Risk bounds for deep learning

Bos, J.M.

Chapter 1

Introduction

Deep learning is, broadly speaking, the training of artificial neural networks on data for prediction tasks. It became popular in the 2010s after achieving hugely improved performance on benchmark datasets used in machine-learning competitions [60, 80, 51, 126]. This was the start of a revolution. Within a few years, deep neural networks achieved (super)human level performance in visual object recognition and speech recognition tasks [126, 30, 58, 3]. By now, deep learning is also successfully used in various other applications.

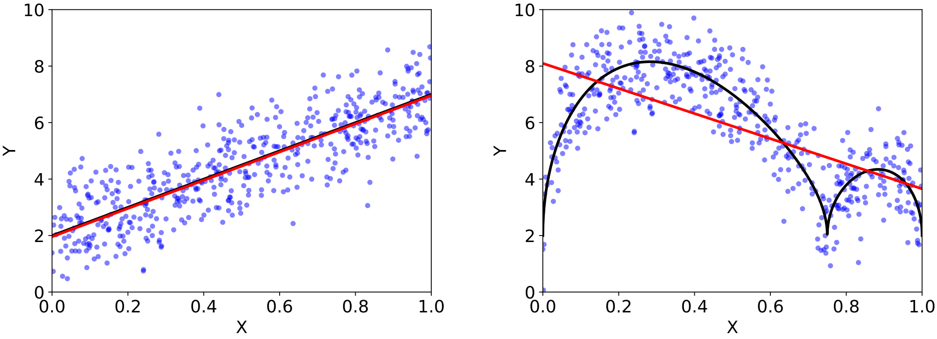
In this thesis, we approach deep learning from the statistical viewpoint: what can be said about the expected error if we view deep learning as a statistical estimation method. This perspective connects machine learning and statistics.

The introduction is structured as follows. The statistical background is introduced in Section 1.1. Section 1.2 provides an overview of deep learning. Sections 1.3, 1.4 and 1.5 briefly introduce the subjects of each of the later chapters.

1.1 Statistical background

Deep learning gained momentum by providing state-of-the-art procedures for supervised learning tasks, where one observes data-pairs $(\mathbf{X}_i, \mathbf{Y}_i)$, with $\mathbf{X}_i$ an input vector and $\mathbf{Y}_i$ the corresponding output. This setting is called supervised because the output $\mathbf{Y}_i$ is already provided (as if by a teacher), contrary to unsupervised learning where the algorithm is fed with unlabeled data. Regression and classification are the two main sub-classes of supervised learning. In classification, the goal is to predict the class a new data point belongs to. Examples include spam detection and image recognition. For instance, in the latter case, the data-pairs could consist of images of different types of animals (the input) and their corresponding labels 'dog', 'cat', etc (the output). For a new image, the neural network must predict which category this image belongs to.

In regression, the output is a real-valued response. For example, predicting income based on years of education. Various (nonparametric) methods have been developed for regression problems. Many of them are also theoretically well-understood. First, we will give a concise introduction of nonparametric statistics, mostly restricting ourselves to regression. Classification will be treated in Section 1.3. After introducing nonparametric methods, we will describe the decision theoretic concepts of loss, risk, and convergence rates. This is followed by sections on empirical risk minimization and the challenges arising in high-dimensional input-spaces.



(a) Linear

(b) Non-linear

Figure 1.1: Plots of the samples (blue), linear least squares estimator (red) and the true regression function (black) of a linear and a non-linear model.

1.1.1 Nonparametric statistics

In statistics, recurring questions are: what does one already know about the distribution of the data or what can one reasonably assume about it? This is of particular importance for complex statistical models. Below, we illustrate this using the nonparametric regression model, as this is a widely accepted framework to study supervised learning.

Example 1.1.1 (Regression)**.** In the (multivariate) regression model we observe n random pairs $(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \ldots, (\mathbf{X}_n, Y_n)$ satisfying

$$Y_i = f_0(\mathbf{X}_i) + \epsilon_i.$$

Here the ϵ_i are noise variables and f_0 is an unknown deterministic function. Models of this form are also called signal plus noise models. The regression problem is to infer f_0 from the data. It is common to assume that the noise variables ϵ_i are independent and this assumption will be used throughout this chapter.

The available prior information plays a crucial role. For instance, suspecting that the regression function f_0 is linear, all that remains is to estimate the parameters of the linear function. In Figure 1.1a, data from a linear regression problem are plotted in blue and the parametric linear least squares estimator is displayed in red. Linear regression is an example of a parametric model. Parametric models are statistical models that depend only on a finite number of parameters. See for example [152, 119] for an introduction and overview of parametric methods. However, often far less is known about the underlying model. For instance, it is clear that the (blue) data-sample in Figure 1.1b was not generated from a linear regression function. The estimate of the (red) linear least squares estimator does not reflect the data. It is, however, not immediately clear which parametric family of functions would be a better candidate. In cases like this, a method is required that assumes less about the structure of the data. In nonparametric statistics, problems are studied that do not satisfy a parametric model. Instead, one considers problems that are too big to be parametrized by a finite number of parameters. For instance, in the regression problem, Example 1.1.1, we can assume that the function f_0 is in the class of all continuous functions. This class is so large that it is impossible to represent it by a finite basis expansion.

There exist many different nonparametric regression estimation methods. Here we provide two examples: the Nadaraya-Watson estimator, Example 1.1.2, and the Fourier-series estimator, Example 1.1.3.

