MLP Coursework 1: Activation Functions

S1655829

Abstract

In this work, we investigate the performance of the various type of rectified activation functions (non-linear) for the classification of images of handwritten digits on EMNIST balanced dataset. We will firstly introduce some background knowledge relates to the investigation topics and then introduce a few activation functions. The introduced activation functions are ReLU, Leaky ReLU, Random ReLU, Parametric ReLU, and ELU. They will be explicitly explained in the paper including their initial settings. Among these functions, a model with layers using ReLU function will be regarded as a baseline for our investigation, due to the factor that others are extensions or variances base on ReLU. At the end of the work, we also investigate whether the non-linear functions are necessary for fitting deep neural networks on large datasets such as EMNIST

1. Introduction

Lately, deep learning has become so famous because of its supremacy in terms of accuracy when trained with a huge amount of data. Deep learning is simply learning from examples using a neural network(1). Neural networks contain s set of algorithms, an analogy to the human brain, that is designed to recognize the patterns in a dataset. It contains multiple layers where each layer contains multiple nodes. A node is just where computation has taken place.(2) Then an activation function bypasses the result from one node to another as mentioned above. An activation function takes a input (x) from a node and applies the function to it (f(x)), then passes the output (f(x)) from the function to a next node.(3)

Deep learning has many usages in real life such as computer vision and pattern recognition, computer games, robots, self-driving cars...(4) In this paper, we will use deep learning for handwriting recognition task. Handwriting recognition has grown rapidly in the current globalization. It enables a computer system can recognize characters. This has many advantages such as data storage (files, contracts, personal records...), data retrieval, historical preservation, using the stylus on cell phone...(5) In our investigation, we will try to find out how each of the non-linear functions (ReLUs) performs for our task and find out whether non-linear functions are necessary for fitting deep neural networks on large datasets. Also, we will find which hyper parameters affect more on our models. The dataset we use in our investigation is EMNIST balanced dataset, which is an extended version of the MNIST (which contains only digits dataset). EMNIST extends MNIST by including images of handwritten letters (upper and lower case) as well as handwritten digits. It is extracted from NIST Special Dataset 19. It uses a conversion processing resulting in centred images of dimension 28X28. There are 62 potential classes for EMNIST but only 47 different labels will be used in the experiment due to the fact that 15 letters are confusing to discriminate between upper-case and lower-case versions (C, I, J, K, L, M, O, P, S, U, V, W, X, Y, Z).(6) The dataset has 100000 training images (100000X785, where the last column is images' labels), 15800 validation images, 15800 and testing images, each showing a 28x28 grey-scale pixel image of one of the 47 labels (10 digits plus 37 letters).

2. Baseline systems

We use ReLU as our activation function for the model of baseline system.

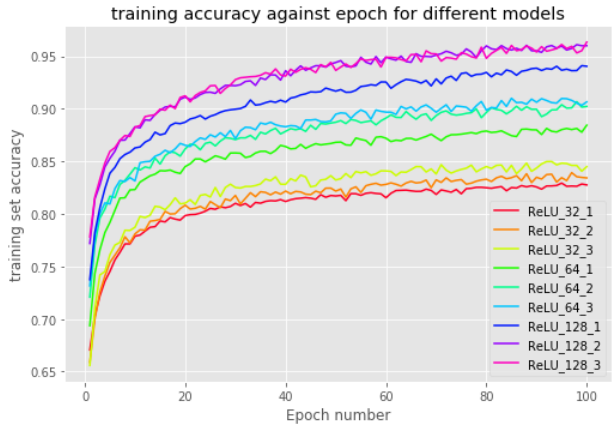
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \qquad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

