Assignment 3 – WGAN-GP

Due Date: Nov 18, 2022 Midnight

In this assignment we will implement a Generative Adversarial Network using the Wasserstein GAN and Gradient Penalty (WGAN -GP) to generate samples like the digits in the MNIST dataset. MNIST is a popular digit dataset that is available with most of the deep learning frameworks like PyTorch, TensorFlow, etc. Or, feel free to use the `digits.py` file provided with Assignment 1 to download the MNIST dataset. It consists of 60,000 grayscale images, each 28x28 pixels. The dataset has 10 categories of digits, {0,1,…,9}. **Normalize the pixel values to [-1,1]**. Use the entire dataset of 60,000 images to train the GAN. The architecture of the neural network is as follows:

1. The Generator takes in a vector $z \in \mathbb{R}^d$ which is sampled from a Gaussian distribution $N(0,1)$ with $d = 100$. The Generator is a fully connected neural network with the following architecture:

   z → 256 → 512 → 1024 → 784

   Use a Leaky ReLU as activation for the layers (for e.g., LeakyReLU with negative slope 0.2), expect for the last layer, where you use a tanh() activation. The output of the last layer (784 dimensions) is converted to an image 28 x 28 pixels. The output of the Generator is $G(z)$

2. The Discriminator is also a fully connected network with LeakyReLU activations except for the last layer, which does not have any activation. The architecture is as follows:

   784 → 512 → 256 → 1

   The Discriminator takes as input real MNIST images x and generated MNIST images (output of the Generator $G(z)$) and outputs a scalar value $D(x)$ and $D(G(z))$ respectively.

3. The objective function for the WGAN-GP is given by:

   $$\min_G \max_D \mathbb{E}_{x\sim P_r(x)}[D(x)] - \mathbb{E}_{z\sim P(z)}[D(G(z))] + \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2]$$

   The first two terms are the WGAN loss where x is sampled from real data $P_r(x)$ and $z \in \mathbb{R}^d$ is the input to the Generator and is sampled from a Gaussian distribution $N(0,1)$ with $d = 100$. The 3rd term is the gradient penalty to ensure $D(.)$ is Lipschitz. $\hat{x}$ is a linear interpolation between real and generated samples, i.e., $\hat{x} = \epsilon * x + (1 - \epsilon) * G(z)$ and $\nabla_{\hat{x}} D(\hat{x})$ is the derivative of $D(\hat{x})$ w.r.t. $\hat{x}$ and $\epsilon \in (0,1)$.

4. Tips to implement the WGAN-GP:
   Gradient Penalty:

a. Implement a function
`get_gradient_penalty(discrimniator, real_images, gen_images, lambda=10)`
This function takes as input the discriminator ($D(.)$), real_images (x) and gen_images ($G(z)$) and returns the gradient_penalty (3rd term of the objective function)
b. Create $\hat{x} = \epsilon * x + (1 - \epsilon) * G(z)$
c. Use the autograd function to estimate the gradient $\nabla_{\hat{x}} D(\hat{x})$ – For e.g., in PyTorch the autograd function is given by:
```
gradients = autograd.grad(inputs=x̂, outputs=D(x̂),
        grad_outputs=torch.ones(D(x̂).size()).to(device),
         create_graph=True, retain_graph=True,
        only_inputs=True)[0]
```
d. Calculate `gradient_penalty = ((gradients.norm(2, dim=1) - 1) ** 2).mean() * lambda`

Discriminator Training:
a. Generate a batch of samples using the Generator – $G(z)$.
b. Detach the generated images (For e.g., $G(z)$.detach() in PyTorch) because we do not want to update the generator using the Discriminator loss.
c. Sample a batch of images from the dataset – x.
d. Calculate mean $D(G(z))$.mean()
e. Calculate mean $D(x)$.mean()
f. The Discriminator is optimized using the loss
D_loss = $D(G(z))$.mean() - $D(x)$.mean() + gradient_penalty
g. Estimate D_loss.backward() and optimize the Discriminator, for example using Adam optimizer.

Generator Training:
a. Generate a batch of samples using the Generator – $G(z)$.
b. Calculate mean $D(G(z))$.mean()
c. The Generator is optimized using the loss
G_loss = $-D(G(z))$.mean()
d. Estimate G_loss.backward() and optimize the Generator, for example using Adam optimizer.

Submission Format and Grading:
1. Implement the assignment in Python and submit a Jupyter Notebook file with the following name format `Assignment3_FirstName_LastName.ipynb`
2. Convert your notebook to pdf after saving it with all the outputs and save the file as `Assignment3_FirstName_LastName.pdf`
3. Do not save images of your code in the pdf file. The pdf file has to run through the plagiarism check and code needs to be in text format, not image.
4. Generator implemented with the prescribed architecture (20 pts)
5. Discriminator implemented with the prescribed architecture (20 pts)
6. Gradient Penalty (20 pts)

7. When executed the notebook should output the following:
    a. Training curves (Discriminator/Generator loss vs epochs) (20 pts)
    b. A 10 x 10 grid plot of 100 randomly generated images which are the outputs of the Generator. (20 pts)

Your submissions <u>will be executed</u> to verify your solutions. The code and report will be evaluated for plagiarism. Ensure the notebook will execute on a generic Google Colab environment without the need for change/debugging. If it is not possible to execute your code, it is not possible to verify your results. The submission will also be evaluated for the quality of images in each of these outputs.