Autoencoders

December 22, 2017

1 Autoencoders

1.1 Import the appropriate modules

```
In [1]: from __future__ import print_function, division
        from builtins import range, input
        import numpy as np
        import theano
        import theano.tensor as T
        import matplotlib.pyplot as plt
        from sklearn.utils import shuffle
        from util import relu, error_rate, getKaggleMNIST, init_weights
```

1.2 AutoEncoder Class

```
In [2]: class AutoEncoder(object):
            """
            M is an arbitrary parameter
            an_id will be used to set names of theano variables
            """
            def __init__(self, M, an_id):
                self.M = M
                self.id = an_id
            """
            Unsupervised, so only takes X
            Epoch and learning rate can be adjusted
            """
            def fit(self, X, learning_rate=0.5, mu=0.99,
                    epochs=1, batch_sz=100, show_fig=False):
                N, D = X.shape
                n_batches = N // batch_sz

                # Initialize weights
                W0 = init_weights((D, self.M))
                # Theano variables
                self.W = theano.shared(W0, 'W_%s' % self.id)
                self.bh = theano.shared(
                    np.zeros(self.M), 'bh_%s' % self.id)
```

```
self.bo = theano.shared(
    np.zeros(D), 'bo_%s' % self.id)
# Save these in an array for use in the gradient
# descent part of this process.
self.params = [self.W, self.bh, self.bo]
self.forward_params = [self.W, self.bh]

# TODO: technically these should be reset before doing backprop
# These are defined for the use of momentum
self.dW = theano.shared(
    np.zeros(W0.shape), 'dW_%s' % self.id)
self.dbh = theano.shared(
    np.zeros(self.M), 'dbh_%s' % self.id)
self.dbo = theano.shared(
    np.zeros(D), 'dbo_%s' % self.id)
# Parameters collected for use in gradient descent
self.dparams = [self.dW, self.dbh, self.dbo]
self.forward_dparams = [self.dW, self.dbh]

# Define tensor input
X_in = T.matrix('X_%s' % self.id)
X_hat = self.forward_output(X_in)

# attach it to the object so it can be used later
# must be sigmoidal because the output is also a sigmoid!
H = T.nnet.sigmoid(X_in.dot(self.W) + self.bh)
self.hidden_op = theano.function(
    inputs=[X_in],
    outputs=H,)

# save this for later so we can call it to
# create reconstructions of input
self.predict = theano.function(
    inputs=[X_in],
    outputs=X_hat,)

# Can also use cross entropy here if needed:
cost=-(X_in*T.log(X_hat)+(1-X_in)*
         T.log(1-X_hat)).sum()  / N
#cost = -(X_in * T.log(X_hat)
 #        + (1 - X_in)
  #       * T.log(1 - X_hat)).flatten().mean()
cost_op = theano.function(
    inputs=[X_in],
    outputs=cost,)

# Gradient Descent
updates = [(p, p + mu*dp - learning_rate*T.grad(cost, p)
                ) for p, dp in zip(self.params, self.dparams)]
        + [(dp, mu*dp - learning_rate*T.grad(cost, p)
            ) for p, dp in zip(self.params, self.dparams)]

        # Train function
        train_op = theano.function(
            inputs=[X_in],
            updates=updates,)

        # Saving costs to plot later
        costs = []
        print("training autoencoder: %s" % self.id)
        for i in range(epochs):
            print("epoch:", i)
            X = shuffle(X)
            for j in range(n_batches):
                batch = X[j*batch_sz:(j*batch_sz + batch_sz)]
                train_op(batch)
                the_cost = cost_op(batch)
                if(j%400 == 0 ):
                    if j % 10 == 0:
                        print("j / n_batches:", j,
                              "/", n_batches, "cost:", the_cost)
                costs.append(the_cost)
        if show_fig:
            plt.plot(costs)
            plt.show()

    """
    Hidden layers ( sigmoid )
    """
    def forward_hidden(self, X):
        Z = T.nnet.sigmoid(X.dot(self.W) + self.bh)
        return Z

    """
    Output layers ( sigmoid )
    """
    def forward_output(self, X):
        Z = self.forward_hidden(X)
        Y = T.nnet.sigmoid(Z.dot(self.W.T) + self.bo)
        return Y

    @staticmethod
    def createFromArrays(W, bh, bo, an_id):
        ae = AutoEncoder(W.shape[1], an_id)
        ae.W = theano.shared(W, 'W_%s' % ae.id)
        ae.bh = theano.shared(bh, 'bh_%s' % ae.id)
        ae.bo = theano.shared(bo, 'bo_%s' % ae.id)
        ae.params = [ae.W, ae.bh, ae.bo]
        ae.forward_params = [ae.W, ae.bh]
        return ae
```

1.3 DNN Class

```
In [3]: class DNN(object):
            """
            hidden layer sizes will be an array of numbers
            """
            def __init__(self, hidden_layer_sizes,
                         UnsupervisedModel=AutoEncoder):
                self.hidden_layers = []
                # Keeps ID of each autoencoder
                count = 0
                for M in hidden_layer_sizes:
                    ae = UnsupervisedModel(M, count)
                    self.hidden_layers.append(ae)
                    count += 1

            """
            Supervised, so we have X and y
            Usually don't want to use test data here.
            This fitting uses momentum and regularization.
            """
            def fit(self, X, Y, Xtest, Ytest,
                    pretrain=True, learning_rate=0.01,
                    mu=0.99, reg=0.1, epochs=1, batch_sz=100):
                # greedy layer-wise training of autoencoders
                pretrain_epochs = 1
                if not pretrain:
                    pretrain_epochs = 0

                current_input = X
                for ae in self.hidden_layers:
                    ae.fit(current_input, epochs=pretrain_epochs)
                    # create current_input for the next layer
                    current_input = ae.hidden_op(current_input)

                # initialize logistic regression layer
                N = len(Y)
                K = len(set(Y))
                W0 = init_weights((self.hidden_layers[-1].M, K))

                # Theano Variables
                self.W = theano.shared(W0, "W_logreg")
                self.b = theano.shared(np.zeros(K), "b_logreg")
```

```python
# Parameters for gradient descent
self.params = [self.W, self.b]
for ae in self.hidden_layers:
    self.params += ae.forward_params

# for momentum
self.dW = theano.shared(
    np.zeros(W0.shape), "dW_logreg")
self.db = theano.shared(
    np.zeros(K), "db_logreg")
self.dparams = [self.dW, self.db]
for ae in self.hidden_layers:
    self.dparams += ae.forward_dparams

# Define input and output tensors
X_in = T.matrix('X_in')
targets = T.ivector('Targets')
pY = self.forward(X_in)

squared_magnitude = [
    (p*p).sum() for p in self.params]
reg_cost = T.sum(squared_magnitude)
cost = -T.mean(
    T.log(pY[T.arange(pY.shape[0]),
             targets]))+reg_cost
prediction = self.predict(X_in)

# Regularization cost
cost_predict_op = theano.function(
    inputs=[X_in, targets],
    outputs=[cost, prediction],)

updates = [(p, p + mu*dp - learning_rate*T.grad(cost, p)
           ) for p, dp in zip(self.params, self.dparams)]
+ [(dp, mu*dp - learning_rate*T.grad(cost, p)
   ) for p, dp in zip(self.params, self.dparams)]
# updates = [(p, p - learning_rate*
#    T.grad(cost, p)) for p in self.params]
train_op = theano.function(
    inputs=[X_in, targets],
    updates=updates,)

n_batches = N // batch_sz
costs = []
print("supervised training...")
for i in range(epochs):
    print("epoch:", i)
```

```
            X, Y = shuffle(X, Y)
            for j in range(n_batches):
                Xbatch = X[j*batch_sz:(j*batch_sz + batch_sz)]
                Ybatch = Y[j*batch_sz:(j*batch_sz + batch_sz)]
                train_op(Xbatch, Ybatch)
                the_cost, the_prediction = cost_predict_op(Xtest, Ytest)
                error = error_rate(the_prediction, Ytest)
                if(j%400==0):
                    print("j / n_batches:", j, "/",
                          n_batches, "cost:", the_cost, "error:", error)
                costs.append(the_cost)
        plt.plot(costs)
        plt.show()

    """
    Prediction
    """
    def predict(self, X):
        return T.argmax(self.forward(X), axis=1)

    """
    Next node
    """
    def forward(self, X):
        current_input = X
        for ae in self.hidden_layers:
            Z = ae.forward_hidden(current_input)
            current_input = Z

        # logistic layer
        Y = T.nnet.softmax(T.dot(current_input, self.W) + self.b)
        return Y
```

1.4 Testing a single autoencoder

```
In [4]: def test_single_autoencoder():
            Xtrain, Ytrain, Xtest, Ytest = getKaggleMNIST()

            autoencoder = AutoEncoder(300, 0)
            autoencoder.fit(Xtrain, epochs=2, show_fig=True)

            done = False
            while not done:
                i = np.random.choice(len(Xtest))
                x = Xtest[i]
                y = autoencoder.predict([x])
                plt.subplot(1,2,1)
                plt.imshow(x.reshape(28,28), cmap='gray')
            plt.title('Original')

            plt.subplot(1,2,2)
            plt.imshow(y.reshape(28,28), cmap='gray')
            plt.title('Reconstructed')

            plt.show()

            ans = input("Generate another?")
            if ans and ans[0] in ('n' or 'N'):
                done = True
```

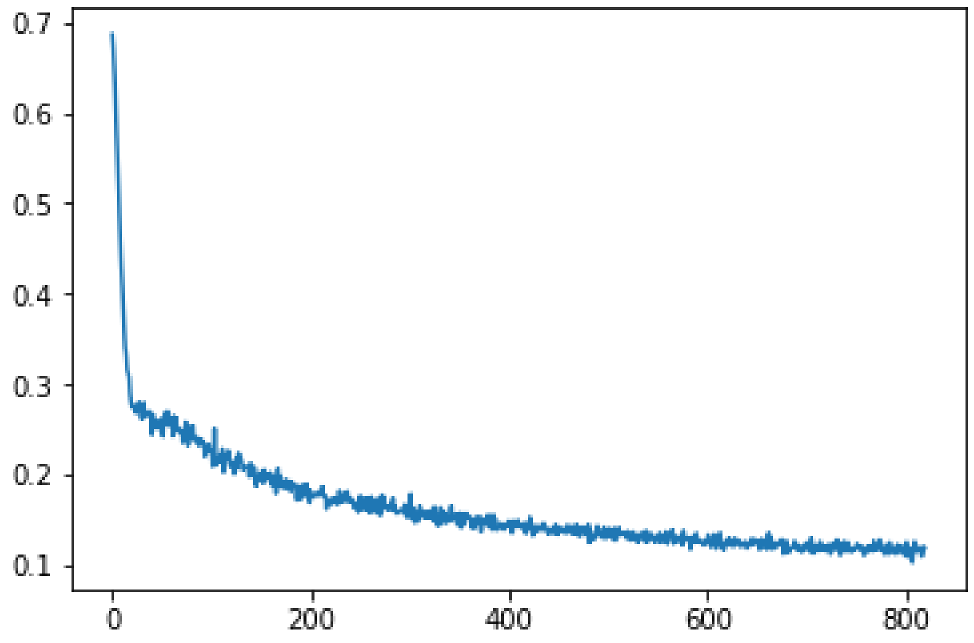
1.5 Running the test

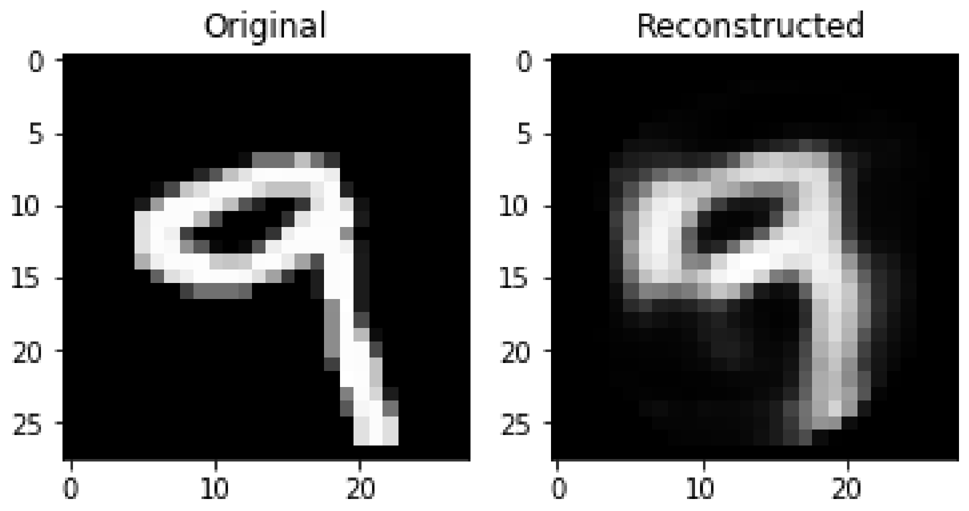
```
In [9]: test_single_autoencoder()
```

```
training autoencoder: 0
epoch: 0
j / n_batches: 0 / 410 cost: 0.6864583118699961
j / n_batches: 10 / 410 cost: 0.4083545705704121
j / n_batches: 20 / 410 cost: 0.27487047764985295
j / n_batches: 30 / 410 cost: 0.2612866868552625
j / n_batches: 40 / 410 cost: 0.2442525046929714
j / n_batches: 50 / 410 cost: 0.2625823924904911
j / n_batches: 60 / 410 cost: 0.2627323088835687
j / n_batches: 70 / 410 cost: 0.24922317948741846
j / n_batches: 80 / 410 cost: 0.24668690594169393
j / n_batches: 90 / 410 cost: 0.23435230221539577
j / n_batches: 100 / 410 cost: 0.22599702608964328
j / n_batches: 110 / 410 cost: 0.2144034667492117
j / n_batches: 120 / 410 cost: 0.2086142283171573
j / n_batches: 130 / 410 cost: 0.215949612274895
j / n_batches: 140 / 410 cost: 0.21212727696929196
j / n_batches: 150 / 410 cost: 0.1963612543874787
j / n_batches: 160 / 410 cost: 0.20187505307670223
j / n_batches: 170 / 410 cost: 0.1926247852954334
j / n_batches: 180 / 410 cost: 0.18188612219283973
j / n_batches: 190 / 410 cost: 0.17305866590304253
j / n_batches: 200 / 410 cost: 0.18162069286156615
j / n_batches: 210 / 410 cost: 0.18287897023227376
j / n_batches: 220 / 410 cost: 0.16841578450521244
j / n_batches: 230 / 410 cost: 0.17216829997298366
j / n_batches: 240 / 410 cost: 0.1749025536865232
j / n_batches: 250 / 410 cost: 0.1770876358835834
j / n_batches: 260 / 410 cost: 0.15796518384386343
j / n_batches: 270 / 410 cost: 0.16729787760253564
j / n_batches: 280 / 410 cost: 0.16715377528452502
j / n_batches: 290 / 410 cost: 0.16109798609028864
```

