Variational Auto Encoder - Mnist

https://github.com/keras-team/keras/blob/master/examples/variational_autoencoder.py

because this one is broken: https://blog.keras.io/building-autoencoders-in-keras.html

[1]

```
from keras.datasets import mnist
import numpy as np
```

```
Using TensorFlow backend.
```

[2]

```
from keras.layers import Input, Lambda, Dense
from keras.models import Model
```

[3]

```
from keras import backend as K
from keras.utils import plot_model
```

[4]

```
# network parameters
original_dim=784
input_shape = (original_dim, )
intermediate_dim = 512
batch_size = 128
latent_dim = 2
epochs = 50
```

[5]

```
inputs = Input(shape=input_shape, name='encoder_input')
x = Dense(intermediate_dim, activation='relu')(inputs)

inputs.shape, x.shape
```

```
(TensorShape([None, 784]), TensorShape([None, 512]))
```

[6]

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

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:])))
```

[7]
```
z_mean = Dense(latent_dim, name='z_mean')(x)
z_log_var = Dense(latent_dim, name='z_log_var')(x)
```

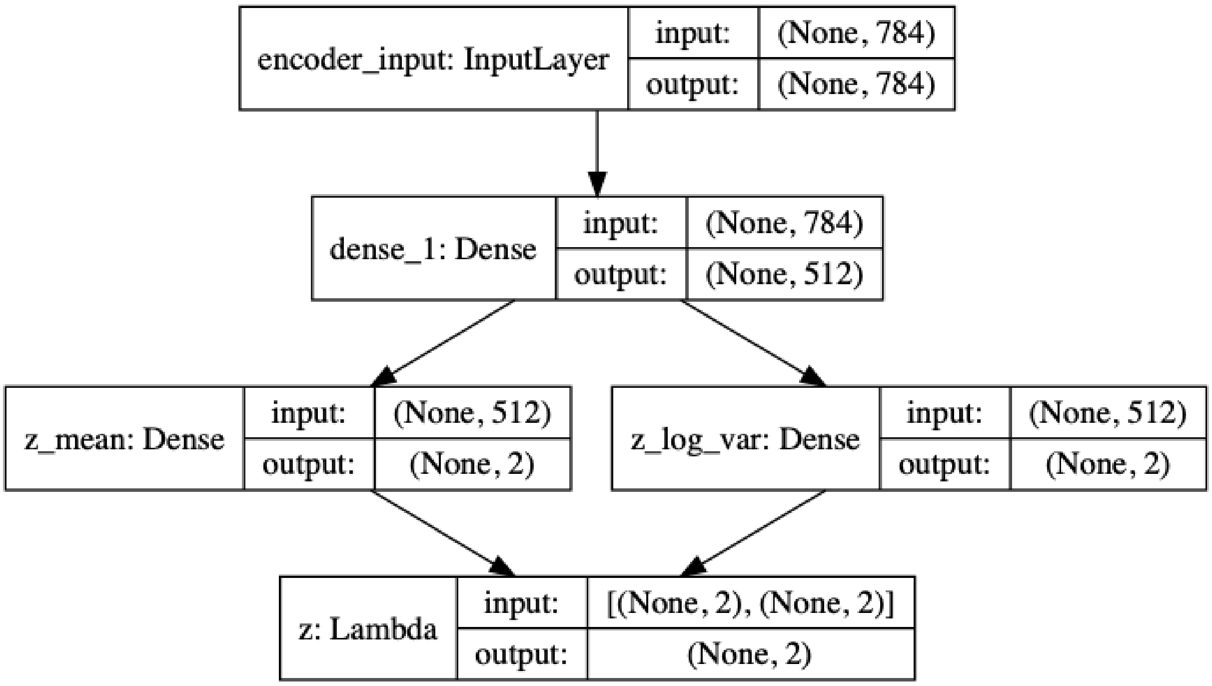
[8]
```
def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    # by default, random_normal has mean = 0 and std = 1.0
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon
```

[9]
```
z = Lambda(sampling, output_shape=(latent_dim,), name='z')
([z_mean, z_log_var])
```

[10]
```
# instantiate encoder model
encoder = Model(inputs, [z_mean, z_log_var, z], name='encoder')
encoder.summary()
plot_model(encoder, to_file='vae_mlp_encoder.png',
show_shapes=True)
```

