Assignment 1 for #70240413 "Statistical Machine Learning"

Kui XU, 2016311209

2017/06/02

1 Preparation

Summary:

(1) Python is my basic language, I use it everyday.

(2) And numpy is a very famous python package and I can not programming without it yet.

(3) For Tensorflow, I am still a junior user, so I learn more in doing this homework.

(4) For ZhuSuan, a very great bayesian network library, a new programming mode for us to design a bayesian network by our self. I am a new user, I read a lot of documents to understanding the concepts of ZhuSuan, and I try to learn the programming mode by following the tutorial of VAE.

2 Variational inference for 2-D Gaussian Mixture

Here, fistly I introduce the the model (named GVAE model). Based on the pre-designed model, it uses $Discrete(\pi)$ as the prior for Z; and uses $N(\mu_{Z^i}, diag\sigma^2_{Z^i})$ as the prior for X.

The dataset is sampled with three fixed point, with $\mu \in (-0.2, 0.2, 0.6)$.

No time for to present more details of my implementation, and I will give more in the final version, now I am proud to show the results below.

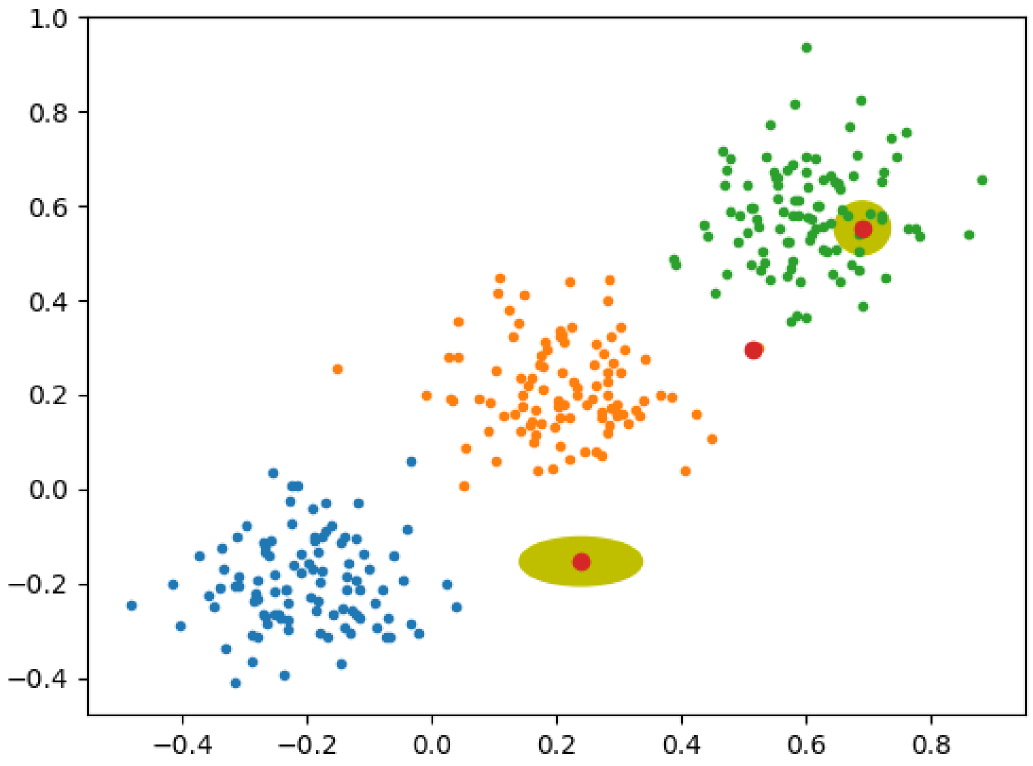
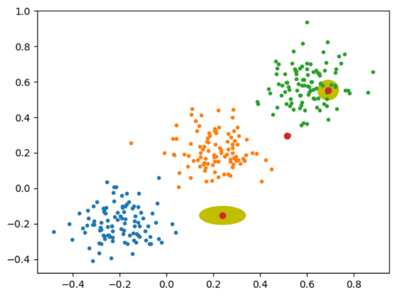
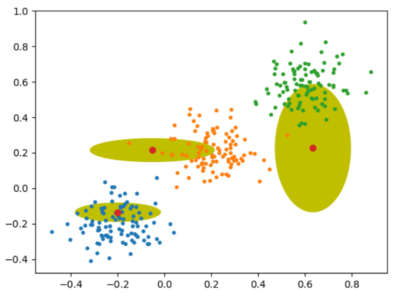
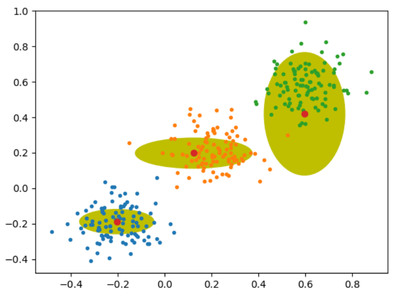
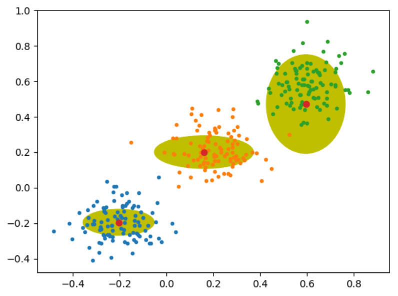


