摘要

生成对抗网络（Generative Adversarial Nets）毫无疑问是当下热门的人工智能技术之一。从2014年Ian Goodfellow发表GAN论文至今，生成对抗网络已经有了数百种模型变种，并且数量仍在上升。可以说生成对抗网络在图像生成模型的质量上实现了飞跃性的提升，它抛弃了传统的单一神经网络生成图片的模式，采用两个神经网络----生成网络及判别网络相互对抗的方式提高生成图片的质量。

本论文采用全连接前馈神经网络结构设计了生成网络的模型，同时将该结构迁移至判别网络，并系统地测试了不同的输入，训练误差及隐藏层个数对上述两个模型的影响。在实验中发现生成网络具有遗忘特性，采用标准BP将只能记录最后一次训练所学到的知识；在对判别网络的训练中发现了全新的训练思路：标准BP的累积化应用。最后将两个网络相结合进行了图像的生成实验，并对实验结果进行了深入细致的分析。

**关键词：**生成对抗网络，GAN，生成网络，判别网络，BP

Abstract

Generative Adversarial Nets is undoubtedly one of the hottest artificial intelligence technologies of the moment. Since the publication of the GAN paper by Ian Goodfellow in 2014, there have been hundreds of model variants in the generation of confrontation networks, and the number is still rising. It can be said that the generation of the confrontation network has achieved a dramatic improvement in the quality of the image generation model. It has abandoned the traditional single neural network to generate pictures, using two neural networks---generating the network and discriminating the way the network confronts each other. Improve the quality of generated images.

In this thesis, the model of generating network is designed by using the fully connected feedforward neural network structure. At the same time, the structure is migrated to the discriminant network, and the effects of different inputs, training errors and hidden layers on the above two models are systematically tested. In the experiment, it is found that the generated network has the forgetting property. The standard BP can only record the knowledge learned in the last training; in the training of the discriminant network, a new training idea is found: the accumulative application of the standard BP. Finally, the two networks were combined to perform image generation experiments, and the experimental results were analyzed in detail.

**Key words:** Generate confrontation network, GAN, generate network, discriminate network, BP

目录

[第一章 绪论 1](#_Toc12959207)

[1.1人工智能的历史与发展 1](#_Toc12959208)

[1.2机器学习与深度学习 2](#_Toc12959209)

[1.3生成对抗网络的诞生 3](#_Toc12959210)

[第二章 传统神经网络与生成对抗网络的异同 4](#_Toc12959211)

[2.1网络结构上的异同 4](#_Toc12959212)

[2.2误差反传播算法上的异同 8](#_Toc12959213)

[第三章 生成对抗网络的拆分与实现 14](#_Toc12959214)

[3.1生成网络设计 14](#_Toc12959215)

[3.2将生成网络结构应用于判别网络 16](#_Toc12959216)

[3.3生成对抗网络的实现 19](#_Toc12959217)

[第四章 总结 24](#_Toc12959218)

[4.1梯度消失及解决思路 24](#_Toc12959219)

[4.2效果不佳及解决思路 25](#_Toc12959220)

[参考文献 26](#_Toc12959221)

[致谢 27](#_Toc12959222)

[附录A 生成网络 28](#_Toc12959223)

[附录B 判别网络 31](#_Toc12959224)

[附录C 生成对抗网络中的判别网络 35](#_Toc12959225)

[声明 38](#_Toc12959226)

# 第一章 绪论

## 1.1人工智能的历史与发展

早在古希腊神话中就已经出现了机器人。古希腊诗人荷马在公元前8世纪曾描述过一位名叫Hephaestus的“锻造之神”，他设计并制造了一组女性黄金机器人，这些机器人能够完成很多高难度的事情，并帮助他在铁匠铺做事。这大概就是能够追溯到的最早的有关人工智能的传说。

当人类第一次对计算机进行编程时,他们已经相信它们可以更智能。如今，伴随着四场著名的人机大战，人们已不再对人工智能陌生。

人工智能的首场胜利要追溯到1997年，那时棋类竞技仍是世界人民的头号消遣活动，卡斯帕罗夫就是当时国际象棋首屈一指的佼佼者。尽管如此，他仍然未能战胜“深蓝”，即IBM公司的超级计算机。

时隔9年，计算机逐渐普及，运算能力也随着时代大幅提升。在运算复杂度上与国际象棋相同难度的中国象棋也迎来了人工智能的挑战。毕竟过了一个十年，浪潮天梭强大的运算能力，即便同时迎战五位中国的顶尖高手仍然取得了胜利。

第三场比赛在国内并不是很知名，因为这场比赛的主场在2011年美国的问答比赛。对于人类语言的理解一直是人们认为机器力所不能的领域，然而在《危险边缘》的答题比拼中，新一代的AI击败对手毫不费力，这个名为“沃森”的AI不得不让人们惊叹如今的人工智能已经能够具备如此不得了的自然语言理解能力。

最后一场也是近几年最火最具话题性的比赛。围棋，一直有最高复杂度的棋类游戏的美誉，也是少数不可能靠穷举进行推测的棋类游戏。然而时代在进步，算法在精进，到了2016年，谷歌公司已经开发出了无需穷举且能自我学习的人工智能----AlphaGo。该人工智能的成功不仅仅是让李世石1：4败北，更引发了一波现象级的人工智能热潮，自此开始，人工智能变得家喻户晓。并且在次年，谷歌公司开发的进阶版下棋AI让柯洁泪洒战场，浑身冷汗。而柯洁，是围棋领域最后一个BOSS。

四场人机大赛，标志着人类最高智力的神话被终结，我们正慢慢进入机械智能的时代。

## 1.2机器学习与深度学习

2012年以后，随着信息爆炸带来的数据量猛增、计算机算力的高速提升和深度学习的出现以及应用，人工智能的研究领域不断扩展，迎来大爆发。除了传统的专家系统、机器学习等，进化计算、模糊逻辑、计算机视觉、自然语言处理、推荐系统也接二连三有了里程碑式的成果[1]。

机器学习属于人工智能的分支之一，且处于核心地位。顾名思义，机器学习的研究旨在让计算机学会学习，能够模拟人类的学习行为，建立学习能力，实现识别和判断。机器学习使用算法来解析海量数据，从中找出规律，并完成学习，用学习出来的思维模型对真实事件做出决策和预测。这种方式也被称为“训练”。深度学习是机器学习的一种实现技术，在2006年被Hinton等人首次提出。深度学习遵循仿生学，源自于神经元以及神经网络的研究，能够 模仿人类神经网络传输和接收信号的方式，进而达到学习人类的思维方式的目的。

简而言之，机器学习是一种实现人工智能的方法，深度学习是一种实现机器学习的技术，而本文的主角生成对抗网络[2]则是深度学习中的一种分类。

神经网络是一种实现机器学习的技术，旨在模拟人脑神经网络的运作机制。1943年，抽象的神经元模型被首次提出。1949年心理学家Hebb提出了“学习率”这一概念，即信息在人脑神经细胞的突触上传递时，强度是可以变化的[3]。于是研究人员们开始使用调整权值的方法进化机器学习算法。1958年，计算科学家Rosenblatt提出了由两层神经元组成的单层神经网络，可以完成线性分类任务[4]。

1986年，BP算法的提出解决了两层神经网络所需要的复杂计算量问题，这个算法在两层神经网络（输入层和输出层）中增加了一个中间层[5]。但尽管使用了BP算法，一次神经网络的训练仍然耗时太久，局部最优解问题作为困扰训练优化的一大问题使得神经网络的优化较为困难[6]。

2006年，Hinton在《Science》和相关期刊上发表了论文，首次提出了“深度学习”的概念，并增加了两个优化技术----“预训练”和“微调”。这两个技术的运用可以让神经网络的权值找到一个接近最优解的值，并使得对整个网络进行优化训练的学习时间大幅度减少。

深度学习实际上指的是深度神经网络学习，普通神经网络由于训练代价较高，一般只有3到4层，而深度神经网络由于采用了特殊的训练方法加上一些技术算法，可以达到8到10层[7]。深度神经网络能够捕捉到数据中的深层联系，从而能够得到更精准的模型，而这些联系不容易被普通的机器学习方法所发觉[8]。

## 1.3生成对抗网络的诞生

在机器学习到深度学习的过程中，计算机不断地向人类靠拢，模仿思想,感受世界,理解人类语言。然而人类最美妙的地方在于创造能力，因而越来越多的研究者将自己的研究方向从机器模仿，转向机器创造。

