中文图书分类号：TP391
密　　级：公开
UDC ：004
学　校　代　码：10005

北京工業大學
BEIJING UNIVERSITY OF TECHNOLOGY

硕 士 学 位 论 文

MASTERAL DISSERTATION

论　文　题　目：Relu 网络的一种新型自适应优化方法研究

论　文　作　者：王铎
学　　科：计算机科学与技术
指　导　教　师：刘波　副教授

论文提交日期:2020 年 6 月

UDC：004
中文图书分类号：TP391

学校代码：10005
学　　号：S201761159
密　　级：公开

北京工业大学工学硕士学位论文

题　　目： Relu 网络的一种新型自适应优化方法研究

英文题目： A Novel Adaptive Optimization Method For ReLU Network

论 文 作 者： 王铎
学 科 专 业： 计算机科学与技术
研 究 方 向： 计算机应用技术
申 请 学 位： 工学硕士
指 导 教 师： 刘波副教授
所 在 单 位： 信息学部
答 辩 日 期： 2020 年 5 月
授予学位单位： 北京工业大学

摘 要

深度神经网络已成为计算机视觉及人工智能领域的研究重点。目前，在图像分类、语义分割等任务中,已有相关研究通过使用不同结构的人工神经网络获得了良好的表现。然而，在训练过程中，一些超参数的选择，如学习率，会对模型的精度产生较大影响，较大的学习率会使模型很难收敛而较小的学习率又会导致训练耗费较长时间。在继往的研究中，学习率的选择往往依赖经验，使得训练效率下降。同时，在优化神经网络时往往使用反向传播算法使梯度由后向前逐层传递，但由于梯度在传递时采用链式法则连乘获得，如若发生损失函数在位于梯度变化较大的“悬崖”区域求梯度，或者某一层网络初始化较差而导致激活函数对梯度信息出现截断等现象，梯度信息将无法稳定传递并且可能导致出现梯度消失或梯度爆炸现象，最终导致训练失败。

因此，本文受神经科学中神经元激活模式启发，通过分析 ReLU 激活函数的数学性质以及网络权值的更新方式，提出了一种基于 ReLU 激活函数且不依赖于反向传播算法的神经网络自适应学习率优化算法。该方法可以根据网络每一层的权值状态寻找适合该层的最优学习率，并不依赖反向传播算法独立更新该层权值，在不需要手动设置学习率的前提下保证算法精度，进而提高优化效率。并且，该方法还能避免出现由于梯度信息无法传递导致的训练失败，对于个数少维度高的样本具有优于随机梯度下降以及目标差传播算法的收敛速度。最后，本文通过实验验证所提出算法的性能。本文的具体工作如下：

第一，本文通过分析采用 ReLU 函数作为激活函数的神经网络的逐层结构，提出一种基于目标传播的新型权值更新方法。该方法可以将多层嵌套的神经网络拆解，借助所训练的“近似逆映射”逐层以目标值传递形式将误差信息传回每一层网络，由此实现不依赖反向传播的神经网络更新，避免了由于梯度信息传递异常导致的训练失败。

第二，在每层神经元更新过程中，神经元的更新方向与步长可通过计算该层神经网络输出值与目标值的 MSE 损失自适应获得。由于此 MSE 损失为凸函数，借助凸优化方法我们可以解析得出该层网络的最优学习率。通过使用此学习率进行训练可以加快收敛速度并降低人工调参成本。

第三，由于最优学习率的计算依赖该层神经元的权值状态以及该层输入值的规模，当神经元个数以及输入规模较大时，计算最优学习率需要耗费一定算力。针对此问题本文提出两种加速算法，即线搜索方法以及批(batch)方法用以加速训练。实验表明本文提出的加速算法具有良好的精度及收敛速度。

第四，本文分析了所提出优化算法的复杂度。通过分析复杂度可知此算法收敛速度为 $O(\frac{1}{r})$，而最大不会超过 O(m)。其中 r 为平均学习率而 m 为样本个数。实验表明，此方法对于样本量较少的数据优于传统反向传播算法，对于样本较多的数据，使用加速算法仍可得到不差于传统反向传播算法的表现。

关键词：神经网络；ReLU；优化算法；目标传播；自适应学习率

Abstract

Deep neural networks have shown potential in the fields of computer vision and artificial intelligence. By using artificial neural networks with different structures, related works have shown good performance in image classification, semantic segmentation, and other tasks. However, the choice of learning rate during the training process often depends on experience. An excessive learning rate will cause the model hard to converge but the small learning rate will increase the training time. Besides, when using the back-propagation algorithm, the gradient error will be transmitted layer by layer from back to front. Due to the gradient is multiplied by the chain rule during transmission, sometimes the gradient information will not be transmitted stably and may lead to the gradient disappearance or the gradient explosion, eventually leading to training failure.

Inspired by the mode of neuron activation in neuroscience, this work proposes a method which disassembles complex neural networks and updates the weights of each layer independently and adaptively. By analyzing the mathematical properties of the ReLU activation function and the optimization method, an adaptive learning rate optimization algorithm for ReLU network which does not rely on the back-propagation algorithm has been proposed. The optimal learning rate can be found from the weights state of each layer and updates independently. Hence, by using this method, the optimization speed can be improved, and training failures from gradient error can be avoided due to the independent updating. It is verified through experiments that the algorithm proposed in this work shows the accuracy of no less than the SGD algorithm and has a better optimization speed for high-dimensional small sample data. The specific works are as follows:

First, by analyzing the structure of the loss function from each layer of the neural network which using the ReLU function as the activation function, a novel weight update method based on target propagation is proposed. This method can disassemble the multi-layered neural network and updating the weights of each layer independently. By using the trained “approximate inverse mapping”, error information can be transmitted layer by layer through the target value. Thereby updating the network without using the back-propagation algorithm and avoid training failure due to abnormal gradient information transmission.

Second, in the updating processes, the update direction and the step size of the neurons can be adaptively obtained from the MSE loss between the output value and the target value. Since this MSE loss is a convex function, the optimal learning rate can be obtained by using the convex optimization method analytically. And this learning rate can speed up the algorithm and reduce the cost of manual tuning.

