Recognizing Hand-Written Digits

Raviteja Anantha Ramesh(raviteja.anantha@nyu.edu, rar555),
Karthik Tiruveedhi(karthik.tvs@nyu.edu, vst216),
Ganapathy Aiyer(ganapathy.aiyer@nyu.edu, gsa277)

Section-1

Objective:

We implemented Neural Net and KNN for recognizing hand written digits. We use the accuracy of the Neural Net to see how it compares with SVM, implementation of sklearn library with Radial Basis Function being the Kernel Function. We used MNIST data to train our models. We chose KNN and Neural Net because they both are very different, KNN doesn't compute any model and it is lazy, on the other hand Neural Net learns from the training-data and computes a model. So we can see how these two entities perform. Detailed description of Neural Net and KNN are given in the following sections. We also created a User Interface where the user can draw the digits and can see the how the Neural Net and SVM classify it.

Section-2

Neural Net:

After researching the literature on the problem, we found the problem is solved by many researchers using Neural Nets[1]. We chose MNIST data for training and testing purposes. As we have a large dataset, using Gradient descent may take too long because in every iteration when we are updating the values of the parameters, we are running through the complete training set. On the other hand, using Stochastic Gradient Descent[2] will be faster because we use only a subset(mini-batch) of training-data and it starts improving itself right away from the first mini-batch. Stochastic Gradient Descent often converges much faster compared to Gradient Descent but the error function is not as well minimized as in the case of Gradient Descent. Often in most cases, the close approximation that you get in Stochastic Gradient Descent for the parameter values are enough because they reach the optimal values and keep oscillating there. We are using Backpropagation because it works faster for most of the Neural Nets[3][4][5] for computing the Gradient and updating weights. We also later compare out Neural Net performance on the test-data with how SVM by sklearn performs with Radial Basis Function as it's Kernel Function.

Network Architecture:

The Neural Net[6] we designed have one input layer, one hidden layer and one output layer. Deciding on the number of neurons in the input and output layer was not hard, the MNIST database contains images of 28x28 pixels and they are all handwritten digits(only 10 classes). So we decided to have 784(28x28) neurons in the input layer and 10 neurons in the output layer. But deciding on the hidden layer was very hard since there is no formal heuristic to follow. So we tried quite a few different numbers and decided it to have 10 neurons as it was giving the best results. We randomly initialized the weights and biases. We used sigmoid function as our activation function, we could have also used tanh function for the same purpose. For the Learning Rate(Eta), we tried the values in steps of 0.10 from 0.10 to 1.0, then went in steps of 1.0 till 5.0, and we noticed 3.0 was giving the best accuracy.

Results of Neural Net:

After training the Neural Net using the proposed algorithms, we found that our Neural Net results in 96% accuracy, we also tried using SVM of sklearn with Radial Basis Function as it's Kernel Function and found it results in 97% accuracy. We also note that the record for the accuracy with the MNIST database is 99.79%, and it was by a committee of 5 members and they used Convolutional Neural Net with 6 Layers and a lot of preprocessing.

KNN:

KNN is a lazy classification algorithm, wherein we calculate the distance of test image with each image in the training set. From the distances calculated we choose the k smallest distances and predict the class of test image based on the class which appears most in k smallest distances. This is a very computation and memory intensive task.

Since the training and testing set are images, we are not using mainstream distance measures like manhattan and euclidean distances. Below is the description of the distance measure we used:

1. Count the number of non zero pixels in a row for training image and test image.

2. Take the absolute value of the difference between the counts and store it in a vector.

3. Do the same for each row.

4. Count the number of non zero pixels in a column for training image and test image.

5. Take the absolute value of the difference between the counts and store it in another vector.

6. The dot product of the horizontal and vertical vector is the distance.

We chose this distance measure as we thought that each digit has a shape and each shape will have the same number of horizontal and vertical pixels.

Results of KNN:

As this is very computation and memory intensive we couldn't use the entire training data set (60k samples) and testing data set(10k samples). However some test results are as follows:

1. 5k train set, and 10 test images - Accuracy - 80%
2. 10k training set and 100 test images - Accuracy - 85%

References:

[1] Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition, Yann LeCun, L. D. Jackel, Leon Bottou, Corinna Cortes, John S. Denker, Harris Drucker, Isabelle Guyon, Url A. Muller, Eduard Sacking, Patrice Simard, and Vladimir Vapnik

[2] Stochastic Gradient Descent Methods for large scale pattern classification, Saswata Chakravarty, Indian Institute of Science

[3] Learning representations by back-propagating errors, David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams

[4] Efficient Backprop, Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Muller

[5] Learning representations by back-propagating errors, David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams

[6] Comparing different neural network architectures for classifying handwritten digits, Yan LeCun

[7] Modeling the manifolds of images of handwritten digits, Geoffrey Hinton, P. Dayan, M. Revow

[8] Learning sets of filters using back-propagation, David C. Plaut, Geoffrey E. Hinton