在使用传统的神经网络生成图片的过程中，尽管研究者尽可能通过模拟人脑的思维方式生成图片，但效果依旧十分不理想。生成对抗网络正是在这种情况下应运而生。

生成对抗网络的诞生源于Ian Goodfellow一个天才的想法。Ian在蒙特利尔大学进行生成模型研究的时候遭遇了上述的图片质量不理想的问题，而更大的问题在于如果要对当前的模型进行优化，需要大量的数据集，并且最终效果依然是个未知数。

为了解决这一问题，Ian对传统神经网络的使用方式产生了怀疑，并最终提出了一个优秀的解决办法：如果不是使用一个神经网络，而是使用两个会怎么样。在Ian天才的构思中，两个神经网络互为敌对方相互对抗，如同人类长久以来的相互竞技才会有更大的进步一样，在最初的模型设计出来后，就取得了非常理想的效果。

生成对抗网络这种全新的技术为机器学习带来了突破，并逐渐成为研究热点，衍生出越来越多有关生成对抗网络的模型。

# 第二章 传统神经网络与生成对抗网络的异同

## 2.1网络结构上的异同

人类建立出神经元的数学模型这一历史壮举发生在1940年。完成这一壮举的人名叫沃伦·麦卡洛克以及沃尔特·皮茨。

他们创建的神经元结构模型见下图：

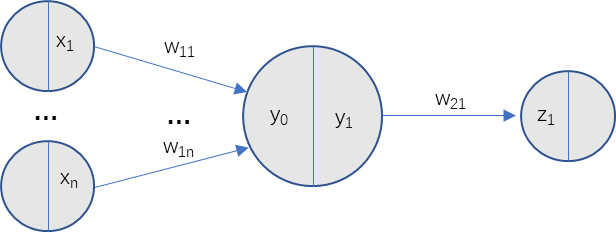


图2-1 M-P神经元模型

该神经元数学模型正是沿用至今的神经网络基本结构。

在此模型中,y神经元接收通过加权连接传输的所有x神经元的所有信号,并将接收的输入值与阈值（）比较后进行“激活函数”处理,随后产生输出[9]。

其数学表达为：

1. 接收n个前一层神经元传递来的信号总和，并与阈值（）作比较：

（2-1）

2.通过“激活函数”处理产生神经元的输出：

（2-2）

在生物神经网络中，一个神经元只有两种电位，“兴奋”或者“抑制”。

在两种状态下神经元存在不同的行为策略：

1.“兴奋”：

神经元仅在受到刺激或接收到上一个神经元的信号时触发该状态，在该状态下，神经元向向下一个神经元传送信号。

2.“抑制”：

神经元在该状态下不会活动，即停止神经元之间信号的传递。

在上述状态下传送的信号为化学物质，该物质能够改变神经元内外的电荷平衡，导致产生电流，即信号。

基于神经元的这种特性，在神经元模型中我们期望能够得到类似的函数以模拟神经元的激活过程，这种函数称为阶跃函数（见下图）：

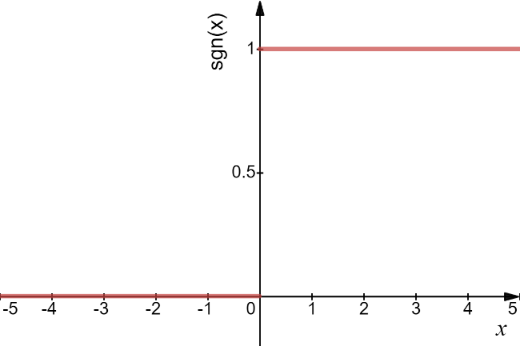


图2-2 阶跃函数

其表达式为：

（2-3）

阶跃函数原本非常理想。但是,阶跃函数存在以下问题：

1. 不连续
2. 不平滑

因此实际中使用Sigmod函数代替：

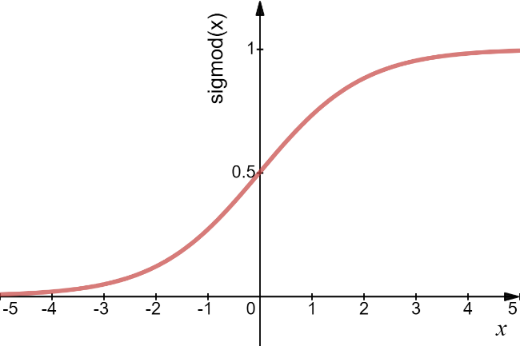


图2-3 Sigmod函数

其表达式为：

（2-4）

Sigmod函数解决了阶跃函数的诸多缺点，且能够将输入值压缩进0到1之间，因此有时也将其称为“挤压函数”。

神经网络是通过连接特定层次结构中的多个这样的神经元而获得的。

前馈神经网络是最简单的神经网络,其中神经元按层次结构排列,以下图为例，每个神经元仅连接到左边的神经元,接收左边的输出,输出到右边,没有跨层反馈。

而单隐层前馈神经网络指的是在输入层与输出层之间增加一个隐层，因而整个神经网络有三层。以每层均含有两个神经元的神经网络为例，其结构如图：

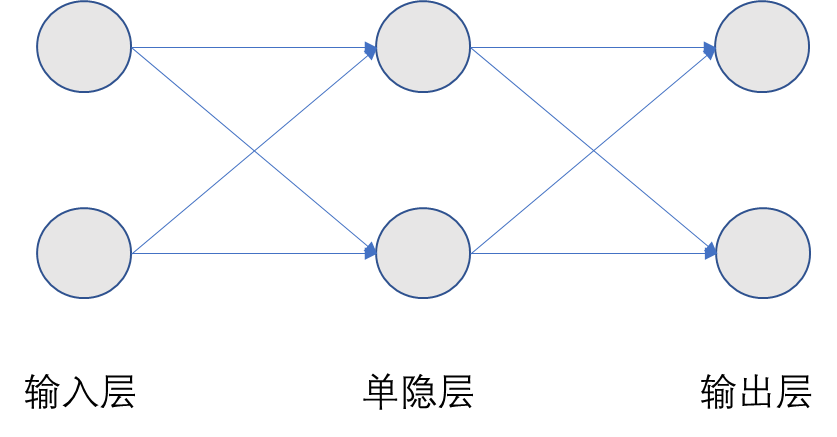


图2-4 单隐层前馈神经网络

上图中每一个结点均与下一层的结点有权值的连接，这种结构叫做全连接，因而上图中的神经网络亦可称为单隐层全连接前馈神经网络。

生成对抗网络与传统神经网络最大的差别在于传统神经网络的基本单元是神经元，而生成对抗网络的基本结构是神经网络。传统神经网络通过神经元的组合形成网络，生成对抗网络通过使用两个不同的神经网络相互对抗形成一个网络集合。在这个过程中，每个网络负责不同的职责：

生成网络：

生成网络是机器学习史上处于重要位置的构建模型。

如果有很多数据(如图像、音频和文本),则可以通过生成模型来模拟此高阶数据的分布,对于多数情况来说,这些数据很有价值。在数据量很少的情况下,构建模型有助于生成数据并增加数据量,并且可以使用半监督学习来提高学习效率。语言模型是生成网络应用最广泛的示例之一,通过合理的建模,语言模型不仅有助于生成语言流畅的句子,而且可以在机器领域具有广泛的辅助应用。翻译和聊天对话。

在本设计中生成网络是整个生成对抗网络的前半部分，用于将随即噪音通过神经网络生成数据。在模型的意义上，具有制造者的角色身份。单独使用时可以通过固定目标随机输入的方式进行训练。

判别网络：

判别网络是整个生成对抗网络的后半部分，用于将输入的数据进行分类。在模型的意义上，具有识别者的角色身份。单独使用时需用对应的输入及其分类对该网络进行训练，通常使用带有分类标签的输入数据集进行一个判别网络权值的更新。判别网络作为单网络使用的情况非常广泛，比较有名的实用事例就是mnist手写数据集的识别[10]。

生成网络与判别网络的对抗：

将两个神经网络结合使用一直被认为是Ian的天才之举。两个网络互相对抗就是制造者与识别者之间的对抗，因此首先需要让识别者具备一定的识别能力，这就是判别网络的预训练。

在一次完整的对抗过程中，生成网络将随机噪声映射成一次输出，该输出交由判别网络进行分类，一般来说经过训练的判别网络能够将生成网络的输出映射到一个“真”或者“假”的分类。而想要让生成网络给出什么样的最终结果取决于判别网络将什么样的误差进行回传。

