HW11: Variational Autoencoders

Yu Che Wang / yuchecw2

December 11, 2018

Abstract

Apply variational autoencoders to MNIST dataset using TensorFlow framework.

1 Single MNIST digits

For 10 pairs of MNIST test images of the same digit, selected at random, compute the code for each image of the pair. Now compute 7 evenly spaced linear interpolates between these codes, and decode the result into images.

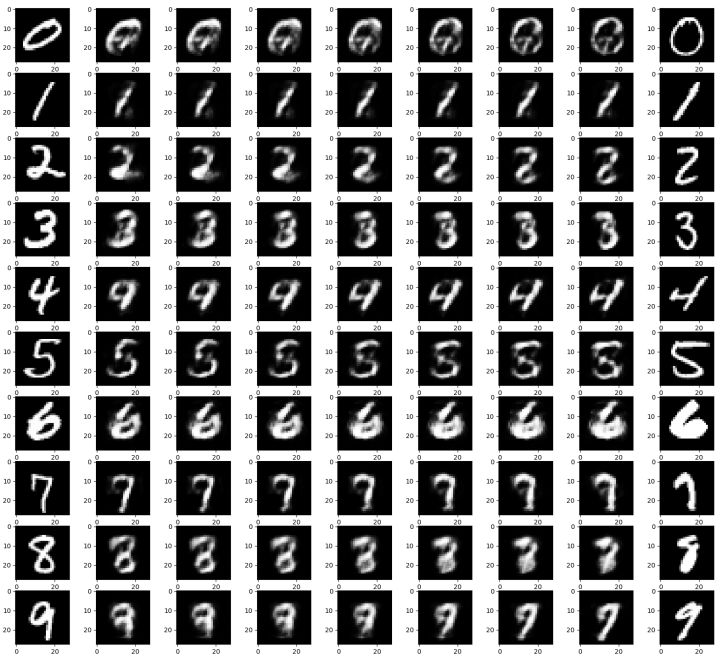

Figure 1: Single MNIST digits.

2 Different MNIST digits

For 10 pairs of MNIST test images of different digits, selected at random, compute the code for each image of the pair. Now compute 7 evenly spaced linear interpolates between these codes, and decode the result into images.

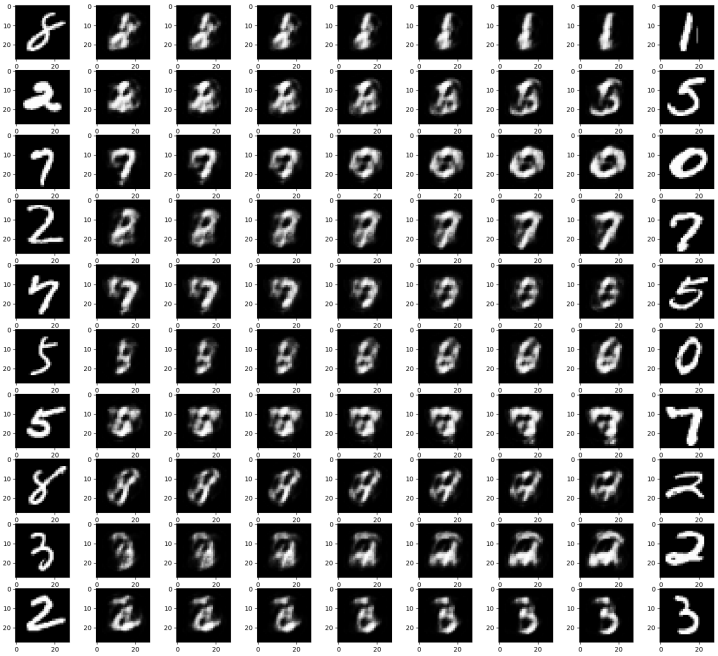

Figure 2: Different MNIST digits.

3 Codes

```python
''' Autoencoder for MNIST dataset'''
from __future__ import division, print_function, absolute_import

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
import matplotlib.pyplot as plt
import argparse

def weight_variable(shape, name):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.1), name=name)

def bias_variable(shape, name):
    return tf.Variable(tf.constant(0.1, shape=shape), name=name)

def encoder(x, in_channels, hid_channels_1, hid_channels_2, name='encoder'):
    with tf.name_scope(name):
        w1 = weight_variable([in_channels, hid_channels_1], name='w1')
        b1 = bias_variable([hid_channels_1], name='b1')
        hid_out_1 = tf.nn.sigmoid(tf.matmul(x, w1) + b1)
        w2 = weight_variable([hid_channels_1, hid_channels_2], name='w2')
        b2 = bias_variable([hid_channels_2], name='b2')
        hid_out_2 = tf.nn.sigmoid(tf.matmul(hid_out_1, w2) + b2)
        return hid_out_2

def decoder(x, hid_channels_2, hid_channels_1, out_channels, name='decoder'):
    with tf.name_scope(name):
        w1 = weight_variable([hid_channels_2, hid_channels_1], name='w1')
        b1 = bias_variable([hid_channels_1], name='b1')
        hid_out_1 = tf.nn.sigmoid(tf.matmul(x, w1) + b1)
        w2 = weight_variable([hid_channels_1, out_channels], name='w2')
        b2 = bias_variable([out_channels], name='b2')
        hid_out_2 = tf.nn.sigmoid(tf.matmul(hid_out_1, w2) + b2)
        return hid_out_2
```