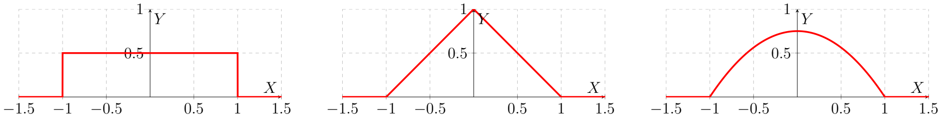
Example 1.1.2 (Nadaraya-Watson estimator)**.** A kernel in d-dimensions is a function $K : \mathbb{R}^d \to \mathbb{R}$ such that $\int_{\mathbb{R}^d} K(\mathbf{u})\, d\mathbf{u} = 1$. Some standard kernels in one dimension are displayed in Figure 1.2.

The Nadaraya-Watson estimator f_{NW} with kernel K and bandwidth $h > 0$ is given by

$$\widehat{f}_{NW}(\mathbf{x}) := \sum_{i=1}^{n} Y_i \frac{K\Big(\frac{1}{h}(\mathbf{X}_i - \mathbf{x})\Big)}{\sum_{j=1}^{n} K\Big(\frac{1}{h}(\mathbf{X}_j - \mathbf{x})\Big)},$$

when $\sum_{j=1}^{n} K\Big(\frac{\mathbf{X}_j - \mathbf{x}}{h}\Big) \neq 0$ and $f_{NW}(\mathbf{x}) = 0$ otherwise.

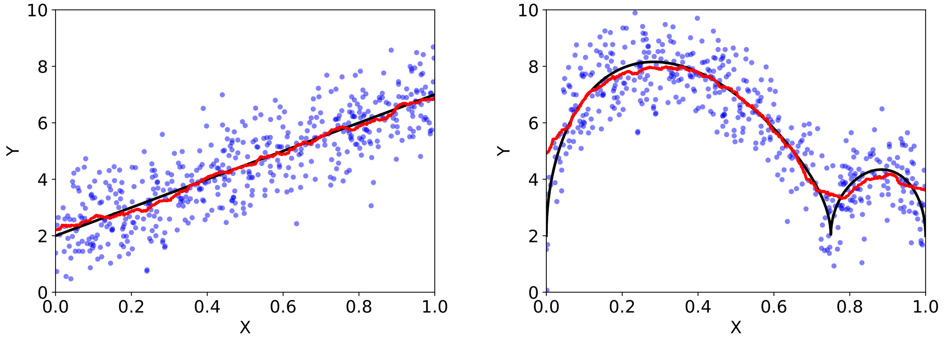
The Nadaraya-Watson estimator was first proposed in 1964 by Nadaraya [100] and Watson [154]. The estimate in a point $\mathbf{x}$ is determined by a weighted average of the output variables Y_i. The weight assigned to each sample is determined by the chosen kernel K and the bandwidth h. The role of the bandwidth h in tuning the



(a) Rectangular kernel (b) Triangular kernel (c) Epanechnikov kernel

Figure 1.2: Three examples of standard kernels in one dimension.

estimator will be discussed in Section 1.1.3. In Figure 1.3 the (red) Nadaraya-Watson estimator with the rectangular kernel is shown on the linear and non-linear regression dataset from before. Compared to the linear least squares estimator in Figure 1.1, this estimator provides a reasonable estimate for both problems.



(a) Linear (b) Non-linear

Figure 1.3: Plots of the samples (blue), Nadaraya-Watson estimator (red) and the true regression function (black) of a linear and a non-linear model.

Example 1.1.3 (Fourier-series estimator)**.** Consider the problem of estimating a function in the function space $L^2[0,1]$: the space of square integrable functions on the interval $[0,1]$. The Fourier basis in $L^2[0,1]$ is given by

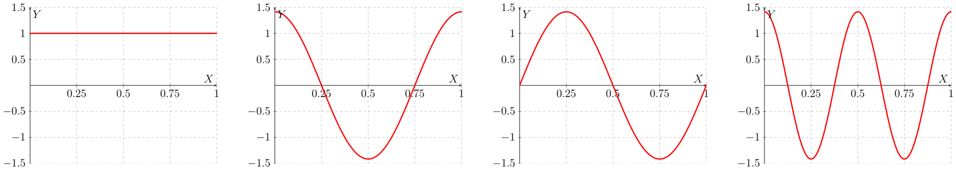
$$\begin{aligned}\psi_1(x) &:= 1,\\ \psi_{2k}(x) &:= \sqrt{2}\cos(2k\pi x),\\ \psi_{2k+1}(x) &:= \sqrt{2}\sin(2k\pi x),\end{aligned}$$

for $k = 1, 2, \ldots$. In other words, for every function $f \in L^2[0,1]$, there exist coefficients $(c_j)_{j=1}^{\infty}$ such that f can be written as $f(x) = \sum_{j=1}^{\infty} c_j \psi_j$. The first four Fourier basis functions are plotted in Figure 1.4.

The Fourier-series estimator of level N and with threshold τ is given by

$$\widehat{f}_F(x) := \sum_{j=1}^{N} \widehat{c}_j \psi_j(x) \mathbb{1}\big(|\widehat{c}_j| \geq \tau\big),$$

where $\widehat{c}_j := n^{-1} \sum_{i=1}^{n} Y_i \psi_j(X_i)$ is the empirical Fourier coefficient. Here $\mathbb{1}(\cdot)$ denotes the indicator function, returning one when $|\widehat{c}_j| \geq \tau$ and zero otherwise.