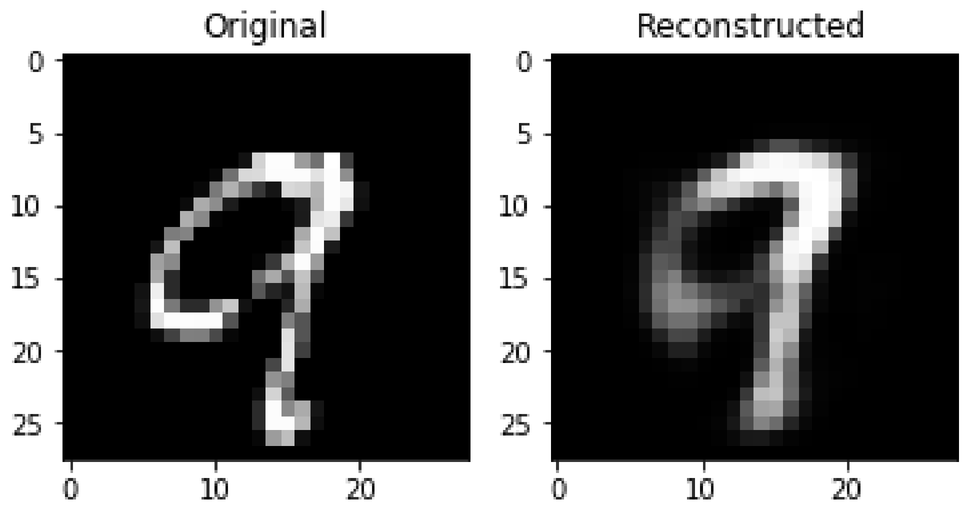
```
j / n_batches: 300 / 410 cost: 0.1602552705329945
j / n_batches: 310 / 410 cost: 0.16578800452184697
j / n_batches: 320 / 410 cost: 0.15996999822075955
j / n_batches: 330 / 410 cost: 0.14497043265334486
j / n_batches: 340 / 410 cost: 0.14925038981073316
j / n_batches: 350 / 410 cost: 0.15799866755001715
j / n_batches: 360 / 410 cost: 0.15423460401872696
j / n_batches: 370 / 410 cost: 0.14569947437960723
j / n_batches: 380 / 410 cost: 0.14286434234284573
j / n_batches: 390 / 410 cost: 0.14232151345659755
j / n_batches: 400 / 410 cost: 0.14935656367765943
epoch: 1
j / n_batches: 0 / 410 cost: 0.1426891373370491
j / n_batches: 10 / 410 cost: 0.13959878924122957
j / n_batches: 20 / 410 cost: 0.1387443103049705
j / n_batches: 30 / 410 cost: 0.134341280857039
j / n_batches: 40 / 410 cost: 0.14029403269525967
j / n_batches: 50 / 410 cost: 0.14006138206619675
j / n_batches: 60 / 410 cost: 0.14077479922775593
j / n_batches: 70 / 410 cost: 0.1452153687563194
j / n_batches: 80 / 410 cost: 0.13769776512859686
j / n_batches: 90 / 410 cost: 0.13290364847947447
j / n_batches: 100 / 410 cost: 0.13872985620831402
j / n_batches: 110 / 410 cost: 0.13747395199815188
j / n_batches: 120 / 410 cost: 0.1254306553766625
j / n_batches: 130 / 410 cost: 0.13312366739530462
j / n_batches: 140 / 410 cost: 0.1294032627560754
j / n_batches: 150 / 410 cost: 0.12377508341318359
j / n_batches: 160 / 410 cost: 0.12874419510862656
j / n_batches: 170 / 410 cost: 0.13080376058878354
j / n_batches: 180 / 410 cost: 0.1285272165624478
j / n_batches: 190 / 410 cost: 0.12429770395396772
j / n_batches: 200 / 410 cost: 0.12248725031231798
j / n_batches: 210 / 410 cost: 0.12347501725268167
j / n_batches: 220 / 410 cost: 0.12320179039398482
j / n_batches: 230 / 410 cost: 0.11930937816884522
j / n_batches: 240 / 410 cost: 0.12759567018606166
j / n_batches: 250 / 410 cost: 0.11813420848168364
j / n_batches: 260 / 410 cost: 0.12041953529593062
j / n_batches: 270 / 410 cost: 0.12710643367264368
j / n_batches: 280 / 410 cost: 0.11772222626107819
j / n_batches: 290 / 410 cost: 0.12510789551871965
j / n_batches: 300 / 410 cost: 0.1202916797628079
j / n_batches: 310 / 410 cost: 0.115885304894515
j / n_batches: 320 / 410 cost: 0.11844020864730756
j / n_batches: 330 / 410 cost: 0.11621788022464621
j / n_batches: 340 / 410 cost: 0.1190199242301229
j / n_batches: 350 / 410 cost: 0.11746316221649122
```

```
j / n_batches: 360 / 410 cost: 0.11948159984961802
j / n_batches: 370 / 410 cost: 0.11843887401738222
j / n_batches: 380 / 410 cost: 0.1125643078090492
j / n_batches: 390 / 410 cost: 0.12452686809768108
j / n_batches: 400 / 410 cost: 0.11922375584212914
```

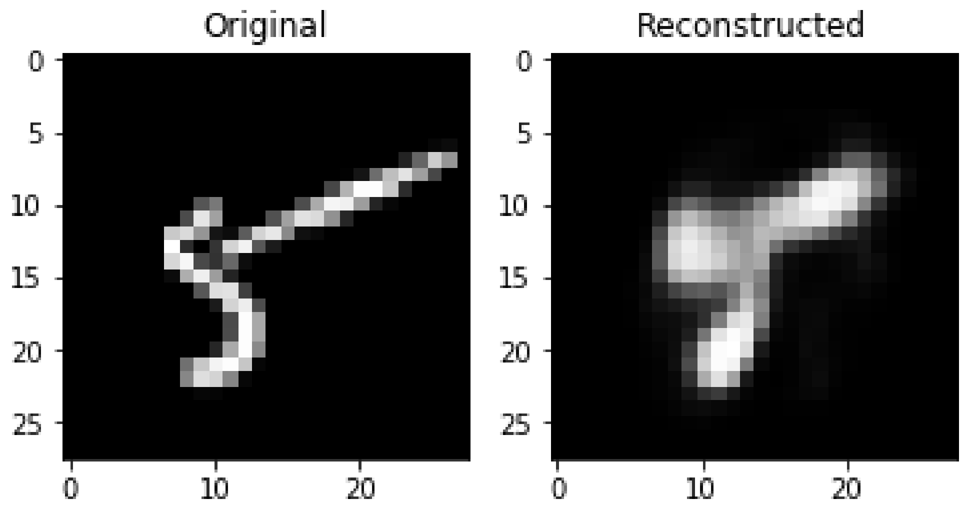




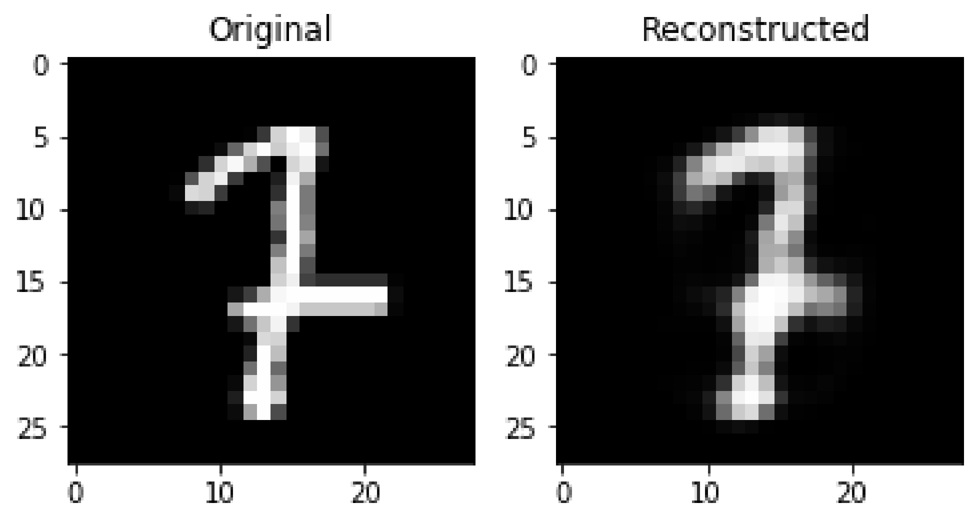
```
Generate another?y
```



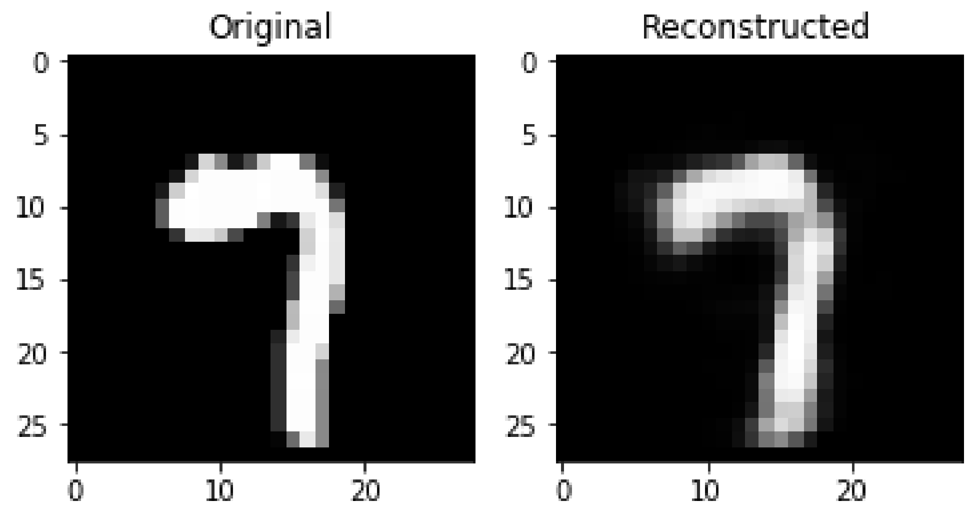
```
Generate another?y
```



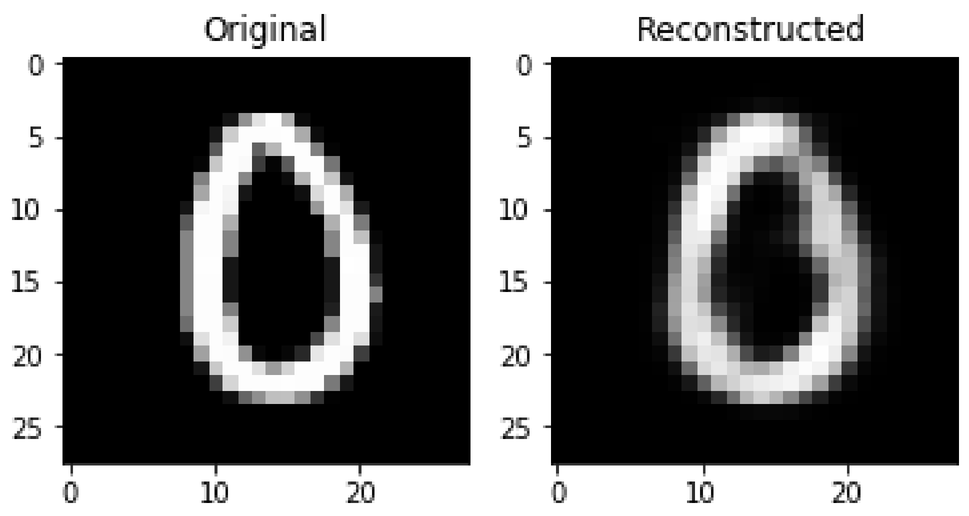
```
Generate another?y
```



```
Generate another?y
```



```
Generate another?y
```



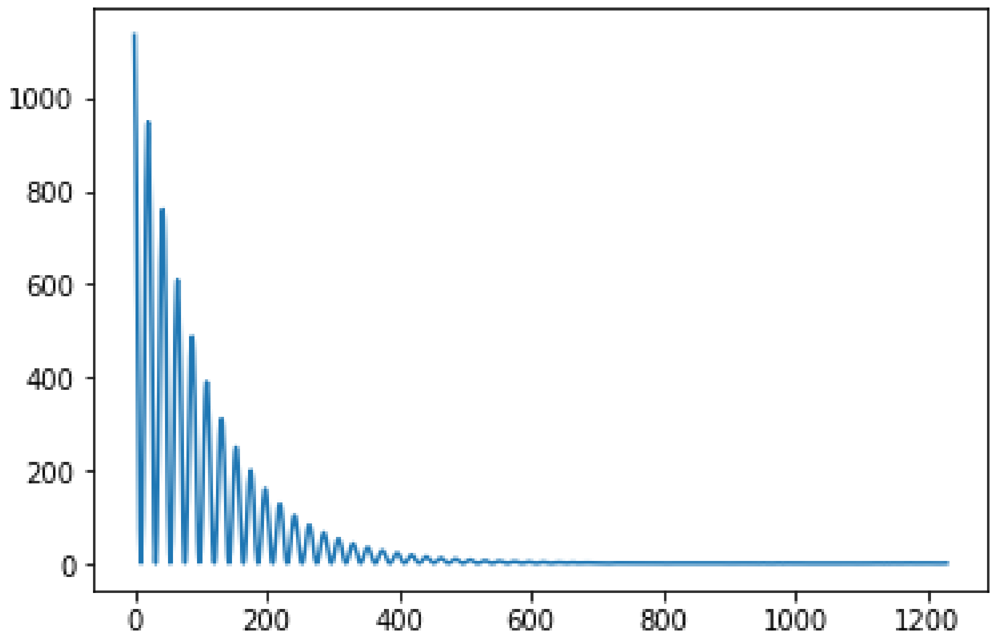
```
Generate another?n
```

Here we see that the autoencoder ran for two epochs and generated a fairly accurate representation of the original numbers.

