Contents

1 A Brief Background on Deep Learning

Nowadays you can find different definitions of deep learning in different sources. According to IBM's modern definition, "deep learning is a subset of machine learning in which multi-layered neural networks -modeled to work like the human brain- 'learn' from large amounts of data". In this chapter, we are going to follow this definition, which expresses deep learning as a subfield of machine learning, and try to understand it.

The beginning of this chapter provides an overview of machine learning and its key components. Then, we are going to discuss about how the basic neural networks work and how they are trained. At the end of this chapter, we will focus on convolutional neural networks (CNNs) and graph convolutional neural networks (GCNNs), which are the backbones of object detection models and this work.

1.1 Machine Learning

What is Intelligence?
"(1) : The ability to learn or understand or to deal with new or trying situations :REASON. Also: the skilled use of reason
(2) : the ability to apply knowledge to manipulate one´s environment or to think abstractly as measured by objective criteria (as tests)." (Webster´s New Collegiate Dictionary)

As we can infer from the above definition, artificial intelligence enables computers and machines to have learning ability, cognition, ability to make a decision and problem solving skills.

It pursues four possible goals: [RN95]

- Systems that can think like a human being

- Systems that are able to think logically
- Systems that can act logically
- Systems that are capable of acting like a human being

Thomas N. Kipf has described machine learning as ”one of the most widely pursued branches of AI (artificial intelligence), which deals with the question of how we can build systems and design algorithms that learn from data and experience (e.g.,by interacting with an environment)”. [Kip20]

Why Machine Learning Matters?
As the human population grows, so does the amount of information. Quantitative changes brought qualitative changes, which led to an emerging of a new term *big data*. ”Big data is very large sets of data that are produced by people using the internet, and that can only be stored, understood, and used with the help of special tools and methods” (Cambridge Dictionary). With the availability of big data, machine learning has become a key technique for solving problems in many fields.[ZPWV17]
Some areas, where machine learning is popular:

- Computational biology
- Image Processing and Computer Vision
- Computational finance
- Automotive, aerospace, and manufacturing
- Natural language processing

There are three kinds of learning problems in machine learning and they differ mostly in the form of the input data.

Supervised Learning
In supervised learning, there is a relationship given between input and output data, and predictions about what output data will be, can be made from labeled input data. [ZLLS20][TK13]

Unsupervised Learning
Unlike supervised learning, there is no relationship between input and output data, nor is there any guidance(input data is not labeled and not categorized). The purpose here is to recognize and find some structure between the layers in the given input data. [ZLLS20][TK13]

Reinforcement Learning
In reinforcement learning, system is in an interaction loop with an environment: the agent has a purpose and receives some observations from environment and must give the best feedback to the environment to get closer to its goal. [ZLLS20]

1.1.1 The Key Components

1.1.1.1 Datasets

There is no data science without data. This subchapter may take pages, but we can think of datasets mainly as collection of samples.

If you were working with video data, each sequence might represent an example.

Machine learning algorithms improve their performance directly proportional to the number of samples. That means, the more data we have, the stronger our model will be. But it is not enough to have a lot of data, we also need right data. If we dont have right data, it doesn't matter, how strong our model is, because: *garbage in, garbage out.*

So a good dataset has both a lot of data and right data. [ZLLS20]

1.1.1.2 Models

A Model is a model of how to transform the input data.[ZLLS20] Deep learning models are to be trained with neural network architectures, that includes lots of layers. There are also several pre-trained model architectures. Instead of creating a new model, we can use the architecture of a pre-trained model, that has been created by someone else to solve the same problem.

As an example of such a model we can mention ResNet18, an 18-layer deep convolutional neural network. There is also a version of this network model pre-trained with (ImageNet) database, that we can download and use for image classification problems. [HZRS15] In the next chapters, we will discuss in more detail about CNN models and pre-trained CNN models.

