Neural Networks and TensorFlow

Dominik Grüneis and Lukas Hubl

I. ARTIFICIAL NEURAL NETWORKS

A. *Introduction to artificial neural networks*

Neural networks are computer programs which try to imitate the principle of the biological brain to solve certain kinds of problems. Normal computer programs compute completely different than the human brain as most programs operate in a linear way. Most tasks the human brain is capable of doing would need days of calculation on a conventional computer program. The reason why the human brain is able to solve tasks like image recognition is, that the brain is able to learn and build up experience to solve such tasks.[1] Most people consider machine learning as a brand-new technology solving computational problems in the future. In fact, Aristotle (384-322 B.C.) was the first who tried to build an informal system of logical conclusions for accurate argumentation.[2] Which is nothing more but the attempt to create some sort of ”artificial intelligence”. The origin of computer based artificial intelligence, machine- and deep learning concepts date back to the 1940s, where it was first called ”cybernetics”. In the 1980s the name changed to ”connectionism” until the terms ”machine learning” and ”deep learning” were commonly used since the 2000s. Although there were different terms over time, all of them are referring to the same elementary concept, trying to solve computational problems by reproducing the brains way of learning and reacting based on its knowledge. Hence the name artificial neural networks.

B. *Architecture of a artificial neural network*

A main goal of an artificial neural network is to reproduce the biological architecture of the human brain in a programmatical way. The cell inside the human brain which computes the signals is called a neuron. A normally operating brain has billions of these cells to process information.[3] So to create an artificial neural network it is essential to create a artificial neuron. Fig.1 shows the biological architecture of a neuron. A neuron consists of dendrites which are recieveing impulses of other neurons, the cell body and the axon which sends impulses to other cells. A neuron is recieving signals from other neurons which increase or decrease the electrical potential of the cells nucleus. If this neuron reaches a specific threshold, the neuron sends out a signal via the axon. This signal is then recieved by other cells and processed in the same way.[4]

To simulate this behaviour a artificial neuron uses weighted inputs to simulate the dendrites. These inputs are combined inside a transfer function which the sends the combined input to an activation function. This activation function simulates the threshold of nucleus of the biological neuron. For example the sigmoid function shown in fig.3is often used as an activation function. The advantages of the sigmoid function is, that it is a non linear function which has the advantage to be capable of non binary activations, which means that it gives an analog output unlike a normal step function. It also has the characteristic to react significantly to changes around zero, which is good for a classifier. The output of the activation function is either sent to another artificial neuron or it is interpreted as an output. Artificial neurons are often called nodes in the technical context. Fig.2 shows how an artificial neuron is built. [3]

