深度卷积生成对抗网络优化

The Optimization of Deep Convolutional Generative Adversarial Networks

计算机科学与技术

姓名　　王悦　　PB13011058

导师 张信明 教授

2017 年 5 月

中国科学技术大学

本科毕业论文

题 目 深度卷积生成对抗网络优化

英 文 The Optimization of
题 目 Deep Convolutional
Generative Adversarial Networks

院 系 计算机科学与技术

姓 名 王悦 学 号 PB13011058

导 师 张信明 教授

日 期 二〇一七年五月

致谢

首先要感谢我的指导老师张信明教授，从毕业设计题目的选定，到中期检查时候的方向调整和后期的论文书写修改过程中，张老师的悉心教导令我受益匪浅。

其次要感谢中国科大超级计算中心，在前期的工作中，由于自己的笔记本是ATI的显卡，不支持CUDA，并且i5三代低压版的CPU性能较弱，在学习过程中，许多实践需要GPU来加速运算，在自己的笔记本上无法实现，超算中心提供了平台的支持，让我得以在平台上学习和实践。

还要感谢我的好朋友李京同学，前期的实验不仅依靠超算中心提供的平台，也依靠他，借用他的笔记本做了许多实验，保证了毕业设计的进度。

更要感谢的是我的父母，养育之恩终生难以回报。父母提供资金给我更换笔记本电脑，让我得以在自己的计算机上继续进行毕业设计，免去了超算中心的排队和借用同学电脑的不便之处。

最后感谢这个城市，这所学校，还有身边的同学、朋友，提供了一个合适的平台环境，让我学习成长。

目录

摘　要 3
Abstract 4

第 1 章　绪论 5
1.1　背景 5
1.2　本文章节安排 5

第 2 章　对抗生成网络基础 7
2.1　神经网络基础 7
2.2　卷积神经网络 10
2.3　生成对抗网络 12

第 3 章　Wasserstein GAN 的关键技术 14
3.1　生成对抗网络的改进方案 14
3.2　Wasserstein 距离的优势 15
3.3　Wasserstein GAN 的优势 16

第 4 章　对 Wasserstein GAN 在图像生成上的改进 18
4.1　深度卷积神经网络识别验证码 18
4.2　深度卷积网络生成验证码 20
4.3　Wasserstein GAN 生成验证码 24
4.4　对 Wasserstein GAN 的优化 28

第 5 章　对抗生成网络在人脸生成上的实验 32
5.1　程序封装说明 32
5.2　DCGAN 32
5.3　WDCGAN 34
5.4　WDCGAN 的优化 35
5.5　WGAN 37
5.6　WGAN 的优化 38

第 6 章　Improved Training of Wasserstein GANs 42
6.1　Martin Arjovsky 对 Wasserstein GAN 所做改进..... 42
6.2　实验效果对比 42
6.3　全文展望 44

参考文献 46

附录 47
附 1　验证码 CNN 实现的网络结构程序 47
附 2　验证码 DCGAN 实现的网络结构和损失函数 47
附 3　二次封装 Tensorflow 函数说明 48
附 4　WGAN-GP 施加限制的方法..... 49

摘　　要

作为无监督学习中最有希望的研究领域之一，生成对抗网络目前被广泛应用于图像生成、序列生成、图像的超分辨率重建、将文字转化成图像等领域。生成对抗网络中的生成器和判别器要能够稳定快速的训练，并且生成器要能够生成在人类看来可以以假乱真的数据。在今年年初提出的 Wasserstein GAN，是一种新型的生成对抗网络的模型，试图解决传统生成对抗网络中存在的梯度不稳定、多样性不足等问题。在介绍了神经网络和对抗生成网络基本原理之后，简要描述了 Wasserstein GAN 前作中揭示的传统生成对抗网络存在问题的原因以及 Wasserstein GAN 的优势，结合深度卷积网络，做实验去检验 Wasserstein GAN 网络模型，揭示了 Wasserstein GAN 的不足之处。通过分析损失函数变化趋势以及大量的实验测试，在图像生成领域对 Wasserstein GAN 做出修改网络模型、调整模型参数等优化。在验证码和人脸两个数据集上的实验表明，Wasserstein GAN 是优于传统 GAN 的，提出的优化方法对 Wasserstein GAN 是有效的。

关键词：神经网络，生成对抗网络，深度卷积网络，图像生成，Wasserstein GAN，人脸，验证码，Tensorflow

Abstract

As one of the most promising research areas in unsupervised learning, generative adversarial network is widely used in the image generation, sequence generation, image super resolution reconstruction, transforming text into images and other fields. The discriminator and the generator of generative adversarial network should train fast and stable, and the generator should be able to generate data that can appear to be fake in human beings. Wasserstein GAN, proposed at the beginning of this year, is a new generative adversarial network model, trying to solve the traditional generative adversarial network exists in the gradient instability, diversity and other issues. After introducing the basic principles of artificial neural network and generative adversarial network, this paper briefly describes the causes of the traditional confrontation network problems and the advantages of Wasserstein GAN, which are revealed in Wasserstein GAN, and combined with deep Convolutional network, tests the Wasserstein GAN network model to reveal the shortcomings of Wasserstein GAN. By analyzing the trend of loss function and a large number of experiments, Wasserstein GAN is modified in the field of image generation to optimize the network model and adjust the model parameters. Experiments on the two data sets of the verification code and the face show that the Wasserstein GAN is superior to the traditional GAN, and the proposed optimization method is effective for the Wasserstein GAN.

Keyword: the neural network, generative adversarial nets, deep Convolutional network, image generate, Wasserstein GAN, human face, Verification code, Tensorflow

第1章 绪论

1.1 背景

近些年来人工智能的发展进入了快车道，层出不穷的黑科技冲击着人们的眼球。比如日常生活中用到Siri、Cortana等个人助理，以及在互联网遨游时遇到有针对性的推送，还有各家厂商进军自动驾驶行业，Alpha Go战胜李世石，Master横扫围棋快棋棋坛，种种事件都表明人工智能进入了一个井喷式的发展阶段。

在人工智能领域，深度学习也爆发了强大的生命力，尤其是随着高性能计算的发展，深度学习不再受到计算能力上的制约，相关的研究取得众多突破。这是一种黑箱式的科技，却吸引着众多学者去探寻奥秘、发掘潜力。从识别到分析，从合成到创作，深度学习在语音、图像、自然语言理解、艺术创作等众多方面都取得惊人的进展，一些模型都已经超过人的能力。

生成对抗网络自从被提出[1]，就以极快的速度发展着。在艺术创造上，生成对抗网络展现的“创造力”令人叹为观止，微软小冰创作的诗歌，被众多读者称赞，读到的人甚至不知道这一首首充满灵性的诗歌，来自一个人工智能。

1.2 本文章节安排

本文对内容做如下安排：

第一章 绪论。介绍对抗生成网络的相关背景。

第二章 对抗生成网络基础。从神经网络入手，介绍神经网络的思想和部分算法，接着是卷积神经网络的算法原理，以及生成对抗网络的基础知识。

第三章 Wasserstein GAN的关键技术。首先介绍的是学习调研的一些关于生成对抗网络的改进方案，其次着重介绍 Wasserstein GAN 的改进方法，包括Wasserstein距离的优势和Wasserstein GAN的优势。

第四章 对Wasserstein GAN在图像生成上的改进。本章所述为毕设中后期的工作，是在图像生成方面对Wasserstein GAN的优化。实验所用的数据集是用爬虫抓取的验证码。

第五章 对抗生成网络在人脸生成上的实验。本章所述为在celebA数据集[11]上实践上一章提出的优化方案，进一步的验证方法的可行性。

第六章 Improved Training of Wasserstein GANs。本章所述是提出Wasserstein GAN的作者Martin Arjovsky于三月提出的gradient penalty改

进[4]。最后是对毕业设计的总结和展望。

第2章　对抗生成网络基础

2.1　神经网络基础

神经网络是一种模拟生物神经元结构和功能的模型，典型的神经网络模型包含结构、激活函数和学习规则三部分，其作用是通过简单的矩阵运算和非线性的激活运算来对数据进行处理。

下面所述的是一个简单的神经网络。

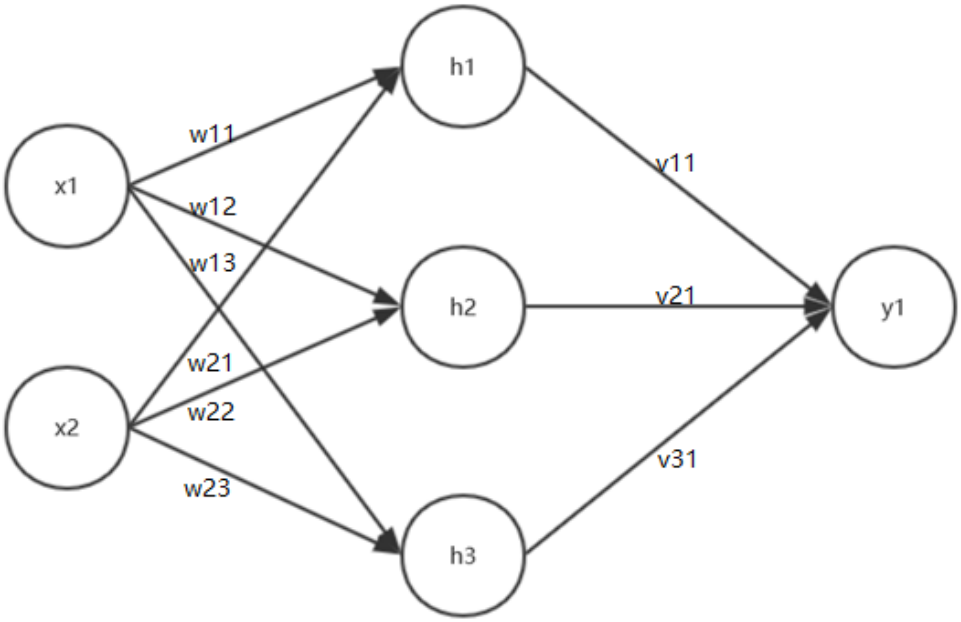

图 1-1 神经网络示意图

如图 1-1 所示，这是一个简单的神经网络，包含两个输入变量x_1、x_2，三个中间变量h_1、h_2、h_3，一个输出变量y_1，将这些结点称为神经元。其中x_i所在为输入层，h_j所在为隐藏层，隐藏层可以有多层，y_k所在为输出层。w_{ij}和v_{jk}被称为权重。计算方式为：

$$\begin{cases} h_j = \sum_i w_{ij} x_i \\ y_k = \sum_j v_{jk} h_j \end{cases} \tag{2.1}$$

例如：

$$\begin{cases} h_1 = w_{11} x_1 + w_{21} x_2 \\ h_2 = w_{12} x_1 + w_{22} x_2 \\ h_3 = w_{13} x_1 + w_{23} x_2 \end{cases} \tag{2.2}$$

为了表达和计算上的方便，用矩阵来表示如上运算过程：

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{2.3}$$

在实际的运用中，往往还会加上一个偏置项 bias，则 1.1、1.2、1.3 式转变为如下：

$$\begin{cases} h_j = \left(\sum_i w_{ij} x_i \right) + b_j \\ y_k = \left(\sum_j v_{jk} h_j \right) + d_k \end{cases} \tag{2.4}$$

$$\begin{cases} h_1 = w_{11} x_1 + w_{21} x_2 + b_1 \\ h_2 = w_{12} x_1 + w_{22} x_2 + b_2 \\ h_3 = w_{13} x_1 + w_{23} x_2 + b_3 \end{cases} \tag{2.5}$$

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{2.6}$$

图 1-1 中没有表现出偏置项，在实现过程中，偏置项有多种方式可以添加进去，一种是如式 2.6 加上一个向量，还可以将偏置项视作 1×w，将偏置项的加法运算放入矩阵相乘中，式 2.6 转化为：

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \tag{2.7}$$

其中 $b_j = w_{3j}$。这种方式在计算上省去了一步向量相加，但是多出了一步向量的拼接操作，将 (x_1, x_2) 拼接为了 $(x_1, x_2, 1)$。如图 1-2：

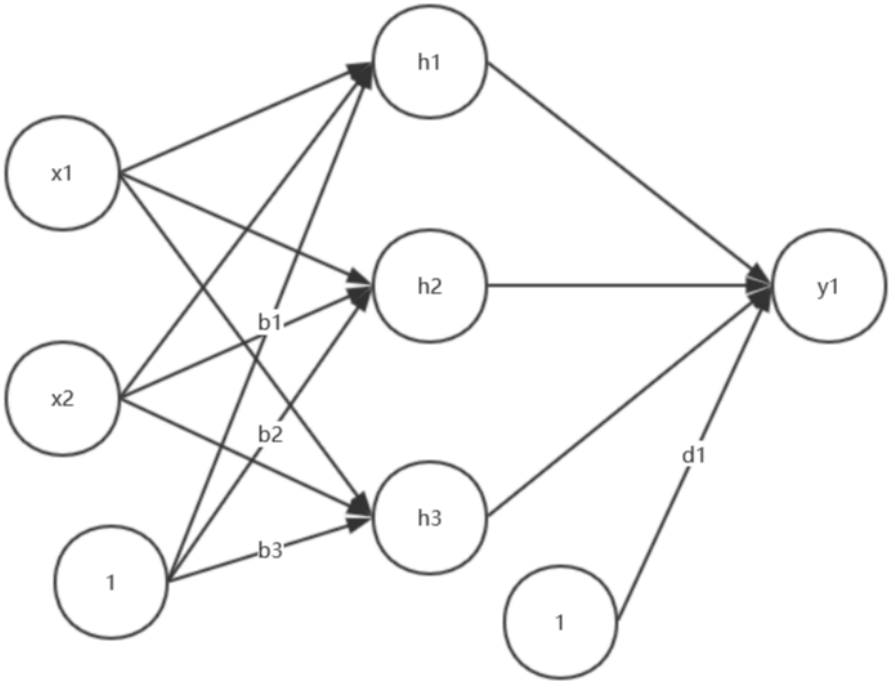

图 1-2 神经网络示意图，带偏置项 bias

在实际的运算中，拼接操作比向量相加消耗更多的计算。于是出现了又一种计算方法。考虑到每次拼接的开销，不妨考虑只在输入层增加一个值为 1 的神经元，在每一层隐藏层，都附加一个神经元，该神经元的值由之前的计算得到，如图 1-3：

