# ELL888

## Assignment 1

Aditi Jha(2015EE10504)*, Saksham Soni(2015EE10534)*, Sanyam Gupta(2015EE30761)*

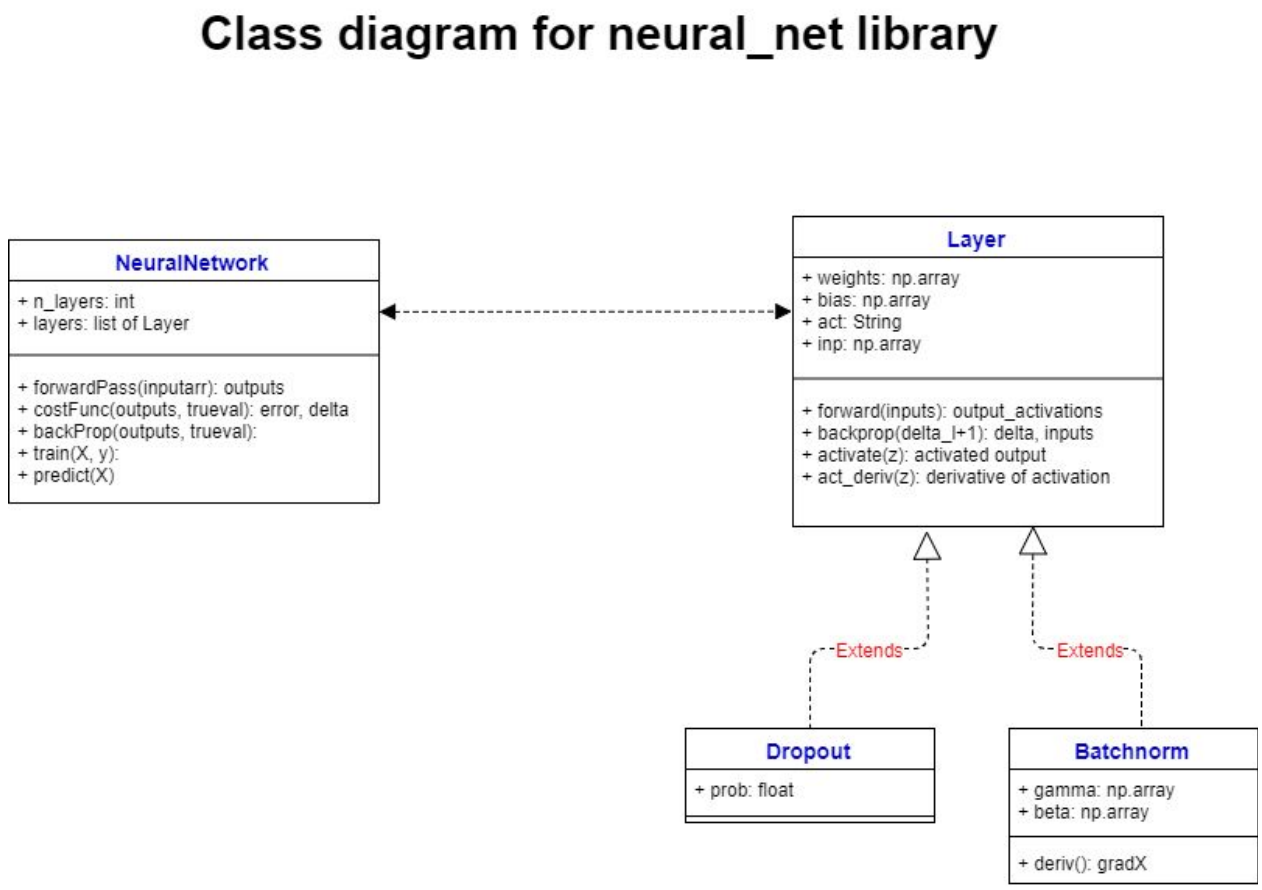



Fig. 1

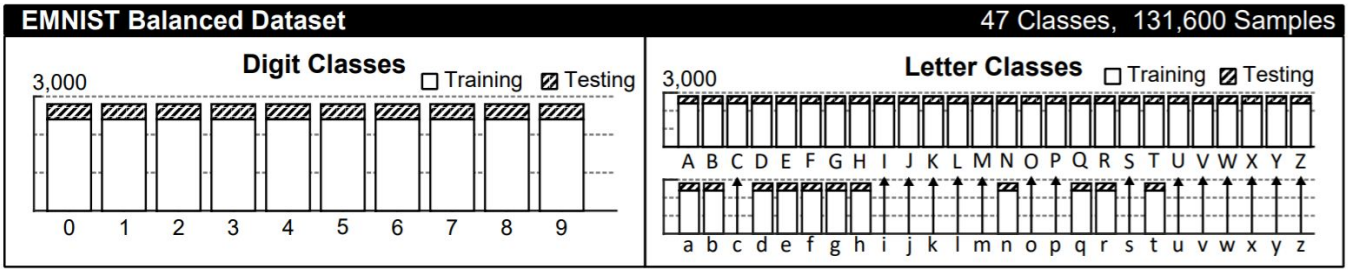



Fig. 2 (Source - https://arxiv.org/pdf/1702.05373.pdf)

**Description of our *neural_net* library:**

The class diagram in *Fig. 1* summarizes the structure of our library. The NeuralNetwork class provides an

*=equal contribution

interface to create a network of given architecture and train it. We have implemented **Mean-Squared-Error** and **Cross-Entropy** loss.