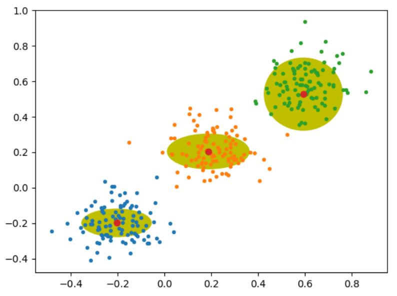
Figure 1: The figure shows the sampled data by three fixed point.

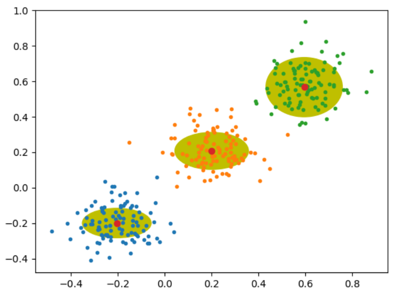

(a) Epoch 100

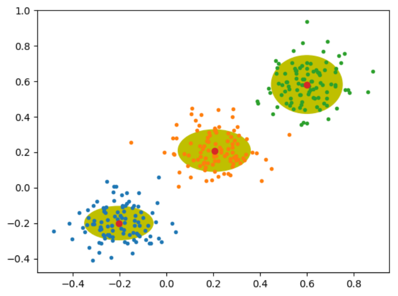

(b) Epoch 200

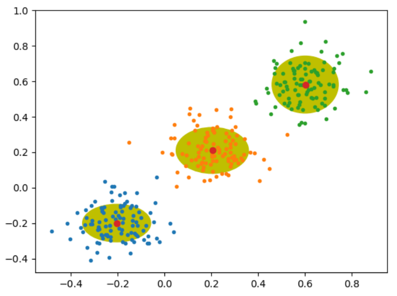

(c) Epoch 1000


(d) Epoch 1200


(e) Epoch 1400


(f) Epoch 1600


(g) Epoch 2000


(h) Epoch 2400

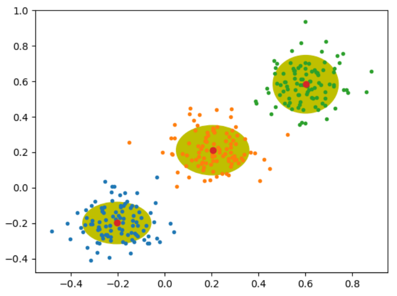
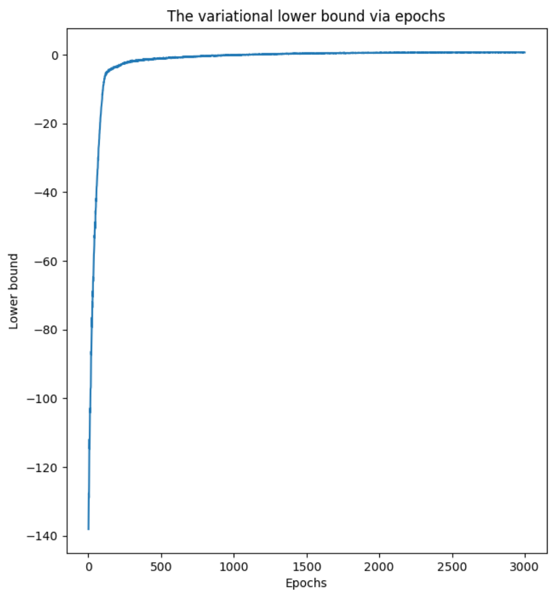

(i) Epoch 2800

Figure 2: The cluster center is changed over the epoch.



