MLP Coursework 1: Activation Functions

s1794066

Abstract

This report compares the effects of using four different activation functions in the hidden layers of a neural network. As a baseline, we used the same architecture, but with a sigmoidal activation function. We also investigate the effects of network depth, and weights initialisation strategies. Evaluated by validation set accuracy on the MNIST dataset, we find Leaky ReLU to perform the best of the activation functions. We also find that using 7 layers and a Glorot weights initialisation scheme gives us the highest validation set accuracy.

1. Introduction

Neural networks have been used extensively for classification problems in machine learning. They can be evaluated for an input $\mathbf{x} \in \mathrm{R}^{Dx1}$ for $l = 1, \ldots, L+1$

$$\begin{aligned}\mathbf{h}^{(1)} &= W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\\ \mathbf{a}^{(1)} &= \mathbf{g}^{(1)}(\mathbf{h}^{(1)})\\ \mathbf{h}^{(l)} &= W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\\ \mathbf{a}^{(l)} &= \mathbf{g}^{(l)}(\mathbf{h}^{(l)})\\ \mathbf{f}(\mathbf{x}) &= \mathbf{a}^{(L+1)}\end{aligned}$$

where L is the number of hidden layers in the network. If $K^{(l)}$ is the dimensionality of hidden layer l, then $W^{(l)} \in \mathrm{R}^{K^{(l)}xK^{(l-1)}}$ and $b \in \mathrm{R}^{K^{(l)}}$. In supervised learning, there is a training dataset with training examples $\mathbf{x}^{(m)}$ and target outputs $\mathbf{y}^{(m)}$. An associated error function E which depends on $\mathbf{x}^{(m)}$, $\mathbf{y}^{(m)}$ and the weights matrices $W^{(l)}$ is used to evaluate predictions of the network, and subsequently train the weights.

The choice of function $\mathbf{g}^{(l)}$ is an important one. First of all, it must be non-linear so that we can learn non-linear functions. The gradient of the non-linear function $\mathbf{g}^{(l)}$ is used in the learning procedure by multiplying with "error signals" calculated from layer $l + 1$. Therefore, it is also important that the gradient of our activation function is sufficiently large enough to allow learning to occur.

This report looks to evaluate the performance of four different activation functions. We will evaluate the performance of the trained models on validation set accuracy. Additionally, we will be looking at the effects of network depth and weights initialisation strategies on this same metric.

Our networks are trained and evaluated on the MNIST dataset of handwritten digits. Each data point is a 28 x 28 image, presented as a 784-dimensional vector, with an associated integer denoting the digit. We split the dataset into a training dataset (with 50000 examples) and a validation set (with 10000 examples).

2. Activation functions

The four activation functions we consider are:

1. Rectified Linear Units (ReLU)
2. Leaky ReLU
3. Exponential Linear Units (ELU)
4. Scaled Exponential Linear Units (SELU)

ReLU:

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

with gradient:

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

Leaky ReLU:

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

with gradient:

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

We set $\alpha = 0.01$, but this can also be considered a parameter. In this case, we would optimise for α which is called Parametric ReLU.

ELU:

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

with gradient:

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

We set $\alpha = 1$, so that we have a smooth function.

SELU:

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

with gradient:

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

(Klambauer et al., 2017) provides values of α and λ such that the activations of each layer converge towards zero mean and unit activation. This results in us setting $\lambda = 1.6733$ and $\alpha = 1.0507$.

3. Experimental comparison of activation functions

In order to investigate the effects of the activation function in hidden layers, we had to fix the rest of the network architecture. We used a network with two hidden layers. The hidden layers have dimension 100. The networks were trained using a batch size of 50, and with 100 epochs (passes through the training data). The weights we used were from the fully trained model, i.e. early-stopping was not implemented. We use the sigmoid function for hidden layer activations as a baseline to compare our other models with. The output layer, as is the case with all the networks used in this report, uses a SoftMax layer for the output activation.

First we look at the learning curves. Figure 1 compares how the error changes as training goes on for each of the different activation functions.

We can do the same with the validation set and see how validation error changes as our model is trained for longer, shown in figure 2.

Figure 3 compares the different models with regards to accuracy of predictions on the validation set, which is summarised in table 1.

Immediately, from figure 1, we can see that the network with sigmoid layers learns much slower than any other network. This is due to the fact that the sigmoid function saturates for extreme inputs, and so we have the problem of vanishing gradients. This results in smaller weight updates, and as a result, slower learning. The other networks seem to learn at a similar rate to each other.

Looking at figure 2, we see that the sigmoid network has the lowest validation error (at end of training) but table 1 shows that it has the worst classification rate. We hypothesise that the other networks may be predicting classes more confidently than the sigmoid network. This means that when there is a misclassification in these models, the error is higher. From this table, we also see that using Leaky ReLU provides the best performance (marginally).

