

Deep Learning 2019

Assignment # 2

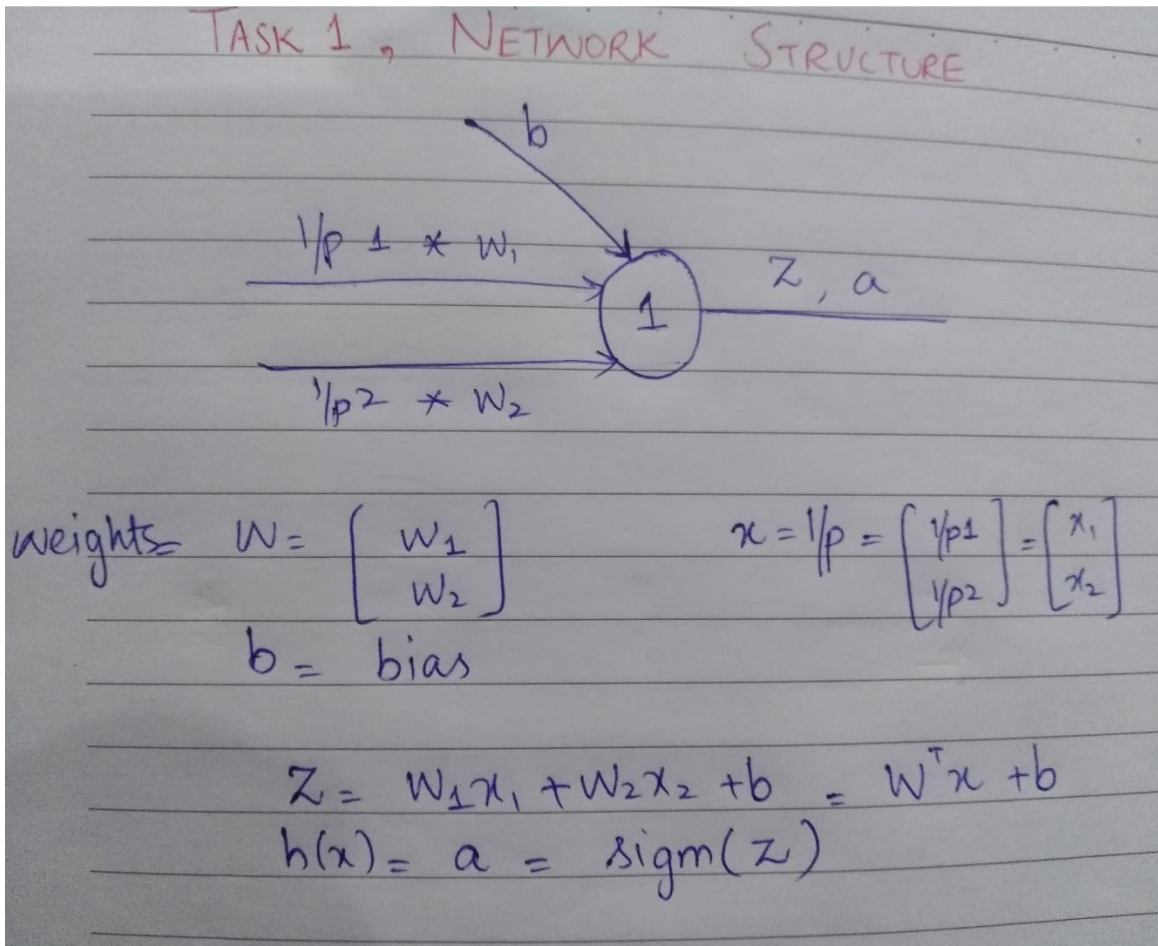
Report

Topic: Neural Networks

➤ Task 1:

Implement a Neural Network with Sigmoid and cross entropy loss function.

We are required to perform binary classification using a single neuron having sigmoid activation and minimize the binary cross entropy loss. Following is the structure of the network.



Goal:

Our goal is to find the values of weights w and bias b which minimize the loss. We'll do this by back propagating the error.

Steps:

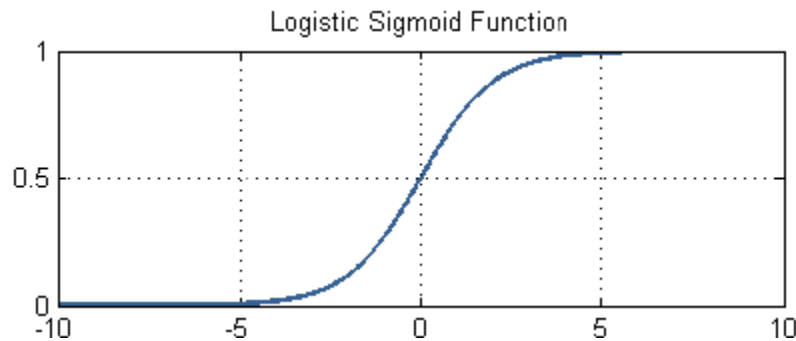
Note: I've performed all experiments using *SGD*.

Feedforward:

1. Randomly initialize weights and bias.
2. Compute z by:

$$z = w^T x + b$$

3. Apply sigmoid on z . Sigmoid is a squashing function. The value of z can be less than 0 or greater than 1, but for binary classification we only need 0 or 1 (or any value between 0 and 1, if we are dealing with probabilities. So, for this purpose we use sigmoid.



$$a = \text{Sigm}(z) = \frac{1}{1 + e^{-z}}$$

The value of a is the probability of the label being equal to 1.

