MLP Coursework 1: Activation Functions

s1450710

Abstract

Evaluation of different activation functions (ELU, SELU, and LReLU) was carried out against two baseline activation functions (ReLU and Sigmoid) on the MNIST Task. It was found that SELU performs best amongst the three, and have comparable performance to ReLU. Further investigation was carried out on SELU activation function: 1) experimental investigation on performance with varying number of hidden layers; 2) experimental investigation of different weights initialisation methods. It was conclusive that SELU experienced less perturbations while converging with larger number of layers. However, no conclusive outcome can be gathered from experimenting SELU with different weight initialisation methods. Further experiments should be carried out to investigate the self-normalising property of SELU.

1. Introduction

In this report, we detail the methodology and finding of experiments set out to investigate: 1) The behaviour of different activation functions, namely: Leaky Rectified Linear Unit (LReLU) (Maas et al., 2013), Exponential Linear Unit (ELU) (Clevert et al., 2015), and Scaled Exponential Linear Unit (SELU) (Klambauer et al., 2017), compared against Sigmoid and Rectified Linear Unit (ReLU) (Glorot et al., 2011). This is elaborated in Section 2 with my findings presented in Section 3. 2) The impact on accuracy due to varying depth of models - number of hidden layers, and weight initialisation methods (Section 4).

The following subsections provides an overview of the MNIST classification task and the methodology for training, validation, and testing our models.

1.1. MNIST Task and Dataset

The Task in our investigation is to classify images of digits written by human into their respective classes (ranging from `0` to `9`). The MNIST dataset (LeCun et al., 1998; 2010) contains training, validation, and testing datasets, where each sample is an a grayscale image of 784 pixels. Samples are labelled according to the class it belongs to. In total, there are 50,000 images for training, and 10,000 images for validation and test each. The performance a model is measured using the accuracy score.

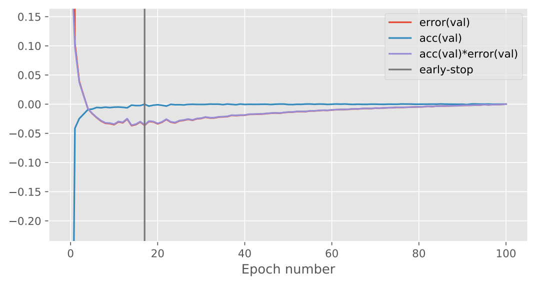

Figure 1. Best epoch for ELU activation function with 2 hidden layers, 100 units each. The best epoch was found (vertical line) where the validation error is low but with a validation accuracy approach the final validation accuracy. This is an attempt to find the early-stopping epoch.

1.2. Training, validation, and overfitting

A cross-entropy softmax error function is used for this supervised learning task with mini-batch of stochastic gradient descent (SGD) of size m. For the nth mini-batch input, the cross-entropy softmax error is:

$$E^n(\mathbf{t}, \mathbf{y}) = -\sum_{i=1}^{m}\sum_{k=1}^{C} t_{ic} \cdot ln(y_{ic}) \tag{1}$$

A mini-batch SGD algorithm was used for training. The training dataset was split into batch-size of 50 images each, and each iteration of an epoch involves forward propagating the images (activations) through the model, back propagating the model error and updating model coefficients (the weights and biases of each layer) to minimise the cross-entropy softmax error. The validation and test dataset was also split into mini-batches of 50 images each.

A total of 100 epochs was carried on each model. It was observed that after 100 epochs, the model has overfitted to the training data (Figure 1). This is because despite the training error approaching 0.0%, the validation error increases after dipping to a local minimum. We can therefore not use the final validation error and accuracy score to compare models, as that is not the best performance of the model on unseen data.

