FACULTY OF LAW AND CRIMINOLOGY

SUPERVISED MACHINE LEARNING IN LAW

Wim De Mulder
Student number: 002000340262

Supervisor: Prof. Dr. Joke Baeck

Co-readers: Thomas Demeester and Pablo Herremans

A dissertation submitted to Ghent University in partial fulfilment of the requirements for the degree of Master of Laws

Academic year: 2019-2020

GHENT UNIVERSITY

ACKNOWLEDGEMENT

I am enormously grateful to my thesis supervisor, Prof. Joke Baeck, for her generous and thoughtful guidance throughout the course of this project. I gratefully acknowledge her respectful and positive attitude, and I will remember the warm and stimulating talks. In my view it is rather exceptional that someone who has achieved a high level of expertise in a specific field is so open-minded and receptive towards other fields. Ronald E. Osborn once said it as follows: “unless you try to do something beyond what you have already mastered, you will never grow”.

Abstract

As a buzzword in technology, *machine learning* has gained the status of a synonym of *super intelligent computing*. The non expert is undoubtedly aware of its existence in many applications, such as speech recognition systems, computer games and chatbots. Yet, as an invisible driving force behind such applications it is surrounded with an aura of mystery. It seems almost as if machine learning is meant not to be comprehended. Understandingly, this results in a reluctance to use it, as it is considered an imperceptible enemy rather than a supportive friend. This holds, at least, for many practitioners in highly specialized fields, such as medicine and law. A high number of experts in these fields are unwilling to believe that machine learning can add value to their tasks, let alone that they can imagine that one day machine learning systems will surpass their expertise.

In 1876, Alexander Graham Bell became the first inventor to be granted a patent for the telephone. He approached American communications company Western Union and offered them rights to his patent for $100,000, but company bigwigs balked at the proposal citing the “obvious limitations of his device, which is hardly more than a toy”. A little bit later, in 1879, Henry Morton, a leading scientific mind and president of the Stevens Institute of Technology, called one man’s tinkering a "conspicuous failure." The man was Thomas Edison. The invention was the light bulb. History clearly tends to repeat itself, as one Nathan Stubblefield invented a wireless communication device in 1892, but he found himself ridiculed for his efforts.

Maybe these stories tell us to be careful in expecting (or hoping) that one day it will become clear that machine learning is unable to touch the limits of humankind’s extraordinary intelligence? At any rate, the fact that many people refuse to accept the power of machine learning, is often either due to ignorance about its principles or due to false knowledge (which is, according to George Bernard Shaw, “even more dangerous than ignorance”).

The purpose of this thesis is to describe machine learning to the non expert, in particular to practitioners in law. Emphasis is on a clear understanding of the basic principles. This means that instead of focusing on the amazing results that machine learning has achieved in the field of law, i.e. *what can be done*, we highlight *how it is done*. However, because machine learning is a very broad field, it is necessary to narrow down the topic. We have chosen to focus on *supervised* machine learning, where data instances on which the system relies for its proper functioning, consist of input-output pairs (in contrast, in unsupervised machine learning data instances only contain an input part). Supervised machine learning is much richer in applications than its unsupervised counterpart, and this also applies to the particular field of law. The general principles of supervised machine learning are illustrated on a particularly interesting and very popular machine learning methodology, namely artificial neural networks. If their interesting history and intuitive, yet subtle, characteristics will not attract the attention of the newcomer in

machine learning, he can be sure that he never will be active in the field of artificial intelligence. Trying to understand the described fundamental principles of machine learning will pay off for the legal practitioner: applications of machine learning in his field will be much better understood, appreciated and -indeed- criticized.

TABLE OF CONTENTS

CHAPTER 1

FUNDAMENTALS OF MACHINE LEARNING

1.1 Introduction to machine learning

Can a computer mimic human behavior in performing complex tasks? Every person, computer scientist or otherwise, knows the answer: "Yes". Indeed, today no serious man considers a computer as the mere equivalence of a mechanical calculator, restricted to the simple task of automatically carrying out the basic operations of arithmetic. The many examples of apparently smart computerized systems in everyday life have induced human beings with a general understanding of machines that are able to intelligently react to changing conditions. Autopilot systems in airplanes [6], intelligent speed adaptation in cars [57], speech recognition systems [75] and smart lawn mowers [80] are just a few of these examples.

What distinguishes the computer scientist from the layman, is that only the first one thoroughly understands the working principles of machine learning, the main field of computer science that is devoted to simulating the outcome of human reasoning in performing a complex task. This is, in fact, quite remarkable, as the basic principles of machine learning are relatively easy to grasp. The crucial idea is to copy the main method by which humans develop ability to solve a certain complex task, namely by learning from examples. This applies, in particular, to tasks that cannot be clearly defined, in the sense that a simple and deterministic stepwise approach to perform the task seems nonexistent. Examples include riding bike, swimming, and recognizing a picture as representing a person, even if the picture is very blurred as in Fig. 1.1. The last case stresses at once what learning from examples means: it refers to being able to *generalize* the information incorporated in the presented examples to new, yet similar, instances. Thus generalization has been achieved when the involved task can still be satisfactorily solved upon presentation of a new instance, e.g. when one still recognizes the silhouette of a man on a blurred

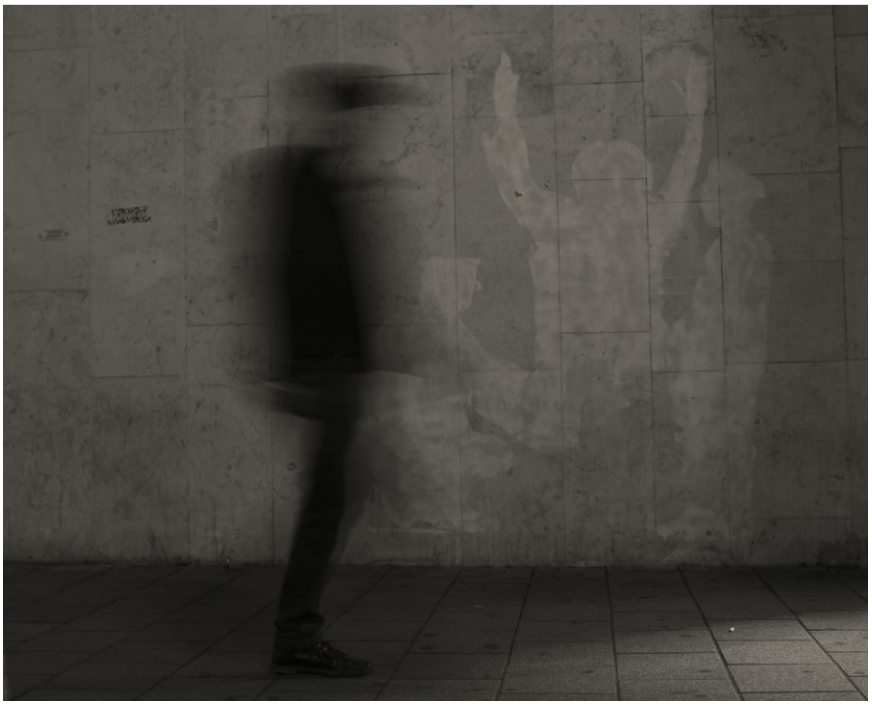

Figure 1.1: Blurred picture of a man. Image adapted from https://pxhere.com/en/photo/646715.

picture although neither the picture nor the man has been seen before. Generalizing certain aspects of a real phenomenon after having been presented instances of its working, is exactly what characterizes machine learning. This feature is significantly different from other well known functions a computer is able to perform, such as finding a word in a text (which merely relates to performing a simple search through words in memory) or switching thermostat modes when certain temperature thresholds are reached (which executes a clearly defined algorithm such as 'if the temperature is lower than X degrees start heating, if the temperature is higher than Y degrees start cooling').

The generalization property of machine learning relates to another important feature of most machine learning systems: the involved processes are often black boxes, meaning that the underlying mechanism that maps the input to the output is obfuscated by a figurative box [39]. A machine learning system is able to identify new blurred pictures, after having been presented a lot of similar blurred pictures, yet the rules that have been learnt by the system to perform this task, cannot be extracted from it. This is, again, not unlike human behavior on performing complex tasks. If an adult is asked to describe in detail the specific rules by which he rides bike, he will fail to do so. At least, not the kind of rules that can be easily implemented by another person in order to allow him to ride bike instantaneously. The black box notion correlates with an important distinction in computer science approaches that are developed to solve complex tasks, namely data-driven versus rule-based systems. In a data-driven system the available

information takes the form of numeric data, representing a set of instances (or examples) of the behavior of the considered phenomenon [20]. In contrast, a rule-based system relies on rules as the representation of knowledge encoded in the system, the purpose being to mimic the reasoning of a human expert in solving a knowledge intensive problem [25]. Simply speaking, one could say that data-driven systems, the class to which most machine learning systems belong, mimic the outcome of human reasoning, while rule-based systems mimic the human reasoning itself. This work restricts attention to data-driven machine learning, as this is the most relevant subfield for law. For example, in client-lawyer relationships, clients may desire that the considered judgment of an experienced lawyer is informed by the most relevant information required to answer their questions, where that information relates to, e.g., case law (which constitutes data, as contrasted to rules) [84].

1.2 Types of machine learning systems

Machine learning systems can be used to perform a variety of tasks. Depending on the task at hand, a different kind of machine learning system might be appropriate. The result is a hierarchical organization of machine learning systems, where the properties of the considered task determine the selection of the right kind of machine learning system to be applied. The top of this hierarchy distinguishes between supervised and unsupervised machine learning. The difference between both types of system lies in the structure of the given examples by which the system learns to grasp a real world phenomenon.

1.2.1 Supervised machine learning

In supervised machine learning, the examples, which represent particular instances of the studied phenomenon, consist of both an input and a corresponding output. Since the examples are to be given in a format that is understandable by a computer system, the collected inputs and outputs are typically numeric. In supervised machine learning, an example is then of the form $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i$ represents the input of the ith example, while $\mathbf{y}_i$ denotes the corresponding output. It is not required that both vectors have the same number of components. Let $\mathbf{x}_i \in \mathbb{R}^n$, i.e. a vector with n components, and $\mathbf{y}_i \in \mathbb{R}^m$, i.e. a vector with m components, with possibly $m \neq n$. The set of all collected examples can then be represented as $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, N\}$, with N denoting the number of examples.

As a simple illustration of how a set of examples may look like in a supervised learning context, suppose that we want to predict Amazon's stock price based on a short history encompassing five stock prices. Then a typical procedure to construct examples for a machine learning system that is to perform this task, entails the collection of all available historical stock prices of Amazon. Suppose that there are 500 historical prices, which we can denote as $s_1, \ldots, s_{500}$. Since we want the system to learn to predict the next price given the five previous prices, we provide

it examples consisting of prices over five consecutive days as input, while the next price serves as the corresponding output. Thus the examples for this particular application are given by $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_{495}, \mathbf{y}_{495})$ with

$$\begin{aligned}
\mathbf{x}_1 &= (s_1, s_2, s_3, s_4, s_5) \\
\mathbf{y}_1 &= s_6 \\
\mathbf{x}_2 &= (s_2, s_3, s_4, s_5, s_6) \\
\mathbf{y}_2 &= s_7 \\
&\ldots \\
\mathbf{x}_{495} &= (s_{495}, s_{496}, s_{497}, s_{498}, s_{499}) \\
\mathbf{y}_{495} &= s_{500}
\end{aligned}$$

and thus $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, 495\}$ for this particular case.

How can these examples be used to understand the behavior of Amazon's stock price over time? The general idea, to be made more concrete below, is that each example entails a small piece of information about the fluctuation of the stock price. An intelligently designed machine learning system might then be able to unify all these examples into a more comprehensive view on the behavior of the stock price. In particular, after the system has digested all the provided examples, we will give the trained system the stock prices $(s_{496}, s_{497}, s_{498}, s_{499}, s_{500})$ of the last five days, and we hope that the system might be able to provide a useful prediction of the as yet unknown stock price s_{501}. Whether the system will succeed in generating a good prediction will depend on several factors, such as the type of machine learning system that is used, the number of examples that has been collected, and the phenomenon itself (if Amazon's stock prices exhibit very chaotic behavior, there is no hope that a system can be constructed that produces reliable predictions).

Supervised machine learning systems are typically applied to *regression* or *classification* tasks. Predicting Amazon's stock price is an example of regression, where the goal is to determine the continuous value of one or more given output variables given the value of one or more input variables. Other examples of regression tasks include the prediction of swimming performance given input variables such as swimming technique, lung capacity, and hand and foot size [49], the prediction of the price of real estate in terms of variables such as built-up area, the number of bedrooms and the location [67], and modelling the relationship between stream water temperature and air temperature, where either of the variables can be taken as input variable [42]. In a classification task, the output variables assume discrete values, and each of these values represents one of the possible classes. The goal is to predict the correct class of a given input. An important example in medicine is the classification of a tumor as either malignant or benign, given input data that consist of medical records relevant to the type of cancer involved [36]. Another widespread application of machine learning classification is to distinguish between spam

emails and non spam ones, where the input variables include email header information [5]. The number of classes in both classification examples is two, but there are plenty of applications where more than two classes are appropriate. Consider as illustration the classification of Spanish red wines into seven distinct groups, based on metals in the wine (B, Ca, Fe, etc) and on certain physico-chemical properties such as pH and acidity [56].

The word *supervised* is properly chosen, as the system can be considered to learn under the supervision of an imaginary master. The master provides the system with the inputs from the examples, and tells the system which output should be produced for any given input. It is then up to the system to behave in accordance with the instructions from the master. Although the explicit goal of supervised machine leaning is often to perform prediction, there are frequently some less pronounced goals, such as gaining understanding of the considered phenomenon. These two goals are often complementary. Determining the relationship between a cancer category (malignant or benign) and information represented by medical records, may increase understanding of cancer, while at the same time doctors might use the trained system to predict the cancer type of a certain patient given his medical records.

1.2.2 Unsupervised machine learning

Unsupervised machine learning is applied when the collected examples only have an input part. The main application of unsupervised machine learning is clustering. In this case the task of the machine learning system is to identify groups (commonly referred to as clusters) in the given data set, with members of the same group sharing certain characteristics. This is analogous to classification, although in the case of clustering no class labels are available, as output data is not provided. A first consequence is that the system has to define meaningful groups without external guidance. A second consequence is that it also has to find an appropriate number of clusters, since in typical applications of clustering a suitable number of clusters is not known. The selected number of clusters is frequently the result of a trade-off. On the one hand, a very large number of clusters is undesired, since then each cluster will only contain a small fraction of the data points, and a cluster with only a few elements is not what we intuitively mean by a *group* of elements. On the other hand, a very small number of clusters might also be undesired, since this might imply that any cluster has a large number of different elements, perhaps meaning that these elements are not very similar, thereby contradicting the very definition of clustering.

The best-known unsupervised machine learning technique to perform clustering is K-means [33]. This is a simple and intuitive algorithm, and it is instructive to describe it, although we will not discuss all the details. Denote the set of given examples by $D = \{\mathbf{x}_i \mid i = 1, \ldots, N\}$, where, as a reminder, each example contains only an input part. K-means assumes the number of clusters to be given, which is denoted by k. As said above, in typical applications an appropriate value for k is not known. Luckily, some heuristic methods have been developed to identify a suitable number

of clusters [12, 76, 79, 82, 92]. We assume that such a method has been applied, and thus that k has been assigned a proper value. K-means defines clusters in terms of so-called centroids. A centroid is an element of the data space that is representative of the cluster. For example, if the data set would contain basic characteristics of people, like age, weight, height and sex, it might be the case that two clusters are detected, the one consisting of males, the other of females. An appropriate centroid of the male cluster would then be the vector with as entries the average age, the average weight and the average height of the men that belong to the considered data set. The same applies to the female cluster. With this background knowledge on clustering, we can now describe the K-means algorithm:

1. Randomly select k elements from the data set D. These elements are used as initial centroids.
2. For each element $\mathbf{x}_i \in D$, compute the distance to each centroid. Assign $\mathbf{x}_i$ to the cluster that has the centroid with smallest distance to $\mathbf{x}_i$. The choice of distance measure depends on the application at hand.
3. Update the k centroids: recompute each centroid as the average of the elements in D that are member of the corresponding cluster.
4. Go back to step 2 unless there is convergence. Convergence is reached when the membership of each $\mathbf{x}_i$ is not changing anymore.

Notice that steps 2 and 3 are repeatedly performed. In step 2, the data elements are assigned to the cluster with the closest centroid, whereby it is possible that some elements are assigned to a different cluster compared to the assigned cluster in the previous iteration. In step 3, the centroids are recomputed, which may result in new centroids compared to the previous iteration due to the possibility that certain elements have changed membership in step 2. The algorithm terminates when the membership that is assigned to each data element in step 2 is the same as in the previous iteration.

