arXiv:1801.05894v1 [math.HO] 17 Jan 2018

Deep Learning: An Introduction for Applied Mathematicians

Catherine F. Higham* Desmond J. Higham†

January 19, 2018

Abstract

Multilayered artificial neural networks are becoming a pervasive tool in a host of application fields. At the heart of this deep learning revolution are familiar concepts from applied and computational mathematics; notably, in calculus, approximation theory, optimization and linear algebra. This article provides a very brief introduction to the basic ideas that underlie deep learning from an applied mathematics perspective. Our target audience includes postgraduate and final year undergraduate students in mathematics who are keen to learn about the area. The article may also be useful for instructors in mathematics who wish to enliven their classes with references to the application of deep learning techniques. We focus on three fundamental questions: what is a deep neural network? how is a network trained? what is the stochastic gradient method? We illustrate the ideas with a short MATLAB code that sets up and trains a network. We also show the use of state-of-the art software on a large scale image classification problem. We finish with references to the current literature.

1 Motivation

Most of us have come across the phrase deep learning. It relates to a set of tools that have become extremely popular in a vast range of application fields, from image recognition, speech recognition and natural language processing to targeted advertising and drug discovery. The field has grown to the extent where sophisticated software packages are available in the public domain, many produced by high-profile technology companies. Chip manufacturers are also customizing their graphics processing units (GPUs) for kernels at the heart of deep learning.

*School of Computing Science, University of Glasgow, UK (Catherine.Higham@glasgow.ac.uk). This author was supported by the EPSRC UK Quantum Technology Programme under grant EP/M01326X/1.

†Department of Mathematics and Statistics, University of Strathclyde, UK (d.j.higham@strath.ac.uk). This author was supported by the EPSRC/RCUK Digital Economy Programme under grant EP/M00158X/1.

Whether or not its current level of attention is fully justified, deep learning is clearly a topic of interest to employers, and therefore to our students. Although there are many useful resources available, we feel that there is a niche for a brief treatment aimed at mathematical scientists. For a mathematics student, gaining some familiarity with deep learning can enhance employment prospects. For mathematics educators, slipping "Applications to Deep Learning" into the syllabus of a class on calculus, approximation theory, optimization, linear algebra, or scientific computing is a great way to attract students and maintain their interest in core topics. The area is also ripe for independent project study.

There is no novel material in this article, and many topics are glossed over or omitted. Our aim is to present some key ideas as clearly as possible while avoiding non-essential detail. The treatment is aimed at readers with a background in mathematics who have completed a course in linear algebra and are familiar with partial differentiation. Some experience of scientific computing is also desirable.

To keep the material concrete, we list and walk through a short MATLAB code that illustrates the main algorithmic steps in setting up, training and applying an artificial neural network. We also demonstrate the high-level use of state-of-the-art software on a larger scale problem.

Section 2 introduces some key ideas by creating and training an artificial neural network using a simple example. Section 3 sets up some useful notation and defines a general network. Training a network, which involves the solution of an optimization problem, is the main computational challenge in this field. In Section 4 we describe the stochastic gradient method, a variation of a traditional optimization technique that is designed to cope with very large scale sets of training data. Section 5 explains how the partial derivatives needed for the stochastic gradient method can be computed efficiently using back propagation. First-principles MATLAB code that illustrates these ideas is provided in section 6. A larger scale problem is treated in section 7. Here we make use of existing software. Rather than repeatedly acknowledge work throughout the text, we have chosen to place the bulk of our citations in Section 8, where pointers to the large and growing literature are given. In that section we also raise issues that were not mentioned elsewhere, and highlight some current hot topics.

2 Example of an Artificial Neural Network

This article takes a data fitting view of artificial neural networks. To be concrete, consider the set of points shown in Figure 1. This shows *labeled data*—some points are in category A, indicated by circles, and the rest are in category B, indicated by crosses. For example, the data may show oil drilling sites on a map, where category A denotes a successful outcome. Can we use this data to categorize a newly proposed drilling site? Our job is to construct a transformation that takes any point in $\mathbb{R}^2$ and returns either a circle or a square. Of course, there are many reasonable ways to construct such a transformation.

