vae_0

October 16, 2018

1 Importing the packages

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread , imresize
import os
import time
```

2 Parameters

In [57]:
```
no_latent = 10
lr = 1e-3
epochs = 100
```

3 Declaring functions

In [3]:
```
# Sigmoid

def sigmoid(x):
    return 1/(1 + np.exp(-x))

#derivative of sigmoid
def deriv_sigmoid(x):
    return np.exp(-x)/((1+ np.multiply(np.exp(-x),np.exp(-x))))

# softmax

def softmax(x):
    out = np.zeros(x.shape)
    for i in range(0,x.shape[0]):
        for j in range(0,x.shape[1]):
            out[i,j] = np.exp(x[i,j])/np.sum(np.exp(x[i]))
    return out

# sum of Squared error
```

```
        def squared_error(y_train, y_predicted):
            return np.sum(np.multiply(y_train - y_predicted , y_train - y_predicted))
```

In [21]:
```
def load_flattened_images(Loc):
    Images = []
    for root, dirs, files in os.walk(Loc):
        for file in files:
            Image = imread(os.path.join(root, file))
            Image = Image / 255.0
            Images.append(Image.flatten())

    Images = np.asmatrix(Images)
    print(Images.shape)

    return Images
```

In [90]:
```
## fitting the model

def net_fit_vae(x_train , y_train , epochs = 100 , no_latent = 10 , lr = 1e-3):
    input_dim = x_train.shape[1]
    training_samples = x_train.shape[0]
    output_dim = y_train.shape[1]
    costs = []
    x_train = np.hstack((np.ones((training_samples , 1)), x_train))
    #initializig the parameters
    alpha = np.asmatrix(np.random.normal(0,1e-5,(input_dim + 1 , no_latent)))
    gaama = np.asmatrix(np.random.normal(0,1e-5,(input_dim + 1 , no_latent)))
    beta = np.asmatrix(np.random.normal(0,1e-5,(no_latent+1 , output_dim)))

    #looping for number of itretions
    for epoch in range(0,epochs):
        mu_raw = x_train * alpha
        mu = sigmoid(mu_raw)
        sigma_raw = x_train * gaama
        sigma = sigmoid(sigma_raw)

        #finding z matrix
        z = np.multiply((np.repeat(np.asmatrix(np.random.multivariate_normal(np.zeros((
        z_biased = np.asmatrix(np.hstack((np.ones((training_samples,1)),z)))

        #finding y matrix
        y_raw = z_biased * beta
        y_predicted = sigmoid(y_raw)

        ##finding the cost
        cost = squared_error(y_train , y_predicted) + 0.5*(np.sum(np.square(mu)) + np.s
        costs.append(cost)
        #finding gradient w.r.t beta
```

```
            delta = np.multiply((y_predicted - y_train), deriv_sigmoid(y_raw))
            d_beta = z_biased.T * delta

            temp_beta = beta[1:,:]

            #finding gradient w.r.t alpha
            ss_alpha = np.multiply((delta * temp_beta.T),deriv_sigmoid(mu_raw))
            d_alpha = x_train.T * (ss_alpha + np.multiply(mu , deriv_sigmoid(mu_raw)))

            #finding gradient w.r.t gaama
            ss_beta = np.multiply((delta * temp_beta.T),deriv_sigmoid(sigma_raw))
            d_gaama = x_train.T * (0.5* np.multiply(ss_beta , np.exp(0.5 * sigma)) + np.mul

            #updating the weights
            beta = beta - lr * d_beta
            alpha = alpha - lr*d_alpha
            gaama = gaama - lr*d_gaama
#           print(np.max(alpha) , np.max(beta) , np.min(alpha) , np.min(beta))
            print("\nEpoch: " + str(epoch+1) + "   cost : " + str(cost))
        return beta , costs
```

In [91]:
```
#prediction

def net_generate(number , weights , no_latent):
    generated = []
    for i in range(0,number):
        x = np.asmatrix(np.random.multivariate_normal(np.zeros((no_latent)) , np.eye(no
        x = np.asmatrix(np.hstack(([[1]],x)))
        gen = sigmoid(x * weights)
        gen_image = gen.reshape(28,28)
        generated.append(gen_image)
    return generated
```

4 Generating training data

In [97]:
```
# x_train = load_flattened_images("/home/snehith/Documents/machine learning/datasets/mn
# np.save("train_set_vae0.npy" , x_train)
x_train = np.asmatrix(np.load("train_set_vae0.npy"))
print(x_train.shape)
```

