called *vanishing gradient* problem. In flat regions of the loss landscape the rate of convergence of gradient descent can be too slow. A way to deal with this problem is by using *momentum* methods. A simple implementation would be the following:

$$\begin{aligned} \boldsymbol{m}_\tau &= \beta \boldsymbol{m}_{\tau-1} + \nabla_{\boldsymbol{w}} E(\boldsymbol{w})\big|_{\boldsymbol{w}^{(\tau-1)}}, \\ \boldsymbol{w}_\tau &= \boldsymbol{w}_{\tau-1} - \eta \boldsymbol{m}_\tau, \end{aligned} \tag{3.29}$$

where $\boldsymbol{m}$ is called the momentum and $\beta \in (0, 1)$. If we let $\boldsymbol{g}_\tau = \nabla_{\boldsymbol{w}} E(\boldsymbol{w})\big|_{\boldsymbol{w}^{(\tau)}}$ we can write:

$$\boldsymbol{m}_\tau = \beta\left(\beta \boldsymbol{m}_{\tau-2} + \boldsymbol{g}_{\tau-2}\right) + \boldsymbol{g}_{\tau-1} = \ldots = \sum_{t=0}^{\tau-1} \beta^t \boldsymbol{g}_{\tau-t-1}. \tag{3.30}$$

If we are in a flat region where most past gradients are approximately constant, with value $\boldsymbol{g}$, we can write:

$$\boldsymbol{m}_\tau = \boldsymbol{g} \sum_{t=0}^{\tau-1} \beta^t. \tag{3.31}$$

As $\tau \to \infty$ we have:

$$\boldsymbol{m}_\tau = \frac{1}{1-\beta}\boldsymbol{g}, \tag{3.32}$$

Thus, if $\beta = 0.9$ for example, the gradient is scaled by 10 and we move faster out of a possible flat region. Several improvements over this idea have been made, see [Mur22] Section 8.4, particularly the popular ADAM method in Section 8.4.6.3.

Because the error function depends on the entire dataset, to evaluate $\nabla E$ one needs to process the entire dataset. A method typically used to reduce this computational cost is known as *stochastic gradient descent*.

**Stochastic Gradient Descent (SGD):** We are usually interested in minimizing a loss function which is averaged across the entire training set, i.e.:

$$\mathcal{L}(\boldsymbol{\theta}_t) = \frac{1}{N}\sum_{n=1}^{N} \ell(t_n, y(x_n, \boldsymbol{\theta}_t)), \tag{3.33}$$

for a general loss function $\ell$ which we could take to be least squares $\ell(t_n, y(x_n, \boldsymbol{\theta}_t)) = \big(t_n - y(x_n, \boldsymbol{\theta}_t)\big)^2$. Computing the gradient we obtain:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{\theta}_t) = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \ell(t_n, y(x_n, \boldsymbol{\theta}_t)) \tag{3.34}$$

In stochastic gradient descent, at each iteration $t$ we draw a training example $x_t$ randomly from the training set and we compute:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, x_t). \tag{3.35}$$

We could also consider a batch approach where a number $S \ll N$ of samples is selected such that:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{S}\sum_{n=1}^{S} \nabla_{\boldsymbol{\theta}} \ell(t_n, y(x_n, \boldsymbol{\theta}_t)). \tag{3.36}$$

In SGD, the update direction is a random vector whose expectation coincides precisely with the gradient of $E(\boldsymbol{w})$ and, the larger the batch we consider, the closer we are to the actual gradient. According to [Bis06], theoretically the rate of convergence of stochastic gradient descent is slower than full or batch gradient descent. However, in practice, the parameter update step in SGD is so much faster that it typically results in faster optimization. The choice of learning rate in SGD is also of major importance, there are several theoretical results in this regard, see Section 8.4.3 in [Mur22]. For other variations and improvements on this optimization step, see Sections 8.4.5-6 of [Mur22]. For a few theoretical results regarding the convergence of SGD see Section 14.3 of [SSBD14].

We now show an example of gradient descent methods in the context of the curve fitting example seen in Chapter 1.

**Example 3.2.1.** *(Polynomial curve fitting via gradient descent in the overparametrized regime)*

*Consider again a dataset* $D = \{(x_n, t_n)\}_{n=1}^N$ *generated as follows: we have a distribution over input space* $\mathcal{X} = [0, 1]$*, following* $x \sim Uni[0, 1]$*, together with an underlying labeling function* $g(x) = \sin(2\pi x)$ *and a Gaussian noise term* $\epsilon \sim \mathcal{N}(0, 0.15^2)$ *such that the target variable* $t$ *is given by* $t(x) = g(x) + \epsilon$*. We consider a linear model of the form:*

$$y(x, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j x^j, \tag{3.37}$$

*where the parameters are estimated via empirical risk minimization of least squares with weight decay, i.e. minimization of:*

$$\tilde{E}(\boldsymbol{w}, \lambda) = \frac{1}{N} \sum_{n=1}^{N} (y(x_n, \boldsymbol{w}) - t_n)^2 + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}. \tag{3.38}$$

*In this particular example we will take* $|D| = 10$ *and* $M = 30$*. We further generate a test dataset* $D_T$ *with* $|D_T| = 100$ *in the same way as specified above. The following figure, fig. 3.2, shows the empirical risk averaged over an ensemble* $\{D_i\}_{i=1}^{100}$ *and the test error on* $D_T$ *as a function of the epoch, for 5000 epochs, using full gradient descent (FGD), on-line SGD and SGD with a batch size of 5 samples.*

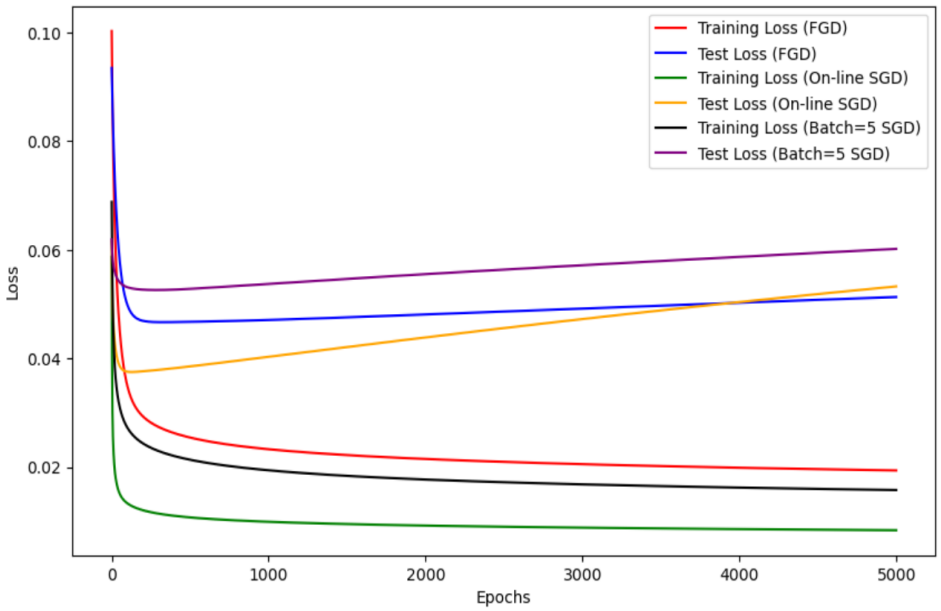


Figure 3.2: Empirical and test risk for FGD, On-line SGD and Batch SGD w/ batch size of 5.

*Note that the loss per epoch values are not directly comparable between the different methods. This is because the model parameters, in FGD, are updated after processing the entire dataset, while in SGD the model parameters are updated after processing each sample or batch of samples. Thus, the frequency of updates is different between the different methods. In particular, at each epoch, on-line SGD updates the weights 10 times while FGD updates once and SGD, with a batch of size 5, updates twice.*

*The analytical solution without regularization* ($\lambda = 0$) *obtains an empirical error averaged over the ensemble of* $E_{train}(\boldsymbol{w}) \approx 0.0244$ *and a test error* $E_{test}(\boldsymbol{w}) \approx 14049283584$ *while the analytical solution with regularization* ($\lambda = 0.01$) *obtains an average empirical error* $\tilde{E}_{train}(\boldsymbol{w}) \approx 0.0066$ *and a test error*

$\tilde{E}_{test}(\boldsymbol{w}) \approx 0.9290$. *We shall comment each curve, and values, in the corresponding methods which we describe now in more detail.*

- ***Analytical solution:*** *In the overparametrized regime, $M > N$, we have an infinite number of solutions where $\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2 = 0$, i.e. where the fitting curve passes exactly through the training data. Because (3.38) is semi-convex, this corresponds to a valley region of the loss landscape. The analytical solution, as given by (2.22), chooses the one among these with the smallest possible norm, by solving an optimization problem via Lagrange multipliers.*

  *Without regularization we have, as expected, a training error of almost zero where the discrepancy is probably a result of poor conditioning for some particular samples, when calculating the inverse to obtain the analytical solution. Due to overparametrization, which reflects a more complex hypothesis class, we obtain tremendous overfiting as is clear by the obtained test error value.*

  *By introducing regularization we allow the model to stop passing exactly through the datapoints, where it has zero unregularized empirical risk, provided that it reflects a significant reduction in the norm of the weights. This trade-off with the norm, mediated via $\lambda$, reflects a decrease in the complexity of the hypothesis class. The trade-off of slightly increasing the empirical risk by lowering the norm of $\boldsymbol{w}$ results in a severe reduction in overfitting, compared to the unregularized solution, this is quite clear by the much smaller test error. However it is a bit unexpected that the empirical error of the regularized version is actually smaller than the unregularized version.*

  *The following figure, fig. 3.3 shows the resulting curve fits for the unregularized and the regularized analytical solutions, respectively, for 5 randomly selected datasets out of the 100 randomly generated datasets.*

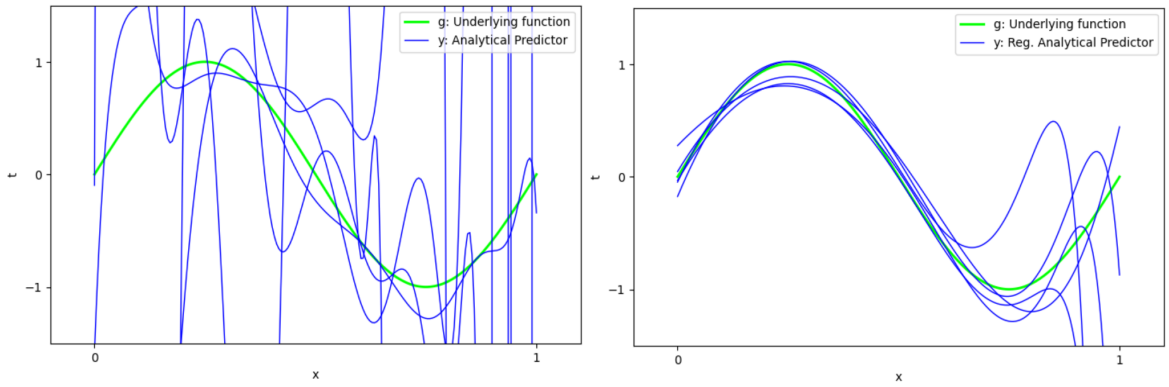


Figure 3.3: i) Curve fits using the unregularized ($\lambda = 0$) analytical solution; ii) Curve fits using the regularized ($\lambda = 0.01$) analytical solution.

- ***Full Gradient Descent Solution:*** *Here we minimize (3.38), using gradient descent on the entire dataset. The partial derivatives are given by:*

$$\frac{\partial \tilde{E}(\boldsymbol{w})}{\partial w_j} = \frac{1}{N}\sum_{n=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)x_n^j + \lambda w_j, \tag{3.39}$$

*and the update follows the following equation:*

$$\boldsymbol{w}_j^{(\tau+1)} = \boldsymbol{w}_j^{(\tau)} - \eta\frac{\partial \tilde{E}(\boldsymbol{w})}{\partial w_j}, \tag{3.40}$$

*where we used in particular $\eta = 0.01$ and $\lambda = 0$.*

*The following figure, fig. 3.4, shows the resulting curve fit for 5 of the 100 randomly generated datasets.*

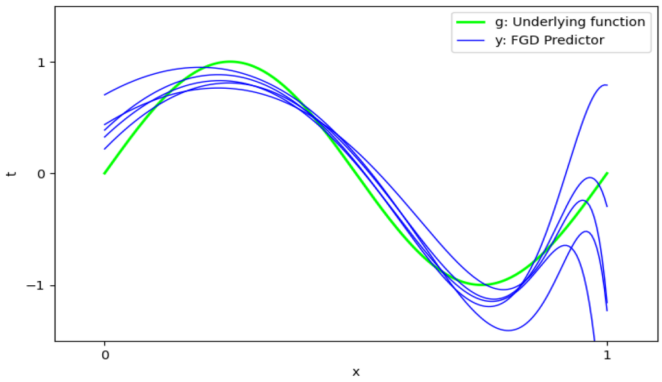


Figure 3.4: Curve fits using the FGD solution after 2500 epochs, with $\lambda = 0$ and $\eta = 0.01$.

- ***On-line and batch SGD Solution:*** *Here we approach the minimization of (3.38), in expectation, by using on-line stochastic gradient descent where, at each epoch, we update the weights with gradient descent by sampling a single datapoint uniformly from the dataset in the online case and sampling five datapoints with replacement in the batch case. The following figure, fig. 3.5, shows the resulting curve fit for 5 of the 100 randomly generated datasets.*

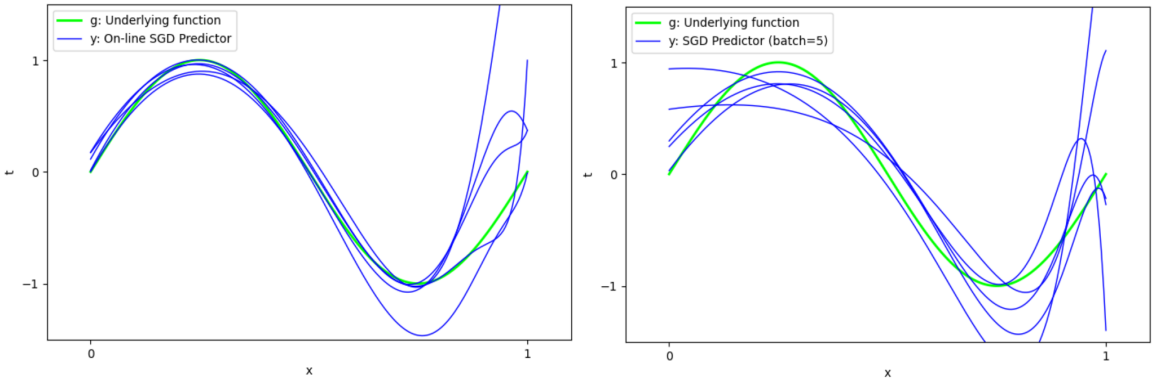


Figure 3.5: i) Curve fits using the on-line SGD solution; ii) Curve fits using the batch of five SGD solution; (both after 2500 epochs, with $\lambda = 0$ and $\eta = 0.01$).

*On stochastic methods we have a random estimation of the empirical risk based on a sample of the datapoints, which is only equal to the true empirical risk in expectation. Because of this, the updates on the weight vector might be noisy and do not always move in a direction which reduces the empirical risk over the whole dataset. This is however hard to see in the figure. Theoretically, because the error function we are considering is semi-convex, when the number of epochs goes to infinity gradient descent should take us to a weight vector which lives in the valley of zero empirical risk over the entire dataset. In particular, it would probably result in a vector of weights with larger norm than the unregularized analytical solution which, for this particular example, would result in an even worth fit (larger oscillations of the predictor due to larger values in the parameters).*

*So, why do gradient methods appear to provide a good fit while also apparently converging to a minimum, which contradicts the above? This seems to be the result of the vanishing gradient problem we mentioned on momentum approaches to gradient descent. The following figure, fig. 3.6, shows the evolution of the norm of the gradient as a function of the epoch for full gradient descent.*

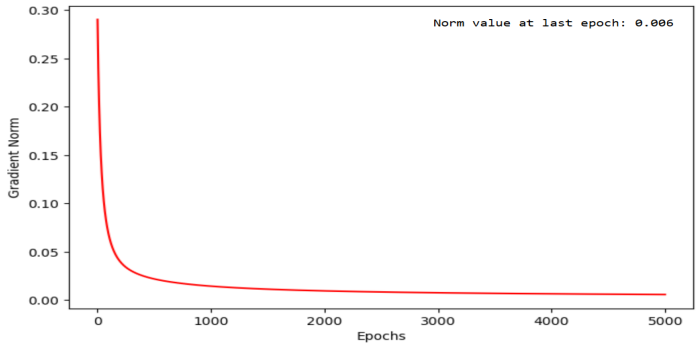


Figure 3.6: Norm of $\|\nabla_{\boldsymbol{w}}\tilde{E}(\boldsymbol{w})\|$ as a function of the training epoch.

*Since the gradient converges quickly to zero, the weight updates will barely change very quickly. As we saw back in the polynomial curve fitting example in Section 2.1.1 the overfitting issue is related to large values of the parameters. Because the parameters were randomly chosen from a Gaussian* $\mathcal{N}(0,1)$*, they will remain quite small which results in a regularization of sorts.*

*If we initialize the weights as* $\boldsymbol{w} \sim \mathcal{N}(0,100)$ *we see again the empirical loss stagnating, because the gradient again goes quickly to zero, and the fit is now much worse even when running for a large number of epochs (20000). Figure 3.7 shows the empirical and test loss on the left panel and the gradient norm, both as functions of the epoch.*

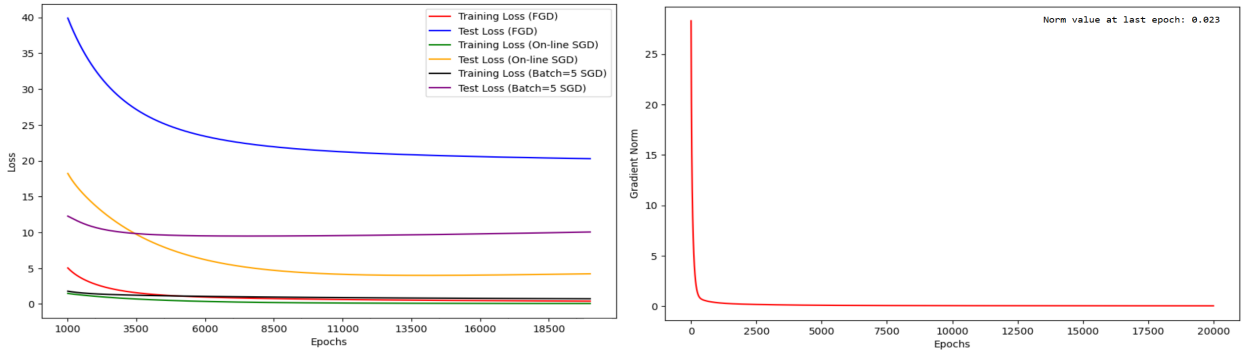


Figure 3.7: i) Training and test loss as a function of the epoch ; ii) Norm of $\|\nabla_{\boldsymbol{w}}\tilde{E}(\boldsymbol{w})\|$ as a function of the training epoch; (using $\boldsymbol{w} \sim \mathcal{N}(0,100)$).

*The next while figure, fig. 3.8 shows the resulting fits for all gradient methods.*

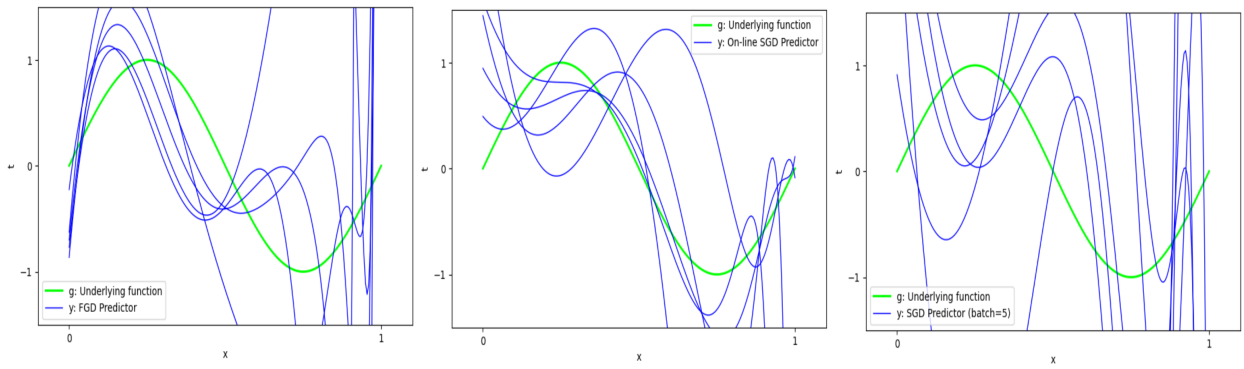


Figure 3.8: i) FGD; ii) On-line SGD; iii) batch-5 SGD; ($\eta = 0.01$, 20000 epochs)

*It would be interesting to experiment with momentum based methods and analyse how quickly gradient descent methods converge to one of the global minimums. This is left for a future iteration of this experiments.*

***Source code:*** *For the source code used to generate all plots see the file "polyfitsgd.ipynb".*

We now present the most common algorithm used in neural networks for applying gradient descent, which is called **backpropagation**.

**Backpropagation:** Although mathematically consisting simply in the use of the chain rule, backpropagation is in fact a computational efficient algorithm to apply gradient descent. We derive this algorithm now for a general neural network, i.e. with arbitrary number of layers $L$, neurons $\{N_\ell\}_{\ell=1}^{L}$ and activation function for each layer $h^{(\ell)}$.

Consider a general loss function:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell(\boldsymbol{w}, (x_n, t_n)). \tag{3.41}$$

Since the the derivative of a sum is the sum of the derivatives, we shall compute the gradient over a single input-output pair $(x_n, t_n)$, i.e. $\nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}, (x_n, t_n))$. Then, for full gradient descent or batch methods we could apply the results repeatedly on each input-output pair and sum. In what follows we will also be considering that the bias terms are implicit in the weights.

Writing $\ell(\boldsymbol{w}, (x_n, t_n))$ just as $\ell_n(\boldsymbol{w})$ we are interested in computing the derivative:

$$\frac{\partial \ell_n(\boldsymbol{w})}{\partial w_{ij}^{(\ell)}}, \tag{3.42}$$

where $i$ corresponds to neuron $i$ of layer $\ell$ and $j$ to the factor which multiplies $z_j^{(\ell-1)}(x_n)$. The dependence on $w_{ij}^{(\ell)}$ is through the sum:

$$y_i^{(\ell)} = \sum_{j=1}^{N_{\ell-1}} w_{ij}^{(\ell)} z_j^{(\ell-1)}(x_n), \tag{3.43}$$

and we have:

$$\frac{\partial \ell_n(\boldsymbol{w})}{\partial w_{ij}^{(\ell)}} = \frac{\partial \ell_n(\boldsymbol{w})}{y_i^{(\ell)}} \frac{\partial y_i^{(\ell)}}{\partial w_{ij}^{(\ell)}}, \tag{3.44}$$

where:

$$\frac{\partial y_i^{(\ell)}}{\partial w_{ij}^{(\ell)}} = z_j^{(\ell-1)}(x_n). \tag{3.45}$$

Defining $\delta_i^{(\ell)} := \frac{\partial \ell(\boldsymbol{w}, x_n)}{\partial y_i^{(\ell)}}$ we have then:

$$\frac{\partial \ell_n(\boldsymbol{w})}{\partial w_{ij}^{(\ell)}} = \delta_i^{(\ell)} z_j^{(\ell-1)}(x_n). \tag{3.46}$$

$$\delta_i^{(\ell)} = \sum_{k=1}^{N_{\ell+1}} \frac{\partial E_n}{\partial y_k^{(\ell+1)}} \frac{\partial y_k^{(\ell+1)}}{\partial y_i^{(\ell)}} := \sum_{k=1}^{N_{\ell+1}} \delta_k^{(\ell+1)} \frac{\partial y_k^{(\ell+1)}}{\partial y_i^{(\ell)}}, \tag{3.47}$$

where:

$$\frac{\partial y_k^{(\ell+1)}}{\partial y_i^{(\ell)}} = \frac{\partial y_k^{(\ell+1)}}{\partial z_i^{(\ell)}} \frac{\partial z_i^{(\ell)}}{\partial y_i^{(\ell)}} = w_{ki}^{(\ell+1)} h'^{(\ell)}\big(y_i^{(\ell)}\big). \tag{3.48}$$

Finally:

$$\delta_i^{(\ell)} = h'^{(\ell)}\big(y_i^{(\ell)}\big) \sum_{k=1}^{N_{\ell+1}} w_{ki}^{(\ell+1)} \delta_k^{(\ell+1)} \tag{3.49}$$

What this shows is that we can write the terms $\delta_i^{(\ell)}$ as linear combinations of the $\delta_k^{(\ell+1)}$ of the following layer. Therefore, we can start by computing the $\delta^{(L)}$ at the last layer and go backward to determine all other such terms.

The following remark, from Section VII.3 in [Str19], shows why backpropagation is computationally more efficient. Suppose we have $M_i$, $N \times N$ matrices, with $i = 1, ..., L$. The forward computation cost of $(((M_1 M_2)M3)...M_L)v$ requires $(L-1)N^3 + N^2$ multiplications while the backward calculation $M_1(M_2(...(M_L v)))$ requires $LN^2$ multiplications. We can write the neural network as a function of the form $y = f_L(f_{L-1}(...(f_1(w))))$, where $f_j : \mathbb{R}^{N_j - 1} \to \mathbb{R}^{N_j}$, $w \in \mathbb{R}^{N_0}$, the derivative becomes:

$$\frac{\partial y}{\partial w} = \frac{\partial f_L}{\partial f_{L-1}} ... \frac{\partial f_1}{\partial w} = J_{f_L}(w_L) J_{f_{L-1}}(w_{L-1}) ... J_{f_1}(w), \tag{3.50}$$

where $J_{f_j}(w_j)$ is the Jacobian matrix computed at $w_j = f_{j-1}(w_{j-1})$. This is a product of matrices which justifies the remark above.

The backpropagation algorithm, for a single input, can be written as follows:

- Choose an input $x_n$ and use forward propagation to compute the following:

$$\begin{aligned} y_i^{(\ell)}(x_n) &= \sum_{j=1}^{N_{(\ell-1)}} w_{ij} z_j^{(\ell-1)}(x_n), \\ z_i^{(\ell)} &= h^{(\ell)}(y_i^{(\ell)}(x_n)), \end{aligned} \tag{3.51}$$

  for all $\ell = 1, ..., L$, and $i = 1, ..., N_\ell$.

- Compute $\boldsymbol{\delta}^{(L)}$, i.e. at the last layer, and propagate it backwards via equation (3.49), to obtain all $\delta_i^{(\ell)}$, with $\ell = 1, ..., L$, and $i = 1, ..., N_\ell$.

- Compute all derivatives via equation (3.46).

Then, to optimize weights, we use (3.28). For full gradient descent, or batch methods, just add the contributions of each $x_n$:

$$\frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial w_{ij}} = \sum_{n=1}^{N} \frac{\partial \ell_n(\boldsymbol{w})}{\partial w_{ij}^{(\ell)}}. \tag{3.52}$$

The method of backpropagation also allows one to evaluate the Hessian of the error function, see Section 5.4 of [Bis06]. This is of special importance when considering Bayesian neural networks and the application of the Laplace approximation method.

We will now show examples of a FFFC neural network implemented from scratch in Python, for any number of layers and neurons, in the setting of regression. The attached code file also has a second implementation in classification, with the UCI digit dataset, which we do not show here.

**Example 3.2.2.** ***(FFFC neural network for regression)***
*To construct our dataset $D$ we generate 20 datapoints uniformly spaced, with $x \in [-2\pi, 2\pi]$, and label them according to the function $t(x) = sin(x)$. The following figure, fig. 3.9, shows a plot of the fit for a neural network with 1 hidden-layer and the following hyperparameters: 30 neurons, 100 neurons and 200 neurons with a learning rate of 0.01 using on-line SGD for 1000 epochs. The activation, in all cases, was the sigmoid function.*

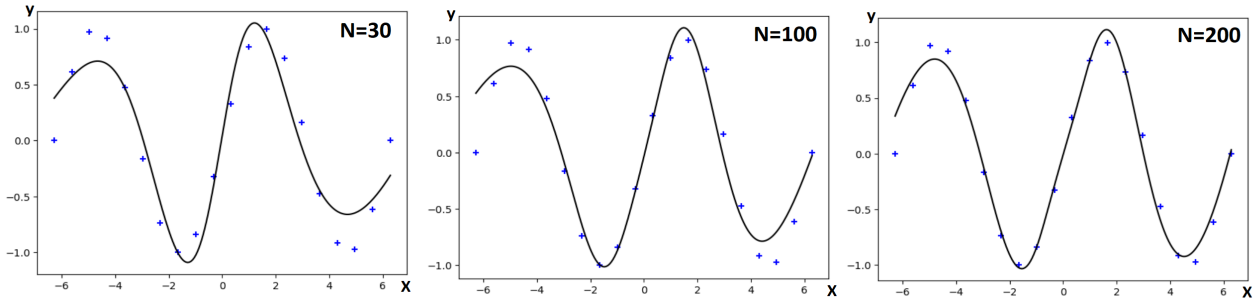


Figure 3.9: Fit with a neural network with a single hidden-layer for $N = 30, 100, 200$ neurons respectively with a learning rate of 0.01 and for 1000 epochs.

*The larger number of neurons seems to allow a better approximation, especially on the edges. The following figure, fig. 3.10, shows a couple of experiments and the hyperpameters that led to the best possible fit that we managed to find (third panel).*

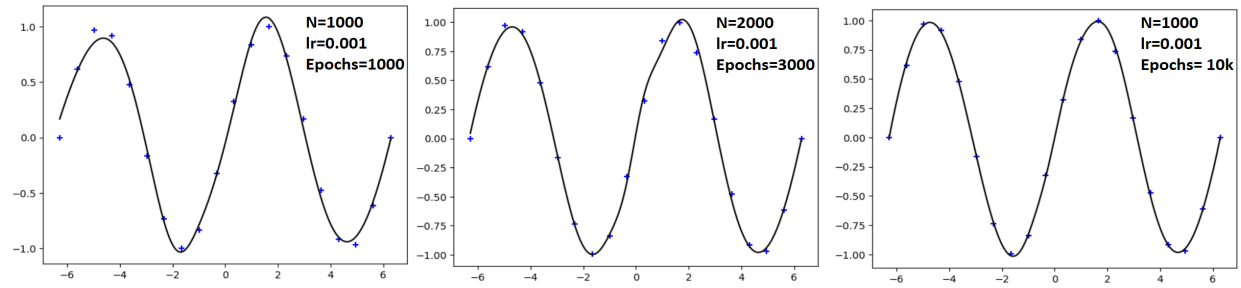


Figure 3.10: Fit with a neural network with a single hidden-layer with hyperparameters shown in the picture.