(a) ψ_1 (b) ψ_2 (c) ψ_3 (d) ψ_4

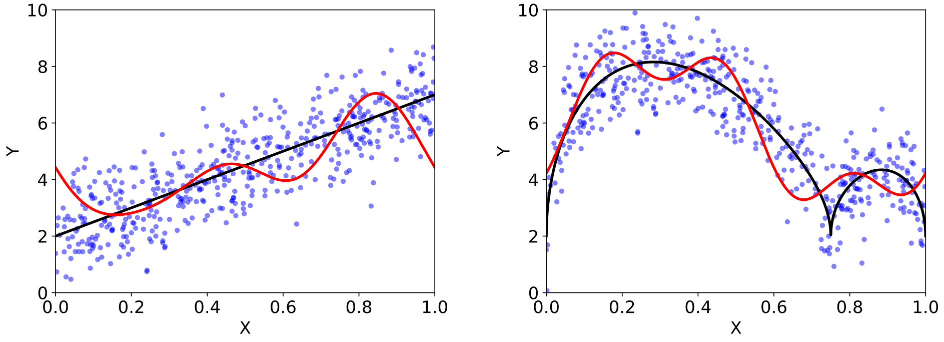
Figure 1.4: The first four Fourier basis functions

The Fourier-series estimator is an example of a projection estimator (also known as orthogonal series estimator), [153, 146, 44], as it projects on the space spanned by the first N basis elements and then estimates the coefficients of these N elements. The choice of the (orthogonal) basis determines how one can interpret an orthogonal series estimator. In the case of the Fourier-series estimator, the basis functions represent frequencies. In Figure 1.5 the (red) Fourier-series estimator is shown on the linear and non-linear regression dataset from before. The waveform of the sine and cosine functions used in the Fourier-basis can be seen back in the estimated function.

For a further introduction and overview of nonparametric estimation methods see for example [153, 146, 54].

1.1.2 Loss, risk and minimax rates

An important aspect of statistical estimation theory is to provide a quantification of how 'good' an estimator is. Consider for instance the regression problem, Example 1.1.1; Given an estimator $\widehat{f}$, how far away is it from the true regression function f_0? The first step is to choose a loss function for measuring the estimation error.



(a) Linear

(b) Non-linear

Figure 1.5: Plots of the samples (blue), Fourier-series estimator (red) and the true regression function (black) of a linear and a non-linear model.

Definition 1.1.4 (Loss-function)**.** Let $\mathcal{G}$ represent the class containing the quantity to be estimated. Denote by $\mathcal{F}$ the class in which the estimator takes its values. A loss function ℓ assigns a non-negative number to any combination of $g \in \mathcal{G}$ and $f \in \mathcal{F}$ and returns zero if $f = g$.

For nonparametric regression, an example is the squared pointwise loss: $\ell(\widehat{f}, f_0) = \big(\widehat{f}(\mathbf{x}) - f_0(\mathbf{x})\big)^2$ with $\mathbf{x}$ a given point. This loss is derived from the log-likelihood of the regression model with Gaussian noise. In classification, the goal is to predict a label y. A common choice for this task is the zero-one loss $\ell(\widehat{f}, y) = \mathbb{1}(\widehat{f} \neq y)$. This loss returns zero if the predicted label is correct and one otherwise.

To evaluate the performance of an estimator we consider the expected average loss over all possible realizations of the data.

Definition 1.1.5 (Risk)**.** The risk of an estimator $\widehat{f}$ with respect to the loss function ℓ is given by $R(\widehat{f}, g) := \mathbb{E}_g\Big[\ell\big(\widehat{f}, g\big)\Big]$. Here the expectation $\mathbb{E}_g$ is over the data distribution with parameter g.

In general the data distribution may also depend on other parameters besides g.

To compare estimators, we look at their worst-case risk over all possible parameters g. In other words, we are interested in $\sup_{g \in \mathcal{G}} R(\widehat{f}, g)$, where $\mathcal{G}$ denotes the class containing the quantity to be estimated. The (worst-case) risk cannot be computed exactly in most cases. Instead, one relies on upper bounds. For a sensible estimator, this upper bound should converge to zero as the sample size n increases. The rate at which the upper bound decreases is called the convergence rate.

Ideally, the worst-case risk should decrease to zero as fast as possible, but how fast can this possibly happen? The answer to this question depends on the imposed assumptions. For instance, for the parametric linear regression problem a faster rate of convergence can be reached, than for a nonparametric regression problem that involves all differentiable functions. It can be shown that without assumptions, the worst-case risk in the nonparametric regression model is lower bounded by some constant no matter how much data is available, see for example Theorem 3.1 of [54] or Section 7 of [38].

For many classes it is possible to prove lower bounds on the rate of convergence of the worst-case risk of any estimator. In other words, it can be shown that $\inf_{\widetilde{f}} \sup_{g \in \mathcal{G}} R(\widetilde{f}, g)$ cannot tend to zero faster than a certain rate. For an introduction to lower bounds, see for example [146]. Lower bounds for standard nonparametric regression settings were proven in [138].

If the rate of the lower bound matches the convergence rate of some estimator for this estimation problem, then the rate is called minimax (rate) optimal.

The main focus in this thesis are convergence rates for worst-case upper bounds in high-dimensional input settings related to deep learning. The empirical counterpart of the risk plays a crucial role in establishing these bounds.

1.1.3 Empirical risk minimization

The chosen risk determines the quality of the estimator and a ‘good’ estimator will have small risk. However, direct minimization of the risk is impossible; Computing the expected value in the definition of the risk requires knowledge of the underlying unknown data-distribution. To overcome this, one can replace the expectation by an average.

