Deep Learning on SpiNNaker

Master Thesis

Jonas Fassbender jonas@fassbender.dev

In the course of studies
High Performance Computing with Data Science

For the degree of
Master of Science

The University of Edinburgh

First supervisor: Caoimhín Laoide-Kemp
EPCC, University of Edinburgh

Second supervisor: Dr Kevin Stratford
EPCC, University of Edinburgh

Third supervisor: Dr Alan Stokes
APT, University of Manchester

Edinburgh, August 2020

Declaration

I declare that this dissertation was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Jonas Fassbender
August 2020

Abstract

Contents

List of Figures

1. Introduction

Deep learning is revolutionizing the world. It has become part of our daily lives as consumers, powering major software products—from recommendation systems and translation tools to web search (LeCun et al., 2015). Major breakthroughs in fields like computer vision (Krizhevsky et al., 2012) or natural language processing (Hinton et al., 2012) were achieved through the use of deep learning. It has emerged as a driving force behind discoveries in numerous domains like particle physics (Ciodaro et al., 2012), drug discovery (Ma et al., 2015), genomics (Leung et al., 2014) and gaming (Silver et al., 2016).

Deep learning has become so ubiquitous that we are changing the way we build modern hardware to account for its computational demands. From the way edge devices like mobile phones or embedded systems are built (Deng, 2019) and modern CPUs (Perez, 2017) to specialized hardware designed only for deep learning models, such as Google's tensor processing unit (TPU) (Jouppi et al., 2017) or NVIDIA's EGX Edge AI platform (Boitano, 2020). Whole state-of-the-art supercomputers are built solely for deep learning. An example would be a supercomputer built by Microsoft for OpenAI, which is part of the Azure cloud (Langston, 2020).

Hardware manufacturer are faced with a major challenge in meeting the computational demands arising from inference, and more importantly, training deep learning models. OpenAI researchers have estimated that the computational costs of training increases exponentially; approximately every 3.4 months the cost doubles (Amodei et al., 2019). Amodei et al. (2019) claims the deep reinforcement learning agent AlphaGo Zero—the successor of the famous AlphaGo program, which was able to beat Go world champion Lee Sedol (Silver et al., 2017)—to be the system with the highest computational demands of approximately 1850 petaflop/s-days. AlphaGo Zero was trained for 40 days on a machine with 4 TPUs (Silver et al., 2017). With the end of Moore's Law (Loeffler, 2018), chip makers have to get creative in scaling up computing, the same way machine learning researchers are scaling up their models (Simonite, 2016). Therefore production and research into new hardware designs for deep learning are well on the way.

Another field which has high computational demands for very specific tasks and algorithms is computational neuroscience. Computational neuroscience has long been linked to deep learning, which has its origin in research done by neuroscientists (McCulloch and Pitts, 1943). While in the recent past deep learning research has been more focused on mathematical topics like statistics and probability theory, optimization or linear algebra, researchers are again looking to neuroscience to further improve the capabilities of deep learning models (Marblestone et al., 2016).

But the algorithms developed by computational neuroscientists are not the only aspect drawing attention from the deep learning community. Computational neuroscience has a long standing history of developing custom hardware for the efficient modeling of the human brain, so called neuromorphic computing. Neuromorphic computing—a computer architecture inspired by the biological nervous system—has been around since the 1980s (Mead, 1989). Today, neuromorphic computers are being developed to meet the demands for efficient computing needed to run large-scale spiking neural networks used for modeling brain functions (Furber, 2016). While being developed mainly for the task of modeling the human brain, deep learning has been linked to neuromorphic computing, especially in the context of commercial usability (Gomes, 2017). Both the low energy demands of neuromorphic computers—such as IBM's True North (Cassidy et al., 2013) or The University of Manchester's Spiking Neural Network Architecture (SpiNNaker) (Furber et al., 2006)—and their scalability and massive-parallelism are intriguing for two very important use cases of deep learning:

(i) edge computing, for example robotics and mobile devices, (ii) supercomputers and the cloud-era (Gomes, 2017).

This thesis investigates the performance of SpiNNaker machines for deep learning by training the state-of-the-art computer vision model ResNet-50 (He et al., 2015) under the closed division rules of the MLPerf benchmark (Mattson et al., 2019). In order to benchmark ResNet-50 on SpiNNaker a prototypical implementation was developed as part of this thesis.

