In [1]:

```
import warnings
warnings.filterwarnings("ignore")

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

In [3]:

```
# Training Parameters
learning_rate = 0.01
num_steps = 30000
batch_size = 256

display_step = 1000
examples_to_show = 10
```

In [4]:

```
# Network Parameters
num_hidden_1 = 256 # 1st layer num features
num_hidden_2 = 128 # 2nd layer num features (the latent dim)
num_input = 784 # MNIST data input (img shape: 28*28)
```

In [5]:

```
# tf Graph input (only pictures)
X = tf.placeholder("float", [None, num_input])

weights = {
    'encoder_h1': tf.Variable(tf.random_normal([num_input, num_hidden_1])),
    'encoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2])),
    'decoder_h1': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_1])),
    'decoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_input])),
}
biases = {
    'encoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
    'encoder_b2': tf.Variable(tf.random_normal([num_hidden_2])),
    'decoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
    'decoder_b2': tf.Variable(tf.random_normal([num_input])),
}
```

In [6]:

```
# Building the encoder
def encoder(x):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
                                   biases['encoder_b1']))
    # Encoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
                                   biases['encoder_b2']))
    return layer_2
```

In [7]:

```
# Building the decoder
def decoder(x):
    # Decoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']),
                                   biases['decoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
                                   biases['decoder_b2']))
    return layer_2
```

In [8]:

```
# Construct model
encoder_op = encoder(X)
decoder_op = decoder(encoder_op)
```

In [9]:

```
# Prediction
y_pred = decoder_op
# Targets (Labels) are the input data.
y_true = X
```

In [10]:

```
# Define loss and optimizer, minimize the squared error
loss = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
#optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(loss)
optimizer = tf.train.AdamOptimizer (learning_rate).minimize(loss)
```

In [11]:

```
# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

In [12]:

```
sess = tf.InteractiveSession()
```

In [13]:

```
sess.run(init)
```

In [14]:

```python
# Training
vloss = []
for i in range(1, num_steps+1):
    # Prepare Data
    # Get the next batch of MNIST data (only images are needed, not labels)
    batch_x, _ = mnist.train.next_batch(batch_size)

    # Run optimization op (backprop) and cost op (to get loss value)
    _, l = sess.run([optimizer, loss], feed_dict={X: batch_x})
    # Display logs per step
    if i % display_step == 0 or i == 1:
        vloss.append(l)
        print('Step %i: Minibatch Loss: %f' % (i, l))
```

```
Step 1: Minibatch Loss: 0.430625
Step 1000: Minibatch Loss: 0.071511
Step 2000: Minibatch Loss: 0.065912
Step 3000: Minibatch Loss: 0.061884
Step 4000: Minibatch Loss: 0.059398
Step 5000: Minibatch Loss: 0.057097
Step 6000: Minibatch Loss: 0.055169
Step 7000: Minibatch Loss: 0.054546
Step 8000: Minibatch Loss: 0.053669
Step 9000: Minibatch Loss: 0.054318
Step 10000: Minibatch Loss: 0.052241
Step 11000: Minibatch Loss: 0.053284
Step 12000: Minibatch Loss: 0.053250
Step 13000: Minibatch Loss: 0.052586
Step 14000: Minibatch Loss: 0.054612
Step 15000: Minibatch Loss: 0.051785
Step 16000: Minibatch Loss: 0.053546
Step 17000: Minibatch Loss: 0.052066
Step 18000: Minibatch Loss: 0.052791
Step 19000: Minibatch Loss: 0.051464
Step 20000: Minibatch Loss: 0.050152
Step 21000: Minibatch Loss: 0.052197
Step 22000: Minibatch Loss: 0.054017
Step 23000: Minibatch Loss: 0.050568
Step 24000: Minibatch Loss: 0.050024
Step 25000: Minibatch Loss: 0.051199
Step 26000: Minibatch Loss: 0.049455
Step 27000: Minibatch Loss: 0.049745
Step 28000: Minibatch Loss: 0.050995
Step 29000: Minibatch Loss: 0.049980
Step 30000: Minibatch Loss: 0.049278
```

