,2
same
MLP Coursework 1 ()

MLP Coursework 1: Activation Functions

s1449640

Abstract

Analyse the training of multi-layer neural networks to address the MNIST digit classification problem. More specifically, comparing different activation functions, including Sigmoid, ReLU, Leaky ReLU, ELU and SELU, by looking at their effects on the error function and the accuracy on the train and validation sets. Furthermore, analysing the behaviour of a ELU-based network with a varying number of hidden layers (between 2 and 8) to determine whether a deeper or a shallower network is suitable for this task. It is clear that very deep networks suffer heavily from overfitting the data, and that there is little difference between ReLU and it's variants when using the same fixed hyperparameters. Finally, different methods of initialising the weight parameters for the hidden layers are also taken into account when measuring performance.

1 Introduction

The task of recognising handwritten symbols has historically been considered to be difficult or impossible to achieve without first developing Artificial General Intelligence the$_e$*conomist*$_2$016.*Throughouttheyears*, *manyalgorithms*, *albeitf armorespecialisedthaninitiallythought*, *havesucc*

1.1 Neural Networks

An (artificial) neural network (ANN) is a network of simple elements called neurons, which are loosely based on the biological neurons found in animals. These neurons receive an input which changes their state according to a weight (and bias) ascribed to it, run it through an activation function (see 2) and finally produce an output. This network then forms a directed graph where the neurons are nodes and the edges are the weighted connection between them. At the end of this, the network propagates the final outputs back into the network, through a process called backpropagation and *"learns"* (by adjusting the weights of the edges) whether it has achieved a correct or incorrect result based on an annotated training dataset. rÃd'dle2010neuronale

1.2 MNIST Dataset

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples lecun$_1$998.*Inthiscoursework*, *Ihaveonlyusethef ir*
scale, *whichisthenf ormattedintoa*784*dimensionalrowvector*, *usedasaninpu*

1.3 Default testing environment

All tests were carried out using the default setting provided in the official GitHub repository: each hidden layer has 100 neurons while the output layer has 10. The learning rate was set to 0.1 as my own brief testing showed little to no improvements in changing this value by comparing it on the baseline Sigmoid activation function (see 2). Furthermore both the batch size and the number of epochs (for the Batch Gradient Descent learning algorithm) were set to 100.

1.4 Activation Functions

Using different activation functions can have large effects on the performance of a Neural Network. It is claimed that modern variations of Rectified Linear Units (ReLUs) can speed up the training of networks significantly elu selu, as opposed to a traditional implementation of ReLU

or, the traditionally used, Sigmoid functions. In this coursework, five different activation functions were analysed: Sigmoid, ReLU, Leaky Relu, ELU (Exponential Linear Unit), SELU (Scaled Exponential Linear Unit).

1.5 Deep Neural Networks

When using a neural network for a supervised classification task, it is usually claimed that shallow networks are easier to train, but have far worse accuracy performance on large datasets compared to deeper networks, which can be very slow to train, but achieve lower error values and better accuracy. I decided to experiment on a network based on the Exponential Linear Unit (ELU) activation function for the hidden layers, because it claims to be able to achieve quicker and better results in deeper networks elu. A comparison of 2, 5 and 8 layers is shown here, although I did train all networks between 2 and 8 layers, to get a sense of which were most interesting to compare.

1.6 Weight Initialisation

Weight initialisation can seem like a small detail to care about when first diving into the subject of NNs, but it can actually have a profound effect on performance intoli. In the case that weights are naively initialised to 0, the network might now work at all! Here we will explore different initialisation methods on ELU- and SELU-based networks, and see if they affect performance in any noticeable way.

2 Activation functions and their experimental comparison

This section will present all of the 5 activation functions tested and will contrast some of their features.

