Mathematical Machine Learning

Yahya SALEH, Tizian WENZEL, Kamal SHARMA

July 2, 2024

Contents

Introduction

Underlying the success of artificial intelligence are learning algorithms, i.e., algorithms that learn from data to perform a certain task. We start by two concrete examples of supervised learning algorithms. In the first example we consider the problem of approximating functions from pointwise evaluations using linear regression. In the second example we look at the task of classifying hand-written digits. In these two examples we identify and familiarize ourselves with the main components of learning algorithms; *datasets, a hypothesis class*, and *optimization algorithms*. We further identify important aspects of supervised learning algorithms, such as overfitting, and underfitting. Finally, we motivate in these examples, two problems at the forefront of research in mathematical machine learning, namely *the curse of dimensionality (CoD)* and *double/multiple descent phenomenon*.

I.1   Supervised Learning: Motivating Examples

In supervised learning tasks the dataset D is made up of two components, input variables $D_x = \{x_i\}_{i=1}^N$ and targets $D_y = \{y_i\}_{i=1}^N$. The dataset is assumed to be generated by an unknown function $f : \text{input} \to \text{target}$. The goal in a supervised learning task is to approximate the unknown function f pointwise, i.e., to find a function h such that $h(x) \approx f(x)$ for any x, whether it belongs to D_x or not. The target value can take finitely many values, e.g., $\text{target} \in \{0, 1, \dots, M\}$. In such a case, the supervised learning task is called a *classification task*. If the target can take infinitely many values, the learning task is called a *regression task*. Supervised learning problems are approached by first choosing a *hypothesis space* $\mathfrak{H}$, in which one looks for an approximation to the unknown function f. For example, if the data x is one-dimensional and the target takes values in $\mathbb{R}$ one can define the hypothesis class to be the set of all affine mappings, i.e.,

$$\mathfrak{H} = \big\{ f \mid f(x) = ax + b, \ a, b, \in \mathbb{R} \big\}. \tag{I.1}$$

Then, one can define a loss function l that measures how well a hypothesis function h approximates an unknown function f at a point x. Using the dataset D, the supervised learning problem can then be formulated as an optimization problem

$$\min_{h \in \mathfrak{H}} \frac{1}{N} \sum_{i=1}^N l(h(x_i), y_i). \tag{I.2}$$

While there are many alternatives to solve this optimization problem, by-far the most used algorithms are variants of the gradient-descent algorithm.

Let's look at some concrete examples.

Example I.1. [Regression] ☛ I.1 Let x be a random variable that takes values in the interval $[-1, 1]$. And assume we have access to a dataset $D = \{(x_i, y_i)_{i=1}^{200}\}$ generated by the unknown function

$$f(x) = x^2 \cos(5x) \exp(-x).$$

Assume that the dataset is corrupted by Gaussian noise. To learn a function h that approximates f, let your hypothesis class be the class of affine functions (I.1). Let the loss function be the absolute error, i.e.,

$$l(h(x_i), y_i) = |h(x_i) - y_i|$$
$$= |ax_i + b - y_i|.$$

Use a gradient-descent-like algorithm to choose the best hypothesis h, i.e., the best scalars a and b.

Change your hypothesis class to the class of all polynomials up to order 20 and repeat the optimization process. Which class is better for optimization? Figure I.1 shows the outcome of such an experiment.
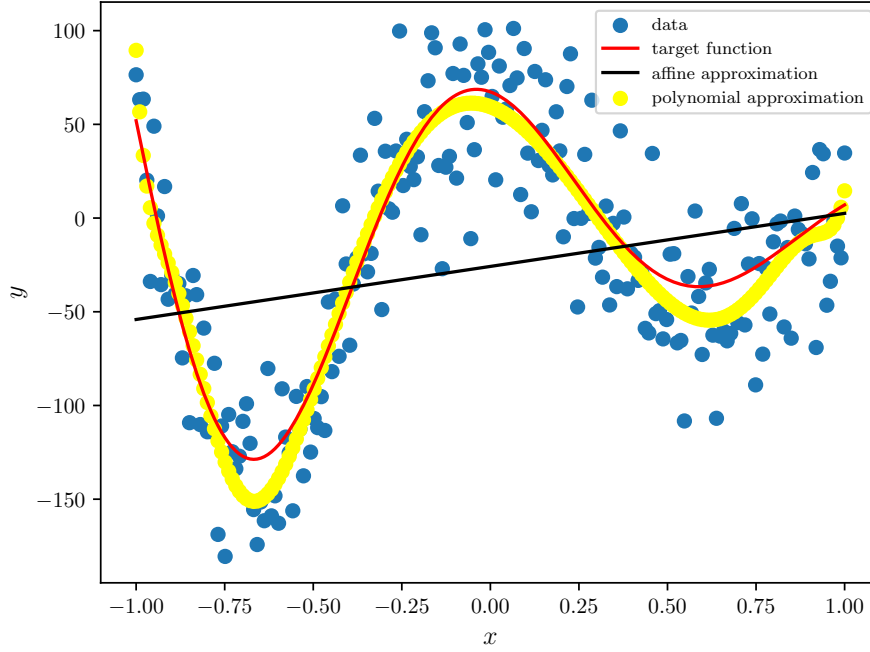

Figure I.1: A regression task; the goal is to fit noisy data (blue dots) assumed to be generated from a true function (solid red line). The data is fitted using an affine mapping (solid black line) and a polynomial mapping (solid yellow line).

Example I.2. [Classification] 👉 I.2 We consider a classification problem of hand-written digits. The input to the problem is an 8×8 image of a hand-written digit, and the output should be the predicted value of the digit. Formally, we consider x to be a random variable taking values in $[0, 16]^{8 \times 8} \subset \mathbf{N}^{8 \times 8}$, i.e., x is a random variables in a matrix representation, where each matrix element takes an integer value between 0 and 16. Here, the value of a certain matrix element represents its color, where 0 denotes black, and 16 denotes white. Let the target value y be a random variable taking values in the discrete set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. To solve this supervised learning problem we consider as a hypothesis class the following multilayer perceptron:

$$\mathfrak{H} = \left\{ \text{softmax } w_2 \left(\sigma(w_1 \cdot x + b_1) \right) + b_2; w_1 \in \mathbb{R}^{\text{dh},8}, b_1 \in \mathbb{R}^{\text{dh}}, w_2 \in \mathbb{R}^{10,\text{dh}}, b_2 \in \mathbb{R}^{10} \right\},$$

where dh is called the number of hidden units. In this class, the linear parameters are the weight matrices w_1, w_2 and the biases b_1, b_2. σ is a nonlinear non-learnable function, often referred to by the *activation function*. A common choice is the ReLU (Rectified Linear Unit) function

$$\text{ReLU}(x) = \max(0, x)$$

The softmax function (or layer) takes a set of real-valued input and transforms it into a probability distribution over multiple classes. In our example we have ten classes and the softmax is given by

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}.$$

Therefore, the output of the hypothesis function is a probability distribution over the 10 classes. Concretely, the output is 10-dimensional, where each entry denotes the probability of the input image to represent a certain digit.

For facilitating the implementation we represent the target value as a one-hot vector. For example, given a target value 4, we represent it as the vector $y = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$. A suitable loss function for such problems is the categorical cross entropy function given by

$$l(h(x_i), y_i) = \sum_{c=0}^{9} y_i^c \log(h(x_i)^c),$$

where (x_i, y_i) is a specific training example. y_i^c refers to the c-th entry of the one-hot vector representation of the target.

Compute the training and test errors and study how they change when changing the number of hidden units or the number of layers.

Figure I.1 shows two hypotheses, one linear and one nonlinear, that we learned to fit the data in Example I.1. The figure depicts an interesting phenomenon; a certain hypothesis h can fit the data too accurately; notice for example in Figure I.1 that the polynomial-regression model fits badly local minima of the target function. In these regions, it is optimized to fit the noise. The outcome of such a result is that the polynomial-regression model will fail to generalize well in these regions, i.e., it will have large error on unseen data in these regions. This is called *overfitting*. On the other hand, the linear-regression model produces largely deviated results from the data everywhere, and would, hence, also generalizes badly to unseen data. This is called an *underfitting* phenomenon.

Think about the influence of the following factors on the underfitting and overfitting:

- Complexity of the model. For a polynomial-regression model this can be the degree of the polynomial.

- Size of the dataset. For example, would adding more data decrease or increase underfitting?

I.2   This Course

In this section we introduce and motivate some questions that guide the structure of this course.

I.2.1   Generalization Error and Minimization Principles

In Example I.2 and Example I.1 we saw that a model trained on a certain dataset D_{train} is expected to generalize well on *unseen* data. This is a crucial difference from standard interpolation paradigms. Formally, the training data is assumed to follow an unknown probability distribution P, i.e., $D \sim P$. It is desirable that the learned model not only performs well on D, but also on any other dataset D_{test} that also follows the distribution P. The problem is somehow ill-posed; *how can one fit a model to a training data D and expect it to perform well on unseen data D_{test}?*

One strategy to tackle this question is *via* induction principle. Formally, obtaining a hypothesis that minimizes the error over the whole distribution, also known as the *true risk*, is impossible. Instead, one can do the next best thing. This is formally done by deriving an upper bound of the true risk that includes, among other terms, the empirical risk, i.e., the loss on the training data. Other terms include the so-called *Rademacher complexity*, a term which describes how complex the hypothesis class is. The original task of minimizing the error over the whole probability distribution is then replaced by the task of minimizing its upperbound. Such induction strategies are called *Empirical Risk Minimization Principles*. These principles show that minimizing only the loss function of the training dataset is not enough to obtain good generalization. One needs to regularize such loss functions, i.e., to add some terms, whose minimization reduces the complexity of the hypothesis class. This is directly linked to our previous discussion on overfitting.

The topic will be discussed in more detail in chapter 2.

I.2.2   Hypothesis Classes and Approximation Capabilities

In Example I.1 and Example I.2, we have seen some examples of hypothesis classes, such as the class of all affine mappings, the class of polynomials up to a predefined degree, and the class of single-layer neural networks. Another major hypothesis class is Kernel methods. Some important questions here are as follows: given a certain learning problem, what class do we use? Are we guaranteed to find an optimal solution in the chosen class? Can we increase accuracy simply by increasing the complexity of our class?

In chapter 3 we will study two important classes of models, neural networks and kernel methods. We will survey some approximation properties of these classes and perform error analysis for approximating important functions, such as continuous functions, L^p- functions and Sobolev functions.

Moreover, we look with some detail at specific important application domains, such as image recognition, and natural language processing.

I.2.3 Curse of Dimensionality

Assume that we want to optimize a potentially multi-variate unknown function $f : \mathbb{R}^d \to \mathbb{R}$ using a dataset of points sampled from it. Let $d = 1$, i.e., assume for now that f is uni-variate and let our hypothesis class be a linear class. Let us denote by T the computational costs needed to achieve a certain accuracy ϵ. It turns out that T grows exponentially with respect to d. In other words, the computational costs required to achieve accuracy ϵ grow exponentially with the dimension of the problem. This is known as the *Curse of Dimensionality* phenomenon.

Formally, when approximating an unknown function f by a linear approximator $\tilde{f}$, *appriori* error estimates are often given by

$$\|f - \tilde{f}\| \le c(d)\|f\|$$

where $\|.\|$ denotes some Sobolev norm of interest, and $c(d)$ is a constant that scales exponentially with the dimension of the problem. See, for example, error bounds for approximating Schwartz functions in the linear span of Hermite functions [1].

Example I.3. [CoD:Fitting] ☛ I.3 Consider fitting a dataset generated from the 1-dimensional target function

$$f(x) = \cos(2x)\exp(-x),$$

where you use the root-mean-squared error as a loss function and a polynomial-regression model as a hypothesis class. What is the degree of the polynomial necessary to achieve a training set error less than 10^{-5}. Similarly, study the same issue for fitting a dataset generated from the 2-dimensional target function

$$f(x, y) = \cos(2xy)\exp(-x).$$

In fact, this phenomenon is a major bottleneck for numerical methods to solve differential equations, such as finite differences, finite volumes or spectral methods.

Example I.4. [CoD:Solving Schrödinger Equation] I.4 Consider the following differential operator over $\mathbb{R}^d$

$$H = -\frac{1}{2}\left(\Delta + |x|^2 + \frac{1}{2}|x|^4\right),$$

where Δ denotes the Laplacian operator, and $|x| = \sqrt{\sum_{i=1}^{d} |x_i|^2}$. Its eigenvalue problem reads as follows: find all eigenpairs (E_n, ψ_n) that satisfy

$$H\psi_n = E_n\psi_n.$$

Assume we are interested only in the smallest eigenvalue E_0 and its corresponding eigenfunction ψ_0. Consider approximating ψ_0 in the linear span of truncated Hermite functions $(\gamma_n)_{n=0}^{\infty}$, i.e.,

$$\psi_0 \approx \tilde{\psi}_0$$
$$= \sum_{n=0}^{N-1} c_n \gamma_n.$$

Set $d = 1$. How many Hermite functions N are necessary to approximate E_0 to a relative accuracy of 10^{-1}? Repeat the same task for $d = 2$ and 3. What do you conclude? Answers are shown in Table I.1. For details on calculations refer to [2].

There is evidence, however, that neural networks are less prone to the CoD phenomenon. In other words, the computational scaling for using them to achieve a certain accuracy on a given task do not scale dramatically with the dimension of the problem. Characterizing such cases and providing

Table I.1: The size of a truncated Hermite basis N that is required to compute the smallest eigenvalue of the differential operator in Example I.4 in 1,2 and 3 dimensions to a relative absolute error $< 10^{-1}$.

d	1	2	3
N	3	45	286

rigorous understanding of this is a crucial point in modern mathematical machine learning. We will touch on this topic in chapter 4. Moreover, this topic will be of major interest to us when considering physics-informed neural networks.

I.2.4 Approximating Highly-Oscillatory Functions

The computational costs of approximation models, whether linear or nonlinear, seem to increase exponentially with an increase in the oscillation of a target function. This is a major bottleneck in some applications such as quantum dynamics. An important example here is approximating solutions to time-independent Schrödinger equations. Similar to Example I.4, the task here is to diagonalize a differential operator that describes a certain quantum system. We look here at a specific example, where H represents an operator, that describes vibrational motions inside a molecule. Given a linear numerical method to compute the eigenvalues, we look in Figure I.2 at the relative accuracy of the first 100 eigenvalues as a function of the truncation parameter N. It is clear that larger eigenvalues are harder to approximate. Morever, increasing the truncation parameter N does a worse job for improving the accuracy of higher eigenvalues than lower eigenvalues. This is partially because, larger eigenvalues correspond to highly-oscillatory functions. For details on the calculations see [3].
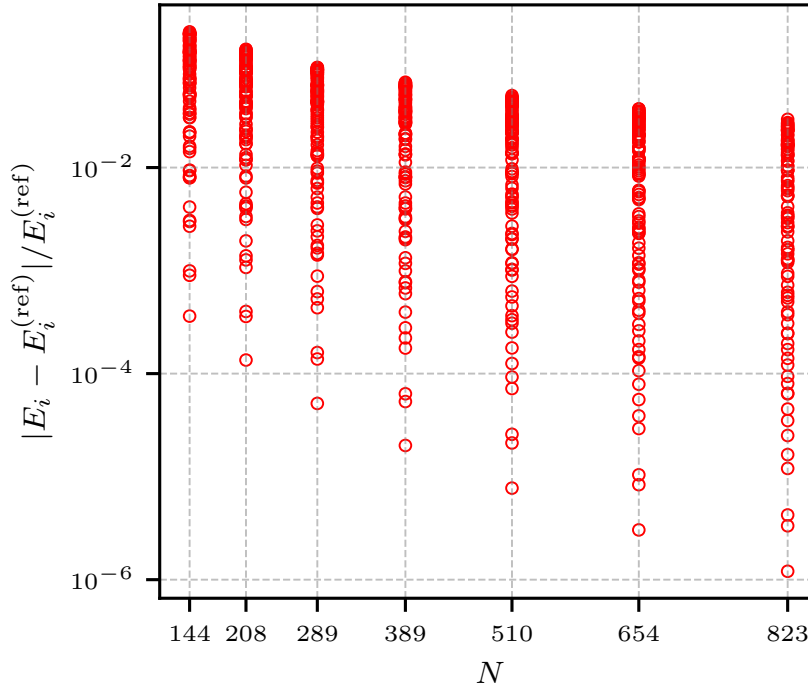

Figure I.2: The relative accuracy of the approximate first 100 eigenvalues of the vibrational Schrödinger equation for H_2S as a function of the truncation parameter N.

We will see in chapter 4 that neural-network approximation methods can significantly improve approximation capabilities for highly-oscillatory functions.

I.2.5 Validity of Occam's Razor and the Overparametrized Regime

Theoretical and empirical results have demonstrated in the past 40 years that Occam's Razor principle is valid when dealing with machine-learning problems: *Simple solutions are favored over unnecessarily complex ones.* Indeed, one can see in, e.g., Example I.1 that using a very high-order polynomial can produce very good results on the training data, but fail badly to produce sensible predictions on unseen data, resulting in an overfitting phenomenon. However, new evidence implies that very complex neural networks seem to generalize well on unseen data. In particular, many of the very successul machine-learning models that we use in our everyday life are heavily overparamterized, i.e., they have way more optimizable parameters than training data. Neverthess they generalize well on unseen data. Understanding the behavior of machine-learning models in this overparameterized regime is an important research direction in modern machine learning.

What is not covered in this course

The field of mathematical machine learning is huge and spans many standard mathematical areas, ranging from (geometric) measure theory to statistical learning theory, optimization theory and functional analysis. It is hence very challenging to cover all topics in a master course. Here is a list of topics that we do not cover in detail in this course and some references for self-study.

- Optimizers and their convergence are largely ignored in this course. We use standard first order optimizers in all our numerical examples and do not comment on their convergence properties.

- Bayesian learning models are largely marginalized.

- Unsupervised learning algorihtms such as clustering, and dimensionality reduction.

Wait! What is what?

Here is a list of questions that can help you check your understanding of key concepts in this chapter.

1. What are some examples of hypothesis classes? Which of them are linear or nonlinear approximation methods?

2. What loss functions do you know for regression and classification tasks? Can you think of other examples than the ones mentioned in this chapter?

3. For a fitting problem, can we get more accurate results by increasing the complexity of the hypothesis class?

4. How are the computational costs related to the dimension of the problem?

Statistical Learning Theory

In the previous chapter we have observed the phenomenon of overfitting; a model trained to minimize the empirical risk on a training dataset can still fail to generalize well over unseen dataset. A fundamental question in statistical learning theory is how to design hypothesis classes that do not overfit.

We will see in this chapter that restricting the complexity of the hypothesis classes can help reduce the overfitting. First, we start by looking at finite hypothesis classes and show that they do not overfit. This result will also motivate a notion of statistical learning, that of *probably approximately correct (PAC)* learning. However, finiteness of the hypothesis class is, indeed, a very restricting condition. We will discuss other measures of the complexity of a hypothesis class, such as the *Vapnik-Chervonenkis (VC) dimension* and the *Rademacher complexity*. At the end of this chapter, we will discuss what these theoretical results imply for the design of machine learning algorithms and link to the concept of *overparameterization*, which is a hot topic in the field of deep learning.

The reader is referred to [4] and [5] for more details.

We start this section by recalling some basic definitions of probability measure theory.

II.1   Probability Measure Theory

As we have seen in Introduction, training datasets are treated as random variables. This makes the following tools from probability theory essential.

Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability measure space, where $\Omega \subseteq \mathbb{R}$. It is common to refer to any $A \in \mathcal{A}$ by an *event*. An event A s.t. $\mathbb{P}(A) = 1$ is said to happen *almost surley*.

Important probability measures are those induced by measurable transformations in the following way.

Definition II.1 (Push-forward Measure)**.** Given two measurable spaces $(\Omega_1, \mathcal{A}_1)$, $(\Omega_2, \mathcal{A}_2)$ and a measurable mapping $h : \Omega_1 \to \Omega_2$ the *push-forward measure* of a measure $\mathbb{P}$ on $(\Omega_2, \mathcal{A}_2)$ is

$$h_\# \mathbb{P}(A) := \mathbb{P}(h^{-1}(A)) \quad \forall A \in \mathcal{A}_2.$$

The push-forward measure is sometimes denoted by $\mathbb{P}h^{-1}$.

We look now at a special kind of measurable mappings and the measures they induce.

Definition II.2 (Real Random Variables and Distributions)**.** Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability measure space.

(i) A measurable mapping $V : \Omega \to \mathbb{R}$ is called a *real random variable.*

(ii) The push-forward measure $\mathcal{P}_V := V_\# \mathbb{P}$ induced by a real random variable V is called the *(probability) distribution of V*.

A mapping $V : \Omega \to \mathbb{R}^n$ for $n > 1$ is usually referred to as a *random vector*. In our treatment we will refer to V by a random variable for any $n \geq 1$.

Definition II.3 (Expected Value)**.**   • The expected value of a random variable $X : \Omega \to \mathbb{R}$, denoted by $\mathbb{E}[X]$ is defined as:

$$\mathbb{E}[X] := \int_\Omega X(\omega) \ d\mathbb{P}(\omega)$$
$$= \int_\mathbb{R} x \ d\mathcal{P}_X(x),$$

where $\mathcal{P}_X$ is the pushforward-measure of X.

- Similarly, the expected value of a measurable mapping $g : \mathbb{R} \to \mathbb{R}$ as a function of the random variable X is given by

$$\mathbb{E}[g(X)] := \int_\Omega g(X)(\omega) \; d\mathbb{P}(\omega)$$
$$= \int_\mathbb{R} g(x) \; d\mathcal{P}_X(x).$$

Sometimes, one writes $\mathbb{E}[g] = \mathbb{E}_{x \sim \mathcal{P}_x}[g]$ to highlight the measure on $\mathbb{R}$ against which one integrates.

Definition II.4 (Independent Events)**.** Two events A, B are said to be *independent* if

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B).$$

Given an index set I, consider the family $A_i \in \mathcal{A}$ for $i \in I$. The family $(A_i)_{i \in I}$ of events is said to be *independent* if

$$\mathbb{P}(\cap_{j \in J} A_j) = \prod_{j \in J} \mathbb{P}(A_j) \quad \forall J \subset I.$$

The independence of events (i.e., sets) can be generalized to independence of families of sets.

Definition II.5 (Independence of Families of Sets)**.** Let I be an index set and consider $\mathcal{E}_i \subseteq \mathcal{A}$ for all $i \in I$. The family $(\mathcal{E}_i)_{i \in I}$ is called *independent* if, for any finite subset $J \in I$ and any choice of $E_j \in \mathcal{E}_j, j \in J$, one has

$$\mathbb{P}(\cap_{j \in J} E_j) = \prod_{j \in J} \mathbb{P}(E_j).$$

The following is an important family of random variables that one often encounters in machine learning and statistics.