1.6 Running the AutoEncoder (LONG OUTPUT)

```
In [5]: Xtrain, Ytrain, Xtest, Ytest = getKaggleMNIST()
        # dnn = DNN([1000, 750, 500])
        # dnn.fit(Xtrain, Ytrain, Xtest, Ytest, epochs=3)
        # vs
        dnn = DNN([1000, 750, 500])
        dnn.fit(Xtrain, Ytrain, Xtest,
                Ytest, pretrain=False, epochs=3)
        # You'll definitely want to crank those epochs up.
```

```
training autoencoder: 0
training autoencoder: 1
training autoencoder: 2
supervised training...
epoch: 0
j / n_batches: 0 / 410 cost: 1133.5626991975137 error: 0.917
j / n_batches: 400 / 410 cost: 21.603151058604755 error: 0.887
epoch: 1
j / n_batches: 0 / 410 cost: 3.802571193880928 error: 0.899
j / n_batches: 400 / 410 cost: 2.4457724252970166 error: 0.899
epoch: 2
j / n_batches: 0 / 410 cost: 2.8124273381509677 error: 0.894
j / n_batches: 400 / 410 cost: 2.6643533773882586 error: 0.901
```



2 TensorFlow Autoencoder

2.1 Importing the appropriate modules

```
In [1]: from __future__ import print_function, division
        from builtins import range, input
        import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt
        from sklearn.utils import shuffle
        from util import error_rate, getKaggleMNIST
```

2.2 Autoencoder class

```
In [2]: class AutoEncoder(object):
            """
            Take input variables then call biuld function.
            """
            def __init__(self, D, M, an_id):
                self.M = M
                self.id = an_id
                self.build(D, M)

            """
            Setter for session
```

```
"""
def set_session(self, session):
    self.session = session


"""
Creates tensorFlow variables for each input
"""
def build(self, D, M):
    # instance variables to call later
    self.W = tf.Variable(
        tf.random_normal(shape=(D, M)))
    self.bh = tf.Variable(
        np.zeros(M).astype(np.float32))
    self.bo = tf.Variable(
        np.zeros(D).astype(np.float32))

    self.X_in = tf.placeholder(
        tf.float32, shape=(None, D))
    self.Z = self.forward_hidden(self.X_in) # for transform() later
    self.X_hat = self.forward_output(self.X_in)


    # using the naive formulation for cross-entropy
    # will have numerical stability issues if X_hat = 0 or 1
    logits = self.forward_logits(self.X_in)
    self.cost = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            labels=self.X_in,
            logits=logits,))

    self.train_op = tf.train.AdamOptimizer(
        1e-1).minimize(self.cost)
    # self.train_op = tf.train.MomentumOptimizer(10e-4, momentum=0.9).minimize(self.


"""
Simple because the NN is already built.
Randomizes and calls through loop
"""
def fit(self, X, epochs=1, batch_sz=100, show_fig=False):
    N, D = X.shape
    n_batches = N // batch_sz

    costs = []
    print("training autoencoder: %s" % self.id)
    for i in range(epochs):
        print("epoch:", i)
```

```
            X = shuffle(X)
            for j in range(n_batches):
                batch = X[j*batch_sz:(j*batch_sz + batch_sz)]
                _, c = self.session.run((
                    self.train_op, self.cost),
                    feed_dict={self.X_in: batch})
                if j % 10 == 0:
                    print("j / n_batches:", j, "/", n_batches, "cost:", c)
                costs.append(c)
        if show_fig:
            plt.plot(costs)
            plt.show()


    """
    Numpy to numpy Hidden layer function
    """
    def transform(self, X):
        # accepts and returns a real numpy array
        # unlike forward_hidden and forward_output
        # which deal with tensorflow variables
        return self.session.run(self.Z, feed_dict={self.X_in: X})

    """
    Numpy to numpy Hidden layer function
    """
    def predict(self, X):
        # accepts and returns a real numpy array
        # unlike forward_hidden and forward_output
        # which deal with tensorflow variables
        return self.session.run(self.X_hat, feed_dict={self.X_in: X})

    """
    Tensorflow to tensorflow
    """
    def forward_hidden(self, X):
        Z = tf.nn.sigmoid(tf.matmul(X, self.W) + self.bh)
        return Z

    """
    Tensorflow to tensorflow
    """
    def forward_logits(self, X):
        Z = self.forward_hidden(X)
        return tf.matmul(Z, tf.transpose(self.W)) + self.bo

    """
    Tensorflow to tensorflow
```

```
        """
        def forward_output(self, X):
            return tf.nn.sigmoid(self.forward_logits(X))
```

2.3 DNN Class

```
In [3]: class DNN(object):
            """
            Hidden deep NN
            """
            def __init__(self, D, hidden_layer_sizes,
                         K, UnsupervisedModel=AutoEncoder):
                self.hidden_layers = []
                count = 0
                input_size = D
                for output_size in hidden_layer_sizes:
                    ae = UnsupervisedModel(
                        input_size, output_size, count)
                    self.hidden_layers.append(ae)
                    count += 1
                    input_size = output_size
                self.build_final_layer(D, hidden_layer_sizes[-1], K)

            """
            Setter for session for each hidden layer
            """
            def set_session(self, session):
                self.session = session
                for layer in self.hidden_layers:
                    layer.set_session(session)

            """
            Final layer
            """
            def build_final_layer(self, D, M, K):
                # initialize logistic regression layer
                self.W = tf.Variable(tf.random_normal(shape=(M, K)))
                self.b = tf.Variable(np.zeros(K).astype(np.float32))

                self.X = tf.placeholder(tf.float32, shape=(None, D))
                labels = tf.placeholder(tf.int32, shape=(None,))
                self.Y = labels
                logits = self.forward(self.X)

                self.cost = tf.reduce_mean(
                    tf.nn.sparse_softmax_cross_entropy_with_logits(
                        logits=logits,
                        labels=labels))
```

```
        self.train_op = tf.train.AdamOptimizer(1e-2).minimize(self.cost)
        self.prediction = tf.argmax(logits, 1)

    """
    Takes in train and test, greedy is boolean
    """
    def fit(self, X, Y, Xtest, Ytest,
            pretrain=True, epochs=1, batch_sz=100):
        N = len(X)

        # greedy layer-wise training of autoencoders
        pretrain_epochs = 1
        if not pretrain:
            pretrain_epochs = 0

        current_input = X
        for ae in self.hidden_layers:
            ae.fit(current_input, epochs=pretrain_epochs)

            # create current_input for the next layer
            current_input = ae.transform(current_input)

        n_batches = N // batch_sz
        costs = []
        print("supervised training...")
        for i in range(epochs):
            print("epoch:", i)
            X, Y = shuffle(X, Y)
            for j in range(n_batches):
                Xbatch = X[j*batch_sz:(j*batch_sz + batch_sz)]
                Ybatch = Y[j*batch_sz:(j*batch_sz + batch_sz)]
                self.session.run(
                    self.train_op,
                    feed_dict={self.X: Xbatch, self.Y: Ybatch})
                c, p = self.session.run(
                    (self.cost, self.prediction),
                    feed_dict={self.X: Xtest, self.Y: Ytest})
                error = error_rate(p, Ytest)
                if j % 10 == 0:
                    print("j / n_batches:", j, "/",
                          n_batches, "cost:", c, "error:", error)
                costs.append(c)
        plt.plot(costs)
        plt.show()

    """
    Goes up to logits, TF-TF
    """
```

```
            def forward(self, X):
                current_input = X
                for ae in self.hidden_layers:
                    Z = ae.forward_hidden(current_input)
                    current_input = Z

                # logistic layer
                logits = tf.matmul(current_input, self.W) + self.b
                return logits
```

2.4 Testing the dnn

```
In [4]: def test_pretraining_dnn():
            Xtrain, Ytrain, Xtest, Ytest = getKaggleMNIST()
            # dnn = DNN([1000, 750, 500])
            # dnn.fit(Xtrain, Ytrain, Xtest, Ytest, epochs=3)
            # vs
            Xtrain = Xtrain.astype(np.float32)
            Xtest = Xtest.astype(np.float32)
            _, D = Xtrain.shape
            K = len(set(Ytrain))
            dnn = DNN(D, [1000, 750, 500], K)
            init_op = tf.global_variables_initializer()
            with tf.Session() as session:
                session.run(init_op)
                dnn.set_session(session)
                dnn.fit(Xtrain, Ytrain, Xtest,
                        Ytest, pretrain=True, epochs=10)
```

2.5 Testing a single autoencoder

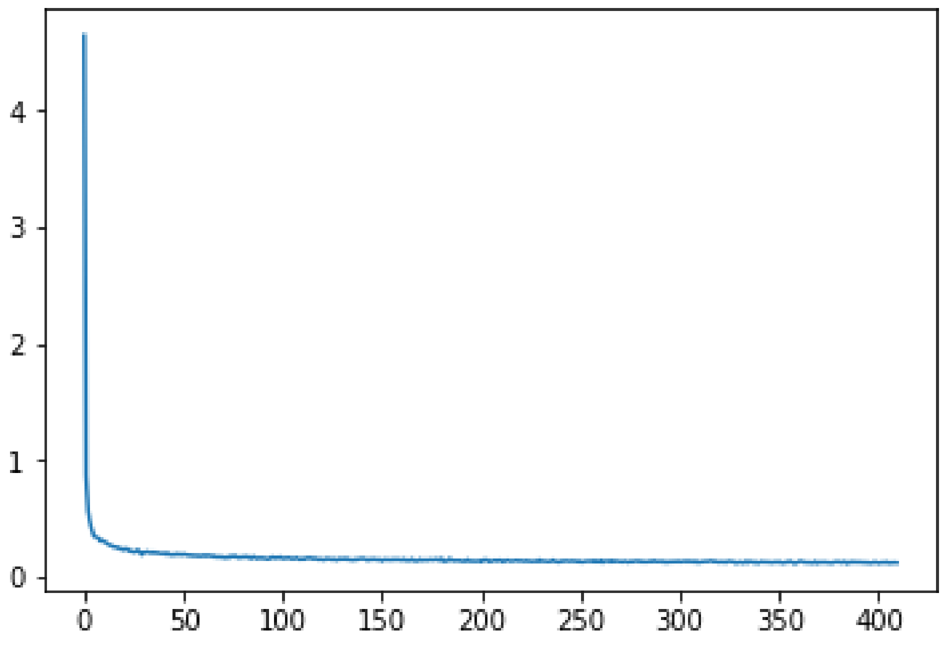
```
In [7]: def test_single_autoencoder():
            Xtrain, Ytrain, Xtest, Ytest = getKaggleMNIST()
            Xtrain = Xtrain.astype(np.float32)
            Xtest = Xtest.astype(np.float32)

            _, D = Xtrain.shape
            print(D)
            autoencoder = AutoEncoder(D, 300,0)
            init_op = tf.global_variables_initializer()
            with tf.Session() as session:
                session.run(init_op)
                autoencoder.set_session(session)
                autoencoder.fit(Xtrain, show_fig=True)

                done = False
                while not done:
                    i = np.random.choice(len(Xtest))
                x = Xtest[i]
                y = autoencoder.predict([x])
                plt.subplot(1,2,1)
                plt.imshow(x.reshape(28,28), cmap='gray')
                plt.title('Original')

                plt.subplot(1,2,2)
                plt.imshow(y.reshape(28,28), cmap='gray')
                plt.title('Reconstructed')

                plt.show()

                ans = input("Generate another?")
                if ans and ans[0] in ('n' or 'N'):
                    done = True
```

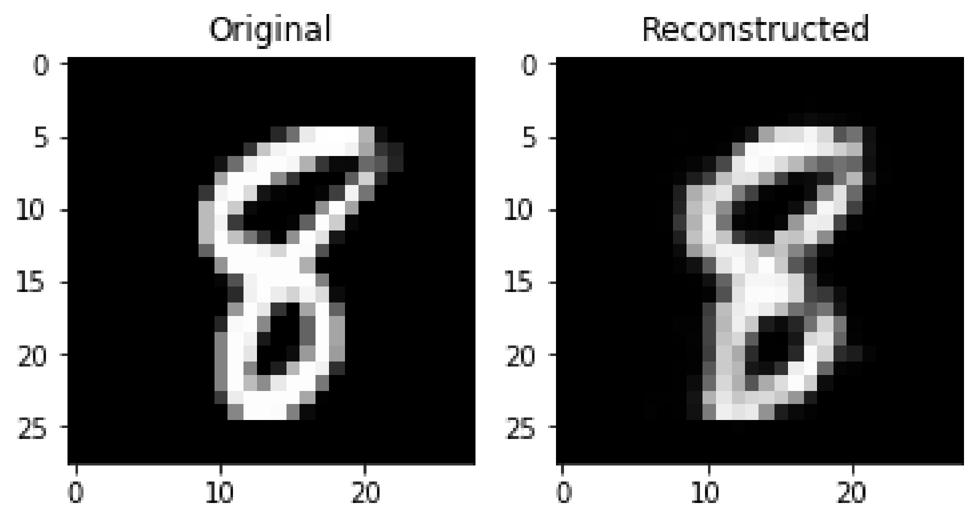
2.6 Running the test

```
In [8]: test_single_autoencoder()
```