Back-propagate:

1. Calculate the gradients for back-propagation of error. We need to find:

$$\frac{\partial L}{\partial w} \text{ \& \; } \frac{\partial L}{\partial b}$$

where L is binary cross-entropy loss, w is set of weights and b is bias.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial a} = \frac{a - y}{a(1 - a)}$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial w} = x$$

So,

$$\frac{\partial L}{\partial w} = (a - y) * x$$

Similarly,

$$\frac{\partial L}{\partial b} = (a - y)$$

Update Weights and Bias:

1. Update Weights:

$$w = w - \alpha * \frac{\partial L}{\partial w}$$

2. Update Bias:

$$b = b - \alpha * \frac{\partial L}{\partial b}$$

Alpha is the learning rate.

Steps of SGD:

1. For every data point:
 - Perform feedforward.
 - Perform back-propagation i.e. find gradients and back propagate the error, update the weights and bias simultaneously.
2. Compute Cross-entropy loss L:

$$L = \frac{1}{n} \sum_{i=1}^n (-(y_i * \log(a_i) + (1 - y_i) * \log(1 - a_i)))$$

where y_i is the actual label and n is the total number of training examples.

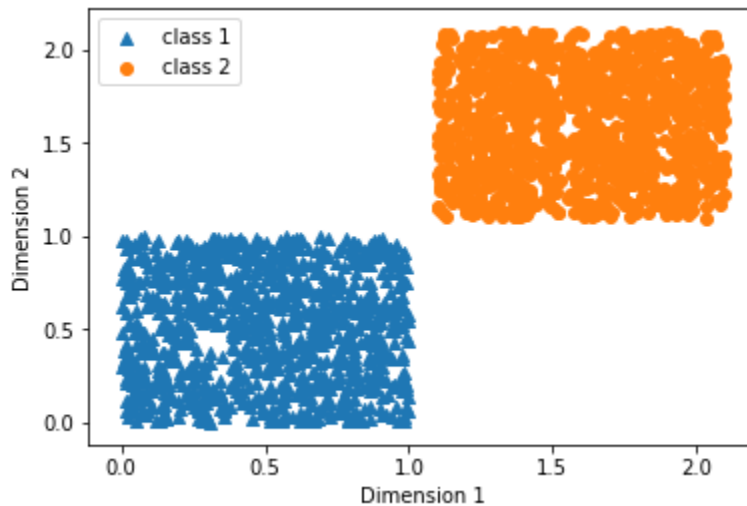
3. Repeat the above steps till L converges.

Results:

I've used random weights initialization using **np.random.rand**

Data:

Two dimensional data for two classes is randomly initialized.



Train, validation, test split:

Data is randomly split into train, validation and test set after shuffling.

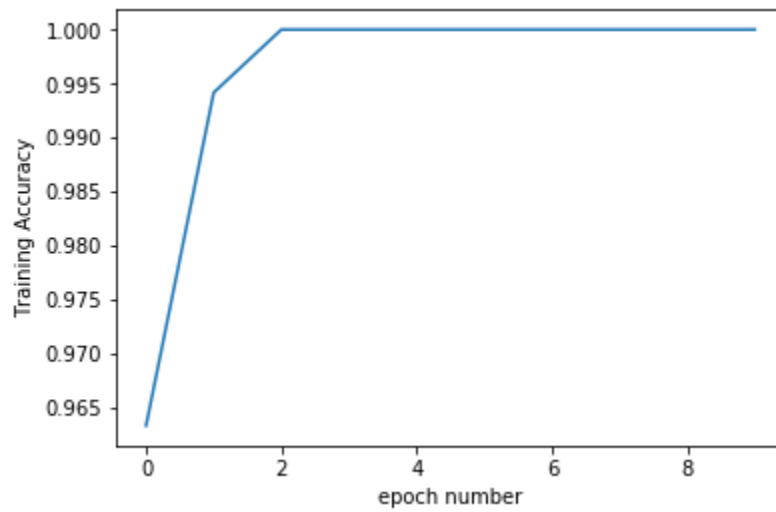
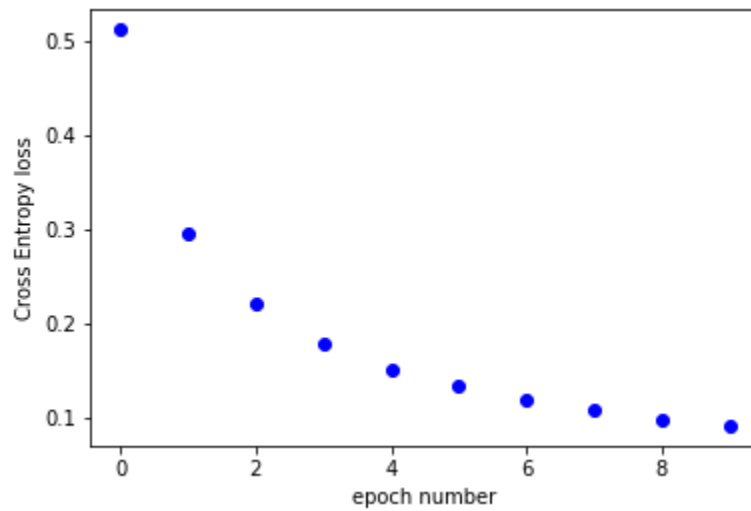
Set	Distribution
Train	60%
Validation	20%
Test	20%