**Activation functions implemented:**

- Sigmoid
- ReLU
- Leaky ReLU
- Tanh
- Softmax

**Challenges faced in the code:**

We had to handle the following problems

- Overflow in values: Because numpy library tries to keep the maximum precision in values, calculating sigmoid when it goes very close to zero or one sometimes causes overflow issues and causes NaN values in the network which give inexplicable results. We handled this by clipping the input to sigmoid activation function.
- Calculating softmax can also sometimes cause overflow while computing exponents. We used the stabilized softmax implementation for this.
- Divide by zero errors while taking log in cross entropy. We added a small epsilon (of the order of 1e-20) to handle this.

**Optimizations:**

- All the calculations in the backprop are vectorized and the forward pass and back-propagation of gradients is efficiently handled inside the *NeuralNetwork* and *Layers* classes.
- The outputs of hidden layers are stored within the class objects itself which makes calculation of gradients easy.
- The computation of derivatives of sigmoid, softmax and tanh is faster if we have their outputs, since intermediate outputs are saved within the layers, this is used for speeding up.
- In the case of softmax the gradient is a square matrix, but for the special case of softmax with cross entropy the delta takes a simple form which we directly compute making the back-prop faster.

**Initialisation:**

We use **Xavier initialisation** to initialise all our weights.

**Input normalization:**

The input features are pixels which are integers between 0 and 255. For better performance of the network and ease of training we normalize the images by subtracting the mean and dividing by 255.

**Optimisation method:**

To update our weights, we use **mini-batch gradient descent** and **adam optimizer**.

**Hyper-parameter tuning:**

To tune the hyper parameters we coded a line search algorithm. We use ***K-fold cross validation*** with 3 folds.

**Performance Metrics:**

We are calculating the accuracy as:
(number of correctly classified samples/total samples) x 100

**Letters for Classification:**

The structure of EMNIST balanced dataset is explained by *Fig. 2*. The subset of letters that we have chosen is ***{ A, D, G, H, I, J, K, N, O}***

**Constants throughout the experiments:**

- The value of alpha in case of Leaky ReLu is always taken to be 0.01.
- All experiments have been conducted with mini-batch Gradient Descent, except where adam optimizer is explicitly mentioned.
- We have used 80% of training data for as training set and 20% for cross validation.
- We have used Cross Entropy Loss loss for all our experiments unless mentioned explicitly.

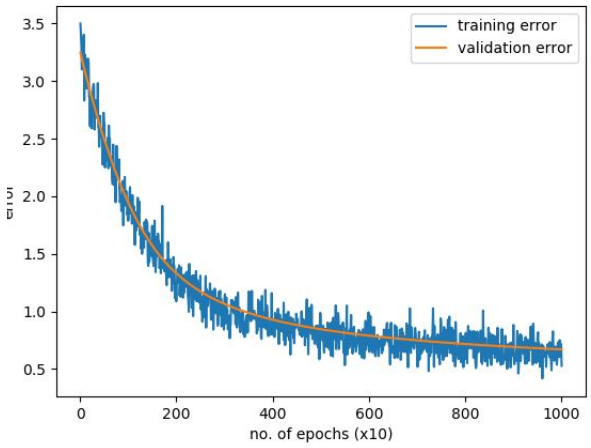
## Experiments Conducted:

### Effect of depth and activation:

We start with a simple neural network with softmax activation on the output layer without any regularisation. We used mini-batch vanilla

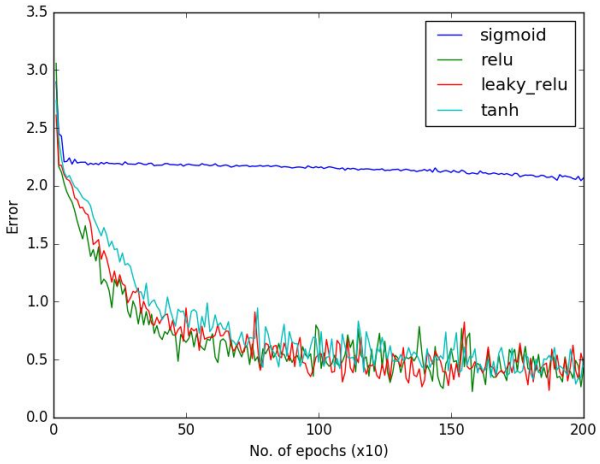
backpropagation technique to train the model with cross-entropy error and obtain the following curve:

**No Hidden Layer:**




*Training and Validation Curve with no Hidden Layer*
Correct-classification accuracy: 62.27%

**Adding one hidden layer with 256 neurons**, we also study the effect of various activation functions in the hidden layers like Sigmoid, ReLu, Leaky ReLu and tanh with mini-batch gradient descent.




