Neural Networks and Analog Circuits: An Overview

Christoph Schröter

Abstract

This paper explains how neural networks work and presents research about their implementation with more focus on analog methods. Thereby different layer types and the neuron as the smallest part of the network are described before the main learning method for neural networks, backpropagation, is briefly covered. A short conclusion about the current state of neural networks and their hardware/software implementation is given, as well as hints into future research topics.

1 Introduction

Neuronal networks (NNs) are becoming increasingly relevant for industry and research. Their power stems from being able to approximate an arbitrary function from just input and output values through training and backpropagation. Therefore they are heavily used already today, for example in image recognition which can be used in medical applications or for autonomous systems such as automotives or roboters. Even in manufacturing they can be used productively, let it be for product design or quality inspection.

In recent research, analog NNs are occurring more and more frequently, as further improvement in general purpose processors slows down, while the demand for powerful NNs increases, slowly forcing research away from digital ones.

Even though over 30 years ago research has been conducted already in this topic [6, 8, 14, 15, 19], new developments and improvements are still being made with success. Therefore, this paper presents past and recent research in this topic to then summarize the current state of this research field and give a brief outlook into the possible future improvements.

Firstly though, the basic structure of NNs is explained to better understand the developments of the topic.

2 Neural Network Structure

Since for implementing a NN whether its digital or analog the structure is crucial, this section provides a brief summary about basic NN components. Because NNs try to recreate the structure of a brain, there are similarities between the two, however, these are not relevant for implementation and therefore not explained in this paper.

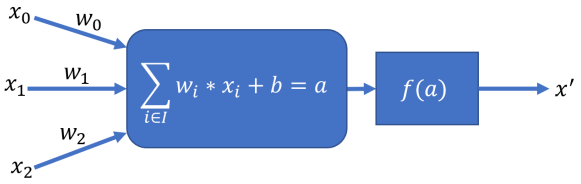

Figure 1. Structure of a neuron. The inputs x_i are multiplied with their corresponding w_i, after that, the bias b is added. Lastly an activation function is applied to determine the output x'.

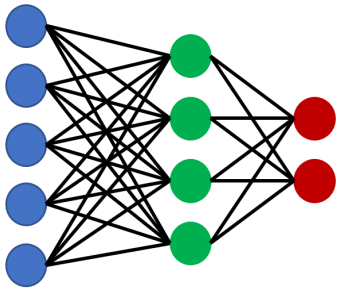

Figure 2. Visualization of a neural network with only fully connected layers. Each input (blue) and output (red) is connected to every neuron in the hidden layer (green).

2.1 Neuron

The smallest piece of a neural network is a *neuron* (also called *perceptron*), whose structure is shown in Figure 1. It sums up an arbitrary number of input values x_i multiplied with their individual weights w_i and adds a bias b to it. The resulting value is called the *activation* a and gets passed to the next neuron (or the output) after applying the *activation function* f. This function plays a huge role in the networks performance, can be selected almost arbitrary and is a research topic on its own. However, simpler activation functions tend to outperform more complex ones, presumably because of a more difficult training process (see section 3). Currently the most widespread function is the *Rectified Linear Unit* which is defined as $ReLU(a) = max(0, a)$. [12]

2.2 Layer

The NN itself consists out of multiple *layers* which in turn consist of the neurons described in subsection 2.1. Figure 2 shows a basic network with three layers, one of

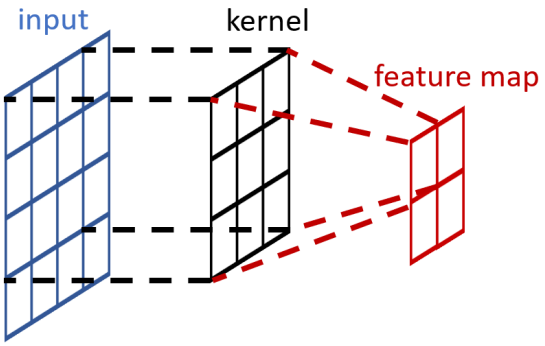

Figure 3. Schematic drawing of a convolutional layer. The two-dimensional input is convolved with a 3×3 kernel to calculate a 2×2 feature map.

