Conditional Variational Autoencoder (VAE)

```
import warnings
import numpy as np
from keras.layers import Input, Dense, Lambda
from keras.layers.merge import concatenate as concat
from keras.models import Model
from keras import backend as K
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from scipy.misc import imsave
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')
%pylab inline
```

```
Using TensorFlow backend.

Populating the interactive namespace from numpy and matplotlib
```

```
# Data import
# MNIST data is separated into training and test partitions, with
separate X (pixel representation) and y (label value)
# The X matrices are 28x28 numpy arrays, while the y is just an
integer.
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

# Reshaping
# Properly represent the pixel information contained in X to a fully-
connected feed forward neural network
X_train = X_train.astype('float32') / 255.
X_test = X_test.astype('float32') / 255.

n_pixels = np.prod(X_train.shape[1:])
X_train = X_train.reshape((len(X_train), n_pixels))
X_test = X_test.reshape((len(X_test), n_pixels))

# Properly represent the label y
y_train = to_categorical(Y_train)
y_test = to_categorical(Y_test)

# Hyperparameters
m = 250 # batch size
n_z = 2 # latent space size
encoder_dim1 = 512 # dim of encoder hidden layer
decoder_dim = 512 # dim of decoder hidden layer
decoder_out_dim = 784 # dim of decoder output layer
activ = 'relu'
optim = Adam(lr=0.001)
```

```python
n_x = X_train.shape[1]
n_y = y_train.shape[1]


n_epoch = 50

# The encoder
X = Input(shape=(n_x,))
label = Input(shape=(n_y,))

inputs = concat([X, label])

encoder_h = Dense(encoder_dim1, activation=activ)(inputs)
mu = Dense(n_z, activation='linear')(encoder_h)
l_sigma = Dense(n_z, activation='linear')(encoder_h)

def sample_z(args):
    mu, l_sigma = args
    eps = K.random_normal(shape=(m, n_z), mean=0., stddev=1.)
    return mu + K.exp(l_sigma / 2) * eps

# Sampling latent space
z = Lambda(sample_z, output_shape = (n_z, ))([mu, l_sigma])

# The latent space
z = Lambda(sample_z, output_shape = (n_z, ))([mu, l_sigma])

# merge latent space with label
zc = concat([z, label])

# Decoder
decoder_hidden = Dense(decoder_dim, activation=activ)
decoder_out = Dense(decoder_out_dim, activation='sigmoid')
h_p = decoder_hidden(zc)
outputs = decoder_out(h_p)

# Defining loss
def vae_loss(y_true, y_pred):
    recon = K.sum(K.binary_crossentropy(y_true, y_pred), axis=-1)
