MLP Coursework 1: Activation Functions

s1307311

Abstract

In this report, I investigate the ability of various deep neural networks to classify images from the Balanced EMNIST dataset. The main comparison involves evaluating models which use variations of the ReLU activation function, as well evaluating a linear model; with a model using the ELU activation function giving the best performance. An investigation is also carried out to determine whether hyperparameters for various activation functions can be set effectively using limited training and validation data, with both a general trend in classification accuracy at different hyperparameter settings, as well as the same best hyperparameter setting for the best performing model identified when the model is evaluated on both the full and limited datasets.

1. Introduction

This report details work carried out which explored classification of images using various neural network architectures. The report has 5 main sections (with Section 1 serving as the introduction you are now reading).
Section 2 describes the experiments conduction to establish baseline scores of various network architecutres on the Balanced EMNIST dataset.
The Balanced EMNIST dataset (Cohen et al., 2017) was used throughout this project, as an alternative to the MNIST dataset (LeCun et al., 1998) which is used as a baseline in many classification tasks. The MNIST dataset is itself derived from an earlier dataset, the NIST dataset (Grother, 1995). The NIST dataset contains handwritten letters and numbers, and was created to enable development of handwritten character recognition. The dataset is very large, and reported by some as too difficult to use (Cohen et al., 2017). The MNIST dataset was therefore generated to provide as a simpler to use baseline for classification systems: the dataset consists only of the digits 0-9, with the images gone through some processing - such as rescaling and anti-aliasing. The EMNIST dataset - which is a collection of various datasets, each providing a different classification task - was also derived from the original NIST dataset, and was developed to provide a more challenging classification task than the MNIST dataset. The EMNIST dataset produced the images in the same manner as the MNIST dataset, but additionally includes handwritten letters. The Balanced EMNIST dataset was created by merging characters for which there is ambiguity in their uppercase and lowercase versions - e.g. 'C', 'W', and 'X' - resulting in a final dataset with 47 classes. Using the provided data-providers class in the mlp package, I generated training, validation, and test sets. For each experiment I ran, I reset the datasets to allow for a direct comparison of results. Every experiment in this project was run multiple times to reduce the variance in the observed results. Where an experiment was repeated N times, the seed used to generate the training, validation, and test datasets were $1..N$. The details of this experiment are also presented in Section 2.
Section 3 describes the work carried out in comparing activation functions. It describes the experiments that were carried out, as well as the results from those experiments. There is also a description of each of the activation functions which were compared, with special interest as to how those activation functions are related to each other, as well as how they differ, and the motivation for the use of different activation functions. In addition to this, Section 3 describes experiments which were conducted in order to answer the question of whether or not a limited training and validation set can be used to effectively set the hyperparameters of a neural network. The motivation for this question comes from the fact that deep neural networks can have a large number of trainable hyperparameters, resulting in the monumental task of finding the best hyperparamter settings which can distract from the original purpose of the research. One approach to setting hyperparameters is a grid-search over all possible hyperparameter values, however, with a large number of training points and a large network, this can take a long time. Some report that hyperparameters need not be trained using all available data, and instead an estimate towards the setting of these parameters can be learned from minimal data (Nielsen, 2015). The search for the best model architecture via comparison of models with different activation functions, as well as comparison of models with the same activation function but different hyperparameter settings, provides an optimal setting to investigate this question.
Section 4 describes the work carried in comparing Linear vs. Non-Linear functions.
Finally, the conclusion in Section 5 draws the results of the experiments together, highlighting the network configuration which gave the best test set accuracy, and details some avenues for future work which could be carried out to improve the performance of the models considered, as well as other architectures which could be used to achieve higher classification architectures on the EMNIST dataset.

# Units	1 layer	2 layers	3 layers
32	0.780 (0.00358)	0.791 (0.00133)	0.796 (0.00182)
64	0.807 (0.00274)	0.816 (0.00276)	0.818 (0.00154)
128	0.824 (0.00159)	0.831 (0.00127)	0.832 (0.00219)

Table 1. Mean test set accuracy for various deep network architectures across 10 samples (SD in brackets).