```
(4132, 784)
```

5 Training the Model

In [98]:
```
# alpha , beta , losses = net_fit(x_Train , y_train_and , hidden_nodes = hidden_nodes ,
tic = time.time()
```

```
weights , lossses = net_fit_vae(x_train , x_train , no_latent = no_latent , epochs = ep
print("time taken: "+ str(time.time() - tic) + "sec")
# print("\nalpha:\n",alpha ,"\nbeta:\n", beta,"\n" ,"\nloss:\n", losses[epochs-1])
np.save("weights_vae.npy" , weights)
# np.save("beta_weights_vae.npy" , beta)
```

```
Epoch: 1   cost : 725086.136243532

Epoch: 2   cost : 322541.51573729515

Epoch: 3   cost : 669175.7539098519

Epoch: 4   cost : 217094.72920800574

Epoch: 5   cost : 235249.71285442586

Epoch: 6   cost : 207249.00742859082

Epoch: 7   cost : 202111.12168735985

Epoch: 8   cost : 1352057.1311466217

Epoch: 9   cost : 893579.1080725547

Epoch: 10   cost : 216532.42286970024

Epoch: 11   cost : 239362.49715282736

Epoch: 12   cost : 293122.44507299695

Epoch: 13   cost : 216216.97785743725

Epoch: 14   cost : 1618535.8314190523

Epoch: 15   cost : 340584.01575966296

Epoch: 16   cost : 1929043.3005736817

Epoch: 17   cost : 263063.90100896836

