南京大学

本 科 毕 业 论 文

院　　系　计算机科学与技术系

专　　业　计算机科学与技术

题　　目　权值初始化对梯度下降的影响

年　　级　**2017** 级　学　　号　**171250623**

学生姓名　姜勇刚

指导老师　林冰凯　职　　称　教授

提交日期　**2021** 年 **5** 月 **25** 日

Influence of Weight Initialization on Gradient Descent

by

Yonggang Jiang

Supervised by

Professor Binkai Lin

An Undergraduate Dissertation Submitted to
Department of Computer Science and Technology, Computer Science and Technology

Department of Computer Science and Technology
Computer Science and Technology

May 25, 2021

南京大学本科生毕业论文 (设计、作品) 中文摘要

题目：权值初始化对梯度下降的影响

院系：计算机科学与技术系
专业：计算机科学与技术
本科生姓名：姜勇刚
指导老师（姓名、职称）：林冰凯教授

摘要：

权值初始化是影响梯度下降行为的一个重要因素。这其中有两个值得探究的方面：一个是如何选择正确的初始化以优化梯度下降的效果，另一个是初始化的值和最后训练结果的值有什么样的关系。

本文给出了这两个方面的实验结论和理论解释。首先说明了随机初始化的重要性，并推导了边权初始化值的合理范围。除此之外，我们也探究了初始化的值和最后训练结果值的关系，证明了在满足一定条件的情况下，初始化的值会和最终训练结果的值相差不大。

关键词： 神经网络；初始化；梯度下降；训练；损失函数

南京大学本科生毕业论文 (设计、作品) 英文摘要

THESIS: Influence of Weight Initialization on Gradient Descent
DEPARTMENT: Department of Computer Science and Technology
SPECIALIZATION: Computer Science and Technology
UNDERGRADUATE: Yonggang Jiang
MENTOR: Professor Binkai Lin
ABSTRACT:

Weight initialization is an important factor affecting gradient descent behavior. There are two aspects worth exploring: one is how to choose a good initialization to optimize the gradient descent effect, the other is the relationship between the initialization value and the final training result value.

The experimental results and theoretical explanations of these two aspects are given in this paper. Firstly, the importance of random initialization is explained, and the reasonable range of edge weight initialization value is deduced. In addition, we also explores the relationship between the initial value and the final training result value, and proves that under certain conditions, the initial value will not differ from the final training result.

KEY WORDS: Neural network; Initialization; Gradient descent; Training; Loss function

前　言

人工智能技术是近些年来不断发展的重要技术，并已经和我们的生活牢牢绑定：无人驾驶，图片识别，机器人技术等都离不开人工智能。人工智能技术也成为当代发展的重点，相关的研究工作层出不穷。正因为人工智能的广泛适用性和重要性，此毕业论文也将进行人工智能方面的探究。

在人工智能的研究中，近些年来不断发展的主流方法是神经网络：用以拟合生物神经元的函数结构。而训练神经网络的最简单，最常用的方法就是梯度下降法。梯度下降训练神经网络的方法由于其实现简洁，实验效果佳而被广泛使用。但由于梯度下降法训练复杂神经网络的数学结构的复杂性，使得理论分析变得困难，这也使人们对这一工具的理解存在着许多缺漏，往往只能通过实验结果得到结论，而不能从理论上解释。

由于笔者对计算机理论方面的兴趣，这次毕设论文打算从理论的角度出发进行探究。主要的想法是研究影响梯度下降的要素，而笔者认为这些要素中初始化值是最为重要的要素之一。通过不断的尝试，最终也是得到了一些实验和理论的结果。虽然未能对科学前沿起到推进的作用，但也是对自己的一次很好的训练和知识拓展，也可以成为以后更加深入研究的基础。

姜勇刚

目　录

第一章　绪论

1.1　研究背景

在现代人工智能的发展和研究中，应用神经网络的学习来拟合一个问题进而产生一个问题的解，已经成为了一种主流的方法。而在神经网络的学习中，我们往往是通过调整一个神经网络的参数来使得输出和真实值定义的损失函数值不断变小，而这种通过调整变量而使得一个函数值变小的问题可以用梯度下降算法：即从一个点出发，在这个点上求出对各个变量的偏导，从而往能在局部使这个函数变小的方向移动。这种梯度下降法由于其简单实用的特性而被广泛运用。

影响梯度下降的性能的有很多因素，包括初始值的设定，步长的选择和调整，以及函数局部最优的位置分布等等。其中初始值的选择十分重要，一个坏的初始位置如果正好选在离全局最优很远的位置，或者选择一个局部最优附近，可能会大大影响梯度下降的性能，甚至可能会使梯度下降法不收敛。在这篇文章中，我们就从实验和理论两个角度来研究初始值对梯度下降法结果的影响。

1.2　相关工作

关于初始化在梯度下降中的重要性，有许多文章给出了讨论（见[1]，[2]）。在这些文章中，作者提出：初始化值往往对梯度下降的结果会有很大的影响，且文章最后给出的高正确率实验结果都来源于精心设计的初始化。

有许多工作都对神经网络梯度下降应该如和初始化进行了研究，包括两篇重要的工作[3]，[4] 分别提出了两个被广泛应用的初始化方法：Xavier 和 He 初始化。我们在后文中会提到这两种方法。

在[5] 中，作者对图神经网络进行了研究。作者指出，图神经网络的随机初始化会相对单一初始化（初始化为同一个值）达到更好的区分效果。由这个结果可以看出初始化方法在神经网络中的重要性。这一结果是本文研究的灵感来源。

还有许多工作对初始化方法和梯度下降的收敛性关系进行了研究，这些工作主要集中在过参数化（over-parameterized）情况下的研究，包括[6]，[7]，[8]，[9]。在这些研究中，作者主要考虑的是网络在中间层节点数比初始节点数多很多的情况下，在一定的随机化条件时，梯度下降最后的收敛效率是否有保障。

1.3 本文主要工作

本文主要探究矩阵参数值初始化对梯度下降的两个角度的影响：

1. 使用什么样的初始化能使梯度下降快速收敛，以及使在训练集上使用梯度下降得到的结果在测试集上表现出高正确率？
2. 考虑比较一开始对参数初始化的值，和训练结束之后得到的最终训练结果参数的值，他们之间是否有一些关系？

本文将分别在这两个角度上设计、进行实验，观察实验数据和现象，并用理论来解释现象。

同时由于初始化对梯度下降的影响是一个比较基本的问题，在科学家前沿也有很多的研究工作。本文也对这些工作中与上述两个角度相关的文章进行了一些整理，并给出了自己的推导和理解。

1.4 本文结构

在章节2，本文将介绍一些基础知识，介绍神经网络的背景并定义清楚神经网络的严格数学模型，介绍几种常用的神经网络。同时也会介绍关于梯度下降的相关知识，介绍常用的损失函数、介绍梯度下降在神经网络中具体的实现方法和重要的正向传播和反向传播公式：

$$\nabla M^{(i)} = \nabla \boldsymbol{y}^{(\boldsymbol{i+1})} \cdot \boldsymbol{y}^{(\boldsymbol{i})} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(\boldsymbol{i+1})}\right)$$

$$\nabla \boldsymbol{y}^{(\boldsymbol{i})} = M^{(i)} \cdot \nabla \boldsymbol{y}^{(\boldsymbol{i+1})} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(\boldsymbol{i+1})}\right)$$

这两个公式将贯穿本文的理论分析，在2.2.2节中定义。

在章节3中，本文将通过实验找出正确的初始化方法（实际上是确定初始化值大小的范围），并对实验现象进行理论分析，同时介绍两种常用初始化Xavier 和 He 初始化的理解和推导。

在章节4中，本文将通过实验先观察初始化值和训练结果值的关系，并根据实验给出猜想，通过理论分析解释实验结论，同时证明了一个理论的定理4-1用以严格的说明实验现象的原因。

最后我们在章节5中给出本文的整体总结和讨论。

第二章　神经网络与梯度下降

本章将介绍神经网络以及梯度下降的基础知识。在 2.1节，我们将介绍神经网络的基本原理，以及几种常见的神经网络。在 2.3，我们将介绍损失函数的定义（也就是我们想用梯度下降法去优化的目标），以及几种常用的损失函数类型。在 2.2.2，我们将介绍梯度下降法，并且介绍一种适用于在神经网络中实现梯度下降法的经典方法——反向传播算法。

2.1　神经网络

自然界中的问题（人类想用计算机解决的问题）往往可以用函数表示：棋类游戏下一步应该下哪里，是一个从棋盘局面到下一步落子位置的函数；给一张图片识别出图片中物体的类别，是一个从图片像素点集排列到物体种类集合的函数；给一个题目和目标想写一篇论文，是一个从题目和目标到特定汉子排序的函数。

神经网络也是这样的一个从输入到输出的映射，用来刻画一个问题的解法。神经网络的设计理念和名字一样，灵感来源于人类是如何思考事物（产生解）的（因此神经网络其实是在模拟人类，所以被适用于人工智能的研究）。人类的大脑由一个一个神经元组成，每个神经元也相当于一个函数，当输入超过一个阈值之后输出就会被激活，一个神经元的输出连接到下一个神经元的输入，这样构成的网络组成了一个复杂的复合函数结构，来处理人类的各种思考和行为。把这个原理抽象成模型之后可以得到如图2-1所示的结构。

如图2-1所示，输入有三个节点，每个节点代表一个输入值（比如一个整数）。每条边有一个权重，每个中间节点的值等于以下所有值的和：左边节点的值乘以连向这个边的权重。输入层和输出中间的节点构成隐藏层。注意到人的神经元是“超过一个阈值之后产生输出”，所以往往神经网络里会安排激励函数，在输入小于 0 的时候输出 0，输入大于 0 的时候输出输入值（这里指常用的 ReLu 函数，也有很多其他种类的激励函数）。这样就完成了一个类似人类思考方式的网络结构。

可以看出，这样的网络定义出了一个函数，这个函数由网络的结构（点数，层数，边的连接方式，激励函数）以及边的权重决定。

以上为从神经网络的由来（模拟人脑神经元）的角度出发介绍的神经网络，实际上可以以一种更数学的方式定义神经网络：假设网络结构如下：

1. 输入是一个 n 维的向量 $\boldsymbol{x}$，输出是一个 m 维向量 $\boldsymbol{y}$，神经网络总共有 $d+1$ 层（第 0 层为输入，第 d 层为输出）。
2. 其中第 i 到第 $i+1$ 层的边权用矩阵 $M^{(i)}$ 表示（也就是说第 i 层的第 l 个节点连到第 $i+1$ 层的第 r 个节点的边的边权是 $M_{l,r}^{(i)}$）。

