**Report describing implementation details and presenting a comparison of results obtained by the three approaches**

**Implementation details of the Perceptron rule**

Here I am describing the implementation details of the Perceptron rule:

As given by the data we will have 784 variables (input features). Each variable is given the input of the pixel on or off from the mat file data. As perceptron is binary classifier we have to use ALL Pair approach. By the all pair approach we have made 47 perceptrons each perceptron representing the one pair. Then we created the array of weights the size of this array is 785. Here position 785 represents the bias weight. First we gave the weights a random value between 0 and 1. After giving the value we loop through the all instances of the two classes which are in one perceptron pair.

During looping through the instances we keep updating the weights and bias. This process involves computing local and global error. Local error is basically the difference between the predicted and actual value and global error is the summation of the local errors of all instances.

At the end we computed the root mean square error. And from this can see that how much is our perceptron is efficient in recognizing digits.

Here is the result of our classifier efficiency :

92.7039 85.8370 78.6046 71.8133 65.4358 57.5826 49.8129 43.6047 36.2907 93.1507 86.1976

79.5594 73.2958 65.8146 58.3588 52.2857 45.2273 92.8785 85.9275 79.2934 71.7634 64.2034

57.7662 50.5212 93.0148 86.2994 78.9751 71.5633 65.0602 57.8944 93.0623 85.8122 78.4194

71.7048 64.4846 92.7756 85.3561 78.3860 71.0664 92.8289 85.8982 78.7591 93.0508

86.0420 93.0508 86.0420 92.8996

**Implementation details of the sigmoid unit using gradient descent learning**

First, we will use one neuron with two 784 inputs. Note that the inputs are given equal weights by assigning the weights (w’s) to ‘1’. The threshold, T, is set to 0. We calculate the output by Computing the total weighted inputs, Now Calculate the output using the logistic sigmoid activation function. O = 1/1+e^-X. for all the values that lie below O is of one type1 and that lie above is of type2. Then find the global error as described above and compute the efficiency.

Same method will be applied to all other 46 pair of classes i.e 46 neuron. And the result will be calculated.

The result of all the perceptron is:

93.1614 85.8787 78.9706 71.6185 63.8708 57.5646 51.2019 43.2588 36.4301 92.7139 85.5261 78.0413 70.2194 63.5122 56.8027 48.7814 41.6909 92.9797 85.788 78.2920 71.6681 65.0915 57.3818 50.4838 92.8441 85.4297 78.6001 71.8690 64.2339 57.2552 92.7980 85.9036 79.1607 71.7266 64.8054 93.0856 86.3781 79.1953 72.3803 93.0304 85.8177 78.8145 92.8091 85.6761 92.8091 85.6761 92.9584

**Implementation details of the learning weights using genetics algorithm**

In this initial process is same as above but difference comes in learning weights of a perceptron for the digit recognition is using genetics algorithm. In my case weights are my chromosome, I first defined the fitness function that finds the maximum fittest value. This is my fitness function f(w) = (w0+w1)-(w2+w3)+ (w4+w5)-(w6+w7)….till 785. And for 785 value bias = (1-weights (785)) ^2.

Then I will pass reference of my fitness function to ga (my fitness, nvars). Here nvars is no of variables. This will optimize the weights value.

Then using the weights value you can find the best solution possible. This process involves computing local and global error. Local error is basically the difference between the predicted and actual value and global error is the summation of the local errors of all instances.

**Comparison between Perceptron rule, Sigmoid activation and learning weights by using GA**

Perceptron rule is not so much efficient in finding the correct solution. Its effiencncy lies in between the 80-85 on avg as can been seen by the above results shown. On the other hand by using GA for learning weights its efficiency increased as rocket. By using GA its efficiency is between 94-97%. If you increased the no of iterations for updating weights, its efficiency can be increased more. But for Perceptron rule it will always lie in between a specific range on avg. In GA it will try to find a more fit solution everytime when you update the weight (the fit solution is calculated by cross mutation). There is one drawback in GA that is for a large no of input variables its processing time increased rapidly. More the no of variables more the processing time to calculate the best solution. On the other hand if we are using perceptron rule its processing time to find the solution is remarkably low then the GA.

For this I can conclude that GA is most efficient to find correct solution but it takes more time and perceptron rule can find the solution in less and it is not much efficient.

In between them lies Sigmoid activation functions are often used in artificial **neural networks** to introduce nonlinearity. A **neural network** element computes a linear combination of its input signals, and applies a **sigmoid** function to the get the result. Sigmoid is better then the perceptron rule in efficiency that can be seen by the result shown above. But it is less then that of GA.