MLP Coursework 2: Learning rules, BatchNorm, and ConvNets

s1788323

Abstract

In this study we investigate the effects of using different hidden unit activation functions (as they appear in deep learning), network architectures, training hyper-parameters, regularization techniques, and learning rules on the Extended Modified National Institute of Standards and Technology database (EMNIST) Balanced dataset. We find that the best model after considering these factors is a four hidden layer network with Exponential Linear Units (ELU) as the activation function, using a learning rate of 0.1 with a L2 penalty of magnitude 10^{-3}, while using the stochastic gradient descent method. Then we look at the effect of Batch Normalization technique on this particular model. We then proceed to compare the performances of a single layer and two layer Convolutional Neural Network (CNN) amongst themselves, and with the best performing deep network. As expected after reading the theory, we find that the two layer CNN has the best performance, and both convolutional networks outperform our best deep network (obtained from the first part of the study) in this classification task.

1. Introduction

Deep learning is an area of machine learning which holds great potential for the future of Artificial Intelligence. Before delving into the intricacies of deep learning, it is important to understand some basics. This paper focuses on experimenting with and analyzing some of these basic concepts, namely, comparison of the behaviour of different activation functions, experimental investigation of the impact of depth of a network with respect to accuracy (number of units in a hidden layer is kept constant throughout the study, at 100), effect of different learning rates, regularization techniques, and three different learning rules: Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMS Prop) (Tieleman & Hinton, 2012), and Adaptive Moment Estimation (Adam) (Kingma & Ba, 2014). After we have experimentally analyzed these factors and fixed a model, we implement the Batch Normalization technique (Ioffe & Szegedy, 2015) and observe its efficacy. We wind up our study by comparing two types of CNN's: a single layer CNN and a two layer CNN.

We perform all our experiments on the EMNIST Balanced data set (Cohen et al., 2017). This is a collection of images of handwritten digits and letters (both lower case and upper case), used in machine learning tasks as an alternate to the MNIST dataset. Since EMNIST Balanced has many more classes (47, compared to 10 in MNIST), the classification task is more challenging, and hence the classification accuracies achieved are lower than on the MNIST. The training set has about 94,000 images, with another 18,800 images in the validation set. The test set also has 18,800 images. For our experiments we use a batch of 100 images and iterate for a maximum of 100 epochs. The results presented in the tables use early stopping (Prechelt, 1998), but it was not implemented during the experiments which produced the graphs presented, because then the graphs would not have been of uniform lengths, making the presentation unattractive.

The next section describes the experiments performed on our baseline systems. The subsections discuss each subsequent step. We begin by discussing about the various activation functions, and results of experiments with them on a 2 hidden layer network. Then we move on to experimenting with depth of the network, different values of the learning rate, and use of regularization in our model. Section 3 discusses two learning rules: RMS Prop and Adam, and compares their performance with SGD on our model. In section 4 we look at Batch Normalization, and then incorporate it into our model. Section 5 has a brief description of Convolutional networks, followed by a comparison of performances of a single convolution layer and two convolution layer network. Then we use the networks we judged to be the best on the test set, and report their performance in Section 6.

2. Baseline systems

2.1. Activation functions used

We use the following activation functions in our experiments. Mathematically, they are defined as follows:

Sigmoid function: this function constrains the values to be between 0 and 1

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

On simplifying, we find its gradient is:

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

Rectified Linear Unit (ReLU): this function overcomes the vanishing gradient problem that occurs with the sigmoid function, especially in deep networks.

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

which has the gradient:

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

Leaky ReLU - this function is a modification of ReLU which fixes the "dying ReLU" problem by scaling the negative values by a small number (α).

