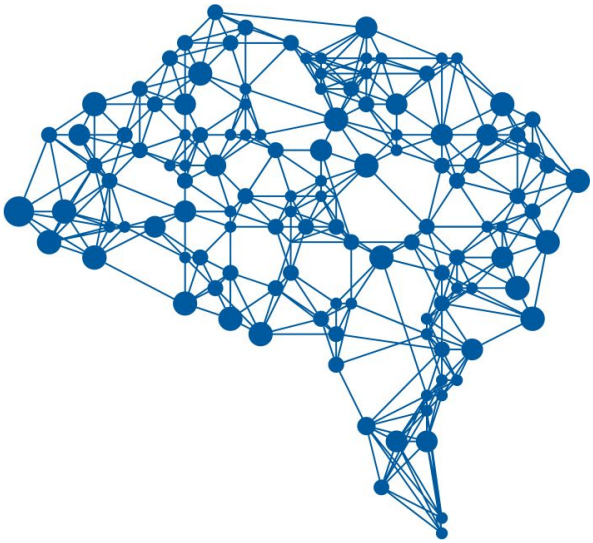


# Deep Learning

Mingming Li

First Created: September 29, 2020
Last Modified: February 27, 2023

# Contents

# List of Figures

# List of Tables

# Part I

# Theory

Chapter 1

# Learning

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell provides a succinct definition: "A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$"

Now, how does the learn happen? The model or algorithm learns by adjusting the parameters contained in it.

How to adjust the parameters? Refer to Chapter 3.

Chapter 2

# Model

What is a model?

You can think a model as a function in math. For example,

$$ax + by = c$$

In this function, $a, b$ and $d$ are the parameters, $x$ and $y$ are input from data. Through adjusting the parameters $a, b$ and $d$, we can implement the learning process. This is also called training in machine learning.

## 2.1 Capacity

The capacity is the function space. It defines the limitation that can learn from data.

For example,

$$ax + by = c$$

This function can only learn linear function from data.

$$ax_1^2 + bx_2 = c$$

This function can learn curve function from data.

### 2.1.1 The no free lunch theorem

For any algorithms (functions) $a_1$ and $a_2$, at iteration step $m$

$$\sum P(d_m^y | f, m, a_1) = \sum P(d_m^y | f, m, a_2) \tag{2.1}$$

where $d_m^y$ denotes the ordered set of size $m$ of the cost values $y$ associated to input values $x \in X$, $f : X \longrightarrow Y$ is the function being optimized and $P(d_m^y | f, m, a)$ is the conditional probability of obtaining a given sequence of cost values from algorithm $a$ run $m$ times on function $f$.

The no free lunch theorem implies that we must design our machine learning algorithms to perform well on a specific task but not a universal task.

### 2.1.2 Universal approximation theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $R^n$ , under mild assumptions on the activation function.

Let $\varphi : \mathrm{R} \to \mathrm{R}$ be a nonconstant, bounded, and continous functions. Let $I_m$ denote the m-dimensional unit hypercube $[0, 1]^m$ . The space of real-value continous function on $I_m$ is denoted by $C(I_m)$ . Then, given any $\varepsilon > 0$ and function $f \in C(I_m)$ , there exist an integer N, real constants $\upsilon_i, b_i \in R$ and real vectors $\omega_i \in R^m$ for $i = 1, \cdots, N$ , such that we may define:

$$F(x) = \sum_{i=1}^{N} \upsilon_i \varphi(\omega_i^T x + b_i) \tag{2.2}$$

as an approximate realization of the function f; that is

$$|F(x) - f(x)| < \varepsilon \tag{2.3}$$

for all $x \in I_m$ . In other words, functions of the form $F(x)$ are dense in $C(I_m)$ .

This still holds when replacing $I_m$ with any compact subset of $R^m$ .

Kurt Hornik showed in 1991 that it is not the specific choice of the activation function, but rather the multi-layer feedforward architecture itself which gives neural networks the potential of being universal approximators. The output units are always assumed to be linear. For notational convenience, only the single output case will be shown. The general case can easily be deduced from the single output case.

In 2017 Lu et al. proved universal approximation theorem for width-bounded deep neural networks.

This is the base of deep learning.