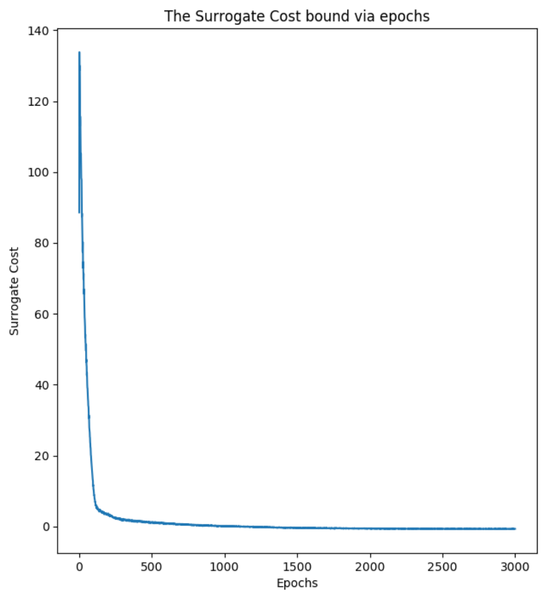


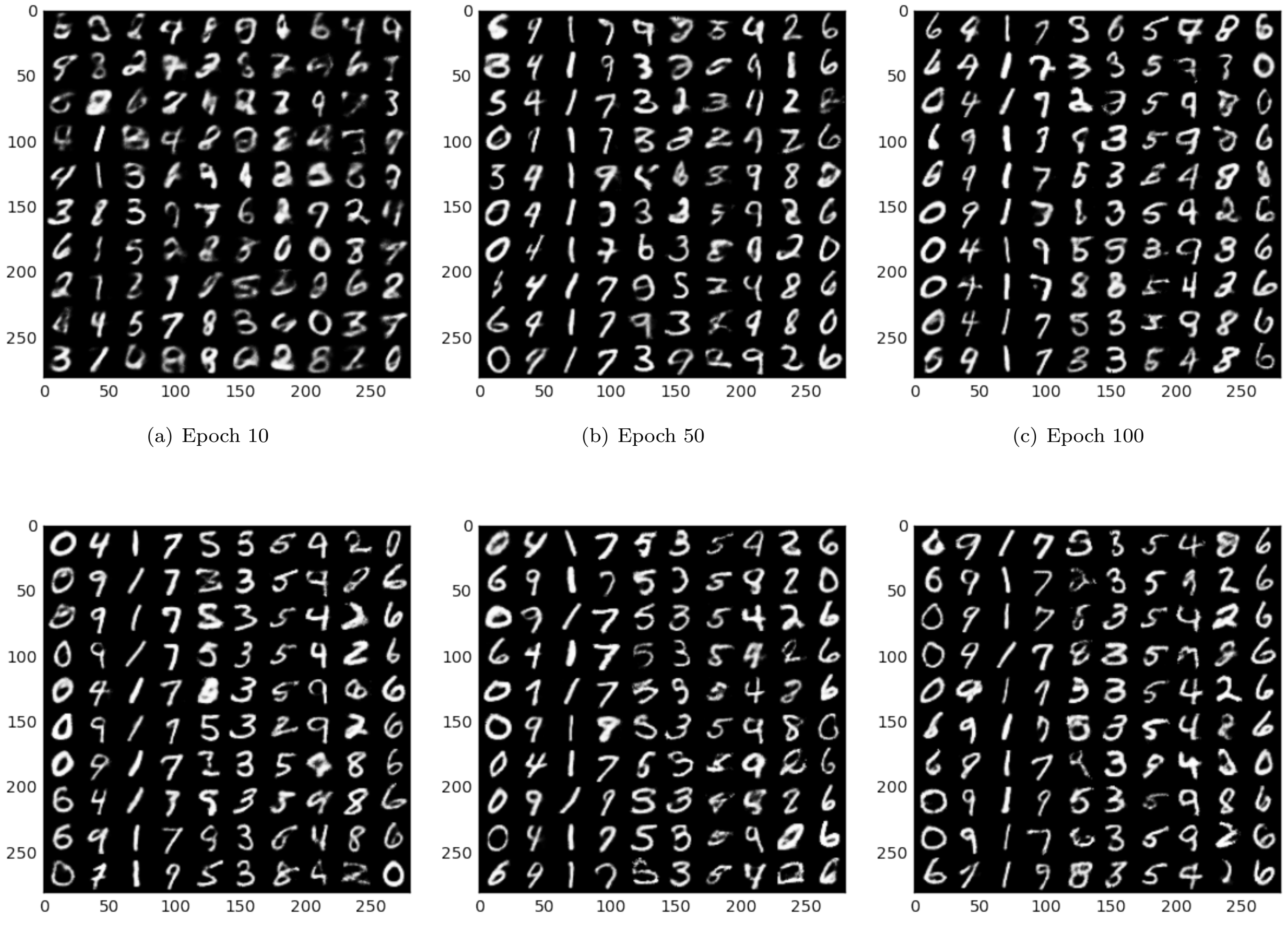
Figure 3: The figure shows the sampled data by three fixed point.