Definition II.6 (Independent and Identically Distributed Random Variables)**.** Let I be an index set and $(V_i)_{i \in I}$ be a family of real random variables. Endow $\mathbb{R}$ with the Borel σ-algebra $\mathcal{B}$.

(i) The family $(V_i)_{i \in I}$ is said to be *identically distributed* if

$$\mathcal{P}_{V_i} = \mathcal{P}_{V_j} \quad \text{for all } i, j \in I.$$

(ii) The family $(V_i)_{i \in I}$ is said to be *independent* if the family of generated σ-algebras $\big(\sigma(V_i)\big)_{i \in I}$, where $\sigma(V_i) = V_i^{-1}(\mathcal{B})$ is independent.

A family of real random variables satisfying both conditions is said to be *independent and identically distributed (i.i.d.)*. In such a case set $\mathcal{P} = \mathcal{P}_{V_i}$.

II.2 A Formal Setting of Learning

We start by defining the standard setting of supervised learning. In our treatment, we will not use this very general setting. We will be imposing restrictions, for example, by considering only binary classification problems, or by using special loss functions. Nevertheless, it is useful to see the general setting first.

Let $z = (x, y)$ be a random variable where $x : \Omega \to \mathbb{X} \subseteq \mathbb{R}^n$ and $y : \Omega \to \mathbb{Y} \subset \mathbb{R}$. Denote by $\mathcal{P}$ the probability distribution of z and by $\mathcal{P}_x$ the marginal probability distribution corresponding to the random variable x. Further, we set $\mathbb{Z} := \mathbb{X} \times \mathbb{Y}$ [1]. We denote by $p(z)$ and $p(x)$ the probability density functions of z and x, respectively. These two densities are related by the formula $p(z) = p(x)p(y|x)$.

Given a hypothesis class $\mathfrak{H}$ of functions $h : \mathbb{X} \to \mathbb{Y}$ and a loss function $l : \mathbb{Y}^2 \to \mathbb{R}_{>0}$, the goal of a supervised-learning algorithm is to solve

$$\underbrace{R_\mathcal{P}(h) := \int_\mathbb{Z} l(y, h(x)) \; d\mathcal{P}}_{\text{True risk}} \longrightarrow \min_{h \in \mathfrak{H}} \implies h^\star,$$

---

[1]Do not confuse the notation with the set of integers.

that is, to minimize the *true risk*. Note that the true risk is also called the *generalization error*. However, one only has access to a finite realiazation of the random variable z, i.e., to a training set $D = \{(x_i, y_i)_{i=1}^m\}$. A reasonable thing to do is, hence, to minimize a finite/empirical representation of the true risk, i.e., to solve

$$\underbrace{\hat{R}_{\mathcal{P}}(h) := \sum_{(x,y) \in D} \frac{1}{|D|} l(y, h(x))}_{\text{Empirical risk}} \longrightarrow \min_{h \in \mathfrak{H}} \implies \hat{h}^\star.$$

A learning algorithm that replaces the original task of minimizing the true risk by the task of minimizing the empirical risk is called an *empirical risk minimization (ERM) learner* or is said to be using the *ERM learning rule*. To highlight the dependence of the empirical risk on the training data, we sometimes write $\hat{R}_{\mathcal{P}}(h; D)$.

One of the main problems of statistical learning theories is to study the validity of approximating h^* by $\hat{h}^*$. In particular, under what conditions on the hypothesis class does $\hat{h}^*$ have a small generalization error? In the next section, we will see that the finiteness of the hypothesis class is a sufficient condition to this end. Is it necessary though?

II.3   Probably Approximately Correct Learning

We start this section by looking closely at the problem of overfitting and show that finite classes do not overfit. Our results to this end motivate a notion of statistical learning that we discuss.

For this section, we will adopt the setting defined in section II.2 and further assume that $\mathbb{Y} = \{0, 1\}$, i.e., we restrict ourselves to the binary classification case. Moreover, we assume that y is generated from x by the deterministic functional relation $y = f(x)$, where $f : \mathbb{X} \to \{0, 1\}$. Unless otherwise stated, we will set the loss function to be the 0-1 loss which we define as follows.

$$l(y, h(x)) := \begin{cases} 1 : & h(x) \neq y \\ 0 : & \text{otherwise.} \end{cases}$$

Note that in this case, the true and empirical risks simplify to

$$R_{\mathcal{P}}(h) = \mathcal{P}_x\big(\{x : h(x) \neq y\}\big)$$

$$\hat{R}_{\mathcal{P}}(h) = \frac{1}{m}|\{x : h(x) \neq y\}|$$

Moreover, we restrict ourselves to working under the *realizability assumption*, i.e., that there exists $h^* \in \mathfrak{H}$ s.t. $R_{\mathcal{P}}(h^*) = 0$. Note that this assumption implies that the empirical risk of the hypothesis obtained by the ERM rule is zero, i.e., $\hat{R}(\hat{h}^*) = 0$ with probability 1 over the choice of the training data. In other words, the realizability assumption implies that the ERM rule provides a hypothesis that is *consistent* on the training data. To see this, note that $R_{\mathcal{P}}(h^*) = 0$ implies that $\hat{R}_{\mathcal{P}}(h^*; D) = 0$ with probability 1 over the choice of a dataset D that is i.i.d. generated by $\mathcal{P}$. In turn, this implies that $\hat{R}_{\mathcal{P}}(\hat{h}^*; D) = 0$ with probability 1 over the choice of D since $\hat{R}_{\mathcal{P}}(\hat{h}^*; D) \leq \hat{R}_{\mathcal{P}}(h^*; D)$ by the definition of the ERM rule.

II.3.1   Finite Hypothesis Classes do not Overfit

The goal in this section is to show that we won't encounter an overfitting problem if the hypothesis class has finitely many elements. We will deal with the overfitting problem in an approximate manner, i.e., we will say that a hypothesis h does not overfit if $R_{\mathcal{P}}(h) \leq \epsilon$ for some small $\epsilon > 0$.

In what follows, given a dataset D, we denote by $D|_x$ the input of the dataset, i.e., $D|_x = \{x_i : i = 1, \ldots, m\}$. Note that since each $x \in D|_x$ is a random variable with a distribution $\mathcal{P}_x$, the dataset $D|_x$ has distribution $\mathcal{P}_x^m$ over $\mathbb{X}^m$.

Lemma II.1. *Under the realizability assumption and for accuracy $\epsilon > 0$ it holds that*

$$\mathcal{P}_x^m\big(\{D|_x : R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\big) \leq |\mathfrak{H}| e^{-\epsilon m}.$$

In words; the probability of sampling m training data points and obtaining a learner $\hat{h}^*$ by the ERM rule that does not generalize well is upperbounded. Note that this upperbound is finite if the cardinality of $\mathfrak{H}$ is finite. Also note that

$$\mathcal{P}_x^m\big(\{D|_x : R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\big) = \mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\big(\{R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\big).$$

Proof. We start by defining the set $\mathfrak{H}_b$ of bad hypotheses, i.e., the set of all hypotheses that lead to a generalization error $> \epsilon$,

$$\mathfrak{H}_b := \{h \in \mathfrak{H} \text{ s.t. } \mathbb{R}_{\mathcal{P}}(h) > \epsilon\}.$$

Next, we define the set of misleading training data, i.e., the set of all training datasets of cardinality m, on which there is at least one hypothesis that produces zero training error and a generalization error $> \epsilon$,

$$M := \{D|_x : \ |D|_x| = m, \text{ and s.t. there exists } h \in \mathfrak{H}_b : \ \hat{R}_{\mathcal{P}}(h; D) = 0\}.$$

Note that the realizability assumption implies that $\hat{R}_{\mathcal{P}}(\hat{h}^*) = 0$ as discussed before. This in turn implies that $\{D|_x : R_{\mathcal{P}}(\hat{h}^*) > \epsilon\} \subseteq M$. It thus follows that

$$
\begin{aligned}
\mathcal{P}_x^m\big(\{D|_x : R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\big) &\leq \mathcal{P}_x^m(M) \\
&\leq \sum_{h \in \mathfrak{H}_b} \mathcal{P}_x^m\big(\{D|_x : \hat{R}_{\mathcal{P}}(h; D) = 0\}\big) \\
&= \sum_{h \in \mathfrak{H}_b} \prod_{i=1}^m \mathcal{P}_x\big(\{x : h(x) = y\}\big) \\
&= \sum_{h \in \mathfrak{H}_b} \prod_{i=1}^m (1 - R_{\mathcal{P}}(h)) \quad \textcolor{red}{(\text{definition of true risk})} \\
&\leq \sum_{h \in \mathfrak{H}_b} (1 - \epsilon)^m \quad \textcolor{red}{(\text{since } h \in \mathfrak{H}_b)} \\
&\leq |\mathfrak{H}_b|(1 - \epsilon)^m \\
&\leq |\mathfrak{H}|(1 - \epsilon)^m \\
&\leq |\mathfrak{H}|e^{-\epsilon m}.
\end{aligned}
$$

$\square$

The above lemma shows that the probability of overfitting is exponentially small with the size of the training data. It also provides us with a way to find the minimal amount of training data that guarantees a small generalization error.

Corollary II.1. *Let $\mathfrak{H}$ be a finite hypothesis class. Let $\delta \in (0, 1)$ be a confidence parameter and $\epsilon \in (0, 1)$ be an accuracy parameter. Let m be an integer that satisfies*

$$m \geq \frac{1}{\epsilon} \log(|\mathfrak{H}|/\delta).$$

Under the realizability assumption it holds that

$$\mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\big(\{R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\big) \leq \delta.$$

In other words,

$$R_{\mathcal{P}}(\hat{h}^*) \leq \epsilon$$

holds with probability of at least $1 - \delta$ over the choice of the training data D.

Note that this result holds for any labeling function f and any distribution $\mathcal{P}_x$.

Proof. Follows straight-forwardly from the previous lemma. $\square$

The results we proved so far show that finite hypothesis classes do not overfit. Here, overfitting is defined in an *approximate sense* controlled by parameter ϵ. The results guarantee that the ERM rule provides a hypothesis that generalizes well in a *probabilistic sense*. The probability here is controlled by a parameter δ. This motivates the following definition.

Definition II.7. A hypothesis class $\mathfrak{H}$ is PAC learnable if there exist a function $m_{\mathfrak{H}} : (0, 1)^2 \to \mathbb{N}$ and a learning algorithm with the following properties

- for every $(\epsilon, \delta) \in (0, 1)^2$
- for every distribution $\mathcal{P}_x$ over $\mathbb{X}$
- for every labeling function $f : \mathbb{X} \to \{0, 1\}$

s.t. if the realizability assumption holds, when running the learning algorithm on $m \geq m_{\mathfrak{H}}(\delta, \epsilon)$ i.i.d. samples generated by $\mathcal{P}_x$ and labeled by f, the algorithm returns a hypothesis h such that

$$R_{\mathcal{P}}(h) \leq \epsilon$$

with probability of at least $1 - \delta$ over the choice of the samples.

Definition II.8 (samples complexity). The sample complexity of leaning a hypothesis class $\mathfrak{H}$ is the minimal integer that satisfies the requirement of PAC learnability with accuracy ϵ and confidence δ.

With these definitions we can restate our corollary in the following manner.

Corollary II.2. *Every finite hypothesis calss is PAC learnable with sample complexity*

$$m \leq \lceil \frac{1}{\epsilon} \log(|\mathfrak{H}|/\delta) \rceil$$

One may wonder whether the finiteness of the hypothesis class is a necessary condition for PAC learnability. We will see now that this is indeed not the case. For example, we consider here the class of threshold functions.

Definition II.9 (Class of Threshold Functions). $\mathfrak{H} := \{h_a : \mathbb{R} \to \{0,1\}, h_a(x) = \mathbf{1}_{x<a}, \ a \in \mathbb{R}\}$

In words, this class consists of all functions that assign 1 to all inputs that are smaller than a threshold a and 0 otherwise. The class of threshold functions has infinite cardinality. However, we will see that this class is PAC learnable.

Lemma II.2. *The class of threshold functions $\mathfrak{H}$ is PAC-learnable using the ERM rule with sample complexity of $m_{\mathfrak{H}} \leq \lceil \frac{1}{\epsilon} \log(2/\delta) \rceil$.*

Proof. It suffices to show that

$$\mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\left(\{R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\right) \leq 2\exp(-\epsilon m).$$

To prove this note that we work under the realizability assumption. Hence, it is possible to find a hypothesis $\hat{h}^* = h_{\hat{a}^*}$ that is consistent with the training data.

Let a^* be the threshold of the labeling function f. Let $a_r^* > a^*$ be such that $\mathbb{P}_{x \sim \mathcal{P}_x}\left(x \in (a^*, \hat{a}_r^*)\right) = \epsilon$. Similarly, let $a_l^* < a^*$ be such that $\mathbb{P}_{x \sim \mathcal{P}_x}\left(x \in (\hat{a}_l^*, a^*)\right) = \epsilon$. Notice that picking datapoints that are outside the interval (a_l^*, a^*) or outside (a^*, a_r^*) is a sufficient condition for picking training datasets that lead to a bad hypothesis. Formally

$$\mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\left(\{R_{\mathcal{P}}(\hat{h}^*) > \epsilon\}\right) \leq \mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\left(\{\forall x \in D|_x, x \notin (a_l^*, a^*) \vee x \notin (a^*, a_r^*)\}\right)$$

$$\leq \underbrace{\mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\left(\{\forall x \in D|_x, x \notin (a_l^*, a^*)\}\right)}_{\text{term1}} + \mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\left(\{\forall x \in D|_x, x \notin (a^*, a_r^*)\}\right).$$

Note that

$$\text{term } 1 \leq \mathbb{P}_{D|_x \sim \mathcal{P}_x^m}\left(\{x_1 \notin (a_l^*, a^*) \wedge x_2 \notin (a_l^*, a^*) \wedge \cdots \wedge x_m \notin (a_l^*, a^*)\}\right)$$

$$= \prod_{i=1}^{m} \mathbb{P}_{x \sim \mathcal{P}_x}\left(x \notin (a_l^*, a^*)\right)$$

$$= (1 - \epsilon)^m.$$

The claim follows by doing the same inequality for the second term and using $(1-\epsilon)^m \leq \exp(-\epsilon m)$. $\qquad \square$

In summary, our results show that finiteness of the hypothesis class is not a good condition for characterizing PAC learnability, since infinite classes can be PAC learnable. We thus move in the next section to introduce a better measure of the complexity of a hypothesis class.

II.3.2 Vapnik-Chervonenkis Dimension

The Vapnik-Chervonenkis (VC) dimension is a measure of the complexity of a hypothesis class. The intuition behind this measure is to define the richness of a class with respect to a given dataset. For example, a class of threshold functions is infinite dimensional. However, given a dataset D of only one point $x \in \mathbb{R}$, the whole class of threshold functions can give this datapoint only two different values, either 0 if the hypothesis has a threshold lying to the left of the datapoint, or 1 otherwise. Now consider a dataset of two points. The class of threshold functions can label these two points in three different ways, (0,0), (1,0) or (1,1). However, the hypothesis class of threshold functions cannot represent the labeling (0,1). Hence, the class of threshold functions cannot exhaust all the possible labeling of a dataset of two points. This is indeed the motivation behind the following definitions.

Definition II.10 (Restriction of $\mathfrak{H}$ to C). Let $\mathfrak{H}$ be a class of functions from $\mathbb{X}$ to $\{0, 1\}$ and let $C = \{x_1, \ldots, x_m\} \subset \mathbb{X}$. The restriction of $\mathfrak{H}$ to C is the set of functions from C to $\{0, 1\}$ that can be derived from $\mathfrak{H}$. That is,

$$\mathfrak{H}_C := \{(h(x_1), \ldots, h(x_m)) : h \in \mathfrak{H}\}.$$

If $\mathfrak{H}_C$ is the set of all functions from C to $\{0, 1\}$ we say that $\mathfrak{H}$ shatters C. Formally we write as follows.

Definition II.11 (Shattering). A hypothesis class $\mathfrak{H}$ shatters a finite set $C \subset \mathbb{X}$ if $|\mathfrak{H}_C| = 2^{|C|}$.

According to this definition we observe that the class of threshold functions shatters a set of one point but does not shatter a set of two points, because it does not exhaust all of its possible labelings.

The following definition quantifies the number of different labelings that a hypothesis class can provide for a dataset of fixed size.

Definition II.12 (Growth Function). Let $\mathfrak{H}$ be a hypothesis class. The growth function of $\mathfrak{H}$, denoted by $\tau_{\mathfrak{H}} : \mathbb{N} \to \mathbb{N}$ is defined as

$$\tau_{\mathfrak{H}}(m) := \max_{C \subset \mathbb{X} : |C| = m} |\mathfrak{H}_c|$$

In words; $\tau_{\mathfrak{H}}(m)$ is the maximum number of different functions from a set C of size m to $\{0, 1\}$ that can be obtained by restricting $\mathfrak{H}$ to C.

It turns out that the growth function is a good quantity to measure the complexity of a hypothesis class, since a bounded growth function implies PAC learnability.

Theorem II.1. *Let $\delta \in (0, 1)$ be a confidence parameter and $\epsilon \in (0, 1)$ be an accuracy parameter. Let m be an integer that satisfies*

$$\epsilon \geq \frac{1}{m} \log(|\tau_{\mathfrak{H}}(2m)|/\delta).$$

Under the realizability assumption it holds that

$$\mathbb{P}_{D|_x \sim \mathcal{P}_x^m} \left(\{R_{\mathcal{P}}(\hat{h}^*) > \epsilon\} \right) \leq \delta.$$

Note that this result is very similar to Lemma II.1. The main difference is that the growth function is used instead of the cardinality of the hypothesis class.

Proof. To prove the result it suffices to show that

$$\mathcal{P}_x^m \left(\{D|_x : R_{\mathcal{P}}(\hat{h}^*) > \epsilon\} \right) \leq 2\tau_{\mathfrak{H}}(2m)2^{-\epsilon m/2}.$$

Given a hypothesis function h we define the quantity $M(h; D)$ as the number of errors that h makes on the dataset D. Since we are using the 0-1 loss we note that $\hat{R}_{\mathcal{P}}(h; D) = M(h; D)/m$. We define the event B as the event that a hypothesis is consistent with the training data but does not generalize well, i.e.,

$$B : \exists h \in \mathfrak{H} : M(h; D) = 0 \wedge R_{\mathcal{P}}(h) > \epsilon.$$

To have an empirical estimate of the true risk we assume that we have access to a test set D' of size m that is i.i.d. generated by $\mathcal{P}_x$. We now define the event B'

$$B' : \exists h \in \mathfrak{H} : M(h; D) = 0 \wedge M(h; D') > \epsilon m/2.$$

Now consider the following experiment. For each element $x \in D|_x$ and $x' \in D'|_x$ we flip a fair coin. If the coin lands heads we swap x with x'. Otherwise we do not swap. We denote the resulting datasets by T and T'. We now define the event B'' as follows

$$B'' : \exists h \in \mathfrak{H} : M(h; T) = 0 \wedge M(h; T') > \epsilon m/2.$$

We now claim that

$$\mathbb{P}(B) \underbrace{\leq}_{*_1} 2\mathbb{P}(B') \underbrace{=}_{*_2} 2\mathbb{P}(B'') \underbrace{\leq}_{*_3} 2\ \tau_{\mathfrak{H}}(2m)2^{-\epsilon m/2}.$$

Proving this claim would complete the proof. We now prove the inequality1 $*_1$,

Note that

$$\mathbb{P}(B') \geq \mathbb{P}(B' \wedge B)$$
$$= \mathbb{P}(B'|B)\mathbb{P}(B)$$

The claim now follows from the fact that $\mathbb{P}(B'|B) \geq 1/2$ if $m \geq 8/\epsilon$. While we will not formally prove this result, here is the intuition behind it. Assume that the event B has occurred. This means that there is a hypothesis h that is consistent with the training data but does not generalize well. Now consider $M(h; D')$ as a random variable and note that $\mathbb{E}_{D'|_x \sim \mathcal{P}_x^m} M(h; D') = R_{\mathcal{P}}(h)m$. Since $R_{\mathcal{P}}(h) > \epsilon$ we have that $\mathbb{E}_{D'|_x \sim \mathcal{P}_x^m} M(h; D') > \epsilon m/2$. Hence, the probability of B' is equivalent to the probability of the random variable $M(h; D')$ being larger than a lower bound to its expected value. The desired result is based on the use of a measure-concentration inequality, i.e., an inequality that study the convergence of an empirical average of a random variable to its true expected value.

To see the inequality $*_2$ note that the samples D, D' are i.i.d. generated by $\mathcal{P}_x$. Since T, T' were generated by equally likely permutations, we have that T, T' are also i.i.d. generated by $\mathcal{P}_x$. Hence, the probability of B'' is the same as the probability of B'.

The inequality $*_3$ follows from the application of the law of total expectation which states that for two random variables X, Y it holds that $\mathbb{E}(X) = \mathbb{E}(\mathbb{E}(X|Y))$. Note that the probability of some event A can be written as the expected value of the indicator function of A. It hence follows that

$$\mathbb{P}_{D,D' \sim \mathcal{P}_x^{2m}}(B'') = \mathbb{E}_{D,D' \sim \mathcal{P}_x^{2m}} \mathbb{P}(B''|D, D').$$

Now let $\mathfrak{H}'$ be a hypothesis class containing one representative for each different labeling of $\mathfrak{H}$ of D, D'. We have that

$$\mathbb{P}(B''|D, D') = \mathbb{P}(\exists h \in \mathfrak{H} : \underbrace{M(h; T) = 0 \wedge M(h; T') > \epsilon m/2}_{:=b(h)} |D, D')$$
$$= \mathbb{P}(\exists h \in \mathfrak{H}' : b(h)|D, D')$$
$$\leq \sum_{h \in \mathfrak{H}'} \mathbb{P}(b(h)|D, D')$$
$$\leq |\mathfrak{H}'|2^{-\epsilon m/2} \quad \text{assume } \mathbb{P}(b(h)|D, D') \leq 2^{-\epsilon m/2}$$
$$\leq \tau_{\mathfrak{H}}(2m)2^{-\epsilon m/2}.$$

We prove the assumption $\mathbb{P}_{D,D' \sim \mathcal{P}_x^{2m}}(b(h)|D, D') \leq 2^{-\epsilon m/2}$ by considering three different cases. For a fixed h, we note that D, D' already occurred. Hence, the only randomness is coming from the random permutation process to construct T, T'.

