Artificial Neural Networks (ANNs)

Suvinava Basak

January 11, 2025

Abstract

Artificial intelligence has been making tremendous strides in closing the gap between human and computer capabilities. Both amateurs and researchers work on many facets of the area to achieve incredible results. The field of computer vision is one of several such fields. The goal of this field is to make it possible for machines to see and perceive the world similarly to humans. They will then be able to use this knowledge for a variety of tasks, including Natural Language Processing, Media Recreation, Image and Video Recognition, Image Analysis and Classification, Recommendation Systems, and more. Deep Learning's contributions to computer vision have been developed and refined throughout time, primarily using an algorithm, **Convolutional Neural Network**.

In this book, we will dig deeper into the realm of CNN and explore many things, starting from the **historical perspective** to the building blocks of a CNN along with their **mathematical foundations**. We will also look at different CNN architectures, e.g., **LeNet**, **AlexNet**, **VGGNet**, **GoogleNet**, **ResNet**, **MobileNet**, **EfficientNet** etc. and some of the real-life applications. Finally, we end this book with some of the current research and future trends of CNNs.

Contents

1 Introduction to Neural Networks

A **Neural Network** is a computational model inspired by the structure and functioning of the human brain. It consists of layers of interconnected nodes, called neurons, that process data in a manner similar to biological neurons. Neural networks are the foundation of **deep learning**, a subset of machine learning, and are used to recognize patterns, make predictions, and solve complex problems.
Key components of a neural network include:

- **Input Layer**: Accepts the raw data for processing.
- **Hidden Layers**: Perform computations to extract patterns or features through weighted connections and activation functions.
- **Output Layer**: Produces the final result, e.g., classification or regression output.

Neural networks learn by adjusting the weights of connections between neurons using algorithms like **backpropagation**, minimizing the error between predicted and actual outputs. They are widely applied in tasks such as image recognition, natural language processing, and time-series forecasting.

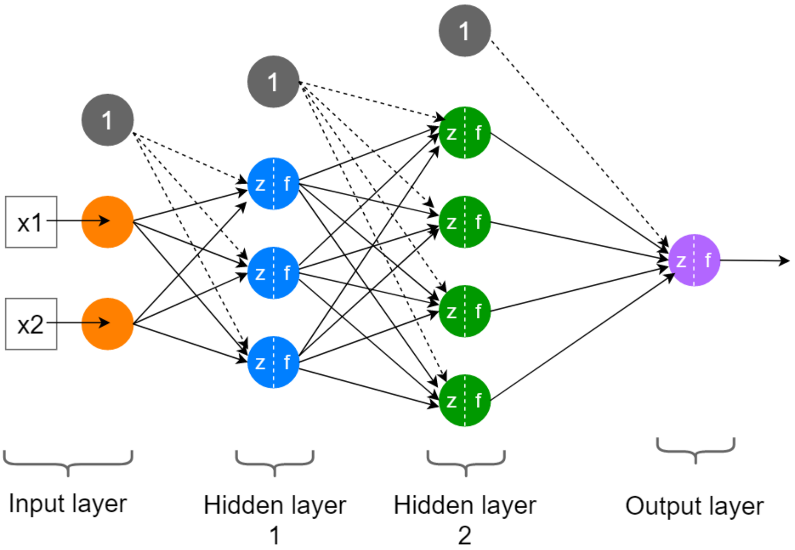

Figure 1.1: A typical Artificial Neural Network

1.1 Background

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of biological neural networks in the human brain. The idea of mimicking biological neurons dates back to the 1940s with the introduction of the **McCulloch-Pitts neuron model**, which laid the foundation for modern neural networks.[MP43] Early research explored how networks of artificial neurons could compute logic functions and recognize patterns.

By the 1980s, the **backpropagation algorithm**, introduced by Rumelhart, Hinton, and Williams, marked a turning point by enabling efficient training of multi-layer networks.[RHW86] This innovation fueled interest in neural networks for tasks such as image recognition, speech processing, and robotics.

Despite their promise, neural networks struggled during the 1990s due to computational limitations and the dominance of simpler models like support vector machines (SVMs). However, the resurgence of interest in the 2010s, termed the **”deep learning revolution”**, was driven by three factors:

- Availability of large datasets (e.g., ImageNet)
- Advancements in computational power (e.g., GPUs and TPUs).
- Development of new architectures and techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).[LBH15][KSH12]

1.2 Applications of Neural Networks

Neural networks are used across diverse domains, revolutionizing traditional methods with their ability to learn from data and generalize well. Prominent applications include:

- **Image Recognition**: Tasks like facial recognition and object detection rely on CNNs to extract hierarchical features from images.[LBH15]
- **Natural Language Processing (NLP)**: Models like transformers power language models such as ChatGPT and BERT, enabling applications like sentiment analysis, translation, and summarization.[VSP+17]
- **Healthcare**: ANN-based solutions are applied for early diagnosis of diseases, drug discovery, and personalized treatment recommendations.[EKN+17]
- **Autonomous Systems**: Self-driving cars use neural networks to process sensor data for navigation and decision-making.[BTD+16]

and many more...

1.3 Importance of Neural Networks in Modern AI

Neural networks form the backbone of modern AI, offering unparalleled flexibility and scalability for solving complex problems. Their key advantages include:

1. **Automatic Feature Extraction**: Unlike traditional machine learning algorithms, ANNs can automatically learn hierarchical features from raw data.
2. **Non-linearity**: Through activation functions, ANNs can model complex, non-linear relationships in data.

3. **Scalability**: Modern architectures, such as deep neural networks, allow scaling to billions of parameters for solving real-world problems at scale.

However, ANNs also have limitations, such as their need for large amounts of labeled data, computational intensity, and susceptibility to overfitting. Addressing these challenges remains a significant focus in the AI research community.[GBC16]

2 Logistic Regression

2.1 Fundamentals of Logistic Regression

Logistic regression is a statistical method for binary classification, where the goal is to predict a binary outcome (e.g., success/failure, 0/1) based on one or more input variables. Unlike linear regression, logistic regression outputs probabilities by applying a **sigmoid function** to a linear combination of the inputs.

Consider a set $\mathcal{S} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\} \subseteq \mathcal{X} \times \mathcal{Y}$ of *training data* with $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$. In this set, every $x^{(i)}$ represents an *object* and the corresponding *label* $y^{(i)}$ indicates which one of two possible cases the object belongs to. *Binary Classification* describes the search for a map $h_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by $\theta \in \Theta$, which maps every object $x \in \mathcal{X}$ to a label $y \in \mathcal{Y}$.

In this context, we call Θ the *parameter space* and h_θ a *hypothesis*. A particular approach for identifying suitable hypotheses for binary classification problems is *logistic regression* (for example, see [Bis06, chapter 4.3.2] and [GBC16, chapter 5.7.1]), which we discuss in what follows.

Definition 2.1.1 (Sigmoid function)**.** The function

$$\sigma : \mathbb{R} \to (0, 1), \quad \sigma(z) := \frac{1}{1 + e^{-z}} \tag{2.1}$$

is called a *sigmoid function* or (more general) *logistic function*.

Here, $z = w^\top x + b$, where w represents the weights, x the input features, and b the bias term.

Key features:

- **Output Interpretation**: The sigmoid function ensures outputs lie in the range $(0, 1)$, making them interpretable as probabilities.
- **Decision Boundary**: The model predicts class 1 if $\sigma(z) \geq 0.5$ and class 0 otherwise. This results in a decision boundary defined by the hyperplane $w^\top x + b = 0$.[HTF09]

2.2 Learning as an Optimization Problem

From a statistical perspective, logistic regression models the probability distribution of the data as a **Bernoulli distribution** with parameter $\sigma(z)$. The model is trained by maximizing the **likelihood function**, which represents the probability of observing the given training data under the model parameters.

Maximum Likelihood Viewpoint

The goal is to come up with a distribution on $\mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^d \times \{0,1\}$ that "prioritizes" the training data. In other words, it should assign small (or even zero) values to an elementary event $(x, y) \in \mathcal{X} \times \mathcal{Y}$, where y is **not** the correct label of x. In order to approach this task, we consider the family

$$\mathcal{D}_{Mod}(y \mid x; \theta) := \sigma(w^\top x + b)^y \cdot (1 - \sigma(w^\top x + b))^{1-y}, \quad \theta = (w, b) \in \Theta \tag{2.2}$$

of *model* measures on $\mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^d \times \{0,1\}$ parametrized by $\Theta = \mathbb{R}^d \times \mathbb{R}$. We have

$$\mathcal{D}_{\mathrm{Mod}}(y \mid x; \theta) = \begin{cases} \sigma(w^\top x + b), & \text{if } y = 1 \\ 1 - \sigma(w^\top x + b), & \text{if } y = 0 \end{cases} \tag{2.3}$$

and $\mathcal{D}_{\mathrm{Mod}}(\cdot \mid x; \theta)$ is a Bernoulli probability distribution with parameter $\sigma(w^\top x + b)$. This parameter hence represents (in our model) the probability that $y = 1$ is the correct label for a given x, while the probability that $y = 0$ is the correct label for x equals $1 - \sigma(w^\top x + b)$.