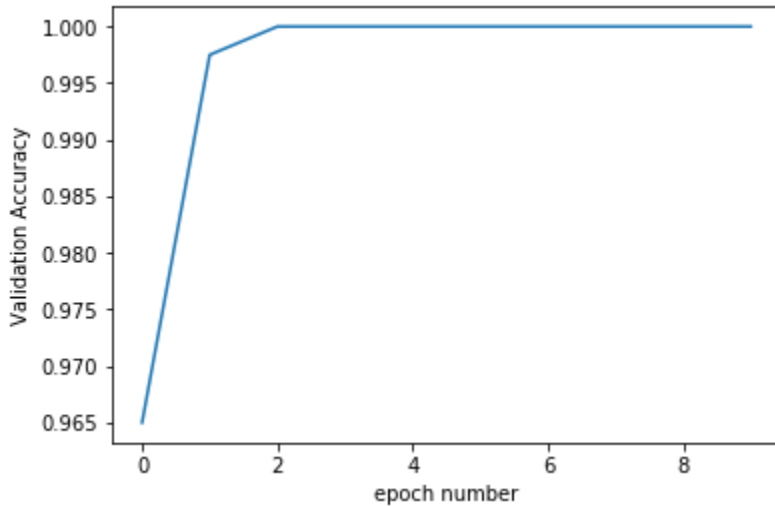
Accuracy:

Number of epochs: 10

Learning Rate: 0.01

Train	100%
Validation	100%
Test	100%





Data Splitting Strategy:

I have split data into three parts 60%, 20%, 20%. I wanted to use validation data for early stopping but I didn't need to as the model didn't over fit. I tried different values of learning rate while finding the best model. Almost all the values gave good results as it's a pretty simple dataset.

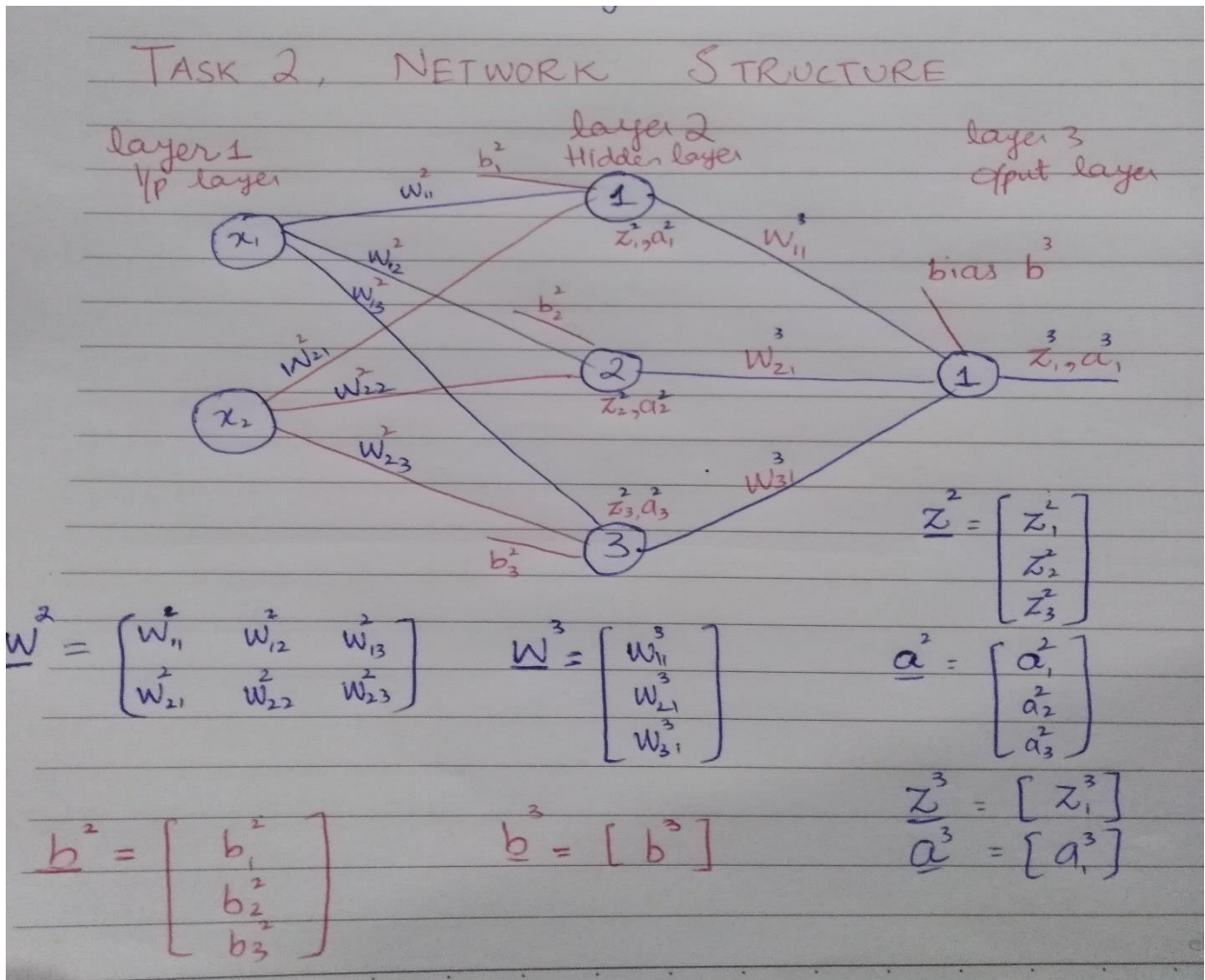
Comments:

I've learnt a lot from implementing this task. First of all, to implement this task, I performed manual calculations of back propagation algorithm which helped me understanding it very well and I figured out it's nothing but simple chain rule. Secondly, implementing this task gave me hands on experience on numpy, using lists and tuples which gave me the confidence and basic structure to implement back propagation with hidden layers in the next tasks.

➤ Task 2:

Implement a Neural Network with a single hidden layer, multiple activation function and cross entropy loss function

We are required to perform binary classification using a neural network on having only one hidden layer, (sigmoid/tanh/relu) activation and minimize the binary cross entropy loss. Following is the structure of the network.