Assume for the first case that there exists an index i s.t. h makes a mistake on both $x_i \in D|_x$ and $x_i' \in D'|_x$. In this case, it is not possible to have $M(h; T) = 0$. Hence, the event $b(h)$ occurs with probability 0.

For the next two cases define r to be the number of pairs $(x_i, x_i'), x_i \in D|_x, x_i' \in D'|_x$ where h is wrong on exactly one between the two, i.e., either on x_i, or on x_i'.

Consider for the second case $r < m\epsilon/2$. Even if we get lucky enough and all elements on which h is correct end up in T, we will not have enough datapoints to satisfy the condition $M(h; T') > \epsilon m/2$. Hence, the event $b(h)$ occurs with probability 0.

For the third case consider $r \geq m\epsilon/2$. In this case, we need that all the coin flips end up in the correct way to have zero mistakes in T and all mistakes in T'. Since the coin flips are independent and fair, we have that
$$\mathbb{P}(b(h)|D, D') = 2^{-r} \leq 2^{-m\epsilon/2}.$$

$\square$

Due to the combinatorial nature of the growth function, its asymptotic behavior is not obvious. Getting hold of it would provide a better bound in Theorem II.1. We will now introduce the more intuitive quantity of the VC-dimension, which will enable us to calculate how many datapoints we need to get PAC learnability with a hypothesis class of infinite cardinality.

Definition II.13 (VC (Vapnik-Chervonenkis) dimension)**.** The VC dimension of a hypothesis class $\mathfrak{H}$, denoted VC-dim($\mathfrak{H}$) is the cardinality of the largest set C that can be shattered by $\mathfrak{H}$.

Example II.5. To show that a finite hypothesis class has a VC-dimension d, it suffices to show that there exists a set of size d that can be shattered by the hypothesis class and that there exists no set of size $d + 1$ that can be shattered by the hypothesis class.

- The VC dimension of threshold functions is 1.

- Consider the following hypothesis class

$$\mathfrak{H} = \{h : h(x) = \mathbf{1}_{x \in (a,b)}, a < b, a, b \in \mathbb{R}\}.$$

The VC dimension of this class is 2.

The previous examples may convey the idea that the VC dimension of a certain class equals the number of free parameters that the class has. This is not always the case. For example, the VC dimension of the class

$$\mathfrak{H} = \{h(x) = \sin(\theta x) : \theta \in \mathbb{R}\}$$

is infinite.

The following lemma allows us to replace the growth function by the VC-dimension in Theorem II.1.

Lemma II.3 (Sauer's lemma)**.** *Let $\mathfrak{H}$ be a hypothesis class. If*

- $VC - dim(\mathfrak{H}) < \infty$ *then* $\tau_{\mathfrak{H}} = O(m^d)$ *for all* $m \in \mathbb{N}$.

- $VC - dim(\mathfrak{H}) = \infty$ *then* $\tau_{\mathfrak{H}} = 2^m$ *for all* $m \in \mathbb{N}$.

Corollary II.3. *Let $\delta \in (0, 1)$ be a confidence parameter and $\epsilon \in (0, 1)$ be an accuracy parameter. Assume VC-dim($\mathfrak{H}$) = d < ∞. Let m be an integer that satisfies*

$$\epsilon \geq C \frac{d \log m + \log(1/\delta)}{m}.$$

for some constant C. Under the realizability assumption it holds that

$$\mathbb{P}_{D|_x \sim \mathcal{P}_x^m} \left(\{ R_{\mathcal{P}}(\hat{h}^*) > \epsilon \} \right) \leq \delta.$$

Proof. The proof follows from applying Sauer's lemma to Theorem II.1. □

So far we have seen that the VC-dimension is a good measure of the complexity of a hypothesis class in the sense that having a finite VC-dimension implies PAC learnability. Now we will see that an infinite VC-dimension implies that the hypothesis class is not PAC learnable. To see this, we use the following theorem that states that a VC-dimension that is larger than twice the size of the training data implies that we might end up with a hypothesis that does not generalize well. Intuitively, the theorem states that having a high VC-dimension implies that the hypothesis class is too rich that it can fit random labels, so it is not capable of learning.

Lemma II.4. *Let $\mathfrak{H}$ be a hypothesis class of functions from $\mathbb{X}$ to $\{0, 1\}$. Let m be a training set size. Assume that VC-dim($\mathfrak{H}$) ≥ 2m. Then, for any learning algorithm A there exist a distribution $\mathcal{P}$ over $\mathbb{X} \times \{0, 1\}$ s.t. $\hat{R}_{\mathcal{P}}(h^A) = 0$ but with probability of at least 1/7 over the choise of $D|_x \sim \mathcal{P}_x^m$ we have that*

$$R_{\mathcal{P}}(h^A) \geq 1/8.$$

Corollary II.4. *Let $\mathfrak{H}$ be a class of inifinite VC-dimension. Then, $\mathfrak{H}$ is not PAC learnable.*

Proof. Since VC-dim($\mathfrak{H}$) = ∞, for any training set of size m, there exists a shattered set of size $2m$. Hence, the result follows from the previous lemma. □

While we managed to show that the VC-dimension is a better measure of complexity than the finiteness of the hypothesis class, it is still not optimal. One of the major limitations of the VC-dimension is its restriction to binary classification. In the next section we will see a more general measure of the complexity of a hypothesis class, that is the Rademacher complexity.

II.4 Agnostic PAC Learning

So far we worked under the realizability assumption, i.e., we assumed that the labeling function f is a member of the hypothesis class $\mathfrak{H}$. We showed that this implies the existence of functions in $\mathfrak{H}$ that have zero empirical risk on the training data. The realizability assumption is, indeed, a very strong assumption and in many practical scenarios it does not hold. In this section we will relax this assumption and consider the more general case where the labeling function f is not necessarily a member of the hypothesis class $\mathfrak{H}$. This is known as the agnostic setting. We will generalize the PAC-learning framework to the agnostic setting.

In the previous section we always assumed that the target value y is generated form x by the functional relation f. From now on, we relax this assumption and assume that there is some randomness in the generation of y from x that is described by the conditional distribution $\mathcal{P}(y|x)$.

Generalizing PAC-learning to the agnostic setting is straightforward.

Definition II.14. A hypothesis class $\mathfrak{H}$ is agnostic PAC learnable if there exist a function $m_{\mathfrak{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following properties

- for every $(\epsilon, \delta) \in (0,1)$

- for every distribution $\mathcal{P}$ over $\mathbb{X} \times \mathbb{Y}$

when running the learning algorithm on $m \geq m_{\mathfrak{H}}(\delta, \epsilon)$ i.i.d. samples generated by $\mathcal{P}$, the algorithm returns a hypothesis $\hat{h}^*$ such that

$$R_{\mathcal{P}}(\hat{h}^*) \leq \min_{h \in \mathfrak{H}} R_{\mathcal{P}}(h) + \epsilon$$

with probability of at least $1 - \delta$ over the choice of the samples.

We note that under the realizability assumption, agnostic PAC-learning reduces to PAC-learning. We now define a tool that will allow us to prove agnostic PAC learnability of a hypothesis class.

Definition II.15 (ϵ-representative sample). A training set D is said to be $\epsilon-$representative (w.r.t. a domain $\mathbb{Z}$, a hypothesis class $\mathfrak{H}$, a loss l, and a distribution $\mathcal{P}$) if for every $h \in \mathfrak{H}$ it holds that

$$|R_{\mathcal{P}}(h) - \hat{R}_{\mathcal{P}}(h)| \leq \epsilon$$

We emphasize that this definition is uniform for $h \in \mathfrak{H}$.

Whenever a dataset D is $\epsilon/2$-representative, the ERM learner is guaranteed to return a good hypothesis (in the agnostic PAC sense).

Theorem II.2. *Assume that a training set D is $\epsilon/2$-representative. Then, for any*

$$\hat{h}^* \in argmin_{h \in \mathfrak{H}} \hat{R}_{\mathcal{P}}(h; D)$$

it holds that

$$R_{\mathcal{P}}(\hat{h}^*) \leq \min_{h \in \mathfrak{H}} R_{\mathcal{P}}(h) + \epsilon$$

Proof. For any h in $\mathfrak{H}$ we have that

$$
\begin{aligned}
R_{\mathcal{P}}(\hat{h}^*) &\leq \hat{R}_{\mathcal{P}}(\hat{h}^*) + \epsilon/2 \quad &\textcolor{red}{D \text{ is } \epsilon/2 - \text{representative}} \\
&\leq \hat{R}_{\mathcal{P}}(h) + \epsilon/2 \quad &\textcolor{red}{\hat{h}^* \in \text{argmin}_{h \in \mathfrak{H}} \hat{R}_{\mathcal{P}}(h)} \\
&\leq R_{\mathcal{P}}(h) + \epsilon \quad &\textcolor{red}{D \text{ is } \epsilon/2 - \text{representative}.}
\end{aligned}
$$

The result holds in particular for the hypothesis $\hat{h}^*$. $\qquad\qquad\square$

The previous theorem suggests that it is enough to prove uniform convergence of the empirical risk to the true risk to show agnostic PAC learnability, i.e., it is enough to show

$$\hat{R}_{\mathcal{P}}(h) \leq R_{\mathcal{P}}(h) + \epsilon/2$$

for all $h \in \mathfrak{H}$ with high probability. We apply this new knowledge to show that a finite hypothesis class is agnostic PAC-learnable.

II.4.1 Finite Hypothesis Classes Revisited

Previously, we showed that a finite hypothesis class is PAC learnable. It turns out that we can extend this to the agnostic setting. For this we rely on the following concentration-of-measure inequality.

Theorem II.3 (Hoeffding's inequality)**.** *Let* $\theta_1, \ldots, \theta_n$ *be a sequence of i.i.d. random variables and assume that for all* i $\mathbb{E}[\theta_i] = \mu$ *and* $\mathbb{P}[a \leq \theta_i \leq b] = 1$. *Then, for any* $\epsilon > 0$

$$\mathbb{P}[|\frac{1}{m} \sum_{i=1}^{m} \theta_i - \mu| > \epsilon] \leq 2 \exp(-2m\epsilon^2/(b-a)^2).$$

We are now ready to state our first result in agnostic PAC learning setting.

Theorem II.4. *Let* $|\mathfrak{H}| < \infty$ *then* $\mathfrak{H}$ *is agnostically PAC learnable with sample complexity*

$$m_{\mathfrak{H}}(\epsilon, \delta) \leq \lceil \frac{2 \log(2|\mathfrak{H}|/\delta)}{\epsilon^2} \rceil$$

Proof. As previously discussed, it is sufficient here to show the uniform convergence property, i.e., to show that

$$\mathbb{P}_{D \sim \mathcal{P}^m}\big(\{D : \exists h \in \mathfrak{H}, |R_{\mathcal{P}}(h) - \hat{R}_{\mathcal{P}}(h)| > \epsilon\}\big) \leq \delta.$$

To this end we write

$$\mathbb{P}_{D \sim \mathcal{P}^m}\big(\{D : \exists h \in \mathfrak{H}, |R_{\mathcal{P}}(h) - \hat{R}_{\mathcal{P}}(h)| > \epsilon\}\big) \leq \sum_{h \in \mathfrak{H}} \mathbb{P}_{D \sim \mathcal{P}^m}\big(\{D : |R_{\mathcal{P}}(h) - \hat{R}_{\mathcal{P}}(h)| > \epsilon\}\big)$$
$$\leq |\mathfrak{H}| \cdot 2 \exp(-2m\epsilon^2)$$
$$\leq \delta,$$

where we used Hoeffding's inequality after having noted that $R_{\mathcal{P}}(h) = \mathbb{E}_{D \sim \mathcal{P}^m} \hat{R}_{\mathcal{P}}(h)$. $\square$

We now talk about a new measure of complexity of a hypothesis class and show agnostic PAC learnability in the new setting.

II.4.2 Rademacher Complexity

We found out in the last section that the VC-dimension is a good measure of the complexity of a hypothesis class. However, the VC-dimension is limited to binary classification problems. While there are ways to rectify this limitation, we will look into a more general measure of the complexity of a hypothesis class, mainly that of the Rademacher complexity.

We start by motivating the Rademacher complexity with a simple example.

Consider the binary classification task with the zero-one loss. Further, assume that the target values $y \in \{-1, 1\}$. Note that

$$\hat{R}_{\mathcal{P}}(h) = \frac{1}{m} \sum_{i=1}^{m} l(y_i, h(x_i))$$
$$= \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{\{h(x_i) \neq y_i\}} \quad \textcolor{red}{\text{since we are using the 0-1 loss}}$$
$$= \frac{1}{m} \sum_{i=1}^{m} \frac{1 - h(x_i)y_i}{2} \quad \textcolor{red}{\text{since } y_i \in \{-1, 1\} \text{ for any } i}$$
$$= \frac{1}{2} - \frac{1}{2m} \sum_{i=1}^{m} h(x_i)y_i.$$

Thus,

$$\frac{1}{2m} \sum_{i=0}^{m} h(x_i)y_i = 1 - 2\hat{R}_{\mathcal{P}}(h).$$

The quantity $\frac{1}{2m} \sum_{i=0}^{m} h(x_i)y_i$ is a good measure of how well the hypothesis h fits the data. Note also that minimizing the empirical risk is equivalent to maximizing this quantity. The Rademacher complexity generalizes this quantity to be a measure of how well h fits any dataset, not only

D. Formally, one replaces y_i by a random variable σ_i that takes values in $\{-1, 1\}$ with equal probability. The Rademacher complexity is defined to be

$$\text{Rad}(\mathfrak{H}) := \mathbb{E}_\sigma \Big[\sup_{h \in \mathfrak{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i h(x_i) \Big].$$

Intuitively, the Rademacher complexity measures how well a hypothesis class can fit random labels. Let us now compute the Rademacher complexity for two simple hypothesis classes. First, let the hypothesis class contain one element, i.e., $\mathfrak{H} = \{h\}$. We then compute

$$\begin{aligned}
\text{Rad}(\mathfrak{H}) &= \mathbb{E}_\sigma \Big[\frac{1}{m} \sum_{i=1}^m \sigma_i h(x_i) \Big] \\
&= \frac{1}{m} \sum_{i=1}^m \mathbb{E}_\sigma [\sigma_i] h(x_i) \\
&= \frac{1}{m} \sum_{i=1}^m 0 h(x_i) \\
&= 0.
\end{aligned}$$

Second, we consider the case, where $\mathfrak{H}$ shatters the dataset D. In this case, we have that the empirical risk of the ERM hypothesis is zero. Hence, the Rademacher complexity is one.

We will now show that our definition of the Rademacher complexity extends in a meaningful way to regression problems.

Consider now a generic supervised learning problem. Given a certain dataset D we define its representativeness to be
$$\text{Rep}(D) := \sup_{h \in \mathfrak{H}} (\hat{R}_\mathcal{P}(h; D) - R_\mathcal{P}(h)).$$

Clearly, one cannot compute this quantity since one cannot compute the true risk. However, one can approximate this quantity by considering a test dataset D' generated from the true distribution $\mathcal{P}$. We define the empirical representativeness of D to be

$$\hat{\text{Rep}}(D) := \sup_{h \in \mathfrak{H}} (\hat{R}_\mathcal{P}(h; D) - \hat{R}_\mathcal{P}(h; D')).$$

Note that

$$\hat{\text{Rep}}(D) = \sup_{h \in \mathfrak{H}} \Big(\frac{1}{|D|} \sum_{(x_i, y_i) \in D} l(h(x_i), y_i) - \frac{1}{|D'|} \sum_{(x_i, y_i) \in D'} l(h(x_i), y_i') \Big)$$

Now assume that $|D| = |D'| = m/2$ and let set

$$\sigma_i = \begin{cases} 1 & \text{if } (x_i, y_i) \in D \\ -1 & \text{if } (x_i, y_i) \in D' \end{cases}.$$

It follows that

$$\hat{\text{Rep}}(D) = \sup_{h \in \mathfrak{H}} \frac{2}{m} \sum_{i=1}^m \sigma_i l(h(x_i), y_i).$$

The Rademacher complexity generalizes this idea by taking the expectation over all possible choices of σ_i.

We note that the Rademacher complexity of a hypothesis class depends on the choice of a loss function and training data set D. Hence, we introduce the following notation to make this dependence explicit.

$$l \circ \mathfrak{H} \circ D := \{ (l(h(x_1), y_1), \ldots, l(h(x_m), y_m)) : h \in \mathfrak{H} \}$$

$$\mathfrak{F} := l \circ \mathfrak{H} := \{ z \rightarrow l(h, z) : h \in \mathfrak{H} \}$$

We are now ready to state the formal definition of the Rademacher complexity.

Definition II.16 (Rademacher Complexity)**.** Let $\sigma = (\sigma_1, \ldots, \sigma_m) \in \{-1, 1\}^m$ be a vector of random variables such that $\mathbb{P}[\sigma_i = 1] = \mathbb{P}[\sigma_i = -1] = 1/2$. The Rademacher complexity of $l \circ \mathfrak{H} \circ D$ is defined as

$$
\mathrm{Rad}(l \circ \mathfrak{H} \circ D) := \frac{1}{2} \mathbb{E}_\sigma \big[\hat{\mathrm{Rep}}(D) \big]
$$

$$
= \mathbb{E}_\sigma \Big[\sup_{f \in \mathfrak{F}} \frac{1}{m} \sum_{i=1}^m \sigma_i f(x_i) \Big]
$$

$$
= \mathbb{E}_\sigma \Big[\sup_{h \in \mathfrak{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i l(h(x_i), y_i) \Big]
$$

The following result shows that a finite Rademacher complexity allows for agnostic PAC learning.

Theorem II.5. *Assume that for all $z = (x, y)$ and $h \in \mathfrak{H}$, $|l(h(x), y)| \leq c$. Then, for any $\delta > 0$ with probability of at least $1 - \delta$ over the choice of the sample D, for any $h \in \mathfrak{H}$ it holds that*

$$
R_\mathcal{P}(h) \leq \hat{R}_\mathcal{P}(h) + 2Rad(l \circ \mathfrak{H} \circ D) + c\sqrt{\frac{2 \log(4/\delta)}{m}} \tag{II.3}
$$

and

$$
\hat{R}_\mathcal{P}(h) \leq R_\mathcal{P}(h) + 2\mathbb{E}_{D' \sim \mathcal{P}^m} \big[Rad(l \circ \mathfrak{H} \circ D') \big] + c\sqrt{\frac{2 \log(2/\delta)}{m}}
$$

These results hold in particular for $h = \hat{h}^*$.

We accept the previous theorem without proof. However, we note that the proof is similar to the proof of the agnostic PAC learnability of finite hypothesis classes. The main difference here lies in the utilization of another concentration-of-measure inequality, namely the McDiarmid's inequality.

Remark II.1 (Empirical Risk Minimization Principle)**.** *Let us now spend some time to discuss the interpretation of the last theorem. (II.3) suggests that the true risk of any hypothesis h is bounded by the empirical risk of h plus terms that depend on the complexity of the hypothesis class and the size of the training data set. This result offers us a way to escape the dilemma of supervised learning, i.e., the fact that we want to minimize the true risk, although we do not have access to the true distribution of data. Since we cannot directly minimize the true risk, we can do the next best thing available to us, i.e., minimize an upperbound to it. Such result is called the ERM principle. We emphasize that the main message here is that we can solve the supervised learning problem by minimizing the empirical risk while controlling the complexity of the hypothesis class.*

We will now derive a meaningful bound to the Rademacher complexity of the linear hypothesis class given by

$$
\mathfrak{H} = \{x \to \langle x, w \rangle : \quad w \in \mathbb{R}^n\},
$$

where $\langle ., . \rangle$ denotes the inner product on $\mathbb{R}^n$. To this end, we need the following result.

Lemma II.5 (Letting out of the Loss Function)**.** *For any training example, let $\phi_i : \mathbb{R} \to \mathbb{R}$ be a $\rho-$Lipschitz function. For $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{R}^m$ let $\phi(a) = (\phi(a_1), \ldots, \phi(a_m))$. Let $\phi \circ A = \{\phi(a) : \quad a \in A\}$. Then, it holds that*

$$
Rad(\phi \circ A) \leq \rho Rad(A)
$$

The previous lemma allows us to consider the Rademacher complexity of $\mathfrak{H} \circ D$ instead of $l \circ \mathfrak{H} \circ D$ under a Lipschitz constraint on the loss function l.

Theorem II.6 (Rademacher complexity of Linear Classes)**.** *It holds that*

$$
Rad(\mathfrak{H} \circ D) \leq \frac{\|w\|_2}{m} \max_i \|x_i\|_2
$$

Proof.

$$mRad(l \circ \mathfrak{H} \circ D) = \mathbb{E}_\sigma \Big[\sup_{a \in \mathfrak{H} \circ D} \sum_{i=1}^m \sigma_i a_i \Big]$$

$$= \mathbb{E}_\sigma \Big[\sup_{w \in \mathbb{R}^n} \sum_{i=1}^m \sigma_i \langle x_i, w \rangle \Big]$$

$$= \mathbb{E}_\sigma \Big[\sup_{w \in \mathbb{R}^n} \big\langle \sum_{i=1}^m \sigma_i x_i, w \big\rangle \Big]$$

$$\leq \|w\|_2 \mathbb{E}_\sigma \Big[\| \sum_{i=1}^m \sigma_i x_i \|_2 \Big] \quad \textcolor{red}{\text{by the Cauchy-Schwarz inequality}}$$

$$\leq \|w\|_2 \mathbb{E}_\sigma \Big[\| \sum_{i=1}^m \sigma_i x_i \|_2^2 \Big]^{1/2} \quad \textcolor{red}{\text{using Jensen's inequality}}$$

$$= \|w\|_2 \mathbb{E}_\sigma \Big[\sum_{i,j} \sigma_i \sigma_j \langle x_i, x_j \rangle \Big]^{1/2}$$

$$= \|w\|_2 \mathbb{E}_\sigma \Big[\sum_{i \neq j} \sigma_i \sigma_j \langle x_i, x_j \rangle + \sum_i \sigma_i^2 \langle x_i, x_i \rangle \Big]^{1/2}$$

$$= \|w\|_2 \Big(\sum_{i \neq j} \mathbb{E}_\sigma[\sigma_i \sigma_j] \langle x_i, x_j \rangle + \sum_i \mathbb{E}_\sigma[\sigma_i^2] \langle x_i, x_i \rangle \Big)^{1/2}$$

$$= \|w\|_2 \Big(\sum_{i \neq j} 0 \ \langle x_i, x_j \rangle + \sum_i 1 \langle x_i, x_i \rangle \Big)^{1/2} \quad \textcolor{red}{\text{since } \sigma_i \text{ are independent}}$$

$$= \|w\|_2 \Big(\sum_{i=1}^m \|x_i\|_2 \Big)^{1/2}$$

$$\leq \|w\|_2 \ m^{1/2} \ \max_i \|x_i\|_2^{1/2}.$$

$\square$

Remark II.2 (Link to Regularization). *The previous result shows that the Rademacher complexity of a linear hypothesis is bounded by the product of the norm of the weight vector and the maximum norm of the data points. This result implies that a smaller 2-norm of the weight vector implies a smaller complexity of the hypothesis class. This, indeed, aligns with a common practice in applications, where one adds the 2-norm of the weight vector to the loss function to prevent overfitting, see also Example I.2 and Example I.1.*

II.5   Occam's Razor and the Overparameterization Regime

We have seen in this chapter that the unachievable task of minimizing the true risk can be replaced by the next best thing, i.e., minimizing an upperbound to it. In particular, this was suggested by upperbounds of the form

$$\text{true risk} \leq \text{empirical risk} + \text{complexity term}.$$

But how do the empirical risk and the complexity term behave? Assume that we have two hypothesis classes that result in roughly similar empirical risk minimizers. In other words, for two different hypothesis classes, running the ERM minimization algorithm returns hypothesis with similar empirical errors. Which of the two hypothesis classes is better? The guiding upperbound suggests that the hypothesis class with the smaller complexity term is better. So, in a sense, the Occam's razor principle is at play here; the simpler answer is better.