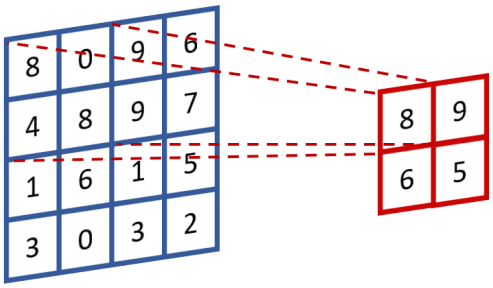

Figure 4. Schematic drawing of a max pooling layer. It extracts the maximum value from a 2×2 array of the two-dimensional input on 4 different positions.

which is considered a *hidden layer*, which means that it is neither input nor output. Some common layer types are explained in the following subsections.

2.2.1 Fully-connected Layer. A *fully-connected layer* denotes a layer, where each neuron consumes input from each neuron of the preceding layer as depicted in Figure 2. Having a lot of fully connected layers can lead to higher processing time and electrical power requirements due to a high number of trainable parameters, even though all the inputs might not be required in most of the layers [4].

2.2.2 Convolutional Layer. *Convolutional layers* as shown in Figure 3 are used to exract features from their input. Therefore, they use a *kernel*, which stands for an array of weights which is smaller than the input. This kernel can then be applied to multiple positions of the input (by "shifting" over it) with a convolution. This leads to a so-called *feature map*. Since the kernel is smaller than the input, convolutional layers are good at detecting for example features of an image (e.g. eyes of a human) regardless of their position. If the output is not supposed to be smaller than the input, *padding* can be used to extend the input, for example by using extrapolation. Typically multiple convolutional layers will be used in parallel to extract multiple features. As this layer type is the core technique, it is name giving for *convolutional neural networks* (CNNs). Layers which perform the inverse of convolutional ones are called *deconvolutional layers*. [7]

2.2.3 Pooling Layer. *Pooling layers* reduce the size of their input by performing a simple pooling operation, such as selecting the maximum (*max pooling*) or average (*average pooling*) value, on different regions of the input. An example of a max pooling layer can be seen in Figure 4. These layers typically follow convolutional ones, as they can be used to extract the existance of a feature in a feature map independent of its position and therefore add robustness for example when rotating or translating an input image. [7]

2.2.4 Normalization Layer. If activation functions such as ReLU (see subsection 2.1) are used in a neural network, inputs into the following layers can take on very different values. Thus, normalizing data inside the network in *normalization layers* can have significant benefits. It allows an easier analysis by the model and speeds up training, as the layers after the normalization one can adapt to a single distribution of inputs. [4]

2.3 Classification

As already explained in Figure 2.2.2, CNNs with their convolutional layers are highly performant when extracting features is required. This makes them especially applicable in topics such as computer vision, image classification, video processing or speech recognition [2]. *Deep neural networks* (DNNs) stand for NNs with many hidden layers, while *recurrent neural networks* (sometimes also called *feed-backward NNs*) correspond to NNs, which unlike the up to this point only discussed *feed-forward NNs* contain feedback connections to previous layers or inbetween them. Having feedback inside the network allows better processing of event sequences or time-based data, thus fitting those NNs better to applications such as language learning or adaptive processes in autonomous systems [11]. As classification of NNs is not the main scope, it needs to be noticed that there are many other types of NNs, which are not further discussed in this paper.

3 Neural Network Training

As an important part of creating a (feed-forward) NN is its training, it is briefly explained in this section. To begin with, sufficient training data needs to be available, in that there must exist a set of inputs to the network with the corresponding correct output for those inputs.

Moreover, since only training by feeding data forward through the network does not lead to acceptable results, *backpropagation* still generally is the main method for training NNs. As an example, Wilamoski et al. [17] proposed a method which was way quicker in the training process, but did not yield satisfactory outcomes. The results could be improved by increasing the number of neurons, however, this lead to the network being *overtrained*, which means that the network performed well on data it had trained on, but could not handle new inputs (which were not part of the training) correctly.

3.1 Backpropagation

Since backpropagation itself is just a way to calculate derivatives, there are other applications than NNs. In most cases a simpler version of backpropagation is used for NNs, as proposed by Werbos et al. [16], as it would require a lot of manual mathematics which can not be translated to executable code easily. Moreover, the derivatives would be highly dependent on the application itself and therefore could not be reused.
The general idea is to propagate the test input through the network to receive an output, which is then compared with the correct output of the training case. This comparison is done via a so-called *loss function*. Just like with the activation function (see subsection 2.1), there exist numerous different versions. Once the loss has been determined, its derivative with respect to the parameter in the network (e.g. weight, bias) calculated. For this to work, every activation function, as well as the loss function has to be differentiable, however there are workarounds for non-differentiable functions such as ReLU. With the derivatives availabe, *gradient descent* can take place. The gradient of the function is calculated and scaled by a (predetermined) *step-size*, before it gets subtracted from the parameter corresponding to the specific entry of the gradient. This process can be executed iteratively which leads to minimizing the function, in the case of NNs the loss function, thereby adjusting the parameters such that the output is closer to the correct output when feeding the network with the same training cases again.