Goal:

Our goal is to find the values of weights w^2 , w^3 and biases b^2 , b^3 which minimize the loss. We'll do this by back propagating the error.

Steps:

Note: I've performed all experiments using *SGD*.

Feedforward:

- Randomly initialize weights and biases.

$$w^2 = w_h, w^3 = w_o \text{ (h for hidden and o for output)}$$

$$b^2 = b_h, b^3 = b_o \text{ (h for hidden and o for output)}$$

- Compute z matrices by:

$$z_h = w_h^T x + b_h$$

$$z_o = w_o^T x + b_o$$

- Apply activation function on z to get a_h and a_o .

Sigmoid:

$$a = \text{Sigm}(z) = \frac{1}{1 + e^{-z}}$$

tanh:

$$a = \tanh(z) = \frac{2}{1 + e^{-2z}} - 1$$

Relu:

$$a = \max(0, z)$$

Back-propagate:

- Calculate the gradients for back-propagation of error. We need to find:

$$\frac{\partial L}{\partial w_o}, \frac{\partial L}{\partial w_h}, \frac{\partial L}{\partial b_o}, \frac{\partial L}{\partial b_h}$$

where L is binary cross-entropy loss, w is set of weights and b is bias.
Always using sigmoid at the output node (to get probabilities), we get:

Output layer:

$$dz_o = \frac{\partial L}{\partial z_o} = (a_o - y) = \delta_o$$

$$\frac{\partial L}{\partial w_o} = \delta_o * a \quad \& \quad \frac{\partial L}{\partial b_o} = \delta_o$$

Hidden Layer:

$$dz_h = \frac{\partial L}{\partial z_h} = \delta_o * w_o^T * a_h = \delta_h$$

Note: * is not dot product. It is just element-wise multiplication of two vectors.

$$\frac{\partial L}{\partial w_o} = \delta_h \mathbf{X} x$$

Note: \mathbf{X} is outer product and x is the input vector.

$$\frac{\partial L}{\partial b_o} = \delta_h$$

Update Weights and Bias:

3. Update Weights:

$$w_o = w_o - \alpha * \frac{\partial L}{\partial w_o}$$

$$w_h = w_h - \alpha * \frac{\partial L}{\partial w_h}$$

4. Update Biases:

$$b_o = b_o - \alpha * \frac{\partial L}{\partial b_o}$$

$$b_h = b_h - \alpha * \frac{\partial L}{\partial b_h}$$

Alpha is the learning rate.

Steps of SGD:

4. For every data point:
 - Perform feedforward.
 - Perform back-propagation i.e. find gradients and back propagate the error, update the weights and bias simultaneously.
5. Compute Cross-entropy loss L:

$$L = \frac{1}{n} \sum_{i=1}^n (-(y_i * \log(a_{oi}) + (1 - y_i) * \log(1 - a_{oi})))$$

where y_i is the actual label and n is the total number of training examples.

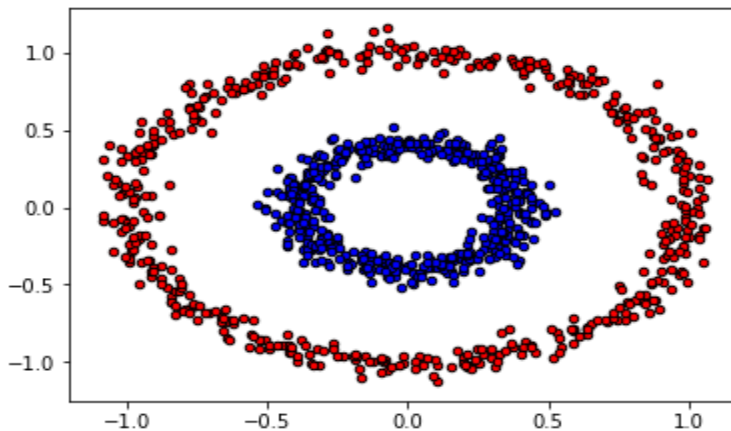
6. Repeat the above steps till L converges.

Task 2 Results:

I've used random weights initialization using **np.random.rand**

Data:

Two dimensional data for two classes is randomly initialized.



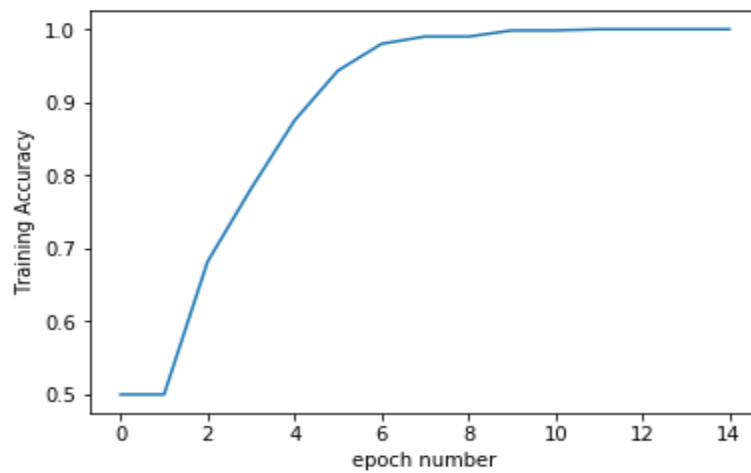
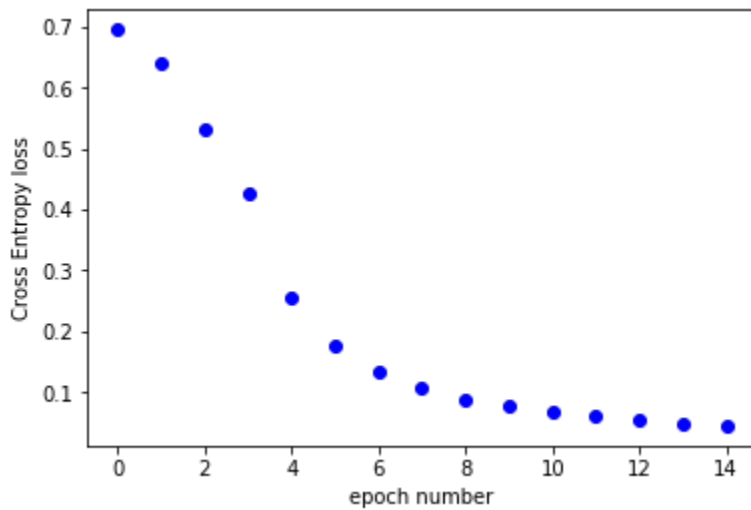
Train, validation, test split:

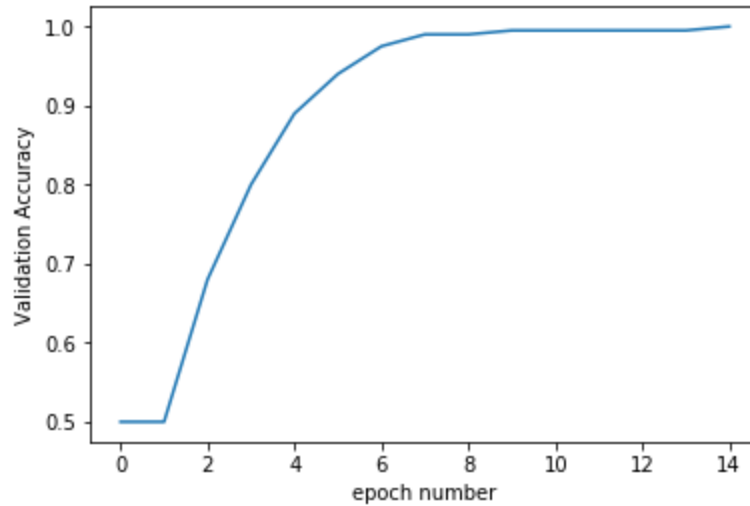
Data is randomly split into train, validation and test set after shuffling.

Set	Distribution
Train	60%
Validation	20%
Test	20%

Accuracy:

Activation Function	No. of epochs	Learning Rate	Data	Accuracy
Sigmoid	100	0.01	Train	81
			Validation	81
			Test	80.5
Tanh	15	0.1	Train	1
			Validation	1
			Test	1
Relu	20	0.005	Train	78.5
			Validation	78.5
			Test	78





Data Splitting Strategy:

I have split data into three parts 60%, 20%, 20%. I wanted to use validation data for early stopping but I didn't need to as the model didn't over fit. I tried different values of learning rate while finding the best model. Almost all the values gave good results as it's a pretty simple dataset.

Comments:

I've learnt a lot from implementing this task. First of all, to implement this task, I performed manual calculations of back propagation algorithm which helped me understanding it very well and I figured out it's nothing but simple chain rule. Secondly, implementing this task gave me hands on experience on numpy, using lists and tuples which gave me the confidence and basic structure to implement back propagation with hidden layers in the next task which is a layered neural network.

➤ Task 3:

Implement a multi-layer Neural Network for multi-class classification.

This task was the toughest. I tried to implement the mini-batch back propagation in vectorized fashion but couldn't complete it as I was short of time, so I implemented it using loops. Backpropagation steps are same as given in task 2 above. The only difference here is that now as there are more hidden layers so forward and backward propagations are performed using loops to make it generic.

Dataset:

MNIST dataset is used. Training is performed on 60000 images while testing on 10000 images.

Data-preprocessing:

Mean-image subtraction is done to normalize the data.

Mean Image:

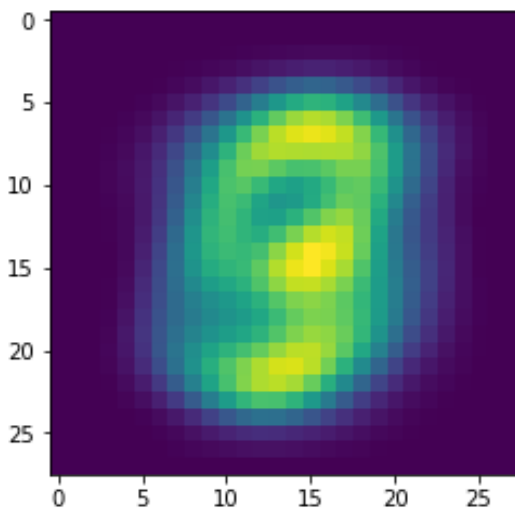


Image before mean-image normalization:

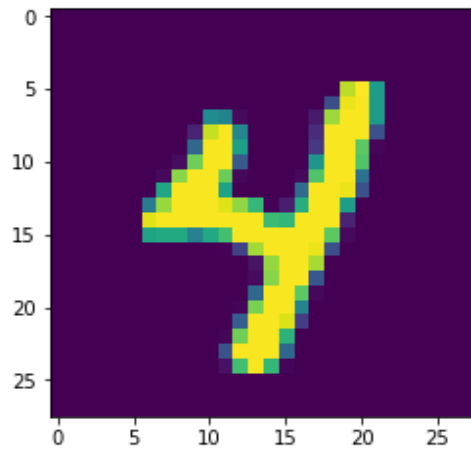
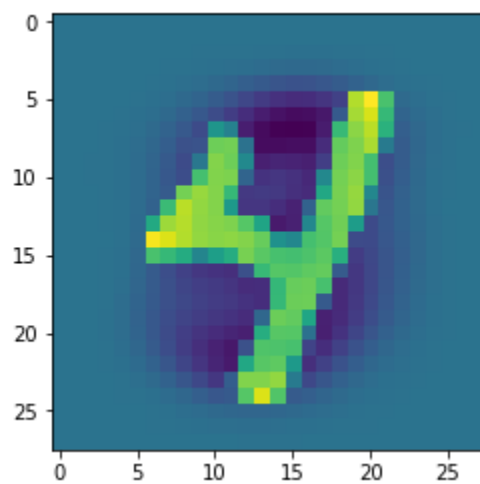


