

# **INTRODUCTION TO MACHINE LEARNING**

## **PROJECT 3: CLASSIFICATION**

**SINDHURA UPPU**

**UBIT NAME: suppu**

**PERSON NUMBER: 50206730**

Aim: This project aims to implement and evaluate classification algorithms to classify a 28\*28 grayscale handwritten digit image and identify as a digit among 0,1,..,9.

This project can be divided into 4 sub tasks as follows:

- Implementing logistic regression
- Implementing single hidden layer neural network
- Implementing convolutional neural network
- Testing MNIST trained models on USPS test data

Feature Extraction and Data Partition:

- The MNIST dataset has been downloaded from the internet and it is loaded into python as a 2-Dimensional array using cPickle.
  - files = 'mnist.pkl.gz'
  - fo = gzip.open(files,'rb')
  - training\_data,valid\_data,\_test\_data = cPickle.load(fo)
- This data is divided into training, validation and test data sets.
- The weights are obtained using training data and the accuracy of these weights is tested upon validation and test data.

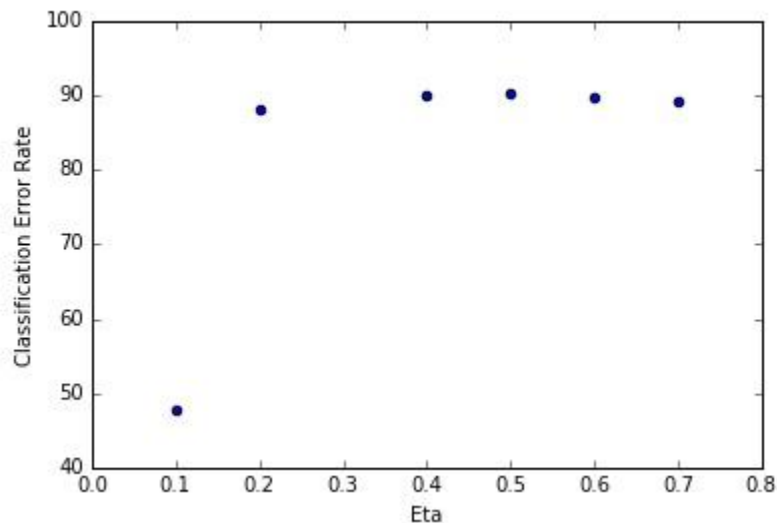
Logistic Regression:

- 1-of-K coding scheme has been used here.
- Initial weights are randomly generated in the confinements of [784,10].
- From these weights, the activation vector  $a$ , is calculated and the resulting  $y(x)$ , which is the logistic regression model is obtained using the exponential probability as below:

$$p(C_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

- The gradient of the error function is calculated using the following equation:
$$\nabla_{\mathbf{w}_j} E(\mathbf{x}) = (y_j - t_j) \mathbf{x}$$
- Initially, when the  $t$  vector is taken from the training data, it will be a [50000,1] array.
- This needs to be converted to [50000,10] array so as to make this equation work.
- It can be done using one hot encoding where the target value in each row of the  $t$  vector is taken, assigned a value 1 in the corresponding index of the  $t_j$  vector while all the other values in the row will be 0.

- Using the gradient of the error function and some random eta value (between 0 and 1), the new weights are calculated and updated.
- This process is run for 100-500 times and the resulting y array is taken and compared to t vector from the MNIST data.
- Based on the number of equalities obtained, accuracy of the algorithm is calculated and the maximum obtained is 91.28 for training data, 90.12 for validation and 89.05 for testing data for eta = 0.5.
- For various values of hyper parameter eta, the accuracy obtained for validation data is plotted as below:



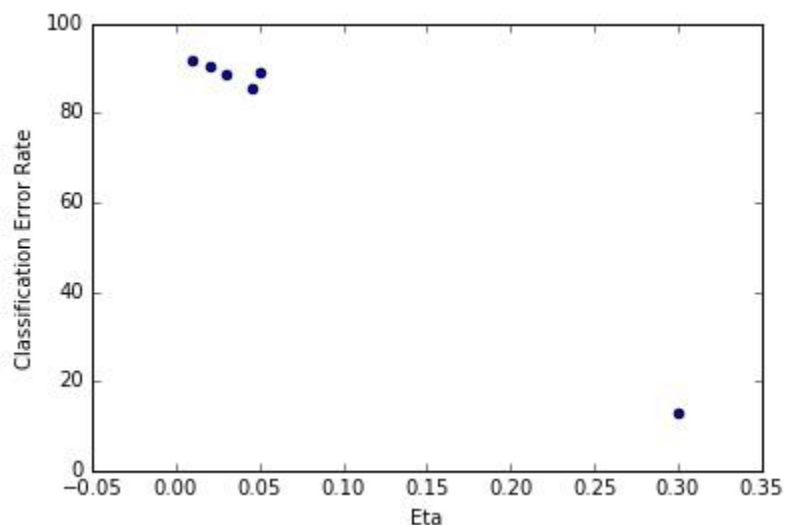
#### Single Layer Neural Network:

- A single layer neural network with one hidden layer is used here.
- The weights of the hidden units  $w_{ji}$  and  $w_{kj}$  are initialized randomly between 0 and 1.
- $W_{ji}$  vector will have dimensions  $[784, M]$  for the training data of MNIST dataset where in M is the number of nodes in the hidden layer.
- $W_{kj}$  vector will have dimensions  $[M, 10]$  where M is the number of nodes in hidden layer and 10 is the classification digit image range (there are 10 digits here).
- For each row, we calculate  $z_j$ ,  $a_k$  and  $y_k$  vectors where  $z_j$  is the activation of the hidden layer and  $h(.)$  function is the activation function of the hidden layer.
- Ideally,  $h(.)$  can be either logistic sigmoid, hyperbolic tangent or rectified linear unit.
- Here, we have considered  $h(.)$  function to be a logistic sigmoid of the form:

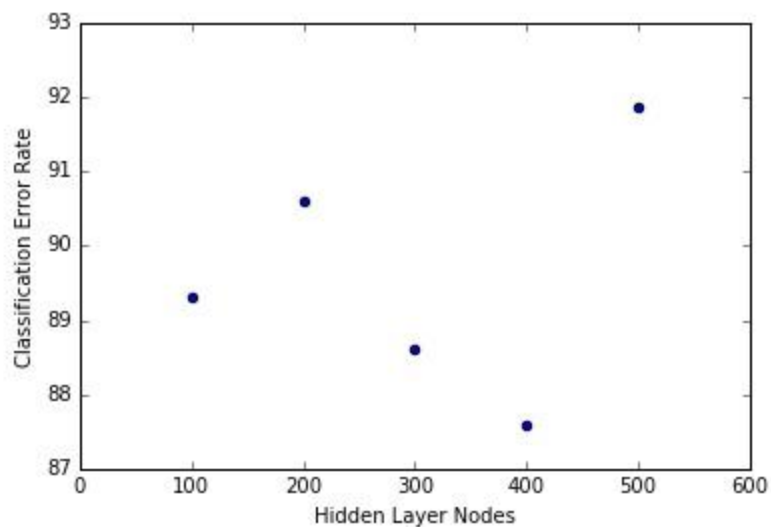
- $h(x) = ( 1 / ( 1 + \exp(-x) ) )$
- Cross entropy function is used to find the error and the gradient of the error function is derived from this using derivation w.r.t both the weights  $w_{ji}$  and  $w_{kj}$  :

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i, \quad \frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

- After obtaining the gradients, stochastic gradient descent can be used to train the neural network.
- The new weights are calculated by subtracting the original weights with (eta\*gradient function).
- After running this process for all the rows in the data for n number of iterations, we get the final y vector which will be compared with the t vector taken from the training data (the original classified data).
- Both these vectors are compared the resulting accuracy of the algorithm will be known.
- When 500 hidden nodes are taken in a single layer and eta is 0.01, the highest accuracy found for the training data is , validation data is 92.51%, for testing data is 92.00%.
- In this algorithm, there are two hyper parameters that we tune so as to get desired results, namely eta and number of nodes.
- The accuracy obtained for different values of eta is plotted as below:



- The accuracy obtained for when different number of nodes in the hidden layer are taken are plotted as below:

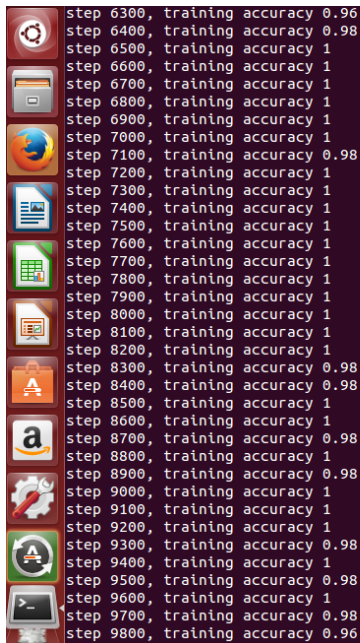


### Convolutional Neural Network:

- Tensorflow is used to build a convolutional neural network.
- Installed tensorflow interface using anaconda in an Ubuntu instance and implemented CNN using the tutorial slides posted by the TA as reference.
- The accuracy obtained over the iterations is displayed along with an installation screenshot below:

```
sindhurau@ubuntu: ~/Downloads/Python-2.7.12
- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify an different location below

[/home/sindhurau/anaconda] >>>
PREFIX=/home/sindhurau/anaconda
installing: python-2.7.8-1 ...
installing: conda-3.7.0-py27_0 ...
installing: conda-build-1.8.2-py27_0 ...
installing: _license-1.1-py27_0 ...
installing: abstract-rendering-0.5.1-np19py27_0 ...
installing: argcomplete-0.8.1-py27_0 ...
installing: astropy-0.4.2-np19py27_0 ...
installing: atom-0.3.9-py27_0 ...
installing: beautiful-soup-4.3.2-py27_0 ...
installing: binstar-0.7.1-py27_0 ...
installing: bitarray-0.8.1-py27_0 ...
installing: blaze-0.6.3-np19py27_0 ...
installing: blz-0.6.2-np19py27_0 ...
installing: bokeh-0.6.1-np19py27_0 ...
installing: boto-2.32.1-py27_0 ...
installing: cairo-1.12.2-2 ...
installing: casuarus-1.1-py27_0 ...
installing: cdecimal-2.3-py27_0 ...
installing: cffi-0.8.6-py27_0 ...
installing: chaco-4.4.1-np19py27_0 ...
installing: colorama-0.3.1-py27_0 ...
installing: configobj-5.0.6-py27_0 ...
installing: cryptography-0.5.4-py27_0 ...
installing: curl-7.38.0-0 ...
installing: cython-0.21-py27_0 ...
installing: cytoolz-0.7.0-py27_0 ...
installing: datashape-0.3.0-np19py27_1 ...
```



```

step 6300, training accuracy 0.96
step 6400, training accuracy 0.98
step 6500, training accuracy 1
step 6600, training accuracy 1
step 6700, training accuracy 1
step 6800, training accuracy 1
step 6900, training accuracy 1
step 7000, training accuracy 1
step 7100, training accuracy 0.98
step 7200, training accuracy 1
step 7300, training accuracy 1
step 7400, training accuracy 1
step 7500, training accuracy 1
step 7600, training accuracy 1
step 7700, training accuracy 1
step 7800, training accuracy 1
step 7900, training accuracy 1
step 8000, training accuracy 1
step 8100, training accuracy 1
step 8200, training accuracy 1
step 8300, training accuracy 0.98
step 8400, training accuracy 0.98
step 8500, training accuracy 1
step 8600, training accuracy 1
step 8700, training accuracy 0.98
step 8800, training accuracy 1
step 8900, training accuracy 0.98
step 9000, training accuracy 1
step 9100, training accuracy 1
step 9200, training accuracy 1
step 9300, training accuracy 0.98
step 9400, training accuracy 1
step 9500, training accuracy 0.98
step 9600, training accuracy 1
step 9700, training accuracy 0.98
step 9800, training accuracy 0.96

```

Output of the training, validation and test data:

\*For logistic regression, the loop is run 500 times, when  $\eta = 0.5$

\*For Single layer hidden neural network, the loop is run 5 times for each row of the x array, number of hidden nodes is taken as 500 and  $\eta = 0.01$

\*This result is the optimal that this algorithm could deliver.