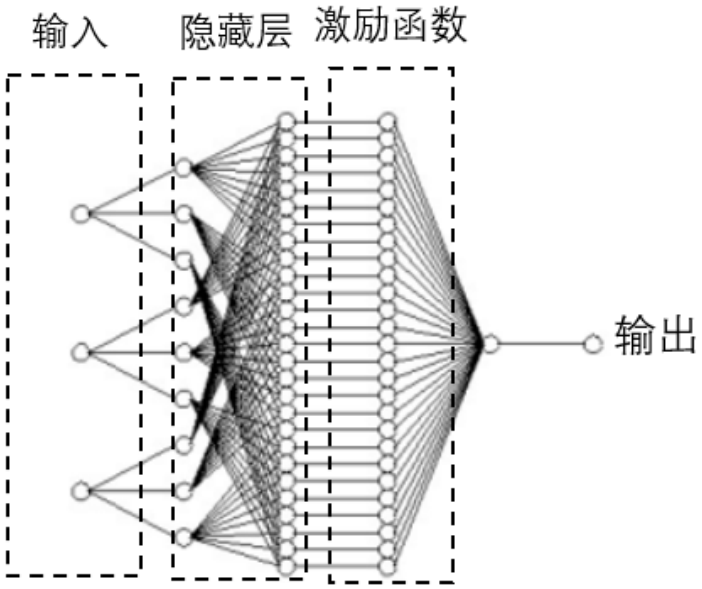


图 2-1: 神经网络示例

3. 第 i 层的输出经过一个激励函数 σ_i。

则这个神经网络递归定义了这样的一个函数：

$$\boldsymbol{y^0} = \boldsymbol{x}$$

$$\forall 0 \le i \le d-1, \boldsymbol{y^{(i+1)}} = \sigma_{i+1}\left(\boldsymbol{y^{(i)}} \cdot M^{(i)}\right)$$

$$\boldsymbol{y} = \boldsymbol{y^{(d)}}$$

也就是说，通用的神经网络函数形如 $\boldsymbol{y} = \sigma_d\left(\sigma_{d-1}\left(\sigma_{d-2}(...) \cdot M^{(d-2)}\right) M^{(d-1)}\right)$。

上面提到，神经网络函数由网络的结构和边权两部分决定。人们在构造一个用于解决一个特定问题的神经网络时，往往是先按照人类对这个问题的经验认识等先固定神经网络的结构，再用计算机采用一定算法（如梯度下降）找到解决这个问题的最佳边权。下面我们先介绍 4 种本文将涉及的常用神经网络。

2.1.1 全连接神经网络

全连接神经网络是最简单的神经网络，如图2-1所示，全连接神经网络将每相邻两层的所有边都连成一个完全二部图。

实际上这样的网络结构也是一种一般性的网络结构，可以用来表示其他网络。其他的结构（非全连接）去掉了一些边，其实可以看成是这些边在全连接神经网络里面的权重恒为零，在后续的边权寻找中也不会改变这些恒为零的边权。

全连接神经网络最大的缺点就是边的数目过多，在两层之间的边和两层的点数成平方关系。这会直接导致后续需要训练的参数（各个 $M^{(i)}$ 矩阵的值）数

目过多，从而导致效率低下。这造成的一个直接后果是为了达到合理的训练时间，点数会做相应的舍弃。但在后面的4中我们会看到，每一层点数（也就是宽度）会影响到神经网络的工作效果。因此，我们往往需要去掉一下边的其他网络结构。

2.1.2 卷积神经网络 (CNN)

下面我们介绍一种在全连接神经网络的基础上删除掉一些特定的边之后得到的卷积神经网络。卷积神经网络最初的概念来自于对图片进行识别。考虑人类对一个图片的识别，往往是寻找图片中一些特定的标志物，再根据这些标志物的位置、组合等识别出一个图片。比如人类在判断一个图片是鸟的过程中，实际上是看到了鸟嘴、鸟眼组合成鸟头，再加上翅膀等，通过正确的位置组合判断出这是一只鸟摆出的可能的动作，如图2-2所示。

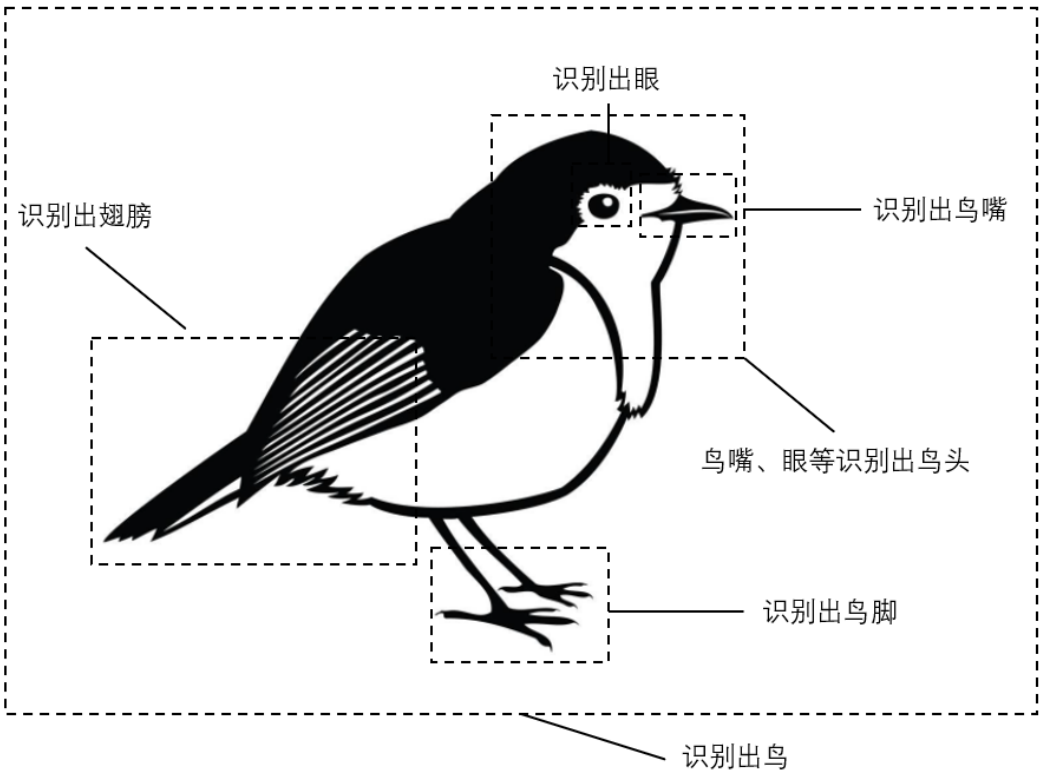


图 2-2: 识别鸟的过程

通过一个区域块识别出一个结构，这很像对一个区域块的输入通过一个节点来处理，这个节点的输出代表了识别出怎样的一种结构（如眼，嘴）。而这些结构的位置排列又可以输入下一个节点，判断是不是一种更高层的结构（如头），最后得到整个图片的一种识别。

基于以上的这种想法，我们可以画出如图2-3所示的卷积神经网络。

图2-3省略了激励函数，且是一个简易的卷积神经网络示例，只有一层隐藏层。实际使用中多层隐藏层会使效果更好。图示将输入的第一个、第二个、

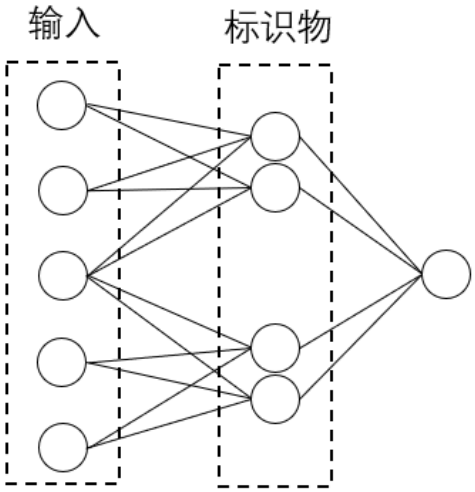


图 2-3: 卷积神经网络示例

第三个节点输入到了隐藏层的两个节点中，分别代表识别这三个节点的两种标志物。而输入的第四个、第五个、第六个节点也被输入两个节点中。可以看出标志物是可以重叠的。这样的网络结构就达到了上面描述的人类识别过程。

可以看出，上面所描述的全连接神经网络在两层之间的边数与点数的关系是平方关系。而卷积神经网络由于输入的每一块点只与下一层的某一块点产生全连接，当这两块点的数目上限为常数（比如 q）时，可以看出两层之间的边数最多是 $n \cdot q^2 = \Theta(n)$，边数减少了很大的数量级，同时由于上述的讨论，神经网络的效果也会好很多。

实际在使用卷积神经网络的时候，会添加其他的一些优化，比如加入池化操作进一步减少节点等。本文后面的实验也会采用卷积神经网络进行训练，往往能达到比较好的效果。

2.1.3 残差神经网络 (ResNet)

神经网络往往层数越多表示能力越强：层数少的网络总是可以被包含在层数多的网络当中。虽然层数会增大训练难度，以及潜在的训练效果问题，但基于表示能力更强的原因，深层的神经网络也被广泛应用。但深层的神经网络也有缺点，比如会带来传播过程中的损失。而残差神经网络的设计思路就来源于这里。

正常的神经网络每一层对输入的处理是把输入乘一个矩阵（线性变换）后产生输出。现在我们希望在网络层数过多的时候每一层只对输入进行一些“修正”，也就是说，每一层做的事情变成了，对输入进行一个线性变换后加到原来的输入中。严格来说，残差神经网络对原本神经网络的定义做了一些改变，变成如下的定义：

$$\boldsymbol{y}^{\mathbf{0}} = \boldsymbol{x}$$

$$\forall 0 \le i \le d-1, \boldsymbol{y}^{(i+1)} = \boldsymbol{y}^{(i)} + \sigma_{i+1}\left(\boldsymbol{y}^{(i)} \cdot M^{(i)}\right)$$

$$\boldsymbol{y} = \boldsymbol{y}^{(d)}$$

在后面的章节我们将看到，残差神经网络对深层的神经网络有很好的训练效果，能有效的避免一般神经网络传播过程中的指数型误差累积。

2.2 损失函数与梯度下降

2.2.1 损失函数

在上一节我们定义了网络模型（也就是函数 f）。记得之前说过 f 在结构确定的情况下，由边权（矩阵）唯一确定。我们想要找到合适的矩阵值使得函数 f 能完成一个实际的任务：让 f 逼近我们想要的真实函数 f^*。这里 f^* 我们是不知道的，但是我们可以找到很多的“训练数据”：一系列 $(x, f^*(x))$ 对。于是问题变成了，如果根据这一系列数据训练出足够逼近 f^* 的 f？

我们把这个问题转化为一个优化问题。假设我们拥有的训练数据集合是 $S = \{(x_i, f^*(x_i)\}_{i\in[n]}$。我们构造一个合适的“损失函数”：$loss(S, f)$ 使得这个损失函数能合理的描述 f 和 f^* 的距离。这个函数以训练集和 f 为自变量，一个自然的想法就是找到所有的 $f(x_i)$，并且和 $f^*(x_i)$ 做比较，将所有的比较出来的差做和，这就是一个合理的损失函数：尽量让这个损失函数小可以让 f 至少在这个训练数据集上不断的逼近真实函数 f^*。在2.3节我们将介绍几个常用的损失函数。