    kl = 0.5 * K.sum(K.exp(l_sigma) + K.square(mu) - 1. - l_sigma,
axis=-1)
    return recon + kl

def KL_loss(y_true, y_pred):
     return(0.5 * K.sum(K.exp(l_sigma) + K.square(mu) - 1. - l_sigma,
axis=1))
```

```
def recon_loss(y_true, y_pred):
     return K.sum(K.binary_crossentropy(y_true, y_pred), axis=-1)

# Defining the graphs
cvae = Model([X, label], outputs)
encoder = Model([X, label], mu)

d_in = Input(shape=(n_z+n_y,))
d_h = decoder_hidden(d_in)
d_out = decoder_out(d_h)
decoder = Model(d_in, d_out)

# Training
cvae.compile(optimizer=optim, loss=vae_loss, metrics = [KL_loss,
recon_loss])

# compile and fit
cvae_hist = cvae.fit([X_train, y_train], X_train, verbose = 1,
batch_size=m, epochs=n_epoch,
                                    validation_data = ([X_test,
y_test], X_test),
                                    callbacks =
[EarlyStopping(patience = 5)])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 2s - loss: 198.5605 -
KL_loss: 9.2840 - recon_loss: 189.2765 - val_loss: 154.5540 -
val_KL_loss: 4.7290 - val_recon_loss: 149.8251
Epoch 2/50
60000/60000 [==============================] - 1s - loss: 148.7884 -
KL_loss: 4.4363 - recon_loss: 144.3522 - val_loss: 144.2921 -
val_KL_loss: 4.2789 - val_recon_loss: 140.0132
Epoch 3/50
60000/60000 [==============================] - 1s - loss: 142.8696 -
KL_loss: 4.2707 - recon_loss: 138.5989 - val_loss: 141.2003 -
val_KL_loss: 4.3155 - val_recon_loss: 136.8848
Epoch 4/50
60000/60000 [==============================] - 1s - loss: 140.4333 -
KL_loss: 4.2716 - recon_loss: 136.1617 - val_loss: 139.4493 -
val_KL_loss: 4.2255 - val_recon_loss: 135.2238
Epoch 5/50
60000/60000 [==============================] - 1s - loss: 138.9394 -
KL_loss: 4.2724 - recon_loss: 134.6669 - val_loss: 138.1692 -
val_KL_loss: 4.1830 - val_recon_loss: 133.9862
Epoch 6/50
60000/60000 [==============================] - 1s - loss: 137.8364 -
KL_loss: 4.2851 - recon_loss: 133.5513 - val_loss: 137.4907 -
val_KL_loss: 4.2063 - val_recon_loss: 133.2844
Epoch 7/50
60000/60000 [==============================] - 1s - loss: 136.9856 -
```

```
KL_loss: 4.3045 - recon_loss: 132.6812 - val_loss: 136.7183 -
val_KL_loss: 4.2148 - val_recon_loss: 132.5035
Epoch 8/50
60000/60000 [==============================] - 1s - loss: 136.3045 -
KL_loss: 4.3180 - recon_loss: 131.9866 - val_loss: 136.0102 -
val_KL_loss: 4.3962 - val_recon_loss: 131.6140
Epoch 9/50
