VAE

Variational autoencoder [1] models inherit autoencoder architecture, but use variational approach homework, we will implement VAE and quantitatively measure the quality of the generated sampl

[1] Auto-Encoding Variational Bayes, Diederik P Kingma, Max Welling 2013 https://arxiv.org/abs/1

[2] Improved techniques for training gans, Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Ra Neural Information Processing Systems

[3] A note on inception score, Shane Barratt, Rishi Sharma 2018 https://arxiv.org/abs/1801.01973

PART I. Train a good VAE model

Setup

```
import tensorflow as tf
if tf.__version__ < '2.0.0':
    tf.enable_eager_execution()
tf.executing_eagerly()

import numpy as np
import os

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# A bunch of utility functions

def show_images(images):
    # images reshape to (batch_size, D)
    images = np.reshape(images, [images.shape[0], -1])
    sqrtn = int(np.ceil(np.sqrt(images.shape[0])))
    sqrtimg = int(np.ceil(np.sqrt(images.shape[1])))

    fig = plt.figure(figsize=(sqrtn, sqrtn))
    gs = gridspec.GridSpec(sqrtn, sqrtn)
    gs.update(wspace=0.05, hspace=0.05)

    for i, img in enumerate(images):
        ax = plt.subplot(gs[i])
        plt.axis('off')
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        ax.set_aspect('equal')
        plt.imshow(img.reshape([sqrtimg,sqrtimg]))
    return

def preprocess_img(x):
    return 2 * x - 1.0

def rel_error(x,y):
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))

def count_params(model):
    """Count the number of parameters in the current TensorFlow graph """
    param_count = np.sum([np.prod(p.shape) for p in model.weights])
    return param_count
```

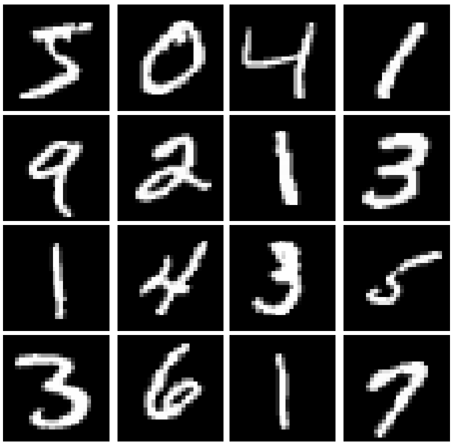
Dataset

We will be working on the MNIST dataset, which is 60,000 training and 10,000 test images. Each p digit on black background (0 through 9). This was one of the first datasets used to train convoluti standard CNN model can easily exceed 99% accuracy.

Heads-up: Our MNIST wrapper returns images as vectors. That is, they're size (batch, 784). If you resize them to (batch,28,28) or (batch,28,28,1). They are also type np.float32 and bounded [0,1].

```
class MNIST(object):
    def __init__(self, batch_size, shuffle=False):
        """
        Construct an iterator object over the MNIST data

        Inputs:
        - batch_size: Integer giving number of elements per minibatch
        - shuffle: (optional) Boolean, whether to shuffle the data on each epoch
        """
        train, _ = tf.keras.datasets.mnist.load_data()
        X, y = train
        X = X.astype(np.float32)/255
        X = X.reshape((X.shape[0], -1))
        self.X, self.y = X, y
        self.batch_size, self.shuffle = batch_size, shuffle

    def __iter__(self):
        N, B = self.X.shape[0], self.batch_size
        idxs = np.arange(N)
        if self.shuffle:
            np.random.shuffle(idxs)
        return iter((self.X[i:i+B], self.y[i:i+B]) for i in range(0, N, B))

# show a batch
mnist = MNIST(batch_size=16)
show_images(mnist.X[:16])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```



```
X_DIM = mnist.X[0].size
num_samples = 100000
num_to_show = 100

# Hyperparamters. Your job to find these.
# TODO:
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
num_epochs = 50
batch_size = 50
Z_DIM = 5
learning_rate = 2e-4
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

Encoder

Our first step is to build a variational encoder network $q_\phi(z \mid x)$.

Hint: You should use the layers in `tf.keras.layers` to build the model. Use four FC layers. All fully
For initialization, just use the default initializer used by the `tf.keras.layers` functions.