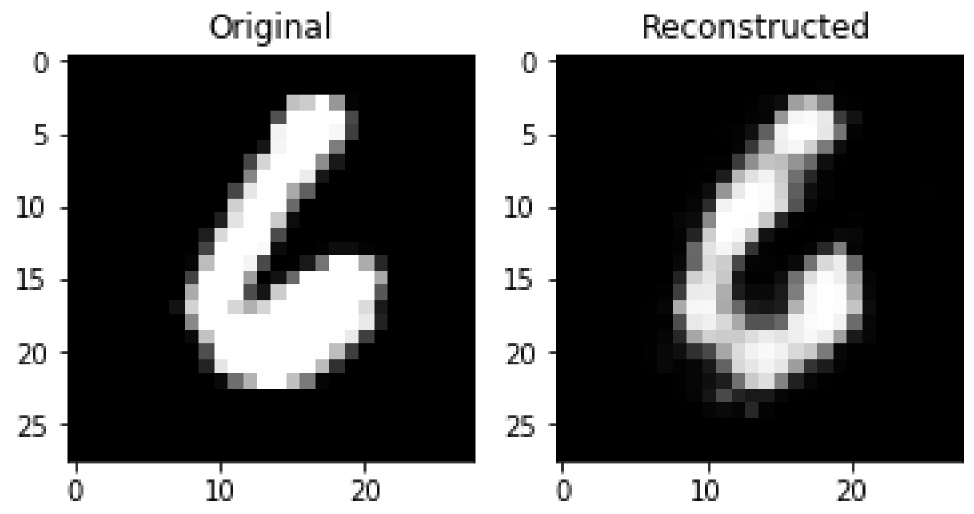
```
784
training autoencoder: 0
epoch: 0
j / n_batches: 0 / 410 cost: 4.64019
j / n_batches: 10 / 410 cost: 0.303542
j / n_batches: 20 / 410 cost: 0.232976
j / n_batches: 30 / 410 cost: 0.212464
j / n_batches: 40 / 410 cost: 0.203958
j / n_batches: 50 / 410 cost: 0.198198
j / n_batches: 60 / 410 cost: 0.186505
j / n_batches: 70 / 410 cost: 0.170297
j / n_batches: 80 / 410 cost: 0.173321
j / n_batches: 90 / 410 cost: 0.161602
j / n_batches: 100 / 410 cost: 0.160524
j / n_batches: 110 / 410 cost: 0.157753
j / n_batches: 120 / 410 cost: 0.151965
j / n_batches: 130 / 410 cost: 0.153976
j / n_batches: 140 / 410 cost: 0.158943
j / n_batches: 150 / 410 cost: 0.138085
j / n_batches: 160 / 410 cost: 0.14589
j / n_batches: 170 / 410 cost: 0.142765
j / n_batches: 180 / 410 cost: 0.159547
j / n_batches: 190 / 410 cost: 0.135499
j / n_batches: 200 / 410 cost: 0.130431
j / n_batches: 210 / 410 cost: 0.142418
j / n_batches: 220 / 410 cost: 0.141606
j / n_batches: 230 / 410 cost: 0.134821
j / n_batches: 240 / 410 cost: 0.139244
```

```
j / n_batches: 250 / 410 cost: 0.129124
j / n_batches: 260 / 410 cost: 0.145343
j / n_batches: 270 / 410 cost: 0.131935
j / n_batches: 280 / 410 cost: 0.128325
j / n_batches: 290 / 410 cost: 0.133767
j / n_batches: 300 / 410 cost: 0.129704
j / n_batches: 310 / 410 cost: 0.127779
j / n_batches: 320 / 410 cost: 0.128025
j / n_batches: 330 / 410 cost: 0.129662
j / n_batches: 340 / 410 cost: 0.124576
j / n_batches: 350 / 410 cost: 0.131516
j / n_batches: 360 / 410 cost: 0.135001
j / n_batches: 370 / 410 cost: 0.127732
j / n_batches: 380 / 410 cost: 0.130802
j / n_batches: 390 / 410 cost: 0.131051
j / n_batches: 400 / 410 cost: 0.118874
```

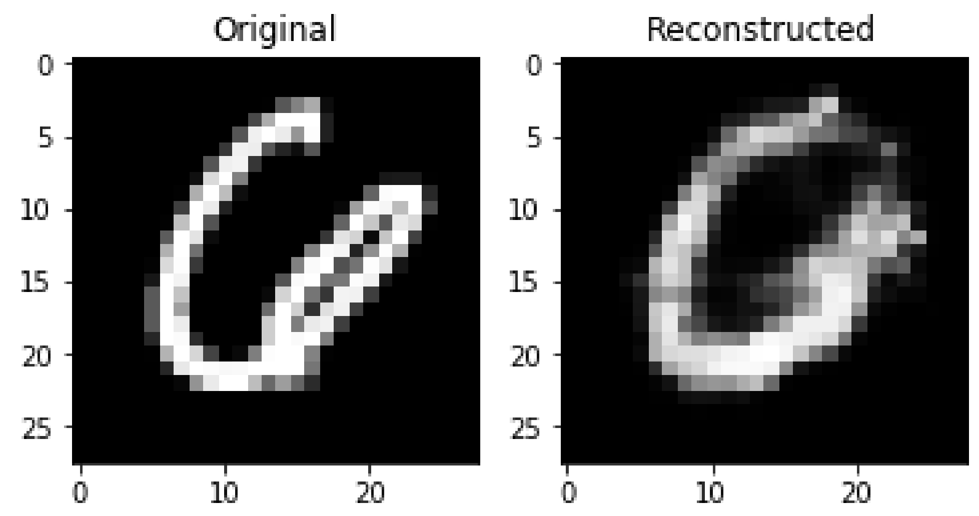




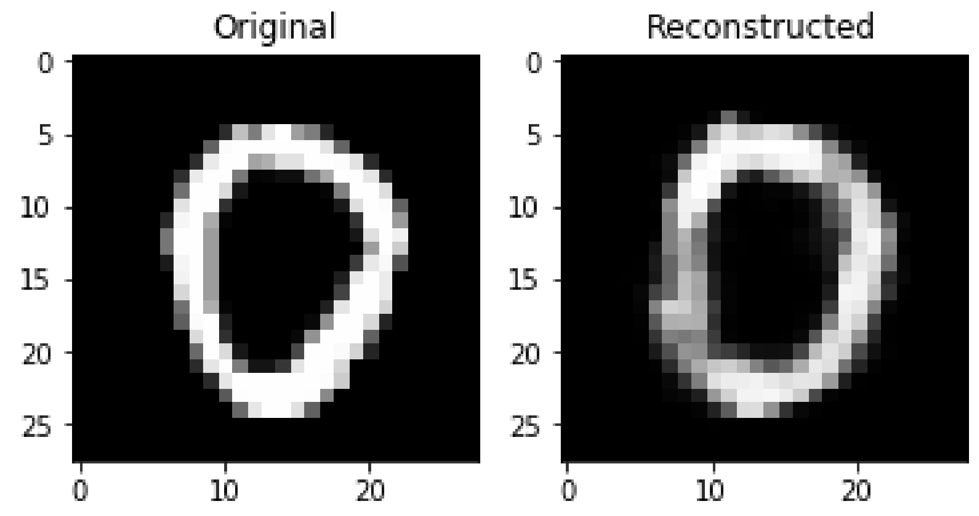
```
Generate another?
```



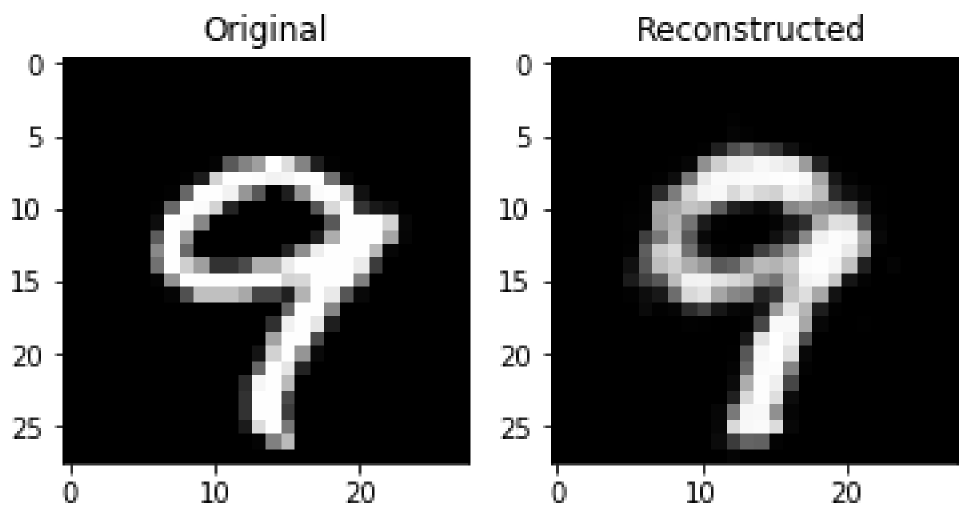
```
Generate another?
```



```
Generate another?
```



```
Generate another?y
```



```
Generate another?n
```