To avoid the problem of overfitting, the validation error and accuracy of the model is used to evaluate if the model have overfitted at a particular epoch. A simple algorithm was used in place of more sophisticated ones suggested by (Prechelt, 1998). The idea is to find the epoch that is approaching the final validation error while having the

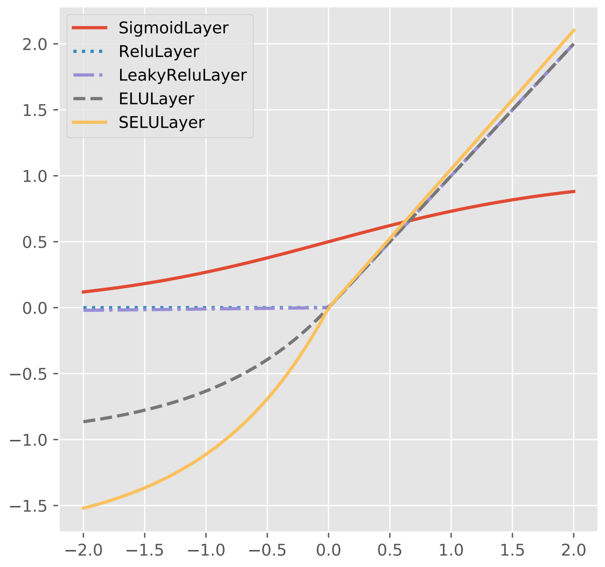

Figure 2. Sigmoid and ReLU are baseline models, while we conducted experiment to compare the performance of models that uses LReLU, ELU, and SELU.

lowest validation error. We then use the test accuracy at this epoch to compare the models. Although it was explained that validation accuracy should be used to compare the models, the algorithm's usage of validation accuracy has bias the the comparison metric. Hence, another metric - test accuracy, which we do not know before hand, is used instead.

2. Activation functions

There are five activation function (Figure 2) of interest to our experiments, Sigmoid and ReLU activation functions are used as baseline models, while LReLU, ELU, and SELU are activation functions for us to compare against. In the following subsections, we elaborate each function's characteristic.

The sigmoid function is a continuous and monotonically increasing S-shaped curve, that is similar to a step function. The sigmoid activation function suffers from the vanishing gradient problem in deep architectures, as the gradient tends to 0 at deeper layers when the activation is saturated at near -1 or 1, and converges slower than non-linear functions (Bengio et al., 1994). Mathematically, sigmoid function and its gradient is:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp^{-x}} \tag{2}$$

which has gradient of:

$$\frac{\mathrm{d}}{\mathrm{d}x}\,\text{sigmoid}(x) = (\,\text{sigmoid}(x))(1 - \text{sigmoid}(x)) \tag{3}$$

The other baseline model is ReLU (relu) and the remaining activation functions use for comparison are non-linear and anti-symmetric. ReLU builds on the old concept of half-wave rectified units where stages of the mammalian visual cortex is built on layers of abstraction(Malik & Perona, 1990), which was also observed in the layers of features learned in deep neural network (Goodfellow et al., 2009; Nair & Hinton, 2010). Glorot et al. investigated deep architecture with rectified linear unit and found that despite the gradient problem and sparse representation, the non-linear sparse network outperforms their saturating sigmoidal and tanh counterparts. Using ReLU units in deep networks have promising results in computation vision problems (Krizhevsky et al., 2012) and speech processing (Dahl et al., 2013).

Formally, ReLU is defined as:

$$\text{relu}(x) = \max(0, x) \tag{4}$$

which has the gradient:

$$\frac{\mathrm{d}}{\mathrm{d}x}\,\text{relu}(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{5}$$

While ReLU have promising results, few researchers question if the zero gradient problem and sparsity is really beneficial for deep networks (Xu et al., 2015). Others created alternate non-linear representations:

LReLU allows a small gradient to be derived, hence avoiding the zero gradient problem in ReLU. Maas et al. used LReLU for speech processing task and found that it performs *nearly identical* to ReLU. The authors further alluded that gradient-based optimisation methods can be used, which could possibly improve networks with ReLU-like properties. Mathematically, LReLU have the following form:

$$\text{lrelu}(x) = \begin{cases} \alpha x & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} \tag{6}$$