6 Full MATLAB Example

We now give a concrete illustration involving back propagation and the stochastic gradient method. Listing 6.1 shows how a network of the form shown in Figure 3 may be used on the data in Figure 1. We note that this MATLAB code has been written for clarity and brevity, rather than efficiency or elegance. In particular, we have "hardwired" the number of layers and iterated through the forward and backward passes line by line. (Because the weights and biases do not have the the same dimension in each layer, it is not convenient to store them in a three-dimensional array. We could use a cell array or structure array, [18], and then implement the forward and backward passes in `for` loops. However, this approach produced a less readable code, and violated our self-imposed one page limit.)

The function `netbp` in Listing 6.1 contains the nested function `cost`, which evaluates a scaled version of Cost in (6). Because this function is nested, it has access to the variables in the main function, notably the training data. We point out that the nested function `cost` is not used directly in the forward and backward passes. It is called at each iteration of the stochastic gradient method so that we can monitor the progress of the training.

Listing 6.2 shows the function `activate`, used by `netbp`, which applies the sigmoid function in vectorized form.

At the start of `netbp` we set up the training data and target y values, as defined in (5). We then initialize all weights and biases using the normal pseudorandom number generator `randn`. For simplicity, we set a constant learning rate `eta = 0.05` and perform a fixed number of iterations `Niter = 1e6`.

We use the the basic stochastic gradient iteration summarized at the end of Section 5. Here, the command `randi(10)` returns a uniformly and independently chosen integer between 1 and 10.

Having stored the value of the cost function at each iteration, we use the `semilogy` command to visualize the progress of the iteration.

In this experiment, our initial guess for the weights and biases produced a cost function value of 5.3. After 10^6 stochastic gradient steps this was reduced to 7.3×10^{-4}. Figure 7 shows the `semilogy` plot, and we see that the decay is not consistent—the cost undergoes a flat period towards the start of the process. After this plateau, we found that the cost decayed at a very slow linear rate—the ratio between successive values was typically within around 10^{-6} of unity.

An extended version of `netbp` can be found in the supplementary material. This version has the extra graphics commands that make Figure 7 more readable. It also takes the trained network and produces Figure 8. This plot shows how the trained network carves up the input space. Eagle-eyed readers will spot that the solution in Figure 8. differs slightly from the version in Figure 4, where the same optimization problem was tackled by the nonlinear least-squares solver `lsqnonlin`. In Figure 9 we show the corresponding result when an extra data point is added; this can be compared with Figure 5.

and past gradient directions. We use mini-batches of size 100 (so $m = 100$ in (17)) and set a fixed number of 45 epochs. We predefine the learning rate for each epoch: $\eta = 0.05$, $\eta = 0.005$ and $\eta = 0.0005$ for the first 30 epochs, next 10 epochs and final 5 epochs, respectively. Running on a Tesla C2075 GPU in single precision, the 45 epochs can be completed in just under 4 hours.

As an additional test, we also train the network with dropout. Here, on each stochastic gradient step, any neuron has its output re-set to zero with independent probability

- 0.15 in block 1,
- 0.15 in block 2,
- 0.15 in block 3,
- 0.35 in block 4,
- 0 in block 5 (no dropout).

We emphasize that in this case all neurons become active when the trained network is applied to the test data.

In Figure 11 we illustrate the training process in the case of no dropout. For the plot on the left, circles are used to show how the objective function (32) decreases after each of the 45 epochs. We also use crosses to indicate the objective function value on the validation data. (More precisely, these error measures are averaged over the individual batches that form the epoch—note that weights are updated after each batch.) Given that our overall aim is to assign images to one of the ten classes, the middle plot in Figure 11 looks at the percentage of errors that take place when we classify with the highest probability choice. Similarly, the plot on the right shows the percentage of cases where the correct category is not among the top five. We see from Figure 11 that the validation error starts to plateau at a stage where the stochastic gradient method continues to make significant reductions on the training error. This gives an indication that we are overfitting—learning fine details about the training data that will not help the network to generalize to unseen data.

Figure 12 shows the analogous results in the case where dropout is used. We see that the training errors are significantly larger than those in Figure 11 and the validation errors are of a similar magnitude. However, two key features in the dropout case are that (a) the validation error is below the training error, and (b) the validation error continues to decrease in sync with the training error, both of which indicate that the optimization procedure is extracting useful information over all epochs.