It turns out, however, that the relationship between the empirical risk and the complexity term is not straightforward. In particular, note that the empirical risk can be made arbitrarily small by increasing the number of learnable parameters of the hypothesis class. However, a hypothesis class with more learnable parameters is also more complex. For example, consider a certain classification task and let the hypothesis class be that of neural networks. Moreover, assume that we measure the complexity of the hypothesis class by the VC-dimension. To better fit the data, one can simply increase the number of learnable parameters n_p in the neural network. However, one can also show that the VC-dimension of the hypothesis class is bounded by $\mathcal{O}(n_p \log n_p)$. This suggests

that increasing the number of parameters in the neural network increases the complexity term. In summary, increasing the expressivity of the hypothesis class allows us to better fit the training data, but it also increases the complexity term, and hence the generalization error. This is known as bias-variance tradeoff and is a central concept in machine learning[2].

Indeed, the bias-variance tradeoff is an established concept in machine learning and statistics. It has been observed for multiple learning settings and a wide variety of hypothesis classes. However, lately, there has been a surge of counter examples. In particular, it has been observed that increasing the number of parameters in a neural network does not necessarily increase the generalization error. In fact, in many applications, one uses neural networks with a number of parameters that is much larger than the number of training data points and still obtains a small generalization error. This phenomenon is known as overparameterization.

Figure section II.5 summarizes the clash between common wisdom and new observations.
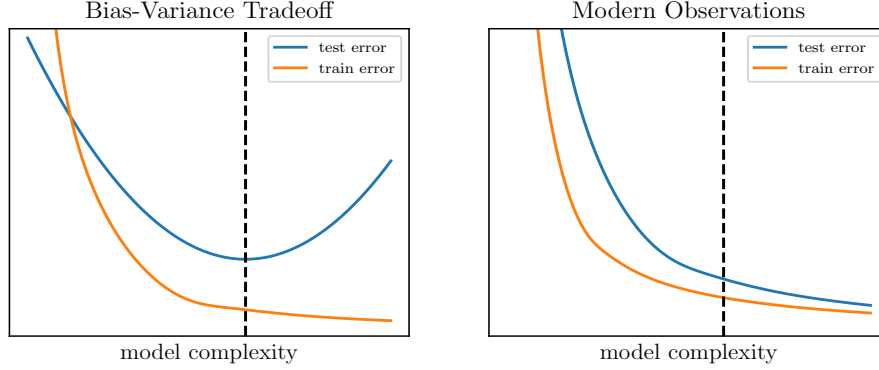

Figure II.3: Common wisdom and many empirical observations suggest that the training error decreases with the complexity of the hypothesis class. The test error decreases as well, only to increase again after a certain threshold as shown in the figure on the left. Here, the threshold usually corresponds to the point where the number of learnable parameters equals the size of the dataset. However, modern observations suggest that deep neural networks can have many parameters and still generalize well as shown in the figure on the right.

Recently, there has been a lot of research to understand the phenomenon of overparameterization. While we will not delve into the details of this research, we look at a one specific example.

Example II.6.   Consider as a hypothesis class the space of 2-layer neural networks with ReLU activation functions. Assume that the learnable weights corresponding to the first layer do not deviate much in the 2-norm from their initial values. Further, assume that the weights of the second layer remain bounded in the 2-norm throughout the training. Formally, this class can be written as

$$\mathfrak{H} := \{h(x) = V \cdot [U \cdot x]_+ : U, V \in W\},$$

where

$$W := \{(U, V) \quad U \in \mathbb{R}^{n \times n_{\text{hu}}}, V \in \mathbb{R}^{n_{\text{hu}} \times n_{\text{class}}} \quad \text{and}$$
$$\|u_i - u_i^0\|_2 \leq \beta, \|v_i\|_2 \leq \alpha \quad \text{for all units}\},$$

and n_{hu} is the number of hidden units, n_{class} is the number of classes, $[.]_+$ denotes the ReLU activation function, u_i, v_i denote the i-th column of U and V, respectively, u_i^0, v_i^0 are the i-th column of the initial weight matrix U, and V, respectively, and β and α are constants.

Train such a neural network on the publicly available CIFAR-10 dataset. Study empirically both the training error and the test error as a function of the number of hidden units. What do you observe? Experiment with n_{hu} ranging from 8 to 30000.

---

[2]Note that in practice, the test error is a proxy of the generalization error

> Now, empirically study the quantities $\|U\|_2$, $\|V\|_2$, and $\|U - U^0\|_2$ as a function of the number of hidden units. What do you observe?

In [6] the authors did the empirical study explained in Example II.6 and observed the following:

- Both the training error and test error decrease as a function of n_{hu}. This indicates that we are in the overparameterization regime.

- The quantity $\|U\|_2$ initially decreases as a function of n_{hu} and then increases. The quantities $\|V\|_2$ and $\|U - U^0\|_2$ decrease as a function of n_{hu}.

The second observation, and specifically the decrease of $\|U - U^0\|_2$ as a function of n_{hu}, is particularly interesting. It suggests that the weights of the neural network do not deviate much from their initial values during training. It appears here that training the neural network is then more focused on optimizing the weights of the second layer. When interpreting the hidden units as features, this result suggests that when one has many features, one has consequently many relevant features that are useful for the task at hand. Hence, there is no need to learn new features.

In a next step, the authors bounded the Rademacher complexity of the specific hypothesis class in Example II.6 by the quantities $\|U - U^0\|_2$ and $\|V\|_2$. In particular, they showed that

$$\mathrm{Rad}(l \circ \mathfrak{H} \circ D) \leq \frac{C}{\sqrt{m}} \max_i \|x_i\|_2 \; \alpha \; (\beta + \|U^0\|_2)$$

This suggests that the Rademacher complexity of the hypothesis class decreases as a function of n_{hu}, which is in line with the empirical observations.

Wait! What is what?

Here is a list of questions that help you check your understanding of key concepts inside this chapter.

1. What is the ultimate goal of a supervised learning problem?

2. Where does the randomness in probably-approximately-correct (PAC) learning come from?

3. What is the realizability assumption in PAC learning?

4. What is the empirical risk minimization principle?

5. How do we judge whether a certain type of a complexity measure of a hypothesis class is good?

6. What is the bias-variance tradeoff? How does it relate to the complexity of the hypothesis class?

7. What are the limitations of the VC-dimension as a complexity measure of a hypothesis class?

8. What is the overparameterization regime?

Neural networks

In the Introduction chapter, we breifly talked about a neural network which can be used to recognize hand-written digits. The neural network takes an image of a hand-written digit as an input and predicts a number which corresponds to the digit as an output. In this chapter, we will delve deeper into the architecture of neural networks and try to answer the following questions:

1. What is the motivation behind using neural networks ?

2. What is the idea behind the structure (neurons, input layer, hidden layers, output layer) of neural networks ?

3. Why do we need an activation function and a loss function ?

4. How to train a neural network ?

5. What are learning algorithms ?

6. Why certain neural networks can approximate any continuous function defined on a compact set ?

Recognition of handwritten digits is a classic problem for introducing the topic of neural networks. Therefore we will use this problem as an example to introduce various concepts related to neural networks.

III.1   Handwritten digit recognition

Most people can easily recognize the handwritten digits shown in figure III.4. Even if these digits were written in a different manner (for example refer figure III.5), one would still be able to recognize them.   This is due to the fact that our visual cortex has evolved over millions of
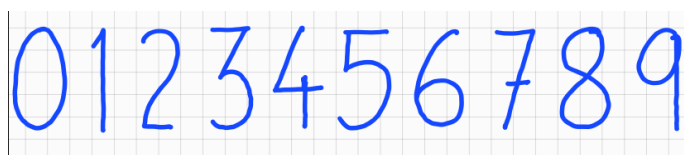

Figure III.4: One of the ways of writing digits from 1 to 9.

Figure III.5: Different ways of writing the digit 3.

years and therefore is magnificently adapted to understand the visual world [7]. The problem of recognizing hand-written digits is so trivial for our brains that we don't have to put an effort to recognize digits. But this task will immediately start to look complicated when one tries to write a computer program which can recognize hand-wriiten digits. Let us try to write a program for this task. We can start by coverting the digits into a pixel image (28×28 for example) and assigning each pixel a value from 0 to 1 based upon the level of darkness/brightness (figure III.6). This can serve as an input to our program. We can identify distinct features of each digit and write code which looks for these features in the input image. Our program will therefore contain a lot of *if*
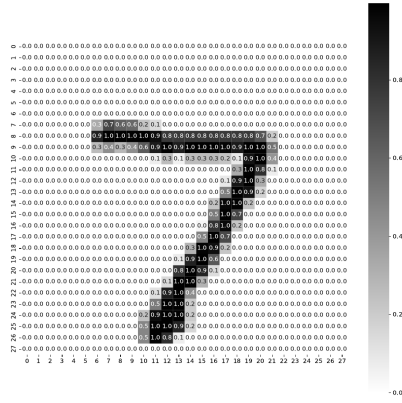
Figure III.6: A 28×28 pixel image of digit 7. Each pixel is assigned a value from 0 to 1.

statements and *for* loops to identify such features but more importantly will contain thousands of line of code to handle exceptions which arise from the fact that each digit can be written in a slightly different manner. It turns out that it is very difficult to express algorithmically the intuitions we have for recognizing digits. Soon we will realize the complexity of this task. What if we could write a program (for such fuzzy and difficult to-reason-about problems) that mimics the structure of our brain ?

III.2 Introduction to neural networks

Neural networks tackles this problem in a different way. The construction of neural network is inspired by the structure of our brain. The idea is to take a large number of handwritten digits
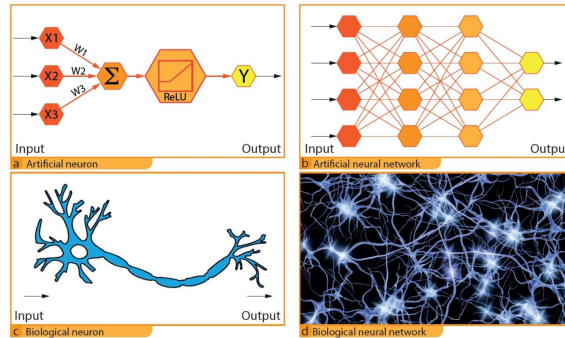

Figure III.7: Similarity between the structure of artificial and biological neuron [8].

together with labels and develop a system which can learn from these examples just like we learn. We leave it to the system to automatically infer rules from these examples and classify hand-written digits.
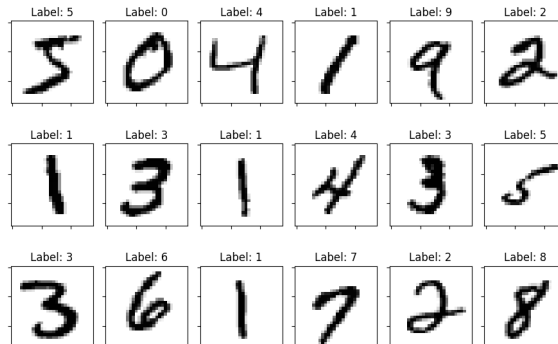

Figure III.8: Some training examples from MNIST dataset [9].

A neural network takes an input and gives an output. Figure III.9 shows a pictorial representation of a neural network which can be used to classify handwritten digits. It has layers (input, hidden, output) which consist of artificial neurons. The neurons are connected. Each neuron stores a value (for example from 0 to 1 in our case of handwritten digit recognition). A neuron
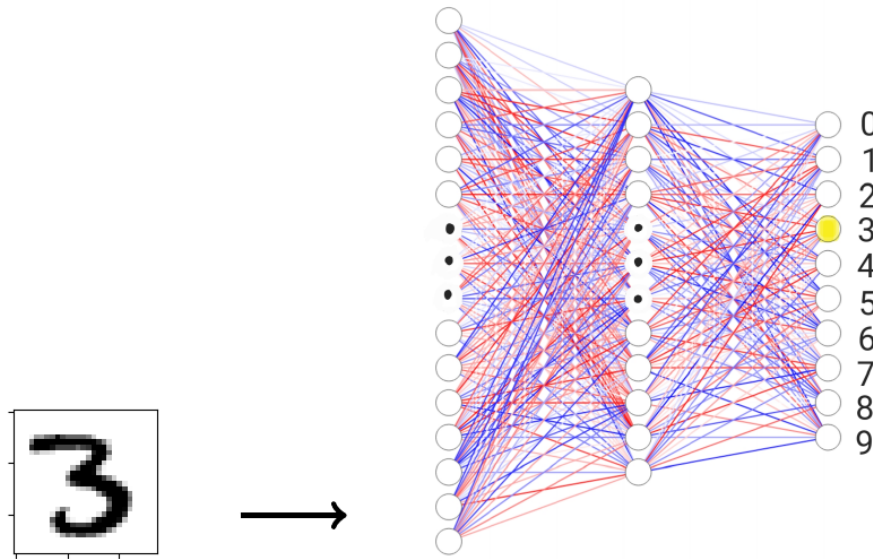

Figure III.9: A neural network to classify handwritten digits [10].

takes multiple inputs (say $a_1, a_2, ..., a_n$) and gives the output $\sigma(\sum_{i=1}^{n} a_i w_i + b)$ where σ is called an activation function. The inputs are multiplied by weights, w_i and added to a bias, b before passing through the activation funciton. The choice of activation function depends on the type of problem one is trying to solve. For example, for hand-written digit recognition one could use sigmoid/logistic function as an activation function.
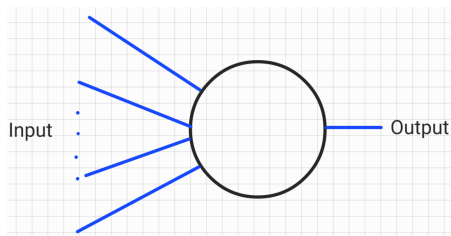

Figure III.10: A pictorial representation of an artificial neuron. It takes multiple inputs (say $a_1, a_2, ..., a_n$) and gives the output $\sigma(\sum_{i=1}^{n} a_i w_i + b)$.
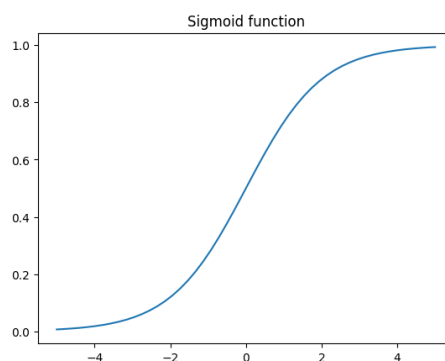

Figure III.11: A sigmoid/logistic activation function, $\sigma(z) = \frac{1}{1+e^{-z}}$

Mathematically speaking, a neural network is simply a function that takes an input which passes through its layers (made up of neurons which are mathematical functions) and results in an output. The neural network shown in figure III.9 has 784 neurons in the input layer (containing

data from a 28×28 pixel image), 15 neurons in the hidden layer and 10 neurons in the output layer. Each neuron in the output layer corresponds to a digit from 0 to 9; if the input image is of digit 3, only the fourth neuron in the outplayer will light up (output is close to 1) while the rest will remain dormant (outputs are closer to 0). Such neural networks are called feed-forward neural networks. When we look at the architecture of a neural network, many questions can come to our minds, for example,

1. Why does the network has a particular structure (i.e. layers with neurons) ?

2. What is the purpose of using an activation function ?

3. How does a neural network able to recognize hand-written digits ?

Until now, it is also not clear that why this approach (of using neural networks) even works. Even if we assume that this approach works, another question arises, which is, how do we get the weights, w_i and biases b_i of the network ? The answer to this question comes from a key ability of neural networks which is *learning*.

III.2.1 Learning from data

Learning is the process of obtaining suitable weights and biases such that a neural network can perform the desired task. There are several steps involved in learning. First step is to assign random values to weights and biases of the network and show the network some training examples. In the case of handwritten digit recognition we use the MNIST dataset [9]. Since the network is initialized with random weights and biases it will wrongly classify handwritten digits. Next step is
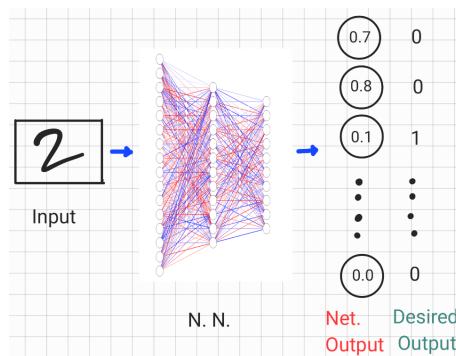

Figure III.12: Results from a randomaly initialized neural network and how it compares to the desired output for a training example

to tell the network to update its weights and biases in order to reduce the classfication error. This is done by introducing a cost/loss function which can quantify the classification error. We use a quadratic loss function,

$$C(w, b) = \frac{1}{N} \sum_x \|y(x) - a\|^2$$

where N denotes the number of training examples, $y(x)$ is the desired output corresponding to training example x, and a is the neural network output. Our aim is to find w, b such that the loss is minimum. The problem of learning from training data thus translates into solving an optimization problem. But this process raises another set of questions,

1. Which optimization algorithm to choose for minimizing the cost function?

2. How to define a suitable loss function ?

3. Does a global minimum exists for a chosen loss function ?

To answer these questions and the questions that we raised earlier, we will have to first understand the origins of neural networks.

III.2.2 Origins of neural networks

The artificial neurons we introduced earlier are called sigmoid neurons. In order to understand why sigmoid neurons are defined the way they are, we need to first understand another artificial neuron called perceptron. Perceptrons were developed in the 1950s and 1960s [11]. Unlike sigmoid neurons whose inputs, $x_i \in \mathbb{R}$, a perceptrons can take only binary inputs and produce a binary output. The mathematical model of a perceptron is,

$$\text{output} = \begin{cases} 0 & \text{if} \quad \sum_i w_i x_i \leq \text{ threshold} \\ 1 & \text{if} \quad \sum_i w_i x_i > \text{ threshold} \end{cases}$$

where x_i are the inputs and w_i are the weights. The output is 1 if $\sum_i w_i x_i$ exceeds a threshold value otherwise the output is 0. Perceptrons can be used as a device to make simple decisions by weighing up evidence [12]. The following example illustrates how perceptrons can be used to make a decision.

Example III.7. Suppose a Rock band is coming to perform in your city. You like Rock music and you are trying to decide whether or not to go to this event. You can make the decision by weighing up three factors:

1. Is the event nearby ?

2. Are your friends accompanying you ?

3. Is the weather good ?

We can assign binary variables x_1, x_2 and x_3 to these three factors. For example, if the event is nearby we would have $x_1 = 1$ but if it is far away we would have $x_1 = 0$. Similarly, if your friends are accompanying you, $x_2 = 1$ otherwise $x_2 = 0$. If the weather is good, $x_3 = 1$ and if the weather is bad, $x_3 = 0$. Now suppose, you absolutely love rock music and it doesn't really matter to you if your friends can accompany you or if the event is nearby but you really loathe bad weather and you won't go if the weather is bad. You can use perceptrons to model this kind of decision-making. One way is to choose, $w_3 = 6$ for the weather and $w_1 = 2, w_2 = 2$ for other conditions. The larger value of w_3 in comparison to w_1 and w_2 suggests that weather matters to you a lot. Finally, suppose you choose a thresold value of 5 for the perceptron. With these choices, the perceptron mimics the desired decision-making model, outputting 1 whenever the weather is good, and 0 whenever the weather is bad. By choosing different values for weights and the thresold, we can get different models of decision making.

It is obvious that the perceptron isn't a complete model of human descision-making. But it is plausible that if we use more layers in the network with more number of perceptrons, the network could make more complex decisions: We can write the mathematical model of a perceptron in a

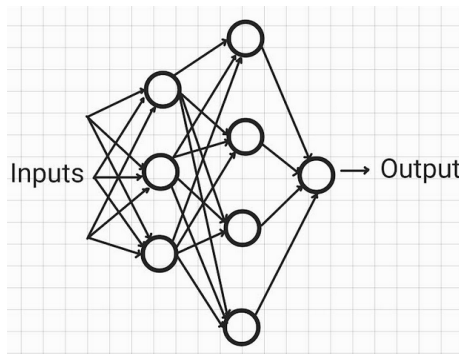Figure III.13: A network built out of perceptrons.

way familiar to us,

$$\text{output} = \begin{cases} 0 & \text{if} \quad \sum_i w_i x_i + b \leq 0 \\ 1 & \text{if} \quad \sum_i w_i x_i + b > 0 \end{cases}$$

where perceptron's bias, $b \equiv -\text{threshold}$. We have seen that a network of perceptrons can be used as a method for weighing evidence to make decisions. We can also use perceptrons to compute the

elementary logical functions such as AND, OR, and NAND [12]. The NAND gates are universal for computation, and therefore it follows that perceptrons are also universal for computation [12]. This property of perceptrons tells us that networks of perceptrons can be as powerful as any other computing device. These properties of perceptrons might make them an attractive option for solving decision-making problems but the property that we are really looking for in the network is the ability to tune its weights and biases in response to external stimuli (for example training data). We want our network to learn from data. In order to devise learning algorithms for a network, we would like that if we make a small change in some weight (or bias) in the network, the output changes only by a small amount. Schematically, this is what we want:
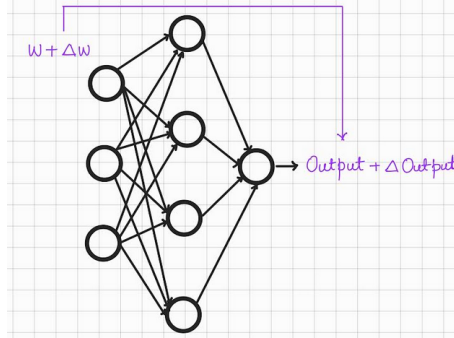
Unfortunately, a network of perceptrons doesn't have this capability. This is the reason why we use sigmoid neurons. Unlike perceptrons, the sigmoid neurons can take input values between 0 and 1. The activation function used in sigmoid neurons can be described as,

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

The smoothness of σ ensures that a small change in the input causes a small change in the output.