We now try a few examples beyond the single-hidden layer case, the chosen architectures were mostly random. The following figure, fig. 3.11, shows a fit with deep networks with the following hyperparameters and architectures (respectively): (40,30,20,10), $\eta = 0.01$, for $10K$ epochs; (500,500), $\eta = 0.01$, for $10K$ epochs and (40,30,20,10), $\eta = 0.01$, for $10K$ epochs.

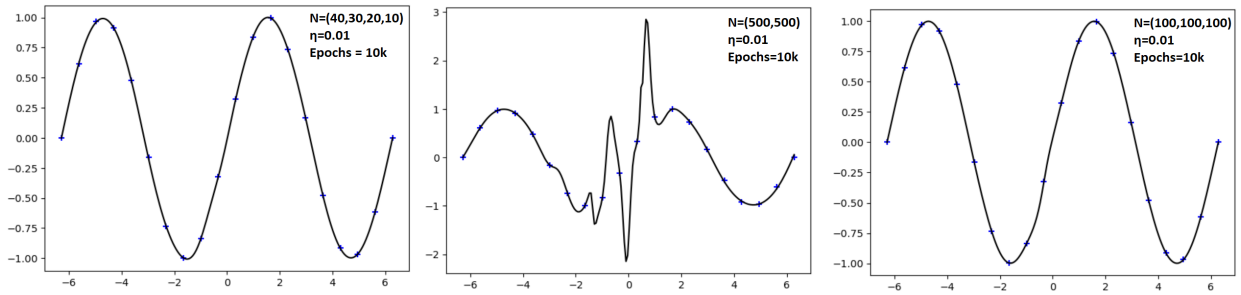


Figure 3.11: Fit using deep networks with hyperparameters and architectures as described above.

A curious observation here is that, despite the first two neural nets having approximately the same number of parameters, $P = 240k$ and $P = 250k$ respectively, the fits are clearly very different, with the latter showing a large overfit of the data. Note that in all cases we had convergence to a minimum, the loss remained constant for a large number of epochs. This might have been simply a question of converging to "good/bad" local minima or might have something to do with different expressive power depending on how the parameters are distributed over the layers. We would need to perform several, more careful, experiments in order to understand this. The last panel shows a great fit with $10^6$ parameters which, again, is quite surprising.

Our great conclusion here is that these results are very sensible to changes in learning rate, stoppage time and of course the neural network architecture. In any case, it is not hard to find a combination of hyperparameters and architecture which leads to good results, that is perhaps the surprising part. In any case, we would like to redo this code carefully and experiment further in the future.

**Source code:** For the source code with these implementations see the file "nnets-reg.ipynb". To see the implementation for the case of classification, using the UCI dataset see the file "nnets-class.ipynb".

## 3.3 Regularization in Neural Networks

The input/output dimension of a neural network is determined by the data but, the number of hidden layers and number of neurons at each hidden layer, are hyperparameters that need to be adjusted. Similar to the polynomial curve fitting example seen at the start of Chapter 2, where different values of the polynomial degree lead to better or worse fits of the data, here, different number of hidden layers and neurons lead to different generalization performance. Before, we dealt with this issue by introducing a regularization term, weight decay, of the form:

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}. \tag{3.53}$$

We could in principle apply the same solution here, where $\lambda$ would control the model complexity. However, this solution ends up favouring certain weights over others that lead to equivalent neural network realization functions. Let us consider an example.

**Example 3.3.1.** *Suppose we have a neural network with a single hidden-layer such that it has input variable $x \in \mathbb{R}^{N_0}$ and corresponding output $y(x) \in \mathbb{R}^{N_2}$. Each neuron in the hidden layer computes:*

$$z_j^{(1)}(\boldsymbol{w}^{(1)}, x) = h\bigg(\sum_{i=1}^{N_0} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\bigg), \;\; j = 1, ..., N_1. \tag{3.54}$$

*While at the output layer we have:*

$$y_k(x) = \sum_{j=1}^{N_1} w_{kj}^{(2)} z_j^{(1)}(\boldsymbol{w}^{(1)}, x) + w_{k0}^{(2)}. \tag{3.55}$$

*Suppose we apply the following linear transformations to the inputs and outputs:*

$$\begin{aligned} x_i \to \tilde{x}_i = a x_i + b, \;\; i = 1, ..., N_0, \\ y_k \to \tilde{y}_k = c y_k + d, \;\; k = 1, ..., N_2, \end{aligned} \tag{3.56}$$

*for some $a, b > 0$ and $b, d \in \mathbb{R}$. By transforming the weights via the following linear transformations:*

$$\begin{aligned} w_{ji}^{(1)} &\to \tilde{w}_{ji}^{(1)} = \frac{1}{a} w_{ji}^{(1)}, \\ w_{j0}^{(1)} &\to \tilde{w}_{j0}^{(1)} = w_{j0}^{(1)} - \frac{b}{a} \sum_{i=1}^{N_0} w_{ji}^{(1)}, \\ w_{kj}^{(2)} &\to \tilde{w}_{kj}^{(2)} = c w_{kj}^{(2)}, \\ w_{k0}^{(2)} &\to \tilde{w}_{k0}^{(2)} = c w_{k0}^{(2)} + d, \end{aligned} \tag{3.57}$$

*the neural network realization function is the same by the straightforward computations:*

$$\begin{aligned} \tilde{z}_j^{(1)}(\tilde{\boldsymbol{w}}^{(1)}, \tilde{x}) &= h\Big( \sum_{i=1}^{N_0} \tilde{w}_{ji}^{(1)} \tilde{x}_i + \tilde{w}_{j0}^{(1)} \Big) = h\Big( \sum_{i=1}^{N_0} \frac{1}{a} w_{ji}^{(1)} (a x_i + b) + w_{j0}^{(1)} - \frac{b}{a} \sum_{i=1}^{N_0} w_{ji}^{(1)} \Big) \\ &= h\Big( \sum_{i=1}^{N_0} w_{ji}^{(1)} x_i + \frac{b}{a} \sum_{i=1}^{N_0} w_{ji}^{(1)} - \frac{b}{a} \sum_{i=1}^{N_0} w_{ji}^{(1)} + w_{j0}^{(1)} \Big) = h\Big( \sum_{i=1}^{N_0} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \Big) \\ &= z_j^{(1)}(\boldsymbol{w}^{(1)}, x). \end{aligned} \tag{3.58}$$

$$\begin{aligned} \tilde{y}_k &= \sum_{j=1}^{N_1} \tilde{w}_{kj}^{(2)} z_j^{(1)}(\tilde{\boldsymbol{w}}^{(1)}, \tilde{x}) + \tilde{w}_{k0}^{(2)} = \sum_{j=1}^{N_1} \tilde{w}_{kj}^{(2)} z_j^{(1)}(\boldsymbol{w}^{(1)}, x) + \tilde{w}_{k0}^{(2)} \\ &= \sum_{j=1}^{N_1} c w_{kj}^{(2)} z_j^{(1)}(\boldsymbol{w}^{(1)}, x) + c w_{k0}^{(2)} + d = c \sum_{j=1}^{N_1} w_{kj}^{(2)} z_j^{(1)}(\boldsymbol{w}^{(1)}, x) + d = c y_k + d. \end{aligned} \tag{3.59}$$

*Thus, training a network with the original data $(x, y)$ or training with the transformed data $(\tilde{x}, \tilde{y})$ (or $(\tilde{x}, y), (x, \tilde{y})$) leads to equivalent neural networks which differ on the weights via the transformations in equation (3.57). However, the weight decay method above will favor the weights with lowest norm where all weights and biases are treated on an equal footing, i.e. we can not change $\lambda$ in a way that compensates distinct re-scalings in the elements of $\boldsymbol{w}$.*

A regularizer that is invariant to this linear transformations in the weights and biases, for the example above, is given by:

$$\frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2, \tag{3.60}$$

where $\mathcal{W}_i$ denotes the weights in the respective layer $i$, with biases not being part of the regularizer. By treating the weights of each layer independently each factor $\lambda_i$ can re-scale making the regularizer unchanged via a weight transformation. In the example above we would have $\lambda_1 \to a^2 \lambda_1$ and $\lambda_2 \to c^{-2} \lambda_2$.

In Section 2.1.4 we saw the equivalence of MAP with weight decayed least squares, i.e. the appearance of an implicit regularization. By similar computations one can see that this regularization term corresponds to a prior of the form:

$$p(\boldsymbol{w} | \lambda_1, \lambda_2) \propto \exp\Big( -\frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 - \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2 \Big). \tag{3.61}$$

Which can naturally be extended to any number of layers $L$ with corresponding weights $\mathcal{W}_\ell$ by:

$$p(\boldsymbol{w} | \boldsymbol{\lambda}) \propto \exp\Big( -\frac{\lambda_\ell}{2} \sum_{\ell} \|\boldsymbol{w}\|_\ell^2 \Big), \text{ where } \|\boldsymbol{w}\|_\ell^2 = \sum_{j \in \mathcal{W}_\ell} w_j^2. \tag{3.62}$$

**Early Stopping:** A different approach to regularization, in order to reduce the overfitting problem, is known as *early stopping*. Over the training set, the error is typically a non-increasing function of the parameters, as they are optimized over each epoch. In general, over unseen data, the test error reduces with each update up to a certain point and then starts to increase as overfitting sets in. Thus, stopping the optimization procedure near this minimum would be an effective way to reduce model complexity. However, because the test error oscillates between epochs, this needs to be implemented in a clever way.

**Invariances:** It is often useful to introduce prior knowledge in the network in the form of invariances that we know our predictor should preserve. For example, in digit recognition the classification of a particular digit should be invariant to scale and translation. If there is a large amount of data with these variations a neural network could pickup on these invariances by itself. However, if data is scarce, this may not work and we can instead "force" the network to recognize these invariances. Four possible approaches to this are the following:

- **Training set augmentation:** We apply the transformations that we wish our network to be invariant to on the training data. For example, having an image of a digit $x$ with a corresponding label $y$ we could re-scale and translate the image of digit $x$ and add it to the training set with the same label $y$.
- **Tangent propagation:** This method introduces a regularization term which penalizes different outputs for invariant input transformations. Let us see a specific example.

  Suppose we have a particular input in d dimensions, $x^{(n)} = (x_1^{(n)}, ..., x_d^{(n)})$. Any one-parameter continuous transformation of this input traces a 1-dimensional submanifold, a curve $\mathcal{M}$ in $\mathbb{R}^d$. Let the parameterization be given by $s(x^{(n)}, \xi)$ such that $s(x^{(n)}, 0) = x^{(n)}$. Then, the tangent vector at $x^{(n)}$ is given by:

$$\tau_n := \left.\frac{ds(x^{(n)}, \xi)}{d\xi}\right|_{\xi=0}. \tag{3.63}$$

  The change in the predictor $y_k$ due to the $\xi$ variation is given by:

$$\left.\frac{dy_k}{d\xi}\right|_{\xi=0} = \sum_{i=1}^{d} \frac{\partial y_k}{\partial x_i} \left.\frac{\partial x_i}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{d} J_{ki}\tau_i, \tag{3.64}$$

  where $J_{ki}$ is the corresponding entry in the Jacobian matrix of the function $y$. By defining the regularization function:

$$\Omega := \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \left( \sum_{i=1}^{d} J_{nki}\tau_{ni} \right)^2, \tag{3.65}$$

  we can define an error function:

$$\tilde{E} = E + \lambda\Omega, \tag{3.66}$$

  where $\Omega = 0$ precisely when the network realization function is invariant under local transformations around each input $x^{(n)}$, $n = 1, ..., N$. The hyperparameter $\lambda$ again determines the balance between fitting the training data and learning the invariance.

  If the transformation depends on $m$ parameters then $\mathcal{M}$ has dimension $m$ and equation (3.65) will have an extra sum for each parameter.

  It is possible to show that the technique of augmenting the training set is closely related to the approach here, see Section 5.5.5 of [Bis06].

- **Feature extraction:** By selecting from the data only features which are invariant under the required transformation forces the network to respect those invariances.
- **Network structure:** The invariances can be implicit in the structure of a neural network. A particular example of this are convolutional neural networks used for image recognition. In images, typically pixels which are closer together are more strongly correlated, imagine the color gradient over an arbitrary picture. By using techniques such as *local receptive fields*, *weight sharing* and *subsampling* one can make a neural network invariant to small distortions over the original inputs. For more details on this topics see Section 5.5.6 and 5.5.7 in [Bis06] or a more recent exposition in [GBC16].

**Dropout:** This method of regularization tries to limit the complexity of the realization function by randomly, at each training example, disconnecting neurons with some probability $p$. Intuitively this forces the parameters at each unit to perform as well as possible in the absence of other units which reduces the overall complexity by not relying on all parameters simultaneously. For a more in depth discussion, and references studying this formally, see Section 13.5.4 of [Mur22] .

## 3.4 Bayesian Neural Networks

In Chapter 2 we saw that the Bayesian approach, in particular the MAP paradigm, includes an implicit regularization term. As mentioned before, this could be particularly useful here due to the usually large number of adaptive parameters on a neural network, which makes it prone to overfitting. We shall now see how we can use a Bayesian setting for neural networks.

**Bayesian Neural Networks for Regression:** Suppose we want to predict $t \in \mathbb{R}$ given $\boldsymbol{x} \in \mathbb{R}^{N_0}$, where we assume a model distribution of the form:

$$p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1}), \tag{3.67}$$

where $y(\boldsymbol{x}, \boldsymbol{w})$ is the realization function of a certain neural network. We further consider a Gaussian prior over the weights given by:

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}). \tag{3.68}$$

Over a dataset $D = \{(\boldsymbol{x}_1, t_1), ..., (\boldsymbol{x}_N, t_N))\}$, the likelihood function is given by:

$$p(D|\boldsymbol{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(\boldsymbol{x}_i, \boldsymbol{w}), \beta^{-1}). \tag{3.69}$$

With posterior given by:

$$p(\boldsymbol{w}|D, \alpha, \beta) \propto p(D|\boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha). \tag{3.70}$$

Unlike the Bayesian linear regression case, because of the non-linear dependence of $y(\boldsymbol{x}, \boldsymbol{w})$ on $\boldsymbol{w}$, this will not be a Gaussian distribution i.e. there won't be a quadratic dependence over $\boldsymbol{w}$ in the exponential term. However, one can approximate the posterior to a Gaussian distribution via the Laplace approximation method introduced in Section 2.2.4.

We start by determining a maximum of the posterior, which may only be a local maximum. We can equivalently maximize the following:

$$\log p(\boldsymbol{w}|D) = -\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} - \frac{\beta}{2}\sum_{i=1}^{N}(y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i)^2 + \text{const}, \tag{3.71}$$

which, as mentioned, requires numerical optimization methods for complex functions such as those of a deep neural network. Let $\boldsymbol{w}_{MAP}$ be the obtained solution, i.e. a local maximum of the previous expression. The matrix $\boldsymbol{A}$ of the Laplace approximation method is given by:

$$\boldsymbol{A} = -\nabla\nabla \log p(\boldsymbol{w}|D, \alpha, \beta) = \alpha \boldsymbol{I} + \beta \boldsymbol{H}, \tag{3.72}$$

where $\boldsymbol{H}$ is the Hessian matrix of the sum-of-squares error with respect to $\boldsymbol{w}$ (approximation methods of the Hessian can be seen in Section 5.4 of [Bis06]). By the Laplace approximation method, the approximated Gaussian posterior is given by:

$$q(\boldsymbol{w}|D) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{MAP}, \boldsymbol{A}^{-1}). \tag{3.73}$$

As usual, the predictor distribution is given by:

$$p(t|\boldsymbol{x}, D) = \int p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) q(\boldsymbol{w}|D) d\boldsymbol{w}. \tag{3.74}$$

Again, due to the complexity of $y(\boldsymbol{x}, \boldsymbol{w})$ as a function of $\boldsymbol{w}$, the integral is still analytically intractable. In order to obtain an analytical solution we make the further assumption that the variance of the posterior $q(\boldsymbol{w}|D)$ is small compared to the space of possible values for $\boldsymbol{w}$, i.e. we assume the true values of $\boldsymbol{w}$ are somewhere close to $\boldsymbol{w}_{MAP}$. Then, Taylor expanding around $\boldsymbol{w}_{MAP}$ and keeping first order terms we get:

$$y(\boldsymbol{x}, \boldsymbol{w}) \simeq y(\boldsymbol{x}, \boldsymbol{w}_{MAP}) + \left( \nabla_{\boldsymbol{w}} y(\boldsymbol{x}, \boldsymbol{w}) \big|_{\boldsymbol{w}_{MAP}} \right)^T (\boldsymbol{w} - \boldsymbol{w}_{MAP}) := y(\boldsymbol{x}, \boldsymbol{w}_{MAP}) + \boldsymbol{g}^T (\boldsymbol{w} - \boldsymbol{w}_{MAP}). \tag{3.75}$$

Since, under this approximation, $y(\boldsymbol{x}, \boldsymbol{w})$ becomes linear in $\boldsymbol{w}$, then the exponential term in $p(t|\boldsymbol{x}, \boldsymbol{w}, \beta)$ is quadratic in $\boldsymbol{w}$ and the product $p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) q(\boldsymbol{w}|D)$ is Gaussian in $\boldsymbol{w}$ making the integral tractable. By Proposition 6.3.14, using the fact that both $q(\boldsymbol{w})$ and $p(t|\boldsymbol{w})$ are Gaussian, we have:

$$p(t|\boldsymbol{x}, D, \alpha, \beta) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}_{MAP}), \boldsymbol{g}^T \boldsymbol{A}^{-1} \boldsymbol{g} + \beta^{-1}). \tag{3.76}$$

This is the analog of (2.64). We have assumed that the parameters $\alpha, \beta$ are known in advance. We can however estimate these parameters from the data as well, see Section 5.7.2 in [Bis06]. For an extension of these results to classification problems see Section 5.7.3 of [Bis06].

This is ends our brief discussion of neural networks. There is much more to say in regards to the training mechanisms of neural networks and especially about different architectures, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Graph Neural Networks (GNNs), etc. Some references to look at in this regard are [SSBD14], for a more theoretical discussion on gradient descent and stochastic gradient descent and [GBC16], for different network architectures and observations with regards to implementation.

## 3.5 Summary

We started this chapter by giving a rigorous definition of a **feedforward fully-connected neural network** based on [BGKP21]. A neural network is essentially a function approximation method via basis functions which are themselves parametrized and where the predictor, called the **realization function** of the neural network has, in general, a highly non-linear dependence on the parameters. One major result, which we did not cover the proof of, is the following:

- **Universal approximation Theorem:** A shallow neural network, with a single hidden-layer, is enough to approximate any continuous function over a compact set provided that the **activation function** is not a polynomial. Despite this result however, there are several arguments favouring deeper networks.

The process of training a neural network follows the paradigm of ERM we saw in Chapter 1. This requires the specification of a loss function which depends on the learning task we are addressing. In particular, for regression problems we considered square loss while for classification we considered the cross-entropy loss. Due to the highly non-convex landscape of the empirical risk function, in order to find a minimum, which will hardly correspond to a global minimum in general, we used the idea of gradient descent.

- **Gradient descent:** Imagining the landscape of the empirical error, as a function of the parameters, because the gradient of a function at a point is a vector which points in the direction of largest increase of the function, by updating parameters in the opposite direction of the gradient we decrease the value of the error function. Applying this iteratively one expects to descend along the landscape of the empirical risk until a minimum is reached. Of course, this minimum will in general not be a global minimum. The idea is simple and to implement it in an efficient manner for neural networks we introduced the following algorithm.

- **Backpropagation:** This a specific method for implementing gradient descent in neural networks. In particular it is a computationally efficient method to obtain the partial derivatives of the loss function with respect to the parameters. The main takeaway is that we can define the partial derivatives of the loss function with respect to the parameters in a recursive manner that depends on a computation over the following layers. To do this we simply need to store the activations and preactivations during the **forward propagation** step and then, during the **backward step**, compute the derivatives of the loss with respect to previous layer preactivations, at each layer, which allows one to obtain the desired result.

- **Stochastic Gradient Descent (SGD):** The empirical risk, which we want to minimize, is defined as the sum of the loss function over the entire training data. In the full gradient descent method we use the entire training data when computing the gradient of the empirical risk with respect to the weights, this is as accurate as possible. However, if the dataset is very large this can be computationally demanding. The idea of stochastic gradient descent is simply to approximates the gradient of the empirical risk with respect to the weights by sampling a single datapoint at each update of the weights (a better estimation can be done by using a batch of datapoints). Because we are minimizing a different function compared to the empirical risk on the full dataset, SGD is not guaranteed to decrease the empirical risk at each update. In expectation however such a minimization is expected. For some theoretical results regarding this see Chapter 14 of [SSBD14].

Because neural networks, in general, depend on a large number of parameters they have huge flexibility when approximating functions. As we have seen in Chapter 1, with such a large hypothesis class we are prone to overffiting issues. However, just like for linear methods, we can apply regularization methods to neural networks. A few of the methods we discussed, some quite briefly, where:

- **Early stopping:** The test error (ignoring double descent phenomena) follows a sort of parabolic curve. At initialization this value is high and, as the training progresses, decreases up to a minimum. Onwards, while the training error keeps decreasing, the test error begins to increase as we start to overfit more and more. The idea of early stopping is to simply stop training the network further when it begins to show signs of overfitting.

- **Invariances:** A way to reduce overfitting is to ignore changes over input parameters that should not influence the outcome of the predictor. We talked about a few methods of doing this but a particularly interesting one is **tangent propagation**. Having defined transformations in input space, with respect to which we want our realization function to be invariant, we add a regularizer term which is minimized precisely when the derivative of the realization function goes to zero along these directions.

- **Dropout:** This controls the flexibility of the network by forcing it to randomly train with access to a limited number of neurons at each epoch, and thus a limited number of adaptable parameters. This results in more robust weight values which are not overly dependent on their inter-relations.

Finally, we introduced the concept of **Bayesian neural networks** in the context of regression problems. This is useful to consider due to what we have seen in Chapter 2, the implicit regularization in a Bayesian treatment. However, in order to obtain an analytically tractable result, several demanding approximations have to be made.

- **Bayesian neural networks:** Similarly to Bayesian logistic regression, due to the complexity of the realization function, the likelihood function will be equally complex, this impedes an analytical consideration of the posterior distribution. Just like before we use the Laplace approximation which requires numerical methods to both estimate a maximum of the prior and the Hessian. Having found these, the obtained predictor given by the average over weights is still intractable and requires a further assumption that the posterior is sharply concentrated on the mean value. These are all sacrifices for the sake of obtaining an analytical solution which we get in the end in the form of a Gaussian distribution. Surely, implementing these methods successfully requires robust numerical solvers so that we do not require such drastic approximations.

# Chapter 4

# Kernel Methods

This final chapter introduces **kernel methods**. So far, the methods we have used relied on learning parameters, either associated to fixed basis functions or to adaptable basis functions such as in the case of neural networks. After the learning process, the data is discarded and we use the learnt parameters to construct a predictor under our model assumption. Kernel methods on the other hand use the training data to make predictions, by using a similarity measure which is captured in the **kernel function**. We start precisely by defining what a kernel function is and motivate its introduction, which will be done via the "**kernel trick**" on a particularly simple example and a dual formulation of the least squares problem. We will then see what property precisely characterizes a kernel function and how we can construct them.

Next we present the most familiar example of a kernel method, a sort of generalization of the perceptron algorithm known as the **Support Vector Machine** (SVM). It can be formulated as an optimization problem which determines a hyperplane separating the training data, or transformations of it, but doing it in a way that maximizes the margin to this data, unlike the perceptron. We will further see that the predictor associated to the (general) SVM problem is completely specified by inner products of the training data with the new point we want to evaluate, or again a transformation of them, which allows to completely specify the predictor via a kernel function.

We introduce another kernel method, known as a **Gaussian process** which, unlike SVM, defines a probability distribution over the target. This can be seen as a generalization of a Gaussian distribution to infinite dimensions where the covariance is defined via a kernel function. We will see as well how Gaussian processes relate to neural networks in the next section.

We end this seminar by introducing a special kernel, the **Neural Tangent Kernel** (NTK), which connects neural networks to kernel methods in the limit of infinite width, i.e. when the number of neurons goes to infinity at each layer. In particular, in this limit, one can specify the realization function of the neural network via the NTK by solving a differential equation resulting from gradient descent.

## 4.1 Kernel Trick

Instead of learning parameters $\boldsymbol{w}$ to construct a predictor $y(\boldsymbol{x}; \boldsymbol{w})$ and then discarding the data, kernel methods store a subset of the training data and use learned coefficients over pairs to make predictions. This is encapsulated in the notion of a kernel function which, over the reals, has the following definition (from [SC08]):

**Definition 4.1.1.** ***(Kernel function)***
*Let $X$ be a nonempty set. A function $\kappa : X \times X \to \mathbb{R}$ is called a **kernel** on $X$ if there exists a Hilbert*

*space $F$ and a map $\phi : X \to F$ such that, for all $x, x' \in X$ we have:*

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle_F . \tag{4.1}$$

*The map $\phi$ is called a **feature map** and $F$ is called a **feature space** of $\kappa$.*

**Remark 4.1.1.** *We have a map $\phi$ and a feature space $F$ of $\kappa$, as opposed to the map or the feature space since the correspondence is not 1-1, i.e. a kernel $\kappa$ does not specify a unique feature map and feature space. Following the example in Section 4.1 of [SC08], consider $X = \mathbb{R}$ and $\kappa(x, x') = xx' \in \mathbb{R}$. Then, $\kappa$ is a kernel since we can consider $\phi = id_{\mathbb{R}}$ and $H = \mathbb{R}$. However, if we consider instead the map $\phi : X \to \mathbb{R}^2$ with $x \mapsto \phi(x) = (x/\sqrt{2}, x/\sqrt{2})$, for all $x, x' \in \mathbb{R}$ we have:*

$$\langle \phi(x), \phi(x') \rangle = \left( \frac{x}{\sqrt{2}}, \frac{x}{\sqrt{2}} \right) \cdot \left( \frac{x'}{\sqrt{2}}, \frac{x'}{\sqrt{2}} \right) = \frac{xx'}{2} + \frac{xx'}{2} = xx' = \kappa(x, x'). \tag{4.2}$$

Let us now motivate the introduction of the kernel function, in connection to linear models we have seen, in two steps. First we motivate, via a simple example following 16.1 in [SSBD14], how a transformation of our data via a feature map to a higher dimensional space might be useful in constructing predictors. Then, in the context of least squares, we will see how the predictor can be entirely formulated by inner products within the feature space which, by connection with the definition above, can then be specified by a kernel function.

### 4.1.1 Motivating the Kernel Formulation

Let $\mathcal{X} = \mathbb{R}$ and suppose we have the following dataset:

$$D = \{(-3, 1), (-2, -1), (-1, -1), (0, -1), (1, -1), (2, -1), (3, 1)\},$$

where the underlying labeling function assigns $+1$ whenever $|x| > 2$ and 0 otherwise. It is not possible to separate this dataset with a halfspace. However, we can consider a (feature) map from the original input space into a space of higher dimension where the data is separable by a halfspace. See the following figure, fig. 4.1.

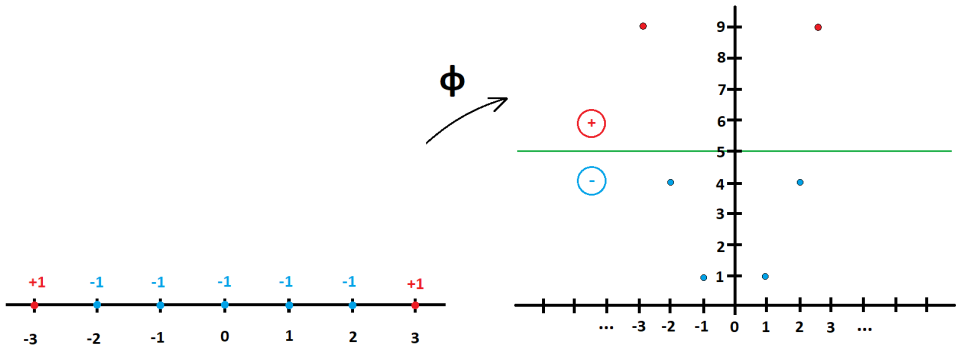


Figure 4.1: i) Dataset in $\mathbb{R}$; ii) Data transformation via feature map $\phi$;

Here we considered the feature map $\phi : \mathbb{R} \mapsto \mathbb{R}^2$ given by $\phi(x) = (x, x^2)$ where the resulting halfspace defines the predictor $h(x) = \text{sign}(\langle \boldsymbol{w}, \phi(x) \rangle - b))$ with $\boldsymbol{w} = (0, 1)$ and $b = 5$ (the separating region is marked in green on the figure above).

This motivates the following general procedure:

1. From the input domain $\mathcal{X}$ find a feature map $\phi : \mathcal{X} \mapsto F$, where $F$ is called the feature space (a Hilbert space in general as seen in the definition of a kernel function).

2. Transform the original dataset:

$$D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N \mapsto \tilde{D} = \{(\phi(\boldsymbol{x}_i), y_i)\}_{i=1}^N \tag{4.3}$$

3. Find a predictor $h$ within the feature space, such as the separating hyperplane in the example above, where, for any new test point $\boldsymbol{x}$ we compute $\hat{y} = h(\phi(\boldsymbol{x}))$.

If the feature space is very high-dimensional, this represents an increase both in the complexity of the hypothesis class and in the computational cost of performing computations in this space. The great benefit of the kernel formulation is that the linear methods we have seen so far can be formulated via inner products in this feature space. This allows one to bypass the explicit construction of the feature map by simply specifying a kernel function which computes these inner-products. Let us see this in detail for a linear regression model, such as those considered in Section 2.1, following Chapter 6 of [Bis06].