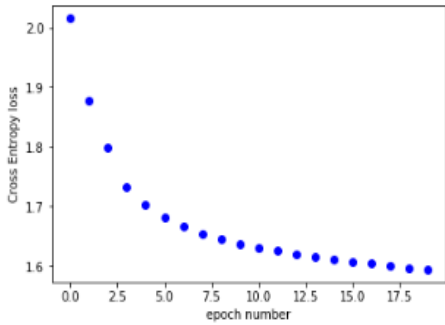
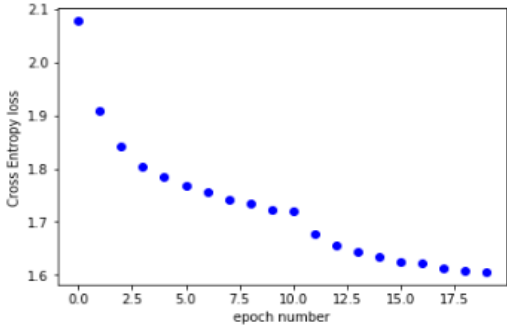
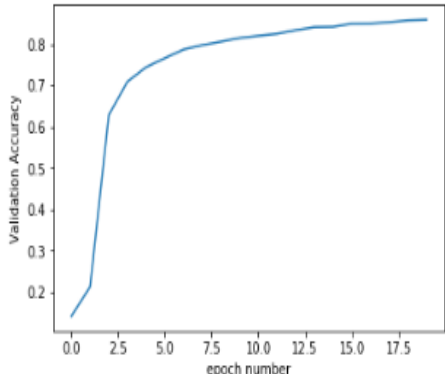
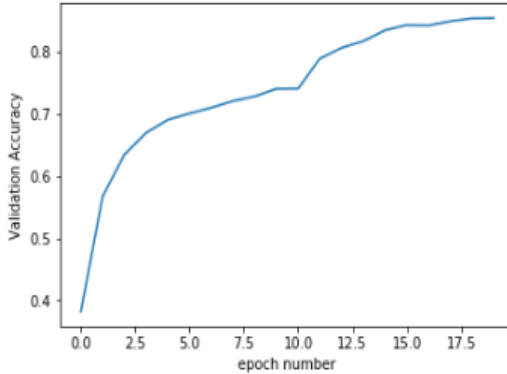
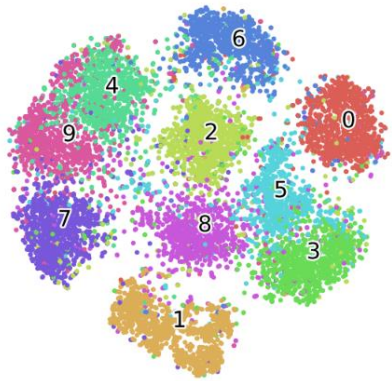
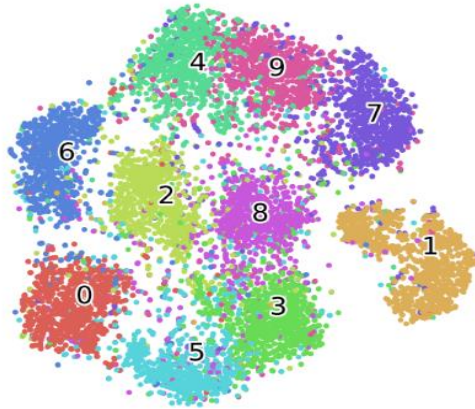
Image after mean-image normalization:



Task 3 Results:

I've used random weights initialization using `np.random.randn`

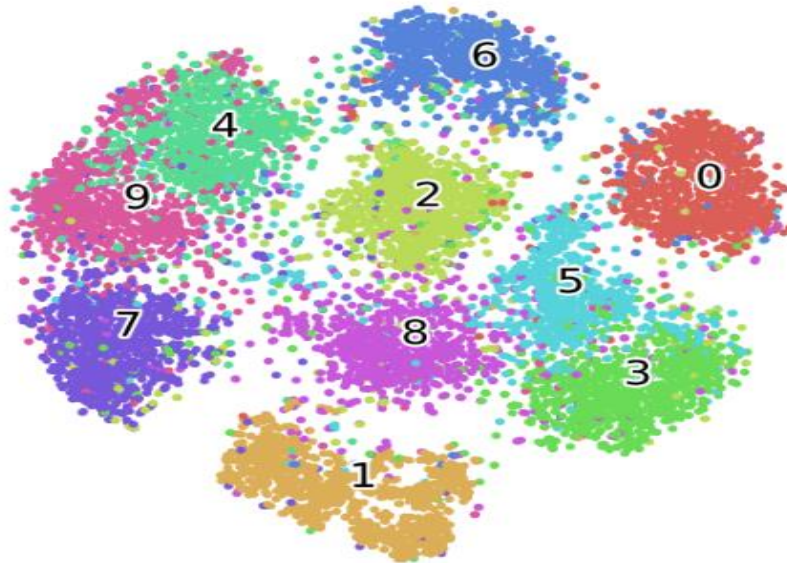
1. Plot loss and accuracy with mean image subtraction and without mean image subtraction. Report the difference in their accuracy and loss curves.