The problem of *Maximum Likelihood Estimation* seeks a parameter $\theta = (w, b)$ such that the probability of observing $\mathcal{S}$ is maximal (among $\mathcal{D}_{\mathrm{Mod}}(\cdot \mid \cdot; \theta)$). Thus, given *pairwise independent* random variables $X^1, \ldots, X^m$ on $\mathcal{X} \times \mathcal{Y}$ distributed according to $\mathcal{D}_{\mathrm{Mod}}(\cdot \mid \cdot; \theta)$, we have to maximize the *likelihood function*:

$$\begin{aligned} \mathcal{L}(\theta) &:= \mathbb{P}(\{X^1 = (x^{(1)}, y^{(1)}), \ldots, X^m = (x^{(m)}, y^{(m)})\}), \quad \theta = (w, b) \\ &\overset{\text{i.i.d.}}{=} \prod_{i=1}^{m} \mathcal{D}_{\mathrm{Mod}}(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \sigma(w^\top x^{(i)} + b)^{y^{(i)}} \cdot (1 - \sigma(w^\top x^{(i)} + b))^{1-y^{(i)}} \end{aligned} \tag{2.4}$$

Equivalently, we can minimize the negative log of the likelihood function as below, which gives us the *cross entropy loss function*:

$$\begin{aligned} -\ln \mathcal{L}(\theta) &= -\ln(\prod_{i=1}^{m} \mathcal{D}_{\mathrm{Mod}}(y^{(i)} \mid x^{(i)}; \theta)) \\ &= -\sum_{i=1}^{m} \ln(\mathcal{D}_{\mathrm{Mod}}(y^{(i)} \mid x^{(i)}; \theta)) \\ &= -\sum_{i=1}^{m} \ln(\sigma(w^\top x^{(i)} + b)^{y^{(i)}} \cdot (1 - \sigma(w^\top x^{(i)} + b))^{1-y^{(i)}}) \\ &= -\sum_{i=1}^{m} [\ln(\sigma(w^\top x^{(i)} + b)^{y^{(i)}}) + \ln(1 - \sigma(w^\top x^{(i)} + b))^{1-y^{(i)}})] \\ &= -\sum_{i=1}^{m} [y^{(i)} \ln(\sigma(w^\top x^{(i)} + b)) + (1 - y^{(i)}) \ln(1 - \sigma(w^\top x^{(i)} + b))] \end{aligned} \tag{2.5}$$

The parameters w and b are learned by minimizing this loss using optimization algorithms such as gradient descent.[Mur12]

Thus, a maximum likelihood estimator $\hat{\theta} \in \Theta$ is a solution to the following non-linear optimization problem:

$$-\ln \mathcal{L}(\hat{\theta}) = \min_{\theta \in \Theta} -\ln \mathcal{L}(\theta) \tag{2.6}$$

2.3 From the Model Distribution to the Hypothesis

Once the parameter $\theta = (w, b)$ is fixed, we apply the hypothesis

$$h_\theta(x) := \begin{cases} 1, & \text{if } \mathcal{D}_{\text{Mod}}(1 \mid x; \theta) \geq 0.5 \\ 0, & \text{else} \end{cases} \tag{2.7}$$

to an object $x \in \mathcal{X}$ with (potentially) unknown label $y \in \mathcal{Y}$. That means we apply, based on our model distribution, “the most likely” label to x (which one can consider to be the natural or intuitive thing to do). In case that x is on the boundary region $\mathcal{D}_{\text{Mod}}(1|x;\theta) = \mathcal{D}_{\text{Mod}}(0|x;\theta) = 0.5$, where both labels have the same probability, we need to take a decision of favoring one label.

Due to the fact that $\sigma(0) = 0.5$ and that σ is a strictly monotonically increasing function (refer Figure 2.3), we conclude that the sets $\{x \in \mathcal{X} \mid h_\theta(x) = 1\}$ and $\{x \in \mathcal{X} \mid h_\theta(x) = 0\}$ are separated by the hyperplane $\mathcal{E} := \{x \in \mathcal{X} \mid w^\top x + b = 0\}$. Therefore, our decision rule 2.7 is described as:

$$h_\theta(x) := \begin{cases} 1, & \text{if } w^\top x + b \geq 0 \\ 0, & \text{else } w^\top x + b < 0 \end{cases} \tag{2.8}$$

and we call the hyperplane $\mathcal{E}$ the *decision boundary* (see Figure 2.1).

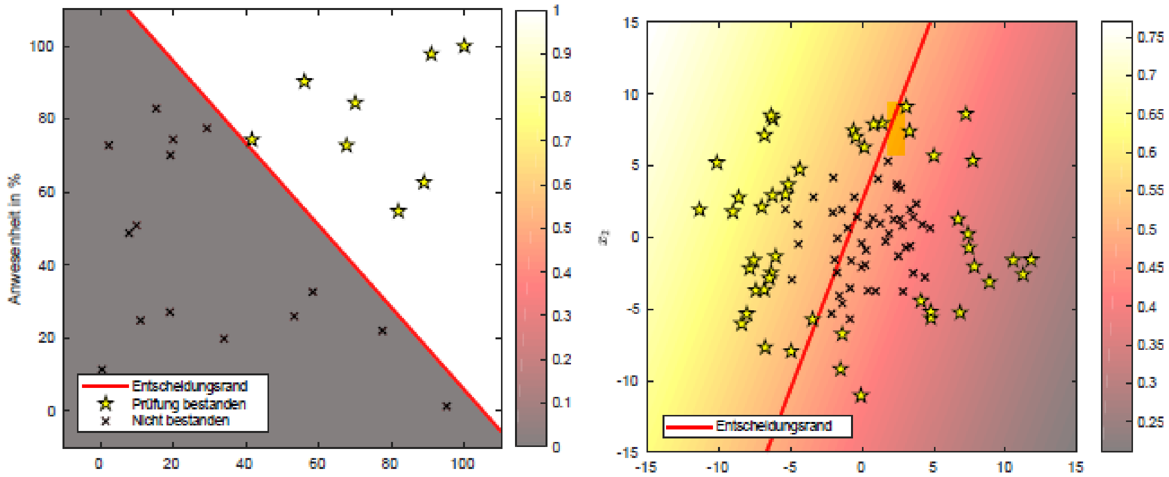

Figure 2.1: **Left:** Example for a logistic regression on linearly separable data. **Right:** Example for logistic regression on linearly non-separable data. The computed hypothesis achieves only an accuracy of 68% on the training data. The coloring represents the values of $\sigma(w^\top x + b)$.

Figure 2.2 shows how logistic regression typically works to predict the label.

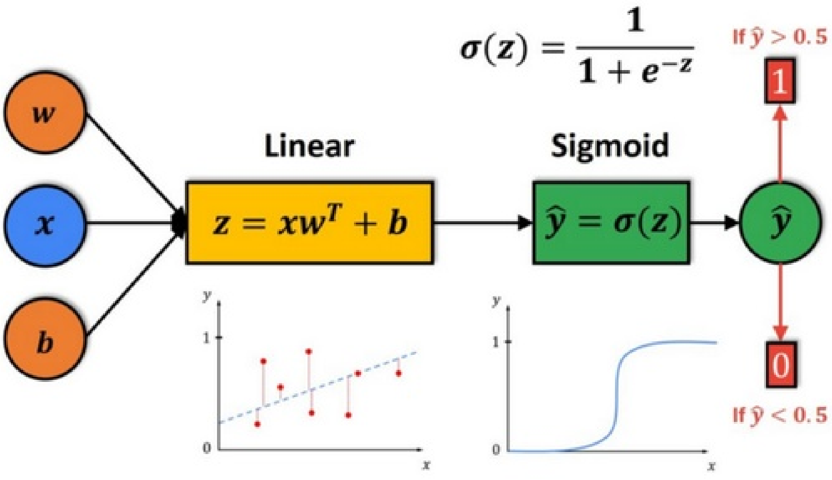

Figure 2.2: How Logistic Regression typically works

2.4 Machine Learning Formulation

Now we embed the adopted statistical point of view into the context of machine learning. We take a closer look at the summand in our target function 2.5, which we have to minimize (refer 2.6).

Definition 2.4.1 (Cross entropy loss function)**.** The function

$$\ell : \{0,1\} \times \{0,1\} \to \mathbb{R}, \quad \ell(y,\hat{y}) := -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y}) \tag{2.9}$$

is called *cross entropy loss function*.

