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.



1. 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})

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

1. 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.

 **** ****

**** ****