**3 NEURAL NETWORKS**

This chapter is dedicated to description of NN in general and its special type called CNN.

**3.1 History**

History of NNs can be arguably dated from 1943, when Warren Mcculloch and Walter Pitts devised mathematical model inspired by Biology of central nervous systems of mammals [25].

This inspired the invention of Perceptron, created in 1958 by Frank Rosenblatt. Perceptron used very simple model mimicking biological neuron that was based on mathematical model of Pitts and Mcculloch. Definition of the Perceptron model also described an algorithm for direct learning from data.

In the beginning Perceptron seemed very promising, but it was soon discovered that it had severe limitations. Most prominent voice of criticism was Marvin Minsky. Minsky published book in which he laid out a case that Perceptron model was unable to solve complex problems [26]. Among others the book contained mathematical proof that Perceptron is unable to solve simple XOR problem. More generally the Perceptron is only capable of solving linearly separable problems. Even though according to Minsky this criticism wasn’t malicious, it in effect stifled the interest in NNs for over a decade.

Interest in NNs was rejuvenated in the early 80’s, when it was shown that any previously raised deficiencies could have been solved by usage of multiple units. This was later exacerbated by invention of back-propagation learning algorithm, which enabled the possibility to gather neurons into groups called layers, which can be stacked into hierarchical structures to form a network. NN of this type were commonly called Multilayer Perceptron (MLP). In 80s and 90s the interest in NNs plateaued again and general research of AI was more focused on other1 machine learning techniques. In the realm of classification problems, it was notably SVM and ensemble model. AI research community also developed several other paradigms of NNs that were similarly inspired by Biology of certain aspect of central nervous system but took different approaches. Most important examples were SOM and Recurrent Neural Network (RNN)(e.g. Hopfield networks).

By the year 2000, there was very few research groups that were devoting enough attention to the NNs. There was also certain disdain for NNs in academia and AI research community. Success of NNs that was promised almost half a century ago was finally encountered around 2009, when the first networks with large number of hidden layers were successfully trained. This led to mainstream adaption of umbrella term deep learning which by and large refers to Deep Neural Network (DNN). The word deep indicates that networks have large number of hidden layers

The key theoretical insight was to learn complicated functions that could represent high-level abstractions (e.g. vision recognition, language understanding etc.). There is a need for deep architecture.

NNs in the times before DNNs had only 1 or 2 hidden layers. These are today often called shallow networks. Typical Deep Networks can have number of hidden layers in order of tens, but in some cases even hundreds [18]

Even though that progress of Neural Network into direction of structures with high number of hidden layers was obvious, its training was unsolved technical problem for very long time. There were basically 3 reasons why this breakthrough didn’t come sooner

1. There were no technique allowing the number of hidden layers to scale.

2. There wasn’t enough of labeled data necessary to train the NN.

3. The computational hardware wasn’t powerful enough to train sufficiently large and complex networks effectively.

First problem was tackled by invention of CNNs [24]. Second problem was solved simply when there was more data available. This was mainly achieved thanks to effort of large companies (Google, Facebook, YouTube, etc.) but also with help of large community of professionals and hobbyists in data sciences.

Both innovation in computational hardware and improvement of training methods were needed to solve the third problem. One of the technical breakthroughs was utilization of Graphics Processing Units (GPUs) for the demanding computation involved in training of a complex network. Thanks to the fact that training process of NNs is typically large number of simple consequent computations, there is a great potential for parallelization.  
**3.2 Structure of Neural Networks**

The term NN is very general and it describes broad family of models. In this context NN is distributed and parallel model that is capable of approximating complex nonlinear functions. Network is composed from multiple computational units called neurons assembled in particular topology.

Description of NN structure will follow the convention laid out in the description of learning algorithm. Meaning that a description of the learning algorithm is composed of model, cost function and optimization procedure. The difference comes into play with the fact that model of NN is much more complex than the model linear regression. Therefore, the analysis is divided into model of neuron and topology of the network.

3.2.1 Model of Neuron

Neuron is computational unit performing nonlinear transformation of its inputs

𝑦 = 𝑔(𝑤𝑇𝑥 + 𝑏). (3.1)

Argument 𝑤𝑇𝑥 + 𝑏 of function 𝑔 is often regarded as 𝑧. Therefore, the equation can be rewritten as

𝑦 = 𝑔(𝑧). (3.2)

Typical schema is shown on Figure 3.1, which depicts inputs, weights bias and activation function.