## 2.2 Overfitting and underfitting

We train model on training data but use test data (not used to train the model) to test out model. The ability to perform on test data is called **generalization**. We can use model on test data because we assume that the train data and the test has the same probability distribution (i.e. they have relationship).

The error on training data is called **training error**. The error on test data is called **test error**. Underfitting occurs when the model is not able to obtain a sufficient low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.

We can control whether a model is more likely to overfit or underfit by altering its **capacity**. The overfitting and underfitting happen because of the mismatch of capacity and the data.

Chapter 3

# Training (optimization)

Training is the process of learning. By inputting the data, we adjust the parameters in the model to achieve better performance. In machine learning this is also called the optimization.

## 3.1 SGD

Stochastic gradient descent (SGD) is a very basic and important algorithm.

The cost function used by a machining learning algorithm often decomposes as a sum over training examples of some per-example loss function.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathrm{x},y\sim\hat{p}_{data}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}), \tag{3.1}$$

Where $L$ is the loss function, $x$ and $y$ are the input data and labels, $\theta$ is the parameters in the model, $m$ is the number of samples.

The gradient is

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}). \tag{3.2}$$

The computational cost of this operation is $O(m)$. As the training set size grows to billions of examples, the time to take a single gradient step becomes prohibitively long.

The insight of SGD is that the gradient is an expectation. The expectation may be approximately estimated using a small set of samples. Specifically, on each step of the algorithm, we can sample a **minibatch** of examples $\mathbb{B} = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ drawn uniformly from the training set. The minibatch size $m'$ is typically chosen to be a relatively small number of examples, ranging from one to a few hundred.

The estimator of the gradient is formed as

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{3.3}$$

using examples from the minibatch $\mathbb{B}$. The stocastic gradient descent algorithm then follows the estimated gradient downhill:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}, \tag{3.4}$$

where $\epsilon$ is the learning rate.

## 3.2 Momentum

The method of momentum is designed to accelerate learning in SGD. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g} \tag{3.5}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{3.6}$$

Where $g$ is the gradient, $\alpha \in [0, 1]$ determines how quickly the contributions of previous gradients exponentially decay.

## 3.3 Nesterov momentum

Nesterov Momentum is a variant of the Momentum algorithm. The difference between Nesterov momentum and standard momentum is where the gradient is evaluated. With Nesterov momentum the gradient is evaluated after the current velocity is applied.

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}, y, \boldsymbol{\theta} + \alpha \boldsymbol{v}) \tag{3.7}$$

## 3.4 AdaGrad

AdaGrad is designed to converge rapidly when applied to a convex function. Comparing to SGD, AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\delta \boldsymbol{I} + \operatorname{diag}(G)}} \odot \boldsymbol{g} \tag{3.8}$$

where $\boldsymbol{\theta}$ is the parameter to be updated, $\epsilon$ is the initial learning rate, $\delta$ is some small quantity that used to avoid the division of zero, $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{g}$ is the gradient estimate.

$$\boldsymbol{G} = \sum_{\tau=1}^{t} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^T \tag{3.9}$$

AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure.

## 3.5 Adam

The RMSProp algorithm modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average. RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl.

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \tag{3.10}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g} \tag{3.11}$$

Where $\boldsymbol{g}$ is the gradient, $\boldsymbol{\theta}$ is the parameters in a model, $\boldsymbol{r}$ is initialized to 0..

Chapter 4

# Cost function

**Loss** is the difference between the predicted value and the actual value. The Function used to quantify this loss during the training phase in the form of a single real number is known as **Loss Function**. **Cost function** refers to an average of the loss functions over an entire training dataset. Cost function helps us reach the optimal solution. The cost function is the technique of evaluating "the performance of our algorithm/model".

There are many cost functions in machine learning and each has its use cases depending on whether it is a regression problem or classification problem.

## 4.1 Regression cost function

Regression models deal with predicting a continuous value. They are calculated on the distance-based error.

$$\text{Error} = y - y' \tag{4.1}$$

### 4.1.1 Mean squared error

$$\text{MSE} = \frac{\sum_{i=0}^{n}(y - y')^2}{n} \tag{4.2}$$

### 4.1.2 Mean absolute error

$$\text{MSE} = \frac{\sum_{i=0}^{n}|y - y'|}{n} \tag{4.3}$$

## 4.2 Classification cost function

In classification, we usually use one hot to encoder the labels. This can eliminate the affect of the distance.

