MLP Coursework 1: Activation Functions

s1448320

Abstract

Image recognition has always been one of the most fundamental tasks in Deep Learning and general Artificial Intelligence.

The task of this experiment is hand-written digit classification (0-9 digits) using MNIST dataset (Modified National Institute of Standards and Technology) and various types of neural networks.

The main focus of this experiment is to explore the influence of different activation functions, number of hidden layers and strategies for weight initialisation on networks' performance (error rate and accuracy on training and validation sets).

1. Introduction

MNIST dataset is commonly used for various image processing tasks (Kussul, 2004) and originally consists of 60,000 training and 10,000 testing images of hand-written digits collected from American Census Bureau employees and American high-school students (Y. LeCun, 2013).

Each image in the dataset is 28 pixels high and 28 pixels wide. This means that a single image can be represented by 28 by 28 matrix, containing 784 entries (each entry represents a single pixel). This number also corresponds to the input dimension of the investigated neural networks. The same applies to the output dimension which corresponds to 10 possible classes (0-9 digits).

Since MNIST is widely used for supervised learning tasks, it also contains of sets of labels for training and testing sets, where each label corresponds to true class (digit) for each image.

For the purpose of this experiment, training set has been divided further into testing and validation sets containing 50,000 and 10,000 images respectively. This split is vital since it will allow us to monitor the network's performance and check if the data has been overfitted using a separate set of images which had not been seen while training the network (validation set).

The error rate, as well as the accuracy, on the validation set is monitored during the training process. In most of the cases, in the initial stage of the training process, the error on validation set decreases along with the error on training set. However, at later stages, the error on the validation set begins to rise. This phenomenon is known as overfitting which means that the network performs very well on the data that it has already seen by learning the irrelevant detail and noise in the training set but performs poorly on newly seen data (validation set). Overfitting impacts network's ability to generalize since the training set is just a sample of all possible configurations of the data.

In this experiment, we will investigate our implementation of different activation functions ("Leaky" ReLU, ELU, and SELU units) and their influence on training with respect to test/validation set error rate and accuracy. In next sections, we will focus on the theory behind each of the activation functions (section 2) and discuss their influence on the investigated network architecture (section 3). For this purpose, all experiments were conducted using 2 hidden layers, with 100 units per hidden layer to make the results more comparable. The performances of each investigated functions were then compared against baseline systems of the same architecture using Sigmoid and ReLU units separately.

This procedure will allow us to determine the best initialization function for the MNIST task and investigate it further in the next part of the experiment where we will determine the impact of the depth of the network with respect to accuracy using chosen activation function from the previous task (section 4). This time, we are considering networks with up to 8 hidden layers and the same number of hidden units per layer as before (100).

Furthermore, we will determine the impact of various initialisation strategies (Fan-in, Fan-out, Glorot and SELUInit) on the network performance.

The final section of this paper are conclusions regarding all the findings drawn throughout the experiment (section 5).

2. Activation functions

In this section we will focus on explaining the purpose of activation function in neural network and its different types used throughout this experiment. As it can be seen from Figure 1, activation functions introduce non-linearity to the neural network but to understand what it actually does, let us focus on the artificial neuron first. Each neuron calculates a "weighted sum" of its input, adds a bias and then decides whether it should be activated ("fired"). This description matches the following equation:

$$y = \sum(input \times weight) + bias \tag{1}$$

The value of the output (y) can take arbitrarily large (positive or negative) values and it has to be decided when the

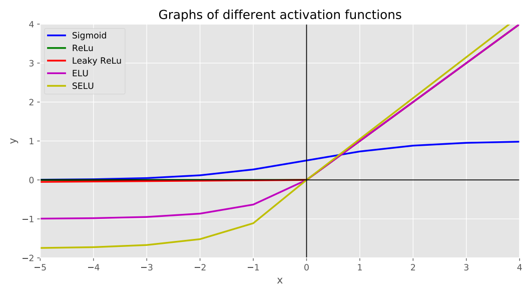

Figure 1. Graph of all the investigated activation functions.

neuron is activated. This can be achieved by introduction of activation function. Thanks to the activation function, outside connections are able to determine when the neuron is activated ("fired" as an analogy to neuroscience).

Without activation function the output would be a linear function and such a neural network would become a linear regression model. Linear equations are not complex enough (degree one polynomial) to capture complex mappings from the data. This is because the real-life tasks are much more complex and usually non-linear (Sharma, 2017).

