LU-TP 22-30
June 2022

Investigating Relations between Regularization and Weight Initialization in Artificial Neural Networks

Rasmus Sjöö

Department of Astronomy and Theoretical Physics, Lund University

Bachelor thesis supervised by Patrik Edén

LUND
UNIVERSITY

Abstract

L2 regularization is a common method used to prevent overtraining in artificial neural networks. However, an issue with this method is that the regularization strength has to be properly adjusted for it to work as intended. This value is usually found by trial and error which can take some time, especially for larger networks. This process could be alleviated if there was a mathematical relationship that predicted the best strength based on the network's hyperparameters. The aim of this project is to prove part of such a relation, specifically if the optimal regularization strength is proportional to the inverse number of training patterns. This was tested using a network that solves binary classification problems. Several regularization strengths were tested for different amounts of training patterns. The best ones were compared to the proposed relation. Additional tests were performed to check if weight initialization had an effect on said relation, and if it works for L1 regularization as well.

Populärvetenskaplig beskrivning

Artificiella Neurala Nätverk (ANN) är en form av maskininlärning som är populär just nu. Den har många användningsområden inom dagens teknik, till exempel som bildigenkänning. Att programmera ett sådant nätverk är dock inte det lättaste man kan göra. Ett vanligt problem som uppstår när ett ANN byggs upp är överträning, vilket sker när nätverket är för kraftfullt för problemet det ska lösa. En vanlig metod som används för att kringgå detta problem är L2 regularisering. Den går ut på att modifiera nätverket så att det minskar sina vikter under träning. Detta kräver att man har en lämplig regulariseringsstyrka. Det vanligaste sättet att hitta den bästa styrkan är att testa flera olika värden och se vilket som ger bäst resultat. Denna process kan dock ta mycket tid, särskilt om man har ett stort nätverk som tar lång tid att träna. Det hade varit lättare att utföra detta arbete om det fanns ett matematiskt förhållande som avgjorde den bästa regulariseringsstyrkan innan träningen. Målet med detta projekt var att ta reda på huruvida ett sådant förhållande existerade och hur det skulle set ut.

Contents

List of acronyms

ANN Artificial Neural Network
RNG Random Number Generator MLP Multilayered Perceptron CEE Cross Entropy Error

1 Introduction

The issue of overtraining is common within the field of artificial neural networks [1]. There are however, several ways to get around this problem. One of them is called L2 regularization which works by adding the sum of all weights squared to the loss function [2]. This will reduce the weight size in the network, making it more stable and reduce overtraining. However, one has to adjust the regularization strength (λ) in order to achieve optimal performance. Currently, the best way to find good hyperparameters, including λ, is through random trials [3]. This process can be very time consuming though, especially if the network is large and takes a long time to train.

This project is part of a collaboration between four bachelor students in theoretical physics (because of this, I will use "we" to refer to this group in the sections that were performed collectively). The shared goal of our projects is to alleviate the training process by finding a mathematical relationship for optimal λ. We present a simple theoretical argument that suggests the optimal λ depends on certain hyperparameters in the network. This theory was explored and put to the test during the project. The artificial neural network that was used for the project was coded by the group as part of the collaboration.

The task of this particular project was to find out if optimal lambdas are inversely dependent on the number of training patterns used in the network. Some additional tests were performed to see if weight initialization had an effect on the lambdas and if L1 regularization has similar dependencies as L2.

2 Theory

2.1 Loss Function

To determine if an ANN has given an output y that is close to the desired target d, a loss function is used to compare them. The weights in the network are then updated proportionally to the size of the loss. This function is also used in training where the network updates its weights in a manner that minimizes the loss function.
There are several loss functions that are good for different situations. In the case of a binary classification problem, one suitable option is the so called "cross entropy error" (CEE) [1]. This function is defined as:

$$E(\boldsymbol{\omega}) = -\frac{1}{N}\sum_{n}[d_n \ln(y_n) + (1-d_n)\ln(1-y_n)] \tag{2.1}$$

