Name: Pratik Walawalkar

UIN: 667624808

**ECE 559: Homework 5 and 6**

Answer 1.:

* Pseudocode:

1. Take “x” input vectors of sample size 300 distributed uniformly on [0,1]
2. Take “v” vectors of sample size 300 distributed uniformly on [-0.1,0.1]
3. Calculate corresponding 300 desired output vectors d[i]= sin(20\*x[i])+(3\*x[i])+v[i]) where i->1 to 300
4. Consider two-layer neural network of order 1X24X1 neurons
5. Initialize weights as:
   * 1st layer:

* 24 Input bias weights: wb1N distributed uniformly on [-1,1] initially
* 24 Input weights: w1N distributed uniformly on [-1,1] initially
  + 2nd layer:
* 1 Output bias weights: wb2N distributed uniformly on [-1,1] initially
* 24 Output weights: w2N distributed uniformly on [-1,1] initially

1. Run while loop with initial epoch=0
   1. Run for loop for 300 input vectors

6.1.1) Do forward propagation and calculate “y” output for every “x” input and above initialized weights.

**Y=wb2N+ (w2N[i]\*tanh(wb1N[i]+(w1N[i]\*x[j])))** where i->1 to 24 hidden neurons and j->1 to 300 input vectors

6.1.2) Using Backpropagation we calculate:

* + - * Final output: y=f(v2)=v2(No activation function)
      * Local induced field for 2nd layer: v2=wb2N+z[i]\*w2N[i] where i-> 1 to 24
      * Output from 1st layer: z[i]=tanh(v1[i]) where i-> 1 to 24
      * Local induced field for 1st layer: v1[i]= wb1N[i]+x\*w1N[i] where i-> 1 to 24
      * Delta1=Delta2\*w2N\*(1-tanh2(v1[i])) where derivative of tanh(x)=1-tanh2(x)
      * Delta2=-2(d-y)\*1 where is -2(d-y) derivation of cost function

6.1.3) Weight update equation will be:

* + 1st layer:
* Input bias weights: **wb1N = wb1N – training parameter\*Delta1\*1**
* Input weights: **w1N = w1N - training parameter\*Delta1\*x**
  + 2nd layer:
* Output bias weights: **wb2N = wb2N – training parameter\*Delta2\*1**
* Input weights: w2N = **w1N - training parameter\*Delta2\*z1**

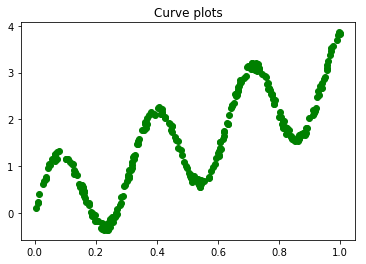
6.1.4) Calculate MSE as MSE = ( (d-y)2)/300

* 1. Epoch=Epoch+1
  2. Assign threshold=MSE to terminate while loop. This is done to minimize MSE. In code MSE is taken as 0.01 to terminate while loop

1. After certain epochs when MSE equals to 0.01, take the updated weights(w\_upd) and calculate

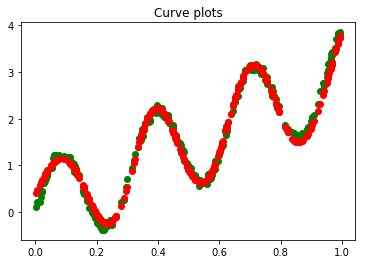
**f(x,w\_upd)= wb2N\_upd+ (w2N\_upd[i]\*tanh(wb1N\_upd[i]+(w1N\_upd[i]\*x[j])))** where i->1 to 24 hidden neurons and j->1 to 300 input vectors

* Plots/Observation:

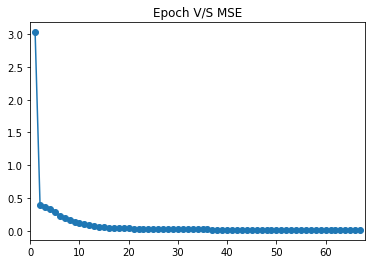


Plot for x v/s d

We get good curve fitting as below. Green curve for d and red curve for f(x,w)



Plot for x v/s f(x,w)



Plot for epochs v/s MSE

MSE saturates for value of 0.012 after 30 epochs. Hence MSE threshold is kept as 0.01 and we get good curve fitting for this minimum MSE value

* Conclusion:
* Initially when weights were chosen between [-1,1] we would get convergence after very large epochs (3000+)
* Hence, with the help of backpropagation we reverse engineered the network and found out minimum and maximum value of each layer weights. So, we initialize the weights by choosing the new range and tried running algorithm. This reduced the number of epochs required to converge (epoch range of 60-150)
* **It was clear that depending upon the choice of initial weights closer to optimal ones, the backpropagation method would yield faster convergence with minimum cost function.**
* Choice of weights given after series of experiments:
  + 1st layer:
* 24 Input bias weights: wb1N distributed uniformly on [-5.48, 6.89] initially
* 24 Input weights: w1N distributed uniformly on [-9.61, 10.09] initially
  + 2nd layer:
* 1 Output bias weights: wb2N as 0.6
* 24 Output weights: w2N distributed uniformly on [4.29,6.18] initially

We can observe that **weights are higher at input layer and lower at output layer**. This is because there is **larger local gradient at output layer** than the ones in input/hidden layers. Hence to maintain faster learning rate we must choose distinct set of weights for different layer. This is also true for learning parameter choice.

Answer 2.

