```python
try:
    import google.colab
    IN_COLAB = True
except:
    IN_COLAB = False
if IN_COLAB:
    print("Downloading Colab files")
    ! shred -u setup_google_colab.py
    ! wget https://raw.githubusercontent.com/hse-aml/bayesian-methods-for-ml/master/setup_google_colab.py
    import setup_google_colab
    setup_google_colab.load_data_week5()
```

```
Downloading Colab files
shred: setup_google_colab.py: failed to open for writing: No such file or directory
--2019-06-12 09:06:23--  https://raw.githubusercontent.com/hse-aml/bayesian-methods-for-ml/master
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... connect
HTTP request sent, awaiting response... 200 OK
Length: 1308 (1.3K) [text/plain]
Saving to: ‘setup_google_colab.py’

setup_google_colab. 100%[===================>]   1.28K  --.-KB/s    in 0s

2019-06-12 09:06:23 (215 MB/s) - ‘setup_google_colab.py’ saved [1308/1308]
```

```python
import tensorflow as tf
import keras
import numpy as np
import matplotlib.pyplot as plt

from keras.layers import Input, Dense, Lambda, InputLayer, concatenate
from keras.models import Model, Sequential
from keras import backend as K
from keras import metrics
from keras.datasets import mnist
from keras.utils import np_utils
from w5_grader import VAEGrader
```

```
Using TensorFlow backend.
```

```python
def vlb_binomial(x, x_decoded_mean, t_mean, t_log_var):
    """Returns the value of negative Variational Lower Bound

    The inputs are tf.Tensor
        x: (batch_size x number_of_pixels) matrix with one image per row with zeros and ones
        x_decoded_mean: (batch_size x number_of_pixels) mean of the distribution p(x | t), real numbers f
        t_mean: (batch_size x latent_dim) mean vector of the (normal) distribution q(t | x)
        t_log_var: (batch_size x latent_dim) logarithm of the variance vector of the (normal) distributio

    Returns:
        A tf.Tensor with one element (averaged across the batch), VLB
    """
    print(tf.shape(x))
    print(x.get_shape().as_list())
    print(x_decoded_mean.get_shape().as_list())
    print(t_mean.get_shape().as_list())
    print(t_log_var.get_shape().as_list())

    # Bernoulli: p^k *(1-p)^(1-k) for k in {0, 1}
    # k: ground truth
    # p: decoded value
    xent_loss = K.sum(x * K.log(x_decoded_mean + 1e-10) + (1 - x) * K.log(1 - x_decoded_mean + 1e-10), ax
    print(xent_loss.get_shape().as_list())

    kl_loss = 0.5 * K.sum((1 + t_log_var - K.square(t_mean) - K.exp(t_log_var)), axis=1)
    print(kl_loss.get_shape().as_list())

    # average across minuibatch
    vae_loss = - K.mean(xent_loss + kl_loss)
    return vae_loss
```

