Subject: Training RBM on the hand-written digits.

Name: Mohit Kumar Soni
SC code: SC15B103

Theory of the RBM: -

RBM also known as Restricted Boltzmann Machines are well known for their generative properties. The main idea behind the training of Boltzmann machine is the Contrastive Divergence and the Gibbs sampling. The data set used for the training RBM is MNIST dataset. First by using the input data we will generate hidden layer approximates of output using Gibbs sampling and pre-activations of visible layers and the using those we will generate the approximate for the input. Then we will update the weights using the Regular RBM update equation.

Procedure: -

The initialization of weights is done using Xavier's initialization. The input layer has 728 neurons and the hidden layer using 500 neurons. The whole model is run for the 15 iterations and the output of the filters and the reconstruction is plotted.

Code Flow: -

The code for sampling: -

```
def sample_h_given_v(self, v0_sample):
    ''' This function infers state of hidden units given visible units '''
    # compute the activation of the hidden units given a sample of
    # the visibles
    pre_sigmoid_h1, h1_mean = self.propup(v0_sample)
    # get a sample of the hiddens given their activation
    # Note that theano_rng.binomial returns a symbolic sample of dtype
    # int64 by default. If we want to keep our computations in floatX
    # for the GPU we need to specify to return the dtype floatX
    h1_sample = self.theano_rng.binomial(size=h1_mean.shape,
                                         n=1, p=h1_mean,
                                         dtype=theano.config.floatX)
    return [pre_sigmoid_h1, h1_mean, h1_sample]
```

The distribution used for sampling is Binomial Distribution as our data is binary. The Theano library is used for regeneration and RBM building.

```
def sample_v_given_h(self, h0_sample):
    ''' This function infers state of visible units given hidden units '''
    # compute the activation of the visible given the hidden sample
    pre_sigmoid_v1, v1_mean = self.propdown(h0_sample)
    # get a sample of the visible given their activation
    # Note that theano_rng.binomial returns a symbolic sample of dtype
    # int64 by default. If we want to keep our computations in floatX
    # for the GPU we need to specify to return the dtype floatX
    v1_sample = self.theano_rng.binomial(size=v1_mean.shape,
                                         n=1, p=v1_mean,
                                         dtype=theano.config.floatX)
    return [pre_sigmoid_v1, v1_mean, v1_sample]
```

Same is done for sampling x from h. The input is the hidden layer output.

```
def free_energy(self, v_sample):
    ''' Function to compute the free energy '''
    wx_b = T.dot(v_sample, self.W) + self.hbias
    vbias_term = T.dot(v_sample, self.vbias)
    hidden_term = T.sum(T.log(1 + T.exp(wx_b)), axis=1)
    return -hidden_term - vbias_term
```

This is for the calculation of free Energy.

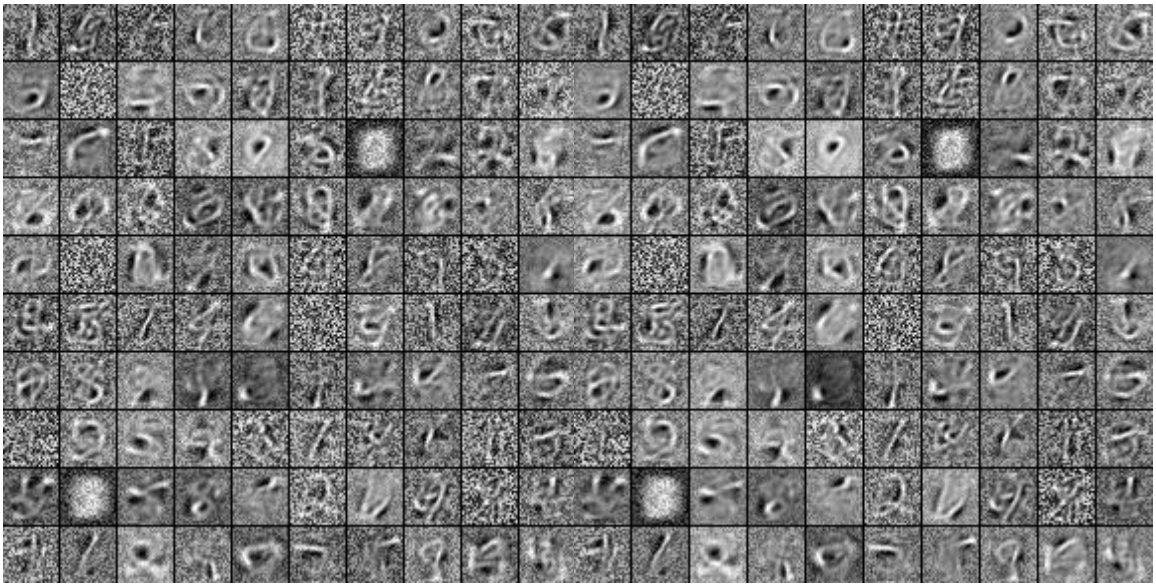
```
def gibbs_hvh(self, h0_sample):
    ''' This function implements one step of Gibbs sampling,
        starting from the hidden state'''
    pre_sigmoid_v1, v1_mean, v1_sample = self.sample_v_given_h(h0_sample)
    pre_sigmoid_h1, h1_mean, h1_sample = self.sample_h_given_v(v1_sample)
    return [pre_sigmoid_v1, v1_mean, v1_sample,
            pre_sigmoid_h1, h1_mean, h1_sample]

def gibbs_vhv(self, v0_sample):
    ''' This function implements one step of Gibbs sampling,
        starting from the visible state'''
    pre_sigmoid_h1, h1_mean, h1_sample = self.sample_h_given_v(v0_sample)
    pre_sigmoid_v1, v1_mean, v1_sample = self.sample_v_given_h(h1_sample)
    return [pre_sigmoid_h1, h1_mean, h1_sample,
            pre_sigmoid_v1, v1_mean, v1_sample]
```