* Architecture/Design & Choice of hyperparameter:
* Network topology: Consider two-layer neural network of order 784 X 50 X 10 (input X hidden X output) neurons.

**Reason: 1 layer with 50 hidden neurons are sufficient for digit classification. As we increase hidden neurons/layer, the training complexity increases which require more time for convergence at the cost of slight increase in training accuracy.**

* Digits represented in output layer is of 10 output neurons i.e. [0100000000] as 1 and so on
* Neuron activation function is taken as **hyperbolic tangent function** for both layers.

**Reason**: **tanh is odd sigmoid function which yields in faster speed of learning. Also at origin its slope is close to unity which makes derivative of function easy to calculate.**

* Energy function is taken as (1/n) (d-y)2

**Reason**: **To minimize mean square function** **to attain minimal error rate and optimal final weights after each update used in online learning. Also for larger dataset online learning is implemented.**

* Other tricks used:

1. **Different values of training parameters** are incorporated at each layer. Higher learning parameter is given at 1st layer weights and lower values at 2nd layer weights

**Reason: All neurons in the multilayer network should ideally learn at the same rate. The last layer usually has larger local gradient than the layer at the front end of the network.** [1]

**2. Normalizing inputs:** Inputs are normalized by factor of 1/255 for each pixel which gray scale value ranges from 0 to 255.

**Reason:** **This is done in to get zero mean value, averaged over the entire training sample. Also, when these inputs are used in along with the hyperbolic tangent function the variance of the individual neural outputs in the multilayer network will be close to unity.** [1]

[1] *Reference: Neural Networks and Learning Machines Third Edition Simon Haykin)*

* Design Process:
* Initial weights were taken uniformly between range of [-1,1]. This approach however failed as the values of delta 1 and delta 2 were either too large or too small which would diverge the algorithm very quickly. In some case it would also increase the initial error rate. Hence lower weights in range of [-0.1,0.1] were taken.
* Large learning parameter=1 was taken. This configuration would also diverge the network quickly. Hence lower rate=0.01 was taken to get finite range of updated weights.
* Pseudocode: Initialize weights as:
  + 1st layer:
* 784 X 50 Input weights: w1N distributed uniformly on [-1,1] initially
  + 2nd layer:
* 50 X 10 Output weights: w2N distributed uniformly on [-1,1] initially

1. Pass these weights to train\_backprop function:

2.1) Import training\_label file and read labels and store in d\_list. Also express respective d\_list element in 10X1 matrix with output = 1 for that index and 0 for rest. Name this matrix des.

2.2) Run while loop with initial epoch as 0

2.2.1) Import and read training\_images file with 784 pixels(28X28) for each image. Assign this as matrix x of order 1X784

2.2.2) Do forward propagation and calculate “y” output for every “x” input and above initialized weights. Equations we get are as follows:

* y=tanh(v2) of matrix order 10X1
* v2=z\_T. w2N of matrix order 10X1
* z=tanh(v1) of matrix order 50X1
* v1=w1N.x\_T of matrix order 50X1

2.2.3) Take maximum magnitude of y matrix as ymax

2.2.4) Using Backpropagation we calculate:

* + - * Delta2= -2\*(des-y) \*(1-tanh2(v2)) of matrix order 10X1 where derivative of tanh(x)=1-tanh2(x)
* Delta1= Delta2\*w2N\*(1-tanh2(v1))

2.2.5) Weight update equations will be:

* + 1st layer:
* w1N= w1N – training parameter\*z\*Delta2\_T
  + 2nd layer:
* w2N= w2N – training parameter\*x\*Delta1

2.2.6) Calculate MSE as MSE = ( (d-y)2)/300

2.2.7) If d\_list is not equal to ymax then there will be classification error. So, error=error+1

2.3) Assign threshold=error/dataset to terminate while loop. This is done to minimize misclassification and achieve higher accuracy for training images network. In code threshold is taken as 0.05 or 5% error to terminate while loop

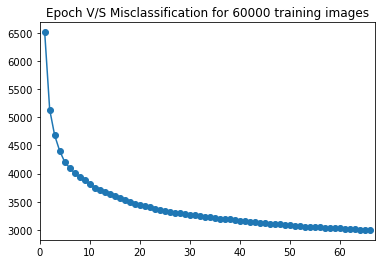
3) After certain epochs and success rate pass update final weights (w1N\_upd and w2N\_upd) to test\_backprop function with test\_image and label file to read and check the final success rate of the network.

* Plots/Observation:

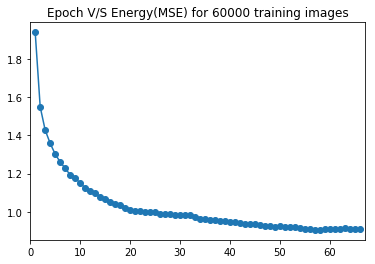
For 60000 training images network we trained with success rate 95%. We observe convergence at epoch: 66 with errors: 2999 and final energy(MSE): 0.9071666666666667

When the updated weights from training set were given to 10000 test images we get final success rate as **94.08** with errors **592**

This concludes that if training set is trained with correct hyperparameters and minimum error rate, the test set will yield higher success rate depending upon choice of hidden layer and neurons. Also backpropagation helps achieve better success rate and easy convergence as well.



Plot for error v/s misclassification for training set



Plot for error v/s Energy for training set

From above graphs its observed that there is a faster learning rate till epoch 20. After that the speed of convergence is slow and gradual and depends upon what error threshold is taken for the convergence