3 Gaussian Mixture VAE

3.1 Detailed Description

Here, fistly I introduce the the model (named GMVAE model). Based on the pre-designed model, it uses $Discrete(\pi)$ as the prior for Z; and uses $N(\mu_{Z^i}, diag\sigma^2_{Z^i})$ as the prior for H, and then a feed-forword neural network is used to learn the lantent representation with the input of H, finally X is generated by a Bernoulli distribution which represent the 10 different type of digit image in the MNIST dataset.
No time for to present more details of my implementation, and I will give more in the final version, now I am proud to show the results below.

3.2 Results



(a) Epoch 10

(b) Epoch 50

(c) Epoch 100

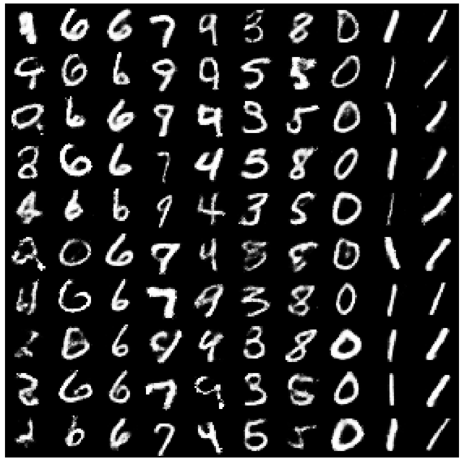
(d) Epoch 150

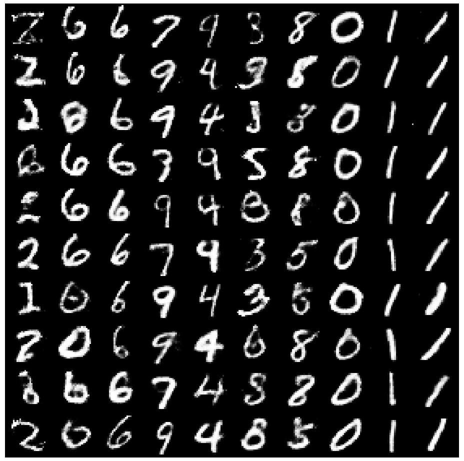
(e) Epoch 200

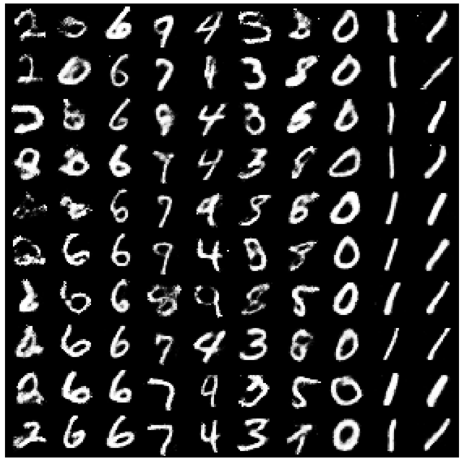
(f) Epoch 500

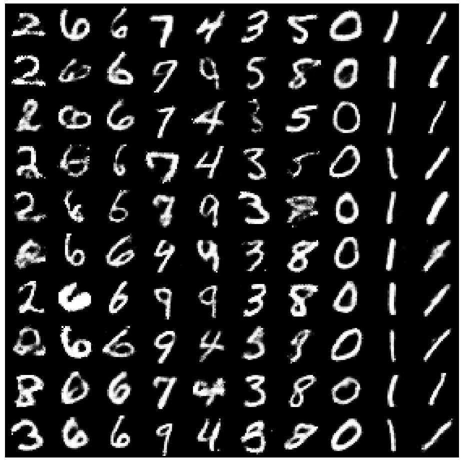
Figure 4: Experiment 1 for generating the image.

(a) Epoch 600


(b) Epoch 700


(c) Epoch 800


(d) Epoch 900


(e) Epoch 950

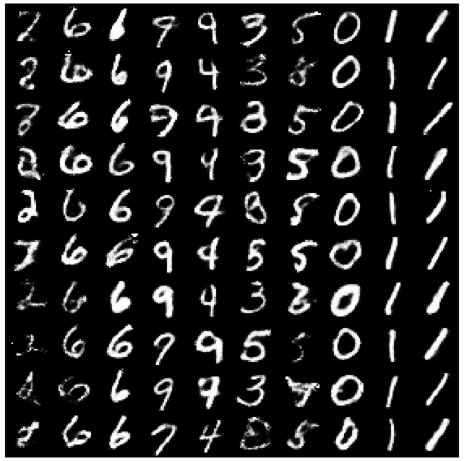

(f) Epoch 1000

Figure 5: Experiment 2 for generating the image.

The low bound curve will be showed later.

3.3 Implemention

I implemented the algorithm using ZhuSuan. Code lists below.

Code of **vGM.py**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from __future__ import absolute_import
from __future__ import print_function
from __future__ import division
import os
import time

import tensorflow as tf
from tensorflow.contrib import layers
from six.moves import range
import numpy as np
import zhusuan as zs

from matplotlib.patches import Ellipse
import matplotlib as mpl
#%matplotlib inline
from matplotlib import pyplot as plt
mpl.use('Agg')

def sampleData(prng, nb_samples=100):
