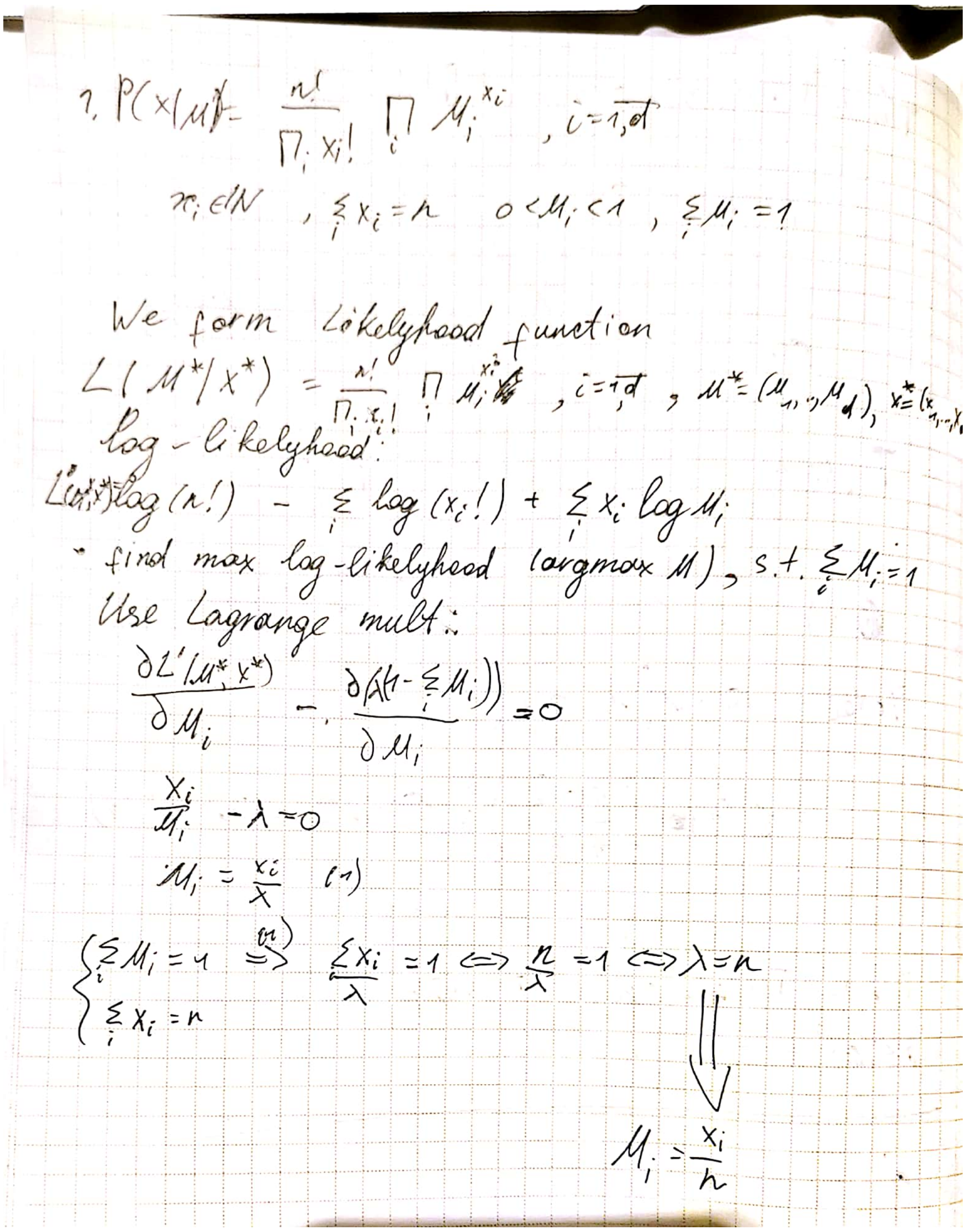
Student id: 2019280373 Name: 费杰

1.1



1. $P(x|\mu) = \frac{n!}{\prod_i x_i!} \prod_i \mu_i^{x_i}$, $i=\overline{1,d}$

$x_i \in \mathbb{N}$, $\sum_i x_i = n$ $\quad 0<\mu_i<1$, $\sum_i \mu_i = 1$

We form Likelyhood function

$L(\mu^*/x^*) = \frac{n!}{\prod_i x_i!} \prod_i \mu_i^{x_i}$, $i=\overline{1,d}$, $\mu^*=(\mu_1,\dots,\mu_d)$, $x^*=(x_1,\dots,x$

log-likelyhood:

$L'(\mu^*,x^*) = \log(n!) - \sum_i \log(x_i!) + \sum_i x_i \log \mu_i$

• find max log-likelyhood (argmax μ), s.t. $\sum_i \mu_i = 1$

Use Lagrange mult.:

$$\frac{\partial L'(\mu^*,x^*)}{\partial \mu_i} - \frac{\partial(\lambda(1-\sum_i \mu_i))}{\partial \mu_i} = 0$$

$$\frac{x_i}{\mu_i} - \lambda = 0$$

$$\mu_i = \frac{x_i}{\lambda} \quad (1)$$

$$\begin{cases} \sum_i \mu_i = 1 \\ \sum_i x_i = n \end{cases} \overset{(1)}{\Rightarrow} \frac{\sum x_i}{\lambda} = 1 \Leftrightarrow \frac{n}{\lambda} = 1 \Leftrightarrow \lambda = n$$

$$\Downarrow$$

$$\mu_i = \frac{x_i}{n}$$

1.2

Probability of document d belonging to the topic c is $P(c_d = k \mid d) = \frac{P(d \mid c_d = k) * P(c_d = k))}{P(d)}$

E-step:We denote $\gamma_d(k) = \frac{\pi_k * \Pi_w \mu_{wk}^{T_{dw}}}{\sum_{j=1}^{K} \pi_j \Pi_w \mu_{wk}^{T_d w}}$, which is the function of expectation. We create on the E-step.
M-step:
We denote $\theta = argmax_{\pi,\mu} \ln L(\theta) = argmax_{\pi,\mu} \sum_D \ln(\sum_{k=1}^{K} \pi_k \frac{n_d!}{\Pi_w T_{dw}!} \Pi_w \mu_{wk}^{T_{dw}})$

We use Lagrange multiplier:
$\theta = argmax_{\pi,\mu} \sum_D \ln(\sum_{k=1}^{K} \pi_k \frac{n_d!}{\Pi_w T_{dw}!} \Pi_w \mu_{wk}^{T_{dw}})$
s.t:
$\sum_k \pi_k = 1$
$\frac{\partial \theta}{\partial \pi_k} = 0 \Leftrightarrow \sum_D \sum_k \gamma_d(k) = \sum_k \lambda \pi_k$
Since $\sum_k \pi_k = 1$ and $\sum_k \gamma_d(k) = 1 : \lambda = n_d$,

$$\pi_k = \frac{\sum_D \gamma_d(k)}{n_d}$$

$$\frac{\partial \theta}{\partial \mu_{wk}} = \frac{\sum_D T_{dw} \gamma_d(k)}{\sum_d \sum_w T_{dw} \gamma_d(k)} = \frac{\sum_D T_{dw} \gamma_d(k)}{\sum_D n_d \gamma_d(k)}$$

2

To maximize the variance, we should minimize the mean-squared error:

$$J = \frac{1}{N} \sum_{n=1}^{N} (x_n^2 + (\sum_{i=1}^{d} z_{ni}\mu_i + \sum_{i=d+1}^{p} b_i \mu_i)^2) = \frac{1}{N} \sum_{n=1}^{N} (x_n^2 + (\sum_{i=1}^{d} z_{ni}\mu_i)^2 + 2(\sum_{i=1}^{d} z_{ni}\mu_i \sum_{i=d+1}^{p} b_i \mu_i) + (\sum_{i=d+1}^{p} b_i \mu_i)^2)$$

$$\frac{\partial J}{\partial z_{ni}} = 0 \Leftrightarrow \frac{1}{N}(x_n^T \mu_i - z_{ni}) = 0$$

We get
$z_{ni} = x_n^T \mu_i$

$$\frac{\partial J}{\partial b_i} = 0 \Leftrightarrow \frac{1}{N} \sum_{n=1}^{N} (x_n^T \mu_i - b_i) = 0 \Leftrightarrow \bar{x}^T \mu_i - b_i = 0$$