4 Analog Implementations

This section provides an overview of selected implementations of NNs in past and recent research.

4.1 Overview

4.1.1 Early Research. As stated in section 1 already, research on analog implementations of neural networks has been conducted many years ago. Graf et al. (1989) [6] as one of those early works found some interesting aspects. First, the sum of products in a neuron is easily implemented with Kirchhoff's law computing the sum, while the product can be performed by a simple resistor. Second, interconnected (fully-connected) layers were difficult to create because of the high number of required connections, which would have led to a lot of space on the chip used up just for those layers. This point is not as prevalent today, as circuitry has decreased extremely in physical size since then. Another problem was the precision of the analog circuits, which was not as high back then. Therefore Graf et al. recommended to train the NN digitally before its analog implementation and to use a learning algorithm which can tolerate low precision network connections. Flexibility in terms of reprogrammable interconnections could be achieved by using a storage cell to contain the weight which then is applied to the actual connection element (e.g. resistor), but would use up a lot of space on the chip too. They concluded that designing an analog chip for a NN is a tradeoff between functionality and network size. If for example training should take place on the chip too or high resolution interconnections are required, the network must be smaller than without those features. Another conclusion was that neural networks in analog circuits can compute results way faster than their digital counterparts, but were not as flexible nor precise. An aspect not mentioned by Graf et al. is the electrical power consumption, which presumably was not as important at that time, because networks were not close to being as big as they are today.
Another example of pioneer work in this area is Zurada et al. [19], who picked up on the problems of that time three years later and tried to combine the advantages of digital and analog implementations by adding a digital-to-analog interface to the chip which allowed the parameters and interconnection of the network to be programmable. Moreover, they implemented on-chip unsupervised (Hebbian[1]) learning as they assessed on-chip learning as being essential for most appliations. Even though their paper was a huge step forwars in terms of design of analog NNs, they correctly conclude that future designs will become way more complex as the networks get more complicated. Also, desirable properties for future NN chips are listed, such as being affordable, being reprogrammable and not taking up a lot of space. In contrast to Graf et al. power consumption as an important criteria is listed too.

4.1.2 Recent Research. Today, many NNs are trained and run on accelerators such as graphics processing units (GPUs[2]). This makes high flexibility in terms of network

[1] https://julien-vitay.net/lecturenotes-neurocomputing/4-neurocomputing/5-Hebbian.html
[2] https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html

design possible while still having acceptable performance. However, it leads to high power consumption and hardware cost and can not reach the performance of an analog NN. With NNs getting a lot bigger to handle more complex tasks, this problem could get worse in the future. Therefore research efforts are made in the direction of analog NNs again. For example Schemmel et al. [13] proposed an architecture to integrate programmable analog NNs on wafers which have low power consumption and high fault tolerance. A problem that is not fully solved yet is on-chip learning, as e.g. Krestinskaya et al. [9] implemented the original backpropagation algorithm in analog circuits, however concluded that their implementation needs to be further optimized in terms of area usage and power consumption in the future.

Other papers propose further research in the topic of accelerators for NNs, which use parts realized in software and hardware to combine the benefits of each approach. So did Ambrogio et al. [1] in 2018. They achieved similar results as GPU trained NNs in terms of accuracy, while heavily outperforming them when it comes to computational speed and energy efficiency. Since field programmable gate arrays (FPGAs[3]), bridge the gap between hardware and software implementations in general, they are an obvious choice when it comes to accelerating a NN. Therefore Lu et al. [10] proposed an accelerator based on a FPGA which operates on sparse CNNs. These were chosen, because removing unnecessary interconnections in a CNN (*pruning*) is highly popular, especially for deep CNNs, to reduce the computational effort. Their experiments show that their accelerator architecture was way faster than previous ones for dense CNNs. When comparing with high-end GPUs, similar speeds could be achieved, while being more than 10 times as energy efficient. Because there are many publications about the topic of NN accelerators, Deng et al. [5], Bouvier et al. [3] and Capra et al. [4] created surveys discussing this topic. Both state that the hardware needs to be optimized further as algorithms get more complex and the demand for neural networks increases. Especially engergy efficient architectures are important for further research, where memory, because it is very engergy consuming, needs to play a big role in.