    """sample data.
    """
    x_coor=[]
    y_coor=[]
    t_train=[]
    #[2, 5.], [5., 2.],[7.,8.]
    classid=0
    for mu, sigma, marker in [(-.2, 0.1, 'o'), (0.2, 0.1, 's'), (0.6, 0.1, '+')]:
        x, y = prng.normal(loc=mu, scale=sigma, size=(2, nb_samples))
        x_coor.extend(x)
        y_coor.extend(y)
        t_train.extend([classid]*100)
        classid+=1
    x_train = np.column_stack([x_coor,y_coor])
    return x_train,t_train


resample=False
if resample:
    prng = np.random.RandomState(96917002)
    x_train, t_train=sampleData( prng)
    np.savez("data.npz",x_train, t_train)
    print("Sampling data")
```

```
r=np.load("data.npz")
x_train, t_train=r['arr_0'],r['arr_1']
x_train_cp = r['arr_0']
n_x = x_train.shape[1]

tf.set_random_seed(1237)
# Define model parameters
n_z = 3
K = 3
D = n_x

@zs.reuse('model')
def gmm(observed, n, K,D): #vae(observed, n, n_x, n_z):
    with zs.BayesianNet(observed=observed) as model:
        # Discrete GMM
        #Step.1: z ~ Discrete(pi)
        pi = tf.get_variable(name="pi",shape=[1,K],initializer=tf.constant_initializer
                                 (np.random.dirichlet(np.ones(3),size=1)),\
                        dtype=tf.float32)
        pi_n = tf.tile(pi,[n,1])
        z = zs.OnehotDiscrete('z', tf.log(pi_n),dtype=tf.float32)
        print("pi_shape:",pi.shape)
        print(z.sample(4).shape)
        miu = tf.get_variable(name="miu", initializer=tf.random_uniform_initializer(\
                minval=-0.3,maxval=0.7,dtype=tf.float32), shape=[K,D], dtype=tf.float32)
        sigma = tf.get_variable(name="sigma", initializer=tf.random_uniform_initializer(\
                minval=0,maxval=0.2,dtype=tf.float32), shape=[K,D], dtype=tf.float32)