The most important properties of activation functions, apart from non-linearity, include:

- Continuous differentiability, which enables the usage of gradient-based optimization methods.
- Complexity (time to compute) since it has to be computed a great number of times during the training phase.
- The effect of saturation which is also known as "killing" the gradient. Saturation relates to the functions which gradients are very small at any point. When a gradient approaches to 0 (almost constant function) almost no signal will flow through the neuron to its weights and recursively to its data during backpropagation. This phenomenon is called "killing" the gradient and results in very slow learning.

Let us focus on each activation function separately to describe their properties.

2.0.1. Rectified Linear Unit (ReLu)

The first function that we are going to consider is ReLu. It simply outputs 0 for all negative inputs. Its equation is as follows:

$$\text{relu}(x) = \max(0, x), \tag{2}$$

and has the following gradient:

$$\frac{d}{dx}\text{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{3}$$

A big advantage of ReLu is that it is very easy to compute compared to some other functions (Sigmoid, Tanh) by simply thresholding a matrix of activations at zero. As a result, the activation is more sparse and less calculation is being performed. The research (Krizhevsky et al., 2012) has shown that using ReLUs results in much faster training for large networks and gives very accurate results (factor of 6). Unfortunately, ReLu tends to be very sensitive to learning rates (performs especially poorly for high learning rates) and some of the units can "die". This is because of its horizontal line for negative inputs, which means that the neurons which fall into that section will not respond to variations in input from that point onwards and "die" since the gradient is equal to zero.

2.0.2. "Leaky" Rectified Linear Unit (Leaky ReLu)

"Leaky" ReLu is a variation of the previous function which tries to fix the problem of "dying" units by introducing a new parameter α.

In this case, any input value less than zero is multiplied by the constant α which is usually equal to 0.01 and this value had been used throughout this experiment. This introduces a minimal slope for the negative values to keep the updates "alive".

"Leaky" ReLu shares all the advantages of normal ReLu unit but additionally deals with "dying" units. However, its improvement over ReLu unit has not been confirmed due to inconsistent results (Clevert et al., 2015).

2.0.3. Exponential Linear Unit (ELU)

The biggest advantage of ELU is that it takes negative values for all inputs ($exp(x) - 1$ term) which are less than zero. This shifts the mean activation closer to zero compared to ReLu and speeds up the learning process. This has a similar effect as batch normalization but is less computationally expensive. Its equation is as follows:

$$\text{elu}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{4}$$

and has the gradient:

$$\frac{d}{dx}\text{elu}(x) = \begin{cases} \alpha e^x & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{5}$$

Here, the value of α has been estimated to 1 and this value has had been used throughout this experiment.

According to some papers, ELU shows promising results in some image recognition tasks similar to MNIST (especially for deeper networks). The research (Clevert et al., 2015) states that ELU is more robust to noise than "Leaky" ReLu and speeds up the overall learning process significantly.

2.0.4. Scaled Exponential Linear Unit (SELU)

An ideal property of an activation function is to keep an activation for every layer as close to zero mean and unit variance as possible. This will ensure that tensors propagated through several layers converge towards zero mean and unit variance. As a result, the learning can be more stable by preventing gradients from exploding or "dying".

SELU is a variation on ELU but with an additional parameter λ and a slightly different value for α. These parameters seem to scale the function in such a way that it satisfies the conditions stated better than the functions described previously (Klambauer et al., 2017).

This had also been confirmed in research, where SELU seems to perform better on various tasks (including MNIST) when compared to other types of functions, especially as the network gets deeper (Klambauer et al., 2017). ELU function looks a follows:

$$\text{elu}(x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} \tag{6}$$

and has the gradient:

$$\frac{d}{dx}\,\text{elu}(x) = \lambda \begin{cases} \alpha e^x & \text{if } x \le 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{7}$$

For standard scaled inputs with mean equal to 0 and standard deviation of 1, the values of α and λ had been estimated to 1.6733 and 1.0507 respectively (Cohen, 2017) and these values had been used throughout this experiment. With value of λ being slightly greater than 1, we guarantee the slope to be greater than 1 for all positive inputs.

3. Experimental comparison of activation functions

In this section we will present and discuss the results of the experiments of comparing networks performance using mentioned activation functions on the MNIST task.

All of the investigated networks consist of 2 hidden layers with 100 hidden units per layer. Each of them had been trained on the MNIST dataset with batch size of 50 for a total of 100 epochs. Additionally, a wide range (0.005, 0.01, 0.05, 0.1, 0.2, 0.4) of different learning rates was chosen in order to investigate sensitivity of different units on this parameter change. We will also compare the results with baseline systems of the same architecture using Sigmoid and ReLU units.