2.7 Running the DNN (LONG OUTPUT)

```
In [9]: test_pretraining_dnn()
```

```
training autoencoder: 0
epoch: 0
j / n_batches: 0 / 410 cost: 8.32554
j / n_batches: 10 / 410 cost: 0.288521
j / n_batches: 20 / 410 cost: 0.240427
j / n_batches: 30 / 410 cost: 0.208209
j / n_batches: 40 / 410 cost: 0.188207
j / n_batches: 50 / 410 cost: 0.197097
j / n_batches: 60 / 410 cost: 0.173092
j / n_batches: 70 / 410 cost: 0.172895
j / n_batches: 80 / 410 cost: 0.150908
j / n_batches: 90 / 410 cost: 0.163357
j / n_batches: 100 / 410 cost: 0.155598
j / n_batches: 110 / 410 cost: 0.15084
j / n_batches: 120 / 410 cost: 0.152299
j / n_batches: 130 / 410 cost: 0.138424
j / n_batches: 140 / 410 cost: 0.14347
j / n_batches: 150 / 410 cost: 0.136149
j / n_batches: 160 / 410 cost: 0.141237
j / n_batches: 170 / 410 cost: 0.134323
j / n_batches: 180 / 410 cost: 0.128744
j / n_batches: 190 / 410 cost: 0.141049
j / n_batches: 200 / 410 cost: 0.129385
```

```
j / n_batches: 210 / 410 cost: 0.133327
j / n_batches: 220 / 410 cost: 0.132665
j / n_batches: 230 / 410 cost: 0.134684
j / n_batches: 240 / 410 cost: 0.134187
j / n_batches: 250 / 410 cost: 0.122915
j / n_batches: 260 / 410 cost: 0.122035
j / n_batches: 270 / 410 cost: 0.122663
j / n_batches: 280 / 410 cost: 0.122112
j / n_batches: 290 / 410 cost: 0.131178
j / n_batches: 300 / 410 cost: 0.118812
j / n_batches: 310 / 410 cost: 0.123254
j / n_batches: 320 / 410 cost: 0.121179
j / n_batches: 330 / 410 cost: 0.123637
j / n_batches: 340 / 410 cost: 0.124784
j / n_batches: 350 / 410 cost: 0.11547
j / n_batches: 360 / 410 cost: 0.118247
j / n_batches: 370 / 410 cost: 0.114037
j / n_batches: 380 / 410 cost: 0.116206
j / n_batches: 390 / 410 cost: 0.122581
j / n_batches: 400 / 410 cost: 0.116064
training autoencoder: 1
epoch: 0
j / n_batches: 0 / 410 cost: 7.29053
j / n_batches: 10 / 410 cost: 0.0539505
j / n_batches: 20 / 410 cost: 0.0354188
j / n_batches: 30 / 410 cost: 0.0293173
j / n_batches: 40 / 410 cost: 0.0287077
j / n_batches: 50 / 410 cost: 0.0226113
j / n_batches: 60 / 410 cost: 0.0208852
j / n_batches: 70 / 410 cost: 0.0216575
j / n_batches: 80 / 410 cost: 0.0189366
j / n_batches: 90 / 410 cost: 0.0188824
j / n_batches: 100 / 410 cost: 0.017168
j / n_batches: 110 / 410 cost: 0.0180982
j / n_batches: 120 / 410 cost: 0.0170854
j / n_batches: 130 / 410 cost: 0.0172834
j / n_batches: 140 / 410 cost: 0.0186722
j / n_batches: 150 / 410 cost: 0.0171835
j / n_batches: 160 / 410 cost: 0.0165436
j / n_batches: 170 / 410 cost: 0.0156026
j / n_batches: 180 / 410 cost: 0.0160052
j / n_batches: 190 / 410 cost: 0.0140984
j / n_batches: 200 / 410 cost: 0.0157722
j / n_batches: 210 / 410 cost: 0.0149274
j / n_batches: 220 / 410 cost: 0.0145885
j / n_batches: 230 / 410 cost: 0.0140103
j / n_batches: 240 / 410 cost: 0.0137
j / n_batches: 250 / 410 cost: 0.014461
```

```
j / n_batches: 260 / 410 cost: 0.0142282
j / n_batches: 270 / 410 cost: 0.0149018
j / n_batches: 280 / 410 cost: 0.0137597
j / n_batches: 290 / 410 cost: 0.0137959
j / n_batches: 300 / 410 cost: 0.0148607
j / n_batches: 310 / 410 cost: 0.0153694
j / n_batches: 320 / 410 cost: 0.0140924
j / n_batches: 330 / 410 cost: 0.0142784
j / n_batches: 340 / 410 cost: 0.0141795
j / n_batches: 350 / 410 cost: 0.0143274
j / n_batches: 360 / 410 cost: 0.0143592
j / n_batches: 370 / 410 cost: 0.0139359
j / n_batches: 380 / 410 cost: 0.0139906
j / n_batches: 390 / 410 cost: 0.0129023
j / n_batches: 400 / 410 cost: 0.013638
training autoencoder: 2
epoch: 0
j / n_batches: 0 / 410 cost: 6.14876
j / n_batches: 10 / 410 cost: 0.164539
j / n_batches: 20 / 410 cost: 0.134975
j / n_batches: 30 / 410 cost: 0.128838
j / n_batches: 40 / 410 cost: 0.115391
j / n_batches: 50 / 410 cost: 0.111785
j / n_batches: 60 / 410 cost: 0.110691
j / n_batches: 70 / 410 cost: 0.110652
j / n_batches: 80 / 410 cost: 0.105406
j / n_batches: 90 / 410 cost: 0.102985
j / n_batches: 100 / 410 cost: 0.0991655
j / n_batches: 110 / 410 cost: 0.0971909
j / n_batches: 120 / 410 cost: 0.100068
j / n_batches: 130 / 410 cost: 0.101509
j / n_batches: 140 / 410 cost: 0.0913432
j / n_batches: 150 / 410 cost: 0.0910203
j / n_batches: 160 / 410 cost: 0.0942244
j / n_batches: 170 / 410 cost: 0.0952375
j / n_batches: 180 / 410 cost: 0.0895847
j / n_batches: 190 / 410 cost: 0.0896965
j / n_batches: 200 / 410 cost: 0.0876174
j / n_batches: 210 / 410 cost: 0.0904252
j / n_batches: 220 / 410 cost: 0.0885089
j / n_batches: 230 / 410 cost: 0.0867954
j / n_batches: 240 / 410 cost: 0.084289
j / n_batches: 250 / 410 cost: 0.0854485
j / n_batches: 260 / 410 cost: 0.0858562
j / n_batches: 270 / 410 cost: 0.0852335
j / n_batches: 280 / 410 cost: 0.0836579
j / n_batches: 290 / 410 cost: 0.0868765
j / n_batches: 300 / 410 cost: 0.0823584
```

```
j / n_batches: 310 / 410 cost: 0.0807055
j / n_batches: 320 / 410 cost: 0.0857531
j / n_batches: 330 / 410 cost: 0.0825072
j / n_batches: 340 / 410 cost: 0.0832088
j / n_batches: 350 / 410 cost: 0.082619
j / n_batches: 360 / 410 cost: 0.0845067
j / n_batches: 370 / 410 cost: 0.0809277
j / n_batches: 380 / 410 cost: 0.0819079
j / n_batches: 390 / 410 cost: 0.0835875
j / n_batches: 400 / 410 cost: 0.0823062
supervised training...
epoch: 0
j / n_batches: 0 / 410 cost: 5.79049 error: 0.883
j / n_batches: 10 / 410 cost: 1.45029 error: 0.421
j / n_batches: 20 / 410 cost: 0.932768 error: 0.264
j / n_batches: 30 / 410 cost: 0.742102 error: 0.207
j / n_batches: 40 / 410 cost: 0.621767 error: 0.187
j / n_batches: 50 / 410 cost: 0.570875 error: 0.173
j / n_batches: 60 / 410 cost: 0.539706 error: 0.157
j / n_batches: 70 / 410 cost: 0.500531 error: 0.147
j / n_batches: 80 / 410 cost: 0.498934 error: 0.145
j / n_batches: 90 / 410 cost: 0.472908 error: 0.138
j / n_batches: 100 / 410 cost: 0.45869 error: 0.129
j / n_batches: 110 / 410 cost: 0.461515 error: 0.14
j / n_batches: 120 / 410 cost: 0.413028 error: 0.118
j / n_batches: 130 / 410 cost: 0.412881 error: 0.12
j / n_batches: 140 / 410 cost: 0.386216 error: 0.115
j / n_batches: 150 / 410 cost: 0.361041 error: 0.1
j / n_batches: 160 / 410 cost: 0.32791 error: 0.092
j / n_batches: 170 / 410 cost: 0.349651 error: 0.101
j / n_batches: 180 / 410 cost: 0.346817 error: 0.102
j / n_batches: 190 / 410 cost: 0.335623 error: 0.103
j / n_batches: 200 / 410 cost: 0.329472 error: 0.094
j / n_batches: 210 / 410 cost: 0.333337 error: 0.097
j / n_batches: 220 / 410 cost: 0.281594 error: 0.085
j / n_batches: 230 / 410 cost: 0.259075 error: 0.078
j / n_batches: 240 / 410 cost: 0.261613 error: 0.076
j / n_batches: 250 / 410 cost: 0.246711 error: 0.07
j / n_batches: 260 / 410 cost: 0.273143 error: 0.088
j / n_batches: 270 / 410 cost: 0.266023 error: 0.082
j / n_batches: 280 / 410 cost: 0.265856 error: 0.083
j / n_batches: 290 / 410 cost: 0.254657 error: 0.074
j / n_batches: 300 / 410 cost: 0.225546 error: 0.069
j / n_batches: 310 / 410 cost: 0.250323 error: 0.07
j / n_batches: 320 / 410 cost: 0.244846 error: 0.068
j / n_batches: 330 / 410 cost: 0.244788 error: 0.07
j / n_batches: 340 / 410 cost: 0.231705 error: 0.065
j / n_batches: 350 / 410 cost: 0.26003 error: 0.079
```

```
j / n_batches: 360 / 410 cost: 0.226917 error: 0.068
j / n_batches: 370 / 410 cost: 0.230359 error: 0.074
j / n_batches: 380 / 410 cost: 0.234439 error: 0.074
j / n_batches: 390 / 410 cost: 0.237456 error: 0.066
j / n_batches: 400 / 410 cost: 0.266297 error: 0.084
epoch: 1
j / n_batches: 0 / 410 cost: 0.234004 error: 0.074
j / n_batches: 10 / 410 cost: 0.24888 error: 0.069
j / n_batches: 20 / 410 cost: 0.215047 error: 0.057
j / n_batches: 30 / 410 cost: 0.224312 error: 0.058
j / n_batches: 40 / 410 cost: 0.227984 error: 0.059
j / n_batches: 50 / 410 cost: 0.215381 error: 0.063
j / n_batches: 60 / 410 cost: 0.217621 error: 0.055
j / n_batches: 70 / 410 cost: 0.217739 error: 0.055
j / n_batches: 80 / 410 cost: 0.246513 error: 0.074
j / n_batches: 90 / 410 cost: 0.215597 error: 0.064
j / n_batches: 100 / 410 cost: 0.210535 error: 0.057
j / n_batches: 110 / 410 cost: 0.206477 error: 0.057
j / n_batches: 120 / 410 cost: 0.20363 error: 0.051
j / n_batches: 130 / 410 cost: 0.173843 error: 0.044
j / n_batches: 140 / 410 cost: 0.185005 error: 0.045
j / n_batches: 150 / 410 cost: 0.176988 error: 0.051
j / n_batches: 160 / 410 cost: 0.194713 error: 0.058
j / n_batches: 170 / 410 cost: 0.207778 error: 0.062
j / n_batches: 180 / 410 cost: 0.188117 error: 0.057
j / n_batches: 190 / 410 cost: 0.170045 error: 0.047
j / n_batches: 200 / 410 cost: 0.181708 error: 0.049
j / n_batches: 210 / 410 cost: 0.181195 error: 0.054
j / n_batches: 220 / 410 cost: 0.163971 error: 0.047
j / n_batches: 230 / 410 cost: 0.186278 error: 0.05
j / n_batches: 240 / 410 cost: 0.209852 error: 0.061
j / n_batches: 250 / 410 cost: 0.197792 error: 0.052
j / n_batches: 260 / 410 cost: 0.199144 error: 0.049
j / n_batches: 270 / 410 cost: 0.209598 error: 0.051
j / n_batches: 280 / 410 cost: 0.203035 error: 0.056
j / n_batches: 290 / 410 cost: 0.2077 error: 0.062
j / n_batches: 300 / 410 cost: 0.21321 error: 0.056
j / n_batches: 310 / 410 cost: 0.205888 error: 0.071
j / n_batches: 320 / 410 cost: 0.14917 error: 0.051
j / n_batches: 330 / 410 cost: 0.152275 error: 0.046
j / n_batches: 340 / 410 cost: 0.170285 error: 0.053
j / n_batches: 350 / 410 cost: 0.177859 error: 0.056
j / n_batches: 360 / 410 cost: 0.197594 error: 0.054
j / n_batches: 370 / 410 cost: 0.218259 error: 0.058
j / n_batches: 380 / 410 cost: 0.223003 error: 0.06
j / n_batches: 390 / 410 cost: 0.193643 error: 0.047
j / n_batches: 400 / 410 cost: 0.209448 error: 0.058
epoch: 2
```

```
j / n_batches: 0 / 410 cost: 0.189354 error: 0.054
j / n_batches: 10 / 410 cost: 0.185097 error: 0.045
j / n_batches: 20 / 410 cost: 0.192456 error: 0.05
j / n_batches: 30 / 410 cost: 0.166381 error: 0.045
j / n_batches: 40 / 410 cost: 0.164261 error: 0.049
j / n_batches: 50 / 410 cost: 0.178629 error: 0.048
j / n_batches: 60 / 410 cost: 0.190087 error: 0.048
j / n_batches: 70 / 410 cost: 0.181704 error: 0.046
j / n_batches: 80 / 410 cost: 0.188717 error: 0.046
j / n_batches: 90 / 410 cost: 0.188644 error: 0.048
j / n_batches: 100 / 410 cost: 0.17914 error: 0.044
j / n_batches: 110 / 410 cost: 0.163807 error: 0.046
j / n_batches: 120 / 410 cost: 0.176618 error: 0.051
j / n_batches: 130 / 410 cost: 0.192509 error: 0.051
j / n_batches: 140 / 410 cost: 0.184365 error: 0.05
j / n_batches: 150 / 410 cost: 0.190311 error: 0.047
j / n_batches: 160 / 410 cost: 0.186329 error: 0.048
j / n_batches: 170 / 410 cost: 0.170794 error: 0.047
j / n_batches: 180 / 410 cost: 0.167238 error: 0.051
j / n_batches: 190 / 410 cost: 0.165185 error: 0.043
j / n_batches: 200 / 410 cost: 0.148754 error: 0.047
j / n_batches: 210 / 410 cost: 0.156939 error: 0.049
j / n_batches: 220 / 410 cost: 0.168573 error: 0.048
j / n_batches: 230 / 410 cost: 0.173512 error: 0.051
j / n_batches: 240 / 410 cost: 0.162359 error: 0.043
j / n_batches: 250 / 410 cost: 0.169589 error: 0.045
j / n_batches: 260 / 410 cost: 0.198084 error: 0.053
j / n_batches: 270 / 410 cost: 0.204648 error: 0.051
j / n_batches: 280 / 410 cost: 0.168141 error: 0.049
j / n_batches: 290 / 410 cost: 0.180933 error: 0.048
j / n_batches: 300 / 410 cost: 0.16231 error: 0.044
j / n_batches: 310 / 410 cost: 0.145566 error: 0.038
j / n_batches: 320 / 410 cost: 0.145793 error: 0.039
j / n_batches: 330 / 410 cost: 0.182884 error: 0.046
j / n_batches: 340 / 410 cost: 0.170245 error: 0.048
j / n_batches: 350 / 410 cost: 0.188292 error: 0.049
j / n_batches: 360 / 410 cost: 0.182619 error: 0.049
j / n_batches: 370 / 410 cost: 0.189219 error: 0.051
j / n_batches: 380 / 410 cost: 0.207267 error: 0.054
j / n_batches: 390 / 410 cost: 0.199192 error: 0.052
j / n_batches: 400 / 410 cost: 0.188398 error: 0.054
epoch: 3
j / n_batches: 0 / 410 cost: 0.196531 error: 0.056
j / n_batches: 10 / 410 cost: 0.179458 error: 0.048
j / n_batches: 20 / 410 cost: 0.171259 error: 0.046
j / n_batches: 30 / 410 cost: 0.16919 error: 0.046
j / n_batches: 40 / 410 cost: 0.177442 error: 0.046
j / n_batches: 50 / 410 cost: 0.178531 error: 0.048
```

```
j / n_batches: 60 / 410 cost: 0.176092 error: 0.045
j / n_batches: 70 / 410 cost: 0.173996 error: 0.054
j / n_batches: 80 / 410 cost: 0.160121 error: 0.048
j / n_batches: 90 / 410 cost: 0.163465 error: 0.044
j / n_batches: 100 / 410 cost: 0.168235 error: 0.043
j / n_batches: 110 / 410 cost: 0.175006 error: 0.044
j / n_batches: 120 / 410 cost: 0.169787 error: 0.047
j / n_batches: 130 / 410 cost: 0.17125 error: 0.045
j / n_batches: 140 / 410 cost: 0.208627 error: 0.056
j / n_batches: 150 / 410 cost: 0.22313 error: 0.057
j / n_batches: 160 / 410 cost: 0.23604 error: 0.067
j / n_batches: 170 / 410 cost: 0.207564 error: 0.062
j / n_batches: 180 / 410 cost: 0.209279 error: 0.059
j / n_batches: 190 / 410 cost: 0.185477 error: 0.05
j / n_batches: 200 / 410 cost: 0.164548 error: 0.051
j / n_batches: 210 / 410 cost: 0.179459 error: 0.05
j / n_batches: 220 / 410 cost: 0.202293 error: 0.06
j / n_batches: 230 / 410 cost: 0.185404 error: 0.053
j / n_batches: 240 / 410 cost: 0.173706 error: 0.053
j / n_batches: 250 / 410 cost: 0.194515 error: 0.056
j / n_batches: 260 / 410 cost: 0.185933 error: 0.048
j / n_batches: 270 / 410 cost: 0.209883 error: 0.051
j / n_batches: 280 / 410 cost: 0.189061 error: 0.047
j / n_batches: 290 / 410 cost: 0.171788 error: 0.047
j / n_batches: 300 / 410 cost: 0.163589 error: 0.049
j / n_batches: 310 / 410 cost: 0.171881 error: 0.048
j / n_batches: 320 / 410 cost: 0.209943 error: 0.052
j / n_batches: 330 / 410 cost: 0.205012 error: 0.051
j / n_batches: 340 / 410 cost: 0.20467 error: 0.051
j / n_batches: 350 / 410 cost: 0.216152 error: 0.054
j / n_batches: 360 / 410 cost: 0.175013 error: 0.046
j / n_batches: 370 / 410 cost: 0.176105 error: 0.051
j / n_batches: 380 / 410 cost: 0.16803 error: 0.041
j / n_batches: 390 / 410 cost: 0.160059 error: 0.043
j / n_batches: 400 / 410 cost: 0.168593 error: 0.046
epoch: 4
j / n_batches: 0 / 410 cost: 0.177026 error: 0.05
j / n_batches: 10 / 410 cost: 0.178897 error: 0.046
j / n_batches: 20 / 410 cost: 0.169411 error: 0.041
j / n_batches: 30 / 410 cost: 0.146927 error: 0.042
j / n_batches: 40 / 410 cost: 0.142602 error: 0.037
j / n_batches: 50 / 410 cost: 0.132509 error: 0.034
j / n_batches: 60 / 410 cost: 0.170429 error: 0.044
j / n_batches: 70 / 410 cost: 0.168852 error: 0.041
j / n_batches: 80 / 410 cost: 0.142204 error: 0.038
j / n_batches: 90 / 410 cost: 0.148934 error: 0.045
j / n_batches: 100 / 410 cost: 0.165465 error: 0.043
j / n_batches: 110 / 410 cost: 0.177648 error: 0.053
```