1.1.1.3 Loss Functions

To find out how well our model has improved or if it has improved at all, we need standardized measured values. In machine learning, we determine these values with *loss functions* or *cost functions.* The return value of our loss function quantifies how much loss we have in model. That means, the lower the value, the better our model has trained. [ZLLS20]

But getting a lower value from loss function doesn't guarantee that our model will perform well on test data, which he has not seen yet. If a student does well in lectures and old examinations but fails in real exam, then perhaps one can speak about learning by rote. In machine learning we call it *overfitting.*[ZLLS20]

Usually, the loss functions are defined taking into account the problem and model's parameter. For example, while the cross-entropy is commonly used as a loss function for classification problems, mean squared loss is used for regression problems.[ZLLS20]

1.1.1.4 Optimization

Optimization is the process of minimizing or maximizing a function $f(x)$ through variation of x. In our case, optimization is used to minimize the loss function. The commonly used approach for optimization (minimizing the loss function) is called *gradient descent.* To gain a better insight of backpropagation, which is the base of training a model, it is important to understand the gradient descent method. [GBC16] [ZLLS20]

Gradient Descent
Assume we have a differentiable and univariate function $y = f(x)$. The derivative of this function ($f'(x)$) at point P ($f'(P)$) gives us the slope of the line that is tangent to the function at point P.

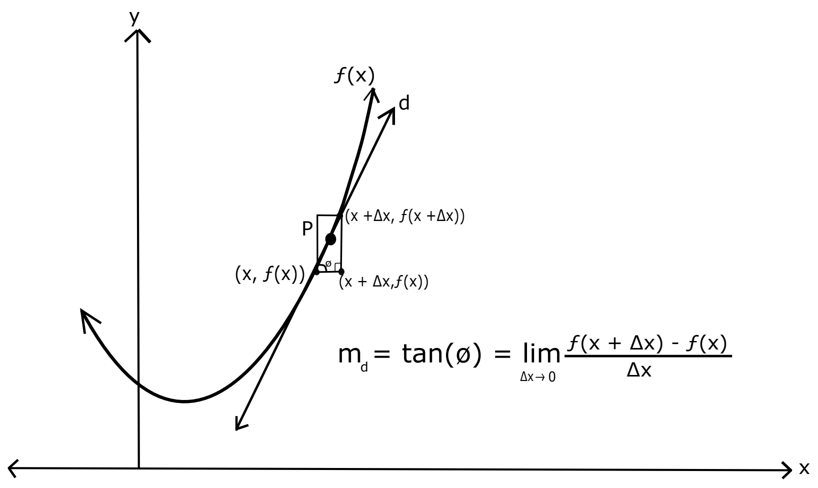

Figure 1.1: To visualize it in our minds, an example function $y = f(x)$ and the slope of the line d, that is tangent to the function at the point P

The slope of the line determines the increasing rate and the direction of $f(x)$. That means, by looking at the slope we can make conclusions about the location of the minimum point, which is in the opposite direction of the increase, except when $f'(P) = 0$ (a point with zero slope, which called *critical point*). We can also make conclusions about the maximum points, but in our case only the minimum points matter. [GBC16]

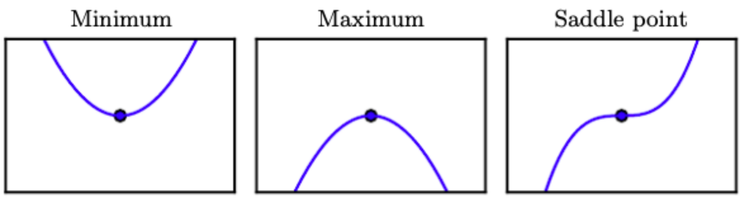

Figure 1.2: Three kinds of critical points in one dimension. The point that is at a lower level than the points arround it called *lokal minimum* and the lowest point of function called *global minimum* . The point that is at a higher level than the points arround it called *lokal maximum*. The *saddle point* has both higher and lower points around it. [GBC16]

In deep learning we often have multiple inputs, which lead us into two-, three- or multi-dimensional space and partial derivatives.