Note that $\ell(y,\hat{y})$ is nothing but the cross entropy between Bernoulli distributions with parameters y and $\hat{y}$. Using this cross-entropy loss function, we reformulate the minimization problem 2.6 as:

$$\min_{\theta=(w,b)\in\Theta} \frac{1}{m}\sum_{i=1}^{m} \ell(y^{(i)}, \sigma(w^\top x^{(i)} + b)) \tag{2.10}$$

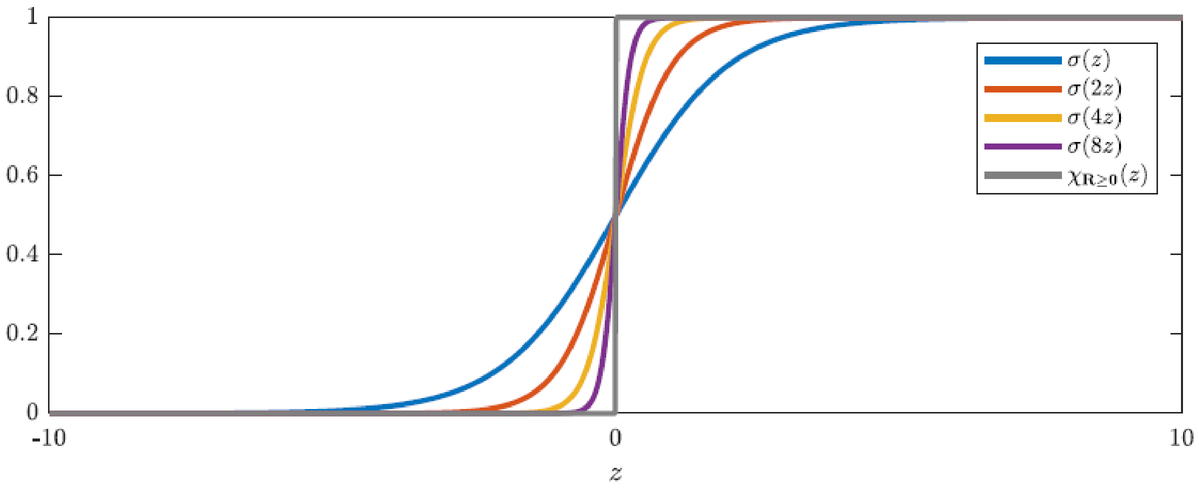

Figure 2.3: The sequence $(\sigma(nz))_{n\in\mathbb{N}}$ converges uniformly to a characteristic function on $\mathbb{R}\setminus\{0\}$.

NOTE: The additional factor $\frac{1}{m}$ does not change the optimal points (i.e., maximum Likelihood estimators $\hat{\theta}$). We can interpret every summand in (2.10) as the distance between a known target value (i.e., label) $y^{(i)}$ and a prediction $\sigma(w^\top x^{(i)} + b)$ based on our model. The more accurate our model makes predictions on the training data, the smaller are the individual values $\ell(y^{(i)}, \sigma(w^\top x^{(i)} + b))$ contributing to the target function of the optimization problem.

2.5 Minimizing Risks

Minimizing sums of the form (2.10) is a (type of) problem that one encounters regularly in the context of machine learning. To make our goal, the prediction of class membership of objects $x \in \mathcal{X}$ with unknown labels, more precise, we investigate another alternative formulation of (2.10).

Definition 2.5.1 (Empirical Risk Distribution)**.** The distribution $\hat{\mathcal{D}}$ with

$$\hat{\mathcal{D}}(A) := \frac{1}{m} \sum_{i=1}^{m} \delta_{(x^{(i)}, y^{(i)})}(A) \tag{2.11}$$

on $\mathcal{X} \times \mathcal{Y}$ and

$$\delta_{(x^{(i)}, y^{(i)})}(A) = \begin{cases} 1, & \text{if } (x^{(i)}, y^{(i)}) \in A \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

is called *empirical distribution* with respect to $\mathcal{S}$.

We can formulate the target function in (2.10) as an expected value as below:

$$\frac{1}{m} \sum_{i=1}^{m} \ell(y^{(i)}, \sigma(w^\top x^{(i)} + b)) = \mathbb{E}_{(x,y) \sim \hat{\mathcal{D}}} \ell(y, \sigma(w^\top x + b)) =: \mathcal{J}(\theta) \tag{2.13}$$

Thus, 2.13 is also called the *empirical risk* and the computation of $\hat{\theta}$ via the problem

$$\mathcal{J}(\hat{\theta}) = \min_{\theta \in \Theta} \mathcal{J}(\theta) \tag{2.14}$$

which is equivalent to (2.10), is called *empirical risk minimization*.

However, our goal is not minimizing the empirical risk, but rather minimizing the *true risk*

$$\mathbb{E}_{(x,y) \sim \mathcal{D}} \ell(y, \sigma(w^\top x + b)) =: \mathcal{J}^*(\theta), \tag{2.15}$$

i.e., we have to find

$$\theta^* \in \arg\min_{\theta \in \Theta} \mathcal{J}^*(\theta) \tag{2.16}$$

But, we do not know the underlying distribution $\mathcal{D}$, and hence we cannot solve the problem (2.16) directly. Empirical risk minimization can therefore be interpreted as *replacing* the unknown (true) distribution $\mathcal{D}$ in (2.16) by the empirical distribution $\hat{\mathcal{D}}$, which is induced by our sample $\mathcal{S}$.

2.6 Limitations of Logistic Regression: Linear Separability

While logistic regression is simple and interpretable, it has significant limitations that restrict its applicability.

2.6.1 Linear Decision Boundaries

Logistic regression assumes that the classes can be separated by a linear boundary. For datasets with complex, non-linear relationships, this assumption leads to poor performance.

In section 2.3, we interpreted logistic regression as learning a hyperplane $\mathcal{E}$ that separates the space $\mathcal{X} = \mathbb{R}^d$ in two half-spaces $\{x \in \mathcal{X} \mid h_\theta(x) = 0\}$ and $\{x \in \mathcal{X} \mid h_\theta(x) = 1\}$. If the hyperplane, which we learn, classifies all objects in our training data correctly (i.e., the training data is linearly separable), then we have $h_\theta(x^{(i)}) = y^{(i)}$ for all training examples. If we define $\mathcal{X}_\mathcal{S} := \{x \in \mathcal{X} \mid \exists y \in \mathcal{Y} : (x, y) \in \mathcal{S}\}$, then it follows immediately that the sets

$$\begin{aligned} \mathcal{X}_0 &:= \{x^{(i)} \in \mathcal{X}_\mathcal{S} \mid y^{(i)} = 0\} \subset \{x \in \mathcal{X} \mid h_\theta(x) = 0\} \\ \mathcal{X}_1 &:= \{x^{(i)} \in \mathcal{X}_\mathcal{S} \mid y^{(i)} = 1\} \subset \{x \in \mathcal{X} \mid h_\theta(x) = 1\} \end{aligned} \tag{2.17}$$

are also linearly separable. Note that the property to be linearly separable depends only on the sample $\mathcal{S}$ and is hence *independent* of the hyperplane which we learned. This means in particular that, if two sets $\mathcal{X}_0$ and $\mathcal{X}_1$ cannot be separated linearly, then it is *impossible* to learn a hyperplane via logistic regression that classifies all training data correctly. In other words, we obtain $h_\theta(x^{(i)}) \neq y^{(i)}$ for a positive percentage of our training data (see right-hand side of Figure 2.4).

2.6.2 Feature Engineering Dependency

In order to learn nonlinear decision boundaries in the context of logistic regression, one can use *basis functions* (see [Bis06, chapter 4.3.2]), $\phi_1, \dots, \phi_n : \mathbb{R}^d \to \mathbb{R}$. Every one of these basis functions $\phi_j(x)$ can be nonlinear in x. We can interpret these functions as representing a particular *pattern* of the object x. Thus, we call the vectors

$$\phi^{(i)} := \phi(x^{(i)}) := \begin{pmatrix} \phi_1(x^{(i)}) \\ \vdots \\ \phi_n(x^{(i)}) \end{pmatrix} \tag{2.18}$$

also *feature vectors*. The feature vector $\phi^{(i)}$ represents the object $x^{(i)}$ via the extracted pattern $\phi_1(x^{(i)}), \dots, \phi_n(x^{(i)})$, which are nothing but the coordinates of $\phi^{(i)}$. If the basis functions are given (or chosen), then we can apply logistic regression as described before with the crucial difference that we now use training data $\{(\phi^{(1)}, y^{(1)}), \dots, (\phi^{(m)}, y^{(m)})\} \subseteq \mathbb{R}^n \times \{0, 1\}$.

Note that the cardinality n of the basis functions does not need to coincide with the dimension d of the objects x.

