Label 4 :
Info GAN

Outline

1. Introduction

2. Experiment setups

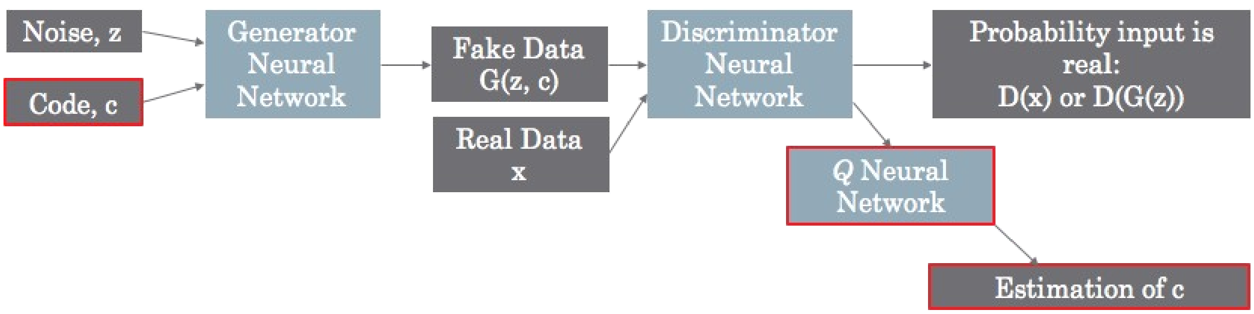
3. Experiment results

4. Discussion

1. Introduction

運用擁有可以找到隱藏 code 與數據關係的抗式網路架構, 搭配 MNIST 資料訓練出用特定的 code 可以輸出特定的 MNIST 數字圖片。

網路架構:



目標公式:

$$\min_{G,Q} \max_{D} V_{\text{InfoGAN}}(D, G, Q) = V(D, G) - \lambda L_I(G, Q)$$

$$L_1(G, Q) = E_{c \sim P(c), x \sim G(z,c)}[\log Q(c \mid x)] + H(c)$$

2. Experiment setups

Setups：

beta1, beta2: 控制 Adam 計算梯度及平方的運行平均值的係數。

criterion_q_continous: 計算 code 對資料的 loss。

```
batch_size = 64
epoch = 100
learning_rate = 1e-4
beta1 = 5e-1
beta2 = 999e-3
```

```
criterion_discriminator = nn.BCELoss()
criterion_q_discrete = nn.CrossEntropyLoss()
criterion_q_continuous = NormalNLLLoss()

optimizer_discriminator = optim.Adam([{'params': model.discriminator.parameters()}, {'params': model.dHead.parameters()}],
                                     lr = learning_rate, betas = (beta1, beta2))
optimizer_generator = optim.Adam([{'params': model.generator.parameters()}, {'params': model.qHead.parameters()}],
                                 lr = learning_rate, betas = (beta1, beta2))
```

```
class NormalNLLLoss:
    def __call__(self, x, mu, var):
        logli = -0.5 * (var.mul(2 * np.pi) + 1e-6).log() - (x - mu).pow(2).div(var.mul(2.0) + 1e-6)
        return -(logli.sum(1).mean())
```

Generator: 加厚了一組的 ConvTranspose2d，以及 activation 改用 LeakyReLU。

```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.generator = nn.Sequential(
            nn.ConvTranspose2d(74, 512, 1, 1, bias = False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(),

            nn.ConvTranspose2d(512, 1024, 1, 1, bias = False),
            nn.BatchNorm2d(1024),
            nn.LeakyReLU(),

            nn.ConvTranspose2d(1024, 128, 7, 1, bias = False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(),

            nn.ConvTranspose2d(128, 64, 4, 2, padding = 1, bias = False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(),

            nn.ConvTranspose2d(64, 1, 4, 2, padding = 1, bias = False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.generator(input)
```