Let us briefly review an application of K-means that has been described in the literature. In [3] the authors show how K-means can be used to make customer service in telecommunication companies more efficient. The grouping of customers is performed according to their profitability. The profitability of a customer is defined as the profit the company makes from serving the customer over a specified period of time. The customers who generate more profit for the company are called high profitability customers. Input variables are defined in terms of RFM, which is the abbreviation of Recency, Frequency and Monetary. These terms refer to how recently a customer has purchased (recency), how often the customer purchases (frequency), and how much the customer spends (monetary). These abstract concepts need to be converted into numeric

Cluster	Average R	Average F	Average M
1	8.62	3.33	5.98
2	2.50	12.67	175.67
3	5.82	5.00	12.38
4	3.47	9.00	59.33

Figure 1.2: The centroids of the four clusters (from [3])

variables, in order to obtain examples in the right format. The set of examples has a suitable format provided that, as we have seen, it can be described as $D = \{\mathbf{x}_i \mid i = 1, \ldots, N\}$, where each $\mathbf{x}_i$ is a vector containing real numbers as components. To this end, the authors introduce three numeric input variables, the first one to capture recency (referred to as variable R), the second one to take frequency into account (variable F), and the third one to incorporate the monetary aspect (variable M). To be concrete, for the ith customer the corresponding example (x_{i1}, x_{i2}, x_{i3}) is defined as follows:

$$
\begin{aligned}
x_{i1} &= \text{Average time duration between two calls over 1 month (in hours)} \\
x_{i2} &= \text{Average number of calls per day over 1 month} \\
x_{i3} &= \text{Bill value over 1 month}
\end{aligned}
$$

The data of 100 customers is used to construct the examples, meaning that $N = 100$. Next, the authors determine a suitable number of clusters, but we do not discuss the algorithm that was used for this task. Using that algorithm, it is found that $k = 4$ is an appropriate number of clusters. Each cluster can then be analyzed in terms of its centroid, which is the vector containing the average value of the three input variables R, F and M, where the average is taken over the customers that are assigned to the corresponding cluster. Fig. 1.2 shows the centroids of the four clusters. Considering the values of the centroids, the authors interpret each cluster in more human-understandable language as follows:

- Cluster 2: High profitable customers.
- Cluster 4: Profitable customers.
- Cluster 3: Medium profitable customers.
- Cluster 1: Low profitable customers.

What is the use of this customer segmentation? The segmentation suggests how a diversified strategy could be implemented to increase revenue. High profitable customers, i.e. members of cluster 2, are very important to the company. Consequently, it is wise to provide them certain

benefits, such as discounts, gifts on special occasions such as their birthday, and a special hotline service where more skilled and senior customer officers handle their questions and concerns. These benefits are intended to reduce the probability that a customer from cluster 2 would terminate his contract with the company. Since these benefits come, of course, also at a cost, the benefits with the highest costs should not be provided to the low profitable customers, which are the customers that have been assigned to cluster 4. Following these ideas, a cost-benefit analysis by the economic department of the company may result in a detailed and refined strategy for each cluster. How to handle a new customer? We simply wait one month to collect his or her data, since the three input variables refer to data over a period of one month, and we assign the customer to the cluster with closest centroid. For example, if the values of the three input variables for this customer are given by (3.50, 8.70, 60) we assign the customer to cluster 4, since the corresponding centroid (3.47,9.00,59.33) is closest to the customer's data (measuring distance according to the Euclidean distance measure[1]). The strategy implemented by the economic department to serve customers in cluster 4 is then to be applied to this customer.

The above example illustrates two properties of machine learning systems that have been referred to in Section 1.1. First, such systems imitate human behavior in solving problems. The segmentation of customers could also have been done by a trained human being without the aid of a computer. However, the processing capacity of humans is far lower, and it would take an incredible amount of time to fulfill the task. The authors did not provide details on the computation time, but it definitely took K-means not longer than a few minutes to produce the four clusters and their accompanying centroids. The imitation of human behavior can also be considered from another perspective: provide the four clusters to a member of the economic department of the company, and he will not be able to tell whether the result was produced by a human being or by a computer. Both are able to generate the non trivial information encapsulated in the four clusters, meaning that even if the result was produced by K-means, it would *seem* as if human intelligence was the sole driving force in fulfilling the segmentation task. Secondly, machine learning systems imitate *complex* human behavior. Indeed, segmenting customers based on some collected data, thereby producing a prototypical customer for each cluster (as represented by its centroid) is a non trivial objective, its complexity far exceeding the task of, for example, finding a specific word in a given text.

1.3 Steps in constructing a machine learning system

In this section we provide a general description of constructing a machine learning system. There are several steps involved in the construction, starting from the collection of examples and ending with a machine learning system that is properly trained. After the construction, the machine learning system is to be used for understanding or prediction, or for some other related

[1] `https://en.wikipedia.org/wiki/Euclidean_distance`

goal. The description is general, because in this chapter we do not yet consider any specific machine learning system, such as artificial neural networks, which are the focus of the next chapter. Our description restricts to supervised machine learning systems, as the remainder of this work will be solely devoted to this type of systems. The construction of a supervised machine learning system depends, to some extent, on its intended use, in particular whether the goal is classification or regression. In avoiding that our description would become too lengthy and possibly confusing, we further restrict our outline to supervised machine learning systems that are used for regression.

1.3.1 Step 1: collecting examples

Given a real world phenomenon that is to be studied, such as the relationship between stream water temperature and air temperature, the first step is to collect examples, as particular instances of the phenomenon. We have already described the use of examples and the proper format in which the examples need to be presented to the system. What remains to be discussed is the precise relationship between $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, N\}$, the set of collected examples, and the phenomenon itself. Let us assume, as is common practice, that the phenomenon can be described by a certain function f (this is not possible for all phenomena, e.g. the points (x, y) on a circle cannot be represented by a function $y = f(x)$). Intuitively, the collected examples are then instances of f, in the sense that $\mathbf{y}_i = f(\mathbf{x}_i)$. However, in almost all case studies measurement error is present. This obviously applies to cases where rather complex measurement instruments are used, such as measurements involving phenomena in physics (e.g. the measurement of the gravitational constant [53, 54, 65]) or measurements related to biology (e.g. related to gene expression arrays [43, 63, 78]), but measurement errors are also to be expected in collected data that may be falsely assumed error-free by researchers not familiar with data mining, such as in survey data [55, 68] and in economic time series [4, 24]. Due to the presence of measurement errors, it is therefore more correct to describe the relationship between the examples $(\mathbf{x}_i, \mathbf{y}_i)$ and the real world phenomenon f as

$$\mathbf{y}_i = f(\mathbf{x}_i) + \boldsymbol{\epsilon}_i \tag{1.1}$$

where $\boldsymbol{\epsilon}_i$ represents the measurement error. The error $\boldsymbol{\epsilon}_i$ is typically not (exactly) known. Data analysis requires that ϵ_{ij} (the jth component of $\boldsymbol{\epsilon}_i$) is small compared to y_{ij} (the jth component of $\mathbf{y}_i$) for all j, otherwise the examples are not a reliable representation of the studied phenomenon, and any machine learning system will fail in correctly understanding the considered event.

1.3.2 Step 2: choice of a specific machine learning system

As outlined in Section 1.2, the phenomenon under study, together with the related examples, determine the choice of machine learning system. For example, it is clear that if the examples do not have an output part, an unsupervised machine learning system should be applied. However,

the categories of systems that we have described still contain plenty of specific machine learning systems to choose from. As an example, a lot of supervised machine learning systems that perform regression have been developed, such as artificial neural networks [7], support vector machines [31], kriging [42] and random forests [34]. Although the phenomenon and the structure of the examples indicate which *type* of machine learning system is applicable, choosing a *specific* system requires further decision criteria. One obvious criterium is the expertise of the researcher, which will further restrict the set of feasible systems to a subset of systems in which the researcher has gained experience. Two other factors that are almost always taken into account are computation time and the degree of sophistication of the system. Intuitively, a more advanced system is to be preferred over a less advanced one. However, there are at least three reasons why this intuitive selection rule should be applied with great care. First, cases have been reported where a simpler model outperforms a more complex one [17]. Secondly, a more complex model often comes with a larger computation time. One of the explanations for this observation is quite simple: complexity often relates to the number of parameters that a model has, and computation time increases as more parameters need to be optimized. Thirdly, a system might even be too complex, in the sense that it results in *overfitting*, an artifact that has to be avoided by all means. Overfitting deserves an extensive description (Section 1.3.4).

1.3.3 Step 3: training

Having chosen a certain machine learning system, such as a neural network, the next step is to ensure that the system learns to understand the considered phenomenon, which it will do by using the examples $D = \{(\mathbf{x}_i, \mathbf{y}_i) \,|\, i = 1, \ldots, N\}$ that have been collected in the first step.

1.3.3.1 Splitting the set of examples into a training set, a validation set and a test set

Before actual training starts, we split the set of examples into three disjoint subsets, referred to as the training set D_{Tr}, the validation set D_V and the test set D_{Te}, such that $D_{Tr} \cup D_V \cup D_{Te} = D$. Each subset will play a central role in further steps in constructing the machine learning system. A common rule of thumb to perform this splitting is the 60/20/20 rule [48, 91], meaning that the training set contains 60% of the examples, the validation set 20% of the examples, and the test set the remaining 20%. That is,

$$D_{Tr} = \{(\mathbf{x}_i, \mathbf{y}_i) \,|\, i = 1, \ldots, \lfloor 0.6N \rfloor\} \tag{1.2}$$

$$D_V = \{(\mathbf{x}_i, \mathbf{y}_i) \,|\, i = \lfloor 0.6N \rfloor + 1, \ldots, \lfloor 0.8N \rfloor\} \tag{1.3}$$

$$D_{Te} = \{(\mathbf{x}_i, \mathbf{y}_i) \,|\, i = \lfloor 0.8N \rfloor + 1, \ldots, N\} \tag{1.4}$$

where $\lfloor . \rfloor$ denotes rounding to the nearest integer.

1.3.3.2 Meaning of training a machine learning system

Essentially, the task of a machine learning system is to provide the user a mathematical representation of the real world phenomenon. In the more exact terminology of Section 1.3.1, the machine learning system that has been chosen in the previous step should, ideally, identify the function f. However, completely identifying f is unfeasible, since we only have a limited number of examples $(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots, N$. Indeed, due to practical or budgetary limitations, measurements will only be collected in a limited number of inputs. This implies that there will exist input values for which the corresponding output is unknown. The given examples thus provide *a limited view* on the considered phenomenon. Furthermore, the examples give *an obfuscated view* on the phenomenon, as the examples are only approximate instances of the phenomenon due to measurement errors, which we expressed by equation (1.1). This means that the best we can hope for is to identify a function $\hat{f}$ that is a satisfactory approximation to f.

The choice of a specific machine learning system implies at once the choice of a functional form $\hat{f}$ that is to be used as approximation to f. For example, a radial basis function network [30, 46, 60], which is a supervised machine learning system that can be used for regression, is in its simplest form expressed as

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{M} w_j \exp\left(-\left|\left|\mathbf{x} - \mathbf{c}_j\right|\right|^2\right) \tag{1.5}$$

where $||.||$ denotes the Euclidean norm, and where M, w_j and $\mathbf{c}_j, j = 1, \ldots, M$, are parameters. The description of $\hat{f}$ is incomplete as long as no values have been assigned to these parameters. Training is then to be understood as the *determination of suitable values* for the parameters, i.e. such that $\hat{f}$ is a good approximation of f.

The relationship between collecting examples (step 1), choosing a specific machine learning system (step 2) and training the system (step 3) can thus be described as follows. The choice of a specific machine learning system amounts to selecting a function $\hat{f}$, which has some parameters with as yet undetermined values, while training refers to determining values for the parameters, using the collected examples, in such a manner that $\hat{f}$ is a good approximation of f.

How to find suitable values for the parameters? An intuitive idea is to randomly try different values for the parameters, each time evaluating the performance of the machine learning system for the given choice of parameter values, and then select the parameter values for which performance is highest. For example, suppose that the inputs of our examples contain three components, and assume that the radial basis function network (1.5) has been chosen as machine learning system. Then we could try, for example, the following parameter values:

$$M = 2, w_1 = 0.2, w_2 = 0.7, \mathbf{c}_1 = (0.9, 1.2, -0.1), \mathbf{c}_2 = (2.3, 0.75, -2) \tag{1.6}$$

and

$$M = 2, w_1 = 0.11, w_2 = 3.4, \mathbf{c}_1 = (1.7, -0.2, 0.75), \mathbf{c}_2 = (3, 1.5, -2.7) \tag{1.7}$$

and possibly many others, and we select the values for which the system performs best. However, the comparison of the performance of different systems requires a *measure*, in the same way as comparing students requires an objective measure, in particular grades. The comparison of machine learning systems that have been assigned different parameter values, is performed using a so-called *error measure*.

1.3.3.3 Error measure

An error measure evaluates the performance of a machine learning system by computing the error between the outputs generated by the system and the output part of the examples. Conceptually speaking, the error that the system makes in representing the ith example $(\mathbf{x}_i, \mathbf{y}_i)$ is typically computed as $\left|\left|\mathbf{y}_i - \hat{f}(\mathbf{x}_i)\right|\right|^2$, where $\hat{f}(\mathbf{x}_i)$ is the output that the machine learning system produces for the given input $\mathbf{x}_i$ (cf. Section 1.3.3.2), and where the notation $||\mathbf{v}||$ refers to the norm (i.e. length) of a vector $\mathbf{v}$. The total error is then simply defined as the average of the errors on all examples:

$$E(\hat{f}, D) = \frac{1}{N} \sum_{i=1}^{N} \left|\left|\mathbf{y}_i - \hat{f}(\mathbf{x}_i)\right|\right|^2 \tag{1.8}$$

The above notation explicitly takes the set of examples D into account, since it is often useful to compute the error measure only for a subset of D, e.g. for the training set D_{Tr} in which case the error measure is given by

$$E(\hat{f}, D_{Tr}) = \frac{1}{|D_{Tr}|} \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in D_{Tr}} \left|\left|\mathbf{y}_i - \hat{f}(\mathbf{x}_i)\right|\right|^2 \tag{1.9}$$

where $|D_{Tr}|$ denotes the number of elements in D_{Tr}. While $E(\hat{f}, D)$ measures the performance with respect to *all* examples, $E(\hat{f}, D_{Tr})$ measures how well the system performs on the training set.

It is important to realize that the error measure $E(\hat{f}, D)$ only approximately tells us how well the system performs in representing the phenomenon. The reason is that the error measure computes the error with respect to the examples. As outlined above, the examples are only approximate instances of the considered phenomenon, because of the presence of measurement error (cf. Section 1.3.1).

When multiple specific machine learning systems are considered where the parameters have been given certain values, it is convenient to use slightly different notations for the different systems, e.g. $\hat{f}_1, \hat{f}_2$, etc. For example, the radial basis function network with parameters given by (1.6) could be denoted by $\hat{f}_1$, while the one with parameters given by (1.7) may be referred to by

$\hat{f}_2$. The error measure then allows to compare their performances, and it is plausible to say that the first radial basis network performs better than the second if

$$E(\hat{f}_1, D) \quad < \quad E(\hat{f}_2, D) \tag{1.10}$$

and vice versa. However, this is not really the way that machine learning experts compare the performance between systems with different parameter values. The next sections will clarify how performance comparison should be appropriately performed. At any rate, the concept of error measure will be an essential ingredient in comparing given systems.

1.3.3.4 Training algorithm

An intuitive way to determine suitable parameter values for a given machine learning system is to try at random several values, then comparing the systems in terms of the error measure E as given by (1.8), and finally selecting the values corresponding to lowest E. There are, however, two problems with this intuitive optimization procedure. The first problem relates to overfitting, which is described in Section 1.3.4. The second problem is that the number of feasible parameter values is very often infinite, as is the case for the radial basis network considered in Section 1.3.3.2. Thus, in case of bad luck, by randomly guessing values for the parameters we might end up with systems that *all* have a large error.

A training algorithm is an algorithm that tries parameter values in a sophisticated way, typically by employing mathematical techniques to update current parameter values to new values for which E is reduced. Generally speaking, a training algorithm typically starts with an initial random guess for the parameter values, but repeatedly applies a mathematical method to obtain an appropriate sequence of parameter values for which the corresponding E is decreasing. That is, only the first value is a random guess, and all other values are derived from the previous parameter values by a theoretically founded method that tries to guarantee that the new values correspond to a lower error than the previous ones.

The best-known training algorithm is undoubtedly gradient descent [72], which is conveniently explained for the case where the machine learning system has only one parameter $\alpha \in \mathbb{R}$. Given a randomly chosen initial value for this parameter, say a_1, gradient descent will compute the derivative of the error measure in a_1. Calculus then tells us that the slope in a_1 describes the behavior of the error function nearby a_1. For example, if the slope is positive, then values smaller then a_1 will result in smaller values of E. Gradient descent will then select such a smaller value a_2 as new and better value for the parameter α. If the slope is negative, a_2 will be chosen as some value that is larger than a_1. This procedure is then repeated by computing the derivative of E in a_2 and the sign of the derivative is again used as guideline to select a new value a_3. To avoid that the procedure would go on forever, a stopping criterion is needed. A plausible stopping criterion is to stop training when the error is below a certain threshold. In Section 1.3.4 we come back to the

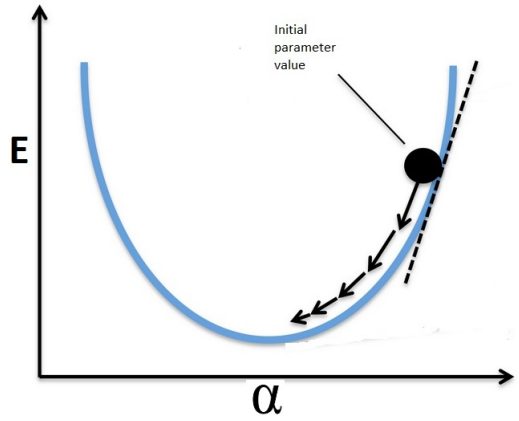

Figure 1.3: Illustration of gradient descent in case there is only one parameter. Image adapted from `https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html`

stopping criterion, as this criterion is of the utmost importance to end up with a reliable machine learning system.

The gradient descent technique is illustrated in Fig. 1.3. Other, more advanced, algorithms to minimize E exist, such as the Levenberg-Marquardt method [35].