2.2.2 梯度下降

梯度下降是一种很好的优化函数的算法。主要思路为对损失函数的各个变量求偏导，找出能达到减小损失函数的方向，加或减在当前位置上对应的偏导，到达下一个位置。如此不断迭代，直到达到一个基本稳定的位置。

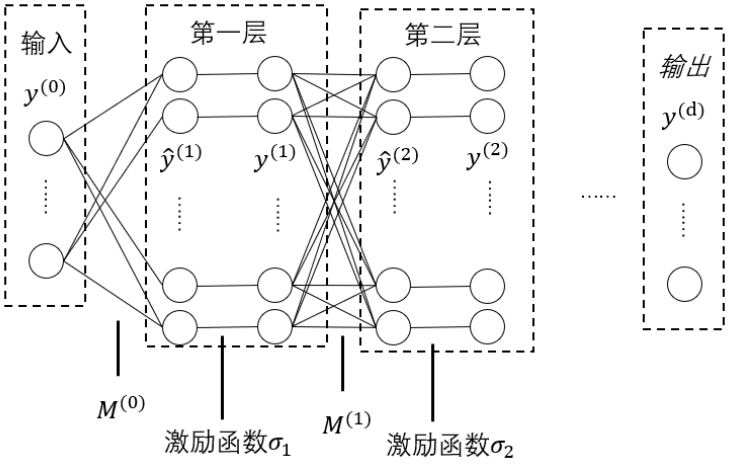


图 2-4: 多层网络示例

在每一轮，我们对影响 $loss(S,f)$ 的每个变量求偏导。注意到我们想优化的目标是每个矩阵的元素。于是对于 $M_{l,r}^{(i)}$ 求偏导得到

$$\frac{\partial loss(S,f)}{\partial M_{l,r}^{(i)}} = \frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i+1)}{}_r} \cdot \frac{\partial \boldsymbol{y}^{(i+1)}{}_r}{\partial M_{l,r}^{(i)}}$$

这里 $\boldsymbol{y}^{(i)}$ 为第 i 层（包括激励函数 σ_i）的输出，如图2-4所示。我们令 $\hat{\boldsymbol{y}}^{(i)}$ 表示第 i 层不包括激励函数的输出，即 $\boldsymbol{y}^{(i)} = \sigma_i\left(\hat{\boldsymbol{y}}^{(i)}\right)$，则有

$$\frac{\partial loss(S,f)}{\partial M_{l,r}^{(i)}} = \frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i+1)}{}_r} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(i+1)}{}_r\right) \cdot \boldsymbol{y}^{(i)}{}_l$$

其中对节点的偏导可以表示成

$$\frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i)}{}_l} = \sum_{r\in[d_{i+1}]} \frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i+1)}{}_r} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(i+1)}{}_r\right) \cdot M_{l,r}^{(i)}$$

于是我们可以写成如下形式的紧凑形式

$$\left(\frac{\partial loss(S,f)}{\partial M^{(i)}}\right)^T = \left(\frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i+1)}}\right)^T \cdot \boldsymbol{y}^{(i)} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(i+1)}\right)$$

$$\left(\frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i)}}\right)^T = M^{(i)} \cdot \left(\frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i+1)}}\right)^T \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(i+1)}\right)$$

这里 $\frac{\partial loss(S,f)}{\partial M^{(i)}}, \frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i)}}$ 表示对矩阵或向量的每一个元素分别求偏导。我们将 $\left(\frac{\partial loss(S,f)}{\partial M^{(i)}}\right)^T$ 写成 $\nabla M^{(i)}$，把 $\left(\frac{\partial loss(S,f)}{\partial \boldsymbol{y}^{(i)}}\right)^T$ 写成 $\nabla \boldsymbol{y}^{(i)}$。这里 $\nabla M^{(i)}$ 表示边权的梯度矩阵，矩阵的第 i,j 个元素表示损失函数对这个矩阵的第 i,j 个元素的偏导。$\nabla \boldsymbol{y}^{(i)}$ 同理是中间隐藏层的梯度，是一个 d_i 维行向量（注意到 $\boldsymbol{y}^{(i)}$ 也是行向量）。σ'_i 是 σ_i 的导数，自变量为向量时，表示对每一个向量的元素使用 σ'_i。于是我们得到

$$\nabla M^{(i)} = \nabla \boldsymbol{y}^{(i+1)} \cdot \boldsymbol{y}^{(i)} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(i+1)}\right) \tag{2-1}$$

$$\nabla \boldsymbol{y}^{(i)} = M^{(i)} \cdot \nabla \boldsymbol{y}^{(i+1)} \cdot \sigma'_{i+1}\left(\hat{\boldsymbol{y}}^{(i+1)}\right) \tag{2-2}$$

在某一次梯度下降的过程中，首先我们需要利用输入 $\boldsymbol{x}$ 和每一层的矩阵 $M^{(i)}$ 和激励函数 σ_i，计算出每一层的输出 $\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}$（如图2-4），这是一次正向传播，直到递推计算到输出 $\boldsymbol{y}^{(d)}$。

然后我们就可以用 $loss(S,f)$ 对输出 $\boldsymbol{y}^{(d)}$ 求偏导（一般来说损失函数 $loss(S,f)$ 是针对输出 $\boldsymbol{y}^{(d)}$ 和真实输出 $f^*(x)$ 之间做比较的函数，所以对 $\boldsymbol{y}^{(d)}$ 有直接的求导表达式）得到最后一层的梯度 $\nabla \boldsymbol{y}^{(d)}$。然后我们通过公式2-2就可以从 $\nabla \boldsymbol{y}^{(d)}$ 开始计算出每一层的梯度 $\nabla \boldsymbol{y}^{(i)}$，这就是反向传播算法。

在计算出每一层的输出和梯度 $\boldsymbol{y}^{(i)}, \nabla \boldsymbol{y}^{(i)}$ 之后，我们就可以用公式2-1计算

出每个矩阵的梯度 $\nabla M^{(i)}$。梯度下降算法采取的步骤是对每个矩阵减去在这个位置的梯度乘上学习率。即

$$M^{(i)} \leftarrow M^{(i)} - \eta \cdot \left(\nabla M^{(i)}\right)^T$$

这里学习率 η 决定了梯度下降每一轮对矩阵大小改变的程度，这可以随着轮数的进行而不断改变。一种比较常用的策略是在刚开始把学习率尽量设的比较大，以便于快速的脱离初始状态进入下降，之后再慢慢减小学习率，更加细化的去找到最优状态。

2.3 损失函数的选择

下面我们介绍几个本文会用到的损失函数。

2.3.1 均方误差

均方误差是最常见的损失函数，适用于神经网络的输出为一个数值的情况。我们将这个数值与训练数据集的数据做差的平方和之后就得到了均方误差，如下式：

$$\sum_{i\in[n]} |f(x_i) - f^*(x_i)|^2$$

2.3.2 交叉熵

上面的均方误差损失函数在面对一些输出并非单一数值的网络时训练效果并不好。典型的例子如分类问题。我们在训练分类问题的网络时往往会有多个输出节点，将输出最大的那个节点编号做为最后的分类结果。在这种情况下，强行使用均方误差的效果并不理想。我们采用交叉熵的方式构造损失函数。

交叉熵来源于度量两个概率分布的距离。假设有两个分布 p, q，定义在离散的概率空间 D 上。交叉熵定义这两个分布的差异为：

$$H(p,q) = \sum_{x\in D} p(x) \cdot \log\left(\frac{1}{q(x)}\right)$$

对于分类问题来说，输出的节点值可以当成是每一种分类的“可能性”，从而构成了一个分布。而训练数据集的标注分布就是一个在正确分类上取 1，其他位置取 0 的分布。我们用交叉熵来度量这两个分布。

严格来说，分类问题的神经网络 f 可以写成 $f = \max_{i\in[m]} f'_i$，其中 m 为可能的类型数量，f' 为 f 的 m 个输出节点，f'_i 为 f' 的第 i 个分量。而交叉熵损失函数定义为

$$\sum_{i\in[n]} -\log\left(f'_{f^*(x_i)}(x_i)\right)$$

优化这个函数可以使输出的分布不断的逼近真实分布，也能达到让 f 不断毕竟 f^* 的效果。

值得一提的是，这里的函数 $f = \max_{i\in[m]} f_i'$ 虽然也是单一输出的函数，但是这里不用均方误差的一个主要原因是我们希望 f 对矩阵的偏导形式简单易得，这有利于梯度下降。而 max 函数对求导并不友好，所以交叉熵损失函数在这种情况下很常用。

2.3.3 无局部最优损失函数

需要注意的是，梯度下降方法优化一个函数很容易陷入局部最优。如图2-5所示。因为梯度下降法在后期主要是在做局部的调整，看不到全局的信息，所以很难进入全局最优解。 而对于前面提到的均方误差、交叉熵这种类

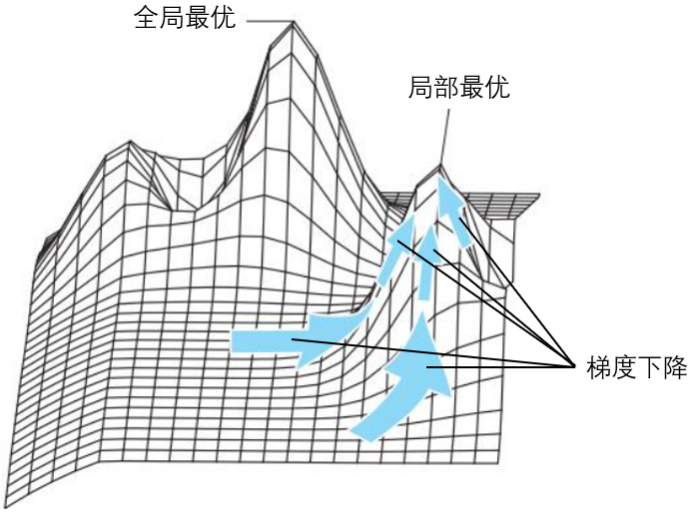


图 2-5: 陷入局部最优

型的损失函数来说，一般存在不少局部最优。甚至有时候梯度下降并不会收敛。因此，初始值的选取影响了最后梯度下降算法的效率，这在下一章会重点讨论。

这里我们介绍一种可以避免局部最优的损失函数，在[10] 中提出。这里我们只考虑最简单的两层、单节点输出的神经网络，即 $f(\boldsymbol{x}) = a\sigma(M\boldsymbol{x})$。令 M_i 表示矩阵 M 的第 i 行向量，则我们可以构造如下函数。

$$loss(M) = \operatorname{sign}(\hat{\sigma}_4)\mathbb{E}\left[y \cdot \sum_{i\neq j}\phi_{i,j}\right] - c_1\operatorname{sign}(\hat{\sigma}_4)\mathbb{E}\left[y \cdot \sum_{i}\psi_i\right] + c_2\sum_{i=1}^{m}(\|M_i\|^2 - 1)^2$$