        #miu_z = tf.matmul(tf.cast(z, dtype=tf.float32),miu)
        mean_x = tf.tensordot(tf.cast(z, dtype=tf.float32),miu, axes=1)
        print("mean_x:",tf.shape(mean_x))
        #sigma_z = tf.matmul(tf.cast(z, dtype=tf.float32),sigma)
        std_x = tf.tensordot(tf.cast(z, dtype=tf.float32),sigma, axes=1)
        print("std_x:",tf.shape(std_x))

        x = zs.Normal('x', mean_x, tf.log(std_x), group_event_ndims=1)
        print("x:",tf.shape(x))

    return model, x, miu, sigma, pi

# z inference
@zs.reuse('variational')
def q_net(x, n, K): #def q_net(x, n_z):
    with zs.BayesianNet() as variational:
        # Discrete GMM
        lz_x = layers.fully_connected(tf.to_float(x), 500)
        lz_x = layers.fully_connected(lz_x, 500)
        log_pi = layers.fully_connected(lz_x, K, activation_fn=None)
        z = zs.OnehotCategorical('z', log_pi,n_samples=n,  group_event_ndims=0,dtype=tf.float32)

    return variational,tf.arg_max(log_pi,dimension=1)

x = tf.placeholder(tf.float32, shape=[None, n_x], name='x')
n = tf.shape(x)[0]

def log_joint(observed):
    model, _ ,_ ,_,_ = gmm(observed, n, K, D)
    log_pz, log_px_z = model.local_log_prob(['z', 'x'])
    print("log_pz:", log_pz.shape)
    print("log_px_z:", log_px_z.shape)
    return tf.cast(log_pz + log_px_z, dtype=tf.float32)

variational,max_pi = q_net(x, n, K)
qz_samples, log_qz = variational.query('z', outputs=True,
                                        local_log_prob=True)

#Ocost, Olower_bound = zs.nvil(log_joint,\
#Ocost, Olower_bound = zs.nvil(log_joint,decay=tf.cast(0.2,dtype=tf.float32), variance_normalization
#Ocost, Olower_bound = zs.nvil(log_joint,decay=tf.cast(0.2,dtype=tf.float32),\
# === Amazing code
Ocost, Olower_bound = \
  zs.nvil(log_joint,\
    decay=tf.cast(0.2,dtype=tf.float32), \
    variance_normalization=True, \
    observed={'x': x}, \
  latent={'z': [tf.cast(qz_samples, dtype=tf.float32), tf.cast(log_qz, dtype=tf.float32)]})

Mcost = Ocost
Mlower_bound = tf.reduce_mean(Olower_bound)
optimizer = tf.train.AdamOptimizer(0.0001)
infer = optimizer.minimize(Mcost)

# Generate data
n_gen = 80
_, _, mean_x,std_x, pi  = gmm({}, n_gen, K, D)

# Define training parameters
epoches = 3000
batch_size = 80
iters = x_train.shape[0] // batch_size
save_freq = 200

def plotResult(iepoch):
    fig, ax = plt.subplots()


    ax.plot(x_train[0:100,0],x_train[0:100,1],".")
    ax.plot(x_train[101:200,0],x_train[101:200,1],".")
    ax.plot(x_train[201:,0],x_train[201:,1],".")

    x_means = sess.run(mean_x)
    print("x_means:",x_means)
    x_std = sess.run(std_x)
    print("x_std:",x_std)
    x_pi = sess.run(pi)

    #x_means= np.matmul(x_means,x_pi)
    ax.plot(x_means[:,0],x_means[:,1],'o')