Figure 13 gives a summary of the performance of the trained network with no dropout (after 45 epochs) in the form of a *confusion matrix*. Here, the integer value in the general i, j entry shows the number of occasions where the network predicted category i for an image from category j. Hence, off-diagonal elements indicate mis-classifications. For example, the (1,1) element equal to 814 in Figure 13 records the number of airplane images that were correctly

8 Of Things Not Treated

This short introductory article is aimed at those who are new to deep learning. In the interests of brevity and accessibility we have ruthlessly omitted many topics. For those wishing to learn more, a good starting point is the free online book [26], which provides a hands-on tutorial style description of deep learning techniques. The survey [22] gives an intuitive and accessible overview of many of the key ideas behind deep learning, and highlights recent success stories. A more detailed overview of the prize-winning performances of deep learning tools can be found in [29], which also traces the development of ideas across more than 800 references. The review [35] discusses the pre-history of deep learning and explains how key ideas evolved. For a comprehensive treatment of the state-of-the-art, we recommend the book [10] which, in particular, does an excellent job of introducing fundamental ideas from computer science/discrete mathematics, applied/computational mathematics and probability/statistics/inference before pulling them all together in the deep learning setting. The recent review article [3] focuses on optimization tasks arising in machine learning. It summarizes the current theory underlying the stochastic gradient method, along with many alternative techniques. Those authors also emphasize that optimization tools must be interpreted and judged carefully when operating within this inherently statistical framework. Leaving aside the training issue, a mathematical framework for understanding the cascade of linear and nonlinear transformations used by deep networks is given in [24].

To give a feel for some of the key issues that can be followed up, we finish with a list of questions that may have occured to interested readers, along with brief answers and further citations.

Why use artificial neural networks? Looking at Figure 4, it is clear that there are many ways to come up with a mapping that divides the x-y axis into two regions; a shaded region containing the circles and an unshaded region containing the crosses. Artificial neural networks provide one useful approach. In real applications, success corresponds to a small *generalization error*; the mapping should perform well when presented with new data. In order to make rigorous, general, statements about performance, we need to make some assumptions about the type of data. For example, we could analyze the situation where the data consists of samples drawn independently from a certain probability distribution. If an algorithm is trained on such data, how will it perform when presented with *new data from the same distribution*? The authors in [15] prove that artificial neural networks trained with the stochastic gradient method can behave well in this sense. Of course, in practice we cannot rely on the existence of such a distribution. Indeed, experiments in [36] indicate that the worst case can be as bad as possible. These authors tested state-of-the-art convolutional networks for image classification. In terms of the heuristic performance indicators used to monitor the progress of the training phase, they found that the stochastic gradient method appears to work just as effectively

weights and biases, and hence the tasks performed by each layer, emerge from the training procedure. We note that the use of back propagation to compute gradients is not restricted to the types of connectivity, activation functions and cost functions discussed here. Indeed, the method fits into a very general framework of techniques known as *automatic differentiation* or *algorithmic differentiation* [13].

How big do deep learning networks get? The AlexNet architecture [21] achieved groundbreaking image classification results in 2012. This network used 650,000 neurons, with five convolutional layers followed by two fully connected layers and a final softmax. The programme *AlphaGo*, developed by the Google DeepMind team to play the board game Go, rose to fame by beating the human European champion by five games to nil in October 2015 [30]. AlphaGo makes use of two artificial neural networks with 13 layers and 15 layers, some convolutional and others fully connected, involving millions of weights.

Didn't my numerical analysis teacher tell me never to use steepest descent? It is known that the steepest descent method can perform poorly on examples where other methods, notably those using information about the second derivative of the objective function, are much more efficient. Hence, optimization textbooks typically downplay steepest descent [9, 27]. However, it is important to note that training an artificial neural network is a very specific optimization task:

- the problem dimension and the expense of computing the objective function and its derivatives, can be extremely high,
- the optimization task is set within a framework that is inherently statistical in nature,
- a great deal of research effort has been devoted to the development of practical improvements to the basic stochastic gradient method in the deep learning context.