- here a paragraph about the results

Section 2 presents the background of this thesis. An introduction to deep learning is given in Section 2.1, as well as an overview of the benchmark in Section 2.2. Section 2.3 describes the SpiNNaker architecture and compares it to current deep learning hardware. Related work can be found in Section 3. Section 4 presents the architecture of the prototype developed for benchmarking and Section 5 presents the benchmarks and its results. In Section 6 the results of the benchmark are discussed, as well as the development process. Section 7 contains the conclusion, while Section 8 outlines the next steps for further increasing the performance of SpiNNaker by enhancing the prototype.

2. Background

This section summarizes the background knowledge needed in the following sections. First a short introduction to deep learning is given in Section 2.1. The main focus lies on the basic concepts and those important for computer vision and therefore the prototype developed as part of this thesis. Next, Section 2.2 outlines the context of the conducted benchmark presented in Section 5. Lastly the SpiNNaker neuromorphic computer architecture is described in Section 2.3. SpiNNaker is also compared against the two state-of-the-art hardware solutions for deep learning which currently produce the best performance in training and inference, namely general purpose graphical processing units (GPGPUs) and Google's tensor processing unit (TPU).

2.1 An Introduction to Deep Learning

While it may seem that deep learning is a recent development in the field of artificial intelligence—due to all the hype and all the announced breakthroughs—it is actually around since the 1940s. McCulloch and Pitts (1943) first described the McCulloch-Pitts neuron as a simple mathematical model of a biological neuron, which should mark the origin of what today is known as deep learning.

Even though deep learning models today are still called *artificial neural networks*, due to their historical context, they are quite different from *spiking neural networks* (which SpiNNaker was designed to run efficiently). While the former has been described as "just nonlinear statistical models" (Hastie et al., 2009), the latter incorporated findings about biological neurons and is therefore closer related to how the nervous system works (Maass, 1997). Spiking neural networks are mostly used for simulation rather than inference like deep learning models.

The history of deep learning can be broken down into three distinct phases and only during the last phase the methodology is called deep learning (Goodfellow et al., 2016); arguably the reason why deep learning seems to be a new development. The first phase, where deep learning was known as cybernetics, ranged from the 1940s to the 1960s (Goodfellow et al., 2016). Like stated above, it was the time where the first biologically inspired representations of neurons where

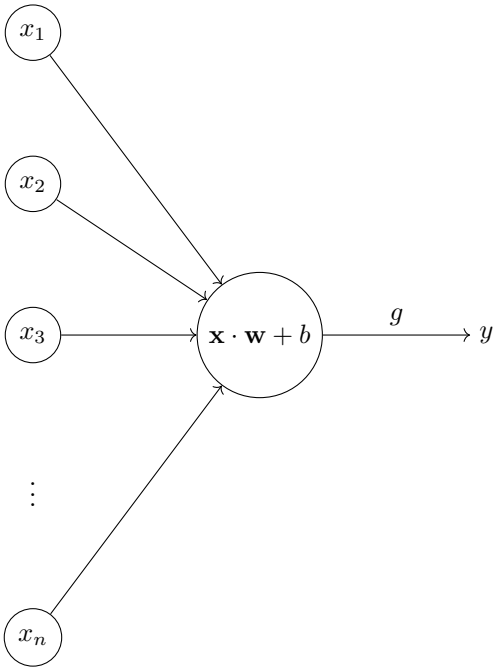

Figure 1: Schema of a perceptron.

developed. Rosenblatt (1958) presents the first model, a single trainable artificial neuron known as the perceptron (see Figure 1).

Today's perceptron receives a real-valued n-vector $\mathbf{x}$ of *input* signals and builds the dot product with another real-valued n-vector known as *weights* $\mathbf{w}$: $\mathbf{x} \cdot \mathbf{w} = \sum_{i=1}^{n} x_i w_i$. The *bias* b is added to the dot product. $\mathbf{x} \cdot \mathbf{w} + b$ is then passed to the *activation function* g—some fixed transformation function appropriate for the application domain—and $y = g(\mathbf{x} \cdot \mathbf{w} + b)$ is the output of the perceptron.