    #mean = [ 19.92977907 ,  5.07380955]
    widths = 3*(x_std[:,0])
    heights = 3*(x_std[:,1])
    angle = 0
    ells = [mpl.patches.Ellipse(xy=x_mean, width=width, height=height, \
            angle = angle, color="y") for i, (x_mean, width, height) in \
            enumerate(zip(x_means, widths, heights))]


    [ax.add_patch(ell) for ell in ells ]
    fig.savefig('./gmm/'+str(iepoch)+'_learn.png')
    #fig.tight_layout()
    #plt.show()

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    lbes= []
    costses = []
    for epoch in range(1, epoches + 1):
        np.random.shuffle(x_train_cp)
        lbs = []
        costs = []
        pis = []
        for t in range(iters):
            x_batch = x_train_cp[t * batch_size:(t + 1) * batch_size]
            _, lb, cost,x_mean, tmp_pi = sess.run([infer,Mlower_bound, Mcost, mean_x, pi],
                             feed_dict={x: x_batch})
            lbs.append(lb)
            costs.append(cost)
            pis.extend(tmp_pi)
        m_pis = np.asarray(pis)
        print('Epoch {}: Lower bound = {}, Cost = {}, Pi = {}'.format(
            epoch, np.mean(lbs), np.mean(costs),np.mean(m_pis,axis=1)))
        lbes.append(np.mean(lbs))
        costses.append(np.mean(costs))
        if epoch % save_freq == 0:
            plotResult(epoch)
    fig_size = [16, 8]
    fig, axes = plt.subplots(ncols=2, nrows=1, num=0,
                             figsize=fig_size, squeeze=True)
    axes[0].plot(range(1, epoches + 1),lbes)
    axes[0].set_ylabel('Lower bound')
    axes[0].set_xlabel('Epochs')
    axes[0].set_title('The variational lower bound via epochs')
    axes[0].legend(loc='center right', shadow=True)
    axes[1].plot(range(1, epoches + 1),costses)
    axes[1].set_ylabel('Surrogate Cost')
    axes[1].set_xlabel('Epochs')
    axes[1].set_title('The Surrogate Cost bound via epochs')
    axes[1].legend(loc='upper right', shadow=True)
    fig.savefig('./gmm/cost_lb.png')
```

Code of running **GMVAE.py**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
from __future__ import absolute_import
from __future__ import print_function
from __future__ import division
import os

import tensorflow as tf
from tensorflow.contrib import layers
from six.moves import range
import numpy as np
import zhusuan as zs

from examples import conf
from examples.utils import dataset, save_image_collections
import warnings
warnings.filterwarnings("ignore")

tf.reset_default_graph()

@zs.reuse('model')
def gmmvae(observed, n_x, N, D, K):
    with zs.BayesianNet(observed=observed) as model:
        pi_ = tf.get_variable(name='pi',shape=[K], dtype = tf.float32, \
                              initializer = tf.random_uniform_initializer(minval=0.,\
                                maxval=0., dtype=tf.float32))
        # N*K
        pi_ = tf.tile(tf.expand_dims(pi_,0),[N,1])
        # N*K
        z = zs.OnehotDiscrete('z',pi_, dtype = tf.float32, group_event_ndims=0)
        # K*D
        h_mean = tf.get_variable(name='mean_h', dtype=tf.float32, shape=[K, D],\
                                  initializer=tf.random_uniform_initializer(minval=0.,\
                                    maxval=1,seed=None, dtype=tf.float32))
        # K*D
        h_sigma = tf.get_variable(name = 'sigma_h', dtype = tf.float32, shape=[K, D],\
                                  initializer=tf.random_uniform_initializer(minval=0.,\
                                    maxval=1, seed=None,dtype=tf.float32))
                                   #initializer = tf.random_uniform([K,D], dtype=tf.float32))
        # N*D
        h_mean = tf.matmul(z, h_mean)
        h_sigma = tf.matmul(z, h_sigma)
        # N*D
        h = zs.Normal('h', h_mean, h_sigma, group_event_ndims=1)