Currently, a theoretical underpinning for the success of the stochastic gradient method in training networks is far from complete [3]. A promising line of research is to connect the stochastic gradient method with discretizations of stochastic differential equations, [31], generalizing the idea that many deterministic optimization methods can be viewed as timestepping methods for gradient ODEs, [17]. We also note that the introduction of more traditional tools from the field of optimization may lead to improved training algorithms.

Is it possible to regularize? As we discussed in section 7, overfitting occurs when a trained network performs accurately on the given data, but cannot *generalize* well to new data. *Regularization* is a broad term that describes attempts to avoid overfitting by rewarding smoothness. One approach is

to alter the cost function in order to encourage small weights. For example, (9) could be extended to

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N} \tfrac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2 + \frac{\lambda}{N}\sum_{l=2}^{L} \| W^{[l]} \|_2^2. \qquad (33)$$

Here $\lambda > 0$ is the regularization parameter. One motivation for (33) is that large weights may lead to neurons that are sensitive to their inputs, and hence less reliable when new data is presented. This argument does not apply to the biases, which typically are not included in such a regularization term. It is straightforward to check that using (33) instead of (9) makes a very minor and inexpensive change to the back propagation algorithm.

What about ethics and accountability? The use of "algorithms" to aid decision-making is not a recent phenomenon. However, the increasing influence of black-box technologies is naturally causing concerns in many quarters. The recent articles [7, 14] raise several relevant issues and illustrate them with concrete examples. They also highlight the particular challenges arising from massively-parameterized artificial neural networks. Professional and governmental institutions are, of course, alert to these matters. In 2017, the Association for Computing Machinery's US Public Policy Council released seven *Principles for Algorithmic Transparency and Accountability*[1]. Among their recommendations are that

- "Systems and institutions that use algorithmic decision-making are encouraged to produce explanations regarding both the procedures followed by the algorithm and the specific decisions that are made", and
- "A description of the way in which the training data was collected should be maintained by the builders of the algorithms, accompanied by an exploration of the potential biases induced by the human or algorithmic data-gathering process."

Article 15 of the European Union's General Data Protection Regulation 2016/679[2], which takes effect in May 2018, concerns "Right of access by the data subject," and includes the requirement that "The data subject shall have the right to obtain from the controller confirmation as to whether or not personal data concerning him or her are being processed, and, where that is the case, access to the personal data and the following information:."Item (h) on the subsequent list covers

- "the existence of automated decision-making, including profiling, referred to in Article 22(1) and (4) and, at least in those cases, meaningful information about the logic involved, as well as the significance

[1] `https://www.acm.org/`

[2] `https://www.privacy-regulation.eu/en/15.htm`

and the envisaged consequences of such processing for the data subject."

What are some current research topics? Deep learning is a fast-moving, high-bandwith field, where many new advances are driven by the needs of specific application areas and the features of new high performance computing architectures. Here, we briefly mention three hot-topic areas that have not yet been discussed.

Training a network can be an extremely expensive task. When a trained network is seen to make a mistake on new data, it is therefore tempting to fix this with a local perturbation to the weights and/or network structure, rather than re-training from scratch. Approaches for this type of *on the fly* tuning can be developed and justified using the theory of measure concentration in high dimensional spaces [12].

Adversarial networks, [11], are based on the concept that an artificial neural network may be viewed as a *generative model*: a way to create realistic data. Such a model may be useful, for example, as a means to produce realistic sentences, or very high resolution images. In the adversarial setting, the generative model is pitted against a *discriminative model*. The role of the discriminative model is to distinguish between real training data and data produced by the generative model. By iteratively improving the performance of these models, the quality of both the generation and discrimination can be increased dramatically.

The idea behind *autoencoders* [28] is, perhaps surprisingly, to produce an overall network whose output matches its input. More precisely, one network, known as the *encoder*, corresponds to a map F that takes an input vector, $x \in \mathbb{R}^s$, and produces a lower dimensional output vector $F(x) \in \mathbb{R}^t$. So $t \ll s$. Then a second network, known as the *decoder*, corresponds to a map G that takes us back to the same dimension as x; that is, $G(F(x)) \in \mathbb{R}^s$. We could then aim to minimize the sum of the squared error $\|x - G(F(x))\|_2^2$ over a set of training data. Note that this technique does not require the use of labelled data—in the case of images we are attempting to reproduce each picture without knowing what it depicts. Intuitively, a good encoder is a tool for dimension reduction. It extracts the key features. Similarly, a good decoder can reconstruct the data from those key features.