1.3.4 Step 4: validation

The previous steps resulted in the construction of examples (the set D, with as subsets the training set D_{Tr}, the validation set D_V and the test set D_{Te}), and some specific machine learning system, expressed as a function $\hat{f}$ that contains parameters and that is to be used as an approximation to the real world phenomenon f. Section 1.3.3.4 describes the use of a training algorithm, which is able to efficiently update the parameter values such that the error measure decreases as consecutive parameter values are selected. We did not specify yet which error measure should be used in applying the training algorithm. Intuitively, we use the error measure $E(\hat{f},D)$, described by (1.8), which determines the error with respect to all examples. After all, we want the machine learning system to be a good representation of all examples. Paradoxically, a system that performs very well with respect to all examples might actually be a very bad system. The anomaly to be blamed for this perplexing result is known under the name overfitting.

1.3.4.1 Overfitting

Overfitting is one of the main concerns of a machine learning practitioner [8]. To understand why practitioners might not be confident that their system will perform well on its intended task, although their training algorithm has identified parameter values for which $E(\hat{f},D)$ is very low, we need to recall the actual task of a machine learning system. That task is to *generalize* the information that is, in some way, encapsulated in the given set of examples (cf. Section 1.1). This

means that it is not sufficient that a system performs well on the examples; actually, our desire is that the system does a good job on *previously unseen* instances of the considered phenomenon. A previously unseen instance refers to some input-output pair $(\mathbf{x}, \mathbf{y})$ that relates to the studied phenomenon, but that is not part of the set of examples D. Although it is not one of the collected examples, it is essential that the system is able to properly process this instance too. For example, consider a system that has been trained to classify the tumour of patients into "benign" and "malignant", using collected examples of patient data for which the correct class is known. The ultimate purpose of such a system is, however, to correctly classify the tumour of a *new* patient, given certain medical data associated with this patient and for whom the correct class is currently unknown. Thus whenever the system has seen a lot of examples, consisting of medical input data together with the correct class (benign or malignant) as output data, it is desired that the system is able to use this experience to assign the right class to a new patient.

The description provided in Section 1.3.1 allows to gain insight into the overfitting issue. Equation (1.1) shows that examples are only *approximate* instances of the real world phenomenon, since these examples are corrupted by noise. If a machine learning system is trained in such a way that the error $E(\hat{f}, D)$ is zero, the undesired consequence is that the system has not only modeled the phenomenon f, but also the measurement errors. In this case, the equation shows that $\hat{f}(\mathbf{x}_i) = f(\mathbf{x}_i) + \boldsymbol{\epsilon}_i$, while what we ideally want to have is that $\hat{f}(\mathbf{x}_i) = f(\mathbf{x}_i)$. The holy grail in machine learning is, therefore, to have ingenious methods to separate the phenomenon, i.e. f, from the measurement errors, i.e. ϵ. The next section describes a commonly used technique to pursue this goal.

Overfitting can be illustrated by a case study in climatology. Fig. 1.4 shows yearly temperature anomalies from 1880 to 2014 as recorded by several institutes, namely NASA, NOAA, the Japan Meteorological Agency, and the Met Office Hadley Centre (United Kingdom). Although the observed phenomenon is, of course, the same, the graphs produced by the different institutes are not identical, due to different measuring techniques. Another observation is that the graphs exhibit minor variations over short time spans, which gives the overall pattern an irregular appearance. Such fluctuations are the result of a host of factors with random behavior, which typically have a temporal and minor influence. A popular research question in climatology is to make predictions of the mean temperature for, e.g., the next 20 years. To answer such a question, it is not meaningful that the machine learning system incorporates the short-term random fluctuations. Rather, the system should be able to detect the underlying trend, which is driven by the non random factors that have a dominant influence on the climate. The temperature anomalies from 1970 on allows the system to gain insight into the evolution of the temperature over the years, but what actually matters is that this experience is used to predict how this evolution is to continue in the future. An acceptable candidate for $\hat{f}$ is the thick black line shown in the figure. Although it is not a perfect fit to the examples, it seems to be a good representation of how the temperature has evolved in the past and how it will probably evolve in the future.

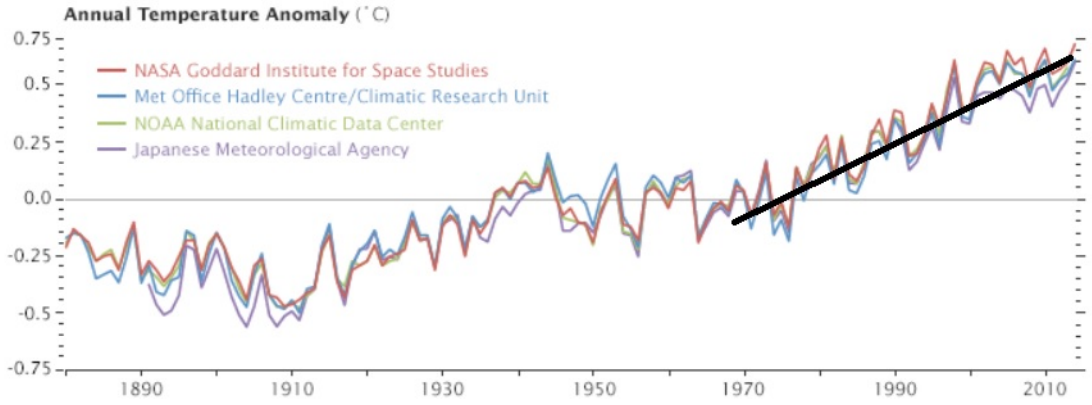

Figure 1.4: A first illustration of overfitting. Image adapted from `https://earthobservatory.nasa.gov/WorldOfChange/DecadalTemp`)

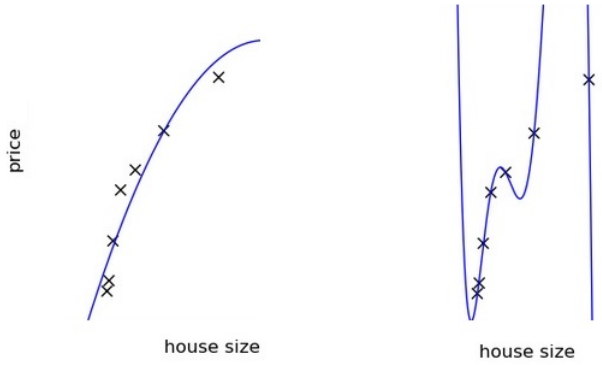

Figure 1.5: A second illustration of overfitting

Another illustration of overfitting relates to modelling the price of a house in terms of its size[2]. A reseacher has collected some examples of this phenomenon, where the examples are in the format (house size, house price), and has trained two different machine learning systems on these examples. The resulting $\hat{f}$ for both systems is shown in Fig. 1.5. It is seen that the second system (on the right) perfectly represents the examples. However, the fact that $\hat{f}$ in this case shows very unrealistic behavior, with extremely large prices for certain house sizes, suggests that it will perform poorly when a new house size is presented to the system with as goal to predict its price. We then say that this system has been overfitted. The machine learning system shown on the left is a much more appealing model of the relationship between the price of a house and its size. Consequently, the system that perfectly fits the examples is *inferior* to the system that makes errors on the given examples.

[2] `https://srdas.github.io/DLBook/ImprovingModelGeneralization.html`

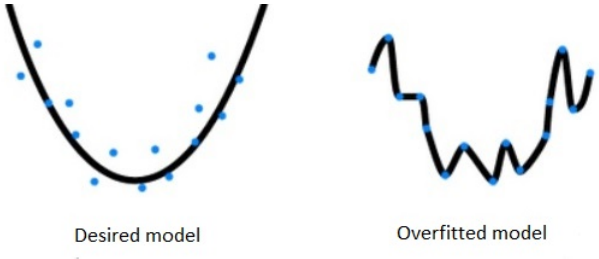

Figure 1.6: A third illustration of overfitting. Image adapted from `https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42`

A final illustration of overfitting is displayed in Fig. 1.6.

1.3.4.2 Avoiding overfitting with the use of the validation set

Training a machine learning system, i.e. optimizing the parameters with the use of a training algorithm, by minimizing the error $E(\hat{f},D)$, might result in overfitting. The previous sections illustrated this observation. Since the ultimate goal is to train a system in such a way that it performs well on unseen examples, the most popular solution to avoid overfitting is to *train* the algorithm on *a certain subset* of the examples, and to *evaluate* its performance on *another subset*. In other words: learning is done using certain examples, while the presence of overfitting is detected by evaluating the performance of the system on other examples.

This is why in Section 1.3.3.1 the set of all examples D is subdivided into several sets, namely a training set D_{Tr}, a validation set D_V and a test set D_{Te}. The use of the test set is described in Section 1.3.5. In this section we explain how the training set and the validation set are useful in training the system, while at the same time overfitting is avoided.

In plain words, the strategy to properly train the system is to apply the training algorithm only on the examples in the training set. This means that the inputs $\mathbf{x}_i$ from the examples in D_{Tr} are presented, one by one, to the system, and depending on the difference between the corresponding output $\mathbf{y}_i$ and the produced output $\hat{f}(\mathbf{x}_i)$, the training algorithm updates the parameter values to new values (cf. Section 1.3.3). More specifically, the training algorithm chooses the new parameter values in order to reduce $E(\hat{f},D_{Tr})$. However, from time to time, for example whenever ten other examples from D_{Tr} have been presented to the system, the error is computed with respect to the validation examples, i.e. the examples in D_V. The examples in D_V are, as far as the system is aware of, new (i.e. previously unseen), since they have not been used to determine the parameter values of the system (remember that parameter values are updated by solely using the examples in the training set D_{Tr}). The *optimal* parameter values are then those for which the error on the validation set $E(\hat{f},D_V)$ is minimal. This accomplishes our goal, since ensuring that the performance is best with respect to unseen examples is equivalent to ensuring generalization.

A typical procedure for training is then as follows:

1. Assign initial values to the parameters, e.g. randomly chosen real numbers.
2. Present an example from the training set D_{Tr} to the system, and apply the training algorithm to find new parameter values that decrease the error $E(\hat{f}, D_{Tr})$.
3. Every time a certain number of examples have been presented to the system, e.g. every ten examples, compute the performance of the machine learning system on the validation set D_V, i.e. compute the error $E(\hat{f}, D_V)$.
4. Repeat the previous two steps until the error $E(\hat{f}, D_V)$ drops below a predefined threshold.

The training error $E(\hat{f}, D_{Tr})$, which is the error that the system makes on the training examples, typically decreases as more examples from the training set are presented. This is straightforward, as the training algorithm updates the parameter values in order to reduce the training error. The story is different for the validation error $E(\hat{f}, D_V)$. Initially, we expect the validation error to decrease as training examples are presented, because by digesting examples the system starts to grasp the phenomenon. However, after many examples have been presented, overfitting might make its appearance. At that point, the training error is becoming very low, which is a bad thing, since it means that the system starts to also represent noise, such as measurement errors. At that same point, however, the validation error will begin to increase, because the fact that overfitting pops in implies that the system starts to perform worse on the unseen validation examples. Thus we expect that the training error keeps decreasing, while the validation error typically decreases until overfitting emerges, and then starts to increase. The evolution of the training error and the validation error as examples from the training set D_{Tr} are presented to the system is illustrated in Fig. 1.7. The vertical line indicates where training should be terminated, corresponding to the minimum value of the validation error. At that moment it is not advised that training is still continued, because then overfitting will come alive, and generalization will be gone forever.

1.3.5 Step 5: testing

Having designed a machine learning system that avoids overfitting, it is ready to be applied to its intended task. However, it is common practice that the machine learning researcher first presents some results that convincingly show that the system will have high performance on this task. It is not helpful that he tries to be convincing by reporting the training error, since a low training error might actually indicate overfitting, as has been repeatedly argued above. Nor is it very meaningful to refer to the validation error, because the system has been developed to minimize that error. So of course the validation error is low. Instead, the researcher should make a case for using his system by applying it to a set of examples that has not be used, in any way,

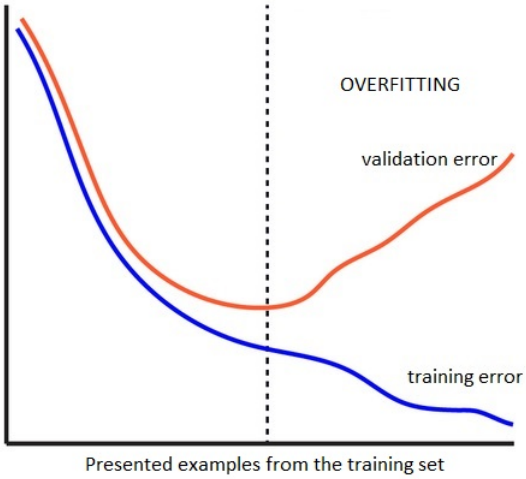

Figure 1.7: Evolution of training error and validation error as examples from the training set are presented to the machine learning system

in training the system. This is why one of the subsets of the set of examples D is the test set D_{Te}: we have kept aside these examples from training to be used for the ultimate performance test of the trained system. Having trained the system using the training set and the validation set, the researcher now presents the inputs from the test examples to his system. The system computes the corresponding output, which is compared to the output parts of the test examples. In other words, the so-called test error $E(\hat{f}, D_{Te})$ is computed. The lower the test error, the more convincing the researcher will be in claiming that his system is amazing.

CHAPTER 2

ARTIFICIAL NEURAL NETWORKS AS A PRIME EXAMPLE OF MACHINE LEARNING

2.1 Introduction

The previous chapter started by raising the question whether a computer is capable to *mimic* human behavior in performing complex tasks. An even more profound question is whether a machine is actually able to *think* [22]?

For a long period, investigating such questions was considered the privilege of the philosophical arena, with philosophers such as Descartes arguing that there is a real distinction between the mind and the body [18], while others, e.g. Hobbes, claimed that everything consists only of matter or is ultimately dependent upon matter [21]. If the materialists are right in stating that the mind is merely a physical substance, there might be some prospect in answering the above question in the affirmative sense. Yet, it is a challenge in itself to define the meaning of intelligence, let alone to answer this deep question in a straightforward way. Already in 1950 Alan Turing devised an elegant test that allows to evaluate to what extent a machine is able to think, without the need to come up with a proper definition of intelligence [83]. His test is still the most popular way to determine a machine's intelligence and has been coined the Turing test. The test implements the idea that when a human being is unable to determine whether he is having a conversation with a machine or with another human being, the machine has reached the level of human intelligence. In a practical setting, a human interrogator is having a written conversation with messages exchanged through, for example, a chatbox with another human being and with a machine. This conversation is supposed to entail a normal, everyday talk with the other two entities. The machine is said to have passed the Turing test if the interrogator is unable to distinguish the machine from the human. The test is illustrated in Fig. 2.1. So far, no

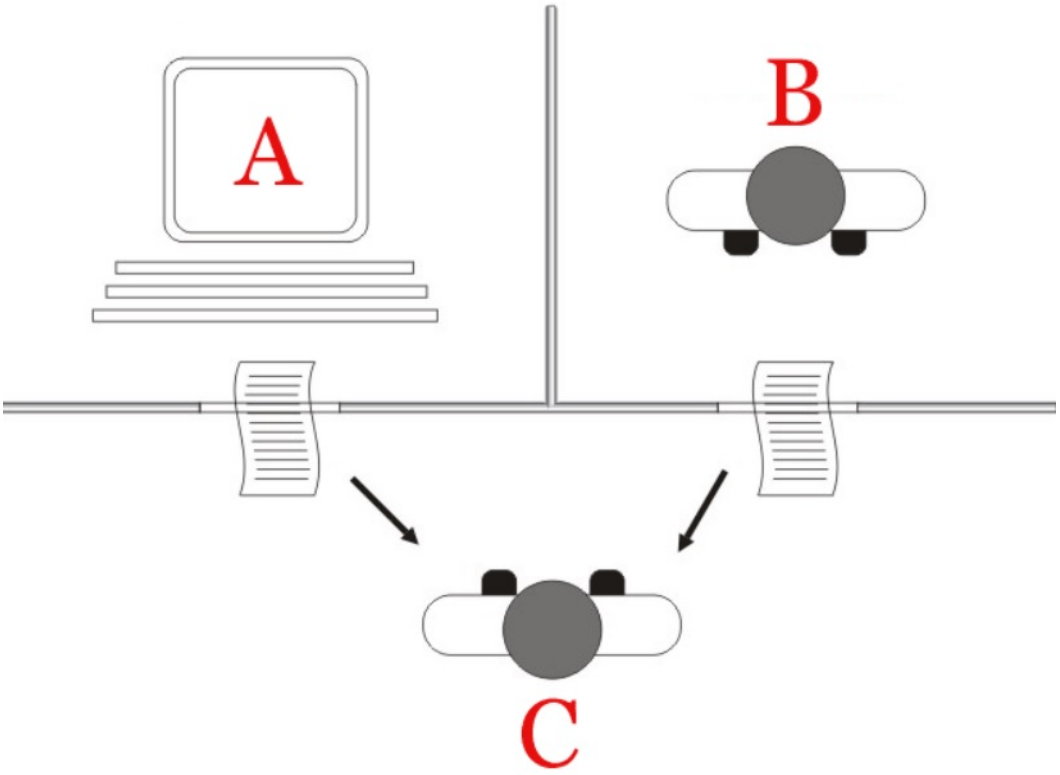

Figure 2.1: Illustration of the Turing test, with C the human interrogator, and A the machine and B the other human being (or vice versa). Image adapted from [69].

machine has been able to pass the Turing test, but Ray Kurzweil, author of some well known books on artificial intelligence (AI) such as "The singularity is near: when humans transcend biology" [38], has made an interesting bet on the future of artificial intelligence. According to his prediction, in 2029 the first computer will pass the Turing test[1].