Where $\boldsymbol{\omega}$ is the set of weights in the ANN and N is the number of training patterns. This loss function has the added benefit of allowing y_n to be interpreted as the probability that $d_n = 1$. This is because d_n can only be 1 or 0 in the case of binary problems. This also

means that the loss function is simplified for the two possible target values:

$$d = 0 \Rightarrow E(\boldsymbol{\omega}) = -\frac{1}{N}\sum_n [\ln(1 - y_n)]$$

$$d = 1 \Rightarrow E(\boldsymbol{\omega}) = -\frac{1}{N}\sum_n [\ln(y_n)]$$

2.2 Regularization terms

One way of reducing overtraining is to modify the loss function $E(\boldsymbol{\omega})$ by adding a regularization term onto it,

$$\tilde{E}(\boldsymbol{\omega}) = E(\boldsymbol{\omega}) + \lambda\Omega(\boldsymbol{\omega}) \tag{2.2}$$

where λ is the regularization strength, $\boldsymbol{\omega}$ is the weights and Ω is the regularization term [1]. This extra term will alter which weight values gives the smallest loss. The regularization strength determines how much of an effect the term will have on the overall loss. There are different ways to define the regularization terms. The ones used in this project are called L2 regularization and L1 regularization. They both work by letting the regularization term grow with the magnitude of the weights in the network, but use different relations each. The loss will then generally be lower for smaller weight values.

2.2.1 L2 Regularization

L2 regularization, sometimes known as weight decay, works by defining the regularization term as

$$\Omega = \frac{1}{2}\sum_i \omega_i^2 \tag{2.3}$$

where ω_i is weight i in the network [4]. The derivative of this term with respect to weight i becomes ω_i. This means that when training the network, the minimized error is often located where the weights are small.

2.2.2 L1 Regularization

L1 regularization, or lasso, has a simpler regularization term [2].

$$\Omega = \sum_i |\omega_i|. \tag{2.4}$$

The derivative of this term with respect to ω_i is -1 for $\omega_i < 0$ and 1 for $\omega_i > 0$. Due to how Ω is defined for L1 means that the derivative is independent of ω_i and therefore does not vanish for very small weights.

2.3 Maximum Likelihood

In order to understand the mathematical relation that we hypothesize to hold for optimal λ, some probability theory needs to be involved. Applying this to the training data set will give the loss function on Bayesian form, which will be needed to explain the relation.

The main goal of training a network is finding a certain set of weights, $\boldsymbol{\omega}_n$ that give a desired target output d_n, given a certain set of inputs, $\boldsymbol{x}_n$ [1].

A data set that is used to train an ANN, $D = \{\boldsymbol{x}_n, d_n\}_{n=1...N}$ can be considered as a sample from an unknown distribution, $p(D)$ [1]. This can be rewritten, assuming the samples are independent, as:

$$p(D) = \prod_n p(d_n, \boldsymbol{x}_n) = \prod_n p(d_n|\boldsymbol{x}_n)p(\boldsymbol{x}_n). \tag{2.5}$$

Here, $p(d_n, \boldsymbol{x}_n)$ is the total probability for both d_n and $\boldsymbol{x}_n$, $p(d|\boldsymbol{x}_n)$ is the probability of getting target d_n *given* an input $\boldsymbol{x}_n$, and $p(\boldsymbol{x}_n)$ is the probability of said input.

The end product of a neural network is an output, $y(\boldsymbol{x}, \boldsymbol{\omega})$, which is dependent on a set of inputs, $\boldsymbol{x}$, and weights, $\boldsymbol{\omega}$. This can be interpreted as a model for the expectation value for d given inputs $\boldsymbol{x}$ [1]. Thus, the weights are also part of a model of $p(D)$. To accommodate this, the distribution is rewritten as $p(D|\boldsymbol{\omega})$ (probability for a certain data set D given a set of weights $\boldsymbol{\omega}$). This affects the probabilities in equation 2.5 as well:

$$p(D|\boldsymbol{\omega}) = \prod_n p(d_n, \boldsymbol{x}_n|\boldsymbol{\omega}) = \prod_n p(d_n|\boldsymbol{x}_n, \boldsymbol{\omega})p(\boldsymbol{x}_n). \tag{2.6}$$

Note that $p(\boldsymbol{x}_n)$ is independent of the weights since the weights only affect the relation between inputs and targets, not the inputs themselves.