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

(For our experiments, we use $\alpha = 0.01$)
Its gradients are:

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

Exponential Linear Unit (ELU) - this is another modification of ReLU, which gives a better performance than leaky ReLU in certain tasks as it exponentiates the negative inputs followed by scaling.

$$\text{elu}(x) = \begin{cases} \alpha(exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{7}$$

(For our experiments, we use $\alpha = 1$)
Its gradients are:

$$\frac{d}{dx}\text{elu}(x) = \begin{cases} \alpha * exp(x) & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{8}$$

Scaled Exponential Linear Unit (SELU) - this is a modification of ELU which scales the function in both intervals (x > 0 and x <= 0)

$$\text{selu}(x) = \lambda \begin{cases} \alpha(exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{9}$$

(For our experiments, we use $\alpha = 1.6733$ and $\lambda = 1.0507$)
The corresponding gradients are:

$$\frac{d}{dx}\text{selu}(x) = \begin{cases} \lambda\alpha * exp(x) & \text{if } x \leq 0 \\ \lambda & \text{if } x > 0 \end{cases} \tag{10}$$

2.2. Experimental comparison of activation functions

For all the activation functions we run a 2 hidden layer network with 100 units each. The learning rate is fixed at 0.1, and we use SGD as our learning rule. The weights are initialized using Glorot uniform initialization (Glorot & Bengio, 2010). These experiments were run to determine which activation function works best on this dataset, so that we use it in all subsequent experiments. Since we are mainly interested in performance on the validation set, we present graphs of their error and accuracy on the validation set only, not the training set. This helps us compare all the activations easily at the same time.

Figure 1 compares all the activation functions together, but due to the sigmoid curve, the others aren't in a scale where it is easy to compare them accurately. Hence Figure 2 compares the activations without the sigmoid. For uniformity, the plotted curves were saved from an experiment which ran for all 100 epochs; however, for our analysis we use early stopping as well, and hence Table 1 lists the values obtained after taking that into account. Figure 3 compares the accuracy on the validation set.

From these graphs and values, we can deduce that ELU activation works best in this case, as it has the lowest validation error and highest validation accuracy. All the activations (except sigmoid) perform best around the 20^{th} epoch, after which they start over-fitting the data, and hence our training time is also not vastly different across the different activations when we use early stopping. Thus for all further experiments, we use ELU activations.

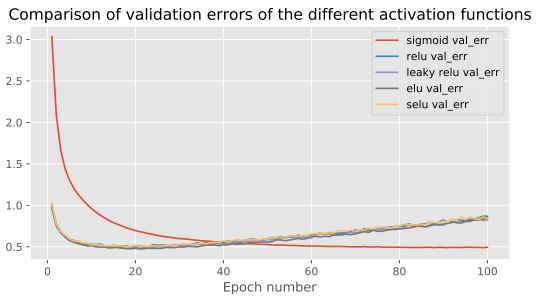

Figure 1. Comparison of errors on the validation set

Table 1. Validation errors and accuracies (using early stopping)

Activation Function	Val error	Val acc
Sigmoid	4.95 e-01	8.39 e-01
ReLU	5.07 e-01	8.42 e-01
Leaky ReLU	4.92 e-01	8.39 e-01
ELU	4.81 e-01	8.48 e-01
SELU	5.04 e-01	8.41 e-01

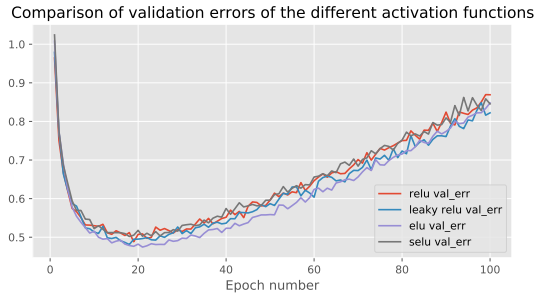

Figure 2. Comparison of errors on the validation set without sigmoid

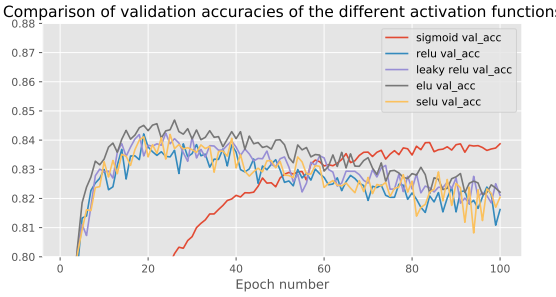

Figure 3. Comparison of accuracy on the validation set

2.3. Experimental comparison of depth of network

Now we run experiments using the same parameters specified in section 2.2, but we use only ELU activations and vary the depth of the deep network from 2 hidden layers to 8 hidden layers. This is done so as to identify which architecture works best, so that we can use it in all the other experiments. Since validation accuracy is what is most important to us, we only present this graph. Figure 4 has a zoomed in comparison of the validation accuracies across the depths (in the legends, h corresponds to the layer depth). Again, for analysis we use early stopping, and Table 2 compares the values we obtain after using early stopping.