*Training Curve for different activation functions on one hidden layer*

We repeat the same experiment with Adam optimiser to observe the effect of activation with Adam optimiser.

| Activation function | Accuracy with mini-batch gradient descent | Accuracy with adam |
|---|---|---|
| LeakyReLu | 72.19% | 83.44% |
| ReLu | 71.88% | 84.277% |
| Tanh | 70.41% | 83.21% |
| Sigmoid | 67.91% | 81.27% |

The graph shows that the **error reduces fastest in case of Leaky ReLu and ReLu, followed by tanh and then sigmoid**. Accuracy of the model is also in the same order. While Leaky ReLu performs slightly better than ReLu in mini-batch stochastic gradient descent, the reverse happens in case of adam. Both have their pros and cons, while ReLu creates a more sparse representation, it suffers from vanishing gradient problem. Leaky ReLu has the opposite effect and hence, the results might vary for them.

Our findings are in line with theoretical claims: ReLu performs best as it does not saturate while invoking sparsity. Also, because of its linear nature, computation time also is less for ReLu. Sigmoid and Tanh both saturate easily and hence suffer from vanishing gradient but tanh is zero-centred, and hence performs better.
Also, we observe that **accuracy increases upon increasing depth** by one hidden layer.

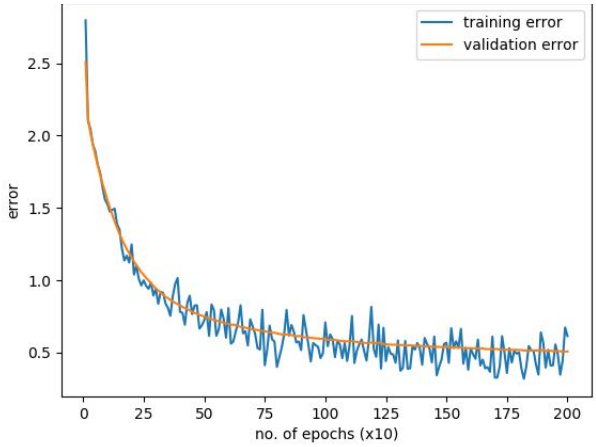
With **2 hidden layers, 784-512-256-9** architecture and Leaky ReLu activation, we obtain an accuracy of 74.77% with mini-batch gradient descent, however with adam optimiser we observe an increase in accuracy to 87.55%. **In either case, the accuracy improves with increasing depth.**

## Effect of Regularisation:

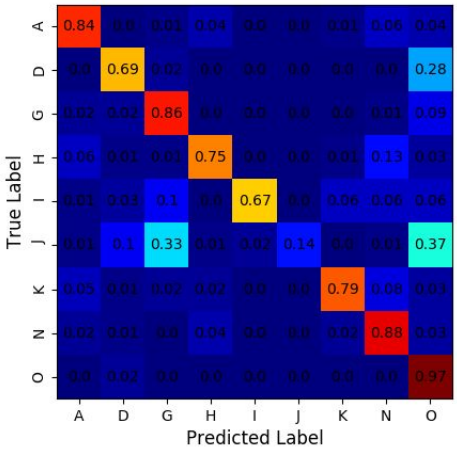
We use L1 and L2 regularisation methods to evaluate these methods as a measure of performance improvement.

**L2 regularisation:**

We first use L2 regularisation with one hidden layer(256 neurons) and Leaky ReLu activation.

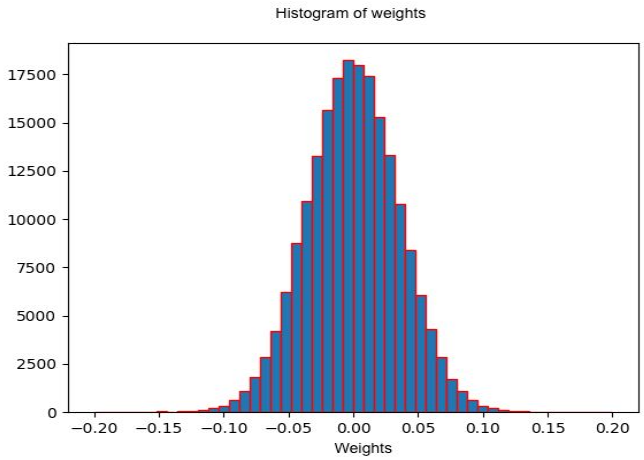



*Training and Validation curve with L2 regularisation with lambda=0.01*




Correct-classification accuracy: 75.72%

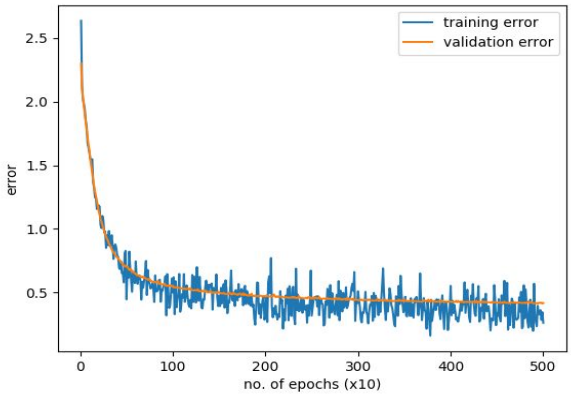
We observe that the accuracy with L2 regularisation improves.




