实验报告

181840013 陈超

1 基本配置

1. 语言：Pytorch
2. GPU：Kaggle 平台提供的 NVIDIA Tesla P100
3. 数据集：51223 个二次元小姐姐的图片 (i 了 i 了)。

(食用本篇实验报告前建议先阅读第四部分)

2 GAN

2.1 原理

GAN 网络 (Generative adversarial network)，原理可以说是相当精妙，针对给定的 source data，定义两个网络，G(Generator) 和 D(Discriminator)。Generator 是一个生成图片的网络，它接收一个随机的噪声 z，通过这个噪声生成图片，记做 G(z)。Discriminator 是一个判别网络，判别一张图片是不是“真实的”。它的输入是 x，x 代表一张图片，输出 D(x) 代表 x 为真实图片的概率，如果为 1，就代表是真实的图片，输出为 0，则代表不可能是真实的图片。

训练时分为两步，首先更新 D，其 loss 的计算方式是 G 把真实图片判断为非真实图片的误差，加上把非真实图片判断为真实图片的误差。loss 计算完进行反向传播的时候需要注意，由噪声 z 计算 G(z) 得到的 fake_x 在传入 D 之前需要先 detach，因为计算得到 fake_x 的梯度与 D 无关，不要要加入计算图内。之后我们再来更新 G，其 loss 就等于非真实图片 fake_x 有多少被 D“发现”了，即被判断为非真实。如此每轮训练都一次更新 G 和 D，就可以逐步优化 G 和 D，让 G 能够生成更逼真的图像，D 对真实和非真实的图像有更强的鉴别能力 [1]。

2.2 代码分析

代码主要分成三个部分：数据读取及预处理，网络构建，网络训练。

2.2.1 数据读取及预处理

对应代码 data loading and preprocessing 注释部分，主要是将图像 resize 到 64×64 的大小并进行标准化。

2.2.2 网络构建

对应代码 net constructing 注释部分，参考了从《Dive into Deep Learning》一书的网站：http://www.d2l.ai/下载的 d2l-en 代码包 [5]。整个网络和 DCGAN 有一些相似。下面分成 Generator 和 Discriminator 分别介绍。

Generator 有 4+1 个 stage，前 4 个 stage 分别由一个 ConvTranspose2d 层，一个 Batchnorm 层和一个 ReLU 层组成，只是输入输出的 channel 数不同，最后一个 stage 则是由一个 ConvTranspose2d 层和一个 Tanh 层组成。具体如下：

Layer (type)	Output Shape	Param #
ConvTranspose2d-1	[-1, 512, 4, 4]	819,200
BatchNorm2d-2	[-1, 512, 4, 4]	1,024
ReLU-3	[-1, 512, 4, 4]	0
G_block-4	[-1, 512, 4, 4]	0
ConvTranspose2d-5	[-1, 256, 8, 8]	2,097,152
BatchNorm2d-6	[-1, 256, 8, 8]	512
ReLU-7	[-1, 256, 8, 8]	0
G_block-8	[-1, 256, 8, 8]	0
ConvTranspose2d-9	[-1, 128, 16, 16]	524,288
BatchNorm2d-10	[-1, 128, 16, 16]	256
ReLU-11	[-1, 128, 16, 16]	0
G_block-12	[-1, 128, 16, 16]	0
ConvTranspose2d-13	[-1, 64, 32, 32]	131,072
BatchNorm2d-14	[-1, 64, 32, 32]	128

ReLU−15	[−1, 64, 32, 32]	0
G_block−16	[−1, 64, 32, 32]	0
ConvTranspose2d−17	[−1, 3, 64, 64]	3,072
Tanh−18	[−1, 3, 64, 64]	0

Total params: 3,576,704
Trainable params: 3,576,704
Non−trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 3.94
Params size (MB): 13.64
Estimated Total Size (MB): 17.58