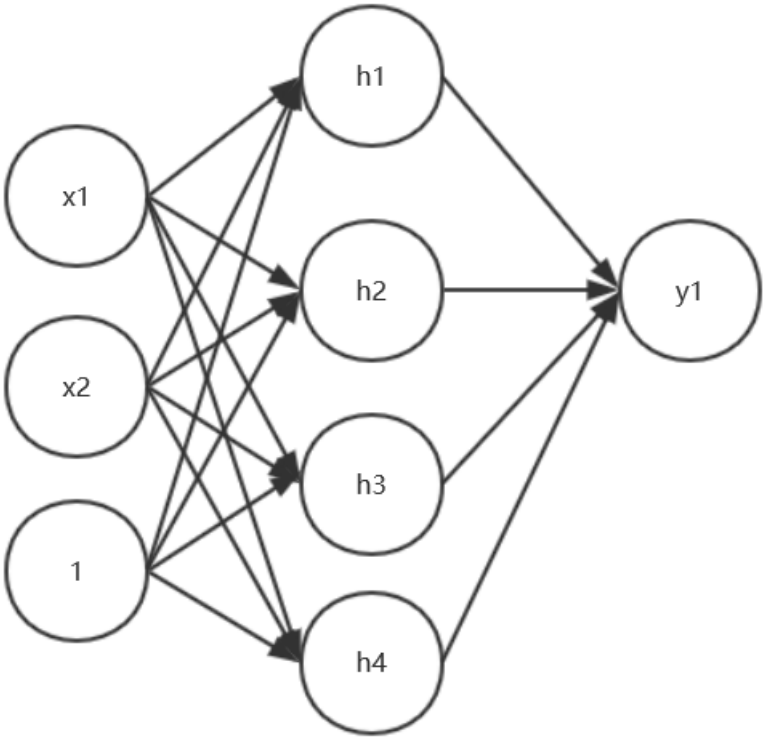

图 1-3 神经网络示意图，转变后的偏置项

相对应的计算公式为：

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}, (y_1) = \begin{pmatrix} v_{11} & v_{12} & v_{13} & v_{14} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{pmatrix} \tag{2.8}$$

其中，$h_4 = w_{41}x_1 + w_{42}x_2 + w_{43}$作为隐藏层的偏置项参与输出层的运算。

神经网络里面一个举足轻重的反向传播算法 back propagation[12]，其作用在于调整网络使之成为我们所需要的状态，简单的来说就是复合函数的链式求导，用梯度下降的方法更新网络。

2.2 卷积神经网络

在后文中用到了另一个重要的网络层是卷积层，熟悉图像处理的人知道，卷积操作在图像中十分常见，对图像的平滑、模糊、锐化、边缘检测、浮雕等操作都可以通过特定的卷积算子来实现。下面简略的介绍卷积在图像中的操作方式。

卷积在图像中往往是在一个平面上进行运算，对于一张图像和一个卷积核，如图 1-4 所示，左边是一个 5x5 的图像，右边是一个 3x3 的卷积核。

x11	x12	x13	x14	x15
x21	x22	x23	x24	x25
x31	x32	x33	x34	x35
x41	x42	x43	x44	x45
x51	x52	x53	x54	x55

w11	w12	w13
w21	w22	w23
w31	w32	w33

图 1-4 图像卷积示意(1)

有一个形象的说法，将卷积核视为窗口，在图像上滑动，每平移一次，得到卷积后的结果。举个例子，在图 1-4 中，将卷积核置于图像的左上角，那么运算方式为，每个像素点的值乘以卷积核上与像素位置对应的值，最后累加得到结果。

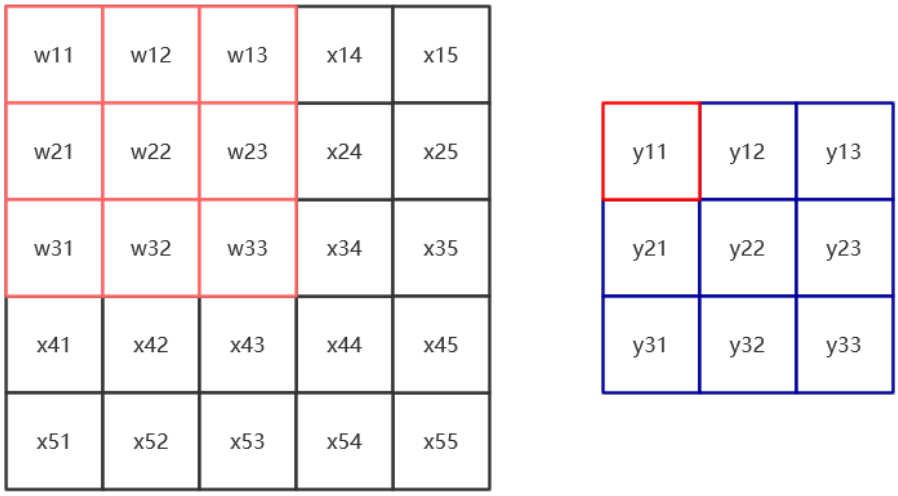

图 1-5 图像卷积示意(2)

运算式为：

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

然后将卷积核向右移动一个像素，如图 1-6：

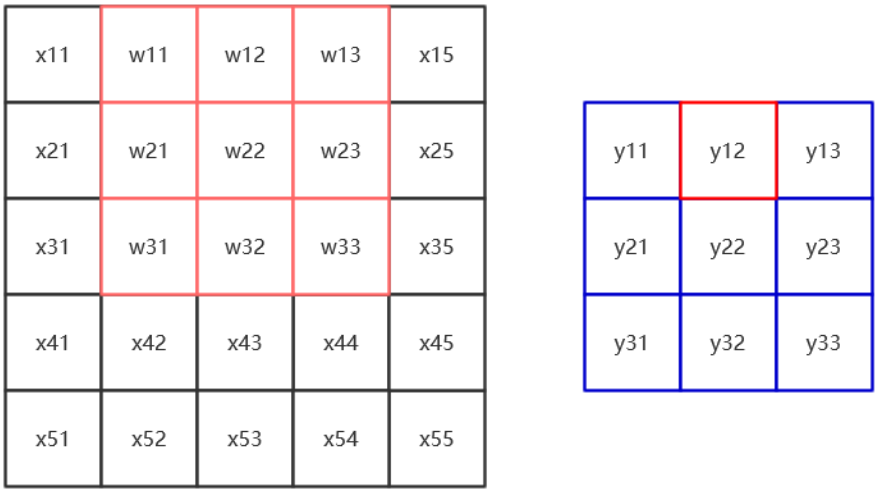

图 1-6 图像卷积示意

对应的运算为：

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

将卷积核在图像上全部滑动一遍后，得到一个 3×3 的新图像。原图尺寸是 5×5，若想维持图像尺寸不变，则在原图像四周补上数据才能应用卷积操作，补充的数据的方式有多种，最简单的是用 0 填充，卷积中心从边缘开始，则可以得到和原图一样大小的图像。上述的卷积的滑动向左和向下的步长都是 1，在实际运用中，步长可以为其他正整数。比如，当步长为 2 时，边缘补充数据，卷积后

的图像大小的长和宽都是原图像的一半。

在实际的运算中，卷积运算会转换成矩阵相乘来实现，首先对原图像和卷积核都做展开，上述的例子则可以转变成：

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} & x_{31} & x_{32} & x_{33} \\ x_{12} & x_{13} & x_{14} & x_{22} & x_{23} & x_{24} & x_{32} & x_{33} & x_{34} \\ x_{13} & x_{14} & x_{15} & x_{23} & x_{24} & x_{25} & x_{33} & x_{34} & x_{35} \\ x_{21} & x_{22} & x_{23} & x_{31} & x_{32} & x_{33} & x_{41} & x_{42} & x_{43} \\ x_{22} & x_{23} & x_{24} & x_{32} & x_{33} & x_{34} & x_{42} & x_{43} & x_{44} \\ x_{23} & x_{24} & x_{25} & x_{33} & x_{34} & x_{35} & x_{43} & x_{44} & x_{45} \\ x_{31} & x_{32} & x_{33} & x_{41} & x_{42} & x_{43} & x_{51} & x_{52} & x_{53} \\ x_{32} & x_{33} & x_{34} & x_{42} & x_{43} & x_{44} & x_{52} & x_{53} & x_{54} \\ x_{33} & x_{34} & x_{35} & x_{43} & x_{44} & x_{45} & x_{53} & x_{54} & x_{55} \end{pmatrix} \begin{pmatrix} w_{11} \\ w_{12} \\ w_{13} \\ w_{21} \\ w_{22} \\ w_{23} \\ w_{31} \\ w_{32} \\ w_{33} \end{pmatrix} = \begin{pmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{21} \\ y_{22} \\ y_{23} \\ y_{31} \\ y_{32} \\ y_{33} \end{pmatrix}$$

对结果重排成 3×3 的矩阵得到最终结果。

卷积的有两点，第一点空间开销小。相比较两层神经元每一对都需要权值连接的全连接网络，卷积运算的空间开销远小于全连接。在深度学习中，卷积网络使用了共享权值的方法，即一个卷积核可以处理整张图像。将一个 5x5 的图像转变成 3x3 的图像，全连接需要 25x9=225 个权重，3×3 的卷积只需要 9 个权重。为了充分的获取图像的特征，通常使用多个卷积核处理同一个图像，这种情况下仍然比全连接的空间开销小。第二点，图像是具有局部性的，通过图像中某一区域得到的信息，和距离该区域较远的区域关系不密切，卷积操作只关注局部的运算方式符合人类观察图像的过程。

简略介绍反卷积操作，反卷积通常被用来实现数据的扩充。卷积和反卷积的运算过程是相反的，卷积的前向计算过程是反卷积的反向传播过程，卷积的反向传播过程是卷积的前向计算过程。

2.3 生成对抗网络

三年前蒙特利尔大学 Ian Goodfellow 等学者提出“生成对抗网络”(Generative Adversarial Networks，GANs)的概念[1]，2016 年，学界、业界对 GANs 的关注度突然增加。

关于生成对抗网络，Ian Goodfellow 给出的描述是“生成对抗网络是一种生成模型（Generative Model）[1]，其背后基本思想是从训练库里获取很多训练样本，从而学习这些训练案例生成的概率分布。”

对抗生成网络包含两个子网络，一个是判别器 Discriminator，一个是生成

器 Generator。生成器将输入的噪声生成与样本格式规模一样的数据，判别器则是要区分真实数据和生成器生成的伪造数据。传统的 GAN 通过如下的方式训练。

判别器接收一组真实数据 real data，进行计算之后的结果和 1 进行比较，计算交叉熵，记为判别器对真实数据的损失函数 dis_loss_real。生成器生成一组伪造数据 fake data，传入判别器进行计算，得到的结果和 0 进行比较，计算交叉熵，记为判别器对伪造数据的损失函数 dis_loss_fake，与 dis_loss_real 相加记为判别器的损失函数 dis_loss；得到的结果和 1 进行比较，计算交叉熵，记为生成器的损失函数 gen_loss。直观上的理解则是判别器尽可能的区分真实数据和伪造数据，所以 dis_loss_real 越小，表明判别器对真实数据的判断结果越接近 1，dis_loss_fake 越小，表明判别器对伪造数据的判别结果越接近 0，gen_loss 越小，表明生成器生成的数据越接近真实数据。

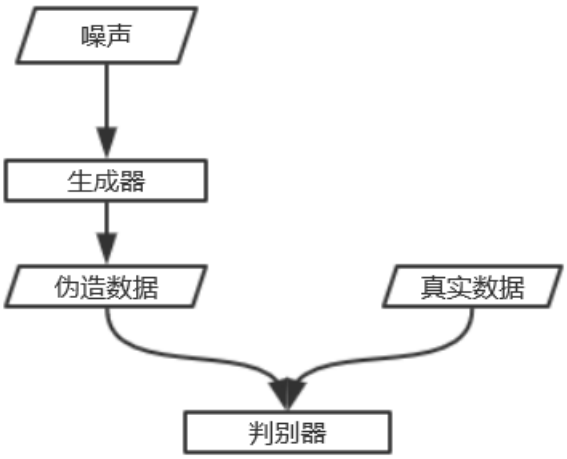

图 1-7 生成对抗网络示意

由上述的描述可以得知，判别器和生成器的训练是相互依赖的，判别器需要生成器提供伪造数据来训练判别器，生成器需要判别器对伪造数据的判断结果训练生成器。训练生成器的时候，判别网络不做更新；训练判别器的时候，生成网络不做更新。两者交替训练，最终的目的是让生成器生成足以欺骗人类的数据。

关于生成对抗网络，已经有很多的改进的版本，比如 ali-bi gan、boundary equilibrium gan、conditional gan、infogan 等，大部分的网络模型是在损失函数的计算方式和网络结构上做了一些改动，本质上没有改变 GANs 的性质。

