BME646 and ECE60146: Homework 7

Spring 2023
Due Date: 11:59pm, Apr 05, 2023
TA: Fangda Li (li1208@purdue.edu)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. Late submissions will be accepted with penalty: **-10 points per-late-day, up to 5 days**.

1 Introduction

Got pizza? The goal for this HW is to create your own pizza-generating Generative Adversarial Networks (GAN). The learning objectives are:

1. Understand how the generator and discriminator networks are trained to compete against each other in a minimax game.

2. Experiment with two different GAN learning criteria: Binary Cross-Entropy (BCE) and the Wasserstein distance.

3. Evaluate your generated images qualitatively, and quantitatively using the Fréchet Inception Distance (FID).

2 Getting Ready for This Homework

Before embarking on this homework, do the following:

1. Go through Slide 28 through 44 of the Week 9 slide deck on Semantic Segmentation [3] and develop a good understanding of the concept of what is meant by Transpose Convolution.

2. Also go through the Slide 43 through 51 of the same set of Week 9 slides to fully understand the relationship between the Kernel Size, Padding, and Output Size for Transpose Convolution. Make sure you understand the example shown on Slide 44 in which a 4-channel 1×1 noise vector is expanded into a 2-channel 4×4 noise image. This example is foundational to designing the Generator side of a GAN.

3. Understand the GAN material on Slide 60 through 77 of the Week 11 slide deck on "Generative Adversarial networks" [2]. For additional

depth, you may wish to read the original GAN paper by Goodfellow et al. [5]:

https://arxiv.org/pdf/1406.2661.pdf

4. When you are learning about a new type of a neural network, playing with an implementation by varying its various parameters and seeing how that affects the results can often help you gain deep insights in a short time. If you believe in that philosophy, execute the following the script in the `ExamplesAdversarialLearning` directory of DLStudio:

   ```
   python dcgan_DG1.py
   ```

   It uses the `PurdueShapes5GAN` dataset that is described on Slide 56 through 61 of the Week 11 slides. Instructions for downloading this dataset are on the main DLStudio webpage.

5. For understanding the Wasserstein distance, you need to first read the explanation on Slide 38 through 43 of the Week 11 slides. Make sure you understand the 1-Lipschitz condition for imposing smoothness constraints on the distance function. Now go over Slide 92 through 97 to understand how to create the Critic part of a Critic-Generator pair for estimating the Wasserstein distance. Finally, look over Slide 100 for how the 1-Lipschitz condition is actually implemented in code.

   To play with the Wasserstein-GAN in DLStudio yourself, execute the following script: `wgan_CG1.py`, or `wgan_CG2.py` if you are also interested in applying the gradient penalty. For a good alternative source of reference on how Wasserstein GAN with gradient penalty can be implemented, you can read the `networks.py` file from [1].

3 Programming Tasks

3.1 Building and Training Your GAN

Here are the steps for making your own (fake) pizza:

1. Before starting, make sure you have downloaded the provided pizza dataset from BrightSpace along with this handout. The dataset contains 8k+ images for training and 1k images for evaluation, all resized to 64×64. Example images are shown in Figure 1.

Figure 1: Real pizzas.

2. Your first task in this homework is to conjure up your own generator and discriminator networks. Just like the previous homeworks, you have total freedom on how you design your networks. The only network-building requirement is that your generator has to be able to generate RGB pizza images of size 64×64 from random noise vectors and must do so while utilizing *transposed convolutions*.

3. Subsequently, you'll need to write your own adversarial training logic. You can refer to Slide 64 through 69 of the Week 11 slides to familiarize yourself with how it can be done. For this HW, we ask you to experiment with two different adversarial learning criteria: the Binary Cross-Entropy (BCE) loss as originally used by Goodfellow *et al.* in [5], and the Wasserstein distance as introduced by Arjovsky *et al.* in [4]. You should train two GANs in total, one with the BCE loss and another with the Wasserstein distance. In the rest of this handout, we shall call them the BCE-GAN and the W-GAN, respectively.

   Note that for the purpose of this HW, we do *not* require you to enforce the 1-Lipschitz constraint on your Critic for W-GAN. For enforcing that constraint, authors of the original W-GAN used the heuristics of weight clipping (as mentioned on Slide 92), while that is generally replaced later by Gradient Penalty [6] (Slide 106 through 108).

4. In your report, plot the adversarial losses over training iterations for both the generator and the discriminator in the same figure. Note that

you should make two separate figures for BCE-GAN and W-GAN.

3.2 Evaluating Your GAN

Here are the steps for evaluating your GANs:

1. First, you should generate 1k images of fake pizza from randomly sampled noise vectors using your trained generator (BCE-GAN or W-GAN).

2. For evaluating your generated images quantitatively, you will use the Fréchet Inception Distance (FID). Originally proposed in [7], the FID is a widely used metrics for measuring both the quality and the diversity of GAN-generated images. More specifically, it does so by measuring how close the distribution of the fake images is to the distribution of the real images. To calculate the FID, one would first encode the set of real images into feature vectors using a pretrained Inception network, and then model the resulting distribution of feature vectors using a multivariate Gaussian distribution. The same is carried out for the set of fake images. Once that is done, the FID is simply the Fréchet distance between the two multivariate Gaussians.

3. For this homework, you will be using the `pytorch-fid` package [8] for calculating the FIDs. To install the package, use the command:

   ```
   pip3 install pytorch-fid
   ```

   Once installed, you can use the `pytorch-fid` package in a Python script as follows:

   ```python
   from pytorch_fid.fid_score \
       import calculate_activation_statistics, \
       calculate_frechet_distance
   from pytorch_fid.inception import InceptionV3

   real_paths = ['/real/0.jpg', '/real/1.jpg', ...]
   fake_paths = ['/fake/0.jpg', '/fake/1.jpg', ...]
   dims = 2048
   block_idx = InceptionV3.BLOCK_INDEX_BY_DIM[dims]
   model = InceptionV3([block_idx]).to(device)
   m1, s1 = calculate_activation_statistics(
       real_paths, model, device=device)
   m2, s2 = calculate_activation_statistics(
       fake_paths, model, device=device)
   fid_value = calculate_frechet_distance(m1, s1, m2, s2)
   print(f'FID: {fid_value:.2f}')
   ```

4. In your report, for qualitative evaluation, display a 4×4 image grid, similar to what is shown in Figure 1, showcasing images randomly generated by your BCE-GAN. Also display the same with images by your W-GAN. You might find the functions in `torchvision.utils` really helpful here. For quantitative evaluation, present the FID values for both GAN variants. Finally, include a paragraph discussing your results: BCE-GAN v.s. W-GAN, which is better?

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Your pdf must include a description of
   - The figures and descriptions as mentioned in Sec. 3.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up.
2. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw7_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.
3. **Make sure your submission zip file is under 10MB.** Compress your figures if needed.
4. **Do NOT submit your network weights.**
5. For all homeworks, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert it to `.py` and submit that as source code.
6. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.
7. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

8. To help better provide feedbacks to you, make sure to **number your figures**.

References

[1] pytorch-CycleGAN-and-pix2pix. URL https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.

[2] Generative Adversarial Networks for Data Modeling, . URL https://engineering.purdue.edu/DeepLearn/pdf-kak/GAN.pdf.

[3] Encoder-Decoder Architectures for Semantic Segmentation of Images, . URL https://engineering.purdue.edu/DeepLearn/pdf-kak/SemanticSeg.pdf.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[8] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.3.0.