Lab 7 Report Gan NF

電控碩一 309512074 黃柏叡

Introduction

這次作業有兩個task

task1: 利用conditional GAN以及conditional Normalizing Flow訓練一個可以生成指定條件圖片的模型，training dataset 為ICLEVR的幾何物體圖片，總共有24種不同的幾何物體，因此condition為24-dimension的vector，ex. [0,0,0,1,0,0,1,...,0,0,0]

task2: 利用conditional Normalizing Flow訓練一個可以生成人臉圖像的模型，training dataset為CelebA-HQ的人臉照片，總共有40種不同的特徵，若這張找片有此屬性則為1，反之為-1，ex. [-1, 1, 1, -1 -1, ..., -1,1 -1]

其中要完成3項任務

- Conditional face generation: 根據condition產生有相對應特徵的人像圖片
- Linear interpolation: 選取兩張照片，利用線性內插疊合
- Attribute manipulation: 選取特徵(ex. smiling)，計算mean vector z_{pos} (smiling) and z_{neg} (without smiling), then pick a image, and modify its attribute using vector z_{pos} - z_{neg} with different scaling

Implementation details

- Describe how I implement my model
  - cGan

  使用conditional DCGAN作為model架構

  Generator會把condition vector和latent z(100-dim)concat在一起，而condition vector會先過一層fully connected layer從24-dim變為300-dim來擴充資訊，最後再做5次ConvTranspose生成fake images，再做ConvTranspose時使用BatchNormalized以及ReLU

```
class Generator(nn.Module):
    def __init__(self,z_dim,c_dim):
        super(Generator,self).__init__()
        self.z_dim=z_dim
        self.c_dim=c_dim
        self.conditionExpand=nn.Sequential(
            nn.Linear(24,c_dim),
            nn.ReLU()
        )
        kernel_size=(4,4)
        channels=[z_dim+c_dim,512,256,128,64]
        paddings=[(0,0),(1,1),(1,1),(1,1)]
        for i in range(1,len(channels)):
            setattr(self,'convT'+str(i),nn.Sequential(
                nn.ConvTranspose2d(channels[i-1],channels[i],kernel_size,stride=(2,2),padding=paddings[i-1]),
                nn.BatchNorm2d(channels[i]),
                nn.ReLU()
```

```
            ))
        self.convT5=nn.ConvTranspose2d(64,3,kernel_size,stride=(2,2),padding=(1,1))
        self.tanh=nn.Tanh()

    def forward(self,z,c):
        """
        :param z: (batch_size,100) tensor
        :param c: (batch_size,24) tensor
        :return: (batch_size,3,64,64) tensor
        """
        z=z.view(-1,self.z_dim,1,1)
        c=self.conditionExpand(c).view(-1,self.c_dim,1,1)
        out=torch.cat((z,c),dim=1)  # become(N,z_dim+c_dim,1,1)
        out=self.convT1(out)  # become(N,512,4,4)
        out=self.convT2(out)  # become(N,256,8,8)
        out=self.convT3(out)  # become(N,128,16,16)
        out=self.convT4(out)  # become(N,64,32,32)
        out=self.convT5(out)  # become(N,3,64,64)
        out=self.tanh(out)    # output value between [-1,+1]
        return out

    def weight_init(self,mean,std):
        for m in self._modules:
            if isinstance(self._modules[m], nn.ConvTranspose2d) or isinstance(self._modules[m], nn.Conv2d):
                self._modules[m].weight.data.normal_(mean, std)
                self._modules[m].bias.data.zero_()
```

Discriminator會把24-dim的condition vector經過fully connected layer在reshape成(1*64*64)的維度，目的也是為了擴充資訊，再與training data(real image)或Generator的圖片(fake image)做concat變成(4*64*64)，最後再做5次Conv得到一個scalar，再做Convolution時使用BatchNormalized以及LeakyReLU