Third, due to the optimal learning rate depends on the weights state and the input value of the neurons. When we have a large number of neurons and input scales, it may take a certain amount of time to calculate the optimal learning rate. Aiming at this problem, this work proposes two acceleration strategies, which are the line-search method and the batch method. Experiments show that the acceleration algorithms proposed show a good performance in accuracy and convergence speed.

At last, we analyze the complexity of the optimization algorithm of this work. It shows this method is more efficient than the traditional back-propagation algorithm with small samples. For data with more samples, the approximate optimal learning rate can still obtain performance that is not worse than the traditional back-propagation algorithm. By analyzing the complexity, the convergence speed of this algorithm is O ($\frac{1}{r}$), and the maximum will not exceed O (m). Where r is the average step size and m is the number of samples.

Keywords: Neural network; ReLU; optimization algorithm; target propagation; self-adaptive learning rate

目 录

第 1 章　绪论

本章将详细介绍深度神经网络的研究背景与意义、关于深度神经网络优化算法在国内外研究现状以及本文所提出的基于 ReLU 网络自适应优化算法的主要研究内容，在本章最后将介绍本文的章节结构。

1.1　研究背景与意义

随着神经科学以及脑科学的不断发展，人们逐渐发现大脑思维活动的本质就是单向流淌在神经通路上的信号。大脑神经元接受之前的数个神经元传来的信号，按照一定标准选择向某一个下级神经元进行信号传递[1] 。同时，在学习和认知过程中，人们发现脑神经的组成具有分层结构，从低级神经细胞到高级神经细胞，不同层次的细胞提取不同层次的特征[2] 。另外，对于每一个神经元，其都有一定的自我调节功能，主要依赖于神经突触的可塑性[3] 。这种多层结构的神经系统使得生物体能够产生复杂的智能行为[4] 。

借助神经科学的发展，计算机科学家通过模拟生物神经结构组建了多种人工神经网络，这些神经网络在计算机视觉、自然语言处理等智能任务中取得了优秀的表现[5-8]。通过对神经多层结构的加深理解，计算机科学家已经认识到深层结构对于神经网络的重要性，即随着神经网络层数的增加，从数据中提取的特征会越来越高级，最终使模型精度越来越好。但是对于神经元的赋值过程，由于目前对于脑神经的认识仍略显不足，人们仍未找到具有生物可解释性的神经元赋值方式[9] 。随着网络层数的加深，多层嵌套的网络结构以及非线性激活函数使得最终的损失函数高度非凸，这也导致了神经网络较难训练[10] 。为解决此问题，计算机科学家采用反向传播算法[11] ，通过梯度下降以链式法则形式将误差逐层传递回每一层网络。但是这种方法依赖诸多训练超参数，如学习率、批大小以及权值衰减等参数，这导致神经网络在训练时依赖神经网络设计者的经验以及在调试时难以直接定位错误。同时，采用反向传播算法在训练神经网络时也可能导致梯度爆炸或梯度消失现象的出现，即如果某一层网络由于自身状态以及激活函数性质无法稳定传递梯度信息，这将导致该层之前的网络训练失败[12] 。另外，这种方法比较缺乏生物合理性，目前脑科学家已经通过突触的结构验证了正向传播的存

在，但是对于反向传播的方式，目前仍未发现相应的细胞结构予以证实[4] 。因此，寻找一种能够自适应学习率，且不依赖于反向传播算法的神经网络优化算法对于提升网络训练效率十分重要。

本文的研究目标在于提出一种基于 ReLU 激活函数的神经网络自适应优化算法，该算法不依赖于反向传播算法，从而避免了由于梯度依赖所造成的训练失败。同时，该算法通过分析网络每一层神经元权值的状态，能够计算出该层最优的学习率，从而提高训练效率。

1.2 国内外研究现状

人工神经网络的研究起源于 1943 年，由 Warren McCulloch 和 Walter Pitts 提出的“M-P 神经元模型”[13] ，即通过简单电路来模拟大脑的神经元行为。该模型接收n个其他神经元的输入信号X_i,通过自身权值w_i对信号进行加工传递。

1958 年，美国心理学家 Frank Rosenblatt 基于 M-P 神经元模型提出感知机模型，该模型由两层神经网络组成，其中输入层负责接收外界信号而输出层为 M-P 神经元[14] 。由于该模型无法解决线性不可分问题，如异或问题，后续学者 Paul J. Werbos[15] ，Hinton[16][11] 等提出多层感知机（Multilayer Perceptron, MLP）以及反向传播算法（Back propagation, BP）用以解决复杂分类问题[17] 。然而，随着神经网络深度的增加，神经网络的训练也越发困难，在训练过程中可能遇到收敛时间过长、梯度爆炸、梯度消失、容易落入局部极小等问题。目前，国内外相关学者针对这些问题提出了不同的改进措施，主要分为基于反向传播算法的改进优化算法以及非反向传播的优化算法。

1.2.1 基于反向传播算法的改进优化算法

反向传播算法的核心内容是对损失函数以梯度下降方式进行优化。对于梯度下降算法来说，由于每次迭代需要使用全部数据，当样本量较多时会消耗大量计算时间。为解决此问题，2010 年 Bottou 等提出随机梯度下降算法（Stochastic gradient descent, SGD）每次迭代只使用一个样本进行训练使得每一轮训练速度大

大加快[18]。但是由于每次训练的样本是随机挑选出来的，这也导致网络的收敛过程充满随机性。Hinton 等人于 2012 年提出 mini-batch 算法对 SGD 加以改进，通过每次只迭代一小批数据加快训练速度[19]，由于该算法能够在保证速度的同时逼近梯度下降算法的收敛效果，因此成为了主流的神经网络训练算法之一。

由于多层嵌套的神经网络损失函数高度非凸，在使用梯度下降及其改进算法时如果学习率等参数调节不当有可能会使网络收敛于局部极小点或鞍点[20]。为解决此问题，Qian 等人提出动量法（Momentum）通过引入动量项给予网络在陷入局部极小时仍有一定跳出能力[21]。但是此方法仍有一定缺陷，如果动量参数选择不合适，当网络处于全局最小点附近时，由于动量影响会导致网络无法收敛到全局最小点，在最坏情况可能直接改变梯度下降方向最终导致算法无法停止。

