Report
Neural Learning

- With reference to http://neuralnetworksanddeeplearning.com/chap1.html
- Using Stochastic Gradient Descent method

C:\WINDOWS\system32\cmd.exe - python

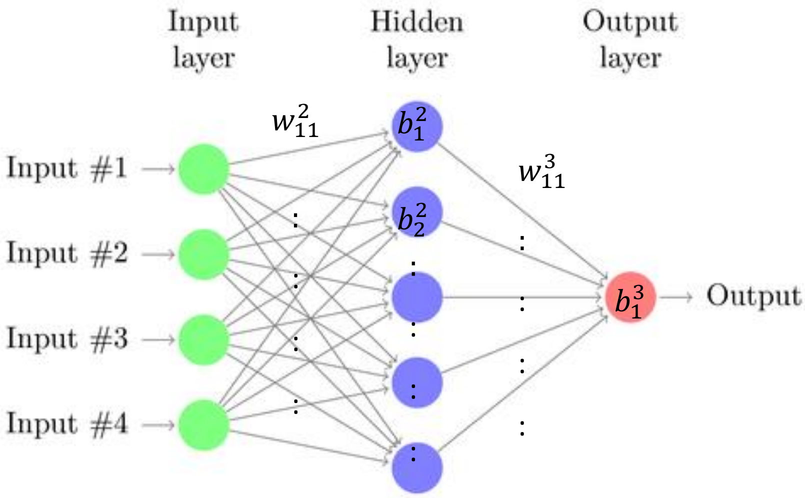
```
C:\Users\Ajinkya\Desktop\Tt\SEM4\ProfProject\python_FILES\pgm>python
Python 2.7.9 (default, Dec 10 2014, 12:28:03) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import mnist_loader
>>> import Network as network
>>> training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
>>> net = network.Network([784, 30, 10])
>>> net.SGD(training_data, 30, 10, 3.0, test_data = test_data)
Epoch 0: 7973 / 10000
Epoch 1: 8065 / 10000
Epoch 2: 8141 / 10000
Epoch 3: 8221 / 10000
Epoch 4: 8175 / 10000
Epoch 5: 8280 / 10000
Epoch 6: 8250 / 10000
Epoch 7: 8283 / 10000
Epoch 8: 8319 / 10000
Epoch 9: 8312 / 10000
Epoch 10: 8331 / 10000
Epoch 11: 8343 / 10000
Epoch 12: 8355 / 10000
Epoch 13: 8341 / 10000
Epoch 14: 8309 / 10000
Epoch 15: 8353 / 10000
Epoch 16: 8360 / 10000
Epoch 17: 8358 / 10000
Epoch 18: 8356 / 10000
Epoch 19: 8368 / 10000
Epoch 20: 8372 / 10000
Epoch 21: 8361 / 10000
Epoch 22: 8382 / 10000
Epoch 23: 8388 / 10000
Epoch 24: 8398 / 10000
Epoch 25: 8370 / 10000
Epoch 26: 8400 / 10000
Epoch 27: 8365 / 10000
Epoch 28: 8409 / 10000
Epoch 29: 8398 / 10000
>>>
```

Fig: Output from simulation of neural network with MNIST handwriting training data

- Neurons in Layers 1,2 & 3 are 784:30:10
- **Epochs :** 30
- **Mini Batch Size :** 10
- **Learning Rate (η)** : 3.0

Code Structure

- Initialization of weights and biases as matrices



- Weights matrix $(W) = \begin{bmatrix} \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & w_{14}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 & w_{24}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 & w_{34}^2 \\ w_{41}^2 & w_{42}^2 & w_{43}^2 & w_{44}^2 \\ w_{51}^2 & w_{52}^2 & w_{53}^2 & w_{54}^2 \end{bmatrix}, [w_{11}^3 \quad w_{12}^3 \quad w_{13}^3 \quad w_{14}^3 \quad w_{15}^3] \end{bmatrix}$
- Biases matrix $(\mathrm{b}) = \begin{bmatrix} \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \\ b_4^2 \\ b_5^2 \end{bmatrix}, [b_1^3] \end{bmatrix}$
- $\boldsymbol{Input}\ to\ layer\ l, (z^l) = {W^l}^T a^{l-1} + b^l$
- $\boldsymbol{Activation}\ to\ layer\ l, (a^l) = \sigma(z^l),\ \ l = 2,3, \ldots$

