MLP Coursework 1: Activation Functions

s1546937

Abstract

Activation functions help Artificial Neural Networks to learn and make sense of the non-linearity between the inputs and outputs. Deep neural networks are difficult to train and making a better learning model is to play around with different hyper parameters, tweaking them and bringing a right fit among them.

This report deals with some of the most commonly used hyper parameters, and the selection process based on the learning model you want to build based on the understanding and adjustments of the hyper parameters. It discusses the layers of depth needed to train the model well and making it robust.

1. Introduction

An Artificial Neural Network (A-NN) is a computational model based on the structure and function of biological neural network. Activation Functions are important for A-NN to learn and make sense of complicated and non-linear functional mapping between the inputs and response parameters.

In Artificial Neural Networks, we do the sum of products of inputs (X) and their corresponding weights (W) and apply a Activation Function $f(x)$ to it to get the output of that layer and feed it as the input to the next layer. This stack of layers lead to deep models.

So considering the significance of depth, a question arises: Is stacking more layers means better learning? A hurdle to answer this question was the vanishing gradient problem(Glorot & Bengio, 2010), which makes it hard to learn and tune the parameters. This problem depends on the choice of the activation function, in which some common activation functions 'squash' their input into a very small output range in a very non-linear fashion.

This report consists of two sections. In the first section we will be seeing different activation functions and then comparing them on the MNIST data set. In the second section, we will be experimenting the deep neural networks on different sets of layers and initialization strategies.

2. Activation functions

Activation functions basically convert an input signal of a node in an Neural Network to an output signal, which intern is used as input in the next layer in the stack.

In this section we will be looking at four activation functions: Rectified Linear Unit (ReLU), Leaky ReLU, Exponential Linear Unit (ELU), and Scaled Exponential Linear Units (SELU).

ReLU (Nair & Hinton, 2010)

Instead of sigmoids, most recent deep learning networks use rectified linear units (ReLUs) for the hidden layers. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input.

$$\text{relu}(x) = \max(0, x), \tag{1}$$

which has the gradient:

$$\frac{d}{dx}\text{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{2}$$

Leaky Relu (Maas et al., 2013) Leaky ReLUs allow a small, non-zero gradient when the unit is not active.

$$\text{lrelu}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases}, \tag{3}$$

which has the gradient:

$$\frac{d}{dx}\text{lrelu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{4}$$

Where α is a constant; typically $\alpha = 0.01$.

ELU (Clevert et al., 2015) Exponential linear units try to make the mean activation closer to zero which speeds up learning. It has been shown that ELUs can obtain higher classification accuracy than ReLUs.

$$\text{elu}(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases}, \tag{5}$$

which has the gradient:

$$\frac{d}{dx}\text{elu}(x) = \begin{cases} \alpha(\exp(x)) & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{6}$$

Where α is a constant; typically $\alpha = 1$.

SELU (Klambauer et al., 2017) Scaled Exponential linear uses custom weight initialization technique. A-NNs initialize weights with zero mean and use standard deviation of the squared root of $1/(sizeofinput)$.

Hyper Parameters	Value
learning rate	0.1
number of epocs	100
number of hidden layers	2
hidden dimension	100
non-linearity	ReLU, Leaky ReLU, ELU, SELU
batch size	100
weight initialiser	Glorot Uniform Initialiser
bias initialiser	Constant Initialiser

Table 1. Hyper Parameters used in training the model.

$$\mathrm{selu}(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \le 0 \\ x & \text{if } x > 0. \end{cases}, \tag{7}$$

which has the gradient:

$$\frac{d}{dx}\mathrm{selu}(x) = \lambda \begin{cases} \alpha(\exp(x)) & \text{if } x \le 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{8}$$

Where α and λ is a constant; typically $\alpha = 1.6733$ and $\lambda = 1.0507$.

3. Experimental comparison of activation functions

We did a comparison of different activation functions - ReLU, Leaky ReLU, ELU and SELU over a set of parameters and hyper parameters to check the performance and how they given over a certain set of hyper parameters as given in Table 1.