The last step is to circle this back to the loss function. This is done with the "Maximum Likelihood" method which is often used in the field of statistics. It works by maximizing the likelihood $p(D|\boldsymbol{\omega})$ [1]. Here, a variation of this method is used where, instead of maximizing $p(D|\boldsymbol{\omega})$, one minimizes the negative logarithm of it,

$$-\ln p(D|\boldsymbol{\omega}) = -\sum_{n=1}^{N} \ln p(d_n|\boldsymbol{x}_n, \boldsymbol{\omega}) - \sum_{n=1}^{N} \ln p(\boldsymbol{x}_n). \tag{2.7}$$

This is represented by minimizing a loss which is proportional to the right hand side of equation 2.7. However, the second term is independent of $\boldsymbol{\omega}$, so it gets thrown away, giving the following loss function:

$$E(\boldsymbol{\omega}) \propto -\sum_{n=1}^{N} \ln p(d_n|\boldsymbol{x}_n, \boldsymbol{\omega}). \tag{2.8}$$

2.4 Bayesian Formalism

A Bayesian framework can be applied to the maximum likelihood approach to further build on to the loss function.

One cannot know what the weights in an ANN will look like before training is performed. There are however always some restrictions applied before training. For instance, the size of the network restricts the size of the weight vector, $\boldsymbol{\omega}$. All such restrictions are represented by a "prior distribution", $p(\boldsymbol{\omega})$ within the Bayesian framework [1].

The Bayesian prior is part of Bayes' theorem:

$$p(\boldsymbol{\omega}|D) = \frac{p(D|\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(D)}. \tag{2.9}$$

The maximum likelihood relationship from earlier, equation (2.6), can be inserted into this theorem if the negative logarithm is applied first:

$$-\ln p(\boldsymbol{\omega}|D) = -\sum_{n=1}^{N} \ln p(d_n|\boldsymbol{x}_n, \boldsymbol{\omega}) - \sum_{n=1}^{N} \ln p(\boldsymbol{x}_n) - \ln p(\boldsymbol{\omega}) + \ln p(D). \tag{2.10}$$

Throwing away all the $\boldsymbol{\omega}$-independent terms gives the loss function as

$$E(\boldsymbol{\omega}) \propto -\sum_{n=1}^{N} \ln p(d_n|\boldsymbol{x}_n, \boldsymbol{\omega}) - \ln p(\boldsymbol{\omega}) \tag{2.11}$$

2.5 Mathematical Relation

Equation 2.11 corresponds to the loss function with regularization term seen in equation 2.2. This means that the Bayesian prior is equivalent to the regularization term.

It is a common convention in the field of ANN to multiply the loss function by $1/N$. This is done so that networks with large data sets give errors on the same scale as smaller ones. Without this factor, the learning rate will vary widely depending on the network size. Applying this to the loss function gives the following relation:

$$E = -\frac{1}{N}\sum_{n=1}^{N} \ln p(d_n|\boldsymbol{x}_n, \boldsymbol{\omega}) - \frac{1}{N}\ln p(\boldsymbol{\omega}). \tag{2.12}$$

Comparing this relation to equation 2.2, the Bayesian prior term is equivalent to the regularization term. This then implies that a good lambda should be inversely dependent on the number of training patterns.

This $1/N$ dependence is not the only factor λ depends on. The other participants in this collaboration have explored if λ depends on other factors, such as the number of input nodes [5], [6] and if it works with dropout [7].

3 Method

3.1 Network Coding

In order to conduct the experiments outlined by the theory, an ANN needed to be created for use in testing. We decided to code our own network, in order to have greater control over its features and hyperparameters. This also allowed us to test different λ in several unconnected ways. The structure used to make the network was a Multilayered Perceptron (MLP) that solved binary classification problems. The amount of nodes and hidden layers were variable and could be changed depending on the problem. As the issue studied was overtraining, the network would often be defined as more powerful than needed to solve the problem. This caused overtraining to occur so that tests could be performed using L2 regularization.

The network tested which λ gave the best performance for different hyperparameters. The code supported testing of λ for different amount of nodes in each layer and increasing amounts of training patterns.

