Report for Assignment 3(B) of COL 774

Ayushi Agarwal
2018ANZ8503
ayushi.agarwal@cse.iitd.ac.in

1 Introduction

The goal of this assignment is to implement Neural Networks to classify the different English alphabets. English Alphabets Dataset is an image dataset consisting of grayscale pixel values of 28×28 size images of 26 Alphabets. It is a multi-class dataset consisting of 26 classes, one for each alphabet. The training and test dataset consists of 13000 and 6500 samples, respectively. In this problem, we use one-hot encoding for the output labels. Given r classes, the output of the neural network is a vector of size r representing the probability of the data belonging to that class. The report will be sectioned based on the following implementation strategies:

- Single Hidden Layer MLP: We will experiment with a neural network with one hidden layer and variable number of hidden units as shown in Section 2.1 and Section 2.2. The activation function used is sigmoid. We will analyse the effects of fixed learning rate and adaptive learning rate (Section 2.3) on the training.
- Multiple Hidden Layers MLP: In Section 2.4, the architecture of the neural network will have 2 hidden layers with 100 units each. We present an analysis on the performance difference between ReLU and Sigmoid activation functions used for hidden layers. In Section 2.6, we use a different cost function (cross entropy) to analyze the difference in network prediction performance.
- Classifier from SKLearn: In Section 2.5, we will train several MLP classifiers with different hyperparameters and compare the results with our own implementation.

The next section of this report discusses the algorithm in each question, the method to implement the algorithm, report the results and present an analysis of the results obtained.

2 Method, Results and Analysis

In this section, we would discuss our algorithm implementation and the results obtained. We would present an analysis of the results as well[I]. The dataset used here has 26 classes from (1,26). To train a neural network to classify them, we would use one-hot encoding of the output labels so that each data sample has an output vector of size 26 and each component is $\{0,1\}$. Each output vector will only have one entry as 1 (the actual label) and the other entries as 0. The network will therefore learn 26 corresponding outputs for each data sample. In the following sections we will present an analysis on the performance of the neural network by varying different hyperparameters of the network like number of layers, number of perceptrons in one layer, activation function used (Sigmoid vs. ReLU vs. Softplus), learning rate, parameter initialization, etc.

2.1 Part A - Generic Neural Network Model for Multi-Class Classification

In this section, we implement a generic neural network or a multi-layer perceptron to learn a model for multi-class classification using one-hot encoding of the output labels. A single perceptron is similar to how a brain neuron functions. It fires according to the linear combination of the inputs into the perceptron when it becomes higher than a threshold function which is also called the activation function. In a multi-layer perceptron, several such perceptron's are repeatedly used in layered manner to learn different features from the input at different time instances. Each layer of perceptron learns to recognise different features according to the feature map the network learns during training.

The cost function used in neural networks can be of varied types ranging from mean squared error to cross-entropy. We use the mean squared error, as shown in Equation 1, to implement the cost function over the entire training set (m samples) in this part of our experimentation. In Section 2.6, we will present the analysis on the performance of the neural network using cross-entropy loss. Equation 2 represents the activation function $g(x)$. In this experiment we have used the sigmoid activation

[I] https://github.com/agarwal-ayushi/Machine-Learning-Assignments/tree/master/Assignment3/

function for all the layers in the network.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 \quad (1)$$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad (2)$$

We use mini-batch stochastic gradient descent (mini-batch SGD) to update the parameters of the network. Equation 3 shows the cost $J(\theta)$ on a mini-batch of size M. Here $y^{(i)}$ is represented using the one-hot encoded vector of the output labels. $o_l^{(i)}$ represents the output of the neural network corresponding to the label l from r such output labels.

$$J^b(\theta) = \frac{1}{2M}\sum_{i=(b-1)M}^{bM}\sum_{l=1}^{r}(y_l^{(i)} - o_l^{(i)})^2 \quad (3)$$

Equation 4 represents the back-propagation algorithm to calculate the delta for updating the network parameters. The delta of the last layer is the derivative of the cost function calculated at the current set of network parameters and current output of the neural network. The delta for the parameters of the hidden layers is calculated using chain rule by using the delta of the downstream layers.

$$\begin{aligned}\delta_L &= \frac{1}{M}(Y_b - O_l)O_l(1 - O_l)\\ \delta_l &= \delta_{l+1}\theta_l O_l(1 - O_l)\end{aligned} \quad (4)$$

After the back-propagation is done, the parameters of the network are updated by the update rule given in Equation 5. The parameters are updated in proportion to the learning rate α of the network.

$$\theta^{t+1} = \theta^t + \alpha X_l \delta_{l+1} \quad (5)$$

The implementation of the fully-connected neural network is generic for the following set of hyperparameters:

- Mini-Batch Size (M)
- Number of features (n)
- Hidden Layer Architecture - List of numbers denoting the number of hidden layer units in the corresponding hidden layer.
- Number of target classes (r)

2.2 Part B - Single Hidden Layer Neural Network Model for Multi-Class Classification

In this section, we experiment with a neural network having a single hidden layer. The neural network will have three layers: Input layer, One Hidden layer and one output layer. In the input layer, the number of perceptron units equal to the number of input features/attributes. In this dataset, we have input samples of alphabet images in the form of 28×28 pixel values. So the number of perceptrons in the input layer would be 784. We have added one extra feature in the input features to act as the intercept term $x_0 = 1$ and the corresponding network bias parameter. So we implement $\theta^T X$ instead of $w^T X + b$, where $b = \theta_0 x_0$. Similarly the intercept terms are added at each layer to act as a bias for that layer.