α is a constant typically assign as 0.01. lrelu has the gradient of:

$$\frac{\mathrm{d}}{\mathrm{d}x}\,\text{lrelu}(x) = \begin{cases} \alpha & \text{if } x \le 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{7}$$

The exponential linear units (ELU and SELU) allows a gradient that is proportionate to the value of exp^x when $x \le 0$ when the unit is deactivated. Such activation function pushes the mean of the activations closer to zero, and allows for the deactivation of the unit to be propagated. This is similar to using Batch Normalisation that regularise the inputs (activations) to each layers to counter the bias shift. Both (Clevert et al., 2015; Klambauer et al., 2017) concluded that batch normalisation with ELU and SELU, respectively, does not affect models with ELU and SELU; but does significantly improve networks with ReLU activation function.

The form of ELU is:

$$\text{elu}(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} \tag{8}$$

which has the gradient of:

$$\frac{\mathrm{d}}{\mathrm{d}x}\,\mathrm{elu}(x) = \begin{cases} \alpha(\exp(x)) & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{9}$$

SELU have two parameters: $\lambda \approx 1.0507$ and $\alpha \approx 1.6733$:

$$\mathrm{selu}(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{10}$$

which has the gradient of:

$$\frac{\mathrm{d}}{\mathrm{d}x}\,\mathrm{selu}(x) = \lambda \begin{cases} \alpha(\exp(x)) & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{11}$$

$$= \lambda \begin{cases} \mathrm{selu}(x) + \lambda\alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{12}$$

It is worth noting that the use of exponentials in ELU and SELU allows us to represent their gradients as the output of the activation function, as shown in equation 12 for the case of SELU. This means that we could save some computation back propagating the gradients with ELU and SELU activation units.

3. Experimental comparison of activation functions

Experiment Setup: I set out to observe if there is any difference in the test accuracies between each model. All the models in this experiment uses the same network architecture: two hidden layer with 100 hidden units each. A mini-batch SGD algorithm was used with similar hyperparameters: learning rate (0.1), weights (using Xavier Initialisation (Glorot & Bengio, 2010)), and bias initialisation methods. The only variable is the activation function.

Hypothesis:

1. Most function under comparison will outperform and converge quicker than baseline sigmoid; the LReLU should perform as good or better than ReLU, as noted in Maas et al. 2013.
2. ELU and SELU should perform as good or better than ReLU, as noted in Klambauer et al. 2017; Clevert et al. 2015

Results: Comparing the three activation function (LReLU, ELU, and SELU) against the Sigmoid and ReLU models (Table 1), most activation function outperforms the baseline sigmoid, and are comparable to baseline ReLU. However, we do not observe that ELU and SELU perform as good as baseline ReLU, as claimed in the papers.

Analysis: The weight initialisation method, although a constant factor, influence the performance of the network. In specific, *Xavier Initialisation* is ideal for units that are linear, such as sigmoid and tanh functions (Glorot & Bengio, 2010); initialisation from uniform distribution of mean

Function	error(valid)	acc(valid)	error(test)	acc(test)
Sigmoid	1.042e-01	9.698e-0	1.073e-	9.671e-01
RELU	9.111e-02	9.770e-0	8.466e-	9.759e-01
LRELU	8.934e-02	9.775e-0	8.276e-	9.774e-01
ELU	8.038e-02	9.773e-0	7.397e-	9.780e-01
SELU	8.974e-02	9.762e-0	8.868e-	9.749e-01

Table 1. Validation accuracy and test accuracy at the best epoch for each activation function. In this experiment, **Xavier uniform weights initialisation** was used.