Consider f function as a hilly landscape and we are standing at the point (x_0,y_0). Here, the slope of the hill depends on which direction we walk: if we walk in the positive x direction the slope is $\frac{\partial f}{\partial x}$ and the slope in the positive y direction is $\frac{\partial f}{\partial y}$. In this instance, the vector that contains all the partial derivatives is called *gradient* and denoted with ∇f. The critical points are, where each element of the gradient is zero. [GBC16]

Decreasing the function f to reach the minimum by making it move in the opposite direction of the gradient, called *gradient descent* and can be defined as follows:

$$\boldsymbol{x'} = \boldsymbol{x} - \eta \nabla_x f(\boldsymbol{x}) \tag{1.1}$$

with the vector $\boldsymbol{x'}$ denoting the a new point and the positive skalar η the *learning rate*. [GBC16] The learning rate is to be chosen by algorithm designer and can be chosen in many different ways (e.g. line search). A small learning rate lead us to a better solution but can cause $\boldsymbol{x'}$ to update very slowly and we have the risk of being stuck in a local minima, but a high learning rate can cause overshoot and gradual divergences. [ZLLS20]

Using backpropagation through the network, we compute the gradient for different points in the input space until a minimum of loss function is found. We are going to discuss this process in the backpropagation chapter.

1.1.1.5 Activation Functions

As the name implies, activation functions determine if the neuron should be activated (fired) or not. They also convert the linear inputs into non-linear outputs, which supports learning of high-order polynomials for deeper networks. [ZLLS20]

1.1.1.5.1 Rectified Linear Unit (ReLu)

ReLu is a non-linear activation function. It is defined as follows, where x is the input of a neuron:

$$ReLu(x) = max(x, 0) \tag{1.2}$$

In short, if the input is negative it outputs zero (derivative 0), and if the input is positive it outputs a linear function (derivative 1) of the input. [ZLLS20] One of the main advantages of ReLu, is the ability to avoid the gradient vanishing effect (gradients do not flow fast in one direction and slow or not at all in another, but rather they flow well). [GBB11]

1.1.1.5.2 Sigmoid Function

The mathematical formulation of the sigmoid function is shown in equation (1.3).

$$sigmoid(x) = \frac{1}{1 + exp(-x)} \tag{1.3}$$

It takes a real input in the range (-inf, inf) and squashes it into a value in the range $(0, 1)$. [ZLLS20]

The derivate of the sigmoid function is as follows:

$$\begin{aligned}\frac{d}{dx}sigmoid(x) &= \frac{d}{dx}(1 + exp(-x))^{-1} = -(1 + exp(-x))^{-2}\frac{d}{dx}(1 + exp(-x) \\ &= -(1 + exp(-x))^{-2}(-exp(-x)) = \frac{exp(-x)}{(1 + exp(-x))^2} \\ &= sigmoid(x)(1 - sigmoid(x))\end{aligned} \tag{1.4}$$

1.2 Neural Networks

From definition of artificial intelligence through machine learning to the key components of deep learning, we arrived at the point where "multiple 'layers' of differentiable non-linear transformations are stacked and a model is trained using gradient descent technique". [Kip20]

1.2.1 Single Layer Neural Networks

A neural network consist of milions of connected nodes. The idea of these nodes was inspired by the neurons in the human brain, so we call them *neurons*, even though they are much simpler than biological neurons. A neuron in a neural network is where all operations take place and they consist of some inputs and outputs. [KLF16]

A single layer neural network is the most basic feedforward neural network architecture. We call it feedforward, because information flows in forward direction and there are no returns, otherwise we would call it *recurrent neural network.*[GBC16]

Since no operations are performed in the input layer of nodes, by single-layer we mean the output. [KLF16]

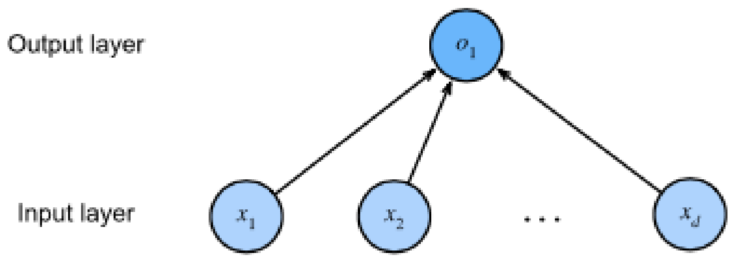

Figure 1.3: Linear regression diagram, which is also a single-layer neural network from the book [ZLLS20]. It contains d inputs $(x_1, x_2, ..., x_d)$ in input layer and has one output (o_1).