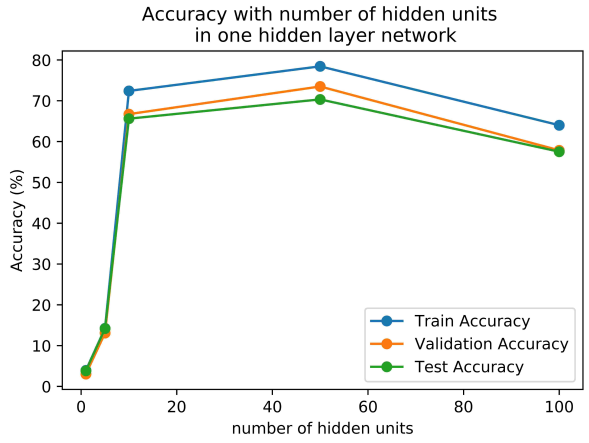
Stopping Criteria	**Theta Init**	**lr (α)**	**epochs**	**Train Acc(%)**	**Test Acc(%)**
$\Delta loss_{train} < 1e^{-7}$	(0, 1)	0.5	9321	96.77	88.58
$\Delta loss_{train} < 1e^{-6}$	(0, 1)	0.1	9457	96.44	86.41
$\Delta loss_{train} < 1e^{-6}$	(0, 0.01)	0.1	7753	97.09	88.75
$\Delta loss_{train} < 1e^{-6}$	(0, 1)	0.5	3807	96.94	85.8
$\Delta loss_{train} < 1e^{-6}$	(0, 0.01)	0.5	3042	97.52	88.35
$\Delta loss_{val} < 1e^{-7}$ (10 epochs)	(0, 1)	0.1	3884	94.91	85.56
$\Delta loss_{val} < 1e^{-7}$ (10 epochs)	(0, 0.01)	0.1	2656	96.17	88.06
$\Delta loss_{val} < 1e^{-6}$ (10 epochs)	(0, 0.01)	0.5	604	96.35	88.43
$\Delta loss_{val} < 1e^{-6}$ (10 epochs)	(0, 0.05)	0.1	2228	95.73	89.18
$\Delta loss_{val} < 1e^{-5}$ (10 epochs)	**(0, 0.05)**	**0.1**	**1385**	94.33	**87.87**
$\Delta loss_{val} < 1e^{-5}$ (10 epochs)	**(0, 0.05)**	**0.5**	**462**	95.65	**88.22**
$\Delta loss_{val} < 1e^{-5}$ (10 epochs)	(0, 1)	0.5	1034	78.42	70.32

Table 1: Accuracy of a neural network with one hidden layer having **50 perceptron units** with different types of early stopping criteria for training to avoid overfitting. The theta initialization is done using numpy random generator from normal distribution. The mean and standard deviation are shown in the table above.

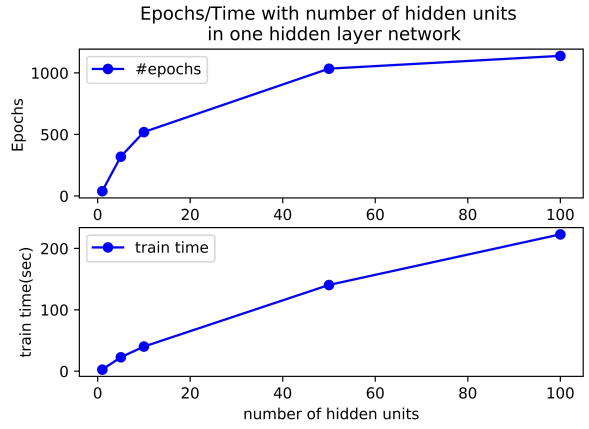
The number of perceptrons in the Output layer are equal to the number of output classes which in this case is 26. The hidden layer units would be varied from the set $\{1, 5, 10, 50, 100\}$ to analyze the performance of the corresponding neural networks. Apart from the hidden layer units, the learning rate of the network will be constant throughout the learning process ($\alpha = 0.1$). However, we

will show in the coming sections that higher learning rate $\alpha = 0.5, \alpha = 1.5$ are considerably better and converge faster with same accuracy. The mini-batch size is 100.

Table 1 shows the results of training a neural network with 50 perceptron units. This table shows a comprehensive analysis of how the network performance varies with the different early stopping criteria and learning rate. There are two possible stopping criteria based on the training loss and the validation loss. The training loss criteria tries to minimize the cost on the training dataset. However, for validation loss, we separate 15% of the training dataset as the validation dataset and minimize the loss on this validation set during training.