其中 $\hat{\sigma}_4$ 是 σ 的第 4 个埃尔米特多项式系数（见[11]），c_1, c_2 是常数，并且

$$\phi_{i,j} = \frac{1}{2}\|M_i\|^2\|M_j\|^2 + \langle M_i, M_j\rangle^2 - \frac{1}{2}\|M_j\|^2(M_i\boldsymbol{x})^2 - \frac{1}{2}\|M_i\|^2(M_j\boldsymbol{x})^2 + 2(M_i\boldsymbol{x})(M_j\boldsymbol{x})\langle M_i, M_j\rangle + \frac{1}{2}(M_i\boldsymbol{x})^2(M_j\boldsymbol{x})^2$$

$$\psi_i = \frac{1}{8}\|M_i\|^4 - \frac{1}{4}(M_i\boldsymbol{x})^2\|M_i\|^2 + \frac{1}{24}(M_i\boldsymbol{x})^4$$

另外函数中的期望概率空间为真实函数 f^* 的自变量空间的均匀分布 x，其中 $y = f^*(x)$。注意到这个损失函数只与 M 有关，这是为了分析方便，省去了训练样本集（实际使用时可以用训练样本集来近似表达式中的期望），并且省去了 a 的训练：当训练出 M 之后再用一次简单的线性回归训练 a。对于上面的这个损失函数，我们有如下的好性质。

定理 2-1 ([10]) 假设真实函数 $f^*(x) = a^*\sigma(M^*x)$。若对任意 a 的坐标 a_i 都有 $R \geq a_i \geq L$，其中 R, L 是常数，则存在与 R, L 有关的常数 c_1, c_2 使得

1. $loss(M)$ 的所有局部最优（极小值点）都是全局最优（即所有极小值点值相同）。
2. $loss(M)$ 的最小值点 M' 满足 $M' = DPM^*$，其中 P 是一个排序矩阵，D 是一个对角矩阵，并且 $D_{ii} \in \{\pm 1 \pm O\left(\frac{\mu R}{c_2}\right)\}$。

定理中的第一个条件保证了梯度下降总是会收敛到函数的全局最优。而第二个条件保证这个损失函数的全局最优可以达到对真实函数的一种近似。

这个函数的设计使得在完全未知目标函数的时候，总能通过梯度下降的方法收敛到近似解。但由于函数过于复杂，操作难度大，并且只适用于两层，本身函数的设计也不好对多层网络进行推广，所以实际运用中依然会用均方差或者交叉熵这种简单易于求导，实际效果好的函数。

第三章　初始化的方法与重要性

上一章介绍了神经网络和梯度下降法优化损失函数的基本知识。梯度下降的一开始有不可避免的一个步骤，就是寻找一个起始点，也就是对权重矩阵进行初始化。在2.3.3中我们提到，我们期望梯度下降从起始点开始收敛到一个附近的极小值，但这个附近的极小值有可能是局部最优，甚至可能不收敛，除非使用2.3.3中介绍的无局部最优损失函数。但在一般情况下我们需要考虑均方差和交叉熵类型的损失函数，避免陷入局部最优，此时初始化方法极为重要。

本章主要探究什么样的初始化是不可以使用的，以及什么样的初始化对一般的损失函数能起到好的效果。在3.1中我们将从实验和理论两个角度来探究初始化值的大小对结果的影响，并给出合理的初始化方式；在3.1.3中我们将介绍并推导两种常用的初始化方法：Xavier 初始化和 He 初始化的合理性。

3.1　实验和思考

本节将展示在用卷积神经网络对手写数字的识别中，权值初始化以及在训练过程中权值的变化，并针对实验现象提出理论解释。

3.1.1　网络结构

实验采用一个两层的卷积神经网络对手写数字进行分类学习。对于每一个手写数字的图片输入，大小为 $1\times28\times28$，其中 28×28 表示长宽都为 28 的像素点阵，每一个像素点有一个实数值表示这个像素点上的黑色深度。第一层卷积层对每一个 5×5 的子像素点阵链接到 16 个节点上，并且允许图片外部 2 行补 0。这种卷积结构可以用图3-1来表示：

这样得到的第二层有 $16\times28\times28$ 个节点。可以看成是 28×28 的像素矩阵，每一个像素点由 16 个实数值来描述。为了使训练效率更高，我们采用池化的办法，对每一个 2×2 的子矩阵，去中间的最大值做为输出值，于是得到了一个 $16\times14\times14$ 的第二层输出。第二层的输出进入激励函数 *ReLU*，定义为

$$ReLU(x)=\begin{cases}0 & x\le 0\\ x & x>0\end{cases} \tag{3-1}$$

第三层对第二层的 14×14 像素点进行卷积，每 5×5 的点阵输入到 32 个节点上，和第一层卷积到第二层一样，允许两行补 0，从而得到 $32\times14\times14$ 的第三层节点。再经过一次和第二层一样的池化得到 $32\times7\times7$ 的输出。

最后一层输出层采用全连接的线性输出。整个网络结构如图3-2所示。

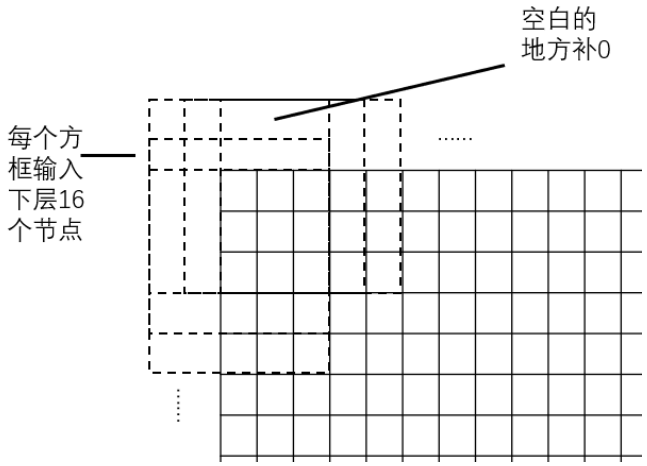


图 3-1: 卷积

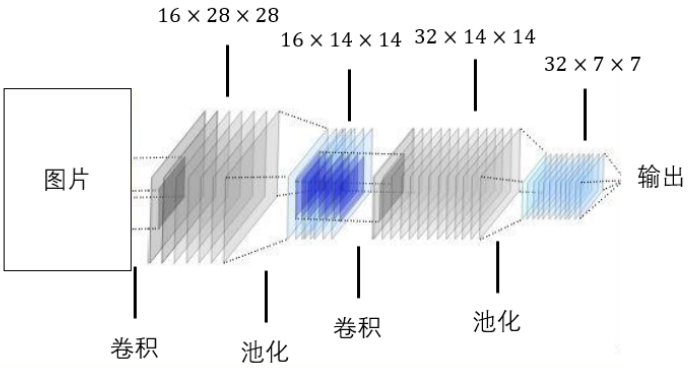


图 3-2: 网络架构

最后的输出是 10 个节点，我们取这 10 个节点中输出最大的节点做为最后预测的图片数字类型。我们用交叉熵来做为损失函数。代码的实现使用 Pytorch 架构，直接使用 minist 数据集。下面我们讨论在不同的权值初始化方法下梯度下降结果的好坏。

3.1.2 单一值初始化

一个最简单方法是把所有的权重初始化为同一个值。而实验结果表明，在权值设置过大的时候往往会获得很坏的结果。比如初始化所有的权重为 100，在第一次训练之后所有权值就会变成上万级别，第二次，第三次之后就会超过计算机的表示范围。这样最后的结果肯定是不能切近测试值的。

我们考虑将所有的权值初始化为 0.1，图 3-3为随着训练次数的增多，两个卷积层矩阵的平均值变化（假设第一个卷积层的矩阵为 M_1，第二个卷积层矩阵为 M_2）。

上图中的情况我们设置学习率为 0.1。记得之前介绍梯度下降是提过学习率 η 是每一次梯度下降减去的梯度前面乘的系数。在这个情况下，初始值为全

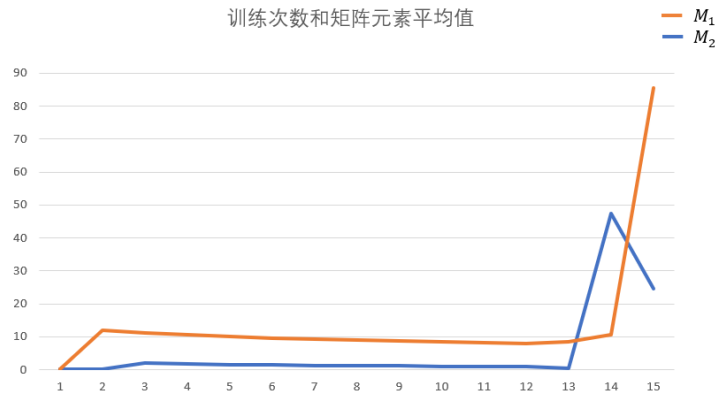


图 3-3: 实验数据

0.1 可以观察到发生了“梯度爆炸”，即矩阵平均值以指数增长的方式不断趋于无穷大。在第 14 次梯度下降的时候，两个矩阵的值都迅速上升到了几十，在第 15 次之后就会成为上万级别，然后超出计算机的表示范围。

在初始值为 0 的时候会达到很好的训练结果，而在初始值为 0.1 的时候会梯度爆炸。很自然的问题是这中间的分界线在哪？为了大致衡量出这其中分界线的位置，我们以 10 倍做为一个划分，分别尝试了初始值为 $1, 0.1, 0.01, 0.001$ 的训练结果，如图 3-4。

初始值	正确率
1	nan
1/10	nan
1/100	0.8
1/1000	1.0

图 3-4: 实验数据

其中 nan 表示发生了梯度爆炸。可以看出，真实的分界线大概在 0.01 到 0.1 之间。为了进一步的观察这个过程中发生了什么，我们将这个区间进一步细化。考虑把 $1/x$ 中的 x 从 10 到 100 进行变化。下面这张图 3-5展示的是发生改变的这一段。 可以看出分界点在 $\frac{1}{20}$ 这个位置附近。另外我们也通过观测 M_1 的 M_2 的平均值来进一步探究初始值是如何整个过程的。可以看出当初始值小于 1/20 的时候，梯度爆炸不会发生，但是训练结果变的很坏。从数据中可以看出两个矩阵在 $\frac{1}{21}$ 这个训练结果好的地方是相差很大的：M_2 比 M_1 要小很多，而其他地方都相差很小。我们在后一节分析中会分析这种现象。从实验中我们可以得出的结论是，单一初始值有一个分界点，正好落在这个分界点时训练结