After having outlined the general principles of machine learning in the previous chapter, it is instructive to take a closer look at one of the many existing machine learning systems. Because of their interesting history and because they have many applications, artificial neural networks (ANNs) will be described in some detail in this chapter. Originally, they were developed in an attempt to apply the working principles from the brain in an artificial environment. That explains why the fundamental unit of an ANN is called a *neuron*, the same term that is used for the basic units in the brain. Given the degree of intelligence with which humans are endowed, it might be surprising that neurons are, in fact, rather simple units [9]. Yet, as gradually became clear, high complexity can be attained by the *interaction* amongst a *very large number* of such simple building blocks (it has been estimated that the brain contains about 100 billion neurons [28]). The realization that complex functions can be built by suitably combining simple functions, is

[1] `https://www.kurzweilai.net/a-wager-on-the-turing-test-why-i-think-i-will-win`

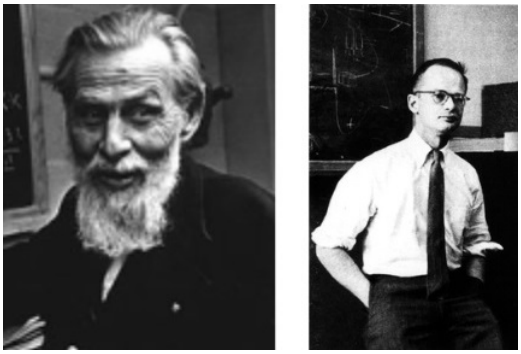

Figure 2.2: McCulloch (1889-1969) and Pitts (1923-1969). Adapted from [74].

the very cornerstone of what was to become machine learning. In the next sections we go back in time to briefly describe the origin and the subsequent evolution of ANNs. This will shed some light on how researchers successfully relied on insights from neurology to create a new kind of entity that in the future might even transcend human intelligence [37]. In describing the origin of neural networks we rely, to some extent, on [88] and on a very well written account by Andrey Kurenkov[2].

2.2 Artificial neural networks with fixed weights: the McCulloch-Pitts neuron

2.2.1 Description

The beginning of the field of artificial neural networks is widely recognized to be the seminal paper by McCulloch and Pitts in 1943 [50] (a picture of McCulloch and Pitts is shown in Fig. 2.2), who introduced the concepts of neuron and artificial neural network. The neuron they introduced was essentially just a very simple function with an arbitrary number of n input variables and a single output variable. The output variable was restricted to be binary, i.e. its value is either 0 or 1. Such a machine learning system can only be used for classification into two classes, where the value 0 corresponds to one class and the value 1 to the other class.

Their neuron, accepting n input values and producing either 0 or 1 as output, operates as follows. Any value x_i of the ith input variable is multiplied by a constant value w_i, referred to as a weight, and the weighted values are then summed to obtain a single value

$$s(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i \tag{2.1}$$

with $\mathbf{x} = (x_1, \ldots, x_n)$. Following the notation of the previous chapter, the network is represented by a function $\hat{f}$, which is assumed to approximate the real world phenomenon f (cf. Section 1.3.3.2).

[2] http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/

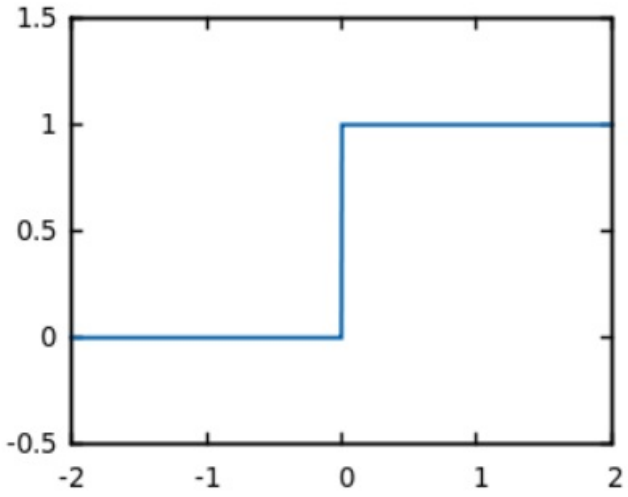

Figure 2.3: Graphical representation of τ with $\alpha = 0$.

McCulloch and Pitts defined the output $\hat{f}(\mathbf{x})$ as 1 if $s(\mathbf{x})$ exceeds a predefined threshold value α and 0 otherwise. In other words:

$$\begin{aligned} \hat{f}(\mathbf{x}) = &\quad 1 \quad \text{if } s(\mathbf{x}) > \alpha \\ = &\quad 0 \quad \text{otherwise} \end{aligned} \tag{2.2}$$

By introducing the so-called step function τ, defined as

$$\begin{aligned} \tau(z) = &\quad 1 \quad \text{if } z > \alpha \\ = &\quad 0 \quad \text{otherwise} \end{aligned} \tag{2.3}$$

we can describe the operation of the neuron, given by (2.2), also as follows:

$$\hat{f}(\mathbf{x}) \quad = \quad \tau\Big(s(\mathbf{x})\Big) \tag{2.4}$$

This representation will prove useful when extensions to this basic ANN are discussed. The function τ is illustrated in Fig. 2.3. In the terminology of neural networks, a function as τ is referred to as an activation function, since it determines when the neuron returns as output one, in which case it is said that the neuron is 'activated'.

An illustration of a McCulloch-Pitts artificial neural network in the case of two input variables (i.e. $n = 2$) is shown in Fig. 2.4. The figure indicates that the values x_1 and x_2 of some given example are multiplied by weights w_1 and w_2 respectively, and these weighted values $w_1 x_1$ and $w_2 x_2$ are then sent to the neuron. The neuron computes $s(\mathbf{x})$, according to (2.1), as an intermediate step in computing its output, and the final output is then calculated according to (2.2). It is

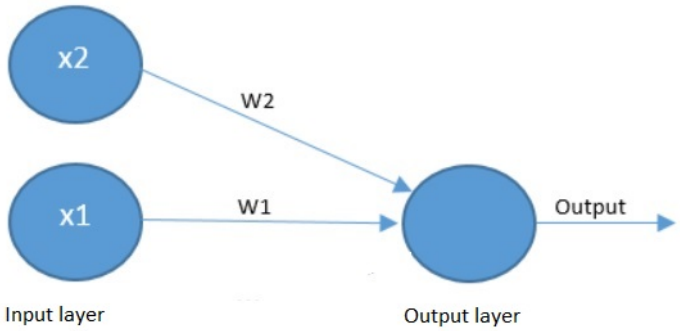

Figure 2.4: Illustration of a McCulloch-Pitts artificial neural network. Adapted from https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b.

common to regard such an ANN as consisting of two layers, where the first layer, containing the input variables, is called the input layer, and where the second layer, containing the neuron that computes the output, is referred to as the output layer. Although the input variables are not really neurons, since they are just placeholders for the input values and do not perform any operation, it is common practice to refer to the input variables also as neurons. Any unit in the network can then be called a neuron. To distinguish between the input variables and the actual neuron in the output layer, we refer to the first as *input neurons* and to the latter as *output neurons*.

2.2.2 Illustration of a two layer ANN: representing the AND function

Notwithstanding the extreme simplicity of this very first artificial neural network, it allows to represent some common logical gates. As an illustration, we demonstrate how such a network can model the logical AND function. The AND function is defined as the function that assumes two binary input values, with the corresponding output being one if both inputs equal one, and zero otherwise. For example, if the input equals (1,0) the AND function returns 0 as output. The AND function can be implemented using the ANN described in the previous section by setting the weights and the threshold to the following values:

$$\begin{aligned} w_1 &= 1 \\ w_2 &= 1 \\ \alpha &= 1.5 \end{aligned}$$

As an example of the operation of this network, consider as input $\mathbf{x} = (1,0)$. We first compute $s(\mathbf{x})$ according to (2.1):

$$s(\mathbf{x}) = 1 \times 1 + 1 \times 0 = 1$$

Next we compute the output $\hat{f}(\mathbf{x})$, which requires to compare $s(\mathbf{x}) = 1$ to the threshold $\alpha = 1.5$. Since $s(\mathbf{x}) \leq \alpha$, we find that $\hat{f}(\mathbf{x}) = 0$, cf. (2.2). The output of the neural network thus equals the

output as given by the AND function. One can easily verify that the other possible inputs, namely (0,0), (0,1) and (1,1), also result in equality between the neural network output and the output produced by the AND function.

2.2.3 Artificial neural networks with more than two layers

Let us consider another logical function. The XOR function is defined as producing as output one if and only if the two binary input variables assume a different value; otherwise, the output is zero. For example, the output corresponding to (1,0) equals 1, while (1,1) produces 0. How can the XOR be represented by a neural network with two input variables and a McCulloch-Pitts neuron? Unfortunately, whatever values are assigned to the two weights and to the threshold, the network will fail to produce the correct output for every possible input vector [89]. The XOR is a too complex function to be modeled by a network with a neuron that is as simple as the McCulloch-Pitts neuron.

Yet, the state of affairs is not as depressing as might seem. Referring to Section 2.1, ANNs were initially developed by borrowing some general principles from neurology, in particular the principle that even simple processing units can exhibit very complex behavior, provided that enough of these units are employed and that they exchange information in an appropriate way. So, the very first machine learning researchers had the idea of using multiple McCulloch-Pitts neurons, expecting that this would improve the complexity of the neural network. They found an ingenious way to incorporate more neurons: by adding an extra layer between the input layer and the output layer. Like the output layer, this extra layer consists of McCulloch-Pitts neurons. The result is an ANN with three layers, with the first layer (i.e. the input layer) containing the input variables, and both the second layer and the third layer (i.e. the output layer) containing a number of McCulloch-Pitts neurons. The operation of the McCulloch-Pitts neurons in the second layer is essentially the same as those in the output layer. The extra layer is commonly referred to as the hidden layer, and the neurons in this layer are called hidden neurons. Furthermore, connections between neurons are again maintained by weights. There is a weight between every neuron in the first layer and every neuron in the hidden layer, as well as between every neuron in the hidden layer and every neuron in the output layer. A graphical representation of an ANN with three layers is shown in Fig. 2.5.

To describe in some detail the working of this more complex network, it is convenient to extend our notation. In the case of two layers, there was only one type of weights, namely the weights connecting the input layer to the output layer. With three layers, there are two kinds of weights, i.e. weights between the input layer and the hidden layer, and weights between the hidden layer and the output layer. The weight that goes from the jth neuron in layer $k-1$ to the ith neuron in layer k is denoted by $w_{ij}^{(k)}$. For example, the weight that connects the second input neuron (which belongs to layer 1) with the first hidden neuron (belonging to layer 2) is denoted by $w_{12}^{(2)}$.

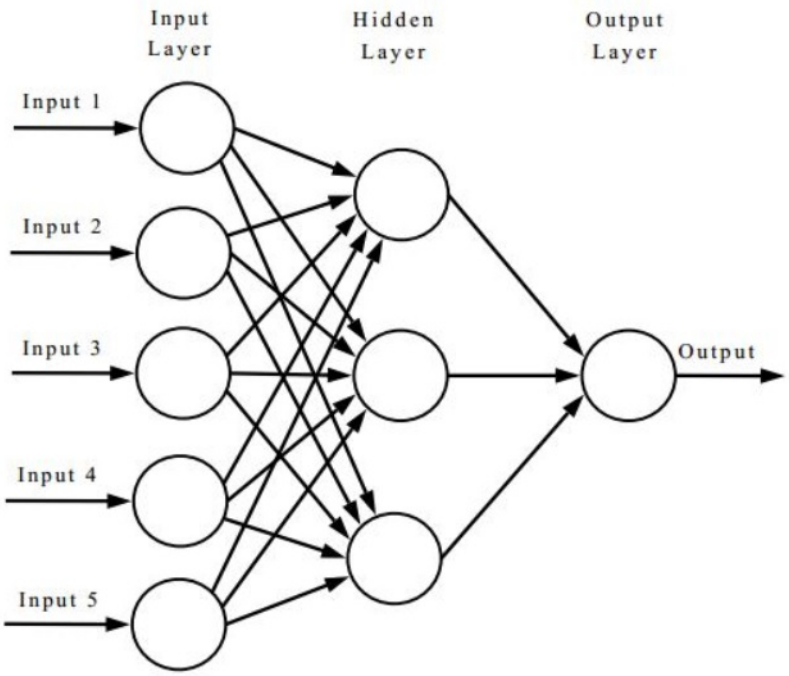

Figure 2.5: Illustration of an artificial neural network with three layers.

Each hidden neuron performs the same operation as the output neuron in the two layer case. This means that the jth hidden neuron receives the input vector from the input neurons, and then computes its output $h_j(\mathbf{x})$ as follows:

$$h_j(\mathbf{x}) = \tau_j\Big(\sum_{i=1}^{n} w_{ji}^{(2)} x_i\Big) \tag{2.5}$$

with the function τ_j defined as

$$\begin{aligned} \tau_j(z) &= 1 \qquad \text{if } z > \beta_j \\ &= 0 \qquad \text{otherwise} \end{aligned} \tag{2.6}$$

The above equations are essentially the same as (2.3)-(2.4). The only difference is that we have taken into account that each hidden neuron might have its own threshold β_j. The values $h_j(\mathbf{x})$ produced by the hidden neurons are then sent to the output neuron. The output neuron computes the final output in a similar way as for the two-layer case, cf. (2.4). There is, however, one important difference: the output neuron is now receiving the values $h_j(\mathbf{x})$ as its input values, while in the two layer case it received the values x_j as input values. The final output, as computed by the output neuron, is then given by:

$$\hat{f}(\mathbf{x}) = \tau\Big(\sum_{j=1}^{m} w_{1j}^{(3)} h_j(\mathbf{x})\Big) \tag{2.7}$$

where m denotes the number of hidden neurons, and where τ is still defined as in (2.3).

In summary, the network computes the output for a given input vector according to the following steps:

1. Each input value $x_i, i = 1,\ldots,n$, is sent to each hidden neuron.

2. Given the input values $\mathbf{x} = (x_1,\ldots,x_n)$, each hidden neuron performs the operation (2.5). This results in the values $h_1(\mathbf{x}),\ldots,h_m(\mathbf{x})$.

3. The values $h_1(\mathbf{x}),\ldots,h_m(\mathbf{x})$ are sent to the output neuron.

4. Given the values $h_1(\mathbf{x}),\ldots,h_m(\mathbf{x})$, the output neuron computes the output of the network by performing operation (2.7).

For convenience, we have described the structure of the network for the case of one output neuron. The same principles are easily extended when more output neurons are needed.

In constructing a network with three layers, there is no freedom in the choice of the number of input neurons and the number of output neurons: the number of input neurons equals the number of input variables, while the number of output neurons equals the number of output variables. Both are determined by the considered application. For example, the AND function has two input variables and one output variable, and thus the network that was constructed to model this function has two input neurons and one output neuron. The story is different for the number of hidden neurons. In principle, the user is completely free to choose the number of hidden neurons. Recalling the above considerations, we might expect that as more hidden neurons are introduced, the network is able to model more complex behavior. Does this suggest to use a very large number of hidden neurons, such that we might be pretty sure that our phenomenon will be properly modelled? No. One reason why a very complex model might not be desired, was discussed in the previous chapter: overfitting. A system with very high complexity will typically also model irregularities that are not intrinsically part of the considered phenomenon, such as measurement errors. Plenty of heuristic methods have been developed to determine an appropriate number of hidden neurons, but we will not discuss these techniques here. The interested reader is referred to, e.g., [19, 43, 59, 81, 90].

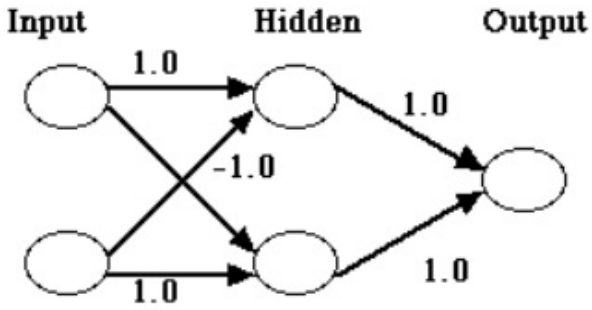

Figure 2.6: Implementing the XOR gate by a neural network consisting of McCulloch-Pitts neurons. Adapted from `http://hollich.psych.purdue.edu/CSS/Lecture1.html`.

2.2.4 Illustration of a three layer ANN: representing the XOR function

The conjecture that higher complexity can be attained by using more neurons is exemplified by modelling the XOR function. As stated above, researchers have shown that a two layer ANN of the kind we have described above, is unable to represent the XOR correctly. However, it can be described by a *three* layer ANN, as we now demonstrate.

The input layer of our three layer ANN contains two input neurons, since the XOR has two input variables, while the output layer has one output neuron, which is determined by the fact that the XOR has one output variable. The threshold α associated with the output neuron is set to $\alpha = 0$. Furthermore, we choose two hidden neurons with associated thresholds $\beta_1 = \beta_2 = 0$. There are six weights, because there is one weight from each neuron in a certain layer to each neuron in the next layer. We choose the weights as follows:

- The weight from the first input neuron to the first hidden neuron: $w_{11}^{(2)} = 1$.
- The weight from the first input neuron to the second hidden neuron: $w_{21}^{(2)} = -1$.
- The weight from the second input neuron to the first hidden neuron: $w_{12}^{(2)} = -1$.
- The weight from the second input neuron to the second hidden neuron: $w_{22}^{(2)} = 1$.
- The weight from the first hidden neuron to the output neuron: $w_{11}^{(3)} = 1$.
- The weight from the second hidden neuron to the output neuron: $w_{12}^{(3)} = 1$.