2.1 Sigmoid

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

which has the gradient:

$$\frac{d}{dx}\,\text{sigmoid}(x) = \text{sigmoid}(x) - (1 - \text{sigmoid}(x)) \tag{2}$$

The sigmoid function, together with the Rectified Linear Unit (ReLU) (see 2.2), were considered baselines for the 2-hidden layer neural networks that had to be analysed in Part2A of the coursework. It managed to achieve the lowest score of all the tested activation functions, being the only function never to achieve a 0 error score, nor perfect accuracy on the training set.

2.2 ReLU

$$\text{relu}(x) = \max(0, x), \tag{3}$$

which has the gradient:

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

In ReLU the activation is simply thresholded at 0. Compared to the sigmoid function, it doesn't involve calculating (computationally) expensive exponentiation of the input vectors, thus it is generally considered to be a faster and almost always better choice than the former. ReLU is the other baseline function to which the next 3 activation functions are compared to, these being slight variations on it.

2.3 Leaky ReLU

$$\text{lrelu}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases} \quad (5)$$

which has the gradient:

$$\frac{d}{dx}\text{lrelu}(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x > 0. \end{cases} \quad (6)$$

Leaky ReLU tries to correct the *"dying neuron"* problem by applying a very small value ($\alpha = 0.01$ is used here) to the slope when $x < 0$. A *"dead"* neuron is one that always outputs the same value regardless of input, and this is usually arrived at by having a negative bias value. Once a neuron is *"dead"*, the slope is equal to 0, so the gradient will never readjust it's weight value.

2.4 ELU

$$\text{elu}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases} \quad (7)$$

which has the gradient:

$$\frac{d}{dx}\text{elu}(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0. \end{cases} \quad (8)$$

The ELU Activation tries to speed up learning in deep neural networks and lead to higher accuracy rates for classification, by alleviating the vanishing gradient problem elu, which is a problem when multiplying small gradient values during backpropagation, leading the very deep networks being very slow to update the weight values of the first few layers. In this coursework, a value of $\alpha = 1$ was used, resulting in a smooth function.

2.5 SELU

$$\text{selu}(x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases} \quad (9)$$

which has the gradient:

$$\frac{d}{dx}\text{selu}(x) = \begin{cases} \lambda\alpha e^x & \text{if } x < 0 \\ \lambda & \text{if } x > 0. \end{cases} \quad (10)$$

SELU induces self-normalising properties meaning that it tries to keep the mean value close to 0 and keeps a unit variance throught the network. This is claimed to speed up the process of training very deep networks, and experimental data seems to confirm this claim selu. In my implementation, I have used the recommended value of $\alpha = 1.6733$ and $\lambda = 1.0507$.

2.6 Experimental Comparison of Activation Functions

We can clearly see that the Sigmoid activation function performed worst, by far, having never reached an error function value of 0 on either datasets, and a peak accuracy of 95% on the validation set. ReLU and it's variants achieved very similar performance in this 2-layer NN, but it's also obvious that they were all overtraining on the data, because of reaching a value of 0 error on the train data, which then led to an increase of the error function on the validation set. They all managed to reach 100% accuracy rate on the training set, and approximately 98% on validation, surprisingly with ReLU having the most consistent and highest score of all.

3 Deep neural network experiments

The data for the 2-, 5-, and 8-layer ELU-based NNs is presented here.