The cross entropy of the distribution $q$ relative to distribution $p$ over a given set is defined as

$$H(p, q) = -E_p[\log q] \tag{4.4}$$

Where the $E$ is the expected value operator respected to the probability $q$.

For discrete probability distribution $p$ and $q$ with the same support $\mathcal{X}$ this means

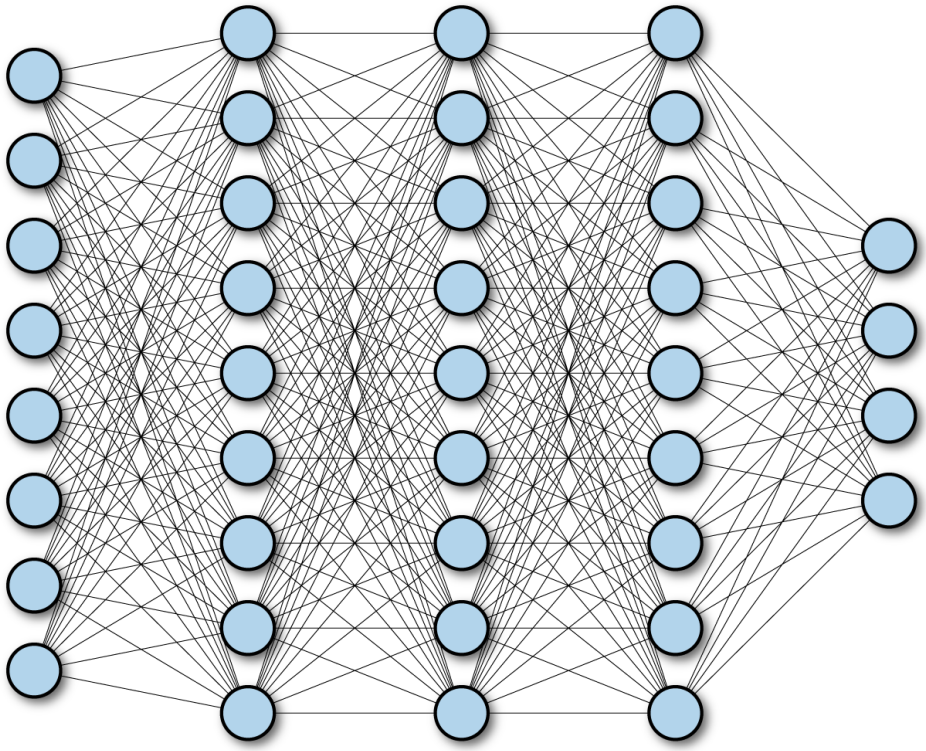
$$H(P, Q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x) \tag{4.5}$$

Chapter 5

# Full Connected Networks

A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer. Full connected neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means that every output unit interacts with every input unit.

Figure 5.1 show the full connects neural network.



**Figures 5.1:** Full connected neural network

Chapter 6

# CNN

CNN stands for convolutional neural network. Convolutional networks are neural networks that have convolutional layers. A typical convolutional layer consists of three stages:

1. convolution stage: affine transform
2. detector stage: nonlinearty
3. pooling stage

## 6.1 Convolution

$$s(t) = \int x(a)w(t-a)da. \tag{6.1}$$

This operation is called **convolution**. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t). \tag{6.2}$$

In convolutional network terminology, the first argument (in this example, the function $x$) to the convolution is often referred to as the **input**, and the second argument (int this example, the function $w$) as the **kernel**. The output is sometimes referred to as the **feature map**.