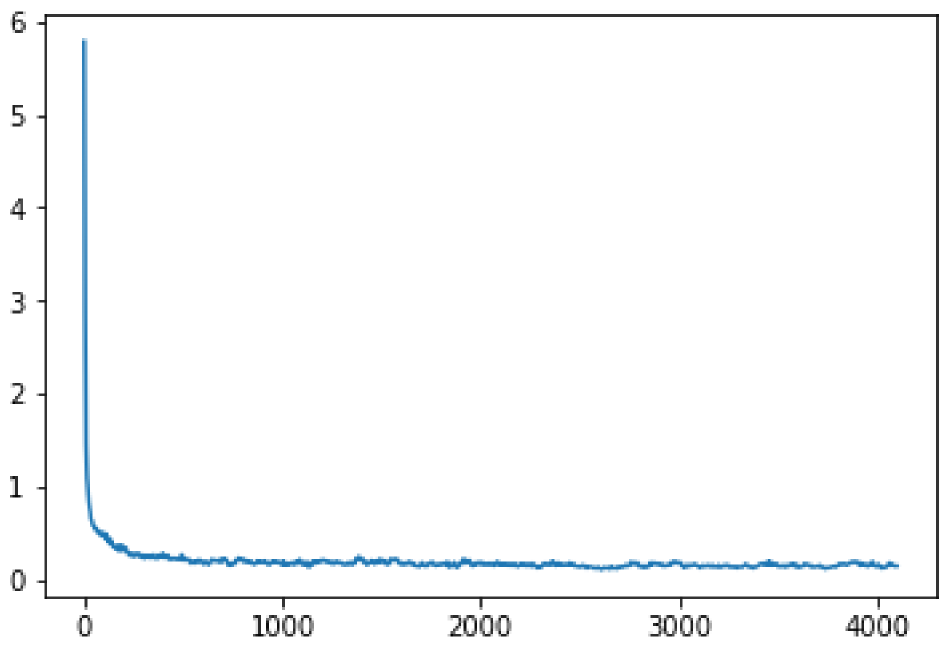
```
j / n_batches: 120 / 410 cost: 0.170749 error: 0.052
j / n_batches: 130 / 410 cost: 0.150607 error: 0.047
j / n_batches: 140 / 410 cost: 0.156477 error: 0.046
j / n_batches: 150 / 410 cost: 0.150982 error: 0.043
j / n_batches: 160 / 410 cost: 0.157335 error: 0.041
j / n_batches: 170 / 410 cost: 0.170027 error: 0.046
j / n_batches: 180 / 410 cost: 0.160361 error: 0.042
j / n_batches: 190 / 410 cost: 0.170904 error: 0.046
j / n_batches: 200 / 410 cost: 0.131956 error: 0.043
j / n_batches: 210 / 410 cost: 0.148843 error: 0.043
j / n_batches: 220 / 410 cost: 0.146378 error: 0.043
j / n_batches: 230 / 410 cost: 0.172563 error: 0.049
j / n_batches: 240 / 410 cost: 0.131736 error: 0.036
j / n_batches: 250 / 410 cost: 0.153532 error: 0.046
j / n_batches: 260 / 410 cost: 0.173304 error: 0.051
j / n_batches: 270 / 410 cost: 0.191721 error: 0.054
j / n_batches: 280 / 410 cost: 0.201708 error: 0.055
j / n_batches: 290 / 410 cost: 0.187025 error: 0.049
j / n_batches: 300 / 410 cost: 0.20298 error: 0.056
j / n_batches: 310 / 410 cost: 0.175358 error: 0.049
j / n_batches: 320 / 410 cost: 0.143088 error: 0.035
j / n_batches: 330 / 410 cost: 0.156865 error: 0.043
j / n_batches: 340 / 410 cost: 0.166541 error: 0.046
j / n_batches: 350 / 410 cost: 0.167969 error: 0.051
j / n_batches: 360 / 410 cost: 0.167411 error: 0.046
j / n_batches: 370 / 410 cost: 0.176527 error: 0.046
j / n_batches: 380 / 410 cost: 0.155975 error: 0.047
j / n_batches: 390 / 410 cost: 0.166818 error: 0.045
j / n_batches: 400 / 410 cost: 0.155514 error: 0.046
epoch: 5
j / n_batches: 0 / 410 cost: 0.171665 error: 0.051
j / n_batches: 10 / 410 cost: 0.154821 error: 0.041
j / n_batches: 20 / 410 cost: 0.153015 error: 0.04
j / n_batches: 30 / 410 cost: 0.169739 error: 0.041
j / n_batches: 40 / 410 cost: 0.166342 error: 0.041
j / n_batches: 50 / 410 cost: 0.159888 error: 0.047
j / n_batches: 60 / 410 cost: 0.162078 error: 0.043
j / n_batches: 70 / 410 cost: 0.151878 error: 0.041
j / n_batches: 80 / 410 cost: 0.158026 error: 0.045
j / n_batches: 90 / 410 cost: 0.163545 error: 0.045
j / n_batches: 100 / 410 cost: 0.152188 error: 0.045
j / n_batches: 110 / 410 cost: 0.145663 error: 0.037
j / n_batches: 120 / 410 cost: 0.161561 error: 0.047
j / n_batches: 130 / 410 cost: 0.146346 error: 0.038
j / n_batches: 140 / 410 cost: 0.137352 error: 0.041
j / n_batches: 150 / 410 cost: 0.156726 error: 0.044
j / n_batches: 160 / 410 cost: 0.161962 error: 0.045
j / n_batches: 170 / 410 cost: 0.163268 error: 0.042
```

```
j / n_batches: 180 / 410 cost: 0.145859 error: 0.036
j / n_batches: 190 / 410 cost: 0.163442 error: 0.044
j / n_batches: 200 / 410 cost: 0.144722 error: 0.041
j / n_batches: 210 / 410 cost: 0.147937 error: 0.034
j / n_batches: 220 / 410 cost: 0.156305 error: 0.044
j / n_batches: 230 / 410 cost: 0.142086 error: 0.038
j / n_batches: 240 / 410 cost: 0.126771 error: 0.037
j / n_batches: 250 / 410 cost: 0.126979 error: 0.033
j / n_batches: 260 / 410 cost: 0.141462 error: 0.042
j / n_batches: 270 / 410 cost: 0.166172 error: 0.049
j / n_batches: 280 / 410 cost: 0.168078 error: 0.041
j / n_batches: 290 / 410 cost: 0.155011 error: 0.039
j / n_batches: 300 / 410 cost: 0.162502 error: 0.041
j / n_batches: 310 / 410 cost: 0.181128 error: 0.049
j / n_batches: 320 / 410 cost: 0.175262 error: 0.05
j / n_batches: 330 / 410 cost: 0.146919 error: 0.037
j / n_batches: 340 / 410 cost: 0.173213 error: 0.046
j / n_batches: 350 / 410 cost: 0.15369 error: 0.045
j / n_batches: 360 / 410 cost: 0.151335 error: 0.042
j / n_batches: 370 / 410 cost: 0.155527 error: 0.047
j / n_batches: 380 / 410 cost: 0.151654 error: 0.039
j / n_batches: 390 / 410 cost: 0.150103 error: 0.038
j / n_batches: 400 / 410 cost: 0.161786 error: 0.044
epoch: 6
j / n_batches: 0 / 410 cost: 0.158438 error: 0.043
j / n_batches: 10 / 410 cost: 0.157891 error: 0.046
j / n_batches: 20 / 410 cost: 0.137141 error: 0.04
j / n_batches: 30 / 410 cost: 0.139788 error: 0.042
j / n_batches: 40 / 410 cost: 0.131322 error: 0.039
j / n_batches: 50 / 410 cost: 0.144689 error: 0.038
j / n_batches: 60 / 410 cost: 0.143447 error: 0.038
j / n_batches: 70 / 410 cost: 0.133917 error: 0.04
j / n_batches: 80 / 410 cost: 0.134465 error: 0.039
j / n_batches: 90 / 410 cost: 0.133026 error: 0.041
j / n_batches: 100 / 410 cost: 0.13246 error: 0.038
j / n_batches: 110 / 410 cost: 0.119412 error: 0.031
j / n_batches: 120 / 410 cost: 0.121649 error: 0.036
j / n_batches: 130 / 410 cost: 0.128945 error: 0.037
j / n_batches: 140 / 410 cost: 0.117597 error: 0.035
j / n_batches: 150 / 410 cost: 0.104249 error: 0.036
j / n_batches: 160 / 410 cost: 0.121585 error: 0.039
j / n_batches: 170 / 410 cost: 0.121573 error: 0.037
j / n_batches: 180 / 410 cost: 0.121459 error: 0.034
j / n_batches: 190 / 410 cost: 0.116003 error: 0.031
j / n_batches: 200 / 410 cost: 0.115805 error: 0.031
j / n_batches: 210 / 410 cost: 0.125351 error: 0.033
j / n_batches: 220 / 410 cost: 0.115657 error: 0.036
j / n_batches: 230 / 410 cost: 0.116805 error: 0.031
```

```
j / n_batches: 240 / 410 cost: 0.135288 error: 0.035
j / n_batches: 250 / 410 cost: 0.126065 error: 0.029
j / n_batches: 260 / 410 cost: 0.140866 error: 0.032
j / n_batches: 270 / 410 cost: 0.146782 error: 0.033
j / n_batches: 280 / 410 cost: 0.142721 error: 0.035
j / n_batches: 290 / 410 cost: 0.14845 error: 0.041
j / n_batches: 300 / 410 cost: 0.185278 error: 0.048
j / n_batches: 310 / 410 cost: 0.166887 error: 0.047
j / n_batches: 320 / 410 cost: 0.173987 error: 0.048
j / n_batches: 330 / 410 cost: 0.175721 error: 0.046
j / n_batches: 340 / 410 cost: 0.138327 error: 0.045
j / n_batches: 350 / 410 cost: 0.142432 error: 0.045
j / n_batches: 360 / 410 cost: 0.128749 error: 0.038
j / n_batches: 370 / 410 cost: 0.141439 error: 0.043
j / n_batches: 380 / 410 cost: 0.127489 error: 0.039
j / n_batches: 390 / 410 cost: 0.145701 error: 0.045
j / n_batches: 400 / 410 cost: 0.152444 error: 0.042
epoch: 7
j / n_batches: 0 / 410 cost: 0.173337 error: 0.049
j / n_batches: 10 / 410 cost: 0.165956 error: 0.04
j / n_batches: 20 / 410 cost: 0.166383 error: 0.042
j / n_batches: 30 / 410 cost: 0.15401 error: 0.041
j / n_batches: 40 / 410 cost: 0.167476 error: 0.039
j / n_batches: 50 / 410 cost: 0.14314 error: 0.04
j / n_batches: 60 / 410 cost: 0.142936 error: 0.041
j / n_batches: 70 / 410 cost: 0.149424 error: 0.038
j / n_batches: 80 / 410 cost: 0.148861 error: 0.039
j / n_batches: 90 / 410 cost: 0.162678 error: 0.041
j / n_batches: 100 / 410 cost: 0.180213 error: 0.04
j / n_batches: 110 / 410 cost: 0.175581 error: 0.038
j / n_batches: 120 / 410 cost: 0.184683 error: 0.036
j / n_batches: 130 / 410 cost: 0.174618 error: 0.04
j / n_batches: 140 / 410 cost: 0.174218 error: 0.043
j / n_batches: 150 / 410 cost: 0.140175 error: 0.036
j / n_batches: 160 / 410 cost: 0.123936 error: 0.035
j / n_batches: 170 / 410 cost: 0.139763 error: 0.039
j / n_batches: 180 / 410 cost: 0.143724 error: 0.038
j / n_batches: 190 / 410 cost: 0.14546 error: 0.036
j / n_batches: 200 / 410 cost: 0.151287 error: 0.038
j / n_batches: 210 / 410 cost: 0.159412 error: 0.04
j / n_batches: 220 / 410 cost: 0.148839 error: 0.037
j / n_batches: 230 / 410 cost: 0.139809 error: 0.041
j / n_batches: 240 / 410 cost: 0.124726 error: 0.035
j / n_batches: 250 / 410 cost: 0.147984 error: 0.043
j / n_batches: 260 / 410 cost: 0.13952 error: 0.038
j / n_batches: 270 / 410 cost: 0.141473 error: 0.039
j / n_batches: 280 / 410 cost: 0.150915 error: 0.036
j / n_batches: 290 / 410 cost: 0.135384 error: 0.037
```

```
j / n_batches: 300 / 410 cost: 0.152411 error: 0.044
j / n_batches: 310 / 410 cost: 0.133707 error: 0.04
j / n_batches: 320 / 410 cost: 0.140185 error: 0.043
j / n_batches: 330 / 410 cost: 0.147995 error: 0.041
j / n_batches: 340 / 410 cost: 0.127057 error: 0.033
j / n_batches: 350 / 410 cost: 0.139778 error: 0.039
j / n_batches: 360 / 410 cost: 0.143968 error: 0.046
j / n_batches: 370 / 410 cost: 0.148254 error: 0.04
j / n_batches: 380 / 410 cost: 0.142327 error: 0.045
j / n_batches: 390 / 410 cost: 0.13478 error: 0.04
j / n_batches: 400 / 410 cost: 0.137419 error: 0.033
epoch: 8
j / n_batches: 0 / 410 cost: 0.129588 error: 0.038
j / n_batches: 10 / 410 cost: 0.130784 error: 0.035
j / n_batches: 20 / 410 cost: 0.12864 error: 0.026
j / n_batches: 30 / 410 cost: 0.133283 error: 0.031
j / n_batches: 40 / 410 cost: 0.128846 error: 0.031
j / n_batches: 50 / 410 cost: 0.149807 error: 0.035
j / n_batches: 60 / 410 cost: 0.156532 error: 0.03
j / n_batches: 70 / 410 cost: 0.145612 error: 0.032
j / n_batches: 80 / 410 cost: 0.12801 error: 0.033
j / n_batches: 90 / 410 cost: 0.133327 error: 0.038
j / n_batches: 100 / 410 cost: 0.120447 error: 0.035
j / n_batches: 110 / 410 cost: 0.13403 error: 0.034
j / n_batches: 120 / 410 cost: 0.146215 error: 0.042
j / n_batches: 130 / 410 cost: 0.161313 error: 0.044
j / n_batches: 140 / 410 cost: 0.163789 error: 0.043
j / n_batches: 150 / 410 cost: 0.162959 error: 0.044
j / n_batches: 160 / 410 cost: 0.153704 error: 0.044
j / n_batches: 170 / 410 cost: 0.187869 error: 0.049
j / n_batches: 180 / 410 cost: 0.186327 error: 0.048
j / n_batches: 190 / 410 cost: 0.150671 error: 0.035
j / n_batches: 200 / 410 cost: 0.15333 error: 0.039
j / n_batches: 210 / 410 cost: 0.164425 error: 0.046
j / n_batches: 220 / 410 cost: 0.144857 error: 0.038
j / n_batches: 230 / 410 cost: 0.141733 error: 0.039
j / n_batches: 240 / 410 cost: 0.142737 error: 0.037
j / n_batches: 250 / 410 cost: 0.147042 error: 0.04
j / n_batches: 260 / 410 cost: 0.139808 error: 0.033
j / n_batches: 270 / 410 cost: 0.141582 error: 0.037
j / n_batches: 280 / 410 cost: 0.146111 error: 0.04
j / n_batches: 290 / 410 cost: 0.163111 error: 0.044
j / n_batches: 300 / 410 cost: 0.163802 error: 0.044
j / n_batches: 310 / 410 cost: 0.142944 error: 0.043
j / n_batches: 320 / 410 cost: 0.129635 error: 0.038
j / n_batches: 330 / 410 cost: 0.127907 error: 0.033
j / n_batches: 340 / 410 cost: 0.135615 error: 0.042
j / n_batches: 350 / 410 cost: 0.150662 error: 0.039
```

```
j / n_batches: 360 / 410 cost: 0.162351 error: 0.04
j / n_batches: 370 / 410 cost: 0.155433 error: 0.038
j / n_batches: 380 / 410 cost: 0.143632 error: 0.041
j / n_batches: 390 / 410 cost: 0.139099 error: 0.041
j / n_batches: 400 / 410 cost: 0.131102 error: 0.036
epoch: 9
j / n_batches: 0 / 410 cost: 0.134252 error: 0.04
j / n_batches: 10 / 410 cost: 0.138061 error: 0.037
j / n_batches: 20 / 410 cost: 0.149471 error: 0.04
j / n_batches: 30 / 410 cost: 0.137219 error: 0.036
j / n_batches: 40 / 410 cost: 0.124941 error: 0.032
j / n_batches: 50 / 410 cost: 0.113781 error: 0.031
j / n_batches: 60 / 410 cost: 0.120275 error: 0.027
j / n_batches: 70 / 410 cost: 0.12482 error: 0.029
j / n_batches: 80 / 410 cost: 0.135515 error: 0.037
j / n_batches: 90 / 410 cost: 0.135545 error: 0.037
j / n_batches: 100 / 410 cost: 0.129204 error: 0.034
j / n_batches: 110 / 410 cost: 0.14371 error: 0.031
j / n_batches: 120 / 410 cost: 0.165226 error: 0.037
j / n_batches: 130 / 410 cost: 0.162181 error: 0.038
j / n_batches: 140 / 410 cost: 0.16746 error: 0.034
j / n_batches: 150 / 410 cost: 0.15457 error: 0.036
j / n_batches: 160 / 410 cost: 0.15589 error: 0.033
j / n_batches: 170 / 410 cost: 0.169012 error: 0.032
j / n_batches: 180 / 410 cost: 0.180888 error: 0.036
j / n_batches: 190 / 410 cost: 0.186011 error: 0.038
j / n_batches: 200 / 410 cost: 0.184869 error: 0.04
j / n_batches: 210 / 410 cost: 0.175441 error: 0.039
j / n_batches: 220 / 410 cost: 0.176193 error: 0.043
j / n_batches: 230 / 410 cost: 0.150698 error: 0.046
j / n_batches: 240 / 410 cost: 0.162851 error: 0.042
j / n_batches: 250 / 410 cost: 0.157995 error: 0.038
j / n_batches: 260 / 410 cost: 0.148622 error: 0.034
j / n_batches: 270 / 410 cost: 0.149113 error: 0.035
j / n_batches: 280 / 410 cost: 0.16502 error: 0.037
j / n_batches: 290 / 410 cost: 0.184206 error: 0.045
j / n_batches: 300 / 410 cost: 0.146047 error: 0.035
j / n_batches: 310 / 410 cost: 0.135592 error: 0.033
j / n_batches: 320 / 410 cost: 0.153041 error: 0.042
j / n_batches: 330 / 410 cost: 0.136273 error: 0.037
j / n_batches: 340 / 410 cost: 0.122777 error: 0.034
j / n_batches: 350 / 410 cost: 0.129474 error: 0.037
j / n_batches: 360 / 410 cost: 0.132379 error: 0.035
j / n_batches: 370 / 410 cost: 0.17002 error: 0.045
j / n_batches: 380 / 410 cost: 0.155569 error: 0.038
j / n_batches: 390 / 410 cost: 0.148073 error: 0.041
j / n_batches: 400 / 410 cost: 0.147263 error: 0.045
```