Discriminator:

```python
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.discriminator = nn.Sequential(
            nn.Conv2d(1, 64, 4, 2, 1),
            nn.LeakyReLU(0.1, True),

            nn.Conv2d(64, 128, 4, 2, 1, bias = False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.1, True),

            nn.Conv2d(128, 1024, 7, bias = False),
            nn.BatchNorm2d(1024),
            nn.LeakyReLU(0.1, True)
        )
    def forward(self, input):
        return self.discriminator(input)
```

DHead: 加厚一層 Conv2d 以及搭配一個 Sigmoid。

```python
class DHead(nn.Module):
    def __init__(self):
        super().__init__()
        self.d = nn.Sequential(
            nn.Conv2d(1024, 512, 1),
            nn.Sigmoid(),
            nn.Conv2d(512, 1, 1),
            nn.Sigmoid()
        )

    def forward(self, inputs):
        return self.d(inputs)
```

QHead:

```python
class QHead(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(1024, 128, 1, bias=False)
        self.bn1 = nn.BatchNorm2d(128)
        self.leakyReLU = nn.LeakyReLU(0.1, True)

        self.conv_disc = nn.Conv2d(128, 10, 1)
        self.conv_mu = nn.Conv2d(128, 2, 1)
        self.conv_var = nn.Conv2d(128, 2, 1)

    def forward(self, x):
        x = self.leakyReLU(self.bn1(self.conv1(x)))

        disc_logits = self.conv_disc(x).squeeze()

        mu = self.conv_mu(x).squeeze()
        var = torch.exp(self.conv_var(x).squeeze())

        return disc_logits, mu, var
```

InfoGAN: 組合以上 Model。

```python
class InfoGAN(nn.Module):

    def __init__(self, device):
        super().__init__()
        self.generator = Generator().to(device)
        self.generator.apply(self.weights_init)

        self.discriminator = Discriminator().to(device)
        self.discriminator.apply(self.weights_init)

        self.qHead = QHead().to(device)
        self.qHead.apply(self.weights_init)

        self.dHead = DHead().to(device)
        self.dHead.apply(self.weights_init)

    def forward_realData(self, inputs):
        outputs = self.discriminator(inputs)
        return self.dHead(outputs).view(-1)

    def forward_noise(self, noise):
        fake_data = self.generator (noise)
        outputs = self.discriminator(fake_data.detach())
        probs_fake = self.dHead(outputs).view(-1)
        return fake_data, probs_fake

    def foward_fake_data_treated_as_real(self, fake_data):
        output_treated = self.discriminator(fake_data)
        probs_treated = self.dHead(output_treated).view(-1)
        return output_treated, probs_treated

    def foward_treated_data_to_Q(self, treated_data):
        return self.qHead(treated_data)

    def foward_genImage(self, fixed_noise):
        return self.generator(fixed_noise).detach().cpu()

    def weights_init(self, model):
        if(type(model) == nn.ConvTranspose2d or type(model) == nn.Conv2d):
            nn.init.normal_(model.weight.data, 0.0, 0.02)
        elif(type(model) == nn.BatchNorm2d):
            nn.init.normal_(model.weight.data, 1.0, 0.02)
            nn.init.constant_(model.bias.data, 0)
```

