CS 763: LAB8

Group 10
Piyush Onkar(160050012) Apoorva Agarwal(203050018)

1. The Original GAN

Generator and Discriminator Models

```
Generator(
  (network): Sequential(
    (0): ConvTranspose2d(100, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(32, 1, kernel_size=(2, 2), stride=(2, 2), padding=(2, 2), bias=False)
    (13): Tanh()
  )
)
Discriminator(
  (network): Sequential(
    (0): Conv2d(1, 32, kernel_size=(2, 2), stride=(2, 2), padding=(2, 2), bias=False)
    (1): LeakyReLU(negative_slope=0.02, inplace=True)
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.02, inplace=True)
    (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.02, inplace=True)
    (8): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.02, inplace=True)
    (11): Conv2d(256, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (12): Sigmoid()
  )
)
```

Training

Discriminator Training

Code snippet for Discriminator Training:

```
optimizer_d.zero_grad()
# Discriminator training on real images
## TODO ##
# .view(-1) flattens the tensor
prediction_on_real = D(images.to(device)).view(-1)
batch_size = images.size(0)
real_label = torch.full((batch_size,), 1, dtype=torch.float, device=device)
loss_real = criterion(prediction_on_real,real_label)
## TODO ##
# Discriminator training on fake images
## TODO ##
noise_dim = 100
# randn gives random tensor that follow std. gaussian distribution
noise = torch.randn(batch_size, noise_dim, 1, 1, device=device)
fake_samples = G(noise)
fake_label = torch.full((batch_size,), 0, dtype=torch.float, device=device)
#detach removes tensor from gradient tracking
prediction_on_fake = D(fake_samples.detach()).view(-1)  # don't forget to use detach() here
loss_fake = criterion(prediction_on_fake,fake_label)
## TODO ##
loss_d = loss_real + loss_fake
loss_d.backward()
optimizer_d.step()
```

Explanation:
Discriminator is trained to correctly classify real and fake images. So loss is computed for predictions on real image and for predictions on fake image and used to optimize the discriminator.

Generator Training
Code snippet for Generator Training:

```
# Generator training
optimizer_g.zero_grad()
## TODO ##
fake_samples = fake_samples
prediction_on_fake = D(fake_samples).view(-1)

loss_g = criterion(prediction_on_fake,real_label)
## TODO ##
loss_g.backward()
optimizer_g.step()
```

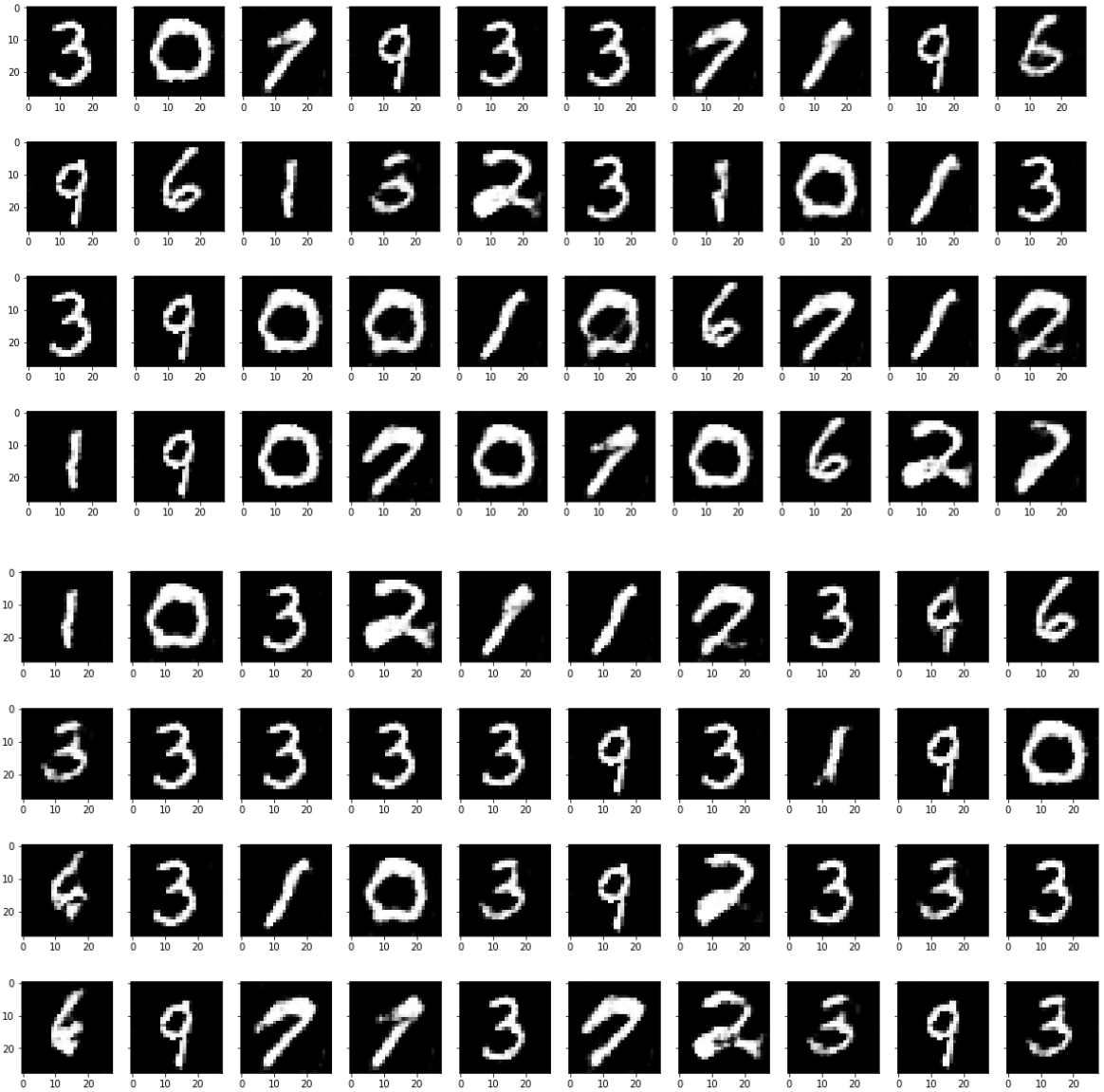
Explanation:
Generator is trained for fake images to be predicted as real by the discriminator.