During *supervised learning*, we have a set of *examples*. Each example consists of an *input* vector $\mathbf{x}$ and a associated *label* y generated by an unknown function $f^*(\mathbf{x})$. A perceptron can be trained to approximate $f^*(\mathbf{x})$. We can describe a perceptron as the mathematical function

$$y = f(\mathbf{x}; \mathbf{w}, b) = g(\mathbf{x} \cdot \mathbf{w} + b). \tag{1}$$

$f(\mathbf{x}; \mathbf{w}, b)$ is known as a *(statistical) model* with $\mathbf{w}$ and b as its *trainable parameters* which are trained/learned in order to approximate f^* with f. How a network of perceptrons—a more complex statistical model better suited for real world applications—is trained via backpropagation and gradient descent, will be explained below.

The second historical phase of deep learning is known as connectionism (1980s-1990s) (Goodfellow et al., 2016). Its main contributions to today's knowledge was the backpropagation algorithm (Rumelhart et al., 1986a) and the approach of parallel distributed processing (Rumelhart et al., 1986b,c), which provided a mathematical framework around the idea that a large number of simple computational units (e.g. the perceptron) can achieve intelligent behavior when connected together

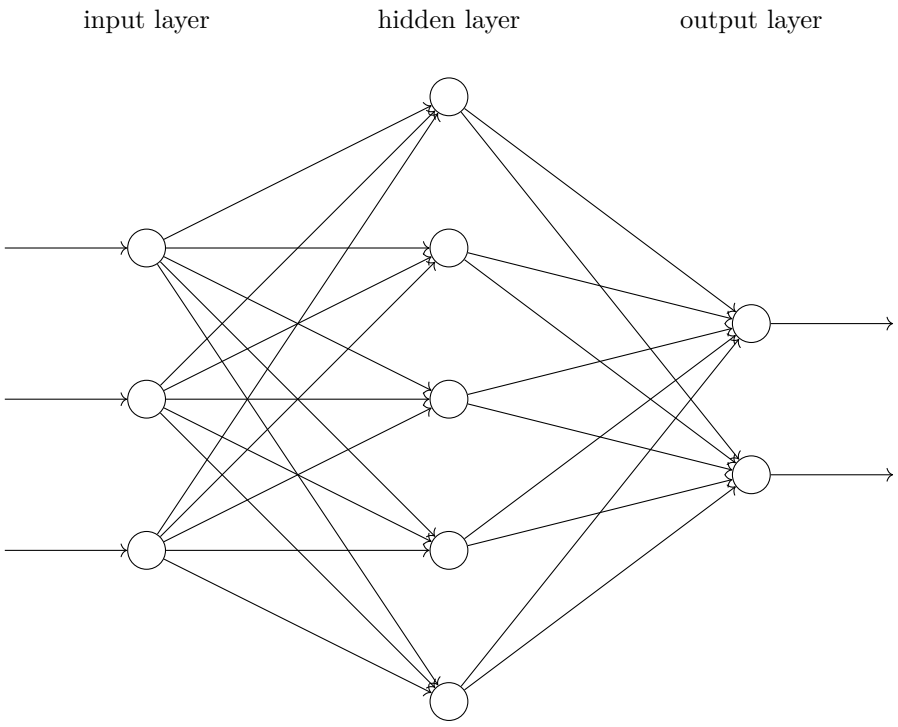

Figure 2: Schema of a MLP or feedforward neural network.

(Goodfellow et al., 2016). Backpropagation enabled the training of networks of perceptrons—artificial neural networks.

The quintessential artificial neural network is the *multilayer perceptron* (MLP), also called a *feedforward neural network* (see Figure 2) (Goodfellow et al., 2016). The MLP consists of multiple perceptrons organized in *layers*. Layers are connected successively such that the output of each of its perceptrons reaches all perceptrons in the next layer it is connected to. Such a layer is known to be *fully-connected* or *dense*. No cycle exists between perceptrons; the MLP is a directed acyclic graph. Unlike the single layer perceptron, the MLP has at least one *hidden layer*.