DataTransformer: 生成 MNIST 以及測試用的 fixed noise 與訓練

用的 noise sample 和 Label 腳本。

```python
class DataTransformer:
    def __init__(self, use_cuda):
        self.root = "./Datasets/"
        self.use_cuda = use_cuda

        self.num_z = 62
        self.num_dis_c = 1
        self.dis_c_dim = 10
        self.num_con_c = 2

        self.real_label = 1
        self.fake_label = 0


    def get_dataloader(self, batch_size):
        transform = transforms.Compose([transforms.Resize(28), transforms.CenterCrop(28), transforms.ToTensor()])
        dataset = datasets.MNIST(self.root, train = 'train', download = True, transform = transform)
        dataloader = torch.utils.data.DataLoader(dataset, batch_size = batch_size, shuffle = True)
        return dataloader
```

```python
def fixedNoise(self, count = 100, number = None):
    fixed_noise = torch.randn(count, self.num_z, 1, 1)

    if number == None:
        idx = np.arange(self.dis_c_dim).repeat(10)
    else:
        number = [self.changeIndex(number[x]) for x in range(len(number))]
        idx = number

    dis_c = torch.zeros(count, self.num_dis_c, self.dis_c_dim)
    for i in range(self.num_dis_c):
        dis_c[torch.arange(0, count), i, idx] = 1.0
