Neural Network architecture for MNSIT Classification - Observations

1. Started with all 60,000 samples of the training data, the desired output labels were one hot encoded, i.e 5 => [0,0,0,0,0,1,0,0,0,0] etc. So a final layer of 10 input neurons were considered.
2. Considered unnormalized inputs, the weight updates were not processing and sometimes it ended up with math overflow, after classifying a few training examples the weight updates were 0. Then the input were normalized by subtracting the inputs of training data by the mean and dividing with its standard deviation - worked well as there were no 0 values.
3. Used weights that were normally distributed, the sigmoid function was exploding.
4. Used weights that were uniformly distributed between -1,1 the network performed well.
5. Sigmoid activation function was used for all layers, as its non-linear. Tried with the tanh activation function as well, close to similar results were found. Avoided ReLU activation function as it's a linear activation function. Sigmoid also gives probabilistic values between 0 and 1.
6. Tried with a single layer network with 100 neurons in hidden layer and sigmoid activation function at 1st hidden layer and output layer, the accuracy achieved for training was at most 75-80%. then another hidden layer was added to improve the accuracies.
7. First 200 neurons was considered for first hidden layer and 100 neurons for 2nd hidden layer, each epoch used to take a significant amount of time to run as there were too many weight calculations. The accuracies with this network architecture increased compared to the network with just a single layer of hidden neurons.
8. Tried to reduce the number of neurons in the 1st hidden layer to 100, similar results were observed compared to the 200 neurons and the time taken per epoch was reduced significantly. So considered just 100 neurons at the first and second hidden layers.
9. Started with a learning rate of 10, at each epoch the number of misclassification kept oscillating, indicating the gradient was oscillating and for a few epochs the accuracy decreased and then increased. The learning rate of 10 was very high.
10. The learning rate was reduced to 1, the network performed better compared to learning rate = 10,The experiment was repeated with learning rate = 0.01 and the network started performing better but it took more number of epochs to converge as the weight updates were very slow, after experimenting with different values a value of 0.1 was considered as the best parameter for learning rate.
11. First the algorithm was performed without using any regularization methods, since there was larger set training examples available, no tricks were added, looking at the number of epochs and accuracy at each epoch, to avoid more computations, no regularization or tweaks were added to the network, the network performed well using just the basic architecture.
12. The most basic energy method i.e Euclidean distance measure was used to calculate the energy at each epoch of the iteration.(Occam's Razor)
13. After performing the experiments many times, the number of epochs were limited to 200 or the epochs were terminated once a 99% accuracy was obtained on train data.
14. Finally the neural network architecture will be
    a. Input layer: 784 neurons
    b. First hidden layer: 100 neurons -> bias + weights (initially random uniform values between -1&1) -> Sigmoid activation function
    c. Second hidden layer: 100 neurons -> bias + weights (initially random uniform values between -1 and 1) -> Sigmoid activation function
    d. Output layer: 10 neurons -> Sigmoid activation function.
    e. Learning rate: 0.1
    f. threshold: 99% accuracy on training data

Pseudo-code to implement a Neural Network for MNSIT Classification

a. Let n = 60,000 be the number of input training samples, each with the dimensions 784 * 1
b. Let N = 10,000 be the number of input test samples, each with dimensions 784 * 1
c. Let D = 60,000 be the labels for the training input samples, the labels one hot encoded with dimension 10 * 1
d. Let D_test = 10,000 be the labels for the test input samples, the labels one hot encoded with dimension 10 * 1
e. Let there be 784 neurons in the input layer, 100 neurons in the first hidden layer, 100 neurons in the second hidden layer and 10 neuron in the output layer. we wish to find (100 * 785) weights in the first hidden layer, (100 * 101) weights at the second hidden layer and (10 * 101) weights at the output layer which are including the bias at each layer.
f. The train and test inputs are standardized by subtracting with its mean and dividing my the standard deviation.
g. The algorithm will be:
   1. Initialize $W_1 \in \mathbb{R}_{(100 \times 785)}$ including the bias, where weights and bias are uniformly distributed between (-1,1)
   2. Initialize $W_2 \in \mathbb{R}_{(100 \times 101)}$ including the bias where bias and weights are uniformly distributed between (-1,1)
   3. Initialize $W_3 \in \mathbb{R}_{(10 \times 101)}$ including the bias, where weights and bias are uniformly distributed between (-1,1)
   4.
      4.1 Let D = $[d_1,d_2,....,d_n]$, n = 60,000 be the set of desired output (labels) for the training data
      4.2 Let D_test = $[d_1,d_2,....,d_n]$, n = 10,000 be the set of desired output (labels) for the test data
   5. Initialize $\eta = 0.1$ and $\varepsilon = 0.01$
   6. Initialize epoch = 0
   7. Initialize errors_train [epoch] = 0 and errors_test[epoch] = 0 for epoch in 0,1,2....

