CSE514A, 2019Fall **CourseProject1 Report** **Xiaoyao Wang 465943**

Intro

This report follows step by step, aiming to answer specifically the required questions in the instruction. A summary of what I learned from the project is offered at the end.

Step 1.1 & 1.2

I found three different ways to access MNIST dataset:

a. Download from Kaggle. This method provides 50,000 train samples.
b. Call directly from /content/sample document in Google Colab. This gives 19,999 train samples.
c. Import MNIST from keras datasets. This gives 70,000 train samples.

One useful way to expand training dataset is to combine the unlabeled test dataset to the unlabeled train dataset as the VAE model does not need the information of labels.

Further structuring with predictor: Aside from latent distribution, another output of encoder could be the ***predictor***, a dense network with two hidden layers that predicts the probabilities of MNIST image labels 0-9 from the mean of the variational Gaussian distribution. The predictor can be used for ***joint training*** of VAE model, additionally organizing the latent space w.r.t. certain continuous properties so that gradient-based navigation into regions of desired properties becomes possible. Refer to http://krasserm.github.io/2018/04/07/latent-space-optimization/ for more details.

Step 1.3.1

In my model construction, the variable for latent variables is defined by:

```
# Dimension of latent space
latent_dim = 8
```

Which is used to define the dimension of latent distribution/output of the encoder in creating the encoder:

```
t_mean = layers.Dense(latent_dim)(x)
t_log_var = layers.Dense(latent_dim)(x)
```

As I change the dimension of the latent variables (k) from 1, 2, 4, 6, 8, to 10, the output images generated by putting the same 100 test input into the VAE models are:

k=1

k=2

k=4

k=6

k=8

k=10

We can observe from the above images that the image quality of k=1 is the worst of six cases and the image has most vague numbers. The overall image quality improves as the dimension of latent variables becomes larger until k=8, though some samples out of 100 have different output numbers as k varies. Such improvement in image quality indicates that when k increases from 1 to 8, the VAE model becomes more complex, and the decrease in error from bias is larger than the increase in error from variance. However, as k goes from 8 to 10, the image quality deteriorates, indicating that the increase in error from variance is larger than the decrease in error from bias as the model complexity increases further and the overfitting problem appears.

Step 1.3.2

2 similarity functions are used:

1) Mean Squared Error

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

where m,n respectively represent the number of pixel rows and the number of pixel columns for both image I and K

A value of 0 for MSE indicates perfect similarity. A value greater than one implies less similarity and will continue to grow as the average difference between pixel intensities increases as well.
While the MSE is substantially faster to compute, it has the major drawback of (1) being applied globally and (2) only estimating the perceived errors of the image.

2) Structural Similarity Index

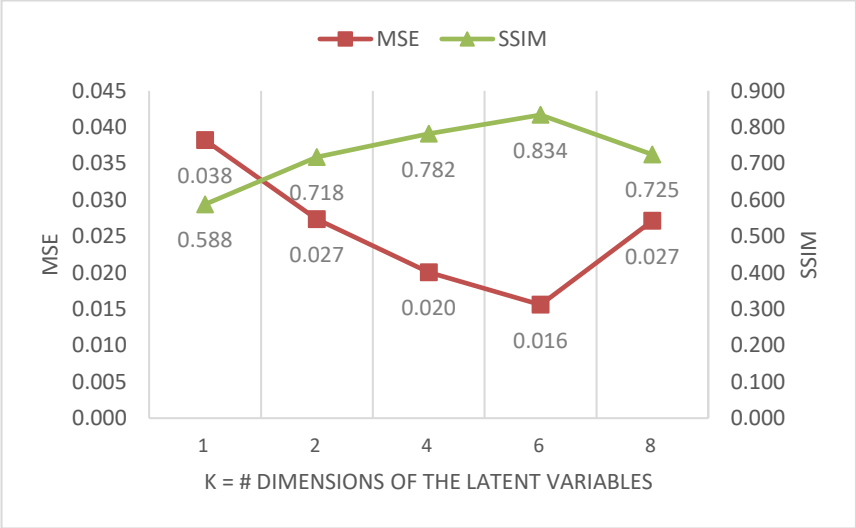
$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

The parameters include the (x, y) location of the N x N window in each image, the mean of the pixel intensities in the x and y direction, the variance of intensities in the x and y direction, along with the covariance.

SSIM can remedy some of the issues associated with MSE for image comparison. The value can vary between -1 and 1, where 1 indicates perfect similarity.
SSIM, while slower, is able to perceive the change in structural information of the image by comparing local regions of the image instead of globally.