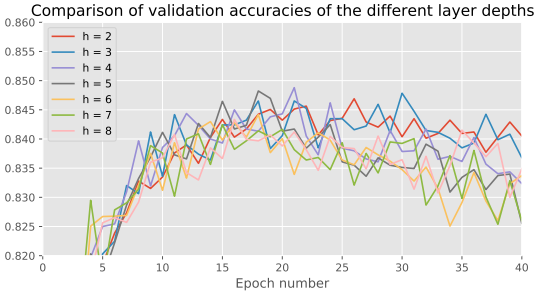

Figure 4. Comparison of accuracy on the validation set

We observe that depths 2 to 5 have very similar accuracies, and hence we run the experiments multiple times. The values presented in Table 2 were an average over 4 trials. Though the values were close each time, the architecture with 4 hidden layers always had the highest accuracy. Thus

Table 2. Validation errors and accuracies (using early stopping)

No. of hidden layers	Val err	Val acc
2	4.81 e-01	8.48 e-01
3	4.88 e-01	8.47 e-01
4	4.97 e-01	8.49 e-01
5	4.98 e-01	8.48 e-01
6	4.98 e-01	8.44 e-01
7	5.15 e-01	8.42 e-01
8	4.90 e-01	8.43 e-01

for all experiments henceforth, we use 4 hidden layer architecture in our experiments.

2.4. Experimental comparison of learning rates

Having fixed our activation function and network architecture, we now proceed to experiment with different learning rates (the other parameters are same as before). This is done to find the best setting of learning rate, which we will then use in subsequent experiments. Figure 5 is a zoomed in plot comparing the different learning rates used, and Table 3 lists the values obtained on using early stopping in our experiments.

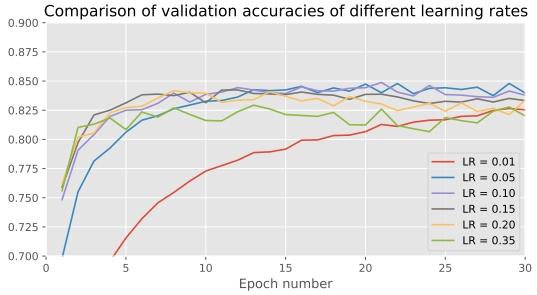

Figure 5. Comparison of accuracy on the validation set

Table 3. Validation errors and accuracies (using early stopping)

Learning rate	Val err	Val acc
0.01	4.90 e-01	8.45 e-01
0.05	4.82 e-01	8.48 e-01
0.10	4.97 e-01	8.49 e-01
0.15	4.83 e-01	8.42 e-01
0.20	4.74 e-01	8.42 e-01
0.35	5.62 e-01	8.29 e-01

Observing the graph and the table, we find that learning rates of 0.05 and 0.10 give similar results. So we run experiments for those two rates multiple times, and compute the average accuracy (the value reported in Table 3). Since a rate of 0.10 is marginally better, we fix this value for further experiments.

2.5. Experimental comparison of different regularization techniques

Continuing from the previous sections, we now add regularization to our model to improve its performance. We try various settings of L1 and L2 regularization techniques. The settings which gave best results have been summarized in Figure 6. To maintain uniformity, the plot is from experiments which were made to run for all 100 epochs, without using early stopping. It is clear that most of the models start over-fitting after the 20^{th} epoch, but the models which perform the best continue improving until the last iteration. The L2 penalty with a setting of 10^{-3} gives the best accuracy, 0.86 or 86%, compared to 84.9% without any regularization (in section 2.4). This result validates the theory that penalizing extreme weights helps reduce over-fitting, and hence improves our generalization to previously unseen data.