Consider a linear regression model, in the parameters $\boldsymbol{w}$ with possible non-linear transformations $\boldsymbol{\phi}$ of the features, where we obtain the following regularized sum-of-squares cost function:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n)^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}. \tag{4.4}$$

with $\lambda \geq 0$. Computing $\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = 0$ we obtain:

$$\boldsymbol{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\right)\boldsymbol{\phi}(\boldsymbol{x_n}) = \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\boldsymbol{x_n}) = \boldsymbol{\Phi}^T\boldsymbol{a} \tag{4.5}$$

where $\boldsymbol{\Phi}$ is the design matrix with $n^{th}$ row equal to $\phi(x_n)^T = [\phi_0(x_n) \dots \phi_{M-1}(x_n)]$. The vector $\boldsymbol{a} = (a_1, ..., a_N)$ is defined by:

$$a_n = -\frac{1}{\lambda}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\right), \;\; n = 1, ..., N. \tag{4.6}$$

The dependency on $\boldsymbol{w}$ can be removed by using eq. (4.5), which shall be done in a moment. Developing the square in eq. (4.4) we obtain:

$$\begin{aligned} E(\boldsymbol{w}) &= \frac{1}{2}\sum_{n=1}^{N}\left((\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}))^2 - 2t_n\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n}) + t_n^2\right) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \\ &= \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x_n})\boldsymbol{\phi}(\boldsymbol{x_n})^T\boldsymbol{w}\right) - \boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \\ &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}. \end{aligned} \tag{4.7}$$

with $\boldsymbol{t} = (t_1, ..., t_N)^T$. Writing the cost as a function of $\boldsymbol{a}$ via eq. (4.5) we obtain:

$$E(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}. \tag{4.8}$$

We define the Gram matrix $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ with entries given by:

$$K_{ij} = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) =: k(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{4.9}$$

where we introduce the kernel function $k$, defined via the inner product of the feature maps. In terms of $\boldsymbol{K}$ we can write (4.8) as:

$$E(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}. \tag{4.10}$$

From eqs. (4.5) and (4.6) we have:

$$\boldsymbol{a} = \frac{1}{\lambda}(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}) \Leftrightarrow \boldsymbol{a} = \big(\boldsymbol{K} + \lambda \boldsymbol{I}_N\big)^{-1}\boldsymbol{t}. \tag{4.11}$$

The predictor, which was given before as a function of the parameters $\boldsymbol{w}$ by:

$$y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T\phi(\boldsymbol{x}), \tag{4.12}$$

can now be recast, as a function of the kernel $k$, as:

$$y(\boldsymbol{x}; k) = \boldsymbol{w}^T\phi(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{\Phi}\phi(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{k}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T\big(\boldsymbol{K} + \lambda \boldsymbol{I}_N\big)^{-1}\boldsymbol{t}. \tag{4.13}$$

where the vector $\boldsymbol{k}(\boldsymbol{x})$ has entries given by the kernel function $k_n(\boldsymbol{x}) = k(\boldsymbol{x}_n, \boldsymbol{x})$ for $n = 1, ..., N$.

Through this dualization process the problem can be formulated entirely via the kernel function $k$. Our first approach was to compute the vector $\boldsymbol{w}$ explicitly and obtain a predictor given by $y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T\phi(\boldsymbol{x})$. The dual formulation computes the predictor $y(\boldsymbol{x}; k) = \boldsymbol{k}(\boldsymbol{x})^T\big(\boldsymbol{K} + \lambda \boldsymbol{I}_N\big)^{-1}\boldsymbol{t}$, which depends on a linear combination of the kernel evaluated at the training data, $\boldsymbol{k}(\boldsymbol{x}) = [k(\boldsymbol{x}_1, \boldsymbol{x}) \, ... \, k(\boldsymbol{x}_N, \boldsymbol{x})]^T$.

The benefit of introducing the kernel function stems from the fact that the entire learning problem can be specified from the point of view of the kernel function, bypassing the need of explicitly specifying the non-linear transformations $\phi$ to compute the inner-products. Let us look at a simple example (Example 2.9 of [STC04]).

**Example 4.1.1.** *Let $X \subseteq \mathbb{R}^2$ and consider the feature map $\phi : X \to \mathbb{R}^3$ given by:*

$$\phi(x) = \phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2). \tag{4.14}$$

*We have then:*

$$\big\langle \phi(x), \phi(x') \big\rangle_{\mathbb{R}^3} = (x_1{}^2, x_2{}^2, \sqrt{2}x_1x_2) \cdot (x'^2_1, x'^2_2, \sqrt{2}x'_1x'_2) = (x_1x'_1 + x_2x'_2)^2 = \big\langle x, x' \big\rangle^2_{\mathbb{R}^2}. \tag{4.15}$$

*Thus, $\kappa(x, x') = \langle x, x' \rangle^2$ is a kernel function and we can compute the inner products over the feature space $F = \mathbb{R}^3$ without explicitly computing the coordinates obtained from the feature map $\phi$.*

*This example can be extended to the feature space $F = \mathbb{R}^{n^2}$, with the same kernel, which shows again that the correspondence is not 1-1, see Example 2.10 in [STC04].*

Because of this, it is useful to understand what property precisely characterizes a kernel function when we are not interested in explicitly constructing a feature map.

### 4.1.2 Characterizing Kernel Functions

As mentioned in the beginning of the section, if we consider the nonlinear transformations $\boldsymbol{\phi}$ over feature space we can construct the kernel function by using the definition provided in the beginning of the section. However, we would like to avoid this dependency and construct valid kernel functions directly.

A full characterization of kernel functions can be obtained via the finitely positive semi-definite property. We state the next results following Section 3 of [STC04] and Section 3.2 of [SC08].

**Definition 4.1.2.** ***Finitely positive semi-definite function***
*Let $X$ be a set and consider the function $\kappa : X \times X \to \mathbb{R}$. We say $\kappa$ satisfies the finitely positive semi-definite property if $\kappa$ is a symmetric function for which the matrices formed by restriction to any*

*finite subset of the space $X$ are positive semi-definite. That is, $\forall n \in \mathbb{N}$ given a subset $\{x_1, ..., x_n\} \subseteq X$ and $\boldsymbol{v} \in \mathbb{R}^n$ we have:*

$$\boldsymbol{v}^T K \boldsymbol{v} = \sum_{i=1}^{n} \sum_{j=1}^{n} v_i v_j k(x_i, x_j) \geq 0 \tag{4.16}$$

*where $K_{ij} := k(x_i, x_j)$.*

Is is possible to show that, assuming only the finitely positive semi-definite property on $\kappa$, one can construct the full feature space. This is stated in the following theorem:

**Theorem 4.1.1.** *Let $X$ be a set. A function $\kappa : X \times X \to \mathbb{R}$ is a kernel, if and only if it satisfies the finitely positive semi-definite property (Theorem 4.16 of [SC08] and Theorem 3.11 of [STC04]).*

*Proof.*
( $\Longrightarrow$ ) Since $\kappa$ is a kernel it is defined via the inner-product on $F$ and thus it is symmetric. Consider a finite subset $S \subseteq X$, with $S = \{x_1, x_2, ..., x_n\}$ for some $n \in \mathbb{N}$. The matrix $K$ defined by the restriction of $\kappa$ to this subset is given by:

$$K_{ij} = \kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_F \tag{4.17}$$

for $i, j = 1, ..., n$. The linear map $K : \mathbb{R}^n \to \mathbb{R}^n$ satisfies the following, given any $v \in \mathbb{R}^n$ we have:

$$\begin{aligned} v^T K v &= \sum_i \sum_j v_i v_j K_{ij} = \sum_i \sum_j v_i v_j \langle \phi(x_i), \phi(x_j) \rangle_F = \sum_i \sum_j \langle v_i \phi(x_i), v_j \phi(x_j) \rangle_F = \\ &= \left\langle \sum_i v_i \phi(x_i), \sum_j v_j \phi(x_j) \right\rangle_F = \left\| \sum_i v_i \phi(x_i) \right\|_F^2 \geq 0. \end{aligned} \tag{4.18}$$

( $\Longleftarrow$ ) Consider the following function space:

$$\mathcal{F} = \left\{ \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot) \, : \, n \in \mathbb{N}, \, x_i \in X, \, \alpha_i \in \mathbb{R}, \, i = 1, ..., n \right\}. \tag{4.19}$$

The space $\mathcal{F}$ is a vector space via the usual definition of addition and scalar multiplication for functions. Let $f, g \in \mathcal{F}$ with $f = \sum_{i=1}^{l} \alpha_i \kappa(x_i, \cdot)$, $g = \sum_{j=1}^{m} \beta_j \kappa(z_j, \cdot)$ and define:

$$\langle f, g \rangle := \sum_{i=1}^{l} \sum_{j=1}^{m} \alpha_i \beta_j \kappa(x_i, z_j) = \sum_{i=1}^{l} \alpha_i g(x_i) = \sum_{j=1}^{m} \beta_j f(z_j), \tag{4.20}$$

where we have used the symmetry of $\kappa$ and the definitions of $f$ and $g$. Since the definition is independent of the representation of $f$ and $g$ this is well defined, furthermore $\langle \cdot, \cdot \rangle$ is a real-valued, symmetric and bilinear map on $\mathcal{F}$. It also satisfies:

$$\langle f, f \rangle \geq 0, \, \forall f \in \mathcal{F}. \tag{4.21}$$

This follows from the fact that $\kappa$ is finitely positive semi-definite. Using the function $f$ defined above we have:

$$\langle f, f \rangle = \sum_{i,j=1}^{l} \alpha_i \alpha_j \kappa(x_i, x_j) = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \geq 0, \tag{4.22}$$

where $\boldsymbol{\alpha} = [\alpha_1 \, ... \, \alpha_l]^T$ and $\boldsymbol{K}$ is the kernel matrix defined on the subset $\{x_1, ..., x_l\} \subseteq X$.

Suppose that $f := \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot)$ such that $\langle f, f \rangle = 0$. Then, $\forall x \in X$ we have:

$$|f(x)|^2 = \left| \sum_{i=1}^{n} \alpha_i \kappa(x_i, x) \right|^2 = |\langle f, k(x, \cdot) \rangle|^2 \leq \langle k(x, \cdot), k(x, \cdot) \rangle = \langle f, f \rangle = 0 \implies f = 0 \tag{4.23}$$

And thus we conclude that $\langle \cdot, \cdot \rangle$ is an inner product on $\mathcal{F}$. The definition of the inner-product $\langle \cdot, \cdot \rangle$ above provides another property, known as the *reproducing property* of the kernel. Given $x \in X$, $\kappa(x, \cdot) \in \mathcal{F}$ and $f$ defined as above, we have:

$$\langle f, \kappa(x, \cdot) \rangle = \sum_{i=1}^{l} \alpha_i \kappa(x_i, x) = f(x). \tag{4.24}$$

We have that $(\mathcal{F}, \langle \cdot, \cdot \rangle)$ is an inner-product space, to show that it is an Hilbert space we still need to verify that it is complete. Let $x \in X$ and consider a Cauchy sequence $(f_n)_{n=1}^{\infty}$. By the reproducing property and Cauchy-Schwartz inequality we have:

$$(f_n(x) - f_m(x))^2 = \langle f_n - f_m, \kappa(x, \cdot) \rangle^2 \leq \|f_n - f_m\|^2 \kappa(x, x). \tag{4.25}$$

Since $f_n(x)$ is a bounded Cauchy sequence of real numbers it has a limit. Defining the function $h$ on $X$ by:

$$h(x) = \lim_{n \to \infty} f_n(x), \tag{4.26}$$

and including all such limiting functions in $\mathcal{F}$, i.e. taking the topological closure of $\mathcal{F}$, one obtains the Hilbert space $(F_\kappa, \langle \cdot, \cdot \rangle)$.

The feature map, $\phi : X \to F_\kappa$, is given by:

$$\phi(x) := \kappa_x = \kappa(x, \cdot). \tag{4.27}$$

■

Because the kernel function will define our predictor it is important to have different kernels, with different degrees of complexity, in order to accurately represent our data. We now show two propositions which allow precisely to construct more complex kernels by applying simple transformations to simpler kernel functions.

### 4.1.3 Constructing Kernels

We saw that specifying the kernel function is sufficient via the dualization process. We now want to construct kernel functions directly, without the need of providing the feature map $\boldsymbol{\phi}$. By Theorem 4.1.1, $k(\boldsymbol{x}, \boldsymbol{x}')$ only has to satisfy the finitely positive semidefinite property. However, it is useful to see what sort of transformations one can apply to kernels that still result in valid kernel functions. This permits the construction of more complicated kernels out of simpler ones.

**Proposition 4.1.2.** ***(Closure properties)***
*Let $k_1$ and $k_2$ be kernels over $X \times X$, $X \subseteq \mathbb{R}^n$, $a \in \mathbb{R}_+$, $f : X \mapsto \mathbb{R}$, $\phi : X \mapsto \mathbb{R}^N$, $k_3$ a kernel over $\mathbb{R}^N \times \mathbb{R}^N$ and $\boldsymbol{B}_{n \times n}$ a symmetric positive semi-definite matrix. Then, the following functions are also kernel functions:*

$$k(x, x') = k_1(x, x') + k_2(x, x'), \tag{4.28}$$
$$k(x, x') = ak1(x, x'), \tag{4.29}$$
$$k(x, x') = k_1(x, x') k_2(x, x'), \tag{4.30}$$
$$k(x, x') = f(x) f(x'), \tag{4.31}$$
$$k(x, x') = k_3(\phi(x), \phi(x')), \tag{4.32}$$
$$k(x, x') = x^T \boldsymbol{B} x'. \tag{4.33}$$

*Proof.* See Proposition 3.22 of [STC04]). ■

**Proposition 4.1.3.** ***(Composition properties)***
*Let $k_1$ be a kernel over $X \times X$ with $p(x)$ a polynomial with positive coefficients. Then, the following functions are also kernel functions:*

$$k(x, x') = p\big(k_1(x, x')\big), \tag{4.34}$$
$$k(x, x') = \exp\big(k1(x, x')\big), \tag{4.35}$$
$$k(x, x') = \exp\bigg(-\frac{\|x - x'\|_2^2}{2\sigma^2}\bigg). \tag{4.36}$$

*Proof.* See Proposition 3.24 of [STC04]). ■

**Kernel functions over sets:** Kernel functions can also be defined over arbitrary sets (Exercise 6.12 of [Bis06]). Suppose we have a set $D$, consider its power set $2^D$. Let $A, B \subseteq D$. We consider a feature space of dimension $2^{|D|}$, i.e. the number of elements in $2^D$ and we define the map $\phi : D \mapsto \{0, 1\}^{2^{|D|}}$ as:

$$\phi_U(A) = \begin{cases} 1, \text{ if } U \subseteq A, \\ 0, \text{ otherwise,} \end{cases} \tag{4.37}$$

for arbitrary $U \subseteq D$. That is, $\phi(A)$ is a vector of dimension $2^{|D|}$ with 1's on the subsets which are contained in $A$ and 0 otherwise. Then, we can define the inner-product:

$$k(A, B) := \langle \phi(A), \phi(B) \rangle = \sum_{U \in 2^D} \phi_U(A)\phi_U(B) = 2^{|A \cap B|}, \tag{4.38}$$

since this is equal to the number of all subsets $U$ which are contained in both $A$ and $B$ and thus is equal to the number of subsets generated by the common elements in $A$ and $B$.
We now introduce SVMs as an optimization problem and, via dualization, show how it is in fact a kernel method.

## 4.2 Support Vector Machines (SVMs)

Given a linearly separable dataset, the perceptron algorithm we saw before is guaranteed to find a separating hyperplane. However, this separating hyperplane is not unique, it depends on the random initialization of the weights. Furthermore, in general, it will not find a separating hyperplane that gives the best possible margin, which is something that we would intuitively want. This is precisely the goal of the support vector machine which we will explore in this section. Let us begin by rigorously defining the concept of a margin and stating in a proposition a result we already mentioned in Section 2.2.1 .

**Definition 4.2.1.** ***(Margin)***
*Given an input space $\mathcal{X} = \mathbb{R}^d$ and a training dataset $D|_{\mathcal{X}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ we define the margin of a hyperplane in $\mathbb{R}^d$ with respect to $D$, defined by parameters $(\boldsymbol{w}, b) \in \mathbb{R}^d \times \mathbb{R}$, to be the solution of:*

$$\begin{aligned} &\min_{\boldsymbol{x} \in D|_{\mathcal{X}},\, \boldsymbol{v} \in \mathbb{R}^d} \|\boldsymbol{x} - \boldsymbol{v}\| \\ &s.t.\ \langle \boldsymbol{w}, \boldsymbol{v} \rangle + b = 0. \end{aligned} \tag{4.39}$$

The margin can be computed via the following proposition.

**Proposition 4.2.1.** ***(Distance between a point and a hyperplane)*** *Let $\boldsymbol{x} \in \mathbb{R}^n$, for any $n \in \mathbb{N}$ and consider the hyperplane in $\mathbb{R}^n$ defined by $\langle \boldsymbol{w}, \boldsymbol{v} \rangle + b = 0$ where $\boldsymbol{w} \in \mathbb{R}^n$ such that $\|\boldsymbol{w}\| = 1$ and $b \in \mathbb{R}$. The distance between $\boldsymbol{x}$ and the hyperplane is given by:*

$$|\boldsymbol{w}^T \boldsymbol{x} + b| = |\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b|. \tag{4.40}$$

*Proof.* The set of distances between the point $\boldsymbol{x}$ and any point $\boldsymbol{v}$ in the hyperplane is given by:

$$\{\|\boldsymbol{x} - \boldsymbol{v}\| \, : \, \langle \boldsymbol{w}, \boldsymbol{v} \rangle + b = 0\}, \tag{4.41}$$

and we are looking for the minimum within such a set. We have the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{v} \in \mathbb{R}^n} \|\boldsymbol{x} - \boldsymbol{v}\|, \\ &\text{s.t. } \langle \boldsymbol{w}, \boldsymbol{v} \rangle + b = 0. \end{aligned} \tag{4.42}$$

Take $\boldsymbol{v} = \boldsymbol{x} - (\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\boldsymbol{w}$, this particular choice comes from the geometric considerations we saw in Section 2.2, then $\boldsymbol{v}$ belongs to the hyperplane since:

$$\langle \boldsymbol{w}, \boldsymbol{v} \rangle + b = \langle \boldsymbol{w}, \boldsymbol{x} \rangle - (\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\|\boldsymbol{w}\|^2 + b = 0. \tag{4.43}$$

Thus, the minimum distance will be at most:

$$\|\boldsymbol{x} - \boldsymbol{v}\| = \|(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\boldsymbol{w}\| = |\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \, | \|\boldsymbol{w}\| = |\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \, |, \tag{4.44}$$

since $\|\boldsymbol{w}\| = 1$. Let $\boldsymbol{u}$ be any other point on the hyperplane, i.e. $\langle \boldsymbol{w}, \boldsymbol{u} \rangle + b = 0$. We have:

$$\begin{aligned} \|\boldsymbol{x} - \boldsymbol{u}\|^2 &= \|\boldsymbol{x} - \boldsymbol{v} + \boldsymbol{v} - \boldsymbol{u}\|^2 = \\ &= \|\boldsymbol{x} - \boldsymbol{v}\|^2 + \|\boldsymbol{x} - \boldsymbol{u}\|^2 + 2\langle \boldsymbol{x} - \boldsymbol{v}, \boldsymbol{v} - \boldsymbol{u} \rangle \\ &\geq \|\boldsymbol{x} - \boldsymbol{v}\|^2 + 2\langle \boldsymbol{x} - \boldsymbol{v}, \boldsymbol{v} - \boldsymbol{u} \rangle \\ &= \|\boldsymbol{x} - \boldsymbol{v}\|^2 + 2(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\langle \boldsymbol{w}, \boldsymbol{v} - \boldsymbol{u} \rangle \\ &= \|\boldsymbol{x} - \boldsymbol{v}\|^2 \end{aligned} \tag{4.45}$$

Where we use $\boldsymbol{x} - \boldsymbol{v} = (\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\boldsymbol{w}$, by the choice of $\boldsymbol{v}$, and, because both $\boldsymbol{v}$ and $\boldsymbol{u}$ belong to the hyperplane, we have $\langle \boldsymbol{w}, \boldsymbol{v} - \boldsymbol{u} \rangle = -b + b = 0$.

We conclude then that the distance between $\boldsymbol{x}$ and the hyperplane must be given by $|\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b|$. ■

The first algorithm we present, called **Hard-SVM**, tries to find precisely a hyperplane which separates the training data $D = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with the largest possible margin by solving the following optimization problem:

$$\begin{aligned} &\arg\max_{(\boldsymbol{w}, b) : \|\boldsymbol{w}\| = 1} \min_{\boldsymbol{x}_n \in D|_{\mathcal{X}}} |\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b|, \\ &\text{s.t. } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) > 0, \ \ \forall n \in 1, ..., N. \end{aligned} \tag{4.46}$$

So, we are looking for the parameters defining a hyperplane which maximizes the margin to the dataset $D$. The constraints on the problem above are such that we have a solution precisely when the data is linearly separable. If such is the case, we can write the problem equivalently as (exercise 15.1 [SSBD14]):

$$\arg\max_{(\boldsymbol{w}, b) : \|\boldsymbol{w}\| = 1} \min_{\boldsymbol{x}_n \in D|_{\mathcal{X}}} y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \tag{4.47}$$

According to [SSBD14] it can be shown that the previous optimization problem is equivalent to the following QP (quadratic program), whose solution can be easily obtained by a solver.

**Algorithm 3** Hard-SVM algorithm

**inputs:**

$D = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$

**Solve:**

$$(\boldsymbol{w}_0, b_0) = \underset{(\boldsymbol{w},b)}{\arg\min} \|\boldsymbol{w}\|^2 \text{ s.t. } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq 1, \ \ \forall n \in 1, ..., N.$$

**outputs:**

$$\boldsymbol{w}^* = \frac{\boldsymbol{w}_0}{\|\boldsymbol{w}_0\|}, \ b^* = \frac{b_0}{\|\boldsymbol{w}_0\|}$$

With this transformation we are searching for $\boldsymbol{w}$ with smallest possible norm where the margin is forced to be 1. However the margin units are now scaling with the norm of $\boldsymbol{w}$. The following lemma shows that the output of the previous algorithm indeed finds a solution of (4.47).

**Lemma 4.2.2.** *The output of the Hard-SVM algorithm is a solution of (4.47).*

*Proof.* See the proof of Lemma 15.2 of [SSBD14]. ■

**Soft-SVM:** The existence of a solution via Hard-SVM requires the dataset to be linearly separable. We now consider a relaxation of this requirement leading to the **Soft-SVM** algorithm.

In the Hard-SVM algorithm (3) we required that $y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq 1$ for all datapoints, i.e. that all datapoints are correctly labeled. These constraints can be relaxed by introducing slack variables $\xi_1, ..., \xi_N$, where $\xi_n \geq 0$ for all $n = 1, ..., N$, and changing the constraints to $y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq 1 - \xi_n$.

The $\xi_n$ is a measure of how much the initial constraint is being violated, in particular, if $\xi_n > 1$, the datapoint $(\boldsymbol{x}_n, y_n)$ is missclassified. We can define an optimization problem where the objective function is a trade-off between minimizing $\|\boldsymbol{w}\|$, which corresponds to the margin, and minimizing the average of the $\xi_n$, i.e. the average amount of violation of the initial constraints. This trade-off is regulated by the parameter $\lambda$. The algorithm is as follows:

**Algorithm 4** Soft-SVM algorithm

**inputs:**

$D = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$

**initialize:**

$\lambda > 0$

**Solve:**

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \left( \lambda \|\boldsymbol{w}\|^2 + \frac{1}{N} \sum_{n=1}^N \xi_n \right)$$
$$\text{s.t. } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq 1 - \xi_n \text{ and } \xi_n \geq 0, \ \ \forall n \in 1, ..., N.$$

**outputs:**

$\boldsymbol{w}, b$

**Support vectors:** The origin of the "Support vector machine" name comes from the fact that the solution of the Hard-SVM can be seen to be a linear combination of vectors which are at a distance $1/\|\boldsymbol{w}^*\|$ from the separating hyperplane. These are called the support vectors. We will not go into detail here but this relies on something called **Fritz John optimality conditions**, see Section 15.3 of [SSBD14] for further details on this. However we see already here a connection to kernel methods, the solution is a linear combination of some datapoints, those for which $|\langle \boldsymbol{w}^*, \boldsymbol{x}_n \rangle| = 1$.

**SVM and Kernel Methods:** We will now see rigorously how SVMs relate to kernel methods via dualization. The optimization problem given by (3) can be modified by considering hyperplanes which

pass through the origin and divide the space in homogeneous half-spaces, setting the bias to zero. The problem is equivalent to the original by either centralizing the data, i.e. removing the mean or adding an extra dimension to input space. The optimization problem becomes the following:

$$\begin{aligned} &\min_{\boldsymbol{w}} \|\boldsymbol{w}\|^2 \\ &\text{s.t. } y_n \langle \boldsymbol{w}, \boldsymbol{x}_n \rangle \geq 1, \ \ \forall n \in 1, ..., N. \end{aligned} \tag{4.48}$$

Consider this as the primal problem. First we write each constraint equation in the form $1 - y_n \langle \boldsymbol{w}, \boldsymbol{x}_n \rangle \leq 0$ and associate to it a dual variable $\alpha_n > 0$, for all $n = 1, ..., N$. Then, the Lagrangian is given by:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n (1 - y_n \langle \boldsymbol{w}, \boldsymbol{x}_n \rangle). \tag{4.49}$$

And the dual problem can be written as:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}} \left( \frac{1}{2} \|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n (1 - y_n \langle \boldsymbol{w}, \boldsymbol{x}_n \rangle) \right) \\ &\text{s.t. } \alpha_n \geq 0, \ \ \forall n \in 1, ..., N. \end{aligned} \tag{4.50}$$

where the $1/2$ is introduced for convenience when taking the derivative, which we are about to do. Because the minimization problem is now unconstrained in $\boldsymbol{w}$ it is actually a convex problem, since the quadratic term is $\boldsymbol{w}^T \boldsymbol{I} \boldsymbol{w}$ and the identity matrix is of course positive semi-definite. Thus, to find the minimum we just compute the gradient:

$$\begin{aligned} &\nabla_{\boldsymbol{w}} \left( \frac{1}{2} \|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \alpha_n (1 - y_n \langle \boldsymbol{w}, \boldsymbol{x}_n \rangle) \right) = 0 \Leftrightarrow \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n = 0 \\ \Rightarrow \ &\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n. \end{aligned} \tag{4.51}$$

This shows that the optimal solution is given by a linear combination of the training data. Replacing this solution in the dual problem above we obtain:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \left( \frac{1}{2} \left\| \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n \right\|^2 + \sum_{n=1}^{N} \alpha_n \left( 1 - y_n \left\langle \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i, \boldsymbol{x}_n \right\rangle \right) \right), \\ &\text{s.t. } \alpha_n \geq 0, \ \ \forall n \in 1, ..., N. \end{aligned} \tag{4.52}$$

We now see that the dual problem only involves inner products between the input datapoints and the connection with kernel methods is made explicit by specifying this inner products via a kernel function.

By dualizing Hard-SVM we have shown that the solution must be given by a linear combination of the training data, however this is more general. The general SVM problem can be formulated as the following optimization problem (see 16.2 [SSBD14]):

$$\min_{\boldsymbol{w}} \left( f(\langle \boldsymbol{w}, \phi(\boldsymbol{x}_1) \rangle, ..., \langle \boldsymbol{w}, \phi(\boldsymbol{x}_N) \rangle) + R\|\boldsymbol{w}\| \right) \tag{4.53}$$

where $f : \mathbb{R}^N \mapsto \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \mapsto \mathbb{R}$ is a monotonically non-decreasing function. The following theorem, a particular case of the more general Mercer's Theorem, shows that there exists a solution given by a linear combination of the training data.

**Theorem 4.2.3.** ***(Representer Theorem)***
*Consider an input space $\mathcal{X}$ and a feature map $\phi : \mathcal{X} \mapsto \mathcal{F}$ to some Hilbert space. There exists $\boldsymbol{\alpha} \in \mathbb{R}^N$ such that:*

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_i \phi(\boldsymbol{x}_n), \tag{4.54}$$

*is an optimal solution for (4.53).*

*Proof.* See the proof of Theorem 16.1 in [SSBD14]. ■

Then, for any $\boldsymbol{x} \in \mathcal{X}$, because the predictor is linear, of the form $y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x})$, we have:

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_i \phi(\boldsymbol{x}_n)^T \phi(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_i \left\langle \phi(\boldsymbol{x}_n), \phi(\boldsymbol{x}) \right\rangle, \tag{4.55}$$

and it can be specified with recourse to a kernel function. Of course, the particular choice of kernel function is still a problem that needs to be addressed. In practice one tries kernels of different families, linear kernels, radial basis function kernels (RBF), polynomial kernels, etc. choosing the ones, and the corresponding hyperparameters, which perform best on the specific dataset.

The SVM is a non-probabilistic discriminative model, we now introduce a second kernel method, in a probabilistic setting, known as a **Gaussian process**.

## 4.3 Gaussian Processes

A Gaussian process defines a probability distribution over a function space and is specified via a kernel function. Here we follow mainly [Bis06] with some inspiration from [RW06] as well.