The neural network is shown in Fig. 2.6.

Let us verify its working on the input vector $\mathbf{x} = (1,0)$. Since the input values differ, the XOR returns one as output, and a correct ANN should thus produce $\hat{f}(\mathbf{x}) = 1$ as output. We follow the steps described in Section 2.2.3 to compute the network output $\hat{f}(\mathbf{x})$:

1. The input values $x_1 = 1$ and $x_2 = 0$ are sent to the hidden neurons.

2. Each hidden neuron performs the operation (2.5):

$$\begin{aligned} h_1(\mathbf{x}) &= \tau_1\Big(w_{11}^{(2)} x_1 + w_{12}^{(2)} x_2\Big) \\ &= \tau_1\Big(1 \times 1 - 1 \times 0\Big) \\ &= \tau_1(1) \\ &= 1 \\ h_2(\mathbf{x}) &= \tau_2\Big(w_{21}^{(2)} x_1 + w_{22}^{(2)} x_2\Big) \\ &= \tau_2\Big(-1 \times 1 + 1 \times 0\Big) \\ &= \tau_2(-1) \\ &= 0 \end{aligned}$$

   Notice that in going from the third line to the fourth for h_1, we set $\tau_1(1) = 1$, which follows from $1 > \beta_1$, cf. (2.6). A similar reasoning applies to $\tau_2(-1)$.

3. The values $h_1(\mathbf{x})$ and $h_2(\mathbf{x})$ are sent to the output neuron.

4. The output neuron computes the network output by applying (2.7):

$$\begin{aligned} \hat{f}(\mathbf{x}) &= \tau\Big(w_{11}^{(3)} h_1(\mathbf{x}) + w_{12}^{(3)} h_2(\mathbf{x})\Big) \\ &= \tau\Big(1 \times 1 + 1 \times 0\Big) \\ &= \tau(1) \\ &= 1 \end{aligned}$$

   where the last line holds because $1 > \alpha = 0$, and thus, according to (2.3), it follows that $\tau(1) = 1$. This demonstrates that the neural network produces the correct output if the input vector equals $(1,0)$. By performing similar calculations one can verify that the network also produces the correct output for the other input vectors, given by $(0,0),(0,1)$ and $(1,1)$.

2.3 Artificial neural networks with variable weights: the Rosenblatt perceptron

2.3.1 The need for an efficient training algorithm

One may wonder how to find suitable values for the parameters (i.e. the weights and the thresholds) of the ANNs that we have considered above. In machine learning terminology, the question is how to properly train an ANN (cf. Section 1.3.3.2). For the ANNs that modelled the AND and the XOR we boldly presented parameter values for which the ANN performs the classification correctly. But it is not clear at all how proper values should be obtained, given any

application at hand. Actually, for the AND and the XOR application we do not need a sophisticated method to identify suitable values for the parameters. Values can be found by trial and error. Two important characteristics shared by the AND and the XOR application allow to apply this brute force method. First, the number of feasible input vectors is very small. Indeed, there are only four possible input vectors. Secondly, all input vectors are known beforehand. That is, we *know* that the four possible vectors are given by $(0,0),(1,0),(0,1)$ and $(1,1)$. Since the network is only required to model four patterns, and since these patterns are known in advance, it is not such a hard task for a computer to try different values for the parameters until a configuration has been found that classifies the four patterns correctly. Given the tremendous speed of today's computers, it is very plausible that correct parameters values will be found in a reasonable amount of time.

This ad hoc procedure breaks down if the number of feasible input vectors is very large, or if not all possible input vectors are known at the moment that training is to be performed. In fact, such a situation is often encountered in practice, e.g. when the task is to relate a handwritten character to the corresponding capital letter that one finds on a keyboard. In this case the set of possible input vectors is both extremely large and not completely known in advance, since it consists of all characters that have ever been written by people by hand as well as all characters that could ever be written by humans. Trying to determine parameter values by trial and error is not very wise in this case. The task of recognizing handwritten characters is very complex. This means that many neurons will be required, because the number of neurons relates to the degree of complexity that can be handled, as has been argued above. Many neurons imply many parameters, i.e. many weights and thresholds, and randomly finding suitable values for all these parameters will be prohibitively time consuming. Indeed, randomly trying values for a large set of parameters is not unlike randomly guessing the correct numbers in a combination lock with many digits.

2.3.2 Rosenblatt's idea

Rosenblatt invented a way to circumvent the need to set the parameters to constant values. He proposed a method to initially assign random values to the parameters and then *update* the parameter values, in particular the weights, as new examples are presented to the system. The new kind of neuron that he introduced in 1957 to accomplish this task was coined the perceptron [64]. In fact, the perceptron shares a lot of characteristics with the McCulloch-Pitts neuron:

- Each value that is given as input to the perceptron is multiplied by a weight.
- The weighted values are summed and compared to a threshold.
- If this sum is larger than the threshold, the output is one; otherwise, it is zero. In other words, his perceptron also produces a binary output.

Figure 2.7: A letter t that is not so easy to identify.

The crucial difference is that the weights are allowed to change during training. Intuitively speaking, every time an example is processed by the ANN, the system will evaluate its current performance. Based on this evaluation the weights may be changed in order to improve the performance.

It is no exaggeration to say that with this idea machine learning in the truest sense was born. As outlined in Chapter 1, the main principle of machine learning is to learn from experience, where experience is coded as examples that represent instances of the phenomenon. In this sense, the neural networks that have been constructed to implement the AND and the XOR, and which rely on McCulloch-Pitts neurons, are only forerunners to machine learning. They lack any learning component, because regardless of the number of times that any certain input vector is given to the network, the same output will be produced. For example, the ANN implementing the XOR will always produce 1 if the vector $(1,0)$ is presented to the network, irrespective of other input vectors that have been processed in the meantime. This is, of course, desired, as the XOR itself always returns one as output upon being presented $(1,0)$ as input. A very different story unfolds for many other real world applications. Consider again the application of the automatic recognition of handwritten characters. The letter shown in Fig. 2.7 may be mistakenly taken for an l by an ANN that has only seen a small number of example characters before, just as a child may have difficulties in correctly identifying this letter. However, if the ANN has already been presented a very large number of handwritten characters, we require that it has become better at its job and that it now correctly returns t as output, although it identified this same letter in a different way before. Thus the same input vector may be mapped to a different output by the network, depending on the characters that has been presented in the meantime. And this is what Rosenblatt understood: an ANN can become gradually better in performing its complex task as more training examples are presented, if we allow the weights to gradually change as the examples are processed.

2.3.3 The training algorithm proposed by Rosenblatt

Proposing to change the nature of the weights, from being constant values to variables, is in itself not particularly mind-blowing. What *was* groundbreaking, was that Rosenblatt also presented an efficient algorithm to update the weights during training. Unfortunately, in developing his

training algorithm he had to give up an important extension of the originally introduced neural networks. The application of his algorithm is restricted to two layer networks.

Let us have a look at the training algorithm. Since the discussion will be restricted to networks with two layers, we can use the notation w_i again (cf. Section 2.2.1) instead of more cumbersome notations as $w_{ij}^{(2)}$ that were needed for the three layer case. For convenience, we will again restrict attention to networks with one output neuron (the description of the training algorithm is easily extended to multiple output neurons).

Algorithm 1 (Training of an ANN with a single perceptron)
The training algorithm proceeds as follows:

1. *Initially, assign random values to the weights w_i.*
2. *Consider a training example $(\mathbf{x}, y)$.*
3. *Compute the weighted sum of the inputs $s(\mathbf{x})$ as given by (2.1).*
4. *Compute $\hat{f}(\mathbf{x})$ as given by (2.2).*
5. *Denote the current value of weight w_i by w_i^{old}. This value is updated to a new value w_i^{new} as follows:*

$$w_i^{new} = w_i^{old} + \left(y - \hat{f}(\mathbf{x})\right) x_i \tag{2.8}$$

The update rule for the weights is contained in the fifth step of the algorithm, in particular equation (2.8). Since there are only four possibilities for the combined values of $\hat{f}(\mathbf{x})$ and y, it is little work to describe the update rule for each of these possibilities:

- $y = 1$ and $\hat{f}(\mathbf{x}) = 0$. In this case (2.8) is simplified to:

$$w_i^{\text{new}} = w_i^{\text{old}} + x_i$$

- $y = 0$ and $\hat{f}(\mathbf{x}) = 1$. In this case (2.8) becomes:

$$w_i^{\text{new}} = w_i^{\text{old}} - x_i$$

- $y = 1$ and $\hat{f}(\mathbf{x}) = 1$. The rule (2.8) reads as:

$$w_i^{\text{new}} = w_i^{\text{old}}$$

- $y = 0$ and $\hat{f}(\mathbf{x}) = 0$. In this case (2.8) becomes:

$$w_i^{\text{new}} = w_i^{\text{old}}$$

The last two cases, with $y = \hat{f}(\mathbf{x}) = 1$ and $y = \hat{f}(\mathbf{x}) = 0$, are easily understood. In both cases it holds that the output produced by the network equals the output of the presented example. The network thus has produced the correct output, so there is no reason to change the weights. Let us consider the case with $y = 1$ and $\hat{f}(\mathbf{x}) = 0$. To be concrete, let us assume that the involved network has two input neurons and one output neuron, and that the current training example is given by $(\mathbf{x}, y)$ with $\mathbf{x} = (1.2, 0)$ and $y = 1$. This training example is presented to the network (step 2 of the algorithm) and $s(\mathbf{x})$ is computed (step 3 of the algorithm):

$$\begin{aligned} s(\mathbf{x}) &= w_1^{\text{old}} x_1 + w_2^{\text{old}} x_2 \\ &= 1.2 w_1^{\text{old}} \end{aligned} \tag{2.9}$$

Next, step 4 dictates to compute $\hat{f}(\mathbf{x})$. This means that $s(\mathbf{x}) = 1.2 w_1^{\text{old}}$ is compared to the threshold α. We have assumed that the produced output equals zero. Therefore, according to (2.2), it necessarily holds that

$$1.2 w_1^{\text{old}} < \alpha \tag{2.10}$$

This makes clear the underlying reason for the zero output: $1.2 w_1^{\text{old}}$ is below the threshold. In other words, the first weight is actually too small to produce the correct output (remember that the true output is $y = 1$). The solution is simple: increase that weight. This is exactly what is done in step 5, since (2.8) shows that w_1^{new} is given by

$$\begin{aligned} w_1^{\text{new}} &= w_1^{\text{old}} + \left(y - \hat{f}(\mathbf{x})\right) x_1 \\ &= w_1^{\text{old}} + 1.2 \end{aligned} \tag{2.11}$$

and thus the current weight w_1^{old} is increased by an amount 1.2 to result in an updated weight w_1^{new}.

In the above illustration we have deliberately set $x_2 = 0$ for convenience. As a more complex illustration, let $x_2 = -0.5$, while maintaining the other details of the considered illustration, i.e. $y = 1$, $\hat{f}(\mathbf{x}) = 0$ and $x_1 = 1.2$. Apply step 3 of the algorithm, which consists of computing $s(\mathbf{x})$:

$$\begin{aligned} s(\mathbf{x}) &= w_1^{\text{old}} x_1 + w_2^{\text{old}} x_2 \\ &= 1.2 w_1^{\text{old}} - 0.5 w_2^{\text{old}} \end{aligned}$$

The next step is the calculation of the network output, which entails comparing the above right hand expression with the threshold α. Since, by assumption, the network produces 0 as output, it follows that

$$1.2 w_1^{\text{old}} - 0.5 w_2^{\text{old}} < \alpha \tag{2.12}$$

This expression again clarifies why the network has computed an incorrect value: $1.2w_1^{\text{old}} - 0.5w_2^{\text{old}}$ is too small. In particular, it is below the threshold to produce one as output, which is the desired value. Consequently, correct output might be produced if the value in (2.12) would be larger. In comparison to the previous illustration, where $s(\mathbf{x})$ only contained one term (given by the left hand side of (2.10)), the expression of interest now consists of two terms, namely $1.2w_1^{\text{old}}$ and $-0.5w_2^{\text{old}}$. That expression can be increased by increasing *both* terms. It is obvious that the first term can be increased by *adding* a positive value to w_1^{old}. The second term can be increased by *subtracting* a positive value from w_2^{old}, because of the negative factor -0.5. This is what a human would intuitively do. How does the training algorithm handle this case? Applying step 5 of the training algorithm, the weights are updated as follows:

$$\begin{aligned} w_1^{\text{new}} &= w_1^{\text{old}} + x_1 \\ &= w_1^{\text{old}} + 1.2 \end{aligned} \tag{2.13}$$

$$\begin{aligned} w_2^{\text{new}} &= w_2^{\text{old}} + x_2 \\ &= w_2^{\text{old}} - 0.5 \end{aligned} \tag{2.14}$$

As a matter of fact, the algorithm follows the same reasoning. The positive value 1.2 is added to w_1^{old}, while the positive value 0.5 is subtracted from w_2^{old}. Of course, the computer, which executes the algorithm, easily beats us in accuracy and speed. Knowing that modern ANNs often have millions of neurons, and thus even much more weights to be updated during training, only stresses the significance of using a training algorithm.

2.4 Further developments in the field of artificial neural networks

Sections 2.2 and 2.3 describe how artificial neural networks originated. The first networks were very limited in practical applicability by restricting to binary inputs and binary outputs, and the weights were set to constant values. Learning was not part of the job of a neural network. Then Rosenblatt came to the scene, introducing his famous learning algorithm that allows to update the values of the weights as training examples are presented. Furthermore, input variables may be either discrete or continuous.

A lot of time has passed since Rosenblatt's original research. Many research results have improved the capabilities of artificial neural networks. It is impossible to extensively describe all the important contributions that have been made in the meantime. Yet, to grasp the full power of ANNs, it is instructive to briefly describe some important extensions that have resulted from research in this popular field of computer science since the 1950s.

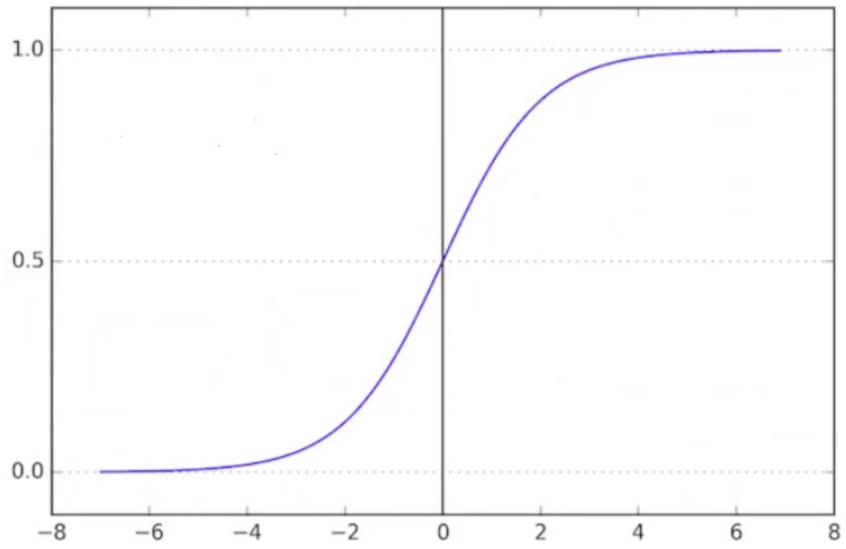

Figure 2.8: The logistic function.

2.4.1 Continuous output

The ANNs described above are limited to binary output values. The applicability of such neural networks is very limited, as they can only perform classification into two classes. Indeed, the value 0 can always be interpreted as denoting a certain class, with the value 1 referring to a second class. One application where such an ANN is useful is in classifying a tumour as either benign or malignant, where the input variables include certain measurements of cell properties [47]. However, many real world applications rely, of course, on continuous output variables. The reason why the above ANNs are forced to produce a binary output is the use of the step function τ as activation function, cf. equations (2.3)-(2.4).

By simply using other kinds of activation functions, it is possible to obtain a continuous output. A very frequently used activation function is the sigmoid, also called logistic, activation function. Using the same notation τ, the output of the network is then still given by (2.4), but where τ is defined as

$$\tau(z) = \frac{1}{1+e^{-z}} \tag{2.15}$$

with e denoting the exponential function. The logistic function is shown in Fig. 2.8. Notice that there are some similarities with the step function, which was shown in Fig. 2.3).

It may be objected that ANNs equipped with a logistic activation function are still very restrictive, in that they require the outputs of the examples to lie between 0 and 1. Thus they would not be of any use to model stock prices, house prices, temperatures, etc. However, this is merely an unimportant practical detail, as before any training is performed the outputs of the training

examples can be rescaled to lie in $[0,1]$. Given $y_1,\ldots,y_N$ as outputs of the examples, we train the ANN with the following *transformed* output values

$$z_i = \frac{y_i - \min_k y_k}{\max_k y_k - \min_k y_k} \tag{2.16}$$

As an illustration, if there are five examples, with outputs of the examples given by 5, 2, 7, 8 and 10, the first example that is actually given to the network will have as output

$$\begin{aligned} z_1 &= \frac{5-2}{10-2} \\ &= 3/8 \end{aligned}$$

which indeed lies between 0 and 1. Such transformations, which ensure that the examples are presented in a suitable form to the network, are very common, and constitute almost a field in itself. This is the research domain of preprocessing [1, 15].