We get

$b_i = \bar{x}^T \mu_i$

3
1st iteration:

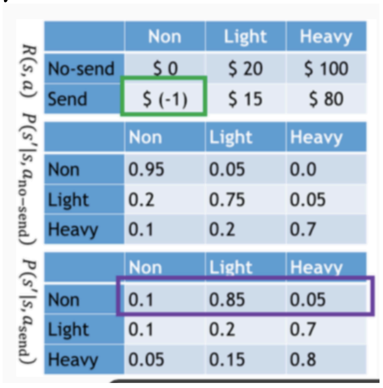
	Non	Light	Heavy
No-send	0,72	45,695	137,71
Send	27,935	52,71	120,68

To get new values we use the following formula:

$Q_k(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_a Q(s',a')$, where

$\max_a Q(s',a') = \{-1,35,50\}$

$\gamma = 0.9$



$R(s,a)$

	Non	Light	Heavy
No-send	\$ 0	\$ 20	\$ 100
Send	$\boxed{\$ (-1)}$	\$ 15	\$ 80

$P(s'|s,a_{no-send})$

	Non	Light	Heavy
Non	0.95	0.05	0.0
Light	0.2	0.75	0.05
Heavy	0.1	0.2	0.7

$P(s'|s,a_{send})$

	Non	Light	Heavy
Non	0.1	0.85	0.05
Light	0.1	0.2	0.7
Heavy	0.05	0.15	0.8

Other values needed are given in these tables.

Take value of non-row in $P(s'|s,a_{no-send})$ table : non-user - no sent = 0 + 0.9*(0.95*(-1) + 0.05*35 + 0*50) = 0.72

Light-row in $P(s'|s,a_{no-send})$ table: light-user - no sent = 20 + 0.9*(0.2*(-1) + 0.75*35 + 0.05*50) = 45.695

Similar for the rest:

heavy-user - no sent = 100 + 0.9*(0.1*(-1) + 0.2*35 + 0.7*50) = 137.71

non-user - sent = -1 + 0.9*(0.1*(-1) + 0.85*35 + 0.05*50) = 27.935

light-user – sent = 15 + 0.9*(0.1*(-1) + 0.2*35 + 0.7*50) = 52.71

heavy-user – sent = 80 + 0.9*(0.05*(-1) + 0.15*35 + 0.8*50) = 120.68

4

4.1

- Following the variational Bayes algorithm of the original VAE, derive the algorithm for this class-conditioned variant (2 points).

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.

4.2 Implementation of algorithm with ZhuSuan:

```
#!/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 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))
    bn.bernoulli("x", x_logits, group_ndims=1)
    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

    # 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
    test_freq = 10
    test_batch_size = 400
    test_iters = x_test.shape[0] // test_batch_size
    result_path = "results/vae"

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

        for epoch in range(1, epochs + 1):
            time_epoch = -time.time()
            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)
            time_epoch += time.time()
            print("Epoch {} ({:.1f}s): Lower bound = {}".format(
                epoch, time_epoch, np.mean(lbs)))

            if epoch % test_freq == 0:
                time_test = -time.time()
                test_lbs, test_lls = [], []
                for t in range(test_iters):
                    test_x_batch = x_test[t * test_batch_size:
                                          (t + 1) * test_batch_size]
                    test_lb = sess.run(lower_bound,
                                       feed_dict={x: test_x_batch,
                                                  n_particles: 1,
                                                  n: test_batch_size})
                    test_ll = sess.run(is_log_likelihood,
                                       feed_dict={x: test_x_batch,
                                                  n_particles: 1000,
                                                  n: test_batch_size})
                    test_lbs.append(test_lb)
                    test_lls.append(test_ll)
                time_test += time.time()
                print(">>> TEST ({:.1f}s)".format(time_test))
                print(">> Test lower bound = {}".format(np.mean(test_lbs)))
                print('>> Test log likelihood (IS) = {}'.format(
                    np.mean(test_lls)))

            if epoch % save_freq == 0:
                images = sess.run(x_gen, feed_dict={n: 100, n_particles: 1})
                name = os.path.join(result_path,
                                    "vae.epoch.{}.png".format(epoch))
                save_image_collections(images, name)


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

Results

