Hw6 Report

R09921026 李育倫

1. Describe the difference between WGAN* and GAN**, list at least two differences

1.

GAN 的生成器目標為 minimize Pdata、PG 的 divergence，而判別器所 maximize 的目標函數與 JS divergence 相關，公式如下:

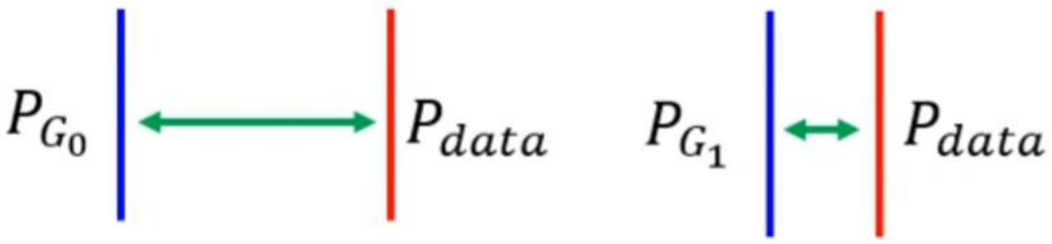
$$D^* = \arg\max_D V(D,G)$$

$$V(G,D) = E_{y\_P_{data}}[\log D(y)] + E_{y\sim P_G}[\log(1-D(y))]$$ (與 JS divergence 相關)

V(G,D) 又為 BCELOSS

$$G^* = \arg\min_G \mathrm{Div}(P_G,P_{data}) \rightarrow G^* = \arg\min_G \max_D V(D,G)$$

PG 和 Pdata 的圖片可以看成是在高維空間中的兩條線，如果兩直線彼此不相交算出來的 JS divergence 都為 log2，這導致在訓練非常困難，因為模型會覺得下面兩種情況是一樣的，但實際上右邊的 case 較好。



因此 WGAN 使用 Wasserstein distance(如下式)取代 JS divergence(BCELOSS) 改善了上述的缺點。

$$\max_{D\in 1-Lipschitz}\{E_{y\sim p_{data}}[D(y)] - E_{y\sim p_G}[D(y)]\}$$

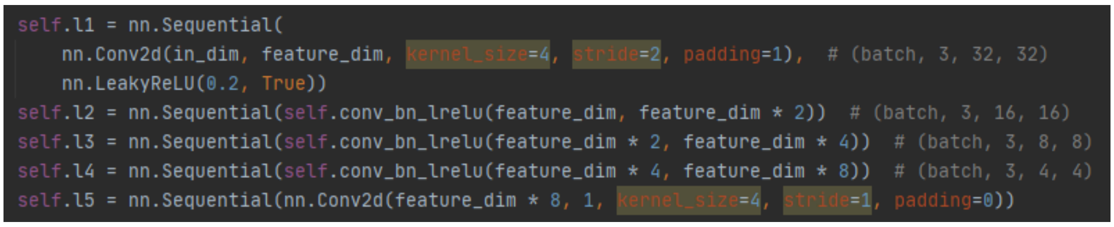
2.

WGAN 需限制判別器夠平滑，也就是 $D\in 1-Lipschitz$，避免在 PG、Pdata 兩不重疊的時候有劇烈的變化，也就是說在 PG 與 Pdata 距離很近的時候 D 無法給 real data 很大的值，generated data 很小的值，因此在 sample code 訓練時需將判別器的參數限制在-1~1 之間(weight clipping)，或使用 Gradient Penalty 技術。

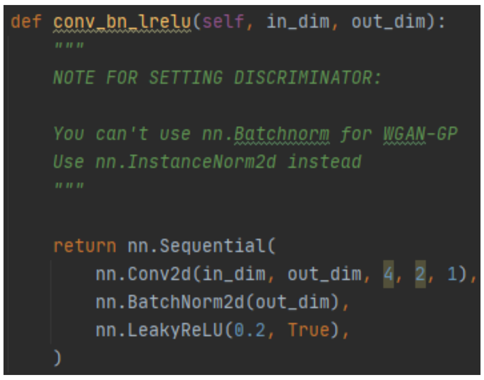
2. Please plot the “Gradient norm” result.

