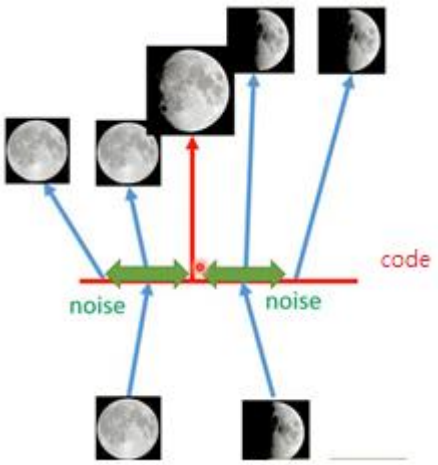
Hw8 Report

R09921026 李育倫

1. Make a brief introduction about variational autoencoder (VAE). List one advantage comparing with vanilla autoencoder and one problem of VAE.

   VAE 有別於一般的 autoencoder，它將圖片進入 encoder 後所得到的 code 上加上了 noise，如下圖，在兩個 code 中 noise 相交的地方為了 decode 出 MSE 最小的圖片那就非常可能產生出你所期待的四分之三弦月，這麼做是因為一般的 autoencoder 中為 non-linear 的 model，在 encode 出來的 code 中隨機 sample 一個點 decode 出來所得到的圖往往不是你所期望的，可能與真實圖片的樣子有一段落差，而 VAE 的技術就是改善了這個問題，也就是 VAE 在 code 中隨機 sample 一個點 decode 出來的圖片會比較好。

   但 VAE 會碰到的問題是他實際上並沒有學會如何重建出較好的圖片，也就是產生出能夠以假亂真的新圖片，而只是學會了盡可能透過模仿的方式，重建出與 database 中某張圖片相似的圖。



2. Train a fully connected autoencoder and adjust at least two different element of the latent representation. Show your model architecture, plot out the original image, the reconstructed images for each adjustment and describe the differences.

   (1) 將 encoder 出來的 tensor，加上 random uniform -10~10 的 noise，結果如下所示:

   由於加入 noise 的大小可能比 encoder 出來的 tensor 數值還大許多，造成 reconstructed image 非常詭異。

   (2) 將 encoder 出來的 tensor，同乘 25.5，結果如下所示:

   由於將 encoder 出來的 tensor 數值同乘 25.5，造成 reconstructed image 的顏色非常極端且有顏色飽合的情況。

```
class fcn_autoencoder(nn.Module):
    def __init__(self):
        super(fcn_autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(64 * 64 * 3, 256),
            # nn.BatchNorm1d(256),
            nn.ReLU(True),
            nn.Linear(256, 2048),
            # nn.BatchNorm1d(2048),
            nn.ReLU(True),
            nn.Linear(2048, 2048),
            nn.ReLU(True),
            nn.Linear(2048, 128),
        )


        self.decoder = nn.Sequential(
            nn.Linear(128, 2048),
            # nn.BatchNorm1d(2048),
            nn.ReLU(True),
            nn.Linear(2048, 2048),
            # nn.BatchNorm1d(2048),
            nn.ReLU(True),
            nn.Linear(2048, 256),
            nn.ReLU(True),
            nn.Linear(256, 64 * 64 * 3),
            nn.Tanh()
        )
```

(model architecture)

(1) 加上 random uniform -10~10 的 noise

```
def forward(self, x):
    x = self.encoder(x)

    x_perturb = x + random.uniform(-10, 10)
    x = self.decoder(x)
    x_perturb = self.decoder(x_perturb)
    return x, x_perturb
```

(2) 同乘 25.5

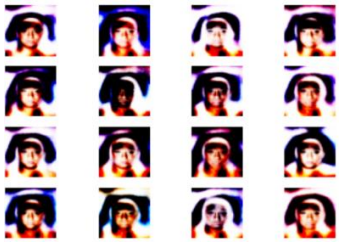
```
def forward(self, x):

    x = self.encoder(x)
    x_perturb = x * 25.5;
    x = self.decoder(x)
    x_perturb = self.decoder(x_perturb)
    return x, x_perturb
```

(1)

Input image:

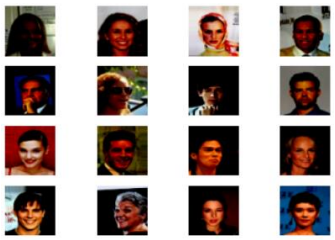
Reconstructed image:

Reconstructed image(adjusted):



(2)

Input image:



Reconstructed image:

Reconstructed image(adjusted):