50.039 Theory and Practice of Deep Learning - Homework 6

tags: `SUTD` `Theory and Practice of Deep Learning` `Homework`

Done by: Lin Huiqing (1003810)

Web view: https://hackmd.io/@ephemeral-instance/rJADkIQ8d

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

```
class Critic(nn.Module):

    def __init__(self, image_size):
        """
        Only forced parameter will be the image size, set to 28.
        """
        super(Critic, self).__init__()
        std_kern_size = image_size // 3
        final_kern_size = image_size - 2 * std_kern_size

        self.conv1 = torch.nn.Conv2d(1, 64, std_kern_size+1)
        self.conv2 = torch.nn.Conv2d(64, 64, std_kern_size+1)
        self.conv3 = torch.nn.Conv2d(64, 1, final_kern_size)

        self.leaky_relu = torch.nn.LeakyReLU(0.2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.leaky_relu(x)
        x = self.conv2(x)
        x = self.leaky_relu(x)
        x = self.conv3(x)
        x = torch.flatten(x, 1)
        return x
```

The Critic model should be able to take in the mini-batch of images, represented by an input tensor of shape [32, 1, 28, 28] and then return the predictions in the form of an output tensor with shape [32, 1].

To do so, the model downsamples the images through convolution layers. Kernel sizes are chosen such that the extent to which the tensor is downsampled is roughly equal throughout the 3-layers of the model. Using the image size, the calculated kernel size to do so is **10** for all layers. The initial input and final output channels should be **1** due to the expected input and output dimensions of the model. For input and output channels of hidden layers, this is set to a relatively high number of **64** to learn as many different features as possible.

LeakyReLU was used as the activation function between layers due to the following reasons:

1. Allows the model to learn even when negative values are calculated, which fixes the "dying ReLU" problem.
2. Speeds up training as it keeps the mean activation close to zero.

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

```
class Generator(nn.Module):

    def __init__(self, latent_size, image_size):
        """
        Only forced parameters will be the image size, set to 28,
        and the latent size set to 64.
        """
        super(Generator, self).__init__()
        std_kern_size = image_size // 3
        final_kern_size = image_size - 2 * std_kern_size

        self.transconv1 = torch.nn.ConvTranspose2d(latent_size, 64, std_kern_size)
        self.transconv2 = torch.nn.ConvTranspose2d(64, 64, std_kern_size)
        self.transconv3 = torch.nn.ConvTranspose2d(64, 1, final_kern_size)

        self.leaky_relu = torch.nn.LeakyReLU()
        self.tanh = torch.nn.Tanh()

    def forward(self, x):
        x = self.transconv1(x)
        x = self.leaky_relu(x)
        x = self.transconv2(x)
        x = self.leaky_relu(x)
        x = self.transconv3(x)
        x = self.tanh(x)
        return x
```

The Generator model should be able to take in a batch of noise samples, represented by a tensor of shape [32, 64, 3, 3], and output an image represented by a tensor of the same shape [32, 1, 28, 28].

To do so, the model upsamples the images through transposed convolution layers. Kernel sizes are chosen such that the extent to which the tensor is upsampled is roughly equal throughout the 3-layers of the model. Using the image size, the calculated kernel size to do so is **9** for the first 2 layers, followed by **10** for the last layer. The initial input and final output channels should be the latent size and **1** respectively due to the expected input and output dimensions of the model. For input and output channels of hidden layers, this is set to a relatively high number of **64** to learn as many different features as possible.

LeakyReLU was used as the activation function between layers due to the following reasons:

1. Allows the model to learn even when negative values are calculated, which fixes the "dying ReLU" problem.
2. Speeds up training as it keeps the mean activation close to zero.

Tanh was used as the final activation function as it has been observed by Radford et al. that the model can learn faster to saturate and cover the colour space of the training distribution with a bounded activation function.

3. For how many iterations did you have to train when using Wasserstein with Conv/TransposeConv layers to get plausible images from the generator? Is it training faster than the Fully Connected Wasserstein/Vanilla GAN?

