Image Generation with GAN

GAN is a framework for teaching a DL model to capture the training data's distribution. It is made of two distinct models, a *generator* and a *discriminator*. The job of the generator is to spawn 'fake' images that look like the training images. The job of the discriminator is to look at an image and predict whether it is a realistic training image or a fake image from the generator. During training, the generator tries to outsmart the discriminator by generating better and better fakes, while the discriminator is working to become a better detective and correctly classify the real and fake images. The equilibrium of this game is when the generator is generating perfect fakes that look as if they came directly from the training data, and the discriminator always guesses at 50% confidence that the generator output is real or fake.

A DCGAN is a direct extension of the GAN, except that it explicitly uses convolutional and transposed convolutional layers in the discriminator and generator, respectively. **In this homework, we implement a DCGAN.**

Discriminator

The discriminator, D, is a binary classification network that takes an image as input and outputs a scalar probability that the input image is real (as opposed to fake). Here, D takes an input image, processes it through a series of Conv2d, BatchNorm2d, and LeakyReLU layers, and outputs the final probability through a Sigmoid activation function. This architecture can be extended with more layers if necessary, but there is significance to the use of the strided convolution, BatchNorm, and LeakyReLUs. The DCGAN paper mentions it is a good practice to use strided convolution rather than pooling to downsample because it lets the network learn its own pooling function. BatchNorm and LeakyRelu functions also promote healthy gradient flow, which is critical for the learning process of both G and D.

Generator

The generator, G, is designed to transform a latent vector (z, drawn from a unit Gaussian distribution $\mathcal{N}(0, I)$) to an image with the same size as the training images. In practice, this is accomplished through a series of strided two-dimensional transposed convolutional layers, each paired with a 2d BatchNorm layer and a Relu activation. The output of the generator is fed through a tanh function to return it to the input data range of $[-1, 1]$. It is worth noting the existence of the BatchNorm functions after the conv-transpose layers, as this is a critical contribution of the DCGAN paper. These layers help with the flow of gradients during training.

Weight Initialization

From the DCGAN paper, the authors specify that all model weights shall be randomly initialized from a Gaussian distribution with mean=0, stdev=0.02.

Loss Function

The loss function is:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_g(z)}[\log(1 - D(G(z)))] \tag{1}$$

Training

Training is split up into two main parts. Part 1 updates the Discriminator, and Part 2 updates the Generator.

- Part 1: Practically, we want to maximize ($\log D(x) + \log(1 - D(G(z)))$). **Firstly**, we will construct a batch of real samples from the training set, forward pass through D, calculate the loss ($\log D(x)$), then calculate the gradients in a backward pass. **Secondly**, we will construct a batch of fake samples with the current generator, forward pass this batch through D, calculate the loss ($\log(1 - D(G(z)))$), and *accumulate* the gradients with a backward pass. (*accumulate* means that we do not zero the gradients between two backward passes.)
- Part 2: As stated in the original paper, we want to train the Generator by minimizing ($\log(1 - D(G(z)))$) to generate better fakes. However, the objective suffers from the gradient vanishing problem. **As a fix, we maximize ($\log D(G(z))$) instead.** (This is the log-D trick in our slides.)

To keep track of the generator's learning progress, we will generate a fixed batch of latent vectors drawn from a Gaussian distribution (i.e., `fixed_noise`). In the training loop, we will periodically feed this fixed_noise into G, and we will see images formed out of the noise over the iterations.

Requirements

- Python 3
- PyTorch >= 1.1
- torchvision
- tensorboard
- scipy < 1.4 // make sure spicy < 1.4 is installed, otherwise computing FID score will be extremely slow (refer to https://zhuanlan.zhihu.com/p/110053716 for more details)

Dataset Description

`dataset.py` defines a class to load the MNIST data and transform the size of each image from 28×28 to $1 \times 32 \times 32$. **Pixel values are normalized in $[-1, 1]$.**

Download MNIST dataset and put all files under `codes/data/`.