    dis_c = dis_c.view(count, -1, 1, 1)

    con_c = torch.rand(count, self.num_con_c, 1, 1) * 2  - 1

    fixed_noise = torch.cat((fixed_noise, dis_c), 1)
    fixed_noise = torch.cat((fixed_noise, con_c), 1)

    if self.use_cuda:
        fixed_noise = fixed_noise.cuda()
    return fixed_noise
```

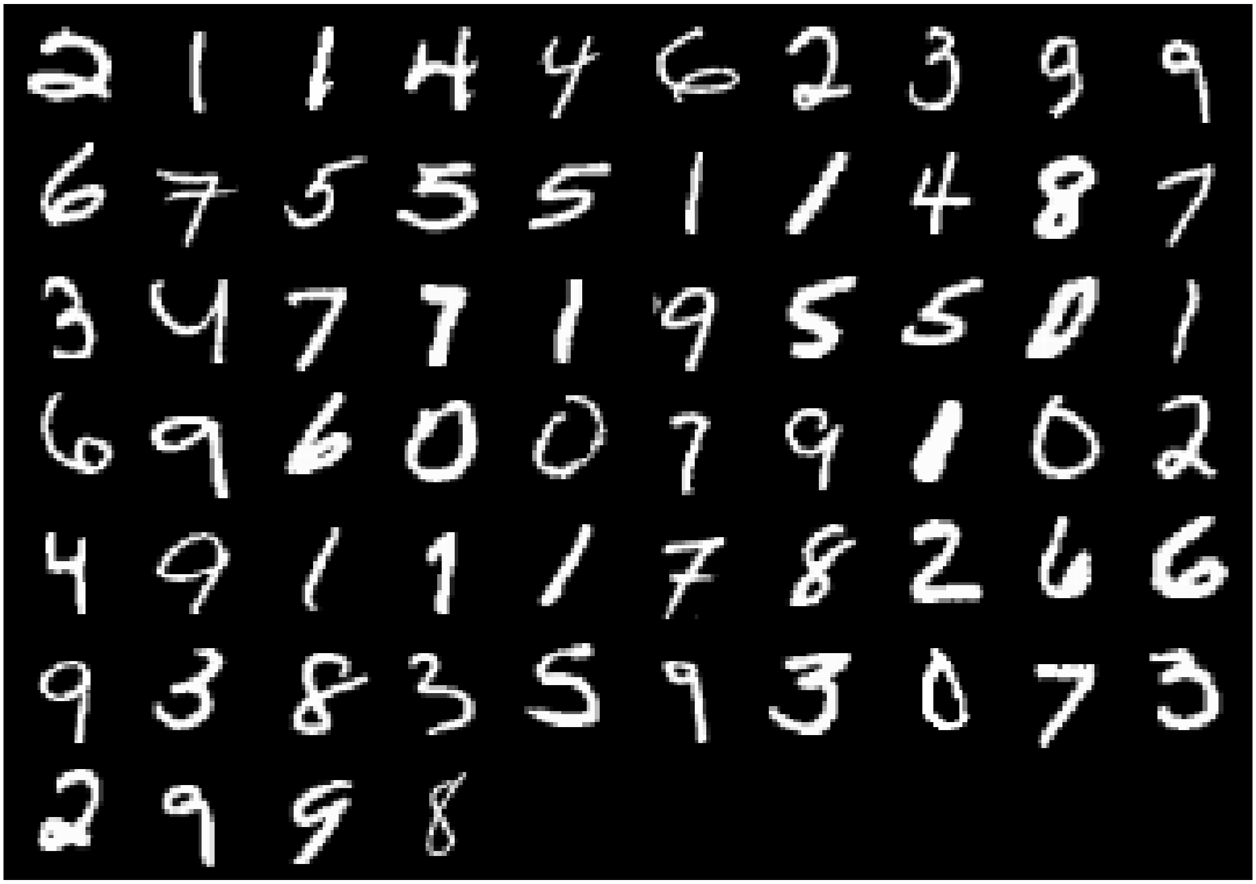
```python
def getNoiseSample(self, batch_size):
    z = torch.randn(batch_size, self.num_z, 1, 1)
    idx = np.zeros((self.num_dis_c, batch_size))
    dis_c = torch.zeros(batch_size, self.num_dis_c, self.dis_c_dim)

    for i in range(self.num_dis_c):
        idx[i] = np.random.randint(self.dis_c_dim, size = batch_size)
        dis_c[torch.arange(0, batch_size), i, idx[i]] = 1.0
    dis_c = dis_c.view(batch_size, -1, 1, 1)

    con_c = torch.rand(batch_size, self.num_con_c, 1, 1) * 2 - 1

    z = torch.cat((z, dis_c), 1)
    z = torch.cat((z, con_c), 1)

    if self.use_cuda:
        z = z.cuda()

    return z, idx
```

```python
def getRealLabel(self, size):
    label = torch.full((size, ), self.real_label)
    if self.use_cuda:
        label = label.cuda()
    return label

def getFakeLabel(self, real_label):
    return real_label.clone().fill_(self.fake_label)
```