通常在实际使用中我们会将类别为“真”的结果定义为“1”，类别为“假”的结果定义为“0”。那么想要生成网络产生类别为“真”的类，就将目标值设定为“1”并将误差进行回传；想要生成网络产生类别为“假”的类，就将目标值设定为“0”并将误差进行回传；然而生成对抗网络最迷人的地方在于它能够混淆判别网络的中间类别，这个时候理想的输出就是判别网络无法归类的“”。

在数学上的表达，生成对抗网络需要优化的目标函数如下：

（2-5）

其中和分别表示在数据空间中定义的真实数据概率分布和潜在空间上定义的潜在变量概率分布，而是二元交叉熵函数，通常用于二元分类问题。

由于想要对真实或虚假样本进行分类，因此是分类问题目标函数的自然选择。 从的角度来看，如果样本来自真实数据，将最大化其输出；而如果样本来自，将最小化其输出。同时，想要欺骗，因此当虚假样本呈现给时，它试图最大化的输出。通过对求导，可计算出最优的鉴别器:

（2-6）

把这个最优鉴别器带入前面的目标函数，则等式变为和之间的Jensen Shannon Divergence（JSD）。 因此，最小化JSD（|| ）的最优生成器是数据分布，并且将最优生成器代入最优鉴别器，D变为，意思是二者难以区分。

## 2.2误差反传播算法上的异同

误差反传播[11]（即BP）如今在该领域无人不知。这是一种相当成功的神经网络学习算法。事实上大多数神经网络都是使用BP算法训练的。BP算法不仅可用于前馈神经网络的训练,还可用于训练其他神经网络,但术语“BP网络”通常指使用该算法训练的多层前馈神经网络[12]。

一次完整的学习过程始于对输入值的正向计算，当计算至最后一层时，输出层将会给出本次计算的结果，其结果必然会与目标值存在误差，为此,我们需根据最后一层神经元获取每个阶段的总误差：