*Histogram of weights with L2 regularisation*

The weight histogram obtained, shows that the weights are restricted by applying regularisation. Also, this **histogram resembles Gaussian distribution** which is what L2 theoretically corresponds to.

Now, when we increase the depth of the network to **two hidden layers,** with ReLu activation:

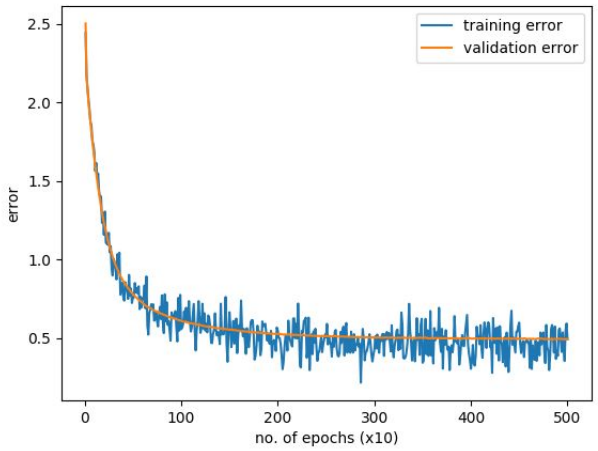



*Training and Validation curve for L2 regularisation with two hidden layers , lambda=0.01*

Correct-classification accuracy: 76.25%

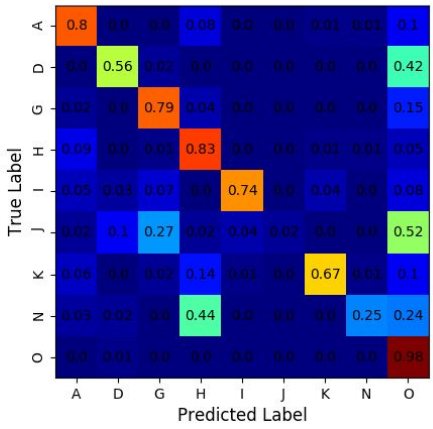
With two hidden layers, when we use no regularisation, the accuracy is 74.77%, and with L2 we **observe a slight increase in accuracy**. Hence, **as the network becomes deeper, the need for restricting weights is more.** Hence, regularisation is beneficial in a deep neural network.

## L1 regularisation:

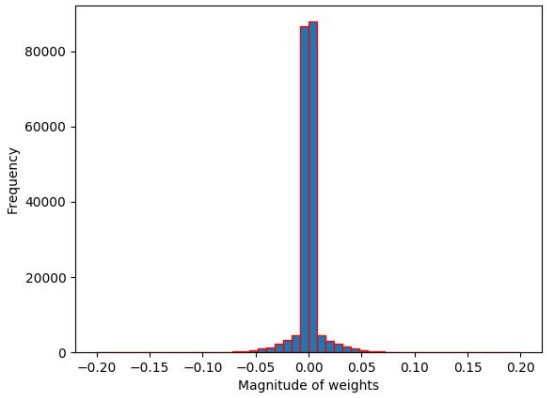



*Training and Validation curve for L1 regularisation with two hidden layers , lambda=0.01*

Correct Classification accuracy: 68.75%




We see that the accuracy is lesser in L1 compared to L2. The confusion matrix above seconds the inference from L1 regularisation that restricting weights to around 0 does not let the network learn J leading to decrease in accuracy.

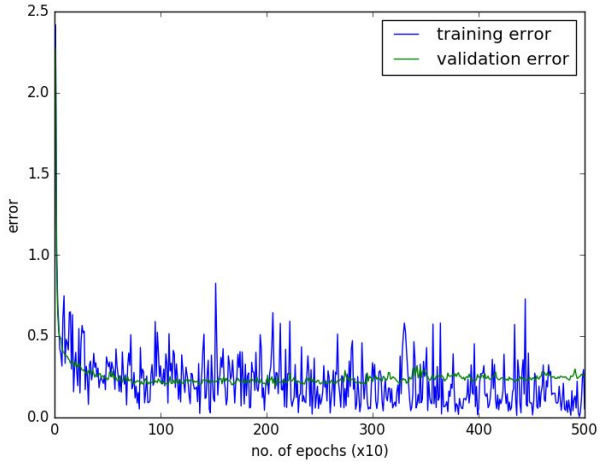



The weight histogram above shows that L1 regularisation restricts most weights to zero, which implies that **L1 invokes sparsity**.

## Dropout:

The idea behind Dropout is to train an ensemble of DNNs and average the results of the whole ensemble instead of train a single DNN. We have performed experiments on shallow and deep networks.