Discriminator 和 Genrerator 则是几乎完全颠倒过来，也是由 4+1 个 stage 组成，前四个 stage 分别由一个 Conv2d 层，一个 Batchnorm 层和一个 ReLU 层组成，只是输入输出的 channel 数不同，最后一个 stage 则是一个 Conv2d 层。具体如下：

Layer (type)	Output Shape	Param #
Conv2d−1	[−1, 64, 32, 32]	3,072
BatchNorm2d−2	[−1, 64, 32, 32]	128
LeakyReLU−3	[−1, 64, 32, 32]	0
D_block−4	[−1, 64, 32, 32]	0
Conv2d−5	[−1, 128, 16, 16]	131,072
BatchNorm2d−6	[−1, 128, 16, 16]	256
LeakyReLU−7	[−1, 128, 16, 16]	0
D_block−8	[−1, 128, 16, 16]	0
Conv2d−9	[−1, 256, 8, 8]	524,288
BatchNorm2d−10	[−1, 256, 8, 8]	512
LeakyReLU−11	[−1, 256, 8, 8]	0
D_block−12	[−1, 256, 8, 8]	0

Conv2d−13	[−1, 512, 4, 4]	2,097,152
BatchNorm2d−14	[−1, 512, 4, 4]	1,024
LeakyReLU−15	[−1, 512, 4, 4]	0
D_block−16	[−1, 512, 4, 4]	0
Conv2d−17	[−1, 1, 1, 1]	8,192

Total params: 2,765,696
Trainable params: 2,765,696
Non−trainable params: 0

Input size (MB): 0.05
Forward/backward pass size (MB): 3.75
Params size (MB): 10.55
Estimated Total Size (MB): 14.35

2.2.3 模型训练

对应代码 training 注释部分代码，train 函数会调用 update_D 和 update_G 两个函数，在每轮训练中对 D 和 G 依次进行更新。对 D 和 G 更新的具体原理和上一节中所述一致。在训练过程中也会记录 D 和 G 的 loss 情况，最后画出图像。

2.2.4 若干参数

最后详细说一下代码中用到的参数。

- optimizer: Adam(lr=0.005)
- loss function: BCEWithLogitsLoss(reduction='sum')，这个损失函数主要针对多分类问题，并且把 Sigmoid 和 BCE-Loss 合成一步。
- latent dimension: 100，这一项表示传入 G 的随机向量的维度。
- activation function: G 采用 ReLU(输出层为 Tanh),D 采用 LeakyReLU
- batch size: 512，epochs：100

2.3 训练结果

经过 100 轮训练，我们就得到了训练好的 D 和 G 两个网络。可以看到随着训练轮数的增加，G 的 loss 逐渐上升，而 D 的 loss 则是逐渐下降，具体如下图所示：

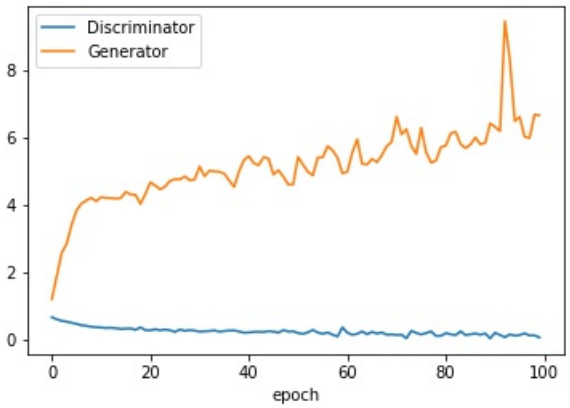


图 1: Result(Normal GAN)

整个训练过程大体上来说比较稳定，随着训练的进行，生成的图像也确实越来越清楚河里。从 G 生成的图像来看，前 10 个 epoch 主要是图像亮度的逐渐增强，变得越来越清楚，10~80 个 epoch 主要是图像逐渐河里 (其实 30 个 epoch 左右图像就已经可以看了)，80~100 个 epoch 则是一些微调 (换句话说就是没啥用)。下面展示的是第 10，20，40，80，100 个 epoch 时生成的小姐姐的图像：