From now on, we name the objects we want to predict, labels or targets and we name our independent variables (inputs), features. [ZLLS20]

In single-layer neural networks information (input) x_i is weighted, which is used to determine the effect of the information and summed with a bias b. Bias is the value, telling us what the predicted value should be if all features have the value 0 (a limit for our model). [ZLLS20] The net input can be written as

$$\begin{aligned} n &= \sum\nolimits_i w_i x_i + b, \\ n &= \boldsymbol{w}^T \boldsymbol{x} + b, \end{aligned} \tag{1.5}$$

where n denotes the net input, $\boldsymbol{w}$ the weight vector $\boldsymbol{w} \in \mathbb{R}^i$, x_i the feature vector $\boldsymbol{x} \in \mathbb{R}^i$ and b the bias. This net input will be processed after some transformations via an activation function σ. [DBDJH14]

Now the output can be written as

$$y = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b). \tag{1.6}$$

We may solve our linear regression or logistic regression problems with a single layer neural network, but what if our problems get more complicated and we need more powerful networks? Then we need to get rid of the monotonicity and add one or more hidden layer to our network.

1.2.2 Multi-Layer Neural Networks

Multilayer neural networks have several layers, that we divide into three categories: input layer, hidden layers and output layer.

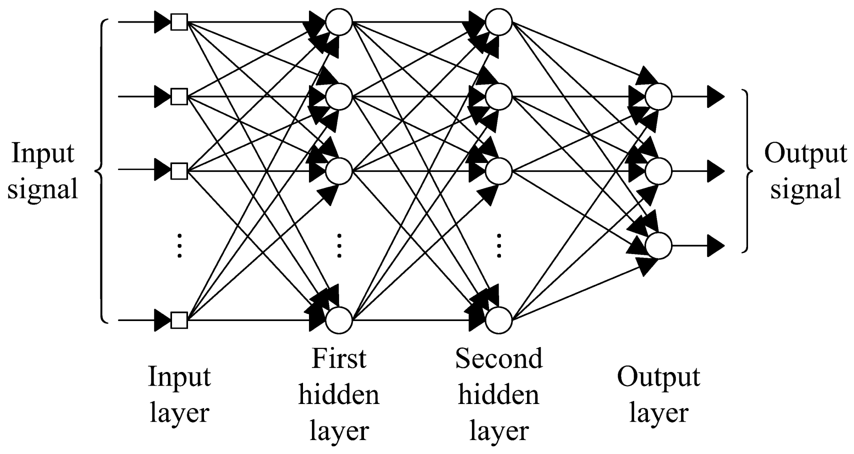

Figure 1.4: A Multilayer neural network graph with two hidden layers from the book [KLF16]. Note that different layers may contain different numbers of neurons.

We can build more powerful models with multilayer neural networks than single layer neural networks, because since each layer has its own weight matrix $\mathcal{W}$, its own bias vector $\boldsymbol{b}$ and its own activation function f, it provides diversity, which a single layer neural network can not provide. [DBDJH14]

1.2.2.1 Forward Propagation

Forward propagation, also called forward pass, is the process of calculating and storing variables in a sequence that starts from the input layer and ends in the output layer. In other words, forward pass makes the steps to generate an output [ZLLS20]

