Deep Learning PhD course
HW#3

Muhammad Osama

June 14, 2019

1 Ex1

Assuming $\boldsymbol{x}^{(i)}$ is d dimensional and is either 0 or 1, we use the following data model conditioned on the latent variable $\boldsymbol{z}$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}) = \prod_{k=1}^{d} Bernoulli(\theta_k) \tag{1}$$

where $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_d\}$. For $MNIST$, $d = 784$. Hence given $\boldsymbol{z}$, we take every pixel to be a Bernoulli random variable. Ideally, the pixel values lie in the interval $[0, 1]$. However, for simplicity we use the above approximation.

2 Ex2

The reason we cannot work with eq. (5) directly is because it is difficult to evaluate the expectation $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big]$ analytically.

3 Ex 3

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \;=\; \mathcal{N}(\boldsymbol{\mu}^{(i)}, diag(\boldsymbol{\sigma}^{2(i)})) \qquad p(\boldsymbol{z}) \;=\; \mathcal{N}(0, \mathbf{I})$$

Let $\Sigma = diag(\boldsymbol{\sigma}^{2(i)})$, hence the KL divergence is

$$\begin{aligned}\Delta_{KL}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p(\boldsymbol{z})\big) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\cdot)}\big[-\frac{(\boldsymbol{z}-\boldsymbol{\mu}^{(i)})^\top \Sigma^{-1}(\boldsymbol{z}-\boldsymbol{\mu}^{(i)})}{2}\big]+ \\ &\mathbb{E}_{q_{\boldsymbol{\phi}}(\cdot)}\big[\frac{\boldsymbol{z}^\top \boldsymbol{z}}{2}\big] - \frac{1}{2}\sum_{j=1}^{K} \log \sigma_j^{2(i)},\end{aligned} \tag{2}$$

where

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\cdot)}\big[-\frac{(\boldsymbol{z}-\boldsymbol{\mu}^{(i)})^\top\Sigma^{-1}(\boldsymbol{z}-\boldsymbol{\mu}^{(i)})}{2}\big] = \frac{1}{2}\sum_{j=1}^{K}(1),$$

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\cdot)}\big[\frac{\boldsymbol{z}^\top\boldsymbol{z}}{2}\big] = \sum_{j=1}^{K}\sigma_j^{2(i)} + \mu_j^{2(i)}.$$

Hence,

$$\Delta_{KL}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p(\boldsymbol{z})\big) = -\frac{1}{2}\big[\sum_{j=1}^{K}1 + \log\sigma_j^{2(i)} - \sigma_j^{2(i)} - \mu_j^{2(i)}\big] \tag{3}$$

Moreover,

$$g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(i,l)}, \boldsymbol{x}^{(i)}) \ = \ \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(i,l)}$$

where

$$\boldsymbol{\epsilon}^{(i,l)} \sim \mathcal{N}(0, \mathbf{I})$$

4 Ex 4

The model summaries are given in figures 1 and 5 for latent dimensions of $k = 3$ and $k = 20$ respectively. For sampling from the generative network, a k dimensional vector $\boldsymbol{z}_s$ was picked randomly in space $\mathbb{R}^k$. I looked at the mean and variance of the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ for the test data to decide on what would be reasonable values for the elements of $\boldsymbol{z}_s$. The 784 probability parameters of the Bernoulli distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ corresponding to this latent variable were obtained by using $\boldsymbol{p}_B = \textit{decoder.predict}(\boldsymbol{z}_s)$. Then we simply independently sampled 784 random variables with these probabilities from the Bernoulli distribution i.e. the sample is given by $\boldsymbol{x}_s = \textit{Bernoulli}(\boldsymbol{p}_B)$.