output代表是否為real image的scalar，因此選用 binary cross entropy作為loss function

```
class Discriminator(nn.Module):
    def __init__(self,img_shape,c_dim):
        super(Discriminator, self).__init__()
        self.C,self.H,self.W=img_shape
        self.conditionExpand=nn.Sequential(
            nn.Linear(24,self.H*self.W*1),
            nn.LeakyReLU()
        )
        kernel_size=(4,4)
        channels=[4,64,128,256,512]
        for i in range(1,len(channels)):
            setattr(self,'conv'+str(i),nn.Sequential(
                nn.Conv2d(channels[i-1],channels[i],kernel_size,stride=(2,2),padding=(1,1)),
                nn.BatchNorm2d(channels[i]),
                nn.LeakyReLU()
            ))
        self.conv5=nn.Conv2d(512,1,kernel_size,stride=(1,1))
        self.sigmoid=nn.Sigmoid()

    def forward(self,X,c):
        """
        :param X: (batch_size,3,64,64) tensor
        :param c: (batch_size,24) tensor
        :return: (batch_size) tensor
        """
```

```
        c=self.conditionExpand(c).view(-1,1,self.H,self.W)
        out=torch.cat((X,c),dim=1)  # become(N,4,64,64)
        out=self.conv1(out)  # become(N,64,32,32)
        out=self.conv2(out)  # become(N,128,16,16)
        out=self.conv3(out)  # become(N,256,8,8)
        out=self.conv4(out)  # become(N,512,4,4)
        out=self.conv5(out)  # become(N,1,1,1)
        out=self.sigmoid(out)  # output value between [0,1]
        out=out.view(-1)
        return out

    def weight_init(self,mean,std):
        for m in self._modules:
            if isinstance(self._modules[m], nn.ConvTranspose2d) or isinstance(self._modules[m], nn.Conv2d):
                self._modules[m].weight.data.normal_(mean, std)
                self._modules[m].bias.data.zero_()
```

在訓練時，會希望generator loss越小越好，而discriminator loss中的loss_fake越大越好，為了使training符合這個趨勢，將generator與discriminator的訓練次數比例條為4:1，使loss可以符合這個趨勢

```
for epoch in range(1, epochs+1):
        total_loss_D = 0
        total_loss_G = 0
        for size, (images, condition) in enumerate(dataloader):
            D_model.train()
            G_model.train()
            batch_size = len(images)
            images = images.to(device, dtype=torch.float)
            # print(type(condition))
            condition = condition.to(device, dtype=torch.float)
            # print(type(condition))
            real = torch.ones(batch_size).to(device)
            fake = torch.zeros(batch_size).to(device)

            ##train discriminator
            optimizer_D.zero_grad()

            ##real images
            predict = D_model(images, condition)
            loss_real = criterion_D(predict, real)

            ##fake images
            latent_z = random_z(batch_size, z_dim).to(device)
            gen_imgs = G_model(latent_z, condition)
            predict = D_model(gen_imgs.detach(), condition)
            loss_fake = criterion_D(predict, fake)

            ##back propagation
            loss_D = loss_real+loss_fake
            # print(loss_D)
            loss_D.backward()
            optimizer_D.step()

            ##train generator
            for _ in range(4):
                optimizer_G.zero_grad()
                latent_z = random_z(batch_size, z_dim).to(device)
                gen_imgs = G_model(latent_z, condition)
                predict = D_model(gen_imgs, condition)
                loss_G = criterion_G(predict, real)
```

```
            ##bp
            loss_G.backward()
            optimizer_G.step()
```

- conditional NF

  使用conditional Glow作為model架構

  task1:

  參考https://github.com/5yearsKim/Conditional-Normalizing-Flow這個repo

  其中repo中的condition為image，而我們的task的condition為24-dim的vector，因此我對condition先做fully connected再做reshape使condition的維度符合glow的model, 同時調整glow model中的參數(K,L,C)來嘗試使evaluate的效果進步

  task2:

  參考https://github.com/y0ast/Glow-PyTorch這個repo

  將z2逐漸加上(z1-z2)的value，生成的圖片會從z2開始逐漸變成z1

```
def interpolations(z1, z2, n):
  # print('interpolations input size',z1.size())
  z_list = torch.Tensor([]).cuda()
  for j in range(n):
    top = z1
    down = z2
    value = down + 1.0 * j*(top-down)/n
    z_list = torch.cat((z_list,value.unsqueeze(0)),0)
    # print('interpolations output size', z_list.size())
  return z_list
```