第3章 Wasserstein GAN的关键技术

3.1 生成对抗网络的改进方案

首先总结不同版本的GANs对损失函数的改进方案。考虑原始的GANs，对一个样本，判别器的输出是一个数值，对该值做sigmoid激活后的结果与0或者1计算交叉熵。在实现中，输入为一组样本，记为一个batch，batch中样本的数量成为batch_size，一个batch中的样本同时训练，提升了训练的速度。在深度学习框架tensorflow中，sigmoid_cross_entropy_with_logits函数，传入的两个参数，logits是判别器输出的结果（不需要sigmoid激活，因为这个函数内部实现了sigmoid激活），labels是与logits规模相同的0或者1。

在调研中[13]，发现对此的改进主要有三个方向，一种是使用sigmoid激活之后用其他函数来处理，第二种是对输出的结果直接做处理，第三种，引入其他的度量标准，修改损失函数。

第一种以Ali bigan为例，经过sigmoid激活后的结果在[0,1]之间，用log函数可以拉开差距。记判别器对一个batch真实数据的判断结果经过sigmoid激活后为X_real,记判别器对一个batch伪造数据的判断结果经过sigmoid激活后为X_fake，判别器的损失函数为-reduce_mean(log(X_real)+log(1-X_fake))，生成器的损失函数为-reduce_mean(log(X_fake)+log(1-X_real))，其中reduce_mean表示对所有数据计算均值。使用梯度下降的方法最小化损失函数，达到对抗训练的效果。在Least Square GAN[13]中，真实数据和伪造数据经过判别器和sigmoid激活后的结果记为X_real,X_fake，判别器的损失函数为$(X_real-1)^2+(X_fake)^2$,生成器的损失函数为$(X_fake-1)^2$。关键思想在于判别器让真实数据的判定为1，伪造数据的判定为0，生成器让伪造的结果判定为1。

第二种，对输出的结果直接处理。比如Auxiliary Classifier GAN[13]，真实数据具有不同的标签，对标签进行独热码编码，将判别器的结果进行softmax，两者计算交叉熵。判别器对真实数据和伪造数据以及对应的标签，都希望得到正确的分类，但是，只凭借这个无法区分真实数据和伪造数据，因此需要最小化伪造数据经过判别器之后的结果，这就要求判别器有两组输出，一组是分类后的结果，另一组是一个值，经sigmoid激活之后和前面用相同的方法，判别器需要让伪造数据的第二种输出尽可能接近0,生成器则是希望分类正确并且输出接近1。

至于第三种，方法不唯一。比如MaGAN[13]使用阈值m，来对伪造数据的判断

结果做限定；比如 Mode Regularized GAN[13]通过计算真实数据和伪造数据的差值，来优化网络的。因为方法繁多，并且还使用了其他的优化，在此不做详细说明。

除了改进损失函数的计算方法，另一大类是对网络做修改，并且通常两者会结合使用。在对网络模型的优化中，是 Conditional GAN[13]，即条件生成对抗网络，很具有价值。CGAN 的最大特点是将数据的标签引入到判别器和生成器中，与 Auxiliary Classifier GAN 中使用标签的方法不同的是，Condition GAN 在数据的输入和计算过程加入标签，达到引入数据和控制数据变化方向的作用。比如，在手写数字的生成中，对于判别器，输入的是图像和标签，将标签独热码编码，一方面是加强标签的作用，另一方面是便于数据的拼接操作。在图像和标签的共同作用下，判别器对真实数据的判断为 1。对于生成器，输入为噪声和标签，输出的是生成的图像，再将图像和对应的标签传入判别器，若训练生成器则让判断结果和 1 比较，若训练判别器则让结果和 0 比较。借鉴了 CGAN 方法的 DCGAN 在手写数字生成上效果很好，生成的数字足以以假乱真。

3.2 Wasserstein 距离的优势

Wasserstein 距离[3][14]又叫 Earth-Mover（EM）距离，定义如下：

$$W(P_r,P_g)=\inf_{\gamma\sim\Pi(P_r,P_g)}\mathrm{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{3.1}$$

$\Pi(P_r,P_g)$ 表示 P_r 和 P_g 组合的所有可能的联合分布的集合，意味着 $\Pi(P_r,P_g)$ 中，每一个分布的边缘分布都是 P_r 和 P_g。对于每一个可能的联合分布 γ 来说，可以从中采样 $(x,y)\sim\gamma$ 得到一个真实样本 x 和一个生成样本 y，并计算这对样本的距离 $\|x-y\|$，因此可以计算该联合分布 γ 下样本对距离的期望值 $\mathrm{E}_{(x,y)\sim\gamma}[\|x-y\|]$。在所有可能的联合分布中对这个期望取下界 $\inf_{\gamma\sim\Pi(P_r,P_g)}\mathrm{E}_{(x,y)\sim\gamma}[\|x-y\|]$，定义成 Wasserstein 距离。

关于 Wasserstein 距离，用一个形象的比喻来解释，在 γ 这个“路径规划”下，把 P_r 这堆“沙土”挪到 P_g 位置所需要的消耗看作 $\mathrm{E}_{(x,y)\sim\gamma}[\|x-y\|]$，那么 $W(P_r,P_g)$ 则是“最优路径规划”下的“最小消耗”，故此又被称为 Earth-Mover（推土机）距离[14]。

在原始的对抗生成网络中，当判别器近似最优的时候，最小化生成器的损失函数等价于最小化 P_r 和 P_g 之间的 JS 散度。关键问题是，P_r 和 P_g 之间几乎不可能有不可忽略的重叠，所以 JS 散度为常数 log2[2]，对常数的求导得到 0，因此生成器的梯度近似为 0 了，出现梯度消失问题，无法继续有效的训练下去[14]。

原始的 GAN 不稳定，原因是如果判别器训练的太好，生成器梯度消失严重，那么生成器的 loss 降不下去；如果判别器训练的不够，生成器梯度不准确，那么会导致生成器的梯度不稳定。故此需要判别器训练状态处于一个既不能太好又不能太坏的状态，才能让生成器有个稳定并且不消失的梯度，找到这个状态很困难，训练的前期和后期都可能不一样，也就导致设想很好的 GAN 训练起来很困难。

那么 Wasserstein 距离的优势在于，相比 JS 散度，即使 P_r 和 P_g 分布没有重叠，它都可以反应它们的远近。并且，相比 JS 散度的突变，Wasserstein 距离是平滑的，能够提供梯度。

尽管 Wasserstein 距离很有优势，但是实际的计算中，$\inf_{\gamma\sim\Pi(P_r,P_g)}$ 没法直接求解，作者用一个已有的定理将 Wasserstein 距离转变为了如下形式[3]：

$$W(P_r,P_g)=\frac{1}{K}\sup_{\|f\|_L\le K}\mathrm{E}_{x\sim P_r}[f(x)]-E_{x\sim P_g}[f(x)] \tag{3.2}$$

式(3.2)表示，在保证函数 f 的 Lipschitz 常数 $\|f\|_L$ 不超过 K 的条件下，对所有可能满足条件的函数 f 取到 $\mathrm{E}_{x\sim P_r}[f(x)]-E_{x\sim P_g}[f(x)]$ 的上界，然后再除以 K。式(3.2)变成[3]：

$$K\cdot W(P_r,P_g)\approx\max_{w:|f_w|_L\le K}\mathrm{E}_{x\sim P_r}[f_w(x)]-E_{x\sim P_g}[f_w(x)] \tag{3.3}$$

利用神经网络强大的拟合能力，足以找到一系列的近似公式 3.2 要求的 f_w。至于 $\|f\|_L\le K$，对权重做范围的限制，即令所有的参数 w_i 不超过范围[-c,c]即可保证关于输入样本 x 的导数 $\frac{\partial f_w}{\partial x}$ 也不会超过某个范围。在 tensorflow 中，使用 clip_by_value 函数对判别器网络的权重进行限制。

3.3 Wasserstein GAN 的优势

上一节提到，原始 GAN 的判别器越好，生成器梯度消失越严重；最小化生成器的损失函数 loss，会等价于最小化一个不合理的距离衡量。导致原始 GAN 出现两个问题，梯度不稳定和多样性不足[14]。多样性不足指的是和真实数据相比，

生成的结果可能偏向于某一种或者少量的几种，例如手写数字的生成中，原始的GAN在没有标签控制生成方向的时候，结果会向着特征简单的方向生成，比如生成的样本大部分是1，而那些特征复杂的数字，在样本中出现的次数很少。

Wasserstein GAN与原始的GAN相比，只修改了四点[14]：

- 判别器的最后一层去掉sigmoid
- 生成器和判别器的loss不取log
- 每次更新判别器的参数之后把它们的绝对值截断到不超过一个固定常熟c
- 不要用基于动量的优化算法(包括momentum和Adam),推荐使用RMSProp或者SGD。

在实验中发现，对于已有的较为成熟有效的生成对抗网络模型中，在不改变网络结构的情况下，结合Wasserstein GAN可以达到相同的效果。但是，较为成熟的生成对抗网络模型中，使用了很多的优化方法，诸如批归一化等操作。比如DCGAN中，使用了反卷积和批归一化，并且引入标签来补充数据，去掉这些优化方法，模型将无法正常工作。但是，使用Wasserstein GAN，仅使用简单的全链接网络，就可以生成一些图片或者出现轮廓。在没有标签参与的情况下，多数GANs 都出现了多样性不足的情况，Wasserstein GAN 却避开了这一点。此外，Wasserstein GAN的损失函数可以指示训练进度，传统的GAN的损失函数上下抖动的较为剧烈，无法作为指示。

第4章 对 Wasserstein GAN 在图像生成上的改进

由于神经网络结构和损失函数以及参数的复杂性，下述在进行对比实验的时候，均是经过大量调参，找到尽可能适合当前网络模型的参数。在使用不同的损失函数时，对学习率相同的限制没有意义。对于 GAN 生成的图像效果，尚未找到合适的衡量标准，将以本人和同学的主观感受作为衡量标准，判断生成图像的质量，再结合网络训练的用时，评估模型的性能。

4.1 深度卷积神经网络识别验证码

正如程序员的第一课是 hello world，神经网络的第一课是手写数字识别，使用的数据集是 mnist[7]，本人在实践了 caffe[8]训练并识别 mnist 数据集之后，修改网络结构，做学校教务系统验证码的识别，并移植到 tensorflow[6]上，因为 python 语言的灵活性，以下全部用 tensorflow 作为框架来描述。

4.1.1 网络构建

首先描述数据集的特征，验证码图片从 http://mis.teach.ustc.edu.cn/randomImage.do 这个地址使用 python 爬虫抓取，此 url 是原先学校教务系统本科生登陆验证码的地址，现在已经弃用，每一张验证码由四个字母组成，图像大小是 20x80 的，水平四等分之后能把验证码分隔开。验证码包含数字和大写字母，除了数字 0、1 和字母 I、O，一共 32 种字符，用 python 将验证码转化为灰度图做阈值分割并且四等分之后，人工的给分割后的验证码分类，并制作标签，用 python 转化成 mat 文件格式（matlab 的文件格式）。

图 4-1 验证码原图示例

图 4-2 验证码初步处理并做分割后的图片

参考 lenet-5 对手写数字做分类的网络模型，我自己的网络模型如下：

(1) 数据输入：(batch_size,20,20,1)的图像，表示 batch_size 个长和宽为 20 的灰度图；

(2) 第一层卷积层，卷积核大小是(5,5)，卷积核个数 20，卷积步长(1,1,1,1)，即在各个维度上步长都为 1，卷积的结果需要加上偏置项，偏置项的长度和卷积核个数相同。一张图像经过 20 个卷积核之后，得到 20 张图像，每一张图像都加上各自的偏置项，一张图的所有像素点的数据所加的值相同。卷积

的设置是不填充边界的，因此卷积之后的数据规模是(batch_sizde,16,16,20)；

(3) 第二层激活层，对数据进行 ReLU 激活，即只保留正数部分，负数部分置为0，数据规模不变化；

(4) 第三层池化层，使用最大值池化，池化的核大小为(1,2,2,1)，步长为(1,2,2,1)，数据在第二、三维度上缩小一倍，等同于一张图像按比例放缩为原来的 25%，并且每个像素点对应的是原来 4 个像素点中的最大值。此时数据规模是(batch_size,8,8,20)；

(5) 第四层卷积层，卷积核大小是(5,5)，卷积核个数 50，步长均为 1，得到的结果仍旧加上偏置项。每个核都对 20 个从上一层接收的(8,8)的图像做处理，将 20 张图像按权重线性累加得到当前卷积核输出的结果，累加操作由 tensorflow 框架内部实现。卷积图像不填充边界，结果的数据规模为(batch_size,4,4,50)；

(6) 第五层激活层，ReLU 激活，数据规模仍是(batch_size,4,4,50)；

(7) 第六层池化层，池化方式和第三层相同，结果的数据规模为(batch_size,2,2,50)；

(8) 第七层全连接层，对 batch_size 个第六层得到的规模为(2,2,50)的数据，每一个重排为一个规模为 2*2*50=200 的一维向量，叉乘一个(200,500)的矩阵，加上长为 500 的偏置项，得到长为 500 的一维向量。最后的结果是(batch_size,500)的二维向量；

(9) 第八层激活层，ReLU 激活，数据规模不变；

(10) 第九层全连接层，上层结果叉乘(500,32)的矩阵之后加上长 32 的偏置项，结果的数据规模为(batch_size,32)；

(11) 第十层 softmax 层，对上层结果做 softmax 回归，得到类似独热码的结果，其中一个数较大，接近 1，其他数较小，接近 0，数据都在[0,1]之间，数据规模不变；

(12) 接下来计算损失函数，将进行独热码编码后的真实标签与取对数后的上一层结果相乘并累加，取相反数作为损失函数。

(13) 以上是训练的网络，训练过程中计算准确率，将第十层的结果中最大的数的索引和独热码编码后的真实标签中为 1 的数的索引进行比较，统计相等的个数。

4.1.2 关键程序实现

训练模型的输入为图像数据和标签，在附 1 中为按照本节 a)中所述网络结构展示关键代码。

训练使用 AdamOptimizer 算法梯度下降最小化 loss。

```
train_operate = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

训练过程如下：