(a) Accuracy with different number of hidden layer units



(b) Epochs and Train Time

Figure 1: Accuracy, Epochs and Training Time for neural networks with different number of hidden layer units with Theta Initialization from normal dist. with $\mu = 0$ and $\sigma = 1$ and using $\alpha = 0.5$

Observations on the Stopping Criteria, Learning Rate and Theta Initialization :

We observe from the table that **the best stopping criteria** is when the **change in validation loss is less than** $1e^{-5}$ **over 10 continuous epochs**. We achieve similar test accuracy's using this criteria with less number of epochs. The table also points out that the use of training loss as stopping criteria might lead to overfitting and also take more epochs to meet that stopping criteria. We have observed that with the Stopping Criteria based on the Training Loss minimization, the network achieves the optimum accuracy within 1000 epochs but keeps running to minimize the training loss further leading to overfitting. The loss degradation becomes very slow close to the minima and so the test accuracy does not change very significantly.

The table also shows that the learning rate of $\alpha = 0.5$ is better than $\alpha = 0.1$ achieving higher accuracy in less number of epochs.

The table also shows that the **effective theta initialization** for the number of hidden units {50,100} is from normal distribution with $\mu = 0$ and $\sigma = 0.05$. However, as shown in Figure 2, this theta initialization is not effective for lesser number of hidden units which are {1, 5, 10} as these networks achieve random accuracy with this initialization.

Observations on Accuracy and Train Time:

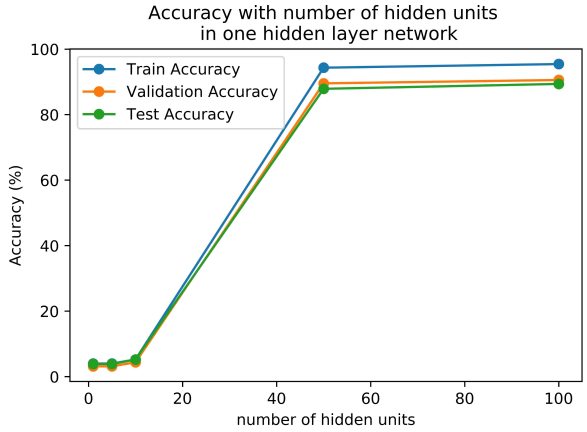
Figure 1 shows the accuracy, number of epochs and the training time achieved by network with one hidden layer but different number of perceptrons in that layer. Figure 1a shows the performance of networks with different number of hidden units using a theta initialization from a normal distribution with $\mu = 0$ and $\sigma = 1$. The accuracy keeps increasing with the number of hidden units. The test accuracy obtained with 1 hidden unit is random probability because one unit might not be enough to learn all the input features. The Test accuracy improves with 5 and 10 hidden units going upto 65% but takes significant epochs to train. The test accuracy with 50 and 100 hidden units is around 70%, showing that this theta initialization proves detrimental to larger networks. The number of epochs and hence the training time is large using this theta initialization as shown in Figure 1b.

Figure 2 shows the performance obtained by the networks with a different theta initialization from normal distribution of $\mu = 0$ and $\sigma = 0.05$. The performance of smaller networks is poor however, the larger networks achieve good performance is smaller number of epochs. With $\alpha = 0.1$, as shown in Figure 2a and 2b, the **test accuracy** obtained by networks with 50 and 100 units is close to **90%** and is achieved within **1400 epochs** and around 5 minutes of training time. With $\alpha = 0.5$, as shown in Figure 2c and 2d, the **test accuracy** obtained by networks with 50 and 100 units is close to **90%**. In contrast to $\alpha = 0.1$, similar accuracy is achieved within **600 epochs** and 2 minutes of training time.

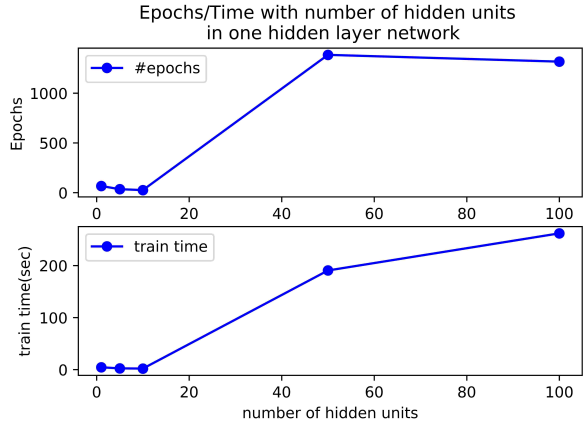
2.3 Part C - Adaptive Learning Rate in Single Hidden Layer Network

In this section, we will use adaptive learning rate to train the single hidden layer neural network with different hidden units from {1, 5, 10, 50, 100}. The learning rate will continuously decrease inversely proportional to the number of epochs i.e. $\alpha_t = \eta_0 / epoch^{1/p}$. We experiment with different values of η_0 and p to decide the factor by which learning rate will decrease.