The output of the encoder should thus have shape `[batch_size, 2*z_dim]`, and contain real numbe
diagonal log variance $\log \sigma(x_i)^2$ of each of the `batch_size` input images. Note, we want to make
stability.

WARNING: Do not apply any non-linearity to the last activation.

```
def q_phi(z_dim=Z_DIM, x_dim=X_DIM):
  model = tf.keras.models.Sequential([
    # TODO: implement architecture
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    tf.keras.layers.Dense(392, activation="relu", use_bias=True, input_shape=(x_dim,)),
    tf.keras.layers.Dense(196, activation="relu", use_bias=True),
    tf.keras.layers.Dense(128, activation="tanh", use_bias=True),
    tf.keras.layers.Dense(2 * z_dim,  use_bias=True)
    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
  ])
  return model
```

```
# TODO: implement reparameterization trick
def sample_z(mu, log_var):
  # Your code here for the reparameterization trick.
  # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
  samples = None
  #print(mu.shape)
  z = tf.random.normal(tf.shape(mu))
  s = tf.math.exp(0.5 * log_var)
  samples = mu + s * z
  return samples
  # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

Decoder

Now to build a decoder network $p_\theta(x \mid z)$. You should use the layers in `tf.keras.layers` to constr
connected layers should include bias terms. Note that you can use the tf.nn module to access act
initializers for parameters.

In this exercise, we will use Bernoulli MLP decoder where $p_\theta(x \mid z)$ is modeled with multivariate B
Gaussian distribution we discussed in the lecture, as following (see Appendix C.1 in the original pa

$\log p(x \mid z) = \sum_{i=1} x_i \log z_i + (1 - x_i) \log(1 - z_i)$

Note, the output of the decoder should have shape `[batch_size, x_dim]` and should output the unr

WARNING: Do not apply any non-linearity to the last activation.

```
def p_theta(z_dim=Z_DIM, x_dim=X_DIM):
  model = tf.keras.models.Sequential([
    # TODO: implement architecture
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    tf.keras.layers.Dense(128, activation="tanh", use_bias=True, input_shape=(z_dim,)),
    tf.keras.layers.Dense(196, activation="relu", use_bias=True),
    tf.keras.layers.Dense(392, activation="relu", use_bias=True),
    tf.keras.layers.Dense(x_dim, use_bias=True)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
  ])
  return model
```

Loss definition

Compute the VAE loss.

1. For the reconstruction loss, you might find `tf.nn.sigmoid_cross_entropy_with_logits` or `tf.ker`
2. For the kl loss, we discussed the closed form kl divergence between two gaussians in the le

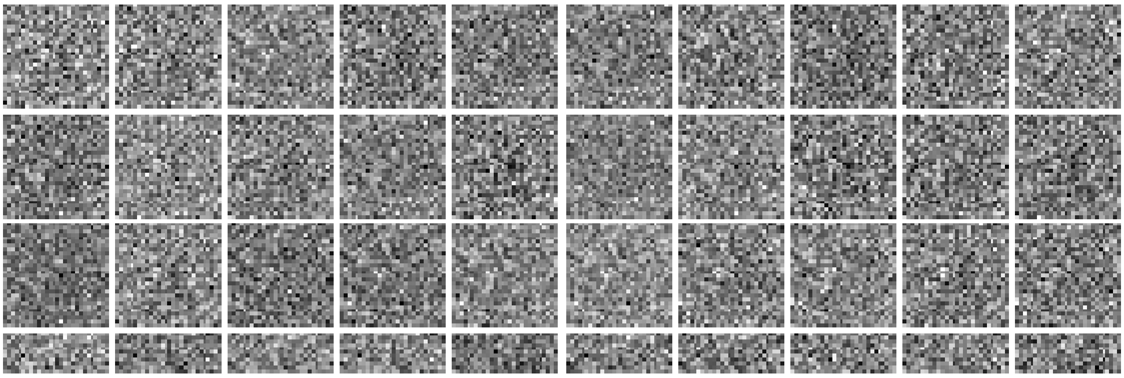
```
def vae_loss(x, x_logit, z_mu, z_logvar):
  recon_loss = None
  kl_loss = None # negative value
  # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
  #print(x, x_logit)
  #bce = tf.keras.losses.BinaryCrossentropy(from_logits=True, reduction=tf.keras.losses.Reduction.N
  recon_loss = tf.reduce_sum(tf.nn.sigmoid_cross_entropy_with_logits(x, x_logit), axis=1)
  #entropy = bce(x, x_logit)
  #recon_loss = tf.reduce_sum(entropy)
  #print(entropy.shape, recon_loss.shape)
  temp =  1 + z_logvar - tf.square(z_mu) - tf.math.exp(z_logvar)
  kl_loss = -0.5 * (tf.reduce_sum(temp, axis = 1))
  #print(kl_loss, z_mu, z_logvar)
  #print(recon_loss, kl_loss)
  # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
  #print(kl_loss)
  vae_loss = tf.reduce_mean(recon_loss + kl_loss)
  return vae_loss, tf.reduce_mean(recon_loss)
```

Optimizing our loss