**Definition 4.3.1.** ***(Gaussian Process)***
*A Gaussian Process is a collection of random variables $X_t$, indexed on some (possibly infinite) set $T$, such that the joint distribution of any finite subset $\{X_{t_1}, ..., X_{t_k}\}$, $k \in \mathbb{N}$, is a joint Gaussian distribution.*

The Gaussian Process is completely specified by the mean and covariance functions. We write a process as $f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right)$ where:

$$\begin{aligned} &m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]. \\ &k(\boldsymbol{x}, \boldsymbol{x}') = \operatorname{Cov}\left(f(\boldsymbol{x}), f(\boldsymbol{x}')\right) = \mathbb{E}\left[\left(f(\boldsymbol{x}) - m(\boldsymbol{x})\right)\left(f(\boldsymbol{x}') - m(\boldsymbol{x}')\right)\right]. \end{aligned} \tag{4.56}$$

**Example 4.3.1.** *Consider again the linear regression given by:*

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}),$$

*with a prior on the parameters $\boldsymbol{w} \sim \mathcal{N}(0, \Sigma)$. The joint distribution over $f(\boldsymbol{w})$ and $f(\boldsymbol{w}')$ (or any finite set) is Gaussian, since it can be obtained by a linear combination of Gaussian distributions, and we can compute:*

$$\begin{aligned} &\mathbb{E}[f(\boldsymbol{x})] = \boldsymbol{\phi}(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}] = 0. \\ &\operatorname{Cov}\left(f(\boldsymbol{x}), f(\boldsymbol{x}')\right) = \mathbb{E}\left[f(\boldsymbol{x}) f(\boldsymbol{x}')\right] = \boldsymbol{\phi}(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^T] \boldsymbol{\phi}(\boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \Sigma \boldsymbol{\phi}(\boldsymbol{x}'). \end{aligned} \tag{4.57}$$

*In the particular case of an isotropic Gaussian prior $\boldsymbol{w} \sim \mathcal{N}(0, \alpha^{-1}\boldsymbol{I})$ we will have:*

$$\operatorname{Cov}\left(f(\boldsymbol{x}), f(\boldsymbol{x}')\right) = \mathbb{E}\left[f(\boldsymbol{x}) f(\boldsymbol{x}')\right] = \boldsymbol{\phi}(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^T] \boldsymbol{\phi}(\boldsymbol{x}') = \frac{1}{\alpha} \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') \tag{4.58}$$

**Remark 4.3.1.** *Note that, by taking the mean to be zero, one can characterize the Gaussian process completely by specifying the covariance matrix which can be specified via the kernel function $k$ as $\mathbb{E}[f(\boldsymbol{x}_n) f(\boldsymbol{x}_m)] = k(\boldsymbol{x}_n, \boldsymbol{x}_m)$.*

**Regression via Gaussian Processes:** Suppose we have a noisy dataset $\{(\boldsymbol{x}_1, t_1), ..., (\boldsymbol{x}_N, t_N)\}$ where we assume the target variable $t$ is of the form:

$$\underbrace{t(\boldsymbol{x}_n)}_{=:t_n} = \underbrace{y(\boldsymbol{x}_n)}_{=:y_n} + \underbrace{\epsilon(x_n)}_{\epsilon_n}, \tag{4.59}$$

where $\epsilon \sim \mathcal{N}(0, \beta^{-1})$ is a random Gaussian noise variable which is independent over each observation $n$. We have then:

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}), \tag{4.60}$$

where the precision $\beta$ is an hyperparemeter and $y_n := y(x_n)$. Considering $\boldsymbol{t} = (t_1, ..., t_N)^T$ and $\boldsymbol{y} = (y_1, ..., y_N)^T$, because the noise is independent on each $x_n$ the marginals are Gaussian with zero covariance therefore we have the joint Gaussian distribution:

$$p(\boldsymbol{t}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{t}|\boldsymbol{y}, \beta^{-1}\boldsymbol{I}_N). \tag{4.61}$$

Now, instead of parametrizing $y$ via basis functions we define a Gaussian process over $y$. Considering $y \sim \mathcal{GP}\big(0, k)\big)$, where $k$ is a kernel function, via the definition of a Gaussian process, on the dataset we have $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K})$, where $\boldsymbol{K}$ is a Gram matrix obtained via the kernel function $k$ with entries $\boldsymbol{K}_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m)$. Of course there is a choice of a kernel function here which is typically chosen such that "similar" points $\boldsymbol{x}_n, \boldsymbol{x}_m$ have larger covariance, i.e. are more strongly correlated.

The marginal distribution $p(\boldsymbol{t})$, conditioned on the datapoints $\boldsymbol{x}_1$,...,$\boldsymbol{x}_N$, is obtained via the marginal of Proposition 6.3.14 in appendix:

$$p(\boldsymbol{t}) = \int p(\boldsymbol{t}|\boldsymbol{y})p(\boldsymbol{y})d\boldsymbol{y} = \mathcal{N}(\boldsymbol{t}|\boldsymbol{0}, \boldsymbol{K} + \beta^{-1}\boldsymbol{I}_N) := \mathcal{N}(\boldsymbol{t}|\boldsymbol{0}, \boldsymbol{C}), \tag{4.62}$$

i.e. the covariance matrix is of the form:

$$C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}. \tag{4.63}$$

A typically used (parametric) kernel function $k$ for a Gaussian process regression is:

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left( -\frac{\theta_1}{2}\|\boldsymbol{x}_n - \boldsymbol{x}_m\|_2^2 \right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m. \tag{4.64}$$

So far we have modeled a distribution over our datapoints and we want to predict a target for any new vale of the input.

**Predictor:** We want to determine the value $t_{N+1}$ corresponding to some unseen input $\boldsymbol{x}_{N+1}$. We have to evaluate the distribution $p(t_{N+1}|(\boldsymbol{X}, \boldsymbol{t}), \boldsymbol{x}_{N+1})$, where $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N)^T$ and $D = (\boldsymbol{X}, \boldsymbol{t})$ is the dataset we have access to. To do this we define first the joint probability distribution $p(t_{N+1}, \boldsymbol{t}) \sim \mathcal{N}(t_{N+1}, \boldsymbol{t}|\boldsymbol{0}, \boldsymbol{C}_{N+1})$ and then use the conditional Gaussian results found in appendix, particularly Section 6.3.5. We first partition the covariance matrix $\boldsymbol{C}_{N+1}$, which is $(N+1) \times (N+1)$ in the following way:

$$\boldsymbol{C}_{N+1} := \begin{bmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{bmatrix}, \tag{4.65}$$

where $\boldsymbol{C}_N$ is the $N \times N$ covariance matrix over the distribution on $\boldsymbol{t}$ (the previous $\boldsymbol{C}$) and $\boldsymbol{k}$ is given by the same kernel but evaluated at $\boldsymbol{x}_{N+1}$, i.e. it is of the form $k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1})$ for $n = 1, ..., N$, and $c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$. Via Proposition 6.3.12, the distribution $p(t_{N+1}|(\boldsymbol{X}, \boldsymbol{t}), \boldsymbol{x}_{N+1})$ is Gaussian with mean and covariance given by:

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}, \tag{4.66}$$

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}. \tag{4.67}$$

were the variance depends as well on the point $\boldsymbol{x}_{N+1}$, via the kernel function $k$. This result defines the predictor for a Gaussian process with the only restriction on this construction being that $\boldsymbol{K}$ be positive semidefinite, or equivalently, that $k$ is a valid kernel function.

It is instructive to show that, if we know the feature map $\boldsymbol{\phi}$ which defines the kernel $k$, we can recover the results obtained in the linear regression section by starting with the Gaussian process formulation (exercise 6.21 in [Bis06]).

**Recovering the previous formulation by using an explicit feature map $\phi$:** In the Gaussian process regression we obtained:

$$p(t_{N+1}|\boldsymbol{t}, \boldsymbol{X}, \boldsymbol{x}_{N+1}) = \mathcal{N}\left(t_{N+1}|\boldsymbol{k}^T\boldsymbol{C}_N^{-1}\boldsymbol{t}, c - \boldsymbol{k}^T\boldsymbol{C}_N^{-1}\boldsymbol{k}\right), \tag{4.68}$$

while in Bayesian linear regression, see Section 2.1.4, we obtained:

$$p(t|D, \alpha, \beta, \boldsymbol{x}) = \mathcal{N}\big(t|\boldsymbol{m}_N^T\boldsymbol{\phi}(\boldsymbol{x}), \frac{1}{\beta} + \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x})\big), \tag{4.69}$$

where:

$$\begin{aligned} \boldsymbol{m}_N &= \beta\boldsymbol{S}_N\Phi^T\boldsymbol{t}, \\ \boldsymbol{S}_N^{-1} &= \alpha\boldsymbol{I} + \beta\Phi^T\Phi, \end{aligned} \tag{4.70}$$

and considering the prior $p(\boldsymbol{w}|\alpha) = \mathcal{N}\big(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}\big)$. Since these are both normally distributed we need only to compare the mean and the covariance. Since we now have the feature map, such that $k(\boldsymbol{x}, \boldsymbol{x}') = \langle\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')\rangle$, equation (4.63) becomes:

$$\boldsymbol{C}_N(\boldsymbol{x}_n, \boldsymbol{x}_m) = \langle\boldsymbol{\phi}(\boldsymbol{x}_n), \boldsymbol{\phi}(\boldsymbol{x}_m)\rangle + \beta^{-1}\delta_{nm} \implies \boldsymbol{C}_N = \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1}\boldsymbol{I}_N, \tag{4.71}$$

where the entries of the matrix $\boldsymbol{\Phi}$ are given by $\boldsymbol{\Phi}_{nk} = \boldsymbol{\phi}_k(\boldsymbol{x}_n)$ (as defined in the linear regression section). We can write the mean as:

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1}\boldsymbol{I}_N\right)^{-1}\boldsymbol{t} = \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T\boldsymbol{\Phi}^T\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1}\boldsymbol{I}_N\right)^{-1}\boldsymbol{t}. \tag{4.72}$$

Using the identity $(\boldsymbol{I} + \boldsymbol{A}\boldsymbol{B})^{-1}\boldsymbol{A} = \boldsymbol{A}(\boldsymbol{I} + \boldsymbol{B}\boldsymbol{A})^{-1}$, as suggested, we obtain:

$$\begin{aligned} m(\boldsymbol{x}_{N+1}) &= \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T\boldsymbol{\Phi}^T\beta\left(\beta\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{I}_N\right)^{-1}\boldsymbol{t} \\ &= \beta\boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T\left(\beta\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{I}_N\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} \\ &= \beta\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{S}_N\boldsymbol{\Phi}^T\boldsymbol{t}. \end{aligned} \tag{4.73}$$

This is equivalent to the mean of equation (4.69) up to the $\alpha$ term. To obtain the exact expression, with the $\alpha$ we need to consider $k(\boldsymbol{x}, \boldsymbol{x}') = \alpha^{-1}\langle\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')\rangle$. For the covariance, using the $\alpha$, we compute:

$$\begin{aligned} &c - \boldsymbol{k}^T\boldsymbol{C}_N^{-1}\boldsymbol{k} = \\ &\quad = \beta^{-1} + \alpha^{-1}\boldsymbol{\phi}(\boldsymbol{x_{N+1}})^T\boldsymbol{\phi}(\boldsymbol{x_{N+1}}) - \alpha^{-2}\boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T\boldsymbol{\Phi}^T\big(\alpha^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1}\boldsymbol{I}_N\big)^{-1}\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x}_{N+1}) \\ &\quad = \beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x_{N+1}})^T\left(\alpha^{-1}\boldsymbol{I}_M - \alpha^{-2}\boldsymbol{\Phi}^T\big(\alpha^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1}\boldsymbol{I}_N\big)^{-1}\boldsymbol{\Phi}\right)\boldsymbol{\phi}(\boldsymbol{x}_{N+1}). \end{aligned} \tag{4.74}$$

Using now the identity:

$$(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1}=\boldsymbol{A}^{-1}-\boldsymbol{A}^{-1}\boldsymbol{B}\big(\boldsymbol{D}+\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}\big)^{-1}\boldsymbol{C}\boldsymbol{A}^{-1}, \tag{4.75}$$

as suggested (in reverse), we can transform the term in large brackets to obtain:

$$\begin{aligned}\Big(\alpha^{-1}\boldsymbol{I}_M-\alpha^{-2}\boldsymbol{\Phi}^T\big(\alpha^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T+\beta^{-1}\boldsymbol{I}_N\big)^{-1}\boldsymbol{\Phi}\Big)&=\\ &=\alpha^{-1}\boldsymbol{I}_M-\alpha^{-1}\boldsymbol{I}_M\boldsymbol{\Phi}^T\big(\boldsymbol{\Phi}(\alpha^{-1}\boldsymbol{I}_M)\boldsymbol{\Phi}^T+\beta^{-1}\boldsymbol{I}_N\big)\boldsymbol{\Phi}\alpha^{-1}\boldsymbol{I}_M\\ &=\big(\alpha\boldsymbol{I}_M+\beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}\big)^{-1}\\ &=\boldsymbol{S}_N.\end{aligned} \tag{4.76}$$

And we finally obtain:

$$c-\boldsymbol{k}^T\boldsymbol{C}_N^{-1}\boldsymbol{k}=\beta^{-1}+\boldsymbol{\phi}(\boldsymbol{x_{N+1}})^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x_{N+1}}). \tag{4.77}$$

which is equal to the covariance for the Bayesian linear regression distribution.

**Remark 4.3.2.** *As in equation (4.64) one can also consider a parameterized kernel. For methods which estimate the best parameters $\theta$ see Section 6.4.3 in [Bis06].*

Now we shall explore a quite recent result, from [JGH18], which connects kernel methods and neural networks in the infinite width limit. In particular we will show that neural networks and Gaussian processes are connected, which was already shown in [Nea94] in the case of one hidden layer at initialization.

## 4.4 Neural Tangent Kernel

A useful tool to understand the training process of neural networks via gradient descent, called the **Neural Tangent Kernel** (NTK), was formalized in 2018 with the paper by Jacot et al. [JGH18]. However, the connection between kernel methods and neural networks, at initialization, was first made in 1994 by Neal [Nea94]. There, he shows that a single hidden-layer neural network, at initialization and in the infinite width limit (when the number of neurons go to infinity), is equivalent to a Gaussian process which, with zero mean priors over weights, can be completely specified by a kernel function, as seen in Section 4.3. In [JGH18] this connection between kernel methods and neural networks is further extended by showing that, in this infinite width limit , the latter can be described by a kernel, not only at initialization but also during the training process, known as the neural tangent kernel. The main results shown in [JGH18] are the following:

- **Theorem 1 of [JGH18]** The neural tangent kernel, in general, depends on the parameters of the network which are randomly initialized. Therefore, at initialization, the NTK will be random as well. The first theorem shows that, when taking the width of the network to infinity, by the law of large numbers the NTK converges in probability to a deterministic kernel.
- **Theorem 2 of [JGH18]:** This is the main result. It states that, during the training process of the neural network and in the infinite width limit, the NTK remains constant and the evolution of the realization function follows a differential equation specified by the NTK. Thus, at any point in the training process, the realization function of the neural network is completely determined by the NTK via the solution of this differential equation.

Before presenting the formal statements and their respective proofs let us recall the general definition of a feedforward fully connected neural network of Section 3.1.

**Definition 4.4.1.** ***(FFFC Neural Network)***
*A feed-forward fully connected neural network is defined by its architecture $a := (\boldsymbol{N}, h)$, where $\boldsymbol{N} \in \mathbb{N}^{L+1}$ is the number of neurons which, for each layer $\ell = 0, ..., L$, has $N_\ell$ neurons, and where $h : \mathbb{R} \to \mathbb{R}$ is the activation function. The number of parameters $P$ of the network is given by:*

$$P(\boldsymbol{N}) := \sum_{\ell=1}^{L} N_\ell N_{\ell-1} + N_\ell. \tag{4.78}$$

*The output at each layer $\ell$ is defined by $f^{(\ell)} : \mathbb{R}^{N_0} \times \mathbb{R}^{P(\boldsymbol{N})} \mapsto \mathbb{R}^{N_\ell}$, even though not all parameters are required, and the realization function of the network is defined as $f_a : \mathbb{R}^{N_0} \times \mathbb{R}^{P(\boldsymbol{N})} \to \mathbb{R}^{N_L}$, i.e. $f_a := f^{(L)}(x, \theta)$. For each input $x \in \mathbb{R}^{N_0}$ and the parameters:*

$$\theta = \{\theta^{(\ell)}\}_{\ell=1}^{L} := \{(W^{(\ell)}, b^{(\ell)})\}_{\ell=1}^{L} \in \mathbb{R}^{P(N)}, \tag{4.79}$$

*we have:*

$$\begin{aligned} f^{(0)}(x, \theta) &= x, \\ f^{(1)}(x, \theta) &= \frac{1}{\sqrt{N_0}} W^{(1)} x + \beta b^{(1)}, \\ \hat{f}^{(\ell)}(x, \theta) &= h(f^{(\ell)}(x, \theta)), \quad \ell = 1, ..., L-1, \quad \textit{(h applied component wise)}, \\ f^{(\ell+1)}(x, \theta) &= \frac{1}{\sqrt{N_{(\ell)}}} W^{(\ell+1)} \hat{f}^{\ell}(x, \theta) + \beta b^{(\ell+1)}, \quad \ell = 1, ..., L-1. \end{aligned} \tag{4.80}$$

*The $W^{(\ell)} \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ are called the weight matrices, $b^{(l)} \in \mathbb{R}^{N_\ell}$ the bias vectors and $\beta > 0$ is a scalar parameter which allows to modify the influence of the bias parameters. The FFFC neural network is said to have width given by $\max\{N_\ell\}_{\ell=1}^{L}$ and is called deep if $L > 2$ and shallow if $L = 2$.*

**Remark 4.4.1.** *The introduction of the factors $1/\sqrt{N_\ell}$ are necessary for the asymptotic results that will be shown ahead, in the limit of infinite width. Because the weights are being scaled down the parameter $\beta$ is introduced to balance the influence of the bias. In particular the bias will have to be toned down, i.e. $\beta << 1$, depending on how wide the network is taken to be.*

In general, a neural network of depth $L$ defines a map as a function of the parameters given by $F^{(L)} : \mathbb{R}^P \to \mathcal{F}$, mapping $\theta \mapsto f_\theta$, where $P$ is the number of parameters given by (4.78) and $\mathcal{F} = \{f : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}\}$. Define the functional cost $C : \mathcal{F} \to \mathbb{R}_+$ where, as a function of the parameters we write $\hat{C} := C \circ F^{(L)} : \mathbb{R}^P \to \mathbb{R}_+$. Under gradient descent, each parameter evolves according to the following differential equation:

$$\theta_p^{t+1} = \theta_p^t - \eta \frac{\partial}{\partial \theta_p} \hat{C}(\theta^t), \tag{4.81}$$

with $p = 1, ..., P$ and $\eta \in \mathbb{R}_{>0}$. Which, in the continuous limit we can write:

$$\frac{d\theta_p(t)}{dt} := \lim_{\eta \to 0} \frac{\theta_p^{(t+1)} - \theta_p^t}{\eta} = -\frac{\partial}{\partial \theta_p} \hat{C}(\theta^t). \tag{4.82}$$

Let us now define the neural tangent kernel in the case of a 1-dimensional output.

### 4.4.1 NTK for a 1-dim Output ($N_L = 1$)

Given a dataset $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(N)}, y^{(N)})\}$, where each $\boldsymbol{x}^{(i)} \in \mathbb{R}^{N_0}$ and $y^{(i)} \in \mathbb{R}$, we can write the cost function as:

$$\hat{C}(\theta(t)) = \frac{1}{N} \sum_{j=1}^{N} \ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)}), \tag{4.83}$$

for a general loss function $\ell : \mathbb{R}^P \to \mathbb{R}_+$, where $f_{\theta(t)}(\boldsymbol{x}^{(j)}) \in \mathbb{R}$ for each $j = 1, ..., N$. Note that we consider the loss as a function of the parameters defining the predictor, i.e. the realization function of the network, compare with Definition 1.1.7. In this setting, equation (4.82) becomes:

$$\frac{d\theta_p(t)}{dt} = -\frac{1}{N}\sum_{j=1}^{N}\frac{\partial}{\partial\theta_p}\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)}) = -\frac{1}{N}\sum_{j=1}^{N}\frac{\partial\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)})}{\partial f_{\theta(t)}}\frac{\partial f_{\theta(t)}(\boldsymbol{x}^{(j)})}{\partial\theta_p}. \tag{4.84}$$

Consider now the evolution of the realization function $f_{\theta(t)}$ during training, we can write:

$$\begin{aligned}\frac{df_{\theta(t)}(\boldsymbol{x})}{dt} &= \sum_{p=1}^{P}\frac{\partial f_{\theta(t)}(\boldsymbol{x})}{\partial\theta_p}\frac{d\theta_p(t)}{dt} \\ &= -\frac{1}{N}\sum_{p=1}^{P}\frac{\partial f_{\theta(t)}(\boldsymbol{x})}{\partial\theta_p}\sum_{j=1}^{N}\frac{\partial\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)})}{\partial f_{\theta(t)}}\frac{\partial f_{\theta(t)}(\boldsymbol{x}^{(j)})}{\partial\theta_p} \\ &= -\frac{1}{N}\sum_{j=1}^{N}\frac{\partial\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)})}{\partial f_{\theta(t)}}\underbrace{\sum_{p=1}^{P}\frac{\partial f_{\theta(t)}(\boldsymbol{x})}{\partial\theta_p}\cdot\frac{\partial f_{\theta(t)}(\boldsymbol{x}^{(j)})}{\partial\theta_p}}_{=:\,\Theta(\boldsymbol{x},\boldsymbol{x}^{(j)};\,\theta(t))}.\end{aligned} \tag{4.85}$$

The NTK is defined as $\Theta(\boldsymbol{x}, \boldsymbol{x}'; \theta) : \mathbb{R}^{N_0} \times \mathbb{R}^{N_0} \to \mathbb{R}$ which is a kernel function defined by an inner product associated to the feature map $\boldsymbol{x} \to \sum_{p=1}^{P}\partial_{\theta_p} f_{\theta(t)}(\boldsymbol{x})\hat{\theta}_p$, with $\hat{\theta}_p$ a unit vector.

Since the realization function $f_{\theta(t)}$ is non-linear in general, due to the non-linear activation functions $h$ in the hidden layer outputs, the NTK depends on $\theta(t)$ which varies during training. So, in general, the NTK will be random at initialization, since the parameters $\theta$ are initialized randomly, and will vary during the training process as the parameters are updated. Let us now see how the NTK changes for a general $K-$dim output.

### 4.4.2 NTK for a K-dim Output ($N_L = K$)

Consider now a dataset $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(N)}, y^{(N)})\}$, where each $\boldsymbol{x}^{(i)} \in \mathbb{R}^{N_0}$ and $y^{(i)} \in \mathbb{R}^{N_L}$. We can write the cost function again as :

$$\hat{C}(\theta(t)) = \frac{1}{N}\sum_{j=1}^{N}\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)}), \tag{4.86}$$

where now $f_{\theta(t)}(\boldsymbol{x}^{(j)}) \in \mathbb{R}^{N_L}$. Equation (4.82) now becomes:

$$\frac{d\theta_p(t)}{dt} = -\frac{1}{N}\sum_{j=1}^{N}\frac{\partial}{\partial\theta_p}\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)}) = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{N_L}\frac{\partial\ell(f_{\theta(t)}(\boldsymbol{x}^{(j)}), y^{(j)})}{\partial f_{i,\theta(t)}}\frac{\partial f_{i,\theta(t)}(\boldsymbol{x}^{(j)})}{\partial\theta_p}e_i. \tag{4.87}$$

The evolution of the realization function, $f_{\theta(t)}$, during training can be written as:

$$\begin{aligned}\frac{df_{\theta(t)}}{dt} &= \sum_{p=1}^{P}\frac{\partial f_{\theta(t)}}{\partial\theta_p}\frac{d\theta_p(t)}{dt} = -\frac{1}{N}\sum_{k=1}^{N_L}\sum_{p=1}^{P}\frac{\partial f_{k,\theta(t)}}{\partial\theta_p}e_k \otimes \left(\sum_{j=1}^{N}\sum_{i=1}^{N_L}\frac{\partial\ell}{\partial f_{i,\theta(t)}}\frac{\partial f_{i,\theta(t)}}{\partial\theta_p}e_i\right) \\ &= -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{N_L}\frac{\partial\ell(f_{\theta(t)}), y^{(j)})}{\partial f_{i,\theta(t)}}\sum_{k=1}^{N_L}\underbrace{\left(\sum_{p=1}^{P}\frac{\partial f_{k,\theta(t)}}{\partial\theta_p}\frac{\partial f_{i,\theta(t)}}{\partial\theta_p}(e_k \otimes e_i)\right)}_{=:\,\Theta_{(k,i)}(\boldsymbol{x},\boldsymbol{x}';\theta(t))}.\end{aligned} \tag{4.88}$$

In this case the NTK is a map to matrices of the form $\Theta : \mathbb{R}^{N_0} \times \mathbb{R}^{N_0} \to \mathbb{R}^{N_L \times N_L}$.

Before looking at the first main result, Theorem 1 from [JGH18] let us see the connection between neural networks and Gaussian processes, which corresponds to Proposition 1 of [JGH18].

### 4.4.3 Relation with Gaussian Processes

As we have mentioned, the connection between neural networks of infinite width at initialization and Gaussian processes was established in [Nea94], for the case of a single hidden layer, with the extension to deep networks coming later in [LBN+17]. The following proposition shows this result in detail, where we use [Nea94], [LBN+17] and [JGH18] as guidelines.

**Proposition 4.4.1.** *(A FFFC neural network is a Gaussian process at initialization)*
*Consider a FFFC neural network with $L$ layers at initialization, Lipschitz activation function $h$ and i.i.d. parameter priors[1] given by $W_{ij}^{(\ell)} \sim \mathcal{N}(0,1)$ and $b_i^{(\ell)} \sim \mathcal{N}(0,1)$. In the limit of infinite width, $N_1, ..., N_{L-1} \to \infty$ sequentially, the output function $f_\theta^{(L)}$ tends in distribution to i.i.d. Gaussian processes of covariance $\Sigma^{(L)}$ defined recursively by:*

$$\begin{aligned} \Sigma^{(1)}(x, x') &= \frac{1}{N_0} x^T x' + \beta^2, \\ \Sigma^{(\ell+1)}(x, x') &= \mathbb{E}_{f \sim \mathcal{N}(0, \Sigma^{(\ell)})}[h(f(x))h(f(x'))] + \beta^2, \quad \ell \geq 1. \end{aligned} \tag{4.89}$$

*Proof.* We will look first at each output dimension of $f_\theta^{(L)}$, which we define as $f_{\theta,k}^{(L)}$ for $k = 1, ..., N_L$. The goal is to prove that, given any $n \in \mathbb{N}$ and a finite set of inputs $\{x^{(1)}, ..., x^{(n)}\}$, the random vector $[f_{\theta,k}^{(L)}(x^{(1)}) \dots f_{\theta,k}^{(L)}(x^{(n)})]$ has a joint Gaussian distribution with zero mean and the specified covariance matrix, i.e. $f_{\theta,k}^{(L)} \sim \mathcal{GP}(0, \Sigma^{(L)})$, where, in the end, we justify the independence over all $k = 1, ..., N_L$. The proof goes by induction on the number of layers.

- **No hidden layers ($L = 1$)** Consider the random vector $X = [f_{\theta,k}^{(1)}(x^{(1)}) \dots f_{\theta,k}^{(1)}(x^{(n)})]^T$, we have:

$$\mathbb{E}[f_{\theta,k}^{(1)}(x)] = \frac{1}{\sqrt{N_0}} \sum_{j=1}^{N_0} \mathbb{E}[W_{kj}^{(1)}] x_j + \beta \mathbb{E}[b_k^{(1)}] = 0, \ \forall x \in \mathbb{R}^{N_0}. \tag{4.90}$$

  Thus we have $\mathbb{E}[X] = [0 \dots 0]^T$. For any $x, x' \in \{x^{(1)}, ..., x^{(n)}\}$ the covariance is given by:

$$\begin{aligned} \mathrm{Cov}\big[f_{\theta,k}^{(1)}(x), f_{\theta,k}^{(1)}(x')\big] &= \mathbb{E}\left[\left(\frac{1}{\sqrt{N_0}} {W_k^{(1)}}^T x + \beta b_k^{(1)}\right)\left(\frac{1}{\sqrt{N_0}} {W_k^{(1)}}^T x' + \beta b_k^{(1)}\right)\right] \\ &= \frac{1}{N_0} \mathbb{E}\big[x^T W_k^{(1)} {W_k^{(1)}}^T x'\big] + \beta^2 \mathbb{E}[{b_k^{(1)}}^2] \\ &= \frac{1}{N_0} x^T \underbrace{\mathbb{E}\big[W_k^{(1)} {W_k^{(1)}}^T\big]}_{=\mathrm{Var}[W_k^{(1)}]=1} x' + \beta^2 \underbrace{\big(\mathbb{E}[{b_k^{(1)}}^2] - \mathbb{E}[b_k^{(1)}]^2\big)}_{=\mathrm{Var}[b_k^{(1)}]=1} \\ &= \frac{1}{N_0} x^T x' + \beta^2 := \Sigma^{(1)}(x, x'). \end{aligned} \tag{4.91}$$

- **One hidden layer ($L = 2$)** Consider a network with one hidden layer, i.e. $L = 2$, which has

[1] We could consider any finite variance on the priors in general, as done in [Nea94].

realization function $f_\theta$ that, for each output dimension $k = 1, ..., N_2$ and $x \in \mathbb{R}^{N_0}$ is given by:

$$\begin{aligned} f^{(2)}_{\theta,k}(x) &= \frac{1}{\sqrt{N_1}} \sum_{j=1}^{N_1} W^{(2)}_{kj} \hat{f}^{(1)}_j(x) + \beta b^{(2)}_k, \;\; k = 1, ..., N_2, \\ \hat{f}^{(1)}_j(x) &= h\Big( \frac{1}{\sqrt{N_0}} \sum_{i=1}^{N_0} W^{(1)}_{ji} x_i + \beta b^{(1)}_j \Big). \end{aligned} \tag{4.92}$$

We have the i.i.d. priors $W^{(\ell)}_{ij} \sim \mathcal{N}(0,1)$ and $b^{(\ell)}_i \sim \mathcal{N}(0,1)$ for $\ell = 1, 2$. Due to independence, the $f^{(2)}_{\theta,k}(x)$ are independent for different $k$ and, each $f^{(2)}_{\theta,k}(x)$ is a sum of i.i.d terms. The expectation on each weight of $f^{(2)}_{\theta,k}(x)$ is given by:

$$\mathbb{E}\Big[ W^{(2)}_{kj} \hat{f}^{(1)}_j(x) \Big] = \mathbb{E}\Big[ W^{(2)}_{kj} \Big] \mathbb{E}\Big[ \hat{f}^{(1)}_j(x) \Big] = 0, \tag{4.93}$$

since the $W^{(2)}_{kj}$ are independent from the $\hat{f}^{(1)}_j(x)$ (via independence with $W^{(1)}_{ki}$ and the $b^{(1)}_k$). The variance of each weight of $f^{(2)}_{\theta,k}(x)$ is given by:

$$\begin{aligned} \mathrm{Var}\Big[ W^{(2)}_{kj} \hat{f}^{(1)}_j(x) \Big] &= \mathbb{E}\Big[ \Big( W^{(2)}_{kj} \hat{f}^{(1)}_j(x) \Big)^2 \Big] \\ &= \mathbb{E}\Big[ {W^{(2)}_{kj}}^2 \Big] \mathbb{E}\Big[ \hat{f}^{(1)}_j(x)^2 \Big] = \mathbb{E}\Big[ \hat{f}^{(1)}_j(x)^2 \Big] \\ &:= V\Big[ \hat{f}^{(1)}_j(x) \Big], \end{aligned} \tag{4.94}$$

where, on the first equality cross terms vanish due to independence and where the last equality follows from $\mathbb{E}[{W^{(2)}_{kj}}^2] = \mathrm{Var}[W^{(2)}_{kj}] = 1$.

Because $V\big[\hat{f}^{(1)}_j(x)\big]$ is independent of $j$, since the weights are independent, and finite, because $h$ is Lipschitz and the parameters are normally distributed, i.e. they go to zero faster than $h$. By the central limit Theorem (see appendix, Theorem 6.3.5) we have:

$$\frac{1}{\sqrt{N_1}} \sum_{j=1}^{N_1} W^{(2)}_{kj} \hat{f}^{(1)}_j(x) \to \mathcal{N}\big(0, V[\hat{f}^{(1)}_j(x)]\big), \tag{4.95}$$

as $N_1 \to \infty$. Since $b^{(2)}_k \sim \mathcal{N}(0,1)$ and is independent of the other parameters we obtain a sum of Gaussians which is Gaussian (the property of being normally distributed is closed under linear operations) and given by:

$$f^{(2)}_{\theta,k}(x) \sim \mathcal{N}(0, V[\hat{f}^{(1)}_j(x)] + \beta^2). \tag{4.96}$$

We have seen that, for any $x \in \mathbb{R}^{N_0}$, $f^{(2)}_{\theta,k}(x)$ is Gaussian in the limit $N_1 \to \infty$. We are now going to apply the same idea to a random vector of the form $\Big[ f^{(2)}_{\theta,k}(x^{(1)}), ..., f^{(2)}_{\theta,k}(x^{(n)}) \Big]$, for any $n \in \mathbb{N}$.