Figure 3.1: Model of artificial neuron [29].

As it was already mentioned model of neuron was inspired by biology. First attempts to create model of neuron had multiple elements equivalent with neurons of human brain. As research progressed this equivalence ceased being as important and modern NN models correspond to their biological counterparts only superficially.

**Inputs**

Each neuron has multiple inputs 𝑥 that are combined together to execute some operation. Each input has designated weight assigned to it.

**Weights**

Inputs of a neuron are weighted by parameters 𝑤 that are modified during learning process. Each weight gives strength to each individual input into the neuron. The basic idea is that when the weight is small the particular input doesn’t influence the output of the neuron very much. Its influence is large in the opposite case.

**Bias**

Another modifiable parameter is bias 𝑏 that controls influence of the neuron as a whole.

**Activation Function**

For NN to approximate nonlinear function each neuron has to perform nonlinear transformation of its input. This is done with activation function 𝑔(𝑧) that preforms nonlinear transformation. There are several different commonly used activation functions. Its usage depends on the type of network and also on the type of layer in which they operate.

One of the oldest and historically most commonly used activation functions if sigmoid function. It is defined by

𝑔(𝑧) = 1 1 + 𝑒 −𝑧 . (3.3)

Problem with sigmoid is that its gradient becomes really flat on both extremes and as such it slows down the learning process [23].

Another activation function is hyperbolic tangent. It is defined as

𝑔(𝑧) = 𝑡𝑎𝑛ℎ(−𝑧). (3.4)

Hyperbolic tangent function is less common in feed forward NN, but it is largely used in RNN. Currently most frequently used activation function is Restricted Linear Unit (ReLU). It is very commonly used in both convolutional and fully connected layers. It is defined by

𝑔(𝑧) = max{0, 𝑧}. (3.5)

It has a drawback because it is not differentiable for 𝑧 = 0, but it is not a problem in software implementation and one of its biggest advantages is that it can learn very quickly.

All three activation functions are illustrated in Figure 3.2.

**3.2.2 Topology of the Network**

There are several different commonly used topologies. Two most commonly used in deep learning are feed-forward and recurrent. Feed forward networks are characterized by the fact that during activation the information flows only in forward direction from inputs to output. A recurrent network has some sort of feedback loop.

Another criterion of topology is how are individual neurons in the network connected. Most commonly are NNs ordered in layers. In each layer there can be from 1 to n neurons. Layers are hierarchically stacked. In typical terminology the first layer is called input layer, the last layer is called output layer and the layers in-between are called hidden.

Description of the network rests on interconnections between individual layers. Most common scheme is called fully connected where each neuron in hidden layer 𝑙 has input connections from all neurons from previous layer 𝑙 − 1 and its output is connected to input of each neuron in following 𝑙 + 1 layer. Entire structure is illustrated on Figure 3.3.



Figure 3.2: Activation Functions

From this point on the term NN will refer to Feed-forward Fully Connected Neural Network.

Types of neurons are dependent on the type of the layer. Currently the main difference is in their activation function, which wasn’t the case for a long time. Historically all layers had neurons with sigmoid activation function. It was mainly because the output sigmoid layer can be easily mapped onto probability distribution, since it acquires vales between 0 and 1. Only relatively recently2 it was found that network composed of neurons with ReLU activation function in the hidden layers can be trained very quickly and are more resistant against over-fitting. Activation functions are still subject of ongoing research.

Neurons in output layer need output that can produce probability distribution



Figure 3.3: Fully connected Feed Forward Neural Network [21].

that can be used to estimate the probability of individual classes. For this reason, most commonly used activation function of output neuron is called softmax.

Softmax is normalized exponential function. It is used to represent probability of an instance being member of class 𝑗 as

𝑔(𝑧)𝑗 = 𝑒 𝑧𝑗 ∑︀𝑘=1 𝐾 𝑒 𝑧𝑘 , (3.6)

where 𝐾 is total number of classes.

**3.2.3 Cost Function**

Cost functions of NNs is a complex topic that exceeds scope of this thesis. One of the most common cost function used in NNs for classification into multiple classes is categorical cross entropy. For softmax activation function from Equation 3.6 is cost function defined as

𝐶 = − 1 𝑛 ∑︁𝑛 𝑖=1 𝑦 (𝑖) ln 𝑔(𝑧 (𝑖) ) + (1 − 𝑦 (𝑖) ) ln(1 − 𝑔(𝑧 (𝑖) )), (3.7)

where 𝑦 (𝑖) if correct class of the instance and 𝑛 is total number of instances.

