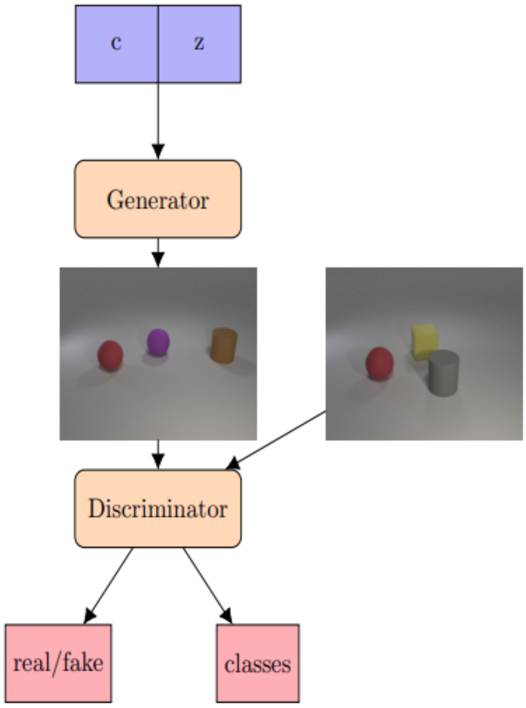
DLP Lab7 Let's Play GANs with Flows and friends

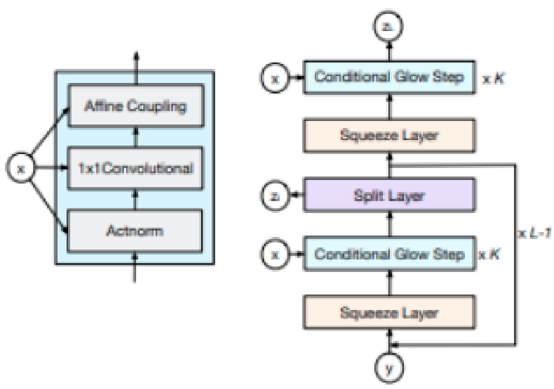
- Introduction
  - c-GAN

    利用conditional GAN 訓練一個可以生成指定條件的圖片。training data為ICLEVR的幾何物體圖片，共有24種不同的幾何物體，因此condition為一個24 dimension的one-hot vector如:[0, 0, 0, 1, 0, 0, 1,...0, 0, 0]



  - c-Glow

    c-Glow是一種用於結構化輸出學習的conditional Glow。c-Glow受益於基於flow的模型準確有效地計算p(y|x)的能力。使用c-Glow學習不需要替代目標或在訓練期間進行推理。一旦經過訓練，就可以直接有效地生成條件樣本。



- Implementation details
    - c-GAN

      這邊使用cDCGAN作為model的架構。

      Generator會把condition vector與雜訊z(100-dim)concat起來，不過conditiona vector會先經過fully connected layer把dimension從24變成200來擴充資訊，最後成為一個300-dim的vector，再連續做5次ConvTranspose變成fake image。

      在做ConvTranspose時，也會一併使用BatchNormalized2D和ReLU function。

```python
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
```

```python
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=(4, 4), stride=(2,2), padding=paddings[i-1]),
                nn.BatchNorm2d(channels[i]),
                nn.ReLU()
            ))
        self.convT5 = nn.ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2,2), padding=(1,1))
        self.tanh = nn.Tanh()
```

Discriminator會把24-dim的conditoinal vector經由fully connected layer和reshape變成一張1*64*64的圖，同樣是為

了擴充資訊，之後再與training data或Generator出來的圖片做concat變成3+1*64*64的圖片，接著連續做5次Conv就可以到一個scalar。
在做Conv時，也會一併使用BatchNormalized2D和LeakyReLU function。
這邊的loss function選擇binary cross entropy。

```
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
```

```
class Discriminator(nn.Module):
    def __init__(self, img_shape, c_dim):
        super(Discriminator, self).__init__()
        self.H, self.W, self.C = img_shape

        self.conditionExpand = nn.Sequential(
            nn.Linear(24, self.H * self.W * 1),
            nn.LeakyReLU()
        )

        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=(4, 4), stride=(2,2), padding=(1,1)),
                nn.BatchNorm2d(channels[i]),
                nn.LeakyReLU()
            ))
        self.conv5 = nn.Conv2d(512, 1, kernel_size=(4, 4), stride=(1,1))
        self.sigmoid = nn.Sigmoid()
```

