Neural Networks and Deep Learning 2021-22

Menglu Tao

Feb 2022

1 Introduction

1.1 Homework Goals

The goal of this homework is to implement and test an autoencoder model, and use it to perform image reconstruction tasks.

1.2 Main Implementation Strategies

In this homework, I used pytorch and matlibplot packages to build an autoencoder model. And using MSEloss function to evaluate the result.

2 Method

2.1 Model architecture

```
class Encoder(nn.Module):
    def __init__(self, encoded_space_dim):
        super().__init__()

        self.encoder_cnn = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=8, kernel_size=3,
                      stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,
                      stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,
                      stride=2, padding=0),
            nn.ReLU(True)
        )
        self.flatten = nn.Flatten(start_dim=1)
        self.encoder_lin = nn.Sequential(
            nn.Linear(in_features=3*3*32, out_features=64),
            nn.ReLU(True),
            nn.Linear(in_features=64, out_features=encoded_space_dim)
        )

    def forward(self, x):
        x = self.encoder_cnn(x)
        x = self.flatten(x)
        x = self.encoder_lin(x)
        return x
```

Figure 1: Encoder Model

```
class Decoder(nn.Module):
    def __init__(self, encoded_space_dim):

        super().__init__()
        self.decoder_lin = nn.Sequential(
            nn.Linear(in_features = encoded_space_dim, out_features=64),
            nn.ReLU(True),
            nn.Linear(in_features=64,out_features=3*3*32),
            nn.ReLU(True)
        )

        self.unflatten = nn.Unflatten(dim=1,unflattened_size=(32, 3, 3))
        self.decoder_conv = nn.Sequential(
            nn.ConvTranspose2d(in_channels=32, out_channels=16, kernel_size=3,
                               stride=2, output_padding=0),
            nn.ReLU(True),
            nn.ConvTranspose2d(in_channels=16, out_channels=8, kernel_size=3,
                               stride=2, padding=1, output_padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(in_channels=8, out_channels=1, kernel_size=3,
                               stride=2, padding=1, output_padding=1)


        )
    def forward(self,x):
        x = self.decoder_lin(x)
        x = self.unflatten(x)
        x = self.decoder_conv(x)
        x = torch.sigmoid(x)
        return x
```

Figure 2: Decoder Model

This autoencoder model is made up by an encoder and a decoder model. In encoder model, there are three convolutional layers and two activation functions. In decoder model, there are three convolutional layers and two activation functions too.

2.2 Hyperparameters

Hyperparameters are as follows: Batch size is 256, Number of epoches is 10, Learning rate is 5e-4, Encoded feature dimension is 2, Loss function is MSELoss() Optimizer is Adam.

3 Result

3.1 Autoencoder Result

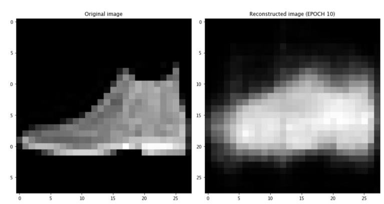

Figure 3: Reconstructed Image Compare with Original One

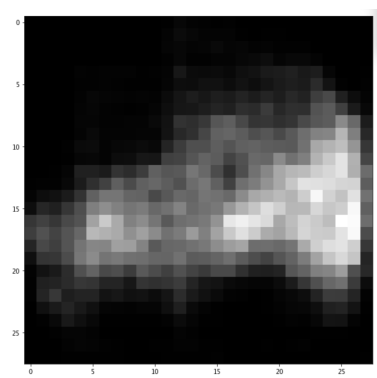

Figure 4: Image generated from latent space

The validation loss result is 0.032616,

3.2 Implementation of VAE

```python
class VAE(nn.Module):
    def __init__(self, x_dim, h_dim1, h_dim2, z_dim):
        super(VAE, self).__init__()

        # encoder part
        self.fc1 = nn.Linear(x_dim, h_dim1)
        self.fc2 = nn.Linear(h_dim1, h_dim2)
        self.fc31 = nn.Linear(h_dim2, z_dim)
        self.fc32 = nn.Linear(h_dim2, z_dim)
        # decoder part
        self.fc4 = nn.Linear(z_dim, h_dim2)
        self.fc5 = nn.Linear(h_dim2, h_dim1)
        self.fc6 = nn.Linear(h_dim1, x_dim)

    def encoder(self, x):
        h = F.relu(self.fc1(x))
        h = F.relu(self.fc2(h))
        return self.fc31(h), self.fc32(h) # mu, log_var

    def sampling(self, mu, log_var):
        std = torch.exp(0.5*log_var)
        eps = torch.randn_like(std)
        return eps.mul(std).add_(mu) # return z sample

    def decoder(self, z):
        h = F.relu(self.fc4(z))
        h = F.relu(self.fc5(h))
        return F.sigmoid(self.fc6(h))

    def forward(self, x):
        mu, log_var = self.encoder(x.view(-1, 784))
        z = self.sampling(mu, log_var)
        return self.decoder(z), mu, log_var

# build model
vae = VAE(x_dim=784, h_dim1= 512, h_dim2=256, z_dim=2)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f'Selected device: {device}')
```

Figure 5: VAE Model