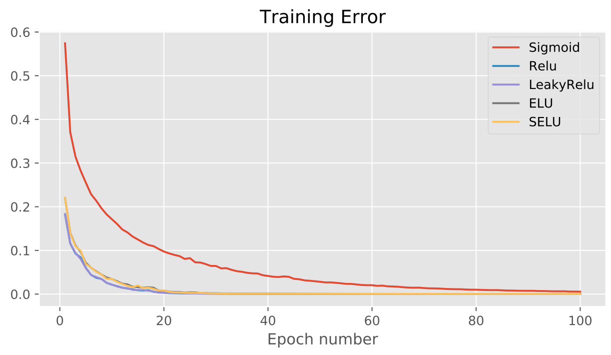

Figure 1. Comparison of using different activation functions on training error

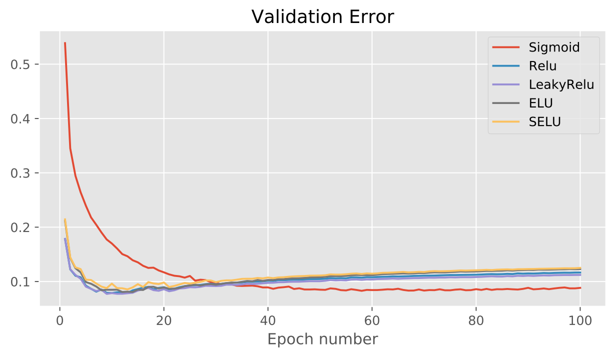

Figure 2. Comparison of using different activation functions on validation error

4. Deep neural network experiments

4.1. Changing network depth

In the next part of our investigation, we look at the effects of network depth on validation accuracy.

Using the results from the previous section, we decided to use Leaky ReLU as our hidden layer activation function. For $L \in \{2, 3, 4, 5, 6, 7, 8\}$, we trained a network with L hidden layers using a batch size of 50 for 100 epochs, like before. Again, early-stopping was not implemented so we use the fully-trained network in our evaluations.

The graphs in this section (and the next subsection) are presented in the same order as in the previous section: training error (or learning curve), followed by validation error, and finally validation accuracy. Validation accuracy is summarised in table 2.

We see from figure 4 that changing network depth doesn't seem to impact how quickly the different networks learn; all networks seem to have fully fit to the training data by epoch 40.

Activation function	Validation Accuracy
Sigmoid	97.7
ReLU	98.0
Leaky ReLU	98.0
ELU	97.9
SELU	97.9

Table 1. Validation accuracies using different activation functions

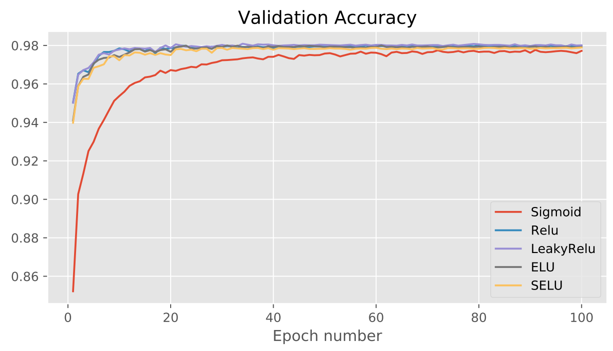

Figure 3. Comparison of using different activation functions on validation accuracy