The A-NN with ReLU, Leaky ReLU ($\alpha = 0.01$), ELU ($\alpha = 1$) and SELU ($\alpha = 1.6733$ and $\lambda = 1.0507$) were trained on the MNIST digit classification dataset using 2 hidden layers with 100 hidden units per layer for these experiments. For each we compared the learning curves (error vs epoch) for validation, and the validation set accuracies.

Here are the Figures 1, 2 showing the validation error and validation accuracy, comparing among activation functions.

In this section, we compared different activation functions - sigmoid, ReLU, Leaky ReLU, ELU and SELU. The sigmoid and ReLU performed bad on training and validation sets over 100 epocs. Leaky ReLU, ELU and SELU were monotonic and shown to generalise better in some cases. These were smooth functions with monotonic derivatives.

The average validation error and accuracy for Leaky ReLU, ELU and SELU are as Table 2

As per the validation accuracy and error, ELU seems to possess a slightly better behaviour over Leaky ReLU and SELU. The ELU network achieved lowest validation error and highest validation accuracy over the average of 100 epocs at learning rate 0.1. Moreover, the ELU actiation function fluctuates less and has a smooth learning curve.

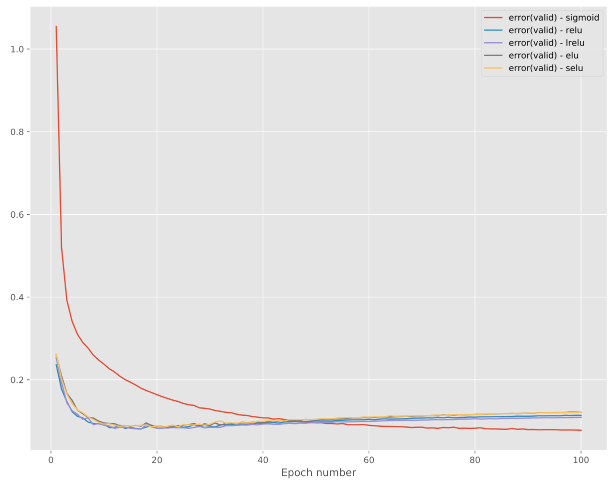

Figure 1. Validation Error over 100 epocs using 2 hidden layer with 100 units per layer, of different activation functions.

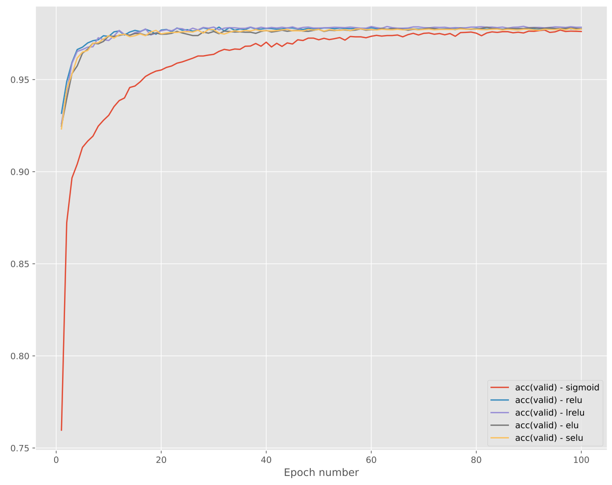

Figure 2. Validation Accuracy over 100 epocs using 2 hidden layer with 100 units per layer, of different activation functions.

Both the Accuracy and Error performance over the validation set was better for ELU over other activation functions.

The variation of median is more in ReLU networks as compared to ELU. Leaky ReLU and SELU also seems to perform better than ReLU, as ReLU ontinuously try to correct the bias shift which is introduced by previous updates of weights.

After comparing the MNIST dataset using different activation functions, ELU seems to be the better choice, as it being Robust and perform better using the 2 hidden layer.

In the next section we will be expanding our work ELU activation function using multi-layer networks and different weight initialization schemes.

Activation Function	Validation Accuracy	Validation Error
Leaky ReLU	96.7	4.6
ELU	97.5	3.9
SELU	97.2	4.2

Table 2. Validation accuracy and error .