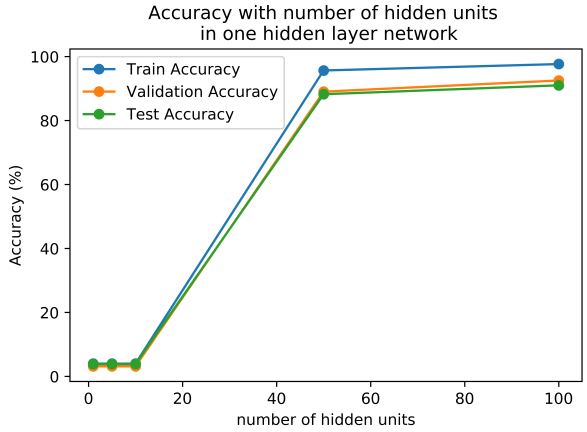
The **stopping criteria remains the same** for the adaptive



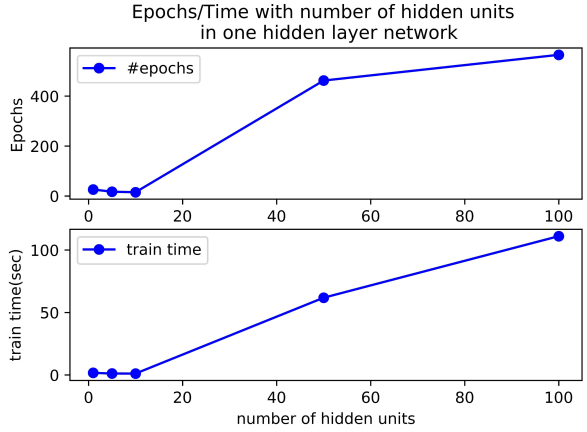
(a) Accuracy with LR=0.1



(b) Epochs and Train time with LR=0.1



(c) Accuracy with LR=0.5



(d) Epochs and Train time with LR=0.5

Figure 2: Accuracy, Epochs and Training Time for neural networks with different number of hidden layer units with Theta Initialization from normal dist. with $\mu = 0$ and $\sigma = 0.05$ and using $\alpha = 0.1$ and $\alpha = 0.5$

learning as in the previous section. So, the training stops when the change in the validation loss becomes less than $1e^{-5}$ across 10 continuous epochs. The **Theta Initialization** also remains same that is from a normal distribution with $\mu = 0$ and $\sigma = 0.05$.

Adaptive Policy $\eta_0/epoch^{(1/p)}$	**epochs**	**Train Acc(%)**	**Test Acc(%)**
$\eta_0 = 0.5, p = 2$	2014	90	85.91
$\eta_0 = 0.5, p = 3$	1236	95.72	89.63
$\eta_0 = 1.5, p = 2$	1189	94.15	89.34
$\eta_0 = 1.5, p = 3$	782	96.5	90.23
$\eta_0 = 1.5, p = 4$	602	97.18	90.54

Table 2: Accuracy of a neural network with one hidden layer having **100 perceptron units** with adaptive learning rate $\alpha = \eta_0/epoch^{(1/p)}$. The theta initialization and the stopping criteria remains same as reported in Section 2.2

Table 2 and Figure 3 shows the performance comparison of a single hidden layer neural network with 100 hidden units and different adaptive learning rate policies. The network with 100 hidden units performs best with $\eta_0 = 1.5$ and $p = 4$ so the learning rate α reduces as $\alpha_t = 1.5/epoch^{1/4}$ as the training progresses. The adaptive policy with $\eta_0 = 0.5$ and $p = 2$ results in degradation of performance from the performance reported in Section 2.2. So this criteria should not be used.