總共train了250個epochs，learning rate為0.0002，batch size為16，雜訊z_dim=100，條件c_dim=200。

- c-NF

  這邊的架構使用c-Glow，主要參考https://github.com/y0ast/Glow-PyTorch這個repo去做修改，將

num_class改成40，以及dataloader的相關地方就能使用。

針對不同的task也有不同的inference可以使用。

總共train 50個epochs，batch size為16，K=6，L=3。

在normal_flow中的flow_coupling有additive和affine兩種版本可以選擇，作者說affine在還原能力上比較好，additive在條件生成的表現上較佳。

```python
def normal_flow(self, input, logdet):
    assert input.size(1) % 2 == 0

    # 1. actnorm
    z, logdet = self.actnorm(input, logdet=logdet, reverse=False)

    # 2. permute
    z, logdet = self.flow_permutation(z, logdet, False)

    # 3. coupling
    z1, z2 = split_feature(z, "split")
    if self.flow_coupling == "additive":
        z2 = z2 + self.block(z1)
    elif self.flow_coupling == "affine":
        h = self.block(z1)
        shift, scale = split_feature(h, "cross")
        scale = torch.sigmoid(scale + 2.0)
        z2 = z2 + shift
        z2 = z2 * scale
        logdet = torch.sum(torch.log(scale), dim=[1, 2, 3]) + logdet
    z = torch.cat((z1, z2), dim=1)

    return z, logdet
```

Task2的第一題，是根據給定的不同條件生成具有不同特徵的臉。這裡讀進condition後再透過reverse model就可生出特定特徵的臉。

```python
attribute_list = [20, 31, 26, 16] # Male, Smiling, Pale_Skin, Goatee
generate_x_list = torch.Tensor([]).cuda()
for yes in [1,0]:
    for i,attribute in enumerate(attribute_list):
        z = torch.rand( (1, 48, 8, 8) )
        y = torch.zeros(40).unsqueeze(dim=1)
        y[attribute] = yes
        predict_x = model(y_onehot=y.cuda(), z=z.cuda(), temperature=1, reverse=True)
        generate_x_list = torch.cat((generate_x_list,predict_x), 0)
save_image(generate_x_list, 'images/task2_Conditional_face.png',normalize=True)
```

Task2的第二題，是要從兩張臉內插出中間的多張臉，這邊隨機選擇三對臉，將各個照片x和對應的y餵進forward model，得到各別的z。然後對這些z做線性內插，算出中間過程中的多個z，最後再將這些z餵進reverse model，生成漸進變化的多張臉。

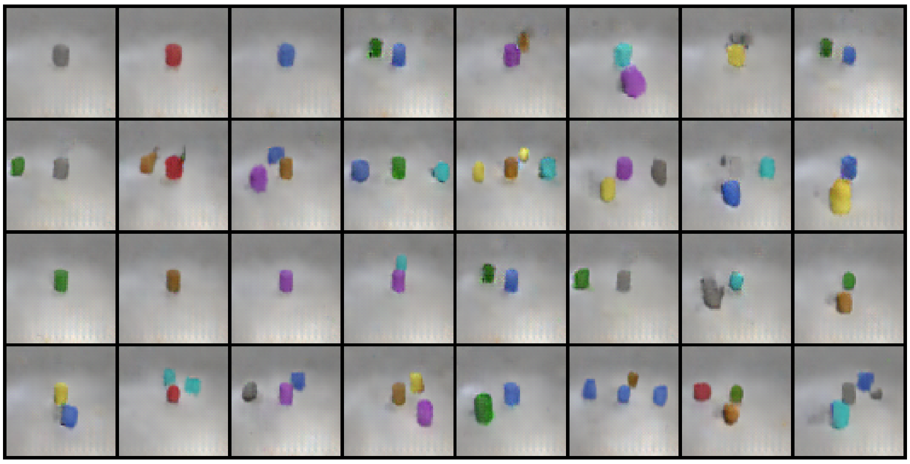
```
def interpolations(z1, z2, n):
    z_list = []
    for j in range(n):
        list_n = []
        for i in range(len(z1)):
            top = z1[i]
            down = z2[i]
            value = down + 1.0 * j * (top - down) / n
            list_n.append(value)
        z_list.append(list_n)
    return np.array(z_list)
```