Figure III.14 shows an architecture of a typical (feed-forward) neural network. It consists of an input layer, multiple hidden layers and an output layer. The choice of the number of neurons in the input and output layers is governed by the problem one is trying to solve. For example, in the case of hand-wriiten digit recognition, if the input image consists of data from $28 \times 28 = 784$ pixels, the input layer will have 784 neurons. We want to classify the images into digits from 0 to 9 and therefore the output layer will have 10 neurons. The design (number of layers and number of neurons) of hidden layers is based on heuristics. Let us now look at a neural network (in detail) which can classify handwritten digits.

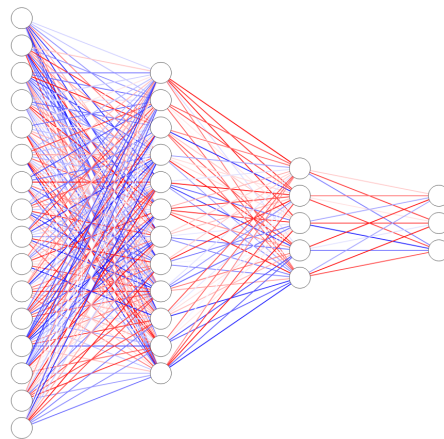Figure III.14: A feedforward neural network with an input layer $\in \mathbb{R}^{16}$, two hidden layers and an output layer $\in \mathbb{R}^2$ [10].

III.2.3   A network to classify handwritten digits

We use a network consisting of an input layer $\in \mathbb{R}^{784}$, one hidden layer and an output layer $\in \mathbb{R}^{10}$ to classify handwritten digits (see figure III.15). Let us try to understand how this neural network
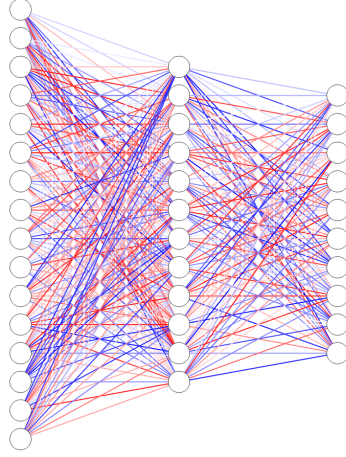
Figure III.15: A pictorial represenation of a neural network which can classify handwritten digits. It consists of an input layer $\in \mathbb{R}^{784}$ (only a few neurons are shown in the figure), one hidden layer and an output layer $\in \mathbb{R}^{10}$.

works. The main feature of this network is its ability to learn from data. We use MNIST data set which consists of scanned images of digits written by 250 people [9]. The MNIST data comes in two parts; a training data set containing 60000 imgaes and a test data set containing 10000 images. The images are in grescale and have size of 28×28 pixels. The MNIST data, along with hand-written digits, also provide correct labels corresponding to the digits. The input x to the
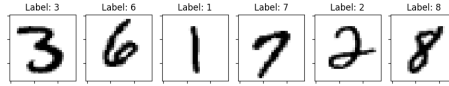

Figure III.16: A sample of training data from MNIST data set.

network is a $28 \times 28 = 784$ dimensional vector and the output a is a 10 dimensional vector. In order to quantify, the accuracy of this network's prediction, we define a cost/loss function,

$$C(w,b) = \frac{1}{N} \sum_x \|y(x) - a\|^2$$

where N denotes the number of training examples, $y(x)$ is the desired output (a 10 dimensional vector) corresponding to a training example x, and a is the neural network output. The idea is to find weights and biases which minimizes the difference between the desired output (labels) and the network prediction and we want this to happen for all training examples. For example, if an input x is an image of digit 6, we want our network's output a to be as close as possible to $y(x) = (0,0,0,0,0,0,1,0,0,0)^T$. The problem of minimizing the loss function is an optimization problem. In the context of neural networks, the most popular methods which are used to minimize the loss function are inspired from the gradient descent method.

Let us consider a simple example to understand the gradient descent method. Suppose we want to minimize a function $C(\mathbf{v})$ of two variables v_1 and v_2 (see figure III.17). We can use Calculus to calculate the gradients of $C(\mathbf{v})$ with respect to v_1, v_2 and use them to find places where C is minimum. This method could work well when C is a function of few variables but soon will turn into a nightmare when we increase the number of variables. Typically, the cost function of a neural network depends on a large number of weights and biases; in fact the biggest neural networks consists of billions of weigths and biases. Therefore, one has to come up with cost effective methods of finding the minima of cost functions. Grandient descent is one such method. It is an interative method, where one starts with an initial guess for variables and update them over multiple iterations in such a way that the cost function reduces with each iteration. Consider the case of cost function $C(\mathbf{v})$. For small changes Δv_1 and Δv_2 in variables v_1 and v_2 respectively, we can write the change in cost function ΔC as

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 = \nabla C \cdot \Delta \mathbf{v} \tag{III.4}$$
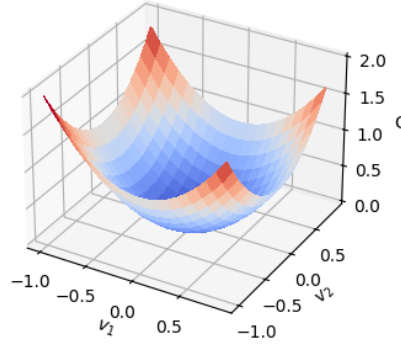
31

Figure III.17: A function C of two variables v_1 and v_2.

The aim is to find suitable change $\Delta \mathbf{v}$ which leads to reduction in C i.e. $\Delta C < 0$. One such choice is $\Delta \mathbf{v} = -\eta \nabla C$ provided $\eta > 0$,

$$\Delta C \approx \nabla C \cdot \eta \nabla C = \eta \|\nabla C\|^2 < 0$$

To conclude, in the gradient descent method we start with an initial guess $\mathbf{v}$ and update it by following

$$\mathbf{v} \to \mathbf{v}' = \mathbf{v} - \eta \nabla C$$

untill we reach the minimum of $C(\mathbf{v})$. In the context of neural networks, a gradient descent update might look like this,

$$
\begin{aligned}
w_j \to w_j' &= w_j - \eta \frac{\partial C}{\partial w_j} \\
b_k \to b_k' &= b_k - \eta \frac{\partial C}{\partial b_k}.
\end{aligned}
\tag{III.5}
$$

Note that the parameter η should be sufficiently small in order to justify the assumption in equation (III.4). We call this parameter the *learning rate*. The use of gradient descent method might seem as a good alternative to using analytical methods but there are still a number of challenges in applying this method. Notice that the cost function has the form $C = \frac{1}{N} \sum_x C_x$, that is, it's an average overs costs $C_x = \|y(x) - a\|^2$ for individual training examples. In order to make an update of gradient descent, we need to compute the gradients ∇C_x separately for each training input and then average them, $\nabla C = \frac{1}{N} \sum_x \nabla C_x$. This update can take extremely long time to make when the number of training inputs are very large, thus making the *learning* very slow.

We can overcome the issue of slow *learning* by using a variant of gradient descent method called *stochastic gradient descent (SGD)* method. The idea is to estimate the gradient ∇C by computing ∇C_x for a small sample of training examples and use it to make updates in weights and biases. The following steps are involved in using the stochastic gradient descent method,

1. Shuffle the training inputs and divide into batches of size m. Each batch (we call them mini-batches) consists of training examples say, $X_1, X_2, \ldots X_m$.

2. Calculate gradients for one mini-batch:

$$\frac{1}{m} \sum_{i=1}^{m} \nabla C_{X_i} \approx \frac{1}{N} \sum_x \nabla C_x = \nabla C$$

3. Make an update in the weights and biases:

$$w_j \to w_j' = w_j - \frac{\eta}{m} \sum_i \frac{\partial C_{X_i}}{\partial w_j}$$

$$b_k \to b_k' = b_k - \frac{\eta}{m} \sum_i \frac{\partial C_{X_i}}{\partial b_k}$$

4. Pick another mini-batch from the shuffled data and repeat steps 2 and 3 until all training inputs are exhausted. The end of this step is referred to as an *epoch* of training.

5. Repeat steps 1 to 5 until the network is sufficiently trained.

The use of SGD significantly increases the speed at which a network is trained. We may be making less accurate updates (hence the name *stochastic*) in the weights and biases in comparison to standard gradient descent method but nevertheless the updates are faster and over multiple iterations the cost (on an average) does go down.

III.2.4 Implementation of a neural network to classify digits

III.3 Backpropagation

Backpropagation is a fast algorithm for computing gradients [12]. Today, it is the workhorse of learning in neural networks. It uses chain-rule from Calculus to compute gradients $\frac{\partial C}{\partial w_i}, \frac{\partial C}{\partial b_i}$ and in the process reveals how changing the weights and biases changes the overall behaviour of the network. We will first see how backpropagation works for a very simple neural network and then generalize the procedure for big neural networks [13]. Consider a neural network having one neuron in the input layer, two hidden layers each with one neuron and one neuron in the output layer (see figure III.18). We can write the loss function as
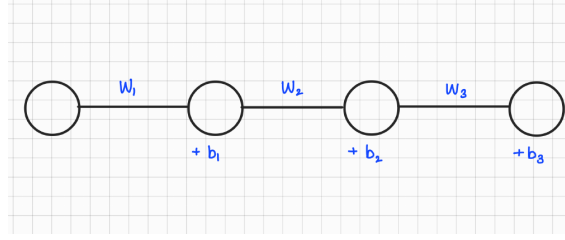

Figure III.18: A simple neural network with only neuron in each layer

$$C = \frac{1}{N} \sum_{k=0}^{N-1} C_k.$$

Let us focus on the cost correponding to one training example,

$$C_0 = (a^{(L)} - y)^2$$

where $a^{(L)}$ is the network output and y is the desired output. We label the activation/output from the last neuron with a superscript L, indicating which layer it's in, so the activation of the previous neuron becomes $a^{(L-1)}$. We can express the last activation $a^{(L)}$ by

$$a^{(L)} = \sigma(w^{(L)} a^{(L-1)} + b^{(L)})$$

where $w^{(L)}$ and $b^{(L)}$ denote the weight and bias of the last neuron respectively. We introduce a new variable z (called weighted sum) for the input to the activation function and keep the same superscript as the activation for labeling it:

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

We first calculate $\frac{\partial C_0}{\partial w^{(L)}}$ which gives us a measure of how changes in $w^{(L)}$ affects C_0. We use the chain-rule from Calculus and write,

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \tag{III.6}$$

The terms in equation (III.6) become clear if we refer to the tree of variables of the network (figure III.19). We can express the terms in equation (III.6) by

$$\frac{\partial z^{(L)}}{w^{(L)}} = a^{(L-1)} \quad \text{since } z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}) \quad \text{since } a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y) \quad \text{since } C_0 = (a^{(L)} - y)^2.$$

where $\sigma'(z^{(L)})$ denotes the derivative of the activation function. This reduces equation (III.6) to

$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y). \tag{III.7}$$

Next, we calculate the gradient of loss with respect to bias, $b^{(L)}$.
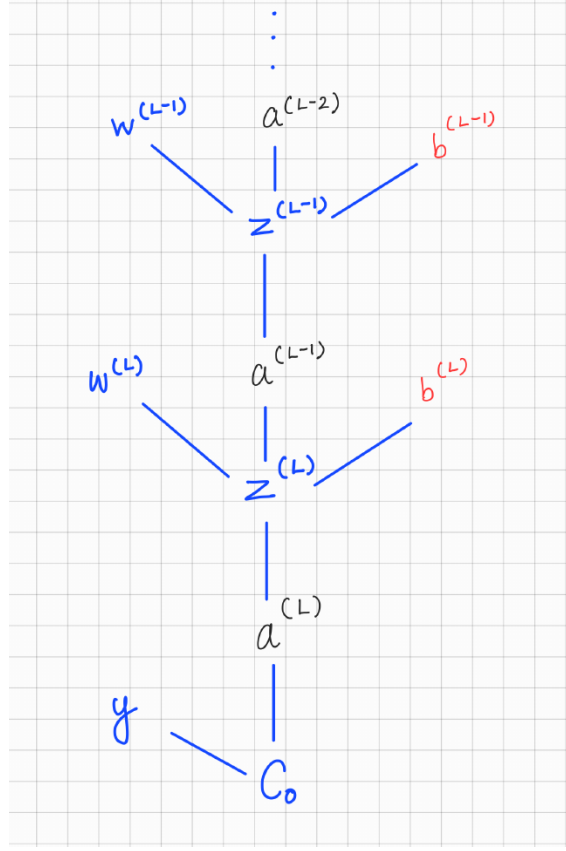

Figure III.19: This tree shows how the variables are connected in the neural network

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

We need to just calculate $\frac{\partial z^{(L)}}{\partial b^{(L)}}$ in order to find $\frac{\partial C_0}{b^{(L)}}$ since the other terms are already available to us from equation (III.6). Therefore, we can write,

$$\frac{\partial C_0}{\partial b^{(L)}} = 1 \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y). \tag{III.8}$$

At last, we calculate $\frac{\partial C_0}{\partial a^{(L-1)}}$.

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = w^{(L)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y). \tag{III.9}$$

The procedure to obtain the cost gradient for second-last layer variables is pretty straightforward now. We just replace the superscript L by $L - 1$.

$$\frac{\partial C_0}{\partial w^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial C_0}{\partial a^{(L-1)}}$$
$$\frac{\partial C_0}{\partial b^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial b^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial C_0}{\partial a^{(L-1)}} \tag{III.10}$$

We have already calculated $\frac{\partial C_0}{\partial a^{(L-1)}}$ in the previous step. For the remaining terms, we know that, $\frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} = a^{(L-2)}$, $\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} = \sigma'(z^{(L-1)})$ and $\frac{\partial z^{(L-1)}}{\partial b^{(L-1)}} = 1$ thus reducing the equations (III.10) to

$$\frac{\partial C_0}{\partial w^{(L-1)}} = a^{(L-2)} \cdot \sigma'(z^{(L-1)}) \cdot \frac{\partial C_0}{\partial a^{(L-1)}}$$
$$\frac{\partial C_0}{\partial b^{(L-1)}} = 1 \cdot \sigma'(z^{(L-1)}) \cdot \frac{\partial C_0}{\partial a^{(L-1)}} \tag{III.11}$$

We have now finished the calculation of all gradients of the network III.18. Although the above procedure is described for a network with only two hidden layers, we can easily generalize the methodology for networks with more hidden layers. In fact, the procedure is essentially the same even if we increase the number of neurons in each layer. We demonstrate this by considering a general neural network with multiple hidden layers each having multiple neurons (see figure III.20). We assign the following notations to weights, biases, weighted sums and activations of the network:
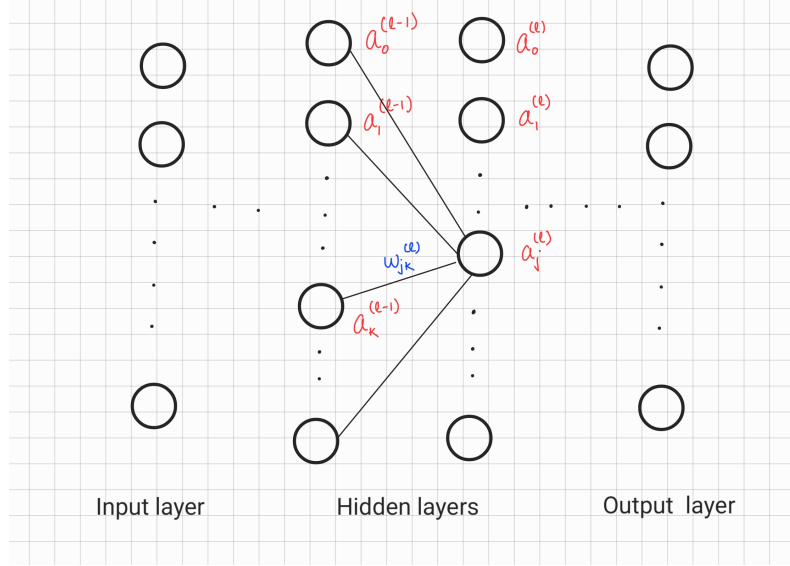

Figure III.20: Figure shows the notations assigned to different variables in a neural network while implementing backpropagation

- The activations are denoted by $a_j^{(l)}$ where the superscript represents the layer number and subscript represents the neuron number. The layers are numbered starting from $l = 1$ (the first layer i.e. input layer) while the neurons are numbered starting from $j = 0$ (the first neuron).

- The weight for the connection between kth neuron of layer $l-1$ and jth neuron of layer l is denoted by $w_{jk}^{(l)}$. The bias for the jth neuron in layer l is denoted by $b_j^{(l)}$.

- The weighted sums are denoted by $z_j^{(l)}$ which are passed to an activation function resulting in activation $a_j^{(l)}$.

We have the following expressions for the cost function (for one training example), activations and weighted sums:

$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2, \quad a_j^{(l)} = \sigma(z_j^{(l)}), \quad z_j^{(l)} = \sum_{k=0}^{n_{l-1}-1} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}$$

We can now derive the gradients following the chain-rule. We start by first calculating the gradients for the last layer,

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$
$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

(III.12)

We also derive the gradient with respect to activations in the second-last layer,

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

We can evaluate the gradients $\frac{\partial C_0}{\partial w_{jk}^{(L-1)}}$ and $\frac{\partial C_0}{\partial b_j^{(L-1)}}$ using $\frac{\partial C_0}{\partial a_j^{(L-1)}}$. In this way we can keep moving backwards in the network (hence the name backpropagation) and find gradients on the way back.

Notice that the only major change we see in the expressions for gradients when moving to complex neural networks from simple networks is for $\frac{\partial C_0}{\partial a_k^{(L-1)}}$. This is attributed to that fact that now we have multiple neurons in each layer. A change in the activation, say $\frac{\partial C_0}{\partial a_k^{(L-1)}}$, can affect the cost through multiple paths. In general, we can write the derivatives of cost/loss function with respect to weights and biases for neurons in any layer l as,

$$
\begin{aligned}
\frac{\partial C_0}{\partial w_{jk}^{(l)}} &= a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C_0}{\partial a_j^{(l)}} \\
\frac{\partial C_0}{\partial b_j^{(l)}} &= \sigma'(z_j^{(l)}) \frac{\partial C_0}{\partial a_j^{(l)}}
\end{aligned}
\tag{III.13}
$$

where

$$
\frac{\partial C_0}{\partial a_k^{(l)}} = \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C_0}{\partial a_j^{(l+1)}} \quad \text{or} \quad 2(a_k^{(L)} - y_k) \text{ when } l = L.
$$

The backpropagation algorithm is generally combined with a learning algorithm like stochastic gradient descent to update weights and biases. Let us see how this works for a mini-batch of m training examples:

1. Input a set of training examples.

2. For each training example x: Set the corresponding input activation $a^{x,1}$ and perform the following steps:

   - Feedforward: For each $l = 2, 3, ...L$, compute $z_j^{(l)} = \sum_{k=0}^{n_{l-1}-1} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}$ and $a_j^{(l)} = \sigma(z_j^{(l)})$
   - Compute gradients for the last layer:

   $$
   \frac{\partial C_x}{\partial a_j^{(L)}}, \frac{\partial C_x}{\partial w_{jk}^{(L)}}, \frac{\partial C_x}{\partial b_j^{(L)}}
   $$

   - Backpropagate: for $l = L-1, L-2, ..., 2$ compute

   $$
   \frac{\partial C_x}{\partial a_j^{(l)}}, \frac{\partial C_x}{\partial w_{jk}^{(l)}}, \frac{\partial C_x}{\partial b_j^{(l)}}
   $$

3. Gradient descent: for each $l = L, L-1, ..., 2$ update weights and biases

$$
w_{jk}^l \rightarrow w_{jk}^l - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial w_{jk}^{(l)}}
$$

$$
b_j^l \rightarrow b_j^l - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial b_j^{(l)}}
$$

III.3.1   Why backpropagation is a fast algorithm ?

In the last section, we have illustrated the procedure to find gradient of the cost function. At first glance, the backpropagation algorithm may seem complicated since it involves a lot of steps and one has to be very careful about the notations. In reality, the backpropagation algorithm is very simple to implement and is considered one of the most efficient methods for computing gradients. Most of the terms like $a_k^{(l-1)}, w_{jk}^{(l+1)}$, in the expressions for gradients are already available to the network when making a forward pass; the rest, for example $\sigma'(z_j^{(l)})$ can be easily computed. We can see the advantages of using backpropagation when we compare it to another popular method for calculating gradients. Let us denote (for simplicity) the cost function by $C(w, b)$ where w are the weights $w_1, w_2, ...$ and b are the biases $b_1, b_2, ...$ of a network. We can use the fundamental definition of partial derivatives to approximate gradients,

$$
\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j, b) - C(w, b)}{\epsilon}
\tag{III.14}
$$

where $\epsilon > 0$ is a small positive number, and e_j is a unit-vector in the $j-$th direction. In other words, we can compute $\frac{\partial C}{\partial w_j}$ by first calculating the cost C for two slightly different values of w_j and then applying equation (III.14). We can use this approach to calculate derivatives with respect to all weight and biases of the network. This method is extremly easy to implement and certainly looks better than the chain-rule. Therefore, one might be tempted to use this instead of backpropagation. Unfortunately, this method is extremely slow !

Suppose we have a million (10^6) weights and biases in a network. In order to get gradients using equation (III.14), we have to compute the cost $10^6 + 1$ times; once to compute $C(w)$ and 10^6 times to calculate $C(w + \epsilon e_j, b)$ for different j's. This is equivalent to making $10^6 + 1$ forward passes of the network. On the other hand, with backpropagation, we need to make just one forward pass and one backward pass to calculate all the gradients. The computational cost of the backward pass is similar to the forward pass since both involve similar computations. Therefore, rougly speaking, (for this example) the backpropagation method is a milion times faster than the numerical gradient computation method, (III.14). The backpropagation is also more accurate than the numerical gradient method. The expression for gradient used in equation (III.14) is an approximation of gradients dependent on the value of ϵ whereas the expressions used in backpropagation are exact.