# Samples	(Limited) Validation set accuracy
5	0.584 (0.0445)
10	0.571 (0.0423)
25	0.573 (0.0533)

Table 2. Mean validation set accuracy of ELU model for various sample sizes (SD in brackets).

2. Baseline systems

2.1. Establishing a baseline

In order to establish a baseline, I examined models with 1, 2, and 3 hidden layers, each of which had 32, 64, and 128 hidden units per layer, resulting in 9 models in total. For optimisation I used I used Adam (Kingma & Ba, 2014), rather than vanilla Stochastic Gradient Descent (SGD), as the original researchers which produced the method report that using Adam for a Logistic Regression task on the MNIST dataset gave a higher test set accuracy than simple SGD (Kingma & Ba, 2014). I used the default learning rate of 0.001 since this is the value used in the original paper. Each model was run for 100 epochs, at the end of which the model setting with the highest accuracy on the validation set was selected to evaluate on the test set. The experiment was repeated 10 times (a discussion as to how this sample size was chosen is detailed in Section 2.2), with different seeds used for any random elements, in order to gather the mean test set accuracy for each architecture, along with the standard deviation. The results of this experiment are summarised in Table 1.

Choosing any single column (keeping the number of hidden layers constant) in Table 1 and reading down each row (increasing the number of units per hidden layer), we can see that for every single model, increasing the number of hidden units for layer leads to an increase in the mean test set accuracy.
Similarly, if we choose any single row (keeping the number of units per hidden layer constant) in Table 1 and move across the columns (increasing the number of hidden layers), we can see that increasing the number of hidden layers also leads to an increase in the mean test set accuracy.
By increasing the complexity of the network - either by adding additional layers or increasing the number of hidden units per layer - the model is able to learn more complex functions, and since we are dealing with a high-dimensional problem with a large number of classes, it is clear that the model benefits from this increased complexity. Of course, by increasing the complexity we are more likely to generate a model which overfits on the training data. If we were to greatly increase the number of training epochs and choose the model with the highest training set accuracy we would likely produce a model which has overfit and is not able to generalise. By selecting the model which gives the best validation set accuracy however we are aiming to reduce the likelihood of overfitting and produce a model capable of generalisation.
Moving forward from here to experiments involving altering the activation function used, the recommended model architecture was 3 hidden layers with 128 units per layer, unless a better baseline architecture was found. Since this recommended architecture is also the same one that gave the best mean test set accuracy across 10 samples, it was the one I used to compare activation functions.

2.2. Choosing a sample size

In order to decide on the number of times to repeat each experiment, I chose a model at random - the ELU model with 1 hidden layer and 32 hidden units - and trained it in on the training data for 10 epochs - only a small number of training epochs were used in order to save on time, and since I was only interested in finding the number of samples which would give me a reliable result, not the best result, I felt that this was an acceptable modification to make. I trained on only a small sample of the data - 100 batches with 100 data points per batch - in order to reduce runtime again (a further discussion of how the number of batches were chosen is found in Section 3.2). I then chose the sample size which gave a good tradeoff between runtime and reliability of results (measured by standard deviation in validation set accuracy of a small sample - 10 batches, 100 data points per batch - of the validation set). The results are summarised in Table 2.

The mean validation set accuracies were all very similar, with 10 samples giving the lowest standard deviation. This itself may have been a side effect of the seeds used, since I would expect more samples giving a lower SD which isn't seen with 25 samples. Having said that, since all of the means and standard deviations were very similar, I decided to use 10 samples since I felt it gave a good tradeoff between runtime and reliability of results.

3. Activation Function Comparisons

In this section, I will begin with a description and comparison of the 4 activation functions considered. I will then describe the steps taken to find the best hyperparameter settings for each activation function. Next, I will report the final test set accuracy for each model, with each model having the hyperparameter which gave the best validation set accuracy found in the previous experiments. Finally, I will describe the experiments which were conducted to answer the question of whether hyperparameters can be learned effectively with limited training data.