初始值	成功率	M_1平均	M_2平均
1/19	nan	nan	nan
1/20	nan	nan	nan
1/21	0.8	-1.82	0.017
1/22	0.2	0.043	0.046
1/23	0.3	0.043	0.043
1/24	0.2	0.042	0.041
1/25	0.1	0.040	0.039
1/26	0.2	0.038	0.038
1/27	0.2	0.037	0.037

图 3-5: 实验数据

果最好，超过这个分界点会造成梯度爆炸，而低于这个分界点会使得训练结果变的很差。

3.1.3 随机初始化

在上一节我们采用的初始化方案都是基于单一值。而交叉熵和均方差误差函数一般不可避免的会存在不少的局部最优，在收敛的时候也不一定能得到最优解，比如在上一节的实验中初始值小于分割值的位置，就是陷入了局部最优解中。

我们考虑一种随机初始化，每一条边的权重采用均匀分布。根据上一节的讨论，我们需要让正态分布的边权尽量集中在 $\frac{1}{n}$ 附近。我们用实验来验证这一猜想。依然采用与上一节相同的网络架构，我们通过改变均匀分布的取值范围来产生不同的初始值分布进行训练。假设均匀分布的取值范围为 $[-1/k, 1/k]$，对于不同的 k 值我们可以得到如图3-6的结果。

K值	正确率	M_1平均	M_2平均
6	nan	nan	nan
7	nan	nan	nan
8	1.0	0.017	-0.0054
9	1.0	0.029	-0.0046
10	1.0	0.024	-0.0039

图 3-6: 实验数据

由于均匀分布的期望是 $\frac{1}{k}$ 的一半，所以可以看出这里的分界线是原先的两倍。值得注意的是加入随机化之后，训练的正确率一直稳定在 1。从这个实验中可以看出随机化确实能大大减小落入局部最优的情况。

3.1.4 梯度爆炸原因分析

回忆之前写的梯度下降的梯度表达式，如下

$$\nabla M^{(i)} = \nabla \boldsymbol{y}^{(\boldsymbol{i+1})} \cdot \boldsymbol{y}^{(\boldsymbol{i})}$$

$$\nabla \boldsymbol{y}^{(\boldsymbol{i})} = M^{(i)} \cdot \nabla \boldsymbol{y}^{(\boldsymbol{i+1})}$$

这里省略了对激励函数的求导，因为 ReLU 的求导只有可能是 1 是 0，为了研究梯度爆炸的原因这里我们假设导数为 1。可以看出在反向传播的过程中，梯度都会乘上一个 $\boldsymbol{y}^{(\boldsymbol{i})}$ 或者 $M^{(i)}$。如果这两项过大，都会导致一次梯度下降对矩阵值的加减过多。而加减过多又会导致矩阵值更大，这样不断恶性循环造成了梯度爆炸。

具体来说，我们来看相邻两层的矩阵是如何互相影响的，我们有如下的式子

$$\nabla M^{(i)} = \nabla \boldsymbol{y}^{(\boldsymbol{i+1})} \cdot \boldsymbol{y}^{(\boldsymbol{i})} \qquad \nabla \boldsymbol{y}^{(\boldsymbol{i+1})} = M^{(i+1)} \cdot \nabla \boldsymbol{y}^{(\boldsymbol{i+2})}$$

$$\Longrightarrow \triangle M^{(i)} = M^{(i+1)} \cdot \nabla \boldsymbol{y}^{(\boldsymbol{i+2})} \cdot \boldsymbol{y}^{(\boldsymbol{i})} \tag{3-2}$$

$$\nabla M^{(i+1)} = \nabla \boldsymbol{y}^{(\boldsymbol{i+2})} \cdot \boldsymbol{y}^{(\boldsymbol{i+1})} \qquad \boldsymbol{y}^{(\boldsymbol{i+1})} = \boldsymbol{y}^{(\boldsymbol{i})} \cdot M^{(i)}$$

$$\Longrightarrow \nabla M^{(i+1)} = \nabla \boldsymbol{y}^{(\boldsymbol{i+2})} \cdot \boldsymbol{y}^{(\boldsymbol{i})} \cdot M^{(i)} \tag{3-3}$$

可以看出，$M^{(i)}$ 通过式3-3影响到 $\nabla M^{(i+1)}$ 的大小，进而影响到下一次 $M^{(i+1)}$ 的大小，从而通过式3-2影响到 $\nabla M^{(i)}$ 的大小，从而又影响了下一次的 $M^{(i)}$。如果我们假定 $\nabla M_{old}^{(i+1)} \approx M^{(i+1)}$（实际上 $M^{(i+1)} = M_{old}^{(i+1)} - \eta \cdot \left(\nabla M_{old}^{(i+1)}\right)^T$，这里我们假定 $\eta \cdot \left(\nabla M_{old}^{(i+1)}\right)^T$ 压过了 $M_{old}^{(i+1)}$），则通过合并3-3和3-2可以得到

$$\triangle M^{(i)} \approx \nabla \boldsymbol{y}^{(\boldsymbol{i+2})}{}_{old} \cdot \boldsymbol{y}^{(\boldsymbol{i})}{}_{old} \cdot M_{old}^{(i)} \cdot \nabla \boldsymbol{y}^{(\boldsymbol{i+2})} \cdot \boldsymbol{y}^{(\boldsymbol{i})}$$

如果我们再假定 $\nabla M_{new}^{(i)} \approx M^{(i)}$（实际上 $M_{new}^{(i)} = M^{(i)} - \eta \cdot \left(\nabla M^{(i)}\right)^T$，同样我们假定 $\eta \cdot \left(\nabla M^{(i)}\right)^T$ 压过了 $M_{new}^{(i)}$）则我们得到

$$M_{new}^{(i)} \approx \nabla \boldsymbol{y}^{(\boldsymbol{i+2})}{}_{old} \cdot \boldsymbol{y}^{(\boldsymbol{i})}{}_{old} \cdot M_{old}^{(i)} \cdot \nabla \boldsymbol{y}^{(\boldsymbol{i+2})} \cdot \boldsymbol{y}^{(\boldsymbol{i})}$$

可以看到，从 $M_{old}^{(i)}$ 到 $M_{new}^{(i)}$ 经过了两次梯度下降，把整个值乘上了两次第 $i+2$ 输出层的梯度和两次第 i 输出层的值。

如果第 $i+1$ 层本身就是最后一层（也就是说不存在第 $i+2$ 层），我们也可

以将3-2代入3-3中，得到一个类似的关系式

$$M_{new}^{(i)} \approx \nabla \boldsymbol{y}^{(i+1)} \cdot \boldsymbol{y}^{(i-1)} \cdot M_{old}^{(i)} \cdot \nabla \boldsymbol{y}^{(i+1)}{}_{old} \cdot \boldsymbol{y}^{(i-1)}{}_{old}$$

可以看出，无论如何都会和附近两层的中间层节点的输出和梯度有关。而如果中间层节点的输出以及梯度大于 1，将在不断的梯度下降中对矩阵的值产生一个指数爆炸。所以我们实际上希望每一个输出层的值和梯度都近似为 1。

注意到每一层的输出和梯度满足如下递归式：

$$\boldsymbol{y}^{(k)} = \boldsymbol{x} \prod_{i=0}^{k-1} M^{(i)}$$

$$\nabla \boldsymbol{y}^{(k)} = \left(\prod_{i=k}^{d-1} M^{i} \right) \cdot \nabla \boldsymbol{y}^{(d)}$$

我们一般认为输入 $\boldsymbol{x}$ 和最后一层的梯度 $\nabla \boldsymbol{y}^{(d)}$ 的每一个元素都被归一化为 1 附近。根据上面两式的迭代过程，如果要中间每一层的元素都和 $\boldsymbol{x}$ 以及 $\nabla \boldsymbol{y}^{(d)}$ 一样被归一化到 1 附近，我们实际上希望矩阵的每一行、每一列值的和都近似为 1。

综上，如果一个矩阵为 $d_i \times d_{i+1}$ 大小的矩阵，即这个矩阵在第 i 个输出层和第 $i+1$ 个输出层之间，则我们可以考虑将这个矩阵中间每个值初始化到 $\frac{1}{d_i+d_{i+1}}$ 附近：这和我们之前实验中的分界点非常接近。这一思想也和我们后面要介绍的 Xavier 初始化非常像。

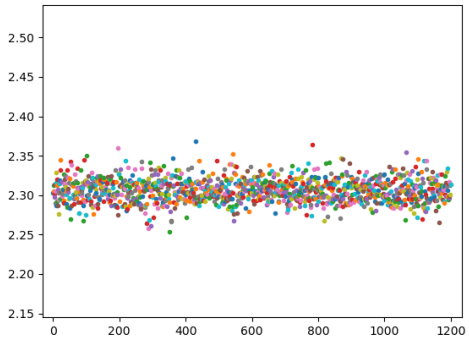
3.2 缩小学习率抵消梯度爆炸

在上一节我们提到，我们需要让矩阵的每一行、每一列的值的和都近似为 1 才能防止让每一次的矩阵梯度 ∇M 指数增长。但梯度下降每一次对矩阵做出的改变值是 $\eta \nabla M$。一个值得思考的问题是，如果通过改变 η 的值来对 ∇M 进行适当的调整，使得 $\|\nabla M\|_2$ 维持在一个合理的范围，既不会太大也不会太小，那么训练结果会如何？下面我们就通过实验来探究这一问题。

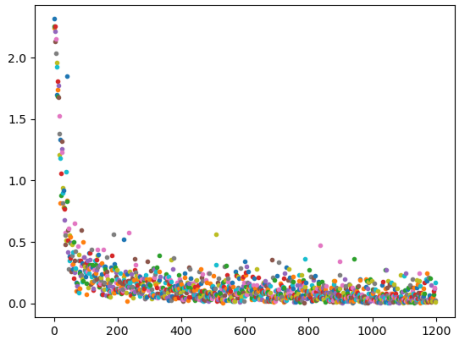
我们的实验依然使用之前的数字图片网络架构。根据表3-6中的数据，我们可以看到：在 $\eta = 0.1$ 的时候，我们将矩阵初始值均匀分布在 $[-0.1, 0, 1]$ 之间，可以达到一个很好的训练效果。由于梯度下降对矩阵的改变是 $\eta \nabla M$，我们考虑将初始值均匀分布在 $\left[-\frac{1}{100\eta}, \frac{1}{100\eta}\right]$ 之间（这样可以保证 $\frac{1}{100\eta} \cdot \eta = 0.1 \cdot 0.1$）。我们通过设置 η 为 $1, 0.1, 0.01, 0.001$ 来分别看损失函数的收敛速度，如下面四张图表示。横坐标为梯度下降的训练次数，纵坐标为损失函数的值。