```
Q = q_phi()
P = p_theta()
solver = tf.keras.optimizers.Adam(learning_rate)
mnist = MNIST(batch_size=batch_size, shuffle=True)
```

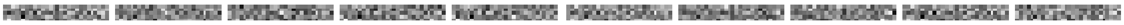
Visualize generated samples before training

```
z_gen = tf.random.normal(shape=[num_to_show, Z_DIM])
x_gen = P(z_gen)
imgs_numpy = tf.nn.sigmoid(x_gen).numpy()
show_images(imgs_numpy)
plt.show()
```



Training a VAE!

If everything works, your batch average reconstruction loss should drop below 95.

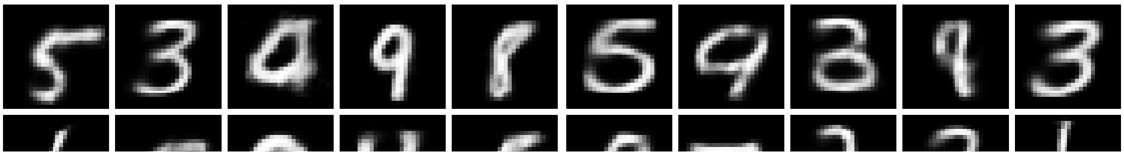


```
iter_count = 0
show_every = 400
for epoch in range(num_epochs):
  for (x_i, _) in mnist:
    with tf.GradientTape() as tape:
      z_concat = Q(preprocess_img(x_i))
      z_mu, z_logvar = tf.split(z_concat, num_or_size_splits=2, axis=1)
      z_i = sample_z(z_mu, z_logvar)

      x_logit = P(z_i)
      loss, recon_loss = vae_loss(x_i, x_logit, z_mu, z_logvar)

      grads = tape.gradient(loss,
                [Q.trainable_variables, P.trainable_variables])

      solver.apply_gradients(zip([*grads[0],*grads[1]],
                [*Q.trainable_variables, *P.trainable_variables]))

      if (iter_count % show_every == 0):
        print('Epoch: {}, Iter: {}, Loss: {:.4}, Recon: {:.4}'.format(
            epoch, iter_count, loss, recon_loss))
        #imgs_numpy = tf.nn.sigmoid(x_logit).numpy()
        #show_images(imgs_numpy[0:16])
        #plt.show()
      iter_count += 1
```