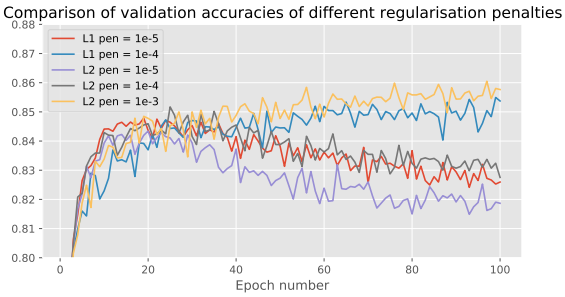

Figure 6. Comparison of accuracy on the validation set

3. Learning rules

Now we test our best model till now (section 2.5) with different learning rules to see which rule works best with our model. We used SGD until now, and we will try RMSProp and Adam now. Before we proceed, lets look at the three learning rules first.
Mathematically, we can define the weight updates of SGD as:

$$g_i(t) = \nabla_{w_i(t)} E(w)$$

$$\triangle w_i(t) = -\eta g_i(t)$$
$$w_i(t+1) = w_i(t) + \triangle w_i(t)$$

Basically we just update weights with a scaled version of the gradient of the loss function with respect to the weights, where η, the learning rate, is the scaling factor.

The weight updates for RMS Prop are:

$$S_i(0) = 0$$
$$S_i(t) = \beta S_i(t-1) + (1-\beta) g_i(t)^2$$
$$\triangle w_i(t) = -\frac{\eta}{\sqrt{S_i(t)} + \epsilon} g_i(t)$$

The value of β, the decay coefficient, is set as 0.9 and ϵ is a small value (10^{-8}) to maintain numerical stability of the weight update. RMSProp is a method in which the learning rate is adapted for each of the parameters. We divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight.

The weight updates for Adam are:

$$M_i(0) = 0 \quad , \quad S_i(0) = 0$$
$$M_i(t) = \alpha M_i(t-1) + (1-\alpha) g_i(t)$$
$$S_i(t) = \beta S_i(t-1) + (1-\beta) g_i(t)^2$$
$$\triangle w_i(t) = -\frac{\eta}{\sqrt{S_i(t)} + \epsilon} M_i(t)$$

Here, α is set as 0.9, and β is set as 0.999. Adam is basically a variant of RMSProp with momentum. In this learning rule, running averages of both the gradients and the second moments of the gradients are used.

After running a few baseline experiments, we find out that we cannot use the same learning rates as before for RMSProp and Adam. Thus we run experiments trying out a few different learning rates on both these learning rules. For these, all our settings are same as in section 2.5, except for our learning rule and learning rate. Figure 7 compares the validation accuracy of different rates using RMSProp, and Figure 8 compares the validation accuracies of different rates using Adam as the learning rule.

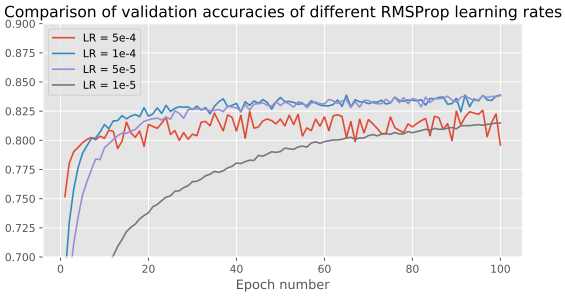

Figure 7. Comparison of accuracy on the validation set

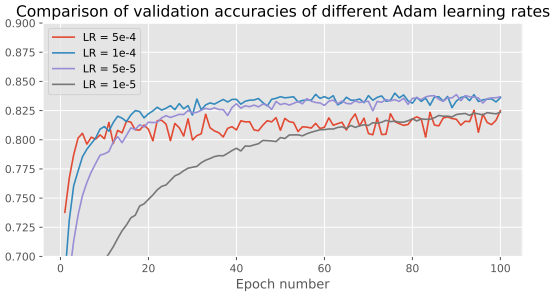

Figure 8. Comparison of accuracy on the validation set

From the graphs, we see that learning rates of 10^{-4} and $5\text{x}10^{-5}$ have similar results. Thus we average over multiple trials, and find that a learning rate of 10^{-4} has a marginally better accuracy on the validation set for both RMSProp and

Adam. Thus for our comparison with SGD, we use both rules with a learning rate of 10^{-4}. Figure 9 compares the accuracies of the three learning rules.