Answers to questions
Q: Explain the nomenclature DC
Ans: DC is Deep Convolution in DCGAN. It signifies the use of convolution and transpose convolution layers in discriminator and generator respectively.

Q: What does the number 100 signify?
Ans: 100 signifies latent vector(noise vector) dimension. It is also the input size to the generator input layer.

Q What is the probability distribution of the output images generated by the GAN? Are all digits generated with equal probability? Speculate.
Ans: We generated 80 images . As labels are not given as input to model so the probability distribution of output images generated will depend on the input distribution. The generation of an image for a particular number is dependent on the complexity of the shape of the number. Therefore the distribution of digits generated is skewed .

Outputs generated



2. Conditional Wasserstein GAN

Generator and Discriminator Models

```
Generator(
  (network): Sequential(
    (0): ConvTranspose2d(110, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(32, 1, kernel_size=(2, 2), stride=(2, 2), padding=(2, 2), bias=False)
    (13): Tanh()
  )
)
Discriminator(
  (network): Sequential(
    (0): Conv2d(11, 32, kernel_size=(2, 2), stride=(2, 2), padding=(2, 2), bias=False)
    (1): LeakyReLU(negative_slope=0.02, inplace=True)
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.02, inplace=True)
    (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.02, inplace=True)
    (8): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.02, inplace=True)
    (11): Conv2d(256, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
)
```

Answers to questions

Q What is tricky about the original GAN?

We want the original GAN to produce a wide variety of outputs with almost equal probability for each digit. After generating more than 60 images for the above part, we observed that no image for digits 5 and 8 is generated. The generator is always trying to find the one output that seems most plausible to the discriminator.
Each iteration of generator over-optimizes for a particular discriminator, and the discriminator never manages to learn its way out of the trap.

Q Assume your assignment is successful. How would you use this in a business, real world setting?

1. We can use this to generate as many images for particular clothing.The designer can select some images generated based on design, pattern and appearance. He can then develop clothing products based on selected images. This can save a lot of his time as he doesn't have to design from scratch. He just has to work on improving the models generated by image.
2. The clothing business can put a lot of images on its website after generating the image from this assignment. The customer can select any clothing which he likes and the clothing store can create clothing based on the order placed by the customer. This would help customers choose between a wide variety of options and help businesses to effectively manage their inventory.

Q: Describe your process of conditioning.

Ans In Generator :

We took the size of the random vector to be equal to 100. We then appended one hot vector of label we wanted and passed the vector to Generator Model for generating images.
In Discriminator:
We first converted the image to 1X28x28.
One hot vector for the label is created. The shape of one hot vector is reshaped to 10x1.
We then repeated this vector to create a matrix of size 10x28x28.
We then appended this matrix to the reshaped image to generate matrix of size 11x28x28 and then passed this matrix to Discriminator Model.

Outputs generated