Download Inception model and put the file under `codes/inception/`

Python Files Description

In this homework, we provide unfinished implementation of DCGAN using **PyTorch**. The code structure is as follows:

- `main.py`: the main script for running the whole program.
- `GAN.py`: the main script for model implementation, including some utility functions.
- `dataset.py`: the script that loads the data
- `trainer.py`: the script that trains a generative model

Usage

- Training: `python main.py --do_train`. It will load the latest checkpoint (initialize if not existed) and run the training process. After training, the model will be evaluated by FID.
- Test: `python main.py`. It will load the latest checkpoint (initialize if not existed) and evaluate the model by FID.
- Tensorboard: `tensorboard --logdir ./runs`. It will open a web server to show the training curves.

DCGAN

You are supposed to:

1. Implement the Generator of the following architecture in `GAN.py`. BN2d denotes BatchNorm2d. Both `latent-dim` and `hidden-dim` are hyper parameters of the Generator. **In this homework, the default values of `latent-dim` and `hidden-dim` are 16.**

   The input latent variable z is of shape $[\text{latent-dim}, 1, 1]$

Layer id	Layer type	Output volume	Kernel size	Stride	Padding
1	Transposed conv + BN2d + ReLU	4*hidden-dim $\times 4 \times 4$	4	1	0
2	Transposed conv + BN2d + ReLU	2*hidden-dim $\times 8 \times 8$	4	2	1
3	Transposed conv + BN2d + ReLU	hidden-dim $\times 16 \times 16$	4	2	1
4	Transposed conv + Tanh	$1 \times 32 \times 32$	4	2	1

2. Implement the `train_step()` function in `trainer.py`. **See the comments in codes for some hints.**

   You are required to average the losses over all instances in a batch.

Report

In the experiment report, you need to answer the following questions:

1. Train a GAN with default hyper-parameters. Increase either `latent_dim` or `hidden_dim` from 16 to 100. How about changing both to 100? (**You need to run at least four experiments in this homework.**)
   1. The code we provide will plot a number of graphs (e.g. loss curves) during training on the Tensorboard. Show them in your report. (5 curves for a GAN. Make sure that TAs can see each curves clearly.)
   2. The code we provide will also paint some images on the Tensorboard. Show the model-generated images **at the last record step**. (One group of images for a GAN. Make sure that TAs can see each graph clearly.)
   3. **State the experimental settings of each curve/image clearly, which include `latent_dim`, `hidden_dim`, `batch_size`, and `num_train_steps`.**
2. We use FID score to evaluate the quality of model-generated images. FID score measures the similarity between model-generated images and real images. Specifically, we extract features from each image with a well-trained model, fit a multivariate Gaussian to the feature vectors of generated images and real images separately, and compute the similarity between the two Gaussians. **A lower FID score means better performance.** For more details, refer to the reference[2].

Report FID scores of the four trained GANs mentioned above. (The best GAN's FID should be lower than 60.)

3. Discuss how `latent_dim` and `hidden_dim` influence the performance of a GAN.
4. Derive the optimal solution for discriminator. Has your GAN converged to the Nash equilibrium? Give your explanation with your training curves.
5. Interpolation in the latent space is a common way to qualitatively evaluate how well a GAN learns the latent representations. Linear interpolation can be conducted by first sampling two latent points denoted as z_1 and z_2, and then decoding an image from $z_1 + \frac{i}{K}(z_2 - z_1)$ for $i \in \{0, 1, \ldots, K\}$ (One can also try setting i to a value bigger than K or smaller than 0 to evaluate its generalization, which is the case of linear extrapolation.)

   Choose the best GAN, add some codes and apply linear interpolation in the latent space. Show the interpolated images (10 images for a pair z_1 and z_2, at least 5 pairs) and describe what you see (the connections between interpolated images and the generation performance).
6. **[bonus point: no more than 1] [Mode Collapse]** Choose one of your GANs and investigate whether it suffers from the mode collapse problem. You should inspect at least 50 randomly generated samples and count the numbers of images that contain each digit. For example, your GAN may generate 10 images containing the '0' digit, 5 images containing the '1' digit , and so on. You can manually do the labeling or use the MNIST classifier in HW1. Finally, give your conclusion (whether the GAN suffers from the mode collapse) and explanations based on your statistics.
7. **[bonus point: no more than 1] [Ablation]** Implement a GAN with an MLP-based generator and discriminator. Report your FID score and show no less than 10 generated images. Repeat the process by trying either one of the following under the DCGAN implementation:

   1. Replace the LeakyReLU implementation with ReLU
   2. Remove Batchnorm layer in both Discriminator and Generator

   Finally, compare their performance with the CNN-based GAN here and rank the modification influence.

