Generative Adversarial Network 数据生成实验

基于 *MNIST* 和简单的几何图形的测试

0.1： 版本号

姓名： 胡学仕

学号： U201514545

班级： ACM1501

完成日期： 2018 年 7 月 8 日

文档作者联系方式：hubachelar@gmail.com

目　录

第 1 章

摘要

GAN(Generative Adversarial Network) 自提出之时候就备受的深度学习的关注。和判别模型不同，GAN 不需要数据具有标签，它试图解决的问题是如何根据现有的数据生成的数据集中间从来都没有数据，但是该数据和数据集中间的数据类似。在本课设中间，将会使用 MNIST 和各种自制的几何图形来验证和分析 GAN 的能力。

§1.1 选题背景

GAN 是 Ian Goodfellow 在 2004 年提出的神经网络模型，该模型可以通过 Discriminator 和 Generator 的两个神经网络的对抗学习，通过现用的数据生成出新的数据。Discriminator 网络是一个用于判别的网络，输入为数据 X, 输出为一个 bool 值，当该 bool 值表示该数据是的 Generator 创造的伪造的数据否者是从数据库中随机取出来的数据。Generator 的输入为随机噪音，输出值为伪造的数据。

GAN 无论是从理论上还是从应用上都具有高意义。GAN 利用了博弈论的思想，Discriminator 和 Generator 两者互相博弈，让 Discriminator 的判别能力逐步强化，而同时让 Generator 生成图片逐步和测试的样例中间的图片的更加相似。GAN 启发研究者，只有通过坚实的理论基础，采用可能打开的深度学习的新大门。当 GAN 可以通过的噪音获取到正确的图形的时候，说明 Generator 已经可以抓取到数据库中间的数据分布的特点，而先用的分类器的分类的依据也是通过不同标签的数据分布的位置不同，从而实现对于数据的分类的。GAN 的应用非常的广泛，包括的字体生成，文字到图片的装换，实时人脸重建等等应用。

在本次课设中，使用 MNIST 以及多种自己创建的数据，用于测试 GAN 在具体的样本上的能力如何，通过多种简单几何图形的数据集的测试，比对 GAN 对于特殊的几何的图形的性质是否同样可以识别并且生成出来。

§1.2 理论介绍

理解 GAN, 首先需要理解生成模型和判别模型直接的区别是什么：公式 $p(y|x)$ 用于描述 “给定 x, y 的概率是多少” , 在 MNIST 中间，其含义是给定一张图片，该数字是 0 的可能性。而 $p(x|y)$ 的用于表示给定出一个类，该类数据的含有特定特征的概率。

G(Generator) 和 Discriminator(D) 的分别是两个神经网络，前者使用生成模型，后者使用判别模

型。如图 1.2-1所示，G 使用 latent space 和噪音混合输入，然后生成出来伪造数据，混合在真实数据中让 D 来分辨。但使用 MNIST 的数据集合的时候，该具体化为 1.2-2所示的结果。