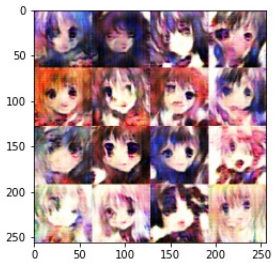


图 2: epoch 10

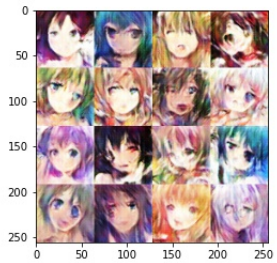


图 3: epoch 20

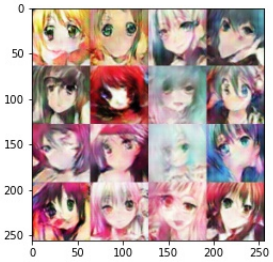


图 4: epoch 40

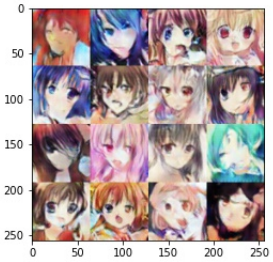


图 5: epoch 60

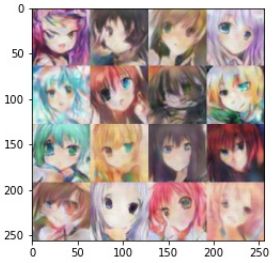


图 6: epoch 80

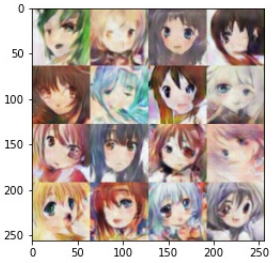
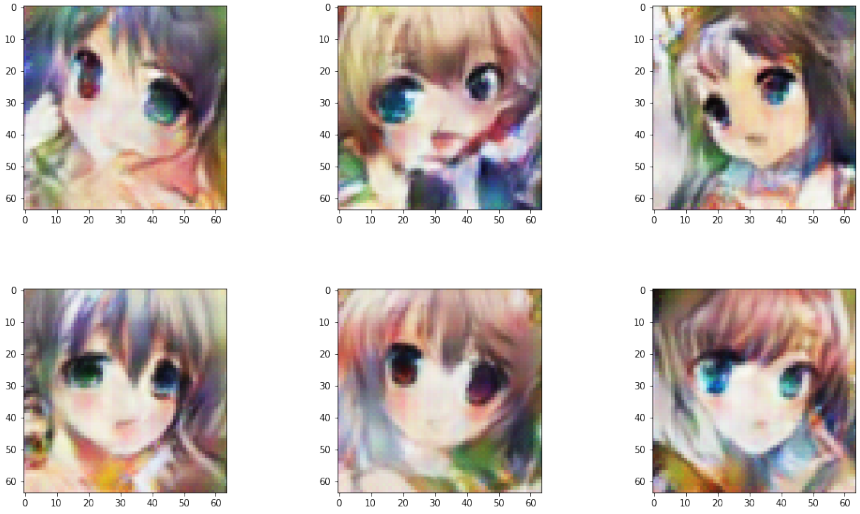


图 7: epoch 100

最后，我还精心挑选了生成的 5 张国色天香的小姐姐，是真的好看：



看到这些小姐姐的我就长这样：

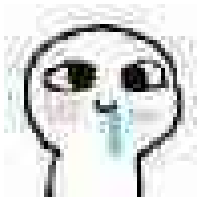


图 8: ↑ cc 的自画像

2.4 一些分析

1. 有一说一，GAN 让我感觉非常奇怪的一点是训练时候的参数对训练结果有很大的影响，2.2.4 中的参数是我调了好久才调出来的还算像样的参数组合，稍微改一点都不行。我把学习率从 0.005 调整到 0.001 或者 0.01，训练就会变得稀里哗啦，G 要么一直生成全黑的图像，要么生成让人不明所以的图像，反正就很迷惑，这和问题在 WGAN-GP 中还会体现得更加明显。

