Homework 7

By Fedor Ivachev 费杰 2019280373

1. **Following the variational Bayes algorithm of the original VAE, derive the algorithm for this class-conditional variant. Specifically, you need to design the variational distribution q(z|x, y) and write down the variational lower bound.**

$$\log p(x|y) \geq E_q(z|x,y) \log p(x|z,y) - KL(q(z|x,y)||p(z)$$

Infer $p(z|x,y)$ using $q(z|x,y)$. Other words, we try to use simpler distribution.

CVAE objective function:

$$\log p(x|y) - D_{KL}(q_\phi(z|x,y)||p_\psi(z|x,y)) = E[\log p(z|x,y)] - D_{KL}(q_\phi(z|x,y)||p_\psi(z|y))$$

The data is described using this $\log p(x|y)$, error is shown by $D_{KL}(q_\phi(z|x,y)||p_\psi(z|x,y))$, $q_\phi(z|x,y)$ is Gaussian distribution, which is determined by neural network (distribution $N(\mu(x),\sigma(x))$.

We change the upper equation right part into $L(x,y,\theta,\phi)$ - variational lower bound.

Since the model uses z independently from y, we use $N(0,1)$.
Here we use Bernoulli distribution.
$E(\log p(z|x,y)) = \frac{1}{L}\sum_L \log p(x|z,y)$, where L is the number of samples drawn a.t. re-parametrization trick.