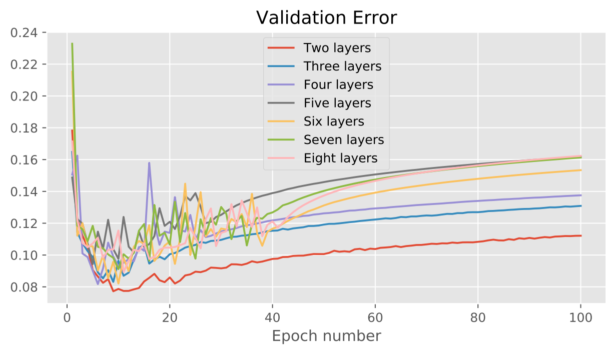

Figure 5. Comparison of network depth on validation error

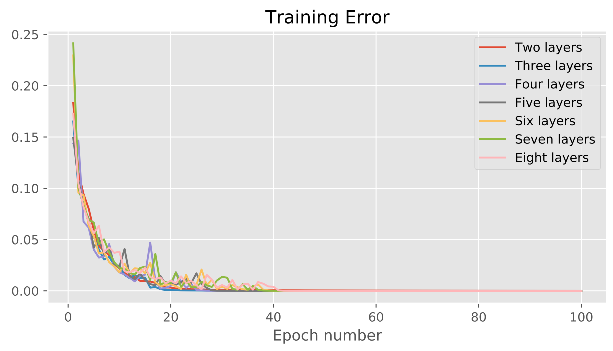

Figure 4. Comparison of network depth on training error

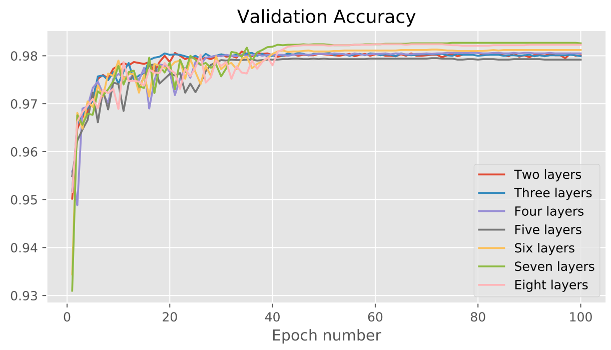

Figure 6. Comparison of network depth on validation accuracy

Figure 5 shows the two layer model having the lowest validation error, but this doesn't lead to good performance, as seen in table 2. Again, we hypothesise that this model is making weaker predictions on the whole than the deeper networks. From this table, we see that the seven layer network has the highest validation accuracy. We suspect that the eight layer network overfits to the training data so much so as to cause the most probable class for certain examples to change. This would cause a lower validation accuracy than the seven layer model.

4.2. Changing weights initialisation

In this subsection, we explore the effects of changing the weights initialisation on validation accuracy. Again, using the results from the previous section, we decided to use a network with seven hidden layers, using the Leaky ReLU activation function for hidden layers.

We explored three different weights initialisations: FanIn, FanOut and Glorot. FanIn sets uniformly-distributed weights so that the variance of the weights for a layer is independent of the number of incoming connections. FanOut sets uniformly-distributed weights so that the variance of the weights for a layer is independent of the number of outgoing connections. Glorot initialisation makes a compromise between the two.

Table 3 shows the Glorot initialisation providing the best performance. We suspect this is because this initialisation keeps weights in a more suitable range so that the network has the chance to learn in more depth.

Number of hidden layers	Validation Accuracy
Two	98.0
Three	98.0
Four	98.0
Five	97.9
Six	98.1
Seven	98.3
Eight	98.2

Table 2. Validation accuracies with different network depths

Weights initialisation	Validation Accuracy
FanIn	98.0
FanOut	98.2
Glorot	98.3

Table 3. Validation accuracies with different weights initialisations

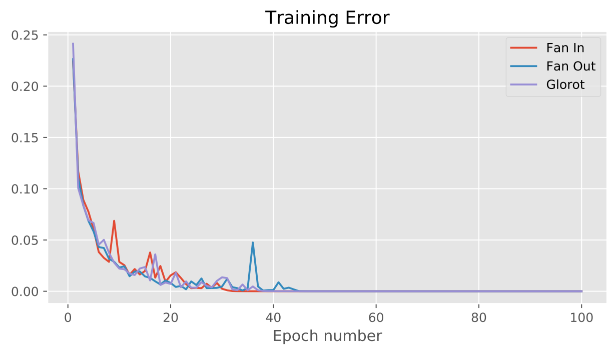

Figure 7. Comparison of different weights initialisations on training error

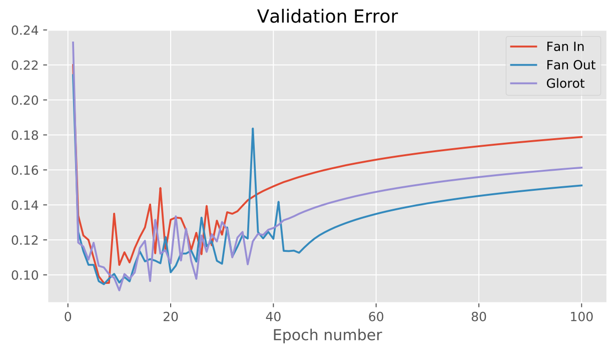

Figure 8. Comparison of different weights initialisations on validation error

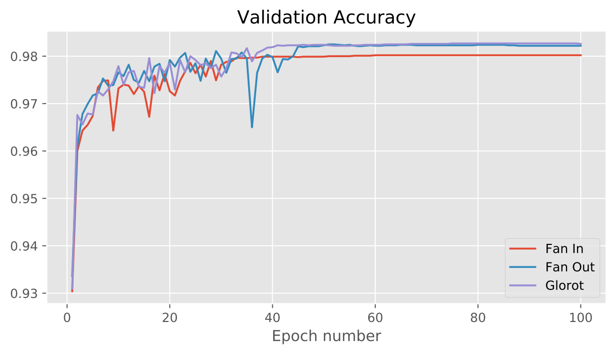

Figure 9. Comparison of different weights initialisations on validation accuracy

5. Conclusions

From our experiments, we conclude that in a network with two hidden layers, using the Leaky ReLU activation function provides the highest validation accuracy. This provided better results than the baseline (sigmoid) due to there no longer being a vanishing gradient. Using this activation function, but varying depth (i.e. number of hidden layers) showed us that using seven hidden layers generalised the best. Changing the weights initialisation seemed to have a small impact on validation accuracy: with FanIn performing worse than FanOut and Glorot.

Further work could look at whether similar results hold for different data sets (apart from network depth, which we hypothesise is largely dependent on the task at hand). Additionally, we could explore self-normalising networks (Klambauer et al., 2017) and investigate how network depth affects learning in this case.

References

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.