（2-7）

该函数名为均方误差函数。其中为总误差，target为训练目标，output为最后一层的输出结果。在该公式中所引入的系数 只是为了后续求导的时候使得结果简洁。

BP算法根据梯度下降策略调整目标的负梯度方向参数。对于权值w，总误差，给定学习率η，有：

（2-8）

我们使用反向传播的目标是更新网络中的每个权值，以便它们使实际输出更接近目标输出，从而最小化每个输出神经元和整个网络的误差[13]。

由于BP算法最终要更新权值，因此，首先要知道总误差对每层权值的偏导。在这个过程中有两种不同的情况需要分别作讨论：

1. 总误差对输出层与前一隐层之间的权值更新。
2. 总误差对隐层与前一层之间的权值更新。

两种情况均需要计算总误差对权值的偏导，对于最后一层，也就是第一种情况，是最基本的过程：

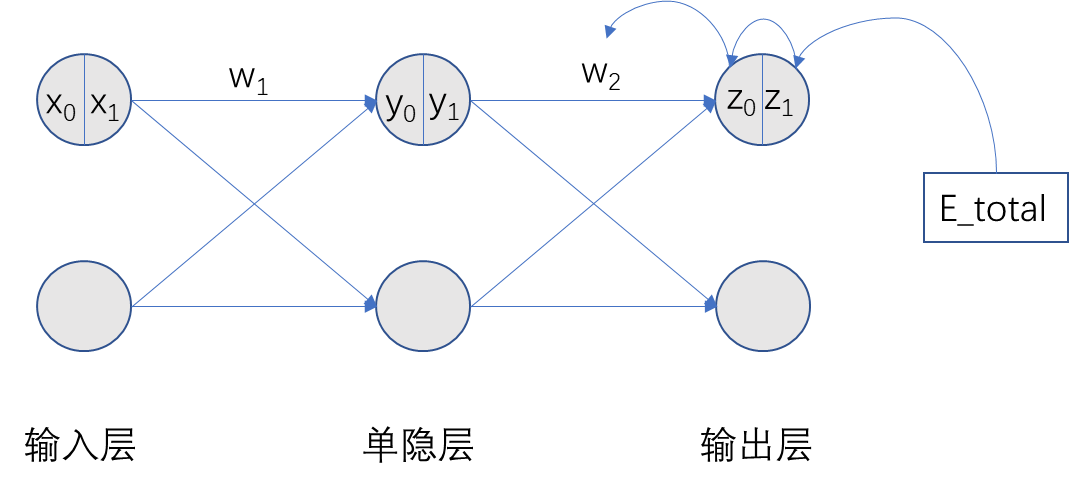


图2-5 总误差对最后一层权值求偏导

总误差对权值w2的偏导可以通过“链式法则”拆分成三步：

（2-9）

其中第一项为总误差对神经元输出的偏导，单独计算可得：

（2-10）

第二项为激活函数的导数，Sigmod函数的导数有一个很好的性质就在于此：

（2-11）

第三项的结果显然就是上一层的输出：

（2-12）

至此第一种情况的求偏导过程分析完毕，讨论第二种，当总误差要跨层回传的情况：

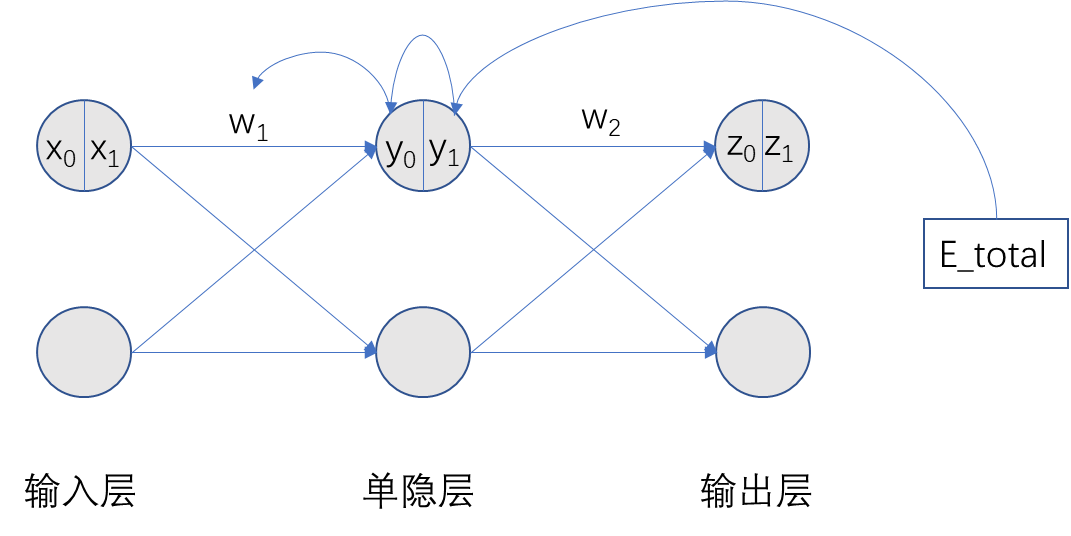


图2-6 总误差跨层对权值求偏导

在这种情况下，总误差需要跨越输出层对隐层神经元的输出进行求偏导：

（2-13）

而第一项可以通过“链式法则”进一步拆分：

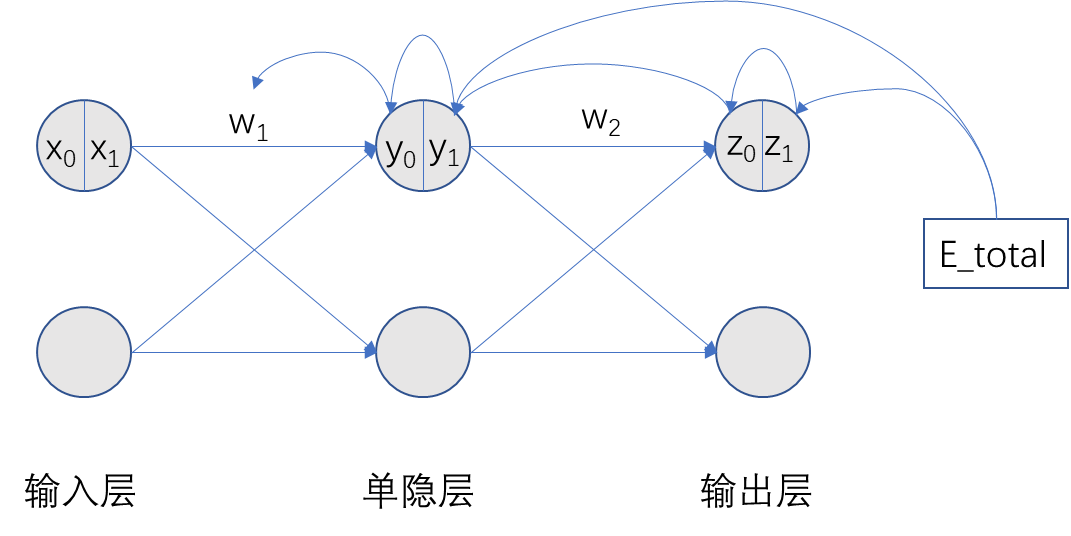


图2-7 总误差跨层对权值求偏导的拆解

因此，整个过程变为：

（2-14）

我们注意到无论是第一种情况还是第二种情况都会涉及总误差对神经元输出的偏导以及该神经元激活函数的导数，因此将它们记为δ，并重写上述运算：

（2-15）

（2-16）

由此我们发现前一层反传计算需要用到的误差对输出的偏导，就是后一层的δ乘以它们之间权值。

值得一提的是，在神经网络的计算中多数情况下是使用矩阵进行运算，因此反传播计算中要特别注意矩阵运算的逆运算。

这一点在一维神经元排列中体现为权值矩阵的转置，在二维神经元排列中体现为权值矩阵在其平面内旋转180度。

根据δ、权值及激活函数导数，逐层计算出所需的误差，最终得到更新权值所需要的变化量，并据此更新权值：

（2-17）

（2-18）

然而在生成对抗网络中，误差反传播存在一种特殊的情况。

在判别网络将误差反传给生成网络的过程中，判别网络固定权值不动，修改输入图，因此并不是将误差反传给权值矩阵，而是修改每层的神经元，最终修改输入图，而这张输入图就是生成网络下一次训练的目标图。

以之前的图例为例：

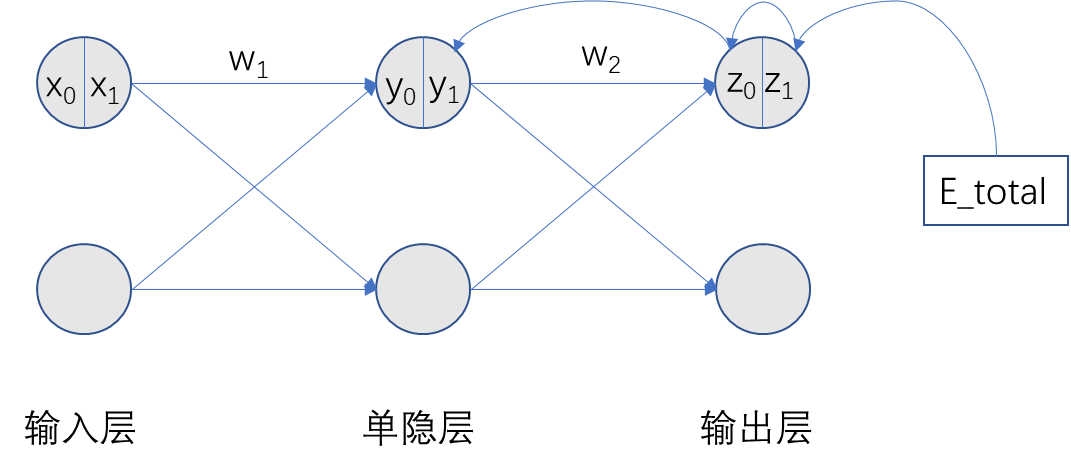


图2-8 生成对抗网络中判别网络的误差反传

最后一层的误差不再对权进行求偏导，而是直接对上一层的输出求偏导，因此公式变为：

（2-19）

而在多层反传的过程中，计算前一层的误差时，与训练判别网络时无异，并不改变δ：

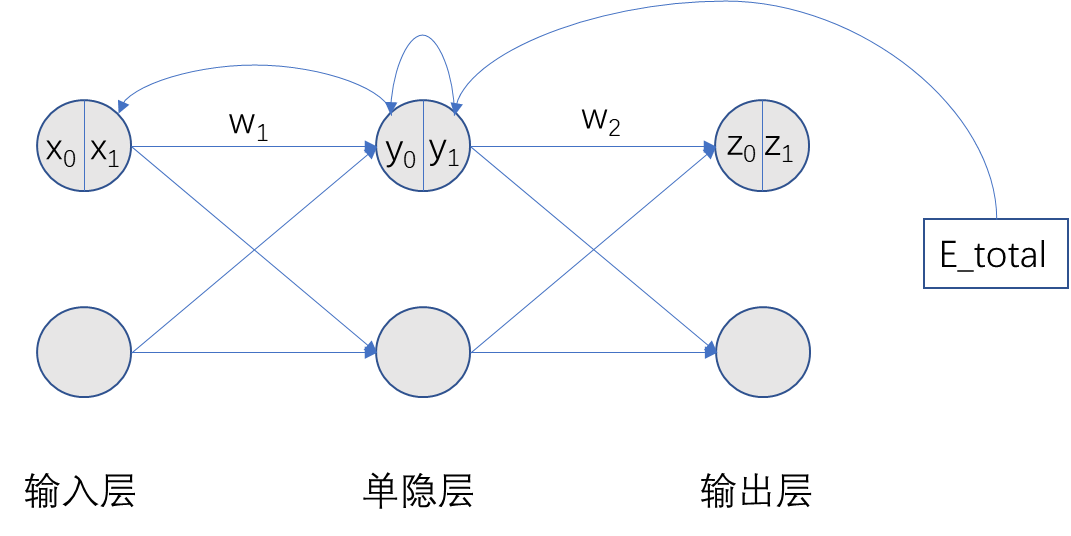


图2-9 生成对抗网络中判别网络的多层误差反传

# 第三章 生成对抗网络的拆分与实现

## 3.1生成网络设计

Hornik在1989年的证明，只需一个包含足够多神经元的隐层，多层前馈网络就能以任意精度逼近任意复杂度的连续函数[14]。事实上增加神经元的个数，就意味着增多了权值，基于这个理念，在设计生成网络的结构时采用曾被弃用的全连接结构[15]。在早期提出全连接结构的年代，由于计算机计算能力不足，内存不够大等时代因素致使全连接神经网络的计算成为妄想。如今计算机计算能力大幅提升，内存空间成倍增长，计算机算力已经不再是阻碍，所以，鉴于等神经元全连接神经网络方便计算推导以及权值足够的特性，将其作为基本结构应用至生成网络中。

在本设计中为了在多项式时间内能够迅速得到程序运行结果，并根据结果对生成网络结构进行调整，以及检测生成网络的计算是否出错，因而采取最简化的模型进行实验。

本设计在代码实现上未采用任何神经网络框架，纯手工搭建网络结构。考虑到矩阵运算的便利性，以及各层间运算方便推导，最终结构定稿为各层等神经元数量的单隐层前馈神经网络，并且可根据需求任意增加隐层数量。

以3×3的输入为例，在生成网络中，输入神经元的结构本应为：

（3-1）

为了计算方便，将每层的神经元通过如下1×9的矩阵结构进行存储：

（3-2）

每层权值记录为如下9×9的矩阵：

（3-3）

如此存储的优势在于方便计算，当1×9的矩阵与9×9的矩阵进行点积运算时，计算结果将依然是1×9的矩阵，这恰好对应了下一层神经元的结构。

由于全连接的特殊结构，因此无须在意原图上下层间可能存在的关系被计算忽视。

最后根据输出神经元的结果，将各个神经元的值对应至原3×3的图片相应的位置上，将其还原成图片。

另外，在生成网络的训练思路上，考虑到误差反传播存在两种思路：

1. 标准BP，将训练集的每个样本进行单独误差反传播训练。
2. 累积BP，将训练集所有样本的误差取平均值后，进行对整个训练集权值的平均值进行更新。

在本设计中采用第一种误差反传播思路----标准BP。当然，在训练时会多次重复总训练集以期望降低误差。

在训练集方面，采用最简单的两类输入：“横”和“竖”。将两个及两个以上垂直排列的图案称为“竖”；将两个及两个以上水平排列的图案称为“横”。通过枚举所有情况制作出样本集。