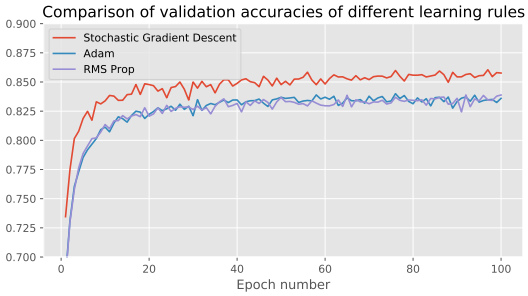

Figure 9. Comparison of accuracy on the validation set

This result tells us that with our model SGD is the best learning rule. We could perhaps do a more extensive grid search of the learning rate for both RMSProp and Adam to find a setting which works better than SGD, but we limit our search to just the values shown in Figures 7 and 8, and hence we conclude that SGD is the best learning rule for our model.

4. Batch normalisation

Batch Normalization is a technique which was developed in 2015. It solves the problems which occur due to internal co-variate shift in deep networks. The equations which govern Batch normalization are given below:

$$\mu_i = \frac{1}{M}\sum_{m=1}^{M} u_i^m \tag{11}$$

$$\sigma_i^2 = \frac{1}{M}\sum_{m=1}^{M}(u_i^m - \mu_i)^2 \tag{12}$$

$$u_i = w_i x \tag{13}$$

$$\hat{u}_i = \frac{u_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{14}$$

$$z_i = \gamma_i \hat{u}_i + \beta_i = Batchnorm(u_i)) \tag{15}$$

Here μ_i and σ_i refer to mean and covariance of the minibatch, and γ_i and β_i are learn-able parameters.

Now we incorporate Batch normalization into our model. Our best model is a 4 hidden layer ELU activation architecture, with a learning rate of 0.1 and using SGD learning rule. The weights are initialized using Glorot uniform initialization. Since Batch normalization enables faster learning, we experiment with three different learning rates to compare systems with our previous model. These experiments are summarized in Figure 10.

From this graph we infer that the system with a learning rate of 0.05 performs best on the validation set (in terms

Figure 10. Comparison of accuracy on the validation set

of accuracy). Thus we select this model to compare performance between models with and without application of batch normalization. Figure 11 shows this comparison.

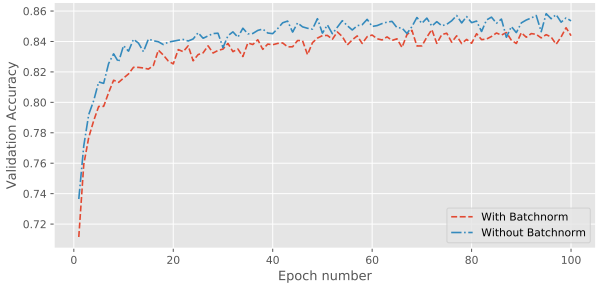

Figure 11. Comparison of accuracy on the validation set

We see a surprising result, that the system without batch normalization works better. This could be due to the fact that we did not explore the learning rates and learning rules that truly unlock the potential of the Batch normalization technique.

5. Convolutional networks

Convolutional networks (Conv nets) are deep feed forward neural networks that improve performance in image and video recognition tasks (among others). They do this by taking into account the spatial configuration of an image, rather than just using individual pixels. A conv net extracts features from the input, with the help of pooling layers. A convolution layer applies a filter (or multiple filters) to the input layer, and moves through the 2-D space to cover the entire input layer. This process corresponds to the convolution of the inputs with the weights. This layer is then subjected to pooling (usually max-pooling) to create a pooling layer, and then this may either be connected to more convolution and pooling layers, or be connected fully to an affine layer.

We implemented a convolution layer by working through the maths, and using for-loops to implement each subsequent step of the process, such as choosing the right input feature space which would be subjected to convolution with the weights. The convolution in itself was

also implemented using for loops, to better understand how it actually works, rather than use the SciPy function for convolution.