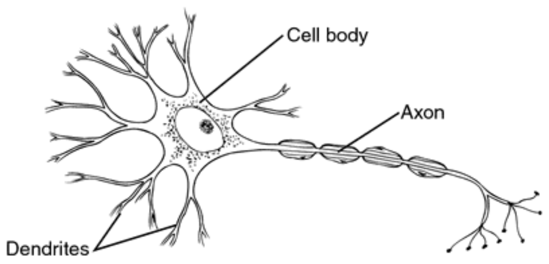

Fig. 1. *neuron[5]*

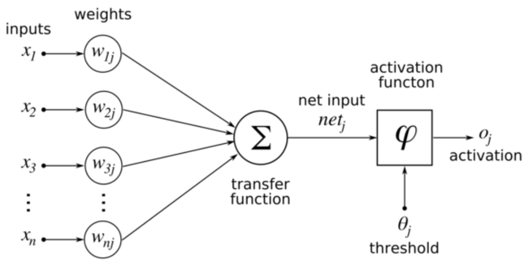

Fig. 2. *artificial neuron[6]*

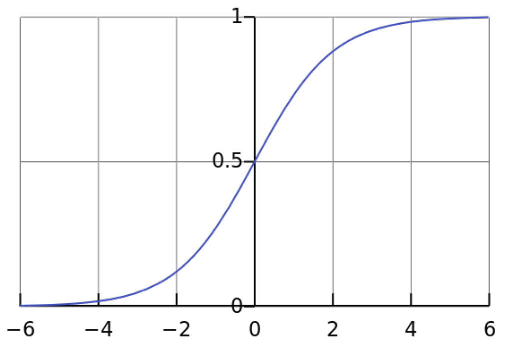

Fig. 3. *sigmoid function*

As a human brain consist of many neurons, an artificial neural network has to consist of many artificial neurons as well. The artificial neurons are organised in layers as shown in Fig.4. It is differentiated between input layers, hidden layers and output layers. Input layers get their input from the general system input and send their output to the hidden layers. Hidden layers get their input from the input layer nodes and send their output either to another layer of hidden layer nodes or to output layer nodes. Output layer nodes get their input from hidden layer nodes and send their output to the general system output. It is also possible to design an artificial neural network without a hidden layer,but most of the time these aren't as performant.[1]

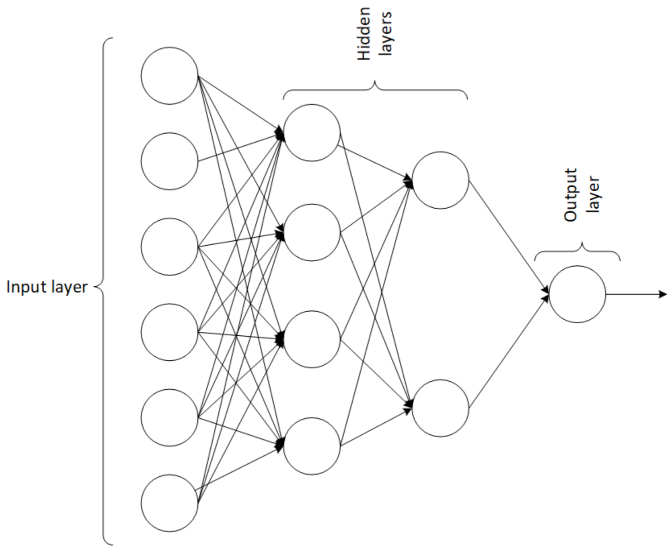

Fig. 4. *artificial neural network*

C. Functionality of a artificial neural network

As stated in the I-A the main reason why a brain is able to solve complex tasks rather quickly and easily is because the brain can adapt and learn. So to make a functioning artificial neural network, a learning feature has to be implemented. In the world of programming, learning means adapting specific variables. This is were the weigths of an artificial neuron are used. These weights are scaling the input of a node and decide how much an input is contributing to the combined input of a node. Therefore for a working artificial neural network, the weights have to be set the right way. A way to set the weights is by feeding teaching patterns to the network and let it set it's weight according to an algorithm. The probably most used algorithm for such learning is the backpropagation algorithm.[3] The backpropagation algorithm propagates the error back to the different weights of the nodes. The error is the square of the target output value and the actual output value of the neural network.

$$error = (t - o)^2 \tag{1}$$

This error has to be distributed to the weigths of every node in every layer with the function. For example to distribute this error in a network with one hidden layer and the sigmoid function as activation function the following formula can used:

$$\frac{\delta E}{\delta w_{jk}} = -(t_k - o_k) - sigmoid(\sum_j w_{jk} \cdot o_j) \cdot (1 - sigmoid(\sum_j w_{jk} \cdot o_j)) \cdot o_j \tag{2}$$

This formula means that the delta of a specific weight is the negative overall error of the network multiplied with the differentiated output of the node k $(sigmoid(\sum_j w_{jk} \cdot o_j)(1-sigmoid(\sum_j w_{jk} \cdot o_j)))$and multiplied with the output of the node j(o_j). This partial error isn't completely added or substracted from the specific weight. A factor is added called the learning rate α which prevents that a weight is is reacting too much to a single error. This prevents the system to be completely disfunctional if a teaching pattern would be wrong. So the value of the updated weight is:

$$w_{jkNew} = w_{jkOld} - \alpha \cdot \frac{\delta E}{\delta w_{jk}} \tag{3}$$

[7]

D. Use Cases for artificial neural networks

Now the question is, where is it beneficial to use artificial neural networks. The following use cases should show the variety of utilization of artificial neural networks.