Discriminator 架構:

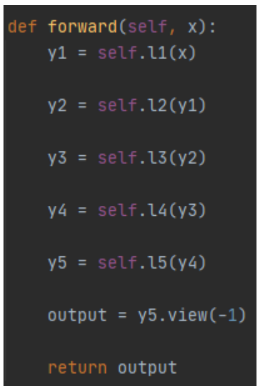
與原始 sample code 架構相同，共 5 層 Convolution Layer，中間三層有使用 Batch norm，如下圖:



```
self.l1 = nn.Sequential(
    nn.Conv2d(in_dim, feature_dim, kernel_size=4, stride=2, padding=1),  # (batch, 3, 32, 32)
    nn.LeakyReLU(0.2, True))
self.l2 = nn.Sequential(self.conv_bn_lrelu(feature_dim, feature_dim * 2))  # (batch, 3, 16, 16)
self.l3 = nn.Sequential(self.conv_bn_lrelu(feature_dim * 2, feature_dim * 4))  # (batch, 3, 8, 8)
self.l4 = nn.Sequential(self.conv_bn_lrelu(feature_dim * 4, feature_dim * 8))  # (batch, 3, 4, 4)
self.l5 = nn.Sequential(nn.Conv2d(feature_dim * 8, 1, kernel_size=4, stride=1, padding=0))
```

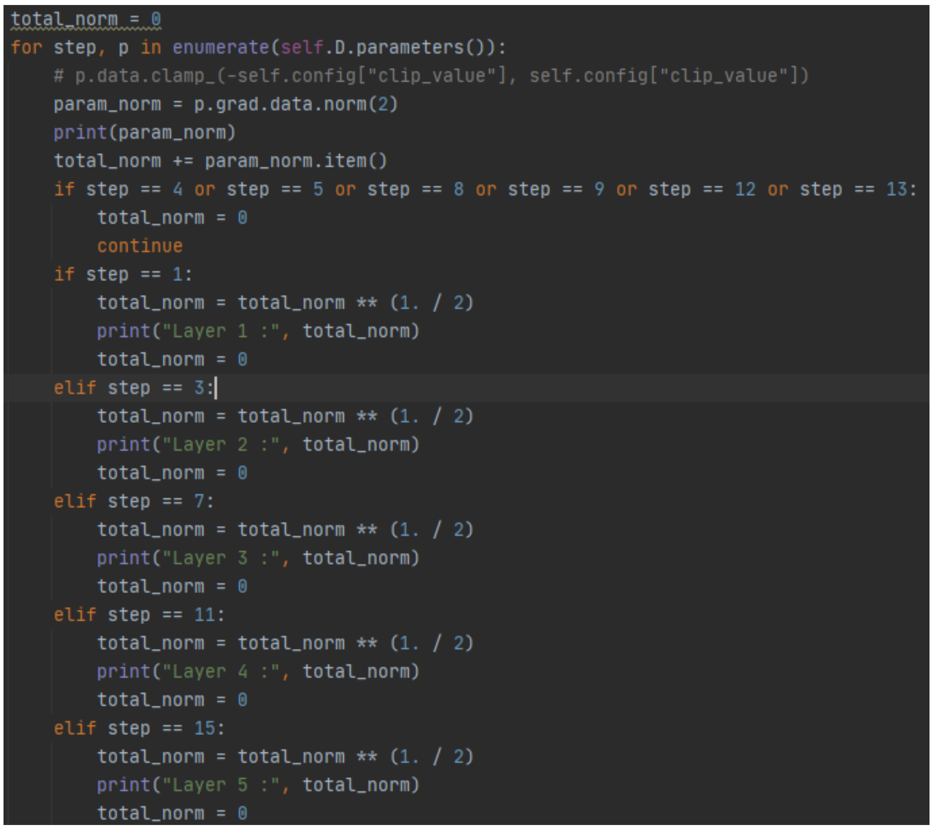


```
def conv_bn_lrelu(self, in_dim, out_dim):
    """
    NOTE FOR SETTING DISCRIMINATOR:

    You can't use nn.Batchnorm for WGAN-GP
    Use nn.InstanceNorm2d instead
    """

    return nn.Sequential(
        nn.Conv2d(in_dim, out_dim, 4, 2, 1),
        nn.BatchNorm2d(out_dim),
        nn.LeakyReLU(0.2, True),
    )
```