```
Epoch: 31, Iter: 38000, Loss: 113.1, Recon: 100.2
Epoch: 32, Iter: 38400, Loss: 104.9, Recon: 92.39
Epoch: 32, Iter: 38800, Loss: 111.9, Recon: 98.44
Epoch: 32, Iter: 39200, Loss: 112.0, Recon: 99.23
Epoch: 33, Iter: 39600, Loss: 104.7, Recon: 91.81
Epoch: 33, Iter: 40000, Loss: 111.4, Recon: 97.96
Epoch: 33, Iter: 40400, Loss: 112.5, Recon: 99.61
Epoch: 34, Iter: 40800, Loss: 104.3, Recon: 91.34
Epoch: 34, Iter: 41200, Loss: 111.9, Recon: 98.52
Epoch: 34, Iter: 41600, Loss: 112.2, Recon: 99.32
Epoch: 35, Iter: 42000, Loss: 104.0, Recon: 91.23
Epoch: 35, Iter: 42400, Loss: 111.9, Recon: 98.36
Epoch: 35, Iter: 42800, Loss: 112.7, Recon: 99.86
Epoch: 36, Iter: 43200, Loss: 104.2, Recon: 91.33
Epoch: 36, Iter: 43600, Loss: 111.7, Recon: 98.21
Epoch: 36, Iter: 44000, Loss: 111.8, Recon: 99.0
Epoch: 37, Iter: 44400, Loss: 105.8, Recon: 93.12
Epoch: 37, Iter: 44800, Loss: 112.3, Recon: 99.06
Epoch: 37, Iter: 45200, Loss: 111.6, Recon: 98.83
Epoch: 38, Iter: 45600, Loss: 104.0, Recon: 91.21
Epoch: 38, Iter: 46000, Loss: 111.5, Recon: 98.06
Epoch: 38, Iter: 46400, Loss: 111.9, Recon: 98.88
Epoch: 39, Iter: 46800, Loss: 104.8, Recon: 92.02
Epoch: 39, Iter: 47200, Loss: 110.9, Recon: 97.78
Epoch: 39, Iter: 47600, Loss: 112.4, Recon: 99.66
Epoch: 40, Iter: 48000, Loss: 104.7, Recon: 91.87
Epoch: 40, Iter: 48400, Loss: 111.7, Recon: 98.27
Epoch: 40, Iter: 48800, Loss: 112.3, Recon: 99.5
Epoch: 41, Iter: 49200, Loss: 104.6, Recon: 91.76
Epoch: 41, Iter: 49600, Loss: 110.8, Recon: 97.45
Epoch: 41, Iter: 50000, Loss: 112.0, Recon: 99.16
Epoch: 42, Iter: 50400, Loss: 103.7, Recon: 90.87
Epoch: 42, Iter: 50800, Loss: 111.6, Recon: 98.27
Epoch: 42, Iter: 51200, Loss: 112.0, Recon: 99.07
Epoch: 43, Iter: 51600, Loss: 104.0, Recon: 91.23
Epoch: 43, Iter: 52000, Loss: 110.8, Recon: 97.29
Epoch: 43, Iter: 52400, Loss: 112.2, Recon: 99.22
Epoch: 44, Iter: 52800, Loss: 103.7, Recon: 90.8
Epoch: 44, Iter: 53200, Loss: 111.4, Recon: 98.2
Epoch: 44, Iter: 53600, Loss: 111.6, Recon: 98.74
Epoch: 45, Iter: 54000, Loss: 104.6, Recon: 91.89
Epoch: 45, Iter: 54400, Loss: 111.5, Recon: 98.24
Epoch: 45, Iter: 54800, Loss: 112.2, Recon: 99.37
Epoch: 46, Iter: 55200, Loss: 103.9, Recon: 91.24
Epoch: 46, Iter: 55600, Loss: 110.8, Recon: 97.39
Epoch: 46, Iter: 56000, Loss: 111.3, Recon: 98.46
Epoch: 47, Iter: 56400, Loss: 103.8, Recon: 91.02
Epoch: 47, Iter: 56800, Loss: 110.7, Recon: 97.16
Epoch: 47, Iter: 57200, Loss: 112.9, Recon: 100.0
Epoch: 48, Iter: 57600, Loss: 103.7, Recon: 91.0
Epoch: 48, Iter: 58000, Loss: 110.7, Recon: 97.32
Epoch: 48, Iter: 58400, Loss: 110.9, Recon: 97.98
Epoch: 49, Iter: 58800, Loss: 104.4, Recon: 91.59
Epoch: 49, Iter: 59200, Loss: 111.4, Recon: 97.83
Epoch: 49, Iter: 59600, Loss: 111.5, Recon: 98.58
```

Visualize generated samples after training

```
z_gen = tf.random.normal(shape=[num_to_show, Z_DIM])
x_gen = P(z_gen)
imgs_numpy = tf.nn.sigmoid(x_gen).numpy()
show_images(imgs_numpy)
plt.show()
```



PART II. Compute the inception score for your trained VAE m

In this part, we will quantitatively measure how good your VAE model is.

Train a classifier

We first need to train a classifier.