Infrastructure: Electric load forecasting: Electric load forecasting is getting more important with the increasing use of sustainable energy sources like wind or solar energy where the energy production isn't constant over time. Therefore artificial neural networks are used to predict the power usage and to allocate energy reserves. [8]

Medicine: Diagnosis of Myocardial Infarction: In this case a artificial neural network model is used to determine if a patient with anterior chest pain has an myocardial infarction [9]

Medicine: Mammography: Using a artificial neural network to help radiologist in the analysis of mammographic data with the ambition to identify breast cancer [10]

Economy: Stock market prediction: The main goal of this usage is to train a artificial neural network to forecast the stock market exchange rates.[11]

E. Usage in todays technology

Artificial neural networks aren't just experimental anymore. Following companies use them in their products.

Netflix: Recommendation engine: Netflix uses artificial neural networks to improve their user experience by showing better movie and series recommendations to them.

Amazon: Alexa speech recognition: Amazon uses artificial neural networks to improve their speech recognition in their Alexa system

Microsoft: Skype language translation: Used in a skype feature which recognizes users speech and converts it to translated text in real time.

Apple: Face ID: Apple is using this technology in their Face ID system.

II. TENSORFLOW ™

A. Introduction to TensorFlow™

TensorFlow™is an open source software library for numerical computation particularly designed for large-scale machine and deep learning. It was originally developed by Googles brain team, conceived for both production and research. First released on November 9th of 2015, its now running version 1.8.0. A major benefit of TensorFlow™is its ability to break up the computation graph in multiple sub-graphs and run them across serveral CPUs, GPUs or TPUs on either Windows, MacOS or Linux as well as iOS and Android. TensorFlow™is certainly the most popular machine learning framework by now due to its scalability, flexibiliy and great documentation. SAP, Dropbox, Intel, AMD, NVIDIA and Airbnb are only a few companies using TensorFlow™for their needs.[12][5]

B. Architecture and Principles of TensorFlow™

TensorFlows™approach on creating numerical computations is to differentiate between the definition of computations by designing a data flow graph in Python and the actual computation of the graph by TensorFlow™using highly optimized C++ code.[5] The graph defining what and how to calculate later on consists of multiple nodes (tensors, operations) in order to form a numerical computation.

C. Tensors

A Tensor is a mathematical unit used to integrate scalars, vectors, matrices and other units of analog structure into a consistent scheme to describe mathematical and physical relations. [13]. So in TensorFlow™tensors represent data in a specified dimension. This means a 0-D tensor is a scalar, a 1-D tensor represents a vector, a 2-D tensor an array and so on.

D. Constants

Constants are constant tensors created by TensorFlows™`tf.constant()` operation returning a constant tensor on success.[12] Compared to variables, the difference beyond the missing initialisation is that constants are stored in the graph definition. Assuming a lot of big constants in the graph definition might make loading them expensive. So constants should only be used for small data whereas variables for bigger data.

E. Variables

Variables are tensors whose value could be manipulated by the program. In contrast to constants which are created by the `tf.constant()` operation, variables have to be initialized in order to be able to use them. The session itself allocates memory to save the variable values so it is not saved in the graph definition like constants. To define a variable the first step is to call the `tf.get_variable()` operation which takes several parameters.

Creating variables with `tf.get_variable()` performs reuse checks.[12] This means, if there is already a variable defined in the current scope whose parameters are identically, it gets this variable, otherwise it creates a new one. To be able to use the variable the second step is to run the variables initializer in a session like `session.run(my_variable.initializer)`. To initialize all the variables in the graph at once use the `session.run(tf.global_variables_initializer())` function.[14]

F. Placeholders

Placeholders are used when its not known which values should be fed into a tensor later on. It is similar to defining a function in programming. The data type of an input parameter is defined but the concrete value of it is unknown until the function is being called. Like variables which may not have been initialized, placeholders also produce an error if there was no value fed before evaluation. Creating placeholders is done using the `tf.placeholder()` operation.

Feeding values to placeholders uses TensorFlows ™ feeding mechanism `feed_dict`. [12]

G. Operators

To be able to work with different constants, variables and placeholders, the graph is lacking of operations between the data. TensorFlow™provides a huge variety of operations

- arithmetic operations
- basic math functions
- matrix math functions
- tensor math functions
- complex number functions

and many more.[12] These functions are also used when it comes to defining a graph and they work like links between the variety of data coming into the data flow.