Define first the random vector $Y_j := \Big[ W^{(2)}_{kj} \hat{f}^{(1)}_j(x^{(1)}) \; ... \; W^{(2)}_{kj} \hat{f}^{(1)}_j(x^{(n)}) \Big]^T$, which as we will see below does not depend on $k$. By the calculations above we have $\mathbb{E}[Y_j] = [0 \, ... \, 0]^T$ and the

covariance between inputs $x^{(p)}$ and $x^{(q)}$ is given by:

$$\begin{aligned}\mathrm{Cov}\big[W_{kj}^{(2)}\hat{f}_j^{(1)}(x^{(p)}), W_{kj}^{(2)}\hat{f}_j^{(1)}(x^{(q)})\big] &= \mathbb{E}\left[\left(W_{kj}^{(2)}\hat{f}_j^{(1)}(x^{(p)})W_{kj}^{(2)}\hat{f}_j^{(1)}(x^{(q)})\right)\right] \\ = \mathbb{E}\left[{W_{kj}^{(2)}}^2\right]\mathbb{E}\left[\hat{f}_j^{(1)}(x^{(p)})\hat{f}_j^{(1)}(x^{(q)})\right] &= \mathbb{E}\left[\hat{f}_j^{(1)}(x^{(p)})\hat{f}_j^{(1)}(x^{(q)})\right] =: \Sigma(x^{(p)}, x^{(q)}),\end{aligned} \tag{4.97}$$

where $\Sigma$ is the covariance matrix of $Y_j$ for any $j = 1, ..., N_1$, i.e. it is independent of $j$, and is finite by the same reasons as above. Thus, by taking the random vector $Y := \sum_{j=1}^{N_1} Y_j$, which is a sum of i.i.d. random vectors with $\mathbb{E}[Y_j] = [0 \dots 0]$ and $\mathrm{Cov}[Y_j] = \Sigma$, for $j = 1, ..., N_1$, by the multidimensional central limit Theorem (see appendix, Theorem 6.3.10) we have:

$$\frac{1}{\sqrt{N_1}}\sum_{j=1}^{N_1} Y_j \to \mathcal{N}\big(\mathbf{0}, \Sigma\big), \tag{4.98}$$

in the limit $N_1 \to \infty$. Again, since the random vector $[b_k^{(2)} \dots b_k^{(2)}] \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and is independent of the other parameters we obtain a sum of Gaussians which is Gaussian and given by:

$$Y \sim \mathcal{N}(\mathbf{0}, \Sigma + \beta^2\mathbf{1}), \tag{4.99}$$

where $\mathbf{1}$ is the matrix with all entries equal to one. We conclude, for a FFFC neural network with one hidden layer, that:

$$f_{\theta,k}^{(2)} \sim \mathcal{GP}(\mathbf{0}, \Sigma + \beta^2\mathbf{1}) \tag{4.100}$$

- **For $L$ layers, i.e. $L-1$ hidden-layers:** Consider now a network with $L$ layers, which is a composition of a network with $L-1$ layers, with realization function $f^{(L-1)} : \mathbb{R}^{N_0} \to \mathbb{R}^{N_{L-1}}$, followed by the element wise application of the activation function $h$ and an affine transformation $\mathbb{R}^{N_{L-1}} \to \mathbb{R}^{n_L}$, i.e.:

$$f_{\theta,k}^{(L)} = \frac{1}{\sqrt{N_{L-1}}}\sum_{j=1}^{N_{L-1}} W_{kj}^{(L)}\hat{f}_j^{(L-1)} + \beta b_k^{(L)}, \;\; k = 1, ..., N_L \tag{4.101}$$

where:

$$\hat{f}_j^{(L-1)} = h(f_j^{(L-1)}), \;\; h \text{ applied component wise.} \tag{4.102}$$

By the induction hypothesis, where each $n_i \to \infty$ sequentially, $i = 1, ..., L-2$, we have that $f_j^{(L-1)} \sim \mathcal{GP}(\mathbf{0}, \Sigma^{(L-1)})$. Thus, $f_{\theta,k}^{(L)}$ is a sum of i.i.d. random variables with finite variance and, by the central limit Theorem, when $N_{L-1} \to \infty$, has a Gaussian distribution. So, over an arbitrary finite set of inputs $\{x^{(1)}, ..., x^{(n)}\}$ we have the random vector:

$$\begin{aligned}X &= \big[f_{\theta,k}^{(L)}(x^{(1)}) \dots f_{\theta,k}^{(L)}(x^{(n)})\big]^T \\ &= \left[\frac{1}{\sqrt{N_{L-1}}}\sum_{j=1}^{N_{L-1}} W_{kj}^{(L)}\hat{f}_j^{(L-1)}(x^{(1)}) + \beta b_k^{(L)} \dots \frac{1}{\sqrt{N_{L-1}}}\sum_{j=1}^{N_{L-1}} W_{kj}^{(L)}\hat{f}_j^{(L-1)}(x^{(n)}) + \beta b_k^{(L)}\right]^T,\end{aligned} \tag{4.103}$$

which will be jointly Gaussian by using the multidimensional central limit Theorem as before

when taking $N_{L-1} \to \infty$, i.e. $f_{\theta,k}^{(L)} \sim \mathcal{GP}(\mathbf{0}, \Sigma^{(L)})$, where $\Sigma^{(L)}$ can be computed as follows:

$$\begin{aligned}
\Sigma^{(L)}(x,x') &= \mathbb{E}\big[f_{\theta,k}^{(L)}(x) f_{\theta,k}^{(L)}(x')\big] \\
&= \frac{1}{N_{L-1}} \sum_{j=1}^{N_{L-1}} \mathbb{E}\big[W_{kj}^{(L)} \hat{f}_j^{(L-1)}(x) W_{kj}^{(L)} \hat{f}_j^{(L-1)}(x')\big] + \beta^2 \mathbb{E}\big[{b_k^{(L)}}^2\big] \\
&= \frac{1}{N_{L-1}} \sum_{j=1}^{N_{L-1}} \mathbb{E}\big[{W_{kj}^{(L)}}^2\big] \mathbb{E}\big[\hat{f}_j^{(L-1)}(x) \hat{f}_j^{(L-1)}(x')\big] + \beta^2 \\
&\to \mathbb{E}_{f_j^{(L-1)} \sim \mathcal{GP}(\mathbf{0}, \Sigma^{(L-1)})}\big[h(f_j^{(L-1)}(x)) h(f_j^{(L-1)}(x'))\big] + \beta^2,
\end{aligned} \tag{4.104}$$

where cross terms, in the second equality, disappear due to independence and a zero mean prior over weights.

At this point we have shown that, over each dimension $k$ of the output, we have a Gaussian process. We still need to verify[2] however that over the entire output this remains the case. Although the outputs have shared weights, because the parameters on the hidden-layers go to zero in the infinite width limit, due to the $1/\sqrt{N_\ell}$ factors, the different dimensions of the output are independent. In particular, in the limit where $n_i \to \infty$ sequentially, $i = 1, ..., L-1$, we have $\mathrm{Cov}[f_{\theta,k_i}^{(L)}(x^{(p)}), f_{\theta,k_j}^{(L)}(x^{(q)})] = 0$, $\forall k_1 \neq, k_2$, due to independence of the weights at the last layer. Because zero covariance implies independence, in the case of Gaussian distributions, the result follows. ■

We now look at Theorem 1 of [JGH18], which shows that the NTK is deterministic at initialization in the infinite width limit.

### 4.4.4 Main results

The following theorem shows that, when taking the number of neurons to infinity sequentially, i.e. taking first $N_1 \to \infty$ then $N_2 \to \infty$ and so on, the NTK converges in probability do a deterministic limiting kernel.

**Remark 4.4.2.** *In general the order with which one takes limits matters, however it is stated in the appendix in [JGH18] that this could be, in principle, strengthened.*

**Theorem 4.4.2.** *(The NTK is deterministic at initialization)*
*Consider a FFFC neural network with $L$ layers at initialization and Lipschitz activation function $h$. In the infinite width limit, $N_1, ..., N_{L-1} \to \infty$ sequentially, the NTK $\Theta^{(L)}$ converges in probability to a deterministic limiting kernel:*

$$\Theta^{(L)} \to \Theta_\infty^{(L)} \otimes \boldsymbol{I}_{N_L}.^3 \tag{4.105}$$

*The kernel $\Theta_\infty^{(L)} : \mathbb{R}^{N_0} \times \mathbb{R}^{N_0} \to \mathbb{R}$ is defined recursively via the following relations:*

$$\begin{aligned}
\Theta_\infty^{(1)} &= \Sigma^{(1)}(x,x'), \\
\Theta_\infty^{(\ell+1)} &= \Theta_\infty^{(\ell)}(x,x') \dot{\Sigma}^{(\ell+1)}(x,x') + \Sigma^{(\ell+1)}(x,x'), \quad \ell \geq 1,
\end{aligned} \tag{4.106}$$

*where:*

$$\dot{\Sigma}^{(\ell+1)}(x,x') = \mathbb{E}_{f_j^{(\ell)} \sim \mathcal{GP}(\mathbf{0}, \Sigma^{(\ell)})}\big[\dot{h}(f_j^{(\ell)}(x)) \dot{h}(f_j^{(\ell)}(x'))\big]. \tag{4.107}$$

[2] This should probably be done more rigorously.
[3] This is the statement in [JGH18], not completely sure about the tensor product here.

*Proof.* The proof is again by induction on the number of layers. By definition the NTK is given by:

$$\Theta^{(L)}(\theta) = \sum_{p=1}^{P} \partial_{\theta_p} f^{(L)}(\theta) \otimes \partial_{\theta_p} f^{(L)}(\theta). \tag{4.108}$$

- For one layer $L = 1$, i.e. no hidden layers, we have:

$$\begin{aligned} f^{(L)}(x;\theta) &= \big[f_1^{(L)}(x;\theta) \dots f_{N_L}^{(L)}(x;\theta)\big], \\ \partial_{\theta_p} f^{(L)}(x;\theta) &= \big[\partial_{\theta_p} f_1^{(L)}(x;\theta) \dots \partial_{\theta_p} f_{N_L}^{(L)}(x;\theta)\big], \\ f_k^{(1)}(x;\theta) &= \frac{1}{\sqrt{N_0}} \sum_{j=1}^{N_0} W_{kj}^{(1)} x_j + \beta b_k^{(1)}, \text{ for } k = 1, ..., N_L. \end{aligned} \tag{4.109}$$

We can compute then:

$$\begin{aligned} \Theta_{k,k'}^{(1)}(x,x') &= \sum_{p=1}^{P} \left(\frac{\partial}{\partial\theta_p}\left(\frac{1}{\sqrt{N_0}} \sum_{j=1}^{N_0} W_{kj}^{(1)} x_j + \beta b_k^{(1)}\right)\right)\left(\frac{\partial}{\partial\theta_p}\left(\frac{1}{\sqrt{N_0}} \sum_{i=1}^{N_0} W_{k'i}^{(1)} x_i' + \beta b_{k'}^{(1)}\right)\right) \\ &= \frac{1}{N_0} \sum_{i=1}^{N_0} \sum_{j=1}^{N_1} x_i x_i' \delta_{jk} \delta_{jk'} + \beta^2 \delta_{jk} \delta_{jk'} \\ &= \frac{1}{N_0} \sum_{i=1}^{N_0} x_i x_i' \delta_{kk'} + \beta^2 \delta_{kk'} = \frac{1}{N_0} x^T x' \delta_{kk'} + \beta^2 \delta_{kk'} \\ &= \Sigma^{(1)}(x,x') \delta_{kk'}. \end{aligned} \tag{4.110}$$

- For $L + 1$ layers:

Consider now a $L+1$ FFFC neural network. As done in the previous proposition we can see it as the composition of a $L$-layered neural network mapping $\mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ together with a component wise application of the activation $h$ and an affine map $\mathbb{R}^{N_L} \to \mathbb{R}^{n_{L+1}}$, i.e:

$$f_{\theta,k}^{(L+1)} = \frac{1}{\sqrt{n_L}} \sum_{j=1}^{n_L} W_{kj}^{(L+1)} \hat{f}_j^{(L)} + \beta b_k^{(L+1)}, \;\; k = 1, ..., n_{L+1}, \tag{4.111}$$

where:

$$\hat{f}_j^{(L)} = h(f_j^{(L)}), \;\; h \text{ applied component wise.} \tag{4.112}$$

The NTK is given by:

$$\begin{aligned} \Theta_{kk'}^{(L+1)}(x,x') = \sum_{p=1}^{P} &\frac{\partial}{\partial\theta_p}\left(\frac{1}{\sqrt{n_L}} \sum_{j=1}^{n_L} W_{kj}^{(L+1)} \hat{f}_j^{(L)}(x) + \beta b_k^{(L+1)}\right) \\ &\times \frac{\partial}{\partial\theta_p}\left(\frac{1}{\sqrt{n_L}} \sum_{j=1}^{n_L} W_{k'j}^{(L+1)} \hat{f}_j^{(L)}(x') + \beta b_{k'}^{(L+1)}\right). \end{aligned} \tag{4.113}$$

We can now separate the sum on the $P$ parameters into a sum on $\tilde{P}$ parameters, which constitute the network up to layer $L$, and the remaining parameters of layer $L + 1$, i.e. equation (4.113) becomes:

$$\sum_{\tilde{p}=1}^{\tilde{P}} \frac{\partial}{\partial\theta_{\tilde{p}}}\left(\hat{f}_k^{(L+1)}(x;\theta)\right) \frac{\partial}{\partial\theta_{\tilde{p}}}\left(\hat{f}_{k'}^{(L+1)}(x';\theta)\right) + \sum_{p=1}^{P-\tilde{P}} \frac{\partial}{\partial\theta_p}\left(\hat{f}_k^{(L+1)}(x;\theta)\right) \frac{\partial}{\partial\theta_p}\left(\hat{f}_{k'}^{(L+1)}(x';\theta)\right). \tag{4.114}$$

The partial derivatives on the left term are given by:

$$\begin{aligned}\frac{\partial}{\partial\theta_{\tilde{p}}}\Big(\hat{f}_k^{(L+1)}(x;\theta)\Big) &= \frac{1}{\sqrt{N_L}}\sum_{j=1}^{N_L} W_{kj}^{(L+1)}\frac{\partial}{\partial\theta_{\tilde{p}}}\hat{f}_j^{(L)}(x)\\ &= \frac{1}{\sqrt{N_L}}\sum_{j=1}^{N_L} W_{kj}^{(L+1)}\dot{h}\big(f_j^{(L)}(x)\big)\frac{\partial}{\partial\theta_{\tilde{p}}}f_j^{(L)}(x).\end{aligned} \tag{4.115}$$

The whole left term of equation 4.114 becomes:

$$\begin{aligned}&\sum_{\tilde{p}=1}^{\tilde{P}}\frac{1}{N_L}\Big(\sum_{j=1}^{N_L} W_{kj}^{(L+1)}\dot{h}\big(f_j^{(L)}(x)\big)\frac{\partial}{\partial\theta_{\tilde{p}}}f_j^{(L)}(x)\Big)\Big(\sum_{i=1}^{N_L} W_{k'i}^{(L+1)}\dot{h}\big(f_i^{(L)}(x')\big)\frac{\partial}{\partial\theta_{\tilde{p}}}f_i^{(L)}(x')\Big)\\ &= \frac{1}{N_L}\sum_{\tilde{p}=1}^{\tilde{P}}\sum_{j=1}^{N_L}\sum_{i=1}^{N_L} W_{kj}^{(L+1)}W_{k'i}^{(L+1)}\dot{h}\big(f_j^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big)\frac{\partial}{\partial\theta_{\tilde{p}}}f_j^{(L)}(x)\frac{\partial}{\partial\theta_{\tilde{p}}}f_i^{(L)}(x')\\ &= \frac{1}{N_L}\sum_{j=1}^{N_L}\sum_{i=1}^{N_L}\Theta_{ji}^{(L)}(x,x')W_{kj}^{(L+1)}W_{k'i}^{(L+1)}\dot{h}\big(f_j^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big).\end{aligned} \tag{4.116}$$

By the induction hypothesis, as $N_1,...,N_{L-1}\to\infty$ sequentially, we have the convergence:

$$\begin{aligned}&\frac{1}{N_L}\sum_{j=1}^{N_L}\sum_{i=1}^{N_L}\Theta_{ji}^{(L)}(x,x')W_{kj}^{(L+1)}W_{k'i}^{(L+1)}\dot{h}\big(f_j^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big)\\ &\to \frac{1}{N_L}\sum_{j=1}^{N_L}\sum_{i=1}^{N_L}\Theta_{\infty}^{(L)}(x,x')W_{kj}^{(L+1)}W_{k'i}^{(L+1)}\dot{h}\big(f_j^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big)\delta_{ji}\\ &= \frac{1}{N_L}\sum_{i=1}^{N_L}\Theta_{\infty}^{(L)}(x,x')W_{ki}^{(L+1)}W_{k'i}^{(L+1)}\dot{h}\big(f_i^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big)\end{aligned} \tag{4.117}$$

By the law of large numbers, taking $N_L\to\infty$, the random variable converges in probability to its expectation where:

$$\begin{aligned}&\mathbb{E}\Big[\Theta_{\infty}^{(L)}(x,x')W_{ki}^{(L+1)}W_{k'i}^{(L+1)}\dot{h}\big(f_i^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big)\Big]\\ &= \Theta_{\infty}^{(L)}(x,x')\mathbb{E}\Big[\dot{h}\big(f_i^{(L)}(x)\big)\dot{h}\big(f_i^{(L)}(x')\big)\Big]\underbrace{\mathbb{E}\Big[W_{ki}^{(L+1)}W_{k'i}^{(L+1)}\Big]}_{=\delta_{kk'}}\\ &= \Theta_{\infty}^{(L)}(x,x')\mathbb{E}_{f_i^{(L)}\sim\mathcal{GP}(\mathbf{0},\Sigma^{(L)})}\big[\dot{h}(f_i^{(L)}(x))\dot{h}(f_i^{(L)}(x'))\big]\delta_{kk'}\\ &= \Theta_{\infty}^{(L)}(x,x')\dot{\Sigma}^{(L+1)}\delta_{kk'},\end{aligned} \tag{4.118}$$

where we used the previous proposition. At this point we are missing the sum on the parameters of the last layer $L+1$ which is similar to the calculation done for $L=1$. Thus, we finally obtain:

$$\begin{aligned}\Theta_{\infty}^{(L+1)}(x,x') &= \Theta_{\infty}^{(L)}(x,x')\dot{\Sigma}^{(L+1)} + \Sigma^{(L+1)}(x,x'),\\ \dot{\Sigma}^{(L+1)}(x,x') &= \mathbb{E}_{f^{(L)}\sim\mathcal{GP}(\mathbf{0},\Sigma^{(L)})}\big[\dot{h}(f^{(L)}(x))\dot{h}(f^{(L)}(x'))\big].\end{aligned} \tag{4.119}$$

■

The main result of [JGH18] states that, in the infinite width limit, the NTK is constant during the training process. This allows, by solving the ODEs given by (4.85) and (4.88), to determine the realization function as a function of the NTK. This is a bridge between kernel methods and neural networks which was previously only formalized at initialization. We do not show here the proof of the theorem, but it will hopefully appear in the sequel to this work, stay tuned!

## 4.5 Summary

In this last chapter we introduced **kernel methods**, methods which define predictors with recourse to the training data and the specification of a **kernel function**. This was done via a procedure known as the "kernel trick".

- **Kernel trick:** The motivation for the kernel trick can be given in two steps. First, we saw an example of how a higher-dimensional embedding of our data, via a **feature map** to a Hilbert space in general, may lead to linearly separable data. This then allows one to find a separating hyperplane which perfectly labels the training data, with recourse to any of the linear classification methods we studied before. In the second step we showed that, for a linear regression model, we only require the values of inner-products in the embedded space to obtain the predictor. This captures the essence of kernel methods, to find high-dimensional representations of the data that may lead to better predictors. However, the higher-dimensional setting increases the computational complexity of the problem. One is required to compute the feature transformations of the entire training set to compute the predictor by inner-products. By exploiting the fact that linear predictors can be entirely specified by a **kernel function**, which measures precisely the inner-products in the feature space, we can completely bypass the feature transformations which reduces the increased complexity of the higher-dimensional feature transformation.

Since we can build these predictors out of kernel functions it is necessary to understand what exactly determines a valid kernel function and how to construct one.

- **Kernel function:** This is a map which intuitively measures the similarity between two points in the input space and, rigorously, the inner-product of a feature representation of those points. The property that completely characterizes such a function was shown to be the **finitely positive semi-definite** property, i.e. that the matrix formed by the kernel restricted to any finite set is positive semi-definite. We further saw how to construct more complex kernels out of simpler ones via the **closure** and **composition** properties. This is very useful since our model now depends completely on the choice of a kernel function. Thus, the complexity allowed by the kernel function is crucial in determining a predictor with good generalization properties depending on the data we are considering.

One of the most widely used kernel methods is the **support vector machine** (SVM), a non-probabilistic discriminant model developed by Vapnik, Chervonenkis and others in the 1990s.

- **Support Vector Machine:** The SVM can be seen as an extension of the perceptron algortihm. The **Hard-SVM** algorithm, if the data is linearly separable, not only finds a hyperplane separating the data it finds the one with best possible **margin**. However, because most datasets will not be linearly separable in general, one can define a softer version, called **Soft-SVM**, which allows some slack over correctly classifying the training data. This results in a trade-off between maximizing the margin and minimizing incorrect classifications. We further saw how the Hard-SVM problem, via dualization, results in parameters which are linear combinations of the training data. Via a general formulation of the SVM optimization problem and the **representer Theorem** the general SVM solution is given by parameters which are linear combinations of the feature transformed training datapoints. Then, because the predictor is given by inner products with the weights, the whole solution can be specified with recourse to a kernel function.

Next we introduced the notion of a Gaussian process, a kernel method which, unlike the SVM, allows to construct probability distributions over the predicted values.

- **Gaussian Processes:** A Gaussian process is an extension of the notion of a Gaussian distribution in infinite dimensional spaces. Bypassing possible difficulties by stating that, over any finite subset of a possibly infinite set, we have a joint Gaussian distribution. We saw how a Gaussian process can be defined with recourse to a kernel function, which defines the covariance matrix in particular, together with a mean function, in general. We further saw how to do linear regression via a Gaussian process. Instead of specifying a parametrized predictor via basis functions we instead define a predictor based on a Gaussian process on the training dataset with an associated kernel. Then, via several properties of Gaussian distributions, which were proven in appendix, one can build a predictor for any new input. We saw furthermore, by specifying directly the feature map, that the Gaussian process, with a kernel function corresponding to this feature map, is completely equivalent to Bayesian linear regression.

Finally, besides looking at the relation of Gaussian processes and neural networks at initialization and in the infinite width limit, we explored the relation of neural networks and kernel methods via the so called **Neural Tangent Kernel** (NTK).

- **Neural Tangent Kernel:** The NTK is a kernel function which specifies the evolution of the realization function of a neural network via gradient descent. In general, the NTK depends on the network parameters, which are randomly initialized and evolve during the training process. However, in the infinite width limit we have seen that the NTK is deterministic at initialization. Furthermore, via Theorem 2 of [JGH18], that we have not yet proven, the NTK remains constant during the training process and defines a differential equation by which we can solve for the realization function as a function of the NTK. Thus, neural networks trained with gradient descent, and in the infinite width limit, are kernel methods specified by the neural tangent kernel function.

# Chapter 5

# Concluding remarks

With the benefit of hindsight we now see that although studying [Bis06] was very informative, and a lot was learned, with the goal of a rigorous mathematical treatment we should have probably focused more on [SSBD14]. In any case, looking at applications was very helpful in placing the formal treatment into context. With this theoretical direction in mind we have the following tentative structure for the master's thesis:

- **Chapter 1: Statistical Learning Theory**
  On Chapter 1 we expect to introduce one or more working examples alongside the development of the theory. Besides what is shown in the seminar, we would like to further study Chapter 7, on non-uniform learnability, Chapter 26 on Rademacher complexities and also the proof of the fundamental Theorem of statistical learning theory in Chapter 28. In order to not be stuck on a single reference it would be interesting to also look at [Vap98], the original source of many of these results, and further consult references contained in both of these works.

- **Chapter 2: Neural Networks**
  Here we would like to look more formally at results concerning the training process of neural networks. In a first approach we would like to cover Chapters 14 (SGD) and 20 (Neural Networks) of [SSBD14] which show a few convergence proofs but ideally we would look at recent results, such as those shown in [BGKP21] and the references therein. I would probably remove most of the content in the corresponding seminar's chapter and focus only on the more mathematical aspects.

- **Chapter 3: Kernel Methods**
  Although we looked at the definition and some properties of RKHS we have not included it in the seminar. This is mainly because we have yet to make progress (due to time being allocated elsewhere) in understanding the influence of RKHS in machine learning methods and in particular their relation with neural networks. This is one of the main areas to expand on for the thesis. The main references here would be [STC04], [SC08], [HSS08] and [PR16]. Possibly also [CX20] and other related work.

- **Chapter 4: "What's missing?"**
  Proper look at why the concepts of classical learning theory seem insufficient at explaining deep learning phenomena. In particular looking at [ZBH$^+$16] and other related references.

- **Chapter 5: Neural Tangent Kernel**
  This would essentially be a reconstruction of the NTK paper. We have looked at proposition 1 and Theorem 1 of [JGH18] but the main theorem, Theorem 2, still needs to be tackled. The idea would be to present the NTK and the proofs in a less abstract way and show an application of the results in a simple case. Maybe do some computational implementation and compare the realization function of the network to the NTK solution as we vary the width of the network.

- **Chapter 6: "Beyond the NTK"**
  This would consist on explaining the result by Prof. João Costa (and others), if everyone agrees of course, and possibly generalize it further. Not sure how much work this would consist of, we are not yet familiar with the result and what a possible generalization entails. Although, according to Prof. João Costa, the generalization has a mostly computational aspect to it. It would be interesting to add some computational experiments here as well, perhaps.

- **Chapter 7: "Something else?"**
  We are open to other possible directions. The previous chapters are mainly to provide some sense of direction to the thesis since the amount of information surrounding these topics can be a bit daunting.

# Bibliography


[BGKP21] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The modern mathematics of deep learning, 2021.

[Bis06] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[Çın11] E. Çınlar. *Probability and Stochastics*. Graduate Texts in Mathematics. Springer New York, 2011.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory 2nd Edition (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 12 2006.

[CX20] Lin Chen and Sheng Xu. Deep neural tangent kernel and laplace kernel have the same rkhs, 2020.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[HSS08] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3), jun 2008.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[JGH18] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[JW02] Richard Arnold Johnson and Dean W. Wichern. *Applied multivariate statistical analysis*. Prentice Hall, Upper Saddle River, NJ, 5. ed edition, 2002.

[KK94] Věra Kůrková and Paul C. Kainen. Functionally Equivalent Feedforward Neural Networks. *Neural Computation*, 6(3):543–558, 05 1994.

[LBN+17] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes, 2017.

[Mur22] K.P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2022.

[Nea94] Radford M Neal. Priors for infinite networks. In *Technical Report CRG-TR-94-1*. University of Toronto, 1994.

[PR16] Vern I. Paulsen and Mrinal Raghupathi. *An Introduction to the Theory of Reproducing Kernel Hilbert Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2016.

[PV17] Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks, 2017.

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

[SC08] I. Steinwart and A. Christmann. *Support Vector Machines*. Information Science and Statistics. Springer New York, 2008.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[SS16] Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks, 2016.

[SSBD14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, illustrated edition edition, 2004.

[Str19] G. Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, 2019.

[Tel16] Matus Telgarsky. Benefits of depth in neural networks, 2016.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[ZBH+16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2016.

# Chapter 6

# Appendix

## 6.1 Probability Theory

This section is a brief, and informal, collection of definitions and results in probability theory following chapter 1 of [Bis06] and parts of chapter 2 and 3 of [Mur22]. We start by considering univariate, real-valued, discrete random variables and some basic concepts such as expectation, variance and their properties. We then show these same results in the setting of univariate continuous random variables and extend them for the case of multivariate distributions.

### 6.1.1 Discrete Random Variables

We start by exploring the the notion of a joint probability distribution considering a **frequentist interpretation**, i.e. interpreting probabilities as experiments repeated an infinite number of times. Let $X$ and $Y$ be two discrete random variables. We define the joint probability of $X = x_i$ and $Y = y_j$ as:

$$p(X = x_i, Y = y_j) = \lim_{N\to\infty} \frac{n_{ij}}{N}, \tag{6.1}$$

where $n_{ij}$ is the number of trials for $X = x_i$ and $Y = y_i$ and $N$ is the total number of trials. We see that, to compute $p(X = x_i)$ we should sum over all possible values of $y_j$, i.e.:

$$p(X = x_i) = \sum_j \lim_{N\to\infty} \frac{n_{ij}}{N} = \lim_{N\to\infty} \frac{c_i}{N} = \sum_j p(X = x_i, Y = y_j), \tag{6.2}$$

where $c_i = \sum_j n_{ij}$. This is known as the **sum rule** (or **marginalization rule**) of probability which we write as:

$$p(X) = \sum_Y p(X, Y). \tag{6.3}$$

Suppose we are interested in the probability of those situations where $Y = y_j$ knowing that $X = x_i$. This is know as **conditional probability** which is given by:

$$p(Y = y_j|X = x_i) = \lim_{N\to\infty} \frac{n_{ij}}{c_i}, \tag{6.4}$$

where this is well defined when taking the limit $N \to \infty$. Manipulating this expressions we can find the relation between the joint and conditional probabilities:

$$p(X = x_i, Y = y_j) = \lim_{N\to\infty} \frac{n_{ij}}{N} = \lim_{N\to\infty} \frac{n_{ij}}{c_i}\frac{c_i}{N} = p(Y = y_j|X = x_i)p(X = x_i). \tag{6.5}$$

This is known as the **product rule** of probability which we write as:

$$p(X, Y) = p(Y|X)p(X). \tag{6.6}$$

From these results we can easily obtain **Bayes' theorem**:

$$p(Y|X) = \frac{p(X,Y)}{p(X)} = \frac{p(X|Y)p(Y)}{p(X)}, \tag{6.7}$$

using the product rule and the fact that $p(X,Y) = p(Y,X)$. This theorem allows for a different interpretation of probabilities, called the **Bayesian interpretation** which we explore in Section 2.1.4.