3.0.1. "Leaky" ReLu

Networks with "Leaky" ReLu units seems to learn relatively faster than those with Sigmoid for corresponding learning rates. Even for small learning rates such as 0.005, final accuracy for validation set was 0.974 compared to 0.932 for Sigmoid. This difference decreases as the learning rate gets higher.

"Leaky" ReLu units make the network much more prone to overfitting the data than in case of Sigmoid. When comparing graph shapes of their accuracies for the same learning rates (Figure 2), the accuracy on the validation set is much smaller compared to training set at the same point. The split happens sooner for ReLU and LReLu than Sigmoid and indicates overfitting. The same principle applies for error rates.

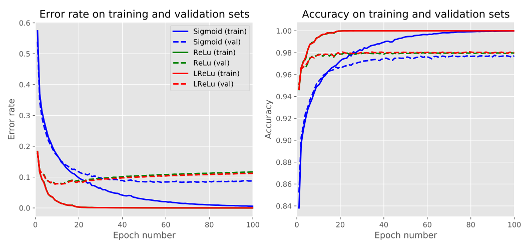

Figure 2. Graphs of error rates and accuracies on training and validation set using Sigmoid, ReLu and "Leaky" ReLu units with the learning rate of 0.1.

There was almost no difference recorded between values for "Leaky" ReLu and ReLu units in case of error rates and accuracy (both training and validation), small value of alpha did not prevent the units from "dying" almost at all (especially for smaller learning rates). The difference increases slightly with the growth of learning rate (0.1455 compared to 0.1388 for "Leaky" ReLu, corresponding to 4.6% decrease of final error rate on validation for learning rate of 0.4). This matches the conclusions of the research mentioned before (Clevert et al., 2015).

3.0.2. ELU

Since "Leaky" ReLu unit had relatively the same effect on the learning process as ReLu, we will not include it in our graphs for the sake of clarity of the plots. Moreover, the same applies to comparison of ELU and SELU to Sigmoid unit since the former are relatively very similar to ReLu (in case of error rate and accuracy values considering the scale). Hence, the same describes the difference between those and Sigmoid units as for "Leaky" ReLu.

ELU unit seems to have very similar effect on training as "Leaky" ReLu. However, the error rate gets smaller for ELU when compared to ReLu for corresponding points for higher learning rates and the opposite for accuracy. For the learning rate of 0.1, the difference between the two is negligible (Figure 3) but grows with the number of epochs, indicating poorer performance for ELU (its accuracy graph gets below ReLu). However, as the learning rate increases (0.4), ELU tends to achieve higher accuracy and lower error on the validation set. This suggests that ELU unit is better at preventing neurons from "dying" as the learning rate increases but the differences are relatively small. Final error on validation set of 0.1358 for ELU compared to 0.1455 for ReLu which corresponds to 6.7% decrease in error rate. This is a slightly bigger decrease of the error rate but still not very significant.

3.0.3. SELU

SELU performs in a similar pattern as ELU for higher learning rates but it seems more sensitive to the small changes of

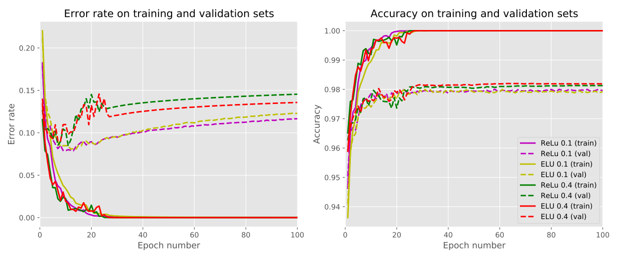

Figure 3. Graphs of error rates and accuracies on training and validation set using ReLu and ELU units with the learning rate of 0.1 and 0.4.