在基于反向传播算法的神经网络的训练过程中，优化更新方向由梯度信息所决定，由于每次更新时损失函数的数学性质和形状不同，因此会导致梯度在不同维度的差异较大。可能出现部分方向梯度值较大更新较快而部分方向梯度值较小更新较慢的情况，为平衡不同维度之间梯度值差异导致的更新问题，2012 年 Hinton 提出 RMSProp（Root Mean Square Prop）算法对更新方向进行平滑约束[22]。该算法通过使用梯度平方的加权平均作为更新方向以缓解更新中由于梯度方向不平衡导致的损失函数大幅抖动的问题。然而，该算法在迭代过程中对加权平均的计算会使得学习率惩罚越来越大，导致学习率可能越来越小并最终使得模型更新极为缓慢或者提前停止。

当网络层数加深时，由于不同深度的网络层学习的内容可能不同，因此对不同层采用不同学习率可能会提高优化效率。基于此思想，John Duchi 等人于 2011 年提出 AdaGrad（Adaptive Gradient Algorithm）算法用于对不同维度的神经元采用自适应对应维度的学习率以提升模型表现[23]。该算法通过采用二阶动量的方式对学习率进行动态约束。通过采用这种方式，可以控制更新频繁的神经元学习率惩罚较大而更新不频繁的神经元学习率惩罚较小从而对于稀疏数据能够有较好的优化效率。然而，该算法初始的全局学习率仍需依赖经验手动设定，如果学习率过大仍会导致算法震荡。同时，在训练中后期，由于二阶动量惩罚的累加导致该方法对部分学习率惩罚趋近于 0 从而在未达到极小点的情况下可能提前停止训练。

对于 AdaGrad 算法的相应问题，Matthew D. Zeiler 提出 Adadelta（Adaptive

delta）算法予以改善[24]。该算法通过使用一阶方法模拟二阶牛顿法，但是不采用全部历史梯度计算二阶动量而仅采用一个局部时间窗口的梯度进行二阶动量计算从而避免梯度累加导致的训练提前终止。同时，该方法通过使用更新量平方的指数加权平均替代初始化全局学习率从而降低对学习率设置的敏感性。但是此方法对初始化扰动系数ε很敏感，如果设置不当，ε太小会导致前期学习率过小收敛缓慢而ε太大会导致后期算法震荡无法收敛。

2014 年，Jimmy Ba 等人通过结合使用一阶动量修正以及二阶动量修正提出 Adam（Adaptive Moment Estimation）算法改善优化[25]。该算法通过结合 AdaGrad 和 AdaDelta 两种方法的优点，利用二阶动量自适应学习率以及动量法对更新方向的修正能够达到较好的计算效率，此方法对于稀疏数据和非平稳目标数据有较好的表现。但是由于 SGD 算法和 AdaGrad 算法可以保证学习率总是单调下降的，而基于局部时间窗口的算法却无法保证，这导致 Adadelta 和 Adam 算法有无法收敛或者收敛于局部极小的风险，并可能使模型无法达到 SGD 算法的精度表现[26]。

1.2.2 非反向传播的优化算法

由于反向传播算法的全局更新策略以及梯度依赖性致使其缺乏生物合理解释并且在实现上依赖调参经验，相关科学家对使用其他方法优化神经网络展开研究。上世纪 90 年代起，科学家在使用遗传算法训练神经网络进行了诸多尝试，然而对比反向传播算法来说遗传算法并没有得到足够良好的表现[27] [28]。

对于反向传播算法来说，导致其缺乏生物合理性的主要原因是以梯度通过链式法则形式对误差进行传递。由此相关科学家针对误差传递的方式开展研究并结合神经科学提出了一些新的更新方法。2014 年，由 Bengio 等人提出目标传播（Target Propagation）概念用以解决随机梯度下降对于深层网络训练困难的问题[29]。该方法通过使用目标值传递误差而不是以梯度传递误差完成整个网络的更新。目标值通过构造辅助网络训练逆映射自动编码器获得。然而，此方法的收敛条件为所训练出的逆映射与原网络完全可逆，这在实际应用中很难实现，因此该方法在使用时仍会有不收敛的情况出现。

为缓解目标传播算法的收敛性缺陷，2015 年 Bengio 的学生 Dong-Hyun Lee 提出目标差传播对目标传播算法进行改善[30]。该算法通过构造伪逆自动编码器

获取目标值而不是完全逆映射从而加强了网络的收敛性，削弱了其收敛条件[31] 。

考虑到生物学中，正向传播与反向传播的权重可能不是共享且对称的，2016 年 Lillicrap 等人提出反馈对比算法[32] 。该算法在反向传播时，不再使用前馈传播权重的转置W^T，而是使用一个固定的随机突触权重矩阵用于网络更新，实验表明该方法也可完成网络的训练过程。

由于生物学中神经元具有分层组织结构，高层神经元所获得的信息由底层神经元组合构成。相关学者于上世纪 80 年代开始对此展开研究并认为逐层的分布式表示是神经网络学习成功的关键。2018 年南京大学周志华提出使用多层梯度提升决策树获取逐层分布式表示以训练神经网络[33] 。对于非图像领域的数据，使用梯度提升决策树方法获取的逐层分布表示可能更有意义，即使用梯度提升决策树算法也可以进行特征提取而不是仅仅用来完成分类任务。

在一些应用场景，比如嵌入式芯片中，网络被二值化表示或网络离散化激活函数出现不可导不连续的情况。此时反向传播算法无法将误差信息传递回网络每一层。为解决此问题，2017 年 Abram, L. Friesen 等人提出可行目标传播算法[34] 通过目标传播算法将网络拆分为若干独立层并使用凸组合优化每一层网络，实验证明，对于非连续硬阈值损失函数构成的神经网络，可行目标传播算法得到了不错的表现。

1.3　主要研究内容

截止到目前，世界各国的研究者在研究深度神经网络时仍主要使用反向传播及其衍生算法，尽管相关学者已经提出非反向传播算法的研究必要性但是相关算法仍有待进一步研究。对于一些自适应算法，如 AdaGrad 等来说，尽管其能够自适应部分超参数，但是在其自适应的过程中又带来更多的超参数依赖，使得整个训练过程仍旧依赖网络设计者的经验。同时，对于损失函数的形状及数学性质的研究刚刚起步，尽管已有相关研究表明对损失函数进行分析将有助于算法性能的提升，但是目前仍鲜有相关算法通过分析损失函数的形状对具体优化过程进行改进。

