Report

2016025196

김동규

1. 코드 설명

Discriminator

```
class Discriminator(nn.Module):
    # 모델의 코드는 여기서 작성해주세요

    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1=nn.Conv2d(3,16,5,1,1)
        self.pool=nn.MaxPool2d(2,2)
        self.conv2=nn.Conv2d(16,32,3,1,1)
        self.conv3=nn.Conv2d(32,64,3,1,1)
        #64*7*7
        #32*15*15
        self.fc1=nn.Linear(64*7*7, 16*16)
        self.fc2=nn.Linear(16*16,1)
        self.activation=nn.LeakyReLU(0.2)
        self.sigmoid=nn.Sigmoid()
        self.dropout=nn.Dropout(0.3)


    def forward(self, input):
        x1=self.pool(self.activation(self.conv1(input)))
        x1=self.dropout(x1)
        x2=self.pool(self.activation(self.conv2(x1)))
        x2=self.dropout(x2)
        x3=self.pool(self.activation(self.conv3(x2)))
        x3=self.dropout(x3)
        #print(x3.shape)
        x_flatten=torch.flatten(x3,1)
        #print(x_flatten.shape)
        x4=self.activation(self.fc1(x_flatten))
        x4=self.dropout(x4)
        output=self.sigmoid(self.fc2(x4))


        return output
```

총 3개의 convolution network와 2개의 fully connected network로 구성되어 있다. 4단계 layer까지는 leaky relu함수를 이용하였으며, 마지막 layer에서는 sigmoid함수를 사용하였다. Drop out은 0.3이다.

main함수

```
if __name__ == "__main__":
    # 학습코드는 모두 여기서 작성해주세요


    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5),(0.5))
        ])



    data_path = 'training_data/'

    dataset = datasets.ImageFolder(root=data_path,
                                   transform=transform
                                   )
    print(torch.cuda.is_available())
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

이미지 특성상 normalize를 하기 위해 tranform에 nomalize를 추가했다.

```
if True:
    generator = Generator().to(device)
    discriminator = Discriminator().to(device)
else:
    generator = Generator()
    discriminator = Discriminator()
    model=torch.load('models/wieght15.pt')
    generator.load_state_dict(model['generator'])
    generator.to(device)
    discriminator.load_state_dict(model['discriminator'])
    discriminator.to(device)
```

학습 및 결과 확인을 위해 사용한 코드로 True를 False로 고쳐 기존에 학습한 모델을 불러오는 기능을 구현했다. True일경우 새로운 모델을 생성하여 학습한다.

```
criterion=torch.nn.BCELoss()
g_optimizer=torch.optim.Adam(generator.parameters(),lr=0.001,betas=(0.9,0.999))
d_optimizer=torch.optim.Adam(discriminator.parameters(), lr=0.001,betas=(0.9,0.999))

epochs=75
total_batch_num=len(dataset)
batch_size=64
```

Loss function은 BCE, optimizer는 ADAM을 사용하였으며, 총 epoch 75,rbatch_size 64로 실험을 진행했다.

```
for epoch in range(epochs):
    print(epoch)
    generator.train()
    discriminator.train()
    avg_g_cost=0
    avg_d_cost=0
    for step,batch in enumerate(train_dataloader):
        if step%(160*5)==0:
            print(str(step*batch_size)+" / "+str(total_batch_num))
        b_x,_=batch
        b_x=b_x.to(device)

        #b_x=b_x.view(3,64,64).to(device)
        num_img=len(b_x)
        real_label=torch.ones((num_img,1)).to(device)
        fake_label=torch.zeros((num_img,1)).to(device)

        real_logit=discriminator(b_x)
        d_real_loss=criterion(real_logit,real_label)

        z=torch.randn((num_img,100,1,1),requires_grad=False).to(device)

        fake_data=generator(z)
        fake_logit=discriminator(fake_data)
        d_fake_loss=criterion(fake_logit,fake_label)


        z=torch.randn((num_img,100,1,1),requires_grad=False).to(device)

        fake_data=generator(z)
        fake_logit=discriminator(fake_data)
        d_fake_loss=criterion(fake_logit,fake_label)

        d_loss=d_real_loss+d_fake_loss
        d_optimizer.zero_grad()
        d_loss.backward()
        d_optimizer.step()

        z=torch.randn((num_img,100,1,1),requires_grad=False).to(device)
        fake_data=generator(z)
        fake_logit=discriminator(fake_data)
        g_loss=criterion(fake_logit,real_label)
        d_optimizer.zero_grad()
        g_optimizer.zero_grad()
        g_loss.backward()
        g_optimizer.step()

        avg_d_cost+=d_loss
        avg_g_cost+=g_loss
    avg_d_cost/=total_batch_num
    avg_g_cost/=total_batch_num
    print(avg_d_cost.item(),avg_g_cost.item())
```

주요 학습 코드로, 실제데이터, 가짜데이터 및 0라벨, 가짜데이터 및 1라벨 순서

로 생성하며 학습한다.

```
if False:
    if epoch%10==0:
        if epoch!=0:
            print("save model and fake image")
            torch.save({'generator' : generator.state_dict(),
            'discriminator': discriminator.state_dict()},'models/wieght'+str(int(epoch)+15)+'.pt')
            generator.eval()
            test_noise = torch.randn(3000, 100, 1, 1, device=device)
            with torch.no_grad():
                test_fake = generator(test_noise).detach().cpu()

                for index, img in enumerate(test_fake):
                    fake = np.transpose(img.detach().cpu().numpy(), [1, 2, 0])
                    fake = (fake * 127.5 + 127.5).astype(np.uint8)
                    im = Image.fromarray(fake)
                    im.save("./fake_img{}/fake_sample{}.jpeg".format(int(epoch)+15,index))
```

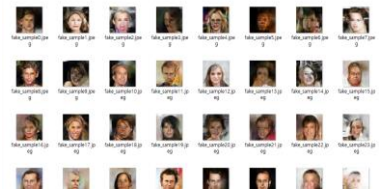
마지막으로 모델 저장코드로 10 epochs마다 모델을 저장한다.

2. 실험결과

초기에는 convolution network로 conv2d(3,64,3)로 시작하는 discriminator를 구성하였다. 그러나 이는 학습시간이 너무 오래 걸리는 문제점으로 인해 (3,16,5), (16,32,3), (32,64,3)으로 구성하고, 1epoch에 15분정도 소요되게 되었다.

이후, 정상동작을 확인하기 위해 15번까지의 epoch를 돌린 결과, 사람의 얼굴과는 전혀 다른 방향으로 fake data가 형성되는 것을 확인했다.

이에 따라, transform normalize추가, dropout추가를 한 결과이다.

 - epoch 5

5번 epoch를 돌린 결과부터 학습이 잘 되고 있다는 것을 확인하고, model을 저장하는 코드를 추가하여 75회까지 10회마다 모델과 fake data를 저장하였다.

그 결과, fake data는 아직 불안정하지만 fid score는 44점이 나오는 것을 확인했다.