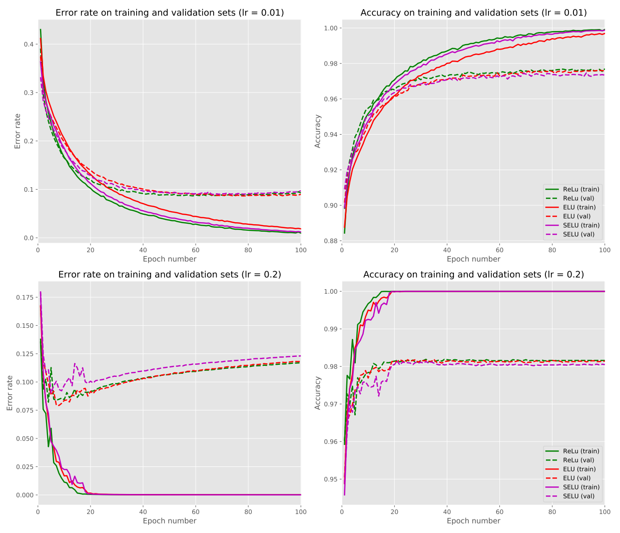

Figure 4. Error rates and accuracies (training and validation) for SELU unit with learning rate of 0.01 (top) and 0.2 (bottom)

this hyperparameter. If we focus on smaller learning rates (0.005 to 0.01), SELU tends to perform better than both ReLu and ELU at very early stages of training (ca. first 5 epochs). This applies to both training and validation sets for accuracy and error rate. This difference vanished in the later stages of training but ELU's performance remains very close to ReLu's rather than ELU's. SELU performs much better with respect to error rate and accuracy for training data set than ELU and relatively similar to ReLu. However, this cannot be said about validation set. In the later stages of training, the accuracy of the validation set falls slightly below values of ELU at the corresponding points. This is especially visible when the learning increases (0.2). This being said, it should be noticed that an overflow of the gradient occurs for much lower learning rate (0.4) in case of SELU than for any other investigated unit. This has been indicated by a warning message generated by Python and error values of "*nan*". This was caused by the learning rate being set too high for particular layer, since we are taking very big steps, and at some point enter a region of the parameter space which is completely different from the previous point. The loss increases drastically until the overflow occurs ("*nan*" values).

No. of hidden layers	Final error (train)	Final error (valid)
2	4.063×10^{-2}	9.478×10^{-2}
4	7.479×10^{-3}	1.059×10^{-1}
6	1.725×10^{-3}	1.316×10^{-1}
8	6.346×10^{-4}	1.391×10^{-1}

Table 1. Final error rates on training and validation set for different numbers of hidden layers and learning rate of 0.005 for SELU unit and Grolot initialisation.

4. Deep neural network experiments

In this section we will investigate the impact of depth (number of hidden layers) of the network with respect to its accuracy using a chosen activation function (SELU). We will also consider different approaches to weight initialisation and their impact on the overall network's performance.

To make the results more comparable, the activation function and the learning rate were decided to be kept constant for the rest of this experiment.

We decided to explore SELU unit due to its very promising results on the network performance for smaller learning rates. This choice was also motivated by the recent papers (Klambauer et al., 2017) where SELU performed particularly well when used in deeper networks.

Due to SELU's sensitivity for change of learning rate and its very good performance for very low values of this hyperparameter, we decided to set the learning rate for the lowest value used in the previous section, namely 0.005.

4.0.1. Number of hidden layers

The final error rate on the training set, consistently decreases as more layers are added to the network (Table 1). This is consistent with the theory, according to which the increase in number of hidden layers (with the same width), results in the increase of total number of parameters in the model. This makes the model fit the data even more accurately. However, an opposite pattern has been recorded for the error on validation set. As the number of hidden layers increases, the validation error also increases.

Looking at the graph (Figure 5) we can notice that after a particular amount of epochs (depending on the depth but sooner for deeper networks), the error on the validation set begins to increase while the the error on the training set still decreases. This is a clear indication of overfitting. This is due to the fact that as the networks gets deeper, it is able to learn more complex representations (Heaton, 2017) and it can overfit previously seen data much sooner (training set).

Therefore, we can conclude that deeper networks are more prone to overfitting and particular caution should be kept when dealing with deeper architectures. However, it does not mean that models with less hidden layers are gener-

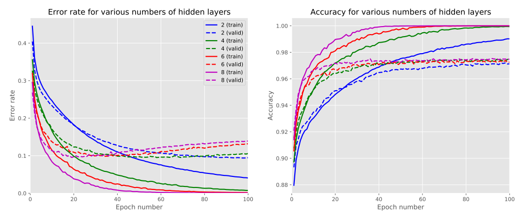

Figure 5. Graphs of error rates and accuracies on training and validation set for different numbers of hidden layers and learning rate of 0.005 for SELU unit and Grolot initialisation.