  當label中有這項特徵時，存入z_pos, 反之存入z_neg，而特徵選取的資料集我是選取前20000(batch_size=4)筆資料的特徵再做平均，最後再做內插時，若已有該項特徵，則和z_neg做內插，若沒有該項特徵，則與z_pos做內插，最後將latent z丟入model生成圖片

```
with torch.no_grad():
    for i,(x,y) in enumerate(test_loader):
      print('reading data: ',i)
      if i >= 5000: break
      for j,attribute_num in enumerate(attribute_list):
        x = x.to(device)
        y = y.to(device)
        z, bpd, y_logits = model(x, y_onehot=y)  # return: z, bpd, y_logits

        for k in range(len(z)):
          if  y[k][attribute_num] == 1:
            z_pos_list[j] = torch.cat((z_pos_list[j],z[k].unsqueeze(0)),0)
          else:
            z_neg_list[j] = torch.cat((z_neg_list[j],z[k].unsqueeze(0)),0)
    ### z_pos_list size: ( N_attribute * N_pos * 48 * 8 * 8 )

    z_pos_mean = torch.Tensor([]).cuda()
```

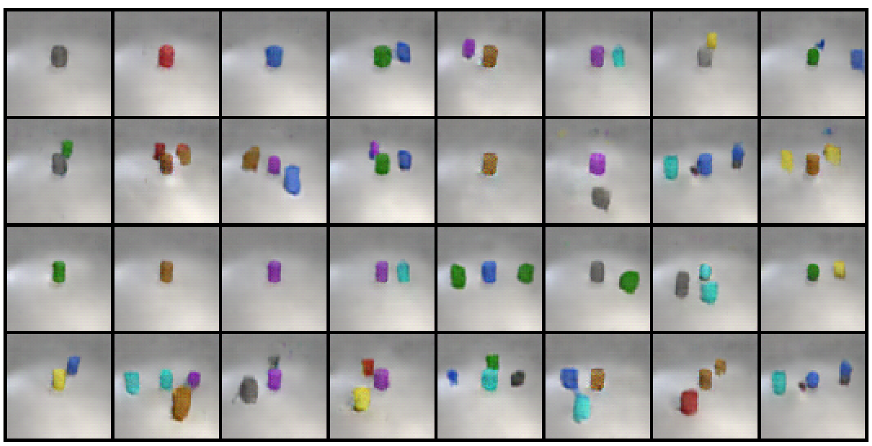
```
    z_neg_mean = torch.Tensor([]).cuda()
    for i in range(len(attribute_list)):
      pos_mean = torch.mean(z_pos_list[i], 0) #pos_mean size: torch.Size([48, 8, 8])
      z_pos_mean = torch.cat((z_pos_mean, pos_mean.unsqueeze(0)), 0)
      neg_mean = torch.mean(z_neg_list[i], 0)
      z_neg_mean = torch.cat((z_neg_mean, neg_mean.unsqueeze(0)), 0)
    ### z_pos_mean size: torch.Size([N_attribute, 48, 8, 8])

    z_input_img = z[0].clone()
    y_input_img = y[0].clone()
    y_rand = torch.rand(40).cuda()
    generate_images = torch.Tensor([]).cuda()
    for i,attribute_num in enumerate(attribute_list):
      if y_input_img[attribute_num] == 1:
        inters = interpolations(z_input_img, z_neg_mean[i], n=N)
      else:
        inters = interpolations(z_pos_mean[i], z_input_img, n=N)
      ### inters size: torch.Size([N, 48, 8, 8]), N=num of interpolations

      for n in range(N):
        z_for_generate = inters[n].unsqueeze(0)
        y_for_generate = y_input_img.clone().unsqueeze(0)
        predict_x = model(y_onehot=y_for_generate, z=z_for_generate, temperature=1, reverse=True)
        generate_images = torch.cat((generate_images,predict_x), 0)
        print('generate_images',generate_images.size())

    ### generate_x_list size: torch.Size([N_attribute*N_interpolation, 3, 64, 64])

    save_image(generate_images, 'images/Attribute_manipulation.png', normalize=True)
```