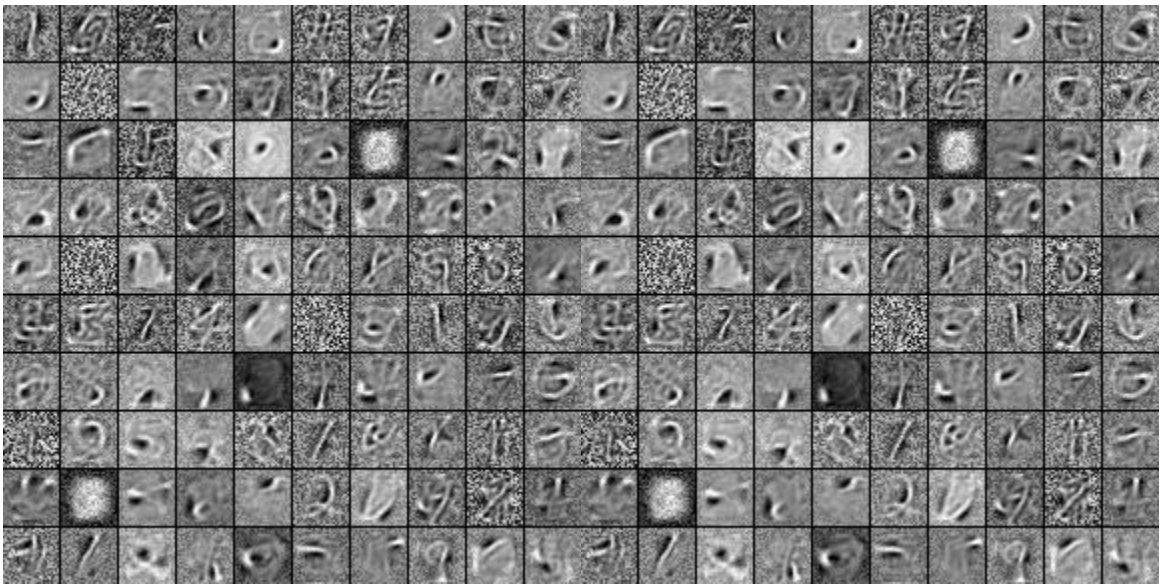
Gibbs sampling is done here.

For the calculation of the cost function the cross-entropy loss has been calculated.

Filter responses of all 15 epochs are being shown here which is done using 20 different chains of Gibbs sampling.