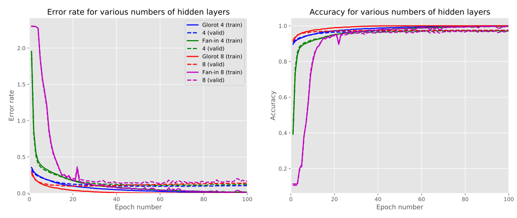

Figure 6. Graphs of error rates and accuracies on training and validation set for different numbers of hidden layers (4,8) for Glorot and Fan-in initialisations.

ally preferred. As indicated in most recent papers, the most accurate networks increased in depth from 7 layers (AlexNet, (Krizhevsky et al., 2012)) to over 1000 (ResNet, (He et al., 2015)) in a span of just 4 years with various number of parameters in each layer and additional methods being added to minimize the effect overfitting. The most trivial example of this is "early stopping" which simply means stopping the training process as soon as the validation error starts to increase (gets bigger than the error on the training set). Moreover, learning more complex representations by adding more hidden layers turns to be crucial in most recent image recognition tasks where each layer of deep convolution neural network (CNN) is distinguishing different and more complex features on an image (Nguyen et al., 2016).

It should also be mentioned that deeper models take longer time to train due to the increased number of parameters (more neurons) which can be a serious constraint in some tasks. This issue could be overcame by using modern hardware (GPUs) and parallelisation (Nemire, 2014).

4.0.2. Initialisation strategies

As a next step, we will investigate different weight initialisation strategies of network's performance.

Why is it so important to initialise weight properly? If the weights are initialised with too small values, the signal shrinks as it is propagated through each layer until its value becomes too small to have any effect on the final result. The same principle applies to the weights being too big to be any useful for the optimisation. They may even cause an overflow. Therefore, a proper initialization of the weights in a neural network is critical to its convergence (Kumar, 2017).

It is generally concluded that an appropriate parameter initialisation scale should depend on layer dimensionality. This way, the variance of activations and backpropagated gradients will be stabilised throughout the network and we will avoid vanishing or diverging activations which can make training very slow or even impossible.

Up to this point, the only initialisation strategy used was one proposed by Glorot and Baggio (Glorot & Bengio, 2010) where the weight for each layer are initialised from a uniform distribution with variance inversely proportional to the sum of the input and output dimensions of the layer. This gave us reasonable results and allowed convergence. However, we will consider other strategies, namely constraining the estimated variance of a unit to be independent of the number of incoming connections (n_{in}), and a corresponding strategy independent of the number of outgoing connections (n_{out}):

$$Fan-in : w_i \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}}) \tag{8}$$

$$Fan-out : w_i \sim U(-\sqrt{3/n_{out}}, \sqrt{3/n_{out}}) \tag{9}$$

$$Glorot : w_i \sim U(-\sqrt{6/(n_{in}+n_{out})}, \sqrt{6/(n_{in}+n_{out})}) \tag{10}$$

Where U is the uniform distribution. As follows from the above equation, Glorot initialisation is simply a combination of the first two strategies. Looking at the graph (Figure 6.) we can notice a very poor performance of the network with Fan-in initialisation strategy, especially during the first 15 epochs of training. This is due to the backpropagation condition for variance not being considered for this strategy:

$$Var(\Delta^{(i)}) = n_{i+1} Var(\Delta^{(i+1)}) Var(w^{(i)}) \tag{11}$$

Here, variance of the input gradient and the output gradient is not kept a stable level throughout the training which results in less appropriate weights initialisation for Fan-in compared wit Glorot.

Fan-out strategy, for which only backpropagation case is considered, turns out to have even worse influence on network's performance. This strategy allows the weights to be initialised with more varied values which led to overflow during the training for deeper networks with more than 4 layers.

The last strategy considered in our experiment was SELUInit strategy recommended (Klambauer et al., 2017) which draws from a Gaussian distribution (as opposed to uniform for the previous strategies) with mean 0 and variance of $1/n_{in}$. By using Gaussian instead of uniform distribution, "more probability" is assigned to the values closer to 0 since the mean is 0 (cumulative distribution function grows sharply near 0). Uniform distribution on the other hand assigns the same probability to all values within its range. Hence, values closer to 0 are equally probable as

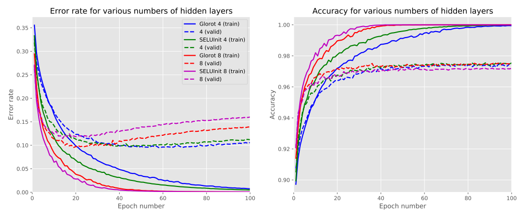

Figure 7. Graphs of error rates and accuracies on training and validation set for different numbers of hidden layers (4,8) for Glorot and SELUInit initialisations.

the values further from it. Drawing from Gaussian distribution turns out to have a positive impact on the network performance compared to Glorot initialisation (uniform distribution).