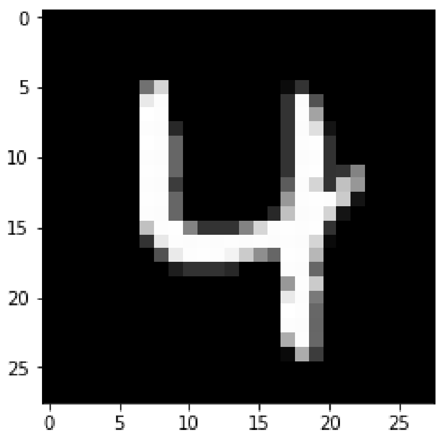
```
# Start tf session so we can run code.
sess = tf.InteractiveSession()
# Connect keras to the created session.
K.set_session(sess)


batch_size = 100
original_dim = 784 # Number of pixels in MNIST images.
latent_dim = 10 #3 # d, dimensionality of the latent code t.
intermediate_dim = 256 # Size of the hidden layer.
epochs = 3


x = Input(batch_shape=(batch_size, original_dim))
x1 = Input(batch_shape=(1, original_dim))
def create_encoder(input_dim):
    # Encoder network.
    # We instantiate these layers separately so as to reuse them later
    encoder = Sequential(name='encoder')
    encoder.add(InputLayer([input_dim]))
    encoder.add(Dense(intermediate_dim, activation='relu'))
    encoder.add(Dense(2 * latent_dim))
    return encoder
encoder = create_encoder(original_dim)

get_t_mean = Lambda(lambda h: h[:, :latent_dim])
get_t_log_var = Lambda(lambda h: h[:, latent_dim:])
h = encoder(x)
t_mean = get_t_mean(h)
t_log_var = get_t_log_var(h)

h1 = encoder(x1)
t1_mean = get_t_mean(h1)
t1_log_var = get_t_log_var(h1)

# Sampling from the distribution
#     q(t | x) = N(t_mean, exp(t_log_var))
# with reparametrization trick.
def sampling(args):
    """Returns sample from a distribution N(args[0], diag(args[1]))

    The sample should be computed with reparametrization trick.

    The inputs are tf.Tensor
        args[0]: (batch_size x latent_dim) mean of the desired distribution
        args[1]: (batch_size x latent_dim) logarithm of the variance vector of the desired distribution

    Returns:
        A tf.Tensor of size (batch_size x latent_dim), the samples.
    """
    t_mean, t_log_var = args
    epsilon = K.random_normal(shape=K.shape(t_mean), mean=0.0, stddev=1.0)
    return t_mean + K.exp(0.5 * t_log_var) * epsilon


t = Lambda(sampling)([t_mean, t_log_var])
t1 = Lambda(sampling)([t1_mean, t1_log_var])


def create_decoder(input_dim):
    # Decoder network
    # We instantiate these layers separately so as to reuse them later
    decoder = Sequential(name='decoder')
    decoder.add(InputLayer([input_dim]))
    decoder.add(Dense(intermediate_dim, activation='relu'))
    decoder.add(Dense(original_dim, activation='sigmoid'))
    return decoder
decoder = create_decoder(latent_dim)
x_decoded_mean = decoder(t)
x1_decoded_mean = decoder(t1)


loss = vlb_binomial(x, x_decoded_mean, t_mean, t_log_var)
vae = Model(x, x_decoded_mean)
# Keras will provide input (x) and output (x_decoded_mean) to the function that
# should construct loss, but since our function also depends on other
# things (e.g. t_means), it is easier to build the loss in advance and pass
# a function that always returns it.
vae.compile(optimizer=keras.optimizers.RMSprop(lr=0.001), loss=lambda x, y: loss)
```

```
Tensor("Shape_6:0", shape=(2,), dtype=int32)
[100, 784]
[100, 784]
[100, 10]
[100, 10]
[100]
[100]
```

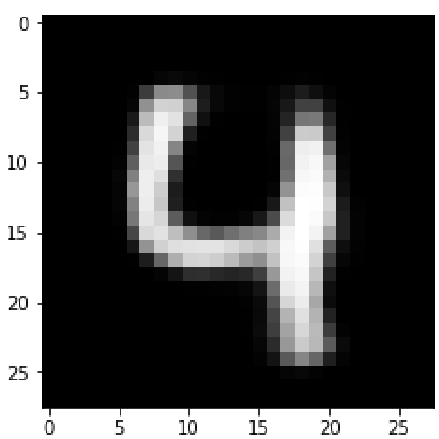
```
# train the VAE on MNIST digits
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# One hot encoding.
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))


epochs = 20
hist = vae.fit(x=x_train, y=x_train,
               shuffle=True,
               epochs=epochs,
               batch_size=batch_size,
               validation_data=(x_test, x_test),
               verbose=2)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

```python
img_flat = x_test[49]
img = img_flat.reshape((28,28))
plt.imshow(img,  cmap='gray')
plt.show()

enc, enc_mean, enc_log_var = sess.run([h1, t1_mean, t1_log_var], feed_dict={x1: [img_flat]})
print(enc)
print(enc_mean)
print(enc_log_var)
print(h1)