We now present several basic notions, that of independent random variables, the expectation of a random variable and its variance.

**Definition 6.1.1.** *(**Independent random variables**)*
*Let $\Omega_X$ and $\Omega_Y$ be the sample space of two discrete random variables $X, Y$. We say that $X$ and $Y$ are independent if, $\forall x \in \Omega_X, y \in \Omega_Y$ we have:*

$$p(X = x, Y = y) = p(X = x)p(Y = y), \tag{6.8}$$

*i.e. the joint distribution factorizes $p(X,Y) = p(X)p(Y)$.*

**Definition 6.1.2.** *(**Expectation for a discrete random variable**)*
*Given a discrete random variable $X$, with sample space $\Omega_X$, we define the expectation of $X$ by:*

$$\mathbb{E}[X] := \sum_{x \in \Omega_X} xp(X = x). \tag{6.9}$$

**Definition 6.1.3.** *(**Variance for a discrete random variable**)*
*Let $X$ be a discrete random variable with sample space $\Omega_X$, we define the variance of $X$ by:*

$$\mathrm{Var}[X] := \mathbb{E}[(X - \mathbb{E}[X])^2]. \tag{6.10}$$

**Definition 6.1.4.** *(**Covariance for discrete random variables**)*
*Let $X$ and $Y$ be discrete random variables we define their covariance as:*

$$\mathrm{Cov}(X,Y) := \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]. \tag{6.11}$$

We now present a few simple properties related to these definitions that will be used frequently and often without mention.

**Proposition 6.1.1.** *(**Expectation is linear**)*
*Given two discrete random variables $X, Y$, with sample spaces $\Omega_X$, $\Omega_Y$ respectively, we have:*

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]. \tag{6.12}$$

*Proof.*

$$\begin{aligned}
\mathbb{E}[aX + bY] &= \sum_{x \in \Omega_X} \sum_{y \in \Omega_Y} (ax + by)p(X = x, Y = y) = \\
&= \sum_{x \in \Omega_X} ax \sum_{y \in \Omega_Y} p(X = x, Y = y) + \sum_{y \in \Omega_Y} by \sum_{x \in \Omega_X} p(X = x, Y = y) \\
&= \sum_{x \in \Omega_X} axp(X = x) + \sum_{y \in \Omega_Y} byp(Y = y) \\
&= a\mathbb{E}[X] + b\mathbb{E}[Y].
\end{aligned} \tag{6.13}$$

Where we used the marginalization rule in the previous to last equality. ■

**Proposition 6.1.2.** *(**Expectation over independent random variables**)*
*Let $X, Y$ be two discrete and independent random variables $X$, with sample space $\Omega_X$ and sample space $\Omega_Y$ respectively. We have:*

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]. \tag{6.14}$$

*Proof.*

$$\begin{aligned}
\mathbb{E}[XY] &= \sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} xyp(X=x, Y=y) \\
&= \sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} xyp(X=x)p(Y=y) \\
&= \sum_{x\in\Omega_X} xp(X=x)\sum_{y\in\Omega_Y} yp(Y=y) \\
&= \mathbb{E}[X]\mathbb{E}[Y].
\end{aligned} \tag{6.15}$$

Where we used first the definition of independence, then the definition of expectation and finally the linear property of the expectation. ■

The n-th moment of a random variable $X$ is defined as $\mathbb{E}[X^n]$. We have the following useful relationship between the variance of a random variables and its second moment.

**Proposition 6.1.3.** *(**Variance and relation to $2^{nd}$ moment**)*

$$\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2. \tag{6.16}$$

*Proof.*

$$\begin{aligned}
\mathrm{Var}[X] :&= \mathbb{E}[(X - \mathbb{E}[X])^2] \\
&= \mathbb{E}[X^2 - 2\mathbb{E}[X]X + \mathbb{E}[X]^2) \\
&= \mathbb{E}[X^2] - 2\mathbb{E}[\mathbb{E}[X]X] + \mathbb{E}[\mathbb{E}[X]^2] \\
&= \mathbb{E}[X^2] - 2\mathbb{E}[X]^2 + \mathbb{E}[X]^2 \\
&= \mathbb{E}[X^2] - \mathbb{E}[X]^2.
\end{aligned} \tag{6.17}$$

■

We now extend these results to the setting of continuous random variables.

### 6.1.2 Continuous Random Variables

A formal treatment of continuous random variables requires the use of measure theory which we will not cover. Here we simply compile a few results non-rigorously.

Informally, for a univariate continuous random variable $X$, the **probability density** $p(x)$ is such that the probability of $x$ being over the interval $(x, x+dx)$ is given by $p(x)dx$, where $dx \to 0$. It needs to satisfy the following properties:

$$p(x) \geq 0, \tag{6.18}$$

$$\int_{-\infty}^{\infty} p(x)dx = 1. \tag{6.19}$$

Over a change of variables the transformation of the probability density depends on the transformation of the measure through the determinant of the Jacobian. Writing $x = g(y)$, where $g$ is a differentiable function, we have:

$$p_y(y) = p_x(x)\left|\frac{dx}{dy}\right| = p_x(g(y))|g'(y)|, \tag{6.20}$$

where the suffixes emphasize the fact that these are two different densities.

**Definition 6.1.5.** *(**Cumulative distribution function (CDF)**)*
*Given a continuous random variable $X$, with probability density function $p(x)$, we define the cumulative distribution function as:*

$$P(z) = \int_{-\infty}^{z} p(x)dx. \tag{6.21}$$

Given two continuous random variables $X, Y$, just like in the discrete case, we have equivalent formulations of the **sum rule**, **product rule** and **Bayes' theorem**.

$$p(x) = \int p(x, y)dy, \tag{6.22}$$

$$p(x, y) = p(y|x)p(x), \tag{6.23}$$

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}. \tag{6.24}$$

**Definition 6.1.6.** *(**Expectation for a continuous random variable**)*
*Given a continuous random variable $X$ we define the expectation of $X$ by:*

$$\mathbb{E}[X] := \int_{-\infty}^{\infty} xp(X = x) := \mu. \tag{6.25}$$

The definition of variance and covariance are the same as in the discrete case, where we use now the expectation as defined above. The results of Propositions 6.1.1, 6.1.2 and 6.1.3 are also still valid in the continuous case.

Given a function $f : \mathbb{R} \to \mathbb{R}$, then $f(X)$ is a random variable and we can similarly compute its expectation and variance. If $X$ and $Y$ are random variables distributed according to distributions $X \sim \mathcal{D}_X$ and $Y \sim \mathcal{D}_Y$ and we have a function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ we define:

$$\mathbb{E}_{X\sim\mathcal{D}_X}[g(X, Y)] := \int_{-\infty}^{\infty} g(x, y)p(x)dx, \tag{6.26}$$

$$\mathbb{E}_{X\sim\mathcal{D}_X}[g|Y = \hat{y}] := \int_{-\infty}^{\infty} g(x, \hat{y})p(x|Y = \hat{y})dx. \tag{6.27}$$

An often useful result, known as the law of total expectation, is as follows:

**Proposition 6.1.4.** *(**Law of total Expectation**)*
*Let $X$ and $Y$ be two continuous random variables defined on the same probability space, the following holds (under some technical assumptions):*

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]] \tag{6.28}$$

*Proof.* A non-rigorous proof is as follows:

$$\begin{aligned}\mathbb{E}[X] &= \int_{-\infty}^{\infty} xp(x)dx \\ &= \int_{-\infty}^{\infty} x\bigg(\int_{-\infty}^{\infty} p(x,y)dy\bigg)dx \\ &= \int_{-\infty}^{\infty} x\bigg(\int_{-\infty}^{\infty} p(x|y)p(y)dy\bigg)dx \\ &= \int_{-\infty}^{\infty} p(y)\bigg(\int_{-\infty}^{\infty} xp(x|y)dx\bigg)dy \\ &= \mathbb{E}[\mathbb{E}[X|Y]].\end{aligned} \tag{6.29}$$

■

We introduce here the notion of correlation, which is defined similarly for the discrete case.

**Definition 6.1.7.** ***(Pearson correlation)***
*Given random variables $X, Y$ we define the Pearson correlation coefficient as:*

$$\rho(X,Y) := \text{Corr}[X,Y] := \frac{\text{Cov}[X,Y]}{\sqrt{\text{Var}[X]\text{Var}[Y]}}. \tag{6.30}$$

This is a useful notion that measures the linear relationship between the two random variables $X$ and $Y$ via the following proposition:

**Proposition 6.1.5.** *Given random variables $X, Y$, then $\text{Corr}[X,Y] = 1$ if and only if $Y = aX + b$, with $a \in \mathbb{R}_{>0}, b \in \mathbb{R}$.*

*Proof.* We prove the if part only. Suppose $Y = aX + b$ then:

$$\begin{aligned}\text{Cov}[X,Y] &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \\ &= \mathbb{E}[X(aX+b)] - \mathbb{E}[X]\mathbb{E}[aX+b] \\ &= a\mathbb{E}[X^2] + b\mathbb{E}[X] - a\mathbb{E}[X]^2 - b\mathbb{E}[X] \\ &= a(\mathbb{E}[X^2] - \mathbb{E}[X]^2) \\ &= a\text{Var}[X].\end{aligned} \tag{6.31}$$

and we can compute:

$$\text{Corr}[X,Y] = \frac{\text{Cov}[X,Y]}{\sqrt{\text{Var}[X]\text{Var}[Y]}} = a\frac{\sqrt{\text{Var}[X]}}{\sqrt{\text{Var}[Y]}} = a\frac{\sqrt{\text{Var}[X]}}{\sqrt{a^2\text{Var}[X]}} = 1, \tag{6.32}$$

since $\text{Var}[Y] = \text{Var}\big[\big(aX + b - (a\mathbb{E}[X] - b)\big)^2\big] = a^2\mathbb{E}\big[(X - \mathbb{E}[X])^2\big] = a^2\text{Var}[X]$. We can compute the $a$ coefficient in general to be:

$$a = \frac{\text{Cov}[X,Y]}{\text{Var}[X]} = \frac{\text{Corr}[X,Y]\sqrt{\text{Var}[Y]}}{\sqrt{\text{Var}[X]}}, \tag{6.33}$$

which shows how the slope of the regression line relates with the Pearson correlation. In particular how the covariance measures the linear relationship between the data. ■

**Remark 6.1.1.** *It is easy to see, from proposition 6.1.2, that if $X$ and $Y$ are independent then $\text{Cov}[X,Y] = 0$, and therefore $\text{Corr}[X,Y] = 0$. However, note that if $\text{Corr}[X,Y] = 0$, we do not necessarily have that $X$ and $Y$ are independent. As an example, suppose that $X \sim Uni[-1,1]$ and let*

*$Y = X^2$. The random variable $Y$ is completely specified by $X$, and thus dependent, however we can compute:*

$$\text{Cov}[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(X^2 - \mathbb{E}[X^2])] = \mathbb{E}[X^3] - \mathbb{E}[X]\mathbb{E}[X^2] - \mathbb{E}[X]\mathbb{E}[X^2] + \mathbb{E}[X]\mathbb{E}[X^2]. \quad (6.34)$$

*We have $\mathbb{E}[X] = 0$, since we are integrating an odd function between $-1$ and $1$. For the same reason we also have $\mathbb{E}[X^3] = 0$ and certainly $\mathbb{E}[X^2]$ is finite, therefore $\text{Cov}[X, Y] = 0$ and thus $\text{Corr}[X, Y] = 0$. A better notion of dependence between random variables will be introduced later in the information theory section, called **mutual information** (definition 6.2.5), which will also capture nonlinear relations between the random variables.*

**Remark 6.1.2.** *It is often the case that two random variables are highly correlated but there is in fact no causal link between the two. This is known as **spurious correlation** and there are typically other hidden variables responsible for this correlation.*

### 6.1.3 Multivariate Random Variables

We now introduce the notion of random vectors for the case of continuous random variables. Similar results can also be defined in the discrete case.

**Definition 6.1.8.** ***(Continuous random vector)***
*A continuous random vector $\boldsymbol{X} = (X_1, ..., X_n)^T$ is a vector whose components $X_i$, $i = 1, ..., n$, are continuous random variables defined over the same probability space.*

The expectation is a natural extension of what we had before and remains a linear function.

**Definition 6.1.9.** ***(Expectation of a random vector)***
*Given a random vector $\boldsymbol{X} = (X_1, ..., X_n)^T$ we define the expectation:*

$$\mathbb{E}[\boldsymbol{X}] = (\mathbb{E}[X_1], ..., \mathbb{E}[X_n])^T := \boldsymbol{\mu}. \quad (6.35)$$

**Proposition 6.1.6.** ***(Expectation is linear)***
*Given a random vector $\boldsymbol{X} = (X_1, ..., X_n)^T$, $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ we have:*

$$\mathbb{E}[\boldsymbol{a}^T \boldsymbol{X} + \boldsymbol{b}] = \boldsymbol{a}^T \mathbb{E}[\boldsymbol{X}] + \boldsymbol{b}. \quad (6.36)$$

*Proof.*

$$\mathbb{E}[\boldsymbol{a}^T \boldsymbol{X} + \boldsymbol{b}] = \mathbb{E}\left[\sum_{i=1}^{n} a_i X_i + b_i\right] = \sum_{i=1}^{n} a_i \mathbb{E}[X_i] + b_i = \boldsymbol{a}^T \mathbb{E}[\boldsymbol{X}] + \boldsymbol{b}, \quad (6.37)$$

by proposition 6.1.1. ■

**Definition 6.1.10.** ***(Covariance matrix of a random vector)***
*Given a random vector $\boldsymbol{X} = (X_1, ..., X_n)^T$ we define the covariance matrix:*

$$\text{Var}[\boldsymbol{X}] = \text{Cov}[\boldsymbol{X}, \boldsymbol{X}] := \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^T] := \boldsymbol{\Sigma}, \quad (6.38)$$

*where each entry is of the form:*

$$\boldsymbol{\Sigma}_{ij} = \begin{cases} \text{Var}[X_i], & \text{if } i = j, \\ \text{Cov}[X_i, X_j], & \text{if } i \neq j. \end{cases} \quad (6.39)$$

**Proposition 6.1.7.** ***(Relation of covariance with $2^{nd}$-moment)***

$$\text{Cov}[\boldsymbol{X}, \boldsymbol{X}] = \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T. \quad (6.40)$$

*Proof.* The proof is basically the same as for proposition 6.1.3.

$$\begin{aligned}\mathrm{Cov}[\boldsymbol{X},\boldsymbol{X}] &= \mathbb{E}[(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])^T] = \mathbb{E}\big[\boldsymbol{X}\boldsymbol{X}^T - \boldsymbol{X}\mathbb{E}[\boldsymbol{X}]^T - \mathbb{E}[\boldsymbol{X}]\boldsymbol{X}^T + \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T\big] \\ &= \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}^T] + \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T \\ &= \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}]^T \\ &= \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T,\end{aligned} \tag{6.41}$$

where we used $\mathbb{E}[\boldsymbol{X}^T] = \mathbb{E}[\boldsymbol{X}]^T$. We can write equivalently:

$$\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] = \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T. \tag{6.42}$$

■

**Proposition 6.1.8.** ***(Covariance matrix of a linear transformation)***

$$\mathrm{Var}[\boldsymbol{A}\boldsymbol{X}+\boldsymbol{b}] = \boldsymbol{A}\boldsymbol{X}\boldsymbol{A}^T. \tag{6.43}$$

*Proof.* We have $\mathbb{E}[\boldsymbol{A}\boldsymbol{X}+\boldsymbol{b}] = \boldsymbol{A}\mathbb{E}[\boldsymbol{X}] + \boldsymbol{b}$ thus:

$$\begin{aligned}&\mathrm{Cov}[\boldsymbol{A}\boldsymbol{X}+\boldsymbol{b}, \boldsymbol{A}\boldsymbol{X}+\boldsymbol{b}] = \\ &\quad= \mathbb{E}[\big((\boldsymbol{A}\boldsymbol{X}+\boldsymbol{b}) - (\boldsymbol{A}\mathbb{E}[X]+\boldsymbol{b})\big)\big((\boldsymbol{A}\boldsymbol{X}+\boldsymbol{b}) - (\boldsymbol{A}\mathbb{E}[X]+\boldsymbol{b})\big)^T] \\ &\quad= \mathbb{E}[\boldsymbol{A}(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])\big(\boldsymbol{A}(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])\big)^T] \\ &\quad= \mathbb{E}[\boldsymbol{A}(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])\big)^T\boldsymbol{A}^T] \\ &\quad= \boldsymbol{A}\big((\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])\big)^T\big)\boldsymbol{A}^T \\ &\quad= \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^T.\end{aligned} \tag{6.44}$$

■

**Definition 6.1.11.** ***(Cross-covariance)***
*Given random vectors $\boldsymbol{X}, \boldsymbol{Y}$ we define the cross-covariance matrix:*

$$\mathrm{Cov}[\boldsymbol{X},\boldsymbol{Y}] := \mathbb{E}[(\boldsymbol{X}-\mathbb{E}[\boldsymbol{X}])(\boldsymbol{Y}-\mathbb{E}[\boldsymbol{Y}])^T]. \tag{6.45}$$

By repeating the calculation in the proof of 6.1.7 we have the following property:

**Proposition 6.1.9.** *Given two random vectors $\boldsymbol{X}, \boldsymbol{Y}$ the cross-covariance is given by:*

$$\mathrm{Cov}[\boldsymbol{X},\boldsymbol{Y}] = \mathbb{E}[\boldsymbol{X}\boldsymbol{Y}^T] - \boldsymbol{\mu}_{\boldsymbol{X}}\boldsymbol{\mu}_{\boldsymbol{Y}}^T. \tag{6.46}$$

**Definition 6.1.12.** ***(Pearson correlation matrix)***
*Given a random vector $\boldsymbol{X} = (X_1, ..., X_n)^T$ we define the correlation matrix:*

$$\boldsymbol{R}_{ij} := \mathrm{Corr}[\boldsymbol{X}]_{ij} = \frac{\mathrm{Cov}[X_i, X_j]}{\sqrt{\mathrm{Var}[X_i]}\sqrt{\mathrm{Var}[X_j]}}. \tag{6.47}$$

*which we can write as:*

$$\boldsymbol{R} = \boldsymbol{K}^{-\frac{1}{2}}\boldsymbol{\Sigma}\boldsymbol{K}^{-\frac{1}{2}}, \tag{6.48}$$

*where $\boldsymbol{K} = diag(\sqrt{\mathrm{Var}[X_1]}, ..., \sqrt{\mathrm{Var}[X_n]})$.*

## 6.2 Information Theory

### 6.2.1 Basics of Information Theory

In this section we introduce the concept of entropy, due to Shannon [Sha48], which intuitively measures the uncertainty on the outcome of a random variable. We follow again [Bis06], [Mur22], particularly chapter 6, and also [CT06].

### 6.2.2 Entropy of Discrete Random Variables

Given a discrete random variable $X$ with $n$ possible outcomes $\{x_1, ..., x_n\}$, with probability $P(X = x_i) = p_i$, $i = 1, ..., n$, following [Sha48] we require that the entropy $H : [0, 1]^n \to \mathbb{R}$, $(p_1, ..., p_n) \mapsto H(p_1, ..., p_n)$, have the following properties:

- $H$ is continuous in the $p_i$, $i = 1, ..., n$.
- If $p_i = \frac{1}{n}$, for $i = 1, ..., n$, or equivalently $X \sim \text{Uni}(1, n)$, we require that $H$ be a monotonic increasing function of $n$.
- If a choice is broken down into two successive choices we require that the entropy of the original choice be the weighted sum of the individual values of the entropy for the two successive choices. See the example in property 3, section 6, of [Sha48].

**Theorem 6.2.1.** *The only function $H$ satisfying the three requirements above has the form:*

$$H(p_1, ..., p_n) = -K \sum_{i=1}^{n} p_i \log p_i \tag{6.49}$$

*where $K \in \mathbb{R}_{>0}$.*

*Proof.* See appendix 2 of [Sha48]. ■

**Remark 6.2.1.** *The choice on the base of the logarithm in the proof is arbitrary. Typically one considers base 2, base 10 or base $e$. For each case the entropy is named bits, dits and nats respectively.*

**Definition 6.2.1.** ***(Entropy of a discrete random variable)***
*The entropy of a discrete random variable variable $X$ with $n$ possible outcomes $\{x_1, ..., x_n\}$, with probability $P(X = x_i) = p_i$, $i = 1, ..., n$, $\sum_i P(X = x_i) = 1$ is defined as:*

$$H(X) := -\sum_{i=1}^{n} P(X = x_i) \log(P(X = x_i)) = -\mathbb{E}[\log(P(X = x))] \tag{6.50}$$

**Remark 6.2.2.** *If we consider an event $X = x_i$ for which $P(X = x_i) = 0$ then we take $P(X = x_i) \log\big(P(X = x_i)\big) = 0$, from a continuity argument since $\lim_{p \to 0^+} p \log p = 0$, by using L'Hôpital's rule for example.*

**Proposition 6.2.2.** *Properties of the entropy $H$.*

1. *By definition $H \geq 0$.*
2. *$H = 0$ if and only if all but one of the $p_i \neq 0$, and thus $p_i = 1$. The entropy vanishes when there is no uncertainty in regards to the outcome of a discrete random variable.*
3. *$H$ is maximum for a discrete uniform distribution, $H(X) = -\sum_{i=1}^{n} \frac{1}{n} \log(1/n) = \log(n)$. This reflects the situation where we are most uncertain over the outcome.*

*Proof.* We prove only the last statement (3) by using Jensen's inequality. Consider the concave function $h(y) = -y\log(y)$ in the interval $y \in [0,1]$. Jensen's inequality states that:

$$f\left(\frac{\sum_{i=1}^n a_i y_i}{\sum_{i=1}^n a_i}\right) \geq \frac{\sum_{i=1}^n a_i f(y_i)}{\sum_{i=1}^n a_i}, \tag{6.51}$$

for a concave function $f$ and $a_i \geq 0$ such that $\sum_{i=1}^n a_i = 1$. Taking $a_i = \frac{1}{n}$ we have then:

$$f\left(\frac{\sum_{i=1}^n y_i}{n}\right) \geq \frac{\sum_{i=1}^n f(y_i)}{n}. \tag{6.52}$$

By considering $y_i = p(x_i)$ and $f$ to be $h$ we obtain:

$$\begin{aligned} -n\left(\frac{\sum_{i=1}^n p(x_i)}{n}\right)\log\left(\frac{\sum_{i=1}^n p(x_i)}{n}\right) &\geq -\sum_{i=1}^n p(x_i)\log(p(x_i)) \\ \Leftrightarrow \quad \log(n) &\geq -\sum_{i=1}^n p(x_i)\log(p(x_i)). \end{aligned} \tag{6.53}$$

Where we use the fact that $\sum_{i=1}^n p(x_i) = 1$. We see that, in the case of a discrete distribution, the entropy is bounded by $\log(n)$ and we get equality precisely when $p$ is the discrete uniform distribution, i.e.:

$$-\sum_{i=1}^n \frac{1}{n}\log\left(\frac{1}{n}\right) = \frac{1}{n}(n\log(n)) = \log(n). \tag{6.54}$$

■

**Definition 6.2.2.** ***(Joint entropy of a discrete random vector)***
*The entropy of a discrete random vector $\boldsymbol{X} = (X_1, X_2, ..., X_n)$, with sample space $\Omega = \bigtimes_{i=1}^n \Omega_i$ and joint probability distribution $p(\boldsymbol{X} = \boldsymbol{x})$ is defined as:*

$$H(\boldsymbol{X}) := -\sum_{\boldsymbol{x}\in\Omega} p(\boldsymbol{X} = \boldsymbol{x})\log\big(p(\boldsymbol{X} = \boldsymbol{x})\big). \tag{6.55}$$

**Definition 6.2.3.** ***(Conditional entropy)***
*The conditional entropy of a discrete random variable $X$, given another discrete random variable $Y$, with joint probability function $p(X = x, Y = y)$, with sample space $\Omega_X \times \Omega_Y$ is defined by*

$$H(X|Y) = -\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X = x, Y = y)\log\big(p(X = x|Y = y)\big). \tag{6.56}$$

**Remark 6.2.3.** *Note that this is different from the entropy of a conditional discrete random variable $H(X|Y = y) = -\sum_{x\in\Omega_X} p(X = x|Y = y)\log\big(p(X = x|Y = y)\big)$. In particular, $H(X|Y) = \sum_{y\in\Omega_Y} H(X|Y = y)p(Y = y)$.*

**Proposition 6.2.3.** ***(Relation of joint and conditional entropy)***
*Let $X, Y$ be discrete random variables, with sample spaces $\Omega_X, \Omega_Y$ respectively, and joint probability function $p(X, Y)$. We have the following relation:*

$$H(Y|X) = H(X, Y) - H(X). \tag{6.57}$$

*Proof.*

$$\begin{aligned} H(Y|X) &:= -\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\log\big(p(Y=y|X=x)\big) \\ &= -\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\log\left(\frac{p(X=x,Y=y)}{p(X=x)}\right) \\ &= -\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\big(\log p(X=x,Y=y)-\log p(X=x)\big) \\ &= H(X,Y)+\sum_{x\in\Omega_X}\log p(X=x)\sum_{y\in\Omega_Y} p(X=x,Y=y) \\ &= H(X,Y)+\sum_{x\in\Omega_X} p(X=x)\log p(X=x) \\ &= H(X,Y)-H(X). \end{aligned} \tag{6.58}$$

■

**Proposition 6.2.4.** *Let $X, Y$ be two discrete random variables. If $Y$ is completely specified by $X$, i.e. $Y = f(X)$, where $f$ is a deterministic and bijective, then:*

$$H(Y|X) = 0. \tag{6.59}$$

*Knowing $X$ implies knowledge of $Y$ so there is no uncertainty over $Y$.*

*Proof.* Suppose $Y = f(X)$ then, $p(Y = y) = p(Y = f(x))$ for some $x \in \Omega_X$ and we can write:

$$\begin{aligned} H(X,Y) &= -\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\log p(X=x,Y=y) \\ &= -\sum_{x\in\Omega_X} p(X=x,Y=f(x))\log p(X=x,Y=f(x)) \\ &= H(X), \end{aligned} \tag{6.60}$$

since $p(X = x, Y = f(x)) = p(X = x)$. By the previous proposition we have the desired result. ■

**Proposition 6.2.5.** *If $Y$ and $X$ are independent discrete random variables then:*

$$H(Y|X) = H(Y). \tag{6.61}$$

*Having information of $X$ does not change the information we have about $Y$ if $X$ and $Y$ are independent.*

*Proof.* From the previous proof and using the independence of $X$ and $Y$ we can write:

$$\begin{aligned} H(Y|X) &= -\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x)p(Y=y)\big(\log(p(X=x)p(Y=y))-\log p(X=x)\big) \\ &= -\sum_{y\in\Omega_Y} p(Y=y)\log p(Y=y)\sum_{x\in\Omega_X} p(X=x) \\ &= H(Y). \end{aligned} \tag{6.62}$$

■

**Proposition 6.2.6.** *Given discrete random variables $X$ and $Y$ we have:*

$$H(X,Y) \leq H(X) + H(Y). \tag{6.63}$$

*Proof.*

$$\begin{aligned} H(X,Y) - H(X) - H(Y) = &- \sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\log p(X=x,Y=y) + \\ &+ \sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(Y=y,X=x)\log p(X=x) \\ &+ \sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\log p(Y=y) \\ = &\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\log\left(\frac{p(X=x)p(Y=y)}{p(X=x,Y=y)}\right) \\ \leq &\log\left(\sum_{x\in\Omega_X}\sum_{y\in\Omega_Y} p(X=x,Y=y)\frac{p(X=x)p(Y=y)}{p(X=x,Y=y)}\right) \\ = &\log(1) = 0. \end{aligned} \tag{6.64}$$

■

Together with proposition 6.2.3 we see that $H(Y|X) \leq H(Y)$ and, by proposition 6.2.5, the equality holds precisely when $X$ and $Y$ are independent. Thus, conditioning on a random variable never increases the uncertainty, which makes intuitive sense.

It is useful to have in mind the information diagram of the following figure 6.1 to better visualize the relation between the entropies defined above. The $\mathbb{I}(X,Y)$ term is known as mutual information and will be defined shortly.

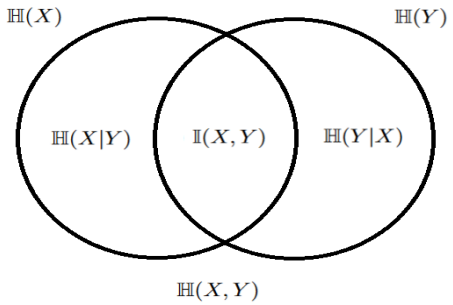


Figure 6.1: Venn diagram relating $H(X,Y), H(X|Y), H(Y,X), H(X), H(Y)$ and $\mathbb{I}(X,Y)$.

Another important concept in information theory is that of relative entropy. As stated in [Bis06], the underlying idea is related to coding schemes but essentially the goal is to measure the *increase* in entropy when approximating a probability mass function (pmf) $p(x)$ with another $q(x)$ or, to have a notion of distance metric between distributions (although it is not an actual metric since it is not symmetric and fails to satisfy the triangle inequality).