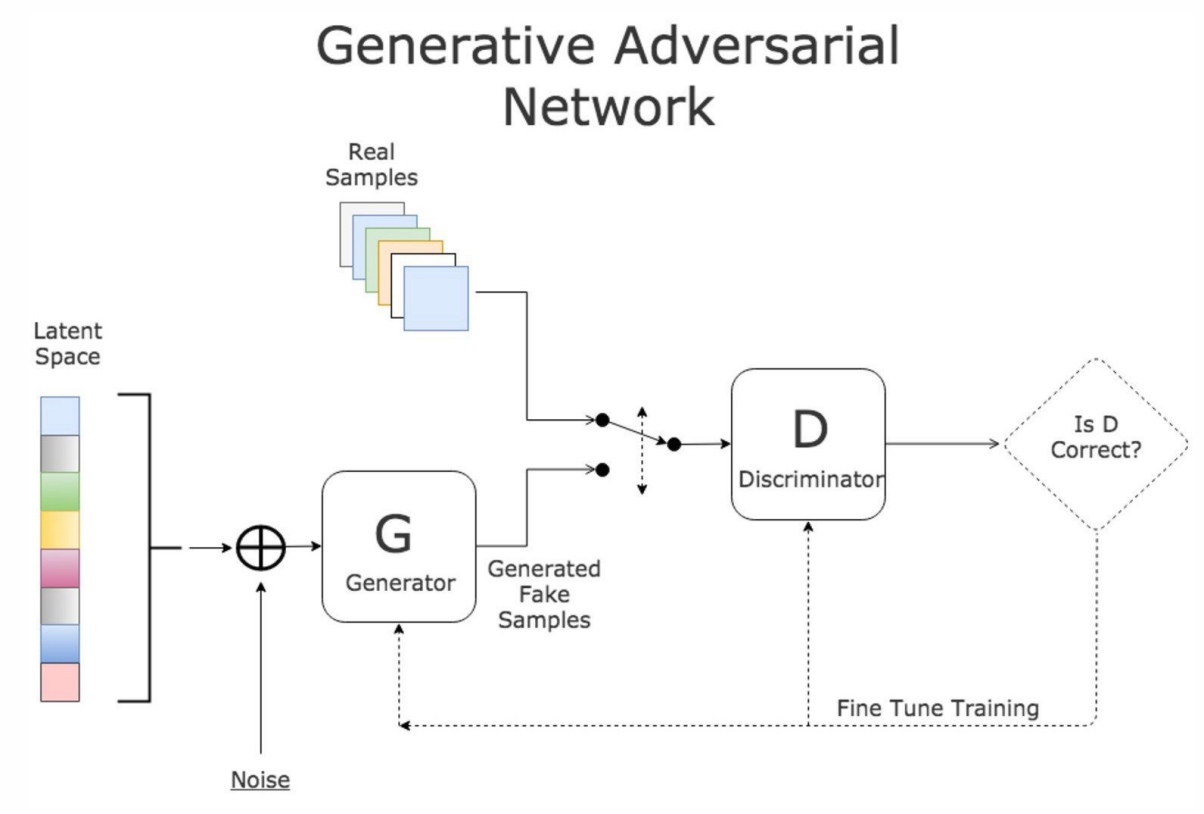

图 1.2-1 GAN

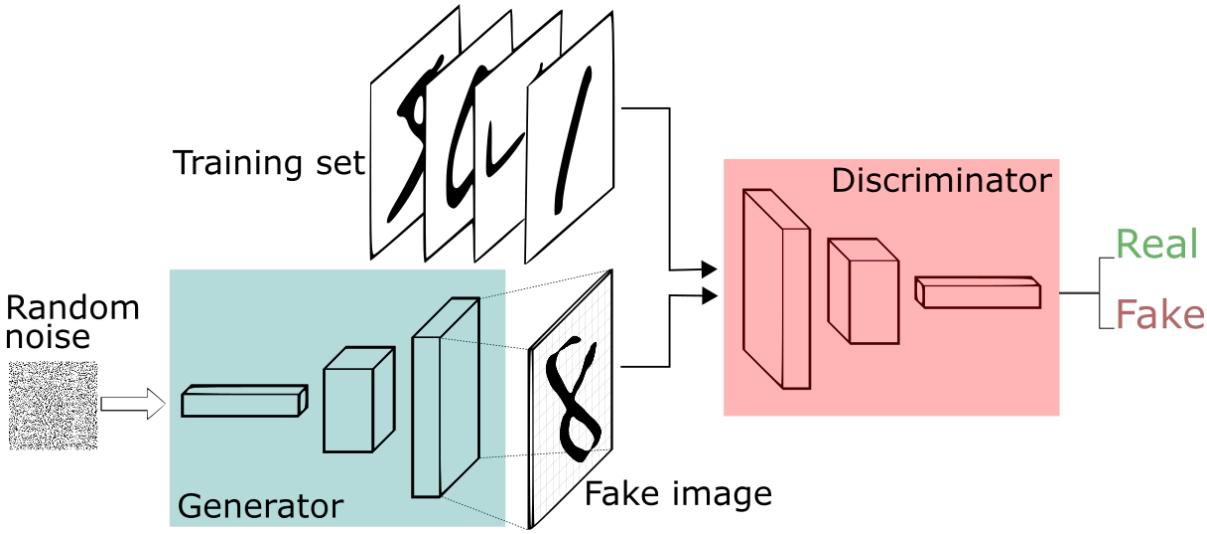

图 1.2-2 GAN with mnist

该算法的具体描述 1.2-3, 从该算法需要注意的是，总是多次训练 D 然后在训练一次 G, 其中原因是为了让 D 更快获取较强的分类能力，避免陷入到分类器 D 和生成器 G 的能力都的不高的状态中间。