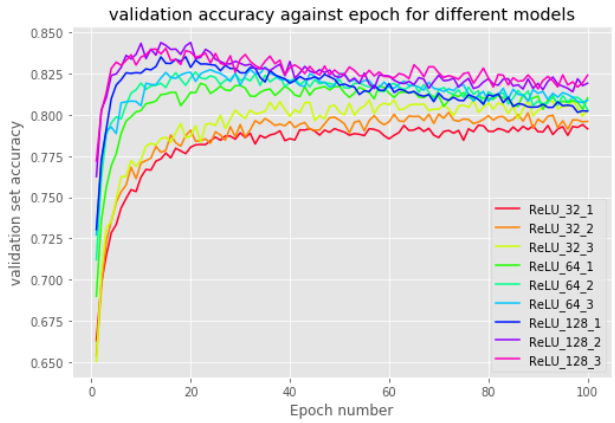
ReLU is very important for state-of-the-art neural networks. It has many advantages over other non-linear functions such as Sigmoid and Tanh activation functions. The major problems both Sigmoid and Tanh functions have are that they are comparatively computationally expensive and they all have the vanishing gradient problem. Gradient vanishing problem arises when the architecture becomes deeper, the derivative terms of the function that are less than 1 or larger than 1 will be multiplied each other many times so that the values will become either very small towards zero hence vanishing or the value tends to infinite large and explode the gradient. This can cause the network stopping or being to slow to learn further. A good solution to the problem is to keep the values to 1 so they can remain unchanged under multiplications. This is explicitly what ReLU does: it has gradient 1 for positive and the gradient is zero otherwise. Comparatively, ReLU functions also are much more computationally efficient because of its sparsity(less dense network due to gradient zero).

In our investigation, we use a multilayers model with various (1,2,3) hidden layers and various hidden units (32,64,128). In total, we test 9 different configurations for the model with ReLU activation function. This allows us to see how different hidden unit and layers can affect the prediction results. The learning rate may be the most impor-

tant hyperparameter for the neural network (7). It affects the learning speed and the outcome of the training process. We set the learning rate = 0.1 because learning rate = 0.1 is very commonly used as a value to set initially for an experiment according to many papers and articles (8)(9)(10). Instead of using learning rule as original Stochastic Gradient Descent (SGD), we use Adam Learning rule because of its superiority compare to the SGD. We set the epoch number to be 100 to ensure the model is not 'underfitting'. We use cross entropy with softmanx for our error measures because of advantages suggested in (8)(10). We initialised the weights by Xavier uniform initializer and bias by constant parameter initialiser as suggested in (8)(10) and labs. The seed used here is 11102019. We will not apply any regularisations method to the model for now since we are now simply designing a baseline for our investigation and we are more interested at the effect of number of hidden units and layers on the model.

The results of different models have shown below:

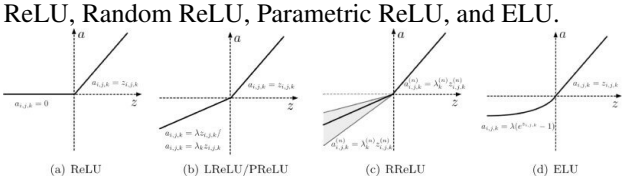




The graph shows an interesting factor that the accuracy of both the training and validation set can be divided by the number of hidden units into 3 main areas. The graph shows that the model with more hidden units and hidden layers reaches the higher part of the graph, so the model will have higher accuracy. However, with more layers and hidden units, the models are more easily to be overfitted, which is indicated at the downward trend of the line. Due to the restriction of this work, we have not explored further on more hidden units and layers. ReLU_128_3 (Model with ReLU function and 128 hidden units, 3 hidden layers) and ReLU_128_2 have really similar best validation accuracy, with ReLU_128_2 slightly higher (less than 0.005) than ReLU_128_3. However, we use ReLU_128_3 with best epoch (epoch with the highest accuracy)as baseline model because the difference is very small and it has advantage of being more comparable to others' work. As the result shows, the best epoch number is epoch number 15, with validation accuracy = 0.8403 (4d.p.) The baseline system has the test accuracy around 82.6%. This is a very good baseline model because the accuracy is very high, hence the accuracy of our final model with best configuration shall not below 82.6%.

3. Activation Function Comparisons

ReLU is a very powerful activation compare to Sigmoid and Tanh, but it also causes another problem called the dying ReLU problem. This is because all negative input in ReLU function will be mapped to 0 which can cause gradient go toward. The weights so will not get adjusted during descent. It causes those neurons into a state that being stop to respond to variations in error/input because of the 0 gradient. (3)(11)(12)(13)We call the neurons in such a state dead neurons. This can be solved by having some adaptations to original ReLU function. Here comes Leaky ReLU, Random ReLU, Parametric ReLU, and ELU.