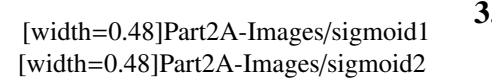

Figure 1: Sigmoid activation function

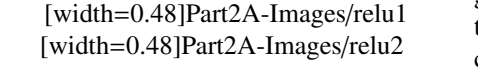

Figure 2: ReLU activation function

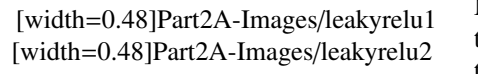

Figure 3: Leaky ReLU activation function

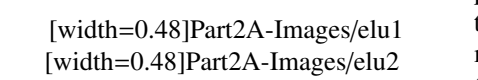

Figure 4: ELU activation function

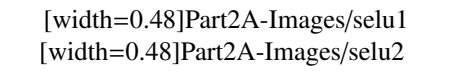

Figure 5: SELU activation function

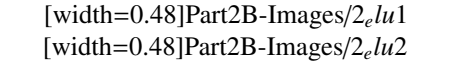

Figure 6: 2-layer ELU Neural Network

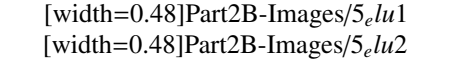

Figure 7: 5-layer ELU Neural Network

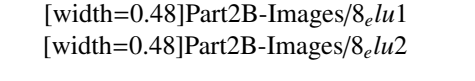

Figure 8: 8-layer ELU Neural Network

3.1 Analysis of the number of layers

We can see that the 8-layer NN reached the highest performance, as expected. However, there was significant noise in the beginning of the training process, as opposed to the shallower networks. Since the accuracy of the deepest network barely improved by less than 1%, and the training time was significantly higher, even when using the purportedly faster ELU-based NN, we can safely conclude that for the task of recognising handwritten digits from the MNIST database it is appropriate to use a shallower, ideally 2-layer, Neural Network. This result is consistent with the pattern seen in previous experiments on the MNIST website, where recent developments have been using shallow Convolutional Neural Networks lecun$_1$998.

3.2 Fan-In Initialisation

If the initial weights are too small, then by the time a forward propagation pass goes through the network, a useful weight might not be seen as important fast enough. Similarly, if a weights starts out massive, by the time it reaches the output layer, it might be too large to useful. Instead of using a simple uniform initialisation scheme, it's better to use ones with better mean and variance. andyljones$_2$015

In the case of fan-in, we care about the number of connection to a neuron, and use that to calculate the variance of the distribution from which we draw the initial numbers.andyljones$_2$015

3.3 Fan-In and Fan-Out Initialisation

Similarly to the previous example, here we care about both the number of incoming connections and the outgoing ones.

3.4 Gaussian Initialisation on SELU

Gaussian distribution is particularly well suited to SELU, since it is a self-normalising activation function, and the gaussian can be set to have it's variance and mean close to the mean at which it normalises. Thus, this initialisation scheme, has by far the best performance in the accuracy of both sets, but it seems to have an issue when starting up. However, once it does, it takes only a few passes in the network to reach convergence on a local minimum, with an accuracy of over 98%.

3.5 Analysis of initialisation schemes

By far the most performant one seems to be the Gaussian Initialisation using SELU. This however has a weird and unpredictable behaviour in the beginning, and achieves great performance once it does get started. Both fan-in and the fan-in and fan-out seem to achieve similar performance, and do not offer much over the usual Uniform Distribution-based initialisation scheme, used in 2.

4 Conclusions

Overall, there seems to be a strong correlation between using the more advanced, ReLU-based Neural Networks as opposed to a Sigmoid-based one, but there are very small differences between them. These achieve very high performance and low error rates. Furthermore, for this task, it doesn't seem to be beneficial to use more than 2 hidden layers when training the data, because of the issue of overfitting. Finally, initialisation schemes seem to have a bigger influence than I expected, it being the determining factor of the performance of my SELU-based network, achieving an almost perfect accuracy score.

Further improvements to these networks could be easily made by analysing when to stop the gradient descent algorithms, since all ReLU-based NNs started overtraining on the data approximately when reaching epoch no. 20. Additionally, determining a way to better initialise the learning rate could also help in creating faster, more robust networks.

References

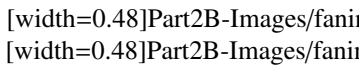

Figure 10: Fan-In and Fan-Out Initialisation

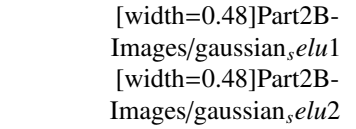

Figure 11: Gaussian intialisation on SELU