- Specify the hyperparameters
  - cGan
    - z_dim=100
    - c_dim=300
    - image_shape=(C,H,W)=(3,64,64)
    - epochs=500
    - learning_rate_generator=0.0001
    - learning_rate_discriminator=0.0004
    - batch_size=64
  - cNF(task1)
    - batch_size=16
    - learning_rate=0.0002
    - channels=512
    - levels(L)=4
    - steps(K)=6

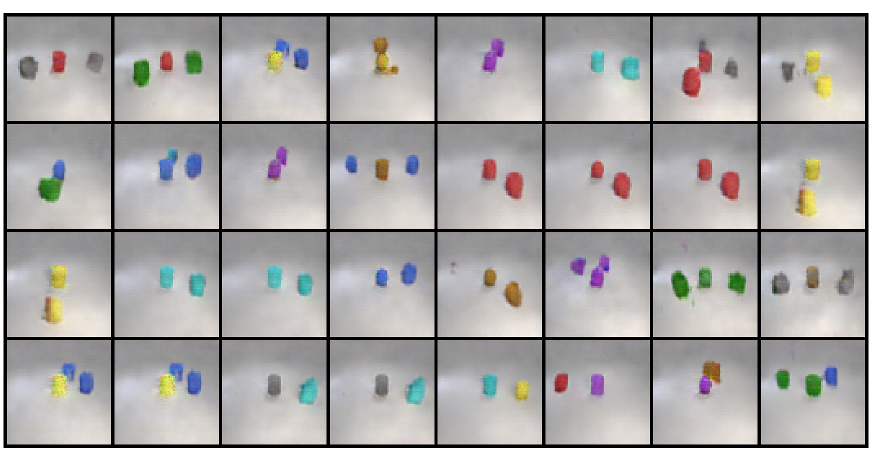
- epochs=500
- cNF(task2)
  - batch_size=16
  - learning_rate=0.0005
  - epochs=50
  - channels=512
  - K=6
  - L=3

Results and discussion

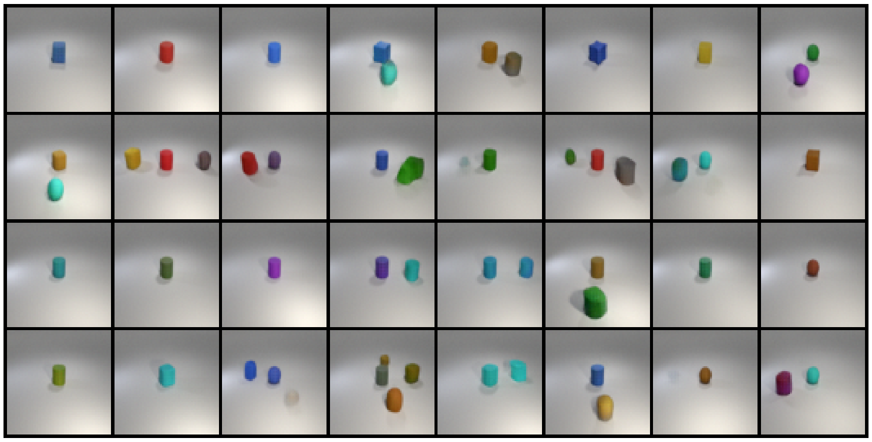
- Task1
  - Results
    - cGan test.json images