4.1 Number of latent variables K=3

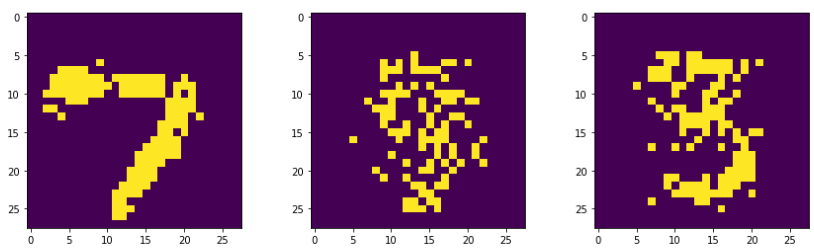
(a) Loss on test data: $KL\ loss = 7.89$, Reconstruction error $= 131.80$
(b) Some samples $\boldsymbol{x}_s$ from the generative network for $k = 2$ are shown in figure 2 (c) Some original test examples and their reconstructions are shown in figures 3 and 4.

4.2 Number of Latent variables $K = 20$

(a) Loss on test data: $KL\ loss = 25.23$, Reconstruction error $= 78.33$
(b) Some samples $\boldsymbol{x}_s$ from the generative network for $k = 20$ are shown in figure 6. (c) Some original test examples and their reconstructions are shown in figures 7 and 8.
We observe that as the dimension of the latent variable increases from $k = 3$

```
In [553]: vae.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 784)               0
_________________________________________________________________
encoder (Model)              [(None, 2), (None, 2), (N 403972
_________________________________________________________________
decoder (Model)              (None, 784)               403728
=================================================================
Total params: 807,700
Trainable params: 807,700
Non-trainable params: 0
_________________________________________________________________
```

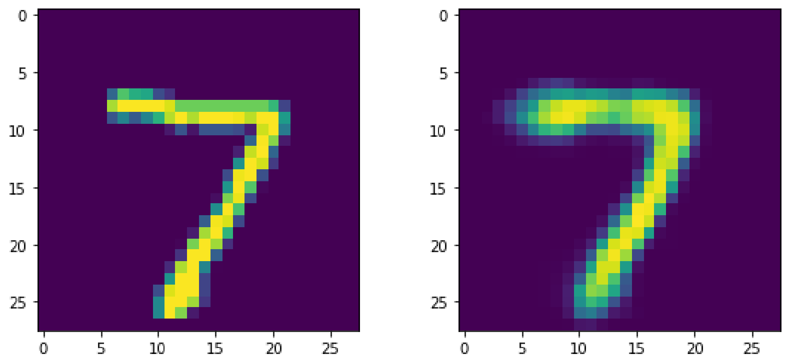
Figure 1



(a) (b) (c)

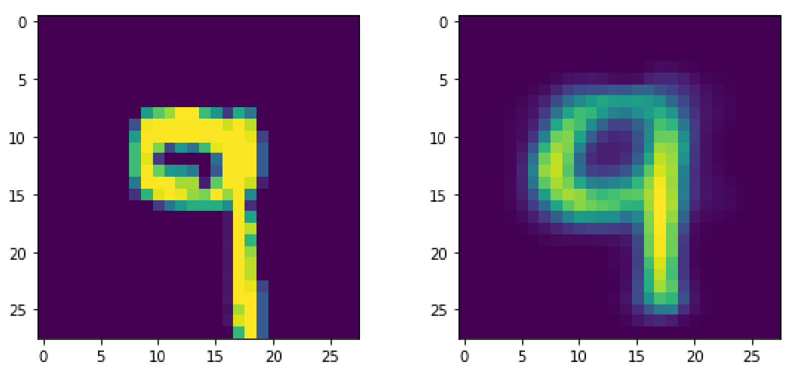
Figure 2: Samples $\boldsymbol{x}_s$ for latent variable (a) $\boldsymbol{z}_s = [-2, 4]$ (b) $\boldsymbol{z}_s = [-0.2, 0.4]$ (c) $\boldsymbol{z}_s = [0.1, 0.1]$

to $k = 20$, the overall loss decreases. It is interesting to see that the KL loss actually increases with dimension so that the posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is less close to the prior $p(\boldsymbol{z})$. However, the reconstruction error decreases with dimension.