	With Mean-Image Subtraction	Without Mean-Image Subtraction
LOSS		
ACC		
TSNE		

2. Visualize data points using t-SNE technique

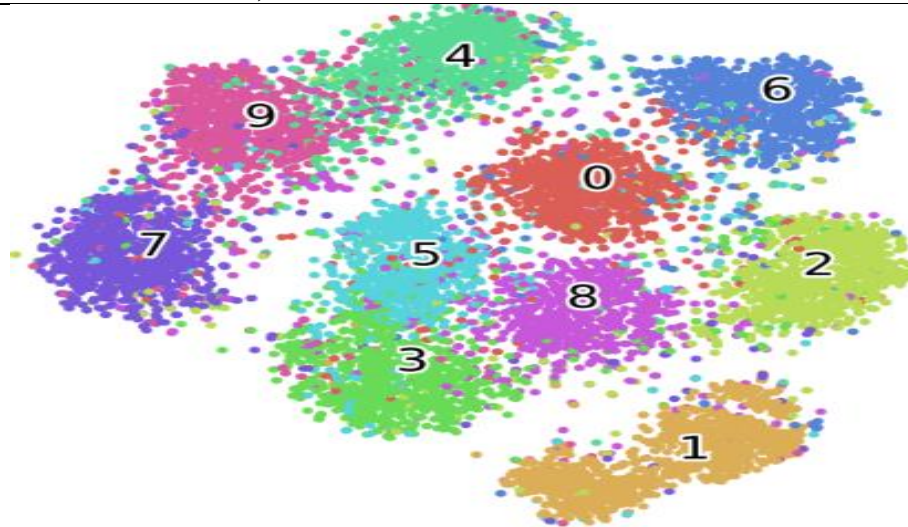
Two Hidden Layers

(784,128,64,10) (DATA IS LESS SEPARABLE HERE)

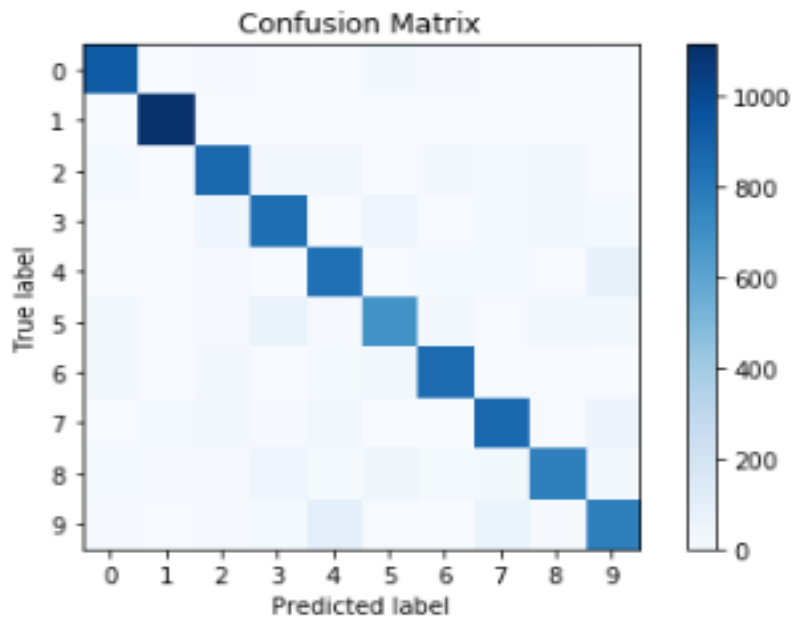


Three Hidden Layer

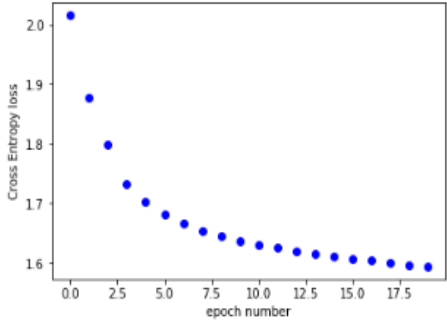
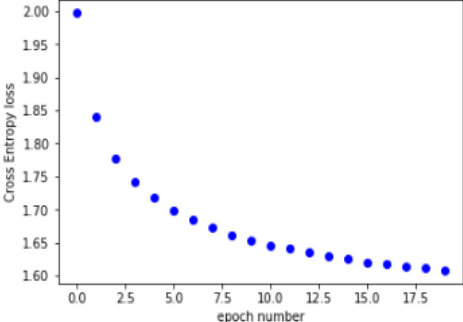
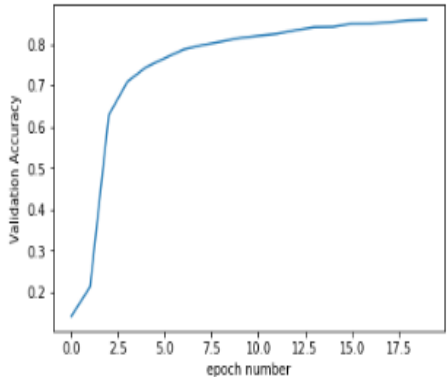
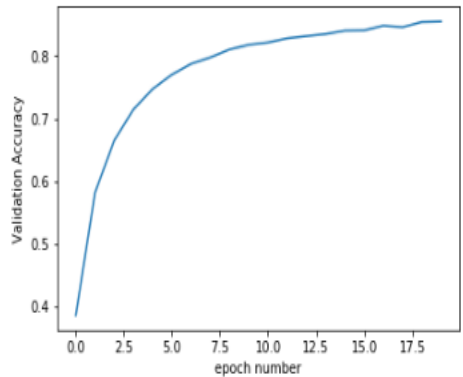
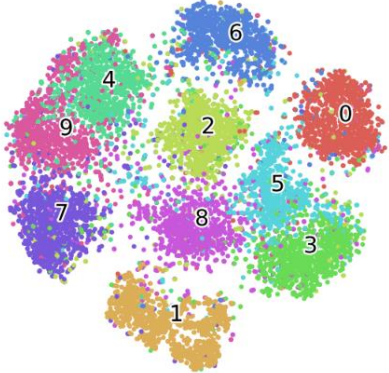
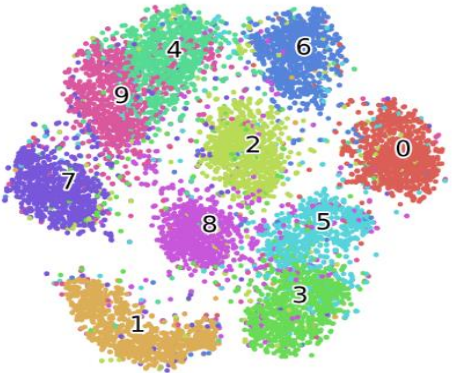
(784,256,128,64,10) (DATA IS MORE SEPARABLE HERE, less no. of mis-classifications)