A MLP can also be represented as a statistical model $f(\mathbf{x}; \theta)$. Computing $f(\mathbf{x})$—also called *inference* or the *forward pass*—can be described as a layer-wise composition of functions $f^{(1)}, f^{(2)}, \ldots, f^{(l)}$, each function $f^{(i)}, i < l$ being a hidden layer and $f^{(l)}$ being the output layer. The perceptron has the weight vector $\mathbf{w}$ and the bias b as its parameters (see Equation 1). The parameters of a layer is the combination of $\mathbf{w}$ and b for each of its perceptrons. For example, if the first hidden layer contains m perceptrons and $\mathbf{x}$ is a n-vector, then the parameters of $f^{(1)}$ would be a matrix $\mathbf{W} : n \times m$ and a m-vector $\mathbf{b}$. The output of layer $f^{(1)}$ would be a m-vector computed as follows:

$$f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = g(\mathbf{x}^\top \mathbf{W} + \mathbf{b})^\top. \tag{2}$$

The second layer takes the output of the first and so forth. The forward pass of the MLP is computed as:

$$y = f^{(l)}(f^{(l-1)}(\dots f^{(1)}(\mathbf{x}))). \tag{3}$$

The backpropagation algorithm is a way to train the parameters of a MLP (or other deep learning models) so that it approximates the unknown function f^* which generates the labels of the examples we have in our data set. The data set used for training a model is called the *training set*. Additionally to the training set there normally exists a *test set* with examples the model has not seen before. The test set is used to determine the generalization performance of the model. Backpropagation is an algorithm that allows to train a deep learning model with *(stochastic or batch) gradient descent*. For example, $\hat{y} = f(\mathbf{x})$ and y is the true label (y and $\hat{y}$ are k-vectors), the error of f is computed using a *loss function* L, for example mean squared error: $1/k \sum_{i=1}^{k}(y_k - \hat{y}_k)^2$. In order to get the gradients of the weights of the output layer we calculate the derivative of the loss according to each weight w_{ij} in $\mathbf{W}$ with the chain rule:

$$\frac{\delta L}{\delta w_{ij}} = \frac{\delta L}{\delta g}\frac{\delta g}{\delta h}\frac{\delta h}{\delta w_{ij}}, \tag{4}$$

h being $f^{(l-1)\top}\mathbf{W} + \mathbf{b}$. w_{ij} is updated by performing the stochastic gradient descent[1]:

$$w_{ij}^{+} = w_{ij} - \mu\frac{\delta L}{\delta w_{ij}}, \tag{5}$$

with μ as the *learning rate*. The same procedure is applied to the following hidden layers, only that the loss is now computed as:

$$L^{(l-1)} = \sum_{i=1}^{k}\frac{\delta L}{\delta h_k}, \tag{6}$$

which can be interpreted as the sum over the error of each perceptron in the previous layer.

Hornik et al. (1989) demonstrated that a non-linear MLP (the activation functions are non-linear transformations of $\mathbf{x}^\top\mathbf{W} + \mathbf{b}$) can overcome the famous XOR problem of a single layer perceptron demonstrated in Minsky and Papert (1969). Another major contribution of the phase of connectionism was the neocognitron (Fukushima, 1980), the origin of today's *convolutional neural networks* (CNNs)—which are the state-of-the-art approach for building computer vision models—and the application of the backpropagation algorithm to fully automate the training of CNNs (LeCun et al., 1989).

Goodfellow et al. (2016) claims that the third and current phase of deep learning—where the name deep learning was established—starts with Hinton et al. (2006) describing a new learning algorithm called greedy layer-wise pretraining, which they applied to deep belief networks. Greedy layer-wise pretraining was soon generalized to work with other deep artificial neural network architectures (Ranzato et al., 2006; Bengio et al., 2007). While these papers may have resulted in the

1. Or (batch) gradient descent. With gradient descend the whole training set is passed through the MLP before the weights are updated with the sum over the loss of each example in the training set. Batch or Mini-batch gradient descent takes a subset of the whole training set and updates the weights after each mini-batch. Deep learning models are normally trained by passing the training set multiple times to the model. Each pass over the whole training set is called an *epoch*.

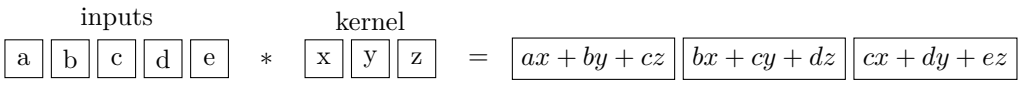

Figure 3: Example of a 1D cross-correlation operation with a kernel size of three, a single channel, a single filter, a stride of one and valid padding.

term deep learning, they were not the reason for the resurrected interest in this methodology. The two most important factors are the increase of available data and computation. The former enables better generalization while the latter allows training bigger models (more hidden layers—the *depth* of the neural networks increased) which can solve more complex problems (Goodfellow et al., 2016).