```
Model: "encoder"
______________________________________________________________________
__________________________
Layer (type)                   Output Shape         Param #
Connected to
======================================================================
==========================
encoder_input (InputLayer)     (None, 784)          0                ...
______________________________________________________________________
__________________________
dense_1 (Dense)                (None, 512)          401920
encoder_input[0][0]
______________________________________________________________________
__________________________
z_mean (Dense)                 (None, 2)            1026
dense_1[0][0]
______________________________________________________________________
__________________________
z_log_var (Dense)              (None, 2)            1026
dense_1[0][0]
______________________________________________________________________
__________________________
z (Lambda)                     (None, 2)            0
z_mean[0][0]
```

```
z_log_var[0][0]
==================================================================================================
Total params: 403,972
Trainable params: 403,972
Non-trainable params: 0
__________________________________________________________________________________________________
```



[11]

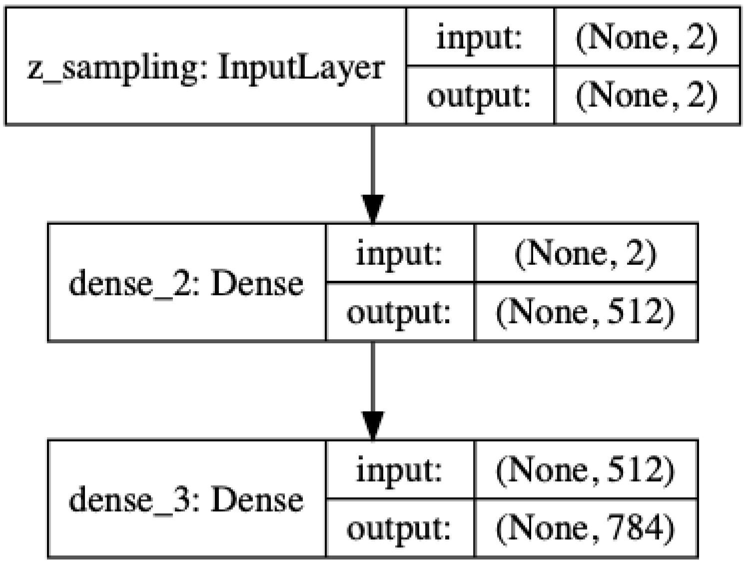
```python
# build decoder model
latent_inputs = Input(shape=(latent_dim,), name='z_sampling')
x = Dense(intermediate_dim, activation='relu')(latent_inputs)
outputs = Dense(original_dim, activation='sigmoid')(x)
```

[12]

```python
# instantiate decoder model
decoder = Model(latent_inputs, outputs, name='decoder')
decoder.summary()
plot_model(decoder, to_file='vae_mlp_decoder.png',
show_shapes=True)
```

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      (None, 2)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1536
_________________________________________________________________
dense_3 (Dense)              (None, 784)               402192
=================================================================
Total params: 403,728
```

```
Trainable params: 403,728
Non-trainable params: 0
_________________________________________________________________
```



[13]

```
# end-to-end autoencoder
#vae = Model(x, x_decoded_mean)
# instantiate VAE model
outputs = decoder(encoder(inputs)[2])
vae = Model(inputs, outputs, name='vae_mlp')
```