此算法的来源是 1.2-4中公式，对于目标公式，G 和 D 分别将该的公式最大化和最小化，希望达到的结果 G 可以产生近似真实的图片，而 D 可以具有很强分类能力。

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

图 1.2-3 GAN algorithm

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

图 1.2-4 GAN formula

§1.3 参考文档

- https://deeplearning4j.org/generative-adversarial-network
- https://arxiv.org/pdf/1701.00160.pdf
- https://github.com/nashory/gans-awesome-applications
- https://arxiv.org/pdf/1703.10580v1.pdf
- http://yann.lecun.com/exdb/mnist/

第 2 章

数据来源

本课设的特色指出在于使用学界公认的数据 MNIST 和自制多种的数据，通过的不同的数据，相同或者相似的网络来处理这些数据，然后比对结果，所以数据的特点的理解是下一步实现的基础。

§2.1 MNIST

MNIST 数据集是公认的较为可信的入门级的数据集，当深度学习没有流行的时候，传统的机器学习方法已经可以在 MNIST 上实现较为良好的结果。MNIST 的所有图片 $28 * 28$ 的大小，数字分别标签为 0 到 9，图 2.1-1 图 2.1-2分别是 MNIST 数据中间的两种形态的 2, 图 2.1-3是 MNIST 数据集中 9。MNIST 的测试数据和训练数据一共含有 60000 张。猜测可能是由于的 MNIST 中间的数据更加多以及其中的数据的识别标准更加宽容，最终在 MNIST 上的数据的结果总是最佳的。

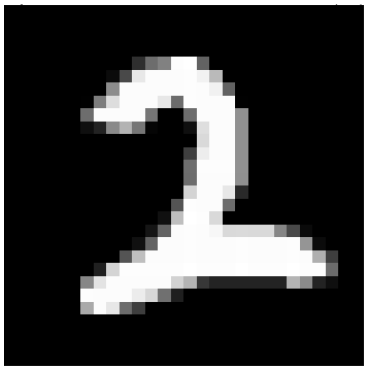

图 2.1-1 手写字 2

§2.2 线条

此数据是使用 python 的工具绘制出来的数据，在本数据集合中间，图片的大小是 60 * 60 的大小，图片中间线条的数目从 0 到 9 各有 1000 张，一共含有 10000 张图片。图 2.2-4 和图 2.2-5 展示的其中的两个样张。

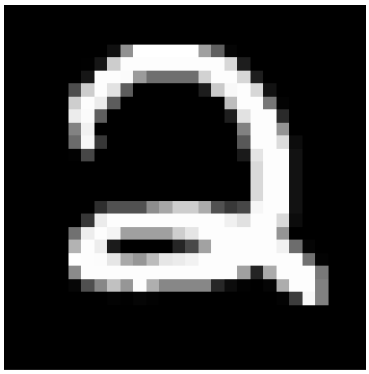

图 2.1-2 手写字 2

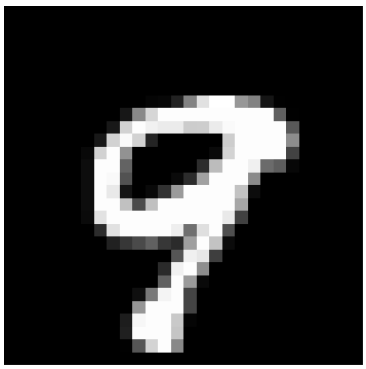

图 2.1-3 手写字 9

图 2.2-4 含有 9 个线条

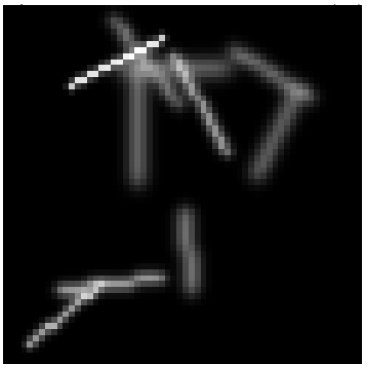

图 2.2-5 含有 10 个线条

§2.3 拼图

在此处的拼图的定义是图像中间含有两个由一个矩形拆分成得到的两个部分。2.3-6 和 2.3-7显示的其中的两个样张。每一张图片的大小是 28 * 28, 一共含有 10000 张图片。

