Advanced Machine Learning Techniques - GANs

Rassin, Royi

isroei5700@gmail.com

1 Line

The parameters used for the Line and Parabola distributions are: $Zdim = 1$, $BatchSize = 4$, ADAM optimizer for both the Generator and the Discriminator (with the default learning rate and betas), 1000 samples were generated, the models were trained for 80 epochs; so a total of $80 * 250 = 20,000$ iterations. Further, I applied BCEloss as criterion, and defined the loss as: $D_{loss} = mean(D_{real_loss} + D_{fake_loss})$, and G_{loss} is the criterion output on the Discriminator's output on the generator's fake data, when compared to a label of the 'real' class. In terms of network architecture and parameters: the Generator had one hidden layer, with LeakyReLU(0.3) right after it. Other than that, the network is straightforward. The hidden dim is 64. The Discriminator is almost identical. The only difference is that there is a sigmoid after the LeakyReLU(0.3).

2 Parabola

Everything is identical to the Line configuration with the exception of ADAM's initial LRs. I noticed the Generator seems to be overpowering the Discriminator, so I initialized the LRs to be: $G_{LR} = 0.001$ and $D_{LR} = 0.00999$.

3 Spiral

The Spiral distribution required more complex architectures, modifying how we train the generator, and the sample size. First, the Discriminator was modified to have a hidden dimension of 32, however, extra hidden layers were added, and another LeakyReLU(0.3), between the first two hidden layers. The Generator's architecture was changed, and is identical to the new Discriminator's architecture. However, two hidden layers were of the dimension 128, and the other two, of 32. The Generator was training every other epoch, since it was still learning faster than the Discriminator, and the Discriminator's initial learning rates was changed again: $D_{LR} = 0.003$. Despite an improvement in the results, it felt like more dimensions, different ReLU values, DropOuts, different loss functions, and

holding back one of the models so the other one will catch up (like I did with the Generator), was not gonna cut it. Thus, I increased the sample size to 10K, and the number of epochs to 250. However, the Batch Size was also increased to 32. So the total number of iterations was 78K (although, the problem was solved long before that).

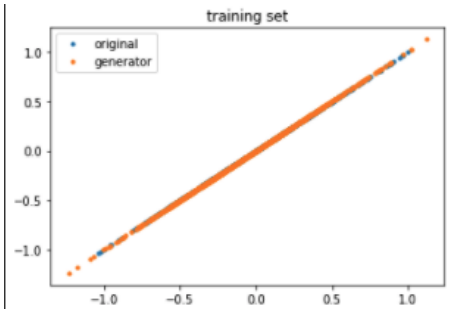
Please note: sometimes the model achieves the best results after going through a small number of epochs, and other times it needs all of the epochs, it highly depends on how it was initialized.

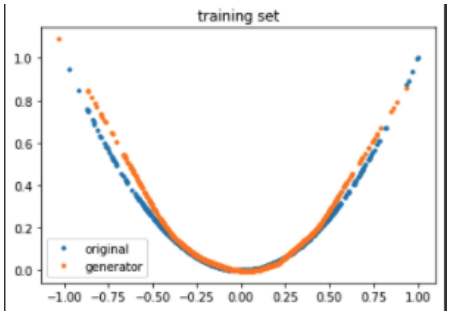
```python
class Discriminator(nn.Module):
    def __init__(self, in_dim: int, hidden_dim: int, out_dim: int):
        super(Discriminator, self).__init__()

        self.layers = nn.Sequential(
            nn.Linear(in_dim, hidden_dim),
            nn.LeakyReLU(0.3),
            nn.Linear(hidden_dim, out_dim),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.layers(x.to(device))


class Generator(nn.Module):
    def __init__(self, in_dim: int, hidden_dim: int, out_dim: int):
        super(Generator, self).__init__()

        self.layers = nn.Sequential(
            nn.Linear(Z_DIM, in_dim),
            nn.Linear(in_dim, hidden_dim),
            nn.LeakyReLU(0.3),
            nn.Linear(hidden_dim, out_dim),
        )

    def forward(self, x):
        return self.layers(x)
```

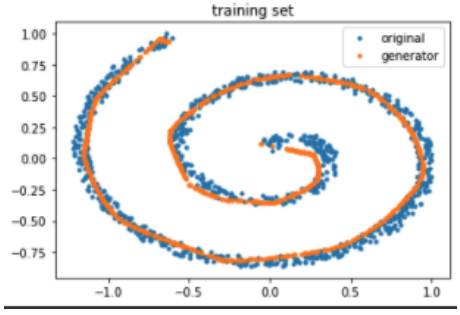
Figure 1: Line and Parabola Architecture

```python
class Discriminator(nn.Module):
    def __init__(self, in_dim: int, hidden_dim: int, out_dim: int):
        super(Discriminator, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(in_dim, hidden_dim),
            nn.LeakyReLU(0.3),
            nn.Linear(hidden_dim, hidden_dim),
            nn.LeakyReLU(0.3),
            nn.Linear(hidden_dim, out_dim),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.layers(x.to(device))


class Generator(nn.Module):
    def __init__(self, in_dim: int, hidden_dim: int, out_dim: int):
        super(Generator, self).__init__()

        self.layers = nn.Sequential(
            nn.Linear(Z_DIM, in_dim),
            nn.Linear(in_dim, 128),
            nn.LeakyReLU(0.3),
            nn.Linear(128, 32),
            nn.LeakyReLU(0.3),
            nn.Linear(32, out_dim),
        )

    def forward(self, x):
        return self.layers(x)
```

Figure 2: Spiral Architecture






training set
original
generator
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0