四次训练最后在 10 个测试集上的正确率如表3-9所示。

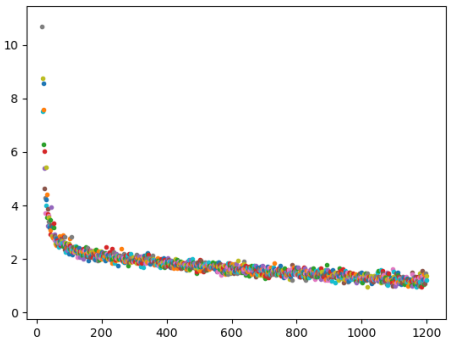
结合上面的图标可以发现，即使我们通过调配 η 和初始值的取值范围使得梯度爆炸不发生，不同的 η 值依然对应这不同的训练结果。在 $\eta = 0.1$ 的时候，训练结果最好，此时的矩阵初始值基本位于 $\frac{1}{d_i+d_{i+1}}$ 附近。而在 $\eta = 1$ 的时候，我



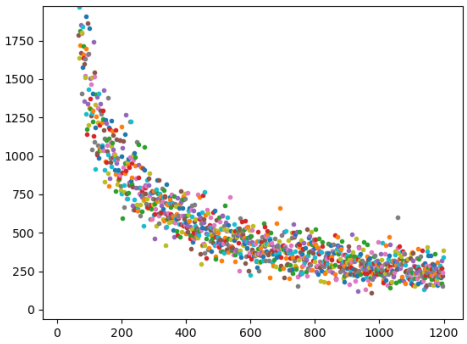
((a)) $\eta = 1$



((b)) $\eta = 0.1$



((a)) $\eta = 0.01$



((b)) $\eta = 0.001$

们矩阵的初始值位于 $\frac{0.1}{d_i+d_{i+1}}$ 附近（非常小），此时的损失函数完全不收敛（如图 4-2所示），训练效果也最差。而在 $\eta = 0.01, 0.001$ 时，虽然有收敛，但是收敛的速度都随着 η 的减小而减缓。这是合理的：η 决定了在接近全局最优时的步长，而 $\eta = 0.01, 0.001$ 时步长过小，从而导致没有完全收敛到全局最优。而训练结果也是随着 η 的减小而变差（分布为 $0.7, 0.6$）。

从这个实验中我们可以看出，虽然通过调配 η 来处理随机初始化权重过大或者过小的情况虽然理论上可行，但是实验表明效果会随着 η 的变化而产生较大波动。其主要原因是 η 本身代表的学习率就应该有一个合适的取值（比如

η值	正确率
1	0.1
0.1	1.0
0.01	0.7
0.001	0.6

图 3-9: 不同 η 值对应的正确率

0.1）以达到一个合适的步长；如果我们不把初始值设置为 $\frac{1}{d_i+d_{i+1}}$ 附近，而通过改变 η 的方式开控制矩阵梯度的大小，改变的 η 本身也会影响训练的效果。

3.3 相关工作

下面介绍两种常用的随机初始化方法。

3.3.1 Xavier 初始化

Xavier 初始化采用的是和我们上一节的随机初始化方案类似的均匀分布。在上一节我们探究出均匀分布的范围应该在 $\left[-\frac{1}{n}, \frac{1}{n}\right]$ 附近。而在论文[3] 中，这一思想被进一步严格化。

[3] 主要考虑的是在随机化输入值、权值初始值的情况下每一层节点输出的方差，使这些方差都一样。具体来说，注意到我们的网络结构满足

$$\boldsymbol{y}^{(i+1)} = \sigma_i\left(M^{(i)} \cdot \boldsymbol{y}^{(i)}{}_l\right)$$

$$\nabla \boldsymbol{y}^{(i)} = M^{(i)} \cdot \nabla \boldsymbol{y}^{(i+1)}$$

$$\nabla M^{(i)} = \nabla \boldsymbol{y}^{(i+1)} \cdot \boldsymbol{y}^{(i)}$$

注意到这里我们省略了激励函数的导数。我们默认激励函数的导数取得 1（实际上 ReLU 激励函数也会取到 0，而这确实会干扰 Xavier 初始化的效果，这一点在下一节介绍 He 初始化的时候会提到）。现在我们将输入变量 $x = \boldsymbol{y}^{(0)}$，权值矩阵 $W^{(i)}$，以及输出层的梯度 $\nabla y^{(d)}$ 当成随机变量的情况下，来计算每一层 $\boldsymbol{y}^{(i)}$ 和 $\nabla \boldsymbol{y}^{(i)}$ 的方差。这两个量决定了 $\nabla M^{(i)}$ 的值。注意到对第 $i+1$ 层的第 r 个节点 $\boldsymbol{y}^{(i+1)}{}_r$ 我们有

$$Var\left(\boldsymbol{y}^{(i+1)}{}_r\right) = Var\left(\sum_l \boldsymbol{y}^{(i)}{}_l \cdot M_{l,r}^{(i)}\right) = \sum_l Var\left(\boldsymbol{y}^{(i)}{}_l \cdot M_{l,r}^{(i)}\right)$$

这里我们假设权值初始化和输入是独立的，并且 Xavier 初始化总是假设 $\mathbb{E}[\boldsymbol{y}^{(i)}{}_l] = \mathbb{E}[M_{l,r}^{(i)}] = 0$，并且每一条边的初始化都是独立的，这样 $\boldsymbol{y}^{(i)}{}_l$ 和 $M_{l,r}^{(i)}$ 是独立的，从而我们得到

$$Var\left(\boldsymbol{y}^{(i+1)}\right) = Var\left(\boldsymbol{y}^{(i)}\right) \cdot Var(M^{(i)})$$

这里 $Var\left(\boldsymbol{y}^{(i)}\right)$ 和 $Var(M^{(i)})$ 都表示对矩阵的每一个元素求方差。同理我们可以得到

$$Var\left(\nabla \boldsymbol{y}^{(i)}\right) = Var\left(M^{(i)}\right) \cdot Var\left(\nabla \boldsymbol{y}^{(i+1)}\right)$$

从而我们可以得到两个递推式，根据这两个递推式可以得到

$$Var\left(\boldsymbol{y}^{(i+1)}\right) = Var(\boldsymbol{x}) \cdot \prod_{j=0}^{i} Var(M^{(j)})$$

以及

$$Var\left(\nabla \boldsymbol{y}^{(i)}\right) = \prod_{i+1}^{j=d-1} Var\left(M^{(j)}\right) \cdot Var\left(\nabla \boldsymbol{y}^{(d)}\right)$$

很自然的想到，如果要让每一层的方差保持稳定，我们必须要这个乘积的每一项都等于 1。由此可知，$M^{(j)}$ 的每行每列的方差和应该为 1。为了方便起见，我们再假设两层中间的所有边权初始化方式相同，则我们得到

$$Var\left(M_{l,r}^{(j)}\right) \approx \frac{1}{n_j}$$

$$Var\left(M_{l,r}^{(j)}\right) \approx \frac{1}{n_{j+1}}$$

于是我们规定

$$Var\left(M_{l,r}^{(j)}\right) = \frac{2}{n_j + n_{j+1}}$$

由于一开始我们考虑的是均匀分布，而在 $[a,b]$ 上的均匀分布得到的方差是 $\frac{(a-b)^2}{12}$，从而我们令 $M_{l,r}^{(j)}$ 的初始化值为 $\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right]$，这就是 Xavier 初始化。

这里解释一下为什么需要让每一层输出的方差都为 1。这是由于在每一次梯度下降的训练过程中，方差都会以乘积的形式乘到下一次的方差上，从而造成指数爆炸。如果方差大于 1，很容易造成之前实验中出现的突然超过计算机最大表示整数的形式。而方差小于 1 时，可以从之前的实验中看到，最后的训练结果基本在初始值上没有什么改动，这很容易陷入局部最优，不能得到一个很好的最优解。

我们用 Xavier 初始化在之前的网络上，表现出的训练正确率全部为 1。

3.3.2 He 初始化

在论文[4] 中，作者在 Xavier 初始化的基础上提出了一个优化的初始化方法，称为 He 初始化。He 初始化对之前 Xavier 初始化中计算出的方差都除了 2，用以处理激励函数为 ReLU 的情况。ReLU 的求导得到的值要么是 0，要么是 1。还记得我们在之前分析 Xavier 初始化中省略了激励函数的求导，在 He 初始化中我们不省略这个导数，也就是说大量的输入值会变成 0，同时每一层节点的期望也不再是 0：小于 0 的会被 ReLU 变为 0。基于这样的假设我们在做同样的分析。

我们依然有同之前分析一样的对于每一层输出和梯度的递推关系式，不同

的是在计算方差时，我们有

$$Var\left(\hat{\boldsymbol{y}}^{(\boldsymbol{i+1})}\right) = Var\left(\boldsymbol{y}^{(\boldsymbol{i})} \cdot M^{(i)}\right) = \mathbb{E}\left[\left(\boldsymbol{y}^{(\boldsymbol{i})}\right)^2 \cdot \left(M^{(i)}\right)^2\right] - \mathbb{E}\left[\boldsymbol{y}^{(\boldsymbol{i})} \cdot M^{(i)}\right]^2$$

这里 $\hat{\boldsymbol{y}}^{(\boldsymbol{i+1})}$ 是未经过激励函数时的输出。我们对权值的初始化总是令期望为 0，所以我们可以假设 $mathbbE\left[M^{(i)}\right] = 0$，另外权值的初始化也是和输入独立的，从而我们得到

$$Var\left(\boldsymbol{y}^{(\boldsymbol{i+1})}\right) = \mathbb{E}\left[\left(\boldsymbol{y}^{(\boldsymbol{i})}\right)^2\right] \cdot \mathbb{E}\left[\left(M^{(i)}\right)^2\right] = \mathbb{E}\left[\left(\boldsymbol{y}^{(\boldsymbol{i})}\right)^2\right] \cdot Var\left(M^{(i)}\right)$$

记得我们令 $\boldsymbol{y}^{(\boldsymbol{i})} = \sigma_i\left(\hat{y}^{(i)}\right) = \sigma_i\left(y^{(i-1)} \cdot M^{(i-1)}\right)$，其中 σ_i 是 *ReLU* 函数，在自变量小于 0 的时候变为 0。由于 $M^{(i-1)}$ 满足一个期望为 0 的随机分布，并且我们假设这个分布关于零点对称，则乘上 $y^{(i-1)}$ 之后，$\hat{y}^{(i)}$ 依然是一个期望为 0、关于零点对称的分布。于是我们得到

$$\mathbb{E}\left[\left(\boldsymbol{y}^{(\boldsymbol{i})}\right)^2\right] = \int_0^{\infty} a^2 \Pr\left[\boldsymbol{y}^{(\boldsymbol{i})} = a\right] da = \frac{1}{2}\int_{-\infty}^{\infty} a^2 \Pr\left[\hat{\boldsymbol{y}}^{(\boldsymbol{i})} = a\right] da = \frac{1}{2} Var\left(\hat{\boldsymbol{y}}^{(\boldsymbol{i})}\right)$$