H. Visualization

To get a better overview of the graph TensorFlow™offers a tool called TensorBoard. To be able to visualize the data flow graph TensorBoard needs a directory it can find the graph information in a specified format in order to be able to visualize it. To provide these serialized form of graph data TensorFlow™offers `FileWriter` objects. FileWriters are able to take the data flow graph, serialize it and save it to a specified location in the file system. Now that the graph is available in the right format, TensorBoard can be launched by `tensorboard --logdir=path/to/log-directory`. This opens a Web-Application as shown in Fig. 5, accessible from your browser at `localhost:6006`, allowing the user to see the data flow graph with all its nodes and tensors.[5][14]

I. Optimizer

As described in `C. Functionality of a artificial neural network`, the major challenge is to simulate the human brains capability to learn. So in artificial neural networks there are a bunch of techniques and algorithms trying to decrease the output error to its minimum, each of them working better than others in different situations. One of the simplest, basic optimizers

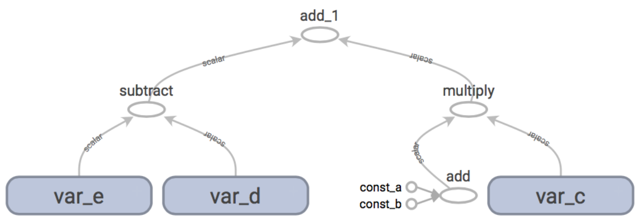

Fig. 5. *TensorBoard Interface*

however is the gradient descent algorithm due to its wide range of applicability. The general idea behind the algorithm is to adjust the parameters iteratively in order to find the best fitting parameter setup for the specified problem. It is pretty much like being on a mountain and trying to get down to the valley as fast as possible, given there is so much fog someone could barely see. The way gradient descent would try to solve this problem is trying to find the steepest slope downwards from the actual position and take a step down that direction. Now that the position has changed it would again check for the steepest slope and take another step. Continuing this strategy over and over again would probably lead to the foot of the mountain. [5] Each of these steps towards the valley could be seen as a single iterative step of learning towards the minimum of the error function. Fig. 6 shows the learning process decreasing the error in each learning step. Whilst the concept behind Gradient Descent seems pretty straight forward, there are two characteristics which are very important.

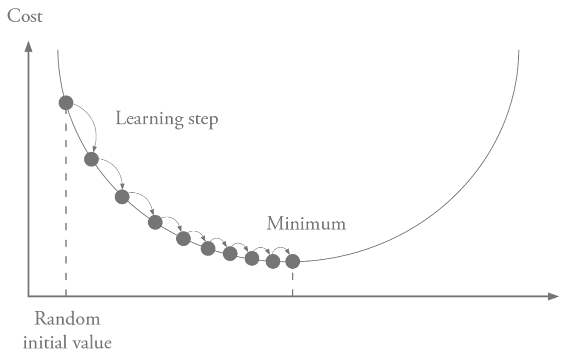

Fig. 6. *Gradient Descent [5]*

1) Learning rate: Defines the amount of change which will be applied to the parameters of a neural network in a single learning iteration. The key goal is to pick the learning rate in a way that on the one hand, learning the network will not take away too much time which would happen if the learning rate is too small, visualized in Fig. 7. And on the other hand it must not be too large which in turn could lead to the problem of getting worse with every iteration.

2) Initial parameter setup: As Fig. 6-9 show, there is always a certain point on the error curve which serves as a starting point. The position of that starting point depends on the initial parameter setup of the neural network. Taken into consideration that the error function curve is mostly not as trivial as shown in these figures, means, that there may be certain situations in which Gradient Descent will not be able to find the global minimum of the error function due to local minima or too less iterations, ending up on a plateau. Fig. 9 shows these two cases.