**Definition 6.2.4.** ***(Kullback-Leibler divergence or relative entropy)***
*The Kullback-Leibler divergence, or relative entropy, between two discrete pmfs $p(x)$, $q(x)$, on the same probability space with sample space $\Omega_X$, is defined as:*

$$\begin{aligned} \mathbb{KL}(p||q) :&= -\sum_{x\in\Omega_X} p(x)\log\big(q(x)\big) - \left(-\sum_{x\in\Omega_X} p(x)\log\big(p(x)\big)\right) \\ &= \sum_{x\in\Omega_X} p(x)\log\left(\frac{p(x)}{q(x)}\right). \end{aligned} \tag{6.65}$$

**Theorem 6.2.7.** ***(Information inequality)***
*Let $X$ be a random variable with two probability mass functions $p(x), q(x)$ on $x \in \Omega_X$. Then, $\mathbb{KL}(p||q) \geq 0$ with equality if and only if $p(x) = q(x)$ for all $x$.*

*Proof.* We follow the proof from [CT06]. Let $A \subseteq \Omega_X$ be the support of $p$, i.e. $A = \{x \in \Omega_X : p(x) > 0\}$. We have:

$$\begin{aligned} -\mathbb{KL}(p||q) &= -\sum_{x\in A} p(x) \log\left(\frac{p(x)}{q(x)}\right) = \sum_{x\in A} p(x) \log\left(\frac{q(x)}{p(x)}\right) \\ &\leq \log\left(\sum_{x\in A} p(x)\frac{q(x)}{p(x)}\right) \\ &= \log\left(\sum_{x\in A} q(x)\right) \\ &\leq \log\left(\sum_{x\in \Omega_X} q(x)\right) \\ &= \log(1) \\ &= 0. \end{aligned} \tag{6.66}$$

Where we used Jensen's inequality on the first inequality (see the proof of 6.2.2), this shows the first part. For the equality part, since the log is strictly concave, on the first inequality of equation (6.66), the equality holds if and only if $q(x)/p(x) = c$ for all $x \in A$, for some constant $c \in [0, 1]$.

$$\frac{\sum_{x\in A} q(x)}{\sum_{x\in A} p(x)} = \sum_{x\in A} q(x) = c, \tag{6.67}$$

where $c$ measures the fraction of the support of $q$ that is in $A$. The last inequality holds if and only if $\sum_{x\in A} q(x) = \sum_{x\in\Omega_X} q(x)$, i.e. the support of $q$ is contained in $A$ and $c = 1$. This implies then that $p(x) = q(x)$ for all $x \in \Omega_X$ and we obtain the desired result. ■

We define now the notion of mutual information mentioned above.

**Definition 6.2.5.** ***(Mutual information $\mathbb{I}$)***
*Let $X, Y$ be discrete random variables, with sample spaces $\Omega_X, \Omega_Y$ respectively, and joint probability function $p(X, Y)$. The mutual information $\mathbb{I}(X, Y)$ is defined as:*

$$\mathbb{I}(X, Y) := \mathbb{KL}\big(p(X, Y)||p(X)p(Y)\big) = \sum_{y\in\Omega_Y}\sum_{x\in\Omega_X} p(x, y) \log\frac{p(x, y)}{p(x)p(y)}. \tag{6.68}$$

Being defined in terms of the KL-divergence we see that $\mathbb{I}(X, Y)$ measures the difference when approximating the joint distribution $p(X, Y)$ by a distribution which takes $X$ and $Y$ to be independent variables, i.e. $p(X)p(Y)$.

### 6.2.3 Entropy of Continuous Random Variables

In this section we present similar results to the previous one, in the setting of continuous random variables. An important distinction in the continuous setting, unlike the discrete case, is that we may have $h(X) < 0$. Thus, the concept of differential entropy does not lend itself naively to the same interpretation of entropy as in the discrete case.

**Definition 6.2.6.** *(**Entropy of a continuous random variable**)*
*The entropy of a continuous random variable $X$ with probability density function (pdf) $p$ is defined as:*

$$h(X) := -\int_{\infty}^{\infty} p(x)\log\big(p(x)\big)dx. \tag{6.69}$$

**Example 6.2.1.** *To exemplify the starting comment, consider the uniformly distributed random variable $X \sim Uni(0,a)$ with $a > 0$. The differential entropy is given by:*

$$h(x) = -\int_0^a \frac{1}{a}\log\left(\frac{1}{a}\right)dx = \log(a). \tag{6.70}$$

*Thus, having $h(x) > 0$, $h(X) = 0$ or $h(X) < 0$ depends on the value of $a$. And it is certainly not the case that we have no uncertainty over $X$ when $a = 1$ which implies $h(X) = 0$.*

**Definition 6.2.7.** *(**Entropy of a continuous random vector**)*
*The entropy of a continuous random vector $\boldsymbol{X} = (X_1, X_2, ..., X_n)$ with joint pdf $p$ is defined as*

$$h(\boldsymbol{X}) := -\int_{\mathbb{R}^n} P(\boldsymbol{X} = \boldsymbol{x})\log\big(P(\boldsymbol{X} = \boldsymbol{x})\big)d\boldsymbol{x}. \tag{6.71}$$

**Definition 6.2.8.** *(**Conditional entropy**)*
*The conditional entropy of a continuous random variable $X$ given another continuous random variable $Y$, with joint pdf $p(X,Y)$, is defined as:*

$$h(X|Y) = -\int_{\mathbb{R}^2} p(X = x, Y = y)\log\big(p(X = x|Y = y)\big)dxdy. \tag{6.72}$$

**Definition 6.2.9.** ***Kullback-Leibler divergence (relative entropy)***
*The Kullback-Leibler divergence or relative entropy between the pdfs $p(x)$ and $q(x)$, on the same probability space, is defined by:*

$$\begin{aligned}\mathbb{KL}(p||q) &:= -\int_{\mathbb{R}} p(X = x)\log\big(q(X = x)\big)dx - \left(-\int_{\mathbb{R}} p(X = x)\log\big(p(X = x)\big)\right)dx \\ &= -\int_{\mathbb{R}} p(X = x)\log\left(\frac{q(X = x)}{p(X = x)}\right)dx.\end{aligned} \tag{6.73}$$

**Proposition 6.2.8.** *The Kullback-Leibler divergence satisfies $\mathbb{KL}(p||q) \geq 0$ with equality if and only if $p = q$ almost everywhere (a.e.).*

*Proof.* The proof is similar to the discrete case, see theorem 8.6.1 of [CT06]. ■

Suppose we have data generated by an unknown distribution $p(x)$ and we approximate it via a parameterized distribution $q(x|w)$. One way of doing this is by minimizing the Kullback-Leibler divergence between $p$ and $q$. Of course, since $p$ is unknown, we can not use the definition directly but we can consider an approximation via the training data $\{(x_1, p_1), ..., (x_N, p_N)\}$. We have then:

$$\mathbb{E}[\mathbb{KL}(p||q)] \simeq \frac{1}{N}\sum_{n=1}^{N}\mathbb{KL}(p(x_n)||q(x_n)) = \frac{1}{N}\sum_{n=1}^{N}\log p(x_n) - \log q(x_n|w). \tag{6.74}$$

Thus, we have the following optimization problem:

$$\begin{aligned}&\arg\min_w \; -\frac{1}{N}\sum_{n=1}^{N}\log q(x_n|w) \\ &= \arg\max_w \; \frac{1}{N}\log\left(\prod_{n=1}^{N} q(x_n|w)\right) \\ &= \arg\max_w \; \prod_{n=1}^{N} q(x_n|w).\end{aligned} \tag{6.75}$$

This shows the equivalence, in this setting, of minimizing the Kullback-Leibler divergence and maximizing the likelihood function.

Another example where one can see the usefulness of information theory is in feature selection. By using the concept of entropy one can devise several methods for detecting the relevance and redundancy between different features when performing classification (see the notes *Information Theory and Filter Feature Selection* by Prof. M. Rosário Oliveira).

We shall see in the next section that the differential entropy is maximized precisely when we have a Gaussian distribution, Proposition 6.3.4.

## 6.3 Gaussian Distribution

In this section we introduce and explore several properties of the Gaussian distribution in both the univariate and multivariate cases. The results obtained here will be useful throughout most other topics, particularly Propositions 6.3.13 and 6.3.14 which will be required to obtain analytically tractable models when performing Bayesian linear regression in Section 6.3.8.

### 6.3.1 Univariate Case

**Definition 6.3.1.** ***(Univariate normal (or Gaussian) distribution)***
*The Gaussian distribution over a continuous single real-valued variable $x$ has the following expression:*

$$\mathcal{N}(x|\mu,\sigma^2) := \frac{1}{\sqrt{2\pi\sigma^2}}\exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big). \tag{6.76}$$

The distribution is specified by two parameters, the mean $\mu \in \mathbb{R}$ and the variance $\sigma^2 \in \mathbb{R}_{>0}$. It is easily seen to satisfy the necessary conditions for a probability density, i.e.:

**Proposition 6.3.1.** *The normal distribution satisfies the following two conditions:*

$$\mathcal{N}(x|\mu,\sigma^2) > 0,\ \forall x \in \mathbb{R}, \tag{6.77}$$

$$\int_{-\infty}^{\infty} \mathcal{N}(x|\mu,\sigma^2)dx = 1. \tag{6.78}$$

*Proof.* It is clear that $\mathcal{N}(x|\mu,\sigma^2) > 0$. Consider now the following integral:

$$I = \int_{-\infty}^{\infty} \exp\Big(-\frac{x^2}{2\sigma^2}\Big)dx. \tag{6.79}$$

The square of which is given by:

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp\Big(-\frac{x^2}{2\sigma^2}-\frac{y^2}{2\sigma^2}\Big)dxdy. \tag{6.80}$$

Changing to polar coordinates, i.e. $(x,y) = (r\cos\theta, r\sin\theta)$ with Jacobian determinant $|\boldsymbol{J}| = r$, we obtain:

$$\begin{aligned} I^2 &= \int_0^{2\pi}\int_0^{\infty} \exp\Big(-\frac{r^2}{2\sigma^2}\Big)rdrd\theta \\ &= 2\pi\big(-\sigma^2\big)\int_0^{\infty}\Big(-\frac{r^2}{2\sigma^2}\Big)\Big(-\frac{r}{\sigma^2}\Big)dr \\ &= -2\pi\sigma^2\exp\Big(-\frac{r^2}{2\sigma^2}\Big)\Big|_0^{\infty} \\ &= 2\pi\sigma^2. \end{aligned} \tag{6.81}$$

And we conclude $I = \sqrt{2\pi\sigma^2}$ which, by the change of variable $y = x - \mu$, implies:

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{(x-\mu)^2}{2\sigma^2} \right) dx = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{y^2}{2\sigma^2} \right) dy = \frac{I}{\sqrt{2\pi\sigma^2}} = 1. \tag{6.82}$$

■

The mean and variance of a random variable $X \sim \mathcal{N}(x|\mu, \sigma^2)$ can be computed via:

**Proposition 6.3.2.** ***(Mean and variance of a Gaussian distribution)***

$$\mathbb{E}[X] := \int_{-\infty}^{\infty} x\mathcal{N}(x|\mu, \sigma^2)dx = \mu, \tag{6.83}$$

$$\mathbb{E}[X^2] := \int_{-\infty}^{\infty} x^2\mathcal{N}(x|\mu, \sigma^2)dx = \mu^2 + \sigma^2, \tag{6.84}$$

$$\mathrm{Var}[X] := \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \sigma^2. \tag{6.85}$$

*Proof.* We want to compute:

$$\mathbb{E}[X] := \int_{-\infty}^{\infty} \frac{x}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{(x-\mu)^2}{2\sigma^2} \right) dx. \tag{6.86}$$

Changing to the variable $y = x - \mu$ we obtain:

$$\begin{aligned} \mathbb{E}[X] &= \int_{-\infty}^{\infty} \frac{(y+\mu)}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{y^2}{2\sigma^2} \right) dy \\ &= \int_{-\infty}^{\infty} \frac{y}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{y^2}{2\sigma^2} \right) dy \; + \mu \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{y^2}{2\sigma^2} \right) dy \\ &= 0 + \mu \cdot 1 \\ &= \mu \end{aligned} \tag{6.87}$$

Where the first integral is zero since the integrand is an odd function and the second is obtained via the normalization condition.

To compute $\mathbb{E}[X^2]$ we differentiate the normalization condition (eq. 6.78) with respect to $\sigma^2$, i.e.:

$$\begin{aligned} & \frac{d}{d\sigma^2}\left( \int_{-\infty}^{\infty} \exp\left( - \frac{(x-\mu)^2}{2\sigma^2} \right) dx \right) = \frac{d}{d\sigma^2}\left( \sqrt{2\pi\sigma^2} \right) \\ \Leftrightarrow \quad & \int_{-\infty}^{\infty} \left( \frac{(x-\mu)^2}{2\sigma^4} \right) \exp\left( - \frac{(x-\mu)^2}{2\sigma^2} \right) dx = \frac{\sqrt{2\pi}}{2\sqrt{\sigma^2}} \\ \Leftrightarrow \quad & \int_{-\infty}^{\infty} \frac{(x-\mu)^2}{\sqrt{2\pi\sigma^2}} \exp\left( - \frac{(x-\mu)^2}{2\sigma^2} \right) dx = \sigma^2 \\ \Leftrightarrow \quad & \mathbb{E}[(x-\mu)^2] = \sigma^2. \end{aligned} \tag{6.88}$$

This shows that $\mathrm{Var}[X] = \sigma^2$ and, since:

$$\mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - 2\mu\mathbb{E}[X] + \mu^2 = \mathbb{E}[X^2] - \mu^2, \tag{6.89}$$

one finally obtains $\mathbb{E}[X^2] = \sigma^2 + \mu^2$. ■

The plot of the univariate Gaussian can be seen in the following figure (fig. 6.2) for different values of the mean and variance.

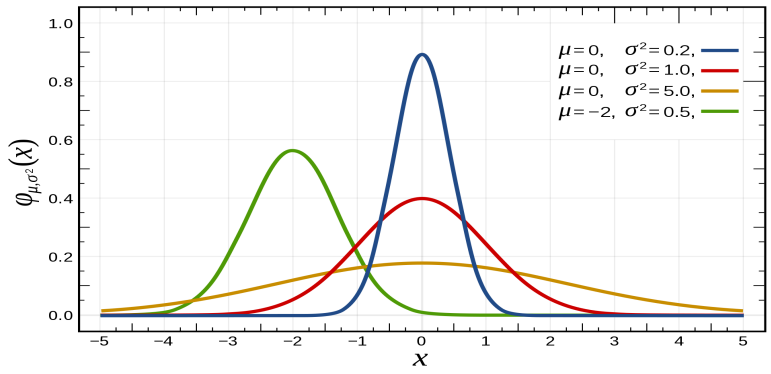


Figure 6.2: $\mathcal{N}(x|\mu, \sigma^2)$ for different values of $\mu$ and $\sigma^2$. Source: Wikipedia.

**Proposition 6.3.3.** ***(Unimodal distribution and mode)***
*The normal distribution is unimodal with mode equal to the mean.*

*Proof.* We just need to compute $\frac{d}{dx}\mathcal{N}(x|\mu, \sigma^2) = 0$ in order to obtain the maximum. However, since the $\ln()$ function is monotonic we can just as well maximize $\ln\big(\mathcal{N}(x|\mu, \sigma^2)\big)$ which is equivalent to solving $(x-\mu)^2 = 0$. Thus, we obtain $x = \mu$ as the only solution, i.e. it is unimodal with mode equal to the mean. ■

**Proposition 6.3.4.** ***(Maximum entropy distribution)***
*The maximum entropy configuration for a random variable $X$, with finite mean $\mu$ and finite variance $\sigma^2$, is given by the Gaussian distribution. In particular, the differential entropy of the Gaussian is given by:*

$$h[X] = \frac{1}{2}\big(1 + \ln(2\pi\sigma^2)\big). \tag{6.90}$$

*Proof.* We formulate this as the following functional optimization problem:

$$\begin{aligned} \max_{p(x)} \quad & h[p(x)] := \int_{-\infty}^{\infty} p(x)\ln(p(x)), \\ \text{s.t.} \quad & \int_{-\infty}^{\infty} p(x)dx = 1, \\ & \int_{-\infty}^{\infty} xp(x)dx = \mu, \\ & \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \sigma^2. \end{aligned} \tag{6.91}$$

Since all constraints are equalities we can use the method of Lagrange multipliers with Lagrangian given by:

$$\begin{aligned} L(p(x), \lambda_1, \lambda_2, \lambda_3) = &-\int_{-\infty}^{\infty} p(x)\ln(p(x)) + \lambda_1\bigg(\int_{-\infty}^{\infty} p(x)dx - 1\bigg) + \lambda_2\bigg(\int_{-\infty}^{\infty} xp(x)dx - \mu\bigg) + \\ &+\lambda_3\bigg(\int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx - \sigma^2\bigg). \end{aligned} \tag{6.92}$$

We now need to look at the stationary points of $L$ with respect to $p(x)$ and the $\lambda_i$, $i = 1, 2, 3$. To find the stationary point with respect to $p(x)$ we use methods from calculus of variations (see appendix D

in [Bis06]). Finding the stationary points of the functional derivative $\frac{\delta h}{\delta p(x)}$ is equivalent to finding the stationary points of $\frac{\partial F}{\partial p(x)}$ where $F = -p(x)\ln(p(x)) + \lambda_1 p(x) + \lambda_2 x p(x) + \lambda_3(x-\mu)^2 p(x)$. We have then:

$$\begin{aligned} \frac{\partial F}{\partial p(x)} &= -\ln(p(x)) - 1 + \lambda_1 + x\lambda_2 + \lambda_3(x-\mu)^2 = 0 \\ \Leftrightarrow\ p(x) &= \exp\big(-1 + \lambda_1 + x\lambda_2 + \lambda_3(x-\mu)^2\big) \end{aligned} \tag{6.93}$$

To easily identify the coefficients we compare the following two expressions:

$$\int_{-\infty}^{\infty} \exp\big(-1 + \lambda_1 + x\lambda_2 + \lambda_3(x-\mu)^2\big)dx = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)dx. \tag{6.94}$$

This allows us to read of the values:

$$\begin{aligned} \exp(\lambda_1 - 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \Leftrightarrow \lambda_1 &= 1 - \frac{1}{2}\ln(2\pi\sigma^2), \\ \lambda_2 &= 0, \\ \lambda_3 &= -\frac{1}{2\sigma^2}. \end{aligned} \tag{6.95}$$

Thus, all constraints are satisfied and we finally have:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{6.96}$$

To obtain the entropy we just need to perform the following computation:

$$\begin{aligned} &-\int_{-\infty}^{\infty} p(x)\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\right)dx \\ =& -\int_{-\infty}^{\infty} p(x)\left(-\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}\right)dx \\ =& \frac{1}{2}\ln(2\pi\sigma^2) + \frac{1}{2}. \end{aligned} \tag{6.97}$$

■

A crucial result, which justifies the particular importance of the normal distribution, is the following theorem:

**Theorem 6.3.5.** ***(Central limit theorem (Lindeberg–Lévy))***
*Let $\{X_1, X_2, ..., X_n, ...\}$ be i.i.d random variables with mean $\mu$ and variance $\sigma^2 < \infty$. Then,*

$$\lim_{n\to\infty} \sqrt{n}(\bar{X}_n - \mu) \to \mathcal{N}(0, \sigma^2), \tag{6.98}$$

*in distribution, where $\bar{X}_n = (X_1 + ... + X_n)/n$.*

*Proof.* See for example theorem 8.1 in chapter 3 of [Çın11]. ■

### 6.3.2 Multivariate Case

We now extend the previous results to the multivariate case and explore interesting behaviours and geometrical properties which arise in the case of larger dimensions.

**Definition 6.3.2.** *(**Multivariate normal distribution**)*
*For a D-dimensional vector* $\mathbf{x} = (x_1, ..., x_D)$*, we have:*

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left( -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\boldsymbol{T}} \boldsymbol{\Sigma}^{-\mathbf{1}} (\mathbf{x} - \boldsymbol{\mu}) \right), \tag{6.99}$$

*where* $\boldsymbol{\mu}$ *is the mean vector and* $\boldsymbol{\Sigma}$ *is a* $D \times D$ *matrix called the covariance matrix.*

**Proposition 6.3.6.** *The multivariate normal distribution satisfies the following two conditions:*

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) > 0, \ \forall \mathbf{x} \in \mathbb{R}^D, \tag{6.100}$$

$$\int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x} = 1. \tag{6.101}$$

**Proposition 6.3.7.** *(**Mean and variance**)*
*Given a random vector* $\boldsymbol{X} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ *we can compute:*

$$\mathbb{E}[\mathbf{x}] = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} \mathbf{x} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x} = \boldsymbol{\mu}, \tag{6.102}$$

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} \mathbf{x}\mathbf{x}^T \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x} = \boldsymbol{\mu}\boldsymbol{\mu}^T + \boldsymbol{\Sigma}, \tag{6.103}$$

$$\mathrm{Cov}[\mathbf{x}] = \boldsymbol{\Sigma}. \tag{6.104}$$

We shall prove this results soon, after looking at an appropriate change of coordinates in section 6.3.3. The plot of the multivariate Gaussian, in 2 dimensions, can be seen in the following figure (fig. 6.3).

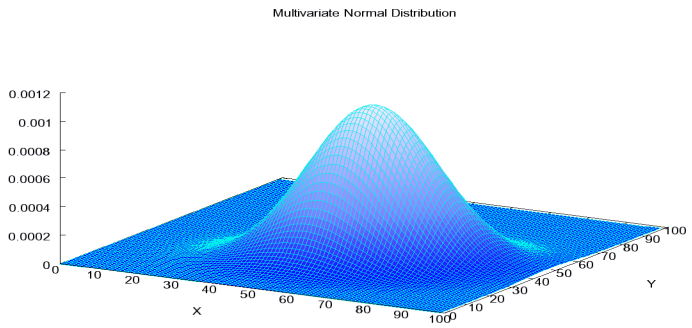


Figure 6.3: $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in 2D. Source: Wikipedia.

**Proposition 6.3.8.** *(**Unimodal distribution and mode**)*
*The multivariate normal distribution is unimodal with mode equal to the mean.*

*Proof.* Similar to 6.3.3. ■

**Proposition 6.3.9.** *(**Maximum Entropy**)*
*The maximum entropy configuration for a multivariate random variable* $\boldsymbol{X}$*, with mean* $\boldsymbol{\mu}$ *and finite variance* $\boldsymbol{\Sigma}$*, is given by the Gaussian distribution. In particular, the differential entropy of the Gaussian is given by:*

$$h[\boldsymbol{X}] = \frac{1}{2} \ln |\Sigma| + \frac{D}{2}(1 + \ln(2\pi)) \tag{6.105}$$

*Proof.* This is an extension of the previous results, see exercises 2.14 and 2.15 of [Bis06]. ■

**Theorem 6.3.10.** ***(Multivariate central limit theorem (Lindeberg–Lévy))*** *Let $\{\boldsymbol{X}_1, .., \boldsymbol{X}_n, ...\}$ be a sequence of i.i.d. $\mathbb{R}^K-$valued random vectors each with $\mathbb{E}[\boldsymbol{X}_i] = \boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Then,*

$$\lim_{n\to\infty} \sqrt{n}(\bar{\boldsymbol{X}}_n - \boldsymbol{\mu}) \to \mathcal{N}_K(\mathbf{0}, \boldsymbol{\Sigma}), \tag{6.106}$$

*in distribution where $\bar{\boldsymbol{X}}_n = (\boldsymbol{X}_1 + ... + \boldsymbol{X}_n)/n$.*

### 6.3.3 Geometrical Properties

For the multivariate Gaussian, the dependence on $\mathbf{x}$ comes from the following term:

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}), \tag{6.107}$$

where $\Delta$ is known as the Mahalanobis distance which, in the particular case where $\Sigma = I$, is just the Euclidean distance.

**Proposition 6.3.11.** *The matrix $\Sigma$ can be taken to be symmetric without loss of generality.*

*Proof.* Any square matrix $A$ can be written as $A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T)$, where the first term $A_s = A + A^T$ is a symmetric matrix while the second $A_a = A + A^T$ is antisymmetric. Thus, when we consider the square of the Mahalanobis distance on the antisymmetric part we get:

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T A_a (\mathbf{x} - \boldsymbol{\mu}) = \sum_{i,j}^{D} (A_a)_{ij} (x_j - \mu_j)(x_i - \mu_i). \tag{6.108}$$

Since $(A_a)_{ii} = 0$ and $(A_a)_{ij} = -(A_a)_{ji}$ the sum is zero. We conclude then that $\Sigma^{-1}$ can be taken to be symmetric. Furthermore, we have $A^T(A^{-1})^T = (A^{-1}A)^T = I$ and $(A^{-1})^T A^T = (AA^{-1})^T = I$, i.e. $(A^{-1})^T = (A^T)^{-1}$ and thus we conclude $\boldsymbol{\Sigma}^{-1} = (\boldsymbol{\Sigma}^{-1})^T \implies \boldsymbol{\Sigma} = \boldsymbol{\Sigma}^T$. ■

Any real symmetric matrix $S$ can be decomposed as $S = Q\Lambda Q^T$, where $Q$ is an orthogonal matrix with columns the eigenvectors of $S$ and where $\Lambda$ is a diagonal matrix with the eigenvalues of $S$. We can write then $\boldsymbol{\Sigma} = \sum_{i=1}^{D} \lambda_i \boldsymbol{q_i} \boldsymbol{q_i}^T$ and $\boldsymbol{\Sigma}^{-1} = \sum_{i=1}^{D} \frac{1}{\lambda_i} \boldsymbol{q_i} \boldsymbol{q_i}^T$, considering that all $\lambda_i > 0$ (i.e. $\boldsymbol{\Sigma}$ to be positive-definite). Replacing this in equation (6.107) we obtain:

$$\Delta^2 = \sum_{i=1}^{D} \frac{1}{\lambda_i} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{q_i} \boldsymbol{q_i}^T (\mathbf{x} - \boldsymbol{\mu}) := \sum_{i=1}^{D} \frac{y_i^2}{\lambda_i} \tag{6.109}$$

We can consider a new coordinate system given by the $\{y_i\}$ where we define:

$$\mathbf{y} = (y_1, ..., y_D)^T := \boldsymbol{Q}^T (\mathbf{x} - \boldsymbol{\mu}), \tag{6.110}$$

where the rows of $\boldsymbol{Q}^T$ are the $q_i^T$. These new coordinates are seen to be shifted and rotated versions of the original $x_i$.

The Gaussian density is constant whenever $\Delta^2 = c$, for some $c \in \mathbb{R}_{\geq 0}$. Again, considering all eigenvalues $\lambda_i > 0$, these surfaces represent ellipsoids with center $\boldsymbol{\mu}$ and axes along the $\boldsymbol{q_i}$, scaled by $\lambda_i^{1/2}$. The green slice in the next figure illustrates the idea:

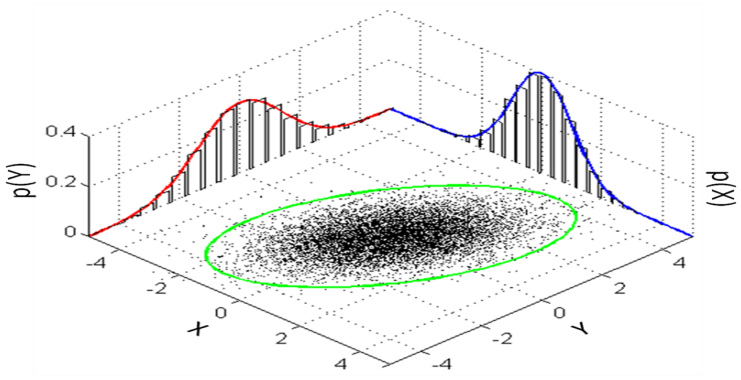


Figure 6.4: Region of constant probability density (ellipses in green) in 2D. Source: Wikipedia.

With this change of coordinates the entries of the Jacobian matrix are given by $J_{ij} = \frac{\partial x_i}{\partial y_j} = \frac{\partial (u_i^T \mathbf{y})}{\partial y_j} = Q_{ij}^T = Q_{ji}$. Thus, $|\boldsymbol{J}|^2 = |\boldsymbol{Q}^T|^2 = |\boldsymbol{Q}^T||\boldsymbol{Q}^T| = |\boldsymbol{Q}^T||\boldsymbol{Q}| = |\boldsymbol{Q}^T\boldsymbol{Q}| = |\boldsymbol{I}| = 1$. Since $|\Sigma| = \prod_{i=1}^D \lambda_i$ we can write:

$$p(\mathbf{y}) = p(\mathbf{x})|\boldsymbol{J}| = \prod_{i=1}^{D} \frac{1}{(2\pi\lambda_i)^{1/2}} \exp\left( -\frac{y_i^2}{2\lambda_i} \right), \tag{6.111}$$

which is the product of $D$ independent univariate Gaussian distributions. Over these new coordinates we can easily prove proposition 6.3.6 and 6.3.7.

*Proof.* (**Proof of proposition (6.3.6)**)

$$\int_{-\infty}^{\infty} p(\mathbf{y}) d\mathbf{y} = \prod_{i=1}^{D} \int_{-\infty}^{\infty} \frac{1}{(2\pi\lambda_i)^{1/2}} \exp\left( -\frac{y_i^2}{2\lambda_i} \right) dy_i = 1, \tag{6.112}$$

by using the result of proposition 6.3.1. ■

*Proof.* (**Proof of proposition (6.3.7)**)
To compute the first moment of $\mathbf{X}$ we write:

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \int \exp\left( -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^{\boldsymbol{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \mathbf{x} d\mathbf{x} \\ &= \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \int \exp\left( -\frac{1}{2} \mathbf{z}^T \boldsymbol{\Sigma}^{-1} \mathbf{z} \right) (\mathbf{z} + \boldsymbol{\mu}) d\mathbf{z} = \boldsymbol{\mu}, \end{aligned} \tag{6.113}$$

again by changing variables and symmetry arguments over the integrand, as in the univariate case, and also using the previous result. The second order moment can be computed as follows:

$$\begin{aligned} \mathbb{E}[\mathbf{x}\mathbf{x}^T] &= \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \int \exp\left( -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \mathbf{x}\mathbf{x}^T d\mathbf{x} \\ &= \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \int \exp\left( -\frac{1}{2} \mathbf{z}^T \boldsymbol{\Sigma}^{-1} \mathbf{z} \right) (\mathbf{z} + \boldsymbol{\mu})(\mathbf{z} + \boldsymbol{\mu})^T d\mathbf{z}. \end{aligned} \tag{6.114}$$

This gives us terms in $\mathbf{z}\boldsymbol{\mu}$ and $\boldsymbol{\mu}\mathbf{z}$, which vanish by symmetry arguments, a constant term $\boldsymbol{\mu}\boldsymbol{\mu}^T$, since the remaining part integrates to 1, and a last term of the form $\mathbf{z}\mathbf{z}^T$ which we still need to evaluate.