所以我们得到的递推式是

$$Var\left(\boldsymbol{y}^{(\boldsymbol{i+1})}\right) = \frac{1}{2} Var\left(\hat{\boldsymbol{y}}^{(\boldsymbol{i})}\right) \cdot Var\left(M^{(i)}\right)$$

这个递推式由于 ReLU 的关系，和之前相比缩小了 $\frac{1}{2}$。所以相应的，我们需要将 $Var\left(M^{(i)}\right)$ 扩大为原来的 $\frac{1}{2}$ 以达到同样的效果。

同理，对于反向传播的梯度我们也有 $\frac{1}{2}$ 的缩小。综上，我们希望初始化的值满足

$$Var\left(M_{l,r}^{(j)}\right) = \frac{4}{n_j + n_{j+1}}$$

在[4] 的原文中，作者是认为 n_{j+1} 和 n_j 不会相差太多，并且认为高斯分布（正态分布）更有效，所以原文中将 $M_{l,r}^{(j)}$ 初始化为方差为 $\frac{2}{n_j}$，期望为 0 的正态分布。当然我们也可以令方差为 $\frac{4}{n_j+n_{j+1}}$，可能会达到更好的效果。

用同样的实验，加上 He 初始化之后的结果表明，无论方差是 $\frac{2}{n_j}$ 还是 $\frac{4}{n_j+n_{j+1}}$，无论是用正态分布还是均匀分布，都能达到 1 的正确率。从中可以看出进行一段范围内的期望为 0 的随机初始化、合理控制方差在一个范围内，能对梯度下降的结果产生非常积极的影响。

第四章　初始化值与训练结果值的关系

本章将探究初始化的值与训练出来的结果矩阵的值有怎样的关系。研究灵感来自于上一章在探究初始化值的大小时，曾观察到训练出来的结果与初始值差别不大的情况。本章旨在进一步探究这一现象。先通过实验找到现象出现的条件，再用理论分析解释现象，最后引用一些对这个现象进行的前沿的研究。

4.1　实验

为了方便起见，这一次实验考虑两层隐藏层的全连接网络，使用 ReLU 函数。即网络的数学表达式为 $y = \sigma_2(\sigma_1(\boldsymbol{x}M_1)M_2)\cdot a$。其中 $\boldsymbol{x}$ 为一个长度为 1 的行向量；M_1 为一个 $1\times m$ 矩阵，M_2 为一个 $m\times m$ 的矩阵；m 为隐藏层的节点数目；σ_1,σ_2 为 ReLU 函数；a 为一个长度为 m 的列向量，表示隐藏层到输出节点的边权（我们只考虑只有一个输出节点的情况）。网络结构在 $m = 7$ 时如图 4-1所示。

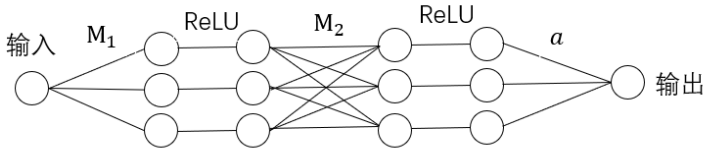


图 4-1: 网络结构

这次实验我们考虑训练一个函数 $y = x^2$。训练的数据采用从 -1 到 1 之间均匀采样的 100 个数据点 $(x_i, x_i^2 + \epsilon_i)$，其中 ϵ_i 是从 -0.1 到 0.1 中采样的均匀分布，做为数据点的扰动。最后我们的测试数据是从 -1 到 1 之间随机采样的一系列 x，我们计算网络训练结果值 $net(x)$ 和 x^2 之间的差距。损失函数我们采用均方误差。权值初始化的方法我们采用之前讨论过的随机初始化，即按照方差为 $\frac{4}{n_i+n_{i+1}}$ 设置第 i 层矩阵值的方差，并按照正态分布进行初始化。

我们不断改变 m 的值，通过 100 次训练来观察损失函数的值以及矩阵值在训练中的变化。具体来说，我们监控以下两个量：

1、假设测试集为 $S = (x_i, x_i^2)_{i\in[10]}$，我们观察 100 次梯度下降，每一次结束之后 $\frac{\sum_{i\in[10]}(net(x_i)-x_i^2)^2}{10}$，其中 $net(x_i)$ 是结束权值更新之后网络对 x_i 的输出。这个值在下面的训练数据中被称作“损失值”。

2、每一次训练后矩阵和原来矩阵差的方差，即

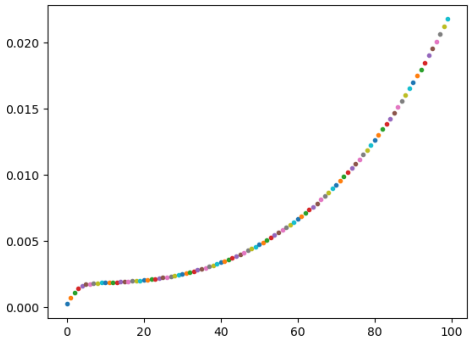
$$||M_1 - M_1^{init}||_2$$

$$||M_2 - M_2^{init}||_2$$

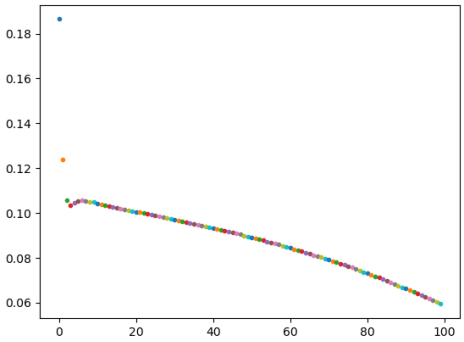
$$\|a - a^{init}\|_2$$

这里 $\|\cdot\|_2$ 是对矩阵（或向量）的每一个元素的平方求平均。由于我们使用的是方差为 $\frac{4}{n_i+n_{i+1}}$ 的初始化方案，所以在初始化的时候 M 和 a 并不差太多（一个是 $\frac{4}{n+m}$，一个是 $\frac{4}{m+1}$，但一般 m 远大于 n），所以在训练时一般两个也不会差太远。在下面的实验数据中，“矩阵差”代表 $\|M_1 - M_1^{init}\|_2 + \|M_2 - M_2^{init}\|_2 + \|a - a^{init}\|_2$。

下面我们给出 m 为 10 时的图 4-2，m 为 50 时的图 4-3，m 为 100 时的图 4-4。这三个值分别是 n 的 10 倍，50 倍，100 倍，具有很好的区分度。梯度下降的学习率设置为 0.1。

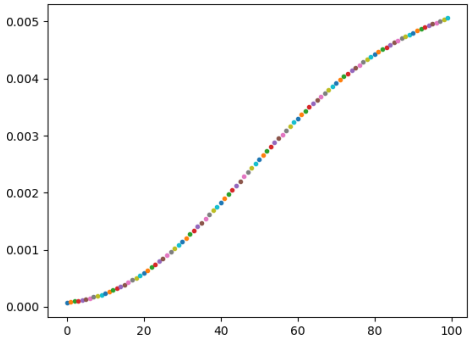


((a)) 矩阵差

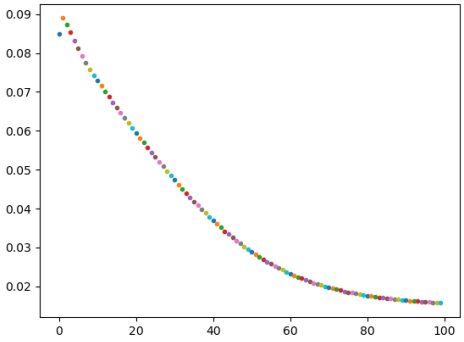


((b)) 损失值

图 4-2: m=10



((a)) 矩阵差



((b)) 损失值

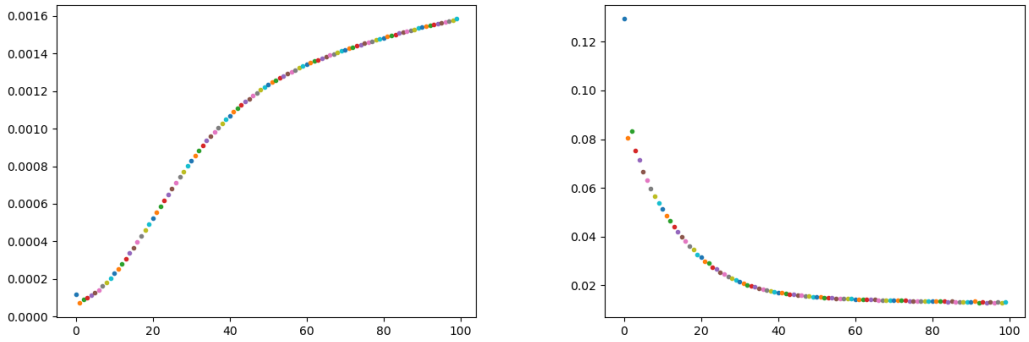
图 4-3: m=50

通过观察这两个值的变化，我们可以得到以下两个个结论：

1、随着梯度下降训练的进行，矩阵差会越来越大，但是随着 m 的增大，整体的差会越来越小。

2、随着 m 的增大，损失值收敛到 0 的收敛速度越来越快。

也就是说，隐藏层相对于输入的 n 越大，训练效果越好，并且训练过程中矩阵对初始值的改变也会越来越小。



((a)) 矩阵差 ((b)) 损失值

图 4-4: m=100

可以看出，在 m 为 10，50，100 时，矩阵差最高能达到的位置分别在 0.2，0.05，0.0016，非常类似于一个正比例的分布。我们将在下一节从理论上分析这个结果。

4.2 分析

我们先计算第一层矩阵 M 和输出层 a 的梯度。我们记隐藏层的输出和最后一层的输出分别为 $\boldsymbol{y}^{(1)}$ 和 $y^{(2)}$。这里 $y^{(2)}$ 是一个数而不是一个向量，这是由于网络只有一个输出。我们记损失函数为 $L = (y - y^{(2)})^2$，这里为了方便起见假设只有一个训练数据 (x, y)。根据之前推导的公式，有

$$\nabla a = \boldsymbol{y}^{(1)} \cdot \nabla y^{(2)}$$

这里 $\nabla y^{(2)}$ 是损失函数对输出的偏导，为 $2(y - \boldsymbol{y}^{(2)})$，$\boldsymbol{y}^{(1)}$ 是隐藏层输出，于是

$$\nabla a = \sigma(x \cdot M) \cdot 2(y - y^{(2)})$$

可以看出 a 的梯度大小主要和输出和真实值的关系、矩阵 M 的大小有关。而同样的，我们可以得到 M 的梯度

$$\nabla M = \nabla \boldsymbol{y}^{(1)} \cdot x = a \cdot \nabla y \cdot x = a \cdot x \cdot 2(y - y^{(2)})$$