```
sess.run(train_operate, feed_dict={x: input_x, y: input_y})
```

4.1.3 实验结果

使用 tensorflow 自带的工具[9]tensorboard 显示 loss 和准确率：

图 4-3 训练过程中的损失函数 loss 和准确率

从图 4-3 中可以看出，本次的训练结果非常好，损失函数接近 0，准确率近乎为 1。保存模型并另抓取验证码图片测试：

图 4-4 抓取的验证码图片和识别结果

实际上，在实验中，最初获取的数据较少，约 400 张，进行手工的分类，用手动分类之后的结果做训练，准确率大概为 92%。抓取更多的数据使用模型进行分类，再人工检查，修正一些错误，用新的数据进行训练，准确率继续上升。在准确率达到 97%左右的时候，用模型去检测数据集，又找出了之前人工没有发现的数据分类的错误。最后实验准确率达到 100%，已经超过人的识别准确度。

4.2 深度卷积网络生成验证码

上一节详细的讲述识别验证码所用的卷积神经网络 CNN 的结构和具体细节，为本节的深度卷积 DCGAN 生成对抗网络打下一个良好的基础。

在 CNN 的识别过程中，用到了池化层，将 2x2 的像素缩减为了 1x1 的像素，只保留最大值，故此损失了许多数据。为了减少数据的损失，将使用步长为 2 的卷积代替 CNN 中的卷积和池化操作。

对于对抗生成网络 GAN，判别器和生成器没有规定要结构相逆，但是，实际操作中我们构建的判别器和生成器结构相逆，不仅在逻辑上更清晰，视生成器为判别器的逆过程，而且，效果会较好。原因是若一个良好的判别器既然能够做到良好的识别，则意味着它对特征的捕捉是准确的，那么和它结构相逆的生成器网络在一定程度上会还原相应的特征。

基于上述，首先所做的是 DCGAN 的实现。

4.1.4 网络构建

判别器的网络结构：

(1) 数据输入和上一节中相同，但是不包含标签；

(2) 第一层卷积层，卷积核大小(5,5)，个数 64，步长为(1,2,2,1)，用 0 填充边界，得到规模为(batch_size,10,10,64)的数据；

(3) 第二层激活层，使用 leaky ReLU 激活，正数部分保持不变，负数部分乘以一个微小量，本程序设置为 0.2，数据规模不变；

(4) 第三层卷积层，卷积核大小(5,5)，个数 128，步长为(1,2,2,1)，用 0 填充边界，得到规模为(batch_size,5,5,128)的数据。

(5) 第四层激活层，使用 leaky ReLU 激活，数据规模不变；

(6) 第五层全连接层，将上层得到的每个样本的数据排列成长为 5*5*128=3200 的一维向量，叉乘 3200*1024 的矩阵，得到长 1024 的一维向量，数据规模为(batch_size,1024)；

(7) 第六层归一化层，对数据进行批归一化操作，数据规模不变；

(8) 第七层激活层，使用 leaky ReLU 激活，数据规模不变；

(9) 第八层全连接层，上层得到的数据叉乘 1024*1 的矩阵，得到数据规模为(batch_size,1)的二维向量；

(10)第九层激活层，使用 sigmoid 函数激活，数据规模不变。

生成器的网络结构：

(1) 数据输入为 batch_size 个长 128 的噪声向量；

(2) 第一层全连接层，噪声向量叉乘 128*1024 的矩阵，得到数据规模为(batch_size,1024)的二维向量；

(3) 第二层归一化层，对上层得到的数据进行批归一化操作，数据规模不变；

(4) 第三层激活层，使用 ReLU 激活，数据规模不变；

(5) 第四层全连接层，上层数据叉乘 1024*3200 的矩阵，得到数据规模为

(batch_size,3200)的二维向量。

(6) 第五层归一化层，对上层得到的数据进行批归一化操作，数据规模不变；

(7) 第六层激活层，使用 ReLU 激活，数据规模不变；

(8) 第七层反卷积层，将上层得到的数据重新排列成(batch_size,5,5,128)四维向量，进行反卷积操作，卷积核大小(5,5)，共有 128 个卷积核，使用卷积反向传播的算法得到 64 张图像，卷积步长为(1,2,2,1)，得到的数据规模为(batch_size,10,10,64)的四维向量；

(9) 第八层归一化层，对上层得到的数据进行归一化操作，数据规模不变；

(10)第九层激活层，使用 ReLU 激活，数据规模不变；

(11)第十层反卷积层，卷积核大小(5,5)，个数 64，输出 1，步长为(1,2,2,1)，得到数据规模为(batch_size,20,20,1)的四维向量；

(12)第十一层激活层，使用 sigmoid 激活，使数据范围转为[0,1]之间，数据规模不变，得到生成的图像。

损失函数的计算：

将一组噪声向量输入生成器，得到一组生成的图片，将这组图片放入判别器，输出结果记为 fake；将一组真实图像放入判别器，输出的结果记为 real。

判别器损失函数由两部分组成，第一部分对真实图片的判别准确程度，将 real 与一组 1 进行比较计算交叉熵，第二部分对伪造图片的判断准确程度，将 fake 与一组 0 进行比较计算交叉熵，两部分加起来得到判别器的损失函数。

生成器的损失函数是将 fake 与一组 1 进行比较计算交叉熵，体现了生成器“造假”的程度。

4.1.5 关键程序实现

生成器网络和判别器网络以及损失函数的程序见附 2。

训练过程使用 AdamOptimizer 梯度更新的方法最小化判别器和生成器的损失函数。

```
train_op_dis = tf.train.AdamOptimizer(learning_rate, beta1=0.5).minimize(dis_loss, var_list=dis_vars)
train_op_gen = tf.train.AdamOptimizer(learning_rate, beta1=0.5).minimize(gen_loss, var_list=gen_vars)
```

训练过程如下：

```
sess.run(train_op_dis, feed_dict={x: input_x, z: input_z})
sess.run(train_op_gen, feed_dict={z: input_z})
```

经过实验得到的结果是判别器和生成器的训练比例是 1:5，只代表此网络的

训练判别器的生成器的训练比例。

4.1.6 实验结果

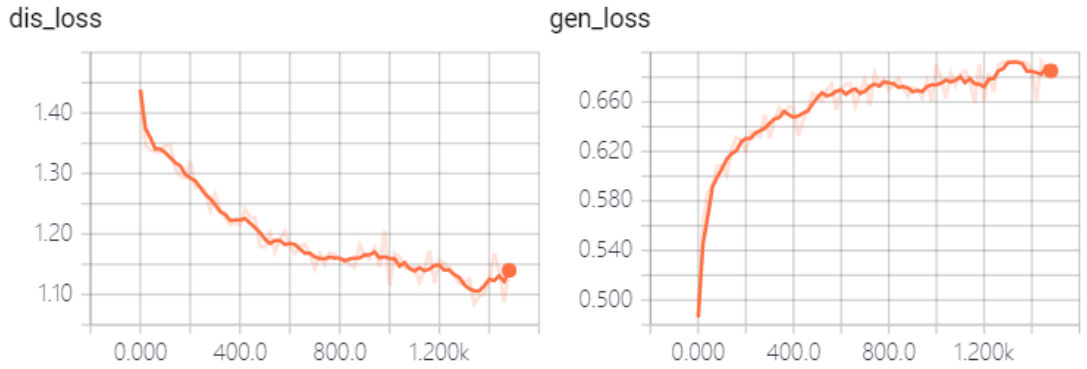

图 4-5 DCGAN 的损失函数状态

从损失函数中并不能看出训练的效果，在第 800 次迭代、耗时 184 秒的结果如图 4-6：

图 4-6 DCGAN 生成验证码示例

此时已经可以看出大部分的验证码，效果尚可，倘若继续训练下去，则会因为判别器训练的过好，生成器梯度消失，无法生成验证码。800 之后的判别器的损失函数变换不明显，图像生成不准确，图 4-7 表示了在第 1260 次迭代（左）和第 1480 次（右）的结果。

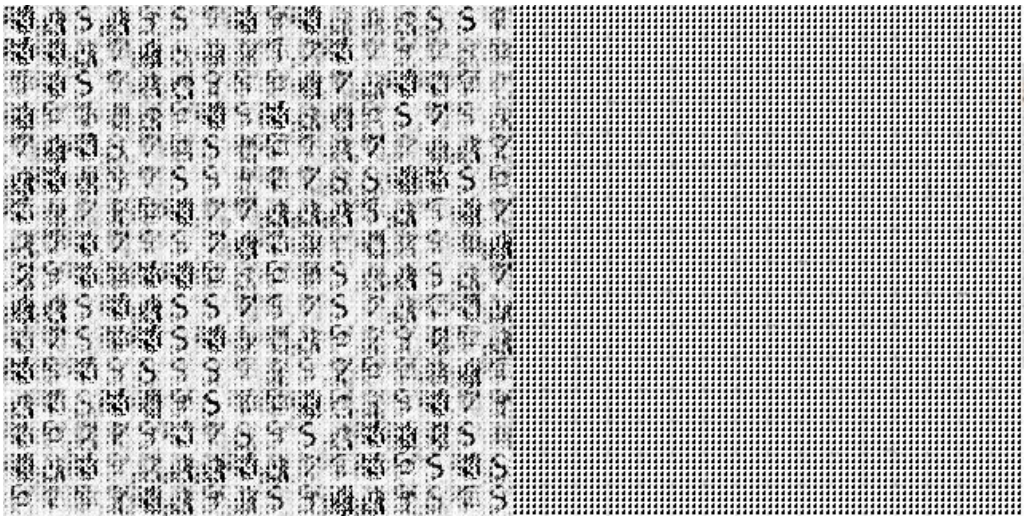

图 4-7 DCGAN 生成验证码示例

4.3 Wasserstein GAN 生成验证码

在 3.3 节中提到，即使是使用简单的全连接网络，Wasserstein GAN（以下简称 WGAN）都有很好的效果。下面的实验包含两个，一个是使用简单的全连接网络进行验证码的生成，另一个是基于 4.2 中的 DCGAN 进行修改后的网络进行验证码的生成。

首先是全连接网络[13]。

判别器的网络结构：第一层将图像转变形状为一个长 400 的一维向量，叉乘 400*500 的矩阵之后加上一个长 500 的偏置项，得到长 500 的一维向量，然后使用 ReLU 激活；第二层全连接到长为 1 的一维向量并加上偏置项，作为判别器的输出。

生成器的网络结构：第一层将噪声向量全连接成长 500 的一维向量并加偏置项，进行 ReLU 激活；然后全连接成长 400 的一维向量并加偏置项；最后使用 sigmoid 激活。

整个网络十分简单，没有复杂的卷积、反卷积，没有归一化操作，生成器和判别器只有简单的三四层，因此每次网络的迭代都非常快。

损失函数的计算是 WGAN 的重点，记 real 为真实数据经过判别器之后输出的结果，记 fake 为伪造数据经过判别器之后输出的结果。那么判别器的损失函数为 fake 的均值减去 real 的均值，生成器的损失函数为 fake 的均值的相反数。使用的 tensorflow 的程序为：

```
dis_loss = tf.reduce_mean(fake) - tf.reduce_mean(real)
gen_loss = -tf.reduce_mean(fake)
```

同时为了满足式 3.3 中关于 Lipschitz 常数的限定，$\| f \|_L \leq K$，对判别器网络中的参数进行了范围限制，其中限制的范围为 0.05，这个数值是需要多次实验才能确定的。进行 clip 的操作为：

```
clip_dis = [p.assign(tf.clip_by_value(p, -0.05, 0.05)) for p in dis_vars]
```

其中 dis_vars 表示的是判别器中参与计算的参数，包括两层全连接网络的权重和偏置项。

梯度更新使用非动量的算法，程序为：

```
train_op_dis = tf.train.RMSPropOptimizer(learning_rate).minimize(dis_loss,
var_list=dis_vars)
train_op_gen = tf.train.RMSPropOptimizer(learning_rate).minimize(gen_loss,
var_list=gen_vars)
```

训练过程为：