2.4.2 Multiple layers

Rosenblatt's ANN is restricted to an input layer and an output layer. A hidden layer is not possible. This is very restrictive, as many real world applications are too complex to be modelled by an ANN without hidden layer. This already applies to the rather simple XOR, which cannot be modelled without hidden neurons, as we have seen. It is tempting to think that the above described training algorithm still applies if an extra layer is added. Unfortunately, matters are more complicated. To see this, let us consider, for convenience, the update rule for the weights in the binary input case. To be concrete, assume that the presented training example resulted in the situation $\hat{f}(\mathbf{x}) = 1$ while $y = 0$. The update rule then decreases the weights that are connected to each input variable that has value one, i.e. $x_i = 1$, as follows:

$$\begin{aligned} w_i^{\text{new}} &= w_i^{\text{old}} + \Big(y - \hat{f}(\mathbf{x})\Big) x_i \\ &= w_i^{\text{old}} + \Big(0-1\Big) \times 1 \\ &= w_i^{\text{old}} - 1 \end{aligned}$$

while the weights that are connected to the input variables that have value 0, i.e. $x_i = 0$, are left unchanged:

$$\begin{aligned} w_i^{\text{new}} &= w_i^{\text{old}} + \Big(y - \hat{f}(\mathbf{x})\Big) x_i \\ &= w_i^{\text{old}} + \Big(0-1\Big) \times 0 \\ &= w_i^{\text{old}} \end{aligned}$$

Remember that the reasoning behind this rule was that the output generated by the ANN is too high, and thus we reduce the weights that are responsible for this too large output.

If an extra layer is added, there are weights between the input variables and the hidden neurons, as well as between the hidden neurons and the output neurons. In this case, and without going into detail, it is not clear which weights are to be blamed for an incorrect output. Rosenblatt simply avoided this issue by restricting to two layer neural networks. It was only later that an update rule was developed that could handle this much more complex case. That algorithm, which will not be described here, is the famous backpropagation algorithm [66].

2.4.3 Approximation capabilities of ANNs

The first ANNs could barely model a function as simple as the AND. The XOR function, as intuitive and simple to understand as it may be, was beyond the power of these premature ANNs. Things can change. ANNs have been gaining capabilities that were beyond the imagination of the first machine learning researchers. By adopting ideas from neurology and mathematics, the power of ANNs was increased by equipping it with multiple layers and with many neurons, supplemented by efficient training algorithms. Can an artificial neural network actually represent *any* reasonable function? That is to say, given any sufficiently large set of training examples from a real world phenomenon, is an ANN, in principle, able to accurately model these examples, and thus the related phenomenon? This question, restated in a much more abstract form, has attracted mathematicians. It is a triumph for ANN researchers that the answer to this question is 'yes' [26, 32].

2.5 An illustrative case study

As an illustration of the application of neural networks, we describe a case study from the literature. The selected case study involves the prediction of graduation success at the United States Military Academy [41]. Each year more than 15,000 candidates apply for admission to the US Military Academy. Approximately 1,200 applications are accepted, receiving a full scholarship with an estimated value of $372,000. Recently there has been a spike in the number of first term course failures. Due to the large costs associated with an acceptance, it is important to have a model that can predict graduation of a student who applies for admission. An accurate prediction model can both inform admission decisions as well as identify students requiring remediation.

The authors collected 5100 training examples from an admissions database and from the annual Cooperative Institutional Research Program (CIRP) survey. Nine input variables were considered:

- The rank that was obtained in high school.
- A variable that indicates the quality of the high school that has been attended.
- SAT math score, where the SAT is a standardized test that is widely used for college admissions in the United States.

- SAT English score.
- Faculty assessment score.
- A variable that relates to the extra-curricular activity.
- A variable that indicates the education status of the father.
- A variable that indicates the education status of the mother.
- Time that has passed since high school graduation.

The ANN thus contains nine input neurons. The authors considered three binary output variables:

- A variable that indicates whether the student did not graduate from United States Military Academy. If graduation was not achieved, this output variable has the value one. Otherwise, it is zero.
- A variable that indicates whether the student was a late graduate. If so, the value is one; otherwise, it is zero.
- A variable that has value one if the student did graduate within the normal period, and has value zero otherwise.

The task of the ANN is to learn the relationship between the given input variables and the output variables, based on the information contained in the large set of training examples. This task is clearly a classification task (cf. Section 1.2.1), where the three possible classes can be described as 'graduate', 'late graduate' and 'non graduate'.

The set of training examples is divided into a training set, a validation set and a test set (cf. Section 1.3.3.1). The authors use a 70/15/15 rule: 70% of the examples are included in the training set, 15% in the validation set, and the remaining 15% are assigned to the test set.

The neural network contains a hidden layer, and the backpropagation algorithm is chosen as training algorithm. A graphical representation of the network is shown in Fig. 2.9. The number of hidden neurons is determined with the use of the validation set. As outlined in Section 1.3.4.2, it is not wise to determine the number of hidden neurons by simply training several ANNs, each with a different number of hidden neurons, and to select the one that performs best on the set of all examples (i.e. for which the error is lowest). Such a procedure easily results in the dramatic effect of overfitting (cf. Section 1.3.4.1). Instead, the authors follow the widely accepted procedure to determine the number of hidden neurons by training several ANNs, but where training is done solely on the *training set*, while selecting the ANN that performs best on the *validation set*. They found that 50 hidden neurons resulted in the best performance on the validation set.

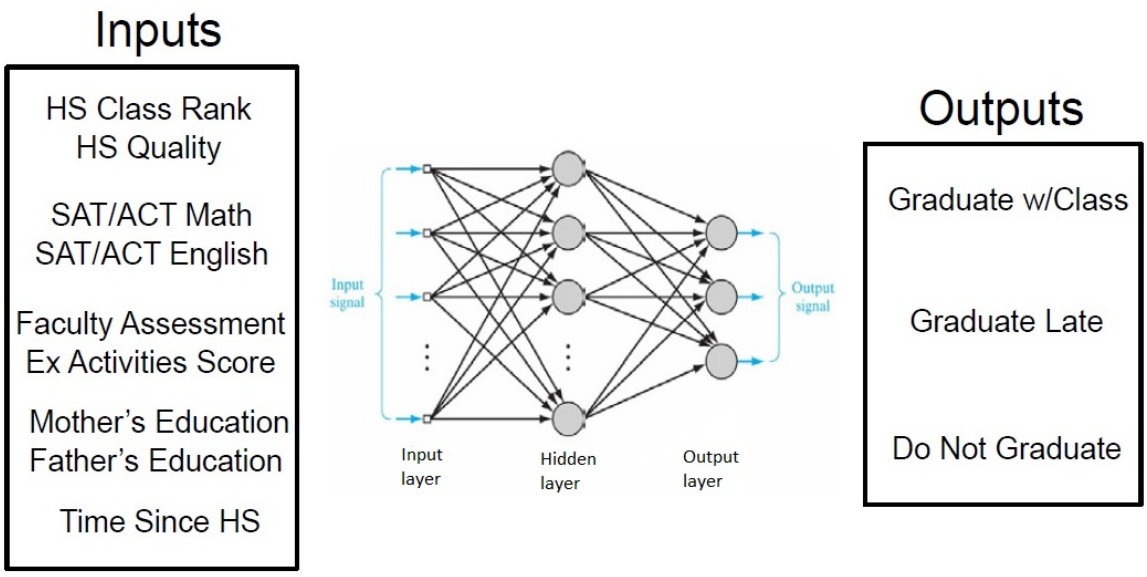

Figure 2.9: Graphical representation of the ANN described in [41].

Finally, the selected ANN with 50 hidden neurons was tested (cf. Section 1.3.5). The classification accuracy turned out to be about 95%. This means that when the ANN was given the input vectors of the examples in the test set, which are examples that had not been seen before and thus were new to the ANN, it predicted in 95% of the cases the correct class ('graduate', 'late graduate' or 'non graduate').

What is the use of having developed this ANN? The answer is essentially contained in Section 1.1: the purpose of machine learning is *not* to perform very well on a *given* set of examples, but on *previously unseen* examples. The (only) purpose of a given set of examples is to let the ANN gain insight into the studied phenomenon. After this learning phase, the ANN is expected to understand the phenomenon and to give reliable answers on new cases that derive from this phenomenon. Since the ANN for the presented case study shows very good performance on the test set, people responsible for the admissions at the US Military Academy might consider to use it in future procedures, instead of previously employed admission evaluations. To be concrete, suppose that a person wants to enter the military school. The board can then ask the potential student to provide the information that is necessary to determine the values of the nine input variables (the rank that was obtained in high school, the SAT English score, etc). This information is then given as input vector to the trained ANN, and the ANN returns an output. If the output is, e.g., (0,0,1), it means that it is predicted that the person will not graduate from the military school if admission would be allowed (since the third class corresponds to 'non graduate'). The admission board might then decide not to permit admission. Such a procedure is very objective as well as very efficient. There is no need to organize endless meetings, requiring many time and even more coffee, to decide on a matter that is to result in the same outcome, using similar

reasonings, as when an ANN would be used.

CHAPTER 3

APPLICATIONS OF SUPERVISED MACHINE LEARNING IN LAW AND CRIMINOLOGY

3.1 Use of machine learning in law and criminology

Section 1.1 mentioned several applications where machine learning has demonstrated its use. The domains of law and criminology are not an exception: without doubt, machine learning is able to generate a paradigm shift in these domains too. And yet, the implementation of machine learning in these fields has lagged other fields [61]. Somehow it seems that judges and lawyers cannot believe that a computerized system is able to be their equal, let alone to surpass them, given their knowledge of, and years of experience in, complex legal matters. The sections in this chapter show how machine learning can be useful in law and criminology by reviewing some applications that has been described in the literature.

3.2 Predicting criminal recidivism

Palocsay, Wang and Brookshire [58] developed an ANN that performs the following classification task (classification was described in Section 1.2.1). Given certain input variables that relate to characteristics of an individual who has been released from prison, predict whether this individual will eventually return. In the following subsections, we describe the data set used by the authors to train the ANN, some architectural details of the ANN that was developed to perform this task, and the results.

3.2.1 Data set

Data was obtained from the Inter-university Consortium for Political and Social Research [71]. The criminal recidivism data contains information on two sets of releases from North Carolina prisons: 9457 individuals released from 1 July, 1977 to 30 June, 1978 (to which the authors refer as the 1978 data set), and 9679 individuals released from 1 July, 1979 to 30 June, 1980 (referred to as the 1980 data set).

For each individual in the data set it is known whether he returned to a North Carolina prison. This information serves as output variable: the output variable is equal to 1 if the individual returned to a North Carolina prison, and 0 otherwise. Nine input variables are considered:

1. Whether the individual was African-American or not.
2. Whether the individual had a past alcohol problem.
3. Whether the individual had a history of using hard drugs.
4. Whether the sample sentence was for a felony or misdemeanor ('sample sentence' refers to the prison sentence from which individuals were released).
5. Whether the sample sentence was for a crime against property or not.
6. The individual's gender.
7. The number of previous incarcerations, not including the sample sentence.
8. The age at the time of release.
9. The time served for the sample sentence.

3.2.2 Some architectural details of the neural network

The ANN used by the authors is a network with multiple layers, having nine input neurons and one output neuron. The authors used logistic activation functions (cf. Section 2.4.1). However, this results in a continuous output between 0 and 1, while the task is to classify an individual as being a recidivist or not (which requires a binary output value). This is easily resolved by mapping values that are larger than 0.5 to 1, and smaller values to 0.

The given data set was divided into a training set (cf. Section 1.3.3), a validation set that is used to avoid overfitting (cf. Section 1.3.4), and a test set to evaluate the performance of the network on unseen examples (cf. Section 1.3.5).

The network was trained on the training set using the backpropagation algorithm (cf. Section 2.4.2).

	1978 Training results			1978 Test results		
Hidden nodes	Recidivist correct (%)	Non-recidivist correct (%)	Total correct (%)	Recidivist correct (%)	Non-recidivist correct (%)	Total correct (%)
39	35.79	87.42	68.31	37.97	87.86	69.20
26	36.49	88.04	68.96	38.84	87.29	69.17
20	35.26	89.28	69.29	37.97	87.65	69.07
30	38.95	86.60	68.96	40.66	85.78	68.91
33	37.02	87.01	68.51	39.88	86.25	68.91
13	36.14	88.97	69.42	38.58	86.92	68.84
29	34.56	89.90	69.42	35.10	88.89	68.78
38	38.07	87.01	68.90	40.05	85.78	68.68
28	39.47	85.57	68.51	41.96	84.59	68.65
44	37.72	86.91	68.70	40.40	85.52	68.65

Figure 3.1: Results on the 1978 data set for the ANN developed by Palocsay, Wang and Brookshire.

3.2.3 Results

The results for all experiments were recorded in terms of the percentage of recidivists correctly classified as recidivists, the percentage of non-recidivists correctly classified as non-recidivists,and the total percentage of correct classifications. The authors describe these results on the 1978 data set, both on the training set and on the test set, for different choices of the number of hidden neurons, see Fig. 3.1. As stated in Section 2.2.3, choosing the number of hidden neurons is not easy, and one simple solution is to apply the ANN on the validation set for different numbers of hidden neurons, and selecting that number for which the corresponding error is smallest.

The authors made a comparison with the results obtained by applying logistic regression, which is a popular method from statistics. They found that the ANN (slightly) outperformed logistic regression. Although in terms of predictability ANNs might be better than traditional statistical methods, ANNs have the disadvantage of lacking explanatory capability in comparison to statistical models. This issue will be studied in more detail in the next chapter.

3.3 Predicting decisions of the European Court of Human Rights

In most research projects, case law is manually collected and hand-coded, although some researchers already use computerized techniques to collect case law and automatically generate usable information from it, e.g. [40], [44] and [73]. The case study described by Medvedeva, Vols and Wieling [52] is an interesting illustration of how machine learning can be used to analyze complex textual data. The goal of the study is to create a system that automatically predicts whether any particular article of the ECHR is violated, given the facts of the case. To this end, the authors developed a support vector machine (SVM), a very popular machine learning technique for classification, which was developed by Cortes and Vapnik [14].

Article	Title	'Violation' cases	'Non-violation' cases
2	Right to life	559	161
3	Prohibition of torture	1446	595
4	Prohibition of slavery and forced labour	7	10
5	Right to liberty and security	1511	393
6	Right to a fair trial	4828	736
7	No punishment without law	35	47
8	Right to respect for private and family life	854	358
9	Freedom of thought, conscience and religion	65	31
10	Freedom of expression	394	142
11	Freedom of assembly and association	131	42
12	Right to marry	9	8
13	Right to an effective remedy	1230	170
14	Prohibition of discrimination	195	239
18	Limitation on use of restrictions on rights	7	32

Figure 3.2: Overview of the data set used by Medvedeva, Vols and Wieling.

3.3.1 Data set

The authors use the publicly available data published by the ECtHR[1]. More concretely, they use all texts of admissible cases available on the HUDOC website as of September 11, 2017. Fig. 3.2 shows an overview of the data set.

The considered texts need to be converted into training examples, which can then be fed to the machine learning system. This requires to create examples consisting of input-output pairs. The input of the examples does not include the entire text, as court decisions contain some information that is redundant in terms of the envisaged goal. For example, the dissenting/concurring opinions can be removed from the texts. Of course, also the decision itself is removed, since this aspect is to be predicted and thus constitutes the output part. The remaining text (after removing the redundant parts) is split into short consecutive sequences of words, and these items are then considered as the inputs to the system. For example, the following sentence:

> By a decision of 4 March 2003 the Chamber declared this application admissible.

is split into sequences of two words as follows:

> By a, a decision, decision of, of 4, 4 March, March 2003, 2003 the, the Chamber, Chamber declared, declared this, this application, application admissible, admissible.

[1] `https://hudoc.echr.coe.int/eng`

Any contiguous sequence of two words is called a 2-gram, while, more generally, any contiguous sequence of n words is referred to as an n-gram. The authors split the text into 1-grams, 2-grams, 3-grams, and 4-grams. Splitting into certain n-grams ensures a standard format for all possible court decisions, and having this standard format it is possible to represent any court decision as an input-output pair, where both input and output are vectors of real numbers.

3.3.2 Results

The authors apply their system to a test set, and evaluate the performance in terms of precision, recall and F-score. Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. If non-violation is to be predicted, a false positive refers to a case the model incorrectly labels as a non-violation, i.e. the court actually decided that there was a violation. A true positive refers to a case the model correctly labels as a non-violation. In case violation is to be predicted, the roles are reversed, e.g. a true positive then refers to a case the model correctly labels as a violation. Recall is the number of true positives divided by the number of true positives plus the number of false negatives. If non-violation is to be predicted, a false negative refers to a case that the model identifies as a violation, while the court decided that there was no violation. The F-score is the harmonic mean of precision and recall. The results are shown in Fig. 3.3 for each article separately. To interpret the results, it is useful to notice that if we would just randomly guess the outcome, we would be correct in about 50% of the cases. Percentages substantially higher than 50% thus indicate that the model is able to use textual information about the facts of the case to improve the prediction of the outcome (violation or non-violation).

The results, with an average performance of 0.75, show substantial variability across articles. It is possible that the differences are caused by differences in the amount of training data. The lower the amount of training data, the less the model is able to learn from the data. However, it is also plausible that cases related to certain articles are more complex than cases related to other articles, resulting in a lower predictive accuracy.