(a) ReLU (b) LReLU/PReLU (c) RReLU (d) ELU

3.1. Leaky ReLU

For Leaky ReLU (LReLU), it has form of:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases} \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{if } x \leq 0 \end{cases}$$

where a is a manully selected value, usually in the range of 0.001-0.2. It is initially set to be 0.01 for our experiments. Both Random ReLU and Parametric ReLU share same form of equation as Leaky except the way how a is selected.

In LReLU, the term a overcomes the dying problem because it adds a very small value to gradient when input is negative. This prevents the problems caused by gradient zero. (13)(14)

3.2. Random ReLU

Random ReLU (RReLU) is the the randomized version of ReLU. It is first proposed in Kaggle NDSB Competition(14). Instead of being having only one fixed and manually selected a, each unit of network has a unique uniformly drawn a, which is also usually ranging from 0.001-0.2. a is initially set to have a lower bound 0.125, upper bound 0.3333333333333333 and a seed (rng=1234) for our experiments.

3.3. Parametric ReLU

The proposer of Parametric ReLU (PReLU) reported its performance is much better than ReLU in large scale image classification task. It is similar to LReLU except that the a is a learnable parameter and a is initialized uniformaly in the range 0.001-0.2 as well. It is initially set to 0.25 for our experiments, and a is learned from using backpropagation.

3.4. ELU

For ELU (Exponential Linear Unit), it has form of:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{if } x < 0 \end{cases}$$

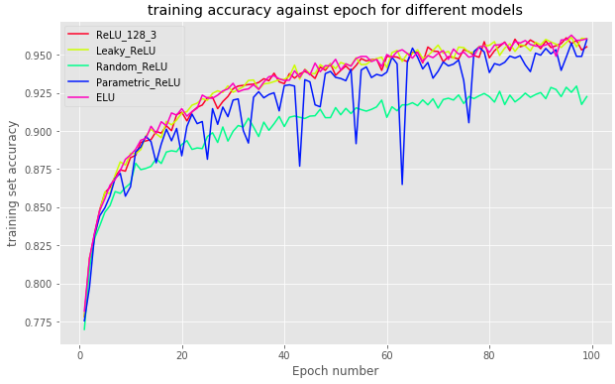
$$f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ f(x) + a & \text{if } x < 0 \end{cases}$$

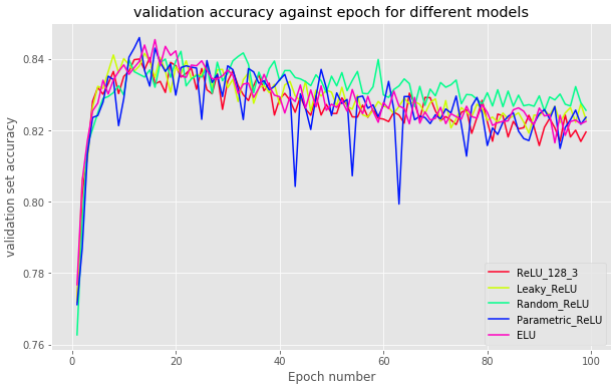
where a is a single manually selected scalar value. ReLU' learning is comparatively slow because of the bias shift. ELU uses the activation function in order to achive mean zero rather than using batch normalisation so learning can be made faster. Therefore, ELU is a good adaption to ReLU since it decreases the bias shift by pushing the meaning activation towards zero. (14). We initially set a to be 0.01 for our experiments.

3.5. Experiment

3.5.1. Glipse on different activation functions

The purpose of our investigation is to compare the generalization performance of each of those activation functions above as well as discovering the best possible network configuration, where the only available options to choose from are the activation functions and their hyper-parameters. The initial values are set according to (14)(15) Before we start tune the hyper parameters of the models with those activation functions just introduced, we firstly use same hyper parameters setting as baseline model except set the epoch number back to 100, so it gives us a better overview of the performance of each models. The results shown below:





Overfitting occurs while accuracy on validation set stops improving while the accuracy on the training set is continous to increase. This factor is shown in the graphs for all the models. Therefore, we say the models are overfitted(8). The graph suggests that 40 epochs is enough to prevent underfitting for all models. However, we will use epoch 50 for further exploration because it gives us more information about the result.