Epoch: 18   cost : 278276.83750033745

Epoch: 19   cost : 455252.4498715482

Epoch: 20   cost : 266590.7384369603

Epoch: 21   cost : 342828.47516364977
```

```
Epoch: 22   cost : 341424.9807052815

Epoch: 23   cost : 242469.1570861982

Epoch: 24   cost : 268553.1406988095

Epoch: 25   cost : 613325.8634369563

Epoch: 26   cost : 335122.8197375136

Epoch: 27   cost : 327478.1361114827

Epoch: 28   cost : 270573.4093212919

Epoch: 29   cost : 303350.3632110347

Epoch: 30   cost : 313924.2986322339

Epoch: 31   cost : 510444.90961341193

Epoch: 32   cost : 273754.20274591004

Epoch: 33   cost : 269493.0551844204

Epoch: 34   cost : 465949.97291683586

Epoch: 35   cost : 269408.1866637219

Epoch: 36   cost : 309300.84601512825

Epoch: 37   cost : 315044.0857985285

Epoch: 38   cost : 292167.88923250284

Epoch: 39   cost : 265703.69650440343

Epoch: 40   cost : 314929.6049868213

Epoch: 41   cost : 276949.5077828565

Epoch: 42   cost : 293999.8293699517

Epoch: 43   cost : 272338.56761919986

Epoch: 44   cost : 278515.2582508874

Epoch: 45   cost : 265505.519945937
```

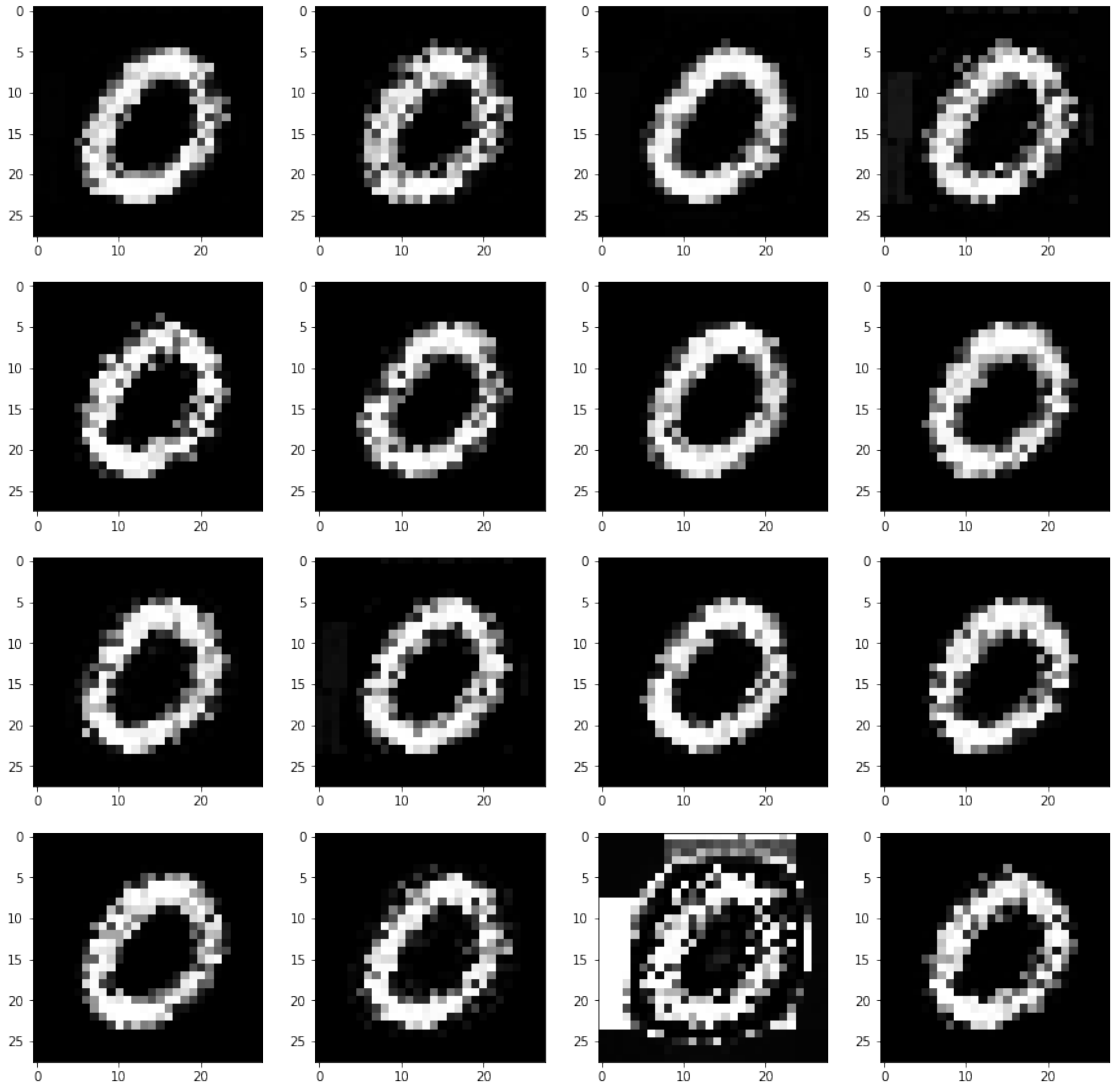
```
Epoch: 46   cost : 315889.0693662229