3.4 Automation of legal reasoning in the discretionary domain of family law in Australia

As a final illustration of the application of supervised machine learning to law and criminology, this section discusses the work by Stranieri, Zeleznikow, Gawler and Lewis [77]. Their work is very interesting, especially from a machine learning point of view, because the majority of applications of artificial intelligence to legal reasoning have focused on domains of law that are typically not regarded as discretionary. In contrast, the authors describe the application of machine learning to judicial decisions that are based on the highly discretionary rules of the

Art #	Class	Precision	Recall	F-score
Art 2	Non-violation	0.72	0.68	0.70
Art 2	Violation	0.70	0.74	0.72
Art 3	Non-violation	0.80	0.77	0.79
Art 3	Violation	0.78	0.81	0.80
Art 5	Non-violation	0.77	0.75	0.76
Art 5	Violation	0.76	0.77	0.77
Art 6	Non-violation	0.78	0.87	0.82
Art 6	Violation	0.85	0.76	0.80
Art 8	Non-violation	0.69	0.76	0.72
Art 8	Violation	0.73	0.66	0.69
Art 10	Non-violation	0.63	0.66	0.65
Art 10	Violation	0.64	0.61	0.63
Art 11	Non-violation	0.86	0.78	0.82
Art 11	Violation	0.80	0.88	0.8
Art 13	Non-violation	0.83	0.86	0.85
Art 13	Violation	0.85	0.83	0.84
Art 14	Non-violation	0.77	0.76	0.77
Art 14	Violation	0.77	0.77	0.77

Figure 3.3: Results of the machine learning system developed by Medvedeva, Vols and Wieling.

Family Law Act of Australia of 1975, in particular in distributing property. The Act makes explicit a number of factors that must be taken into account by a judge in altering the property interests of parties to a marriage, but the statute is silent on the relative importance of each factor. The goal of the author's work is to discover how judges weigh the different factors.

3.4.1 Some background on the Family Law Act

Taking into account section 79(1) of the Family Law Act, judges of the Family Court follow a five step process in order to arrive at a property order:

1. Ascertain the property of the parties.
2. Value all property of both parties.
3. Determine which assets will be paramount in property considerations (referred to as common pool property).
4. Determine a percentage of the property to be awarded to each party.

5. Create an order altering property interest to realise the percentage.

The system developed by the authors implements steps 3 and 4 above, i.e. the common pool determination and the prediction of a percentage split. The discretionary power of the judge is mainly exercised in step 4.

3.4.2 Common pool determination

We do not describe the details of the implementation of this task, as the most interesting machine learning task is the prediction of a percentage split, since this is the step in the above five step process where discretionary power is prevalent. It may suffice to state that the authors rely on directed graphs to determine whether or not a certain asset will be included in the pool by the Family Court. A directed graph is a set of objects that are connected together, where all the edges are directed from one object to another. A graph is intuitive to understand, as it bears many similarities with the sequential decision process by human beings. Furthermore, representation rules have been developed to efficiently store a graph in a computer and to perform certain operations on it in an automated way. Fig. 3.4 is an illustration of a directed graph that is used to determine if a given vehicle will belong to the common pool.

3.4.3 Percentage split determination

As stated above, the Act describes some factors that must be taken into account by a judge in determining the property interests of parties to a marriage, but deciding on the relative importance of these factors is left to the judge. Generally speaking, the factors relevant for a percentage split determination are past contributions of a husband relative to those of the wife, the husband's future needs relative to those of the wife, and the wealth of the marriage. The authors realised that the factors could be placed in a hierarchy, and they developed such a hierarchy in collaboration with domain experts. Actually, this provides a nice illustration of how human expertise can be combined with machine learning abilities in obtaining a superior system, refuting the often held idea (especially by people not acquainted with machine learning) that machine learning considers human experts as unnecessary or, worse, as undesirable objects. Their hierarchy contains 94 factors, part of which is shown in Fig. 3.5.

The hierarchy is used to decompose the task of predicting an outcome in 35 sub-tasks. Outputs of sub-tasks further down the hierarchy are used as inputs to sub-tasks higher in the hierarchy. In this way the overall task is converted into the simpler task of making multiple small-scale inferences, and this is done level by level in the hierarchy. The authors use two different systems to perform these inferences: the inferences represented by solid arcs in the figure are performed with the use of so-called rule sets, while the dashed arcs depict inferences performed using neural networks. We describe only the inferences performed by the neural networks.

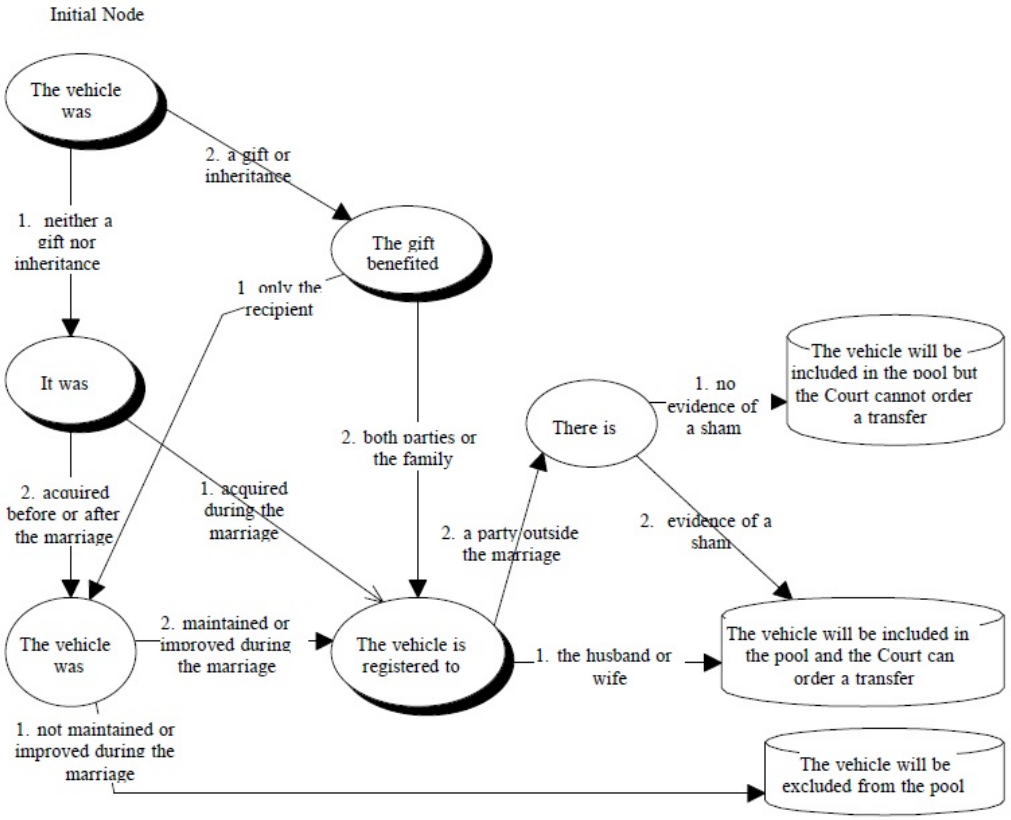

Figure 3.4: Illustration of a directed graph.

Fig. 3.6 illustrates how the ANN is used to infer a percentage split outcome. It expands the factors on the right of Fig. 3.5. The inputs to the displayed neural network are the values of each of these three factors (contributions of husband relative to wife, level of wealth, and future needs of husband relative to wife). The output is the predicted percentage split. This is the ANN that is highest in the hierarchy, and for lower levels the same procedure is employed. For example, referring to Fig. 3.5, there is also an ANN that takes as input the common pool value and that infers the level of wealth.

3.4.4 Data set

The authors rely on written judgments handed down by judicial decision makers in common place cases. These concern the vast majority of cases that come before the first instance decision maker and are never published, never appealed and constitute cases that set no precedents. They prefer to use such cases for training over landmark cases because the intention is to apply neural networks to lean how judges combine factors in actual day to day practice. Data was gathered from cases decided between 1992 and 1994, and of the common place cases concerning family law

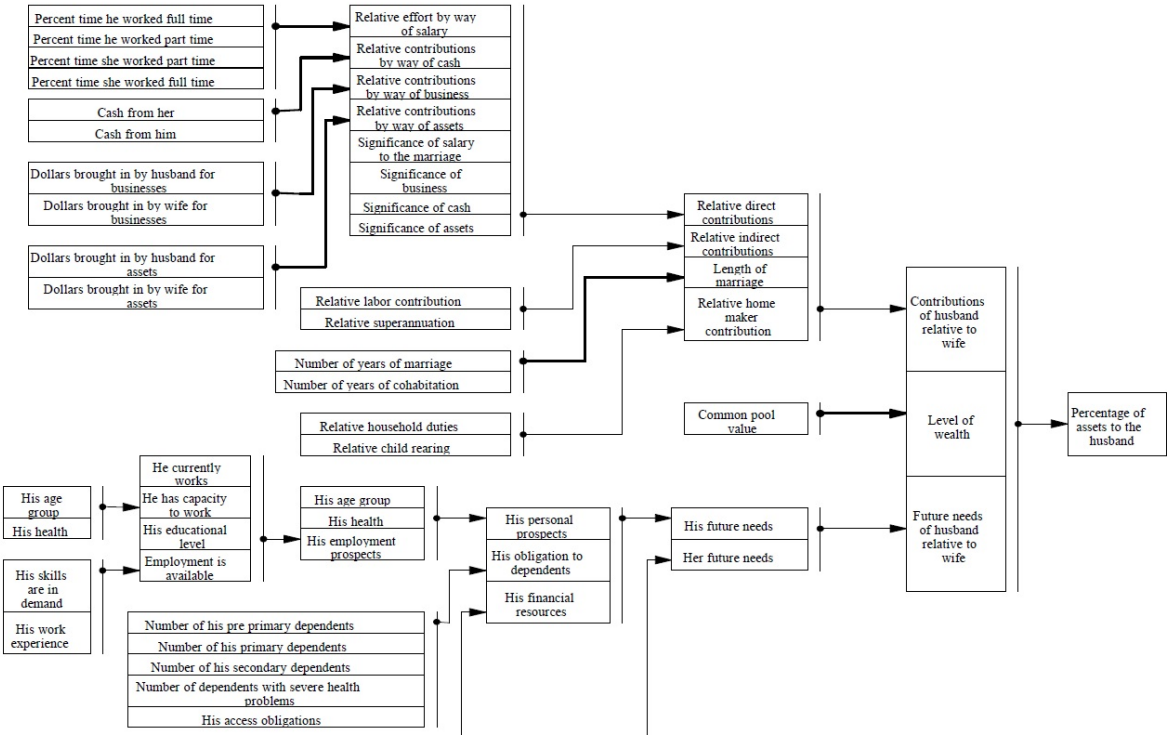

Figure 3.5: Part of the hierarchy of 94 factors that are used to determine the percentage split.

Proportion of errors of magnitude > 3	0.03
Proportion of errors of magnitude > 2	0.12
Proportion of errors of magnitude > 1	0.16
Proportion of errors of magnitude > 0.5	0.31

Table 3.1: Results of Split Up on a test set

103 cases involved property alone. Three raters extracted data from these cases by reading the text of the judgment and recording the values of the 94 variables.

3.4.5 Results

The authors evaluated their system, called Split Up, on a test set. They counted the proportion of test cases on which the error in predicting the percentage was larger than 3, larger than 2, larger than 1 and larger than 0.5. Table 3.1 presents the results.

Interestingly, the authors made also a comparison with an analysis performed by lawyers in family law. Eight specialist family law solicitors were asked to analyse three cases. These cases were devised to test diverse marriage scenarios. The results are shown in Fig. 3.7. Cases B and C indicate compatibility between Split Up predictions and those of the lawyers. Case A was more controversial. It involved a marriage where domestic duties were performed by paid staff and not

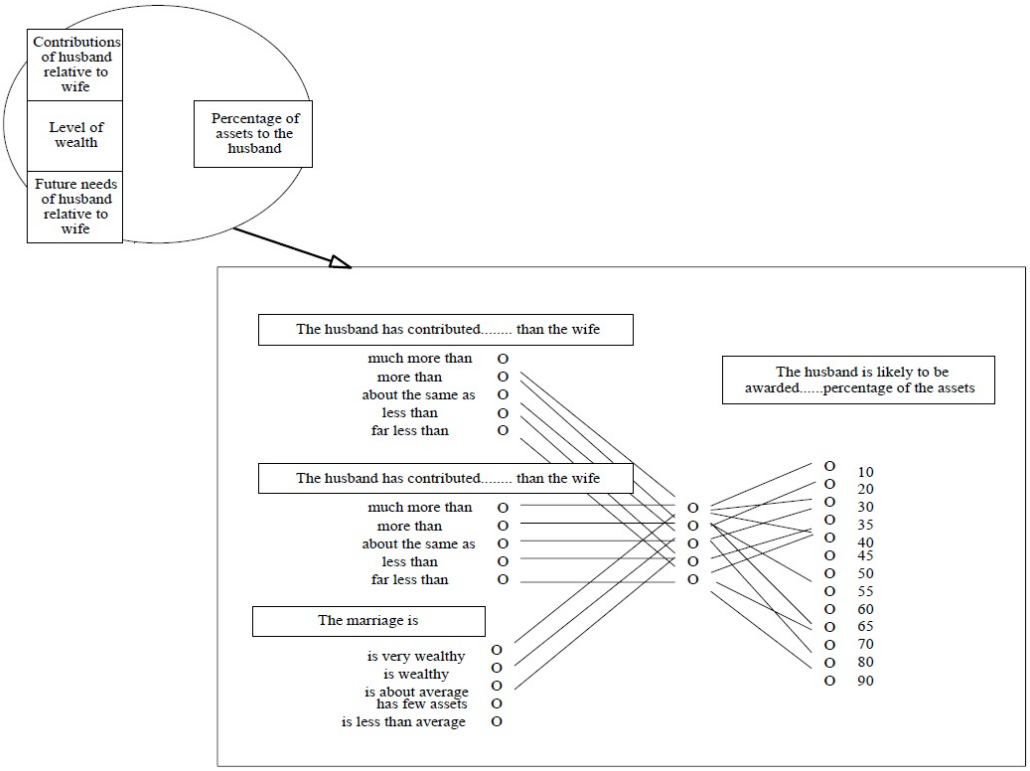

Figure 3.6: Inferring percentage split outcome with an artificial neural network.

	Case A	**Case B**	**Case C**
Split Up	55%	50%	40%
Lawyer 1	55-60%	50%	35%
Lawyer 2	55%	50%	35-40%
Lawyer 3	50-55%	50%	40%
Lawyer 4	45%	50%	50%
Lawyer 5	45-50%	50%	40%
Lawyer 6	40%	50%	35%
Lawyer 7	45-50%	50%	35%
Lawyer 8	50%	50%	40%

Figure 3.7: Percentage of assets awarded to husband by Split Up and family lawyers.

by either party to the marriage. Split Up and four lawyers interpreted the situation as one where both parties had contributed to the home in equal measure. The remaining lawyers regarded this situation as improbable and, despite evidence to the contrary, assigned the majority of the home-maker role to the spouse who had not engaged in paid employment.

CHAPTER 4

EPILOGUE

Machine learning can be a very helpful tool for practitioners in law and for judges. That is the message that is put forward in this work. Currently, machine learning is the practitioner's helper, but one day roles might be reversed. Ray Kurzweil and other artificial intelligence experts are convinced that the singularity is an inevitable future event. At that moment computers will surpass human's intelligence. Whether or not this prediction will come true is, actually, not the main concern of the legal practitioner. What he *should* be concerned about is the rapid evolution of machine learning in his field. Therefore, this is the moment to start studying it, and to use machine learning tools in his daily practice. The many surprising and outstanding results that machine learning has produced, in very diverse domains, leave no doubt about its use.

Yet, the field of machine learning is also encountering challenges, as if it has to reinvent itself. The main worry is the black box behavior of many machine learning systems (cf. Section 1.1). Black box techniques are thought to take inputs and provide outputs, but not to yield physically interpretable information to the user [51]. The process or the reasoning that produces the given output is hidden in model parameters that are extremely difficult to interpret. In this sense, machine learning methods are "oracular inference engines that render verdicts without any accompanying justification" [86].

This black box behavior might be highly undesirable depending on the intended application. At any rate, it can be considered a disadvantage, and this for at least the following reasons:

- According to a longstanding philosophical rationale (going back to at least Aristotle), experts, and thus also intelligent systems, should be able to justify their actions by marshaling knowledge of causal relationships [45].

- No single system has a zero error rate in making predictions. Thus understanding how a system arrived at its decision is critical to ascertain how a certain error occurred, and to be able to subsequently adjust the system such that the observed kind of errors will not appear again.
- For a lot of applications it holds that unravelling the underlying mechanism is intrinsically linked to the purpose of the application itself. For example, in medical applications an informed consent process can only proceed appropriately if the physicians are sufficiently knowledgeable to explain to patients how an artificial intelligence device works, which is rendered difficult by the black-box problem [70].
- In the specific context of applications in law, it is well known that a judge has to motivate its decision. This implies that his decision cannot be based on a black box tool, no matter how sophisticated this black box system is.

Although the black box character of machine learning systems had already been recognized as a severe deficiency before, at least for certain applications (e.g. [13], [16], [23], [62], [87]), it is only very recently that turmoil has reached its peak. This has led to the emergence of a new domain in the broader field of artificial intelligence, namely explainable artificial intelligence, where the ambitious goal is to construct interpretable machine learning algorithms, providing some explanation or justification for decisions obtained by black box mechanisms [2], [10], [11], [27], [29], [85].

Not only scientific researchers, but also major companies, such as Google[1] , and even governments[2], have recently declared that explaining AI systems is an important goal.

As long as the singularity has not been realized, humans and machines depend on each other for their performance. Machine learning systems are capable to supplement human expertise by their speed and their seemingly intelligent reasoning. Conversely, humans are needed to incorporate further characteristics of high level intelligence into systems, such as explainability.