If we apply logistic regression to data, which was transformed via basis functions, then we need to adjust the number of parameters. Now, we have $\theta = (w, b) \in \mathbb{R}^n \times \mathbb{R}$. Thus, possible hypotheses are of the form

$$h_\theta(x) = \begin{cases} 1, & \text{if } w^\top \phi(x) + b \geq 0 \\ 0, & \text{if } w^\top \phi(x) + b < 0 \end{cases} \tag{2.19}$$

with associated decision boundaries $\{x \in \mathcal{X} \mid w^\top \phi(x) + b = 0\}$.

2.6.3 High-dimensional Data

In high-dimensional data, where the number of features greatly exceeds the number of observations (e.g., image data with thousands of pixels as features), logistic regression faces two main challenges:

- *Overfitting*: With so many features, the model tends to fit the noise in the data instead of capturing the true underlying patterns, resulting in poor generalization to new data.
- *Computational inefficiency*: Logistic regression involves solving optimization problems that become computationally expensive as the number of features increases, especially when regularization techniques (like ℓ_1 or ℓ_2 penalties) are applied to mitigate overfitting.

These issues make logistic regression less suitable for high-dimensional data, often requiring feature reduction techniques (e.g., principal component analysis) or alternative models like neural networks or tree-based methods.

2.6.4 Multi-class Classification

Simply put, logistic regression for binary classification predicts the probability of an outcome belonging to class $y = 1$ using the *sigmoid function*:

$$\mathbb{P}(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{(-z)}}, \quad \text{where } z = w^\top x + b \tag{2.20}$$

For binary classification, the predicted class is:

$$\hat{y} = \begin{cases} 1, & \text{if } \mathbb{P}(y = 1 \mid x) \geq 0.5, \\ 0, & \text{otherwise} \end{cases} \tag{2.21}$$

As we can see, logistic regression is naturally designed for binary classification, where the output is either 0 or 1. To handle multi-class classification, two common approaches are used:

- *One-vs-Rest (OvR)*: The multi-class problem is divided into multiple binary classification problems. Mathematically, for K-classes, we train K binary logistic regression models. For each class, a separate binary classifier is trained to distinguish that class from all others. So, each model k predicts $\mathbb{P}(y = k \mid x)$, i.e., whether the input belongs to class k or not:

  $$\mathbb{P}(y = k \mid x) = \frac{1}{1 + e^{(-z_k)}}, \quad \text{where } z_k = w_k^\top x + b_k \tag{2.22}$$

  And then during prediction, compute probabilities for all K models and assign the class with the highest probability:

  $$\hat{y} = \arg\max_k \mathbb{P}(y = k \mid x) \tag{2.23}$$

- *Softmax Regression (Multinomial Logistic Regression)*: This extends logistic regression by using the softmax function to compute probabilities for all classes simultaneously. Instead of separate models, softmax regression models all classes simultaneously. It predicts the probability for each class k using the softmax function:
$$\mathbb{P}(y = k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}, \quad \text{where } z_k = w_k^\top x + b_k \tag{2.24}$$
The model outputs a probability distribution over all classes, and the predicted class is the one with the highest softmax probability:
$$\hat{y} = \arg\max_k \mathbb{P}(y = k \mid x) \tag{2.25}$$

While OvR is simpler to implement, it requires training K separate models (one per class) and combines their outputs during prediction. Softmax regression, on the other hand, handles all classes in a single model but involves computing exponentials and normalizing over all classes, increasing computational cost. In both approaches, the objective function is typically cross-entropy loss, adapted for multi-class scenarios.

2.7 Role of Basis Functions in Logistic Regression

As we have seen in section 2.6, to address the limitation of linear decision boundaries, *basis functions* are introduced. These are transformations of the input features that allow logistic regression to model non-linear relationships(see [Bis06, chapter 4.3.2])

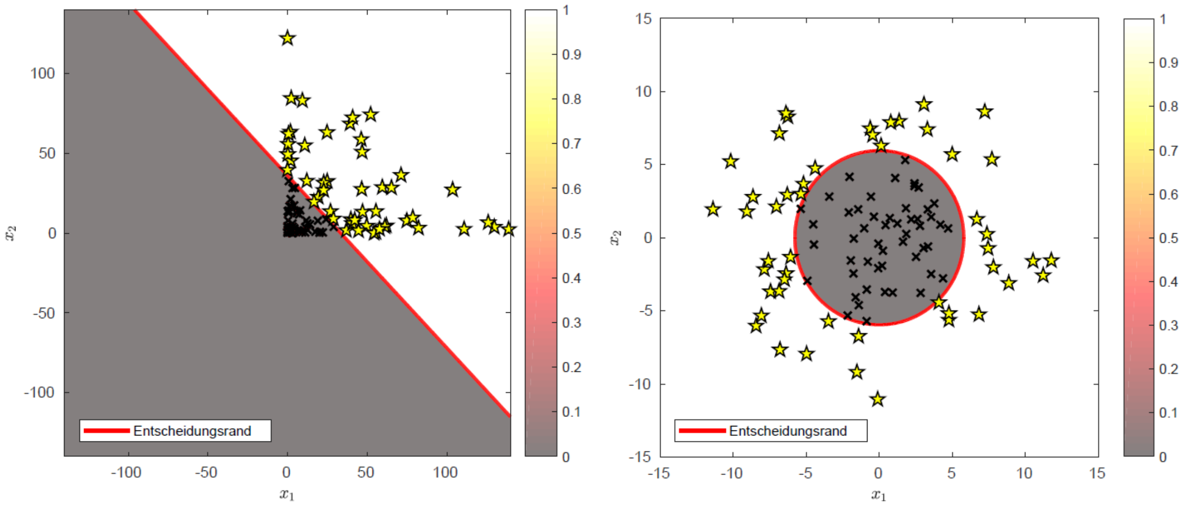

Figure 2.4: Example of logistic regression with basis functions applied to data, which is not linearly separable $\phi_1(x) = x_1^2$ and $\phi_2(x) = x_2^2$. The computed hypothesis is 100% accurate on the training data. The coloring represents the values of $\sigma(w^\top \phi(x) + b)$. On the left-hand side one can see the feature vectors $\phi^{(i)}$, which are linearly separated by the decision boundary $\{\phi \in \mathbb{R}^n \mid w^\top \phi + b = 0\}$. On the right-hand side one can see the original objects $x^{(i)}$, which are separated by the nonlinear decision boundary $\{x \in \mathbb{R}^d \mid w^\top \phi(x) + b = 0\}$ given by the basis functions.

Example 2.7.1 (Quadratic Basis Functions)**.** In Figure 2.4, we reconsider the example from Figure 2.1 (right-hand side). By visualization of the data, it is obvious that the training objects cannot be separated linearly by hyperplane according to their labels (stars vs. crosses). The depicted decision boundary is optimal with respect to a (standard) logistic regression performed on the training data. However, it classifies only 68% of the training data correctly. But if we (simply) introduce the basis functions $\phi_1(x) = x_1^2$ and $\phi_2(x) = x_2^2$, then the feature vectors $\phi^{(1)}, \ldots, \phi^{(m)}$ are linearly separable according to their labels and we find a corresponding linear hyperplane (w, b) via logistic regression; see left-hand side of the Figure 2.4. In the space $\mathcal{X}$ of objects this corresponds to a *nonlinear* separation. It holds that

$$h_\theta(x) = \begin{cases} 1, & \text{if } w_1x_1^2 + w_2x_2^2 + b \geq 0 \\ 0, & \text{if } w_1x_1^2 + w_2x_2^2 + b < 0 \end{cases} \tag{2.26}$$

The sets $x \in \mathcal{X} \mid h_\theta(x) = 0$ and $x \in \mathcal{X} \mid h_\theta(x) = 1$ are hence separated by a quadratic function; see right-hand side of the Figure 2.4.

This example was quite simple in the sense that we were able to predict by a simple visualization of the training data that they should be linearly separable by projecting the data into a higher-dimensional space by using a quadratic function (a circle). In general, one has to handle data in dimensions much larger than 2 or 3 and hence it is not always possible to make an educated preliminary guess about suitable basis functions ϕ_i.