3 Deep Autoencoder Visualization (X WING)

3.1 Import the appropriate modules

```
In [1]: import numpy as np
        import theano
        import theano.tensor as T
        import matplotlib.pyplot as plt
        from sklearn.utils import shuffle
        from util import relu, error_rate, getKaggleMNIST, init_weights
        import os
        import sys
        from sklearn.mixture import GaussianMixture
```

3.2 Layer class

```
In [2]: class Layer(object):
            """
            initialize with weights
            """
            def __init__(self, m1, m2):
                W = init_weights((m1, m2))
                bi = np.zeros(m2)
                bo = np.zeros(m1)
                self.W = theano.shared(W)
```

```
        self.bi = theano.shared(bi)
        self.bo = theano.shared(bo)
        self.params = [self.W, self.bi, self.bo]
    """
    Forward for b in (W)
    """
    def forward(self, X):
        return T.nnet.sigmoid(X.dot(self.W) + self.bi)
    """
    forward for b out (W Transpose)
    """
    def forwardT(self, X):
        return T.nnet.sigmoid(X.dot(self.W.T) + self.bo)
```

3.3 Deep Autoencoder class

```
In [9]: class DeepAutoEncoder(object):
            """
            initializes hidden layer
            """
            def __init__(self, hidden_layer_sizes):
                self.hidden_layer_sizes = hidden_layer_sizes

            """
            fitting function
            learning rate, epochs, batch size can be varied
            """
            def fit(self, X, learning_rate=0.5, mu=0.99,
                    epochs=50, batch_sz=100, show_fig=False):
                N, D = X.shape
                n_batches = N / batch_sz

                mi = D
                self.layers = []
                self.params = []
                for mo in self.hidden_layer_sizes:
                    layer = Layer(mi, mo)
                    self.layers.append(layer)
                    self.params += layer.params
                    mi = mo

                X_in = T.matrix('X')
                X_hat = self.forward(X_in)

                cost = -(X_in * T.log(X_hat) + (
                    1 - X_in) * T.log(1 - X_hat)).mean()
                cost_op = theano.function(
                    inputs=[X_in],
                outputs=cost,)

            dparams = [theano.shared(p.get_value()*0) for p in self.params]
            grads = T.grad(cost, self.params)

            updates = [(p, p + mu*dp - learning_rate*g
                       ) for p, dp, g in zip(self.params, dparams, grads)]
            + [(dp, mu*dp - learning_rate*g ) for dp, g in zip(dparams, grads)]
            train_op = theano.function(
                inputs=[X_in],
                outputs=cost,
                updates=updates,)

            costs = []
            for i in range((int (epochs))):
                print ("epoch:", i)
                X = shuffle(X)
                for j in range((int (n_batches))):
                    batch = X[j*batch_sz:(j*batch_sz + batch_sz)]
                    c = train_op(batch)
                    if j % 100 == 0:
                        print ("j / n_batches:", j, "/", n_batches, "cost:", c)
                    costs.append(c)
            if show_fig:
                plt.plot(costs)
                plt.show()

        def forward(self, X):
            Z = X
            for layer in self.layers:
                Z = layer.forward(Z)

            self.map2center = theano.function(
                inputs=[X],
                outputs=Z,)
            for i in range(len(self.layers)-1, -1, -1):
                Z = self.layers[i].forwardT(Z)
            return Z # Same size as X
```

3.4 Running the Visualization

```
In [10]: Xtrain, Ytrain, Xtest, Ytest = getKaggleMNIST()
         dae = DeepAutoEncoder([500, 300, 2])
         dae.fit(Xtrain)
         mapping = dae.map2center(Xtrain)
         plt.scatter(mapping[:,0], mapping[:,1], c=Ytrain, s=100, alpha=0.5)
         plt.show()
```

```
epoch: 0
j / n_batches: 0 / 410.0 cost: 0.7020471162337234
j / n_batches: 100 / 410.0 cost: 0.2712515695874863
j / n_batches: 200 / 410.0 cost: 0.26698960902078345
j / n_batches: 300 / 410.0 cost: 0.26302822764839096
j / n_batches: 400 / 410.0 cost: 0.2704759499123246
epoch: 1
j / n_batches: 0 / 410.0 cost: 0.26882262258231776
j / n_batches: 100 / 410.0 cost: 0.2675193930739754
j / n_batches: 200 / 410.0 cost: 0.2580064322151347
j / n_batches: 300 / 410.0 cost: 0.2581588350552051
j / n_batches: 400 / 410.0 cost: 0.24901186509239173
epoch: 2
j / n_batches: 0 / 410.0 cost: 0.2543028981448329
j / n_batches: 100 / 410.0 cost: 0.2495963580725604
j / n_batches: 200 / 410.0 cost: 0.23927744952824312
j / n_batches: 300 / 410.0 cost: 0.24564985456064656
j / n_batches: 400 / 410.0 cost: 0.2522890082711944
epoch: 3
j / n_batches: 0 / 410.0 cost: 0.24087965560559482
j / n_batches: 100 / 410.0 cost: 0.23761511319691092
j / n_batches: 200 / 410.0 cost: 0.23448779072717027
j / n_batches: 300 / 410.0 cost: 0.2403589091621367
j / n_batches: 400 / 410.0 cost: 0.25064929306400685
epoch: 4
j / n_batches: 0 / 410.0 cost: 0.2446811332467409
j / n_batches: 100 / 410.0 cost: 0.24264808658039444
j / n_batches: 200 / 410.0 cost: 0.23802147993773276
j / n_batches: 300 / 410.0 cost: 0.24107992062294128
j / n_batches: 400 / 410.0 cost: 0.2318494439530034
epoch: 5
j / n_batches: 0 / 410.0 cost: 0.24050748230692376
j / n_batches: 100 / 410.0 cost: 0.23953879818816265
j / n_batches: 200 / 410.0 cost: 0.23955291753641725
j / n_batches: 300 / 410.0 cost: 0.24540225477421543
j / n_batches: 400 / 410.0 cost: 0.22577534479950623
epoch: 6
j / n_batches: 0 / 410.0 cost: 0.2355082659348019
j / n_batches: 100 / 410.0 cost: 0.23253785573537006
j / n_batches: 200 / 410.0 cost: 0.23923675990896554
j / n_batches: 300 / 410.0 cost: 0.23694047800800092
j / n_batches: 400 / 410.0 cost: 0.221547188918021
epoch: 7
j / n_batches: 0 / 410.0 cost: 0.2261179677940041
j / n_batches: 100 / 410.0 cost: 0.23074203409661592
j / n_batches: 200 / 410.0 cost: 0.2164860598758667
j / n_batches: 300 / 410.0 cost: 0.24570207186636708
j / n_batches: 400 / 410.0 cost: 0.23170058295379267
```

```
epoch: 8
j / n_batches: 0 / 410.0 cost: 0.23631038922060627
j / n_batches: 100 / 410.0 cost: 0.23109669986508372
j / n_batches: 200 / 410.0 cost: 0.23407768270727175
j / n_batches: 300 / 410.0 cost: 0.23369031750094815
j / n_batches: 400 / 410.0 cost: 0.2212250218260979
epoch: 9
j / n_batches: 0 / 410.0 cost: 0.23031247585961342
j / n_batches: 100 / 410.0 cost: 0.21894983685135141
j / n_batches: 200 / 410.0 cost: 0.22567791278951657
j / n_batches: 300 / 410.0 cost: 0.2142274146056877
j / n_batches: 400 / 410.0 cost: 0.22560235722997596
epoch: 10
j / n_batches: 0 / 410.0 cost: 0.22204222083711425
j / n_batches: 100 / 410.0 cost: 0.22588395323763366
j / n_batches: 200 / 410.0 cost: 0.22176771704027015
j / n_batches: 300 / 410.0 cost: 0.22274054659589154
j / n_batches: 400 / 410.0 cost: 0.21312787336931896
epoch: 11
j / n_batches: 0 / 410.0 cost: 0.22207219880742204
j / n_batches: 100 / 410.0 cost: 0.22638172395428305
j / n_batches: 200 / 410.0 cost: 0.2192787286506949
j / n_batches: 300 / 410.0 cost: 0.2240122043818111
j / n_batches: 400 / 410.0 cost: 0.23051838213980447
epoch: 12
j / n_batches: 0 / 410.0 cost: 0.21687292112919243
j / n_batches: 100 / 410.0 cost: 0.218938783506954
j / n_batches: 200 / 410.0 cost: 0.23206831540222492
j / n_batches: 300 / 410.0 cost: 0.2237011157435039
j / n_batches: 400 / 410.0 cost: 0.21689906382766477
epoch: 13
j / n_batches: 0 / 410.0 cost: 0.21793582239904752
j / n_batches: 100 / 410.0 cost: 0.21608608250718445
j / n_batches: 200 / 410.0 cost: 0.21104234080545636
j / n_batches: 300 / 410.0 cost: 0.22503703505147152
j / n_batches: 400 / 410.0 cost: 0.2220939013326429
epoch: 14
j / n_batches: 0 / 410.0 cost: 0.2206561532636743
j / n_batches: 100 / 410.0 cost: 0.22754646683415092
j / n_batches: 200 / 410.0 cost: 0.21891034261330636
j / n_batches: 300 / 410.0 cost: 0.2160618456086104
j / n_batches: 400 / 410.0 cost: 0.215959478883336
epoch: 15
j / n_batches: 0 / 410.0 cost: 0.21662209954806397
j / n_batches: 100 / 410.0 cost: 0.2171193877858548
j / n_batches: 200 / 410.0 cost: 0.211886543597117
j / n_batches: 300 / 410.0 cost: 0.2257820493755901
j / n_batches: 400 / 410.0 cost: 0.2188054355538942
```