Function	error(valid)	acc(valid)	error(test)	acc(test)
Sigmoid	1.805e-01	9.492e-0	1.953e-	9.427e-01
RELU	9.379e-02	9.765e-0	9.252e-	9.751e-01
LRELU	9.929e-02	9.752e-0	9.650e-	9.731e-01
ELU	8.930e-02	9.752e-0	8.805e-	9.736e-01
SELU	9.205e-02	9.744e-0	7.993e-	9.753e-01

Table 2. The experiment was repeated with **normal weight initiation** with mean 0 and variance 0.01. All non-linear functions have comparable test accuracies. The best performing activation function is the SELU, comparable to ReLU.

0 and variance $1/n_{in}$ is ideal for SELU functions (Klambauer et al., 2017); initialisation from uniform distribution of mean 0 and variance $2/n_{in}$ is ideal for rectified non-linear units (ReLU, LReLU) (He et al., 2015).

3.1. Using normal weight initialisation

To adjust for the different ideal weight initialisation methods and not use one initialisation method that leads to better result for one, normal initialisation with mean 0 standard deviation of 0.01 was used. This was a popular weight initialisation technique in training neural networks, although it was observed that it can be problematic to train deep networks (Mishkin & Matas, 2015).

Results: From the results (Table 2) we can observe that the baseline sigmoid still performs poorer than the rest of the activation function. We, however observe an interesting pattern where the SELU performs better than ReLU, instead of observing that LReLU is as good as ReLU. The slightly better results of SELU to ELU is expected..

3.2. Analysis of activation functions

While the above experiments (Figure 3) have provided some evidence of merits for SELU layer, it could be that the “neutral" weight initialisation method in the previous experiment actually aid SELU to perform better. Nevertheless, it is clear that rectified units does indeed allow the network to converge faster than sigmoid function.

4. Deep neural network experiments

In this section, results from experiments on deeper networks for MNIST is presented. Following the results from previous section, **SELU activation function** is used for the hidden layers for all experimental models. In the first experiment we vary the number of hidden layers, from two to eight, for each model while keeping other parameters unchanged. In the second set of experiments, we use dif-

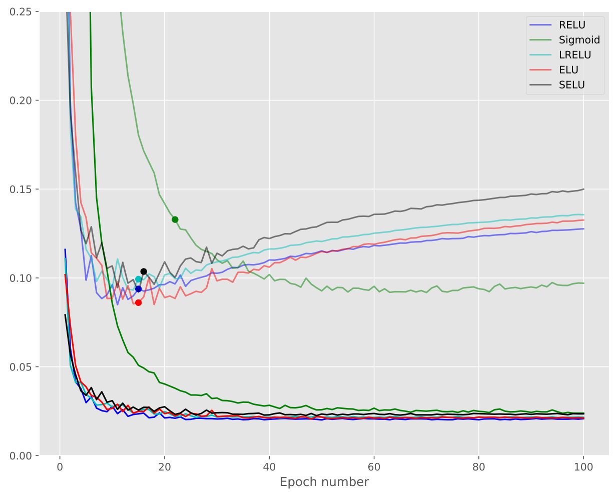

Figure 3. Validation errors (lines in lighter tone) and validation accuracies of the activation functions with using two hidden layer with 100 hidden units each