```
batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

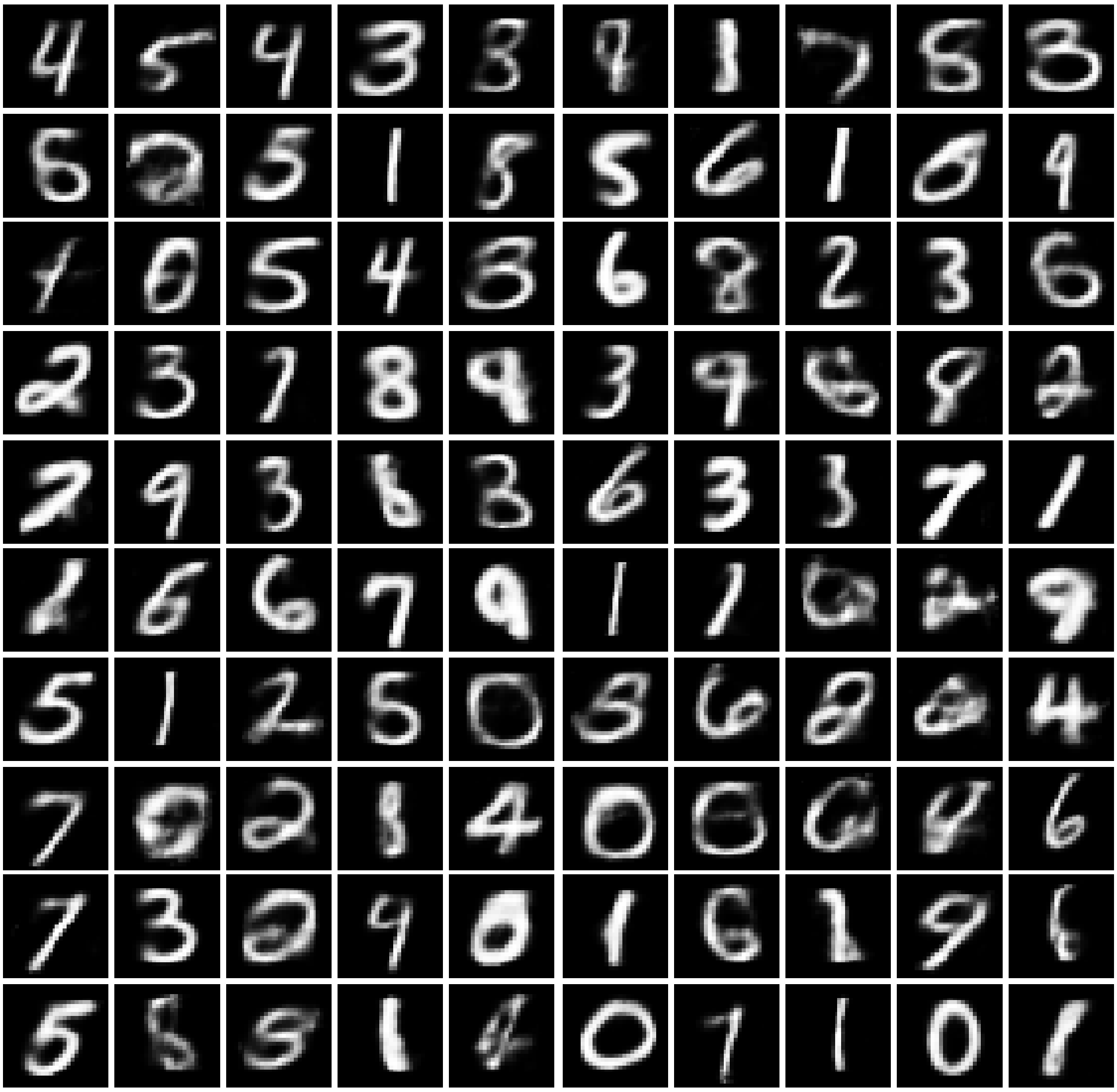
```
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
```

```
dropout_2 (Dropout)          (None, 512)              0
_________________________________________________________________
dense_44 (Dense)             (None, 512)              262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)              0
_________________________________________________________________
dense_45 (Dense)             (None, 10)               5130
```

Verify the trained classifier on the generated samples

Generate samples and visually inspect if the predicted labels on the samples match the actual dig

```
z_gen = tf.random.normal(shape=[num_samples, Z_DIM])
x_gen = P(z_gen)
imgs_numpy = tf.nn.sigmoid(x_gen[:num_to_show]).numpy()
show_images(imgs_numpy)
plt.show()
```



```
Epoch 20/20
```

```
np.argmax(model.predict(tf.nn.sigmoid(x_gen[:20])), axis=-1)
```

```
array([4, 5, 4, 3, 3, 8, 1, 7, 5, 3, 5, 0, 5, 1, 5, 5, 6, 1, 0, 9])
```