With **1 hidden layers(784-256-9), ReLu activation**, using Cross Entropy Loss, and dropout with prob = 0.5, we obtain the following results

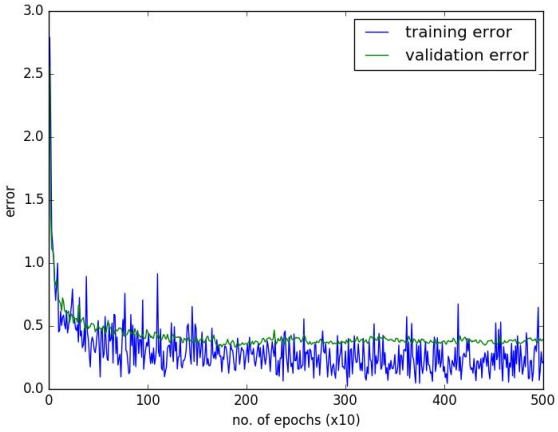



*Training and Validation curve for dropout prob = 0.5 and 1 hidden layer*

**Correct Classification Accuracy: 72.62%**

This is comparable to the accuracy without dropout which was 72.19%. This is because for a single hidden layer and high value of dropout probability, the network loses more abstraction power than it gains through ensemble.

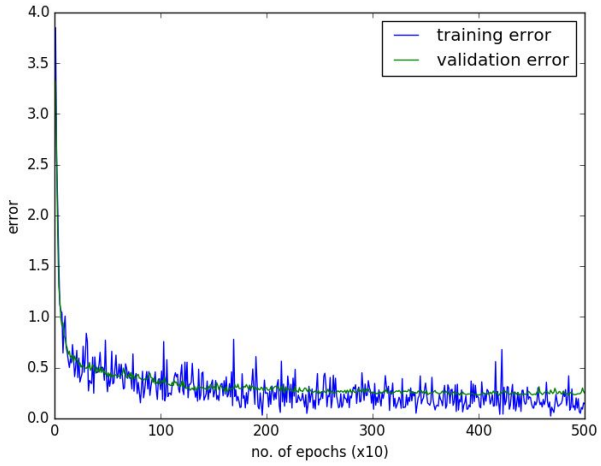
However, if we **decrease the dropout probability to 0.3** for the same architecture,




*Training and Validation curve for dropout prob = 0.3 and 1 hidden layer*

**Correct Classification Accuracy: 85.37%**
This is higher than the accuracy without dropout which was 72.19%. Hence, we can see that the if value of probability is chosen wisely, it can improve performance even for a single layer network.

With **2 hidden layers(784-256-128-9), ReLu activation**, using Cross Entropy Loss, and dropout with prob = 0.5,



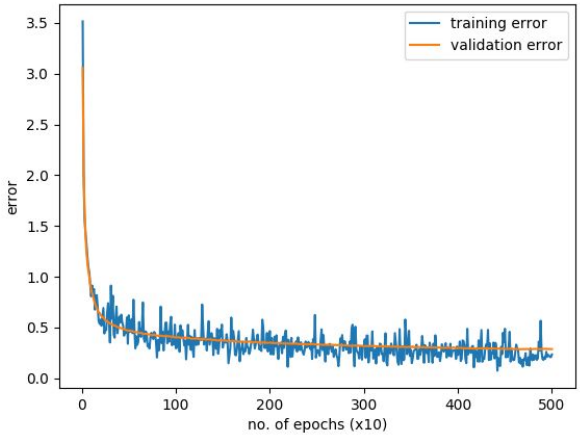

*Training and Validation curve for dropout prob = 0.5 and 2 hidden layers*

**Correct Classification Accuracy: 90.42%**
This is much higher than the accuracy without dropout which was 74.77%. Hence, we see a massive improvement in case of a deeper network. This can be attributed to the **power of ensemble of several weaker deep neural nets**. Even for a higher value of probability, performance increases because of increase in the number of hidden layers.

## Batchnorm:

Batch-normalisation increases our accuracy in all settings. Batch normalisation, being a form of regularisation is more efficient in larger networks where is leads to a drastic increase in accuracy.

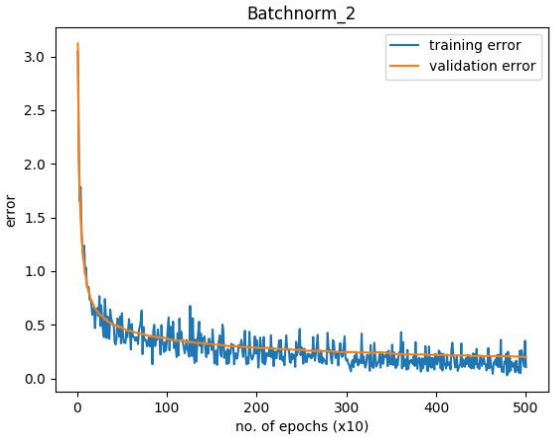
With **1 hidden layer(784-256-9), ReLu activation**, using Cross Entropy Loss, and batch-normalisation, we obtain the following results:




*Training and Validation curve for batch-normalisation and 1 hidden layer*

**Correct Classification Accuracy: 84.33%**
This is higher than the accuracy without batch-normalisation which was 72.19%. Also, convergence is attained faster with batch-normalisation.