In the simplest way, we can define this calculation process as concatenation of linear functions $f^k(.)$ $k \in \{1, 2, ..., K\}$, each also concatenated by a nonlinear function (activation functions) $\sigma(.)$ [Kip20]

$$\begin{aligned} h: \quad & \boldsymbol{x} \to (\sigma \circ f^K \circ \sigma \circ f^{K-1} ... \circ \sigma \circ f^2 \circ \sigma \circ f^1)(\boldsymbol{x}), \\ \sigma : \mathbb{R}^p \to \mathbb{R}^p, \qquad & \boldsymbol{x} \to \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ .. \\ .. \\ \sigma(x_p) \end{bmatrix} \end{aligned} \tag{1.7}$$

where $\circ$ is the function composition and h the forward function. Note that the length of the chain shows the depth of the network. [GBC16] Linear function is defined as follows:

$$f^{(k)}(\boldsymbol{m}) = \mathcal{W}^{(k)}\boldsymbol{m} + \boldsymbol{b}^{(k)}, \tag{1.8}$$

where k denotes the layer number.[Kip20]

Let us consider a multilayer neural network with two hidden layers (as shown in Fig.1.4), that is trained to solve a classification problem.

Here, the calculations in the first hidden layer can be shown as follows:

$$\begin{aligned} \boldsymbol{y}^{(1)} &= (\sigma \circ f^{(1)})(\boldsymbol{x}) \\ &= \sigma(f^{(1)}(\boldsymbol{x})) \\ &= \sigma(\mathcal{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \end{aligned} \tag{1.9}$$

where $\hat{\boldsymbol{y}}^{(1)}$ denotes the output [1] vector of the first hidden layer, $\mathcal{W}^{(1)}$ the weight matrix, $\boldsymbol{x}$ the input vector of our network, $\boldsymbol{b}^{(1)}$ the bias vector of the first hidden layer and σ the activation function.

As the output of the first hidden layer is the input of the second hidden layer, we can write the output of the second layer as follows:

$$\begin{aligned} \boldsymbol{y}^{(2)} &= (\sigma \circ f^{(2)} \circ \sigma \circ f^{(1)})(\boldsymbol{x}) \\ &= (\sigma \circ f^{(2)})(\boldsymbol{y}^{(1)}) \\ &= \sigma(\mathcal{W}^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}) \\ &= \sigma(\mathcal{W}^{(2)}\sigma(\mathcal{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}). \end{aligned} \tag{1.10}$$

Finally, based on this, we can write the output of our network as

$$\begin{aligned} \boldsymbol{y} &= (\sigma \circ f^{(3)} \circ \sigma \circ f^{(2)} \circ \sigma \circ f^{(1)})(\boldsymbol{x}) \\ &= (\sigma \circ f^{(3)})(\boldsymbol{y}^{(2)}) \\ &= \sigma(\mathcal{W}^{(3)}\boldsymbol{y}^{(2)} + \boldsymbol{b}^{(3)}) \\ &= \sigma(\mathcal{W}^{(3)}\sigma(\mathcal{W}^{(2)}\sigma(\mathcal{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}) + \boldsymbol{b}^{(3)}). \end{aligned} \tag{1.11}$$

The loss term for a single data example can be calculated with

$$L = l(\boldsymbol{y}, \hat{\boldsymbol{y}}), \tag{1.12}$$

where l is our loss function and $\hat{\boldsymbol{y}}$ is the label of the data example.[ZLLS20]

Let us go into more detail. Since we have a classification problem, it would be best if we use the cross-entropy loss as loss function, because it measures the difference between the number of bits in our model and the given data, which is a good comparison approach. [ZLLS20]

Then our loss function over M classes (dog, cat, bird etc.) is

$$l(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{m=1}^{M} \hat{\boldsymbol{y}}_{\boldsymbol{m}} \log \boldsymbol{y}_{\boldsymbol{m}}. \tag{1.13}$$

[1] Actually, we do not directly observe the output from each of these layers, that's why we call them hidden layers. We just assume that we have the output vector.

