50.039 Theory and Practice of Deep Learning
Homework 11

Xie Han Keong
1003876
23 Apr. 21

Part 1

Copy and paste the code for your `Critic` *class. Briefly explain your choice of architecture.*

```
class Critic(nn.Module):
    def __init__(self, in_channels, num_features):
        """
        Parameters:
        in_channels (int) - Number of input channels for the first convolutional layer
        num_features (int) - Number of features for the first convolutional layer
        """
        super(Critic, self).__init__()

        # Output score for real or fake image
        out_channels = 1

        self.f = nn.Sequential(                                   # In: 28 x 28
            nn.Conv2d(in_channels, num_features, 3, 2, 1),        # 14 x 14
            nn.LeakyReLU(0.2),
            self.Conv(num_features, num_features*2, 3, 2, 1),    # 7 x 7
            self.Conv(num_features*2, num_features*4, 3, 2, 0),  # 3 x 3
            nn.Conv2d(num_features*4, out_channels, 3, 1, 1),    # 1 x 1
        )                                                         # Out: 1 x 1

    def Conv(self, in_channels, out_channels, kernel_size,
             stride=1, padding=0, dilation=1):
        return nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size,
                      stride, padding, dilation),
            nn.InstanceNorm2d(out_channels),
            nn.LeakyReLU(0.2),
        )

    def forward(self, x):
        return self.f(x).flatten(1)
```

I wrote the architecture for the `Critic` class based on concepts from DCGAN and WGAN-GP. Three convolutional layers are used with kernel size 3, stride 2, and output channels increasing by a factor of 2. This progressively downsamples the input from 1x28x28 to nx14x14 to 2nx7x7 to 4nx3x3, with n=32 giving the most plausible results after five experiments. A final convolutional layer with kernel size 3, stride 1, and 1 output channel is used to compress the output to a single dimension to predict if an image is real or not. In each convolutional block, `InstanceNorm2d` is used instead of `BatchNorm2d` (as recommended in WGAN-GP) to stabilize training whilst allowing the 1-Lipschitz constraint to be enforced by the gradient penalty. Also, to further stabilize training, `LeakyReLU` with negative slope 0.2 is used (as recommended in DCGAN) to avoid the problem of sparse gradients and strided convolutions are used for downsampling instead of pooling layers to increase the expressiveness of the Critic model.