Figure 3: Helper functions.

```python
def train():
    mnist = input_data.read_data_sets('/tmp/data', one_hot=True)
    img_test = mnist.test.images
    label_test = mnist.test.labels
    label_img_dict = {}
    for i in range(label_test.shape[0]):
        num = np.argmax(label_test[i,:])
        if num not in label_img_dict.keys():
            label_img_dict[num] = img_test[i,:]
        else:
            label_img_dict[num] = np.vstack((label_img_dict[num], img_test[i,:]))

    # Network parameters
    in_channels = 784 # = out_channels
    hid_channels_1 = 256
    hid_channels_2 = 32
    # Training parameters
    learning_rate = 0.01
    num_epochs = 100000
    display_epoch = 1000
    batch_size = 1

    x = tf.placeholder(tf.float32, shape=[None, in_channels], name='x') # Input image a.k.a groud truth
    y = tf.placeholder(tf.float32, shape=[None, hid_channels_2], name='y') # Encoder output

    # Auto-encoder Model
    encoder_out = encoder(x, in_channels, hid_channels_1, hid_channels_2) # Latent representation
    decoder_out = decoder(y, hid_channels_2, hid_channels_1, in_channels) # Reconstructed image

    # Calculate loss
    loss = tf.losses.mean_squared_error(x, decoder_out)
    optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(loss)

    # Initialize the variables
    init = tf.global_variables_initializer()
```

Figure 4: Main function.

```
    with tf.Session() as sess:
        sess.run(init)
        print('Start session!')
        # Training
        loss_list = []
        for epoch in range(num_epochs+1):
            batch = mnist.train.next_batch(batch_size)
            en_out = sess.run(encoder_out, feed_dict={x: batch[0]})
            _, mse_loss = sess.run([optimizer, loss], feed_dict={x: batch[0], y: en_out})
            loss_list.append(mse_loss)
            if epoch % display_epoch == 0:
                print('Epoch %d, loss: %.3f'%(epoch, mse_loss))
        plt.figure()
        plt.plot(list(range(len(loss_list))), loss_list)
        plt.xlabel('Epoch number')
        plt.ylabel('Loss')
        plt.title('Loss v.s. Epoch')
        plt.show()
        # Testing
        for are_same_digits in [True, False]:
            plt.figure(figsize=(20,18))
            for digit in range(10):
                digit1, digit2 = None, None
                if are_same_digits: # Same digits
                    digit_arr = label_img_dict[digit]
                    # Randomly select two same digits
                    same_digits = digit_arr[np.random.choice(digit_arr.shape[0], 2, replace=False), :]
                    digit1 = same_digits[0,:].reshape(1,-1)
                    digit2 = same_digits[1,:].reshape(1,-1)
                else:   # Different digits
                    # Randomly select two different digits
                    permutation_arr = np.random.permutation(10)
                    num1 = permutation_arr[0]
                    num2 = permutation_arr[1]
                    digit1 = label_img_dict[num1][np.random.choice(label_img_dict[num1].shape[0], 1, replace=False), :]
                    digit2 = label_img_dict[num2][np.random.choice(label_img_dict[num2].shape[0], 1, replace=False), :]
                # Calculate corresponding codes
                code1 = sess.run(encoder_out, feed_dict={x: digit1})
                code2 = sess.run(encoder_out, feed_dict={x: digit2})
                reconstructed_img = [None]*9
                reconstructed_img[0], reconstructed_img[8] = digit1, digit2
                # Compute 7 evenly spaced linear interpolates
                for t in range(1, 8):
                    code = ((8-t)/8)*code1 + (t/8)*code2
                    reconstructed_img[t] = sess.run(decoder_out, feed_dict={y: code})
                for idx in range(9):
                    plt.subplot(10,9,9*digit+idx+1)
                    plt.imshow(reconstructed_img[idx].reshape(28,28), cmap='gray')
            if are_same_digits:
                plt.savefig('same.png')
            else:
                plt.savefig('different.png')
```

Figure 5: (Continued)Main function.