Example 2.7.2 (Polynomial Basis Functions)**.** Since in real life, it is not always possible to guess a suitable basis functions to make the high-dimensional training data linearly separable, we resort to a general approach. A reasonable general approach is to assume that our objects are separable by the real zero set of a multivariate real polynomial of (total) degree κ (with κ sufficiently large). If our objects are located in a d-dimensional space, the resulting basis functions are monomials in d variables of degree at most κ. These are, for example, $x_1, x_2^{\kappa-2}x_3$ or $x_1x_2x_3$ (if $\kappa \geq 3$). The associated nonlinear functions are then polynomials

$$\sum_{j=1}^{d} w_j \underbrace{x_j}_{\phi_j(x)} + \sum_{j=1}^{d}\sum_{k=j}^{d} w_{jk} \underbrace{x_jx_k}_{\phi_{jk}(x)} + \sum_{j=1}^{d}\sum_{k=j}^{d}\sum_{l=k}^{d} w_{jkl} \underbrace{x_jx_kx_l}_{\phi_{jkl}(x)} + \cdots \tag{2.27}$$

of degree (at most) κ. While this approach is very general and therefore powerful, it has a crucial downside: the number of monomials in d variable of total degree at most κ, is

$$\binom{d+\kappa}{d} \models \frac{(d+1)\cdots(d+\kappa)}{\kappa!} = \mathcal{O}(d^\kappa) \tag{2.28}$$

While introducing basis functions solve the primary problem of handling linearly non-separable data, this approach has challenges:

- **Choosing the Right Basis Functions**: It requires domain knowledge or trial-and-error to select suitable transformations.
- **Dimensional Explosion**: The number of basis functions grows exponentially with the degree of the polynomial and the number of input features. For example, if we consider objects in dimension $d = 400$, e.g., in a small grey scale pictures with 20×20 pixels, and simply consider all monomials of degree at most $\kappa = 2$, then we already obtain 80601 basis functions.[HTF09]

2.8 Transition to Neural Networks

Artificial Neural Networks (ANNs) (see [Bis06, chapter 5] and [GBC16, chapter 6]) solve the limitations of logistic regression by *learning* the basis functions automatically using layers of neurons, instead of manually defining transformations. Thus, on the one hand, they allow us to avoid choosing basis functions by hand; on the other hand, we do not have to create gigantic polynomials as displayed before.

Using learned basis functions, neural networks compute very effective hierarchical representations of input data (i.e., hierarchically extract features from the data) such that we only need to consider a relatively small number of basis functions.

How ANNs Improve on Logistic Regression:

- **Automatic Feature Learning**: Neural networks learn non-linear transformations directly from the data. For example, convolutional layers in CNNs extract spatial patterns like edges and shapes.
- **Scalability**: ANNs can scale to millions of parameters, enabling them to model high-dimensional data efficiently.
- **Universal Approximation**: A neural network with sufficient depth and width can approximate any function, linear or non-linear.
- **Multi-Class Capability**: Using the **softmax activation function**, ANNs natively support multi-class classification tasks.

Thus, ANNs represent a significant advancement over traditional logistic regression, paving the way for breakthroughs in deep learning.

3 Artificial Neural Networks

We saw in section 2.6, that the practical applicability of linear models with fixed basis functions is limited. One alternative is choosing a fixed number of *adaptive basis functions*. Here, basis functions are parametrized in a suitable way, and the parameters are adapted during the training in such a way such that the model explains (or "tuned" to) the given data. This approach can be carried out via using *artificial neural networks*, which we define in what follows.

3.1 Fundamental Aspects

Definition 3.1.1 (Artificial Neural Network)**.** For a fixed $L \in \mathbb{N}$, let $n_0, \dots, n_L \in \mathbb{N}, W^{[\ell]} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, and $b^{[\ell]} \in \mathbb{R}^{n_\ell}$ for all $\ell = 1, \dots, L$. Furthermore, let $g^{[\ell]} : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell}$ for $\ell = 1, \dots, L$ be some functions. Then the function $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$, which maps an *input vector* $x \in \mathbb{R}^{n_0}$ to

$$
\begin{aligned}
f_\theta(x) &:= a^{[L]}, \quad \text{where} && (3.1)\\
a^{[0]} &:= x, && (3.2)\\
z^{[\ell]} &:= W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, && (3.3)\\
a^{[\ell]} &:= g^{[\ell]}(z^{[\ell]}), \Big\} \quad \ell = 1, \dots, L && (3.4)
\end{aligned}
$$

is called an *artificial neural network* with *parameters* $\theta = (W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$ and *activation functions* $g^{[1]}, \dots, g^{[L]}$. The matrices $W^{[1]}, \dots, W^{[L]}$ are called *weight matrices*, and the vectors $b^{[1]}, \dots, b^{[L]}$ are called *bias vectors* of f.

For fixed $x \in \mathbb{R}^n_0$ and $\ell \in \{1, \dots, L\}$, we call $z^{[\ell]} \in \mathbb{R}^{n_\ell}$ *(net) input vector)* and $a^{[\ell]} \in \mathbb{R}^{n_\ell}$ the *activation vector)* in the ℓ-th layer, where n_ℓ is the *width)* of the ℓ-th layer. Finally, $f_\theta(x) = a^{[L]}$ is the *output vector)*, and L is called the *depth)* of f_θ.

Remark 3.1.1 (Nonlinearity)**.** The vectors $a^{[0]}, \dots, a^{[L]}$ and $z^{[1]}, \dots, z^{[L]}$ defined in 3.4 are, like f_θ, functions of the input vector x. For simplicity, we usually omit this dependency in our notation as long as the context is clear. Nevertheless, we can understand f_θ as a composition of several linear and nonlinear functions. For example, for $L = 1$ we have

$$f_\theta(x) = g^{[1]}(W^{[1]} x + b^{[1]}), \tag{3.5}$$

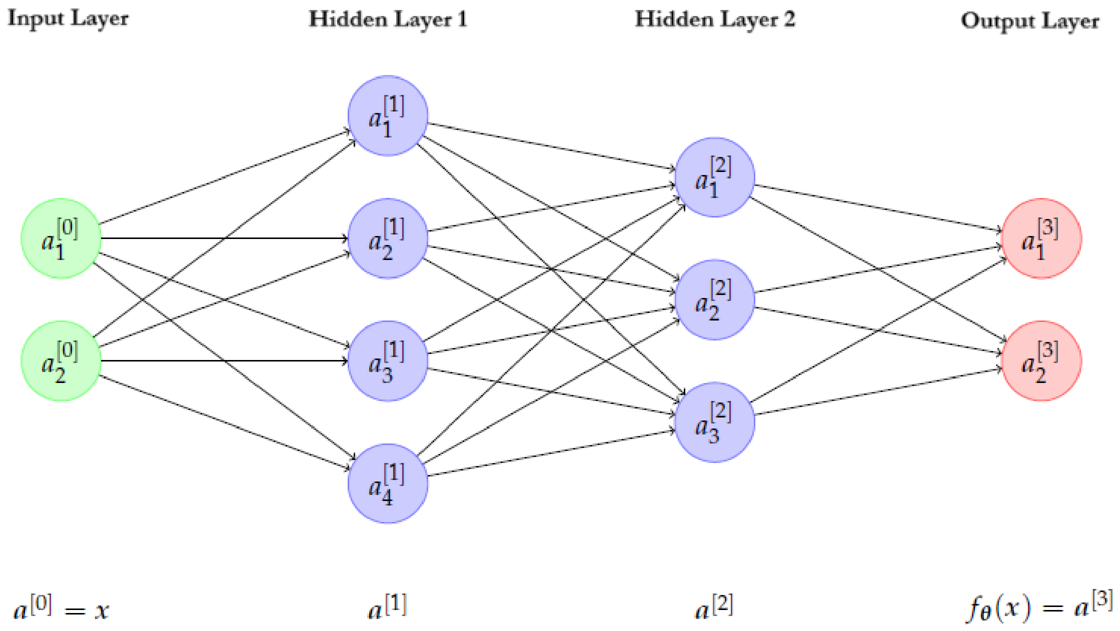

Figure 3.1: Example of an artificial neural network with two hidden layers.

meaning that f_θ is a composition of $g^{[1]}$ with $z^{[1]} = W^{[1]}x + b^{[1]}$. Also for $L = 2$ we obtain a function

$$f_\theta(x) = g^{[2]}(W^{[2]}(g^{[1]}(W^{[1]}x + b^{[1]})) + b^{[2]}), \tag{3.6}$$

and the process continues in the same way for $L \geq 3$.

In general at least some of the functions $g^{[\ell]}$ are chosen to be nonlinear. If this was not the case, then f_θ, as a composition of linear functions, would also be linear. The additional and more complex structure 3.4 would therefore have no advantage over a simple representation of the form $f_\theta(x) = Wx + b$ with $W \in \mathbb{R}^{n_L \times n_0}$ and $b \in \mathbb{R}^{n_L}$ (as every affine-linear function can be represented in such a way).

Remark 3.1.2 (Scalar Activation Functions)**.** The activation function $g^{[\ell]} : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell}$ can often be interpreted as a component-wise application of a real function from $\mathbb{R}$ to $\mathbb{R}$. In this case, unless stated differently, for a scalar function $g^{[\ell]} : \mathbb{R} \to \mathbb{R}$, we define $g^{[\ell]}(z) \in \mathbb{R}^{n_\ell}$ as a component-wise application of $g^{[\ell]}$ to $z \in \mathbb{R}^{n_\ell}$,

$$g^{[\ell]}(z) := \begin{pmatrix} g^{[\ell]}(z_1) \\ \vdots \\ g^{[\ell]}(z_{n_\ell}) \end{pmatrix}. \tag{3.7}$$

When defining a neural network, we decide on the type of activation functions that we use. Below we give a few example of commonly used activation functions.