Definition 1.1.6 (Empirical risk)**.** The empirical risk of an estimator $\widehat{f}$ with respect to the loss function ℓ is given by $R_n(\widehat{f}) := \frac{1}{n} \sum_{i=1}^{n} \ell\big(\widehat{f}(\mathbf{X}_i), Y_i\big)$. In other words, the empirical risk is the average loss over all data-pairs $(\mathbf{X}_i, Y_i)$ in the dataset. An empirical risk minimizer with respect to the class $\mathcal{F}$ is any estimator $\widehat{f}$ satisfying $\widehat{f} \in \arg\min_{f \in \mathcal{F}} R_n(f)$.

The hope is that minimizing the empirical risk results in an estimator with low risk. In other words, it leads to a procedure that generalizes well to unseen samples. Plenty of theory has been developed showing that this is indeed true if the estimator class is not too large or too small, see for example [148, 150]. If the estimator class is too large, then too much of the noise gets incorporated into the estimator. In the extreme case, the estimator interpolates the data points. In this instance the empirical risk is zero, but depending on the loss function, the risk may be arbitrarily large. The estimator class in empirical risk minimization should also be not too small. Otherwise, the best function in the class could still be far away from the true function. The error that arises from this difference is known as the approximation error, while the error that comes from the vulnerability of the estimator to noise is called the stochastic error. A major challenge of empirical risk minimization is to choose function classes that balance approximation and stochastic error.

Both the Nadaraya-Watson estimator, Example 1.1.2, and the Fourier-series estimator, Example 1.1.3, can be derived as variants of empirical risk minimizers for different estimator classes. The linear least squares estimator for linear regression, used in Figure 1.1, is a parametric example of an empirical risk minimizer. It minimizes the empirical risk for the squared loss $\ell(\widehat{f}(\mathbf{x}), y) = (\widehat{f}(\mathbf{x}) - y)^2$ over the class of all linear functions.

The Nadaraya-Watson estimator $\widehat{f}_{NW}$, with nonnegative kernel K is a minimizer of a localized version of the empirical risk,

$$\widehat{f}_{NW}(\mathbf{x}) = \operatorname*{arg\,min}_{\theta \in \mathbb{R}} \sum_{i=1}^{n} (Y_i - \theta)^2 K\Big(\frac{1}{h}\Big(\mathbf{X}_i - \mathbf{x}\Big)\Big). \tag{1.1.1}$$

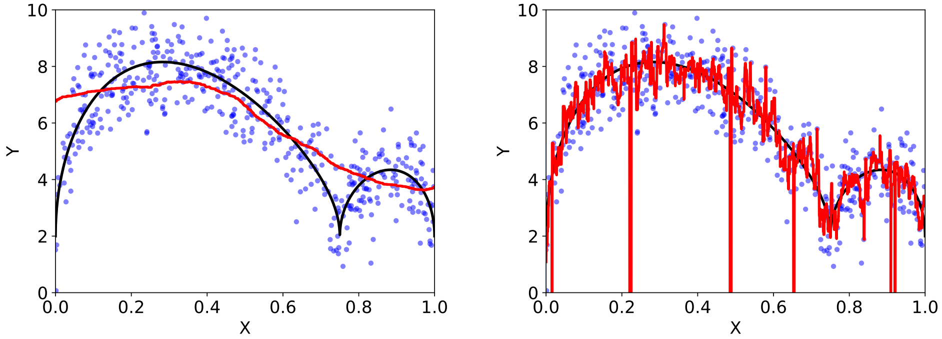
The family of local polynomial estimators can be obtained by replacing θ in (1.1.1) by a polynomial, see for example [146, 153, 93, 54]. The Nadaraya-Watson estimator is therefore a specific local polynomial estimator. For kernel estimators, the bandwidth parameter h controls the trade-off between the approximation and stochastic error. Larger h means that the neighborhood that is included in the kernel increases. This leads to a smoother estimate with a smaller stochastic error, but with a larger approximation error. On the other hand, a smaller h means that a smaller neighborhood is considered by the kernel. This results in a less smooth estimate with smaller approximation error, but a larger stochastic error. In Figure 1.6 the Nadaraya-Watson estimator for large and small bandwidth is plotted, using the same kernel and dataset as in Figure 1.3b.

Now consider the Fourier-series estimator $\widehat{f}_F(x)$ of level N with threshold τ as defined in Example 1.1.3. The linear combinations of the first N Fourier-basis functions form the class

$$\mathcal{F}_N := \Bigg\{ \sum_{j=1}^{N} c_j \psi_j, c_j \in \mathbb{R} \text{ for } j = 1, 2, \ldots, N \Bigg\}.$$

Using this class, the Fourier-series estimator can be rewritten as

$$\widehat{f}_F(x) = \operatorname*{arg\,min}_{f \in \mathcal{F}_N} \frac{1}{n} \sum_{i=1}^{n} \big(f(X_i) - Y_i\big)^2 + \frac{\lambda_\tau}{n} \operatorname{pen}(f), \tag{1.1.2}$$