If we assume that $x$ and $w$ are defined only on integer $t$, we can define the discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{6.3}$$

We often use convolutions over more than one axis at a time. For example, if we use a two-dimensinal image $I$ as our input, we probably also want to use a two-dimensional kernel $K$:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{6.4}$$

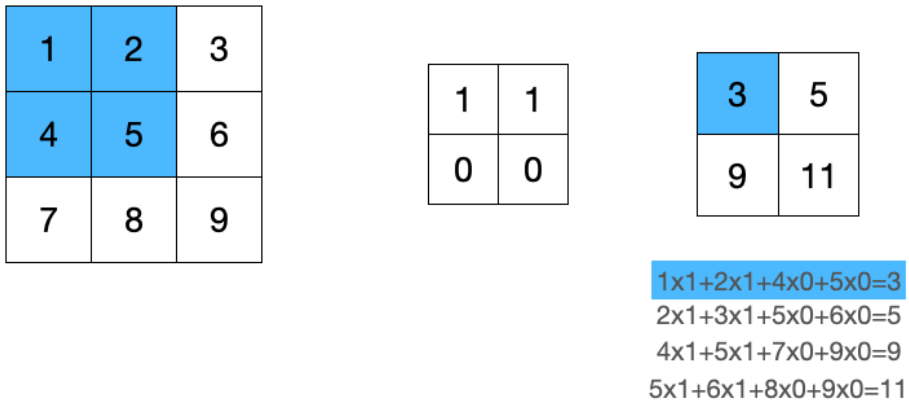
The following formula can be used to calculate the output dimension.

$$h_o = \frac{h_i - h_k}{h_s} + 1 \tag{6.5}$$

$$w_o = \frac{w_i - w_k}{w_s} + 1 \tag{6.6}$$

where $h_o$ is the output height, $h_i$ is the input height, $h_k$ is the kernel height, $h_s$ is the stride height, $w_o$ is the output width, $w_i$ is the input width, $w_k$ is the kernel width, $w_s$ is the stride width.

The convolution operation is shown in Figure 6.1.




**Figures 6.1:** Convoluation operation

## 6.2 Properties

CNN leverages three important ideas:

- ♥ sparse interaction.
- ♥ parameter sharing.
- ♥ equivariant representations.

### 6.2.1 Sparse interaction

This is accomplished by making the kernel smaller than the input.

### 6.2.2 Parameter sharing

In convolutional layers, the same parameter defined in one kernel are used at every position of the input.

### 6.2.3 Equivariant representations

In the case of convolution, the particular form of a parameter sharing causes the layer to have a property called **equivariance** to translation. To say a function is equivariant means that if the input changes, the output changes in the same way.

## 6.3 Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling oeration reports the maximum output within a rectangular neighborhood. Pooling helps to make the representation approximately invariant to small translations of the input. Invariant to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.

The following formula can be used to calculate the output dimension.

$$h_o = \frac{h_i - h_k}{h_s} + 1 \tag{6.7}$$

$$w_o = \frac{w_i - w_k}{w_s} + 1 \tag{6.8}$$

where $h_o$ is the output height, $h_i$ is the input height, $h_k$ is the pooling height, $h_s$ is the stride height, $w_o$ is the output width, $w_i$ is the input width, $w_k$ is the pooling width, $w_s$ is the stride width.

Chapter 7

# Metric

| | | Real value | |
|---|---|---|---|
| | | positive | negative |
| Predict value | positive | true positive | false positive |
| | negative | false negative | true negative |

**Tables 7.1:** Confusion matrix

## 7.1 Precision

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7.1}$$

Of all the predicted positive values, the ratio of the true positive values (the real value is positive and the predicted value is positive).

## 7.2 Recall

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{7.2}$$

Of all the real positive values, the ratio of the true positive values.

## 7.3 Accuracy

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{7.3}$$

Of all the values, the ratio of correctly predicted values. The disadvantage is that it is not suitable for unbalanced data.

## 7.4 F-score

$$\mathrm{F} = \frac{(\alpha^2 + 1) \times \text{precision} \times \text{recall}}{\alpha^2 \times \text{precision} + \text{recall}} \tag{7.4}$$

When determining the value of the parameter $\alpha$, if we pay more attention to recall (compared to precision), we should choose a larger $\alpha$. The F-1 score is the expression when $\alpha = 1$

Chapter 8

# Regularization

We train model on training data but use test data (not used to train the model) to test out model. The ability to perform on test data is called **generalization**. We can use model on test data because we assume that the train data and the test has the same probability distribution (i.e. they have relationship).

**Regularization** is any modification we make to a learnining algorithm that is intended to reduce its generalization error.

In practice, it is very difficult to find a model with the right number of parameters. Indeed, we often use a larger model that has been regularized appropriately.