Part 2

Copy and paste the code for your `Generator` *class. Briefly explain your choice of architecture.*

```python
class Generator(nn.Module):
    def __init__(self, in_channels, out_channels, num_features):
        """
        Parameters:
        in_channels (int) - Number of input channels for the first transposed convolutional layer
        out_channels (int) - Number of output channels for the last transposed convolutional layer
        num_features (int) - Number of features for the last transposed convolutional layer
        """
        super(Generator, self).__init__()

        self.G = nn.Sequential(                                          # In: 1 x 1
            self.ConvTranspose(in_channels, num_features*8, 3, 1, 0, 0),     # 3 x 3
            self.ConvTranspose(num_features*8, num_features*4, 3, 2, 0, 0),  # 7 x 7
            self.ConvTranspose(num_features*4, num_features*2, 3, 2, 1, 1),  # 14 x 14
            nn.ConvTranspose2d(num_features*2, out_channels, 3, 2, 1, 1),    # 28 x 28
            nn.Tanh(),
        )                                                                # Out: 28 x 28

    def ConvTranspose(self, in_channels, out_channels, kernel_size,
                      stride=1, padding=0, output_padding=0, dilation=1):
        return nn.Sequential(
            nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride,
                      stride, padding, dilation),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
        )

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

I wrote the architecture for the `Generator` class based on concepts from DCGAN and WGAN-GP as well. The architecture closely mirrors the `Critic` class—it uses three transposed convolutional layers with kernel size 3, stride 2, and output channels decreasing by a factor of 2. This progressively upsamples the input from 64x1x1 to 8nx3x3 to 4nx7x7 to 2nx14x14, with n=32 as well. A final transposed convolutional layer with kernel size 3, stride 2, and 1 output channel is used to convert the output into 1x28x28—the correct dimension for an MNIST image. `Tanh` is used as the final activation function to map the outputs to the range of [-1, 1]. `BatchNorm2d` and `ReLU` are used in each transposed convolutional block for similar reasons to the Critic.

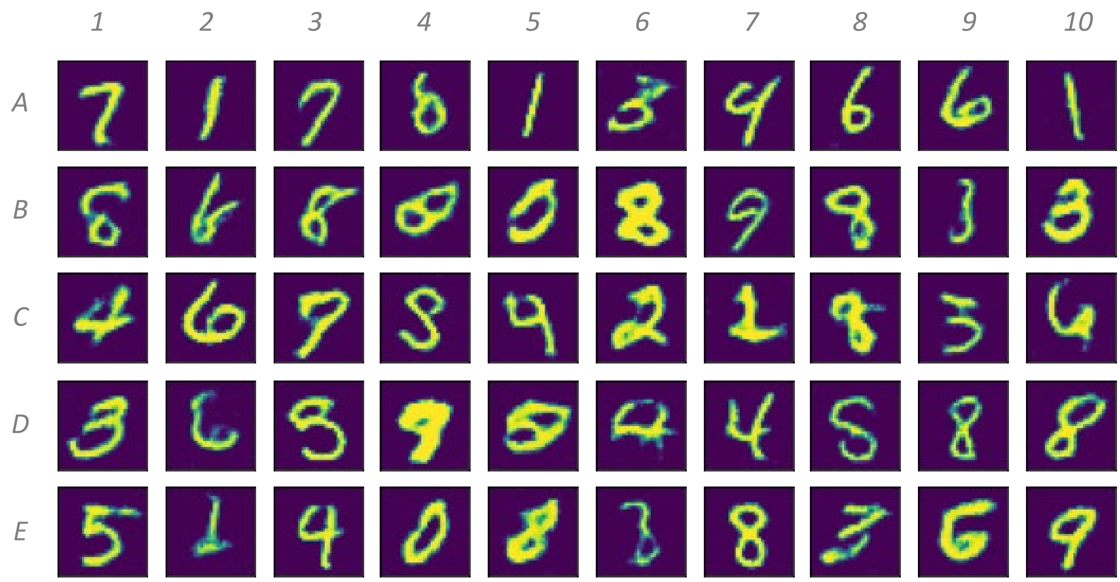
Part 3

For how many iterations did you have to train when using Wasserstein with `Conv2d`/`ConvTranspose2d` *layers to get plausible images from the generator? Is it training faster than the Fully Connected Wasserstein/Vanilla GAN?*

In my fifth and final experiment, I used 20 epochs and a batch size of 128 for a total of 9,380 iterations to train the WGAN model to get plausible images from the generator. That is much faster than the Fully Connected WGAN/GAN, which needed 300 epochs with a batch size of 32 for a total of 562,500 iterations to get some plausible images from the generator.

Part 4

Display some samples generated by your trained generator. Do they look plausible?



From this sample of 50 images, it can be observed that most of them are quite plausible. In fact, there is at least one plausible example for each digit from 0-9: B5, A2, C6, D1, E3, E1, A8, A1, E7, and E10 (respectively). However, there are still some images that do not look plausible. For example, B1, B2, D2, D5, D6, D8, and E6.

Part 5

Let us assume we use `Conv2d` *layers in the Critic. We do NOT use* `ConvTranspose2d` *layers, but only Fully Connected layers in the Generator. Would the GAN still be able to train both models or would it encounter difficulties? Discuss.*

The GAN would encounter difficulties in training the Generator. This is because the Critic will be much more expressive than the Generator as convolutional layers are able to learn more intricate features than fully connected layers. In a deep convolutional neural network, the bottom layers are able to pick up low-level features such as curves and edges, and the top layers are able to learn high-level features such as shapes and patterns. Fully-connected layers are not able to capture these granular details and can only 'see' general patterns across the entire image. This imbalance will cause the Generator to take a much longer time to learn how to 'outsmart' the Critic.

References

Arturml. (2019, March 21). Arturml/pytorch-wgan-gp. Retrieved April 23, 2021, from https://github.com/arturml/pytorch-wgan-gp/blob/master/models.py

Brownlee, J. (2019, September 11). Tips for training stable generative adversarial networks. Retrieved April 23, 2021, from https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/

DCGAN implementation from scratch. (2020, November 02). Retrieved April 23, 2021, from https://www.youtube.com/watch?v=IZtv9s_Wx9I

GANs in Action. (n.d.). Chapter 4: DCGAN. Retrieved April 23, 2021, from https://github.com/GANs-in-Action/gans-in-action/blob/master/chapter-4/Chapter_4_DCGAN.ipynb

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017, December 25). Improved Training of Wasserstein GANs. Retrieved April 23, 2021, from https://arxiv.org/abs/1704.00028

Radford, A., Metz, L., & Chintala, S. (2016, January 07). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. Retrieved April 23, 2021, from https://arxiv.org/abs/1511.06434

Shorten, C. (2019, May 09). DCGANs (Deep Convolutional Generative Adversarial Networks). Retrieved April 23, 2021, from https://towardsdatascience.com/dcgans-deep-convolutional-generative-adversarial-networks-c7f392c2c8f8

Soumith. (n.d.). Ganhacks. Retrieved April 23, 2021, from https://github.com/soumith/ganhacks

WGAN implementation from scratch (with gradient penalty). (2020, November 03). Retrieved April 23, 2021, from https://www.youtube.com/watch?v=pG0QZ7OddX4