CSE474/574: Introduction to Machine Learning(Fall 2012)
Part 2: Classification on Hand Writing Digits

1. PROCESSING DATA:
   - Extracted Binary GSC features: The data set is divided in two parts: Training set and Test set. Size of training set matrix is 20000 x (512+1), which constitutes the feature vector (20000 x 512) and Class labels (20000 x 1). Size of test set matrix is 1500 x (512+1), which constitutes the feature vector (1500 x 512) and Class labels (1500 x 1).
2. IMPLEMENTING FOLLOWING CLASSIFIERS:
   - Logistic Regression
   - Neural Networks

LOGISTIC REGRESSION:

1. **Training**:
   Divide the training set into two parts: training data and validation data. I took 80% of train data for every class (k=1 to 10) as training set and remaining for the validation part (using the import function of matlab). The data given here is for 10 classes (k=10), we need to classify the data (test data) into 10 classes using appropriate model. For this, we compute the following steps using training data to set up our model for classifying the data:

   a) To calculate the posterior probability:
   The posterior probability for multiclass is given by a softmax transformation of linear functions of the feature variable (Ø),

$$p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

   where,
   P(Ck/Ø) is the posterior probability, and ak is the activation, which is given by:

$$a_k = \mathbf{w}_k^{\mathrm{T}}\phi.$$

   where, Ø is our feature vector.

   b) To determine Error:

To test the data, we need to train our model by finding optimized values of the model parameters, in order to get low error. So first we need to calculate error:

This is done by using likelihood function which is given by:

$$p(\mathbf{T}|\mathbf{w}_1,\ldots,\mathbf{w}_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(\mathcal{C}_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

where ynk=yk(φn), and T is an N×K matrix of target variables with elements tnk. Initially I took the value of wk=0.1 (we can choose any initial value, since we are going to train our model to get the optimized value)

Taking the negative logarithm of the likelihood function will give us the error of the training data for k=10:

$$E(\mathbf{w}_1,\ldots,\mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1,\ldots,\mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln y_{nk}$$

Using the above equation, I got the normalized initial error=2.306
P.S. normalized initial error is the per sample error i.e. (error/no of data points)

c) To determine optimized value of parameter {wk}:

To get low error, we need to train our model by finding optimized values of the model parameters. One of the parameter being {wk}, to calculate {wk}, we consider the use of maximum likelihood Newton-Raphson iterative optimization scheme and gradient descent optimization.

The error function can be minimized by an efficient iterative technique based on the Newton-Raphson iterative optimization scheme

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}).$$

Since Hessian is implemented, I implemented Gradient descent optimization to update the weight. I choose the weight update (using formula given below) to comprise a small step in the direction of the negative gradient, where the parameter η>0 is known as the learning rate

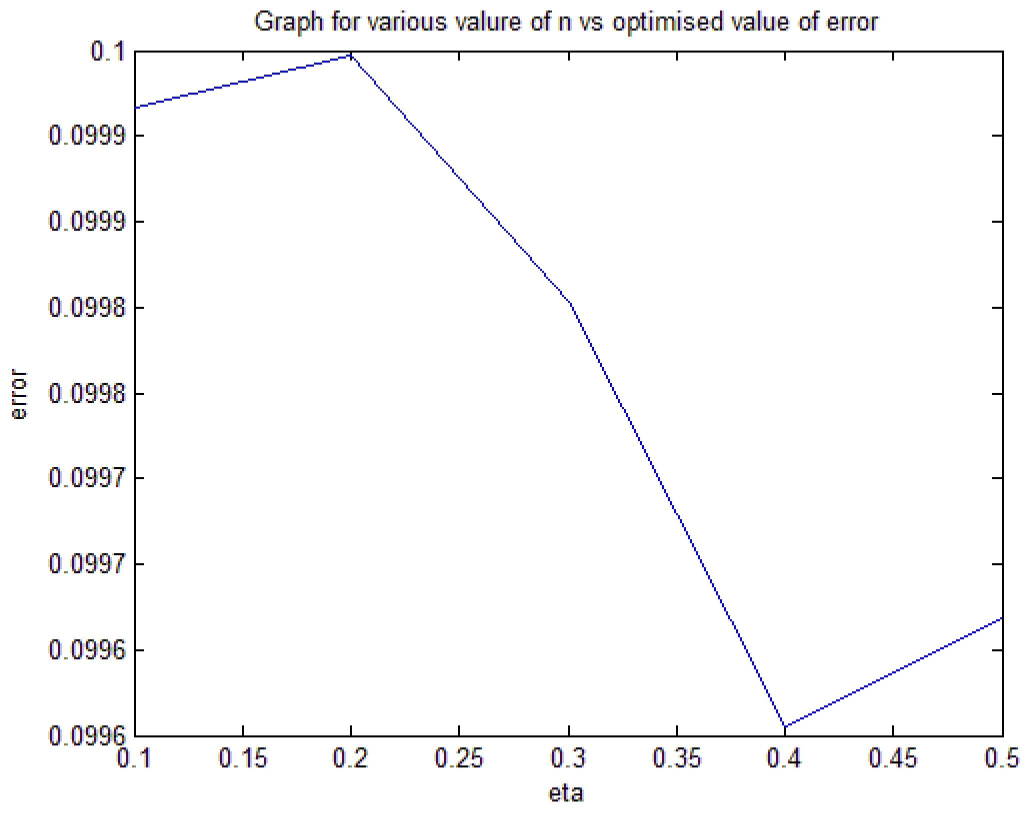
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