在实验中分别采取枚举图片像素大小为3×3及5×5的所有“横”和“竖”的样本进行训练，其中训练变量为：误差精度、每层的神经元数量、隐层数。

训练完成后，将训练好的权值保存并并应用在用于测试的同结构神经网络。

在多次训练后得到的实验结果如下：

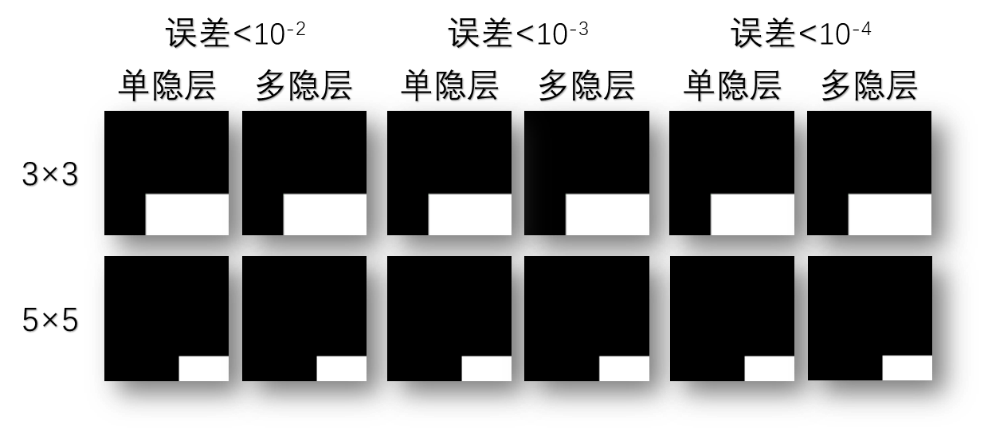


图3-1 全连接前馈神经网络训练结果输出图

该实验结果生成的图片分别为3×3及5×5的训练集最后一张图片，通过调节训练集范围定向调查原因后发现，本设计的生成网络输出结果只与训练集最后一张训练图有关，无论改变训练次数或是隐层数量或是总训练集误差级数都不能对该结果造成影响；当且仅当调整训练集最后一张训练图时，生成结果发生显著变化，变化为生成器最终训练结果为训练集最后一张图的理想输出。

由于在进行神经网络误差反传播的策略时采用标准BP的思路，即按训练集顺序，分别将单个训练目标的总误差训练至误差要求以下后在进行下一个目标的训练，据此判断，当神经网络误差反传播采用标准BP的思路时，生成网络将训练收敛为生成最后一次训练的理想输出的生成器。我将这种现象称为生成网络的遗忘性。所谓遗忘性，是指生成网络将遗忘最后一次训练之前所学到的所有知识，并只能记住最后一次训练所学到的知识，并将其储存于权值中。由此可知，训练好的生成网络只能将各种随机输入映射到唯一的结果。

根据实验结果进行预测，当神经网络误差反传播采用累积BP的思路时，生成网络应该会学习到与总训练集有关的综合特征。所谓累积BP，是将整个训练集跑完一遍后求出总误差的平均值，并回传给训练集权值的平均值。由于误差与权值直接与总训练集相关联，因此，根据生成网络的遗忘特性推断，累计BP能够学习到综合特征并生成与训练集中各个样例均不同的生成结果。

## 3.2将生成网络结构应用于判别网络

判别网络所做的事情与所有分类用的神经网络所做的事情是一样的，它们的目的只有一个，就是将训练集中的样例不同的特征提取出来，并以此作为分类依据。

然而在实现上，判别网络毕竟也是神经网络，生成网络同样是基于神经网络的一种应用，其区别在于生成网络将随机噪音变成目标图片，而判别网络将输入图映射到一个分类的标签。因此，生成网络的结构可以直接复制到判别网络中，所要改变的只有输出部分，将原先生成网络的输出图变成分类的标签。

如下是以3×3为例的生成网络输出神经元的结构

（3-4）

由于判别网络的分类设定仅采用输出神经元的前两个神经元，因此，当结果为“横”的时候，n11神经元为1，其余为0：

（3-5）

而当结果为“竖”的时候，n12神经元为1，其余为0：

（3-6）

这样就巧妙地将生成网络的图片输出结果，变为了判别网络的分类输出结果。

在使用训练集进行判别网络的训练时，我发现了对于标准BP的累积化使用方法。其具体原理如下：

在数学上，我们判断一个曲线与另一个曲线相同采取的办法是：

1. 定义域与值域相同
2. 映射法则相同

判别网络所在做的工作实际上就是在寻求一种映射法则，能够将训练集中的样例映射到其理想结果。在判别网络中，其定义域与值域就是训练集中的样例与标签，由于使用神经网络进行分类的训练话就是寻求样例与标签的映射法则，我们不能将未知的法则用来做比较，因此判断我们的神经网络能够寻找到该映射法则的方法要进行调整：

1. 定义域与值域相同
2. 两函数在取值范围内处处相等

当两个函数在其定义域范围内处处相等时，即便映射法则可能略有不同，但我们依然可以认为在该定义域范围内，生成的映射法则与目标映射法则效果相同。

由于在实际的判别网络训练中，处处拟合意味着每个样例的训练误差均为0，这在生产生活实际中过于理想以至于不能够实现，因此要设定一个误差级数，实验中采取三个级数作为每次的训练目标，分别是百分之一、千分之一和万分之一。当误差小于该级数，我们便认为该样例已训练成功。

根据这个思路，在训练判别网络时，对每一个样例进行单独拟合，直至训练到设定的目标误差一下后进行下一个目标的训练。

由此对于整个训练集的单次训练流程的思路已经设定完毕，但对于整个训练集来说仍然没有一个指标用于衡量训练集在该神经网络上的表现。因此对于整个训练集也要有一个误差级数，用以衡量整个训练集的训练效果。这个总训练集的总误差由每次跑完的样例的误差求和而得，所设定的级数可以与单个样例的训练误差级数相同，经过测试，在该模型下运行效果良好，并可在多项式时间内完成训练。

在进行判别网络的训练时，由于涉及到测试集，因此将所有样本均分，一半用于训练，一般用于测试。对于3×3的“横”和“竖”图集而言，均分别存在9种不同长度及位置的情况，因此总训练集共18种样例，“横”和“竖”分别取一半作为训练集一半作为测试集，实验结果如下：

表3-1 3×3训练集测试结果



由于训练集数量太少导致正确率偏低。从实验结果上来看，本设计的判别网络效果与训练误差级别相关性最大，与隐层数量关联性不高，甚至在多隐层情况下正确率不升反降，并且单隐层时效果最好。

当输入为3×3的图片时“横”和“竖”情况简单，且数量稀少致使判别网络效果不优，因此将输入图片加大尺寸变为5×5再进行测试。由于输入图尺寸变大，相应的训练时间就会加长，考虑到3×3的训练结果，5×5仅给出单隐层各误差级数下的测试结果，因为多隐层效果不好且耗时巨大。当图片扩大到5×5时，相应的“横”和“竖”的情况分别有50种，按照3×3的方法进行训练集与测试集的划分，测试结果如下：

表3-2 5×5训练集测试结果



当训练集数量提升，判别网络所能学习到的特征变得丰富，与训练集的拟合程度越高，正确率就得到了相应的提升。

## 3.3生成对抗网络的实现

生成对抗网络是生成网络与判别网络的综合运用，在判别网络与生成网络的共同作用下生成一张新的图片。在这个过程中，要预先训练好判别网络，判别网络扮演指导者，生成网络扮演学习者，其二者关系如同师生，生成网络将一张随机噪音变成图片，判别网络将其错误反馈给生成网络，生成网络重新学习后将错误更正，继续生成新图片。在这个过程中，占主导地位的并不是生成网络，而是判别网络，判别网络拥有绝对的主导权，他将直接影响生成网络的最终走向。以生成“横”和“竖”为例，如果判别网络希望生成网络学会“横”的特征，就将生成网络的生成图与“横”的类别作比较，将误差反馈给生成网络；如果判别网络希望生成网络学会“竖”的特征，就将生成网络的生成图与“竖”的类别作比较，并将误差反馈给生成网络。在这个过程中，我们使用概率来判断生成图的最终分类，例如判别网络通过计算得出该图有80%的概率为“横”，有20%的概率为“竖”，那么我们就认为这张图的类别是“横”。那么有一种特殊的情况就是判别网络认为生成网络的生成图是“横”是“竖”的概率各有50%的时候，在这种情况下我们就可以认为这张生成图同时具有了“横”和“竖”的特征，以至于可以混淆判别网络。当然实际种并不能如此理想，因此如果我们想要生成一张中间类，或者说叫“图灵类“的图片，就可以将判别网络分类判断概率的50%作为目标，将误差反传，多次训练后便可使生成网络能够生成”图灵类“的图片。