Hyper Parameters	Value
learning rate	0.1
number of epocs	100
number of hidden layers	2
hidden dimension	100
non-linearity	ELU
batch size	100
weight initialiser	Fan-in, Fan-out, Fan-in-Fan-out (Glorot Uniform Initialiser), Gaussian Initialiser
bias initialiser	Constant Initialiser

Table 3. Hyper Parameters used in training the model.

4. Deep neural network experiments

In this section we extended our experiments on deeper networks for MNIST using ELU activation function. The two sets of experiments we will be exploring are the impact of the depth of the network (number of hidden layers), and a comparison of different approaches to weight initialisation.

I have performed experiments on layer 2 to layer 8 on different initialization weights. The hyper parameters used for these experiments are mentioned in Figure 3.

The Figure 3 and Figure 4 shows the accuracy and error rate over multiple layers over 100 epocs.

As you can see in Figure 4, deeper networks lead to higher training error. To understand the counter intuitiveness, consider the following argument.

Consider a network having m layers and this network produces some training error. Now consider a network with n layers such that $n > m$. When we train this network we expect this deep network to perform at least as well as the shallow network. As the part of deep network n, is the shallow network, thus our deeper model can easily learn the shallow network. But we just saw above in the Figure 4, deeper layer leads to higher error.

The difference in accuracy as seen in Figure 3, is minimal among layers. So considering the huge gap in the error at different layers, I would consider using the model with layer 2 and extending the experiment with different weight initialiser.

Weight Initialisation

The weights need to be initialised carefully to break the symmetry between hidden hidden units of the same layer.

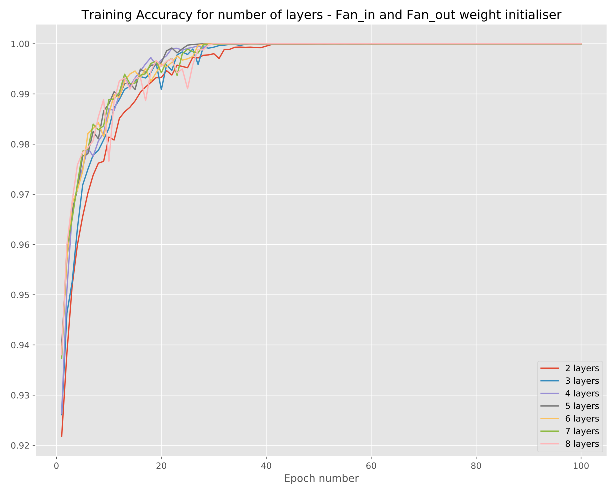

Figure 3. Training Accuracy to Number of Layers - using Fan-In-Fan-Out weight initialiser.

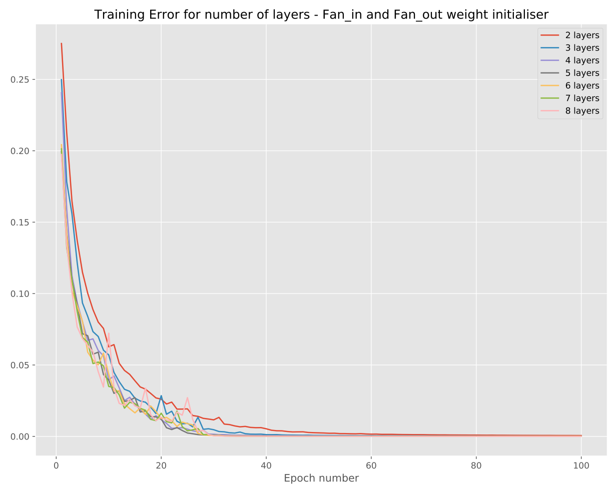

Figure 4. Training Error to Number of Layers - using Fan-In-Fan-Out weight initialiser.