2. 仔细观察生成的图像，可以看到 GAN 在学习头发，脸型，五官还有衣服上都还算不错，但是细节方面学得却并不好，很明显的一个问题就是瞳色，有大量生成的小姐姐是左右眼瞳色不一样的。虽说确实有这样的动漫角色 (比如“吃饭睡觉打六花”的小鸟游六花)，但是这毕竟是少数。当然还有别的细节方面的问题，比如大量出现的“歪嘴战神”(epoch 100 第 2 排第 3 个)，“单眼美瞳”(epoch 100 第 2 排第 4 个)，这些都反映出我使用的 GAN 对细节的学习能力不够。

3. GAN 的训练过程虽说大致平稳，但仍然存在不小的波动，比如在我的训练过程中，Generator 的 loss 在第 90 个 epoch 左右就出现了较大的波动。另外，整个训练过程中 Generator 的 loss 不断上升，而 Discriminator 的 loss 不断下降，但对一个“优秀”的 Generator 而言，其 loss 应该是越小越好。我想造成这种情况的可能有两个原因，一是训练还不够充分，二是 Discriminator 的训练进度比 Generator 快。针对第二个原因，我看网上的代码，有一些是每训练 Generator 三到五轮再训练 Discriminator 一轮，这或许是一个可行的解决方案。

3 WGAN-GP

3.1 原理

在介绍 WGAN-GP 之前先介绍一下 WGAN，在介绍 WGAN 之前又要先说一下 GAN 存在的问题。GAN 存在的问题有这样几个：一是训练困难，需要精心设计模型结构，并小心协调 D 和 G 的训练程度，就像我上面痛苦的调参过程；二是 G 和 D 的 loss 并不能指示训练的过程，缺乏一个指标来指示训练进行程度；三是模式崩坏，即生成的图像缺乏多样性。为了解决这些问题，WGAN 做了几个很小而改动，一是 D 的最后一层去掉了 Sigmoid；二是 G 和 D 的 loss 不取 log；三是每次更新 D 的参数之后，将其绝对值截断到不超过一个固定常数 c 以满足论文中提到的 lipschitz 连续性条件；四是 optimizer 使用不带动量的算法，比如 SGD。有了这四点改进，根据作者的论证，就可以很好地解决 GAN 存在的问题 [2]。

WGAN-GP 则是进一步针对 lipschitz 连续性条件改进了 WGAN。WGAN 中的强制梯度截断可能会导致大多是权重是阶段范围的上下限，大大降低的神经网路的拟合能力，如下图所示。同时，强制截断也可能导致梯度消失或者梯度爆炸问题。

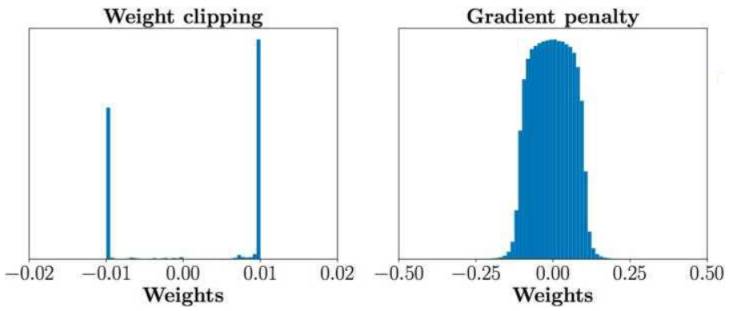


图 9: Contrast between weight clipping and gradient penalty

于是作者提出了使用梯度惩罚 gradient penalty 的方式以满足此连续性条件。既然 Lipschitz 限制是要求判别器的梯度不超过 K，那么就可以先求出判别器的梯度，然后建立与 K 之间的二范数就可以实现一个简单的损失函数设计。虽然 D 的梯度的数值空间是整个样本空间，但是作者提出没必要对整个数据集 (包括真的和生成的) 做采样，只要从每一批次的样本中采样就可以了，比如可以产生一个随机数，在生成数据和真实数据上做一个插