以3×3的随即噪音为输入进行全训练集训练，误差级数为万分之一，分别以“横”和“竖”及“图灵类”为目标，得到以下实验结果：

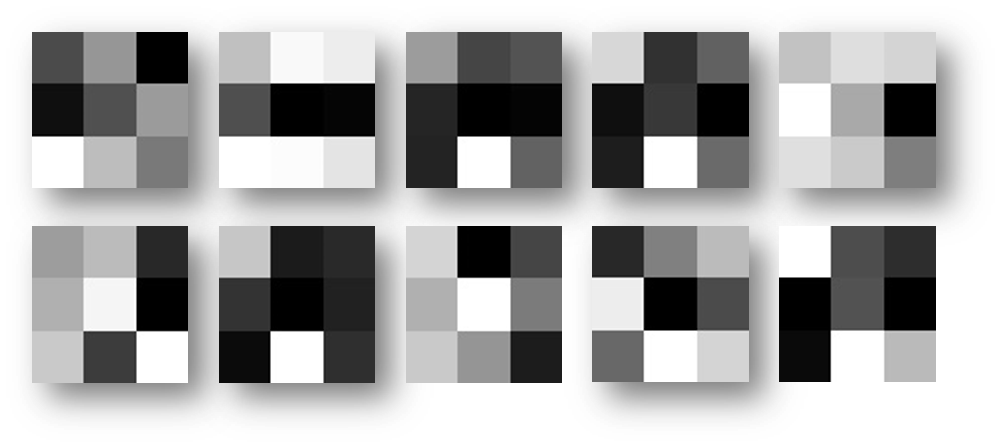


图3-1 生成对抗网络生成的3×3“横“图

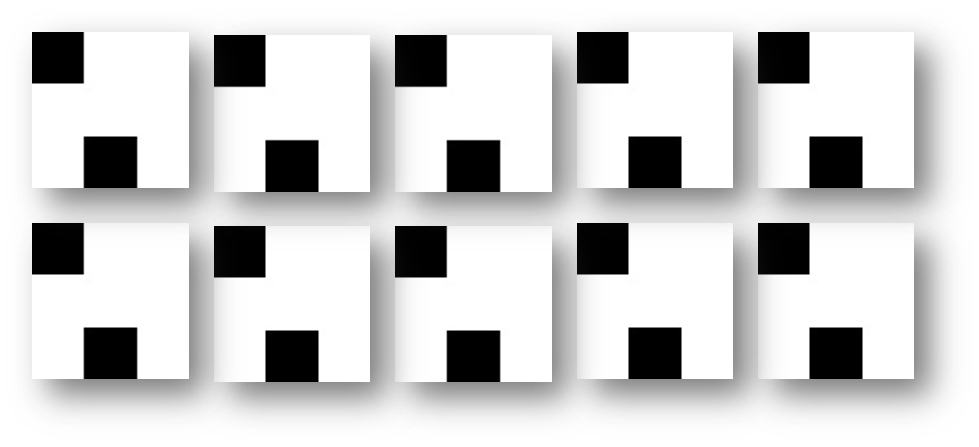


图3-2 生成对抗网络生成的3×3 “竖“图

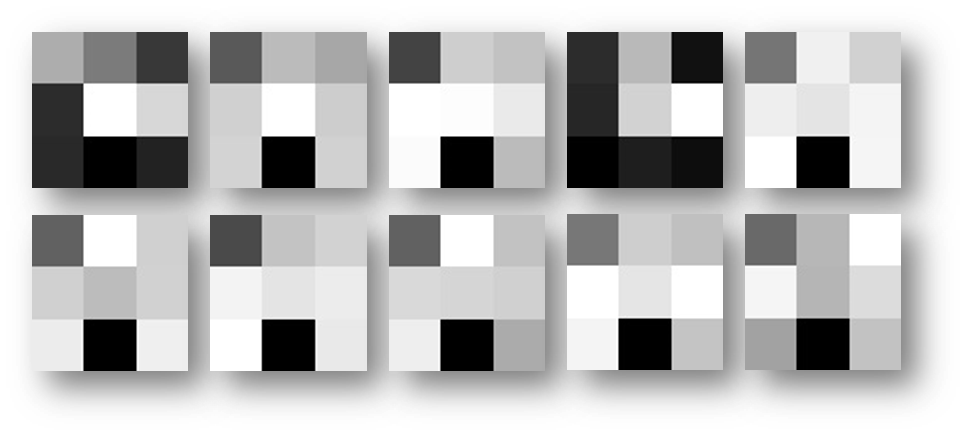


图3-3 生成对抗网络生成的3×3 “图灵类“图

在以上实验中生成的“横”图是判别网络认为99%以上的概率分类为“横”，“竖”图是判别器认为95%以上的概率分类为“竖”，在生成“竖”图时，误差最终会收敛至0.0023。而“图灵类”的图为判别网络认为50%概率为“横”，50%概率为“竖”。

当输入的随机噪音扩大为5×5，生成“横”图时，误差会收敛至0.0031，生成网络会给出一张94.27%概率分类为“横”的图：



图3-4 生成对抗网络生成的5×5 “横“图

生成“竖”图时，误差会收敛至0.90，生成网络会给出一张94.78%概率分类为“竖”的图：



图3-5 生成对抗网络生成的5×5 “竖“图

生成“图灵类”图时，误差会收敛至0.20，生成网络会给出一张94.78%概率分类为“横”的图：



图3-6 生成对抗网络生成的5×5 “横“图

生成的“图灵类”的图时判别器给出了“横”图，该图和生成的“竖”图在宏观上看毫无区别，但是在数据上，这两个图存在微小的差别：

这是生成“图灵类”的图时给出的“横”图数据：

[1.14871894e-110 1.00000000e+000 3.03079895e-038 1.00000000e+000

1.00000000e+000 1.00000000e+000 5.33855547e-001 1.00000000e+000

5.59523950e-027 1.00000000e+000 4.09522698e-023 8.79909574e-138

7.35131795e-030 1.00000000e+000 9.99999291e-001 4.66947106e-097

2.83824118e-005 3.96655240e-069 4.81161048e-016 1.07853054e-007

5.91901834e-082 1.00000000e+000 1.00000000e+000 1.00000000e+000

2.32759474e-003]

这是生成“竖”的图时给出的“竖”图数据：

[5.72340151e-219 1.00000000e+000 2.83347079e-081 1.00000000e+000

1.00000000e+000 1.00000000e+000 5.32951440e-001 1.00000000e+000

8.76313279e-054 1.00000000e+000 1.57892914e-045 6.83279125e-290

2.63784402e-058 1.00000000e+000 1.00000000e+000 2.29849244e-189

8.04866804e-016 3.15968536e-135 2.79572932e-029 5.67952581e-018

1.27559619e-165 1.00000000e+000 1.00000000e+000 1.00000000e+000

1.11638426e-004]

可以看到两图的数据最微小时出现了10的-290次方的数据，并且存在大量数据为1的地方，这说明在训练过程中出现了梯度消失的问题，这也是之前模型误差无法下降的问题。

# 第四章 总结

## 4.1梯度消失及解决思路

在实验的一开始，本设计尝试使用mnist手写数据集做训练，该数据集中的图片为正方形，边长像素28，这意味着全连接的矩阵大小为784×784，因此在第二层的神经元接收到参数并求和后数据就会变得巨大，致使激活函数直接得出1，并失去梯度。这也是本设计采用3×3和5×5作为输入的原因。

该问题的解决方案有两种：

第一种，更换激活函数。sigmod函数在两个极端会出现平滑的收敛导致梯度消失，因此可以更换为ReLU函数，该函数表达式如下：

（4-1）

该函数另一大优点就是再求导的时候导数为1，这可以大大简化误差反传时的计算。

第二种，采用卷积神经网络。卷积神经网络的卷积核小，不会造成大量参数求和导致下一节点接收到无穷大的参数致使梯度消失。然而卷积神经网络在计算上会有额外的问题，例如：池化层在误差反传时的恢复，卷积神经网络使用二维的小矩阵进行相关性运算，因此不能将输入图拆成一维进行矩阵点乘运算，运算难度大大增加。同时在误差反传时，全连接网络直接乘以权值矩阵的转置，而该情况在卷积神经网络中需要点积平面内旋转180°的权值矩阵。

小卷积核也意味着连接权数量的减少，因此在模型中需要存在多层卷积共同提取特征，在该过程中并不能保证所有的卷积核提取到不重复的特征，因此卷积核的数量也是待解决的问题之一。