3.1. Activation functions

I conducted the following experiments with 4 different activation functions: Leaky ReLU (LReLU), Parametric ReLU (PReLU), Random ReLU (RReLU), and the Exponential Linear Unit (ELU). All of these activation functions are extensions of the Rectified Linear Unit (ReLU) function which is defined by Equation 1:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Activation functions are used to introduce non-linearity into the outputs of units within a network, as this has been repeatedly found to increase the performance of those networks and enable them to become universal approximaters (Hornik et al., 1989). The ReLU function is a commonly used activation function as it has been found to outperform other functions such as sigmoid or hyperbolic tangent functions (Glorot et al., 2011).

The first extension to the ReLU function I will describe is the Leaky ReLU; the equation for this activation function is shown in Equation 2:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2}$$

As can be seen by comparing Equations 1 and 2, the LReLU function introduces a parameter α which scales the input x when $x \leq 0$. The purpose of this parameter is to avoid 0 gradients, which can lead to slower learning.
Next, we'll consider the Parametric ReLU function. The equation for the PReLU function is the same as the LReLU function, with the key difference between the activation functions being how the value of α is set: when using LReLU, α is determined and set before the model is run, while with PReLu, the value of α is learned during training via backpropagation. Learning the value of α, rather than setting it explicitly, has been shown to improve model fitting at no extra computational cost (He et al., 2015).
The Random ReLU function, once again, has the same equation as both LReLU and PReLu, with the difference being both the structure of α, as well as way that the value of α is set: with RReLU, α is a vetor of values, uniformly drawn at the beginning of every forward propagation step. The motivation for this function is that by introducing randomness, our models are less likely to overfit to training data. Empirical studies have shown that RReLU outperforms all aforementioned activation functions (Xu et al., 2015).

Finally, the Exponential Linear Unit is defined by Equation 3.

$$f(x) = \begin{cases} x & \text{if } x >= 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{3}$$

The authors of the original paper detailing the ELU describe the motivation for ELUs best: "*ELUs saturate to a negative value with smaller inputs and thereby decrease the forward propagated variation and information*" (Clevert et al., 2015). This saturation has proven advantageous, as the authors report that by using ELUs, their models were able to learn better than models using all previously mentioned activation functions, as well as decreasing the time taken to train the models.

With the activation functions lain out, I will now describe the experiments conducted in order to find settings of hyperparamters for these models, as well as comparing models using these activation functions.

3.2. Finding the best hyperparameter settings

As mentioned at the end of Section 2, I used a constant architecture of 3 layers and 128 hidden units per layer for all of my experiments. In order to compare the activation functions to find the model setting which gave the highest overall test set accuracy, I wanted to ensure that each model type (where by "model type" I mean models with the same architectures described above, but with different activation functions used for the hidden layers) was given an opportunity to achieve the best possible test set accuracy. In order to ensure this, I experimented with changing the hyperparameter for each model, choosing the best hyperparameter setting based on the validation set accuracy.
Encouraged by the method detailed by Nielsen 2015 (Section 3.5) to find hyperparameters, I decided to find the general magnitude of the hyperparameters which gave the highest validation set accuracy, choosing to find the general magnitude over specific values since this is what the aforementioned method allows for. Since all models involve a single parameter α (and a vector of α values in the case of RReLU, all set to the same initial value of α), I tested each model with the following α values: $[0.001, 0.01, 0.1, 1.0, 10.0]$.

Similar to when I established the baselines, I also ran each experiment 10 times and took the mean of the best validation set accuracies. Since I was drawing 10 samples per model, across 4 models, and with 5 α values per experiment, in order to complete the experiments within the time frame of the assignment I decided to reduce the number of training samples used. Since I was only comparing models with different hyperparameter settings and attempting to identify any signal to be found when varying the hyperparameters of a model, rather than searching for the hyperparameters which gave the best overall model performance, I believed this was a reasonable step to take, and wouldn't reduce the reliability of the results by too much compared to an experiment which would use all available data. I believed the idea to reduce the number of training samples during hyperparameter search was also justified by Nielsen who reports when setting hyperparameters for models trained

# (Training, Validation) batches	Validation set accuracy
(10, 1)	0.568 (0.0513)
(50, 5)	0.701 (0.0230)
(100, 10)	0.773 (0.0113)
(1000, 100)	0.843 (0.00383)
(All, All)	0.845 (0.00183)

Table 3. Mean validation set accuracy of ELU model for dataset sizes (SD in brackets).