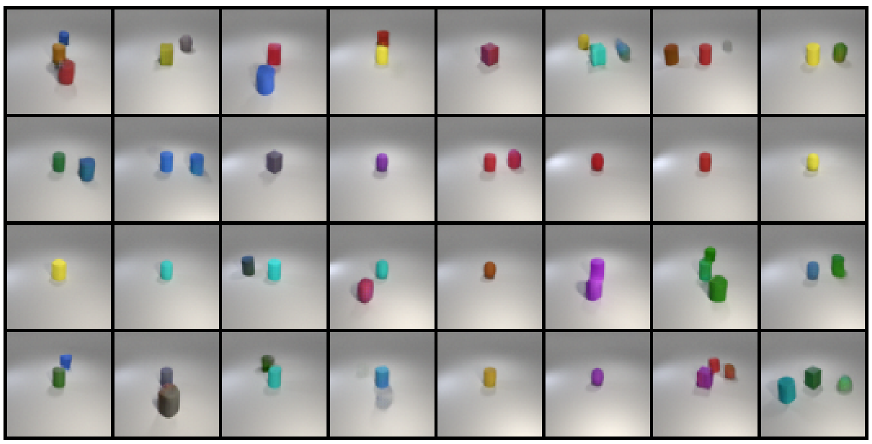
- cGan new_test.json images



- cNF test.json images



- cNF new_test.json images



- Classification accuracy
  - cGan test.json

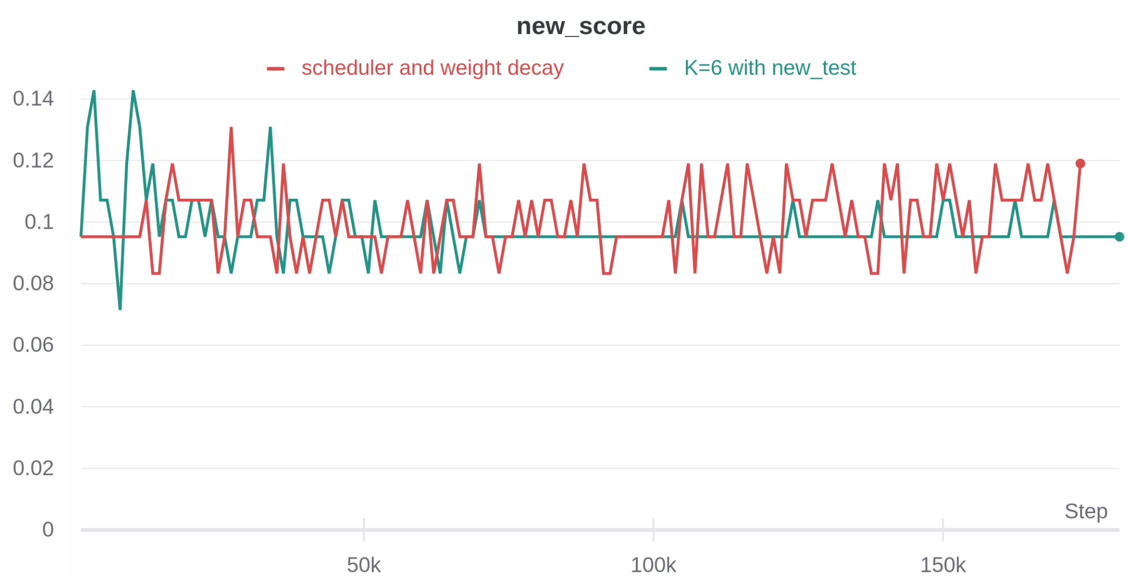
```
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ python3 evaluate_model.py --test_mode test
score: 0.66
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ python3 evaluate_model.py --test_mode test
score: 0.65
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ python3 evaluate_model.py --test_mode test
score: 0.66
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ 
```

  - cGan new_test.json

```
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ python3 evaluate_model.py --test_mode new_test
score: 0.70
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ python3 evaluate_model.py --test_mode new_test
score: 0.71
^[[Aray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ python3 evaluate_model.py --test_mode new_test
score: 0.71
ray@ray-ubuntu:~/Deep-Learning-and-Practice/HW7/dataset/task_1$ 
```