卷积神经网络很复杂，应用于生成网络的结构也并不能直接照搬到判别网络，如果说判别网络是金字塔型的结构，生成网络就是倒金字塔的结构，使用卷积神经网络作为基本结构实现生成对抗网络时，生成网络与判别网络呈对称结构。

尽管卷积神经网络的实现难度较高，但仍不失为解决梯度消失问题的一大利器。

## 4.2效果不佳及解决思路

全连接结构的神经网络带来的另一大弊端仍然是是计算量上的问题，尽管计算机的运算速度在近十几年间得到了飞跃性的提升，但大图片带来的天文数字级别的计算量仍然是现代计算机无法逾越的一座高峰，原因在于图片的增大，会导致权值矩阵的尺寸以指数爆炸的幅度扩大，这意味着在同等运算能力下，如果想要运算更多的训练数据，就必须缩小输入图的尺寸，而如果要输入图的尺寸较大，带来的运算量的负担就要由减少训练数据量来弥补。在本实验中，跑满整个训练集的情况下，将训练误差级数设定在万分之一，3×3的输入图在极短的时间内便可看见结果输出，而5×5的数据集就需要训练长达6分钟以上的时间。

而使用3×3的训练集，会有额外的问题。当输入图小，数据量也会自然减少，在这种情况下得到的训练结果精度不高容易犯错。从实验数据上看，拟合的效果不够理想。在这种情况下要想提升输出结果的质量，只能将更多的非目标情况列举出来，从反面类别提升神经网络的拟合度。

在之前的实验中，训练集用且仅用了具备目标特征的所有情况，对于非目标情况没有做特定的区分，如要改进可以遍历所有非目标情况，用以从反面加强全连接网络的拟合效果。

另外，在实验中理想的目标图都是二值化的样例，但输出图均是灰度图，因此也可以总灰度的不同进行样本数量的提升，但该方法效果预计不如前一种，让仍不失为一种弥补模型缺点的优秀策略。

# 参考文献

1. 郭睿, 宋忠江. 机器学习: 人工智能的未来[J]. 电子世界, 2018 (4): 33-35.
2. Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[C]//Advances in neural information processing systems. 2014: 2672-2680.
3. 邱智勇. 脉冲耦合神经网络在图像配准和插值中的应用[D]. 厦门大学, 2008.
4. 朱大奇. 人工神经网络研究现状及其展望[D]. , 2004.
5. 蒋宗礼. 人工神经网络[M]. 北京: 高等教育出版社, 2001.
6. 刘涵, 贺霖. 深度学习进展及其在图像处理领域的应用[J]. ZTE TECHNOLOGY JOURNAL, 2017.
7. 罗建豪, 吴建鑫. 基于深度卷积特征的细粒度图像分类研究综述[J]. 自动化学报, 2017, 43(8): 1306-1318.
8. 刘飞, 张俊然, 杨豪. 基于深度学习的医学图像识别研究进展[J]. 中国生物医学工程学报, 2018, 37(1): 86-94.
9. 胡长胜, 詹曙, 吴从中. 基于深度特征学习的图像超分辨率重建[J]. 自动化学报, 2017, 43(5): 814-821.
10. 李卫. 深度学习在图像识别中的研究及应用 [D][D]. 武汉理工大学, 2014.
11. 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.
12. 张翔, 丁晶. 提高多层前馈网络推广能力的权值控制算法[D]. , 1998.
13. 周开利, 康耀红. 神经网络模型及其 MATLAB 仿真程序设计[M]. 清华大学出版社有限公司, 2005.
14. Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators[J]. Neural networks, 1989, 2(5): 359-366.
15. 阎平凡, 张长水. 人工神经网络与模拟进化计算[M]. 清华大学出版社有限公司, 2005.

# 致谢

感谢林小竹老师的悉心教导，让我能够在大学接触到如此前沿的机器学习知识，有幸得到林老师的指导是我的幸运，跟随着林老师，我不仅学到了图像处理的知识，还学到了Q-Learning强化学习的算法，最后在我最喜欢的老师的指导下，我得以完成自己的本科毕业设计。

感谢王万霖同学将我带入Python世界，我一直记得大一的那一天，万霖拿着一本Python书来到我的宿舍改变了我的人生。直到现在，我的主流编程语言依然是Python，并且凭借这门语言和所学的机器学习知识找到了实习工作。

感谢我的师兄师姐在每周三的碰面中对我提出的问题的指引和解答，很多关键性的突破都是师兄师姐的功劳，尤其是当我发现生成网络在使用标准BP时出现的遗忘性，朱师兄即刻给出了累积BP时的结果。

最后感谢所有给予过我帮助的老师及同学，随不能一一赘述，然感激之情溢于言表。

# 附录A 生成网络

#!/usr/bin/env python3

# 可变隐层数全连接生成器，将随机噪音生成横或竖图片，用于训练权值并存储

import numpy as np

import math

# 各层神经元输入

Neuron\_Input = []

# 各层间权值矩阵

Weight = []

# 各层神经元输出

Neuron\_Output = []

# 神经网络隐层层数设定

n = 2

# 神经元个数设定

size = 5

Neuron\_Num = size \* size

# 神经网络节点层输入输出构建

for i in range(n+2):

Neuron\_Input.append(np.zeros((Neuron\_Num), float))

Neuron\_Output.append(np.zeros((Neuron\_Num), float))

# 神经网络权值矩阵层构建

for i in range(n+1):

Weight.append(np.random.rand(Neuron\_Num,Neuron\_Num))

# 输出层误差集合定义

E = np.zeros((Neuron\_Num), float)

# 学习率设定

η = 0.5

# 设定训练误差级数

E\_level = 2

# 读取数据集

with open('横'+str(size)+'×'+str(size)+'.csv', 'r') as f:

Original\_String = f.read()

First\_Process = Original\_String.split('\n')

E\_Sum = 1

Number\_of\_training = 0

while E\_Sum > 10 \*\* (-E\_level) and Number\_of\_training < 3:

E\_Sum = 0

for ele in First\_Process:# 训练集范围更改在这里

Second\_Process = ele.split(',')

Third\_Process = list(map(int, Second\_Process))

# 目标值设定

Target = np.array(Third\_Process[1:])

# 底层神经元设定

Neuron\_Input[0] = np.random.rand(Neuron\_Num)

Neuron\_Output[0] = Neuron\_Input[0]

E\_total = 1

# dqcs = 0

while E\_total > 10 \*\* (-E\_level):

for j in range(n+1):

Neuron\_Input[j+1] = np.dot(Neuron\_Output[j], Weight[j])

for i in range(len(Neuron\_Input[j+1])):

Neuron\_Output[j+1][i] = 1 / (1 + math.exp(-Neuron\_Input[j+1][i]))

# 计算总误差

for i in range(len(E)):

E[i] = 0.5 \* (Target[i] - Neuron\_Output[n+1][i]) \*\* 2

E\_total = 0

for ele in E:

E\_total += ele

# 反向计算

# 计算并存储每层节点输出对输入的偏导

Node\_PD = []

for ele in Neuron\_Output[-1:0:-1]:

Node\_PD.append(list(map(lambda x: x\*(1-x), ele)))

# 计算δ

Delta = []

Delta.append((Neuron\_Output[n+1]-Target)\* Node\_PD[0])# Node\_PD[0]

for i in range(1,n+1):

Delta.append(np.dot(Delta[i-1],Weight[len(Weight)-i].T) \* Node\_PD[i])# Node\_PD[i]

# 修正权

Error = []

for i in range(n+1):

a = Neuron\_Output[i].reshape([Neuron\_Num,1])

b = Delta[-1-i].reshape([1,Neuron\_Num])

c = np.dot(a,b)

Error.append(c)

for i in range(n+1):

Weight[i] = Weight[i] - η \* Error[i]

# if dqcs%10000 == 0:

# print('第', int(dqcs/10000), '万次训练：')

# # print('输出：', Neuron\_Output[2])

# print('误差：', E\_total)

# dqcs += 1

E\_Sum += E\_total

Number\_of\_training += 1

print('第', Number\_of\_training, '次误差：', E\_Sum)

with open('生成器权值.csv', 'w') as f:

n = 1

for ele in Weight:

if n == 1:

n = 0

else:

f.write('\n')

for x in range(Neuron\_Num):

for y in range(Neuron\_Num):

f.write(str(ele[x][y]))

if x != Neuron\_Num-1 or y != Neuron\_Num-1:

f.write(',')