60000/60000 [==============================] - 1s - loss: 135.7264 -
KL_loss: 4.3400 - recon_loss: 131.3864 - val_loss: 135.5455 -
val_KL_loss: 4.4053 - val_recon_loss: 131.1402
Epoch 10/50
60000/60000 [==============================] - 1s - loss: 135.2136 -
KL_loss: 4.3617 - recon_loss: 130.8519 - val_loss: 135.1110 -
val_KL_loss: 4.3112 - val_recon_loss: 130.7998
Epoch 11/50
60000/60000 [==============================] - 1s - loss: 134.7685 -
KL_loss: 4.3776 - recon_loss: 130.3910 - val_loss: 134.8785 -
val_KL_loss: 4.2932 - val_recon_loss: 130.5853
Epoch 12/50
60000/60000 [==============================] - 1s - loss: 134.3657 -
KL_loss: 4.3935 - recon_loss: 129.9721 - val_loss: 134.5110 -
val_KL_loss: 4.2835 - val_recon_loss: 130.2275
Epoch 13/50
60000/60000 [==============================] - 1s - loss: 134.0535 -
KL_loss: 4.4186 - recon_loss: 129.6349 - val_loss: 134.1402 -
val_KL_loss: 4.3904 - val_recon_loss: 129.7497
Epoch 14/50
60000/60000 [==============================] - 1s - loss: 133.7379 -
KL_loss: 4.4327 - recon_loss: 129.3052 - val_loss: 133.8768 -
val_KL_loss: 4.2755 - val_recon_loss: 129.6013
Epoch 15/50
60000/60000 [==============================] - 1s - loss: 133.4439 -
KL_loss: 4.4483 - recon_loss: 128.9956 - val_loss: 133.7845 -
val_KL_loss: 4.4687 - val_recon_loss: 129.3158
Epoch 16/50
60000/60000 [==============================] - 1s - loss: 133.1637 -
KL_loss: 4.4669 - recon_loss: 128.6968 - val_loss: 133.5564 -
val_KL_loss: 4.3429 - val_recon_loss: 129.2135
Epoch 17/50
60000/60000 [==============================] - 1s - loss: 132.9419 -
KL_loss: 4.4708 - recon_loss: 128.4711 - val_loss: 133.2376 -
val_KL_loss: 4.4014 - val_recon_loss: 128.8363
Epoch 18/50
60000/60000 [==============================] - 1s - loss: 132.7160 -
KL_loss: 4.4896 - recon_loss: 128.2264 - val_loss: 132.9692 -
val_KL_loss: 4.4500 - val_recon_loss: 128.5193
Epoch 19/50
60000/60000 [==============================] - 1s - loss: 132.5305 -
KL_loss: 4.4989 - recon_loss: 128.0316 - val_loss: 132.8874 -
val_KL_loss: 4.4893 - val_recon_loss: 128.3981
```

```
Epoch 20/50
60000/60000 [==============================] - 1s - loss: 132.3395 -
KL_loss: 4.5129 - recon_loss: 127.8266 - val_loss: 132.7876 -
val_KL_loss: 4.3981 - val_recon_loss: 128.3896
Epoch 21/50
60000/60000 [==============================] - 1s - loss: 132.1601 -
KL_loss: 4.5210 - recon_loss: 127.6391 - val_loss: 132.5717 -
val_KL_loss: 4.4895 - val_recon_loss: 128.0822