因此，针对上述问题，本文通过分析 ReLU 网络的损失函数形状提出一种不依赖反向传播的自适应深度神经网络优化方法。该优化方法能够根据网络每一层

的状态自行计算最优学习率从而使训练不再依赖经验设置学习率并加快收敛速度，提升训练精度。

首先，为了实现非反向传播自适应更新算法，基于目标传播更新方式，本文提出以目标值作为误差监督信息而不是以梯度传递误差信息进行网络更新的方法，其中目标值的获取由所训练的辅助结构“近似逆映射”获得，由此可打破梯度的链式法则从而解决由梯度带来的一系列训练问题，如梯度消失，梯度爆炸等。同时，由于不依赖梯度信息，本网络可以不使用 Batch Normalization 或残差学习等操作用于调整误差信息的尺度或传递残差信息即可完成训练。

其次，目标值获取完成后，通过神经网络每一层输出值与目标值所构成的逐层损失获得该层权值的更新方向和步长。由于每一层逐层损失采用最小化均方误差（MSE）损失构建，为凸函数，因此在计算最优更新步长时，本文通过采用适当的凸优化方法对最优步长进行解析求解，由此达到自适应学习率，加速训练的目的。

接下来，为提升训练效率，本文在所提出的在线自适应更新方法基础上提出两种加速更新方式，分别为：1）批（batch）方法：每次使用一批数据计算最优步长，并以此最优步长作为全局最优步长的近似更新神经元权值。2）线搜索方法：通过使用启发式线搜索获取近似最优步长，只要保证损失函数下降算法仍能快速收敛。由于采用自适应学习率，在训练初期损失函数的下降过程可以不再“小步试探”，而是通过采用动态最优步长快速下降，由此达到加速收敛的效果。实验表明，本文所提出的加速更新算法拥有良好的精度与收敛速度。

最后，本文分析了所提出算法的复杂度，给出了收敛速度与样本个数与维度的关系。通过分析复杂度可知，随着神经元个数的增加，最优步长也跟随快速增大，迭代次数与单次步长成反比，因此算法收敛速度为 $O(\frac{1}{r})$，而最大不会超过 O(m)。其中 r 为平均步长而 m 为样本个数。实验证明，本文所提出的三种自适应优化算法，即在线方法，批（batch）方法以及线搜索方法均能够获得不亚于随机梯度下降的优化精度，并能够一定程度上提高收敛速度。

1.4 论文的组织结构

本文的组织结构如下：

第 1 章：介绍论文的研究背景与意义，并对神经网络以及神经网络的优化算法国内外研究现状做了简要介绍。针对解决反向传播优化算法梯度依赖以及调参不当导致的一系列问题，本文提出了一种基于 ReLU 激活函数并且不依赖于反向传播算法的神经网络自适应学习率优化算法，并阐述了本文的主要研究内容和组织结构。

第 2 章：详细介绍本文的相关技术。阐述深度神经网络的原理和基本结构；介绍反向传播以及目标传播算法的原理及特点；介绍多种基于反向传播的优化算法如梯度下降，SGD，动量法，RMSProp 以及一系列自适应算法如 AdaGrad 以及 Adam，以及非反向传播优化算法如目标传播和多层梯度提升决策树。

第 3 章：本章将详细介绍基于 ReLU 激活函数的神经网络自适应优化算法，主要包括在线方法、批（batch）方法和线搜索方法。还将详细阐述自适应最优学习率的获取方法以及更新策略。在本章最后，将对所提出的基于 ReLU 激活函数的神经网络自适应优化算法进行复杂度分析。通过分析最优学习率与样本个数以及神经元的关系给出复杂度的数学表达。

第 4 章：为了评估不同优化算法的效率和优势，本文设计了多组对比试验对 SGD，目标差传播算法以及本文提出的多种基于 ReLU 激活函数的神经网络自适应优化算法进行比较。本章详细描述了实验的数据集，网络结构以及实现方式。并针对不同实验结果进行分析，得出实验结论。

1.5　本章小结

通过本章介绍，我们可以详细了解基于 ReLU 激活函数的神经网络自适应优化算法的研究背景和意义，以及神经网络优化算法的国内外研究现状。由于目前对于非反向传播的神经网络优化算法仍有待研究，本文提出了基于 ReLU 激活函数的在线自适应优化算法、批（batch）自适应优化算法和线搜索自适应优化算法三种不同的优化方法用于对神经网络进行训练。本文还将对所提出的基于 ReLU 激活函数的神经网络自适应优化算法进行复杂度分析，并给出数学表达。最后，本章详细介绍了本文的研究内容和组织结构。

第 2 章　相关技术介绍

本章将阐述与课题相关的一些技术，如神经网络的原理以及训练方法等。主要介绍多层神经网络的组成结构，如神经元，损失函数，激活函数等，以及基于反向传播算法的神经网络常用的优化算法，包括梯度下降法，随机梯度下降法，动量法，RMSProp，以及一系列自适应算法如 AdaGrad 以及 Adam 等。并对可能导致训练失败的原因如梯度消失，梯度爆炸问题以及其解决方法如 Batch Normalization 及残差结构做出介绍。同时，对一些非反向传播算法进行介绍，重点介绍目标传播算法的原理及特点，它是本文所用方法的基础之一。

2.1　深度神经网络

2.1.1 M-P 神经元模型与激活函数

人工神经网络的最基本单位为神经元，并由若干神经元和其有向连接构成。单个神经元通过模拟生物神经元特征，接收n个输入信号X_i,通过自身权重w_i对信号进行加工传递最终输出y。神经元接收的总输入为$\sum_{i=1}^{n} w_i x_i$,通过与该神经元阈值θ做差后经激活函数f处理得到最终的神经元输出$y = f(\sum_{i=1}^{n} w_i x_i - \theta)$。