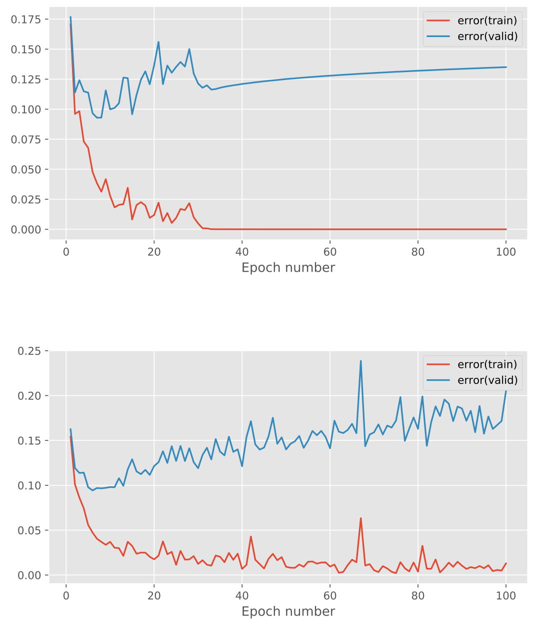

Figure 4. Using stochastic gradient descent with increasing number of hidden layers (left: 4 hidden layers; right: 6 hidden layers) causes the errors to be fluctuating while approaching the local minima. This makes finding the best epoch for early-stopping hard (Prechelt, 1998).

ferent weights initialisation methods. Similar to previous experiments, each hidden layer have 100 units. Constant initialisation to 0 for the biases; initialisation using uniform distribution with mean 0 and variance of $1/n_{in}$ is used for weight initialisation according to Klambauer et al. 2017

4.1. Momentum Learning Rule

For deep neural network experiments, the optimised gradient descent algorithm - **momentum learning rule**, was used instead of the “vanilla" **stochastic learning rule**, because it was observed that the error value was fluctuating drastically as the number of hidden layers increases (Figure 4). This imply that while we are trying to minimise the error, the SGD algorithm was moving around the local minima, but did not manage to descent gracefully. In other words, SGD was oscillating across this steep ravine, going up steep slopes only to be guided down by its gradient, and repeat. In contrast, the momentum learning rule dampens this drastic change by adding a fraction of the previous update vector to the next, preventing the gradient descent algorithm from freely moving based on its current gradient vector (LeCun et al., 2012). Hence, for the remaining experiments, SGD with momentum was used.

4.2. Part 2A: Increasing Hidden Layers

Hypothesis: Klambauer et al. showed that the beauty of a self-normalising network[1] is that it converges faster and able to handle large number of layers. Hence, we should expect that too as compared to ReLU and Sigmoid.

Results: Fixing all other parameters but the number of hidden layers yields engaging results. As number of layers increase for SNN, the test accuracies of the network increases before dropping.

[1]Self-normalising network uses SELU activation unit with weight initialisation from normal distribution of mean 0 and variance $1/n_{in}$

Borrowing the findings from the mammalian visual cortex in Malik & Perona 1990, each additional layer of the cortex is an abstraction of the previous layer. Too many layers, provides too much abstraction such that the model's performance degrade. Another view of the model with larger number of layers with non-linearities activation function is the network eventually become very sparse with multiple “dead" nodes (Glorot et al., 2011).

Comparing multi-layer networks with SELU layers against those with ReLU or Sigmoid layers, (Figure 5), it is evident that Sigmoid layers suffers from the vanishing gradient with increasing number hidden layers. In the experiment, at layers 7 and 8, the sigmoid models failed to converge.

Our experiment also confirm the claim that SELU layers do not have problems converging with larger number of layers (Klambauer et al., 2017). Unlike the noisy high variance baseline ReLU models with larger number of hidden layers, the variance for SELU models is smaller, and have lower validation error.

While increasing from the number of hidden layers from 2 to 8, there are no significant improvement in accuracy results (Table 3). Rather, the network seems to perform marginally worse. This observation is similar to the ReLU baseline model (Table 4).

From the experiments in this section, it is conclusive that SELU have some good property for convergence.