Epoch 22/50
60000/60000 [==============================] - 1s - loss: 131.9843 -
KL_loss: 4.5407 - recon_loss: 127.4437 - val_loss: 132.4869 -
val_KL_loss: 4.4143 - val_recon_loss: 128.0726
Epoch 23/50
60000/60000 [==============================] - 1s - loss: 131.8412 -
KL_loss: 4.5412 - recon_loss: 127.3000 - val_loss: 132.3437 -
val_KL_loss: 4.5734 - val_recon_loss: 127.7703
Epoch 24/50
60000/60000 [==============================] - 1s - loss: 131.6655 -
KL_loss: 4.5555 - recon_loss: 127.1100 - val_loss: 132.1972 -
val_KL_loss: 4.4066 - val_recon_loss: 127.7906
Epoch 25/50
60000/60000 [==============================] - 1s - loss: 131.5629 -
KL_loss: 4.5702 - recon_loss: 126.9928 - val_loss: 132.1781 -
val_KL_loss: 4.4901 - val_recon_loss: 127.6880
Epoch 26/50
60000/60000 [==============================] - 1s - loss: 131.4057 -
KL_loss: 4.5783 - recon_loss: 126.8275 - val_loss: 132.1023 -
val_KL_loss: 4.4415 - val_recon_loss: 127.6608
Epoch 27/50
60000/60000 [==============================] - 1s - loss: 131.2994 -
KL_loss: 4.5753 - recon_loss: 126.7241 - val_loss: 132.0479 -
val_KL_loss: 4.5562 - val_recon_loss: 127.4917
Epoch 28/50
60000/60000 [==============================] - 1s - loss: 131.1553 -
KL_loss: 4.5959 - recon_loss: 126.5594 - val_loss: 131.8009 -
val_KL_loss: 4.5111 - val_recon_loss: 127.2898
Epoch 29/50
60000/60000 [==============================] - 1s - loss: 131.0562 -
KL_loss: 4.5988 - recon_loss: 126.4575 - val_loss: 131.8298 -
val_KL_loss: 4.5768 - val_recon_loss: 127.2530
Epoch 30/50
60000/60000 [==============================] - 1s - loss: 130.9569 -
KL_loss: 4.6155 - recon_loss: 126.3414 - val_loss: 131.6415 -
val_KL_loss: 4.4890 - val_recon_loss: 127.1526
Epoch 31/50
60000/60000 [==============================] - 1s - loss: 130.8494 -
KL_loss: 4.6155 - recon_loss: 126.2339 - val_loss: 131.6697 -
val_KL_loss: 4.5291 - val_recon_loss: 127.1405
Epoch 32/50
60000/60000 [==============================] - 1s - loss: 130.7144 -
```

```
KL_loss: 4.6240 - recon_loss: 126.0904 - val_loss: 131.5643 -
val_KL_loss: 4.6457 - val_recon_loss: 126.9186
Epoch 33/50
60000/60000 [==============================] - 1s - loss: 130.6359 -
KL_loss: 4.6379 - recon_loss: 125.9980 - val_loss: 131.4646 -
val_KL_loss: 4.5177 - val_recon_loss: 126.9469
Epoch 34/50
60000/60000 [==============================] - 1s - loss: 130.5499 -
KL_loss: 4.6443 - recon_loss: 125.9056 - val_loss: 131.4492 -