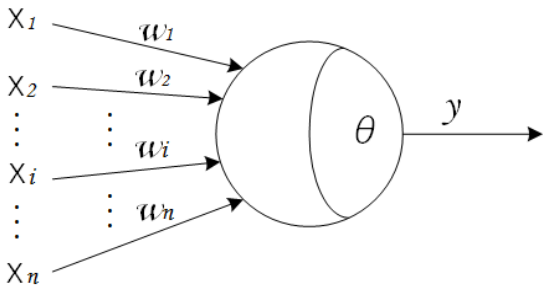


图 2-1　M-P 神经元模型结构图

Figure 2-1　The structure of M-P neuron

由于生物神经元具有“兴奋”和“抑制”状态，为模拟此状态，理想的激活函数为阶跃函数。即：

$$f = sgn(x) = \begin{cases} 1 & x \geq 0; \\ 0 & x < 0; \end{cases} \tag{2-1}$$

然而由于此函数在光滑性和连续性上性质较差，在实际中通常会使用

sigmoid、ReLU、tanh 等非线性函数进行替代。在这些激活函数中，由于 ReLU 函数具有形式简单，计算量小，避免梯度截断等特点已成为神经网络设计的常用选择[16]

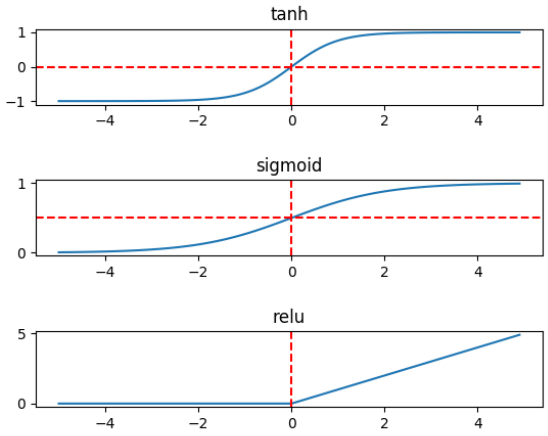


图 2-2　sigmoid、ReLU、tanh 激活函数图形解释

Figure 2-2　The Visualization of sigmoid, Relu and tanh function

其中，sigmoid 函数形式为$f = sigmoid(x) = \frac{1}{1+e^{-x}}$,是一个$\mathrm{R} \to (0,1)$的单调递增非线性映射。通过将数值映射到(0,1)区间从而模拟神经元的激活与抑制，此函数成为阶跃函数$sgn(x)$的一个优秀替代。同时，由于该函数的倒数满足

$$f'(x) = f(x) * (1 - f(x)) \qquad (2\text{-}2)$$

这种特殊形式，使得其在反向传播算法中由于计算简单具有一定优势，从而在早期神经网络结构中被广泛使用。

tanh 函数形式为$f = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$，是一个$\mathrm{R} \to (-1,1)$的单调递增非线性映射。由于其同样拥有非线性映射的性质在激活函数中也得到了良好的表现。

ReLU 函数即线性整流函数，是一种分段线性函数。其形式为：$f = ReLU(x) = \begin{cases} x & x \geq 0; \\ 0 & x < 0; \end{cases}$。此函数通过对小于 0 数值的抑制也起到了非线性激活的效果，能够增强网络非线性拟合的能力。同时，此函数形式简单，计算方便，在近年的神经网络设计中成为了常用的激活函数。另外，由于该函数有良好的数学性质，除$x = 0$一点不可导以外，剩余部分皆为线性表达。由此对于单层基于 ReLU 损失函数的网络层，其损失函数性质可以明确分段表示。本文也基于此性质提出了自适应最优步长概念，将于后文详述。

2.1.2 深度神经网络与多层感知机

最简单的神经网络被称作感知机模型，是由多个神经元构成的仅有输入层和输出层的神经网络。图 2-3 展示了感知机模型的基本架构，其中$X = \{x_i, i = 1,2 \dots n\}$为输入数据，$Y = \{y_j, j = 1,2 \dots m\}$为输出数据，共有 m 个神经元构成此感知机。

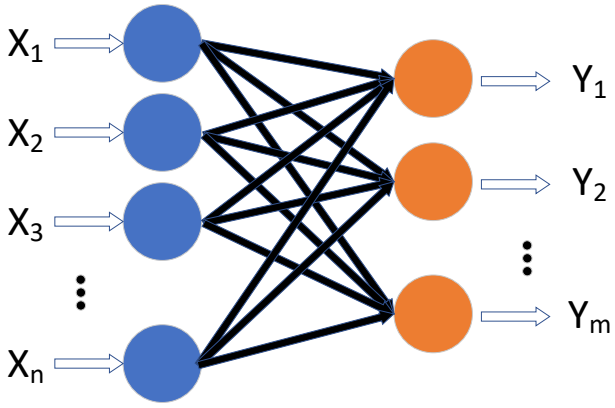


图 2-3　感知机模型

Figure 2-3　Perceptron model

其假设空间是定义在特征空间中的所有线性分类器，即函数集合$\{f|f_i(x) = w_j \cdot x_i + b\}$，其中$w_j$为第$j$个神经元权值。线性方程$w_j \cdot x_i + b = 0$为特征空间$R^n$中一个超平面 S，其中$w_j$为超平面法向量方向而$b$为超平面截距。此超平面将特征空间分为两部分，分别对应分类器的正负样本两类，因此称此超平面为决策超平面。