1.2.2.2 Backpropagation

Backpropagation is the algorithm that uses the chain rule (from calculus) to propagate back from the output of the network to the input and enables us to calculate the gradient. [ZLLS20]

1.2.2.2.1 Chain Rule

As we saw in the forward propagation section 1.2.2.1, in deep learning multidimensional functions are often concatenated and using the chain rule we can differentiate concatened functions. [ZLLS20]

Assume $f(x)$ can be differentiable at the point $x = g(h)$ and $g(h)$ can be differentiable at the point h, then the composite function $(f \circ g)(h) = f(g(h))$ is differentiable at the point h.

$$(f \circ g)'(h) = f'(g(h))g'(h) \tag{1.14}$$

Using Leibniz's notation, the chain rule states that [ZLLS20]

$$\begin{aligned} &y = f(x), \quad x = g(h), \\ &\frac{dy}{dh} = \frac{dy}{dx}\frac{dx}{dh}. \end{aligned} \tag{1.15}$$

Back to backpropagation, at the end of this algorithm, our expectation is

$$\min_{\boldsymbol{\Theta}} \underbrace{-\sum_{m=1}^{M} \hat{\boldsymbol{y}}_{\boldsymbol{m}} \log \boldsymbol{y}_{\boldsymbol{m}},}_{\boldsymbol{T}(\boldsymbol{\Theta})} \tag{1.16}$$

where $\Theta = (\mathcal{W}^{(1)}, \mathcal{W}^{(2)}, \mathcal{W}^{(3)})$ are the parameters of the network from the last section 1.2.2.1.

In our network, the gradient descent is calculated as shown below

$$\Theta_{t+1} = \Theta_{t+1} - \sum_{i=1}^{h} \frac{\partial}{\partial \Theta} \boldsymbol{T}(\boldsymbol{\Theta}). \tag{1.17}$$

And backpropagation algorithm computes the gradient as follows:

$$\begin{aligned} \frac{\partial \boldsymbol{T}(\boldsymbol{\Theta})}{\partial \mathcal{W}^{(l)}} &= \frac{\partial \boldsymbol{T}(\boldsymbol{\Theta})}{\partial f^{(l)}} \frac{\partial f^{(l)}}{\partial \mathcal{W}^{(l)}} \\ &= \frac{\partial \boldsymbol{T}(\boldsymbol{\Theta})}{\partial f^{(l)}} \boldsymbol{y}^{(l-1)} \\ &= \sum_{m} \frac{\partial \boldsymbol{T}(\boldsymbol{\Theta})}{\partial f^{(l+1)}} \frac{\partial f^{(l+1)}}{\partial f^{(l)}} \boldsymbol{y}^{(l-1)} \\ &= \sum_{m} \frac{\partial \boldsymbol{T}(\boldsymbol{\Theta})}{\partial f^{(l+1)}} \mathcal{W}^{(l+1)} \sigma'(f^{(l)}) \boldsymbol{y}^{(l-1)} \end{aligned} \tag{1.18}$$

where $y^{(0)} = x$.

1.2.2.3 Training

In training, forward propagation and backward propagation are not considered independent. When we have one forward and one backward pass we call it *one pass.* Also, one forward and one backward pass of all the training examples is called *one epoch.* [ZLLS20]

	Training process of a neural network
1	Define the neural network you want to train with some weights
2	Iterate over a dataset of features (inputs)
3	Process input across the network and generate an output (forwardpass)
4	Calculate the loss with loss function that you have already defined
5	Compute the gradients and propagate them back to the weights of the network
6	Update the weights and biases of the network with the gradient descent rule

Table 1.1: A small overview table that explains step by step how the networks are trained, which is based on the tutorial "A 60 Minute Blitz" by PyTorch.

1.2.3 Convolutional Neural Networks

Bibliography

[DBDJH14] Howard B. Demuth, Mark H. Beale, Orlando De Jess, and Martin T. Hagan. *Neural Network Design*. Martin Hagan, Stillwater, OK, USA, 2nd edition, 2014.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[Kip20] Thomas Norbert Kipf. *Deep Learning with Graph-Structured Representations*. PhD thesis, University of Amsterdam, 2020.

[KLF16] J.M. Keller, D. Liu, and David Fogel. *Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation*. 07 2016.

[RN95] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 1995.

[TK13] Anish Talwar and Y. Kumar. Machine learning: An artificial intelligence methodology. 2013.

[ZLLS20] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. `https://d2l.ai`.

[ZPWV17] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V. Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237:350 – 361, 2017.