3.2 Synthetic Data Generation

In addition to the network, data was needed for training. Rather than using an already existing data set, we opted to use synthetic data for both training and validation. We coded a synthetic data generator using python. This allowed us to try several sets of the same type and make changes to it if need be. The data sets used for testing was a binary classification problem where the network had to sort between two different types of points in a high-dimensional space. The two point-types were scattered as spherically symmetrical Gaussian clouds around two respective central coordinates. The generator had options to change the dimensionality of the data, how many points of each type, and the size of the Gaussian clouds. We could also control how much data was created. This was mostly important for validation, as that needed to be very accurate.

One issue we ran into was finding a good data set for regularization tests. It needed to be easy enough for the network to solve it, but also hard enough that overtraining would occur in the network without regularization. The one I settled on was a 10-dimensional data set with with equally large classes. The two central coordinates were placed about 13.86 length units apart from each other. The Gaussian clouds had standard deviation of 8 length units in each dimension.

3.3 Initialization

When training the network, the weights were initialized using a Random Number Generator (RNG) according to a normal distribution. This distribution was defined to have zero

mean and variance $1/K$, where K is the number of input weights [1]. This distribution is commonly used when training ANNs. There are alternative ways to define the distribution [8] [9], but for the scope of this project, the mean and variance defined earlier sufficed.

One of the tests performed on the network was to see if weight initialization had any effect on the training. In order to test this, the variance $1/K$ was scaled by different constant factors to be much smaller or larger than usual. This intentionally bad initialization of the weights would in theory make it harder for the network to minimize the loss, as the weights would have had a worse starting distribution.

3.4 Testing

The first tests were done to see if the optimal regularization strength, λ^*, varied at all with different hyperparameters and if so, behaved according to the theorized $1/N$ relation. My own testing consisted of comparing L2 with L1 regularization, as well as trying different methods to initialize the weights when training the network. The purpose of these tests were to see if the relations had more dependencies than theorized.

All tests except for one used the same hyperparameters. These included: 500 epochs, learning rate of 0.1, one hidden layer with 15 nodes and one output node. CEE was used as the loss function in the network. The tested λ were on a logarithmic scale. They varied between 10^{-5} and 0.02. Specifically 1, 2, 4 and 8 multiplied by a power of 10.

To test the validity of the $1/N$ dependence, the network performed training for each test several times, using different seeds for the data generation each time. The best λ was determined by checking which one gave the lowest validation error. An example of this can be seen in figure 1, where $\lambda = 2 \times 10^{-3}$ gives the lowest validation error. Thus it is determined to be the best value in this case.

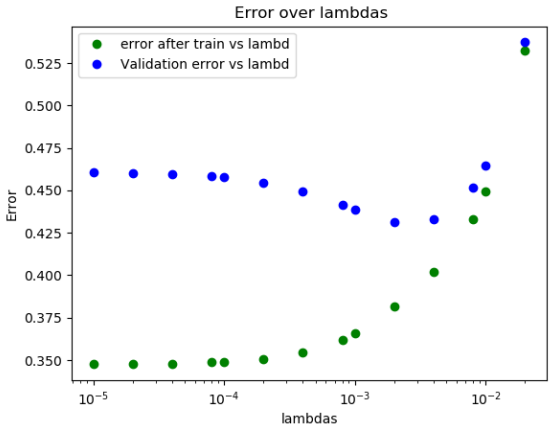


Figure 1: Training and Validation error over λ for $N = 3240$. The one with lowest validation error is chosen as λ^*.

The results from these runs were then used to make error bars using the standard error of the mean. Theses bars were then put in two plots with the mean of λ^* for each amount of training patterns. The plots had a linear scale and a logarithmic scale respectively. A linear regression was then performed in the logarithmic plot. To show a clear $1/N$ dependence, said regression would need a slope of around -1.