val_KL_loss: 4.6452 - val_recon_loss: 126.8040
Epoch 35/50
60000/60000 [==============================] - 1s - loss: 130.4587 -
KL_loss: 4.6537 - recon_loss: 125.8050 - val_loss: 131.3858 -
val_KL_loss: 4.6927 - val_recon_loss: 126.6932
Epoch 36/50
60000/60000 [==============================] - 1s - loss: 130.3741 -
KL_loss: 4.6564 - recon_loss: 125.7178 - val_loss: 131.3302 -
val_KL_loss: 4.5754 - val_recon_loss: 126.7548
Epoch 37/50
60000/60000 [==============================] - 1s - loss: 130.2796 -
KL_loss: 4.6561 - recon_loss: 125.6235 - val_loss: 131.2440 -
val_KL_loss: 4.5587 - val_recon_loss: 126.6853
Epoch 38/50
60000/60000 [==============================] - 1s - loss: 130.2118 -
KL_loss: 4.6805 - recon_loss: 125.5314 - val_loss: 131.2306 -
val_KL_loss: 4.6014 - val_recon_loss: 126.6292
Epoch 39/50
60000/60000 [==============================] - 1s - loss: 130.1300 -
KL_loss: 4.6743 - recon_loss: 125.4556 - val_loss: 131.0826 -
val_KL_loss: 4.7000 - val_recon_loss: 126.3826
Epoch 40/50
60000/60000 [==============================] - 1s - loss: 130.0341 -
KL_loss: 4.6793 - recon_loss: 125.3548 - val_loss: 131.0892 -
val_KL_loss: 4.6052 - val_recon_loss: 126.4840
Epoch 41/50
60000/60000 [==============================] - 1s - loss: 129.9807 -
KL_loss: 4.6819 - recon_loss: 125.2988 - val_loss: 131.1555 -
val_KL_loss: 4.6426 - val_recon_loss: 126.5129
Epoch 42/50
60000/60000 [==============================] - 1s - loss: 129.9151 -
KL_loss: 4.6922 - recon_loss: 125.2229 - val_loss: 131.0138 -
val_KL_loss: 4.5813 - val_recon_loss: 126.4325
Epoch 43/50

60000/60000 [==============================] - 1s - loss: 129.8415 -
KL_loss: 4.7006 - recon_loss: 125.1409 - val_loss: 130.9835 -
val_KL_loss: 4.5907 - val_recon_loss: 126.3928
Epoch 44/50
60000/60000 [==============================] - 1s - loss: 129.7658 -
KL_loss: 4.7099 - recon_loss: 125.0559 - val_loss: 130.8835 -
```

```
val_KL_loss: 4.6845 - val_recon_loss: 126.1990
Epoch 45/50
60000/60000 [==============================] - 1s - loss: 129.6964 -
KL_loss: 4.7125 - recon_loss: 124.9839 - val_loss: 130.9387 -
val_KL_loss: 4.5784 - val_recon_loss: 126.3604
Epoch 46/50
60000/60000 [==============================] - 1s - loss: 129.6490 -
KL_loss: 4.7272 - recon_loss: 124.9218 - val_loss: 130.8578 -
val_KL_loss: 4.5969 - val_recon_loss: 126.2610
Epoch 47/50
60000/60000 [==============================] - 1s - loss: 129.5759 -
KL_loss: 4.7186 - recon_loss: 124.8573 - val_loss: 130.9210 -