Example 3.1.1 (Typical Activation Functions)**.** Depending on the context and goal of the problem, we can use any of the below activation functions (see Figure 3.2) while defining the neural network:

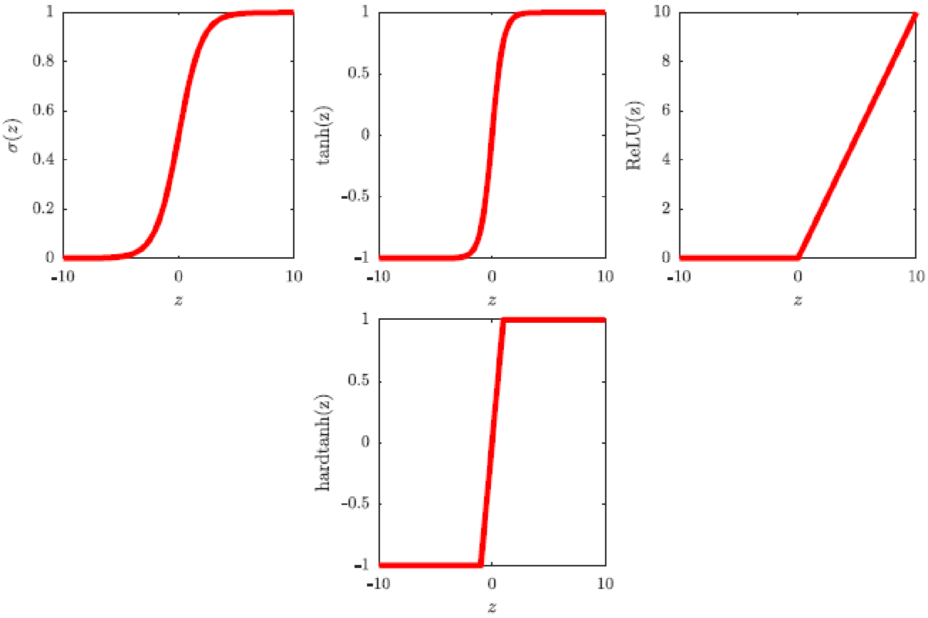

Figure 3.2: Typical Activation Functions

- *Sigmoid function*: Commonly used in the output layer of a binary classification task, as it maps values to the range $(0, 1)$, representing probabilities. Suitable when the model needs to output probabilities or interpret values as likelihoods, but it can suffer from vanishing gradients.[HS97, RHW86]

$$\sigma : \mathbb{R} \to (0,1), \quad \sigma(z) := \frac{1}{1+e^{-z}} \tag{3.8}$$

- *ReLU (Rectified Linear Unit) function*: It ensures sparsity by retaining only positive activations, which not only introduces non-linearity but also improves computational efficiency and helps avoid the vanishing gradient problem.[NH10]

$$\mathrm{ReLU}(z) : \mathbb{R} \to \mathbb{R}_{\geq 0}, \quad \mathrm{ReLU}(z) := \max(0, z) \tag{3.9}$$

- *Hyperbolic Tangent (tanh) function*: Preferred in hidden layers when inputs are zero-centered, as it maps the input in the range $(-1, 1)$, which helps faster convergence during training. It has more efficient gradient flow than sigmoid function.

$$\tanh : \mathbb{R} \to (-1,1), \quad \tanh(z) := \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.10}$$

- *Hard hyperbolic tangent (hardtanh) function*: Provides a faster approximation of the traditional tanh function, maintaining a balance between non-linearity and computational simplicity.

$$\mathrm{hardtanh}(z) : \mathbb{R} \to [-1,1], \quad \mathrm{hardtanh}(z) := \max(-1, \min(1, z)) \tag{3.11}$$

- *Identity function*: While it does not introduce non-linearity, it is occasionally used in skip connections or output layers when no transformation is needed.

$$\mathrm{id}(z) : \mathbb{R} \to \mathbb{R}, \quad \mathrm{id}(z) := z \tag{3.12}$$

- *Softmax function*: Typically applied at the output layer for multi-class classification tasks, converting raw scores into normalized probabilities.

$$\mathrm{softmax}(z) : \mathbb{R}^n \to \mathbb{R}^n, \quad \mathrm{softmax}(z)_j := \frac{e^{z_j}}{\sum_{j=1}^{n} e^{z_j}} \tag{3.13}$$

3.2 Multiclass Classification

Assume that, in contrast to binary classification, we have not only two but K different classes that we want to assign to elements $x \in \mathcal{X}$ of the sample space. Thus, we have the set $\{0, \dots, K-1\}$ of K different labels and we seek a *hypothesis* of the form $h_\theta : \mathbb{R}^d \to \{0, \dots, K-1\}$. We realize this approach via the *One-Hot encoding*

$$\{0, \dots, K-1\} \to \mathbb{R}^K, \quad k \mapsto e_{k+1} \tag{3.14}$$

where e_{k+1} denotes the $(k+1)$-st unit vector. We map every label $y^{(i)} \in \{0, \dots, K-1\}$ of an element of the training dataset to the corresponding unit vector:

$$y^{(i)} = e_{y^{(i)}+1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{position } y^{(i)} + 1 \in \mathbb{R}^K. \tag{3.15}$$

We then use the training data $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}) \subseteq \mathcal{X} \times \{0,1\}^K \subseteq \mathbb{R}^d \times \{0,1\}^K$ and a neural network with input dimension $n_0 = d$ and output dimension $n_L = K$. Further, we determine the depth L, widths $n_1, \dots, n_{L-1}$ of each layer, suitable activation functions $g^{[1]}, \dots, g^{[L-1]}$ and choose $g^{[L]} := \mathrm{softmax}$ to be 3.13.

Definition 3.2.1 (Softmax Loss)**.** The function

$$\ell : \mathbb{R}^K \times \mathbb{R}^K_{>0} \to \mathbb{R}, \quad \ell(y, \hat{y}) := -\sum_{k=1}^{K} y_k \ln(\hat{y}_k) \tag{3.16}$$

is called *softmax loss function*.

Using the softmax loss function, we will train an artificial neural network $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ by applying an optimization algorithm to the problem:

$$\min_{\theta \in \Theta} -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \ln(\hat{y}^{(i)_k}), \tag{3.17}$$

where $\theta = (W^{(1)}, b^{(1)}, \dots, W^{(L)}, b^{(L)})$ ranges in the set Θ of parameters of the underlying artificial neural network. To identify the output of f_θ for the i-th training sample, we write $\hat{y}^{(i)} := f_\theta(x^{(i)})$ in 3.17. For an object x with unknown label and output $\hat{y} = f_\theta(x)$ we then assign the class

$$h_\theta(x) = \left[\operatorname*{arg\,max}_{k=1,\dots,K} \ \hat{y}_k\right] - 1 \tag{3.18}$$

The softmax function defined in 3.13 represents a probability distribution over a discrete set. This makes it particularly suitable for the role of the output activation function $g^{[L]}$ for multiclass classification. For a fixed input vector $x \in \mathbb{R}^d$ and the associated input vector $z^{[L]}$ of the L-th layer (output layer), we interpret

$$\hat{y}_k = \operatorname{softmax}(z^{[L]})_k = \mathcal{D}_{\text{Mod}}(y = k-1 \mid x; \theta), \tag{3.19}$$

which explains why we choose the class with the highest probability in 3.18.

For $L = 1$ and $z^{[1]} = (-(w^\top x + b), 0)$, we obtain

$$\hat{y} = \operatorname{softmax}(z^{[L]}) = \big(1 - \sigma(w^\top x + b), \sigma(w^\top x + b)\big) \tag{3.20}$$

and we see that the Bernoulli distribution 2.3 arises as a particular case of 3.19. In particular, logistic regression 2.7 is a very special instance of the multiclass classification (with $L = 1$ and 3.13 as an output activation function) given by 3.18.

Remark 3.2.1 (Maximum Likelihood Interpretation)**.** As for the cross entropy loss for binary classification, the softmax loss for multiclass classification can be derived via a maximum likelihood approach, where the statistical model is the family of distributions on $\mathcal{X} \times \{0,1\}^K \subseteq \mathbb{R}^d \times \{0,1\}^K$ given by 3.19. Using the vector $y \in \{e_1, \dots, e_K\} \subseteq \mathbb{R}^K$, which is a one-hot encoded label $y \in \{0, \dots, K-1\}$, we write 3.19 as

$$\mathcal{D}_{\text{Mod}}(y|x;\theta) = \mathcal{D}_{\text{Mod}}(y|x;\theta) := \prod_{k=1}^{K} [f_\theta(x)_k]^{y_k} \tag{3.21}$$

Then

$$\mathcal{L}(\theta) = -\ln \mathcal{D}_{\text{Mod}}(S;\theta) = -\ln \prod_{i=1}^{m} \prod_{k=1}^{K} \left[f_\theta(x^{(i)})_k\right]^{y_k^{(i)}} = -\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \ln f_\theta(x^{(i)})_k \tag{3.22}$$

is a negative log likelihood function, where the inner sum corresponds exactly to $\ell(y^{(i)}, f_\theta(x^{(i)}))$ from 3.16.