With **2 hidden layers(784-256-128-9), ReLu activation**, using Cross Entropy Loss, and batch-normalisation, we obtain the following results with mini-batch gradient descent:



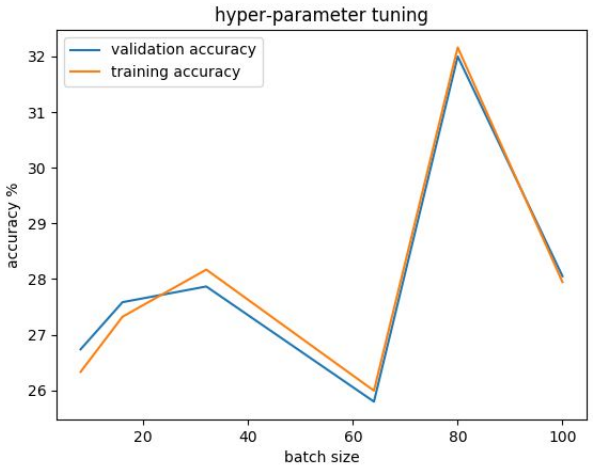

*Training and Validation curve for batch-normalisation and 2 hidden layers*

**Correct Classification Accuracy: 91.11%**
This is a great improvement over the same neural network without batch-norm, reported in the first section(Effect of depth and activation), which gives as accuracy of 74.77%. Hence, **batch-normalisation improves the performance much more in a deeper network.** This is partly because the shift and scale factor in batch normalisation provide a flexible regularisation option.

## Hyperparameter tuning:

We used line search with k-fold cross validation to tune the following hyperparameters:

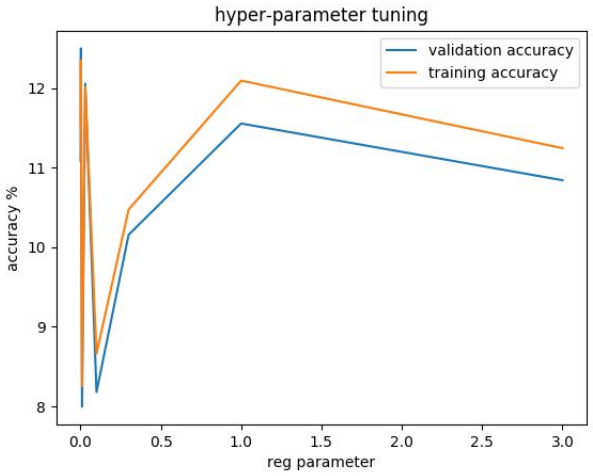
### Batch size:




*Hyperparameter tuning for batch size with one hidden layer*

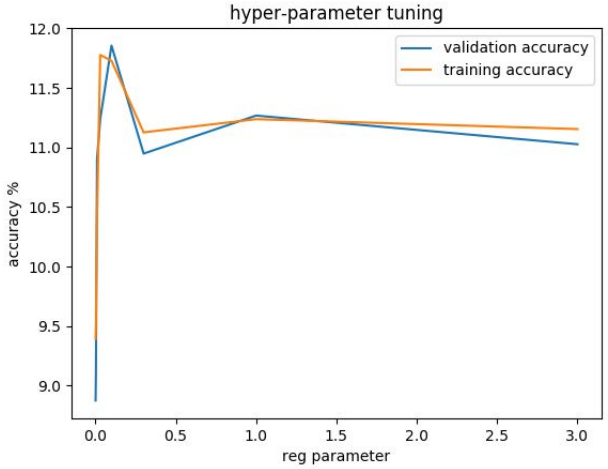
We obtain optimal batch size at around 80 which is used for our experiments

### Regularisation parameter for L2:




*Tuning regularisation parameter for L2 in case of one hidden layer*

We find a peak very close to 0, which shows that when number of hidden layers is less, need for regularisation is less as is also satisfied by the accuracy observed with L2 regularisation with one hidden layer.

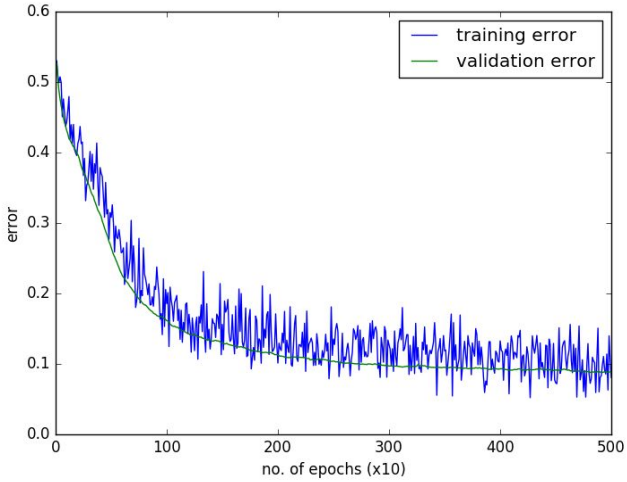



*Tuning regularisation parameter for L2 in case of two hidden layers*

Here the peak shifts towards the right emphasising the **increasing need for regularisation in case of dense networks.**

## Cost function:

We also use a second objective function: Mean Squared Error. Cross Entropy gives us better results than MSE, justifying its use for classification purposes. The log in Cross Entropy removes the saturation problems created by all exponential activation functions, giving us better accuracies.