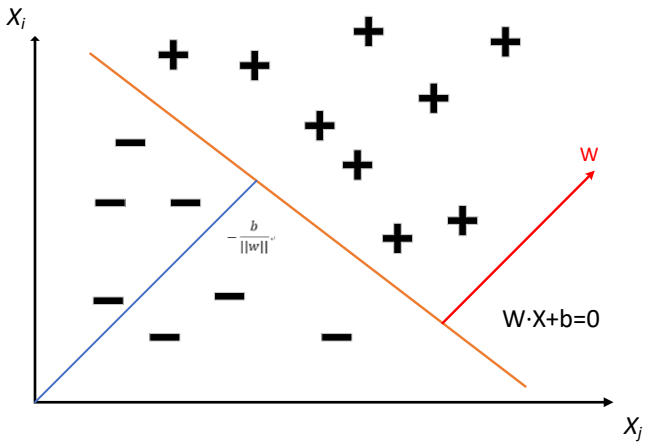


图 2-4　决策超平面

Figure 2-4　Decision hyperplane

然而对于一些线性不可分问题，如异或问题或环状数据集分类等问题，使用一个超平面无法对这类问题的特征空间做出有效分割，此时使用单层的感知机模

型将无法完成此类问题的分类任务，为解决此问题，Hinton 等人提出多层感知机，通过引入隐藏层获得复杂非线性函数的拟合能力以解决复杂线性不可分问题。

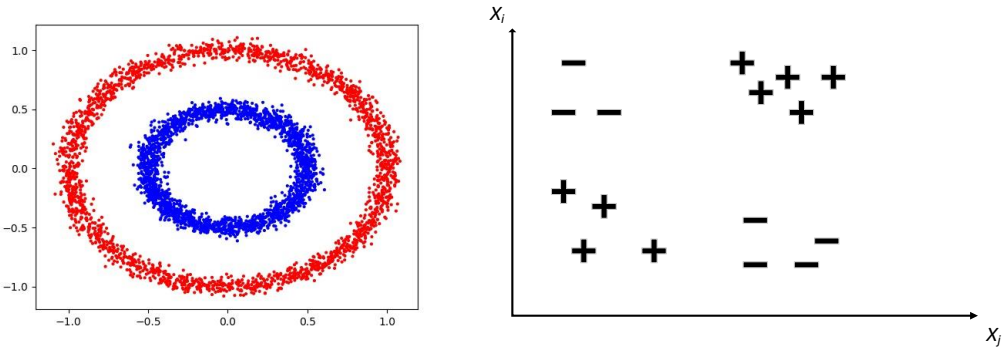


图 2-5　a)环状数据　　b)异或数据

Figure 2-5　a) Ring data　　b) XOR data

多层感知机（MLP）是深度神经网络（DNN）的一种，相比于单层感知机来说，其在输入层与输出层之间增加了若干隐藏层，每一层隐藏层由若干神经元构成，并接收上一层的全部输出。层与层之间完全相连，构成嵌套关系。假设拥有l层隐藏层的 MLP 输入为$\mathrm{x_i} \in \mathrm{R^n}$，其隐藏层神经元权值$\mathrm{W} = \{\mathrm{W_d}: \mathrm{W_d} \in \mathrm{R^{n_d * n_{d-1}}}\}_{d=1}^{l}$那么该 MLP 输出可表示为：

$$y_i = f(x_i; W) = W_l(f_{l-1}(W_{l-1}(f_2(W_{l-2}(\dots W_2(f_1(W_1 x_i)))))))\quad(2\text{-}3)$$

其中f_i为第i层激活函数。

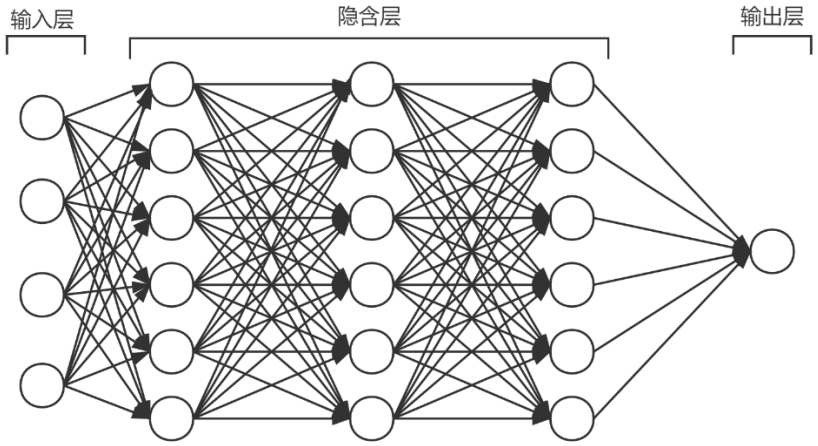


图 2-6　多层感知机

Fugue 2-6　Multilayer Perceptron

2.1.3 损失函数

损失函数通常是一种对误差进行度量的函数，其通过计算估计值与真值的差异来评估神经网络模型的误差。损失函数越小，代表模型预测值与真值的误差越小。神经网络通过优化寻找合适的神经元权值使得损失函数达到最小值，此过程称为神经网络的训练过程。根据任务不同，对于损失函数的选取也有所不同。常见的分类损失函数主要为均方误差损失（MSE）以及交叉熵损失（Cross Entropy）。

均方误差损失通过统计模型估计值$f(x_i)$与真值t_i之差平方的期望表示模型的误差。其形式为：

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - t_i)^2 \tag{2-4}$$

由于该损失对每一个样本预测值与真值的欧氏距离进行了良好度量，因此在分类以及回归等任务中应用比较广泛。但是，此损失函数对于离群点较为敏感，如果数据集存在较多异常离群点，使用此损失函数可能导致模型错误率的升高。

交叉熵损失用于评估预测概率分布与真值概率分布之间的差异，其表达形式如下：

$$H(p,q) = -\sum_{i=1}^{N} p(x_i) log\, q(x_i) \tag{2-5}$$

其中$p(x_i)$为真值的概率分布而$q(x_i)$为预测值的概率分布，交叉熵越小代表两分布越接近。由于交叉熵损失度量的是两个分布的差异，即在给定真实分布的情况下，使用预测分布所指定策略消除系统不确定性的成本。只有预测正确的样本才对交叉熵损失有贡献而预测错误的没有，因此该损失能够因此在分类任务中表现良好。

2.2 反向传播与基于反向传播的优化算法

2.2.1 反向传播算法

反向传播（Back propagation）算法是一种以梯度信息作为误差信息进行误差反向传播的神经网络迭代更新算法。该算法通过计算损失函数对各网络层神经元权值的梯度，并以此梯度结合优化算法对神经元权值进行更新，最终获得使损失函数最小的权值组合。由于梯度通过链式法则进行传递，每一层神经网络层都能

获得属于该层的误差信息，因此该方法成为主流的多层神经网络训练方法。然而，同样由于链式法则，第l层神经网络权值的梯度∇W_l依赖于第l+1 层神经网络权值的梯度∇W_{l+1}，当第l+1 层神经网络权值的梯度∇W_{l+1}出现误差时，此误差会随着梯度链影响第l层以及第l层之前的每一层网络的更新，当出现极端情况如梯度消失或梯度爆炸问题会导致第l层以前的网络更新失效。