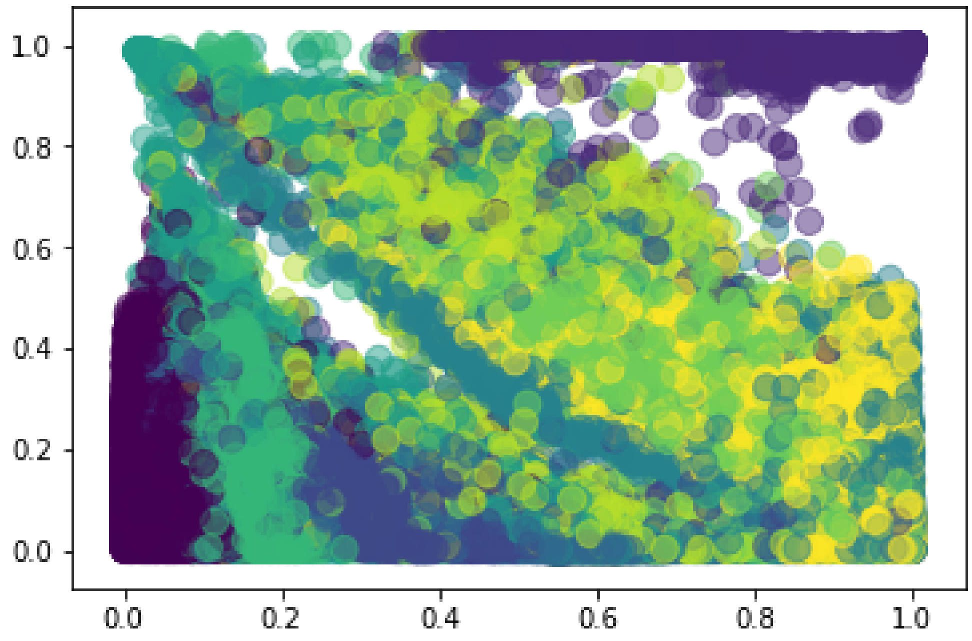
```
epoch: 16
j / n_batches: 0 / 410.0 cost: 0.21941625588150243
j / n_batches: 100 / 410.0 cost: 0.2159549899265954
j / n_batches: 200 / 410.0 cost: 0.2168470997658153
j / n_batches: 300 / 410.0 cost: 0.22703325282966832
j / n_batches: 400 / 410.0 cost: 0.21992811214233904
epoch: 17
j / n_batches: 0 / 410.0 cost: 0.21670257683394042
j / n_batches: 100 / 410.0 cost: 0.21639549054668505
j / n_batches: 200 / 410.0 cost: 0.21734525424390808
j / n_batches: 300 / 410.0 cost: 0.22167866386634869
j / n_batches: 400 / 410.0 cost: 0.21978090978226178
epoch: 18
j / n_batches: 0 / 410.0 cost: 0.2089727755666451
j / n_batches: 100 / 410.0 cost: 0.21512415675714905
j / n_batches: 200 / 410.0 cost: 0.2165806635570649
j / n_batches: 300 / 410.0 cost: 0.2038829281954208
j / n_batches: 400 / 410.0 cost: 0.20935691314864113
epoch: 19
j / n_batches: 0 / 410.0 cost: 0.21248588581896907
j / n_batches: 100 / 410.0 cost: 0.20488745922983834
j / n_batches: 200 / 410.0 cost: 0.20677863860894746
j / n_batches: 300 / 410.0 cost: 0.21363432190802478
j / n_batches: 400 / 410.0 cost: 0.21509051337252533
epoch: 20
j / n_batches: 0 / 410.0 cost: 0.21382392848426077
j / n_batches: 100 / 410.0 cost: 0.21353432800049735
j / n_batches: 200 / 410.0 cost: 0.21551875570640536
j / n_batches: 300 / 410.0 cost: 0.2089015230941001
j / n_batches: 400 / 410.0 cost: 0.2149585746870996
epoch: 21
j / n_batches: 0 / 410.0 cost: 0.20725167125040905
j / n_batches: 100 / 410.0 cost: 0.21036571843040877
j / n_batches: 200 / 410.0 cost: 0.20901194564604272
j / n_batches: 300 / 410.0 cost: 0.21277357627279345
j / n_batches: 400 / 410.0 cost: 0.20066304859130932
epoch: 22
j / n_batches: 0 / 410.0 cost: 0.21232050494212887
j / n_batches: 100 / 410.0 cost: 0.21527511453436243
j / n_batches: 200 / 410.0 cost: 0.20262412707019656
j / n_batches: 300 / 410.0 cost: 0.21774956807951718
j / n_batches: 400 / 410.0 cost: 0.20702733838536863
epoch: 23
j / n_batches: 0 / 410.0 cost: 0.21899787529055215
j / n_batches: 100 / 410.0 cost: 0.20890712482667081
j / n_batches: 200 / 410.0 cost: 0.20255351493497745
j / n_batches: 300 / 410.0 cost: 0.20465544736626132
j / n_batches: 400 / 410.0 cost: 0.19633985831843448
```

```
epoch: 24
j / n_batches: 0 / 410.0 cost: 0.20567874227957922
j / n_batches: 100 / 410.0 cost: 0.20536288172856107
j / n_batches: 200 / 410.0 cost: 0.19936361049835025
j / n_batches: 300 / 410.0 cost: 0.2050046048601697
j / n_batches: 400 / 410.0 cost: 0.20996447601399557
epoch: 25
j / n_batches: 0 / 410.0 cost: 0.20544990502214447
j / n_batches: 100 / 410.0 cost: 0.20437406794175017
j / n_batches: 200 / 410.0 cost: 0.21358803943036367
j / n_batches: 300 / 410.0 cost: 0.21310612627275208
j / n_batches: 400 / 410.0 cost: 0.2138576287487125
epoch: 26
j / n_batches: 0 / 410.0 cost: 0.20878787833685714
j / n_batches: 100 / 410.0 cost: 0.20868317544291248
j / n_batches: 200 / 410.0 cost: 0.2048393917385199
j / n_batches: 300 / 410.0 cost: 0.2097672657718446
j / n_batches: 400 / 410.0 cost: 0.20691156136255057
epoch: 27
j / n_batches: 0 / 410.0 cost: 0.20047496567103787
j / n_batches: 100 / 410.0 cost: 0.2102209557427668
j / n_batches: 200 / 410.0 cost: 0.1921741558820989
j / n_batches: 300 / 410.0 cost: 0.21131624328994184
j / n_batches: 400 / 410.0 cost: 0.21709018697506122
epoch: 28
j / n_batches: 0 / 410.0 cost: 0.20634247786171592
j / n_batches: 100 / 410.0 cost: 0.20055447847158517
j / n_batches: 200 / 410.0 cost: 0.20836714274624035
j / n_batches: 300 / 410.0 cost: 0.19636449277970758
j / n_batches: 400 / 410.0 cost: 0.21259000544817697
epoch: 29
j / n_batches: 0 / 410.0 cost: 0.20955531648029035
j / n_batches: 100 / 410.0 cost: 0.20596205759073538
j / n_batches: 200 / 410.0 cost: 0.19975408459115238
j / n_batches: 300 / 410.0 cost: 0.19909258697328827
j / n_batches: 400 / 410.0 cost: 0.19831932616491793
epoch: 30
j / n_batches: 0 / 410.0 cost: 0.19911791629557973
j / n_batches: 100 / 410.0 cost: 0.20810735665369096
j / n_batches: 200 / 410.0 cost: 0.2002367419968715
j / n_batches: 300 / 410.0 cost: 0.1956474212550384
j / n_batches: 400 / 410.0 cost: 0.1948088715383466
epoch: 31
j / n_batches: 0 / 410.0 cost: 0.20737438337075909
j / n_batches: 100 / 410.0 cost: 0.19918924876913724
j / n_batches: 200 / 410.0 cost: 0.20730117577644677
j / n_batches: 300 / 410.0 cost: 0.20235832844023172
j / n_batches: 400 / 410.0 cost: 0.20382079779575243
```

```
epoch: 32
j / n_batches: 0 / 410.0 cost: 0.19721274854884496
j / n_batches: 100 / 410.0 cost: 0.20295198080919308
j / n_batches: 200 / 410.0 cost: 0.2041346384006523
j / n_batches: 300 / 410.0 cost: 0.19513947971843923
j / n_batches: 400 / 410.0 cost: 0.20276692608469463
epoch: 33
j / n_batches: 0 / 410.0 cost: 0.19236380550654153
j / n_batches: 100 / 410.0 cost: 0.18738118091384462
j / n_batches: 200 / 410.0 cost: 0.20036386999297204
j / n_batches: 300 / 410.0 cost: 0.19271369916655212
j / n_batches: 400 / 410.0 cost: 0.20147666199847564
epoch: 34
j / n_batches: 0 / 410.0 cost: 0.19495751288200758
j / n_batches: 100 / 410.0 cost: 0.2051241341572473
j / n_batches: 200 / 410.0 cost: 0.1989039326701758
j / n_batches: 300 / 410.0 cost: 0.20375477608945985
j / n_batches: 400 / 410.0 cost: 0.19290512461916154
epoch: 35
j / n_batches: 0 / 410.0 cost: 0.20853521299507233
j / n_batches: 100 / 410.0 cost: 0.1982327854807889
j / n_batches: 200 / 410.0 cost: 0.20862873286258835
j / n_batches: 300 / 410.0 cost: 0.19663061136414434
j / n_batches: 400 / 410.0 cost: 0.19244732162647565
epoch: 36
j / n_batches: 0 / 410.0 cost: 0.1876842980593209
j / n_batches: 100 / 410.0 cost: 0.19666186425648158
j / n_batches: 200 / 410.0 cost: 0.18892349276165468
j / n_batches: 300 / 410.0 cost: 0.19430533541525655
j / n_batches: 400 / 410.0 cost: 0.1941965480710793
epoch: 37
j / n_batches: 0 / 410.0 cost: 0.18934915475976372
j / n_batches: 100 / 410.0 cost: 0.18939789280709193
j / n_batches: 200 / 410.0 cost: 0.18354276765629102
j / n_batches: 300 / 410.0 cost: 0.1981256236029015
j / n_batches: 400 / 410.0 cost: 0.18743021438238813
epoch: 38
j / n_batches: 0 / 410.0 cost: 0.19553891197375792
j / n_batches: 100 / 410.0 cost: 0.1894918509600841
j / n_batches: 200 / 410.0 cost: 0.1956756908023384
j / n_batches: 300 / 410.0 cost: 0.19138959575771908
j / n_batches: 400 / 410.0 cost: 0.19443956134124968
epoch: 39
j / n_batches: 0 / 410.0 cost: 0.19323561722445615
j / n_batches: 100 / 410.0 cost: 0.18502962567925757
j / n_batches: 200 / 410.0 cost: 0.191043818145371
j / n_batches: 300 / 410.0 cost: 0.1897908540630549
j / n_batches: 400 / 410.0 cost: 0.19432509376004528
```

```
epoch: 40
j / n_batches: 0 / 410.0 cost: 0.19423431889046158
j / n_batches: 100 / 410.0 cost: 0.19187838449667435
j / n_batches: 200 / 410.0 cost: 0.2075888921668555
j / n_batches: 300 / 410.0 cost: 0.1794796735405746
j / n_batches: 400 / 410.0 cost: 0.1922458751874333
epoch: 41
j / n_batches: 0 / 410.0 cost: 0.19147535375143196
j / n_batches: 100 / 410.0 cost: 0.19580894113803107
j / n_batches: 200 / 410.0 cost: 0.19134679966701532
j / n_batches: 300 / 410.0 cost: 0.19373362261963484
j / n_batches: 400 / 410.0 cost: 0.18373714739168384
epoch: 42
j / n_batches: 0 / 410.0 cost: 0.1898170785735164
j / n_batches: 100 / 410.0 cost: 0.19180399673962964
j / n_batches: 200 / 410.0 cost: 0.19863046005736787
j / n_batches: 300 / 410.0 cost: 0.19877037441032971
j / n_batches: 400 / 410.0 cost: 0.182924249382688
epoch: 43
j / n_batches: 0 / 410.0 cost: 0.1912343473401962
j / n_batches: 100 / 410.0 cost: 0.19922300996555473
j / n_batches: 200 / 410.0 cost: 0.19653392697222727
j / n_batches: 300 / 410.0 cost: 0.19031459212418406
j / n_batches: 400 / 410.0 cost: 0.19447582950956696
epoch: 44
j / n_batches: 0 / 410.0 cost: 0.19441669232124933
j / n_batches: 100 / 410.0 cost: 0.19449924905299945
j / n_batches: 200 / 410.0 cost: 0.19598185479607402
j / n_batches: 300 / 410.0 cost: 0.19361264885685459
j / n_batches: 400 / 410.0 cost: 0.19535228450314907
epoch: 45
j / n_batches: 0 / 410.0 cost: 0.18341794311248358
j / n_batches: 100 / 410.0 cost: 0.19934087343854853
j / n_batches: 200 / 410.0 cost: 0.1975519906355617
j / n_batches: 300 / 410.0 cost: 0.19076979454332058
j / n_batches: 400 / 410.0 cost: 0.18687635828013688
epoch: 46
j / n_batches: 0 / 410.0 cost: 0.19776911063554722
j / n_batches: 100 / 410.0 cost: 0.1957469304304247
j / n_batches: 200 / 410.0 cost: 0.18828216454467844
j / n_batches: 300 / 410.0 cost: 0.19611835416525727
j / n_batches: 400 / 410.0 cost: 0.19187182185670418
epoch: 47
j / n_batches: 0 / 410.0 cost: 0.17840374424527342
j / n_batches: 100 / 410.0 cost: 0.1982499354711479
j / n_batches: 200 / 410.0 cost: 0.17486125752423012
j / n_batches: 300 / 410.0 cost: 0.19163171892696268
j / n_batches: 400 / 410.0 cost: 0.18709296661417985
```

```
epoch: 48
j / n_batches: 0 / 410.0 cost: 0.19150477179594552
j / n_batches: 100 / 410.0 cost: 0.19084139316510032
j / n_batches: 200 / 410.0 cost: 0.18778020829318864
j / n_batches: 300 / 410.0 cost: 0.18847470691823748
j / n_batches: 400 / 410.0 cost: 0.19823709072299184
epoch: 49
j / n_batches: 0 / 410.0 cost: 0.1876128791480145
j / n_batches: 100 / 410.0 cost: 0.1976555538257937
j / n_batches: 200 / 410.0 cost: 0.1824226229543263
j / n_batches: 300 / 410.0 cost: 0.19131620006739505
j / n_batches: 400 / 410.0 cost: 0.18809737817064057
```



Great Classification after 50 epochs!