   8. Do (This loop is where we iterate the algorithm until convergence…)

      8.1 for i in 1 to n (60000) do (This loop is where we perform the forward and backward propagation on training data)
      ---Forward Propagation---
      8.1.1 Calculate the induced local field v_1 with current input samples and weights W_1. i.e $v_1 = W_1*[1,x_i] \in \mathbb{R}_{(100 \times 1)}$, where $W_1 \in \mathbb{R}_{(100 \times 785)}$ and $x_i \in \mathbb{R}_{(785 \times 1)}$ is i_{th} training sample
      8.1.2 The neurons in the first hidden layer will use the sigmoid(v) activation function. Let $y_1 = \Phi(v_1)$ where $\Phi(v_1) = \text{sigmoid}(v_1)$, here y_1 is the output of the neuron in the hidden layer
      8.1.3 Calculate the induced local field v_2 with inputs from first hidden layer and weights W_2. i.e. $v_2 = W_2*[1,y_1] \in \mathbb{R}_{(100 \times 1)}$, where $W_2 \in \mathbb{R}_{(100 \times 101)}$ and $y1 \in \mathbb{R}_{(101 \times 1)}$, where y_1 is the input from the first hidden layer.
      8.1.4 The neurons in the second hidden layer will use the sigmoid(v) activation function. Let $y_2 = \Phi(v_2)$ where $\Phi(v_2) = \text{sigmoid}(v_2)$, here y_2 is the output of the neuron in the hidden layer
      8.1.5 Calculate the induced local field v_3 with inputs from second hidden layer and weights W_3. i.e. $v_3 = W_3*[1,y_2] \in \mathbb{R}_{(10 \times 1)}$, where $W_3 \in \mathbb{R}_{(10 \times 101)}$ and $y1 \in \mathbb{R}_{(101 \times 1)}$, where y_2 is the input from the second hidden layer.
      8.1.6 The neurons at the output layer will use the sigmoid activation function. Let $y_3 = \Phi(v_3)$ where $\Phi(v_3) = \text{sigmoid}(v_3)$, here y_3 is the final output of the network = f(x,w).

8.1.7 Define Energy $E = 1/2 \|D_i - y_3\|_2$,where D_i is desired response for the same and y_3 is the output obtained by the network.

8.1.8 Calculate the number of misclassification for the training data by comparing it with Desired response i.e (if $D_i = Y3$).

---Back Propagation---

8.1.9 Calculate the signal at the output using $\delta_3 = (D_i - y_3) * \Phi'(v_3)$ where $\Phi'(v_3)$ is the derivative of the function $\Phi(v_3)$ and D_i is the desired output for input x_i and y_3 is the actual output of the network

$\Phi'(v_3) = \partial\Phi(v_3)/\partial v = (\text{sigmoid}(v_3)) * (1 - \text{sigmoid}(v_3))$

8.1.10 Calculate the signal at the output of 2nd hidden layer $\delta_2 = (W_3 * \delta_3) * \Phi'(v_2)$ where $\Phi'(v_2)$ is the derivative of the function $\Phi(v_2)$ i.e.

$\Phi'(v_2) = \partial\Phi(v_2)/\partial v = (\text{sigmoid}(v_2)) * (1 - \text{sigmoid}(v_2))$

8.1.11 Calculate the signal at the output of 1st hidden layer $\delta_1 = (W_2 * \delta_2) * \Phi'(v_1)$ where $\Phi'(v_1)$ is the derivative of the function $\Phi(v_1)$ i.e.

$\Phi'(v_1) = \partial\Phi(v_1)/\partial v = (\text{sigmoid}(v_1)) * (1 - \text{sigmoid}(v_1))$

8.1.12 Calculate the partial derivatives of the energy function with respect to the weight

8.1.12.1 $\partial E/\partial W_1 = -\delta_1 * [1,x]_T$.

8.1.12.2 $\partial E/\partial W_2 = -\delta_2 * [1,y_1]_T$.