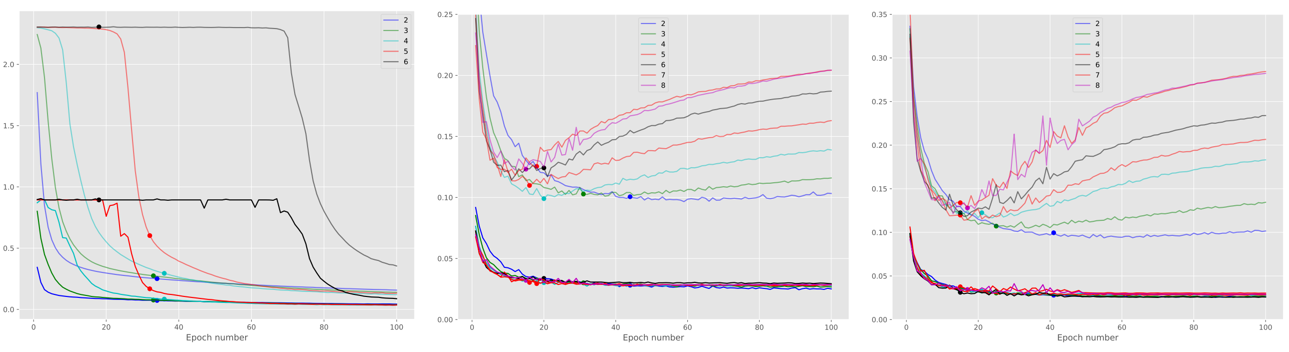

Figure 5. The performance of SELU activation function (centre) compared to Sigmoid activation function(left) and ReLU activation function (right) as the number of hidden layers increases. Graphs with lighter tone illustrate the validation error; while those with darker tone illustrate (1-validation accuracy). The corresponding scatters on the graph indicates the best-epoch that is determined by the finding a local minima of the validation error, while having the validation accuracy close to the final validation accuracy.

#Layers	error(valid)	acc(valid)	error(test)	acc(test)
2	1.006e-01	9.719e-01	9.607e-02	9.709e-01
3	1.028e-01	9.699e-01	9.434e-02	9.729e-01
4	9.903e-02	9.698e-01	1.016e-01	9.682e-01
5	1.098e-01	9.696e-01	1.109e-01	9.670e-01
6	1.242e-01	9.663e-01	1.247e-01	9.656e-01
7	1.255e-01	9.706e-01	1.318e-01	9.659e-01
8	1.232e-01	9.679e-01	1.209e-01	9.652e-01

Table 3. Performance of multilayer network with **SELU** activation function varying number of hidden layers from 2 to 8.

#Layers	error(valid)	acc(valid)	error(test)	acc(test)
2	9.950e-02	9.721e-01	9.693e-02	9.688e-01
3	1.071e-01	9.696e-01	1.085e-01	9.686e-01
4	1.225e-01	9.675e-01	1.218e-01	9.651e-01
5	1.197e-01	9.650e-01	1.182e-01	9.639e-01
6	1.224e-01	9.688e-01	1.196e-01	9.652e-01
7	1.340e-01	9.624e-01	1.418e-01	9.574e-01
8	1.280e-01	9.655e-01	1.273e-01	9.636e-01

Table 4. Performance of models with **ReLU** activation function varying number of hidden layers from 2 to 8. No significant changes in test accuracies.

4.3. Part 2B: Initialisation Methods

The next experiment carried out aims to investigate the different initialisation methods and its impact on the performance of the network.

Experimental setup: The experiment varies the initialisation methods:

FanIn	Constraint the estimated variance to be independent of the number of incoming connections: $w_i \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}})$
FanOut	Constraint the estimated variance to be independent of the number of outgoing connections: $w_i \sim U(-\sqrt{3/n_{out}}, \sqrt{3/n_{out}})$
Glorot	Constraint the estimated variance to be independent of both the number of incoming and outgoing connections: $w_i \sim U(-\sqrt{6/(n_{in}+n_{out})}, \sqrt{6/(n_{in}+n_{out})})$
SELUInit	Weights drawn from a normal distribution: $w_i \sim N(0, 1/\sqrt{n_{in}})$

As our experiment from Section 4.2 uses the SELUInit above, this set of experiments will include varying the initialisation of weights using the first three constraints listed above. In addition to varying the weight, I ran the experiment for each weight initialisation methods with different number of hidden layers (4, 6, and 8 layers). As above, momentum learning rule is used instead of stochastic gradient descent, and SELU layers are used instead.