Example 3.2.1 (MNIST)**.** A dataset often used as a benchmark in the context of multiclass classification is the MNIST dataset ([LC98, LBBH98]), containing over 60000 training examples $(x^{(i)}, y^{(i)}) \in \mathbb{R}^{28\times 28} \times \{0, \dots, 9\}$. Every object $x^{(i)}$ is a grayscale image of a handwritten digit (Figure 3.3) and the associated label $y^{(i)}$ corresponds exactly to the respective digit. In order to use the images $x^{(i)}$ (available as matrices) as an input for a neural network, they must be vectorized (usually row-wise), such that we have an input dimension of $n_0 = 28^2 = 784$ and $n_L = 10$ as output dimension. In this example we use a neural network with depth $L = 3$ and two hidden layers of width $n_1 = 100$ and $n_2 = 50$. Additionally, we apply *batch normalization* and *dropout*, two techniques allowing for a

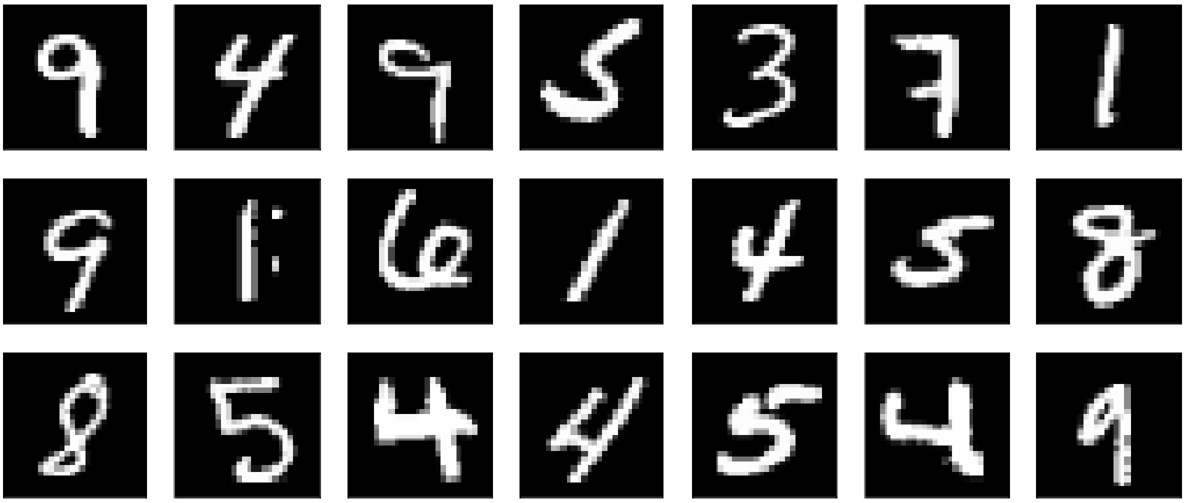

Figure 3.3: Random sampled objects from the MNIST dataset

more efficient training and improving the model performance on unseen data. We will later discuss these techniques in detail. With a sufficient number of gradient steps optimizing the problem 3.17, we can determine the parameters $\hat{\theta} = (W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]})$, and obtain a classification accuracy of 98.37% on the training data and 98.38% on 10000 unseen test images. IN Figure 3.4, we visualize a subset of rows of the matrix $W^{[1]}$ after training. There we can see that some rows extract certain patterns from the input data, like line segments with a certain orientation on fixed positions in the image. The activation vector $a^{[1]}$ for a fixed input x thus contains information about the presence (or absence) of these simple geometric patterns. These information are combined in the second hidden layer to detect more complex patterns in the input images, which are encoded in the activation vector $a^{[2]}$. Based on these extracted patterns, the output layer then assigns the probability of the image belonging to each class.

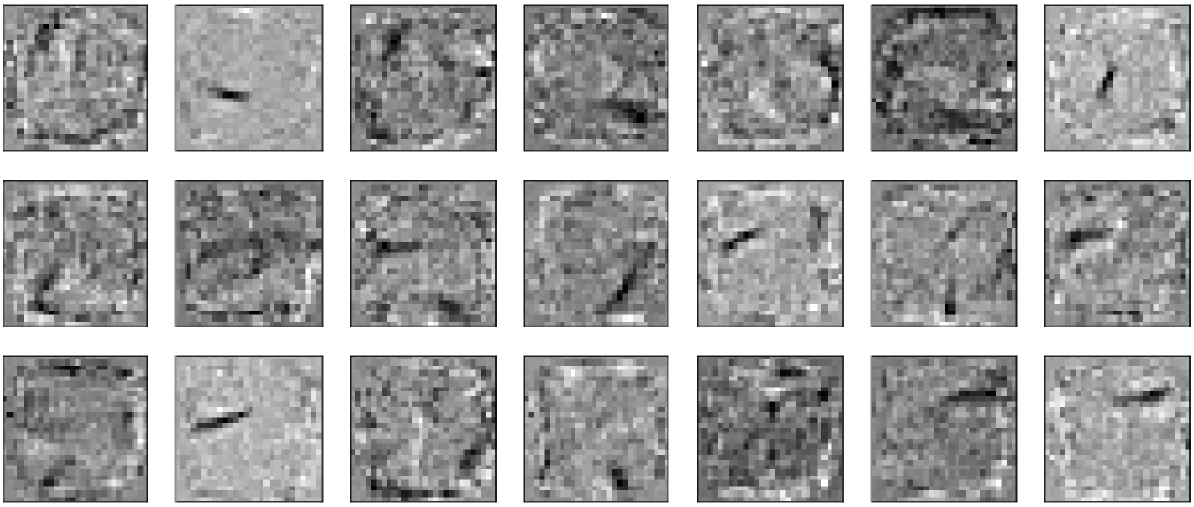

Figure 3.4: Subset of rows of the weight matrix $W^{[1]}$.The length of each row is equal to the input dimension $n_0 = 784$. Here, the rows are displayed as 28×28 grayscale images, which is the original format of the images $x^{(i)}$. Light pixels tend to correspond to positive entries, while dark pixels corresponds to negative entries and gray pixels to small absolute values.

3.3 Error Backpropagation

In order to solve the optimization problem

$$\hat{\theta} \in \arg\min_{\theta\in\Theta} \mathcal{J}(\theta) = \arg\min_{\theta\in\Theta} \frac{1}{m}\sum_{i=1}^{m} \ell(y^{(i)}, f_\theta(x^{(i)})) \tag{3.23}$$

for a loss function ℓ and an artificial neural network f_θ with a *gradient descent method*, we need to compute partial derivatives of $\ell(y^{(i)}, f_\theta(x^{(i)}))$ along the parameters $\hat{\theta} = (W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$. To compute a derivative of $\mathcal{J}$ (for a fixed training data set) we simply have to sum and divide by m the derivatives of $\ell(y^{(i)}, f_\theta(x^{(i)}))$. In the following, we derive the *backpropagation* algorithm used for this purpose (see for example [GBC16, section 6.5], [Bis06, section 5.3]).

3.3.1 Derivatives for one Object on Parameter Basis

Lemma 3.3.1 (Partial derivatives with respect to parameters)**.** Let $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ be an artificial neural network with parameters $\hat{\theta} = (W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$, $\ell : \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \to \mathbb{R}$, a loss function, and $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$. If we define for all $\ell \in \{1, \dots, L\}$ and all $j \in \{1, \dots, n_\ell$

$$\delta_j^{[\ell]} := \frac{\partial \ell(y, f_\theta(x))}{\partial z_j^{[\ell]}}, \tag{3.24}$$

then it holds

$$\frac{\partial \ell(y, f_\theta(x))}{\partial w_{ji}^{[\ell]}} = \delta_j^{[\ell]} a_i^{[\ell-1]} \quad \text{and} \quad \frac{\partial \ell(y, f_\theta(x))}{\partial b_j^{[\ell]}} = \delta_j^{[\ell]}. \tag{3.25}$$

Proof. The value of the objective function $\ell(y, f_\theta(x))$ depends exclusively on the weights $w_{ji}^{[\ell]}$ via the relation

$$z_j^{[\ell]} = \sum_{i=1}^{n_{\ell-1}} w_{ji}^{[\ell]} a_i^{[\ell-1]} + b_j^{[\ell]}. \tag{3.26}$$

Therefore, we can apply the chain rule for the derivative and obtain

$$\frac{\partial \ell(y, f_\theta(x))}{\partial w_{ji}^{[\ell]}} = \frac{\partial \ell(y, f_\theta(x))}{\partial z_j^{[\ell]}} \cdot \frac{\partial z_j^{[\ell]}}{\partial w_{ji}^{[\ell]}} = \delta_j^{[\ell]} a_i^{[\ell-1]}. \tag{3.27}$$

Analogously, we obtain the derivative with respect to $b_j^{[\ell]}$ with the difference that $\frac{\partial z_j^{[\ell]}}{\partial b_j^{[\ell]}} = 1$.