(a) Large bandwidth

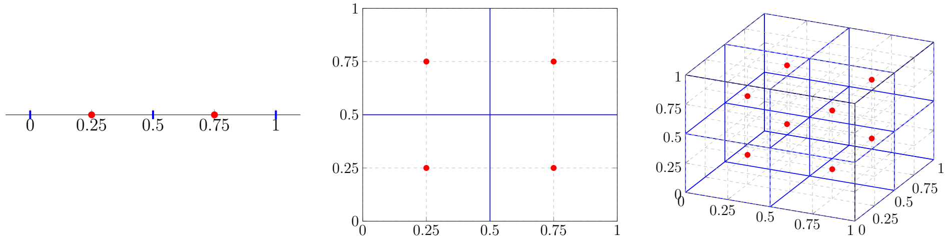
(b) Small bandwidth

Figure 1.6: Plots of the samples (blue), Nadaraya-Watson estimator (red) with a large bandwidth h on the left and a small bandwidth h on the right.

where the penalty pen(f) counts the number of non-zero coefficients c_j of f and the parameter λ_τ depends on the value of the threshold τ. This estimator thus minimizes a penalized/regularized version of the empirical risk. The number of terms considered in the series estimator N and the threshold τ control the approximation and stochastic error. Including more terms, thus increasing N and decreasing τ, leads to a smaller approximation error and a larger stochastic error. Including less terms decreases the stochastic error and increases the approximation error.

1.1.4 Curse of dimensionality

Nonparametric methods work well for low-dimensional input problems. However, in higher dimensions their performance degrades. The (input) dimension d appears in the minimax rate of many nonparametric problems in the power of n. For example, the minimax rate for the squared loss for estimating a β times differentiable function is of order $n^{-2\beta/(2\beta+d)}$, [137, 138, 54, 44]. As d grows, this rate gets slower. The reason for this is that in higher dimensions one needs (many) more data-points before every local neighborhood contains at least one observation. To see this, consider the problem of placing points in the space $[0,1]^d$ such that everywhere in $[0,1]^d$ one is at most $1/4$ in the maximum-norm distance away from one of these points. This is depicted in Figure 1.7. One needs $2 = 2^1$ points in one dimension, $4 = 2^2$ points in two dimensions and $8 = 2^3$ points in three dimensions. In general, one needs 2^d points in dimension d, an exponential growth in the dimension.



(a) Dimension 1 (b) Dimension 2 (c) Dimension 3

Figure 1.7: Required number of points to cover the cube $[0, 1]^d$ with maximum norm balls of radius 1/4.

In contrast, the dimension d appears in the minimax rate for parametric problems only as a multiplicative constant. For example, for linear regression the minimax rate for the squared loss is d/n, [63, 98, 144]. The strong structural assumptions in parametric models reduce the intrinsic dimension of the problem. This is related to the main idea behind approaches to tackle the curse of dimensionality in nonparametric regression: Introduce additional assumptions on the structure such that the intrinsic dimension of the function class becomes smaller. An example are generalized additive models, [56], assuming that the multivariate function can be written as a linear combination of univariate functions. Methods that make use of this structure are able to achieve the dimensionless rate $n^{-2\beta/(2\beta+1)}$ for β times differentiable functions instead of $n^{-2\beta/(2\beta+d)}$, see for example [139] or Section 22 of [54]. The possibility that deep neural networks are able to circumvent the curse of dimensionality is one of the motivations for the statistical analysis of deep learning.

1.2 Deep learning

Around 1960, the first trainable artificial neural networks were developed: The perceptron of Rosenblatt [121, 122] has zero-one output, and the adaptive linear element (ADALINE) [156] is a linear model trained with (a version of) gradient descent. Both the perceptron and ADALINE existed as dedicated physical machines, in contrast to modern neural networks which are software implementations.

In essence, these first neural networks consisted of one single trainable neuron. More neurons and layers were proposed [121, 122, 156], but in those setups only the parameters of the last neuron were changed during training. The other parameters were kept fixed [121, 150]. This means that, from a statistical point of view, all these early neural networks are parametric methods.

In Figure 1.8 a neural network with a single neuron is shown. This neural network multiplies each input coefficient x_i with a weight W_i. It then takes the sum over all these weighted inputs and adds the shift v to this sum. Finally, the activation function σ is applied in the single neuron. The trainable parameters in this neural network are the weights W_i and the shift v. This simple neural network already can be used for various tasks by choosing a suitable activation function σ. The original perceptron used the heaviside function $\sigma(x) = \mathbb{1}(x \geq 0)$, Figure 1.9a. With this activation function the neural network can be used for binary classification, e.g., for answering yes or no questions. When, as in ADALINE, σ is the identity function, Figure 1.9c, this neural network does linear regression. Taking σ as the logistic function $\sigma(x) = 1/(1 + e^{-x})$, Figure 1.9b, results in a neural network that does logistic regression, estimating the probability of an event. Sigmoid activation functions, such as the logistic function, can be considered as smooth alternatives for the heaviside function. These activation functions became the standard in the late eighties.

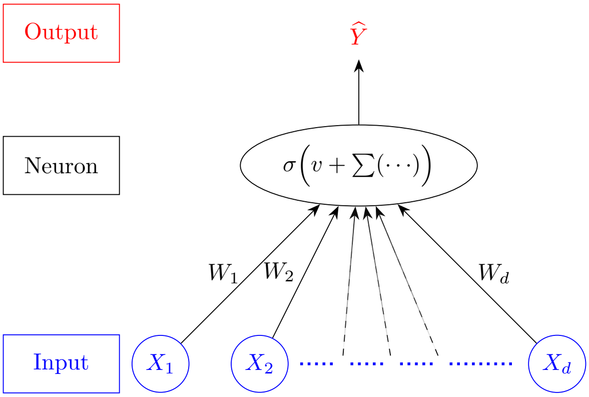
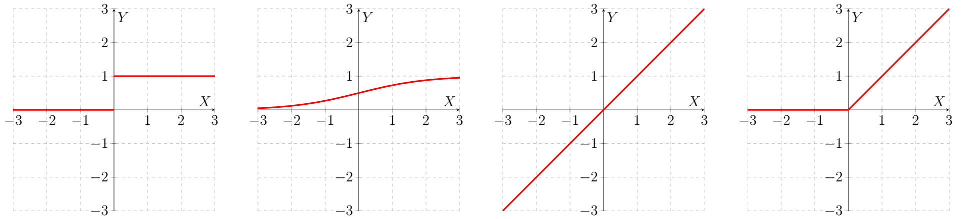


Figure 1.8: A neural network with one single neuron.