```
sess.run([train_op_dis, clip_dis], feed_dict={x: input_x, z: input_z})
sess.run([train_op_gen], feed_dict={z: input_z})
```

和前面的网络相比，判别器的训练多出了 clip_dis 操作，即对参数范围的限定。

判别器和生成器的训练比例设置，经实验选择了 3:1。

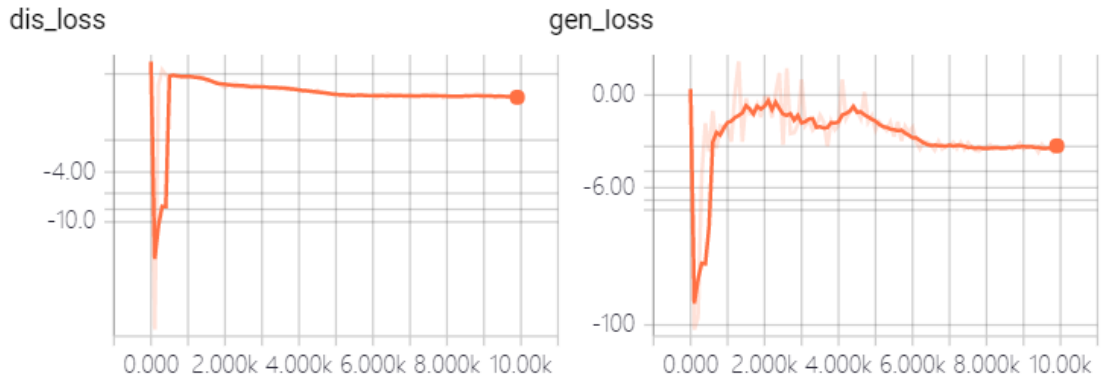

图 4-8 损失函数变化趋势，对数纵坐标。

图 4-9 WGAN 生成示例

从图 4-8 可以观察到，大约 800 次迭代之后，判别器的距离趋近 Wasserstein 距离，两个损失函数都开始稳步下降。6000 次之后趋于平缓，意味着网络已经训练不动。经过 8500 次迭代，耗时 171s，得到的图像如图 4-9 所示，人肉眼能识别少部分的图像，在使用简单的网络结构就可以达到这种效果，Wasserstein GAN 确实很有优势。

对比实验，使用原始的 GAN 的损失函数以及相同的网络结构，无论怎样调参，生成的结果都是千篇一律的样式，并且很容易出现梯度消失训练不动的情况。生成的图像组合在一起仿佛是墙纸花纹，之前在某处看到的关于 GAN 功能的描述，在生成花纹和重复样式上效果很好。如图 4-10：

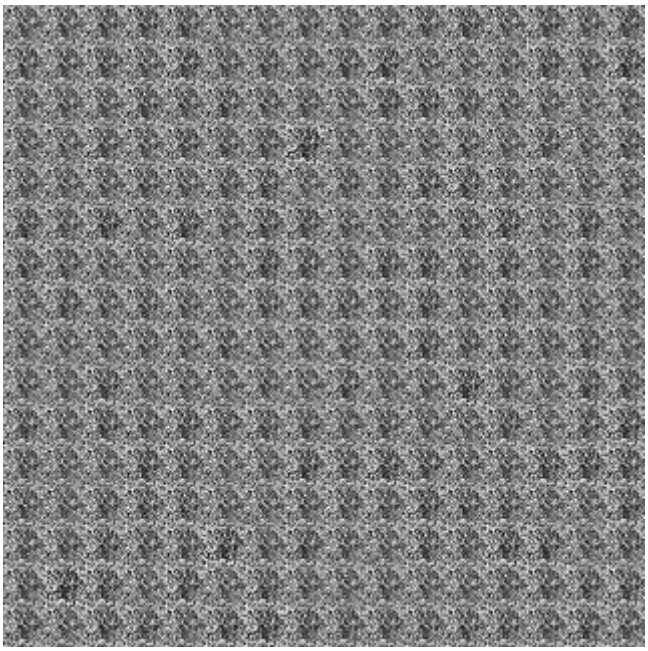

图 4-10 GAN 生成示例

对比了全连接网络，下面结合 Wasserstein GAN 的方式对 4.2 中的 DCGAN 进行改进。

在网络结构不变的情况下，只修改了损失函数和梯度更新的方式，并增加了对判别器中参数的范围限制。判别器和生成器训练比例为 1:5。称此网络为 WDCGAN

在第 340 次迭代，耗时 77 秒时，生成图像如图 4-11（左）所示，效果已和 4.2 中 DCGAN 在第 800 次迭代、耗时 184 秒时效果(图 4-11 右)接近。

图 4-11 WDCGAN 与 DCGAN 示例对比

在第 460 次迭代、104 秒时，效果已经很可观，如图 4-11，此时不仅在分辨程度上相比 DCGAN 有了提升，对比度也增强了许多，用时上也更短，足以表现出

Wasserstein GAN 的优势。

使用 Tensorflow 自带的工具绘制损失函数变化如图 4-12 所示，从第 500 次迭代之后，判别器损失函数下降减缓，生成器的损失函数平稳，略有上升，此时应该增加生成器的训练次数，考虑到此时生成的效果，后续的调整不再做。

图 4-11 WDCGAN 生成效果示意

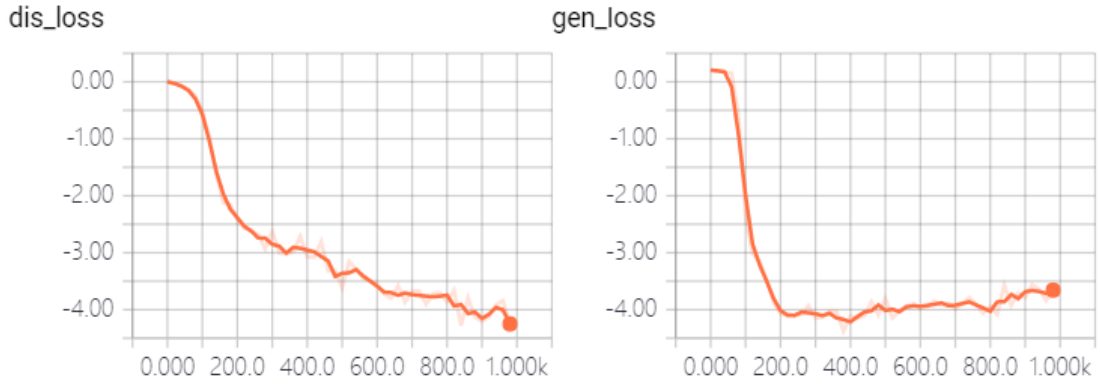

图 4-12 WDCGAN 损失函数

4.4 对 Wasserstein GAN 的优化

考虑 Wasserstein GAN 中做的优化方案，对选择 Wassertein 距离作为损失函数深表叹服，同时在多次实验之后发现问题，第一，若是单纯的使用全连接网络，将会因全连接网络参数规模大而过于消耗内存，需要保存的权重和偏置项很多，并且生成的图像不够平滑，噪点很多。第二，使用现有的较好的模型，只修改判别函数，单次的训练速度会下降，不足以充分利用 Wasserstein 距离的优越性。第三，对于复杂的但是共性很多的图像，如 celeba 的人脸数据集，使用较

为简单的网络，如全连接网络，仍会出现多样性不足的情况。第四，经过限制范围的判别器网络的参数，将会出现两极分化的情况，近似的变成了二值网络，没有充分利用网络的泛化能力。

故此，本人试图对 Wasserstain GAN 做一些改进。

对于全连接网络会出现噪声，一个原因是生成器中的某些参数可能过大或者过小，设想对判别器所做的限制也可以加到生成器里面去，以此来减少某些参数过大或者过小的情况，同时在最后一层增加一层卷积层来对图像起平滑的作用。

对 WDCGAN 做如下调整：

(1) 对原生成器倒数第二层的反卷积操作，最后一层的 sigmoid 激活改为 tanh 激活。增加一层卷积层，卷积核大小视图像而定，当前程序定为(5,5)，卷积步长为 1，边缘用 0 填充保证图像大小不变，卷积输出为(batch_size,20,20,1)的四维向量。增加一层 sigmoid 激活层。从附 2 的(11)开始修改。

```
gen_h3 = tf.nn.tanh(gen_h3)
gen_h4 = tf.nn.conv2d(gen_h3, self.gen_W5, [1, 1, 1, 1], "SAME")
gen_h4 = tf.nn.sigmoid(gen_h4)
```

(2) 修改参数范围限制的操作。对判别器只限制全连接层的权重，不限制偏置项和其他。对生成器的反卷积和全连接的权重加以限制，其他不做限制，相应的训练过程也做修改。

```
clip_dis = [p.assign(tf.clip_by_value(p, -0.05, 0.05)) for p in dis_vars if
"dis_f_W" in p.name]
clip_gen = [p.assign(tf.clip_by_value(p, -0.05, 0.05)) for p in gen_vars if
"gen_f_W" in p.name or "gen_dc_W" in p.name]
sess.run([train_op_dis, clip_dis], feed_dict={x: input_x, z: input_z})
sess.run([train_op_gen, clip_gen], feed_dict={z: input_z})
```

实验结果如下，对于这个已经效果很好的 WDCGAN，优化的作用不是很明显，第 300 次迭代、耗时 83s 时效果（图 4-13 左）足以媲美 WDCGAN 中第 460 次迭代、104s 时的效果（图 4-13 右）。

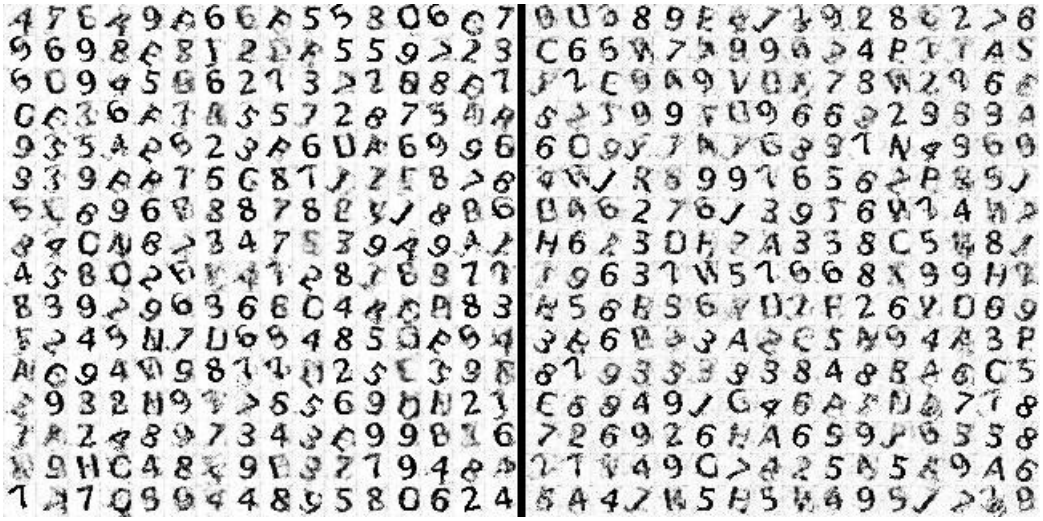

图 4-13 优化后 WDCGAN（左）和原 WDCGAN（右）对比

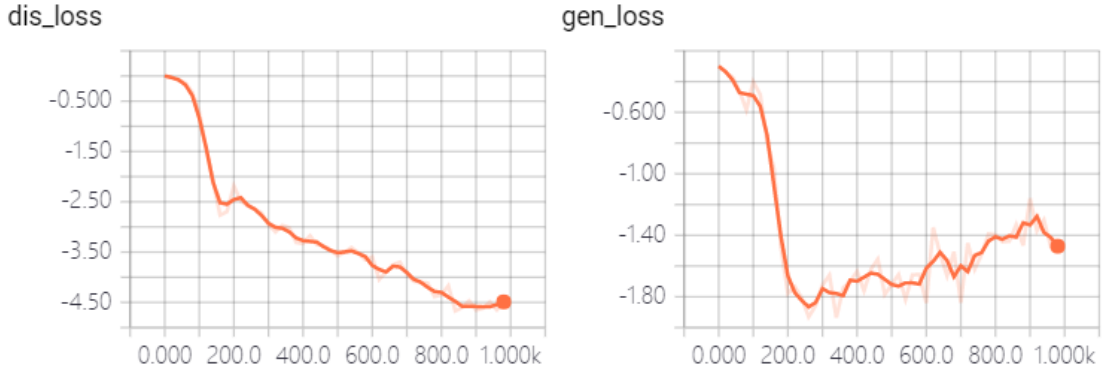

图 4-14 优化后的 WDCGAN 的损失函数

从图 4-14 的损失函数变化可以看出，生成器的变化更稳定，但是由于学习率以及判别器和生成器的训练比例未做调整，生成器的损失函数后期有小幅上升，修改方案应当增加生成器的训练次数。

上述的实验均是不引入标签控制对抗生成网络生成方向的，引入标签之后效果效果更好。引入标签的方式是将标签进行独热码编码，再拼接到网络的神经元中。例如对全连接的输出，使用 concat 函数将标签拼接到输出上，作为下一层的输入。