Like the perceptron, "neurons" in a *convolutional layer* are inspired by findings of neuroscientists. In this case by research done by Hubel and Wiesel about the mammalian visual cortex (Hubel and Wiesel, 1959, 1962, 1968). CNNs are just deep learning models which have at least one convolutional layer. They are applied to problems which have a grid-like topology, like time-series (1D), images (2D) or videos (3D) (Goodfellow et al., 2016).

Unlike dense layers of perceptrons, convolutional layers do not apply a full matrix multiplication $\mathbf{x}^\top \mathbf{W}$ but instead a linear operation $*$ called convolution. A one dimensional discrete convolution can be described as:

$$s(i) = (x * w)(i) = \sum_n x(i+n)w(n). \tag{7}$$

Equation 7 is not really a convolution but is referred to as *cross-correlation*. Unlike true convolution, cross-correlation is not commutative (Goodfellow et al., 2016). But commutativity is not a factor in practice, so many deep learning libraries, like Keras (Chollet et al., 2015) or the prototype developed for this thesis implement cross-correlation rather than true convolution. Below convolution will refer to cross-correlation.

In the case of deep learning, x is a nD array called the *input* and w is another nD array referred to as the *kernel*. The kernel elements are the trainable parameters (Goodfellow et al., 2016). In Equation 7, x and w are one dimensional. If we say x to be a m-vector, the function $x(i)$ is defined as:

$$x(i) = \begin{cases} x_i & \text{if } 1 \leq i \leq m \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

n is the size of the kernel in the first dimension. Figure 3 shows an example of how the output of a 1D convolutional layer is computed. Figure 4 shows the schema of the convolutional layer performing the operation from Figure 3.

Normally a convolutional layer does not consist of a single convolution, but applies multiple kernels to the output of the previous layer. A single convolution is called a *filter* and a layer consists of a predefined amount of filters, each with its own kernel (Brownlee, 2019). The output of a convolutional layer is often called a *feature map* (Goodfellow et al., 2016). Even though an image may seem to be a two dimensional structure of pixels, in most cases it is actually three dimensional, the third dimension being the RGB color values for each pixel. The third dimension of the three RGB colors are called the *channels* (Goodfellow et al., 2016). For example, we have a data set of

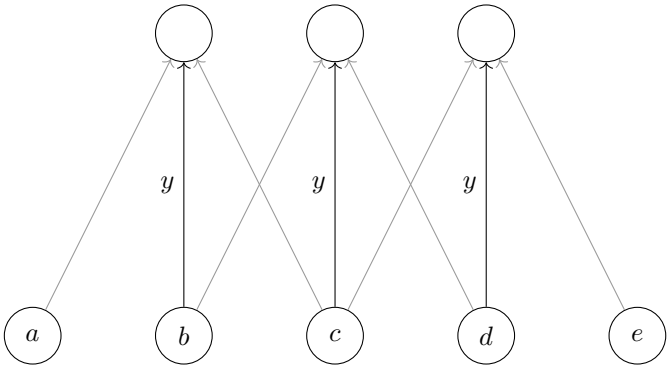

Figure 4: Schema for the convolutional layer performing the convolution shown in Figure 3. Each neuron represents one convolution. The schema shows the property of shared weights and sparse connectivity (Goodfellow et al., 2016). The black edges all have the same associated weight y, while one can see that there are much less edges compared to a dense layer shown in Figure 2.

images with 256×256 pixels and three channels (red, green and blue). We pass the image to a convolutional layer with a 3×3 kernel shape and 64 filters. A kernel consists of 18 elements, the kernel size (for the two spatial dimensions) times the three channels of each pixel. The shape of the feature map of that layer—if we assume "same" padding (see below)—would be $256 \times 256 \times 64$, so the next layer would have 64 channels.