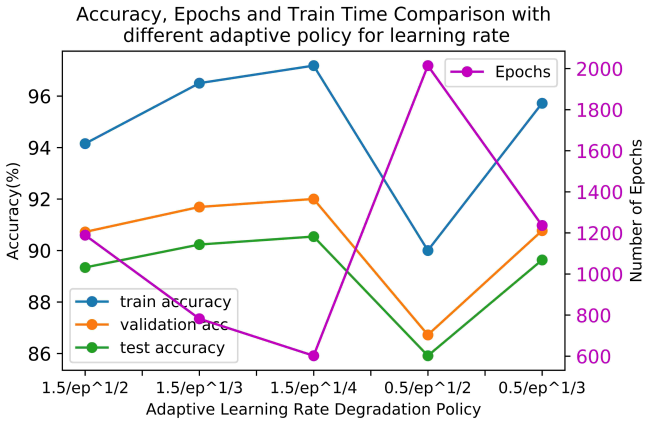
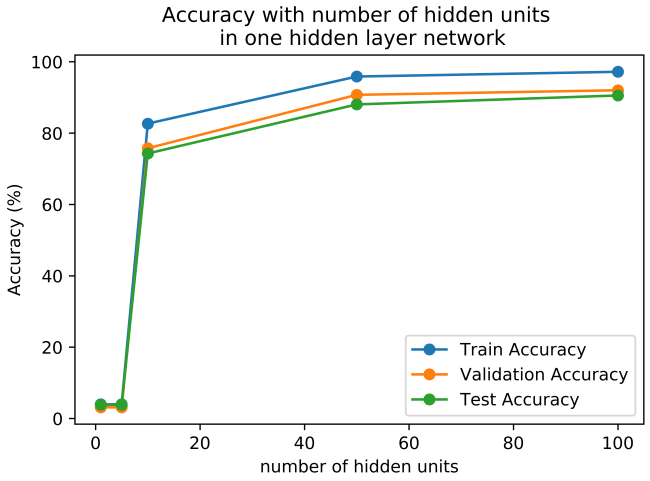
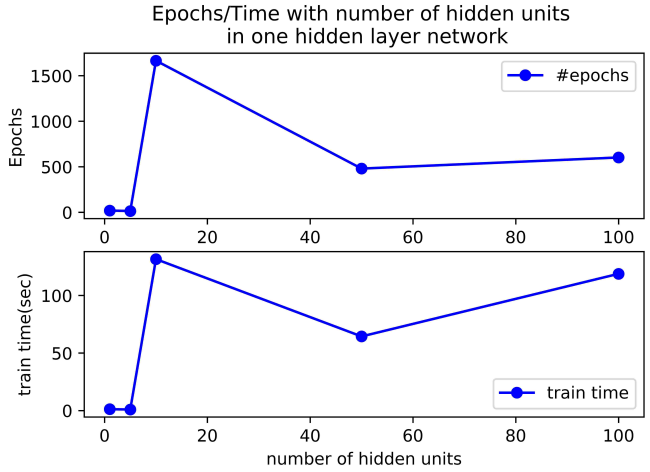


Figure 3: Comparison of different adaptive policies based on different values of η_0 and p in $\alpha = \eta_0/epoch^{1/p}$.

Figure 4 shows the performance of different neural networks with different number of hidden layer perceptrons by using the adaptive policy $\alpha_t = 1.5/epoch^{1/4}$. This policy performs best to train the neural network and the training time is very low. The figure also shows that this adaptive policy proves beneficial for Neural Network with 10 perceptrons which, in contrast to Section 2.2, shows an accuracy of 74.28% with the same theta initialization in around 1600 epochs and 2 minutes of training time. The figure also shows that networks with {50, 100} hidden units are able to achieve 90% accuracy within 500 epochs (approximately 1 minute of training time).



(a) Accuracy obtained with adaptive learning rate and $eta_0 = 1.5$ and $p = 4$



(b) Epochs and Training time with adaptive learning rate and $eta_0 = 1.5$ and $p = 4$

Figure 4: Accuracy, Epochs and Training Time of single hidden layer neural network with different number of hidden units trained with adaptive learning rate $\alpha = 1.5/epoch^{1/4}$.

Hence, the adaptive learning does make the training faster provided the hyperparameters η_0 and p are tuned properly based on the network architecture.

2.4 Part D - Multi-Layer Neural Network with ReLU Activation for Multi-Class Classification

In this section, the neural network architecture will have 2 hidden layers, each with 100 perceptron units. The activation function used for the hidden layers is Rectified Linear Unit (ReLU), as shown in Equation 6 and the activation function used for the output layer is Sigmoid, as shown in Equation 2, to get probabilistic outputs for the labels in the range of $(0, 1)$.

$$h_\theta(x) = g(\theta^T x) = \max(0.0, x) \tag{6}$$

$$g'(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases} \tag{7}$$

The derivative of the ReLU is shown in Equation 7 which will replace the derivative of Sigmoid in the Equation 4 to calculate the δ_l of the hidden layers. The derivative of ReLU does not exist at $x = 0$ so any value between 0 and 1 could be used by using sub-gradients. In the later part of the section, we would also analyze the performance of Softplus activation which is a sigmoid approximation of ReLU at $x = 0$.

Adaptive Learning rate is used as in the previous section to train the network and we will use $\alpha_t = \eta_0/epoch^{1/4}$ with different values of η_0.

Theta Initialization: The theta initialization for ReLU is done from a uniform distribution between $(-0.05, 0.05)$ instead of a normal distribution. The network performs more consistently with this theta initialization.

Stopping Criteria: The Stopping Criteria that works most consistently with ReLU is based on the validation loss. While training the network with ReLU activation, the validation loss starts to increase after the network converges close to the minima. It is also observed that while training the network with ReLU activation, when the loss is close to the minima, it starts oscillating and after some time it starts increasing. So, the training is stopped as soon as the validation loss starts increasing and increases continuously for 10 epochs. We choose 10 epochs because it stops the training at the most optimum point. It allows the network to come out of local minima and also doesn't degrade the performance too much.