III.4 The universal approximation theorem

Neural networks are mathematical functions. We use training data to look for suitable weights and biases such that the neural network can map/obtain the underlying function from which the data is drawn. But, how can one believe that a neural network have the capability of finding the underlying function. What is so special about the neural network structure that after training it is able to give us great results on unseen data/ test data ? The answer to these questions lies in the universal approximation theorem (UAT) proposed by George Cybenko in 1989. George Cybenko gave a rigorous proof that feed-forward neural networks can approximate any continuous function defined on a compact set. To understand the proof of this theorem, one should be familiar with some concepts from measure theory and functional analysis. Let us first briefly go through those mathematical preliminaries.

- Consier an n-dimensional unit cube, $I_n = [0,1]^n$. In the universal approximation theorem, the continuous functions that a neural network aims to approximate are defined on compacts sets, I_n i.e. the domain of continuous functions is I_n.

- Let $C(I_n)$ denote the space of continuous functions on I_n. For any function $f \in C(I_n), f : [0,1]^n \to \mathbb{R}$, one can define a suitable norm, $\|f\| = \sup_{x \in I_n} |f(x)|$ known as uniform/supremum norm. We can associate a dual space to $C(I_n)$ defined as $C(I_n)* = \{L : C(I_n) \to \mathbb{R}\}$ where L is a linear operator.

- UAT is defined for feed-forward neural networks having an input layer (with multiple neurons), one hidden layer and an output layer with only one neuron (see figure III.21). Therefore, the functions generated by the neural network can be described as

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j \cdot x + b_j), \quad w_j \in \mathbb{R}^n, \alpha_j, b_j \in \mathbb{R}, x \in I_n$$

where N is the number of neurons in the hidden layer. We can note that bias is not added to output from the hidden layer. Also the output is not passed to an activation function. We denote the set of functions of the form $G(x)$ by $\mathcal{N}$. In the problem of hand-written digit recognition, we assumed that the activation function, σ is logistic/sigmoid. However, UAT was proved for a general class of activation functions. The actvation functions are assumed to be continuous and discriminatory.

Definition III.17 (Discriminatory function)**.** We say that σ is discriminatory if for a measure $\mu \in M(I_n)$ (where $M(I_n)$ is the space of finite, signed regular Borel measures on I_n)

$$\int_{I_n} \sigma(\mathbf{w} \cdot \mathbf{x} + b) \, d\mu(\mathbf{x}) = 0$$

for all $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ implies that $\mu = 0$.
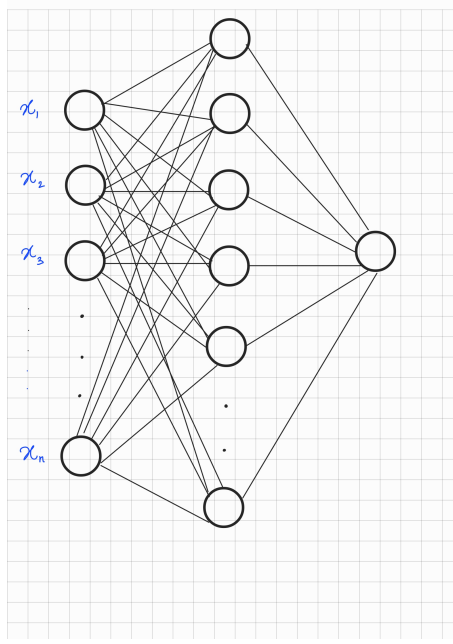
Figure III.21: A feed-forward neural network with an input layer (having multiple neurons), one hidden layer and an output layer (having only one neuron).

Theorem III.7 (Universal approximation theorem)**.** *Let σ be any continuous discriminatory function. Then the set $\mathcal{N}$ of functions of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j \cdot x + b_j), \quad w_j \in \mathbb{R}^n, \alpha_j, b_j \in \mathbb{R}, x \in I_n$$

are dense in $C(I_n)$. In other words, given any function $f \in C(I_n)$ and $\epsilon > 0$, there exists a $G(x) \in \mathcal{N}$ such that

$$\|G(x) - f(x)\| < \epsilon \text{ for all } x \in I_n$$

Before going into the proof of UAT, we will familiarize ourselfves with some key ideas from measure theory.

- Our aim to measure subsets of $I_n = [0,1]^n$. A collection $\sum$ of subsets of I_n is called a σ-algebra if

  1. $I_n \in \sum$
  2. If $A \in \sum$, then $A^c = I_n \backslash A \in \sum$
  3. If $A_1, A_2, ...$ is a countable collection of subsets of $\sum$ then $\bigcup^{\infty} A_i \in \sum$.

- Borel σ-algebra on I_n is the smallest σ-algebra containing all open sets in I_n. Sets in Borel σ-algebra are called Borel sets.

- A finite signed Borel measure μ on I_n is a real valued function, $\mu : \sum \to \mathbb{R}$ (where $\sum$ is a Borel σ-algebra defined on I_n) such that

  1. $\mu(\phi) = 0$
  2. $\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i), A_i$ are disjoint sets
  3. $\mu(A_i) < \infty$

- A Borel measure is regular if

$$\mu(A) = \sup\{\mu(K)|K \subseteq A, K \text{ is compact}\}$$
$$\mu(A) = \inf\{\mu(U)|A \subseteq U, U \text{ is open}\}$$

In other words, what regular means is every measurable set can be approximated from above by open measurabel sets and from below by compact measurable sets.

- We define $M(I_n)$ as the set of all finite signed regular Borel measures on I_n.

The proof of UAT also uses ideas from Hahn-Banach theorem and Riesz representation theorem. These theorems have various equivalent versions. We now define the versions which we will refer to later.

Theorem III.8 (Riesz representation theorem). *If $K \subset \mathbb{R}^d$ is a compact set, then every linear functional L defined on $C(K)$ can represented by a unique regular signed Borel measure $\mu \in M(K)$ in the sense that,*

$$L(f) = \int_K f \ d\mu$$

where $f : K \to \mathbb{R}$.

Theorem III.9 (Hahn-Banach theorem). *Let M be a linear subspace of a normed linear space X, and $f_0 \in X$. Then f_0 is in the closure $\overline{M}$ of M if and only if there is no bounded linear functional L on X such that $L(f) = 0$ for all $f \in M$ but $L(f_0) \neq 0$.*

We now give the proof of universal approximation theorem III.7.

Proof. $C(I_n)$ is a vector space equipped with a supremum norm and hence is a normed vector space. It is clear that the set formed by functions of the form $G(x)$ is a subset of $C(I_n)$ i.e. $\mathcal{N} \subset C(I_n)$. We claim that $\mathcal{N}$ is also a linear subspace of $C(I_n)$. This is true due to the fact that if we take two functions $G_1(x), G_2(x) \in \mathcal{N}$, we can always find a function $G_3(x) \in \mathcal{N}$ such that $G_3(x) = \alpha G_1(x) + \beta G_2(x)$ for all $\alpha, \beta \in \mathbb{R}$. Our next claim is that the closure of $\mathcal{N}$ is all of $C(I_n)$ i.e. $\mathcal{N}$ is dense in $C(I_n)$. We prove this by contradiction.

Assume that the closure of $\mathcal{N}$, $\overline{\mathcal{N}}$ is not all of $C(I_n)$ i.e. $\overline{\mathcal{N}} \neq 0$. It can be proved (refer any standard functional analysis book) that $\overline{\mathcal{N}}$ is a closed proper subspace of $C(I_n)$. By Hahn Banach theorem III.9, there exists a bounded linear functional on $C(I_n)$, let's call it L, with the property that $L \neq 0$ but $L(\overline{\mathcal{N}}) = L(\mathcal{N}) = 0$. By following Riesz representation theorem III.8, we can write this linear operator, L as

$$L(h) = \int_{I_n} h(x) \ d\mu(x)$$

for some $\mu \in M(I_n) \forall h \in C(I_n)$. Since this operator returns 0 for all elements of $\mathcal{N}$, we must have for all $w \in \mathbb{R}^n, b \in \mathbb{R}$

$$\int_{I_n} \sigma(w \cdot x + b) \ d\mu(x) = 0.$$

However, we assumed that σ is a discriminatory functions therefore this condition implies that $\mu = 0$. This contradicts our initial assumption that $\overline{\mathcal{N}} \neq 0$ which led to $L \neq= 0 \equiv \mu \neq 0$ via Hahn Banach theorem. Hence, the subspace $\mathcal{N}$ must be dense in $C(I_n)$. $\qquad\square$

The universal approximation theorem gives us confidence that neural networks can indeed approximate any continuous function provided that the activation function is continuous and discriminatory. However, it is not clear how to build activation functions which are discriminatory.

Definition III.18. We say that σ is sigmoidal if

$$\sigma(t) \to \begin{cases} 1 & \text{as} \ \ t \to +\infty \\ 0 & \text{as} \ \ t \to -\infty. \end{cases}$$

Lemma III.6. *Any continuous sigmoidal function is discriminatory.*

Proof. Refer Cybenko paper. $\qquad\square$

The lemma III.6 gives us a way to build suitable activation functions. In fact, the sigmoid/logistic function we used as an activation function in the neural network for solving handwritten recognition problem satisfies the conditions of definition III.18 and is therefore, discriminatory. Although logistic function is monotonically increasing, no monotonicity is required by definition III.18.

Kernel based approximation

Kernel based methods used to be one of the main tools for machine learning up about 15 years ago, when neural networks became more popular. Nowadays they are still in use due to their rich theoretical background. Recent interest was also spiked due to the so-called neural tangent kernel (NTK).

The first part on the following sections on kernel based approximation in mainly based on [14], while the last part on the neural tangent kernel is based on recent literature [15].

IV.1   Introduction & Motivation

We start by discussing the following problem:

Problem IV.1. *Consider $\Omega \subset \mathbb{R}^d$ and pairwise distinct points $X_N := \{x_i\}_{i=1}^N \subset \Omega$ and target values $\{f_i\}_{i=1}^N \subset \mathbb{R}$. Find a continuous function $s : \Omega \to \mathbb{R}$ that interpolates these values, i.e.*

$$s(x_i) = f_i, \qquad i = 1, ..., N. \tag{IV.15}$$

There are two key important ingredients:

- Multivariate: We are not only interested in the univariate case $d = 1$, but especially in the multivariate case $d > 1$.

- Meshless: The points X_N may be arbitrarily unstructured and scattered, in particular we do not need any mesh.

As a first example to solve this problem, we can consider the univariate case $d = 1$: For the approximation of our problem, we pick an ansatz space V with basis $\{\phi_j\}_{j=1}^N \subset V$ and use the ansatz function

$$s(x) = \sum_{j=1}^N \alpha_j \phi_j(x).$$

This ansatz function can be plugged into the system of equations from Eq. (IV.15), which yields the linear equation system

$$A_{\phi,X_N}\alpha := \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_N(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_N(x_N) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}. \tag{IV.16}$$

In order to make this more explicit, we pick the space of polynomials of degree $N-1$ as our ansatz space, i.e. $V := \pi_{N-1}(\mathbb{R})$, and as basis we choose the monomials $\{\phi_j\}_{j=1}^N = \{x^{j-1}\}_{j=1}^N \subset \pi_{N-1}(\mathbb{R})$. Thus the linear equation system Eq. (IV.16) turns into

$$A_{\phi,X_N}\alpha := \begin{pmatrix} 1 & x_1 & \dots & x_1^{N-1} \\ 1 & x_2 & \dots & x_2^{N-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & \dots & x_N^{N-1} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}.$$

The occuring matrix A_{ϕ,X_N} is the so-called Vandermonde matrix, and we recall that its determinant is given as

$$\det(A_{\phi,X_n}) = \prod_{1 \le i < j \le N} (x_j - x_i).$$

Like this we can easily see, that the linear equation system Eq. (IV.16) is uniquely solvable as soon as the points X_N are pairwise distinct.

This example raises the following question:

Question IV.1.1. *Does this procedure – fixing a basis, and then doing interpolation with arbitrary points X_N – always work, also in higher dimensions?*

The answer to this question will be *no*, as we will see in the Mairhuber Curtis theorem **??**. In order to state and proof this theorem, we first need to introduce some mathematical notation:

Definition IV.19. Let $\Omega \subset \mathbb{R}^d$ contain at least N points and let $V \subseteq \mathcal{C}(\Omega)$ be an N-dimensional linear space.
The space V is called a Haar space of dimension N on Ω, if for arbitrary points $\{x_i\} \subset \Omega$ and arbitrary values $\{f_i\}_{i=1}^n \subset \mathbb{R}$ there exists a unique function $s \in V$ that satisfies

$$s(x_i) = f_i \qquad i = 1, ..., N.$$

The polynomial spaces which we encountered before are a first example of such a Haar space:

Example IV.8. Let $\Omega = [a, b] \subset \mathbb{R}$ for $a < b \in \mathbb{R}$. Then $V = \pi_{N-1}(\Omega)$ (space of polynomials up to degree $N - 1$ on Ω) is a Haar space of dimension N of Ω.

The following proposition gives several equivalent statements on Haar spaces:

Proposition IV.1. *Let $\Omega \subset \mathbb{R}^d$ contain at least N points and let $V \subseteq \mathcal{C}(\Omega)$ be an N-dimensional linear space.*
Then the following statements are equivalent:

1. *V is an N-dimensional Haar space on Ω*

2. *Every $v \in V \setminus \{0\}$ has at most $N - 1$ distinct zeros*

3. *For any set of pairwise distinct points $X_N \subset \Omega$ and any basis $\{\phi_j\}_{j=1}^N$ of V, the interpolation matrix A_{ϕ, X_N} is invertible, i.e. $\det(A_{\phi, X_N}) \neq 0$.*

Proof.
- 1. $\Rightarrow$ 2.: We assume that V is an N-dimensional Haar space on Ω, and that there is a $v \in V \setminus \{0\}$ with (at least) N distinct zeros $X_N \subset \Omega$. Additionaly we consider $u := 0$ (zero-function) and observe that $u \in V$ (as V is a linear space). Especially we have $u \neq v$. Now, both u and v are distinct interpolants for the points X_N with target values $f_i = 0$ for $i = 1, ..., N$, which is a contradiction to the uniqueness of the interpolant within the definition of a Haar space.

- 2. $\Rightarrow$ 3.: We assume that there is a set of pairwise distinct points $X_N \subset \Omega$ and a basis $\{\phi_j\}_{j=1}^N$ of V such that the interpolation matrix is not invertible, i.e. it is singular. Then there is a vector $\alpha \in \mathbb{R}^N \setminus \{0\}$ such that $A_{\phi, X_N} \alpha = 0$. As $\alpha \neq 0$, the function $v(x) := \sum_{j=1}^N \alpha_j \phi_j(x)$ is not the zero function. However v has N distinct zeros (namely in the points X_N), which is thus a contradiction:

$$v(x_i) = \sum_{j=1}^N \alpha_j \phi_j(x_i) = (A_{\phi, X_n} \alpha)_i = 0 \qquad \text{for all } i = 1, ..., N.$$

- 3. $\Leftrightarrow$ 1.: The matrix A_{ϕ, X_N} is invertible if and only if $A_{\phi, X_N} \alpha = b$ has a unique solution of any $b = \{f_i\}_{i=1}^N$, which holds if and only if $\sum_{j=1}^N \alpha_j \phi_j(x_i) = f_i, i = 1, ..., N$ has a unique solution $\alpha = \{\alpha_j\}_{j=1}^N$, which holds if and only if $s(x) := \sum_{j=1}^N \alpha_j \phi_j(x)$ is the unique interpolant. $\square$

With these background knowledge on Haar spaces, we can frame Question IV.1.1 more precisely:

Question IV.1.2. *Is there any Haar space in dimension $d > 1$?*

As announced already before, the answer to this question will be no, as established by the famous Mairhuber-Curtis theorem:

Theorem IV.10 (Mairhuber-Curtis). *Let $\Omega \subset \mathbb{R}^d$, $d > 1$ be a set with nonempty interior. Then there exists no Haar space of dimension $N > 1$ on Ω.*

Proof. For any N-dimensional space $V \subset \mathcal{C}(\Omega)$, we will show the existence of a set $X_N \subset \Omega$ of pairwise distinct points, such that the interpolation matrix has determinant zero and is thus singular, which is a contradiction to Proposition IV.1:

We assume $V := \mathrm{span}\{\phi_1, ..., \phi_N\} \subset \mathcal{C}(\Omega)$ to be a Haar space of dimension $N > 1$ on Ω. As Ω has a nonempty interior, there exists a ball $B(x_0, \varepsilon) \subset \Omega$ for some $x_0 \in \Omega, \varepsilon > 0$. Now we consider pairwise distinct points $X_N = \{x_1, ..., x_N\} \subset \Omega$ with $x_1, x_2 \in B(x_0, \varepsilon)$. As V is assumed to be a Haar space, we have $\det(A_{\phi, X_N}) \neq 0$ by Proposition IV.1.

Now we consider continuous curves $\gamma_1, \gamma_2 : [0, 1] \to B(x_0, \varepsilon) \subset \Omega$ with $\gamma_1(0) = \gamma_2(1) = x_1$, $\gamma_2(0) = \gamma_1(1) = x_2$ that do not intersect in any other points, that do not intersect themselves and that have no points in common with $\{x_3, ..., x_N\}$. This is possible due to the assumption $d > 1$. Like this, $X_N(t) := \{\gamma_1(t), \gamma_2(t), x_3, ..., x_N\}$ are pairwise distinct for any $t \in [0, 1]$. Due to Proposition IV.1, we have $\det(A_{\phi, X_N(1)}) \neq 0$.

Now we consider the function $D : [0, 1] \to \mathbb{R}, t \mapsto \det(A_{\phi, X_N(t)})$ and show that it changes sign: As $A_{\phi, X_N} = A_{\phi, X_N(0)}$ and $A_{\phi, X_N(1)}$ only differ by a permuation of the first two rows, we have $D(1) = -D(0)$ and thus $D(0)D(1) < 0$. As the function $D(t)$ is continuous, there needs to be a $\tilde{t} \in (0, 1)$ such that $D(\tilde{t}) = 0$. This means that $X_N(\tilde{t})$ is a set of pairwise distinct points s.t. $A_{\phi, X_N(\tilde{t})}$ has determinant zero, which is a contradiction to Proposition IV.1. $\square$

The Mairhuber-Curtis theorem from Theorem IV.10 essentially tells us, that it is not possible to choose the approximation space V a-priori, i.e. independent of the points X_N. In order to solve this issue, we will make use of kernels, which will allow us to use data-dependent approximation spaces.

IV.2   Kernels

In this section, we will introduce the notion of a kernel as well as special classes of kernels and discuss their basic properties. Finally we will see how interpolation can be done with kernels in a well defined way, circumventing the Mairhuber-Curtis theorem Theorem IV.10.

We start with the basic definition of a kernel:

Definition IV.20. Let Ω be a nonempty set.

- A <u>real-valued kernel on Ω</u> is a symmetric function $k : \Omega \times \Omega \to \mathbb{R}$.

- A <u>complex-valued kernel on Ω</u> is a hermitian function $k : \Omega \times \Omega \to \mathbb{R}$.

In the following we will mainly focus on real-valued kernels and on kernel which are defined on $\Omega \subseteq \mathbb{R}^d$. However we note that kernels can be defined on very general sets that need not posses any structure, such as graphs, molecules or images.

We continue with the definition of a kernel matrix, which enables us to introduce the notion of positive definite (PD) and strictly positive-definite (SPD) kernels:

Definition IV.21. Let Ω be a nonempty set.

- For any $N \in \mathbb{N}$ and any set of N pairwise distinct points $X_N := \{x_i\}_{i=1}^N \subset \Omega$, the <u>kernel matrix</u> $A := A_{k, X_N} \in \mathbb{R}^{N \times N}$ is defined as $A := [k(x_i, x_j)]_{i,j=1}^N$.

- A kernel k on Ω is <u>positive definite</u> (PD), if for all $N \in \mathbb{N}$ and all pairwise distinct points X_N the kernel matrix A is positive semi-definite, i.e.

$$\sum_{i,j=1}^N \alpha_i \alpha_j k(x_i, x_j) = \alpha^\top A \alpha \geq 0 \qquad \forall 0 \neq \alpha = \{\alpha_i\}_{i=1}^N \in \mathbb{R}^N.$$

- A kernel k on Ω is <u>strictly positive definite</u> (SPD), if for all $N \in \mathbb{N}$ and all pairwise distinct points X_N the kernel matrix A is positive definite, i.e.

$$\sum_{i,j=1}^N \alpha_i \alpha_j k(x_i, x_j) = \alpha^\top A \alpha > 0 \qquad \forall 0 \neq \alpha = \{\alpha_i\}_{i=1}^N \in \mathbb{R}^N.$$

There is another important class of kernels, namely conditionally positive definite kernels, which are however not discussed in the following. It is worth to note, that some literature (e.g. [14]) does not use "*strictly* positive definite" kernels, but uses directly "positive definite" kernels.

In order to show why the notion of a strictly positive definite kernel is important, we recall the following statement from linear algebra:

Proposition IV.2. *Let $A \in \mathbb{R}^{N \times N}$ be a symmetric matrix. Then the following are equivalent:*

1. A is positive definite, i.e. $\alpha^\top A\alpha > 0$ for all $0 \neq \alpha \in \mathbb{R}^N$.

2. The eigenvalues $\{\lambda_i\}_{i=1}^N$ of A are positive

In this case, it holds:

- A is invertible with $\det(A) > 0$.

Proof. As A is symmetric, the spectral theorem gives an eigen-decomposition as $A = V\Lambda V^\top$ with $V, \Lambda \in \mathbb{R}^{N \times N}$, V orthogonal and Λ diagonal with the eigenvalues on its diagonal:

- 1. $\Rightarrow$ 2.: As V is orthogonal, its columns V_i form an orthonormal basis. Thus we can decompose any $0 \neq \alpha \in \mathbb{R}^N$ as $\alpha = \sum_{i=1}^N \beta_i V_i = V\beta$. Then we calculate:

$$\alpha^\top A\alpha = (\beta^\top V^\top)A(V\beta) = \beta^\top V^\top V\Lambda V^\top V\beta = \beta^\top \Lambda\beta = \sum_{i=1}^N \beta_i^2 \lambda_i.$$

  Thus $\lambda_i > 0$ implies $\alpha^\top A\alpha > 0$ for all $\alpha \neq 0$.

- 2. $\Rightarrow$ 1.: We have $\alpha^\top A\alpha > 0$ for any $\alpha \neq 0$, thus we pick $\alpha := V_i$ (column of V as specified above), and have

$$0 < V_i^\top A V_i \lambda_i V_i^\top V_i = \lambda_i.$$

If all the eigenvalues are positive, also the determinant is positive due to $\det(A) = \prod_{i=1}^N \lambda_i$. $\square$

Definition IV.21 in conjunction with Proposition IV.2 now allows us to show how strictly positive definite kernels can be used for our intepolation problem Problem IV.1:

Theorem IV.11. *Let $\Omega \subset \mathbb{R}^d$ and let k be a SPD kernel on Ω. For any set $\{x_i\}_{i=1}^N \subset \Omega$ of pairwise distinct points and corresponding target values $\{f_i\}_{i=1}^N \subset \mathbb{R}$, there exists a unique kernel interpolant*

$$s(x) := \sum_{j=1}^N \alpha_j k(x, x_j)$$

that satisfies $s(x_i) = f_i$ for all $i = 1, ..., N$.