[15]

```
from keras.losses import mse, binary_crossentropy
```

[16]

```
# train
def vae_loss(x, x_decoded_mean):
    #xent_loss = objectives.binary_crossentropy(x,
x_decoded_mean)
    inputs, outputs = x, x_decoded_mean
    reconstruction_loss = mse(inputs, outputs)
    reconstruction_loss *= original_dim
    kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
    kl_loss = K.sum(kl_loss, axis=-1)
    kl_loss *= -0.5
    return K.mean(reconstruction_loss + kl_loss)
```

[17]

```
vae.compile(optimizer='adam', loss=vae_loss)
```

[18]

```
vae.summary()
```

```
Model: "vae_mlp"
```

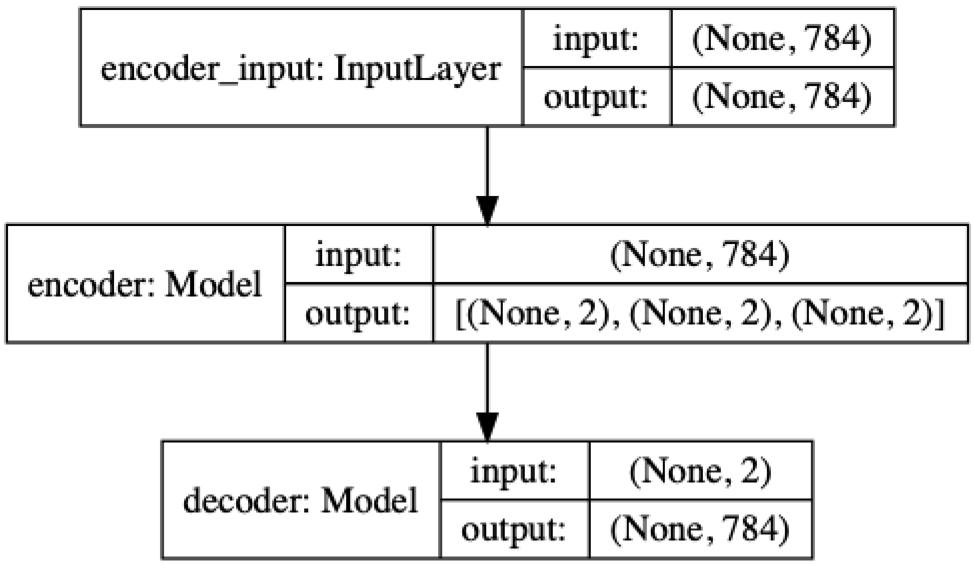
```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
encoder_input (InputLayer)   (None, 784)               0
________________________________________________________________
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
________________________________________________________________
```

[19]

```python
from keras.utils import plot_model
```

[20]

```python
plot_model(vae,
               to_file='vae_mlp.png',
               show_shapes=True)
```