After each such update, the gradient is re-evaluated for the new weight vector and the process repeated. The gradient error is given by:

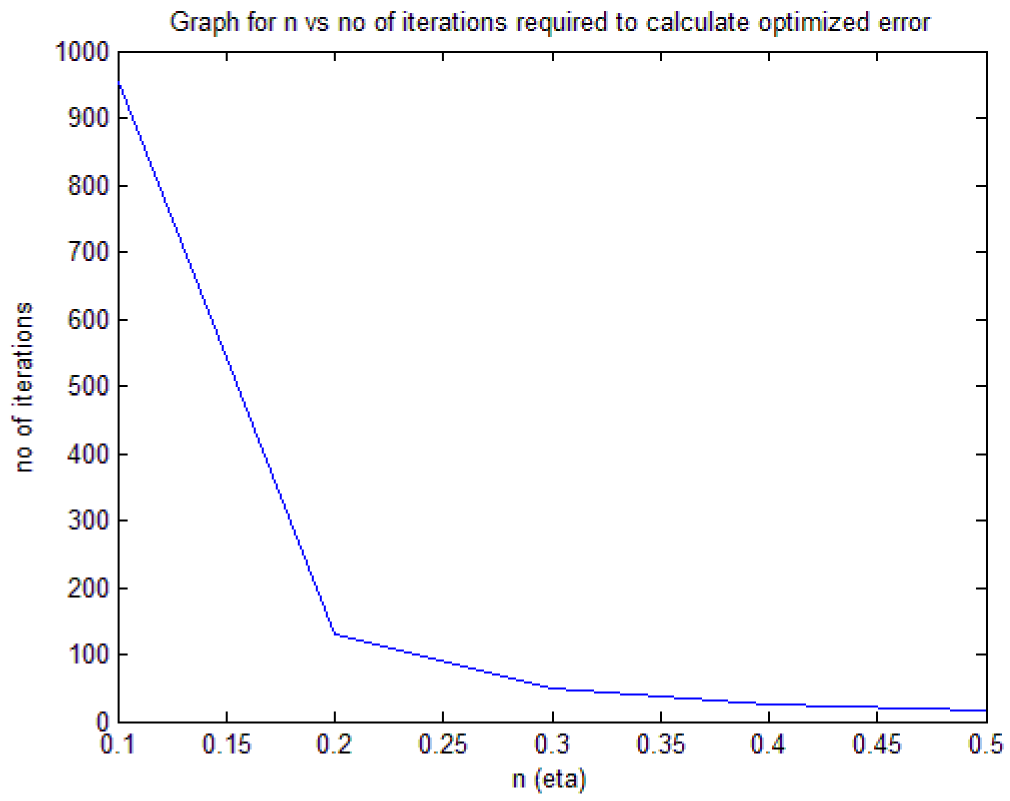
$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\, \boldsymbol{\phi}_n$$

I have taken initial value of w^{old} as 0.1 and n (learning rate) as a set of value and calculated w^{new} repeatedly until I get lowest value of error.