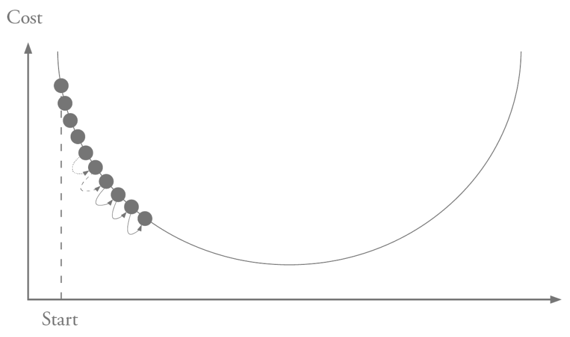

Fig. 7. *Small learning rate[5]*

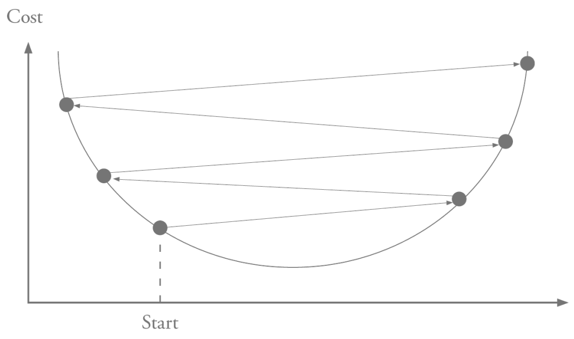

Fig. 8. *Large learning rate[5]*

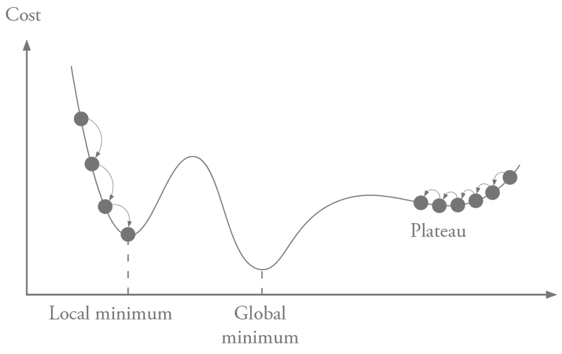

Fig. 9. *Gradient Descent pitfalls[5]*

Aside of implementing the optimizers all from scratch, TensorFlow™provides a variety of different optimizers out of the box which are adjustable in learning rate and other optimizer specific hyperparameters.

J. Decentralized computing

The support of distributed computing is one of TensorFlows ™biggest advantages over other machine learning frameworks. Having the opportunity to save some time by distributing the computations on several devices leads to more flexibility when it comes to retraining the model or experimenting with hyperparameter configurations. [5]

K. Multiple devices on a single machine

The idea is to use multiple devices (GPUs) connected to a single machine (PC) to distribute the computations to. A major advantage over the distribution across multiple machines is that all devices are connected to one computer so there is no network communication needed which would cost time. Using multiple devices in TensorFlow™requires the following 4 steps.[5]

- The GPUs need NVIDIA Compute Capability version 3.0 or higher
- CUDA (NVIDIAs Compute Unified Device Architecture library) hast to be installed in order to use the GPU for other purposes than just graphics rendering
- cuDNN (NVIDIAs CUDA Deep Neural Network library) which serves pre-implemented, commonly used, deep neural network specific computations like normalisation or pooling.
- Install TensorFlow™with GPU support.

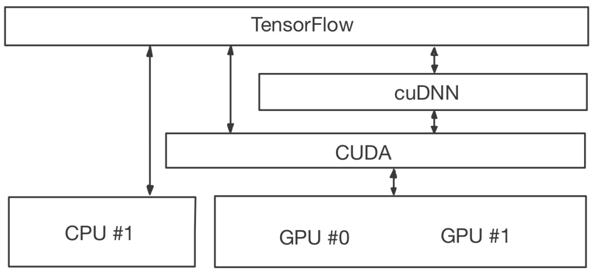

Fig. 10. *TensorFlow uses CUDA and cuDNN to control GPUs and boost DNNs [5]*

L. Simple Placer

To determine which node belongs to which device, TensorFlow™uses the Simple Placer. Simple Placer follows the following three rules:

- Already assigned nodes stay untouched.
- If the user defined a device for a node, the Simple Placer assigns it appropriately (if possible).
- If the node has no user defined device and is not already assigned, Simple Placer assigns it to GPU:0 (GPU mode) or CPU:0 (CPU mode) per default.