**3.2.4 Optimization Procedure**

Every optimization procedure for NN is based on gradient descent. In other words, it is iterative process that aims to lower training error of the network by differentiating of cost function and adjusting parameters 𝜃 of the model by following the negative gradient.

The problem is that cost function of entire network is very complex and has many parameters. To find the gradient of the cost function it is necessary to go through all of the units in the network and estimate their contribution to the overall error. Technique that is used to solve this problem is called back-propagation.

Back-propagation if often confused to be complete learning algorithm which is not the case, it is only the method to compute the gradient [16].

**Back-propagation**

To estimate the influence of individual units in a network the back-propagation is used to compute delta 𝛿 𝑙 𝑗 , where 𝑙 is layer and 𝑗 is index of neuron in that layer. Algorithm starts at the output of NN, more specifically its cost function.

𝛿 𝐿 = ∇𝑥𝐶 ⊙ 𝑔 ′ (𝑧 𝐿 ) (3.8)

where 𝐿 is last layer of the network and ∇𝑥𝐶 is gradient of cost function with respect to 𝑥 and ⊙ is the Hadamard product3 .

In subsequent lower layers the deltas are computed as

𝛿 𝑙 = ((𝑤𝑙+1) 𝑇 𝛿 𝑙+1 ⊙ 𝑔 ′ (𝑧 𝑙 ) (3.9)

where (𝑤𝑙+1) 𝑇 is from Equation 3.1.

Each neuron has two modifiable parameters 𝑏 and 𝑤. To estimate the rate of change for parameter 𝑏 𝑙 𝑗 from Equation 3.1 it needs to be computed as

𝜕𝐶 𝜕𝑏𝑙 𝑗 = 𝛿 𝑙 𝑗 (3.10)

Change of weight 𝑤 𝑙 𝑗𝑘 from Equation 3.1 it needs to be computed as

𝜕𝐶 𝜕𝑤𝑙 𝑗𝑘 = 𝑥 𝑙−1 𝛿 𝑙 𝑗 (3.11)

**Gradient Descent Optimization**

Back-propagation estimates gradient of all modifiable parameters 𝑏 and 𝑤 in the network. These parameters can be referred to by vector 𝜃. Therefore, the gradient of the function to be minimized can be written as ∇𝜃𝑡−1 𝑓(𝜃𝑡−1).

Simplest learning algorithm is called gradient descent. Even though simple, it is very robust learning algorithm.

𝑔𝑡 ← ∇𝜃𝑡−1 𝑓(𝜃𝑡−1) (3.12)

𝜃𝑡 ← 𝜃𝑡−1 − 𝜂𝑔𝑡 (3.13)

Algorithm has one meta-parameter 𝜂, which is often called learning rate. It determines how quickly are 𝜃 parameters updated. Simple gradient descent has the shortcoming that update of parameters is always exactly proportional to change of gradient. This might become a problem when the gradient change slows down. This algorithm is also often called Stochastic Gradient Descent (SGD). The word stochastic indicates that during training the algorithm is using random selection of instances to train. There are many different variations on the gradient descent method. Following definitions are taken from [14].

**Adam**

It is more complex learning algorithm that combines 𝐿2 norm and classical momentum based optimization. It should converge faster than classical Gradient Descent.

𝑔𝑡 ← ∇𝜃𝑡−1 𝑓(𝜃𝑡−1) (3.20)

𝑔^𝑡 ← 𝑔𝑡 1 − ∏︀𝑡 𝑖=1 𝜇𝑖 (3.21)

𝑚𝑡 ← 𝜇𝑚𝑡−1 + (1 − 𝜇)𝑔𝑡 (3.22)

𝑚^ 𝑡 ← 𝑚𝑡 1 − ∏︀𝑡 𝑖=1 𝜇𝑖 (3.23)

𝑛𝑡 ← 𝜈𝑛𝑡−1 + (1 − 𝜈)𝑔 2 𝑡 (3.24)

𝑛^𝑡 ← 𝑛𝑡 1 − 𝜈 𝑡 (3.25)

𝑚¯ 𝑡 ← (1 − 𝜇𝑡)𝑔^𝑡 + 𝜇𝑡+1𝑚^ 𝑡 (3.26)

𝜃𝑡 ← 𝜃𝑡−1 − 𝜂 𝑚¯ 𝑡 √ 𝑛𝑡 + 𝜀 (3.27)