Modern deep neural networks consist of multiple neurons ordered in layers. A neural network consists of an input layer, several hidden layers, and an output layer. The number of hidden layers is also called the depth of the neural network, and the ‘deep’ in deep learning refers to neural networks with multiple layers. In Figure 1.10 an example of a deep neural network is given. This example has d inputs, three hidden layers with four neurons each, and a single neuron as its output. Such a neural network is called a fully connected feedforward network: fully connected since



(a) Heaviside

(b) Logistic

(c) Linear

(d) Rectified Linear Unit (ReLU)

Figure 1.9: Examples of activation functions

every node in a layer is connected to all nodes in the previous and the next layer, feedforward because all connections go forward to the next layer. In fully connected feedforward networks, all hidden units generally have the same activation function σ. Nowadays, the most common choice is the rectified linear unit (ReLU) activation function, Figure 1.9d. This is the activation function used in the hidden layers of the neural networks considered in Chapters 2 and 3. Depending on the learning task, the output may use different activation functions. The linear activation function is used for regression/function estimation problems. Chapter 3 considers neural networks with this function in the output layer. For estimating a binary probability the logistic function is the standard choice for the activation function of the output neuron. Chapter 2 deals with estimating vectors of probabilities. For this task a multivariate version of the logistic function, the softmax function, is used as output activation function.

Around 1990 it was shown that shallow neural networks, neural networks with one hidden layer, have the so-called universal approximation property: If the number of neurons in the hidden layer is allowed to become arbitrarily large, then shallow neural networks are able to approximate any continuous function arbitrarily well, [34, 61, 46]. Multi-neuron networks are thus able to approximate large function classes if the neural network size, and thus the number of parameters, grows as a function of the data. This means that multi-neuron networks can be considered as nonparametric methods.

In the nineties it was proven that shallow neural networks could approximate specific classes of multivariate functions with a rate whose power of n does not contain the input dimension d, [65, 7, 8, 27]. However, these classes come with Fourier transform conditions that depend on d: if the dimension increases, then these conditions become more restrictive. More recently, it has been shown that deep neural networks with ReLU activation function can approximate various classes of functions better than shallow neural networks, [160, 141, 110]. This is in contrast to older works which considered smooth and bounded activation functions, such as the logistic activation function, Figure 1.9b. Following these approximation results, convergence rates for the risk in the regression problem were derived. These results showed that if the regression function has a compositional structure, then deep neural networks can exploit this structure to circumvent the curse of dimensionality, [71, 62, 72, 113, 11, 127, 77].

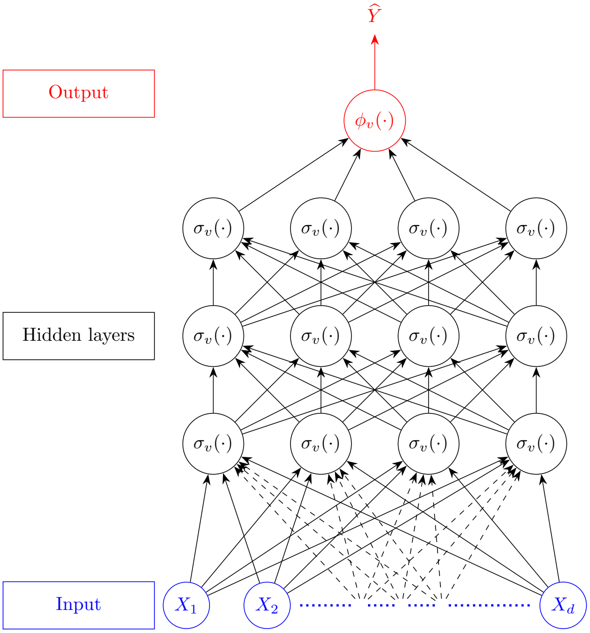


Figure 1.10: A neural network with three hidden layers with four neurons each, an input layer with d inputs, and an output layer with a single output neuron.

1.2.1 Training of neural networks

For supervised learning problems training a neural network means minimizing a training loss. In other words, deep learning is an example of a (regularized) empirical risk minimization approach as discussed in Section 1.1.3. Unlike the examples in that section, it is impossible to derive an explicit solution of the minimization problem. Instead, the empirical risk is minimized stepwise during training by an optimization method. The most common method is gradient descent (or a variation thereof): In each training step the parameters are updated according to the update rule:

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \alpha_k \nabla_{\boldsymbol{\theta}_{k-1}} \left(\frac{1}{n} \sum_{i=1}^{n} \ell\big(f_{\boldsymbol{\theta}_{k-1}}(\mathbf{X}_i), Y_i\big) \right), \tag{1.2.1}$$

for a sequence of positive numbers α_k called the learning rate and $f_{\boldsymbol{\theta}_{k-1}}$ the neural network with parameters $\boldsymbol{\theta}_{k-1}$. In the neural networks as described before, the parameters $\boldsymbol{\theta}$ are all the weight matrices W and all the shift vectors v. For convex problems, gradient descent converges to the global minimum for suitable sequences of learning rates. Neural networks viewed as functions do not lead to a convex function class. Training neural networks is therefore a non-convex problem. Instead of one global minimum, there may exist multiple minima, which may be local or global [37, 125, 29]. As a consequence, a minimum found during training might be a local minimum and in this case there exist neural network-parameters that achieve a lower training loss.