3. Analyze confusion matrix of 10- class classification problem and save that matrix in the form of image. (784x128x64x10)

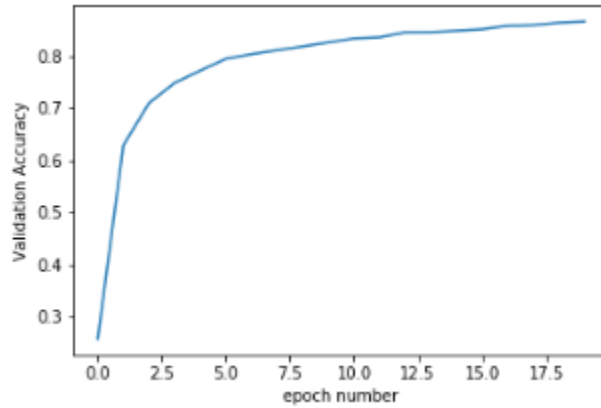
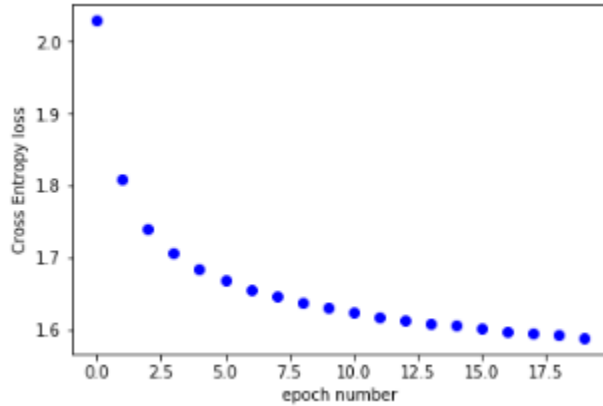


4. Plot loss and accuracy with and without dropout and explain how it effects.

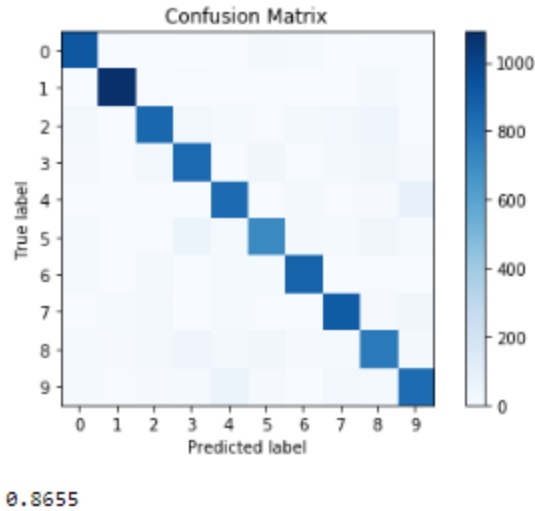
	Without Dropout	With Dropout (0.2%)
LOSS		
ACC		
TSNE		

Dropout has improved the accuracy for same number of hidden layers(2), same number of neurons(128,64), same batch size(1000), same value of learning rate(3), same number of epochs(20), on 10,000 test examples. Accuracy improved slightly as I've used drop-out probability of 0.2 only.

5. Report the accuracy by changing number of neurons in hidden layer. (784x256x100x10)
Accuracy has improved. It reached 80% just on the 6th epoch. I achieved 88% accuracy in 20 epochs, with only two hidden layers just by increasing the number of neurons in the hidden layers. I changed 128 in first hidden layer to 256 and 64 in second hidden layer to 100.



```
[ [ 920  0  7  2  1  23  15  3  6  3]
[  0 1090  5  5  1  5  4  2  23  0]
[ 26  2 856 28 14  5 22 27 44  8]
[ 11  2 30 847 3 41  5 23 38 10]
[  3  3  5  3 839  6 21  6 12 84]
[ 16  3  7 58 13 712 20 13 35 15]
[ 14  4 19  2 17 16 873  4  7  2]
[  4 14 29  4 14  5  2 903 11 42]
[ 13 10 22 44 22 36 11 14 779 23]
[ 14  7 12 15 68  9  2 33 13 836]]
```



➤ References:

https://www.researchgate.net/figure/Logistic-sigmoid-function-Maps-real-numbers-to-the-interval-between-0-and-1_fig6_234049070

<https://www.coursera.org/learn/machine-learning/lecture/1z9WW/backpropagation-algorithm>

<https://www.datacamp.com/community/tutorials/introduction-t-sne>