We need to pick the weights at random following a distribution which helps the optimisation process to converge to the meaningful solution. We will be looking now at the effect of weight initialiser on 2-layer model. The different type of weight initialisers we will be reviewing are:

- Fan-in: $w_i \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}})$
- Fan-out: $w_i \sim U(-\sqrt{3/n_{out}}, \sqrt{3/n_{out}})$
- Fan-in-Fan-out: $w_i \sim U(-\sqrt{6/n_{in}+n_{out}}, \sqrt{6/n_{in}+n_{out}})$
- Gaussian: $\mu = 0, \sigma^2 = 1/n_{in}$

I have performed experiments with layer 2 network on different initialization weights as mentioned above. The Figure 5 and Figure 6 shows the error rate for multiple weight initialiser over 100 epocs.

Table 4 shows the validation value for fan-in, fan-out, fan-

Weight Initialiser	Validation Error	Validation Accuracy
Fan-in	0.114	0.979
Fan-out	0.123	0.978
Fan-in-Fan-out	0.117	0.978
Gaussian	0.115	0.977

Table 4. Validation accuracy and error - different weight initialiser
.

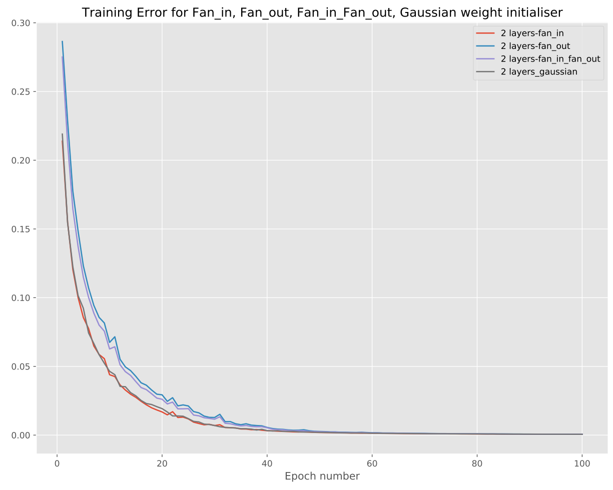

Figure 5. Training Error for 2 Layer network - using different weight initialiser.

in-fan-out and Gaussian weight initialiser.

It is clear from the result obtained from the graph and table that fan-in performs better compare to the other weight initialiser for the layer 2 A-NN model. Fan-in helps inn better optimization process and converge to the meaningful solution.

5. Conclusions

So far we have discussed about the various activation functions and their comparison (section 2, 3) , and about the deep layer networks and how changing hyper parameters like number of hidden layers and the weight initialiser (section 4), have on the model.

We use of ELU activation function, as they are Robust and helps with faster and more precise learning (3).

Deep layers does not means a better learning models, as the layers are too deep, errors are hard to propagate back correctly. If layers are shallow, the model may not learn enough. Also, weight initialiser need to be selected such that they help with the optimization process and converge to the meaningful solutions.

So there is no fixed set of rules in deciding the hyper parameters. It is the combination of these which results in better learning models.

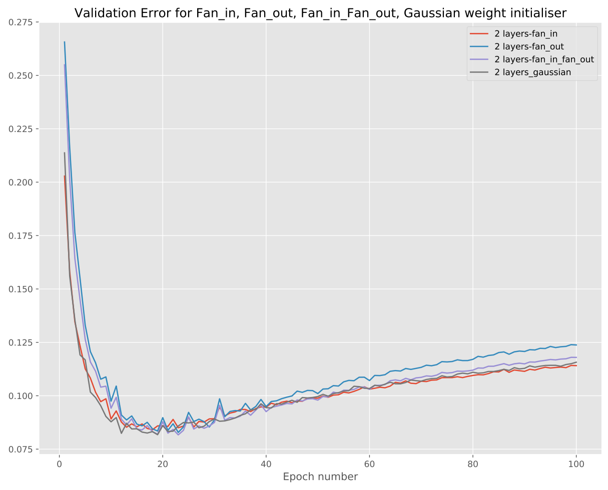

Figure 6. Validation Error for 2 Layer network - using different weight initialiser.

References

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv e-prints*, November 2015.

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, 2010.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-Normalizing Neural Networks. *ArXiv e-prints*, June 2017.

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

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, 2010.