Results: There is no clear trend as to if the different weight initialisation methods changes the performance of the network. It may seems that at a fix number of SELU activation layer, the weight initialisation have lesser effect, than the network architecture. Nevertheless, analysis of the validation error and accuracy (Figure 6), do show that the Fan In initialisation method do reduce the perturbation experienced by a deeper network, as compared to other initialisation methods. This may hint that initialisation methods invariance to the number of incoming connections may help a deeper network converge faster.

Comparing the differences between the performance of the network for a particular initialisation method while varying the number of layers yields no concrete patterns as well (Table 5).

This set of experiments aim to answer the question of: if the weights initialisation methods have significant impact on the performance (test accuracy) of the models. Analysis was carried out on twofold: one, varying the weight initialisation methods while maintaining the number of hidden layers; two, changing the number of hidden layers for each weight initialisation method. There was no clear trend or pattern as to whether the network performance is affected by the weights initialisation scheme.

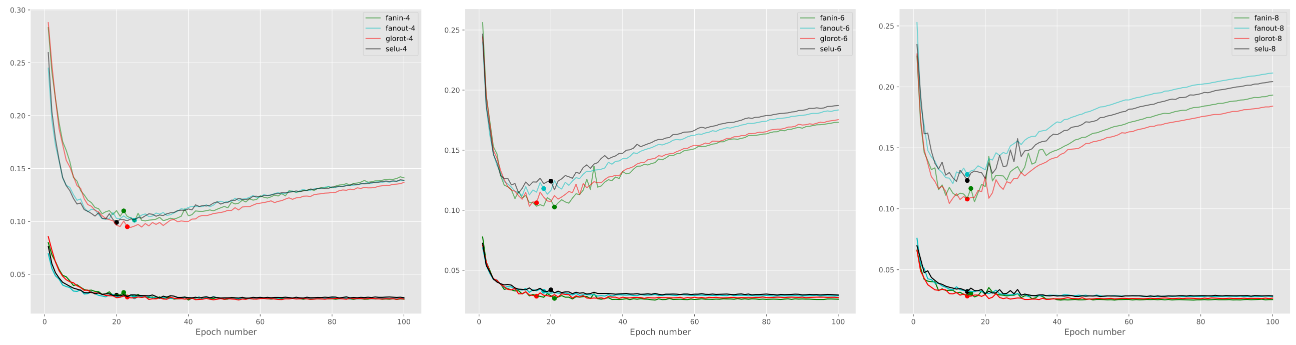

Figure 6. We experiment each initialisation methods (FanIn, FanOut, GlorotInit, and SELUInit) with three different number of hidden layers each. Left: 4 hidden layers; center: 6 hidden layers; right: 8 hidden layers.

init	error(valid)	acc(valid)	error(test)	acc(test)
Fan In Initialisation				
4	1.100e-01	9.672e-01	1.041e-01	9.693e-01
6	1.026e-01	9.734e-01	1.008e-01	9.730e-01
8	1.167e-01	9.693e-01	1.135e-01	9.685e-01
Fan Out Initialisation				
4	1.011e-01	9.713e-01	1.008e-01	9.710e-01
6	1.180e-01	9.675e-01	1.186e-01	9.670e-01
8	1.282e-01	9.678e-01	1.294e-01	9.637e-01
Glorot Benigo Initialisation				
4	9.494e-02	9.715e-01	9.125e-02	9.719e-01
6	1.062e-01	9.716e-01	1.036e-01	9.676e-01
8	1.079e-01	9.716e-01	1.122e-01	9.669e-01
SELU initialisation				
4	9.903e-02	9.698e-01	1.016e-01	9.682e-01
6	1.242e-01	9.663e-01	1.247e-01	9.656e-01
8	1.232e-01	9.679e-01	1.209e-01	9.652e-01

Table 5. The experiment with different methods for weight initialisation shows no clear trend as to where the initialisation methods leads to better network performance in terms of accuracy, or if the network architecture is the main contributor.