8.1.12.3 $\partial E/\partial W_3 = -\delta_3 * [1,y_2]_T$.

--Update Weights---

8.1.13 Update the weights W_1 , W_2 and W_3 based on the eta provided and the partial derivatives calculated earlier

8.1.13.1 $W_1 <- W_1 + (\eta * (\delta_1 * [1,x]_T))$

8.1.13.2 $W_2 <- W_2 + (\eta * (\delta_2 * [1,y_1]_T))$

8.1.13.3 $W_3 <- W_3 + (\eta * (\delta_3 * [1,y_2]_T))$

8.2 for i in 1 to n (10000) do (This loop is where we perform the forward and backward propagation on test data)

8.2.1 Calculate the induced local field v_1 with current input samples and weights W_1. i.e $v_1 = W_1*[1,x_i] \in \mathbb{R}_{(100 \times 1)}$, where $W_1 \in \mathbb{R}_{(100 \times 785)}$ and $x_i \in \mathbb{R}_{(785 \times 1)}$ is i_{th} testing sample

8.2.2 The neurons in the first hidden layer will use the sigmoid(v) activation function. Let $y_1 = \Phi(v_1)$ where $\Phi(v_1) = \text{sigmoid}(v_1)$, here y_1 is the output of the neuron in the hidden layer

8.2.3 Calculate the induced local field v_2 with inputs from first hidden layer and weights W_2. i.e. $v_2 = W_2*[1,y_1] \in \mathbb{R}_{(100 \times 1)}$, where $W_2 \in \mathbb{R}_{(100 \times 101)}$ and $y1 \in \mathbb{R}_{(101 \times 1)}$, where y_1 is the input from the first hidden layer.

8.2.4 The neurons in the second hidden layer will use the sigmoid(v) activation function. Let $y_2 = \Phi(v_2)$ where $\Phi(v_2) = \text{sigmoid}(v_2)$, here y_2 is the output of the neuron in the hidden layer

8.2.5 Calculate the induced local field v_3 with inputs from second hidden layer and weights W_3. i.e. $v_3 = W_3*[1,y_2] \in \mathbb{R}_{(10 \times 1)}$, where $W_3 \in \mathbb{R}_{(10 \times 101)}$ and $y1 \in \mathbb{R}_{(101 \times 1)}$, where y_2 is the input from the second hidden layer.

8.2.6 The neurons at the output layer will use the sigmoid activation function. Let $y_{3(test)} = \Phi(v_3)$ where $\Phi(v_3) = \text{sigmoid}(v_3)$, here $y_{3(test)}$ is the final output of the network = f(x,w).

8.2.7 Define Energy $E = 1/2 \|D - y_{3(test)}\|_2$, where d is desired response for the same and $y_{3(test)}$ is the output obtained by the network.

8.2.8 Calculate the number of misclassification for the training data by comparing it with Desired response i.e (if D_test = Y_3(test)).

8.3 Calculate the Mean Squared Error using the formula MSE = $((D_i - Y_{3i}) ** 2 / n)$ for i in 1,2,3... n where D is the set of desired values and Y_3 is the set if output values obtained in 8.1.6 for the set of training examples.

8.4 Calculate the Mean Squared Error using the formula MSE = $((D\_test_i - Y_{2i}) ** 2 / n)$ for i in 1,2,3... n where D is the set of desired values and Y3 is the set if output values obtained in 8.2.6 for the set of testing examples.

8.5 The algorithm may not always result in a monotonically decreasing MSE, if you find the MSE has increased from the previous iteration i.e.
MSE[epoch] > MSE[epoch -1] then decrease the learning rate η <- η * 0.9

8.6 epoch <- epoch + 1

8.7 Loop to 8 until Number of misclassifications (obtained from 8.1.8) <= ε

---Algorithm Converges---

9. Return the Number of misclassifications and energy values at each epoch and the final weights W_1 , W_2 and W_3

h. Plot the epochs v/s misclassifications for both training and testing images.

i. Plot the epochs v/s Energy values for both training and testing images.

j. The plots in [h] and [i] should have decreasing misclassifications and energy values as we are improving the network performance at each iteration by updating the weights.

By looking at the plots we can see that the error rate and the energy values for both training and test data decreases at each iteration.

Finally, after the algorithm converges, we can see that we have achieved a 99% of training accuracy and 96% of test accuracy using the above mentioned neural network architecture and hyper-parameters.