反向传播算法是一种迭代算法，通过迭代若干次完成神经网络权值更新，每次迭代主要分为前馈传播和反馈传播两部分。其中前馈传播主要流程为：1）将训练数据通过网络每一层，最终得到网络输出。即完成式 2-3。2）由所得到的输出值和已有的真值标签计算损失函数。反馈传播主要流程为：1）根据前馈传播获得的损失函数对网络每一层神经元权值求梯度。2）使用梯度结合优化算法完成每一层神经元的更新。

以下将以 3 层神经网络为例详述反向传播算法，网络结构如图 2-7 所示：

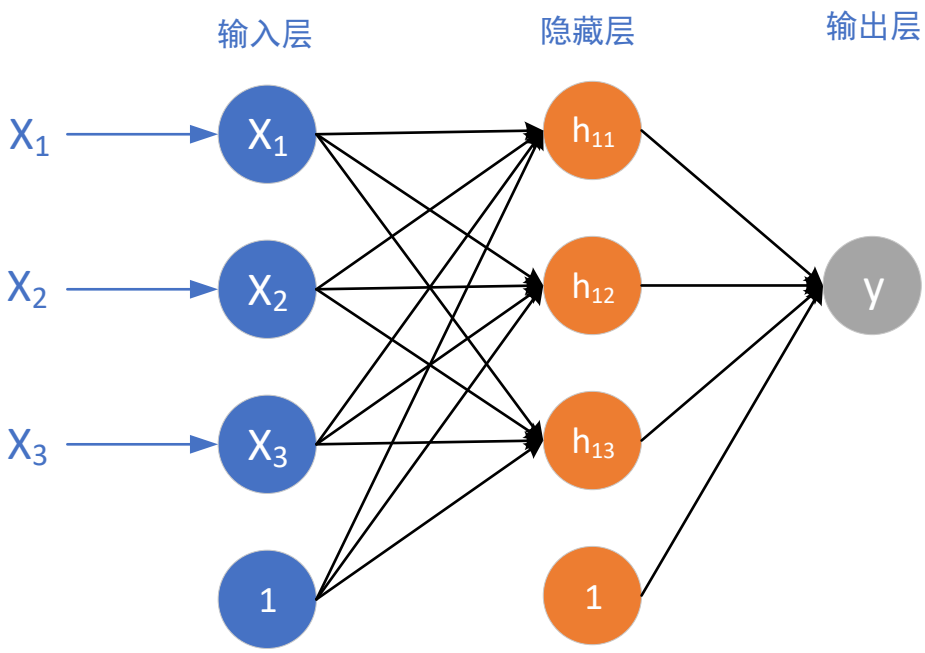


图 2-7　3 层神经网络实例

Figure 2-7　The instance of 3-layer neural network

假设输入数据$\mathrm{X}=(\mathrm{X}_1,\mathrm{X}_2,\mathrm{X}_3)$，其对应真值标签为T，$h_i^l$表示第$l$层隐藏层第$i$个神经元的输出值。$W_{ij}^l$表示第$l$层隐藏层第$j$个神经元与第$l+1$层第$i$个神经元之间连接的权值，$b_i^l$表示第$l$+1 层隐藏层第$i$个神经元偏置值，激活函数以$f$表示，网络输出为 y，损失函数以$\mathrm{Loss}(\mathrm{y},\mathrm{T})$表示。以下将叙述此神经网络完成一次反向传播算法并完成权值更新的过程：

1）完成前馈传播，得到网络输出，即：

$$h_1^1 = f(W_{11}^1 X_1 + W_{12}^1 X_2 + W_{13}^1 X_3 + b_1^1) \quad (2\text{-}6)$$

$$h_2^1 = f(W_{21}^1 X_1 + W_{22}^1 X_2 + W_{23}^1 X_3 + b_2^1) \quad (2\text{-}7)$$

$$h_3^1 = f(W_{31}^1 X_1 + W_{32}^1 X_2 + W_{33}^1 X_3 + b_3^1) \quad (2\text{-}8)$$

$$y = f(W_{11}^2 h_1^1 + W_{12}^2 h_2^1 + W_{13}^2 h_3^1 + b_1^2) \quad (2\text{-}9)$$

2）计算损失函数$\mathrm{Loss(y, T)}$

3）完成反馈传播，使用链式法则计算各神经元梯度，以W_{11}^1为例，即：

$$\frac{\partial Loss}{\partial W_{11}^1} = \frac{\partial Loss}{\partial y} * \frac{\partial y}{\partial h_1^1} * \frac{\partial h_1^1}{\partial W_{11}^1} \quad (2\text{-}10)$$

其中

$$\frac{\partial y}{\partial h_1^1} = \frac{\partial y}{\partial f} * \frac{\partial f}{\partial h_1^1} = \frac{\partial y}{\partial f} * W_{11}^2 \quad (2\text{-}11)$$

$$\frac{\partial h_1^1}{\partial W_{11}^1} = \frac{\partial h_1^1}{\partial f} * \frac{\partial f}{\partial W_{11}^1} = \frac{\partial h_1^1}{\partial f} * X_1 \quad (2\text{-}12)$$

4) 利用各神经元梯度采用优化算法更新权值。

2.2.2 常见优化算法

优化算法是深度学习领域十分重要的一部分，它决定了神经网络权值的更新规则并影响模型的收敛速度与精度。尽管多层嵌套的神经网络损失函数非凸，常见的优化算法通过调整合适的参数和优化策略仍可使网络收敛于损失函数最小的状态。近年来，相关学者针对神经网络优化问题提出一系列基于梯度信息的优化算法，本节将介绍梯度下降法，随机梯度下降，动量法，RMSProp，AdaGrad 以及 Adam 这些常见基于反向传播的优化算法。

梯度下降法（Gradient Descent）是一种寻找函数最小值的方法，常用于无约束优化问题中。从数学角度上看，梯度方向为函数增长最快的方向，那么负梯度方向即为函数减少最快的方向。利用此思想，梯度下降法设定负梯度方向为更新方向，配合合适的学习率作为步长对神经元权值进行迭代更新，即每一次迭代，更新为：

$$W \leftarrow W - \rho \nabla W \quad (2\text{-}13)$$

其中W为神经元权值，$\nabla \mathrm{W}$为该神经元对应梯度，ρ为学习率。对于梯度下降法，学习率的选择依赖经验尝试，合适的学习率会使得算法快速收敛于极小值附近。若学习率设置不当，不论过大还是过小，都会导致收敛速度较慢甚至无法收敛。

