Gradescope

t11902210 張一凡

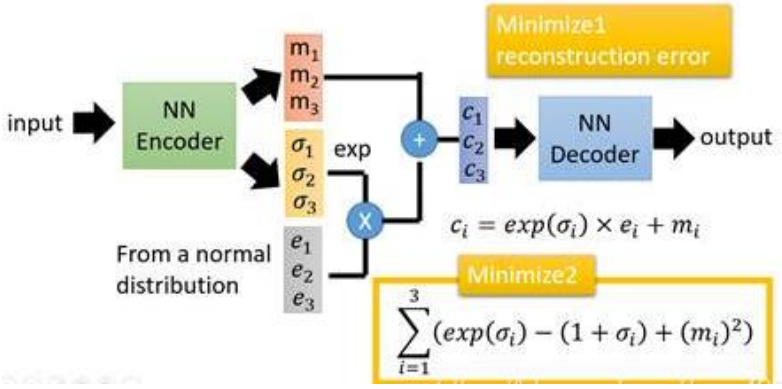
1.Choose a variation of autoencoder. Show an image of the model architecture. Then, list an advantage and a disadvantage comparing with vanilla autoencoder. Also, put on the paper link as reference. Eg, denoising autoencoder, variational autoencoder, etc.

Answer:

Variational Autoencoder (VAE)

Firstly, let's briefly introduce the VAE model: in an automatic encoder, the encoder directly generates the encoding. However, in the VAE model, by adding appropriate noise to the encoding, the encoder will output two types of encoding: one is the original encoding (m1, m2, m3), and the other is the encoding that controls the level of noise interference (σ 1,σ 2,σ 3). The second encoding is to assign weights to random noise codes (e1, e2, e3) and then add exp(σi) The purpose is to ensure that the assigned weights are positive. Finally, by adding the original encoding and noise encoding, the output results of VAE in the code layers (c1,c2,c3) are obtained. Other network architectures are the same as deep automatic encoders. In terms of the loss function, besides the necessary reconstruction loss, VAE also adds a loss function: in order to ensure the higher quality of the generated image, the encoder must hope that the noise will have less interference on the generated image, so the weight assigned to the noise will be smaller, so it only needs to (σ1,σ2,σ3) Simply assign a value close to negative infinity. Therefore, the second loss function has the effect of limiting the encoder to take this extreme path. It can be seen intuitively that exp (σi) - （1+ σi) In the σ In, the minimum value is obtained at i=0, therefore (σ1,σ2,σ3) It will avoid being assigned negative infinity.

The following is the image that I found after searching for many papers and materials, which I believe is the best to showcase the architecture of the VAE model.



Let's briefly introduce the vanilla autoencoder. In the simplest structure of this autoencoder, there are only three network layers, that is, a neural network with only one

hidden layer. Its input and output are the same. You can learn how to reconstruct the input by using the Adam optimizer and the mean square error loss function. The following are the advantages and disadvantages of comparing the VAE model in table form with the vanilla autoencoder.

No	Advantage (comparing with vanilla autoencoder)	Disadvantage (comparing with vanilla autoencoder)
1	The training complexity of the VAE model is higher, especially in terms of generating data augmentation. The advantage of VAE is that it can control the degree of change in the generated data. By manipulating dimensions in different directions in the latent space, selective changes can be made to the generated data. For example, interpolation or scaling operations can be performed on specific dimensions to generate data with specific attributes or varying degrees. This will enable VAE to learn potential variables in the data, thus generating brand new samples instead of simply repeating the input data, resulting in more realistic and realistic samples.	In this case, VAE can be used as a generative model to randomly sample from potential space and generate new data samples. However, at the same time, the increase in model complexity will directly lead to an increase in model computation time and space, which in some cases can lead to a trade-off. Vanilla autoencoders do not need to calculate KL divergence, and their training speed is faster compared to VAE, occupying less memory. At the same time, operations like VAE may generate samples that are different from the input data, because the generated samples are sampled from the potential space, rather than directly copying the input data, and the vanilla autoencoder does not need to worry about this.
2	The VAE model has a more continuous potential space, which allows it to perform interpolation operations in the potential space, and the generated images can transition more naturally. At the same time, it means that compared to other ordinary autoencoders, it can simultaneously process continuous and discrete data, and has a broader application scenario. The most direct example comparison is that the potential space of the vanilla autoencoder is discrete, so interpolation operations cannot be performed in the potential space, and the generated image is not natural enough. In this case, VAE can be used	Although the VAE model has potential continuity space and can deal with many complex continuity problems, it is difficult to explain the meaning of each dimension because the distribution of potential variables of VAE is normal distribution; And at the same time, VAE tends to generate fuzzy or ambiguous samples, because its generation process is random and cannot guarantee that each sample is of high quality. To the same extent, VAE usually has lower image quality than GAN. The potential space of the vanilla autoencoder is interpretable, allowing for a better understanding of the meaning of each dimension, which