[21]

```python
history = vae.fit(x_train, x_train,
        shuffle=True,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_test, x_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 8s 134us/step - loss:
52.4082 - val_loss: 44.3710
Epoch 2/50
60000/60000 [==============================] - 8s 133us/step - loss:
43.6279 - val_loss: 42.6135
```

```
Epoch 3/50
60000/60000 [==============================] - 8s 136us/step - loss:
42.4852 - val_loss: 41.8788
Epoch 4/50
60000/60000 [==============================] - 8s 137us/step - loss:
41.7411 - val_loss: 41.3524
Epoch 5/50
60000/60000 [==============================] - 8s 135us/step - loss:
41.1790 - val_loss: 40.7073
Epoch 6/50
60000/60000 [==============================] - 9s 142us/step - loss:
40.7657 - val_loss: 40.3398
Epoch 7/50
60000/60000 [==============================] - 9s 156us/step - loss:
40.4278 - val_loss: 40.0995
Epoch 8/50
60000/60000 [==============================] - 9s 158us/step - loss:
40.1484 - val_loss: 39.8697
Epoch 9/50
60000/60000 [==============================] - 9s 149us/step - loss:
39.8966 - val_loss: 39.6438
Epoch 10/50
60000/60000 [==============================] - 9s 148us/step - loss:
39.6594 - val_loss: 39.4342
Epoch 11/50
60000/60000 [==============================] - 9s 146us/step - loss:
39.4275 - val_loss: 39.2257
Epoch 12/50
60000/60000 [==============================] - 10s 164us/step - loss:
39.2567 - val_loss: 39.0303
Epoch 13/50
60000/60000 [==============================] - 9s 158us/step - loss:
39.0614 - val_loss: 39.0082
Epoch 14/50
60000/60000 [==============================] - 9s 155us/step - loss:
38.9119 - val_loss: 38.8353
Epoch 15/50
60000/60000 [==============================] - 10s 166us/step - loss:
38.7506 - val_loss: 38.7820
Epoch 16/50
60000/60000 [==============================] - 9s 151us/step - loss:
38.6269 - val_loss: 38.7351
Epoch 17/50
60000/60000 [==============================] - 9s 149us/step - loss:
38.4955 - val_loss: 38.5367
Epoch 18/50
60000/60000 [==============================] - 9s 157us/step - loss:
38.3709 - val_loss: 38.4201
Epoch 19/50
60000/60000 [==============================] - 9s 150us/step - loss:
38.2583 - val_loss: 38.3087
Epoch 20/50
```

```
60000/60000 [==============================] - 9s 149us/step - loss:
38.1325 - val_loss: 38.3267
Epoch 21/50
60000/60000 [==============================] - 9s 149us/step - loss:
38.0338 - val_loss: 38.2504
Epoch 22/50
60000/60000 [==============================] - 9s 150us/step - loss:
37.9441 - val_loss: 38.2131
Epoch 23/50
60000/60000 [==============================] - 9s 150us/step - loss:
37.8620 - val_loss: 38.1799
Epoch 24/50
60000/60000 [==============================] - 9s 149us/step - loss:
37.7863 - val_loss: 38.1288
Epoch 25/50
60000/60000 [==============================] - 9s 152us/step - loss:
37.6941 - val_loss: 38.1089
Epoch 26/50
60000/60000 [==============================] - 9s 148us/step - loss:
37.6172 - val_loss: 38.0469
Epoch 27/50
60000/60000 [==============================] - 9s 149us/step - loss:
37.5183 - val_loss: 37.9198
Epoch 28/50
60000/60000 [==============================] - 9s 156us/step - loss:
37.4641 - val_loss: 37.8002
Epoch 29/50
60000/60000 [==============================] - 9s 147us/step - loss:
37.4065 - val_loss: 37.9389
Epoch 30/50
60000/60000 [==============================] - 9s 151us/step - loss:
37.3149 - val_loss: 37.8055
Epoch 31/50
60000/60000 [==============================] - 9s 154us/step - loss:
37.2568 - val_loss: 37.6736
Epoch 32/50
60000/60000 [==============================] - 10s 159us/step - loss:
37.1849 - val_loss: 37.7372
Epoch 33/50
60000/60000 [==============================] - 10s 162us/step - loss:
37.1077 - val_loss: 37.6555
Epoch 34/50
60000/60000 [==============================] - 10s 171us/step - loss:
37.0747 - val_loss: 37.6494
Epoch 35/50
60000/60000 [==============================] - 9s 156us/step - loss:
36.9943 - val_loss: 37.5102
Epoch 36/50
60000/60000 [==============================] - 9s 153us/step - loss:
36.9459 - val_loss: 37.5556
Epoch 37/50
60000/60000 [==============================] - 10s 164us/step - loss:
```

```
36.8788 - val_loss: 37.5778
Epoch 38/50
60000/60000 [==============================] - 10s 161us/step - loss:
36.8505 - val_loss: 37.4985
Epoch 39/50
60000/60000 [==============================] - 10s 162us/step - loss:
36.7803 - val_loss: 37.4840
Epoch 40/50
60000/60000 [==============================] - 9s 156us/step - loss:
36.7545 - val_loss: 37.4872
Epoch 41/50
60000/60000 [==============================] - 9s 152us/step - loss:
36.7014 - val_loss: 37.4429
Epoch 42/50
60000/60000 [==============================] - 9s 153us/step - loss:
36.6429 - val_loss: 37.4123
Epoch 43/50
60000/60000 [==============================] - 9s 152us/step - loss:
36.5976 - val_loss: 37.3638
Epoch 44/50
60000/60000 [==============================] - 9s 147us/step - loss:
36.5704 - val_loss: 37.3885
Epoch 45/50
60000/60000 [==============================] - 9s 152us/step - loss:
36.5229 - val_loss: 37.4105
Epoch 46/50
60000/60000 [==============================] - 9s 156us/step - loss:
36.4764 - val_loss: 37.2376
Epoch 47/50
60000/60000 [==============================] - 10s 159us/step - loss:
36.4647 - val_loss: 37.3661
Epoch 48/50
60000/60000 [==============================] - 9s 149us/step - loss:
36.4181 - val_loss: 37.1787
Epoch 49/50
60000/60000 [==============================] - 9s 155us/step - loss:
36.3673 - val_loss: 37.3772
Epoch 50/50
60000/60000 [==============================] - 9s 158us/step - loss:
36.3420 - val_loss: 37.2001
```

