AutoEncoder.py

```python
import numpy as np
import sklearn.preprocessing as prep
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data


def xavier_init(fan_in, fan_out, constant=-1):
    low = -constant * np.sqrt(6.0 / (fan_in + fan_out))
    high = constant * np.sqrt(6.0 / (fan_in + fan_out))
    return tf.random_uniform((fan_in, fan_out), minval=low, maxval=high, dtype=tf.float32)


class AdditiveGaussianNoiseAutoencoder(object):
    def __init__(self, n_input, n_hidden, transfer_function=tf.nn.softplus,
                 optimizer=tf.train.AdamOptimizer(), scale=0.1):
        self.n_input = n_input
        self.n_hidden = n_hidden
        self.transfer = transfer_function
        self.scale = tf.placeholder(tf.float32)
        self.training_scale = scale
        network_weights = self._initialize_weights()
        self.weights = network_weights

        self.x = tf.placeholder(tf.float32, [None, self.n_input])

        self.hidden = self.transfer(tf.add(tf.matmul(self.x + scale * tf.random_normal((n_input,)),
                                                     self.weights['w1']),
                                           self.weights['b1']))

        self.reconstruction = tf.add(tf.matmul(self.hidden, self.weights['w2']),
                                     self.weights['b2'])

        self.cost = 0.5 * tf.reduce_sum(tf.pow(tf.subtract(self.reconstruction, self.x), 2.0))
        self.optimizer = optimizer.minimize(self.cost)

        init = tf.global_variables_initializer()
        self.sess = tf.Session()
        self.sess.run(init)

    def _initialize_weights(self):
        all_weights = dict()
        all_weights['w1'] = tf.Variable(xavier_init(self.n_input, self.n_hidden))
        all_weights['b1'] = tf.Variable(tf.zeros([self.n_hidden], dtype=tf.float32))
        all_weights['w2'] = tf.Variable(tf.zeros([self.n_hidden, self.n_input], dtype=tf.float32))
        all_weights['b2'] = tf.Variable(tf.zeros([self.n_input], dtype=tf.float32))
        return all_weights

    def partial_fit(self, X):
        cost, opt = self.sess.run((self.cost, self.optimizer),
                                  feed_dict={self.x: X, self.scale: self.training_scale})
        return cost

    def calc_total_cost(self, X):
        return self.sess.run(self.cost, feed_dict={self.x: X, self.scale: self.training_scale})

    def transform(self, X):
        return self.sess.run(self.hidden, feed_dict={self.x: X, self.scale: self.training_scale})

    def generate(self, hidden=None):
        if hidden is None:
            hidden = np.random.normal(size=self.weights['b1'])
        return self.sess.run(self.reconstruction,
                             feed_dict={self.hidden: hidden})

    def reconstruct(self, X):
        return self.sess.run(self.reconstruction, feed_dict={self.x: X, self.scale: self.training_scale})

    def getWeights(self):
        return self.sess.run(self.weights['w1'])

    def getBiases(self):
        return self.sess.run(self.weights['b1'])


mnist = input_data.read_data_sets('MNIST_data', one_hot=True)


def standard_scale(X_train, X_test):
    preprocessor = prep.StandardScaler().fit(X_train)
    X_train = preprocessor.transform(X_train)
    X_test = preprocessor.transform(X_test)
    return X_train, X_test


def get_random_block_from_data(data, batch_size):
    start_index = np.random.randint(0, len(data) - batch_size)
    print("random_start_index=" + str(start_index), "batch_size=" + str(batch_size))
    return data[start_index:(start_index + batch_size)]


X_train, X_test = standard_scale(mnist.train.images, mnist.test.images)

n_samples = int(mnist.train.num_examples)
training_epochs = 2
batch_size = 128
display_step = 1

autoencoder = AdditiveGaussianNoiseAutoencoder(n_input=784,
                                               n_hidden=200,
                                               transfer_function=tf.nn.softplus,
                                               optimizer=tf.train.AdamOptimizer(learning_rate=0.001),
                                               scale=0.01)

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(n_samples / batch_size)
    print("total_batch:"+str(total_batch))
    for i in range(total_batch):
        batch_xs = get_random_block_from_data(X_train, batch_size)

        cost = autoencoder.partial_fit(batch_xs)
        avg_cost += cost / n_samples * batch_size

    if epoch % display_step == 0:
        print("Epoch", '%04d' % (epoch + 1), "cost=", "{:.9f}".format(avg_cost))
        print(autoencoder.hidden)

print("Total cost:" + str(autoencoder.calc_total_cost(X_test)))
```