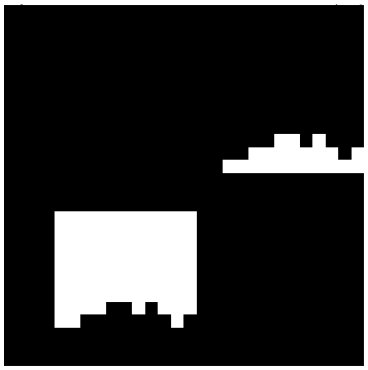

图 2.3-6 拼图 1

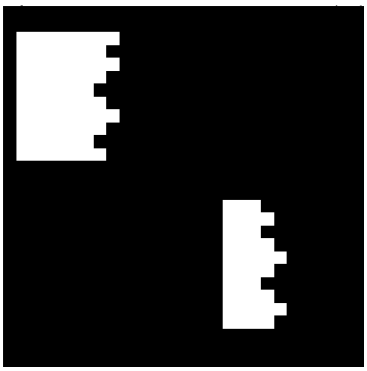

图 2.3-7 拼图 2

§2.4 正多边形

正多边形是从三角形，正方形等一直到正十二边形，相同的图形在在其中含有旋转，大小和中心位置的不同等区别。2.4-8和 2.4-10是两个不同的三角形，其旋转方向不相同。**??**是正十边形。

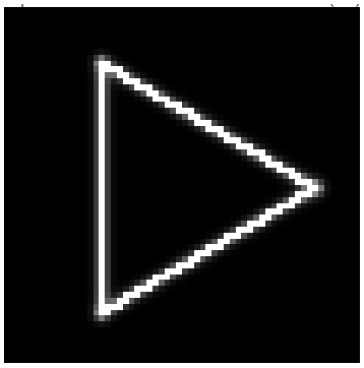

图 2.4-8 拼图 1

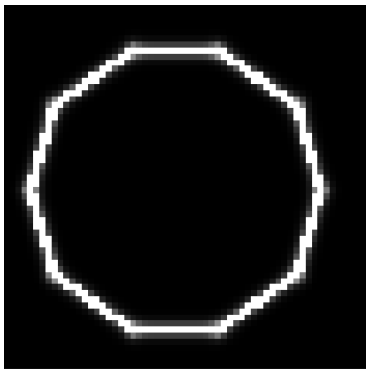

图 2.4-9 三角形

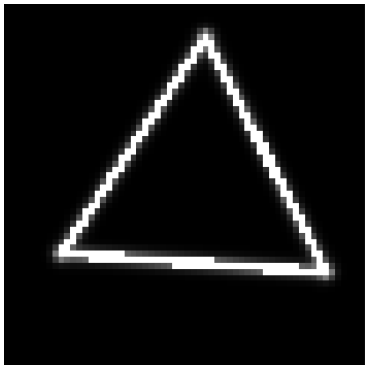

图 2.4-10 正十边形

§2.5 点

点图形指的是 2.5-11以及 2.5-12类似，每一张图片包含的十字状的物体若干个，数目从 0 到 9 不等，一共含有 10000 张图片，图片的大小是 28 * 28.

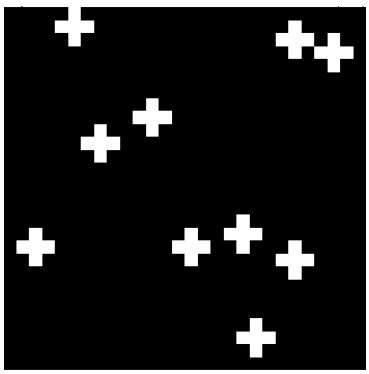

图 2.5-11 包含四个点

§2.6 数据来源总结

2.6展示全部数据的总结, 需要说明一下为何处理正多边形的数目是 800 而不是 10000, 主要原因是绘制的正多边形的算法的实现导致想要得到 10000 张图片需要非常长的时间，除此之外，正多边形的图片之间的变化更加小，所以绘制比较少。

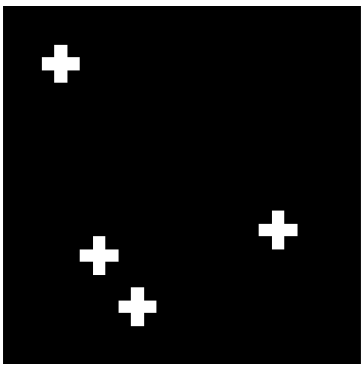

图 2.5-12 包含十个点

名称	图片大小	图片数目	数据是否自制
MNIST	28 * 28	60000	否
线条	60 * 60	10000	是
拼图	28 * 28	100000	是
正多边形	60 * 60	800	是
点	28 * 28	10000	是