[1] https://www.bbc.com/news/technology-50506431

[2] The DARPA division of the Department of Defense is spending $2 billion on its explainable artificial intelligence program, cf. https://www.darpa.mil/program/explainable-artificial-intelligence

Bibliography

[1] S. ALEXANDROPOULOS, S. KOTSIANIS, AND M. VRAHATIS, *Data preprocessing in predictive data mining*, The Knowledge Engineering Review, 34 (2019).

[2] S. ANJOMSHOAE, A. NAJJAR, D. CALVARESI, AND K. FRAMLING, *Explainable agents and robots: results from a systematic literature review*, in Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, 2019, pp. 1078–1088.

[3] H. ARUMAWADU, R. RATHNAYAKA, AND S. ILLANGARATHNE, *Mining profitability of telecommunication customers using K-means clustering*, Journal of Data Analysis and Information Processing, 3 (2015), pp. 63–71.

[4] R. ASHLEY AND D. VAUGHAN, *Measuring measurement error in economic time series* , Journal of Business & Economic Statistics, 4 (1986), pp. 95–103.

[5] A. ASKI AND N. SOURATI, *Proposed efficient algorithm to filter spam using machine learning techniques*, Pacific Science Review A: Natural Science and Engineering, 18 (2016), pp. 145–149.

[6] H. BAOMAR AND P. BENTLEY, *An intelligent autopilot system that learns piloting skills from human pilots by imitation*, in Proceedings of the 2016 International Conference on Unmanned Aircraft Systems, 2016.

[7] M. BATAINEH AND T. MARLER, *Neural network for regression problems with reduced training sets*, Neural Networks, 95 (2017), pp. 1–9.

[8] I. BILBAO AND J. BILBAO, *Overfitting problem and the over-training in the era of data: particularly for artificial neural networks*, in Proceedings of the Eighth International Conference on Intelligent Computing and Information Systems, IEEE, 2017.

[9] S. BONABI, H. ASGHARIAN, R. BAKHTIARI, S. SAFARI, AND M. AHMADABADI, *FPGA implementation of a cortical network based on the Hodgkin-Huxley neuron model*, in International Conference on Neural Information Processing, Springer, 2012.

[10] R. BYRNE, *Counterfactuals in explainable artificial intelligence (xai): evidence from human reasoning*, in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, 2019, pp. 6276–6282.

[11] D. CARVALHO, E. PEREIRA, AND J. CARDOSO, *Machine learning interpretability: a survey on methods and metrics*, Electronics, 8 (2019), pp. 832–865.

[12] M. CHIANG AND B. MIRKIN, *Intelligent choice of the number of clusters in K-means clustering: an experimental study with different cluster spreads*, Journal of Classification, 27 (2010), pp. 3–40.

[13] S. CIMPOERU, *Neural networks and their application in credit risk assessment. Evidence from the Romanian market*, Technological and Economic Development of Economy, 17 (2011), pp. 519–534.

[14] C. CORTES AND V. VAPNIK, *Support-vector networks*, Machine Learning, 20 (1995), pp. 273–297.

[15] S. CRONE, S. LESSMANN, AND R. STAHLBOCK, *The impact of preprocessing on data mining: An evaluationof classifier sensitivity in direct marketing*, European Journal of Operational Research, 173 (2006), pp. 871–800.

[16] J. CRUZ AND D. WISHART, *Applications of machine learning in cancer prediction and prognosis*, Cancer Informatics, 2 (2006), pp. 59–77.

[17] W. DE MULDER, B. RENGS, G. MOLENBERGHS, T. FENT, AND G. VERBEKE, *A comparison of some simple and complex surrogate models: make everything as simple as possible?*, in Proceedings of the Eighth International Conference on Advances in System Simulation, IARIA, 2016.

[18] R. DESCARTES AND J. VEITCH, *meditations on first philosoph*, Rough Draft Printing, 2014.

[19] C. DOUKIM, J. DARGHAM, AND A. CHEKIMA, *Finding the number of hidden neurons for an mlp neural network using coarse to fine search technique*, in Proceedings of the 10th International Conference on Information Science, Signal Processing and their Applications, IEEE, 2010.

[20] D. DUBOIS, P. HÁJEK, AND H. PRADE, *Knowledge-driven versus data-driven logics*, Journal of Logic, Language, and Information, 9 (2000), pp. 65–89.

[21] N. DUNGEY, *Thomas Hobbes's materialism, language, and the possibility of politics*, The Review of Politics, 70 (2008), pp. 190–220.

[22] M. FAZI, *Can a machine think (anything new)? Automation beyond simulation*, AI & Society, 121 (2018), pp. 1–12.

[23] A. FREITAS, D. WIESER, AND R. APWEILER, *On the importance of comprehensible classification models for protein function prediction*, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 7 (2010), pp. 172–182.

[24] K. FUKUDA, *Forecasting economic time series with measurement error* , Applied Economics Letters, 12 (2005), pp. 923–927.

[25] C. GROSAN AND A. ABRAHAM, eds., *Intelligent systems: a modern approach*, Springer, 2011.

[26] N. GULIYEV AND V. ISMAILOV, *A single hidden layer feedforward network with only one neuron in the hidden layer can approximate any univariate function*, Neural Computation, 28 (2016), pp. 1289–1304.

[27] H. HAGRAS, *Towards human-understandable, explainable AI*, Computer, 51 (2018), pp. 28–36.

[28] S. HERCULANO-HOUZEL, *The human brain in numbers: a linearly scaled-up primate brain* , Frontiers in Human Neuroscience, 3 (2009).

[29] R. HOFFMAN, G. KLEIN, AND S. MUELLER, *Explaining explanation for 'explainable AI'*, in Proceedings of the Human Factors and Ergonomics Society 2018 Annual Meeting, 2018.

[30] Y. HON, *A quasi-radial basis functions method for American options pricing*, Computers & Mathematics with Applications, 43 (2002), pp. 513–524.

[31] W. HONG, *A hybrid support vector machine regression for exchange rate prediction*, International Journal of Information and Management Sciences, 17 (2006), pp. 19–32.

[32] K. HORNIK, M. STINCHCOMBE, AND H. WHITE, *Multilayer feedforward networks are universal approximators*, Neural Networks, 2 (1989), pp. 359–366.

[33] A. JAIN, *Data clustering: 50 years beyond K-means*, Pattern Recognition Letters, 31 (2010), pp. 651–666.

[34] A. JOG, A. CARASS, S. ROY, D. PHAM, AND J. PRINCE, *Random forest regression for magnetic resonance image synthesis*, Medical Image Analysis, 35 (2017), pp. 475–488.

[35] S. KONSTANTINIDIS, P. KARAMPIPERIS, AND M. SICILIA, *Enhancing the Levenberg-Marquardt method in neural network training using the direct computation of the error Cost function Hessian*, in Proceedings of the 16th International Conference on Engineering Applications of Neural Networks, 2015.

[36] K. KOUROU, T. EXARCHOS, K. EXARCHOS, M. KARAMOUZIS, AND D. FOTIADIS, *Machine learning applications in cancer prognosis and prediction*, Computational and Structural Biotechnology Journal, 13 (2015), pp. 8–17.

[37] R. KURZWEIL, ed., *The age of spiritual machines: when computers exceed human intelligence*, Penguin Books, 2000.

[38] ——, *The singularity is near: when humans transcend biology*, Penguin Books, 2006.

[39] B. LANTZ, ed., *Machine learning with R*, Packt Publishing, 2013.

[40] D. LAW, *The global language of human rights: a computational linguistic analysis*, Law & Ethics of Human Rights, 12 (2018), pp. 111–150.

[41] G. LESINSKI, S. CORNS, AND C. DAGLI, *Application of an artificial neural network to predict graduation success at the United States Military Academy*, Procedia Computer Science, 95 (2016), pp. 375–382.

[42] H. LI, X. DENG, D. KIM, AND E. SMITH, *Modeling maximum daily temperature using a varying coefficient regression model*, Water Resources Research, 50 (2014), pp. 3073–3087.

[43] X. LIU, N. LAWRENCE, AND M. RATTRAY, *Probe-level measurement error improves accuracy in detecting differential gene expression*, Bioinformatics, 22 (2006), pp. 2107–2113.

[44] M. LIVERMORE, A. RIDDELL, AND D. ROCKMORE, *The Supreme Court and the judicial genre*, Arizona Law Review, 59 (2017), pp. 837–901.

[45] A. LONDON, *Artificial intelligence and black-box medical decisions: accuracy versus explainability*, Hastings Center Report, 49 (2019), pp. 15–21.

[46] Z. MAJDISOVA AND V. SKALA, *Radial basis function approximations: comparison and applications*, Applied Mathematical Modelling, 51 (2017), pp. 728–743.

[47] S. MANDAL AND I. BANERJEE, *Cancer classification using neural network*, International Journal of Emerging Engineering Research and Technology, 3 (2015), pp. 172–178.

[48] S. MARSLAND, ed., *Machine learning: an algorithmic perspective*, Chapman and Hall/CRC, 2009.

[49] A. MASZCZYK, R. ROCZNIOK, Z. WAŚKIEWICZ, M. CZUBA, K. MIKOLAJEC, A. ZAJAC, AND A. STANULA, *Application of regression and neural models to predict competitive swimming performance*, Perceptual and Motor Skills, 114 (2012), pp. 610–626.

[50] W. S. MCCULLOCH AND W. H. PITTS, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics, 5 (1943), pp. 115–133.

[51] A. MCGOVERN, R. LAGERQUIST, D. GAGNE II, G. JERGENSEN, K. ELMORE, C. HOMEYER, AND T. SMITH, *Making the black box more transparent: understanding the physical implications of machine learning*, Bulletin of the American Meteorological Society, 100 (2019), pp. 2175–2199.

[52] M. MEDVEDEVA, M. VOLS, AND M. WIELING, *Using machine learning to predict decisions of the European Court of Human Rights*, Artificial Intelligence and Law, (2019), pp. 1–30.

[53] W. MICHAELIS, H. HAARS, AND R. AUGUSTIN, *A new precise determination of Newton's gravitational constant*, Metrologia, 32 (1995).

[54] V. MILYUKOV, J. LUO, C. TAO, AND A. MIRONOV, *Status of the experiments on measurement of the Newtonian gravitational constant*, Gravitation and Cosmology, 14 (2008), pp. 368–375.

[55] J. MOORE, L. STINSON, AND E. WELNIAK, *Income measurement error in surveys: A review*, Journal of Official Statistics, 16 (2000), pp. 331–361.

[56] I. MORENO, A. GUTIÉRREZ, C. RUBIO, A. GONZÁLEZ, D. GONZALEZ-WELLER, N. BENCHARKI, A. HARDISSON, AND C. REVERT, *Classification of Spanish red wines using artificial neural networks with enological parameters and mineral content*, American Journal of Enology and Viticulture, 69 (2018), pp. 167–175.

[57] H. OEI AND P. POLAK, *Intelligent speed adaptation (ISA) and road safety*, IATSS Research, 26 (2002), pp. 45–51.

[58] S. PALOCSAY, P. WANG, AND R. BROOKSHIRE, *Predicting criminal recidivism using neural networks*, Socio-Economic Planning Sciences, 34 (2000), pp. 271–284.

[59] F. PANCHAL AND M. PANCHAL, *Review on methods of selecting number of hidden nodes in artificial neural network*, International Journal of Computer Science and Mobile Computing, 3 (2014), pp. 455–464.

[60] J. PARK AND I. SANDBERG, *Universal approximation using radial-basis-function networks*, Neural Computation, 3 (1991), pp. 246–257.

[61] J. PHILLIPS, *Integrating machine learning in law: a precis of best practices for initial law firm adoption*, The Journal of Business, Entrepeneurship & the Law, 11 (2018), pp. 321–328.

[62] R. PRADHAN, K. PATHAK, AND V. SINGH, *Application of neural network in prediction of financial viability*, International Journal of Soft Computing and Engineering, 1 (2011), pp. 41–45.

[63] D. ROCKE AND B. DURBIN, *A model for measurement error for gene expression arrays*, Journal of Computational Biology, 8 (2001), pp. 557–569.

[64] F. ROSENBLATT, *The perceptron: a perceiving and recognizing automaton*, Report (Cornell Aeronautical Laboratory), (1957).

[65] C. ROTHLEITNER AND O. FRANCIS, *Measuring the Newtonian constant of gravitation with a differential free-fall gradiometer: a feasibility study*, The Review of Scientific Instruments, 85 (2014).

[66] D. RUMELHART, J. MCCLELLAND, AND P. R. GROUP, *Learning internal representations by error propagation*, in Parallel Distributed Processing, E. Rumelhart, ed., MIT Press, 1986, ch. 8, pp. 318–362.

[67] A. SARIP, M. HAFEZ, AND M. DAUD, *Application of fuzzy regression model for real estate price prediction*, Malaysian Journal of Computer Science, 29 (2016), pp. 15–27.

[68] W. SARIS AND M. REVILLA, *Correction for measurement errors in survey research: necessary and possible*, Social Indicators Research, 127 (2016), pp. 1005–1020.

[69] A. SAYGIN, I. CICEKLI, AND V. AKMAN, *Turing test: 50 years later*, Minds and Machines, 10 (2000), pp. 463—-518.

[70] D. SCHIFF AND J. BORENSTEIN, *How should clinicians communicate with patients about the roles of artificially intelligent team members?*, AMA Journal of Ethics, 21 (2019), pp. E138–E145.

[71] P. SCHMIDT AND A. WITTE, *Predicting recidivism in North Carolina, 1978 and 1980*, ICPSR Inter-university Consortium for Political and Social Research, (1984).

[72] S. SHALEV-SHWARTZ AND S. BEN-DAVID, eds., *Understanding machine learning: from theory to algorithms*, Cambridge University Press, 2014.

[73] O. SHULAYEVA, A. SIDDHARTHAN, AND A. WYNER, *Recognizing cited facts and principles in legal judgements*, Artificial Intelligence and Law, 25 (2017), pp. 107–126.

[74] P. SINČAK, P. HARTONO, M. VIRČIKOVÁ, J. VAŠCÁK, AND R. JAKŠA, eds., *Emergent trends in robotics and intelligent systems*, Springer, 2015.

[75] T. SMADI, H. AL ISSA, E. TRAD, AND K. SMADI, *Artificial Intelligence for Speech Recognition Based on Neural Networks*, Journal of Signal and Information Processing, 6 (2005), pp. 66–72.

[76] D. STEINLEY AND M. BRUSCO, *Choosing the number of clusters in K-means clustering*, Psychological Methods, 16 (2011), pp. 285–297.

[77] A. STRANIERI, J. ZELEZNIKOW, M. GAWLER, AND B. LEWIS, *A hybrid rule-neural approach for the automation of legal reasoning in the discretionary domain of family law in Australia*, Artificial Intelligence and Law, 7 (1999), pp. 153–183.

[78] K. STRIMMER, *Modeling gene expression measurement error: a quasi-likelihood approach*, BMC Bioinformatics, 4 (2003).

[79] C. SUGAR AND G. JAMES, *finding the number of clusters in a dataset: an information-theoretic approach*, Journal of the American Statistical Association, 98 (2003), pp. 750–763.

[80] S. SUJENDRAN AND P. VANITHA, *Smart lawn mower for grass trimming*, International Journal of Science and Research, 3 (2014), pp. 299–303.

[81] E. TEOH, K. TAN, AND C. XIANG, *Estimating the number of hidden neurons in a feedforward network using the singular value decomposition*, IEEE Transactions on Neural Networks, 17 (2006), pp. 1623–1629.

[82] R. TIBSHIRANI, G. WALTHER, AND T. HASTIE, *Estimating the number of clusters in a data set via the gap statistic*, Journal of the Royal Statistical Society Series B (Statistical Methodology), 63 (2001), pp. 411–423.

[83] A. TURING, *Computing machinery and intelligence*, Mind, 59 (1950), pp. 433–460.

[84] E. WALTERS, *Data-Driven law: data analytics and the new legal services*, Auerbach Publications, 2018.

[85] D. WANG, Q. YANG, A. ABDUL, AND B. LIM, *Designing theory-driven user-centric explainable ai*, in Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. 1–15.

[86] D. WATSON, J. KRUTZINNA, I. BRUCE, C. GRIFFITHS, I. MCINNES, M. BARNES, AND L. FLORIDI, *Clinical applications of machine learning algorithms: beyond the black box*, BMJ, 364 (2019), p. I886.

[87] G. WECKMAN, D. MILLIE, C. GANDURI, M. RANGWALA, W. YOUNG, M. RINDER, AND G. FAHNENSTIEL, *Knowledge extraction from the neural 'black box' in ecological monitoring*, Journal of Industrial and Systems Engineering, 3 (2009), pp. 38–55.

[88] N. YADAV, A. YADAV, AND M. KUMA, eds., *An introduction to neural network methods for differential equations*, Springer, 2015.

[89] N. YE, ed., *Data mining: theories, algorithms, and examples*, CRC Press, 2013.

[90] H. YUAN, F. XIONG, AND X. HUAI, *A method for estimating the number of hidden neurons in feed-forward neural networks based on information entropy*, Computers and Electronics in Agriculture, 40 (2003), pp. 57–64.

[91] I. ZAFAR, G. TZANIDOU, R. BURTON, N. PATEL, AND L. ARAUJO, eds., *Hands-On convolutional neural networks with TensorFlow: solve computer vision problems with modeling in TensorFlow and Python*, Packt Publishing, 2018.

[92] S. ZHOU, Z. XU, AND F. LIU, *Method for determining the optimal number of clusters based on agglomerative hierarchical clustering*, IEEE Transactions on Neural Networks and Learning Systems, 28 (2017), pp. 3007–3017.