(a) (b)

Figure 3: (a) Original (b) Reconstructed

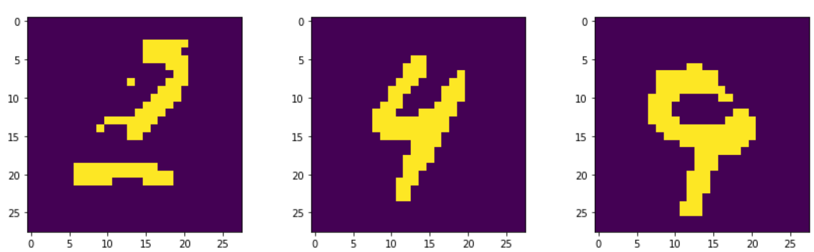


(a) (b)

Figure 4: (a) Original (b) Reconstructed

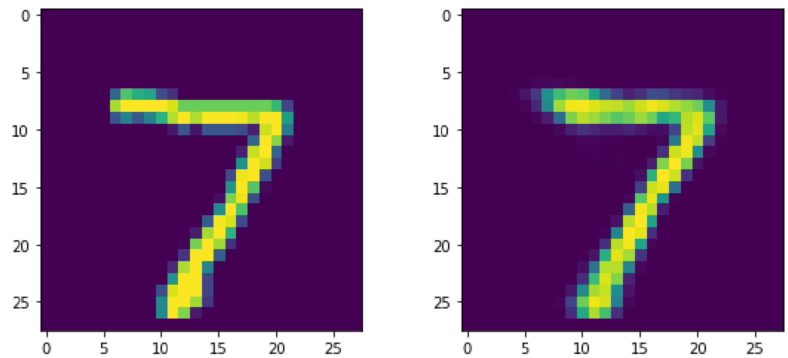
```
In [80]: vae.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 784)               0
_________________________________________________________________
encoder (Model)              [(None, 20), (None, 20),  422440
_________________________________________________________________
decoder (Model)              (None, 784)               412944
=================================================================
Total params: 835,384
Trainable params: 835,384
Non-trainable params: 0
_________________________________________________________________
```

Figure 5



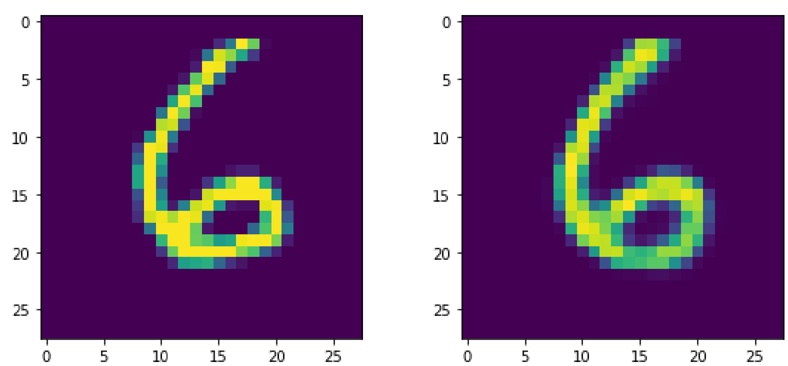
(a) (b) (c)

Figure 6: Samples $\boldsymbol{x}_s$



(a) (b)

Figure 7: (a) Original (b) Reconstructed



(a) (b)