表 2.6-1 各节点硬件配置要求

第 3 章

实验过程

上一个章节获取到了所有的数据之后，下一步就是使用神经网络进行训练，分析结果。

§ 3.1 运行环境

本课设使用的框架是 tensorflow 非 GPU 版本。
运行硬件环境使用 neofetch 得到结果如下:

```
            ............                martin@martin-PC
        .';;;;;.       .,;,.            ----------------
     .,;;;;;;;.       ';;;;;;;.         OS: Deepin 15.6 x86_64
   .;;;;;;;;;'     .,:;;;,'''',.       Model: 20BVA02ACD ThinkPad T450
  ,'.;;;;;;;;    .;;'.          ';      Kernel: 4.15.0-21deepin-generic
 ;'  'cccccc,   ,' :: '..        .:     Uptime: 11 hours
,,    :ccccc.  ;: .c, '' :.       ,;    Packages: 2671
.l.     cllll' ., .lc  :; .l'       l.   Shell: zsh 5.3.1
.c       :lllc  ;cl:  .l' .ll.      :'   Resolution: 1366x768
.l        'looc. .   ,o:  'oo'      c,   DE: Deepin
.o.         .:ool::coc'  .ooo'      o.   WM: Mutter(DeepinGala)
::            .....   .;dddo      ;c    Theme: Deepin [GTK2/3]
 l:...            .';lddddo.     ,o     Icons: Deepin [GTK2/3]
  lxxxxxdoolllodxxxxxxxxxc      :l      Terminal: tilix
   ,dxxxxxxxxxxxxxxxxxl.     'o,       CPU: Intel i5-5200U (4) @ 2.700GHz
     ,dkkkkkkkkkkkkko;.    .;o;        GPU: Intel Integrated Graphics
       .;okkkkkdl;.    .,cl:.          GPU: NVIDIA GeForce 940M
           .,:ccccccc:,.               Memory: 3680MiB / 7677MiB
```

§ 3.2 构建神经网络

在实验的过程中间，曾经尝试过使用 CNN，但是测试的时候发现的 CNN 的运行结果非常的慢的，所以放弃使用，所以以下得到的结果所有的运行的使用的是相同网络模型，只有当数据的维度发生变化的时候，才会自适应做出调整。3.2和 3.2分别是 Generator 和 Discriminator 的网络的配置，两个网络全部使用的是最简单的全连接网络。

层	一	二	三
维度	噪音维度 100	100 300	300 图片维度

表 3.2-1 Generator 网络结构参数

层	一	二	三
维度	图片维度 300	300 100	100 1

表 3.2-2 Discriminator 网络结构参数

由于 GAN 的特殊性，构建网络的时候有几个需要注意的位置

- 当训练 D 的时候，中间包含了 G 生成的噪音数据，如果训练的时候不指定是哪一个变量列表，那么在训练 D 的时候，同时会修改 G 的参数。
- 在大多数的情况之下, G 的实现是通过函数来实现, 如果调用两次该函数，得到的是两个网络, 但是为了让 G 生成数据和 z 生成的数据经过的是同一个网络, 可以采用的方法 tensorflow 提供的 resue_variable 函数, 或者让两个数据连接成为一个整体, 然后调用 G 的函数。

§3.3 完善备份和展示功能

网络构建完成之后就可以了，但是实现更加多的辅助功能可以方便的自己的实验。tensorflow 提供了一个保存运行参数，在每一次循环的时候保存 checkpoint, 当用于种种原因实验中断的之后，只要日志文件依旧存在，就可以继续之前的运行。

第二个是借用 tensorboard 工具，更加容易的显示结果出来，在数据准备阶段中间的数据的截图就是全部来自于使用 tensorboard 工具的截图。

§3.4 测试结果展示

在制作的数据的时候，本来以为所有的数据都是有比较好的结果，但是实际上发现，有的数据可以得到比较好的结果，比如 MNIST, 有的数据只是得到了数据的大体的特征，创建的数据的最主要的特征已经完全消失了，而有的数据得到的结果没有任何的意义，完全看不到生成的数据的含义所在。

§3.4.1 MNIST

MNIST 的测试效果相对于其他的数据的测试结果效果最佳，**??**和 3.4-2中间的数据的结果几乎可以原始的数据相同。

§3.4.2 正多边形和拼图

在这两个数据集中间得到数据丧失了重要的细节，但是保留了对于数据基本观察。3.4-3和 3.4-4显示的是正多边形的训练过程中先后抽取的图片，从图片中间可以感觉到生成的图片正在趋于形成一个圆形。3.4-5中间生成的数据观察到数据会划分为两个部分，但是没有的办法得到可以拼接称为一个图片的数据。