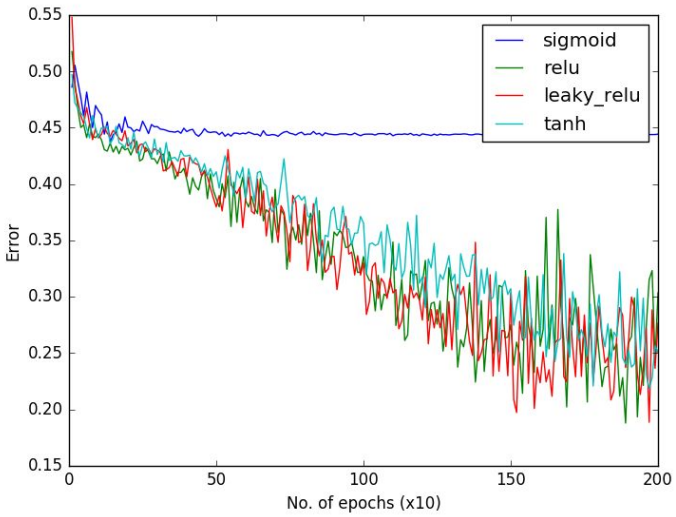

*Training curve for 2 hidden layers using Adam optimiser and MSE error*

Accuracy: 84.71%

This accuracy is lesser than what we obtained with two hidden layers for CE loss using Adam optimiser(87.55%).

We also **study the effect of MSE cost function on activation:**

**Adding one hidden layer with 256 neurons**, we also study the effect of various activation functions in the hidden layers like Sigmoid, ReLu, Leaky ReLu and tanh with mini-batch gradient descent.

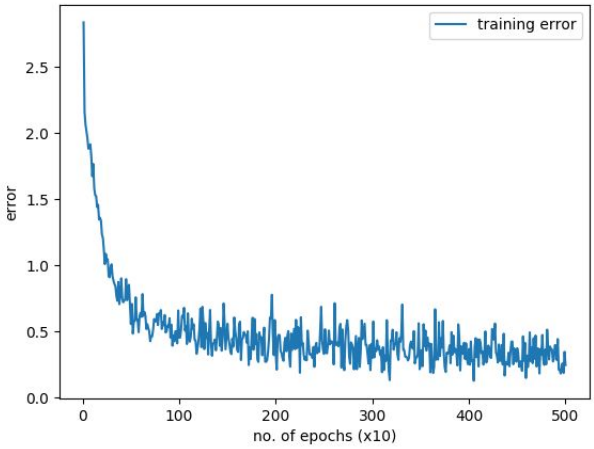



The trend seen in MSE is same as that of CE error function, Leaky Relu and Relu, followed by tanh and then sigmoid. No. of epochs needed to reach convergence, as can be seen in the curve, is more as compared to Cross entropy. Even after 2000 epochs, the learning curve is still steep and error is significant. Also, sigmoid performs very poorly, because of saturation which is not undone by MSE. The learning curve of sigmoid negligibly decreases, supporting theoretical claims about poor performance of sigmoid with Mean Squared error.

Hence, through both the above experiments, we conclude, **Cross entropy is a better loss function for classification.**

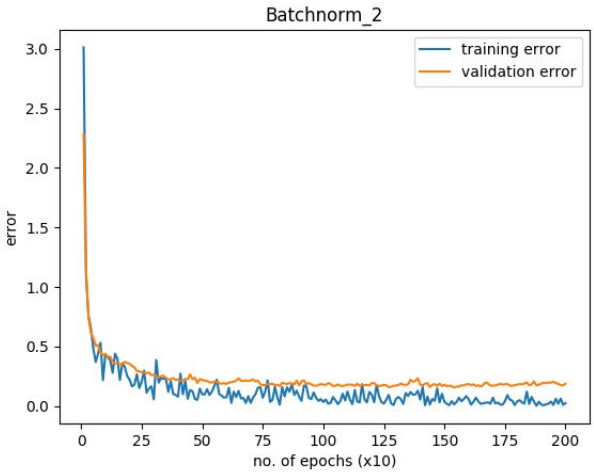
## Effect of using Adam optimiser:

**2 hidden layers, 784-512-256-9** architecture and Leaky ReLu activation with **mini-batch gradient descent:**




*Training curve for 2 hidden layers with mini-batch gradient descent*
Accuracy: 74.77%