        fNN = layers.fully_connected(tf.to_float(h), 500)
        fNN = layers.fully_connected(fNN, 500)
        # N*n_x
        x_logits = layers.fully_connected(fNN, n_x, activation_fn=None)
        x = zs.Bernoulli('x', x_logits, group_event_ndims=1, dtype=tf.float32)
        return model, x_logits

@zs.reuse('variational')
def q_net(x, n_x, N, D, K):
    with zs.BayesianNet() as variational:
        # N*n_x
        fNN1 = layers.fully_connected(tf.to_float(x), 500)
        fNN1 = layers.fully_connected(fNN1, 500)
        h_mean = layers.fully_connected(fNN1, D, activation_fn=None)
        h_sigma = layers.fully_connected(fNN1, D, activation_fn=None)
        h = zs.Normal('h',h_mean,h_sigma,group_event_ndims=1)
        return variational

if __name__ == "__main__":
    # Load MNIST
    data_path = os.path.join(conf.data_dir, 'mnist.pkl.gz')
    x_train, t_train, x_valid, t_valid, x_test, t_test = \
    dataset.load_mnist_realval(data_path)
    x_train = np.random.binomial(1, x_train, size=x_train.shape)
    n_x = x_train.shape[1]  # 784,shape=(50000,784)
    D = 40
    K = 10
    batch_size = 256
    N = batch_size

    x = tf.placeholder(tf.float32, shape=[None, n_x], name='x')

    def sum_z_log_joint(observed,N,K):
        ober_z = np.eye(K)
        sum_z = []
        #sum_z = 0
        for i in range(K):
            ober_zi = np.tile(ober_z[i,], [N,1])
            observed['z'] = ober_zi
            model, _ = gmmvae(observed, n_x, N, D, K)       # only need the model
            log_pz, log_ph_z = model.local_log_prob(['z','h'])
            sum_z.append(log_pz+ log_ph_z)
            #sum_z+= tf.exp(log_pz+ log_ph_z)
        return tf.reduce_logsumexp(sum_z, axis = 0)
        #return tf.log(sum_z)

    def log_joint(observed):
        log_ph = sum_z_log_joint(observed, N, K)
        model, _ = gmmvae(observed, n_x, N, D, K)
        log_px_h = model.local_log_prob(['x'])
        return log_px_h  + log_ph

    variational = q_net(x, n_x, N, D, K)

    qh_samples, log_qh = variational.query('h', outputs=True,local_log_prob=True)
    qh_samples = tf.cast(qh_samples, dtype=tf.float32)
    log_qh = tf.cast(log_qh, dtype=tf.float32)

    lower_bound = tf.reduce_mean(
        zs.sgvb(log_joint,observed={'x': x},latent={'h':[qh_samples, log_qh]}))

    optimizer = tf.train.AdamOptimizer(0.001)
    infer = optimizer.minimize(-lower_bound)

    # Generate images
    n_gen = 100
    ober_z = np.eye(K)
    ober_z = np.tile(ober_z, [n_gen,1])
```

```
_, x_logits= gmmvae({'z':ober_z}, n_x, n_gen, D, K)
# only need the x_logits: N*D
x_gen = tf.reshape(tf.sigmoid(x_logits), [-1, 28, 28, 1]) # N*28*28*1
# Define training parameters
epoches = 1000
iters = x_train.shape[0] // batch_size  #50000/128=390
save_freq = 1

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    lbes= []
    for epoch in range(1, epoches + 1):
        np.random.shuffle(x_train)
        lbs = []
        for t in range(iters):
            x_batch = x_train[t * batch_size:(t + 1) * batch_size]
            _, lb = sess.run([infer, lower_bound], feed_dict={x: x_batch})
            lbs.append(lb)
        print('Epoch {}: Lower bound = {}'.format(epoch, np.mean(lbs)))
        lbes.append(np.mean(lbs))
        if epoch % save_freq == 0:
            images = sess.run(x_gen)  # gen batch-szie images
            name = "results/gmmvae/gmmvae.epoch.{}.png".format(epoch)
            save_image_collections(images, name)
    fig_size = [16, 8]
    fig, ax = plt.subplots()
    ax.plot(range(1, epoches + 1),lbes)
    ax.set_ylabel('Lower bound')
    ax.set_xlabel('Epochs')
    ax.set_title('The variational lower bound via epochs')
    ax.legend(loc='center right', shadow=True)

    fig.savefig('results/gmmvae/lb.png')
```