val_KL_loss: 4.6908 - val_recon_loss: 126.2302
Epoch 48/50
60000/60000 [==============================] - 1s - loss: 129.5281 -
KL_loss: 4.7285 - recon_loss: 124.7996 - val_loss: 130.8939 -
val_KL_loss: 4.6782 - val_recon_loss: 126.2157
Epoch 49/50
60000/60000 [==============================] - 1s - loss: 129.4802 -
KL_loss: 4.7279 - recon_loss: 124.7523 - val_loss: 130.7622 -
val_KL_loss: 4.6026 - val_recon_loss: 126.1596
Epoch 50/50
60000/60000 [==============================] - 1s - loss: 129.4192 -
KL_loss: 4.7410 - recon_loss: 124.6783 - val_loss: 130.7682 -
val_KL_loss: 4.6136 - val_recon_loss: 126.1546
```

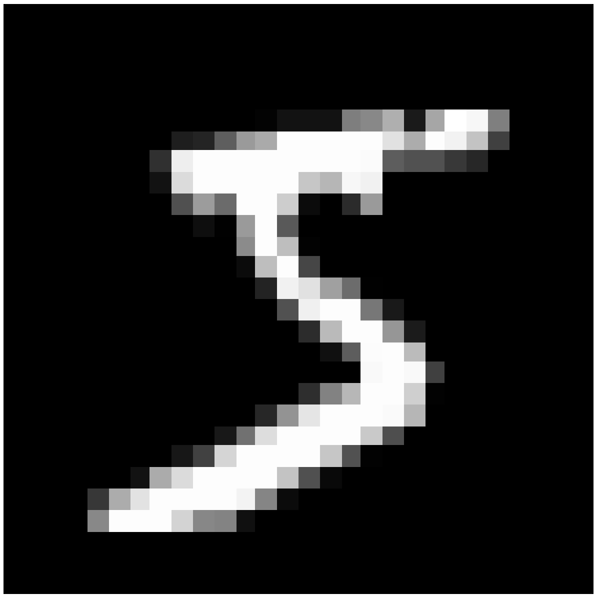
Exploring the model

The latent space should hopefully contain some interesting structural information about the digits we're autoencoding. That's the case in any autoencoding network, but in a VAE the spatial arrangement should make more intuitive 'sense' since the noise added to the latent space representation forces the model to create useful respresentations.

Generating a latent space representation with the encoder

First let's see concretely what happens when we pass an image and class to the encoder. We can take a look at the first image in the training set:

```
# Generating a latent space representation with the encoder
# see what happens when we pass an image and class to the encoder
# take a look at the first image in the training set
plt.imshow(X_train[0].reshape(28, 28), cmap = plt.cm.gray),
axis('off')
plt.show()
```



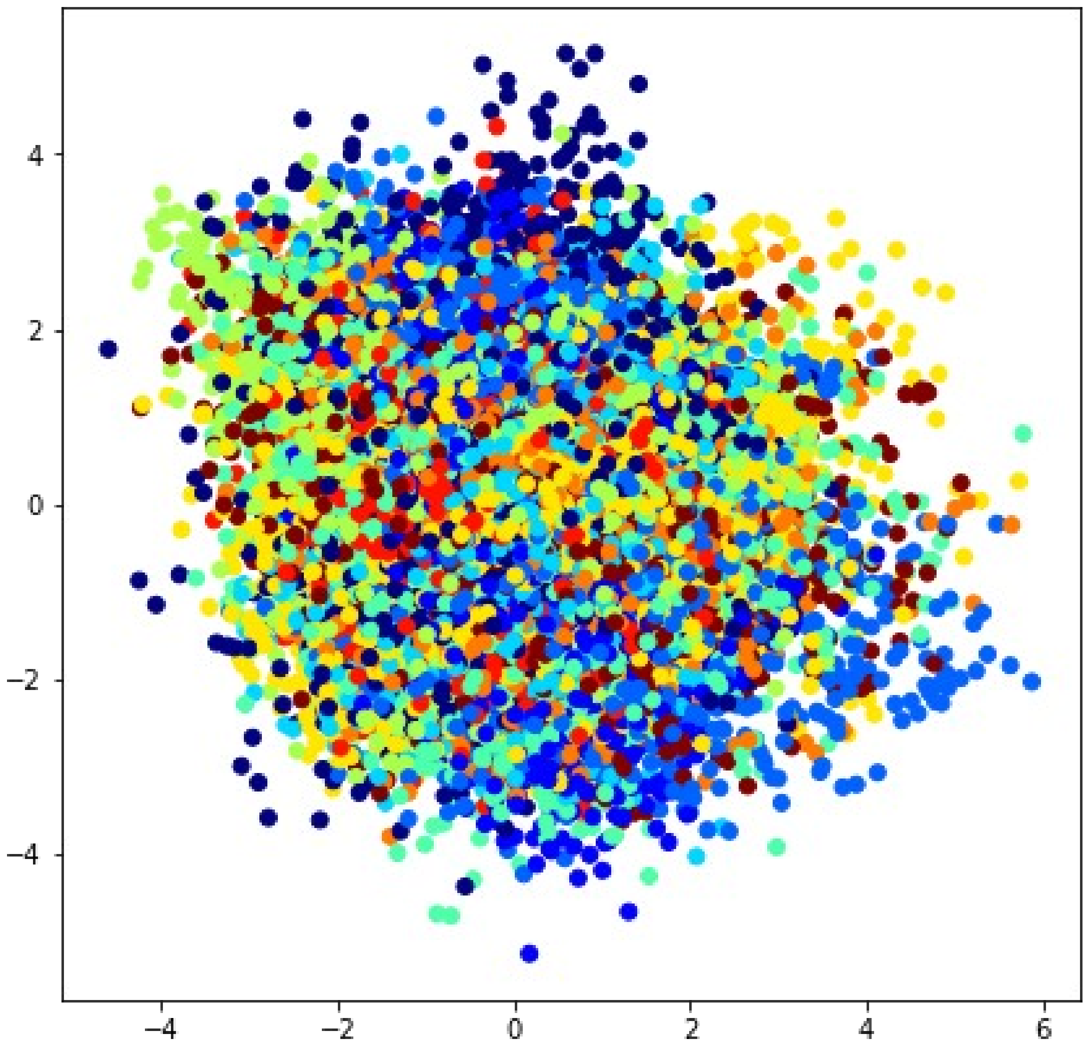
```
# image is 5, the first image in the training set also says it is 5 by
the truth label
print(Y_train[0])
```

```
5
```

```
# how model represents that same digit in the latent space by passing
it through the encoder
encoded_X0 = encoder.predict([X_train[0].reshape((1, 784)),
y_train[0].reshape((1, 10))])
print(encoded_X0)
```

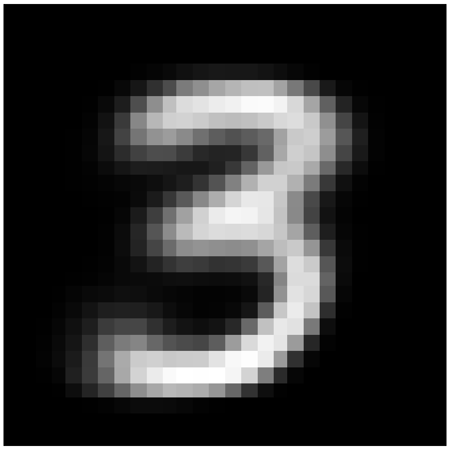
```
[[-0.02542629  0.3583132 ]]
```

```
# Since we append the class label directly to the latent space
representation,
    # our network doesn't need to store any information about which
digit it generates in the latent space
# encode our whole training set
z_train = encoder.predict([X_train, y_train])
encodings= np.asarray(z_train)
encodings = encodings.reshape(X_train.shape[0], n_z)
plt.figure(figsize=(7, 7))
plt.scatter(encodings[:, 0], encodings[:, 1], c=Y_train,
cmap=plt.cm.jet)
plt.show()
