For this problem, I implemented a fully connected neural network architecture with cross-entropy loss and SoftMax output activation layer. For each layer, I have implemented using the Numpy package, fully connected weight matrix which was randomized according to the normal distribution as well as choice of 3 activation functions: Sigmoid(), ReLU(), and Tanh(). I tested my implementation with various different architectures, where I tuned the hyper-parameters of the problem.

The hyperparameters tuned were the number of hidden layers and number of hidden units at each layer, the choice of activation function for each layer, the batch-size for training, number of training epochs, and the learning rate.

1) One of the relatively good architectures and set of hyperparameters used that gave reasonable results for training the MNIST dataset is shown below:

Input layer size: 784 (corresponding to the number of pixels at each image)

Number of Hidden layers: 2

Hidden layer 1 size: 64, *Hidden layer 1 activation:* ReLU()

Hidden layer 2 size: 64, *Hidden layer 2 activation:* ReLU()

Output Layer size: 10 (corresponding to 10 distinct digits)

Output Activation function: SoftMax, *Output Loss Function:* Cross-Entropy

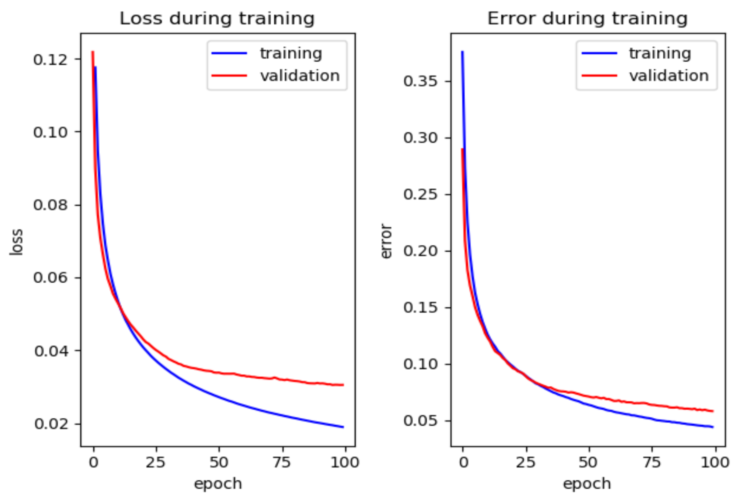
Learning Rate: 0.05, *Number of training epochs:* 100, *Batch-size:* 8

Final Test Loss: 0.03429 Final Test Error: 0.0623

Final Test Accuracy: 1-0.0623 = **93.77%**

Visualization of training process:

For the visualization of training process, we plot both the loss and the percentage error of classification for the training and the separate validation set.



2) In order to achieve a better accuracy than that of 95%. I proceeded to tune the hyperparameters furthermore. Also, upon investigating the literature for improving the accuracy, I used Smith et al., 2017 [1] work to start with a small batch-size that has been shown to have faster convergence to "good" solutions, since with a small batch-size we can start the learning without having to see the entirety of data. Then periodically I increase the batch-size as with a small batch-size we are not guaranteed to converge to a global optimum. In addition, I also decay the learning rate as a common practice (by periodically decreasing it throughout training epochs).

Below present the hyper-parameters of the improved model that can achieve <5% test error:

Input layer size: 784 (corresponding to the number of pixels at each image)

Number of Hidden layers: 2

Hidden layer 1 size: 96, *Hidden layer 1 activation:* ReLU()

Hidden layer 2 size: 36, *Hidden layer 2 activation:* ReLU()

Output Layer size: 10 (corresponding to 10 distinct digits)

Output Activation function: SoftMax, *Output Loss Function:* Cross-Entropy

Learning Rate: Start from **learning-rate of 0.1** and every $\frac{\#epochs}{4}$ halve the rate till **rate of 0.02** is achieved.

Number of training epochs: 300

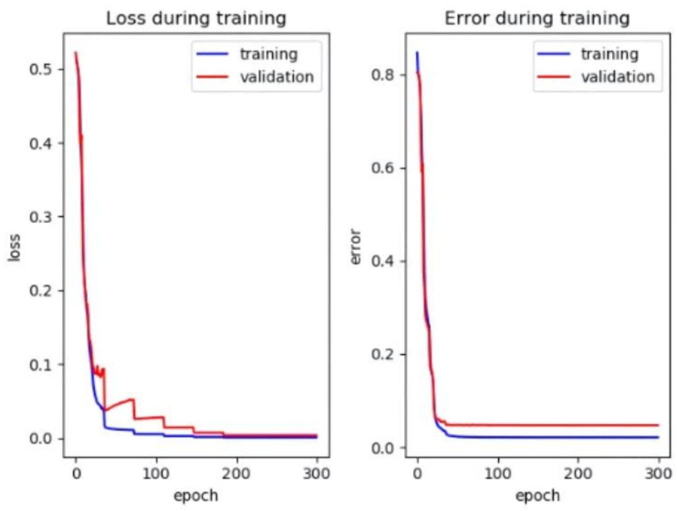
Batch-size: Start from **batch-size of 4** and every $\frac{\#epochs}{8}$ double the size till **batch-size of 128** is achieved.

Final Test Loss: 0.00337 Final Test Error: 0.0463

Final Test Accuracy: 1-0.0463= **95.37%**

Visualization of training process:

For the visualization of training process, we plot both the loss and the percentage error of classification for the training and the separate validation set.



References:

[1] Smith, Samuel L., et al. "Don't decay the learning rate, increase the batch size." *arXiv preprint arXiv:1711.00489* (2017).