Training neural networks with multiple layers of neurons became feasible with the introduction of backpropagation in 1986 in [124] and [86]. Backpropagation consists of a forward and a backward pass through the neural network. In the forward pass, the neural network is given a training sample as input and the output of the neural network is computed for this sample. In the backward pass, the output is used to calculate the training loss, after which the gradient for all parameters is calculated using the chain-rule for differentiation.

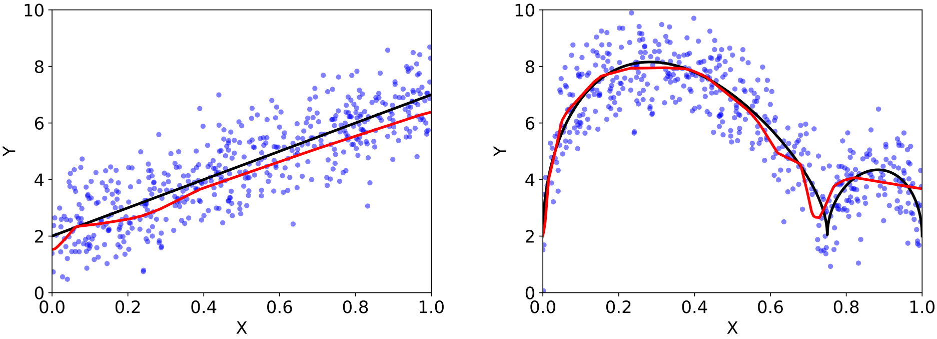
In Figure 1.11 estimates by fully connected feedforward networks for the regression dataset from Section 1.1 are shown. These neural networks have 10 hidden layers with 50 neurons each and use the ReLU activation function in the hidden layers and the linear activation in the output layer.

For a further history of deep learning see [51] or for a more statistics oriented overview of machine learning see [150].

1.3 Introduction for Chapter 2: Classification

The current popularity of deep learning is in part caused by the performance of deep neural networks for image recognition tasks: identifying what object is depicted in an image. In 2012, a deep neural network [80] became state-of-the-art by outperforming other methods in the ImageNet Large Scale Visual Recognition Challenge.

In statistical terms, image recognition is an example of a supervised classification problem. In the classification model with K classes, we observe n random pairs $(\mathbf{X}_1, \mathbf{Y}_1), (\mathbf{X}_2, \mathbf{Y}_2), \ldots, (\mathbf{X}_n, \mathbf{Y}_n)$, with $\mathbf{Y}_i$ the encoding of the observed label or class



(a) Linear

(b) Non-linear

Figure 1.11: Plots of the samples (blue), Deep Neural Network estimator (red) and the true regression function (black) of a linear and a non-linear model.

of the input $\mathbf{X}_i$. For more than two classes, it is standard to represent the labels with one-hot encoding: Each $\mathbf{Y}_i$ is a K-dimensional vector consisting of exactly one 1 and all other coefficients are set to zero. The relationship between $\mathbf{X}_i$ and $\mathbf{Y}_i$ is determined by the (unknown) conditional class probabilities

$$\mathbb{P}(Y_{i,k} = 1 | \mathbf{X}_i = \mathbf{x}),$$

where $Y_{i,k}$ is the k-th coefficient of $\mathbf{Y}_i$ and $\mathbf{x}$ is a specific value taken by the input $\mathbf{X}_i$. In classification, the goal is to predict the class label of a new input. In machine learning, classification methods are often compared based on the fraction of correct classifications. This is equal to one minus the risk corresponding to the 0-1 loss.

To build a classifier, one can try to estimate the decision boundaries directly. Alternatively, one can first estimate the conditional class probabilities and then plug these estimates into a decision rule. For an overview of classification methods see for example [38].

Deep neural networks output estimates of the conditional class probabilities, see for example the seminal work [80]. Furthermore, neural networks are typically compared to other methods based on the faction of correct labels that are contained in the (top five or ten) most likely predicted labels. The activation function commonly used in the output layer of these neural networks is the softmax function

$$\mathbf{\Phi}(\mathbf{x}) := \left(\frac{e^{x_1}}{\sum_{j=1}^K e^{x_j}}, \ldots, \frac{e^{x_K}}{\sum_{j=1}^K e^{x_j}} \right) : \mathbb{R}^K \to \mathcal{S}^K,$$

where $\mathcal{S}^K = \big\{(p_1, \ldots, p_K : \sum_{j=1}^K p_j = 1, p_j \geq 0\big\}$ denotes the probability simplex in $\mathbb{R}^K$. The softmax function is a multivariate version of the logistic function in Figure 1.9b and guarantees that the output of the neural network is a probability vector.