Then we experimented with two types of conv nets to find out which worked better. Both had these parameters being the same: learning rate of 0.1, L2 penalty of 10^{-3}, using SGD learning rule, and having ELU activations. For both architectures, we used convolutions of stride 1, and used a kernel size of 5 by 5. The pooling layer had non-overlapping pooling of size 2 by 2 (stride 2). The first architecture had one convolution and max-pooling layer, with 5 feature maps, while the other architecture had two convolution and max-pooling layers. The first convolution layer had 5 feature maps, and the second one had 10 feature maps.

Figure 12 compares the performance of both architectures with respect to accuracy achieved on the validation set. As expected, the two layer architecture performs better, because it can extract more features from the inputs, enabling it to make better classifications.

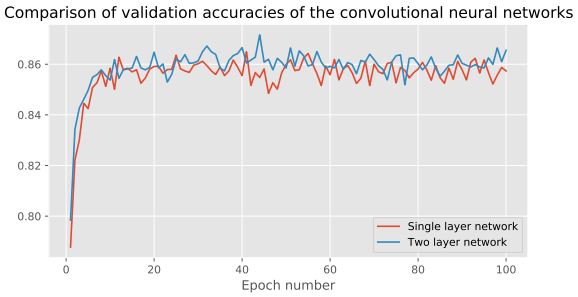

Figure 12. Comparison of accuracy on the validation set

6. Test results

After analyzing the experiments performed from sections 2 to 4, we decided on our best model, which is mentioned at the end of section 4. We arrived at the best model by choosing the model with the highest accuracy on the validation set at each stage. This model was used to evaluate the test set. Averaging over five trials, we obtained the following results:

Mean accuracy on test set: 84.82%
Standard deviation:0.1183

Picking between the two convolutional nets was more straightforward. We used the two layer convolutional net to evaluate the test set, as its validation accuracy was higher. The average over three trials on the test set resulted in these values:

Mean accuracy on test set: 86.03%
Standard deviation: 0.0924

These results demonstrate that our models generalized on the test set in line with our expectations. The accuracy on the test set was slightly lower than that on the validation set, but achieved a good performance nonetheless. Further, we see the convolutional network had a better performance than the deep network.

7. Conclusions

This study was performed to better understand how the performance of a model changed with variations in its parameters, different learning rules and techniques such as batch normalization, and how the deep networks compared to the convolutional networks. We began our study by experimenting with different activation functions. Our results led us to conclude that the ELU activations were best for this task. We then tested the ELU network for different depths of hidden layers, and found that the 4 hidden layer network worked best. This architecture was then tested with various learning rates, and we found that a learning rate of 0.1 achieved highest validation set accuracy. These settings were then tested with various regularization schemes, and our results showed that an L2 penalty of 10^{-3} produced the highest accuracies. These results helped us conclude that regularization and early stopping are powerful techniques to improve performance of the model. Preventing over-fitting, whether through regularization or using a network of appropriate depth, led to a better model. We implemented and looked at different learning rules, and concluded that SGD worked best with our model. We then implemented batch normalization, and incorporated it into our model. For the settings that were tried, the performance of the model with batch normalization did not beat the performance of the model without it. This could be put down to not looking at the best possible settings which would make batch normalization quite effective. Lastly, we experimented with convolutional networks and concluded that the two layer network performed better than the single layer network because it could extract more features from the inputs. As expected, both the convolutional networks outperformed the best deep network model, because they accounted for the spatial structure of the inputs.

This study could be improved and extended in a number of ways. For starters, more experiments can be done on the different learning rules, so that the best possible hyper-parameters are found for each learning rule. This holds true for batch normalization as well. In the literature, we find numerous examples where this technique has helped improve performance of the model, such as (Laurent et al., 2016). Further, we could experiment with other techniques such as dropout (Srivastava et al., 2014). For the convolutional networks, we did not experiment with the hyper-parameters at all, and this could be looked into in great detail. Another good extension would be to examine the effects of using batch normalization with the convolutional layers, and increasing the depth of the layer.

References

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 and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

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

Laurent, César, Pereyra, Gabriel, Brakel, Philémon, Zhang, Ying, and Bengio, Yoshua. Batch normalized recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 2657–2661. IEEE, 2016.

Prechelt, Lutz. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4): 761–767, 1998.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.