Adaptive Policy $\eta_0/epoch^{(1/p)}$	**epochs**	**Train Acc(%)**	**Test Acc(%)**
$\eta_0 = 0.6, p = 4$	130	87.11	82.32
$\eta_0 = 0.5, p = 4$	243	87.78	82.54
$\eta_0 = 0.5, p = 5$	242	88.98	82.55
$\eta_0 = 0.5, p = 3$	286	87	82.3

Table 3: Accuracy of a neural network 2 hidden layers [100,100] with ReLU activation and with adaptive learning rate $\alpha = \eta_0/epoch^{(1/p)}$ with theta initialization from a uniform distribution between $(-0.05, 0.05)$

Table 3 shows the performance of the network with ReLU activation and different values of η_0 in the adaptive learning rate policy. The best performance is achieved by the network is with $\alpha_t = 0.5/epoch^{1/5}$, however, more consistent results are obtained with $\alpha_t = 0.6/epoch^{1/4}$. The **Test Accuracy** achieved by the network is close to **83%** in **130 epochs** with training time of around half a minute.

Softplus Activation:

Softplus is a sigmoid approximation of ReLU at $x = 0$. Equation 8 shows the softplus activation function used to approximate ReLU at $x = 0$. It is also called SmoothReLU. It's derivative as shown in Equation 9 is equal to the logistic sigmoid function.

$$h_\theta(x) = g(\theta^T x) = \log(1 + e^x) \tag{8}$$

$$g'(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

The network performs better than ReLU with Softplus activation and achieves a **test accuracy of 89% in 200 epochs** with train time of about 2 minutes. The stopping criteria and the theta initialization remains the same as ReLU.

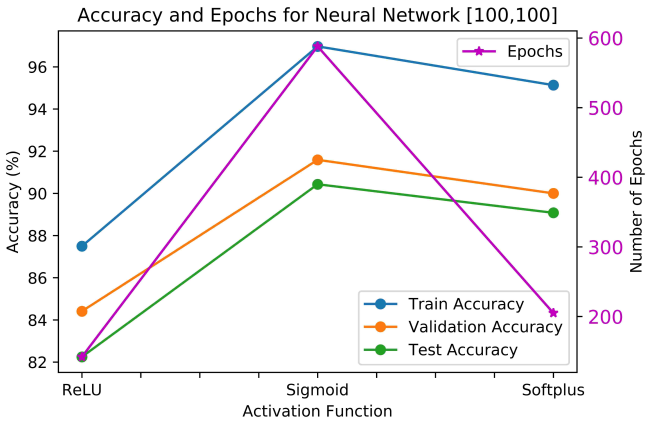


Figure 5: Accuracy and number of Epochs Comparison of 2 hidden layer neural network with different Activation units - ReLU, Sigmoid and Softplus.

Figure 5 shows the comparison of the neural network with 2 hidden layers with 100 perceptrons each with different activation units including ReLU, Softplus and Sigmoid. The **network achieves the maximum accuracy** of 90% with sigmoid, 89% accuracy using softplus and 83% using ReLU. However, the training time of sigmoid is the highest amongst all. The network with sigmoid activation converges in around 600 epochs taking around 3 minutes of time.

Hence, we can **conclude** that the network with ReLU and Softplus are better in performance since they can achieve comparable accuracy in very small amount of time. On comparison with the network with one hidden layer, as in Section 2.2, we can also comment that using ReLU activation is better than sigmoid and a deeper layered network converges faster and achieves higher accuracy.

2.5 Part E: MLP classifier from SKLearn for Multi-Class Classification

In this section, we will use the neural network library from sklearn to train a multi-layer perceptron classifier for the given classification problem. We will use the same one-hot encoded label outputs as the input to the classifier. The architecture of the neural network is same as in Section 2.4. The MLP classifier uses cross-entropy loss over the final network output instead of mean-squared error (MSE). We will also compare our results with the results obtained in Section 2.6 where we have implemented cross entropy loss in our previous implementation. The MLP classifier is trained with Sigmoid Activation and ReLU activation to compare how the different activation units perform and then compare the results obtained in Section 2.4.

The network architecture is two hidden layer with 100 perceptrons each, the learning rate is 0.1, the batch size is 100 and the stopping criteria is used according to what we obtained from our implementation to be able to compare results. We use SGD to optimize the loss function.

Hyper-parameters	**epochs**	**Train Acc(%)**	**Test Acc(%)**	**Time (s)**
Constant $\alpha = 0.1$	289	100	88.06	120
Early Stopping with Validation Loss, $\alpha = 0.1$	63	98.82	87.22	23.08
LR InvScaling with $\eta_0 = 0.3, p = 4$	400	96.94	83.65	166
LR InvScaling with $\eta_0 = 0.3, p = 5$	400	99.91	87.32	160
LR InvScaling with $\eta_0 = 0.3, p = 6$	400	99.96	86.08	161

Table 4: Accuracy of a neural network 2 hidden layers [100,100] with MLP classifier set with different hyperparameter settings. Fixed Settings: Architecture-(100,100), Activation-"Logistic", Solver-"SGD", max iteration-400, Stopping Criteria=$1e^{-5}$

MLP classifier with Logistic Activation:
The MLP classifier is first trained with logistic activations for the two hidden layers. The activation for the output layer is always logistic in this case. We experiment with some of the hyperparameters of the library to report different results and performance in Table 4. The Stopping Criteria used is on the loss when it stops improving in the range of $1e^{-5}$ for over 10 epochs.