There are two more notable concepts of convolutional layers: *stride* and *padding*. The former refers to skipping convolutions in order to reduce the computational cost at the expense of less exact feature extraction (patterns may not be detected by the model due to the increased inaccuracy). The latter is a way of dealing with vanishing spacial dimensions of the feature map if we only perform convolutions on "valid" inputs ($1 \leq i \leq m$ in Equation 8). "valid" padding refers to the fact that the input has no padding, which means the feature map of the convolutional layer will have its kernel size minus one less neurons than its input (see Figure 4). "Same" padding would be to add enough zeros evenly above and below the valid input (along each spacial dimension) so that the feature map of the convolutional layer will have the same spacial dimensions as its input (see Figure 5 (Goodfellow et al., 2016).

1. Pooling

2.2 Benchmarking Deep Learning Systems for Computer Vision

1. short section about imagenet and ilsvrc and their importance for computer vision
2. ResNet50 and residual stuff
3. paragraph about MLPerf

2.3 SpiNNaker as a Neuromorphic Computer Architecture

1. describe spinnaker and the spinnaker architecture

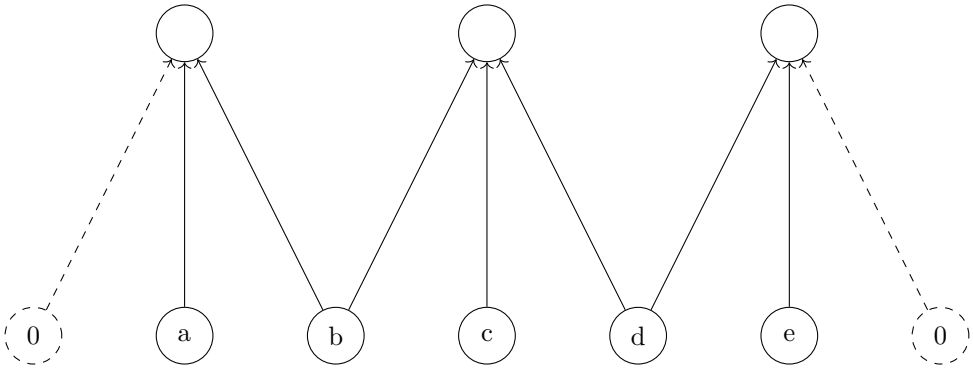

Figure 5: Example showing a layer with same padding and a stride of two.

2. compare to other DL accelerators (GPGPUs and TPUs)

3. Related Work

1. SNNToolbox for translating DNNs to SNNs (only inference)
2. TrueNorth has a paper about its DL implementation
3. (optional) The 2011 paper about mapping MLP's and recurrent networks onto SpiNNaker

4. Deep Learning on SpiNNaker

- concepts (layers, neurons, …)
- partitions and how that allows me to use min keys to discern between what I have received
- communication structure (partitions and global partition manager)
- ping-pong
- graph structure (especially focused on edge and host–SpiNN communication)
- interpreting neurons as domain decomposition over linear algebra compute graph
- backward pass: gradients computed two times so comm fabric is not overly used by unique partitions
- How I crushed nd-kernels into a single blog of weights (same for 2D convolutions even though less interesting)

5. Benchmark

6. Discussion

- space used inefficiently (cores and memory) → better domain decomposition

7. Conclusion

8. Next Steps

- multiple copies of the same network on the same machine → use all resources available
- better domain decomposition (SpiNNaker application graph or custom solution (application graph not helpful for neurons which become too big))
- smart algorithms vs. integrating with state-of-the-art libraries (investing time in stuff like SLIDE and the one paper by the Austrian guys about sparse connections explicitly mentioning SpiNNaker and neuromorphic chips or rather work on a trans-/compiler that efficiently translates linear algebra operations (like TF, PyTorch,…) onto SpiNNaker)
- integrate into compiler projects like Apache-TVM, XLA, Glow, nGraph, etc.
- implementing ONNX spec to make it easy for developers to use SpiNNaker (develop in PyTorch → run on SpiNNaker)

References

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. AI and Compute. `https://openai.com/blog/ai-and-compute/`, 2019.

Y. Bengio, Pascal Lamblin, D. Popovici, Hugo Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. volume 19, 01 2007.

Justin Boitano. How NVIDIA EGX Is Forming Central Nervous System of Global Industries, 05 2020. URL `https://blogs.nvidia.com/blog/2020/05/15/egx-security-resiliency/`.