In certain cases it may occur that the device on which the node should run on is not capable of doing that specific operation. This means the device has no appropriate kernel (implementation to execute the operation). In these cases TensorFlow™would throw an exception unless Soft Placement is activated. If thats the case, TensorFlow™falls back to the CPU. [5]

M. Parallel execution

Once all the nodes are assigned to a specific device, the evaluations can start. As there are multiple nodes assigned to a single device TensorFlow™manages two kinds of thread pools. The inter-op thread pool which is used for parallel evaluation of different nodes and the intra-op thread pool which handles node intern parallelization if the operation has a multi-threaded kernel. [5]

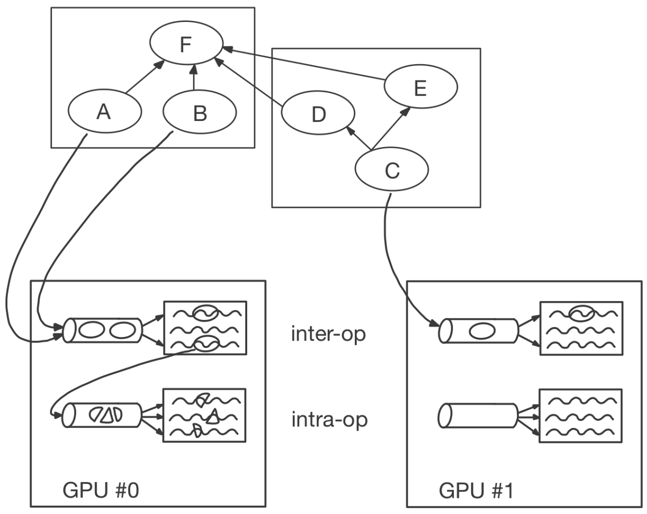

Fig. 11. *Thread Pools*

N. Dependencies

As some nodes require the outputs of other nodes as an input, there may be multiple dependencies. Therefore TensorFlow™counts the dependencies for every node in the graph and starts evaluating the zero dependency nodes. After these nodes are evaluated, other nodes which depend on them are ready to evaluate. In some cases it may be useful to delay the evaluation of certain nodes although all their dependencies are fulfilled. One case would be some kind of memory heavy operation. In this situation it would be useful to postpone the operation until the result is immediately needed in order to save some RAM which may be used by other operations. Another situation would be I/O heavy operations which depend on external data. Its recommended to execute them serially in order to keep some bandwidth for other operations instead of totally occupying all of it. [5]

REFERENCES

[1] S. Haykin and N. Network, "A comprehensive foundation," *Neural networks*, vol. 2, no. 2004, p. 41, 2004.
[2] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2010.
[3] A. Abraham, "Artificial neural networks," *handbook of measuring system design*, 2005.
[4] R. S. Snell, *Clinical neuroanatomy*. Lippincott Williams & Wilkins, 2010.
[5] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[6] M. Cokun, H. Guruler, A. Istanbullu, and M. Peker, “Determining the appropriate amount of anesthetic gas using dwt and emd combined with neural network,” vol. 39, p. 1=10, 02 2015.

[7] T. Rashid, *Make your own neural network*. CreateSpace Independent Publishing Platform, 2016.

[8] D. C. Park, M. El-Sharkawi, R. Marks, L. Atlas, and M. Damborg, “Electric load forecasting using an artificial neural network,” *IEEE transactions on Power Systems*, vol. 6, no. 2, pp. 442–449, 1991.

[9] W. G. Baxt, “Use of an artificial neural network for the diagnosis of myocardial infarction,” *Annals of internal medicine*, vol. 115, no. 11, pp. 843–848, 1991.

[10] Y. Wu, M. L. Giger, K. Doi, C. J. Vyborny, R. A. Schmidt, and C. E. Metz, “Artificial neural networks in mammography: application to decision making in the diagnosis of breast cancer.,” *Radiology*, vol. 187, no. 1, pp. 81–87, 1993.

[11] E. Guresen, G. Kayakutlu, and T. U. Daim, “Using artificial neural network models in stock market index prediction,” *Expert Systems with Applications*, vol. 38, no. 8, pp. 10389–10397, 2011.

[12] TensorFlow, “Official tensorflow website,” *TensorFlow*, 2018.

[13] C. B. Lang and N. Pucker, *Mathematische Methoden in der Physik*, vol. 2. Springer, 2005.

[14] P. G. Chip Huyen, Michael Straka, “Cs 20: Tensorflow for deep learning research,” in *TensorFlow*, 2018.