Lemma 3.3.1 states that we can determine the derivatives of $\ell(y, f_\theta(x))$ with respect to the weights and biases with only little effort, if we know the derivatives of the respective inputs.

Theorem 3.3.1 (Backpropagation)**.** Let $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ be an artificial neural network with parameters $\hat{\theta} = (W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$, loss function $\ell : \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \to \mathbb{R}$ and $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$. Furthermore, let the activation functions $g^{[1]}, \dots, g^{[L-1]} : \mathbb{R} \to \mathbb{R}$ be defined component-wise. Then it holds for every $j \in \{1, \dots, n_L\}$

$$\delta_j^{[L]} = \sum_{k=1}^{n_L} \frac{\partial \ell(y, f_\theta(x))}{\partial a_k^{[L]}} \cdot \frac{\partial a_k^{[L]}}{\partial z_j^{[L]}} \tag{3.28}$$

and further for every $\ell \in \{1, \dots, L-1\}$ and every $j \in \{1, \dots, n_\ell\}$

$$\delta_j^{[\ell]} = g^{[\ell]\prime}(z_j^{[\ell]}) \sum_{k=1}^{n_{\ell+1}} \delta_k^{[\ell+1]} w_{kj}^{[\ell+1]}. \tag{3.29}$$

Proof. The first claim 3.28 follows directly from the definition 3.24 and the chain rule because the function $\ell(y, f_\theta(x))$ only depends on $z_j^{[L]}$ through $a^{[L]}$. Thus

$$\delta_j^{[L]} = \frac{\partial \ell(y, f_\theta(x))}{\partial z_j^{[L]}} = \underbrace{\frac{\partial \ell(y, f_\theta(x))}{\partial a^{[L]}}}_{1 \times n_L} \cdot \underbrace{\frac{\partial a^{[L]}}{\partial z_j^{[L]}}}_{n_L \times 1} = \sum_{k=1}^{n_L} \frac{\partial \ell(y, f_\theta(x))}{\partial a_k^{[L]}} \cdot \frac{\partial a_k^{[L]}}{\partial z_j^{[L]}}. \tag{3.30}$$

For the second claim, we can also apply the chain rule as follows:

$$\begin{aligned} \delta_j^{[\ell]} &= \frac{\partial \ell(y, f_\theta(x))}{\partial z_j^{[\ell]}} = \underbrace{\frac{\partial \ell(y, f_\theta(x))}{\partial z^{[\ell+1]}}}_{1 \times n_{\ell+1}} \cdot \underbrace{\frac{\partial z^{[\ell+1]}}{\partial z_j^{[\ell]}}}_{n_{\ell+1} \times 1} \\ &= \sum_{k=1}^{n_{\ell+1}} \underbrace{\frac{\partial \ell(y, f_\theta(x))}{\partial z_k^{[\ell+1]}}}_{\delta_k^{[\ell+1]}} \cdot \frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}} = \sum_{k=1}^{n_{\ell+1}} \delta_k^{[\ell+1]} \cdot \underbrace{\frac{\partial z_k^{[\ell+1]}}{\partial a_j^{[\ell]}}}_{= w_{kj}^{[\ell+1]}} \cdot \underbrace{\frac{\partial a_j^{[\ell]}}{\partial z_j^{[\ell]}}}_{= g^{[\ell]\prime}\left(z_j^{[\ell]}\right)} \\ &= g^{[\ell]\prime}\left(z_j^{[\ell]}\right) \sum_{k=1}^{n_{\ell+1}} \delta_k^{[\ell+1]} w_{kj}^{[\ell+1]}. \end{aligned} \tag{3.31}$$

Remark 3.3.1 (Activation functions in the output layer)**.** In Theorem 3.3.1, we assumed that the activation functions $g^{[1]}, \dots, g^{[L-1]} : \mathbb{R} \to \mathbb{R}$ are defined component-wise, while we intentionally omitted this assumption for the activation function $g^{[L]}$ in the last layer. The reason for that is that the function softmax $: \mathbb{R}^n \to \mathbb{R}^n$ typically used as an activation in the last layer is not defined component-wise. If, in contrast, $g^{[L]} : \mathbb{R} \to \mathbb{R}$ is also defined component-wise, we obtain a simpler representation of (3.28),

$$\delta_j^{[L]} = \frac{\partial \ell(y, f_\theta(x))}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = g^{[L]\prime}\left(z_j^{[L]}\right) \cdot \frac{\partial \ell(y, f_\theta(x))}{\partial a_j^{[L]}} \tag{3.32}$$

3.3.2 Vectorized Derivatives for one Object

With Lemma 3.3.1 and Theorem 3.3.1 we are now able to compute the derivatives of $\ell(y, f_\theta(x))$ with respect to $w_j i^{[\ell]}$ and $b_j^{[\ell]}$ for a fixed training example (x, y). To achieve a more efficient implementation of the backpropagation it is useful to compute these derivatives not with respect to every single parameter but directly with respect to the whole weight matrix $W^{[\ell]}$ and bias vector $b^{[\ell]}$. For that, we *vectorize* the derivation process.

Definition 3.3.1 (Error)**.** For a fixed $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$ and $\ell \in \{1, \dots, L\}$ we define

$$\delta^{[\ell]} := \begin{pmatrix} \delta_1^{[\ell]} \\ \vdots \\ \delta_{n_\ell}^{[\ell]} \end{pmatrix} \in \mathbb{R}^{n_\ell} \tag{3.33}$$

as the *error* of the ℓ-th layer.

To calculate $\delta^{[\ell]}$, we need to distinguish the two cases $\ell = L$ and $\ell < L$, as we already did in Theorem 3.3.1. In the case $\ell = L$ we can directly expand the definition to

$$\begin{aligned}\delta^{[L]\top} &= \frac{\partial \ell(y, f_\theta(x))}{\partial z^{[L]}} = \frac{\partial \ell(y, f_\theta(x))}{\partial a^{[L]}} \cdot \frac{\partial a^{[L]}}{\partial z^{[L]}} \\ &= \underbrace{\begin{pmatrix} \frac{\partial \ell(y,f_\theta(x))}{\partial a_1^{[L]}} \cdots \frac{\partial \ell(y,f_\theta(x))}{\partial a_{n_L}^{[L]}} \end{pmatrix}}_{1\times n_L} \underbrace{\begin{pmatrix} \frac{\partial a_1^{[L]}}{\partial z_1^{[L]}} & \cdots & \frac{\partial a_1^{[L]}}{\partial z_{n_L}^{[L]}} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_{n_L}^{[L]}}{\partial z_1^{[L]}} & \cdots & \frac{\partial a_{n_L}^{[L]}}{\partial z_{n_L}^{[L]}} \end{pmatrix}}_{n_L\times n_L}\end{aligned} \tag{3.34}$$

By analogy with (3.32) we obtain,

$$\delta^{[L]} = g^{[L]'}\left(z^{[L]}\right) \odot \left[\frac{\partial \ell(y, f_\theta(x))}{\partial a^{[L]}}\right]^\top = g^{[L]'}\left(z^{[L]}\right) \odot \nabla_{a^{[L]}} \ell(y, f_\theta(x)) \tag{3.35}$$

for a component-wise defined $g^{[L]} : \mathbb{R} \to \mathbb{R}$, where $g^{[L]'} : \mathbb{R} \to \mathbb{R}$ is also applied component-wise to $z^{[L]}$. Here, with $\odot$ we denote the component-wise multiplication of two vectors. Note that in this case $\partial a^{[L]}/\partial z^{[L]}$ is a diagonal matrix with entries $g^{[L]'}\left(z^{[L]}\right), j = 1, \ldots, n_L$.

Remark 3.3.2. In several cases of interest (e.g., when $g^{[L]}$ is the softmax function (3.13)) it holds

$$\delta^{[L]} = f_\theta(x) - y \tag{3.36}$$

Now consider the second case $\ell < L$. With (3.29) we can directly see that

TO BE CONTINUED...

Bibliography

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[BTD$^+$16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End-to-end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[EKN$^+$17] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2012.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[LC98] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. *Technical Report*, 1998. Available at http://yann.lecun.com/exdb/mnist/.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagation errors. *Nature*, 323:533–536, 1986.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.