```
Epoch 1 (15.8s): Lower bound = -174.3189239501953
Epoch 2 (15.4s): Lower bound = -126.609375
Epoch 3 (15.6s): Lower bound = -115.97903442382812
Epoch 4 (16.1s): Lower bound = -111.89781951904297
Epoch 5 (15.0s): Lower bound = -109.23577117919922
Epoch 6 (15.7s): Lower bound = -107.38616943359375
Epoch 7 (15.6s): Lower bound = -105.96649169921875
Epoch 8 (15.6s): Lower bound = -104.83159637451172
Epoch 9 (16.0s): Lower bound = -103.60923767089844
Epoch 10 (15.6s): Lower bound = -102.87323760986328
>>> TEST (300.2s)
>> Test lower bound = -102.42675018310547
>> Test log likelihood (IS) = -96.82177734375
Epoch 11 (17.2s): Lower bound = -102.16043853759766
Epoch 12 (15.6s): Lower bound = -101.60298919677734
Epoch 13 (14.4s): Lower bound = -101.01287078857422
Epoch 14 (14.9s): Lower bound = -100.64302825927734
Epoch 15 (14.1s): Lower bound = -100.2274169921875
Epoch 16 (15.2s): Lower bound = -99.846923828125
Epoch 17 (15.9s): Lower bound = -99.53782653808594
Epoch 18 (15.9s): Lower bound = -99.3237533569336
Epoch 19 (15.7s): Lower bound = -98.99461364746094
Epoch 20 (16.2s): Lower bound = -98.66244506835938
>>> TEST (296.5s)
>> Test lower bound = -98.92195129394531
>> Test log likelihood (IS) = -93.45943450927734
Epoch 21 (15.9s): Lower bound = -98.47322082519531
Epoch 22 (14.3s): Lower bound = -98.23519134521484
Epoch 23 (13.6s): Lower bound = -98.07992553710938
Epoch 24 (14.0s): Lower bound = -97.86445617675781
Epoch 25 (13.7s): Lower bound = -97.69099426269531
Epoch 26 (14.0s): Lower bound = -97.52287292480469
Epoch 27 (14.3s): Lower bound = -97.38142395019531
Epoch 28 (14.2s): Lower bound = -97.26490783691406
Epoch 29 (14.3s): Lower bound = -97.12347412109375
Epoch 30 (14.4s): Lower bound = -97.02249908447266
>>> TEST (315.8s)
>> Test lower bound = -97.966064453125
>> Test log likelihood (IS) = -92.20950317382812
Epoch 31 (15.4s): Lower bound = -96.90144348144531
Epoch 32 (13.7s): Lower bound = -96.78202056884766
Epoch 33 (13.3s): Lower bound = -96.67034149169922
Epoch 34 (13.6s): Lower bound = -96.50209045410156
Epoch 35 (13.5s): Lower bound = -96.52604675292969
Epoch 36 (14.1s): Lower bound = -96.42350769042969
Epoch 37 (13.3s): Lower bound = -96.34712982177734
Epoch 38 (13.7s): Lower bound = -96.20309448242188
Epoch 39 (13.6s): Lower bound = -96.1180191040039
Epoch 40 (13.8s): Lower bound = -96.0550765991211
>>> TEST (294.0s)
>> Test lower bound = -97.00977325439453
>> Test log likelihood (IS) = -91.47718811035156
Epoch 41 (16.2s): Lower bound = -95.92442321777344
Epoch 42 (16.1s): Lower bound = -95.84114837646484
Epoch 43 (15.8s): Lower bound = -95.8077621459961
Epoch 44 (15.4s): Lower bound = -95.74011993408203
Epoch 45 (15.2s): Lower bound = -95.70277404785156
Epoch 46 (14.9s): Lower bound = -95.64720153808594
Epoch 47 (14.8s): Lower bound = -95.60064697265625
Epoch 48 (14.4s): Lower bound = -95.55496978759766
Epoch 49 (12.9s): Lower bound = -95.41159057617188
Epoch 50 (14.5s): Lower bound = -95.39166259765625
>>> TEST (293.7s)
>> Test lower bound = -96.52333068847656
>> Test log likelihood (IS) = -91.0076675415039
```

4.3