Epoch 0

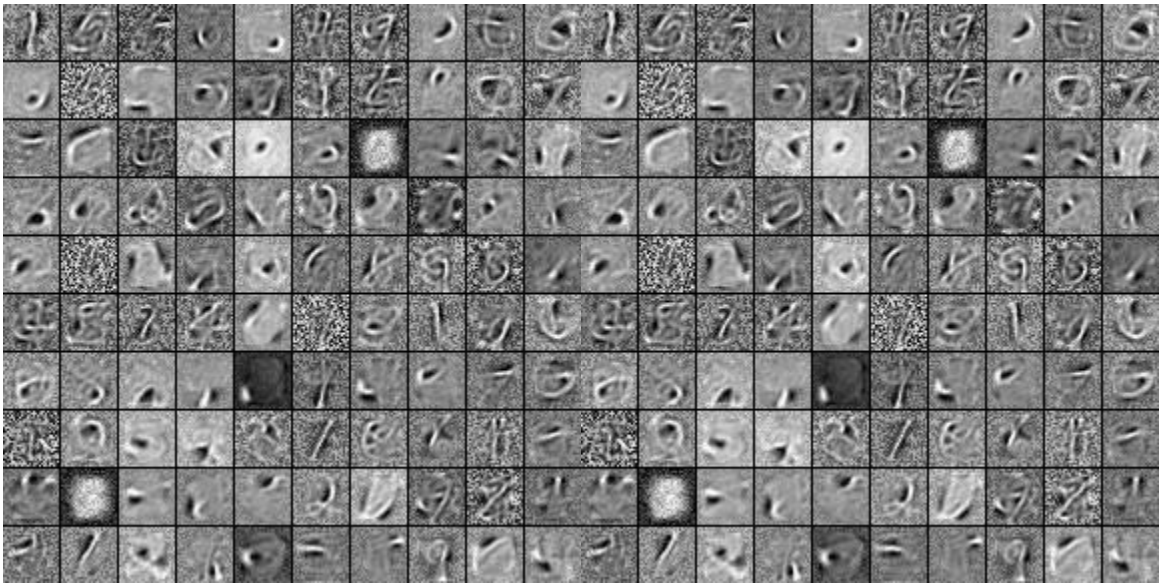


Epoch 1

Epoch 2

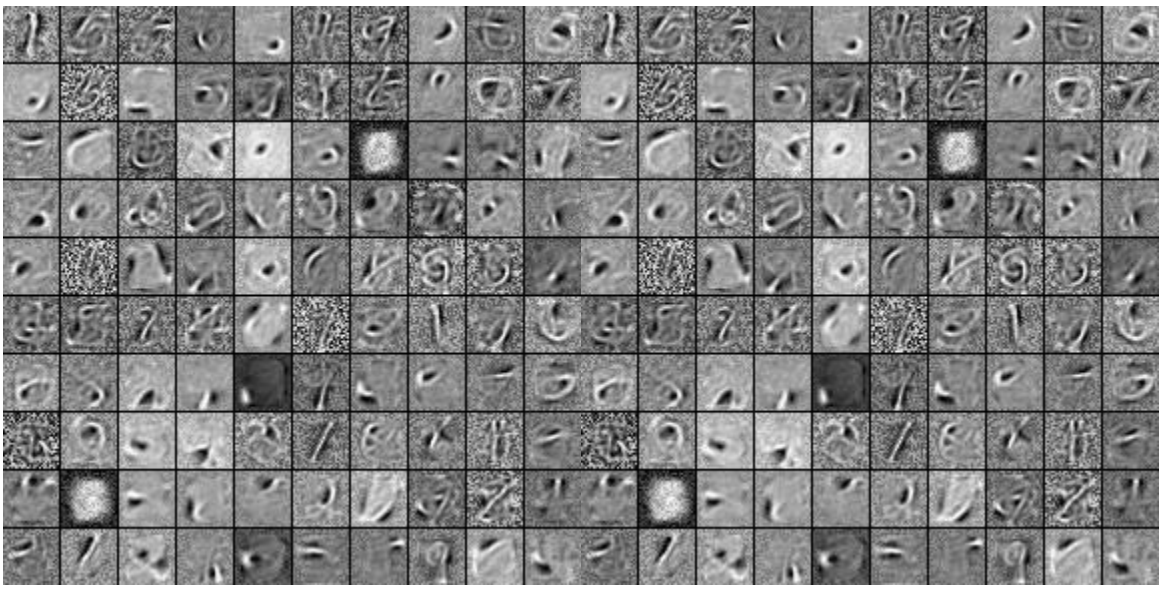
Epoch 3

Epoch 4



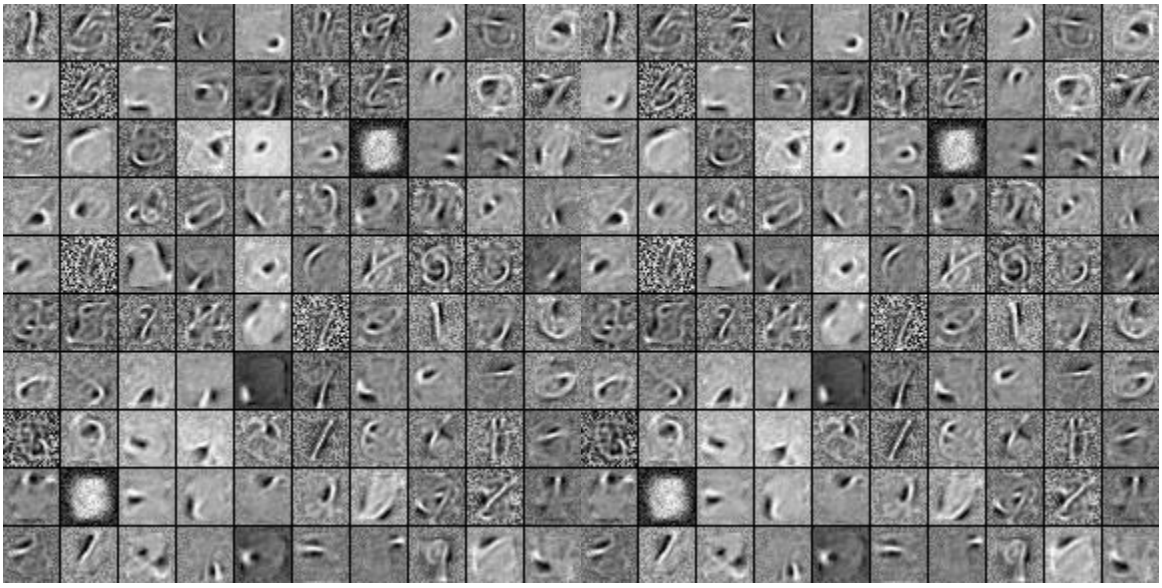
Epoch 5

Epoch 6



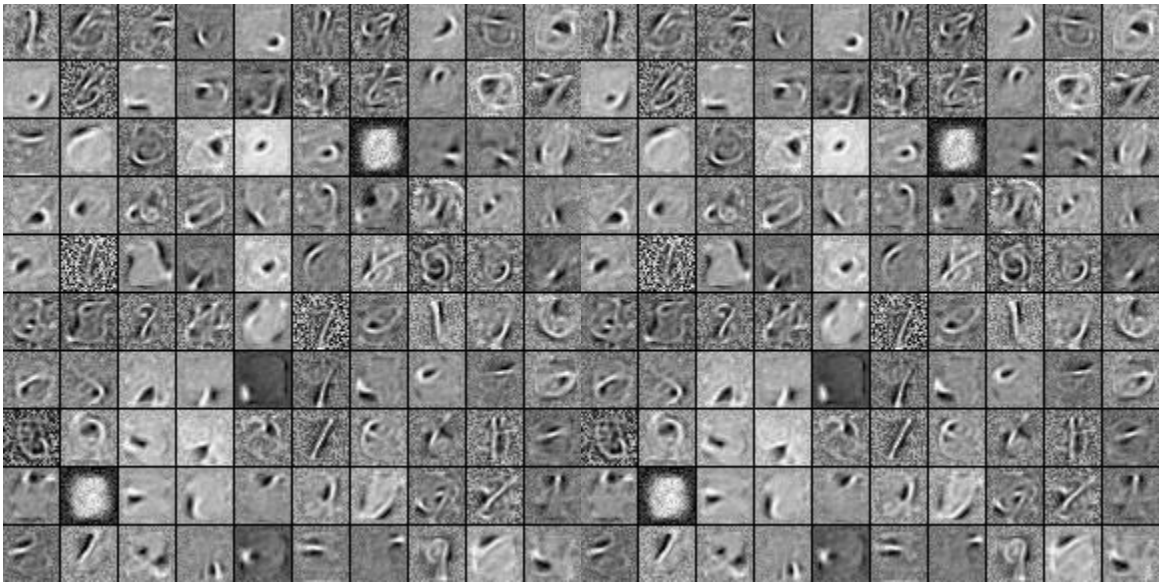
Epoch 7

Epoch 8



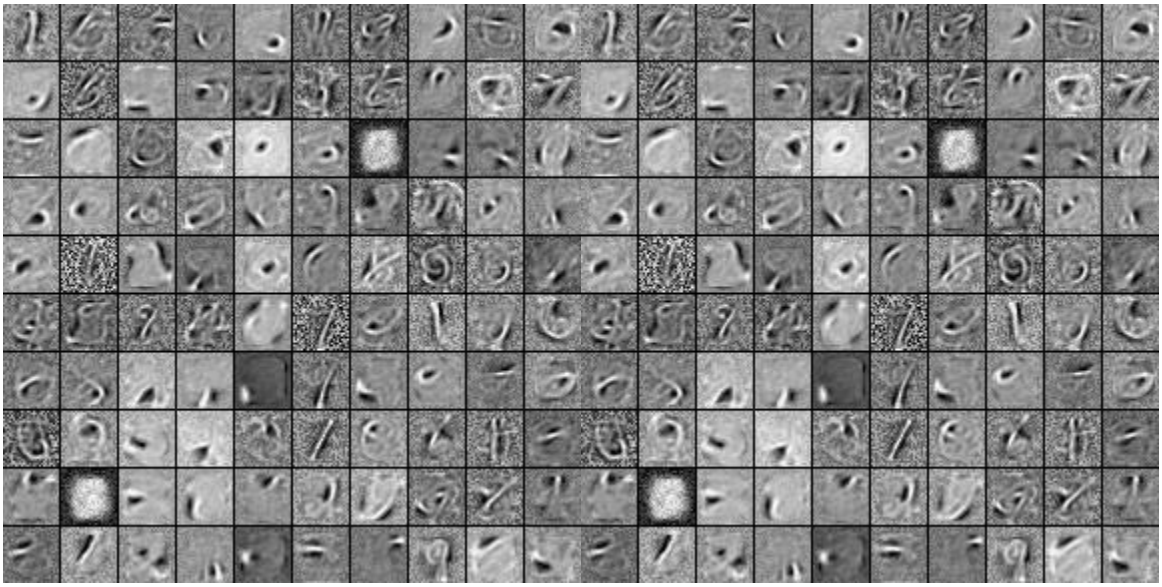
Epoch 9