	as a generative model and has wide applications in many fields, such as image synthesis, natural language generation, etc.	provides greater clarity for model understanding and practical application scenarios.
3	The VAE model is more robust to noise. The distribution of potential variables of the VAE model allows some randomness. VAE is to add appropriate noise to the code on the original AE structure and improve the integrity of the model through new loss function. This makes VAE more robust to noise and deformation in the input. This can better deal with uncontrollable factors such as noise, deformation and outlier in the input data, improve the reliability, stability and generalization ability of the model, and reduce the demand for data pre-processing and data pre-processing.	The VAE model not only increases robustness but also reduces time and space utilization, while sacrificing a certain degree of model accuracy. This is also part of the reason for the limited quality of the generated images in the VAE model. The vanilla autoencoder is specifically used for image processing and other operations, so the accuracy is higher than VAE to the same extent, but it is more susceptible to noise.

The following are the original papers and related links that I referenced.

```
Reference Paper 1:
Auto-Encoding Variational Bayes
Reference link 1:
```
https://arxiv.org/abs/1312.6114

```
Reference Paper 2:
Variational autoencoders
Reference link 2:
```
https://www.jeremyjordan.me/variational-autoencoders/

```
Reference website 3:
Analysis of Variational Autoencoder (VAE)
Reference link 3:
```
https://blog.csdn.net/qq_43753525/article/details/110864755?ops_request_misc=&request_id=&biz_id=102&utm_term=VAE%E7%9A%84%E4%BC%98%E7%BC%BA%E7%82%B9&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-9-110864755.142^v86^insert_down1,239^v2^insert_chatgpt&spm=1018.2226.3001.4187

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.

Answer:

I first demonstrated the self encoder structure I used. I generally used the fcn structure given in hw8, but the difference is that I modified the parameters in the linear in many layers and expanded them to 4-8 times. By increasing the number of input and output features, I made the fcn model more refined, increased the complexity of the model, and improved the final output effect. At the same time, this can also make the distinction between encoder and decoder in this question more obvious, making it easier to answer questions

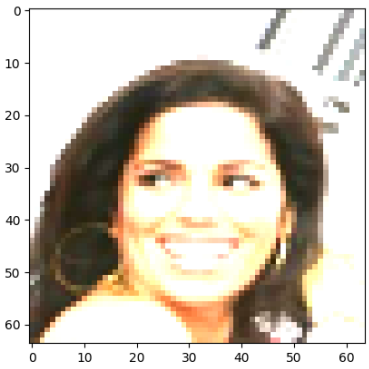
```
class fcn_autoencoder(nn.Module):
    def __init__(self):
        super(fcn_autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(64 * 64 * 3, 1024),
            nn.ReLU(),
            nn.Linear(1024, 256),
            nn.ReLU(),
            nn.Linear(256, 64),
            nn.ReLU(),
            nn.Linear(64, 10)
        )  # Hint: dimension of latent space can be adjusted

        self.decoder = nn.Sequential(
            nn.Linear(10, 64),
            nn.ReLU(),
            nn.Linear(64, 256),
            nn.ReLU(),
            nn.Linear(256, 1024),
            nn.ReLU(),
            nn.Linear(1024, 64 * 64 * 3),
            nn.Tanh()
        )

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

Next, I will draw the original image according to the requirements of the title. For each adjustment of the reconstructed image, I will first display the original image and the normal output image, and then attach the code for two reconstructions to observe the output image and provide an explanation.

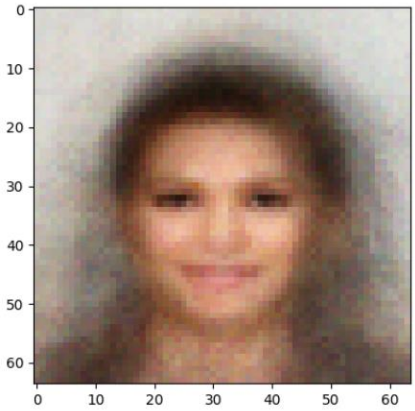
The most original image



The most primitive images are directly extracted from the dataset during the training process. We need to use the FCN model to manually encode and decode them in the following process and observe the results under different adjustment states. The following is the program code for generating the most primitive images.