  Where, $\sigma = \frac{1}{1+e^{-bx}}$ is Activation Function ,also called Logistic Function

  (Note : for layer $(l = 1)$ activation=input, or $\sigma(z) = z$
- **Feed-Forward** (given input X) :

  Compute activation of each layer by iterative process (network with L layers)

$$(z^l) = {W^l}^T a^{l-1} + b^l \quad , for\ l = 2,3,4, \ldots, L$$

- **Back-Propagation** (Algorithm):

Input x

Giving activation for input layer (a^1)

FeedForward :

compute $\left(z^l\right) = {W^l}^T a^{l-1} + b^l \quad , for\ l = 2,3,4, \dots, L$

& compute $a^l = \sigma\left(z^l\right)\ for\ each\ layer$

Output Error:

$compute\ \delta^L = \nabla_a C \odot \sigma'(z^L)$

Backpropagate the error :

For each $l = L-1, L-2, \dots, 2$

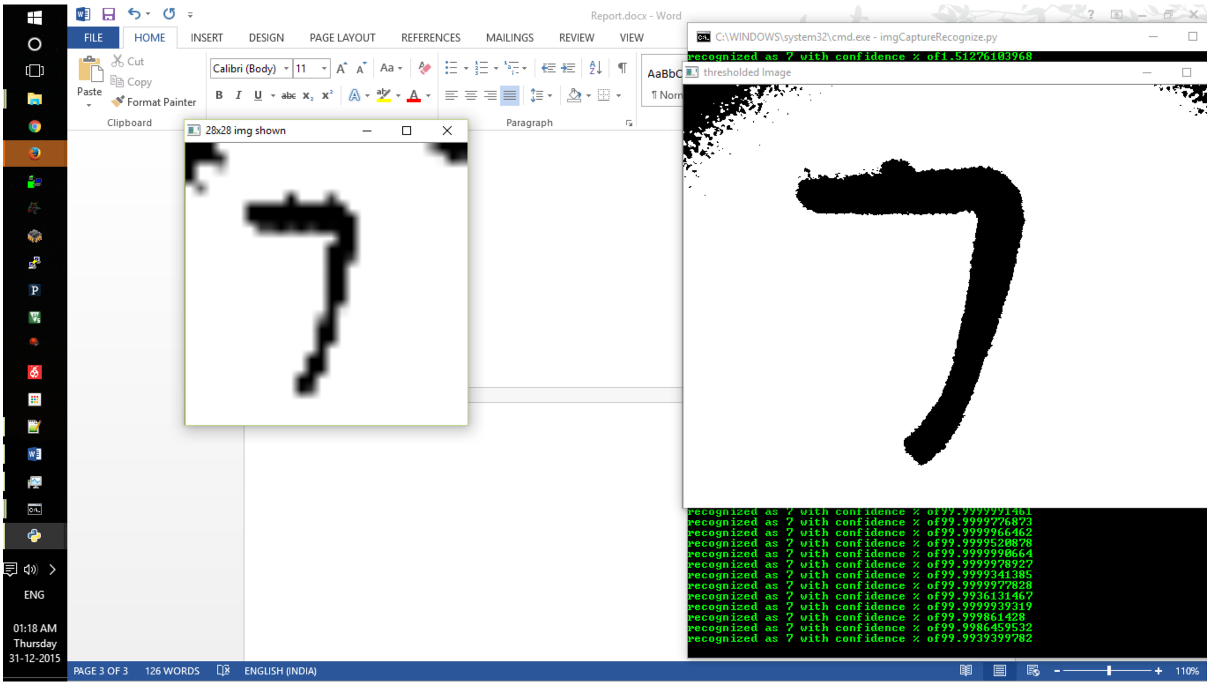
$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^L)$

Final Output:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \quad \& \quad \frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Results of Image Capture & Recognition

- Using openCV on python to capture and threshold the image , then



- Writing Matrices of Input Weights (when connection is of the form [784 ,10])