2. **Implement the algorithm using ZhuSuan, and train the model on the whole training set of MNIST.**

```
from __future__ import absolute_import
from __future__ import print_function
from __future__ import division
import os
import time
import tensorflow as tf
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


@zs.meta_bayesian_net(scope="gen", reuse_variables=True)
def build_gen(x_dim, z_dim, n, n_particles=1):
    bn = zs.BayesianNet()
    z_mean = tf.zeros([n, z_dim])
    z = bn.normal("z", z_mean, std=1., group_ndims=1, n_samples=n_particles)
    h = tf.layers.dense(z, 500, activation=tf.nn.relu)
    h = tf.layers.dense(h, 500, activation=tf.nn.relu)
    x_logits = tf.layers.dense(h, x_dim)
    bn.deterministic("x_mean", tf.sigmoid(x_logits)) ##mapping
    bn.bernoulli("x", x_logits, group_ndims=1) ## Bernoulli
    return bn

@zs.reuse_variables(scope="q_net")
def build_q_net(x, z_dim, n_z_per_x):
    bn = zs.BayesianNet()
    h = tf.layers.dense(tf.cast(x, tf.float32), 500, activation=tf.nn.relu)
    h = tf.layers.dense(h, 500, activation=tf.nn.relu)
    z_mean = tf.layers.dense(h, z_dim)
    z_logstd = tf.layers.dense(h, z_dim)
    bn.normal("z", z_mean, logstd=z_logstd, group_ndims=1, n_samples=n_z_per_x)
    return bn

def 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.vstack([x_train, x_valid])
    x_test = np.random.binomial(1, x_test, size=x_test.shape)
    x_dim = x_train.shape[1]

    # Define model parameters
    z_dim = 40 ## we fix d = 40

    # Build the computation graph
    n_particles = tf.placeholder(tf.int32, shape=[], name="n_particles")
    x_input = tf.placeholder(tf.float32, shape=[None, x_dim], name="x")
    x = tf.cast(tf.less(tf.random_uniform(tf.shape(x_input)), x_input),
                tf.int32)
```

```
n = tf.placeholder(tf.int32, shape=[], name="n")

model = build_gen(x_dim, z_dim, n, n_particles)
variational = build_q_net(x, z_dim, n_particles)

lower_bound = zs.variational.elbo(
    model, {"x": x}, variational=variational, axis=0)
cost = tf.reduce_mean(lower_bound.sgvb())
lower_bound = tf.reduce_mean(lower_bound)

# # Importance sampling estimates of marginal log likelihood
is_log_likelihood = tf.reduce_mean(
    zs.is_loglikelihood(model, {"x": x}, proposal=variational, axis=0))

optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
infer_op = optimizer.minimize(cost)

# Random generation
x_gen = tf.reshape(model.observe()["x_mean"], [-1, 28, 28, 1])

# Define training/evaluation parameters
epochs = 50
batch_size = 128
iters = x_train.shape[0] // batch_size
save_freq = 10

# Run the inference
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for epoch in range(1, epochs + 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_op, lower_bound],
                             feed_dict={x_input: x_batch,
                                        n_particles: 1,
                                        n: batch_size})
            lbs.append(lb)
        print("Epoch {}: Lower bound {}".format(
            epoch, np.mean(lbs)))

        if epoch % save_freq == 0:
            images = sess.run(x_gen, feed_dict={n: 100, n_particles: 1})
```

```python
        name = os.path.join( “res” ,
                            "vae.epoch.{}.png".format(epoch))
        save_image_collections(images, name, shape=(1,10))

if __name__ == "__main__":
    main()
```

```
Epoch 1: Lower bound = -174.3189239501953
Epoch 2: Lower bound = -126.609375
Epoch 3: Lower bound = -115.97903442382812
Epoch 4: Lower bound = -111.89781951904297
Epoch 5: Lower bound = -109.23577117919922
Epoch 6: Lower bound = -107.38616943359375
Epoch 7: Lower bound = -105.96649169921875
Epoch 8: Lower bound = -104.83159637451172
Epoch 9: Lower bound = -103.60923767089844
Epoch 10: Lower bound = -102.87323760986328
Epoch 11: Lower bound = -102.16043853759766
Epoch 12: Lower bound = -101.60298919677734
Epoch 13: Lower bound = -101.01287078857422
Epoch 14: Lower bound = -100.64302825927734
Epoch 15: Lower bound = -100.2274169921875
Epoch 16: Lower bound = -99.846923828125
Epoch 17: Lower bound = -99.53782653808594
Epoch 18: Lower bound = -99.3237533569336
Epoch 19: Lower bound = -98.99461364746094
Epoch 20: Lower bound = -98.66244506835938
Epoch 21: Lower bound = -98.47322082519531
Epoch 22: Lower bound = -98.23519134521484
Epoch 23: Lower bound = -98.07992553710938
Epoch 24: Lower bound = -97.86445617675781
Epoch 25: Lower bound = -97.69099426269531
Epoch 26: Lower bound = -97.52287292480469
Epoch 27: Lower bound = -97.38142395019531
Epoch 28: Lower bound = -97.26490783691406
Epoch 29: Lower bound = -97.12347412109375
Epoch 30: Lower bound = -97.02249908447266
Epoch 31: Lower bound = -96.90144348144531
Epoch 32: Lower bound = -96.78202056884766
Epoch 33: Lower bound = -96.67034149169922
Epoch 34: Lower bound = -96.50209045410156
Epoch 35: Lower bound = -96.52604675292969
Epoch 36: Lower bound = -96.42350769042969
Epoch 37: Lower bound = -96.34712982177734
Epoch 38: Lower bound = -96.20309448242188
Epoch 39: Lower bound = -96.1180191040039
Epoch 40: Lower bound = -96.0550765991211
Epoch 41: Lower bound = -95.92442321777344
Epoch 42: Lower bound = -95.84114837646484
Epoch 43: Lower bound = -95.8077621459961
Epoch 44: Lower bound = -95.74011993408203
Epoch 45: Lower bound = -95.70277404785156
Epoch 46: Lower bound = -95.64720153808594
Epoch 47: Lower bound = -95.60064697265625
Epoch 48: Lower bound = -95.55496978759766
Epoch 49: Lower bound = -95.41159057617188
Epoch 50: Lower bound = -95.39166259765625
```

3. **Visualize the generations of your learned model. Set y observed as {1, 2, . . . , K}, and generate multiple xs for each y using your learned model. Include a few samples in your report.**