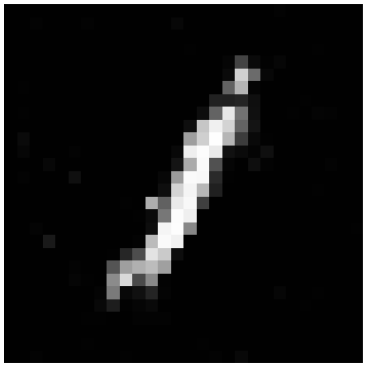

图 3.4-1 生成手写字 1

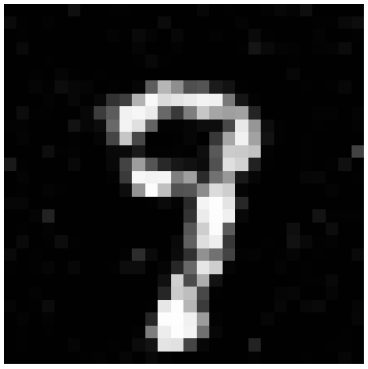

图 3.4-2 生成手写字 2

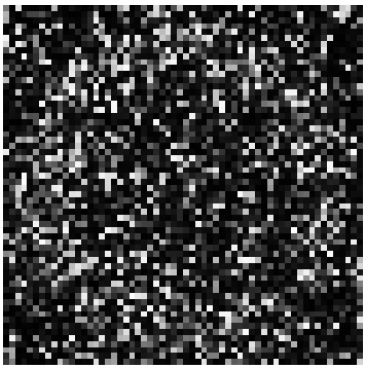

图 3.4-3 正多边形生成结果

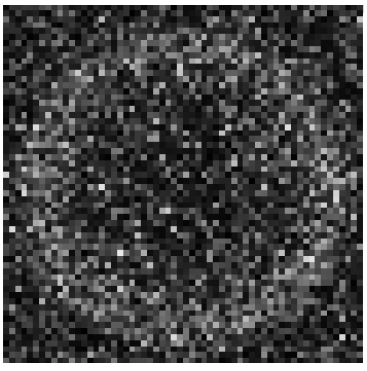

图 3.4-4 正多边形生成结果

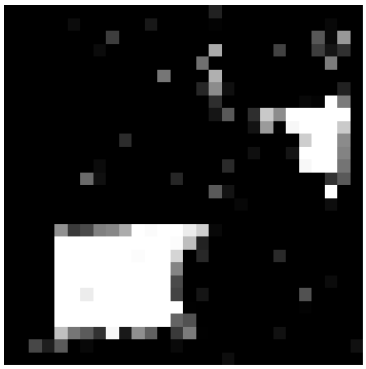

图 3.4-5 拼图生成结果

§3.4.3 点和线条

这两个数据集经过一个晚上的运行，得到的结果依旧是随机造影，看出来任何重要的信息。比如 3.4-6显示的为点图形的结果，该结果中间似乎可以看到点的聚集倾向，但是除非看过原始数据才有这一种错觉。

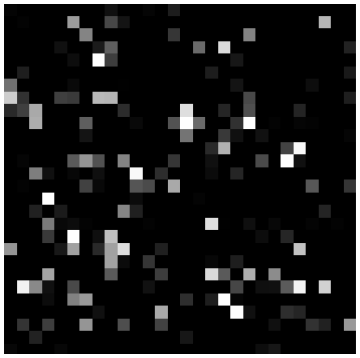

图 3.4-6 点生成结果，几乎看不到任何结构

第 4 章

总结和下一步实验

在本次课设中间，学习了 python 和 tensorflow 的使用，理解了 GAN 模型的运行原理和机制。在实验的过程中间，用于代码运行缓慢，很多错误需要很长时间才可以发现，自己搭建的神经网络的 bug 不断，最后只好借助网上的教程，在其上修改部分内容，才达到实验的目的。对于为什么线条和点的生成结果不理想，目前没有明确的定义，希望在下一步的实验中间希望可以进一步生成数量更加多的数据集来确地是否是数据量不够的原因，以及采用不同的网络结构来实现 Generator 和 Discriminator 网络，改善目前全连接网络的缺陷。