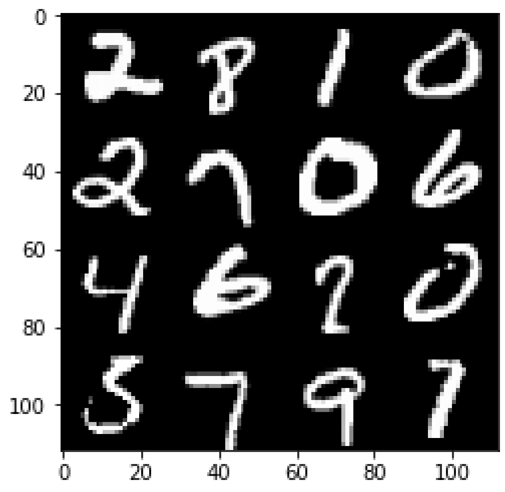
In [15]:

```
# Testing
# Encode and decode images from test set and visualize their reconstruction.
n = 4
canvas_orig = np.empty((28 * n, 28 * n))
canvas_recon = np.empty((28 * n, 28 * n))
for i in range(n):
    # MNIST test set
    batch_x, _ = mnist.test.next_batch(n)
    # Encode and decode the digit image
    g = sess.run(decoder_op, feed_dict={X: batch_x})

    # Display original images
    for j in range(n):
        # Draw the original digits
        canvas_orig[i * 28:(i + 1) * 28, j * 28:(j + 1) * 28] = \
            batch_x[j].reshape([28, 28])
    # Display reconstructed images
    for j in range(n):
        # Draw the reconstructed digits
        canvas_recon[i * 28:(i + 1) * 28, j * 28:(j + 1) * 28] = \
            g[j].reshape([28, 28])
```

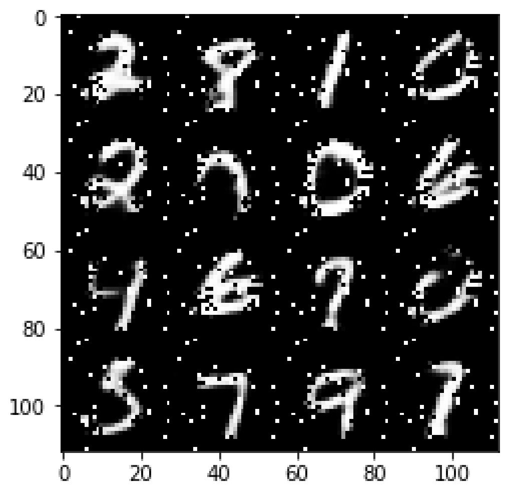
In [16]:

```
print("Original Images")
plt.figure(figsize=(n, n))
plt.imshow(canvas_orig, origin="upper", cmap="gray")
plt.show()

print("Reconstructed Images")
plt.figure(figsize=(n, n))
plt.imshow(canvas_recon, origin="upper", cmap="gray")
plt.show()
```

```
Original Images
```



```
Reconstructed Images
```

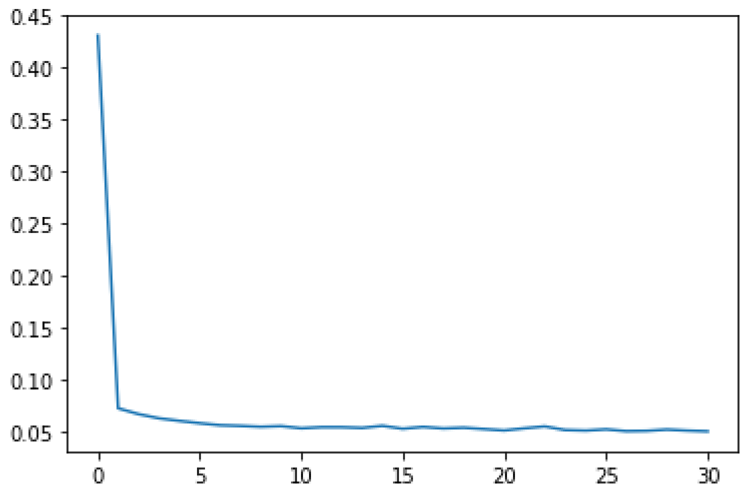


In [17]:

```
plt.figure()
plt.plot(vloss)
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x7f22dff46a58>]
```


0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
5
10
15
20
25
30