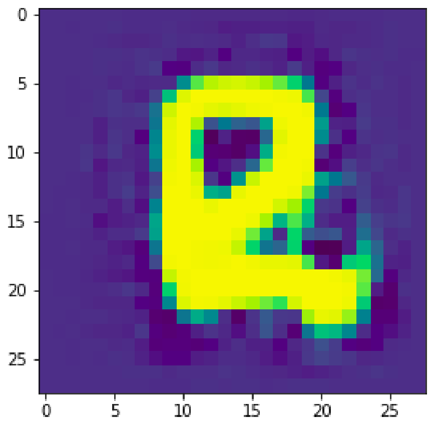
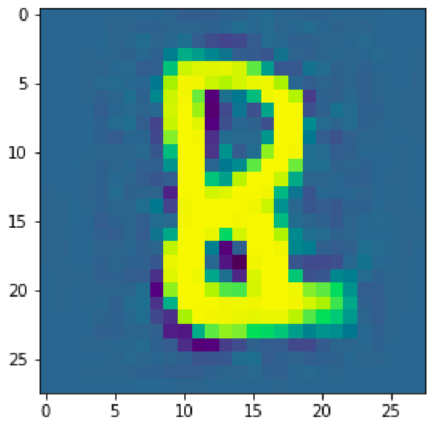
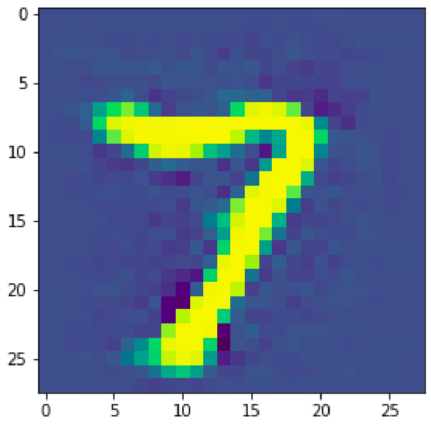
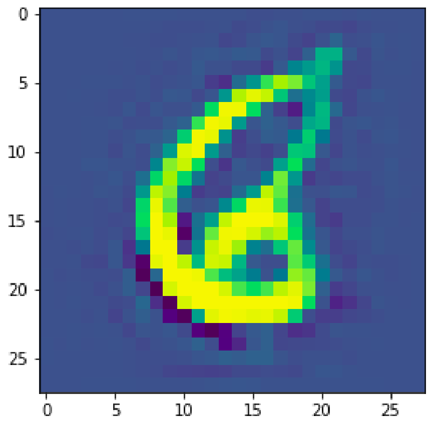
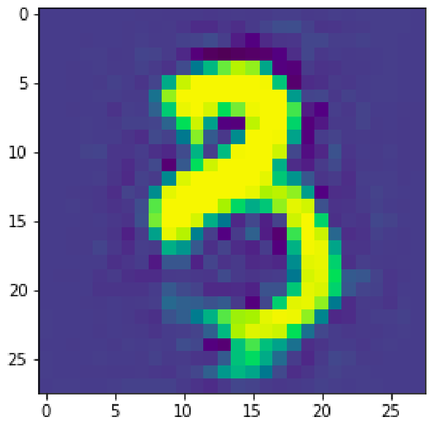
It takes around 60 epochs to get plausible images from the generator.

Yes, the model with Conv/TransposeConv layers is training faster than the model with Fully Connected layers.

4. Display some samples generated by your trained generator. Do they

look plausible?

Below are some samples generated by the trained generator:

Although the first image does not look plausible, the other images do. The 2nd, 3rd, 4th and 5th images look like the numbers 8, 7, 6 and 3 respectively.

5. Let us assume we use Conv2d layers in the Critic. We do NOT use Transposed Conv2d 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.

As Fully Connected layers are not able to use local dependencies when generating images. This means that the Generator would take much longer to train as there are many more weights to train compared to the Conv2d layers which the Critic would be using. Due to imbalanced training, the Critic would have no issues differentiating real and fake images as well, meaning that not much training would occur.

References

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv *preprint arXiv:1511.06434.*