# all of the digits (represented by the different colors) are pretty
much layered on top of each other and are distributed approximately
bivariate normal
# this is what we would expect to happen.
```

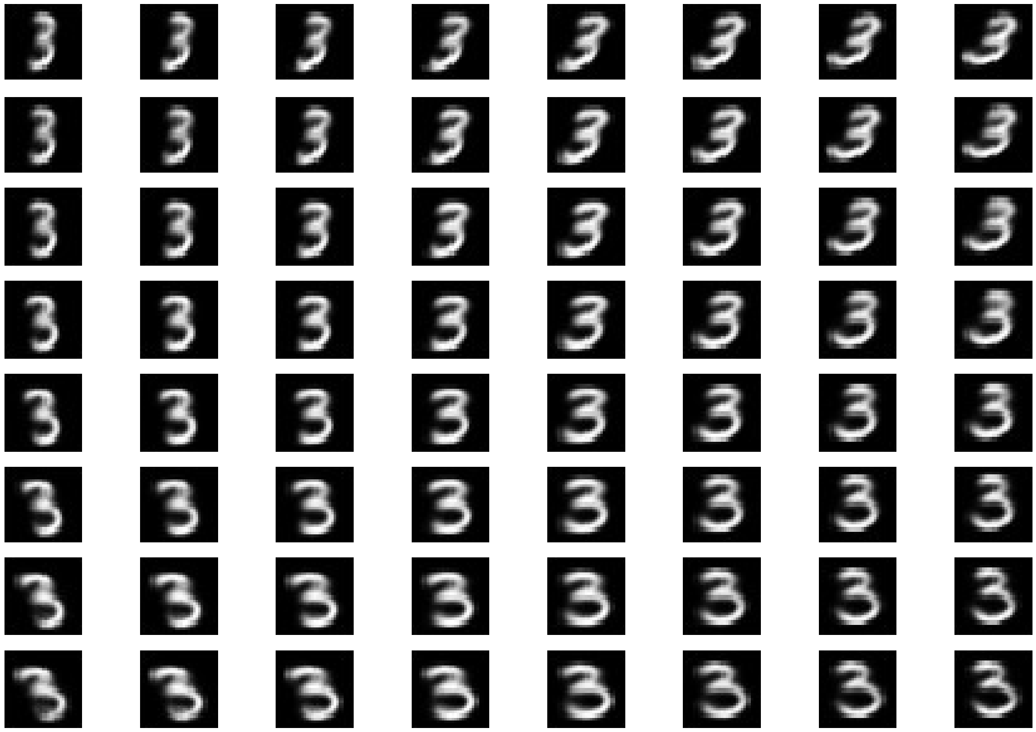



```
# Generating a digit
def construct_numvec(digit, z = None):
    out = np.zeros((1, n_z + n_y))
    out[:, digit + n_z] = 1.
    if z is None:
        return(out)
    else:
        for i in range(len(z)):
            out[:,i] = z[i]
        return(out)

sample_3 = construct_numvec(3)
print(sample_3)

[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]]

plt.figure(figsize=(3, 3))
plt.imshow(decoder.predict(sample_3).reshape(28,28), cmap =
```

```
plt.cm.gray), axis('off')
plt.show()
```



```
# Exploring the latent space variables
dig = 3
sides = 8
max_z = 1.5

img_it = 0
for i in range(0, sides):
    z1 = (((i / (sides-1)) * max_z)*2) - max_z
    for j in range(0, sides):
        z2 = (((j / (sides-1)) * max_z)*2) - max_z
        z_ = [z1, z2]
        vec = construct_numvec(dig, z_)
        decoded = decoder.predict(vec)
        subplot(sides, sides, 1 + img_it)
        img_it +=1
        plt.imshow(decoded.reshape(28, 28), cmap = plt.cm.gray),
axis('off')
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0,
hspace=.2)
plt.show()
# As z1 changes (on the y-axis), the digit style becomes narrower
# Varying the value of z2 (on the x-axis) appears to rotate the digit
slightly and elongate the lower portion in relation to the upper
portion
# There appears to be some interaction between the two values.
```



```
dig = 2
sides = 8
max_z = 1.5

img_it = 0
for i in range(0, sides):
    z1 = (((i / (sides-1)) * max_z)*2) - max_z
    for j in range(0, sides):
        z2 = (((j / (sides-1)) * max_z)*2) - max_z
        z_ = [z1, z2]
        vec = construct_numvec(dig, z_)
        decoded = decoder.predict(vec)
        subplot(sides, sides, 1 + img_it)
        img_it +=1
        plt.imshow(decoded.reshape(28, 28), cmap = plt.cm.gray),
axis('off')
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0,
hspace=.2)
plt.show()
# The latent variable appears to control the "style" of the digit
```