Task2的第三題，要調整整張臉出現出特定特徵。這邊選一張照片，將x,y餵進forward model得到z，接著檢查這張照片是否有該特徵。假設有，接著讀進整個dataset，遇到沒有該特徵的照片就把x,y餵進forward model取得z，最後將所有沒有該特徵的z取平均。然後從所選的，有該特徵的z，內插到算出的平均z，但這樣的結果會導致其他特徵也越趨近平均值。

- Results
  - Task1(c-GAN)

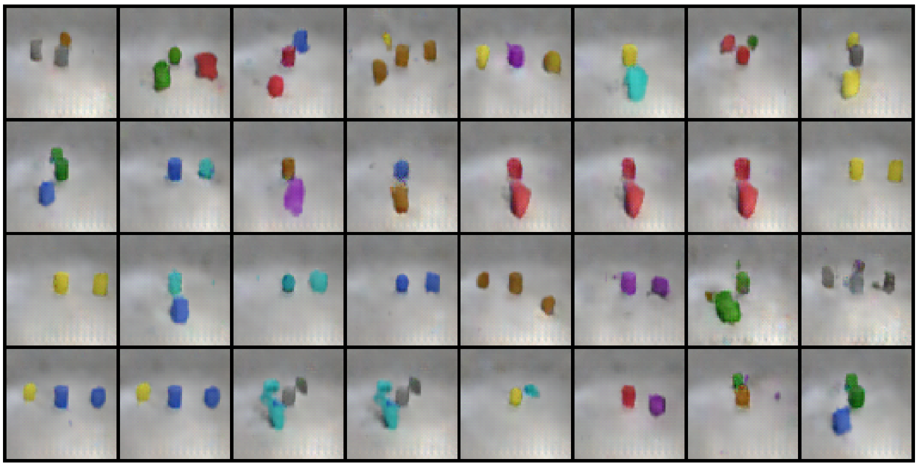
    test:

```
score: 0.75
score: 0.74
score: 0.74
score: 0.74
score: 0.72
score: 0.72
score: 0.74
score: 0.74
score: 0.72
score: 0.72

avg score: 0.73
```



new_test:

```
score: 0.62
score: 0.60
score: 0.60
score: 0.58
score: 0.60
score: 0.60
score: 0.61
score: 0.60
score: 0.60
score: 0.60

avg score: 0.60
```

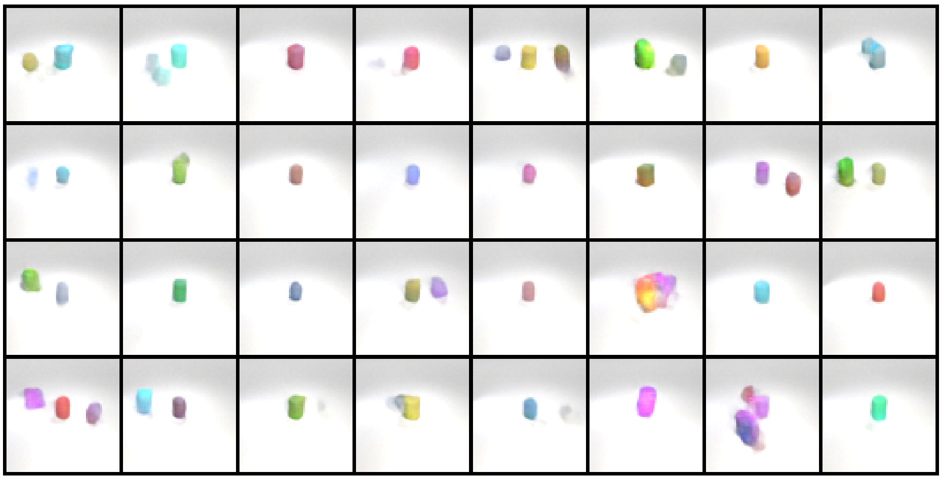


- Task1(c-NF)