Fully on-chip implemented NNs were discussed by Yingge et al. [18]. They conclude that further research needs to be done in terms of NN structure and processing technology to increase their efficiency and lower production cost. Also they claim, that dedicated development environments for (D)NNs would improve the designing process by making it simpler and yielding more results.

[3] https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

5 Conclusion

This paper provided an overview over the general structure of neural networks. After that, research on implementation approaches, especially analog and mixed hardware-software ones, was discussed.

As it turns out, most research today is done for accelerators of neural networks, which try to use both flexibility from software parts and efficiency and speed from hardware parts. Continuous improvements are required in future research in order to keep up with the demand for NNs. One of the main fields of potential improvement, the energy efficiency, might become even more important as NNs are getting bigger and bigger and more widespread.

References

[1] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M Shelby, Irem Boybat, Carmelo Di Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan CP Farinha, et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60–67, 2018.

[2] Dulari Bhatt, Chirag Patel, Hardik Talsania, Jigar Patel, Rasmika Vaghela, Sharnil Pandya, Kirit Modi, and Hemant Ghayvat. Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10 (20), 2021. ISSN 2079-9292. doi: 10.3390/electronics10202470. URL https://www.mdpi.com/2079-9292/10/20/2470.

[3] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. Spiking neural networks hardware implementations and challenges: A survey. *J. Emerg. Technol. Comput. Syst.*, 15(2), apr 2019. ISSN 1550-4832. doi: 10.1145/3304103. URL https://doi.org/10.1145/3304103.

[4] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, 8:225134–225180, 2020. doi: 10.1109/ACCESS.2020.3039858.

[5] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108 (4):485–532, 2020. doi: 10.1109/JPROC.2020.2976475.

[6] H.P. Graf and L.D. Jackel. Analog electronic neural network circuits. *IEEE Circuits and Devices Magazine*, 5(4):44–49, 1989. doi: 10.1109/101.29902.

[7] Tianmei Guo, Jiwen Dong, Henjian Li, and Yunxing Gao. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 721–724, 2017. doi: 10.1109/ICBDA.2017.8078730.

[8] H. Harrer, J.A. Nossek, and R. Stelzl. An analog implementation of discrete-time cellular neural networks. *IEEE Transactions on Neural Networks*, 3(3):466–476, 1992. doi: 10.1109/72.129419.

[9] Olga Krestinskaya, Khaled Nabil Salama, and Alex Pappachen James. Analog backpropagation learning circuits for memristive crossbar neural networks. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018. doi: 10.1109/ISCAS.2018.8351344.

[10] Liqiang Lu, Jiaming Xie, Ruirui Huang, Jiansong Zhang, Wei Lin, and Yun Liang. An efficient hardware accelerator for sparse convolutional neural networks on fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 17–25, 2019. doi: 10.1109/FCCM.2019.00013.

[11] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[12] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017. URL https://arxiv.org/abs/1710.05941.

[13] Johannes Schemmel, Johannes Fieres, and Karlheinz Meier. Wafer-scale integration of analog neural networks. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 431–438, 2008. doi: 10.1109/IJCNN.2008.4633828.

[14] D.B. Schwartz, R.E. Howard, and W.E. Hubbard. A programmable analog neural network chip. *IEEE Journal of Solid-State Circuits*, 24(2):313–319, 1989. doi: 10.1109/4.18590.

[15] E.A. Vittoz. Analog vlsi implementation of neural networks. In *IEEE International Symposium on Circuits and Systems*, pages 2524–2527 vol.4, 1990. doi: 10.1109/ISCAS.1990.112524.

[16] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

[17] Bogdan M. Wilamowski and Hao Yu. Neural network learning without backpropagation. *IEEE Transactions on Neural Networks*, 21(11):1793–1803, 2010. doi: 10.1109/TNN.2010.2073482.

[18] Huo Yingge, Imran Ali, and Kang-Yoon Lee. Deep neural networks on chip - a survey. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 589–592, 2020. doi: 10.1109/BigComp48618.2020.00016.

[19] J.M. Zurada. Analog implementation of neural networks. *IEEE Circuits and Devices Magazine*, 8(5):36–41, 1992. doi: 10.1109/101.158511.