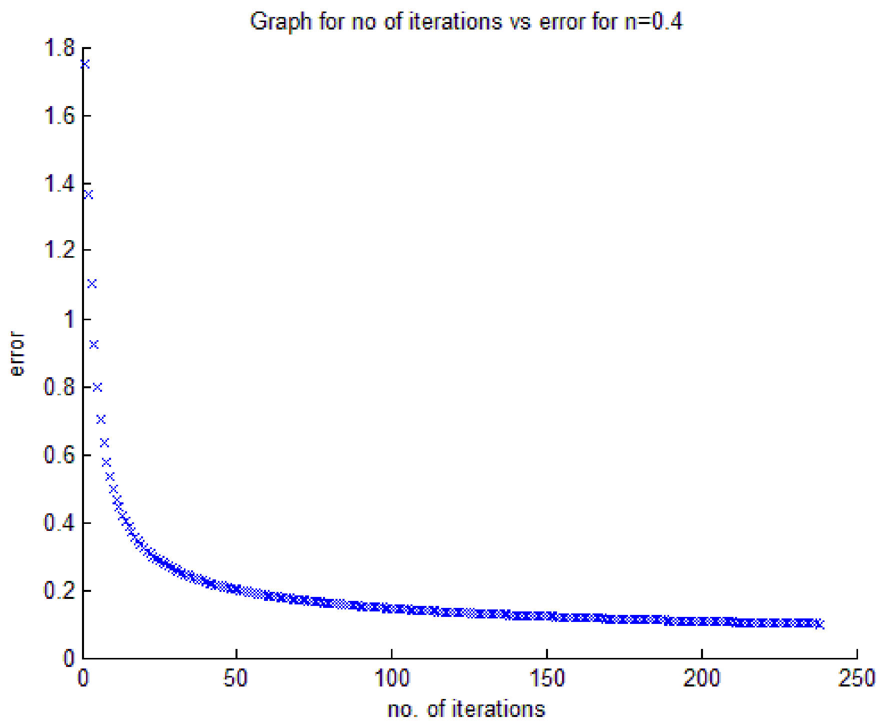
I have used the stopping condition as:
currenterr>0.1|| (preverr-currenterr)>0.001 || (currenterr-nexterr)>0.001
So, using different values of n, I calculated the optimized value of w i.e. for which I get lowest error, and also it satisfies the above condition.
I selected that value of n for which I get the least error and least number of iterations.
Following graphs are attached to justify the selection of n (learning rate).



The above graph shows the different value of n vs the error value calculated (using optimized value of w). So, for n=0.4 I got the least value of error. Also, the next graph shows the variation for different value of n vs number of iteration required for calculating least value of error. It shows that for n=0.4, we need to compute less number of iterations. So taking n=0.4 as our final value for calculating w and computing error.



Below graph shows the plot for finding the least value of error, using n=0.4.



The above graph is obtained by plotting error (y-axis) for every iterations (x-axis).

2. **Validation:**

   Using the optimized value of n and w, I have validated the validation data.

   a) First, I calculated the posterior probability using softmax transformation:

   $$p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

   b) Second, I assigned class labels to the validation data using the above probability. For every data point, the class which is having highest probability is assigned as the class for that data point.

   c) Third, to check the correctness, I have calculated the accuracy i.e. the rate of classification (number of data points classified correctly). I compared the value of predicted class label (as calculated above) with the given class label (training label, and found number of matching points, thus giving the number of hits. Dividing the number of hits by total number of data points give us the accuracy.

   I got the value of **ACCURACY= 97.52%,** thus validated the data using optimized value of w and n.

3. **Testing:**

   Using the testing data, I tested my model by predicting the class labels.

   d) First, I calculated the posterior probability:

   $$p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

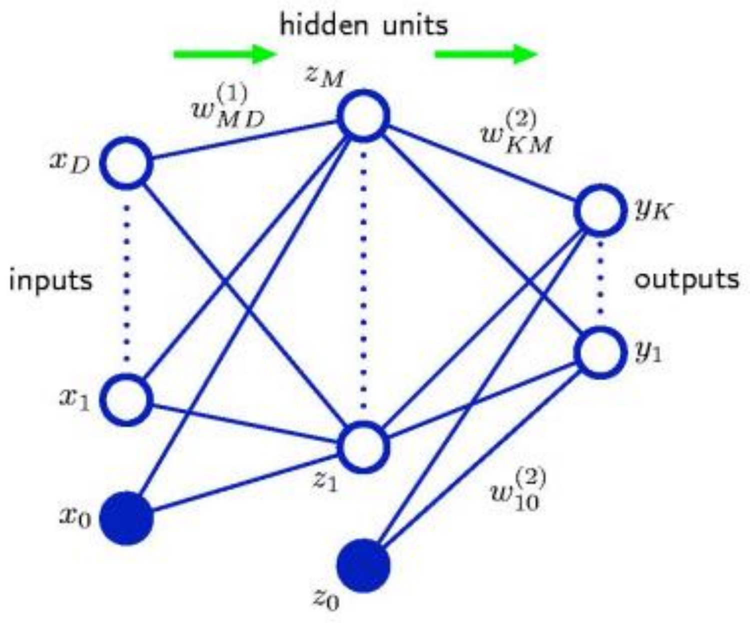
   e) Second, I assigned class labels to the validation data using the above probability.

   f) Third, I have calculated the accuracy i.e. the rate of classification (number of data points classified correctly).

   I got the value of **ACCURACY= 97.4667%**

NEURAL NETWORK:

Our goal in this section is to find an efficient technique for evaluating the gradient of an error function E(w) for a feed-forward neural network.



Can be viewed as a generalization of linear models

$$y_k(\mathrm{x},\mathrm{w}) = \sigma\left(\sum_{j=1}^{M} w^{(2)}_{kj} h\left(\sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0}\right) + w^{(2)}_{k0}\right)$$

I implemented Neural network using above formula:

Used the following feed forward network functions

$$y(\mathrm{x},\mathrm{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathrm{x})\right)$$

Activation function

Basis function

Basis function is implemented using tanh function and activation function is implemented using the sigmoid function as shown below:

$$f(a) = \frac{1}{1+e^{-a}}$$

1. **Training (Approach 1):**

   Divided the training data into two parts: training data and validation data:

   1) Initialized w1 and w2 to 0.1
   2) First we calculate the z i.e. first layer output
      z=tanh(w1*train_data);
   3) Calculate the output layer y using the sigmoid function:
      y=1 / (1+exp(- (w2*z)));
      where z is calculated as above
   4) Calculate the error using the sum-of-squared errors:
      Error=1/2 * ∑ (y - t) ^ 2
   5) Normalized the error for calculating per sample error
   6) Calculate the derivative of E wrt to w1
      $\partial E / \partial w2 = \partial k * \partial a_{nk} / \partial w2$
   7) Calculate the derivative of E wrt to w2
      $\partial E / \partial w1 = \partial j * \partial a_{nj} / \partial w1$
      where
      $\partial j = \partial En / \partial a_{nj} = \partial En / \partial a_{nk} * \partial a_{nk/} \partial a_{nj}$
   8) Assuming n=0.1 (learning rate)
   9) Using the gradient descent rule for finding the updated values of w.