```
dis_h1 = tf.concat([dis_h1, label], 1)
```

以下分别是带标签的DCGAN_label、增加Wasserstein优化的WDCGAN_label、增加作者优化方案的 WDCGAN_label_update 生成的图像。

图 4-15 DCGAN_label(左)、WDCGAN_label(中)、WDCGAN_label(右)效果示意

在达到相似效果时，迭代次数和用时分别为：780 次 187 秒、580 次 137 秒、280 次 78 秒。从最后的结果来看，后者在生成的速度上是优于前者的。

重新描述优化的方法：在 Wasserstein GAN 的基础上，对于简单网络，将生成器最后一层的 sigmoid 激活改为 tanh 激活，增加一层卷积层，最后仍用 sigmoid 激活作为输出。只对判别器的全连接的权重做 clip，对生成器的全连接层和反卷积层（即能起到扩展数据的网络层）的权重做 clip。前期判别器的训练次数较多，等判别器的损失函数开始稳步下降，意味着拟合结果逼近 Wasserstein 距离，此时增加生成器的训练次数，减少判别器的训练次数。

第5章　对抗生成网络在人脸生成上的实验

5.1　程序封装说明

对于验证码的数据集，每一类的字符有其自己的特征，所有的数据共同的特征并不明显，也就会出现 4.3 中两层全连接 GAN 中出现的结果，找出了所有数据的共性。

一组共性明显的数据集为人脸数据集，实验一共使用了两个数据集，UMass 的 LFW 人脸数据集[10]和 CelebA 数据集[11]，前者数量较少并且人脸在图片中所占的比例较少，后者数据量更多并且共性更明显。下面将以 CelebA 数据集为例描述实验，图像转为 128*128 的三通道彩色图像，并转为 tensorflow 专用数据格式，便于从磁盘中并发读取，不占用大量内存。

为了简化网络的构建，本人对 tensorflow 中的部分函数做了二次封装，对程序的封装见附 3。

5.2　DCGAN

网络结构，为方便描述，以 batch 中的一个样本作为输入单元，批归一化操作是在 batch 内的操作，其他操作与 batch 内的其他样本数据无关。

判别器：

(1) 对输入数据进行卷积，卷积核大小为(5,5)，个数为 32、步长为 2，输出的数据规模为(64,64,32)的三维向量；

(2) 对上一层输出的数据进行 ReLU 激活

(3) 对上一层输出的数据进行卷积，卷积核大小为(5,5)，个数为 64，步长为 2，输出的数据规模为(32,32,64)的三维向量；

(4) 对上一层输出的数据做批归一化；

(5) 对上一层输出的数据进行 ReLU 激活；

(6) 对上一层输出的数据进行卷积，卷积核大小为(5,5)，个数为 128，步长为 2，输出的数据规模为(16,16,128)的三维向量；

(7) 对上一层输出的数据做批归一化；

(8) 对上一层输出的数据进行 ReLU 激活；

(9) 对上一层输出的数据进行卷积，卷积核大小为(5,5)，个数为 256，步长为 2，输出的数据规模为(8,8,256)的三维向量；

(10)对上一层数据进行重排，每一张图像对应的数据转为长 16384 的一维向量，全连接到 1 个神经元；

(11)对上一层输出的数据做 sigmoid 激活。

生成器：

(1) 将噪声向量全连接到长 16384 的一维向量；

(2) 对上一层输出的数据做 ReLU 激活，重排为(8,8,256)的三维向量；

(3) 对上一层输出的数据进行反卷积，卷积核大小(5,5)，步长为 2，得到规模为(16,16,128)的三维向量；

(4) 对上一层输出的数据做批归一化；

(5) 对上一层输出的数据做 ReLU 激活；

(6) 对上一层输出的数据进行反卷积，卷积核大小(5,5)，步长为 2，得到规模为(32,32,64)的三维向量；

(7) 对上一层输出的数据做批归一化；

(8) 对上一层输出的数据做 ReLU 激活；

(9) 对上一层输出的数据进行反卷积，卷积核大小(5,5)，步长为 2，得到规模为(64,64,32)的三维向量；

(10)对上一层输出的数据做批归一化；

(11)对上一层输出的数据做 ReLU 激活；

(12)对上一层输出的数据进行反卷积，卷积核大小(5,5)，步长为 2，得到规模为(128,128,3)的三维向量；

(13)对上一层输出的数据做 sigmoid 激活。

损失函数与 4.2 中损失函数计算方式相同，不做赘述。判别器和生成器的训练比例 1:2。

在第 7200 次迭代、耗时 1240s 时得到较为清晰的图片，如图 5-1，图像虽然扭曲严重，但是细节效果尚可。训练过程中判别器和生成器的损失函数变化如图 5-2。

图 5-1 DCGAN 生成图像示例

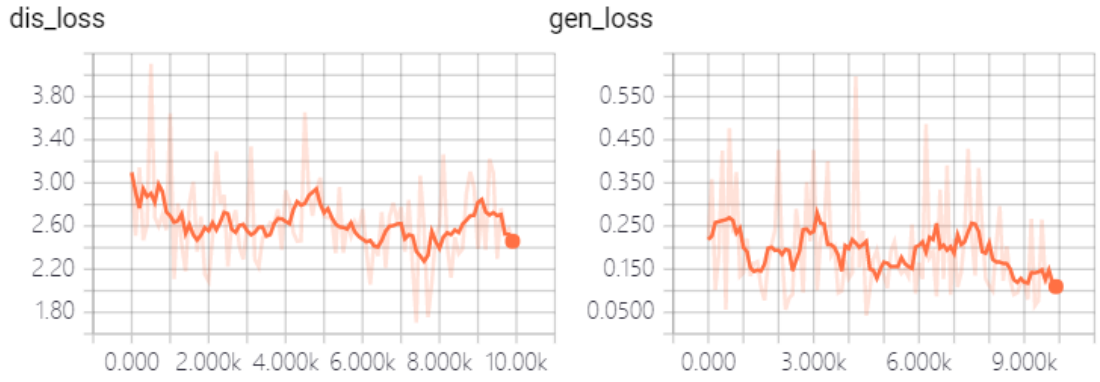

图 5-2 DCGAN 损失函数变化

5.3 WDCGAN

在(a)中修改了损失函数和梯度更新方法，增加了对判别器的 clip，学习率不变，在 25 次迭代以内，判别器与生成器的训练比例为 5:2，之后比例为 2:1，前期增加判别器训练次数是为了使判别器的损失函数大致满足 Wasserstein 距离。

具体实现的细节参考 4.3 中的方案即可，训练到第 5300 次、耗时 846 秒，生成的人脸效果已经接近 5.2 中展示的效果，如图 5-3，损失函数的变化如图 5-4 所示，大约 3000 次迭代之后，判别器的损失函数平稳下降，生成器的损失函数波动较为剧烈，应当适当增加生成器的训练次数。

图 5-3 WDCGAN 生成图像示例

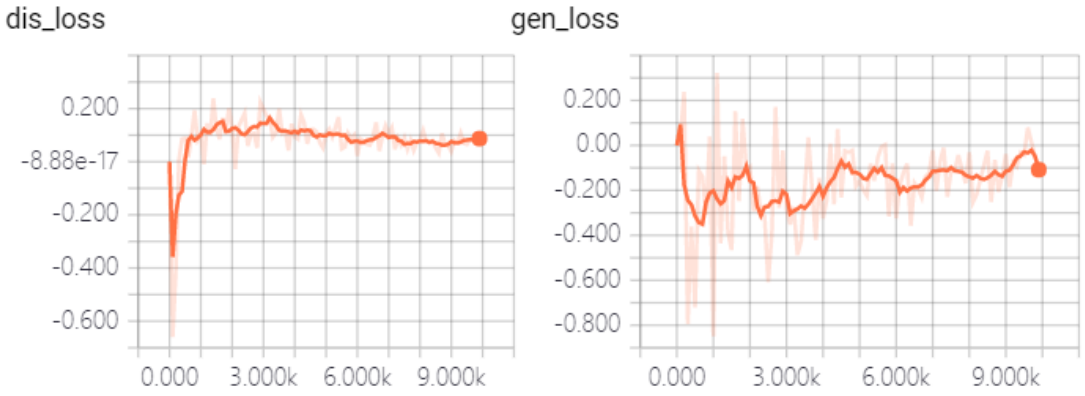

图 5-4 WDCGAN 损失函数变化

5.4 WDCGAN 的优化

应用自己对 WGAN 的改进方法，其他参数不变，对 WDCGAN 的生成器增加一层卷积层平滑噪点，实际上这个改动可以不需要，因为原来的生成效果已经没有多少噪点。对判别器和生成器网络的参数增加限制，只对全连接层和反卷积层这种起到扩充数据作用的网络层的权重限制范围。

具体的效果看实验的结果，在第 3300 次迭代、耗时 687 秒时，生成图像如图 5-5 所示，结合图 5-1、图 5-3 对比表明，优化方案有效，对于复杂网络，可以缩短训练时间。从图 5-6 中损失函数的变化发现，判别器训练的程度不够，应该适当的增加判别器的训练。

图 5-4 WDCGAN 优化后结果示例

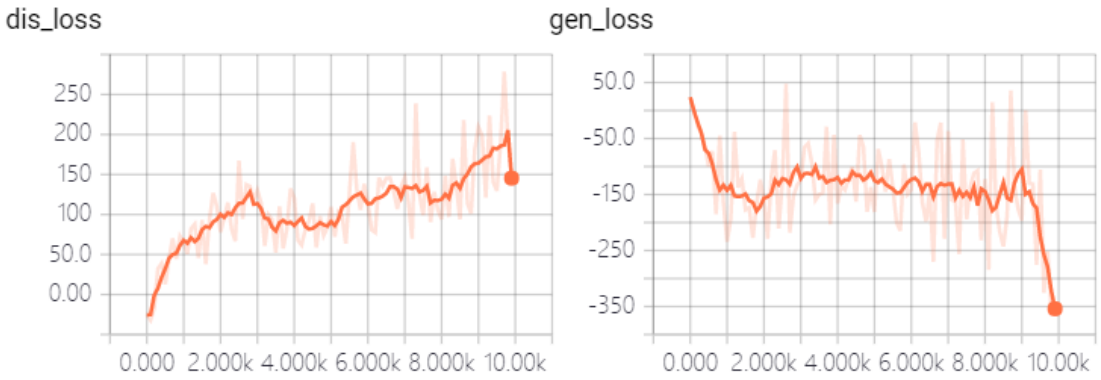

图 5-6 WDCGAN 优化后损失函数变化

5.5 WGAN

DCGAN 的网络结构较为复杂，效果虽好但是速度慢、并且图像扭曲，类似于梵高的画风，由于 Wasserstein GAN 使用简单的全连接网络也可以训练，本次实验则去实践这种方法。

判别器：图像重排成一维向量，全连接到长 1024 的一维向量，做 leaky ReLU 激活，再全连接到长为 1 的向量做输出。

生成器：噪声全连接到长 1024 的一维向量，做 ReLU 激活，再全连接到图像尺寸，做 sigmoid 激活，重排成和图像数据一样的尺寸 128*128*3。

因为网络结构简单，训练花费的时间步长，在进行充分训练之后生成的图像如图 5-6，遗憾的是，多样性不足的问题没有如 WGAN 的作者所说得到解决，网络简单之后，clip 导致网络的泛化能力大大减弱，因此生成了十分相似的图片，这种效果等同于找共性，把人脸数据集中的“大众脸”找了出来。从图 5-8 中的损失函数还发现，判别器不稳定，其原因应该是网络结构过于简单。

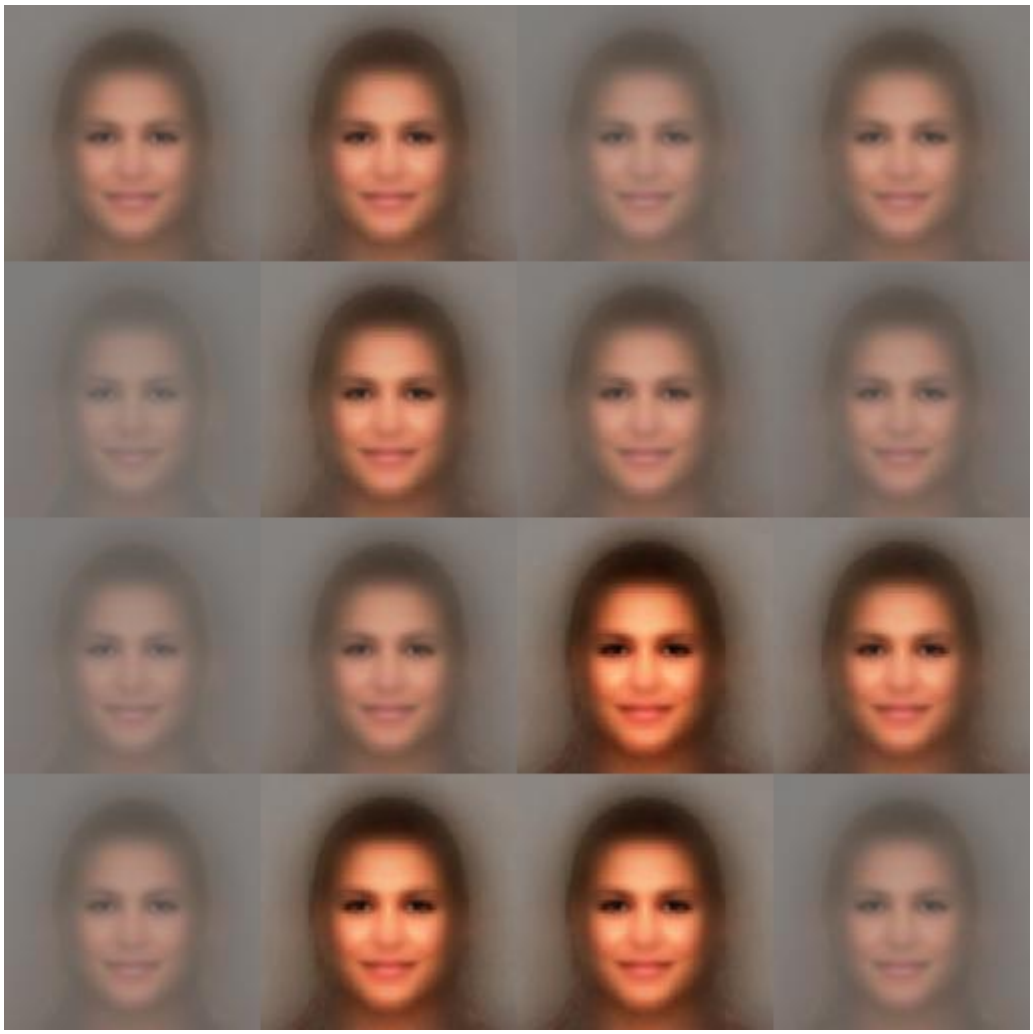

图 5-7 全连接的 WGAN 结果示例

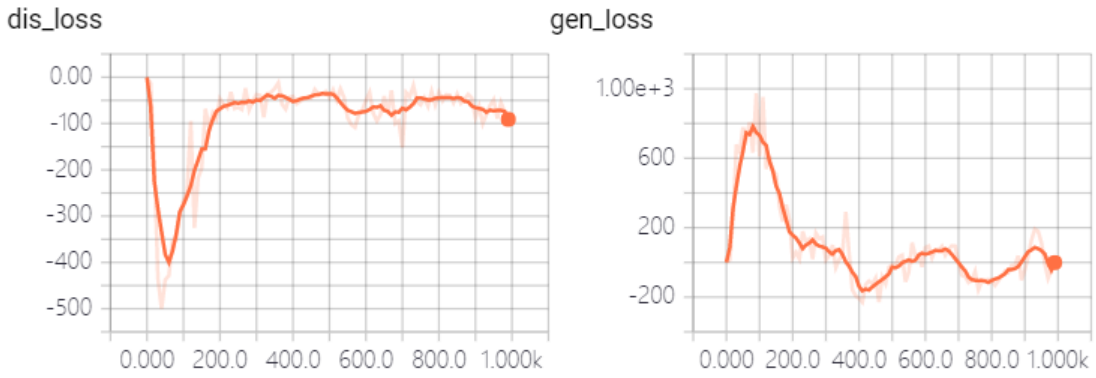

图 5-8 WGAN 损失函数变化

5.6 WGAN 的优化

全连接的网络虽然不复杂，但是有一个很严重的问题，简单的网络泛化能力不足，并且，图像有许多噪点，另外一个问题就是全连接网络空间开销巨大，对于最后的输出是一个(128,128,3)的图像来说，若是第一层全连接输出的是长 M 的一维向量，那么第二层全连接层的权重将有 49152M 个，这种空间的开销是巨

大的。实际上，对 5.5 中的实验，中间一层的神经元的数据若是再多一些，就会因为显卡显存不足，无法运行。反卷积和全连接一样，都具有扩充数据的功能，尽管反卷积可能带来数据的重复。

下面所做的是在生成器中增加一层卷积之后的实验，判别器的网络没有变化，生成器的网络为：全连接→relu 激活→全连接→tanh 激活→卷积→sigmoid 激活。

实验最终的效果如图 5-9，多样性稍有改善，从损失函数来看，网络还有继续训练的空间。

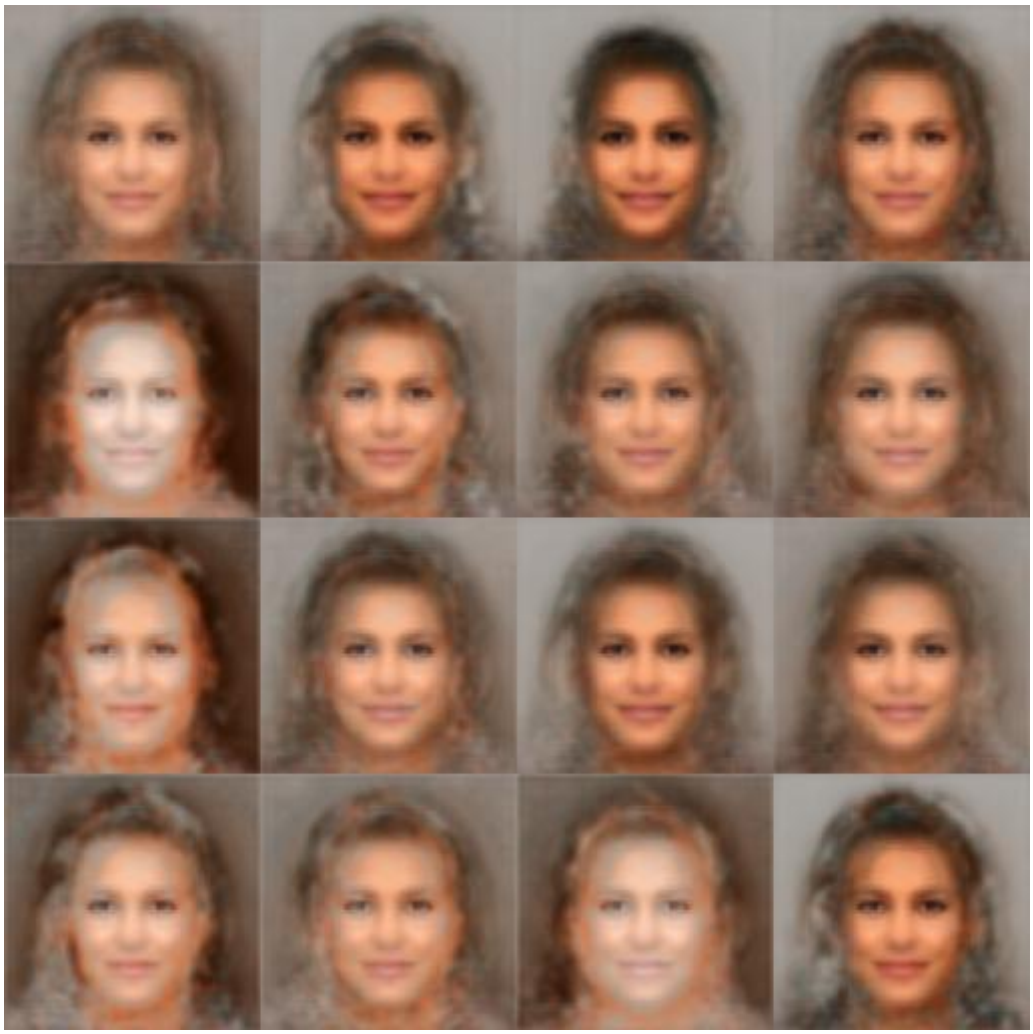

图 5-9 WGAN 优化后示例(1)

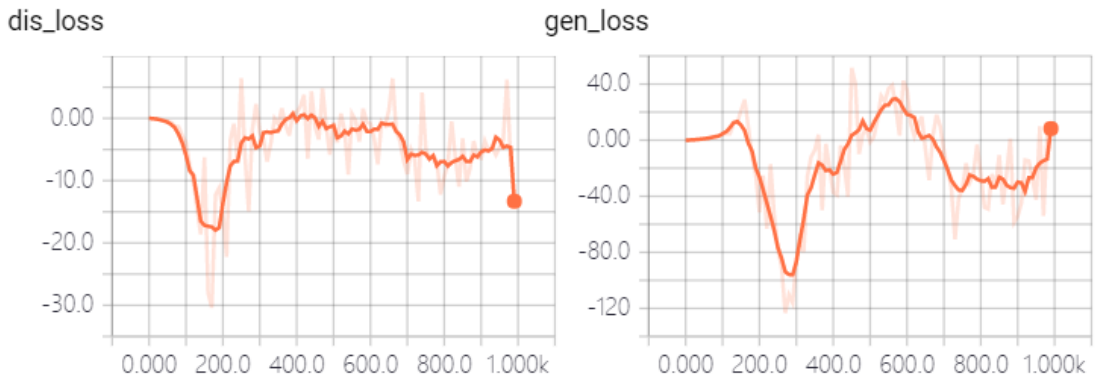

图 5-10 WGAN 优化后损失函数(1)

另一个实验不仅仅增加了一层卷积，还在生成器中增加了两层反卷积，没有使用批归一化，生成器的网络模型如下：

(1)噪声向量全连接到长 4096 的一维向量；

(2)上一层输出的一维向量重排成(32,32,4)的三维向量，做反卷积操作，卷积核大小为(5,5)，步长为 2，输出的数据规模为(64,64,16)；

(3)对上一层输出的数据做 tanh 激活；

(4)对上一层输出的数据做反卷积，卷积核大小为(5,5)，步长为 2，输出的数据规模为(128,128,8)；

(5)对上一层输出的数据做 tanh 激活；

(6)对上一层输出的数据做卷积，卷积核大小为(5,5)，步长为 1，个数为 3，输出的数据规模为(128,128,3)；

(7)对上一层得到的数据做 sigmoid 激活。

从图 5-11 和 5-12 来看，两层反卷积的效果不是很好，比较模糊，其中一个原因就是判别器没有跟着生成器做修改，较难捕捉生成器生成的特征。损失函数也不稳定。

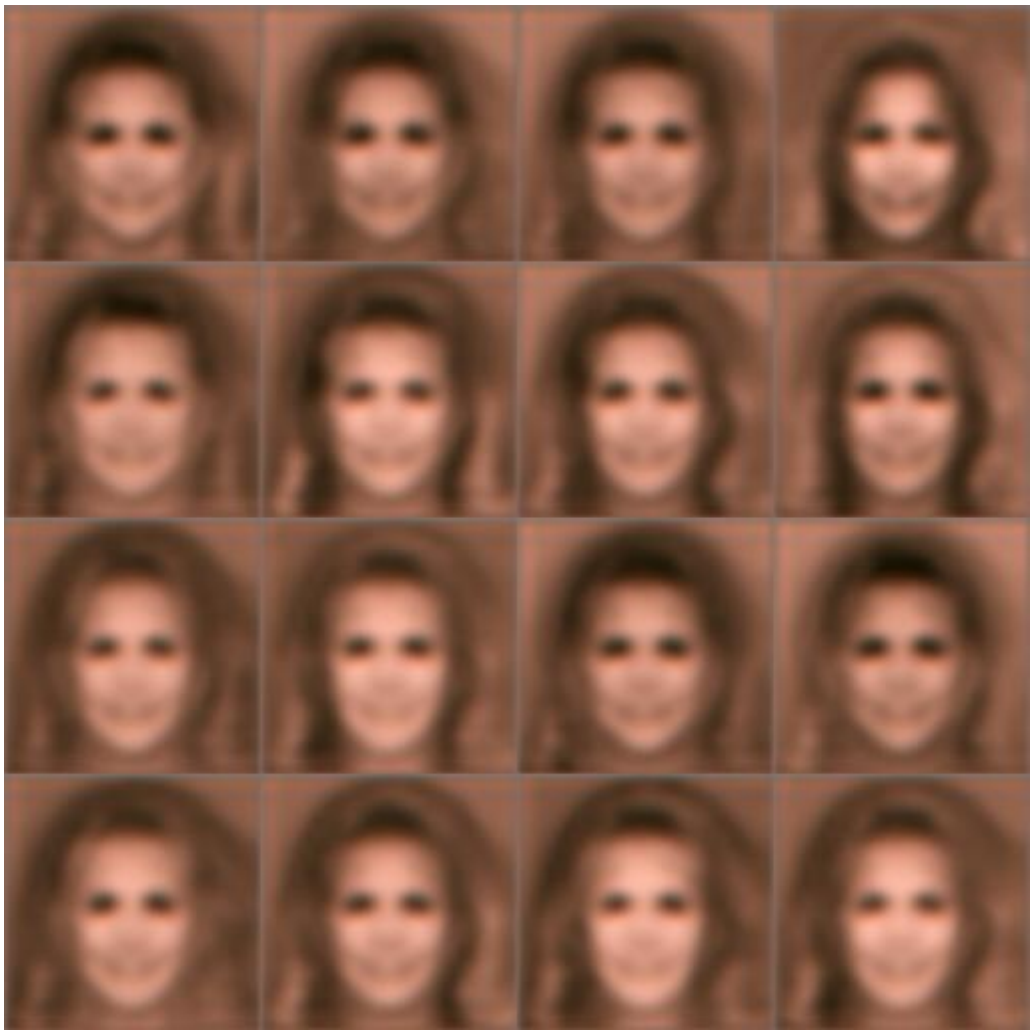

图 5-11 WGAN 优化后结果示意(2)

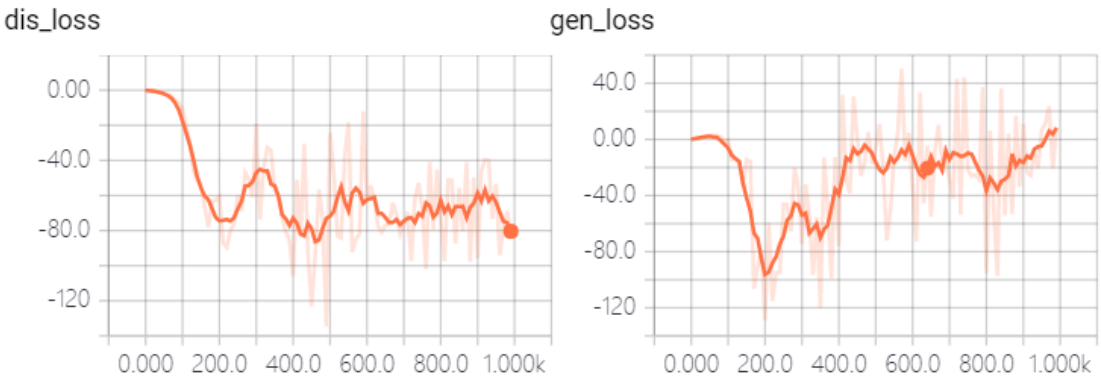

图 5-12 WGAN 优化后损失函数(2)

第6章 Improved Training of Wasserstein GANs

6.1 Martin Arjovsky 对 Wasserstein GAN 所做改进

在五月初的时候，发现了 Wasserstein GAN 的作者 Martin Arjovsky 对 Wasserstein GAN 的改进[4]。Martin 意识到了 Wasserstein GAN 存在训练困难、收敛速度慢等问题[15]，认为关键在于原设计中 Lipschitz 限制的施加方式不对，并在新论文中提出了解决方案。

在对 Lipschitz 施加限制时，原设计使用的是对判别器的网络参数进行 clip，但是这会导致权重的分布集中在最大限制和最小限制两个极端上，判别器趋近一个简单的映射，类似于二值网络，没有充分利用网络的泛化能力。同时，不同的网络 clip 的值也不同，找到这个合适的值是一个很耗时间的事情。clip 的值如果限制的稍小，每经过一层网络，梯度就会变小一点，多层之后就会指数衰减，反之梯度会指数爆炸。这也就是之前对已有的较好的网络施加 Wasserstein GAN 的技术效果并不明显的原因，网络层次深，clip 值不好确定。

6.2 实验效果对比

Martin Arjovsky 在新论文中提出 gradient penalty 方法，相应的网络模型简称为 WGAN-GP。由于时间原因，本人在 WGAN-GP 模型上的实验和学习并不充分，只是单纯的去做了一些验证性的实验。

相比较 Wasserstein GAN，WGAN-GP 模型只修改了 Lipschitz 限制施加的方式，具体的在程序上的改动有两处，一处是判别器损失函数增加了一个惩罚项，用来实现 Lipschitz 限制；另一处则是去除了 WGAN 中对参数 clip 的代码。第二处无须解释，删除代码即可，重点展示第一处程序的修改。

判别器对真实图像的判别结果为 real，对伪造图像的判别结果为 fake，向判别器中输入的数据是一个二维矩阵，每一张图片是一个一维向量，一共 batch_size 张图像。记 alpha 为 batch_size 个[0,1]之间的随机分布，将 batch_size 个真实图像和 batch_size 个伪造图像组合起来，第 i 对按照 alpha[i]:(1-alpha[i])的比例加起来，等同于造了一个介于真实和伪造之间的图像，真实的成分和伪造的成分比例由随机分布来决定，记为 interpolates。对 interpolates 经过判别器的判别结果求梯度，记为 gradients。求 gradients 的 2-范数记为 slopes。计算(slopes-1)的平方和，再乘以一个参数 λ 作为判别器

损失函数的第三项，加到判别器的损失函数中。

具体的程序实现见附 4。

以上则是对判别器损失函数的修改。还需要注意的是，由于对每个样本独立的施加梯度惩罚，不能使用批归一化这种会引入同一批中不同样本的相互依赖关系操作。批归一化可以被层归一化等其他方法代替，或者不使用。

以下是在验证码数据集上的实验。

1. DCGAN 应用 WGAN-GP 优化

第 860 次迭代、耗时 237 秒的结果如图 6-1，大部分验证码清晰可见，效果很好，但是和 4.4 中的优化结果在时间上处于劣势，两个程序所用的模型相同，唯一的不同点在于对 Lipschitz 施加限制的方式。从图 6-2 中可以看出，生成器尚有训练的空间，或许需要增加判别器的训练次数或者增加判别器的预训练。

图 6-1 WDCGAN-GP 结果示例

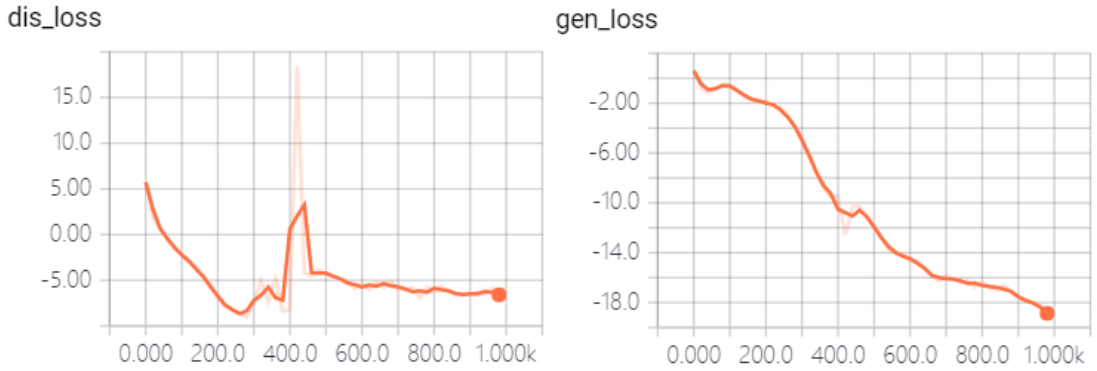

图 6-2 WDCGAN-GP 损失函数

2. WGAN 应用 WGAN-GP 优化

同样的方法施加到全连接的 WGAN 上，在 4500 次、101 秒之后，生成的图像几乎不再变化，如图 6-3 所示。并且从损失函数图 6-4 来看，梯度消失，损失函数趋于水平，整体来说，优化效果在于收敛速度的变快，但对于最终生成的效果，没有多少改变。

图 6-3 全连接 WGAN-GP 生成示例

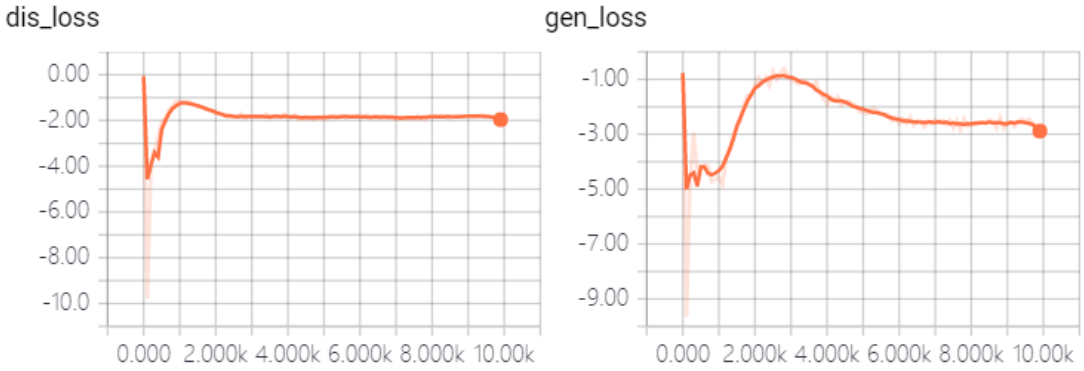

图 6-4 全连接 WGAN-GP 损失函数

6.3 全文展望

在前面的六章中，我们从神经网络入手，逐层递进，从基础的全链接网络、卷积网络，到入门级别的验证码识别，再到当下的热门技术生成对抗网络，关键部分在于对生成对抗网络的优化，不仅仅验证了著名的 Wasserstein GAN，同时也对 WGAN 在图像的生成方面做了针对性的改进。

作为一个机器学习领域的初学者，从毕业设计中学到了很多，然而这只是机

器学习的冰山一角，欲穷千里目，更上一层楼。除了本文涉及的神经网络的入门、GANs、Wasserstein GAN 和优化，还有很多内容值得进一步研究，比如对 WGAN-GP 的优化，简化网络模型。

神经网络囊括的内容有还有很多，随着高性能计算的发展以及更多网络模型的诞生，神经网络将会绽放更多的奇迹。

参考文献

［1］ Goodfellow I, Pouget-Abadie J, Mirza M, et al. *Generative adversarial nets*[C]//Advances in neural information processing systems. 2014: 2672-2680.

［2］ Arjovsky M, Bottou L. *Towards principled methods for training generative adversarial networks*[C]//NIPS 2016 Workshop on Adversarial Training. In review for ICLR. 2017, 2016.

［3］Arjovsky M, Chintala S, Bottou L. *Wasserstein gan*[J]. arXiv preprint arXiv:1701.07875, 2017.

［4］ Gulrajani I, Ahmed F, Arjovsky M, et al. *Improved Training of Wasserstein GANs*[J]. arXiv preprint arXiv:1704.00028, 2017.

［5］ LeCun Y, Cortes C, Burges C J C. MNIST handwritten digit database[J]. AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2010, 2.

［6］https://www.tensorflow.org/ last visit on 2017.5.19

［7］http://yann.lecun.com/exdb/mnist/ last visit on 2017.5.19

［8］http://caffe.berkeleyvision.org/ last visit on 2017.5.19

［9］http://wiki.jikexueyuan.com/project/tensorflow-zh/ last vist on 2017.5.19

［10］http://vis-www.cs.umass.edu/lfw/ last visit on 2017.5.19

［11］ http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html last visit on 2017.5.19

［12］LeCun Y, Boser B E, Denker J S, et al. *Handwritten digit recognition with a back-propagation network*[C]//Advances in neural information processing systems. 1990: 396-404.

［13］https://github.com/wiseodd/generative-models/tree/master/GAN last visit on 2017.5.19

［14］ https://zhuanlan.zhihu.com/p/25071913 last visit on 2017.5.19

［15］ https://www.zhihu.com/question/52602529/answer/158727900 last visit on 2017.5.19

附录

附 1 验证码 CNN 实现的网络结构程序