这里我们省略了 σ 的导数，这是由于 ReLU 的导数只有 0 和 1 这两种。可以看出 M 的梯度与 a 的大小以及输出和真实值的关系有关。结合上面 a 的梯度，可以看出两个矩阵互相影响。同时 $(y - y^{(2)})$ 这个值会随着损失函数的收敛而收敛，而从上面的实验图可以看出损失函数是呈指数型衰减的。这启发我们用归纳法导出如下的结论。

定理 4-1 记第 i 次训练结束之后的隐藏层矩阵、输出向量为 $M^{(i)}$ 和 $a^{(i)}$，

其中 $M^{(0)}$ 和 $a^{(0)}$ 为初始值，满足 $\max(\|x\|_2, \|M^{(0)}\|_2, \|a^{(0)}\|_2) \leq \gamma \leq 1$。记第 i 次训练结束后对 x 的输出为 $y(i) = \sigma\left(x \cdot M^{(i)}\right) \cdot a$。再假设梯度下降的收敛性满足 $e_i \triangleq y(i) - y \leq \frac{1}{2^i}$。假设梯度下降的学习率满足 $\eta \leq 0.01$，则我们有 $\|M^{(i)} - M^{(0)}\|_2 \leq 8\eta\gamma, \|a^{(i)} - a^{(0)}\|_2 \leq 8\eta\gamma$ 对任意的 $i \geq 0$。

证明: 根据上面的讨论，我们有

$$\begin{aligned}\|M^{(i+1)} - M^{(i)}\|_2 &= \eta\|\nabla M^{(i)}\|_2 \\ &\leq \eta\|a^{(i)}\|_2 \cdot \|x\|_2 \cdot 2e_i \\ &\leq \eta\|a^{(i)}\|_2 \cdot \frac{\gamma}{2^{i-1}}\end{aligned}$$

于是我们有

$$\|M^{(i)} - M^{(0)}\|_2 \leq \sum_{j=0} \|M^{(j+1)} - M^{(j)}\|_2 \leq \eta \sum_{j=0}^{i-1} \|a^{(j)}\|_2 \cdot \frac{\gamma}{2^{j-1}} \tag{4-1}$$

而我们可以同时对 $a^{(i)}$ 进行分析。根据上文对 $a^{(i)}$ 的讨论，我们有

$$\begin{aligned}\|a^{(i+1)} - a^{(i)}\|_2 &= \eta\|\nabla a^{(i)}\|_2 \\ &\leq \eta\|M^{(i)}\|_2 \cdot \|x\|_2 \cdot 2e_i \\ &\leq \eta\|M^{(i)}\|_2 \cdot \frac{\gamma}{2^{i-1}}\end{aligned}$$

于是我们有

$$\begin{aligned}\|a^{(i)}\|_2 &\leq \|a^{(0)}\|_2 + \eta \sum_{j=0}^{i-1} \|a^{(j+1)} - a^{(j)}\|_2 \\ &\leq \gamma + \eta \sum_{j=0}^{i-1} \|M^{(j)}\|_2 \cdot \frac{\gamma}{2^{j-1}}\end{aligned}$$

带入式4-1，得到

$$\begin{aligned}\|M^{(i)} - M^{(0)}\|_2 &\leq \eta \sum_{j=0}^{i-1} \left(\gamma + \eta \sum_{k=0}^{j-1} \|M^{(k)}\|_2 \cdot \frac{\gamma}{2^{j-1}}\right) \cdot \frac{\gamma}{2^{j-1}} \\ &\leq \eta \left(\sum_{j=0}^{i-1} \frac{\gamma^2}{2^{j-1}} + \sum_{0 \leq k < j \leq i-1} \|M^{(k)}\|_2 \cdot \frac{\gamma}{2^{k+j-2}}\right) \\ &\leq 4\eta \left(\gamma^2 + \gamma \sum_{k=0}^{i-2} \|M^{(k)}\|_2 \cdot \frac{1}{4^k}\right)\end{aligned}$$

我们提出归纳假设 $\|M^{(i)} - M^{(0)}\|_2 \leq \gamma + 8\eta$，下面我们用归纳法来证明这个假设。

在 $i = 0$ 的时候结论显然成立。当 $i > 0$ 时，由上面的递归式我们得到

$$\|M^{(i)} - M^{(0)}\|_2 \leq 4\eta\left(\gamma^2 + \gamma \cdot (\gamma + 8\eta) \cdot \frac{3}{2}\right) \leq 8\eta\gamma$$

最后一个不等号是由于 $\eta \leq 0.01$ 并且 $\gamma \leq 1$。

于是我们得到了关于 $M^{(i)}$ 的结论，我们可以导出关于 $a^{(i)}$ 的结论

$$\begin{aligned}\|a^{(i)} - a^{(0)}\|_2 &\leq \eta \sum_{j=0}^{i-1} \|M^{(j)}\|_2 \cdot \frac{\gamma}{2^{j-1}} \\ &\leq \eta \sum_{j=0}^{i-1} (\gamma + 8\eta) \cdot \frac{\gamma}{2^{j-1}} \\ &\leq 8\eta\gamma\end{aligned}$$

□

注意到在隐藏层节点增多的时候，由于我们是根据 $\frac{1}{n_i+n_{i+1}}$ 进行的初始化，相应的 γ 的值也会减少，从而 $8\eta\gamma$ 这个上界也会变小，这就完美解释了我们上面的实验结果。

4.3 相关工作

在隐藏层节点数目远大于输入节点数目的时候，训练对单个参数的影响往往微乎其微，但是因为整体的矩阵较大，点数较多，在矩阵单个值改变不大的情况下对输入的会有较大的影响。这个现象被称为过参数化（over-parameterized）。在过参数化的情况下，根据之前的实验和理论分析，我们可以发现两个现象：1、对均方损失函数的梯度下降可以快速收敛。2、结果的矩阵训练值和对矩阵的初始化差距不太大。

这一现象在论文[7]，[6] 都有深入的研究。在论文[6] 中，作者对两层的神经网络进行了分析，并证明了全连接神经网络在中间影藏层节点数目大概为 $\Omega(n^6)$ 时（n 为输入节点数目），对均方误差损失函数进行梯度下降总是可以指数的收敛。而在论文[6] 中，这一现象被推广到了更多层的神经网络：作者证明了在点数大概为 $\Omega(n^4 \cdot 2^d)$ 时（其中 d 为全连接神经网络的深度），可以有指数收敛。可以看到这个值和 d 呈指数关系，在深度神经网络中效果不好。为了避免这个现象，作者同时考虑了残差神经网络，在残差神经网络中证明了和 d 呈线性的结论 $\Omega(n^4 \cdot 2^d)$，从而得到了一个能应用在深度神经网络中的结论。

第五章　总结与讨论

权值初始化对梯度下降的结果的好坏有很大的影响，只有精心设计的初始化才能使梯度下降快速收敛到全局最优，而坏的初始化会直接导致训练的失败。

在初始化的方法上，我们一般使用随机初始化（均匀分布，正态分布等）在避免陷入局部最优；在初始化值的大小设置上，我们一般设置矩阵的值使得矩阵的行、列的和与 1 相差不大，因为矩阵会通过正向传播、反向传播不断在神经网络中往返迭代相乘，超过 1 很多的情况下会造成指数爆炸，从而使神经网络不收敛；而如果初始化值过小又会造成梯度消失：即训练对矩阵值的改变不大，从而陷入局部最优，同样不能达到好的效果。目前常用的初始化 Xavier 和 He 初始化大致是使用均匀分布、正态分布，并使每一项的方差为 $\frac{4}{d_i+d_{i+1}}$ 或 $\frac{2}{d_i+d_{i+1}}$（取决于激励函数的选择），这里的 d_i 和 d_{i+1} 为矩阵两边中间层的点数。

初始化对训练结果值也有很大的影响，在定理 4-1我们证明了当初始化的值大小满足一定条件时，如果梯度下降收敛，那么训练中矩阵值和最初的初始值不会相差的太多。而梯度下降的收敛性在论文[7]，[6] 中都有研究：作者证明了在中间层节点数目很多（过参数化）的情况下，对均方误差的梯度下降总能快速收敛。

参考文献

[1] SUTSKEVER I, MARTENS J, DAHL G, et al. On the importance of initialization and momentum in deep learning[C/OL] // DASGUPTA S, MCALLESTER D. Proceedings of Machine Learning Research, Vol 28 : Proceedings of the 30th International Conference on Machine Learning. Atlanta, Georgia, USA : PMLR, 2013 : 1139 – 1147.
http://proceedings.mlr.press/v28/sutskever13.html.

[2] CHEN Y, CHI Y, FAN J, et al. Gradient descent with random initialization: fast global convergence for nonconvex phase retrieval[J/OL]. Mathematical Programming, 2019, 176(1-2) : 5 – 37.
http://dx.doi.org/10.1007/s10107-019-01363-6.

[3] GLOROT X, BENGIO Y. Understanding the difficulty of training deep feedforward neural networks[J/OL], 2010, 9 : 249 – 256.
http://proceedings.mlr.press/v9/glorot10a.html.

[4] HE K, ZHANG X, REN S, et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification[J/OL]. CoRR, 2015, abs/1502.01852.
http://arxiv.org/abs/1502.01852.

[5] GROHE M. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data[J/OL]. CoRR, 2020, abs/2003.12590.
https://arxiv.org/abs/2003.12590.

[6] DU S S, ZHAI X, PÓCZOS B, et al. Gradient Descent Provably Optimizes Over-parameterized Neural Networks[J/OL]. CoRR, 2018, abs/1810.02054.
http://arxiv.org/abs/1810.02054.

[7] DU S S, LEE J D, LI H, et al. Gradient Descent Finds Global Minima of Deep Neural Networks[J/OL]. CoRR, 2018, abs/1811.03804.
http://arxiv.org/abs/1811.03804.

[8] ALLEN-ZHU Z, LI Y, SONG Z. A Convergence Theory for Deep Learning via Over-Parameterization[J/OL]. CoRR, 2018, abs/1811.03962.
http://arxiv.org/abs/1811.03962.

[9] LI Y, LIANG Y. Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data[J/OL]. CoRR, 2018, abs/1808.01204. http://arxiv.org/abs/1808.01204.

[10] GE R, LEE J D, MA T. Learning One-hidden-layer Neural Networks with Landscape Design[J], 2017.

[11] Wikipedia contributors. Hermite polynomials — Wikipedia, The Free Encyclopedia[J/OL], 2021. https://en.wikipedia.org/w/index.php?title=Hermite_polynomials&oldid=1018860608.

致　　谢

感谢指导老师林冰凯老师。从最开始选题的迷茫到最后论文的完成，林冰凯老师在这段时间里花费了很多的时间和我进行周期性的讨论，联系紧密，时刻关注论文工作的进展。在不断讨论中也渐渐清晰了方向，最终促成了这篇论文的完成。