NOTE: Since the training time may be long, you are not required to tune the hyper-parameters. The default hyper-parameters are enough to produce reasonable results (if you have implemented the model correctly, especially for the loss, which should be averaged over the batch).

NOTE: The bonus point will not exceed 2.

Code Checking

We introduce a code checking tool this year to avoid plagiarism. You **MUST** submit a file named `summary.txt` along with your code, which contains what you modified and referred to. You should follow the instructions below to generate the file:

1. Fill the codes. Notice you should only modify the codes between `# TODO start` and `# TODO end`, the other changes should be explained in `README.md`. **DO NOT** change or remove the lines start with `# TODO`.
2. Add references if you use or refer to a online code, or discuss with your classmates. You should add a comment line just after `# TODO start` in the following formats:

1. If you use a code online: `# Reference: https://github.com/xxxxx`
2. If you discuss with your classmates: `# Reference: Name: Xiao Ming Student ID: 2018xxxxxx`

   You can add multiple references if needed.

   Warning: You should not copy codes from your classmates, or copy codes directly from the Internet, especially for some codes provided by students who did this homework before. In all circumstances, you should at least write more than 70% codes. (You should not provide your codes to others or upload them to Github before the course ended.)

   警告:作业中不允许复制同学或者网上的代码，特别是往年学生上传的答案。我们每年会略微的修 改作业要求，往年的答案极有可能存在错误。一经发现，按照学术不端处理(根据情况报告辅导员或学校)。在任何情况下，你至少应该自己编写70%的代码。在课程结束前，不要将你的代码发送给其他人或者上传到github上。

3. Here is an example of your submitted code:

```
def forward(self, input):
    # TODO START
    # Reference: https://github.com/xxxxx
    # Reference: Name: Xiao Ming Student ID: 2018xxxxxx your codes...
    # TODO END
```

4. At last, run python `./code_analyze/analyze.py` , the result will be generated at `./code_analyze/summary.txt`. Open it and check if it is reasonable. A possible code checking result can be:

```
######################## # Filled Code ######################## #
..\codes\layers.py:1
# Reference: https://github.com/xxxxx
# Reference: Name: Xiao Ming Student ID: 2018xxxxxx your codes...
########################
# References
########################
# https://github.com/xxxxx
# Name: Xiao Ming Student ID: 2018xxxxxx
########################
# Other Modifications
########################
# _codes\layers.py -> ..\codes\layers.py
# 8 - self._saved_tensor = None
# 8 + self._saved_tensor = None # add some thing
```

Submission Guideline

You need to submit both report and codes, which are:

- **Report**: well formatted and readable summary including your results, discussions and ideas. Source codes should not be included in report writing. Only some essential lines of codes are permitted for explaining complicated thoughts.
- **Codes**: organized source code files with README for **extra modifications** (other than `TODO`) or

specific usage. Ensure that TAs can successfully reproduce your results following your instructions. **DO NOT include model weights/raw data/compiled objects/unrelated stuff over 50MB.**
- **Code Checking Result**: You should only submit the generated `summary.txt`. **DO NOT** upload any codes under `code_analysis`. However, TAs will regenerate the code checking result to ensure the correctness of the file.

You should submit a `.zip` file name after your student number, organized as below:

- `Report.pdf/docx`
- `summary.txt`
- `codes/`
  - `GAN/`
    - `*.py`
    - `README.md/txt`

Deadline: Nov. 15th

TA contact info:

- Zhuoer Feng (冯卓尔), fze22@mails.tsinghua.edu.cn

Reference

[1] Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434, 2015

[2] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. arXiv preprint arXiv:1706.08500,2017