```
#  ==================loading====================
img  =  data.float().cuda()
#  Original  image
origin  =  img[0].detach().cpu().numpy().reshape(3,  64,  64).transpose(1,  2,  0)  +  1
plt.imshow(origin)
plt.savefig("origin.png")
```

Output results from normal use of the model



The second display is the final image output by applying the normal model with output=model (img). It can be seen that the normal output color zone is clearly displayed and it can be seen that it is an image of a girl, and the image is relatively dark. Although there is still a gap with the original dataset image shown above, many basic and obvious features can be seen from this image.

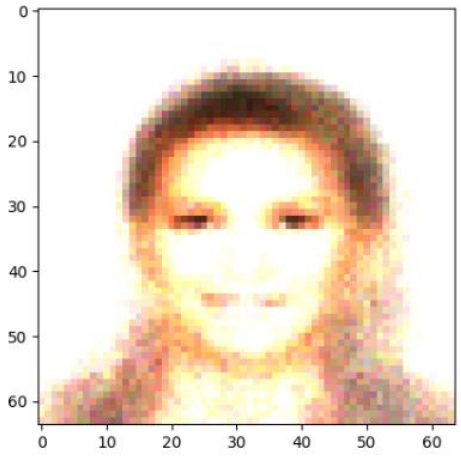
```
#  ==================forward====================
output = model(img)
output_origin  =  output[0].detach().cpu().numpy().reshape(3,  64,  64).transpose(1,  2,  0)  +  1
output_origin  =  (output_origin  *  255.  /  2.).astype(int)
plt.imshow(output_origin)
plt.savefig("output.png")
```

The above shows the basic data preparation for comparison, followed by all adjustments, results display, and analysis explanations related to this question.

（1）Double the zero dimensional data

First of all, I made adjustments to the zero dimensional value after the encoder. The following is the code for my adjustments and output of the image. I first copied img to img1, input img1 into the encoder, and assign a value to z1. For the zero dimensional value of the image in z1, I double it. This is also an example program given in the homework lesson, and the results are also shown below.

```
if model_type in ['fcn']:
        img = img.view(img.shape[0], -1)
        #change 1
        img1 = img
        z1 = model.encoder(img1)
        z1[0][0] *= 2
        output1 = model.decoder(z1)
        output1_img = output1[0].detach().cpu().numpy().reshape(3, 64, 64).transpose(1, 2, 0)
        plt.imshow(output1_img)
        plt.savefig("output1.png")
```

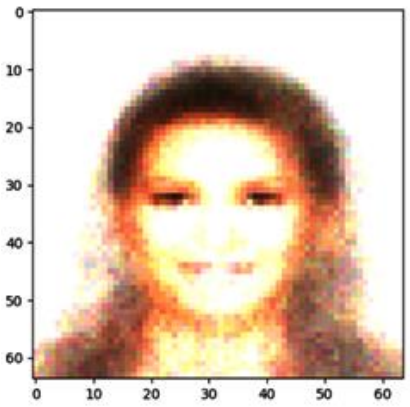


From the newly output model, we can see that the most direct change is that the hair of the characters in the image is less than that in the normal output, and we can hardly see female features from this image when we know that the original result is a female. At the same time, after searching for relevant information, I came to the conclusion that the zero dimensional data in this z is gender controlled. The larger the data, the more biased it is towards men, while the smaller the data, the more biased it is towards women. This is also my first analysis and conclusion made by adjusting the late.

（2）Increase the first dimension by a certain amount

I imitated the first adjustment in the second adjustment, but this time I added the first dimensional data after the image encoder, which is equivalent to slightly increasing its value. The specific code implementation and output results are also shown below.

```
#change 2
img2 = img
z2 = model.encoder(img2)
z2[0][2] += 5
output2 = model.decoder(z2)
output2_img = output2[0].detach().cpu().numpy().reshape(3, 64, 64).
plt.imshow(output2_img)
plt.savefig("output2.png")
```



This contrast is not obvious from the original output, but it can be compared with the last adjusted one. The shoulders on both sides of this picture can be significantly lower to see the color deepening, which indicates that the first dimension data is adjusted for color depth, and the larger the data, the darker the color. After collecting relevant information, I found another possibility that this dimension controls the angle or intensity of light, which is also explained through this comparison.