**2 hidden layers, 784-512-256-9** architecture and Leaky ReLu activation with Adam optimiser on mini-batch:



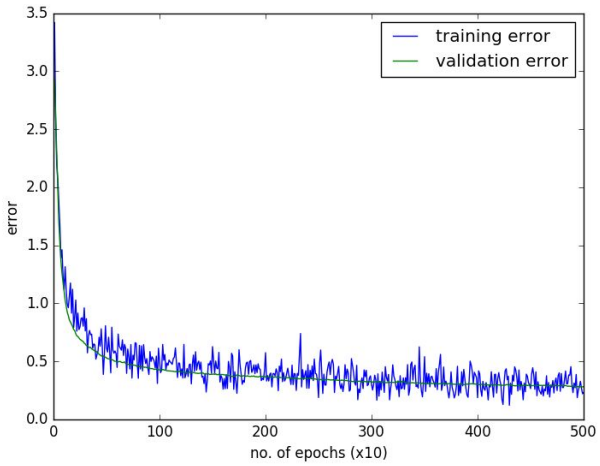

*Training curve for 2 hidden layers with Adam optimizer*

Accuracy: 87.55%

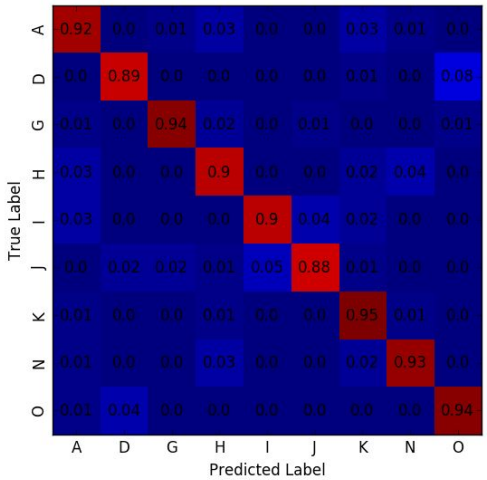
We observe that while **Adam largely improves the accuracy of the model, it also helps achieve convergence faster**. By making the learning rate adaptive and using momentum, Adam helps reach the minima quicker and better. Manual tuning becomes less important with adam, while with classical gradient descent, the effect of tuning parameters is much more.

## Best architecture:

The best architecture we obtain for **mini-batch gradient** is using **2 hidden layers, with dropout and batchnorm on the first, 512 neurons, followed by a second layer of 256 neurons with batchnorm using Leaky ReLu**.



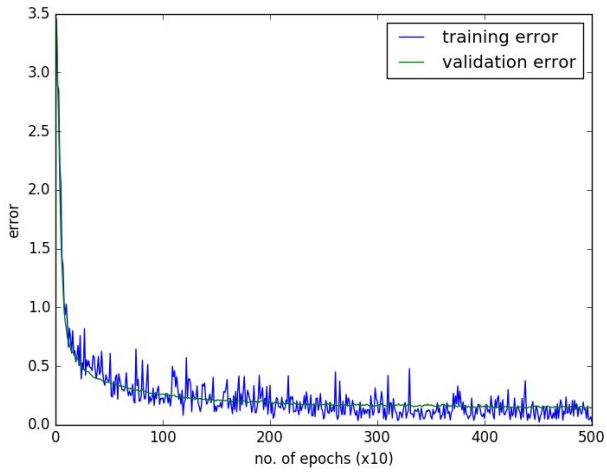

*Training and Validation curve for the above mentioned Dropout+Batchnorm network*
**Correct Classification Accuracy: 91.64%**



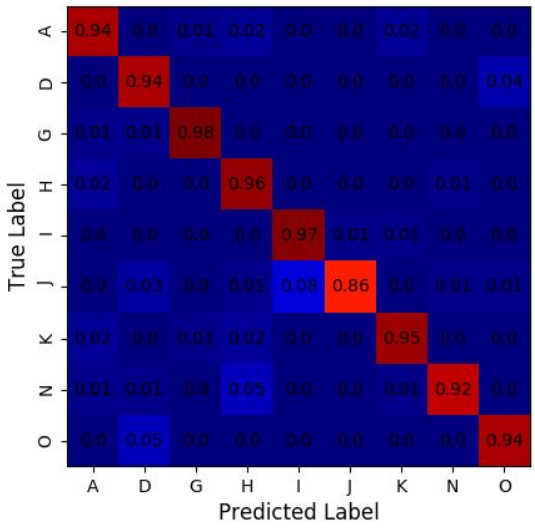

*Confusion matrix for the above mentioned network*

**Batchnorm and dropout together give us the best accuracy.**

Now, we train the same architecture using **Adam optimiser** and obtain even better results: **2 hidden layers, with dropout and batchnorm on the first, 512 neurons, followed by a second layer of 256 neurons with batchnorm using Leaky ReLu.**




*Training and Validation curve for the above mentioned Dropout+Batchnorm network using Adam optimiser on mini-batch*
**Correct-classification Accuracy: 93.88%**




*Confusion matrix for the above mentioned network*

**Conclusions:**

- Increasing depth increases accuracy in general.
- ReLu is the best activation function in terms of: time taken to reach convergence, accuracy and computational complexity, followed by tanh and sigmoid.
- Adam optimiser helps to attain convergence faster and also improves the accuracy by making the learning rate adaptive and adding momentum to the update equation.
- L1 regularisation makes the weights sparse and the histogram is concentrated around 0.
- The weight histogram in L2 regularisation takes the form of Gaussian distribution. Also, L2 increases the accuracy as the network becomes more dense.
- Dropout and Batchnorm increase the accuracy of the model in general, with their effect being more prominent in case of deeper networks.
- Hyperparameter tuning is important to achieve better results. Tuning of batch size, regularisation parameter, etc helps in improving accuracy of model.
- Mean Squared Error does not perform as good as Cross Entropy for classification tasks, and its performance with sigmoid is particularly poor.
- J is sometimes misclassified as G owing to similar curved shapes.
- The best architecture is obtained by combining batchnorm and Dropout. With Adam optimisation, this model gives us an accuracy of 93.88%.