[22]

```
vae.save_weights('vae_mlp_mnist.h5')
```

[23]

```
import os
import matplotlib.pyplot as plt
```

[24]

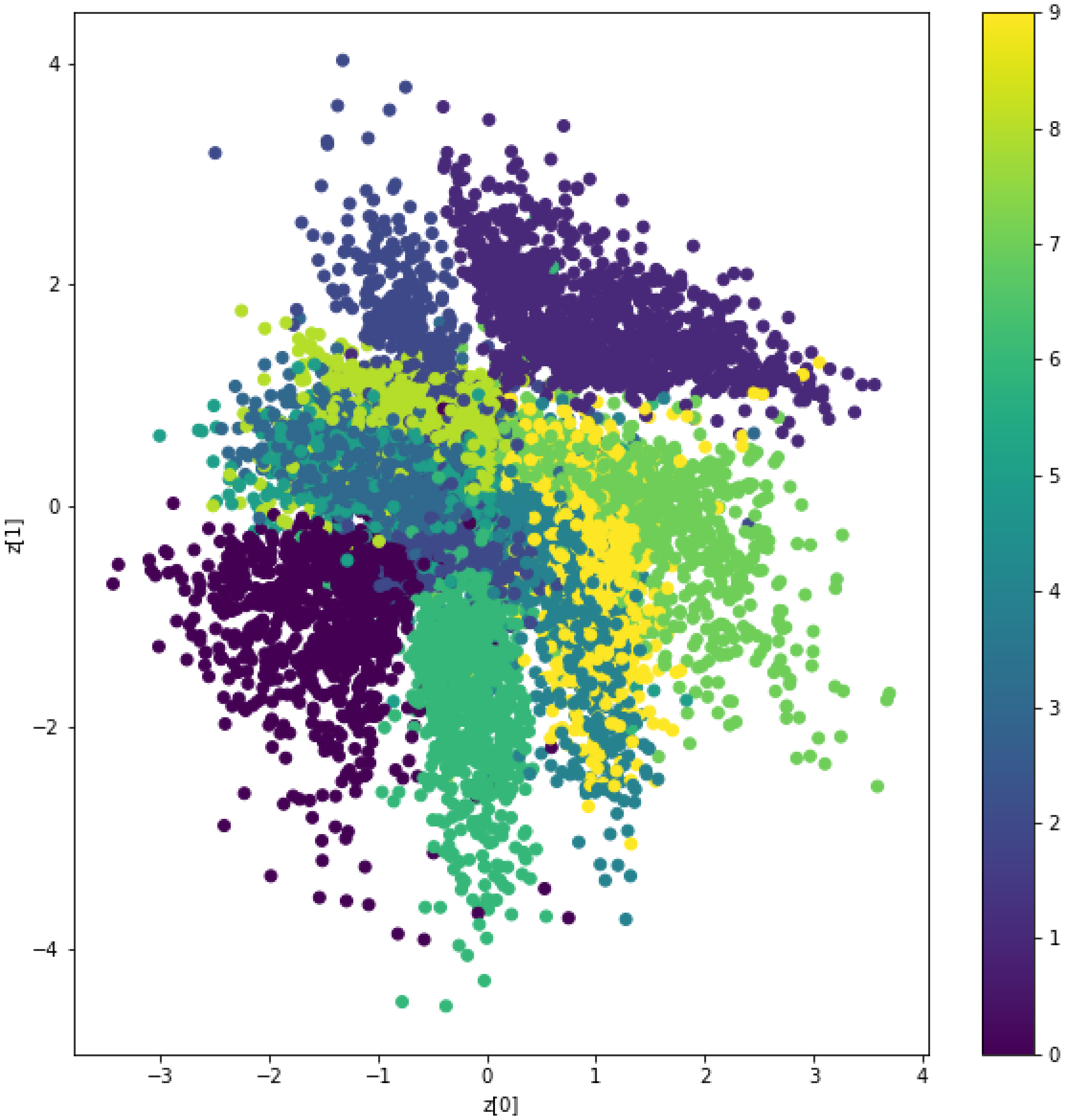
```
def plot_results(models,
                 data,
```