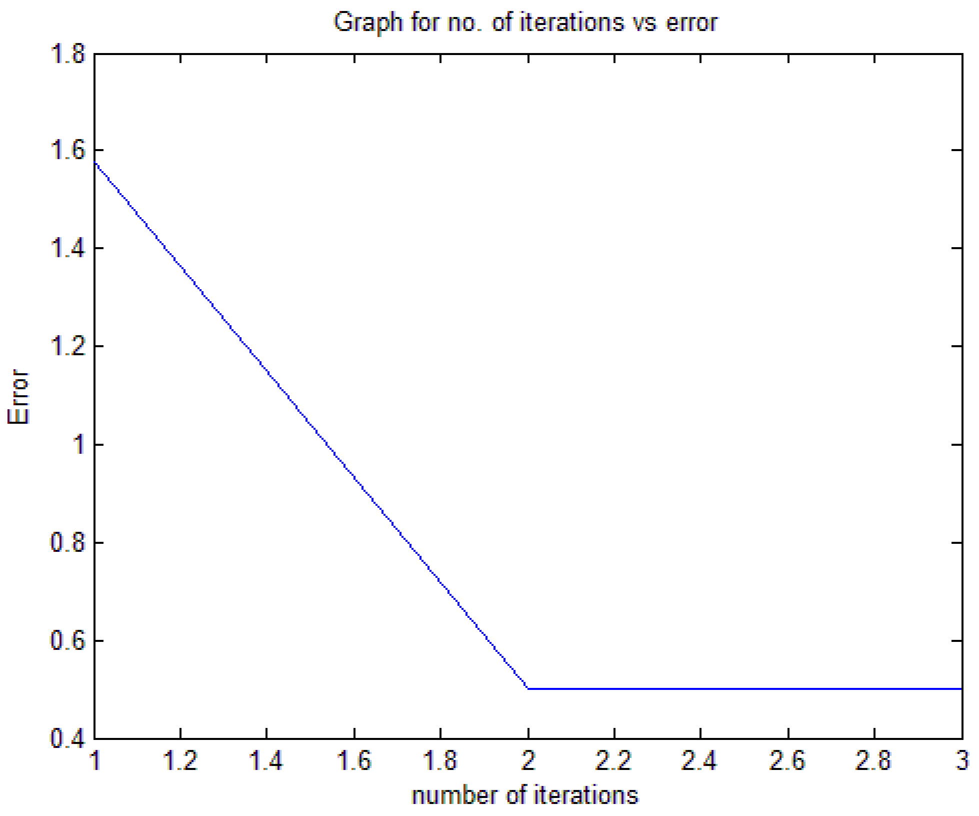
   $$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

   Using this I got the error as showed in the graph (below) for number of iterations vs error. I got initial error=1.5755 (normalized error), and then error got reduced and became constant to 0.5

2. **Validation (Approach 1)**
   - **a.** Using the optimized value of w1 and w2, I calculated first layer and second layer output using the equations given above in 2 and 4 step of training.
   - **b.** Using y, I assigned the class labels to the validation data and calculated the Accuracy of the classification, based on number of hits. Accuracy came out to be 80%

3. **Testing (Approach 1)**
   1. Using the optimized value of w1 and w2, I calculated first layer and second layer output using the equations given above in 2 and 4 step of training.

2. Using y, I assigned the class labels to the validation data and calculated the Accuracy of the classification, based on number of hits. Accuracy came out to be 80%



Approach 2

Used Softmax function at the output layer:

1. **Training**
    1. Initialized w1 and w2 to 0.1
    2. First we calculate the z i.e. first layer output
       z=tanh(w1*train_data);
    3. Calculate the output layer y using the Softmax function:
       $y=\exp(a_k) / \sum (\exp(a_j));$
    4. Calculate the error using the cross entropy error:
       $\text{Error}= - \sum \sum t_{nk} \ln y_{nk}$
    5. Normalized the error for calculating per sample error
    6. Calculate the derivative of E wrt to w1
       $dE/dW2 = - (t_{nk} / y_{nk)} * ((\text{Sum}_{j*} e^{ak}) – (e^{ak} * e^{ak}) / (\text{Sum}_j))* Z_j$
       where, $\text{Sum}_j = \sum e^{aj}$
    7. Calculate the derivative of E wrt to w2
       $dE/dW1 = -(t_{nk} / y_{nk)} * ((\text{Sum}_{j*} e^{ak}) – (e^{ak} * e^{ak}) / (\text{Sum}_j))* W2 * (1-Z_j^{2}) * X;$

8. Assuming n=0.1 (learning rate)
9. Using the gradient descent rule for finding the updated values of w. Using this I got the error as showed in the graph (below) for number of iterations vs error. I got initial error=2.3026 (normalized error) which remained constant for every update step performed

2. **Validation**
   1) Using the optimized value of w1 and w2, I calculated first layer and second layer output using the equations given above in 2 and 4 step of training (approach 2).
   2) Using y, I assigned the class labels to the validation data and calculated the Accuracy of the classification, based on number of hits. Accuracy came out to be 80%

3. **Testing**
   1) Using the optimized value of w1 and w2, I calculated first layer and second layer output using the equations given above in 2 and 4 step of training.
   2) Using y, I assigned the class labels to the validation data and calculated the Accuracy of the classification, based on number of hits. Accuracy came out to be 80%

CONCLUSION:

We classified the data using logistic regression with an accuracy of 97.46%, however using neural network I got the accuracy as 80%.

REFERENCES:

Images and some equations were taken from the pattern recognition and machine learning book written by Christopher Bishop.