dec = sess.run(x1_decoded_mean, feed_dict={t1: enc_mean})
res_img = dec.reshape((28,28))
plt.imshow(res_img,  cmap='gray')
plt.show()
```

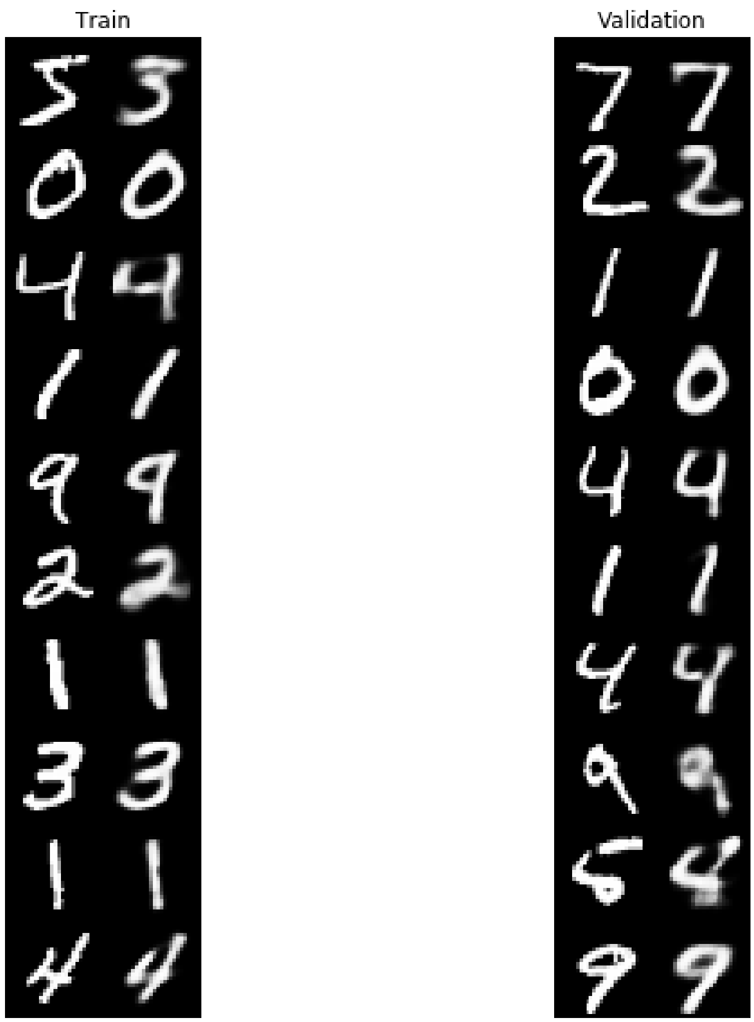



```
[[-0.28441328 -1.3568563   0.5647244  -0.77163976 -0.9561359   2.3299193
   0.4285398  -1.5410898  -1.6350224   0.7509855  -3.110401   -3.1845288
  -3.6625056  -3.6305943  -2.5657816  -3.3691223  -3.0476518  -3.2734396
  -3.321457   -3.748587  ]]
[[-0.28441328 -1.3568563   0.5647244  -0.77163976 -0.9561359   2.3299193
   0.4285398  -1.5410898  -1.6350224   0.7509855 ]]
[[-3.110401  -3.1845288 -3.6625056 -3.6305943 -2.5657816 -3.3691223
  -3.0476518 -3.2734396 -3.321457  -3.748587 ]]
Tensor("encoder_8/dense_18/BiasAdd:0", shape=(1, 20), dtype=float32)
```




```python
fig = plt.figure(figsize=(10, 10))
for fid_idx, (data, title) in enumerate(
            zip([x_train, x_test], ['Train', 'Validation'])):
    n = 10  # figure with 10 x 2 digits
    digit_size = 28
    figure = np.zeros((digit_size * n, digit_size * 2))
    decoded = sess.run(x_decoded_mean, feed_dict={x: data[:batch_size, :]})
    for i in range(10):
        figure[i * digit_size: (i + 1) * digit_size,
               :digit_size] = data[i, :].reshape(digit_size, digit_size)
```

```python
        figure[i * digit_size: (i + 1) * digit_size,
               digit_size:] = decoded[i, :].reshape(digit_size, digit_size)
    ax = fig.add_subplot(1, 2, fid_idx + 1)
    ax.imshow(figure, cmap='Greys_r')
    ax.set_title(title)
    ax.axis('off')
plt.show()
```




```python
n_samples = 10  # To pass automatic grading please use at least 2 samples here.
# sampled_im_mean is a tf.Tensor of size 10 x 784 with 10 random
# images sampled from the vae model.
sample_t = K.random_normal(shape=(n_samples, latent_dim), mean=0.0, stddev=1.0)
sampled_im_mean = decoder(sample_t)
```

```python
sampled_im_mean_np = sess.run(sampled_im_mean)
# Show the sampled images.
plt.figure()
for i in range(n_samples):
    ax = plt.subplot(n_samples // 5 + 1, 5, i + 1)
    plt.imshow(sampled_im_mean_np[i, :].reshape(28, 28), cmap='gray')
    ax.axis('off')
plt.show()
```

```python
# One-hot labels placeholder.
x = Input(batch_shape=(batch_size, original_dim))
label = Input(batch_shape=(batch_size, 10))

xl = concatenate([x, label])

encoder = create_encoder(original_dim + 10)

h = encoder(xl)
cond_t_mean = get_t_mean(h)
cond_t_log_var = get_t_log_var(h)

print(xl.get_shape().as_list())
print(h.get_shape().as_list())
print(cond_t_mean.get_shape().as_list())
print(cond_t_log_var.get_shape().as_list())


t = Lambda(sampling)([cond_t_mean, cond_t_log_var])
tl = concatenate([t, label])

decoder = create_decoder(latent_dim + 10)
cond_x_decoded_mean = decoder(tl)