on MNIST data: "... *instead of using the full 10,000 image validation set to monitor performance, we can get a much faster estimate using just 100 validation images. All that matters is that the network sees enough images to do real learning, and to get a pretty good rough estimate of performance*" (Nielsen, 2015).
In order to further ensure that altering the size of the training data wouldn't reduce the reliability of the results by a large degree, I performed the following experiment to determine the size of the training set to use, similar to the one carried out when determining the number of samples which should be taken to increase reliability: I used the same the same model as in the experiment to determine the number of samples (ELU with 1 hidden layer and 32 units, run for 10 epochs) with varying number of training set and validation set batches (with 100 data points per batch). The results are summarised in Table 3.

The conclusion I drew from this experiment was that 100 batches of training data, and 10 batches of validation data, gave a good tradeoff between runtime and approximation to the true validation set accuracy: the standard deviation of the results at those settings, I believe, is reasonably low, indicating stable results, and each epoch took on average $5s$ to compute, compared to $50s$ per epoch for both 1000 training batches, and using all available training data. Once again, the motivation behind this experiment was to identify a signal resulting from changing the magnitude of hyperparameters across different models, rather than finding the best performing model, and I believe that 100 batches of training data and 10 batches of validation data give reasonably reliable results for identifying such a signal.

With the number of samples and number of training set batches determined, I created models of the described architecture, and tested each one (running each model for 100 epochs) with the 5 values of α described above. The results of this experiment are shown in Table 4, with the best hyperparameter setting for each model identified in bold.

3.3. Comparing test set accuracy

With the best hyperparameters for each model identified, I created 4 models - one for each activation function compared thus far - each with the recommended architecture of 3 hidden layers and 128 hidden units per layer, and each using the parameter setting identified found in Section 3.2. As I was now attempting to find the model which gave the best test set accuracy, I increased the number of epochs to 1000 to give each model enough chance to fit the data well, choosing the model settings which gave the best validation set accuracy at the end of training. Since I knew this would take much longer than the 100 epoch experiments ran so far, I implemented early stopping: the models would stop training if the validation set accuracy did not improve after 10 epochs (choosing 10 to allow small periods of decreasing performance). I also used all available training and validation data, rather than the limited sets used for hyperparameter search. I tested a single model (LReLU) run for 1000 epochs on all training data and found that it took 2 hours to train on my local machine. For that reason, I reduced the number of repetitions from 10 to 3. This allowed the experiment to be run within the time frame of the assignment, however, the results are less accurate than they would be if 10 samples - or even more - were drawn. At the end of training, I selected the model settings with the highest validation set accuracy and applied this to the test set data. The results of the experiment are shown in Table 5.

The best model is ELU, which has both the highest mean test set accuracy across the 3 samples drawn (by a very small margin), as well as the lowest deviation in final accuracy (by a slightly larger model), indicating it is also the most reliable model trained on these samples. This is consistent with the results found in (Clevert et al., 2015) claiming that ELU gives the best model performance over all other activation functions.
With ELU identified as the best model on this data, I wanted to establish whether or not this result holds up when I perform hyperparameter search for this, supposedly best, model with all available training data. This experiment is detailed in the next section.

3.4. Comparing hyperparameter search with different training set sizes

As described in the introduction, a question I was interested in answering was whether hyperparameters for neural networks could be found effectively using limited training data. In order to answer this question, I took the best model setting found so far (ELU with 3 hidden layers, 128 units per layer) and reran the hyperparameter search conducted in Section 3.2, this time using all available training and validation data, to identify whether or not the same signal was present in the training data, and whether the same best hyperparameter was found. A comparison of validation set accuracies for the ELU model at various hyperparameter settings (the log of which is shown in the figure, the absolute values are those reported in Section 3.2), evaluated on the limited training and validation data, as well as the full dataset, are shown in Figure 1

From Figure 1 we can see that the same general trend (an increase from $\alpha = 0.001$ to $\alpha = 1.0$ where the accuracy peaks, followed by a sharp decline at $\alpha = 10.0$) is found

	0.001	0.01	0.1	1.0	10.0
LReLU	0.755 (0.0398)	0.762 (0.0640)	**0.763(0.0471)**	0.643 (0.0384)	0.560 (0.0571)
PReLU	0.641 (0.05145)	0.640 (0.0387)	0.664 (0.0659)	**0.646(0.0280)**	0.641 (0.0589)
RReLU	**0.772(0.0328)**	0.722 (0.0634)	0.759 (0.0399)	0.76 (0.0341)	0.744 (0.0429)
ELU	0.833 (0.0139)	0.834 (0.00829)	0.834 (0.0117)	**0.845(0.0128)**	0.7687 (0.00945)

Table 4. Mean validation set accuracy of various models at different hyperparameter settings (SD in brackets).