Reference: https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/

Step 1.3.3



The trends of both MSE and SSIM correspond exactly to what we observe in image quality variations in Step 1.3.1. The optimal number of dimensions of the latent variables is k=8, at which the minimum MSE and maximum SSIM is obtained and we have the largest similarity between input images and output images from the VAE model.

Step 1.4

The number of layers is specified in both encoder and decoder parts. Take a model with three hidden layers at k=2 as an instance, we use three layers.Conv2D functions to define structures of each layer in the encoder:

```
def create_encoder():
    '''
    Creates a convolutional encoder model for MNIST images.

    - Input for the created model are MNIST images.
    - Output of the created model are the sufficient statistics
      of the variational distriution q(t|x;phi), mean and log
      variance.
    '''
    encoder_iput = layers.Input(shape=image_shape)

    x = layers.Conv2D(32, 3, padding='same', activation='relu')(encoder_iput)
    x = layers.Conv2D(64, 3, padding='same', activation='relu', strides=(2, 2))(x)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.Flatten()(x)
    x = layers.Dense(32, activation='relu')(x)

    t_mean = layers.Dense(latent_dim)(x)
    t_log_var = layers.Dense(latent_dim)(x)

    return Model(encoder_iput, [t_mean, t_log_var], name='encoder')
```

The decoder has a symmetric structure of the encoder:

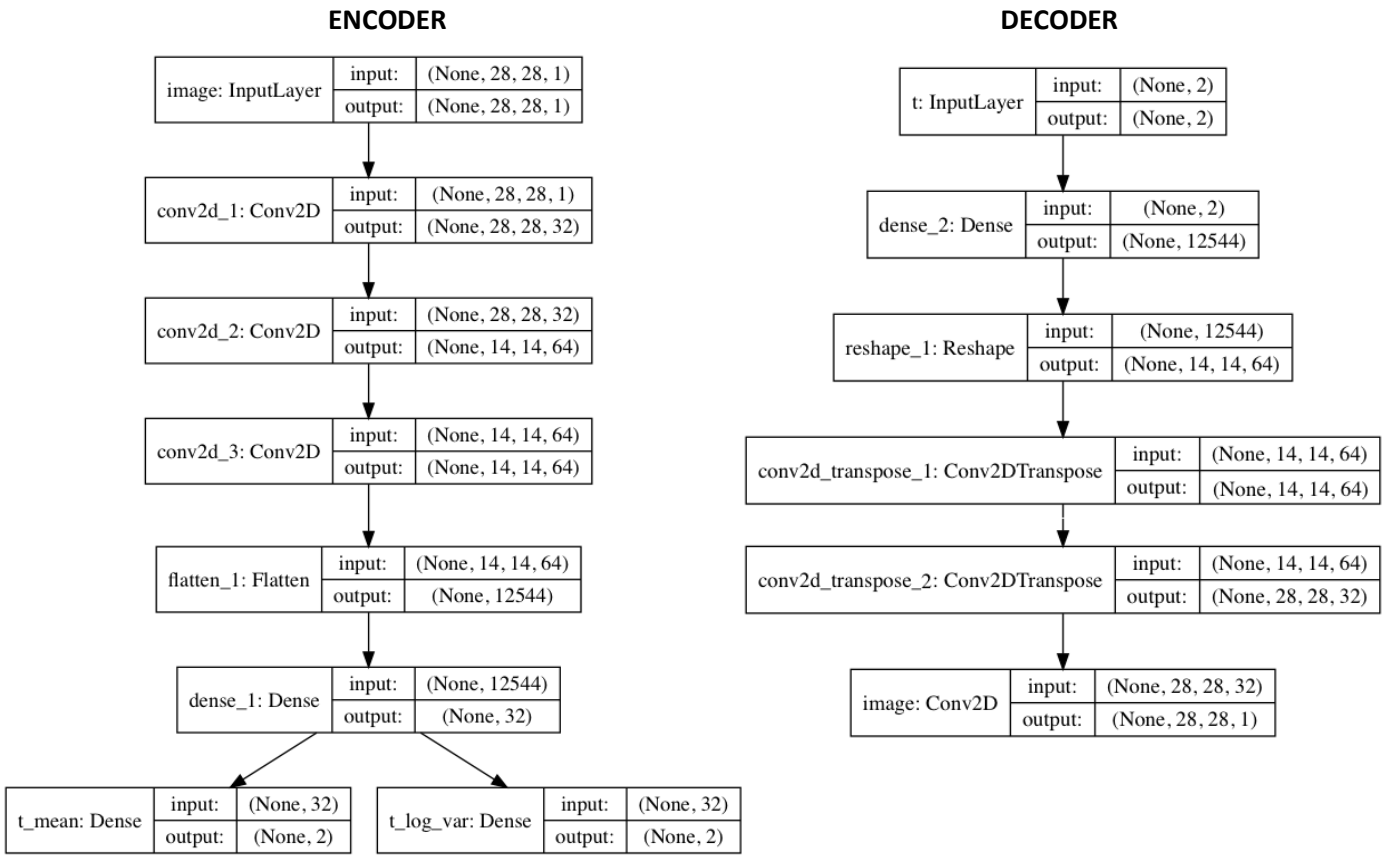
```
def create_decoder():
    '''
    Creates a (de-)convolutional decoder model for MNIST images.

    - Input for the created model are latent vectors t.
    - Output of the model are images of shape (28, 28, 1) where
      the value of each pixel is the probability of being white.
    '''
    decoder_input = layers.Input(shape=(latent_dim,))

    x = layers.Dense(12544, activation='relu')(decoder_input)
    x = layers.Reshape((14, 14, 64))(x)
    x = layers.Conv2DTranspose(64, 3, padding='same', activation='relu')(x)
    x = layers.Conv2DTranspose(32, 3, padding='same', activation='relu', strides=(2, 2))
    x = layers.Conv2D(1, 3, padding='same', activation='sigmoid')(x)

    return Model(decoder_input, x, name='decoder')
```