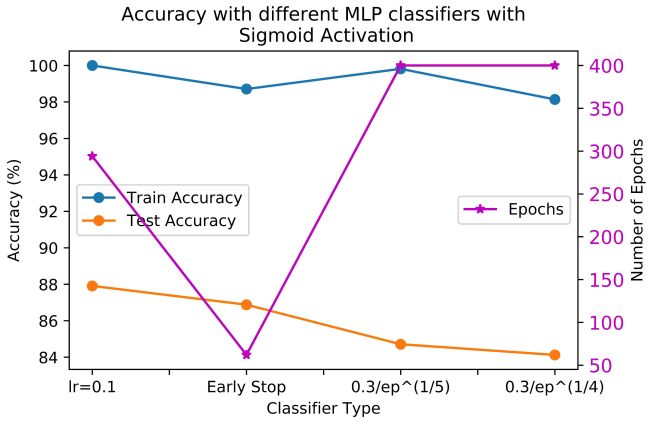


Figure 6: Accuracy achieved by MLP Classifier with different hyperparameter settings for network with 2 hidden layers with Sigmoid Activation.

Observations from Table 4 and Figure 6:

- The maximum test accuracy of 88% is achieved at constant learning rate of 0.1 in around 300 epochs.
- On using Validation Loss as the stopping criteria, similar accuracy is achieved within 70 epochs which is much faster and avoids overfitting. Validation loss is calculated on 10% of the training dataset.

- Using adaptive learning rate as in Section 2.3, we see that the MLP classifier reports a degradation in performance. The network trains much slower and achieves comparable accuracy in around 400 epochs. The most optimum setting found was $\eta_0 = 0.3$ and $p = 5$ for setting $\alpha_t = \eta_0/epoch^{1/p}$ in the adaptive setting. When compared with our implementation in Section 2.3 which achieved around 90% accuracy in 600 epochs with adaptive learning rate $\alpha_t = 1.5/epoch^{1/4}$, this implementation is comparable but faster in training speed but achieves lower maximum accuracy.

Hyper-parameters	**epochs**	**Train Acc(%)**	**Test Acc(%)**	**Time (s)**
Constant $\alpha = 0.03$, stop:$1e^{-4}$	68	100	87.75	28
Constant $\alpha = 0.03$, stop:$1e^{-5}$	43	98.83	86.51	18
Constant $\alpha = 0.1$	32	94.72	84	13
Early Stopping with Validation Loss, $\alpha = 0.1$	31	96.42	85.74	11
LR InvScaling with $\eta_0 = 0.1, p = 2$	400	86.03	80.51	190
LR InvScaling with $\eta_0 = 0.1, p = 3$	400	96.3	84.3	206
LR InvScaling with $\eta_0 = 0.1, p = 4$	400	99.96	84.15	164

Table 5: Accuracy of a neural network 2 hidden layers [100,100] with MLP classifier set with different hyperparameter settings. Fixed Settings: Architecture-(100,100), Activation-"ReLU", Solver-"SGD", max iteration-400, Stopping Criteria=$1e^{-5}$

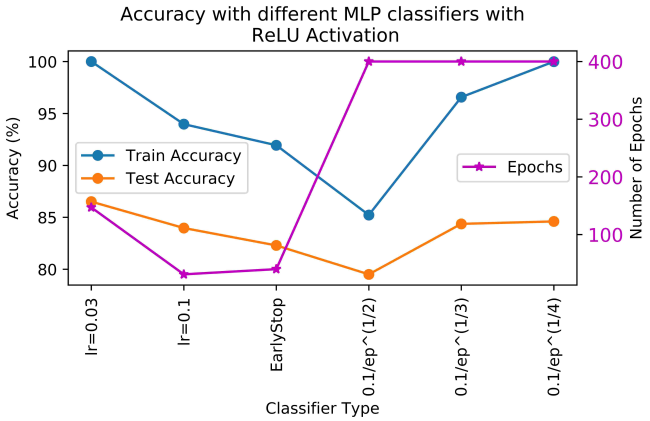


Figure 7: Accuracy achieved by MLP Classifier with different hyperparameter settings for network with 2 hidden layers with ReLU Activation.

MLP classifier with ReLU Activation:
The MLP classifier is trained with ReLU activations for the two hidden layers. The activation for the output layer is always logistic even in this case. We experiment with some of the hyperparameters of the library to report different results and performance in Table 5. The Stopping Criteria used is on the loss when it stops improving in the range of $1e^{-5}$ for over 10 epochs. The table shows that the maximum accuracy of around 88 % is achieved at $\alpha = 0.03$ within 70 epochs.