值，这样就算解决了在整个样本空间上采样的麻烦 [6]。

3.2 代码分析

WGAN-GP 的代码主体框架与 GAN 相同，做了以下一些改动：

1. 把 optimizer 由 Adam 改为 SGD，学习率设置为 0.001。

2. 在更新 D 时加入 gradient penalty，计算方法如原理中所述，并将原来的 D 的 loss 权重设为 1，gradient penalty 权重设为 10，两个相加得到新的 loss。这一部分代码参考了 github 上相关文档 [5]。

3. 由于原先我的代码中 D 的最后一层就没有使用 Sigmoid，G 和 D 的 loss 也没有取 log，这边就无需进行改动了。

4. 我在 github 上看到很多代码中 loss 的计算方式都是直接对输出向量取平均，根据不同情况乘以 1 或-1 得到 loss，但我原先使用的 BCEWithLogitsLoss 这个损失函数效果貌似也不错，这里就沿用了。

3.3 训练结果

同样训练 100 个 epoch，结果却并不如人意。前 10 个 epoch 的训练还很正常，G 生成的图像如下所示：

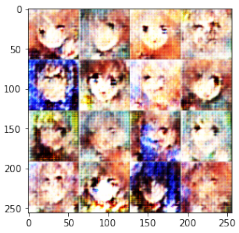


图 10: epoch 2

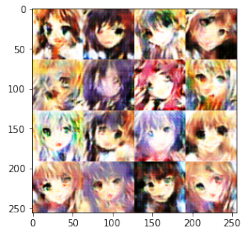


图 11: epoch 4

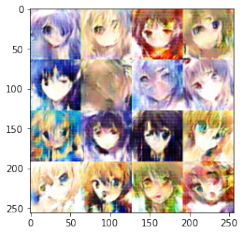


图 12: epoch 6

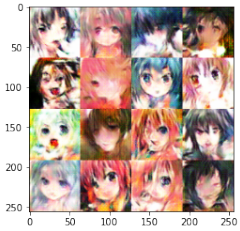


图 13: epoch 8

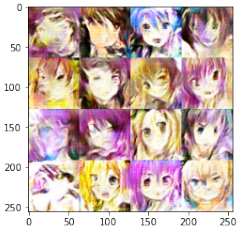


图 14: epoch 10

但从第 12 个 epoch 开始，训练就一发不可收拾，G 生成的图片变得越来越抽象，第 31 个 epoch 之后，G 就永远生成完全一样的没有任何意义的图像，就像下面这样：