```
(01) x = tf.placeholder(tf.float32, [None, 400], "x")
     y = tf.placeholder(tf.float32, [None, 32], "y")
     h0 = tf.reshape(x, shape=[-1, 20, 20, 1], name="reshape")
(02) h1 = tf.nn.conv2d(h0, filter=W1, strides=[1, 1, 1, 1], padding="VALID") + B1
(03) h2 = tf.nn.ReLU(h1)
     h3 = tf.nn.max_pool(h2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="VALID")
     h4 = tf.nn.conv2d(h3, filter=W2, strides=[1, 1, 1, 1], padding="VALID") + B2
(04) h5 = tf.nn.ReLU(h4)
     h6 = tf.nn.max_pool(h5, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="VALID")
(05) h7 = tf.reshape(h6, shape=[-1, 2 * 2 * self.dim_2], name="reshape")
     h8 = tf.matmul(h7, W3) + B3
(06) h9 = tf.nn.ReLU(h8)
(07) h10 = tf.matmul(h9, W4) + B4
(08) y_ = tf.nn.softmax(h10)
(09) loss = -tf.reduce_sum(y * tf.log(y_))
(10) accuarcy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(y_, axis=1), tf.argmax(y,
     axis=1)), tf.float32))
```

附 2 验证码 DCGAN 实现的网络结构和损失函数

判别器主要程序：