## 8.1 Parameter norm penalties

Parameter norm penalties ($\Omega(\boldsymbol{\theta})$) can be added to the object function $J$ to limit the capacity of the model.

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}) \tag{8.1}$$

### 8.1.1 $L^2$ Parameter Regularization

The $L^2$ parameter norm penalty commonly known as **weight decay**.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 \tag{8.2}$$

Where $\boldsymbol{w}$ is the model parameter matrix.

### 8.1.2 $L^1$ Regularization

$L^1$ regularization on the model parameter $\boldsymbol{w}$ is defined as

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 \tag{8.3}$$

## 8.2 Dataset Augmentation

The best way to make a machine learning model generalize better is to train it on more data. Of course, in practice, the amount of data we have is limited. One way to get around this problem is to create fake data and add it to the training set. Dataset augmentation has been a particular effective technique for a specific classification problem: object recognition.

## 8.3 Early stopping

Early stopping is used to avoid overfit.

The algorithm terminates when no parameters have improved over the best recorded validation error for some pre-specified number of iterations. This strategy is known as **early stopping**. It is probably the most commonly used form of regularization in deep learning.

## 8.4 Droupout

Chapter 9

# Activation functions

Activation functions are usually added into neural network in order to help the network learn complex patterns. Because many of the patterns we want to learn are non-linear, we usually add non-linear activation functions into neural network to add the ability to learn non-learn patterns. Its function is to determine what information can be passed to the next neuron.

## 9.1 Desirable features

- ♥ Differentiable. This is needed for optimization.
- ♥ Low computational expense.
- ♥ Zero-centered.

## 9.2 Sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{9.1}$$

It is no longer used in recent deep learning models because it is computationally expensive, causes vanishing gradient problem and not zero-centered.

## 9.3 Tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{9.2}$$

Comparing to sigmoid, it solve the not zero-centered problem.

## 9.4 ReLU

$$\text{relu}(x) = \max(0, x) \tag{9.3}$$

This is a widely used activation function.

## 9.5 Leaky ReLU

$$\text{lrelu}(x) = \max(\alpha x, x) \tag{9.4}$$

## 9.6 Swish

$$\text{swish}(x) = x * \text{sigmoid}(x) = x * (1 + e^{-x})^{-1} \tag{9.5}$$

Chapter 10

# Machine learning process

1. confirm your target
2. observe the data
3. preprocess the data it necessary
4. select the model
5. select the cost function
6. select the optimizer
7. train the model
8. fine-tune the model
9. predict the data using the model
10. deploy the model

# Part II

# Tools

Chapter 11

# PyTorch

Chapter 12

# NumPy

## 12.1 Load data

```
1 import numpy as np
2
3 data = np.load('foo.npz')
```

# Part III

# Practice

Chapter 13

# Preprocessing

Preprocessing is not necessary. We preprocess the data because we can not get what we want from the data directly.

## 13.1 Not valued based data

The computer can only process number-based data. For not number-based data, we need to convert them into number-based data. For example, convert a string into a number list. For null values, we can drop them or compute them with other not null values.

```
from sklearn.preprocessing import LabelBinarizer

data = ['dog', 'cat', 'dog', 'horse']
print('data: ', data)
binarizer = LabelBinarizer()
binarizer.fit(data)
print(binarizer.transform(data))
print(binarizer.transform(['dog']))

"""
data:  ['dog', 'cat', 'dog', 'horse']
[[0 1 0]
 [1 0 0]
 [0 1 0]
 [0 0 1]]
[[0 1 0]]
"""
```

# Part IV

# Projects

# Bibliography


[1] Aston Zhang et al. *Dive into Deep Learning*. 2021.

[2] Debra Cameron et al. *Learning GNU Emacs*. O'Reilly Media, Inc., 2004.

[3] Ian Goodfellow et al. *Deep Learning*. 2016.

[4] Yann LeCun et al. Gradientbased learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[5] D.E. Knuth. *The TEXbook*. Addison Wesley, 1986.

[6] D.E. Knuth. Typesetting concrete mathematics. *TUGboat*, 10(1):31–36, April 1989.

[7] Adrian Rosebrock. *Practical Python and OpenCV*. PyImageSearch, 2016.

[8] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. PyImageSearch, 2017.