#cond_t_mean =  # Mean of the latent code (without label) for cvae model.
#cond_t_log_var = # Logarithm of the variance of the latent code (without label) for cvae model.
#cond_x_decoded_mean =  # Final output of the cvae model.
```

```
[100, 794]
[100, 20]
[100, 10]
[100, 10]
```

```python
conditional_loss = vlb_binomial(x, cond_x_decoded_mean, cond_t_mean, cond_t_log_var)
cvae = Model([x, label], cond_x_decoded_mean)
cvae.compile(optimizer=keras.optimizers.RMSprop(lr=0.001), loss=lambda x, y: conditional_loss)
```

```
Tensor("Shape_7:0", shape=(2,), dtype=int32)
[100, 784]
[100, 784]
[100, 10]
[100, 10]
[100]
[100]
```

```python
hist = cvae.fit(x=[x_train, y_train],
                y=x_train,
                shuffle=True,
                epochs=epochs,
                batch_size=batch_size,
                validation_data=([x_test, y_test], x_test),
                verbose=2)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
 - 2s - loss: 155.4058 - val_loss: 129.0842
Epoch 2/20
 - 2s - loss: 124.1803 - val_loss: 120.4274
Epoch 3/20
 - 2s - loss: 118.4696 - val_loss: 116.5869
Epoch 4/20
 - 2s - loss: 115.4052 - val_loss: 113.4894
Epoch 5/20
 - 2s - loss: 113.3610 - val_loss: 111.9370
Epoch 6/20
 - 2s - loss: 111.9178 - val_loss: 110.4100
Epoch 7/20
 - 2s - loss: 110.7301 - val_loss: 109.6823
Epoch 8/20
 - 2s - loss: 109.8311 - val_loss: 108.2383
Epoch 9/20
 - 2s - loss: 109.0987 - val_loss: 107.7660
Epoch 10/20
 - 2s - loss: 108.4247 - val_loss: 108.0507
Epoch 11/20
 - 2s - loss: 107.8561 - val_loss: 106.8372
Epoch 12/20
 - 2s - loss: 107.3355 - val_loss: 106.8308
Epoch 13/20
 - 2s - loss: 106.8826 - val_loss: 105.6550
```

```python
fig = plt.figure(figsize=(10, 10))
for fid_idx, (x_data, y_data, title) in enumerate(
            zip([x_train, x_test], [y_train, y_test], ['Train', 'Validation'])):
    n = 10  # figure with 10 x 2 digits
    digit_size = 28
    figure = np.zeros((digit_size * n, digit_size * 2))
    decoded = sess.run(cond_x_decoded_mean,
                       feed_dict={x: x_data[:batch_size, :],
                                  label: y_data[:batch_size, :]})
    for i in range(10):
        figure[i * digit_size: (i + 1) * digit_size,
               :digit_size] = x_data[i, :].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) * digit_size,
               digit_size:] = decoded[i, :].reshape(digit_size, digit_size)
    ax = fig.add_subplot(1, 2, fid_idx + 1)
    ax.imshow(figure, cmap='Greys_r')
    ax.set_title(title)
    ax.axis('off')
plt.show()
```


```
# Prepare one hot labels of form
#   0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 ...
# to sample five zeros, five ones, etc
curr_labels = np.eye(10)
curr_labels = np.repeat(curr_labels, 5, axis=0)  # Its shape is 50 x 10.
# cond_sampled_im_mean is a tf.Tensor of size 50 x 784 with 5 random zeros,
# then 5 random ones, etc sampled from the cvae model.
sample_t = K.random_normal(shape=(50, latent_dim), mean=0.0, stddev=0.5)
sample_tl = concatenate([sample_t, tf.convert_to_tensor(curr_labels, dtype=tf.float32)])
cond_sampled_im_mean = decoder(sample_tl)
```

```
cond_sampled_im_mean_np = sess.run(cond_sampled_im_mean)
# Show the sampled images.
plt.figure(figsize=(10, 10))
global_idx = 0
for digit in range(10):
    for _ in range(5):
        ax = plt.subplot(10, 5, global_idx + 1)
        plt.imshow(cond_sampled_im_mean_np[global_idx, :].reshape(28, 28), cmap='gray')
        ax.axis('off')
        global_idx += 1
plt.show()
```

```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 6s 0us/step
```

```
plt.imshow(x_train[7, :])
plt.show()
```