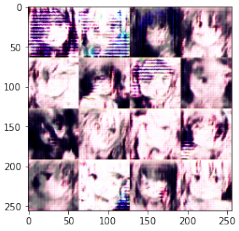


图 15: epoch 12

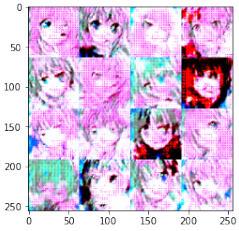


图 16: epoch 14

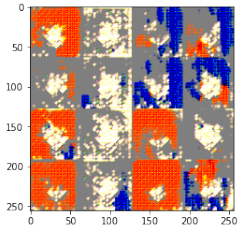


图 17: epoch 16

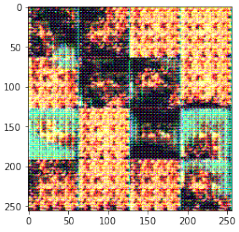


图 18: epoch 18

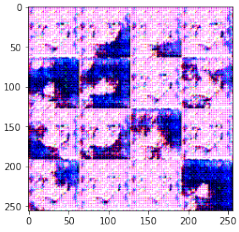


图 19: epoch 20

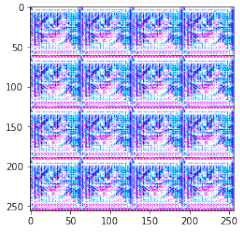


图 20: epoch 31

整个训练过程中 G 和 D 的 loss 如下图所示：

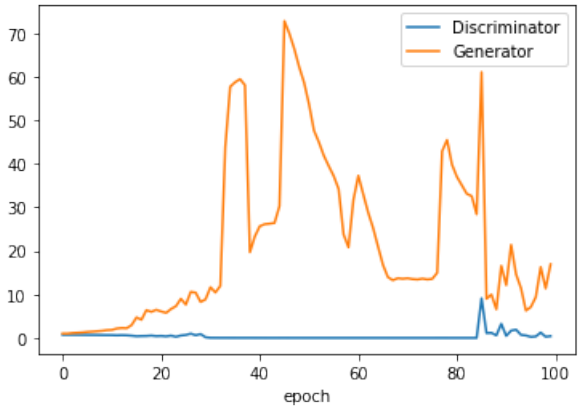


图 21: Result(wgan-gp)

3.4 一些分析

这次我的 WGAN-GP 的尝试应该说是失败的，我也仔细排查过我的代码，并没有发现什么问题，可训练就是稀里哗啦的。我猜想可能有以下两个原因：

1. D 和 G 训练进度相差太大, 这也是我觉得最有可能的原因。应该对 G 训练多轮后再对 D 训练 1 轮，以平衡 G 和 D 的训练，但是这样花费的时间实在太长，我在 kaggle 上试着跑了很多次，都是跑到一半就因为各种原因中断，一直没能跑完。

2. 参数没调整好，包括 gradient panelty 的权重，学习率等。这一点我在 GAN 的训练中就深有体会，虽然 WGAN-GP 对参数要求不高，但是可能也有一点要求？

不过从前面正常训练的 10 个 epoch 可以看出在相同的学习率下,WGAN-GP 的训练速度确实要快一些。第 10 个 epoch 时 G 生成的图像已经大概可以达到普通 GAN20 个 epoch 时的水平。

4 舍不得删掉的废话

1. GAN 部分的代码我写完才发现和 DCGAN 几乎一样，难怪训练效果不错。WGAN-GP 部分的代码在参数和训练方式上可能存在一些小问题，导致 10 个 epoch 之后的训练出现不稳定的情况。我尝试过进行一些修改，但是由于训练一次所需的时间实在太长，我在提交时也没有解决训练不稳定的问题 (呜呜呜)。

2. 提交的代码既可以用 DCGAN 训练，也可以用 WGAN-GP 训练，由 train 函数中 gan_type 这个参量控制。助教如果愿意的话可以复现一下我 WGAN-GP 出现的问题，或许能看出来原因是什么。

3. 如下运行训练脚本：bash train.sh ImageFolder 1，其中 ImageFolder 为包含二次元小姐姐图片的文件夹，第二个参量可选 1 或 2，1 表示用 DCGAN 训练，2 表示用 WGAN-GP 训练。如下运行测试脚本：bash test.sh model.h5，其中 model.h5 为训练好的模型。

4. 提交的 project 中，Figures 文件夹包含了所有原始图像，train.py 为我使用的代码，train.sh 为训练脚本，Net_G.param 为训练好的 Generator，Discriminator 由于文件大小限制没有上传。

5. 这次作业不仅让我认识到参数好坏对于模型训练的重要性，更让我

认识到对参数要求不高的模型和训练方法的重要性。至于为什么 WGAN-GP 在加上 gradient penalty 这一项后会出现训练不稳定的情况，我希望能在我今后的学习中得到解答。

5 参考资料

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial networks. arXiv preprint arXiv:1406.2661.

[2] Arjovsky, M., Chintala, S., & Bottou, L. (2017, July). Wasserstein generative adversarial networks. In International conference on machine learning (pp. 214-223). PMLR.

[3] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). Improved training of wasserstein gans. arXiv preprint arXiv:1704.00028.

[4] http://www.d2l.ai/

[5] https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/wgan_gp

[6] https://www.cnblogs.com/bonelee/p/9166122.html