# 附录B 判别网络

#!/usr/bin/env python3

# 可变隐层数全连接判别器，将横竖图片分为两类

import numpy as np

import math

# 各层神经元输入

Neuron\_Input = []

# 各层间权值矩阵

Weight = []

# 各层神经元输出

Neuron\_Output = []

# 神经网络隐层层数设定

n = 1

# 神经元个数设定

size = 5

Neuron\_Num = size \* size

# 神经网络节点层输入输出构建

for i in range(n+2):

Neuron\_Input.append(np.zeros((Neuron\_Num), float))

Neuron\_Output.append(np.zeros((Neuron\_Num), float))

# 神经网络权值矩阵层构建

for i in range(n+1):

Weight.append(np.random.rand(Neuron\_Num,Neuron\_Num))

# 输出层误差集合定义

E = np.zeros((Neuron\_Num), float)

# 学习率设定

η = 0.5

# 设定训练误差级数

E\_level = 4

# 读取数据集

with open('横'+str(size)+'×'+str(size)+'.csv', 'r') as f:

Original\_String = f.read()

with open('竖'+str(size)+'×'+str(size)+'.csv', 'r') as f:

Original\_String += '\n'+f.read()

# with open('特例.csv', 'r') as f:

# Original\_String = f.read()

First\_Process = Original\_String.split('\n')

# print(First\_Process)

E\_Sum = 1

Number\_of\_training = 0

while E\_Sum > 10 \*\* (-E\_level) and Number\_of\_training < 10\*\*(E\_level+1):

E\_Sum = 0

for ele in First\_Process:# 训练集范围更改在这里

Second\_Process = ele.split(',')

Third\_Process = list(map(int, Second\_Process))

# 目标值设定

Target = np.zeros((Neuron\_Num), float)

Lable = int(Third\_Process[0])

if Lable == 0:

Target[0] = 1

else:

Target[1] = 1

# 底层神经元设定

Neuron\_Input[0] = np.array(Third\_Process[1:])

Neuron\_Output[0] = Neuron\_Input[0]

E\_total = 1

# dqcs = 0

while E\_total > 10 \*\* (-E\_level):

for j in range(n+1):

Neuron\_Input[j+1] = np.dot(Neuron\_Output[j], Weight[j])

for i in range(len(Neuron\_Input[j+1])):

Neuron\_Output[j+1][i] = 1 / (1 + math.exp(-Neuron\_Input[j+1][i]))

# 计算总误差

for i in range(len(E)):

E[i] = 0.5 \* (Target[i] - Neuron\_Output[n+1][i]) \*\* 2

E\_total = 0

for ele in E:

E\_total += ele

# 反向计算

# 计算并存储每层节点输出对输入的偏导

Node\_PD = []

for ele in Neuron\_Output[-1:0:-1]:

Node\_PD.append(list(map(lambda x: x\*(1-x), ele)))

# 计算δ

Delta = []

Delta.append((Neuron\_Output[n+1]-Target)\* Node\_PD[0])# Node\_PD[0]

for i in range(1,n+1):

Delta.append(np.dot(Delta[i-1],Weight[len(Weight)-i].T) \* Node\_PD[i])# Node\_PD[i]

# 修正权

Error = []

for i in range(n+1):

a = Neuron\_Output[i].reshape([Neuron\_Num,1])

b = Delta[-1-i].reshape([1,Neuron\_Num])

c = np.dot(a,b)

Error.append(c)

for i in range(n+1):

Weight[i] = Weight[i] - η \* Error[i]

# if dqcs%10000 == 0:

# print('第', int(dqcs/10000), '万次训练：')

# # print('输出：', Neuron\_Output[2])

# print('误差：', E\_total)

# dqcs += 1

E\_Sum += E\_total

Number\_of\_training += 1

print('第', Number\_of\_training, '次误差：', E\_Sum)

with open('判别器权值.csv', 'w') as f:

n = 1

for ele in Weight:

if n == 1:

n = 0

else:

f.write('\n')

for x in range(Neuron\_Num):

for y in range(Neuron\_Num):

f.write(str(ele[x][y]))

if x != Neuron\_Num-1 or y != Neuron\_Num-1:

f.write(',')

# 附录C 生成对抗网络中的判别网络

#!/usr/bin/env python3

# GAN？

import numpy as np

import math

from scipy import misc

# 搭建判别器部分

# 各层神经元输入

Neuron\_Input = []

# 各层间权值矩阵

Weight = []

# 各层神经元输出

Neuron\_Output = []

# 神经网络隐层层数设定

n = 1

# 神经元个数设定

size = 5

Neuron\_Num = size \* size

# 神经网络节点层输入输出构建

for i in range(n+2):

Neuron\_Input.append(np.zeros((Neuron\_Num), float))

Neuron\_Output.append(np.zeros((Neuron\_Num), float))

# 输出层误差集合定义

E = np.zeros((Neuron\_Num), float)

# 学习率设定

η = 0.1

# 设定训练误差级数

E\_level = 4

# 神经网络权值矩阵层构建

with open('判别器权值.csv', 'r') as f:

Original\_String = f.read()

First\_Process = Original\_String.split('\n')

from My\_Function import Build\_Matrix

for ele in First\_Process:

Second\_Process = ele.split(',')

Weight.append(Build\_Matrix(Second\_Process, Neuron\_Num))

# 目标值设定

Target = np.zeros((Neuron\_Num), float)

Target[0] = 0 #横

Target[1] = 1

# 底层神经元设定

Neuron\_Input[0] = np.random.rand(Neuron\_Num)

Input\_Picture = Build\_Matrix(Neuron\_Input[0], size)

misc.imsave('GAN随机输入.bmp', Input\_Picture)

# Neuron\_Input[0] = np.ones((Neuron\_Num), float)

Neuron\_Output[0] = Neuron\_Input[0]

E\_total = 1

Number\_of\_training = 0

while E\_total > 10 \*\* (-E\_level) and Number\_of\_training < 10\*\*(E\_level+1):

# 正向计算

for j in range(n+1):

Neuron\_Input[j+1] = np.dot(Neuron\_Output[j], Weight[j])

for i in range(len(Neuron\_Input[j+1])):

Neuron\_Output[j+1][i] = 1 / (1 + math.exp(-Neuron\_Input[j+1][i]))

# 计算总误差

for i in range(len(E)):

E[i] = 0.5 \* (Target[i] - Neuron\_Output[n+1][i]) \*\* 2

E\_total = 0

for ele in E:

E\_total += ele

# 反向计算

# 计算并存储每层节点输出对输入的偏导

Node\_PD = []

for ele in Neuron\_Output[-1:0:-1]:

Node\_PD.append(list(map(lambda x: x\*(1-x), ele)))

# 计算δ

Delta = []

Delta.append((Neuron\_Output[n+1]-Target)\* Node\_PD[0])# Node\_PD[0]

for i in range(1,n+1):

Delta.append(np.dot(Delta[i-1],Weight[len(Weight)-i].T) \* Node\_PD[i])# Node\_PD[i]

# 修正输入

Error = np.dot(Delta[-1], Weight[0])

Neuron\_Input[0] = Neuron\_Input[0] - η \* Error

# Neuron\_Output[0] = Neuron\_Input[0]

# for i in range(len(Neuron\_Input[0])):

# if Neuron\_Input[0][i] < 0:

# Neuron\_Output[0][i] = 0

for i in range(len(Neuron\_Input[0])):

Neuron\_Output[0][i] = 1 / (1 + 2.718\*\*(-Neuron\_Input[0][i]))

# if xlcs%10000 == 0:

# print(Neuron\_Output[0])

# print('第', int(xlcs/10000), '万次训练：')

# print('误差：', E\_total)

# xlcs += 1

Number\_of\_training += 1

print('第', Number\_of\_training, '次误差：', E\_total)

print(Neuron\_Output[0])

print(Neuron\_Output[2])

# write\_n = 1

# with open('GAN\_Target.csv', 'w') as f:

# for ele in Neuron\_Output[0]:

# if write\_n == 1:

# write\_n = 0

# else:

# f.write(',')

# f.write(str(ele))

Input\_Picture = Build\_Matrix(Neuron\_Output[0], size)

misc.imsave('GAN理想输出.jpg', Input\_Picture)

# 声明

本人郑重声明：所呈交的学位论文（毕业设计说明书），是本人在导师指导下，独立进行研究（设计）工作的总结。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签 名： 日 期：