If we write down the structure using pen and paper:



As the number of layers (l) changes from 1, 2, 3 to 4, the output images are:

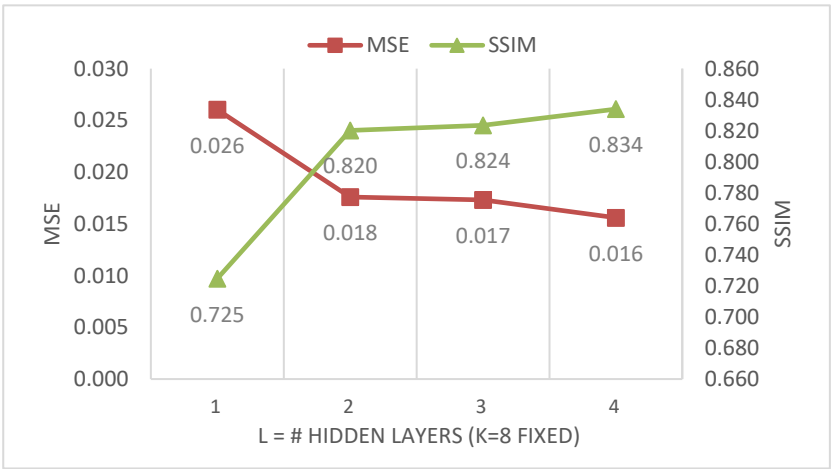
l=1

l=2

l=3

l=4

We can observe from above figures that the image quality improves when the number of layers goes from 1 to 2, but the difference of image quality is minor when l changes from 2, 3 to 4.



The trends of both MSE and SSIM correspond exactly to what we observe in changes of image quality above. As more layers are introduced in our VAE model, the model complexity increases, and the decrease of bias is larger than the increase in variance. However, we need to analyze the cost of more time-consuming computation and the benefit of better output image quality when we construct the VAE model because the improvement in image quality could be minor as the number of layers becomes larger further.

Summary

1) How does VAE capture image features using low dimensional latent variables?

VAE needs to answer **how to define the latent variables z** (i.e. decide what information they present). The information or image features could be very complicated: the digit, angle that the digit id drawn, the stroke width, abstract stylistic properties, etc. These properties may also be correlated through the latent structure between dimensions of z. VAE assumes that there is no simple interpretation of the dimensions of z, and instead assert that samples of z can be drawn from distribution N(0,I), where I id the identity matrix. The key is **that any distribution in d dimensions can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function**. In our MNIST project, the parameters of N(0,I) - [t_mean, t_log_var] for each dimension – is an output of encoder. VAE uses few layers to map the normally distributed z's to the latent values (like digit identity, stroke weight, angle, etc.) with exactly the right statistics (Mellon, 2016). We won't be able to actually interpret the dimensions, but it doesn't really matter.

2) What hyperparameters should we set before training?

There are 4 **hyperparameters that we need to set before training** VAE:

- Dimension of latent variables
- Number of layers
- Number of nodes per layer
- Loss function

3) How can we make VAE more powerful?

Reconstruction quality of images can be increased by choosing a **higher-dimensional latent space** and/or by **using encoder and decoder models with higher capacity** (more layers/more nodes per layer). The VAE model seems to be quite sensitive to the dimensionality of latent variables, but not to the number of hidden layers. We should also be careful that as the model complexity increases further, the increase in error from variance could lead to overfitting problems.