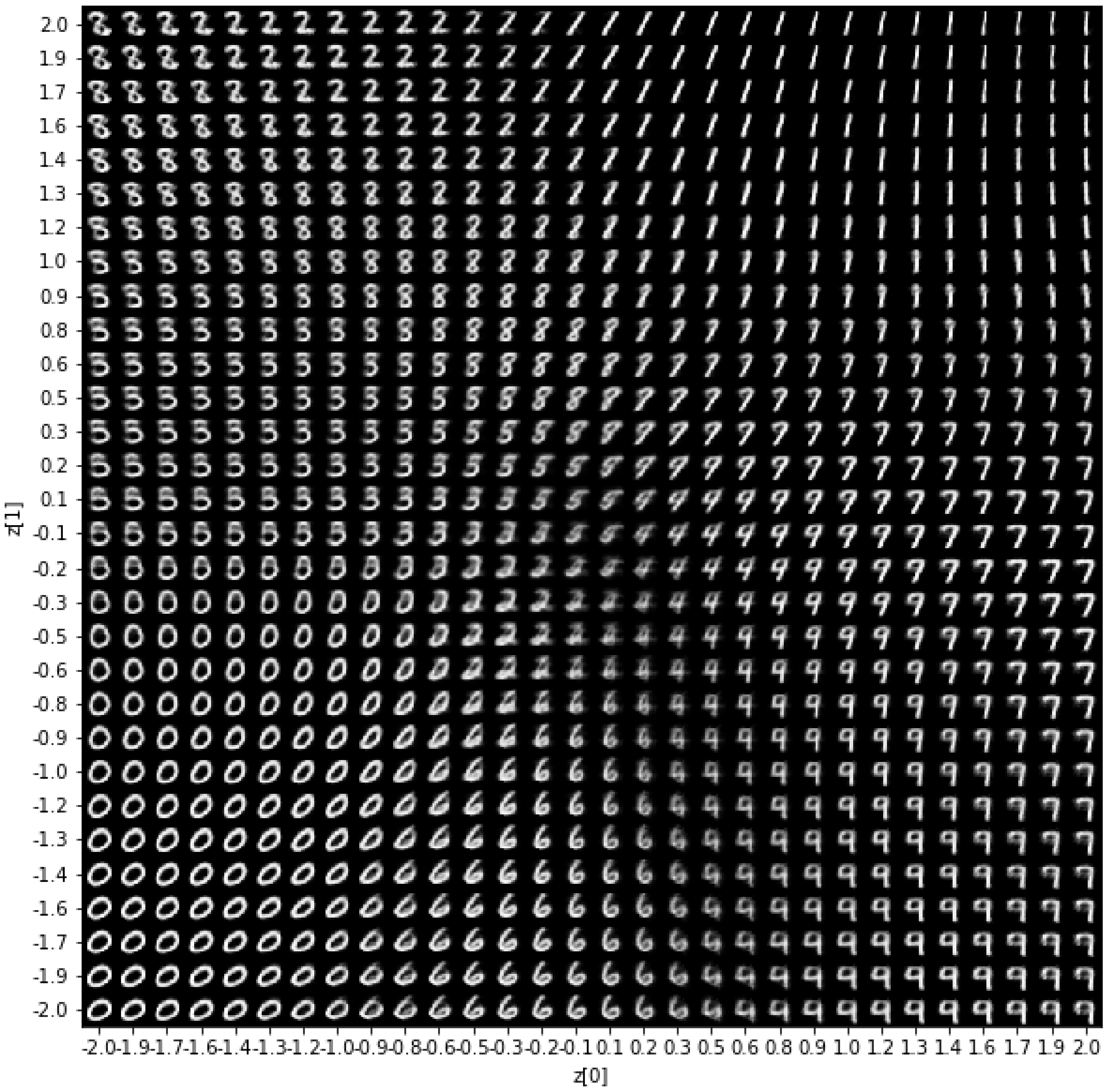
```
                 batch_size=128,
                 model_name="vae_mnist"):
    """Plots labels and MNIST digits as a function of the 2D
latent vector
    # Arguments
        models (tuple): encoder and decoder models
        data (tuple): test data and label
        batch_size (int): prediction batch size
        model_name (string): which model is using this function
    """

    encoder, decoder = models
    x_test, y_test = data
    os.makedirs(model_name, exist_ok=True)

    filename = os.path.join(model_name, "vae_mean.png")
    # display a 2D plot of the digit classes in the latent space
    z_mean, _, _ = encoder.predict(x_test,
                                   batch_size=batch_size)
    plt.figure(figsize=(10, 10))
    plt.scatter(z_mean[:, 0], z_mean[:, 1], c=y_test)
    plt.colorbar()
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.savefig(filename)
    plt.show()

    filename = os.path.join(model_name, "digits_over_latent.png")
    # display a 30x30 2D manifold of digits
    n = 30
    digit_size = 28
    figure = np.zeros((digit_size * n, digit_size * n))
    # linearly spaced coordinates corresponding to the 2D plot
    # of digit classes in the latent space
    grid_x = np.linspace(-2, 2, n)
    grid_y = np.linspace(-2, 2, n)[::-1]

    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = np.array([[xi, yi]])
            x_decoded = decoder.predict(z_sample)
            digit = x_decoded[0].reshape(digit_size, digit_size)
            figure[i * digit_size: (i + 1) * digit_size,
                   j * digit_size: (j + 1) * digit_size] = digit

    plt.figure(figsize=(10, 10))
    start_range = digit_size // 2
    end_range = (n - 1) * digit_size + start_range + 1
    pixel_range = np.arange(start_range, end_range, digit_size)
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
```

```
        plt.yticks(pixel_range, sample_range_y)
        plt.xlabel("z[0]")
        plt.ylabel("z[1]")
        plt.imshow(figure, cmap='Greys_r')
        plt.savefig(filename)
        plt.show()
```

[25]
```
models = (encoder, decoder)
data = (x_test, y_test)
```

[26]
```
plot_results(models,
                 data,
                 batch_size=batch_size,
                 model_name="vae_mlp")
```





[27]

```
# from above -1,1 and -1,1 is pretty tight
# -2,2 and -2,2 leaves the corners empty
# 8 2 1
# 3 8 1
# 53497
# 06497
# 0649
```