  生成出來的圖片都很清晰，看起來很漂亮，但分數都小於0.2

  test:

new_test:



- Task2

  Linear interpolation

Attribute manipulation

第一列Big_Lips, 第二列Smiling, 第三列Wavy_Hair

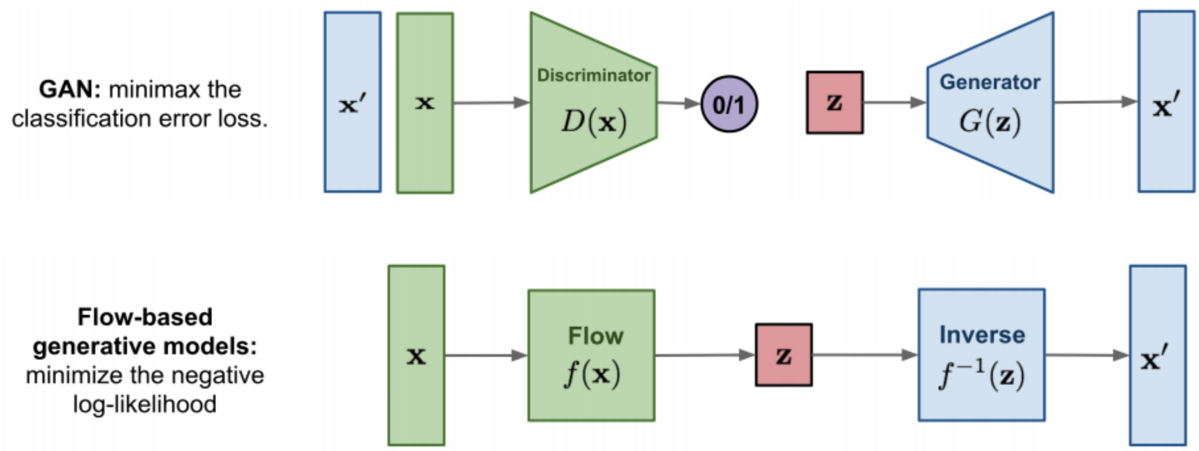
- Discuss

```
"""
train generator
"""
for _ in range(4):
    optimizer_g.zero_grad()

    z = random_z(batch_size, z_dim).to(device)
    gen_imgs = g_model(z, conditions)
    predicts = d_model(gen_imgs, conditions)
    loss_g = Criterion(predicts, real)
    # bp
    loss_g.backward()
    optimizer_g.step()
```

c-GAN的train generator 4次或5次效果差不多。Generator用ReLU且Discriminator要用LeakyReLU。加入Batchnormalized2D可以提高score。在生成fake照片時用的condition vector使用training data已有的條件就好，用自己隨機生成的condition vector反而會train壞掉。

c-Glow在task1中效果很差，似乎沒有學習到condition，loss收斂很快，可能是已經overfitting了，雖然生成的圖片很清晰，但condition和score上卻一直沒有進展。

Flow和其他generative model不一樣的地方在於該模型明確地學習數據分佈p(x)。它的優勢主要有可逆映射，可計算映射後的分佈體積，容易模擬等。但因GAN較容易用隨機sample noise的方式生成未知的新東西，因此還是為generative model的大宗。



GAN: minimax the classification error loss.
$\mathbf{x}'$
$\mathbf{x}$
Discriminator
$D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator
$G(\mathbf{z})$
$\mathbf{x}'$
Flow-based generative models: minimize the negative log-likelihood
$\mathbf{x}$
Flow
$f(\mathbf{x})$
$\mathbf{z}$
Inverse
$f^{-1}(\mathbf{z})$
$\mathbf{x}'$