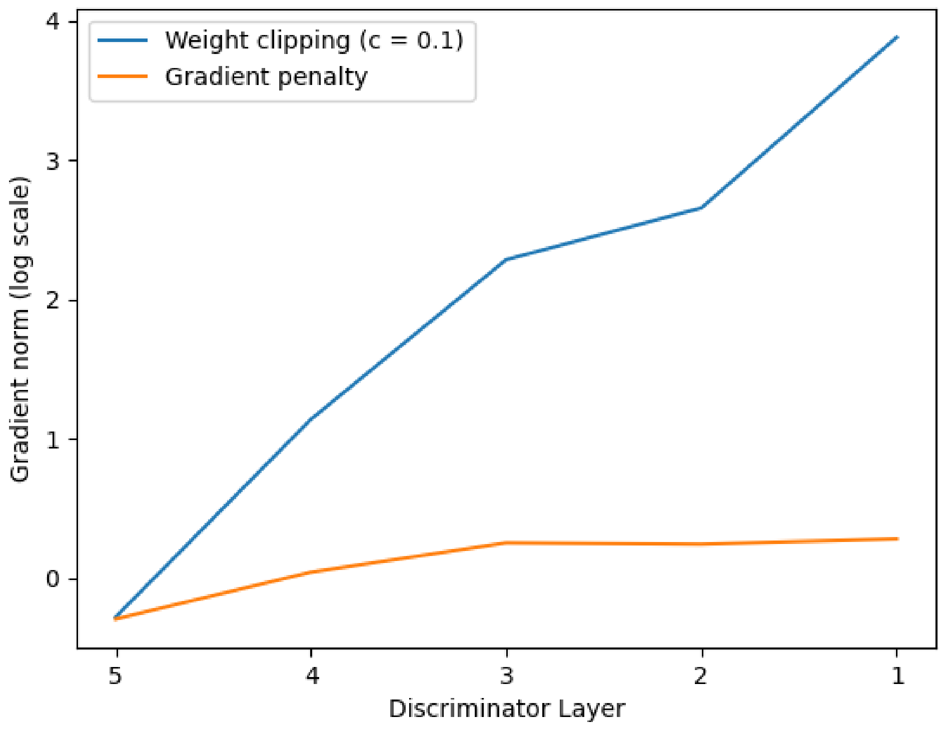
```
def forward(self, x):
    y1 = self.l1(x)

    y2 = self.l2(y1)

    y3 = self.l3(y2)

    y4 = self.l4(y3)

    y5 = self.l5(y4)

    output = y5.view(-1)

    return output
```

每層 grad norm 計算方式如下(取 norm 2)，放置於 loss_D.backward()之後:



```
total_norm = 0
for step, p in enumerate(self.D.parameters()):
    # p.data.clamp_(-self.config["clip_value"], self.config["clip_value"])
    param_norm = p.grad.data.norm(2)
    print(param_norm)
    total_norm += param_norm.item()
    if step == 4 or step == 5 or step == 8 or step == 9 or step == 12 or step == 13:
        total_norm = 0
        continue
    if step == 1:
        total_norm = total_norm ** (1. / 2)
        print("Layer 1 :", total_norm)
        total_norm = 0
    elif step == 3:
        total_norm = total_norm ** (1. / 2)
        print("Layer 2 :", total_norm)
        total_norm = 0
    elif step == 7:
        total_norm = total_norm ** (1. / 2)
        print("Layer 3 :", total_norm)
        total_norm = 0
    elif step == 11:
        total_norm = total_norm ** (1. / 2)
        print("Layer 4 :", total_norm)
        total_norm = 0
    elif step == 15:
        total_norm = total_norm ** (1. / 2)
        print("Layer 5 :", total_norm)
        total_norm = 0
```

圖如下:


Weight clipping (c = 0.1)
Gradient penalty
Gradient norm (log scale)
Discriminator Layer
4
3
2
1
0
5
4
3
2
1