  - cNF test.json

- cNF new_test.json



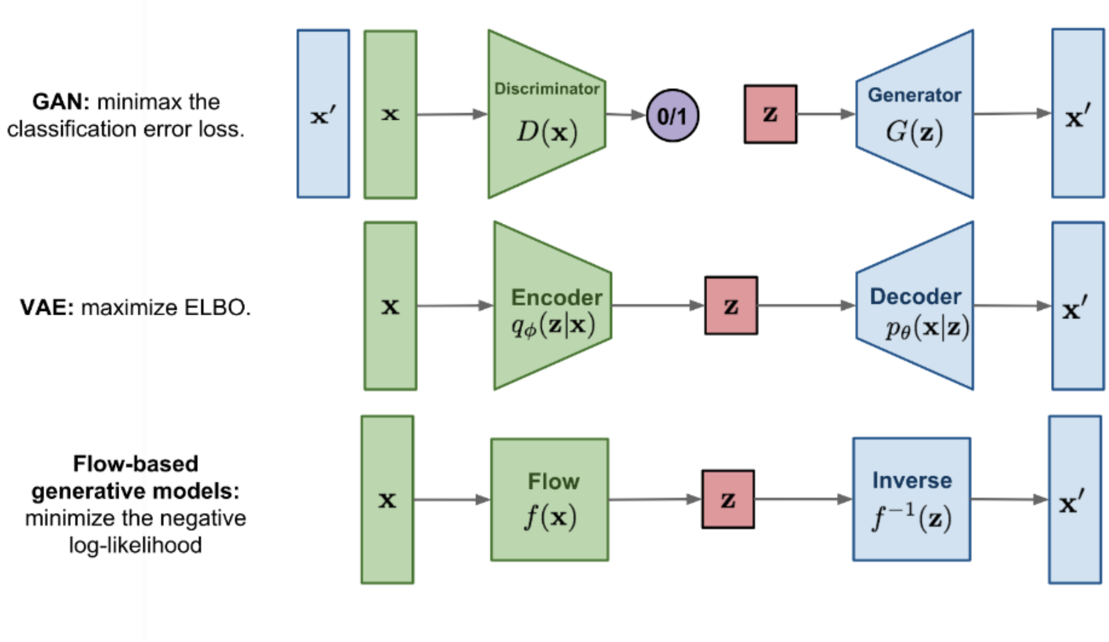
- Discuss the results of different model architecture

DCGan將卷積網路引入Gan的架構，DCGan相比Gan具有有以下特行:

1. 在網路架構中，除了generator model的輸出層及Discriminator的輸入層之外，其餘層都會使用Batch Normalization，可以解決初始狀態效果差的問題，也能防止generator將所有的樣本都收斂到同一個點
2. 去除全連接層，直接透過卷積層連接Generator和Discriminator

然而在task1中，我沒辦法使用cGlow train出好的結果，雖然生成的圖片品質還不錯，但最高的score只有0.14左右，training結果到最後都會overfitting，在學習condition的部份一直沒有進展

Flow和其他generative model布一樣的地方是Flow會明確的學習數據分佈p(x)，優勢有可逆，可以計算映射後的分佈體積，然而目前最流行的generative model還是gan，因為gan比較容易用隨機sample得方式生成未知的新圖片



- Task2
  - Results
    - Conditional facr generation(送分)
    - Linear interpolation

  從CelebA-HQ dataset中隨機選取兩張照片(a,b)(共選3組)，內插兩張照片的latent representation生成照片集，由下圖可看出從最左邊的照片(a)開始，透過內插最終得到最右邊的照片(b)

```
face_pair = [(1,8), (15,20), (57,58)]
    N = 8
    interpolation_result = []

    for p in range(len(face_pair)):
        x1, y1, x2, y2 = None, None, None, None
        for i,(x,y) in enumerate(test_loader):
            x = x.to(device)
            y = y.to(device)
            if i == face_pair[p][0] :
                x1, y1 = x, y
            if i == face_pair[p][1] :
                x2, y2 = x, y
                break

        z1, bpd, y_logits = model(x1, y1)  # return: z, bpd, y_logits
        z2, bpd, y_logits = model(x2, y2)  # return: z, bpd, y_logits
        z1 = z1.cpu()
        z2 = z2.cpu()
        z_list = interpolations(z1[0].detach().numpy(), z2[0].detach().numpy(), N)
```

```
        z_list = torch.Tensor(z_list).cuda()
        y_rand = torch.rand(40).unsqueeze(dim=1)
        predict_x = model(y_onehot=y_rand, z=z_list, temperature=1, reverse=True)
        print(predict_x.size())
        for k in range(len(predict_x)): interpolation_result.append(predict_x[k])


    save_image(interpolation_result, 'interpolation.png', normalize=True)
```

- Attribute manipulation

選取特定屬性，透過 z_{pos} - z_{neg} with different scalar從左往右疊加，我選取的特徵為依序為Brown Hair, Smiling, Wavy Hair，越往右scalar越大，所以越往右特徵會越明顯