Jason Brownlee. Crash Course in Convolutional Neural Networks for Machine Learning, 08 2019. URL `https://machinelearningmastery.com/crash-course-convolutional-neural-networks/`.

Andrew Cassidy, Paul Merolla, John Arthur, S.K. Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta, Jun Sawada, Theodore Wong, Vitaly Feldman, Arnon Amir, Daniel Ben Dayan Rubin, Filipp Akopyan, Emmett McQuinn, W.P. Risk, and Dharmendra Modha. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. 08 2013. doi: 10.1109/IJCNN.2013.6707077.

François Chollet et al. Keras. `https://keras.io`, 2015.

T Ciodaro, D Deva, Joao Seixas, and Denis Oliveira Damazio. Online particle detection with neural networks based on topological calorimetry information. *Journal of Physics: Conference Series*, 368, 06 2012. doi: 10.1088/1742-6596/368/1/012030.

Yunbin Deng. Deep learning on mobile devices - A review. *CoRR*, abs/1904.09274, 2019. URL http://arxiv.org/abs/1904.09274.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 04 1980. doi: 10.1007/bf00344251. URL https://doi.org/10.1007%2Fbf00344251.

Steve Furber. Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, 13, 08 2016. doi: 10.1088/1741-2560/13/5/051001.

Steve Furber, Steve Temple, and Andrew Brown. High-performance computing for systems of spiking neurons. 2, 01 2006.

Lee Gomes. Neuromorphic Chips Are Destined for Deep Learning—or Obscurity, 05 2017. URL https://spectrum.ieee.org/semiconductors/design/neuromorphic-chips-are-destined-for-deep-learningor-obscurity.

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

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, New York, NY, second edition edition, 2009. ISBN 9780387848570.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18:1527–54, 08 2006. doi: 10.1162/neco.2006.18.7.1527.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL http://www.sciencedirect.com/science/article/pii/0893608089900208.

D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962. doi: 10.1113/jphysiol.1962.sp006837. URL https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1962.sp006837.

D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968. doi: 10.1113/jphysiol.1968.sp008455. URL https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008455.

David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148:574–91, 1959.

Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017. URL `http://arxiv.org/abs/1704.04760`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Jennifer Langston. Microsoft announces new supercomputer, lays out vision for future AI work, 05 2020. URL `https://blogs.microsoft.com/ai/openai-azure-supercomputer/`.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

Michael Leung, Hui Xiong, Leo Lee, and Brendan Frey. Deep learning of the tissue-regulated splicing code. *Bioinformatics (Oxford, England)*, 30:i121–i129, 06 2014. doi: 10.1093/bioinformatics/btu277.

John Loeffler. No More Transistors: The End of Moore's Law, 11 2018. URL `https://interestingengineering.com/no-more-transistors-the-end-of-moores-law`.

Junshui Ma, Robert Sheridan, Andy Liaw, George Dahl, and Vladimir Svetnik. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55, 01 2015. doi: 10.1021/ci500747n.

Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(97)00011-7. URL `http://www.sciencedirect.com/science/article/pii/S0893608097000117`.

Adam H. Marblestone, Greg Wayne, and Konrad P. Kording. Towards an integration of deep learning and neuroscience. *bioRxiv*, 2016. doi: 10.1101/058545. URL `https://www.biorxiv.org/content/early/2016/06/13/058545`.

Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen,

Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. Mlperf training benchmark, 2019.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, pages 115–133, 12 1943. doi: 10.1007/BF02478259. URL `http://link.springer.com/10.1007/BF02478259`.

Carver Mead. Analog vlsi and neural systems. 1989.

Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

Andres Felipe Rodriguez Perez. Intel Processors for Deep Learning Training, 11 2017. URL `https://software.intel.com/content/www/us/en/develop/articles/intel-processors-for-deep-learning-training.html`.

Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. 01 2006.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986a. ISBN 026268053X.

David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, Cambridge, MA, USA, 1986b. ISBN 026268053X.

David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2: Psychological and Biological Models*. MIT Press, Cambridge, MA, USA, 1986c. ISBN 0262132184.

David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017. doi: 10.1038/nature24270.

Tom Simonite. Moore's Law Is Dead. Now What?, 05 2016. URL `https://www.technologyreview.com/2016/05/13/245938/moores-law-is-dead-now-what/`.