Where can I find code and data? There are many publicly available codes that provide access to deep learning algorithms. In addition to MatConvNet [33], we mention Caffe [19], Keras [5], TensorFlow [1], Theano [2] and Torch [6]. These packages differ in their underlying platforms and in the extent of expert knowledge required. Your favorite scientific computing environment may also offer a range of proprietary and user-contributed deep learning toolboxes. However, it is currently the case that making serious use of modern deep learning technology requires a strong background in numerical computing. Among the standard benchmark data sets are

the CIFAR-10 collection [20] that we used in section 7, and its big sibling CIFAR-100, ImageNet [8], and the handwritten digit database MNIST [23].

Acknowledgements

We are grateful to the MatConvNet team for making their package available under a permissive BSD license. The MATLAB code in Listings 6.1 and 6.2 can be found at

`http://personal.strath.ac.uk/d.j.higham/algfiles.html`

as well as an exteneded version that produces Figures 7 and 8, and a MATLAB code that uses `lsqnonlin` to produce Figure 4.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: A system for large-scale machine learning*, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[2] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van

Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, *Theano: A Python framework for fast computation of mathematical expressions*, arXiv e-prints, abs/1605.02688 (2016).

[3] L. Bottou, F. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, arXiv:1606.04838, version 2, (2017).

[4] T. B. Brown, D. Mané, A. R. M. Abadi, and J. Gilmer, *Adversarial patch*, arXiv:1712.09665 [cs.CV], (2017).

[5] F. Chollet et al., *Keras*, GitHub, (2015).

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet, *Torch7: A Matlab-like environment for machine learning*, in BigLearn, NIPS Workshop, 2011.

[7] J. H. Davenport, *The debate about algorithms*, Mathematics Today, (2017), p. 162.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, *ImageNet: A large-scale hierarchical image database.*, in CVPR, IEEE Computer Society, 2009, pp. 248–255.

[9] R. Fletcher, *Practical Methods of Optimization*, Wiley, Chichester, second ed., 1987.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Boston, 2016.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, *Generative adversarial nets*, in Advances in Neural Information Processing Systems 27, Montreal, Canada, 2014, pp. 2672–2680.

[12] A. N. Gorban and I. Y. Tyukin, *Stochastic separation theorems*, Neural Networks, 94 (2017), pp. 255–259.

[13] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics, Philadelphia, second ed., 2008.

[14] P. Grindrod, *Beyond privacy and exposure: ethical issues within citizen-facing analytics*, Phil. Trans. of the Royal Society A, 374 (2016), p. 2083.

[15] M. Hardt, B. Recht, and Y. Singer, *Train faster, generalize better: Stability of stochastic gradient descent*, in Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 1225–1234.

[16] C. F. Higham, R. Murray-Smith, M. J. Padgett, and M. P. Edgar, *Deep learning for real-time single-pixel video*, Scientific Reports, (to appear).

[17] D. J. Higham, *Trust region algorithms and timestep selection*, SIAM Journal on Numerical Analysis, 37 (1999), pp. 194–210.

[18] D. J. Higham and N. J. Higham, *MATLAB Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, third ed., 2017.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, *Caffe: Convolutional architecture for fast feature embedding*, arXiv preprint arXiv:1408.5093, (2014).

[20] A. Krizhevsky, *Learning multiple layers of features from tiny images*, tech. rep., 2009.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., 2012, pp. 1097–1105.

[22] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[24] S. Mallat, *Understanding deep convolutional networks*, Philosophical Transactions of the Royal Society of London A, 374 (2016), p. 20150203.

[25] G. Marcus, *Deep learning: A critical appraisal*, arXiv:1801.00631 [cs.AI], (2018).

[26] M. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

[27] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, Berlin, second ed., 2006.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1*, MIT Press, Cambridge, MA, USA, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318–362.

[29] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 61 (2015), pp. 85–117.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of Go with deep neural networks and tree search*, Nature, 2529 (2016), pp. 484–489.