Figure 8: (a) Original (b) Reconstructed

Appendix

```
from keras import backend as K
from keras.losses import
    binary_crossentropy
from keras.datasets import mnist
from keras.layers import Lambda, Input,
    Dense
from keras.models import Model
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

#sampling function-reparametrization
    trick
def sampling(args):

z_mean, z_log_var = args
z_std = K.sqrt(K.exp(z_log_var))
batch = K.shape(z_mean)[0]
dim = K.int_shape(z_mean)[1]
epsl = K.random_normal(shape=(batch,dim))
return z_mean + z_std*epsl

#define losses
def reg_loss(args):

z_mean, z_log_var = args
#loss per datapoint
kl_loss = 1 + z_log_var - K.exp(z_log_var
    ) - K.square(z_mean)
kl_loss = K.sum(kl_loss, axis = -1)
kl_loss *= -0.5;
#average over all datapoints
kl_loss = K.mean(kl_loss)

return kl_loss

def recn_loss(varargs):

inputs, outputs = varargs
#loss per data point assuming nos. of MC
    samples r=1
```

```
#recons_loss = K.sum(inputs * K.log(
    outputs) + (1-inputs)*K.log(1-outputs)
    , axis=-1)
recons_loss = binary_crossentropy(inputs,
     outputs)
recons_loss*= original_dim
#average over all data points
recons_loss  = K.mean(recons_loss)

return recons_loss

# MNIST dataset
(x_train, y_train), (x_test, y_test) =
    mnist.load_data()
image_size = x_train.shape[1]
original_dim = image_size * image_size
x_train = np.reshape(x_train, [-1,
    original_dim])
x_test = np.reshape(x_test, [-1,
    original_dim])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# network parameters
input_shape = (original_dim, )
intermediate_dim = 512
batch_size = 128
latent_dim = 3
epochs = 20

# build encoder model
inputs = Input(shape=input_shape, name='
    encoder_input')
x = Dense(intermediate_dim, activation='
    relu')(inputs)
z_mean = Dense(latent_dim, name='z_mean')
    (x)
z_log_var = Dense(latent_dim, name='
    z_log_var')(x)
z = Lambda(sampling, output_shape=(
    latent_dim,), name='z')([z_mean,
    z_log_var])
encoder = Model(inputs, [z_mean,
    z_log_var, z], name='encoder')

# build decoder model
```

```
latent_inputs = Input(shape=(latent_dim,)
    , name='z_sampling')
x = Dense(intermediate_dim, activation='
    relu')(latent_inputs)
outputs = Dense(original_dim, activation
    ='sigmoid')(x)
decoder = Model(latent_inputs, outputs,
    name='decoder')

outputs = decoder(encoder(inputs)[2])

vae = Model(inputs, outputs, name='vae')
vae_loss = reg_loss([z_mean, z_log_var])
    + recn_loss([inputs, outputs])
vae.add_loss(vae_loss)
vae.compile(optimizer='adam', metrics =
    ['reg_loss','recn_loss'])

#fit
history = vae.fit(x_train,epochs=epochs,
    batch_size=batch_size, validation_data
    =(x_test, None))

#predict
ztest_mean, ztest_log_var, ztest =
    encoder.predict(x_test)
x_pred = decoder.predict(ztest)

#Ex4.1
x_pred = tf.convert_to_tensor(x_pred, np.
    float32)
kl_loss = reg_loss([ztest_mean,
    ztest_log_var])
recons_loss = recn_loss([x_test, x_pred])
with tf.Session() as sess:
kl_loss = kl_loss.eval()

with tf.Session() as sess:
recons_loss = recons_loss.eval()

print(kl_loss)
print(recons_loss)


#Ex4.2
def sample_generative_network(laten_var):
```

```
#probabilities of bernoulli distribution
prb = decoder.predict(laten_var)
#sample from bernoulli
xhat = np.random.binomial(1,prb)
xhat = np.reshape(xhat,(image_size,
    image_size))
plt.imshow(xhat)

#ex4.3
x_pred = decoder.predict(ztest);
idx = 700
#original image
plt.imshow(np.reshape(x_test[idx,:],(
    image_size,image_size)))
plt.show()
#reconstructed image
plt.show()
plt.imshow(np.reshape(x_pred[idx,:],(
    image_size,image_size)))
```