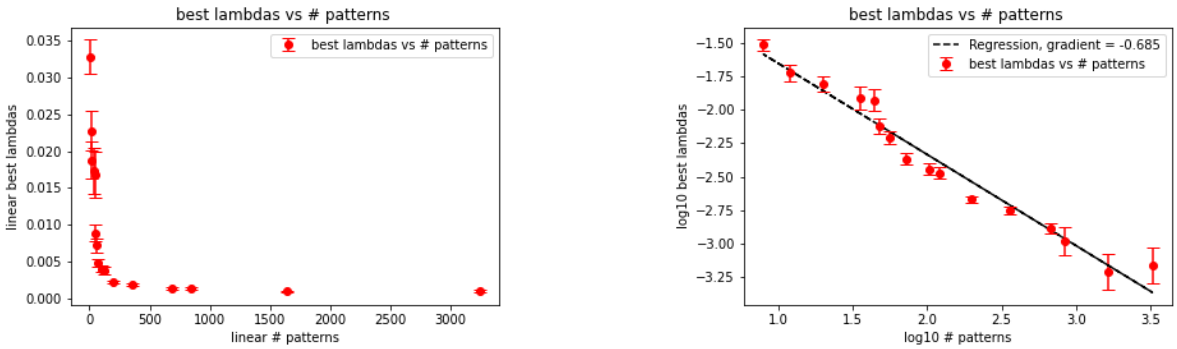
To test if weight initialization had an effect on the optimal lambda for different amounts of patterns, three different weight initializations were tried. The variance was scaled by factor 0.1, 1 and 10 respectively for each test. For each amount of patterns, 14 λ were tested.

The number of patterns used for training were increased on a logarithmic scale. For the initial tests, N was increased 14 times starting at 8 patterns, going up to 3240 patterns. For the weight initialization and L1 vs L2 tests, N was increased 10 times starting at 108 patterns and going up to 3300 patterns.

4 Results

4.1 Initial Testing

The results of our initial testing can be seen in figure 2.



(a) Linear plot of initial results

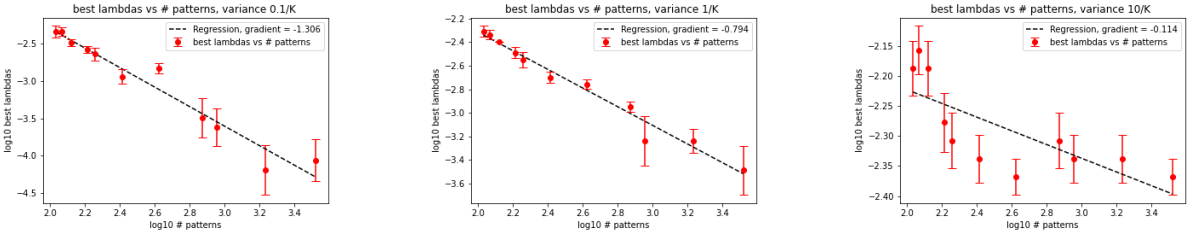
(b) Logarithmic plot of initial results

Figure 2: Results of how λ^* vary with increasing numbers of training patterns.

In both plots, λ^* becomes smaller for increasing amounts of extra patterns. However, the slope of -0.685 is a bit away from -1. This result meant that further testing was warranted since λ^* behaved roughly, but not precisely, according to the relationship.

4.2 Weight initialization

Here, λ^* can be seen in the logarithmic plots in figure 3.



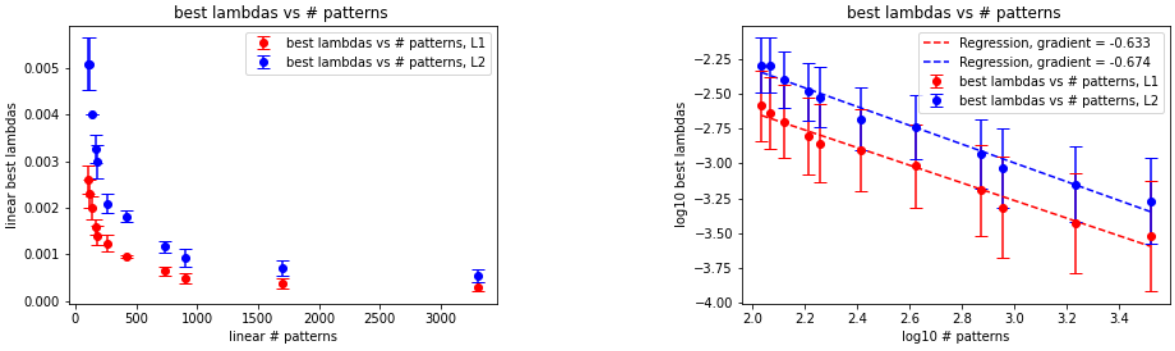
(a) Sigma factor 0.1 (b) Sigma factor 1 (c) Sigma factor 10

Figure 3: λ^* determined for different weight initializations.