Because of the non-differentiability, gradient descent cannot be applied to the 0-1 loss. Therefore, neural networks are trained using a surrogate loss [9, 140]. The cross-entropy loss is commonly used in combination with the softmax output:

$$\ell(\widehat{p}(\mathbf{X}_i), \mathbf{Y}_i) := -\sum_{k=1}^K Y_{i,k} \log(\widehat{p}_k(\mathbf{X}_i)).$$

The cross-entropy loss can be derived from the log-likelihood of the conditional class probabilities, see Section 2.2. Chapter 2 focuses on estimating the conditional class probabilities instead of the final classification. Therefore, the risk corresponding to the cross-entropy loss is considered instead of the risk corresponding to the 0-1 loss. Convergence rates for a neural network estimator are derived with respect to a truncated version of the risk corresponding to the cross-entropy loss.

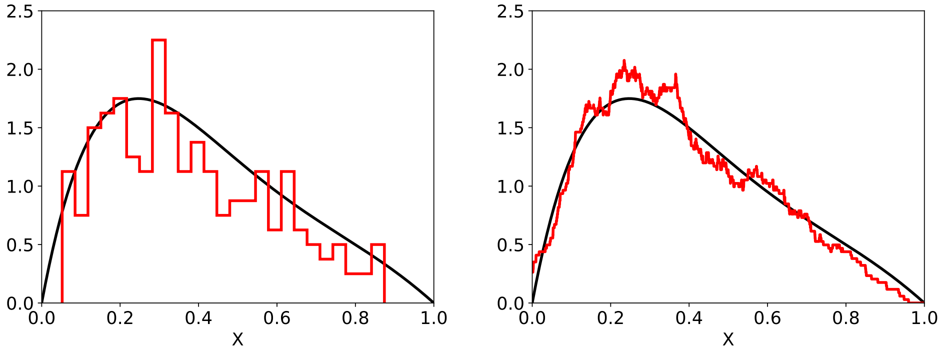
1.4 Introduction for Chapter 3: Multivariate density estimation

In multivariate density estimation, we observe a sequence of independent random vectors $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ distributed according to some multivariate density f_0. The density estimation problem is to estimate this unknown density f_0 from the data. Unlike regression and classification, there are no observed response variables Y_i, making this problem unsupervised.

Kernel density estimators are standard methods for nonparametric density estimation. A kernel in d-dimensions is a function $K : \mathbb{R}^d \to \mathbb{R}$ such that $\int_{\mathbb{R}} K(\mathbf{u})\, d\mathbf{u} = 1$. The kernel density estimator $\widehat{f}_K$ with kernel K and bandwidth h is given by:

$$\widehat{f}_K := \frac{1}{nh^d} \sum_{i=1}^n K\bigg(\frac{1}{h}\Big(\mathbf{X}_i - \mathbf{x}\Big)\bigg). \tag{1.4.1}$$

Kernel density estimators are related to histograms. For the rectangular kernel, Figure 1.2a, the corresponding kernel density estimator can be derived as an average of an infinite number of histograms with shifted bin centers. Alternatively, one can derive this estimator from an approximation of the derivative of the cumulative distribution function (cdf), an approach that was taken in early work on kernel density estimation, [123, 108]. Figure 1.12 compares histogram and kernel density estimator with rectangular kernel.



(a) Histogram

(b) Kernel density estimator

Figure 1.12: Plots, based on the same sample, of the histogram estimator (red) on the left and the kernel density estimator with rectangular kernel (red) on the right. The true density function (black) is a mixture of two beta-distributions.

In Chapter 3 we transform the unsupervised density estimation problem into a supervised regression problem. We first use a kernel density estimator to generate response variables. This allows us to fit a deep ReLU network using existing results for regression. We derive convergence rates showing that this approach can use compositional structures to partly circumvent the curse of dimensionality. We also provide an exploratory simulation study applying this method to several structural density models.

1.5 Introduction for Chapter 4: Optimization motivated by biological neural networks

From the beginning, artificial neural networks have been influenced by theories about biological neural networks (the brain). The first artificial network, the perceptron [121, 122] was based on the McCulloch-Pitts model [95] for neurons in the brain [51, 150]. The firing pattern of neurons in the brain has been cited as motivation for the ReLU activation function, Figure 1.9d, [50]. However, the main interest in ReLU activation functions originates from the empirically observed performance improvement compared to the previously used sigmoid activation functions [50, 102]. Importantly, artificial neural networks are not intended to represent learning in the brain. Popular and successful methods for artificial neural networks, such as gradient descent and backpropagation, are even implausible for biological neural networks [33, 89, 142]. One issue is that these training methods require the capacity to share information about all the parameters with the entire neural network, also known as the weight transportation problem.

Recently there has been renewed interest in the differences and similarities between artificial and biological neural networks, [89, 128, 155]. For example, alternatives to gradient descent have been proposed that are more biologically plausible. One of them is known as (weight-perturbed) forward gradient descent [13, 117]. In this method the gradient update step (1.2.1) is replaced by the update step

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \alpha_k \Big(\nabla_{\boldsymbol{\theta}_{k-1}} \ell\big(f_{\boldsymbol{\theta}_{k-1}}(\mathbf{X}_k), Y_k\big)\Big)^\top \boldsymbol{\xi}_k \boldsymbol{\xi}_k,$$

where $\boldsymbol{\xi}_k$ is distributed as $\mathcal{N}(0, \mathbf{I}_d)$ and is independent of all other randomness. Thus, only random linear combinations of the gradient are required instead of the full gradient. In Chapter 4, we study forward gradient descent in the framework of the linear regression model. We prove that in this setting the mean squared error converges with a rate $d^2 \log(d)/k$, for a large enough number of samples k. This rate has an additional dimension factor $d \log(d)$ compared to the optimal rate for linear regression [144, 63, 98].