```
(01) dis_h0 = tf.nn.conv2d(image, self.dis_W1, strides=[1, 2, 2, 1], padding="SAME")
(02) dis_h0 = lReLU(dis_h0)
(03) dis_h1 = tf.nn.conv2d(dis_h0, self.dis_W2, strides=[1, 2, 2, 1],
     padding="SAME")
(04) dis_h1 = lReLU(dis_h1)
(05) dis_h1 = tf.reshape(dis_h1, [-1, self.dim_dis_2 * (self.shape[0] // 4) *
     (self.shape[1] // 4)])
     dis_h2 = tf.matmul(dis_h1, self.dis_W3)
(06) dis_h2 = batchnormalize(dis_h2)
(07) dis_h2 = lReLU(dis_h2)
(08) dis_h3 = tf.matmul(dis_h2, self.dis_W4)
(09) dis_h3 = tf.nn.sigmoid(dis_h3)
```

生成器主要程序：

```
(01) gen_h0 = tf.matmul(z, self.gen_W1)
(02) gen_h0 = batchnormalize(gen_h0)
(03) gen_h0 = tf.nn.ReLU(gen_h0)
(04) gen_h1 = tf.matmul(gen_h0, self.gen_W2)
(05) gen_h1 = batchnormalize(gen_h1)
(06) gen_h1 = tf.nn.ReLU(gen_h1)
```

```
(07) gen_h1 = tf.reshape(gen_h1, [-1, self.shape[0] // 4, self.shape[1] // 4,
     self.dim_gen_2])
     gen_h2 = tf.nn.conv2d_transpose(gen_h1, self.gen_W3, [batch_size, self.shape[0]
     // 2, self.shape[1] // 2, self.dim_gen_3], strides=[1, 2, 2, 1])
(08) gen_h2 = batchnormalize(gen_h2)
(09) gen_h2 = tf.nn.ReLU(gen_h2)
(10) gen_h3 = tf.nn.conv2d_transpose(gen_h2, self.gen_W4, [batch_size,
     self.shape[0], self.shape[1], self.channel], strides=[1, 2, 2, 1])
(11) gen_h3 = tf.nn.sigmoid(gen_h3)
```

损失函数的计算：

```
real_image = tf.reshape(x, [-1, self.shape[0], self.shape[1], self.channel])
fake_image = self.generate(batch_size, z)
real = self.discriminate(real_image)
fake = self.discriminate(fake_image)
dis_loss_real =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.clip_by_value(
real, 1e-7, 1. - 1e-7), labels=tf.ones_like(real)))
dis_loss_fake =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.clip_by_value(
fake, 1e-7, 1. - 1e-7), labels=tf.zeros_like(fake)))
dis_loss = dis_loss_real + dis_loss_fake
gen_loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.clip_by_value(
fake, 1e-7, 1. - 1e-7), labels=tf.ones_like(fake)))
```

附 3 二次封装 Tensorflow 函数说明

```
def fully_connected(value, output_shape, name="fully_connected")
```

全连接层，输入分别为：输入数据、输出的一维向量长度、网络层名称，返回值为全连接之后的输出数据

```
def lReLU(x, leak=0.2, name="lReLU")
```

leaky ReLU 激活层，x 为输入数据，leak 为负数放缩比例，name 为网络层名称，返回值为激活后数据

```
def ReLU(value, name='ReLU')
```

ReLU 激活层，value 为输入数据，name 为网络层名称，返回值为激活后数据

```
def deconv2d(value, output_shape, k_h=5, k_w=5, strides=(1, 2, 2, 1),
name='deconv2d')
```

二维反卷积层，输入为：输入数据、输出数据规模、卷积核垂直大小、卷积核水平大小、卷积步长、网络层名称，返回值为反卷积后数据

```
def conv2d(value, output_dim, k_h=5, k_w=5, strides=(1, 2, 2, 1), name='conv2d')
```

二维卷积层，输入为：输入数据、卷积核个数（输出数据最后一维大小）、卷积核垂直大小、卷积核水平大小、卷积步长、网络层名称，返回值为卷积后的数据

```
def batch_norm(value, is_train=True, name='batch_norm')
```

批归一化层，输入为：输入数据、是否是训练过程、网络层名称，输出为归一化后结果

附 4 WGAN-GP 施加限制的方法

```
(1) alpha = tf.random_uniform(shape=[batch_size, 1, 1, 1], minval=0., maxval=1.)
(2) interpolates = alpha * real_image + (1. - alpha) * fake_image
(3) gradients = tf.gradients(self.discriminate(interpolates), [interpolates])[0]
(4) slopes = tf.sqrt(tf.reduce_sum(tf.square(gradients), 1))
(5) gradient_penalty = tf.reduce_mean((slopes - 1.) ** 2)
(6) LAMBDA = 10
(7) dis_loss += LAMBDA * gradient_penalty
```