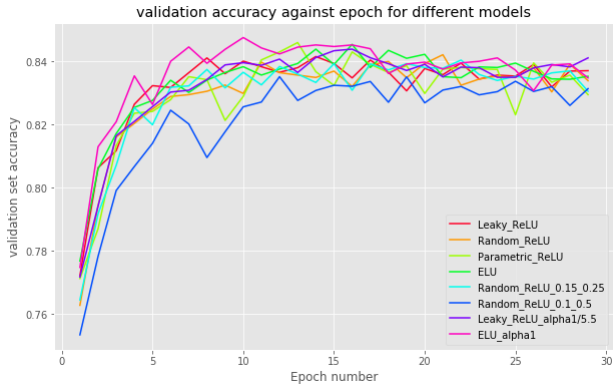
Model	Acc_valid
ReLU$_1$28$_3$	0.8404
LReLu	0.8416
RReLU	0.8422
PReLU	0.8460
ELU	0.8454

Table 1. different models' accuracy on validation set (4 d.p.)

From the table, the result suggests that ELU and PReLU have the highest accuracy. LReLU is slightly better than ReLU but not much better. RReLU is slightly better than LReLU. Those results are very reasonable because LReLU and ELU are adaptions to ReLU, and RReLU and PReLu are adaption to LReLU. Therefore, LReLU and ELU perform better than ReLU, and RReLu and PReLU perform better than LReLU.

3.5.2. alpha adaption

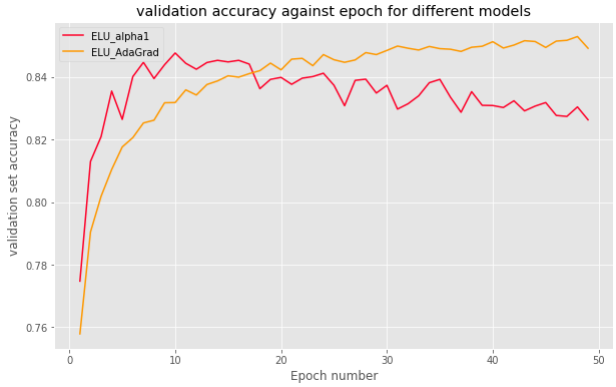
Now, we will tune alpha value in each adapted activation functions to see how alpha values affect on them. For LReLU, we will change alpha to 1/5.5, as a value suggested in another paper (16) to see whether this value will have better performance. Since the initial value we set for RReLU was suggested by NDSB competition winner (16), we have no expectation to find a better range for the RReLU. We investigate to narrow down the range and expand the range to see whether the different range will have better performance than the initial one. For PReLU, we remain it as before due to the factor it had the highest accuracy and a can be learned automatically. For ELU, we change the a to 1 as suggested in many other papers (16)(14)(17) Also, we will use only 30 epoch to train the data since the above graph suggest that epoch 30 is enough to be prevent from underfititing for the models. The reuslt shown below:



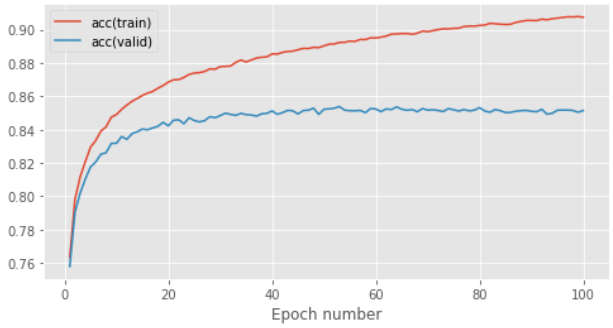
Clearly, the new LReLU has better performance than before. Unsuprisingly, the new ranges have no improvements in performance. ELU model with new value suggested by others papers performs better than PReLU now. The new ELU has better performance than initial ELU and PReLU. Therefore, in the purpose of time saving, we will use alpha =1 for further investigation.