Observations from Table 5 and Figure 7:

- The maximum test accuracy of 88% is achieved at constant learning rate of 0.03 in around 70 epochs.
- On using Validation Loss as the stopping criteria, accuracy reduces by around 4% and the network achieves 84% accuracy within 30 epochs which is much faster and avoids overfitting. Validation loss is calculated on 10% of the training dataset.
- Using adaptive learning rate as in Section 2.3, we see that the MLP classifier reports a degradation in performance. The network trains much slower and achieves comparable accuracy in around 400 epochs. The most optimum setting found was $\eta_0 = 0.1$ and $p = 4$ for setting $\alpha_t = \eta_0/epoch^{1/p}$ in the adaptive setting. When compared with our implementation in Section 2.3 which achieved around 83% accuracy in 130 epochs with adaptive learning rate $\alpha_t = 0.6/epoch^{1/4}$, the MLP classifier achieves a higher accuracy but is very slow in training speed.

Figure 8 shows the performance of Sigmoid activation vs. ReLU activation for training a two hidden layer neural network. When compared to Sigmoid, ReLU trains the network faster but achieves a little lower test accuracy. These observations are in same lines to the observation that we made from Figure 5 which also showed that the Sigmoid achieves higher accuracy but is around 5× slower than ReLU.

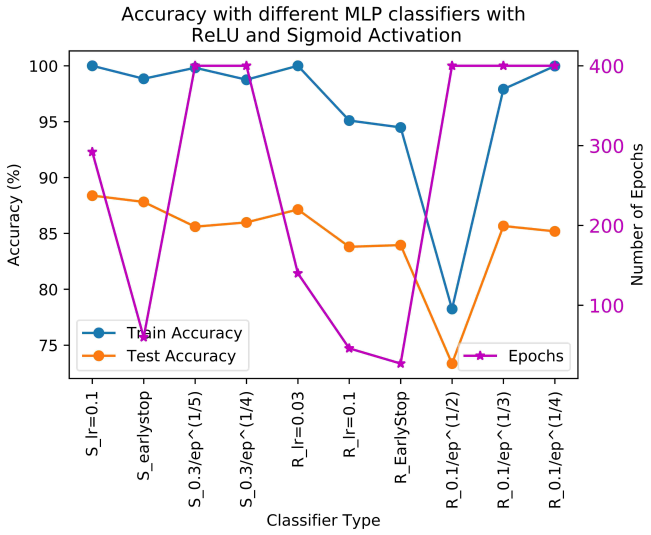


Figure 8: Accuracy achieved by MLP Classifier with different hyperparameter settings for network with 2 hidden layers with ReLU and Sigmoid Activation.

2.6 Part F: Cross-Entropy Loss in Multi-Class Classification with multi-layer neural network

In this section, we use a different loss function to train the network on the given dataset. We use Cross-entropy loss over MSE as used in the previous sections. More specifically, we will use the binary cross entropy loss as shown in Equation 10 because we have one hot encoded out multi-class labels so we will have r different two class problems and we get the total entropy by adding those.

$$H_p(q) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i)) \tag{10}$$

We observe that by using this loss function, the performance of ReLU is more stable and it achieves higher accuracy of 85% at $\alpha_t = 0.1/epoch^{1/3}$, which is similar to the MLP classifier at adaptive learning rate. However, at constant learning rate our cross-entropy implementation only reaches a maximum accuracy of 84.4% as compared to 87% achieved using MLP classifier. Our implementation is however faster than the MLP classifier and achieves comparable performance within 130 epochs. Figure 9 shows the comparison of ReLU and Softplus trained with cross-entropy loss. Network trained with Softplus is better than ReLU activation in terms of accuracy. We observe that using cross-entropy loss the network converges faster than MSE.

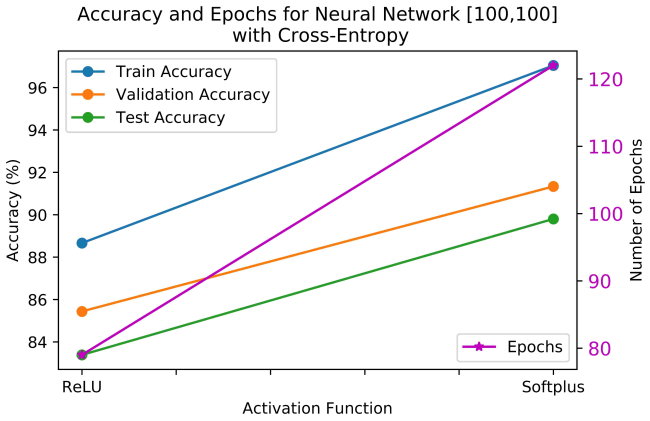


Figure 9: Accuracy achieved by a network with 2 hidden layers with ReLU and Softplus Activation with Cross-Entropy Loss.

3 Conclusions

This report is a summary of the third assignment, second part, done for Machine Learning course. We obtain a good understanding of how the algorithms that we talked about in this report are implemented and we also analyzed the affect of different hyper-parameters on the learning.