Proof. The interpolation conditions $s(x_i) = f_i$ for all $i = 1, ..., N$ yield to the linear system of equations

$$A_{k,X_N}\alpha := \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \dots & k(x_1,x_N) \\ k(x_2,x_1) & k(x_2,x_2) & \dots & k(x_2,x_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_N,x_1) & k(x_N,x_2) & \dots & k(x_N,x_N) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}. \qquad \text{(IV.17)}$$

The matrix A_{k,X_N} is exactly the kernel matrix from Definition IV.21, and due to the assumption on the kernel to be SPD, this matrix is positive definite. By Proposition IV.2 we have $\det(A_{k,X_N}) > 0$, thus this linear system of equations is uniquely solvable. $\square$

As we saw that (strictly positive definite) kernels can be used for interpolation, we will give two examples of kernels:

Example IV.9.
- The linear kernel is defined as $k : \mathbb{R}^d \times \mathbb{R}^d : (x, z) \mapsto (x, z)_{\mathbb{R}^d}$. It simply returns the inner product of two points $x, z \in \mathbb{R}^d$. It is easy to see, that the linear kernel is positive definite kernel, but not strictly positive definite.

- The Gaussian kernel is defined as $k : \mathbb{R}^d \times \mathbb{R}^d : (x, z) \mapsto \exp(-\|x - z\|^2)$. It makes use of the negative squared exponential of the distance between two data points, and is thus a first example of a "radial basis function kernel". The Gaussian kernel can be shown to be strictly positive definite.

The linear kernel was a first example of a kernel, which is defined with help of a "feature map". For these kind of kernels, one can directly show the positive definiteness:

Theorem IV.12. *Let Ω be a set, H a Hilbert space and $\phi : \Omega \to H$ a mapping (<u>feature map</u>). Then the kernel*

$$k(x, z) := (\phi(x), \phi(z))_H$$

is positive definite.

Proof. We verify the condition from the definition in Definition IV.21:

$$\sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) = \sum_{i,j=1}^{N} \alpha_i \alpha_j (\phi(x_i), \phi(x_j))_H = (\sum_{i=1}^{N} \alpha_i \phi(x_i), \sum_{j=1}^{N} \alpha_j \phi(x_j))_H$$

$$= \left\| \sum_{i=1}^{N} \alpha_i \phi(x_i) \right\|_H^2 \geq 0.$$

$\square$

We continue with some further elementary properties on kernels:

Proposition IV.3. *Let $k : \Omega \times \Omega \to \mathbb{R}$ be a PD kernel. Then:*

- *Non-negativity of the diagonal:*

$$k(x, x) \geq 0 \qquad for\ all\ \ x \in \Omega$$

- *Cauchy-Schwarz inequality:*

$$k(x, z)^2 \leq k(x, x)k(z, z) \qquad for\ all\ \ x, z \in \Omega.$$

If k is SPD, then the same holds using ">" instead of "$\geq$".

Proof. • We consider $N := 1$ and $X_1 = \{x\}, \alpha = 1$. Then the positive definiteness implies $\alpha^\top A \alpha = k(x, x) \geq 0$.

- We consider $N := 2$ and $X_2 = \{x, z\}$. Then we have

$$A_{k,X_N} = \begin{pmatrix} k(x, x) & k(x, z) \\ k(z, x) & k(z, z). \end{pmatrix}$$

By Proposition IV.2 we have $0 \leq \det(A_{k,X_N}) = k(x, x)k(z, z) - k(x, z)^2$, which can be rearranged to give the statement.

$\square$

The following proposition allows us to build new kernels, while maintaining their (strict) positive definiteness properties:

Proposition IV.4. *Let $k_1, k_2 : \Omega \times \Omega \to \mathbb{R}$ be PD kernels, $f : \Omega \to \Omega$ and $g : \Omega \to \mathbb{R}$, $\Omega' \subset \Omega$. Then the following kernels are also PD (SPD):*

1. $k : \Omega' \times \Omega' \to \mathbb{R}$ with $k := (k_1)|_{\Omega' \times \Omega'}$ *(SPD if k_1 is SPD)*

2. $k(x, z) := k_1(x, z) + k_2(x, z)$ *(SPD if k_1 or k_2 is SPD)*

3. $k(x, z) := \alpha k_1(x, z)$ for $\alpha \geq 0$ *(SPD if k_1 is SPD and $\alpha > 0$)*

4. $k(x, z) := k_1(x, z)k_2(x, z)$ *(SPD if both k_1 and k_2 are SPD)*

5. $k(x, z) := \exp(k_1(x, z))$

6. $k(x, z) := k_1(f(x), f(z))$ *(SPD if k_1 is SPD and f is injective)*

7. $k(x, z) := g(x)g(z)$

8. $k(x, z) := g(x)k_1(x, z)g(z)$ *(SPD if k_1 is SPD and $g(x) \neq 0$)*

The proofs are rather elementary and mostly directly follow from the definitions – and are thus left as a (voluntary) homework.

Finally we will see, that the Gaussian kernel from Example IV.9 is SPD:

Proposition IV.5. *The Gaussian kernel $k : \Omega \times \Omega \to \mathbb{R}$ for $\Omega \subset \mathbb{R}^d$ given as*

$$k(x, z) = \exp(-\|x - z\|^2)$$

is strictly positive definite.

Proof. We start with the case $d = 1$. We will make use of the Fourier transform and its inverse, which are given as

$$\mathcal{F}[f](\omega) := \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} \exp(-ix\omega) f(x) \, \mathrm{d}x$$

$$\mathcal{F}^{-1}[\hat{f}](x) := \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} \exp(ix\omega) \hat{f}(\omega) \, \mathrm{d}\omega$$

For our Gaussian kernel, we recall the following:

$$\mathcal{F}\left[\exp(-x^2)\right](x) = \frac{1}{\sqrt{2}} \cdot \exp(-\omega^2/4) =: \hat{\Phi}(\omega) \quad \Leftrightarrow \quad \exp(-x^2) = \mathcal{F}^{-1}\left[\frac{1}{\sqrt{2}} \cdot \exp(-\omega^2/4)\right](x).$$

With this, we can straightforward compute:

$$\alpha^\top A_{k,X_N} \alpha = \sum_{j,l=1}^{N} \alpha_j \alpha_l k(x_j, x_l) = \sum_{j,l=1}^{N} \alpha_j \alpha_l e^{-(x_j - x_l)^2}$$

$$= \frac{1}{\sqrt{2\pi}} \cdot \sum_{j,l=1}^{N} \alpha_j \alpha_l \int_{-\infty}^{\infty} e^{i(x_j - x_l)\omega} \hat{\Phi}(\omega) \, \mathrm{d}\omega$$

$$= \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} \hat{\Phi}(\omega) \cdot \left| \sum_{j=1}^{N} \alpha_j e^{ix_j \omega} \right|^2 \, \mathrm{d}\omega.$$

Since $\hat{\Phi}(\omega) > 0$ and $\left| \sum_{j=1}^{N} \alpha_j e^{ix_j \omega} \right|^2 > 0$ a.e. (as X_N are pairwise distinct), we have $\alpha^\top A_{k,X_N} \alpha$.
    The general case $d > 0$ follows from the observation

$$\exp(-\|x - z\|^2) = \exp(-\sum_{i=1}^{d} (x^{(i)} - z^{(i)})^2) = \prod_{i=1}^{d} \exp(-(x^{(i)} - z^{(i)})^d)$$

and an application of Proposition IV.4.

$\square$

It the previous proof it is worth to note, that the whole proof works as long as $\hat{\Phi}(\omega) > 0$, i.e. it can also be leveraged for other kernels. This is indeed done for so-called "translational invariant" or "radial basis funtion" kernels.

IV.3 Reproducing kernel Hilbert spaces

In the previous section we saw that kernel interpolation is well defined. Now we want to consider the approximation of functions with kernel interpolation. The properties of this approximation will for sure depend on the class of functions and on the kernel which we consider. In the following we will see, that there is a "native space" of functions, for which this works well, the so-called reproducing kernel Hilbert space (RKHS). Within this space, we will be able to derive a comprehensive analysis. We start with the definition of a reproducing kernel Hilbert space:

Definition IV.22. Let Ω be a nonempty set, $\mathcal{H}$ a Hilbert space of function $f : \Omega \to \mathbb{R}$ with inner product $(\cdot, \cdot)_{\mathcal{H}}$.
$\mathcal{H}$ is called a <u>Reproducing Kernel Hilbert space</u> (RKHS) on Ω, if there exists a function $k : \Omega \times \Omega \to \mathbb{R}$ (the <u>reproducing kernel</u>) such that

  1. $k(\cdot, x) \in \mathcal{H}$ for all $x \in \Omega$

  2. $(f, k(\cdot, x))_{\mathcal{H}} = f(x)$ for all $x \in \Omega, f \in \mathcal{H}$ (<u>reproducing property</u>)

The first point ensures, that the kernel (with fixed second argument) is a function itself within this Hilbert space of functions. The second point, the so called reproducing property, shows that the Riesz representer of the point evaluation functional δ_x is given by $k(\cdot, x)$.

We continue with an easy example of an RKHS on $\Omega = (0, 1)$:

Example IV.10. Consider $\Omega = (0, 1)$ and the the Sobolev space $H_0^1(\Omega)$:

$$H_0^1(\Omega) := \{f : [0, 1] \to \mathbb{R}, f \in L^2(\Omega), f' \in L^2(\Omega), f(0) = f(1) = 0\}$$

with inner product defined as

$$(f, g)_{H_0^1(\Omega)} := \int_0^1 f'(y) g'(y) \, \mathrm{d}y$$

Then $H_0^1(\Omega)$ is a RKHS on Ω with reproducing kernel given by

$$k(x, z) = \min(x, z) - xz = \begin{cases} x(1 - z) & x < z \\ z(1 - x) & x \geq z. \end{cases}$$

As a technical note it is worth to realize, that $H_0^1(\Omega)$ consists of continuous functions due to the Sobolev embedding theorem.

Now we can quickly verify the two properties of an RKHS from Definition IV.22: For the first property, we can straightforward compute $\frac{\mathrm{d}}{\mathrm{d}y} k(y, x)$ and $k(0, x)$ and $k(1, x)$ to see that $k(\cdot, x) \in H_0^1(\Omega)$. For the second property, we calculate for any $f \in H_0^1(\Omega)$:

$$\begin{aligned}
(f, k(\cdot, x))_{H_0^1(\Omega)} &= \int_0^1 f'(z) \frac{\mathrm{d}}{\mathrm{d}z} k(z, x) \, \mathrm{d}z \\
&= \int_0^x f'(z) \frac{\mathrm{d}}{\mathrm{d}z} k(z, x) \, \mathrm{d}z + \int_x^1 f'(z) \frac{\mathrm{d}}{\mathrm{d}z} k(z, x) \, \mathrm{d}z \\
&= \int_0^x f'(z)(1 - x) \, \mathrm{d}z + \int_x^1 f'(z)(-x) \, \mathrm{d}z \\
&= (1 - x)(f(x) - f(0)) - x(f(1) - f(x)) \\
&= f(x),
\end{aligned}$$

where we used $f(0) = f(1) = 0$ in the final step.

The following proposition shows that linear combinations of $k(\cdot, x_i)$ are included in the RKHS, and how to compute the inner products of such linear combinations. Both properties directly follow from the definition of the RKHS by leveraging the linearity of Hilbert spaces.

Proposition IV.6. *Let $\mathcal{H}$ be a RKHS on Ω with reproducing kernel k. Let $N, M \in \mathbb{N}$, $\alpha \in \mathbb{R}^N, \beta \in \mathbb{R}^M$ and $X_N, Z_M \subset \Omega$ and consider*

$$f = \sum_{i=1}^N \alpha_i k(\cdot, x_i), \quad g = \sum_{j=1}^M \beta_j k(\cdot, z_j).$$

Then it holds

1. *$f, g \in \mathcal{H}$,*

2. *$(f, g)_{\mathcal{H}} = \sum_{i=1}^N \sum_{j=1}^M \alpha_i \beta_j k(x_i, z_j)$*

Proof.   1. By the first property within the definiton of the RKHS in Definition IV.22, we have that $k(\cdot, x_i) \in \mathcal{H}$ for all $i = 1, ..., N$ as well as $k(\cdot, z_j) \in \mathcal{H}$ for all $j = 1, ..., M$. Since H is a Hilbert space (by definition), it is a linear space, and thus also the linear combinations f and g are within $\mathcal{H}$.

2. To compute the inner product between f and g, we leverage the reproducing property (second property within Definition IV.22 and make use of the linearity of the inner product:

$$\begin{aligned}
(f, g) &= (\sum_{i=1}^N \alpha_i k(\cdot, x_i), \sum_{j=1}^M \beta_j k(\cdot, z_j))_{\mathcal{H}} \\
&= \sum_{i=1}^N \sum_{j=1}^M \alpha_i \beta_j (k(\cdot, x_i), k(\cdot, z_j))_{\mathcal{H}} \\
&= \sum_{i=1}^N \sum_{j=1}^M \alpha_i \beta_j k(x_i, z_j).
\end{aligned}$$

$\square$

The following theorem shows that any RKHS has a kernel, that this kernel is unique, and that it is (at least) PD:

Theorem IV.13. *Let $\mathcal{H}$ be a RKHS on Ω with reproducing kernel k. Then k is unique and it is a PD kernel.*

Proof. For any $x, z \in \Omega$ we define $f := k(\cdot, x)$ and $g := k(\cdot, z)$. From Proposition IV.6 we obtain $k(x, z) = (f, g)_{\mathcal{H}} = (g, f)_{\mathcal{H}} = k(z, x)$. Thus this definition yields a valid kernel according to Definition IV.20.

We continue by proving the PDness of the kernel: Let $X_N \subset \Omega$ be pairwise distinct and $0 \neq \alpha \in \mathbb{R}^N$, then we have for the kernel matrix A

$$\alpha^\top A \alpha = \sum_{i,j=1}^N \alpha_i \alpha_j k(x_i, x_j) = \left(\sum_{j=1}^N \alpha_i k(\cdot, x_i), \sum_{j=1}^N \alpha_j k(\cdot, x_j) \right)_{\mathcal{H}}$$

$$= \left\| \sum_{i=1}^N \alpha_i k(\cdot, x_i) \right\|_{\mathcal{H}}^2 \geq 0. \tag{IV.18}$$

Finally it is left to show that the kernel is unique. For this assume that k_1 and k_2 are both reproducing kernels. By Definition IV.22 we know that $k_1(\cdot, x) \in \mathcal{H}$ as well as $k_2(\cdot, z) \in \mathcal{H}$. By using the reproducing properties, we have

$$k_1(x, z) = k_1(z, x) = (k_1(\cdot, x), k_2(\cdot, z))_{\mathcal{H}} = (k_2(\cdot, z), k_1(\cdot, x))_{\mathcal{H}} = k_2(x, z),$$

which shows that $k_1 = k_2$. $\square$

Note that we cannot conclude SPD of the kernel, because within Equation (IV.18) it may happen that $\{k(\cdot, x_i)\}_{i=1}^N$ are linearly independent. In Proposition IV.7 we will see, that the linear independence of $\{k(\cdot, x_i)\}_{i=1}^N$ is actually equivalent to the SPDness of the kernel.

To continue, we need to recall the Riesz representer theorem from Hilbert space theory (functional analysis). We start with the definition of the dual space.

Definition IV.23. For a Hilbert space $\mathcal{H}$, the dual space $\mathcal{H}'$ consists of all linear and continuous functionals $\lambda : \mathcal{H} \to \mathbb{R}$ with norm

$$\|\lambda\|_{\mathcal{H}'} := \sup_{0 \neq f \in \mathcal{H}} \frac{|\lambda(f)|}{\|f\|_{\mathcal{H}}}.$$

Theorem IV.14 (Riesz representer theorem)**.** *Let $\mathcal{H}$ be a Hilbert space and denote as $\mathcal{H}'$ its dual. Then for all $\lambda \in \mathcal{H}'$ there exists a unique $v_\lambda \in \mathcal{H}$ (Riesz representer of λ) such that*

$$\lambda(f) = (v_\lambda, f)_{\mathcal{H}} \qquad \text{for all } f \in \mathcal{H}.$$

Furthermore it holds $\|\lambda\|_{\mathcal{H}'} = \|v_\lambda\|_{\mathcal{H}}$.

The Riesz representer theorem now allows us to further characterize RKHS:

Proposition IV.7. *Let Ω be a nonempty set and $\mathcal{H}$ a Hilbert space of functions $f : \Omega \to \mathbb{R}$. Then*

1. *$\mathcal{H}$ is a RKHS $\Leftrightarrow$ all point evaluation functionals are continuous*

If $\mathcal{H}$ is a RKHS with kernel k, then

2. *$k(\cdot, x)$ is the Riesz representer of the point evaluation functional $\delta_x \in \mathcal{H}'$*

3. *k is SPD $\Leftrightarrow \{\delta_x, x \in \Omega\}$ are linearly independent*

4. *$|f(x)| \leq \sqrt{k(x, x)} \|f\|_{\mathcal{H}}$ for all $f \in \mathcal{H}, x \in \Omega$.*

5. *Convergence in $\mathcal{H}$ implies pointwise convergence*

Proof. 1. Let $\mathcal{H}$ be a RKHS, i.e. there exists a reproducing kernel k. Then we have for the point evaluation functional δ_x

$$|\delta_x(f)| = |f(x)| = |(f, k(\cdot, x))_{\mathcal{H}}| \leq \|f\|_{\mathcal{H}} \cdot \|k(\cdot, x)\|_{\mathcal{H}} = \|f\|_{\mathcal{H}} \cdot \sqrt{k(x, x)}, \qquad \text{(IV.19)}$$

i.e. the point evaluation functional is bounded and thus continuous.

Let $\delta_x \in \mathcal{H}'$, then Theorem IV.14 gives a Riesz representer $v_{\delta_x} \in \mathcal{H}$. We now define $k(\cdot, x) := v_{\delta_x} \in \mathcal{H}$: We have $k(\cdot, x) \in \mathcal{H}$ and $(f, k(\cdot, x)) = (f, v_{\delta_x} = f(x)$ for all $x \in \Omega$. Thus $\mathcal{H}$ has a reproducing kernel and thus is an RKHS.

2. The reproducing property yields $(f, k(\cdot, x)) = f(x)$ for all $x \in \Omega$. Furthermore $k(\cdot, x) \in \mathcal{H}$ and the Riesz representer is unique.

3. First we show that a finite set of linear functionals is linearly independent if and only if the corresponding Riesz representer are linearly independent:

   Let $\lambda_1, ..., \lambda_N$ be N linearly independent functionals and $v_{\lambda_1}, ..., v_{\lambda_n}$ their corresponding Riesz representers. We consider for any $f \in \mathcal{H}$

   $$0 = \sum_{j=1}^{N} \alpha_j \lambda_j(f) = (\sum_{j=1}^{N} \alpha_j v_{\lambda_j}, f)_{\mathcal{H}}$$

   which can only hold for $\alpha_j = 0, j = 1, ..., N$ due to the linear independence of the functionals $\lambda_1, ..., \lambda_N$. Picking $f := \sum_{j=1}^{N} \alpha_j v_{\lambda_j}$ shows that also $v_{\lambda_1}, ..., v_{\lambda_N}$ are linearly independent.

   In order to prove the equivalence of k being SPD to $\{\delta_x, x \in \Omega\}$ being linearly independent, we can leverage the computation within Theorem IV.13, but now we can conclude " $>$" instead of " $\geq$" within Eq. (IV.18) due to $\{\delta_x, x \in \Omega\}$ and thus $\{k(\cdot, x), x \in \Omega\}$ being linearly independent.

4. This was already shown in Eq. (IV.19).

5. Let $f_n \to f$ with $f, f_n \in \mathcal{H}$. Then, by the previous point, we obtain

   $$|(f - f_n)(x)| \leq \sqrt{k(x, x)} \cdot \|f - f_n\|_{\mathcal{H}} \to 0.$$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square$

In practice, we rarely start with the RKHS and think about its kernel. More frequently, we work with a kernel and are interested about its RKHS. The following theorem gives this implication, i.e. how to obtain the RKHS from the kernel.

Theorem IV.15 (Moore-Aronszajn). *Let Ω be a nonempty set and $k : \Omega \times \Omega \to \mathbb{R}$ a positive definite kernel. Then there exists a unique RKHS $\mathcal{H}_k(\Omega)$ with reproducing kernel k.*

Proof. The proof is constructive. Due to Proposition IV.6, we know that the finite linear combinations $\sum_{i=1}^{N} \alpha_i k(\cdot, x_i)$ are contained in the RKHS. We will show, that the RKHS is given by the completion of the space given by the span of all such linear combinations:

We define

$$\mathcal{H}_0 := \text{span}\{k(\cdot, x), x \in \Omega\}$$

as the space of all finite linear combinations of kernel functions with fixed second argument $x \in \Omega$. We consider any two functions $f, g \in \mathcal{H}_0$. By definition of $\mathcal{H}$, they can be expressed as

$$\begin{aligned} f &= \sum_{i=1}^{N} \alpha_i k(\cdot, x_i), \\ g &= \sum_{j=1}^{M} \beta_j k(\cdot, x_j), \end{aligned} \qquad \text{(IV.20)}$$

where we can assume that $\{x_i\}_{i=1}^{N} \subset \Omega$ and $\{x_j\}_{i=1}^{N} \subset \Omega$ are each pairwise distinct. For any such two functions f, g, we define an bilinear mapping $\mathcal{B} : \mathcal{H}_0 \times \mathcal{H}_0 \to \mathbb{R}$ as

$$\mathcal{B}(f, g) = \mathcal{B}\left(\sum_{i=1}^{N} \alpha_i k(\cdot, x_i), \sum_{j=1}^{M} \beta_j k(\cdot, x_j)\right) := \sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_i \beta_j k(x_i, x_j).$$

We show that this mapping is well-defined, symmetric, bilinear and positive semi-definite:

- By definition of f and g, it holds

$$\mathcal{B}(f,g) = \sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_i \beta_j k(x_i, x_j) = \sum_{i=1}^{N} \alpha_i g(x_i) = \sum_{j=1}^{M} \beta_j f(x_j),$$

  i.e. the value of $\mathcal{B}(f,g)$ does not depend on the actual representation of f, g within Eq. (IV.20) (which does not need to be unique), and thus $\mathcal{B}(f,g)$ is well defined.