3.5.3. Learning rule adaption

We will now investigate on how learning rule can affect the model. The learning rule options are the original SGD, Momentum Learning Rule, Adam Learning Rule, and AdaGrad Leaning Rule for our experiment. Since we already have chosen Adam Learning Rule for previous models, we will only compare our ELU_alpha1 model to ELU_alpha1 with AdaGrad Learning. According to many books and articles (8)(10)(18)(19)(7), and others' work observed, Adam and AdaGrad learning rule are better than the original SGD, Momentum Learning Rule. Therefore, we will explore only on Adam and AdaGrad. This time, we use epoch 100 to have more information on result for different learning rule. The results shown below:

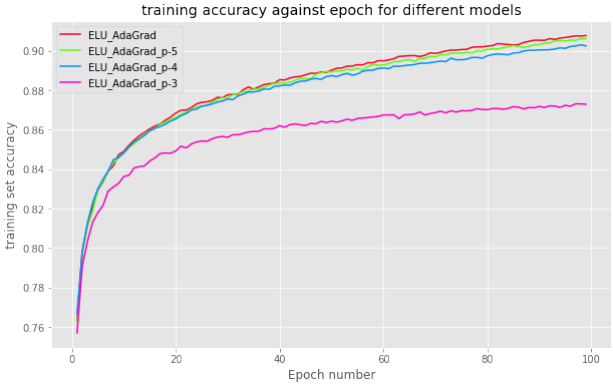


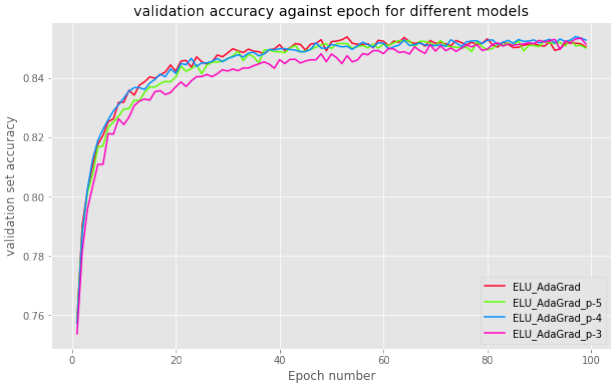
The graph shows clearly that AdaGrad learning rule is better than Adam. According to (18), AdamGrad "is well-suited for dealing with sparse data". Due to the factor that our data is spase (the most of datapoint is 0), I think that is the main reason why AdamGrad has convincing better result than Adam.



3.5.4. penalty

The above graph shows the training and validation accuracy of AdaGrad. There is a clear overfitting on the model. Therefore, we will apply L2 penalty to the model (the reason choosing L2 rather than L1 is because the results shown in Lab5). We have initially chosen the coefficient to be 10^{-5}. Applying penalty to the model has clearly reduce the overfitting but overfitting remains. Therefore, we apply higher and higher coefficients up to 10^{-3}. The results shown below:





As the result shows, the accuracy on the training is now much less than others and is much closer the validation accuracy. This is because applying penalty has very good effects on reducing overfitting (8)(10).

3.5.5. other hyper parameters

- Learning rate: learning rate is very important hyper parameters as we have mentioned above. We decide not to tune it because the learning rule we applied has dynamic learning rate and initial learning rate has smaller effects on it compare to other learning rules. Also, according to some others' work we have seen

and some naive tests we had initially on original ELU model, we decided not to use it. The outcomes of the tests will not be put on here because of the limitation of the pages.

- Weights and bias initialisation: according to (8)(10), it is one of very good way to set up for the neural network. This parameter is not our point of interest, so we decide not to tune it.
- Dropout, hidden layers, hidden units: For similar reason to the learning rate, we have observed some of others work and had some naive test ourselves, it has not much effect on the performance and due to the limitation of pages, we will not put it on.

3.6. Best Model

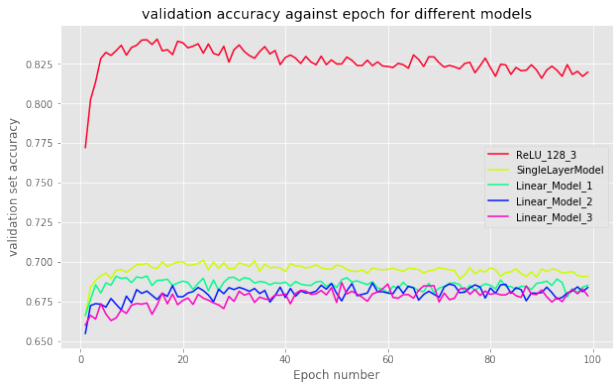
According to the graph above, the ELU model with AdaGrad learning rule, alpha = 1 and penalty coefficient = 10^{-3} is the model with best configuration so far. However, it seems that both training and validation accuracy are still increasing at epoch. Therefore, we further trained the model for another 100 times so epoch number is 200. We found that the result at epoch is 0.858 at epoch 199 (0.856 at epoch 200). Therefore, our best configuration for the model is ELU with AdaGrad Learning rule, alpha =1, penalty coefficient = 10^{-3} (L2 penalty) and epoch number = 199. The accuracy of the final model on validation set is 85.8%