This may also explain why SELUInit's advantage over Glorot decreases with network depth. Since the weights in the network are initialised to smaller values, as the networks gets deeper, the "gradient signal" flowing backwards through a network can diminish and have smaller effect on learning (Figure 7.).

SELUInit seems to outperform Glorot strategy for shallower network architectures (up to 4 layers). However, the improvement is not significant.

5. Conclusions

All investigated networks have been successfully trained and gave us a valuable insight into various possible types of network architecture.

We have tested a variety of initialisation functions using different learning rates (section 3). According to those, no improvement in network performance had been recorded for "Leaky" ReLu. Adding new parameter α did not prevent neurons from "dying". This also matches the hypothesis from the previous researches (Andrew L. Maas & Ng, 2013). Using ELU unit instead, improved the final accuracy on the validation set for higher learning rates. However, this change has not been significant.

SELU has shown very promising a influence on network performance for smaller learning rates especially during the early stages of training. Along with recent literature confirming SELU's positive influence on deeper network's performance, we decided to investigate it further, adding more hidden layers and using different initialisation strategies.

This experiment has shown that appropriate choice of activation function can improve network's performance significantly and it should be carefully considered before designing its architecture.

As we proceeded further with SELU unit, we investigated the influence of number of hidden layers on network's performance (section 4). As the network got deeper (more hidden layers), the training process took longer time to complete the set limit of 100 epochs due to increased number of parameters needed to be calculated. However, the accuracy on the validation set was constantly increasing (from 0.9713 for 2 hidden layers to 0.9748 for 8 hidden layers, using Glorot initialisation).

It should also be mentioned that deeper networks tend to overfit the data sooner, hence additional techniques should be addressed to overcome this issue such as batch normalisation, cross validation or "early stopping" (Klambauer et al., 2017).

These results seem to find confirmation in theory and recent papers where the most accurate networks were build using up to 1202 layers (ResNet, (He et al., 2015)). Hence, it can be confirmed that the architecture with more hidden layers tend to be more accurate.

We have also explored different strategies for weights initialisation (section 4). Fan-in and Fan-out approaches turned out to be less efficient than their combination proposed by Glorot and Baggio (Glorot & Bengio, 2010) due to consideration of just one condition imposed on the variance (independent of inputs or outputs).

Introduction of drawing from Gaussian distribution (SELUInit) rather than uniform (Glorot) seems to have beneficial influence of shallower networks' architectures. However, the change is not significant and should be investigated further using different learning rates and deeper networks.

Due to time constrains, deeper networks with ELU unit have not been tested which could also be an interesting case for future research.

References

Andrew L. Maas, Awni Y. Hannun and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. 2013. URL https://pdfs.semanticscholar.org/367f/2c63a6f6a10b3b64b8729d601e69337ee3cc.pdf.

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. URL http://arxiv.org/abs/1511.07289.

Cohen, E. Selu - make fnns great again (snn), 2017.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In Teh, Yee Whye and Titterington, Mike (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition.

CoRR, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Heaton, J. The number of hidden layers, 2017. URL http://www.heatonresearch.com/2017/06/01/hidden-layers.html.

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017. URL http://arxiv.org/abs/1706.02515.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kumar, Siddharth Krishna. On weight initialization in deep neural networks. *CoRR*, abs/1704.08863, 2017. URL http://arxiv.org/abs/1704.08863.

Kussul, E. Improved method of handwritten digit recognition tested on mnist database. In *Image and Vision Computing*, pp. 971–981, 2004.

Nemire, B. Cuda spotlight: Gpu-accelerated deep neural networks, 2014. URL https://devblogs.nvidia.com/parallelforall/cuda-spotlight-gpu-accelerated-deep-neural-networks/.

Nguyen, Tung Duc, Mori, Kazuki, and Thawonmas, Ruck. Image colorization using a deep convolutional neural network. *CoRR*, abs/1604.07904, 2016. URL http://arxiv.org/abs/1604.07904.

Sharma, A. Understanding activation functions in neural networks, 2017. URL https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

Y. LeCun, C. Cortes. Mnist handwritten digit database. 2013.