- The symmetry follows directly from the definition of $\mathcal{B}$ and the symmetry of the kernel k.

- The bilinearity follows as well directly from the definition of $\mathcal{B}$.

- For the positive semi-definiteness, we compute

$$\mathcal{B}(f,f) = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) = \alpha^\top A \alpha \geq 0$$

  by the positive definiteness of the kernel k.

With these properties it follows that $\mathcal{B}$ is a semi inner product on $\mathcal{H}_0$, i.e. we can define $(f,g)_{\mathcal{H}_0} := \mathcal{B}(f,g)$ for $f, g \in \mathcal{H}_0$. Especially, $\|f\|_{\mathcal{H}_0} \equiv \sqrt{(f,f)_{\mathcal{H}_0}}$ defines a semi-norm.

Now we show that k is a reproducing kernel in $\mathcal{H}_0$: Obviously we have $k(\cdot, x) \in \mathcal{H}_0$ for any $x \in \Omega$. Furthermore we have

$$(f, k(\cdot, x))_{\mathcal{H}} = \sum_{i=1}^{N} \alpha_i k(x, x_i) = f(x),$$

i.e. also the reproducing property is satisfied. This allows us to show that the semi-inner product is actually an inner product: Let $f \in \mathcal{H}_0$ with $\|f\|_{\mathcal{H}_0} = 0$. We would like to conclude $f = 0$. To see this, we use the Cauchy-Schwarz inequality (which is applicable to semi-inner products) to estimate

$$|f(x)| = |(f, k(\cdot, x))_{\mathcal{H}_0}| \leq \|f\|_{\mathcal{H}_0} \cdot \|k(\cdot, x)\|_{\mathcal{H}_0},$$

i.e. $\|f\|_{\mathcal{H}_0} = 0$ implies $f = 0$.

Finally we have shown that $(\mathcal{H}_0, (\cdot, \cdot)_{\mathcal{H}_0})$ is a pre-Hilbert space. From functional analysis it is known, that pre-Hilbert spaces can be completed in an abstract way. We see the details in the following:

$$\mathcal{H}_k(\Omega) := \{f : \Omega \to \mathbb{R} \mid \exists \{f_n\}_{n \in \mathbb{N}} \subset \mathcal{H}_0 \text{ Cauchy sequence s.t. } \lim_{n \to \infty} f_n = f\},$$

$$(f,g)_{\mathcal{H}_k(\Omega)} := (\lim_{n \to \infty} f_n, \lim_{m \to \infty} g_m)_{\mathcal{H}_k(\Omega)} = \lim_{n \to \infty} \lim_{m \to \infty} (f_n, g_m)_{\mathcal{H}_0}$$

for $f = \lim_{n \to \infty} f_n$ and $g = \lim_{m \to \infty} g_m$.

Note that, in general, the definiton of the abstractly completed space $\mathcal{H}_k(\Omega)$ is given directly by all the Cauchy sequences within $\mathcal{H}_0$. However here we can directly identity these Cauchy sequences with a limiting function $f : \Omega \to \mathbb{R}$, because any Cauchy Sequence in $\mathcal{H}_0$ converges pointwise:

$$|f_n(x) - f_m(x)| \leq \|f_n - f_m\|_{\mathcal{H}_0} \sqrt{k(x,x)} \quad \text{for all } x \in \Omega.$$

Also observe that $\mathcal{H}_0 \subset \mathcal{H}_k(\Omega)$, because for any $f \in \mathcal{H}_0$ simply consider the Cauchy sequence $\{f\}_{i=1}^{\infty}$. Next, we show that the reproducing property of k within $\mathcal{H}_0$ carries over due to continuity:

$$f(x) = \lim_{n \to \infty} f_n(x) = \lim_{n \to \infty} (f_n, k(\cdot, x))_{Ha_0} = (\lim_{n \to \infty} f_n, k(\cdot, x))_{Ha_0} = (f, k(\cdot, x)_{\mathcal{H}_k(\Omega)}.$$

Finally we observe that $\mathcal{H}_k(\Omega)$ is unique: Assume there is another RKHS $\mathcal{H}$. Due to Definition IV.22 we have again $\mathcal{H}_0 \subset \mathcal{H}$, thus we can run the same construction and end up with $\mathcal{H} = \mathcal{H}_k(\Omega)$.

$\square$

In Theorem IV.15 we started to use the notation $\mathcal{H}_k(\Omega)$ instead of $\mathcal{H}$ – this is due to the links between a kernel and its RKHS:

- Theorem IV.13 showed that the reproducing kernel of a RKHS is unique and PD.

- Theorem IV.15 showed that every PD kernel gives rise to a unique RKHS.

In Theorem IV.12 we saw, that it is possible to define kernel via a feature map. The following proposition shows, that there is always a special feature map associated to a kernel:

Proposition IV.8 (Kernel feature map)**.** *Let k be a PD kernel on Ω. The* $\underline{\text{kernel feature map}}$ *given by* $\phi : \Omega \to \mathcal{H}_k(\Omega), \phi(x) := k(\cdot, x)$ *defines the kernel.*

Proof. Theorem IV.15 ensures the existence of $\mathcal{H}_k(\Omega)$ and shows that $k(\cdot, x) \in \mathcal{H}_k(\Omega)$. That this kernel feature map indeed defines the kernel, follows from the reproducing property:

$$(\phi(x), \phi(z))_{\mathcal{H}_k(\Omega)} = (k(\cdot, x), k(\cdot, z))_{\mathcal{H}_k(\Omega)} = k(x, z).$$

$\square$

Proposition IV.9. *Let k be a SPD kernel on Ω. If* $\dim(\Omega) = \infty$, *then* $\dim(\mathcal{H}_k(\Omega)) = \infty$.

Proof. Consider N pairwise distinct points $X_N \subset \Omega$. Since k is SPD, we have that $\{k(\cdot, x_j)\}_{j=1}^{N}$ are linearly independent by Proposition IV.7. Furthermore we have $\{k(\cdot, x_j)\}_{j=1}^{N} \subset \mathcal{H}_k(\Omega)$ by definition of an RKHS in Definition IV.22. Thus $\text{span}\{k(\cdot, x_j), j = 1, ..., N\} \subset \mathcal{H}_k(\Omega)$ has dimensionality N. As this holds for any $N \in \mathbb{N}$, we conclude $\dim(\mathcal{H}_k(\Omega)) = \infty$. $\square$

IV.4 Kernel interpololation in RKHS

In the previous sections we saw, that kernel interpolation is well defined and that there is a unique reproducing kernel Hilbert space of functions associated to every PD kernel. This section focusses on analyzing properties of kernel interpolation in RKHS. For this, we will focus on SPD kernels in the following and consider the following setting: Given N pairwise distinct points $X_N \subset \Omega$ and a function $f \in \mathcal{H}_k(\Omega)$, we denote the kernel interpolant as s_f. Then we have:

Proposition IV.10. *Let k be a SPD kernel on Ω, $X_N \subset \Omega$ be N pairwise distinct points and $f \in \mathcal{C}(\Omega)$. Consider*

$$V(X_N) := \text{span}\{k(\cdot, x_i), x_i \in X_N\}.$$

1. *$V(X_N) \subset \mathcal{H}_k(\Omega)$ is an N-dimensional subspace*

2. *$s_f \in V(X_N)$*

3. *If $f \in \mathcal{H}_k(\Omega)$, then $s_f = \Pi_{V(X_N)}(f)$ (using the orthogonal projection $\Pi_{V(X_N)} : \mathcal{H}_k(\Omega) \to V(X_N)$)*

Proof.   1. The definition of the RKHS yields that $V(X_N) \subset \mathcal{H}_k(\Omega)$. As k is SPD, it follows that $V(X_N)$ is N-dimensional by Proposition IV.7.

2. This holds by definition of s_f.

3. We need to show that $f - s_f \perp V(X_N)$. We use that $\{k(\cdot, x_i)\}_{i=1}^{N}$ is a basis of $V(X_N)$ and compute

$$(f - s_f, k(\cdot, x_i))_{\mathcal{H}_k(\Omega)} = (f, k(\cdot, x_i))_{\mathcal{H}_k(\Omega)} - (s_f, k(\cdot, x_i))_{\mathcal{H}_k(\Omega)} = f(x_i) - s_f(x_i) = 0 \quad \text{(IV.21)}$$

for any $i = 1, ..., N$. By linearity, this shows that $f - s_f$ is orthogonal to $V(X_N)$. $\square$

We saw that interpolation amounts to orthogonal projection. By using properties of orthogonal projections, we can conclude the following.

Corollary IV.5. *Let k be a SPD kernel on Ω, $X_N \subset \Omega$ be N pairwise distinct points and $g \in \mathcal{H}_k(\Omega)$.*

- *$g \in V(X_N)^{\perp} \Leftrightarrow g(x_i) = 0$ for all $x_i \in X_N$.*

Each $f \in \mathcal{H}_k(\Omega)$ can be uniquely decomposed as $f = s_f + (f - s_f)$ such that

$$s_f \in V(X_N), s_f(x_i) = f(x_i) \quad \text{for all } x_i \in X_N$$
$$(f - s_f) \in V(X_N)^{\perp}, (f - s_f)(x_i) = 0 \quad \text{for all } x_i \in X_N$$
$$(s_f, f - s_f)_{\mathcal{H}_k(\Omega)} = 0$$
$$\|f\|_{\mathcal{H}_k(\Omega)}^2 = \|s_f\|_{\mathcal{H}_k(\Omega)}^2 + \|f - s_f\|_{\mathcal{H}_k(\Omega)}^2$$

Proof. All the statements follow directly from properties of orthogonal projections by decomposing any $f \in \mathcal{H}_k(\Omega)$ into $f = g + g^\perp$ with $g = \Pi_{V(X_N)}(f)$ and $g^\perp = \Pi_{V(X_N)^\perp}(f)$ and using $s_f = g$ and thus $g^\perp = f - s_f$.

For example, for the last property we have

$$
\begin{aligned}
\|f\|^2_{\mathcal{H}_k(\Omega)} &= (g + g^\perp, g + g^\perp)_{\mathcal{H}_k(\Omega)} \\
&= \|g\|^2_{\mathcal{H}_k(\Omega)} + 2(g, g^\perp)_{\mathcal{H}_k(\Omega)} + \|g^\perp\|^2_{\mathcal{H}_k(\Omega)} \\
&= \|g\|^2_{\mathcal{H}_k(\Omega)} + \|g^\perp\|^2_{\mathcal{H}_k(\Omega)}.
\end{aligned}
$$

$\square$

In the following two propositions, we will analyze two variants of our interpolation problem. First, in Proposition IV.11 we will discuss what will happen, if we keep the ansatz space $V(X_N)$ but do not focus on interpolation anymore. Second, in Proposition IV.12 we will discuss what will happen, if we keep the interpolation conditions, but do not focus on the ansatz space $V(X_N)$ anymore.

Proposition IV.11 (Best approximation by interpolation)**.** *Let k be a SPD kernel on Ω, $X_N \subset \Omega$ be N pairwise distinct points. The kernel interpolant s_f is the unique best-approximant from the space $V(X_N)$ to a function $f \in \mathcal{H}_k(\Omega)$:*

$$
\|f - s_f\|_{\mathcal{H}_k(\Omega)} = \min_{s \in V(X_N)} \|f - s\|_{\mathcal{H}_k(\Omega)}.
$$

Proof. This follows directly from the fact that s_f is the orthogonal projection of f onto $V(X_N)$, see the Proposition IV.10. $\square$

Proposition IV.12 (Minimal norm interpolation)**.** *Let k be a SPD kernel on Ω, $X_N \subset \Omega$ be N pairwise distinct points. Let*

$$
S := \{s \in \mathcal{H}_k(\Omega) : s(x_i) = f(x_i), i = 1, ..., N\}
$$

be the space of interpolating functions in $\mathcal{H}_k(\Omega)$. Then it holds

$$
\|s_f\|_{\mathcal{H}_k(\Omega)} = \min_{s \in S} \|s\|_{\mathcal{H}_k(\Omega)},
$$

with the minimum being unique.

Proof. Apparently $s_f \in S$. Furthermore, within $V(X_N)$, there is no other interpolant. Now we show that any other interpolating function has higher norm: Consider any $s \in S$. By Corollary IV.5 we may decompose $s = g + g^\perp$ with $g \in V(X_N)$ and $g^\perp \perp V(X_N)$. By Corollary IV.5 it furthermore follows that $g(x_i) = s(x_i) = f(x_i)$ and thus that $g = s_f$ by uniqueness of the interpolant within $V(X_N)$. Finally we have

$$
\|s\|^2_{\mathcal{H}_k(\Omega)} = \|g\|^2_{\mathcal{H}_k(\Omega)} + \|g^\perp\|^2_{\mathcal{H}_k(\Omega)} = \|s_f\|^2_{\mathcal{H}_k(\Omega)} + \|g^\perp\|^2_{\mathcal{H}_k(\Omega)} \geq \|s_f\|^2_{\mathcal{H}_k(\Omega)}.
$$

This shows, that as soon as $s \notin V(X_N)$, we have $g^\perp \neq 0$ and thus $\|s\|^2_{\mathcal{H}_k(\Omega)} > \|s_f\|^2_{\mathcal{H}_k(\Omega)}$. $\square$

In the following we turn briefly to error estimates, i.e. ways to bound $f - s_f$. The following proposition is very basic:

Proposition IV.13 (Simple bound)**.** *Let k be a SPD kernel on Ω, $X \subset Y \subset \Omega$ be each pairwise distinct points and $f \in \mathcal{H}_k(\Omega)$. Let $s_{f,X}$ and $s_{f,Y}$ be the corresponding kernel interpolants. Then it holds*

$$
V(X) \subset V(Y)
$$
$$
\|s_{f,X}\|_{\mathcal{H}_k(\Omega)} \leq \|s_{f,Y}\|_{\mathcal{H}_k(\Omega)} \leq \|f\|_{\mathcal{H}_k(\Omega)}
$$
$$
\|f - s_{f,Y}\|_{\mathcal{H}_k(\Omega)} \leq \|f - s_{f,X}\|_{\mathcal{H}_k(\Omega)} \leq \|f\|_{\mathcal{H}_k(\Omega)}
$$

Proof. As $X \subset Y$, it directly follows $V(X) \subset V(Y)$ by definition of the spaces $V(X), V(Y)$. By Corollary IV.5 we furthermore obtain $\|s_{f,X}\|_{\mathcal{H}_k(\Omega)} \leq \|f\|_{\mathcal{H}_k(\Omega)}$ as well as $\|f - s_{f,X}\|_{\mathcal{H}_k(\Omega)} \leq \|f\|_{\mathcal{H}_k(\Omega)}$. The remaining inequalities follow likewise by Corollary IV.5 by observing that the interpolant $s_{f,X}$ of f in X and the interpolant $s_{s_{f,Y},X}$ of $s_{f,Y}$ in X coincide due to $X \subset Y$ and $s_{f,Y}(x_i) = f(x_i)$ for all $x_i \in X$. $\square$

In order to have more sophisticated error estimates, we need to introduced the power function:

Definition IV.24 (Power function)**.** Let $s_{f,X}$ be the kernel interpolant of f in X. The <u>power function</u> is defined as

$$P_{X_N}(x) = \sup_{0 \neq f \in \mathcal{H}_k(\Omega)} \frac{|f(x) - s_{f,X}(x)|}{\|f\|_{\mathcal{H}_k(\Omega)}} \tag{IV.22}$$

We see, that by rearranging the definition of the power function in Eq. (IV.22), we immediately obtain an error bound:

$$|f(x) - s_f(x)| \leq P_{X_N}(x) \cdot \|f\|_{\mathcal{H}_k(\Omega)}$$

This error bound splits into the power function factor, which is independent of the function, and the RKHS norm factor, which is independent of the points X_N.

However, the definition of the power function of Eq. (IV.22) seems infeasbile to compute. Therefore we will see in the following, how the power function can be computed in practice:

Proposition IV.14. *The power function can be computed as*

$$P_{X_N}(x) = \sqrt{k(x,x) - \sum_{i=1}^{N}\sum_{j=1}^{N} k(x,x_i)(A^{-1})_{ij}k(x,x_j)}$$

We do not state the proof here, because it is not necessary or helpful for the following. We stated Proposition IV.14 only to show that there is a computational feasible way to compute the power function: Only the kernel and the input points X_N need to be known.

IV.5 Translational invariant kernels and radial basis function kernels

So far, all the results have been rather abstract. In the following, we will briefly discuss two popular classes of kernels, namely "translational invariant kernels" and "radial basis function kernels". We start with the definition:

Definition IV.25. A kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called

- <u>translational invariant</u>, if

$$k(x,y) = k(x+z, y+z) \quad \text{for all } x, y, z \in \mathbb{R}^d$$

- <u>radial/radially invariant</u>, if

$$k(x,y) = k(z,w)$$

for all $x, y, z, w \in \mathbb{R}^d$ such that $\|x - z\| = \|z - w\|$.

These kernels can be characterized in also in a different way:

Proposition IV.15. *A kernel* $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *is*

- *translational invariant, if and only if there exists a function* $\Phi : \mathbb{R}^d \to \mathbb{R}$ *such that it holds*

$$k(x,y) = \Phi(x-y) \quad \text{for all } x, z \in \mathbb{R}^d.$$

- *radial, if and only if there exists a function* $\Phi : \mathbb{R}_{\geq 0} \to \mathbb{R}$ *such that it holds*

$$k(x,y) = \Phi(\|x-y\|) \quad \text{for all } x, z \in \mathbb{R}^d.$$

Proof. • Let $k(x,y) = \Phi(x-y)$. Then one can directly calculate

$$k(x+z, y+z) = \Phi(x+z-(y+z)) = \Phi(x-y) = k(x,y).$$

Now assume that it holds

$$k(x,y) = k(x+z, y+z) \quad \text{for all } x, y, z \in \mathbb{R}^d.$$

Then, for any fixed $x_0 \in \mathbb{R}^d$, we define $\Phi(x) = k(x_0, x_0 - x)$. With $z := -x + x_0$ we can calculate

$$k(x, y) = k(x - x + x0, y - x + x_0) = k(x_0, x_0 - (x - y)) = \Phi(x - y).$$

It is left to show, that the definition of Φ actually does not depend on the choice of x_0. Thus we consider $\tilde{x}_0 \in \mathbb{R}^d$ and compute

$$\Phi(x) = k(x_0, x_0 - x) = k(x_0 + (\tilde{x}_0 - x_0), x_0 - x + (\tilde{x}_0 - x_0)) = k(\tilde{x}_0, \tilde{x}_0 - x).$$

This shows that the definition of Φ actually does not depend on the choice of x_0.

- Let $k(x, y) = \Phi(\|x - y\|)$ and $z, w \in \mathbb{R}^d$ such that $\|x - y\| = \|z - w\|$. Then we diretly have

$$k(x, y) = \Phi(\|x - y\|) = \Phi(\|z - w\|) = k(z, w).$$

Now assume that it holds $k(x, y) = k(z, w)$ for all $x, y, z, w \in \mathbb{R}^d$ such that $\|x - y\| = \|z - w\|$. Then, for any fixed $x_0 \in \mathbb{R}^d$, we define $\Phi(r) = k(x_0, x_0 + re_1)$ with $e_1 \in \mathbb{R}^d$ being the first unit vector in the Euclidean system. Consider $z := x_0$ and $w := x_0 + \|x - y\|e_1$, which satisfy $\|x - y\| = \|z - w\|$, i.e.

$$k(x, y) = k(x_0, x_0 + \|x - y\|e_1) = \Phi(\|x - y\|).$$

As before, it is left to show that the definition of Φ actually does not depend on the choice of x_0 and e_1. Thus consider $\tilde{x}_0, v \in \mathbb{R}^d$ such that $\|v\| = 1$. We see directly

$$\Phi(r) \equiv k(x_0, x_0 + re_1) = k(\tilde{x}_0, \tilde{x}_0 + rv)$$

due to $\|x_0 + re_1 - x_0\| = r = \|\tilde{x}_0 + rv - \tilde{x}_0\|$.

$\square$

We continue with some commments: The radial kernels are frequently also called radial basis function kernels (RBF kernels). It is obvious, that RBF kernels are also translational invariant, but that not every translational invariant kernel needs to be an RBF kernel. Any RBF kernels can be tuned with help of a shape parameter $\varepsilon > 0$ as

$$k(x, z) = \Phi(\varepsilon\|x - z\|).$$

This shape parameter $\varepsilon > 0$ allows to adapt the kernel to the domain Ω, or in applications to the data X_N and the corresponding target values.

In the following we will see how to deduce properties of k from Φ

Modern Mathematical Machine Learning

Bibliography

[1] C. Lubich, *From quantum to classical molecular dynamics: reduced models and numerical analysis*, European Mathematical Society, 2008.

[2] Y. Saleh, "Spectral and active learning for enhanced and computationally scalable quantum molecular dynamics", Dissertation, Hamburg, Germany: Universität Hamburg, 2023.

[3] Y. Saleh, Á. F. Corral, A. Iske, J. Küpper, and A. Yachmenev, "Computing excited states of molecules using normalizing flows", *arXiv preprint arXiv:2308.16468* (2023).

[4] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, 2014.

[5] R. Schapire, *COS 511: Theoretical Machine Learning*, Lecture Notes, Theoretical Machine Learning, Princeton University Computer Science Department, 2019.

[6] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "Towards understanding the role of over-parametrization in generalization of neural networks", *arXiv preprint arXiv:1805.12076* (2018).

[7] J. Karafin, https://www.lightfieldlab.com/blogposts/ceo-corner-the-evolution-of-visual-perception.

[8] P. A. I. Gimbel R. W. F. Geenen, *What radiologists need to know about Convolutional Neural Networks (CNNs)*, https://epos.myesr.org/poster/esr/ecr2023/C-17664.

[9] Y. LeCun, "The MNIST database of handwritten digits", *http://yann. lecun. com/exdb/mnist/* (1998).

[10] A. Lenail, http://alexlenail.me/NN-SVG/.

[11] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.", *Psychological review* **65** (1958), 386.

[12] M. A. Nielsen, *Neural networks and deep learning*, vol. 25, Determination press San Francisco, CA, USA, 2015.

[13] G. Sanderson, *Backpropagation calculus*, https://www.3blue1brown.com/lessons/backpropagation-calculus.

[14] H. Wendland, *Scattered Data Approximation*, vol. 17, Cambridge Monographs on Applied and Computational Mathematics, Cambridge: Cambridge University Press, 2005.

[15] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks", *Advances in Neural Information Processing Systems* **31** (2018).