4. Linear vs Non-Linear

Linear function has form:

$$f(x) = ax + b \qquad f'(x) = a$$

In order to find out whether non-linear functions are necessary for fitting deep neural networks on large datasets, we make comparison between the model with ReLU ReLU_128_3 (more or less the baseline model except it has 100 epochs) and the models with the linear function. We use same configuration as ReLU_128_3 for the models with linear function. Here is the result of models with linear function and with different hidden layers.



In the graph, it shows that ReLU has significatly better result than the models with linear functions. It is notable that the model with single layer (no hidden layer) has the best performance (accuracy of 70.1%) among the others. We deduce the reason for it is that: firstly, the final activation function of last is a linear function of the first layer independent from the number of layers. Secondly, in linear function, $f'(x) = a$ means that the gradient has no relationship with x. This means that if there is an error in prediction, the changes made by backpropagation is constant and is independent on input x. Therefore, with more layers, it passes more 'noisy information' to the model (since the gradient is independent on input) each time it back propagates, without helping the network learning from the data. Therefore, the model has worse performance for prediction. (13) Due to the fact that the models with linear function have very bad performance on a large dataset such as EMNIST, we can conclude that it is necessary to have non-linear for fitting the deep neural networks on large datasets.

5. Conclusions

Our best model is the Model with ELU activation function where alpha = 1, 128 hidden units, 3 hidden layers, AdamGrad Learning Rule, 0.01 for learning rate, Xavier uniform initializer for weights, constant parameter initialiser for bias, applied L2Penalty with penalty coefficients = 10^{-5} and epoch number = 199. The seed used is 11102019. Our model has test accuracy = 84.5%, which is about 2% better than baseline

MODEL	ACC_TEST
BASELINE	82.6%
FINAL$_{m}$*odel*	84.5%

Table 2. different models' accuracy on test set (4 d.p.)

The most three important things we learned from the experiments are: Firstly, activation functions are important for the neural network. Non-linear function for fitting neural network on large datasets has much better performance than linear function, as described in many articles and books (8)(10)(13). Secondly, ELU and PReLU are indeed much more powerful than LReLU discussed in others papers (17)(14). Thirdly, hyper parameters such as learning are very important and they directly affect the performance of a model (8).

There are few things that can be explored more. We can explore on whether increase in hidden unit and layers will further improve the performance. We can explore further on whether our final remains as the best configuration on other dataset such as MNIST, CIFAR-10 and CIFAR-100. Also, we can further train our final model because it seems that accuracy is still increasing at epoch 200.

References

[1] What is deep learning and how does it work? | content simplicity, 09 2019.

[2] Skymind. A beginner's guide to neural networks and deep learning, 2019.

[3] A gentle introduction to the rectified linear unit (relu) for deep learning neural networks, 04 2019.

[4] yaron. 30 amazing applications of deep learning, 03 2017.

[5] Handwriting recognition software, 2018.

[6] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André Van Schaik. Emnist: an extension of mnist to handwritten letters, 03 2017.

[7] https://www.facebook.com/jason.brownlee.39. Understand the impact of learning rate on neural network performance, 01 2019.

[8] Michael A Nielsen. Neural networks and deep learning, 2018.

[9] Neil Zhang. How to tune hyper-parameters in deep learning, 12 2017.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. The Mit Press, 2017.

[11] geva. Missinglink.ai, 2017.

[12] Avinash Sharma V. Understanding activation functions in neural networks, 03 2017.

[13] ML. Activation functions — ml cheatsheet documentation, 2011.

[14] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task, 04 2018.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[16] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolution network, 11 2015.

[17] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms, 01 2016.

[19] Vitaly Bushaev. Adam—latest trends in deep learning optimization., 10 2018.