Epoch 10



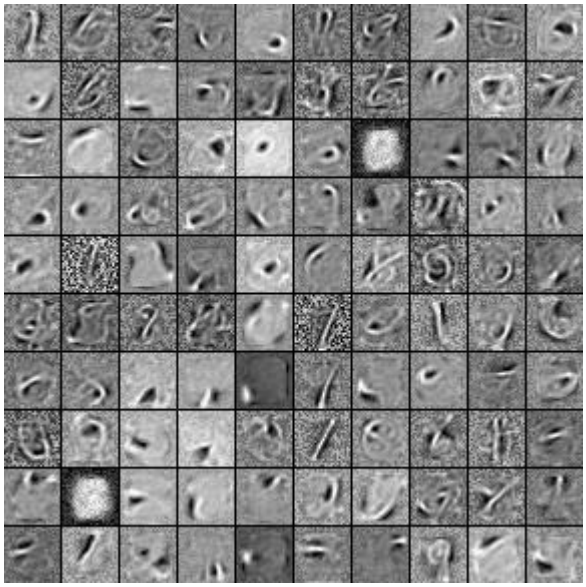
Epoch 11

Epoch 12

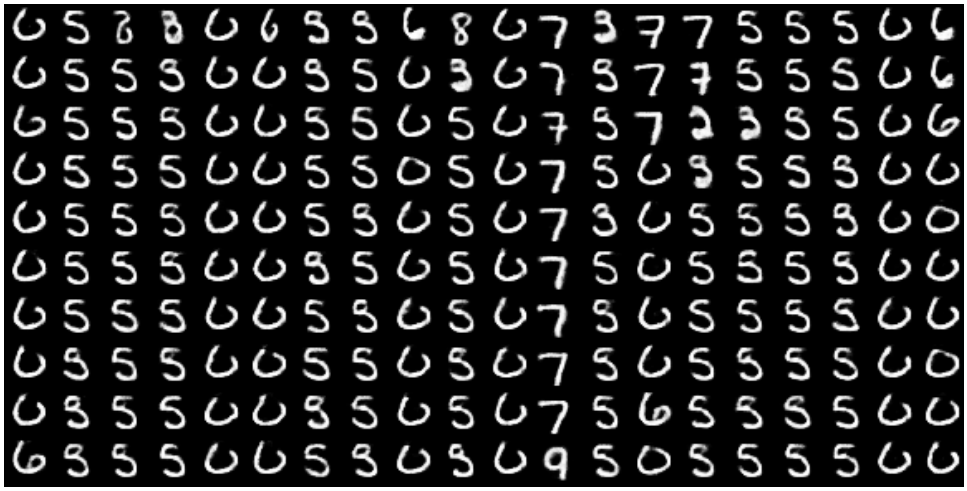


Epoch 13

Epoch 14



Reconstructed Output



The total time taken for the training was about 45 minutes.

Acknowledgements: -

The code majorly constructed using the tutorials and the libraries developed by the **Lisa labs**

The link for their GitHub page is

http://github.com/lisa-lab/DeepLearningTutorials

Link to all the assignments done by me during deep learning course at IIST under **Deepak Mishra Sir** at **IIST is**

https://github.com/mohit1soni/DeepLearningAssignments