N\_right: 45142

Evaluation of Logistic Regression: 90.28

N\_right: 46058

Evaluation of Single Layer Neural Network: 92.116

Weights of Logistic Regression: ', array([[ 0.7938433 , 0.83306663, 0.82554831,  
..., 0.39571299,  
0.56662317, 0.80603846],  
[ 0.37778774, 0.90801221, 0.40026096, ..., 0.17603514,  
0.78453768, 0.49804752],  
[ 0.4512041 , 0.92495684, 0.22163058, ..., 0.48504726,  
0.4499292 , 0.54942159],  
...,

```

[ 0.54578084, 0.7540924 , 0.50783323, ..., 0.97070544,
  0.33582175, 0.10031141],
[ 0.95774925, 0.77296265, 0.07999607, ..., 0.76146727,
  0.83979595, 0.62530324],
[ 0.03227355, 0.83758205, 0.68861277, ..., 0.38122989,
  0.28378375, 0.67778451]]))
  Weights wji in Single Layer Neural Network: ', array([[
    0.08520411, 0.00455009, 0.03931974, ..., 0.00958287,
    0.00212882, 0.01444968],
[ 0.03348533, 0.07465367, 0.05145776, ..., 0.08146869,
  0.06304862, 0.02503765],
[ 0.07828798, 0.01524623, 0.09073858, ..., 0.0925157 ,
  0.0521738 , 0.01652095],
...,
[ 0.04600487, 0.0948537 , 0.02799581, ..., 0.05479937,
  0.05443338, 0.04635508],
[ 0.01940107, 0.00734422, 0.01926921, ..., 0.02366161,
  0.09133039, 0.09311651],
[ 0.08652085, 0.03575448, 0.05764442, ..., 0.06900309,
  0.07826075, 0.03804729]]))
('Weights wkj in Single Layer Neural Network: ', array([[ 6.20621633e-02, -1.87472446e-
01, 4.72410336e-01,
  3.86316862e-01, -6.54209209e-01, -5.65976236e-01,
  4.34648203e-01, 1.13786525e+00, -2.58095399e-01,
  -2.47226356e-01],
[ 3.36802737e-01, -2.20644111e-01, -8.51592801e-02,
  3.73467574e-01, -1.33022802e-01, 7.06108832e-01,
  -1.79227762e-01, 4.52373606e-02, -1.90603905e-01,
  .....
[ -2.58157158e-01, -3.77776173e-01, -6.44144038e-02,
  -4.25190269e-01, 1.57635349e+00, 3.73374056e-02,
  2.69169050e-01, -5.50479860e-01, -1.14374905e-01,
  3.17150857e-01],
[ -4.35751011e-01, -2.13102441e-01, 2.03792120e-01,
  -6.15478596e-02, 9.72000772e-01, -2.02650835e-01,
  5.36067862e-01, -1.05884046e-01, -1.43814226e-01,
  -1.25823084e-01],
[ 7.01452944e-01, -2.11539822e-01, 3.53213482e-01,
  7.10204969e-03, 2.39282130e-01, -3.73545922e-01,
  2.12486417e-01, -1.54464847e-02, -3.98959792e-01,

```

7.10665938e-02],  
[ -2.79176246e-01, 1.28357711e-01, 3.29370503e-01,  
6.92729475e-01, 1.11643442e-01, -1.76147931e-01,  
-2.43105828e-01, 3.04375505e-01, -3.32079291e-02,  
-1.79817805e-01],  
[ -1.20796134e-01, -4.53029197e-01, 1.65045047e-01,  
-3.73773335e-01, 8.12105265e-01, -6.76484480e-01,  
7.99770979e-01, 6.52230751e-01, -5.59824831e-01,  
4.05861937e-01],  
[ 2.38679688e-01, 4.77477364e-01, 1.23600938e+00,  
2.08071464e-01, -3.03321506e-01, 7.73760714e-01,  
-3.82212238e-01, 9.98284619e-02, -1.44663908e+00,  
-2.27484070e-01],  
[ -2.20698796e-01, 2.04208541e-02, 5.25526889e-01,  
-2.68988634e-01, -2.90382139e-02, 4.26783470e-01,  
6.35657489e-01, -1.73759784e-01, -2.44587599e-01,  
-6.21340947e-02],  
[ 9.65396941e-01, -7.88442601e-01, -3.49768307e-01,  
1.34812493e+00, -1.83114601e-01, 2.59557731e-02,  
-7.05891354e-01, 1.72670104e-01, -1.55552848e-01,  
2.05298321e-01]]))

N\_right: 9012

Evaluation of Logistic Regression Validation: 90.12

N\_right: 9251

E-evaluation of Single Layer Neural Network Validation: 92.51

N\_right: 8905

Evaluation of Logistic Regression Test: 89.05

N\_right: 9200

Evaluation of Single Layer Neural Network Test: 92.00