Model	Test set accuracy
LReLU	0.833 (0.00190)
PReLU	0.833 (0.00331)
RReLU	0.833 (0.00204)
ELU	0.834 (0.000908)

Table 5. Mean test set all models at best hyperparameter settings (SD in brackets).

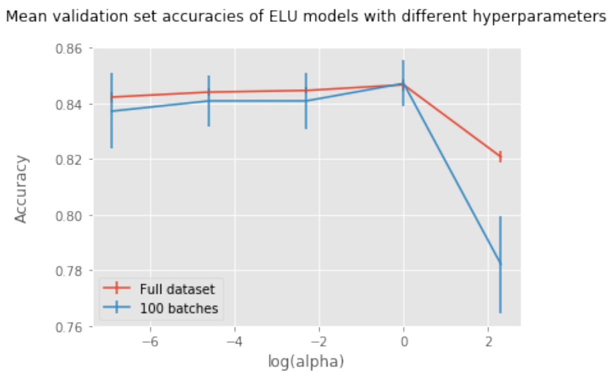

Figure 1. Mean validation set accuracies of ELU on different sizes of training and validation data.

in both datasets, with the only difference being the value of accuracy and the certainty in the results, with less data resulting in lower accuracies and more unstable results, as would be expected. This leads me to conclude that the general magnitude of hyperparameters can be reliably set with limited training data, resulting in quicker turnaround in experiments, with models capable of being trained at a fraction of the time and cost when using all available data.

4. Linear vs Non-Linear

To determine whether non-linear activation functions are necessary for fitting deep neural networks, I created a linear model with the same architecture as that used for all previous experiments (3 layers with 128 units per layer), with the hidden units being affine layers. I randomly initialised the weights of the network, trained it on all available data for 1000 epochs with early stopping after 10 epochs of non-improvement, selected the model with the highest validation set accuracy, and evaluated this on the test set. I repeated this experiment 10 times, taking the mean test set accuracy and the standard deviation in results. The resulting model had a mean test set accuracy of 0.670 and standard deviation 0.00317. This is vastly lower than the best accuracy for non-linear models (0.834 ± 0.000908), indicating that while non-linear activation functions may not be *necessary* for fitting deep neural networks, they are certainly more effective than linear functions.

5. Conclusions

Based on the experiments carried out, the setup which produced the best system on the Balanced EMNIST data was a model with 3 hidden ELU layers, each with an α value of 1.0, and 128 units per layer, trained for 1000 epochs with early-stopping after 10 epochs of non-improvement, with the best model setting selected based on validation set accuracy.
The experiment in Section 4 showed that non-linear activation functions are much more effective at classifying images from the Balanced EMNIST dataset than linear ones. Further work could be carried out here by testing different model architectures (for both the linear and non-linear models), as well as different - and specifically more informative - initialisation of the weights of the linear models, as well as testing the effect of different optimisation techniques. These comments also apply to the experiments comparing activation functions detailed in Section 3. Other techniques, such as regularisation, could also be applied to all models and their effect on model performance analysed.
A further experiment into whether or not hyperparameters can be set effectively with limited training and validation data found that, for the model tested, they can. In order to gather further evidence to test this hypothesis, it would be worthwhile repeating the experiment described in Section 3.4 on all models considered in this report, as well as testing more precise settings of hyperparameters and training and validation set sizes.

References

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

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.

Grother, Patrick J. Nist special database 19. *Handprinted forms and characters database, National Institute of Standards and Technology*, 1995.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Hornik, Kurt, Stinchcombe, Maxwell, and White, Halbert. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Nielsen, Michael A. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

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