The slopes of the linear regressions in the plots are different depending on how the weights were initialized. This would imply that the weight initialization process has an effect on λ^*. The slopes for factor 0.1 and 1 are both fairly close to -1, while the factor 10 one was an order of magnitude away.

4.3 L2 vs L1

Tests were performed to test how L1 and L2 regularization compared to each other when used to solve the same problem. A total of 14 λ-values were tested for each amount of patterns, which increased 10 times. The plots for initial testing can be seen in figure 4.



(a) Linear L1 vs L2 plot (b) Logarithmic L1 vs L2 plot

Figure 4: Comparison of best λ for L1 and L2 regularization, attempt 1.

These plots show that L1 and L2 have similar relations between λ^* and the number of training patterns, implying that the $1/N$ dependence applies to L1 in addition to L2 regularization. The main difference between the slopes is that they intercept the y-axis at different points.

5 Discussion

I have performed tests to see if the regularization strength of an ANN can be determined before training said network using a mathematical relationship. I have also tested whether or not it holds for different weight initializations. Finally, I have tested how λ^* behaves for L1 regularization compared to L2.

The initial testing results showed that λ^* decreased for increasing amounts of training patterns. As shown in figure 2, λ^* agrees to some extent with a $1/N$ curve, which is consistent with the theory. The main take away from this section is that there is a general trend in the results which implies the relationship might be qualitatively correct.

The second round of tests was done to see if the initialization had an effect on λ^*. Figure 3 conveys that scaling the initialization variance by different factors gives different results. The plots for factor 0.1 and 1 appear to be rather similar at a quick glance. However, the y-axes scale differently. One can also look at the gradient on the slopes (-1.31 and -0.79 respectively) which differ by about 0.5. While they were far apart from each other, both were fairly close to -1. This would imply that using a smaller variance for the initialization still gives some agreement with the theorized $1/N$ dependence. On the other hand, using the factor 10 gave a much different result. A slope of around -0.11 differs markedly from the other two results and also -1. This could mean that bad initialization affects what λ^* will be. However, it could also mean that different amounts of epochs are needed for training. Since if the weights are spread far from their optimal values, the network would need more epochs to adjust the weights appropriately. All in all, more testing would be needed to determine if poor weight initialization has an effect on the $1/N$ dependence for λ^*.

The final tests consisted of comparing L1 and L2 regularization, the results of which can can be seen in figure 4. They had close to equal slopes in the linear regressions in the logarithmic plots, and the linear plots showed similar curves between the two methods. The only difference is that there is a constant difference between their respective λ^*. These results suggest that both L1 and L2 regularization have mathematical relationships for their respective λ^*. They are not identical, but have roughly the same qualitative dependence $1/N$ which was theorized.

To conclude these results, there appears to be an approximate $1/N$ behavior for λ^*.

Acknowledgments

This project was made possible through contributions from the other members of this collaboration. Therefore, I would like to thank Joseph Binns, Alexanader Degener and Oskar Bolinder for coding the Artificial Neural Network together with me. I would also like to thank Mattias Ohlsson for helping us with the theory. Finally, I would like to thank

Patrik Edén, who supervised this project, provided much of the theoretical background and answered the many questions I had during the semester.

References

[1] Ohlsson, M. & Edén, P. (2021). *Introduction to Artificial Neural Networks and Deep Learning*. Media Tryck, Lund University.

[2] Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press. `https://www.deeplearningbook.org/`

[3] Bergstra, J. & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305

[4] Krogh, A. & Hertz, J. A. (1991). A simple weight decay can improve generalization. *Proceedings of the 4th International Conference on Neural Information Processing Systems*. pp. 950–957

[5] Bolinder, O. (2022). *Optimizing L2-regularization for Binary Classification Tasks*. LU-TP 22-27. Lund University

[6] Degener, A. (2022). *Prediction of appropriate L2 regularization strengths through Bayesian formalism*. LU-TP 22-29. Lund University

[7] Binns, J. (2022). *Combined Regularization Techniques for Artificial Neural Networks*. LU-TP 22-28. Lund University

[8] Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks.*Proceedings of the thirteenth international conference on artificial intelligence and statistics*. pp. 249–256.

[9] He, K., Zhang, X., Ren, S. & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the IEEE international conference on computer vision* pp. 1026–1034.