Since the orthonormal eigenvectors of the matrix $\Sigma$ furnishes a basis of $\mathbb{R}^D$ we can write:

$$\mathbf{z} = \sum_{j=1}^{D} \langle \boldsymbol{q}_j, \mathbf{z} \rangle \, \boldsymbol{q}_j = \sum_{j=1}^{D} \boldsymbol{q}_j^T \mathbf{z} \boldsymbol{q}_j = \sum_{j=1}^{D} y_j \boldsymbol{q}_j, \tag{6.115}$$

where $\boldsymbol{q}_j^T\mathbf{z} = \boldsymbol{q}_j^T(\mathbf{x}-\boldsymbol{\mu}) = y_j$, as defined previously. The remaining term in $\mathbf{z}\mathbf{z}^T$ becomes:

$$\begin{aligned}
&\frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\int \exp\Big(-\frac{1}{2}\mathbf{z}^T\boldsymbol{\Sigma}^{-1}\mathbf{z}\Big)\mathbf{z}\mathbf{z}^T d\mathbf{z} \\
&= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\int \exp\Big(-\frac{1}{2}\Big(\sum_{k=1}^{D} y_k\boldsymbol{q}_k\Big)^T\boldsymbol{\Sigma}^{-1}\Big(\sum_{l=1}^{D} y_l\boldsymbol{q}_l\Big)\Big)\Big(\sum_{j=1}^{D} y_j\boldsymbol{q}_j\Big)\Big(\sum_{i=1}^{D} y_i\boldsymbol{q}_i\Big)^T d\mathbf{y} \\
&= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\sum_{i=1}^{D}\sum_{j=1}^{D}\boldsymbol{q}_i\boldsymbol{q}_j^T\int \exp\Big(-\sum_{k=1}^{D}\frac{y_k^2}{2\lambda_k}\Big)y_iy_j d\mathbf{y} = \sum_{i=1}^{D}\boldsymbol{q}_i\boldsymbol{q}_i^T\lambda_i = \boldsymbol{\Sigma}.
\end{aligned} \tag{6.116}$$

There are quite a number of intermediate computations here where we use the following facts: the $\{\boldsymbol{q}_j\}$ is an orthonormal basis; $\Sigma^{-1} = \sum_{i=1}^{D}\frac{1}{\lambda_i}\boldsymbol{q_i}\boldsymbol{q_i}^T$; by symmetry the integral vanishes when $i \neq j$; when $i = j$ the resulting integral, with the normalization constants, has the value $\lambda_k$ as seen in equation (6.84). Thus we conclude:

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \boldsymbol{\mu}\boldsymbol{\mu}^T + \boldsymbol{\Sigma}. \tag{6.117}$$

From the definition of covariance of a random vector we have:

$$\mathrm{Cov}[\mathbf{x}] = \mathbb{E}[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T] = \mathbb{E}[\mathbf{x}\mathbf{x}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T = \boldsymbol{\Sigma}, \tag{6.118}$$

which motivates why we have been calling $\boldsymbol{\Sigma}$ the covariance matrix. ■

### 6.3.4 Properties in High-Dimensions

The behaviour of the multivariate normal distribution for large dimension $D$ is interesting and counter intuitive when taking as guide the shape of the distribution in figure 6.3. Here we follow exercise 1.20 of [Bis06] to show that, for a high-dimensional Gaussian distribution, the region with largest probability mass is located in a different region of that of highest probability density, quite unlike the behaviour in lower dimensions.

Consider the following isotropic Gaussian distribution centered at the origin:

$$p(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{D/2}}\exp\Big(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\Big). \tag{6.119}$$

We change to polar coordinates to look at the probability density at a distance $r$ from the origin. Since the distribution is radially symmetric, the density over a thin-shell of infinitesimal thickness $\epsilon$ at radius $r$ is approximately given by:

$$\int_{\text{shell}} p(\mathbf{x})d\mathbf{x} \approx \frac{1}{(2\pi\sigma^2)^{D/2}}\exp\Big(-\frac{r^2}{2\sigma^2}\Big)S_D r^{D-1}\epsilon, \tag{6.120}$$

where $S_D r^{D-1}$ is the surface area of a sphere of radius $r$ in $D$ dimensions, with $S_D = \frac{2\pi^{D/2}}{\Gamma(D/2)}$ and $\Gamma$ being the gamma function. Defining $p(r) := \exp\big(-\frac{r^2}{2\sigma^2}\big)r^{D-1}$, to find the the region of highest density we solve:

$$\frac{d}{dr}p(r) = 0 \Leftrightarrow \exp\Big(-\frac{r^2}{2\sigma^2}\Big)\Big((D-1)r^{D-2} - \frac{r}{\sigma^2}r^{D-1}\Big) = 0 \Leftrightarrow \hat{r} = \sigma\sqrt{D-1}, \tag{6.121}$$

for $r \neq 0$. We now show that, by varying $\hat{r}$ by $\delta \in \mathbb{R}$ with $\delta << \hat{r}$ we obtain $p(\hat{r}+\delta) \approx p(\hat{r})\exp\big(-\frac{\delta^2}{\sigma^2}\big)$.

Consider the ratio:

$$\begin{aligned}\frac{p(\hat{r}+\delta)}{p(\hat{r})} &= \frac{\exp\big(-\frac{(\hat{r}+\delta)^2}{2\sigma^2}\big)(\hat{r}+\delta)^{D-1}}{\exp\big(-\frac{\hat{r}^2}{2\sigma^2}\big)\hat{r}^{D-1}} \\ &= \frac{\hat{r}^{D-1}(1+\delta/\hat{r})^{D-1}}{\hat{r}^{D-1}}\exp\bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}\bigg) \\ &= (1+\delta/\hat{r})^{D-1}\exp\bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}\bigg) \\ &= \exp\big(\ln(1+\delta/\hat{r})^{D-1}\big)\exp\bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}\bigg) \\ &= \exp\bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}+(D-1)\ln(1+\delta/\hat{r})\bigg).\end{aligned} \tag{6.122}$$

We now expand the exponential term using the Taylor's series of $\ln(1+x) \approx x - \frac{x^2}{2} + \frac{x^3}{3} + ...$, for $|x| << 1$, to obtain:

$$\begin{aligned}&\bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}+(D-1)\ln(1+\delta/\hat{r})\bigg) \approx \bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}+(D-1)\bigg(\frac{\delta}{\hat{r}}-\frac{\delta^2}{2\hat{r}^2}\bigg)\bigg) = \\ &= \bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}+(D-1)\bigg(\frac{2\hat{r}\delta-\delta^2}{2\hat{r}^2}\bigg)\bigg) = \bigg(-\frac{2\hat{r}\delta+\delta^2}{2\sigma^2}+\frac{2\hat{r}\delta-\delta^2}{2\sigma^2}\bigg) = -\frac{\delta^2}{\sigma^2}.\end{aligned} \tag{6.123}$$

Combining the results we obtain $p(\hat{r}+\delta) = p(\hat{r})\exp(-\frac{\delta^2}{\sigma^2})$ which shows that the largest concentration of probability is around a thin shell at radius $\hat{r}$. However, computing the probability density at $\hat{r}$ and at $r = 0$ we obtain:

$$\begin{aligned}p(r=0) &= \frac{1}{(2\pi\sigma^2)^{D/2}}, \\ p(r=\hat{r}) &= \frac{1}{(2\pi\sigma^2)^{D/2}}\exp\bigg(-\frac{(D-1)}{2}\bigg) \approx \frac{1}{(2\pi\sigma^2)^{D/2}}\exp\bigg(-\frac{D}{2}\bigg).\end{aligned} \tag{6.124}$$

Where we used the fact that $D >> 1$, and thus we finally obtain:

$$\frac{p(r=0)}{p(r=\hat{r})} = \exp\bigg(\frac{D}{2}\bigg). \tag{6.125}$$

To conclude, we have seen that the point of largest probability density occurs at the origin $\mathbf{x} = 0$ but the region with largest probability mass occurs in a thin shell at an approximate distance $\hat{r} = \sigma\sqrt{D-1}$ from the origin.

The multivariate Gaussian has two important properties when considering conditional distributions and marginalization. When two sets of random variables are jointly Gaussian then the conditional distribution of one set over the other is also Gaussian, similarly when marginalizing one set over the other.

### 6.3.5 Conditional and Marginal

Suppose we have a $D-$dimensional Gaussian random vector $\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\sigma})$ and we partition $\mathbf{x}$ into two disjoint subsets $\mathbf{x}_a$, the first $M$ components of $\mathbf{x}$, and $\mathbf{x}_b$, the last $D-M$ components of $\mathbf{x}$. These naturally define the partitions:

$$\mathbf{x} = \begin{bmatrix}\mathbf{x}_a \\ \mathbf{x}_b\end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b\end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix}\boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb}\end{bmatrix}, \tag{6.126}$$

where, from the fact that $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^T$, we have $\boldsymbol{\Sigma}_{aa} = \boldsymbol{\Sigma}_{aa}^T$, $\boldsymbol{\Sigma}_{bb} = \boldsymbol{\Sigma}_{bb}^T$ and $\boldsymbol{\Sigma}_{ab} = \boldsymbol{\Sigma}_{ba}^T$. In order to simplify some expressions we define the precision matrix $\boldsymbol{\Lambda} := \boldsymbol{\Sigma}^{-1}$, which corresponds to the partition:

$$\boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{bmatrix}. \tag{6.127}$$

Because $\left(\boldsymbol{\Sigma}^T\right)^{-1} = \left(\boldsymbol{\Sigma}^{-1}\right)^T$, $\boldsymbol{\Lambda}$ is also a symmetric matrix and we have similar relations to the ones above over the partitioned blocks. Note that, in general, $\boldsymbol{\Lambda}_{aa} \neq \boldsymbol{\Sigma}_{aa}^{-1}$, similarly for the other blocks.

**Proposition 6.3.12.** *With the partitions defined as above, we have the conditional distribution:*

$$\begin{aligned} p(\mathbf{x}_a|\mathbf{x}_b) &= \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Lambda}_{aa}^{-1}), \\ \boldsymbol{\mu}_{a|b} &= \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}\big(\mathbf{x}_b - \boldsymbol{\mu}_b\big). \end{aligned} \tag{6.128}$$

*Proof.* Considering the partition, the square of the Mahalanobis distance can be expanded in the following way:

$$\begin{aligned} (\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = (\mathbf{x}_a-\boldsymbol{\mu}_a)^T\Lambda_{aa}(\mathbf{x}_a-\boldsymbol{\mu}_a) + (\mathbf{x}_a-\boldsymbol{\mu}_a)^T\Lambda_{ab}(\mathbf{x}_b-\boldsymbol{\mu}_b) \\ + (\mathbf{x}_b-\boldsymbol{\mu}_b)^T\Lambda_{ba}(\mathbf{x}_a-\boldsymbol{\mu}_a) + (\mathbf{x}_b-\boldsymbol{\mu}_b)^T\Lambda_{bb}(\mathbf{x}_b-\boldsymbol{\mu}_b). \end{aligned} \tag{6.129}$$

Since this is again quadratic in $\mathbf{x}_a$, the distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ will be Gaussian. In general, the expansion of the squared Mahalanobis distance is given by:

$$\begin{aligned} (\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) &= \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} - \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} + \text{const} \\ &= \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} - 2\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \text{const}. \end{aligned} \tag{6.130}$$

By writing eq. (6.129) as a function of $\mathbf{x}_a$ we can immediately read of the mean and covariance. We have then:

$$\begin{aligned} (\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) &= \mathbf{x}_a^T\boldsymbol{\Lambda}_{\boldsymbol{aa}}\mathbf{x}_a - 2\mathbf{x}_a^T\boldsymbol{\Lambda}_{\boldsymbol{aa}}\boldsymbol{\mu}_a + 2\mathbf{x}_a^T\Lambda_{ab}(\mathbf{x}_b-\boldsymbol{\mu}_b) + \text{const} \\ &= \mathbf{x}_a^T\boldsymbol{\Lambda}_{\boldsymbol{aa}}\mathbf{x}_a - 2\mathbf{x}_a^T\boldsymbol{\Lambda}_{\boldsymbol{aa}}\left(\boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{\boldsymbol{aa}}^{-1}\boldsymbol{\Lambda}_{ab}(\mathbf{x}_b-\boldsymbol{\mu}_b)\right) + \text{const}. \end{aligned} \tag{6.131}$$

Which, by comparison with eq. (6.130) is just $\mathbf{x}_a^T\boldsymbol{\Sigma}_{a|b}\mathbf{x}_a - 2\mathbf{x}_a^T\boldsymbol{\Sigma}_{a|b}\boldsymbol{\mu}_{a|b} + \text{const}$. Thus, we immediately read of:

$$\begin{aligned} \text{Cov}[\mathbf{x}_a|\mathbf{x}_b] &= \boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Lambda}_{aa}, \\ \mathbb{E}[\mathbf{x}_a|\mathbf{x}_b] &= \boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{\boldsymbol{aa}}^{-1}\boldsymbol{\Lambda}_{ab}(\mathbf{x}_b-\boldsymbol{\mu}_b). \end{aligned} \tag{6.132}$$

■

By using the Schur complement one can restate this result as a function of the partitions over the covariance matrix instead of the precision matrix, see section 2.3 in [Bis06].

**Proposition 6.3.13.** *With the partitions defined as above, the marginal distribution is given by:*

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}). \tag{6.133}$$

*Proof.* We want to compute:

$$p(\mathbf{x}_a) = \int p(\mathbf{x}_a, \mathbf{x}_b) d.\mathbf{x}_b \tag{6.134}$$

Since the integration is over $\mathbf{x}_b$ we can view the joint Gaussian distribution $p(\mathbf{x}_a, \mathbf{x}_b)$ as a function of $\mathbf{x}_b$. Similarly to the previous proof we have:

$$(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = \mathbf{x}_b^T\boldsymbol{\Lambda}_{\boldsymbol{bb}}\mathbf{x}_b - 2\mathbf{x}_b^T\Big(\boldsymbol{\Lambda}_{\boldsymbol{bb}}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\mathbf{x}_a-\boldsymbol{\mu}_a)\Big) + \text{const.} \tag{6.135}$$

Defining $\boldsymbol{m} := \big(\boldsymbol{\Lambda}_{\boldsymbol{bb}}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\mathbf{x}_a-\boldsymbol{\mu}_a)\big)$, which is independent of $\mathbf{x}_b$, in order to obtain a regular integral over a Gaussian distribution we can write:

$$\mathbf{x}_b^T\boldsymbol{\Lambda}_{\boldsymbol{bb}}\mathbf{x}_b - 2\mathbf{x}_b^T\boldsymbol{m} = \big(\mathbf{x}_{\boldsymbol{b}} - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m}\big)^T\boldsymbol{\Lambda}_{bb}\big(\mathbf{x}_{\boldsymbol{b}} - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m}\big) - \boldsymbol{m}^T\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m}. \tag{6.136}$$

Then, up to a constant factor, the integral in equation (6.134) is of the form:

$$\int \exp\Big\{-\frac{1}{2}\big(\mathbf{x}_{\boldsymbol{b}} - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m}\big)^T\boldsymbol{\Lambda}_{bb}\big(\mathbf{x}_{\boldsymbol{b}} - \boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m}\big)\Big\}d\mathbf{x}_b. \tag{6.137}$$

This is the integral over an unnormalized Gaussian so the result is just the reciprocal of the normalization constant. Then, equation (6.134) as a function of $\mathbf{x}_a$ picks up the term $\boldsymbol{m}^T\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m}$ and the other terms from equation (6.129) and is given by:

$$\begin{aligned}
&\alpha\exp\Big\{\frac{1}{2}\boldsymbol{m}^T\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{m} - \frac{1}{2}\mathbf{x}_a^T\boldsymbol{\Lambda}_{aa}\mathbf{x}_a + \mathbf{x}_a^T\big(\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a + \boldsymbol{\Lambda}_{ab}\boldsymbol{\mu}_b\big) + \text{const}\Big\} = \\
&= \alpha\exp\Big\{\frac{1}{2}\Big(\boldsymbol{\Lambda}_{bb}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\mathbf{x}_a-\boldsymbol{\mu}_a)\Big)^T\boldsymbol{\Lambda}_{bb}^{-1}\Big(\boldsymbol{\Lambda}_{bb}\boldsymbol{\mu}_b - \boldsymbol{\Lambda}_{ba}(\mathbf{x}_a-\boldsymbol{\mu}_a)\Big) - \frac{1}{2}\mathbf{x}_a^T\boldsymbol{\Lambda}_{aa}\mathbf{x}_a \\
&\qquad + \mathbf{x}_a^T\big(\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a + \boldsymbol{\Lambda}_{ab}\boldsymbol{\mu}_b\big) + \text{const}\Big\} \\
&= \alpha\exp\Big\{-\frac{1}{2}\mathbf{x}_a^T\big(\boldsymbol{\Lambda}_{aa} - \boldsymbol{\Lambda}_{ab}\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{\Lambda}_{ba}\big)\mathbf{x}_{\boldsymbol{a}} + \mathbf{x}_a^T\big(\boldsymbol{\Lambda}_{aa} - \boldsymbol{\Lambda}_{ab}\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{\Lambda}_{ba}\big)\boldsymbol{\mu}_a + \text{const}\Big\},
\end{aligned} \tag{6.138}$$

where $\alpha$ is some constant. Again, comparing with equation (6.130) we read of:

$$\begin{aligned}
\text{Cov}[\mathbf{x}_a] &= \boldsymbol{\Lambda}_{aa} - \boldsymbol{\Lambda}_{ab}\boldsymbol{\Lambda}_{bb}^{-1}\boldsymbol{\Lambda}_{ba} = \boldsymbol{\Sigma}_{aa}, \\
\mathbb{E}[\mathbf{x}_a] &= \boldsymbol{\mu}_a.
\end{aligned} \tag{6.139}$$

And we conclude:

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}). \tag{6.140}$$

■

### 6.3.6 Bayes' Theorem for a Linear Gaussian Model

We now assume that we have a Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ and a conditional Gaussian $\mathbf{y}|\mathbf{x} \sim \mathcal{N}(\mathbf{y}|A\mathbf{x}+\boldsymbol{b}, \boldsymbol{L}^{-1})$, i.e. the mean is a linear function of $\mathbf{x}$ and the covariance is independent of $\mathbf{x}$. This consideration will be useful when performing Bayesian linear regression in Section 2.1.4. There, we consider a prior over parameters $p(\boldsymbol{w})$ and the likelihood function $p(\mathcal{D}|\boldsymbol{w})$ to be Gaussian distributions. We are then interested in knowing the posterior distribution $p(\boldsymbol{w}|\mathcal{D})$ which means understanding the distribution of $\mathbf{x}|\mathbf{y}$. This is captured in the following proposition:

**Proposition 6.3.14.** *Let $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ and $\boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}, \boldsymbol{L}^{-1})$. Then, we have:*

$$\begin{aligned}
\boldsymbol{y} &\sim \mathcal{N}\Big(\boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{b}, \boldsymbol{L}^{-1} + \boldsymbol{A}\boldsymbol{\Lambda}^{-1}\boldsymbol{A}^T\Big), \\
\boldsymbol{x}|\boldsymbol{y} &\sim \mathcal{N}\Big((\boldsymbol{\Lambda}+\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A})^{-1}(\boldsymbol{A}^T\boldsymbol{L}(\boldsymbol{y}-\boldsymbol{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}), (\boldsymbol{\Lambda}+\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A})^{-1}\Big).
\end{aligned} \tag{6.141}$$

*Proof.* Let us begin by defining the joint distribution $\mathbf{z} := [\mathbf{x}\ \mathbf{y}]^T$ where we can write $p(\boldsymbol{z}) = p(\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x})$. We can compute the log of $p(\boldsymbol{z})$ to be:

$$\begin{aligned}\log p(\boldsymbol{z}) &= \log p(\boldsymbol{x}) + \log p(\boldsymbol{y}|\boldsymbol{x}) \\ &= \log\left(c\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Lambda}(\boldsymbol{x}-\boldsymbol{\mu})\right)\right) \\ &\quad + \log\left(c'\exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^T\boldsymbol{L}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})\right)\right) \\ &= -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Lambda}(\boldsymbol{x}-\boldsymbol{\mu}) - \frac{1}{2}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^T\boldsymbol{L}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) + \alpha,\end{aligned} \tag{6.142}$$

where $c, c'$ are the normalization constants and $\alpha$ is a constant independent of $\boldsymbol{x}$ and $\boldsymbol{y}$. This is a quadratic expansion in $\mathbf{z}$ so it follows a Gaussian distribution. By expanding we obtain the second order terms:

$$\begin{aligned}&-\frac{1}{2}\Big(\boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{x} + \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y} - \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A}\boldsymbol{x}\Big) \\ =&-\frac{1}{2}\Big(\boldsymbol{x}^T(\boldsymbol{\Lambda}+\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y} - \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{A}\boldsymbol{x}\Big).\end{aligned} \tag{6.143}$$

Which we can write as a function of $\mathbf{z}$ in the following way:

$$-\frac{1}{2}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix}^T\begin{bmatrix}\boldsymbol{\Lambda}+\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A} & -\boldsymbol{A}^T\boldsymbol{L}\\ -\boldsymbol{L}\boldsymbol{A} & \boldsymbol{L}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} := -\frac{1}{2}\mathbf{z}^T\boldsymbol{R}\mathbf{z}. \tag{6.144}$$

Comparing with equation (6.130) we have that $\boldsymbol{R}^{-1}$ is the covariance matrix of $\mathbf{z}$, which we can compute to be:

$$\boldsymbol{R}^{-1} = \begin{bmatrix}\boldsymbol{\Lambda}^{-1} & \boldsymbol{\Lambda}^{-1}\boldsymbol{A}^T\\ -\boldsymbol{L}\boldsymbol{A} & \boldsymbol{L}^{-1}+\boldsymbol{A}\boldsymbol{\Lambda}^{-1}\boldsymbol{A}^T\end{bmatrix}. \tag{6.145}$$

Similarly, the linear terms in eq. (6.142) are given by:

$$-\frac{1}{2}\Big(-2\boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{\mu} - 2\boldsymbol{y}^T\boldsymbol{L}\boldsymbol{b} + 2\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{b}\Big) = \begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix}^T\begin{bmatrix}\boldsymbol{\Lambda}\boldsymbol{\mu}-\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{b}\\ \boldsymbol{L}\boldsymbol{b}\end{bmatrix}. \tag{6.146}$$

Comparing with (6.130) we have:

$$\boldsymbol{R}\mathbb{E}[\mathbf{z}] = \begin{bmatrix}\boldsymbol{\Lambda}\boldsymbol{\mu}-\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{b}\\ \boldsymbol{L}\boldsymbol{b}\end{bmatrix} \implies \mathbb{E}[\mathbf{z}] = \boldsymbol{R}^{-1}\begin{bmatrix}\boldsymbol{\Lambda}\boldsymbol{\mu}-\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{b}\\ \boldsymbol{L}\boldsymbol{b}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{b}\end{bmatrix}. \tag{6.147}$$

We conclude then that the joint distribution is Gaussian, i.e.:

$$\mathbf{z} = \begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} \sim \mathcal{N}\Big(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{b}\end{bmatrix}, \boldsymbol{R}^{-1}\Big). \tag{6.148}$$

To marginalize over $\boldsymbol{y}$ we use the results obtained in proposition 6.3.13 together with the partitioned matrix $\boldsymbol{R}^{-1}$ obtained in 6.145. We have then:

$$\boldsymbol{y} \sim \mathcal{N}\Big(\boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{b}, \boldsymbol{L}^{-1}+\boldsymbol{A}\boldsymbol{\Lambda}^{-1}\boldsymbol{A}^T\Big). \tag{6.149}$$

Finally, applying proposition 6.3.12 $p(\boldsymbol{x}|\boldsymbol{y})$ is a Gaussian distribution given by:

$$p(\boldsymbol{x}|\boldsymbol{y}) \sim \mathcal{N}\Big((\boldsymbol{\Lambda}+\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A})^{-1}(\boldsymbol{A}^T\boldsymbol{L}(\boldsymbol{y}-\boldsymbol{b})+\boldsymbol{\Lambda}\boldsymbol{\mu}), (\boldsymbol{\Lambda}+\boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A})^{-1}\Big). \tag{6.150}$$

■

### 6.3.7 Maximum Likelihood

Consider a dataset $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ where we assume that the $\boldsymbol{x}_i$ are independently drawn from a multivariate normal distribution $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with unknown parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. The likelihood function for this dataset is given by:

$$p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left(\frac{1}{2\pi}\right)^{\frac{ND}{2}} \left(\frac{1}{|\boldsymbol{\Sigma}|}\right)^{\frac{N}{2}} \exp\left(\sum_{i=1}^{N} -\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right). \quad (6.151)$$

And the negative log likelihood is:

$$-\frac{ND}{2}\log(2\pi) - \frac{N}{2}\log(|\boldsymbol{\Sigma}|) - \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}) \quad (6.152)$$

We can estimate the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ by considering the derivative with respect to $\boldsymbol{\mu}$:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \log p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}) = 0 \implies \boldsymbol{\mu}^* = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i. \quad (6.153)$$

And with respect to $\boldsymbol{\Sigma}$, which can be computed (exercise 2.34 of [Bis06]) as:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \log p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0 \implies \boldsymbol{\Sigma}^* = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu}^*)(\boldsymbol{x}_i - \boldsymbol{\mu}^*)^T. \quad (6.154)$$

If we now compute the expectation of $\boldsymbol{\mu}^*$ and $\boldsymbol{\Sigma}^*$ under the true distribution we obtain the following:

$$\mathbb{E}[\boldsymbol{\mu}^*] = \boldsymbol{\mu}. \quad (6.155)$$

$$\mathbb{E}[\boldsymbol{\Sigma}^*] = \frac{N-1}{N}\boldsymbol{\Sigma}. \quad (6.156)$$

For proofs of these results see chapter 4 of [JW02]. The expectation of the maximum likelihood solution for the mean corresponds to the true mean while the expectation of the maximum likelihood solution for the covariance is biased and reduced by a factor of $(N-1)/N$. This can be corrected by considering, instead of $\boldsymbol{\Sigma}^*$:

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{N-1}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu}^*)(\boldsymbol{x}_i - \boldsymbol{\mu}^*)^T, \quad (6.157)$$

such that $\mathbb{E}[\tilde{\boldsymbol{\Sigma}}] = \boldsymbol{\Sigma}$.

### 6.3.8 Bayesian Inference

Via maximum likelihood we obtained point estimates of the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Now, via a Bayesian treatment, we shall obtain distributions over these parameters. We have three possible scenarios to consider, we might know the variance parameter $\boldsymbol{\Sigma}$ while $\boldsymbol{\mu}$ is unknown, the reverse scenario or the case were both are unknown. We treat the first case here, the remaining ones can be seen in section 2.3.6 of [Bis06].

Consider a dataset $\boldsymbol{x} = \{x_1, ..., x_N\}$ where each $x_i \in \mathbb{R}$ is obtained independently via the distribution $\mathcal{N}(x|\mu, \sigma^2)$ where $\sigma^2$ is given and $\mu$ is unknown (the multivariate case can be treated similarly). The likelihood function for this dataset is given by:

$$p(\boldsymbol{x}|\mu, \sigma^2) = \prod_{i=1}^{N} p(x_i|\mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{N/2} \exp\left(-\frac{1}{\sigma^2}\sum_{i=1}^{N}(x_i - \mu)^2\right), \quad (6.158)$$

which we view as a function of $\mu$. If we consider a Gaussian prior on $\mu$, i.e. that $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$ for some $\mu_0, \sigma^2 \in \mathbb{R}$ then, the posterior, which is of the following form as seen previously:

$$p(\mu|\boldsymbol{x}) \propto p(\boldsymbol{x}|\mu)p(\mu), \tag{6.159}$$

is also a Gaussian distribution. Let us see why:

$$\begin{aligned} p(\boldsymbol{x}|\mu)p(\mu) &= \left(\frac{1}{2\pi\sigma^2}\right)^{N/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2\right)\left(\frac{1}{2\pi\sigma_0^2}\right)^{1/2}\exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right) \\ &= c\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right) \\ &= c\exp\left(-\mu^2\left(\frac{N}{2\sigma^2}+\frac{1}{2\sigma_0^2}\right) + \mu\left(\frac{\mu_0}{\sigma_0^2}+\frac{\sum_{i=1}^{N}x_i}{\sigma^2}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}x_i^2 - \frac{1}{2\sigma_0^2}\mu_0^2\right). \end{aligned} \tag{6.160}$$

Reading the coefficients by comparing with the exponent expansion of a univariate Gaussian we obtain:

$$\frac{1}{\sigma'^2} = \frac{N}{\sigma^2} + \frac{1}{\sigma_0^2}, \tag{6.161}$$

and for the mean:

$$\frac{\mu\mu'}{\sigma'^2} = \mu\left(\frac{\mu_0}{\sigma_0^2}+\frac{\sum_{i=1}^{N}x_i}{\sigma^2}\right) = \mu\left(\frac{N\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\right)\underbrace{\left(\frac{\sigma^2\sigma_0^2}{N\sigma_0^2+\sigma^2}\right)\left(\frac{\mu_0}{\sigma_0^2}+\frac{\sum_{i=1}^{N}x_i}{\sigma^2}\right)}_{=\mu'}. \tag{6.162}$$

$$\mu' = \frac{\sigma^2\mu_0}{N\sigma_0^2+\sigma^2} + \frac{N\sigma_0^2\mu^*}{N\sigma_0^2+\sigma^2}, \tag{6.163}$$

where $\mu^* = \frac{1}{N}\sum_{i=1}^{N}x_i$, i.e. the maximum likelihood solution for $\mu$. The posterior follows then the Gaussian distribution:

$$p(\mu|\boldsymbol{x}) = \mathcal{N}\big(\mu', \sigma'^2\big). \tag{6.164}$$

If we observe no datapoints, i.e. $N = 0$, we can see from equations (6.163) and (6.161) that the posterior distribution is just the prior distribution. In the limit $N \to \infty$ the mean tends to the maximum likelihood solution $\mu' \to \mu^*$ and the variance $\sigma'^2 \to 0$, we get a Dirac-delta in $\mu^*$. In between, the mean is a compromise between the prior and the maximum likelihood while the variance decreases with each observed datapoint.

It is worth pointing out the sequential nature of the Bayesian approach. Given a prior over a dataset $\{x_1, ..., x_{N-1}\}$ and a new datapoint $x_N$ we can write the update as:

$$p(\mu|\boldsymbol{x}) \propto \left(\prod_{i=1}^{N-1} p(x_i|\mu)p(\mu)\right)p(x_N|\mu), \tag{6.165}$$

which shows how each new datapoint influences our belief over the distribution of $\mu$ up to a normalization factor.

**Limitations of the Gaussian distribution as a model distribution:** Following [Bis06] we list a few undesired properties of the Gaussian when using it as a model distribution, which we have certainly done often.

- The covariance matrix $\mathbf{\Sigma}$, being symmetric, requires the specification of $D(D+1)/2$ independent parameters, while the mean $\boldsymbol{\mu}$ requires another $D$ independent parameters. In total there are $D(D+3)/2$ independent parameters which grow quadratically and involves a large computational cost when inverting $\mathbf{\Sigma}$. If we limit ourselves to diagonal matrices, and thus $D$ independent parameters, or an isotropic covariance where we need only to specify one parameter, this computational cost is averted but we severely limit the models ability to capture the behaviour within the data.
- The Gaussian is unimodal and as such can not capture a multimodal distribution. This hindrance however can be overcome by considering mixtures of Gaussians, see section 2.3.9 of [Bis06].
- When considering periodic random variables the Gaussian is inappropriate since the results will be strongly dependent on an arbitrary choice of origin, see the discussion in section 2.3.8 of [Bis06].