Epoch: 47   cost : 291503.5590985461

Epoch: 48   cost : 306417.650716276

Epoch: 49   cost : 301936.5540661765

Epoch: 50   cost : 293164.8093645576

Epoch: 51   cost : 260634.44523851894

Epoch: 52   cost : 280865.2499739564

Epoch: 53   cost : 287546.406339641

Epoch: 54   cost : 304791.263398947

Epoch: 55   cost : 298195.3723381533

Epoch: 56   cost : 263686.7321894386

Epoch: 57   cost : 304316.8777188359

Epoch: 58   cost : 291130.1328634908

Epoch: 59   cost : 308590.22311147535

Epoch: 60   cost : 295632.9259527002

Epoch: 61   cost : 308008.9466568278

Epoch: 62   cost : 286668.5940175907

Epoch: 63   cost : 292622.22845959343

Epoch: 64   cost : 295329.0954415212

Epoch: 65   cost : 380408.1439519043

Epoch: 66   cost : 353012.7344745292

Epoch: 67   cost : 303965.645655741

Epoch: 68   cost : 272720.75464770774

Epoch: 69   cost : 284110.41535085917
```

```
Epoch: 70   cost : 276925.9681618149

Epoch: 71   cost : 267541.15657566546

Epoch: 72   cost : 279118.1058612356

Epoch: 73   cost : 280200.2389221098

Epoch: 74   cost : 270328.0977017689

Epoch: 75   cost : 307958.0583830132

Epoch: 76   cost : 318970.71227484755

Epoch: 77   cost : 280599.5875669026

Epoch: 78   cost : 302277.2828655003

Epoch: 79   cost : 281598.53797577956

Epoch: 80   cost : 276972.9299321137

Epoch: 81   cost : 319525.1748981624

Epoch: 82   cost : 296603.8190063994

Epoch: 83   cost : 284579.55915852654

Epoch: 84   cost : 303830.6394183136

Epoch: 85   cost : 1133553.5228465071

Epoch: 86   cost : 292639.3053169087

Epoch: 87   cost : 290533.8826376454

Epoch: 88   cost : 2104508.6013035034

Epoch: 89   cost : 287252.76202024845

Epoch: 90   cost : 284428.060792505

Epoch: 91   cost : 291757.6743289549

Epoch: 92   cost : 279999.12864627474

Epoch: 93   cost : 290301.0086395279
```

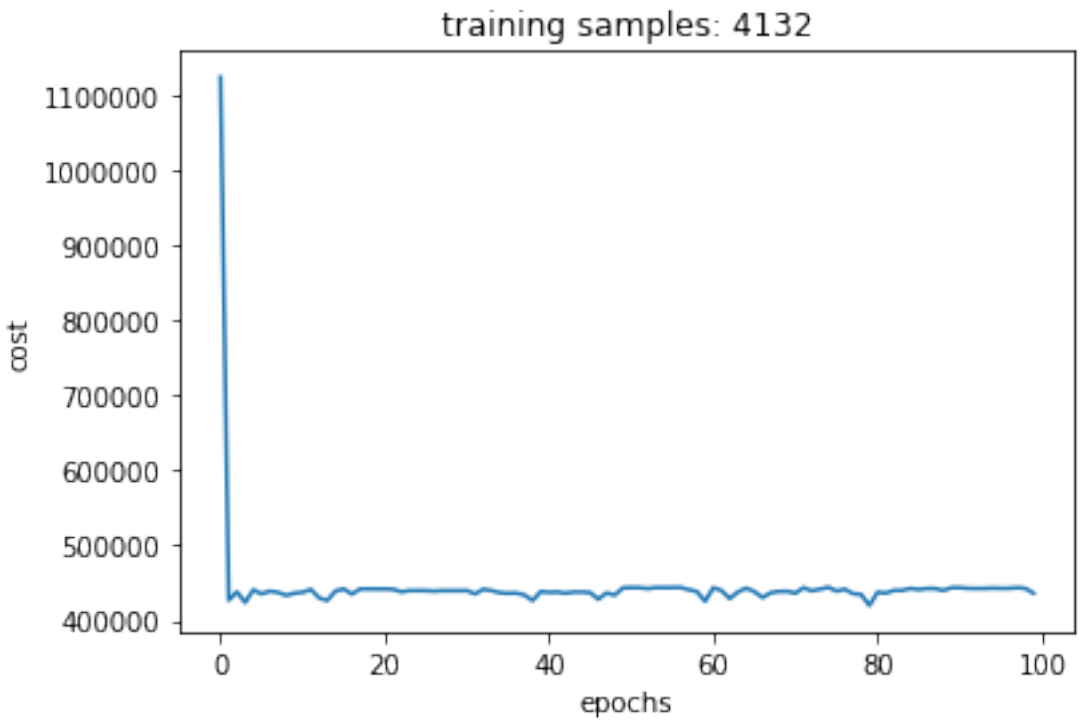
```
Epoch: 94   cost : 287900.6059180534

Epoch: 95   cost : 287993.72173146374

Epoch: 96   cost : 638460.4617442155

Epoch: 97   cost : 426162.55763212324

Epoch: 98   cost : 300674.30607757566

Epoch: 99   cost : 278225.888152339

Epoch: 100   cost : 292688.9862684653
time taken: 48.27308440208435sec
```

5.0.1 Predicting

```
In [102]: #testing samples
          #predicting the output
          result = net_generate(16 , weights , no_latent)
          # print(np.max(res)  , np.max(x_test))
          # print(losses.shape)
          plt.figure(figsize = (15,15))
          for i in range(0,16):
              plt.subplot(4,4,i+1)
              plt.imshow(result[i] , cmap = 'gray')
```



```
In [100]: #ploting the cost vs epochs
          plt.plot(np.arange(epochs),costs)
          # plt.plot(np.arange(epochs),means)
          plt.title("training samples: " + str(x_train.shape[0]))
          plt.xlabel("epochs")
          plt.ylabel("cost")
          plt.show()
```


training samples: 4132
1100000
1000000
900000
800000
700000
600000
500000
400000
cost
0
20
40
60
80
100
epochs