5. Conclusions

In this report, we have analysed the effect of different activation functions (LReLU, ELU, and SELU), compared against baseline activation functions (Sigmoid and ReLU); the SELU activation function and its performance with various weights initialisation methods and number hidden layers. The SELU activation function was dubbed as a self-normalising neural network in Klambauer et al. 2017, and have various good property for deep learning. We have seen that for the MNIST task, the SELU activation function achieve good results, but was not as staring as the RELU activation function (Section 3).

5.1. Imperfections and Risks

The effect of different weight initialisation methods had impacted various decision that I made since the start of the experiments. In specific, comparing activation functions was affected by the type of weight initialisation used in order to be fair to all activation functions, will all else constant. While using normal weight initialisation may not be the ideal way to prevent the any activation function from achieving a better result, it was the best compromised that I had found.

Another imperfection was the use of a self-improvised automated way to the find the best epochs as mentioned in Section 1.2. This method may not have output the best validation accuracy or test accuracy if the validation accuracies observed are rather perturbed with increasing epoch. This motivated the use of Momentum Learning Rule instead of “vanilla" SGD for our deeper networks experiments (Section 4). It was a risk that I had taken to use SGD with momentum, instead of finding a better learning rate, which had impact on convergence for shallower networks (LeCun et al., 2012). These risks and imperfections may have skewed the results we observed in this paper.

5.2. Future Research

More work could be done in the following aspects of the experiments:

1. While SELU have promising performance in the experiments above, Its claims in Klambauer et al. 2017 was not entirely exhibited. In particular, the performance claim due to the self-normalising nature of the network was not compared to another network that was “self-normalising". In other words, future experiments could be carried out to compare SELU with ReLU (or LReLU) with batch-normalisation. Such claim was also seen in Mishkin et al. 2016.

2. The experiments ran could also be carried out with a better algorithm for early-stopping. Instead of naively finding the lowest validation error when the validation accuracy is within a certain range from the final validation accuracy (this was used in our experiments), we could calculate the generalisation error and find the best epoch if the generalisation error increase consecutively for some s epochs. A possible problem with this algorithm is that validation error for deeper network, as seen in in the figures in this report, experience some perturbations after reaching the global minimum.

References

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994.

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.

Dahl, George E, Sainath, Tara N, and Hinton, Geoffrey E. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8609–8613. IEEE, 2013.

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.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

Goodfellow, Ian, Lee, Honglak, Le, Quoc V, Saxe, Andrew, and Ng, Andrew Y. Measuring invariances in deep networks. In *Advances in neural information processing systems*, pp. 646–654, 2009.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

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.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

LeCun, Yann A, Bottou, Léon, Orr, Genevieve B, and Müller, Klaus-Robert. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Maas, Andrew L., Hannun, Awni Y., and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. 2013.

Malik, Jitendra and Perona, Pietro. Preattentive texture discrimination with early vision mechanisms. *J. Opt. Soc. Am. A*, 7(5):923–932, May 1990. doi: 10.1364/JOSAA.7.000923. URL http://josaa.osa.org/abstract.cfm?URI=josaa-7-5-923.

Mishkin, Dmytro and Matas, Jiri. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Mishkin, Dmytro, Sergievskiy, Nikolay, and Matas, Jiri. Systematic evaluation of cnn advances on the imagenet. *arXiv preprint arXiv:1606.02228*, 2016.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Prechelt, Lutz. *Early Stopping - But When?*, pp. 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_3. URL https://doi.org/10.1007/3-540-49430-8_3.

Xu, Bing, Wang, Naiyan, Chen, Tianqi, and Li, Mu. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.