Train: 藍色框針對訓練 discriminator, 紅色框框針對 generator 訓練。

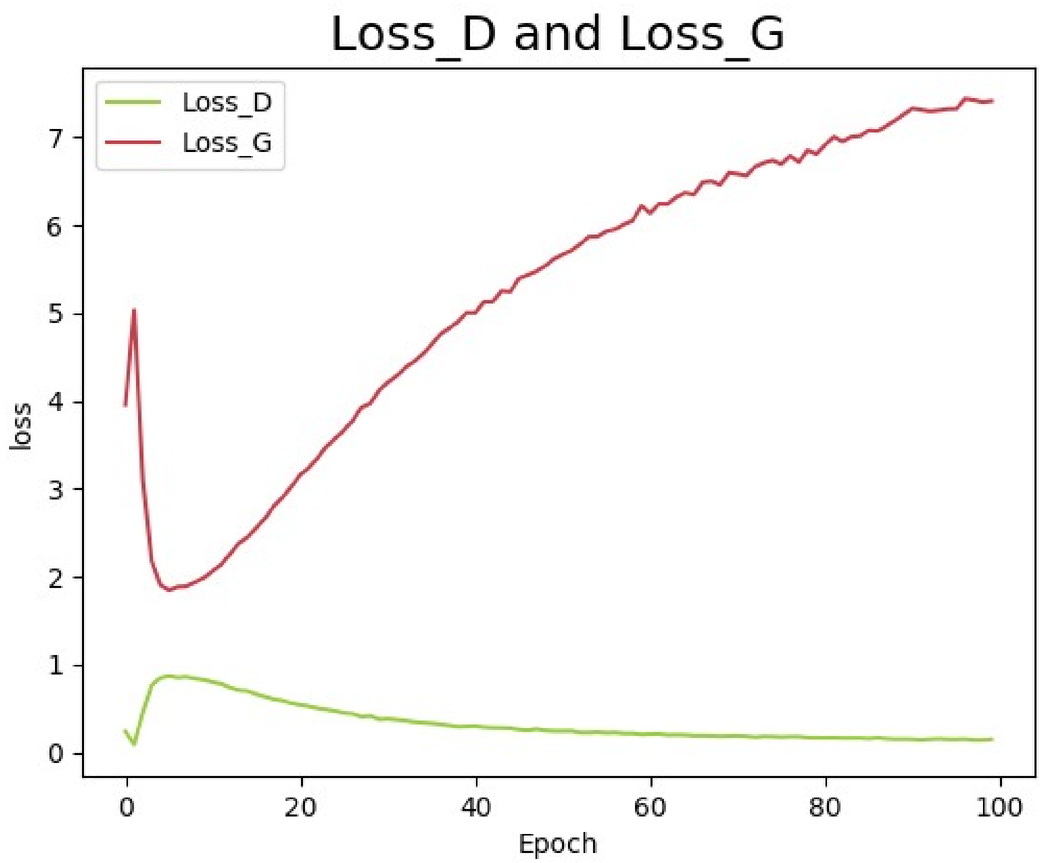
```python
for e in range(1, epoch + 1):
    model.changeToTrainMode()
    D_losses = 0
    G_losses = 0
    for times, (real_data, _) in enumerate(dataloader, 0):

        real_data_size = real_data.size(0)
        if use_cuda:
            real_data = real_data.cuda()

        optimizer_discriminator.zero_grad()
        real_label = dataTransfomer.getRealLabel(real_data_size)
        output_real = model.forward_realData(real_data)
        loss_real = criterion_discriminator(output_real, real_label)
        loss_real.backward()

        fake_label = dataTransfomer.getFakeLabel(real_label)
        noise, idx = dataTransfomer.getNoiseSample(real_data_size)
        fake_data, probs_fake = model.forward_noise(noise)
        loss_fake = criterion_discriminator(probs_fake, fake_label)
        loss_fake.backward()

        D_loss = loss_real + loss_fake
        optimizer_discriminator.step()

        optimizer_generator.zero_grad()
        output_treated, probs_treated = model.foward_fake_data_treated_as_real(fake_data)
        loss_gen = criterion_discriminator(probs_treated, real_label)

        q_logits, q_mu, q_var = model.foward_treated_data_to_Q(output_treated)
        target = torch.LongTensor(idx).to(device)
        dis_loss = 0
        for x in range(dataTransfomer.num_dis_c):
            dis_loss += criterion_q_discrete(q_logits[:, x * 10 : x * 10 + 10], target[x])
        con_loss = criterion_q_continuous(
            noise[:, dataTransfomer.num_z + dataTransfomer.num_dis_c * dataTransfomer.dis_c_dim :].view(-1, dataTransfomer.num_con_c),
            q_mu, q_var) * 0.1

        G_loss = loss_gen + dis_loss + con_loss
        G_loss.backward()
        optimizer_generator.step()

        G_losses += G_loss.item()
        D_losses += D_loss.item()
        if times % 100 == 0:
            print('[%d/%d][%d/%d]' %(e, epoch, times, len(dataloader)))

    G_losses /= times
    D_losses /= times
    Total = D_losses + G_losses

    state = 'e: {}, Loss_D: {}, Loss_G: {}, Total: {}\n'.format(e, D_losses, G_losses, Total)
```

Training Image:

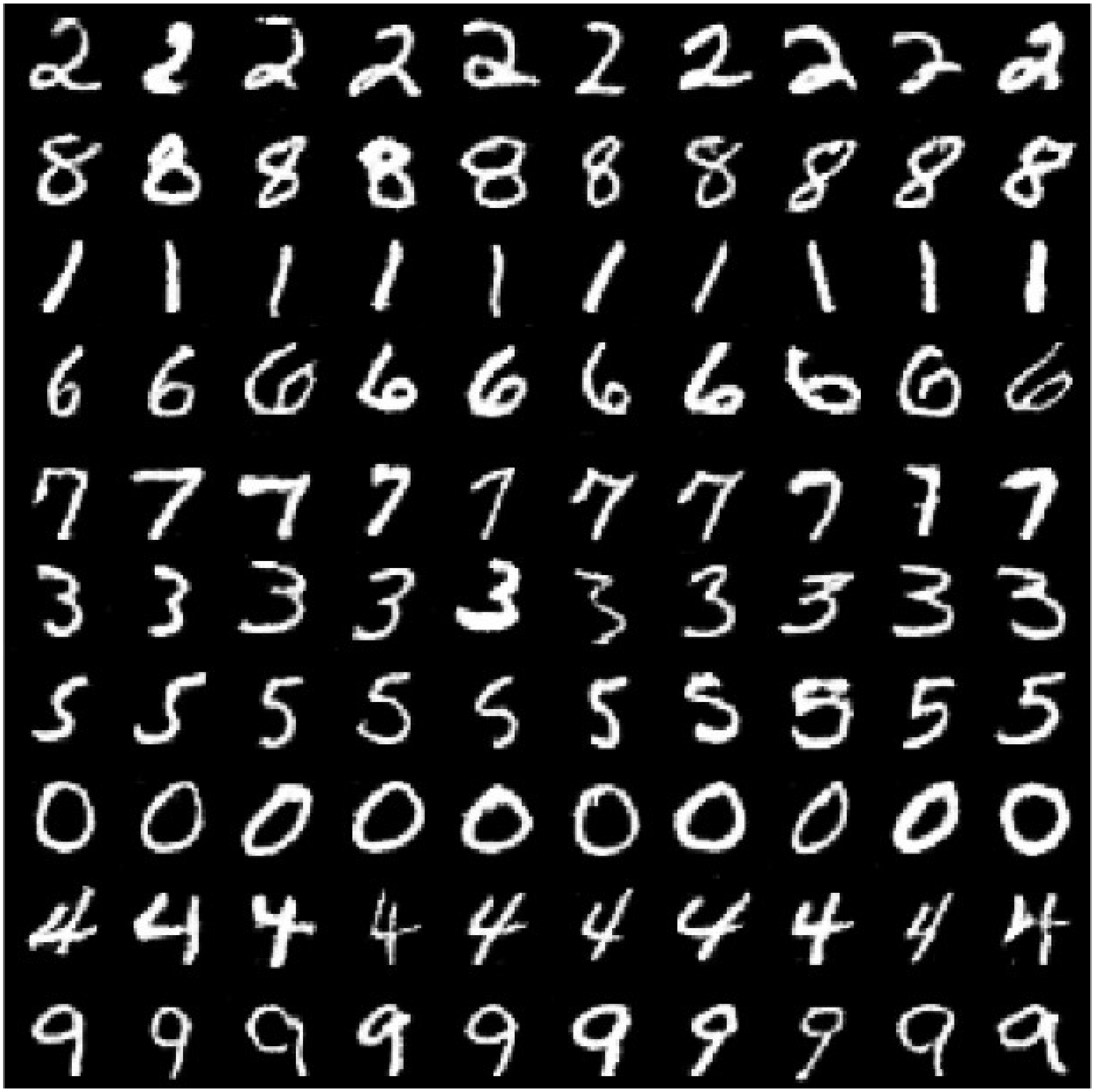


3. Experiment results

Training situation：會發現 LossD 與 LossG 前面會彼此靠近後開始相差越來越遠



Prdict in training:



Testing: 輸入[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

4. Discussion

從 Loss 曲線圖發現, 以及實際 predict 出來的,越往後面的訓練 LossD 與 LossG 的訓練成效就不會有太大的改變或是越來越糟糕, 可能是因為 Discriminator 越來越強,導致 Generator 的訓練變得隨便都可以過。