过大的学习率会导致震荡，而过小的学习率会使得梯度信息无法得到充分表现从而更新停滞。

在梯度下降算法中，每一次迭代使用全部训练数据进行计算，当数据集较大时，使用全部数据计算神经元所有维度的梯度将会消耗大量计算资源使得算法运行较慢。相关学者针对此问题提出了随机梯度下降（SGD）以及小批量梯度下降(mini-batch)算法对 SGD 加以改进。对于 SGD 来说，每次仅使用随机抽取的一个样本x_j计算梯度，并使用此梯度作为优化方向进行更新。即：

$$W \leftarrow W - \rho \frac{\partial Loss(y(x_j), label_j)}{\partial W} \tag{2-14}$$

这种方法使得算法进行每一次迭代(epoch)的速度大大提升，但是，由于单个样本所计算的梯度并不能代表函数值下降最快的方向，同时不同样本的梯度信息可能相差很大，因此使用 SGD 可能会导致训练过程出现随机性的震荡和误差。mini-batch 方法是对梯度下降法及 SGD 方法进行综合的一种训练策略，其每一次迭代（epoch）通过从全部训练数据$\mathrm{X} = \{x_i, \mathrm{i} = 1,2,3 \ldots \mathrm{n}\}$中随机选取一批（batch）$k$个数据$\mathrm{X_k} = \{x_k | x_k \in X, 1 < k < n\}$，每次更新使用这批数据的平均梯度作为优化方向，即：

$$W \leftarrow W - \rho \frac{1}{k} \sum_{i=1}^{k} \frac{\partial Loss(y(x_i), label_i)}{\partial W} \tag{2-15}$$

尽管采用 SGD 或 mini-batch 会导致训练出现震荡或误差，已有研究证明 SGD 或 mini-batch 在参数选取得当的情况下能够收敛于与梯度下降（GD）算法相同的最小值[35] 。因此使用 SGD 或 mini-batch 方法成为深度学习常见的训练方法之一。

梯度下降法对于凸优化问题一定可以达到全局最小值，但是对于非凸问题，如果学习率选取不当算法有可能陷入局部极小，当陷入局部极小时，此处梯度值为 0 会导致算法停止。为解决此问题，动量法引入动量项进行改善。在动量法中并不以当前梯度方向为优化方向，而是对当前梯度方向附加一之前所有梯度的加权平均作为本次优化方向，即第t次迭代的更新规则为：

$$v_t = \beta v_{t-1} + \rho \nabla W \tag{2-16}$$

$$W \leftarrow W - v_t \tag{2-17}$$

其中β为动量超参数，ρ为学习率。使用动量法能够减缓震荡，并有几率跳出局部最小，但是若参数选取不当，会导致接近最小值时由于动量的累加导致网络无法收敛。

对于 SGD 或动量法等优化方法，梯度在各个维度上的学习率是一致的，这会导致梯度较小的维度更新较慢。针对此问题，AdaGrad 算法提出根据自变量在每个维度的梯度值大小调整各个维度上的学习率，从而加速收敛。该算法采用二阶动量的方式对学习率进行动态约束，即第t次迭代的更新规则为：

$$g_t = \nabla W \tag{2-18}$$

$$G = \sum_t g_t^2 \tag{2-19}$$

$$W \leftarrow W - \frac{\eta}{\sqrt{G+\varepsilon}} \cdot g_t \tag{2-20}$$

其中η为全局学习率，ε为维持优化稳定引入的常数。

通过采用这种方式，可以控制更新频繁的神经元学习率惩罚较大而更新不频繁的神经元学习率惩罚较小从而对于稀疏数据能够有较好的优化效率。然而由于二阶动量惩罚的累加存在导致该方法对部分学习率惩罚趋近于 0 从而可能在未达到极小点的情况下提前终止算法。

RMSProp 算法同样使用二阶动量对学习率进行约束，但是与 AdaGrad 不同，RMSProp 积累了前 t 步二阶动量的加权平均，并以此对学习率进行约束，即第t次迭代的更新规则为：

$$g_t = \nabla W \tag{2-21}$$

$$s_t = \gamma s_t + (1-\gamma) g_t^2 \tag{2-22}$$

$$W \leftarrow W - \frac{\eta}{\sqrt{s_t+\varepsilon}} \cdot g_t \tag{2-23}$$

其中γ为二阶动量超参数，η为全局学习率，ε为维持优化稳定引入的常数。此方法能够对 AdaGrad 未达到极小点的情况下提前终止问题进行一定改善,但会引发震荡。

Adam 算法通过对上述算法进行综合吸取了上述算法的优点，通过使用一阶动量以及二阶动量的加权平均对学习率进行自适应约束，即第t次迭代的更新规则为：

$$g_t = \nabla W \tag{2-24}$$

$$v_t = \beta v_{t-1} + (1-\beta) g_t \tag{2-25}$$

$$s_t = \gamma s_t + (1-\gamma) g_t^2 \tag{2-26}$$

$$\hat{v}_t = \frac{v_t}{1-\beta^t} \tag{2-27}$$

$$\hat{s}_t = \frac{s_t}{1-\gamma^t} \tag{2-28}$$

$$W \leftarrow W - \frac{\eta}{\sqrt{\hat{s}_t + \varepsilon}} \cdot \hat{v}_t \tag{2-29}$$

其中β为一阶动量超参数，γ为二阶动量超参数，η为全局学习率，ε为维持优化稳定引入的常数。

2.2.3 梯度消失与梯度爆炸问题

随着网络层数的加深，当较深层梯度信息出现误差时，这种误差会随着链式法则反馈传回前面每一层网络。由于梯度的计算采用连乘形式，如果连乘的每个数字都小于 1，这会导致梯度在向前传播的过程中越传越小，最终趋于 0。此时部分靠前的网络神经元权值将无法更新，网络学习失效，此现象叫做梯度消失问题。同样，如果连乘的每一个数字都大于 1，会导致梯度越乘越大，最终导致部分靠前的网络神经元权值更新极具震荡，无法稳定收敛，此现象叫做梯度爆炸问题。梯度消失和梯度爆炸问题是深层网络训练困难的主要原因[35]，尽管 ReLU 激活函数相比其他激活函数能够缓解由激活函数带来的梯度消失问题，但是梯度消失和梯度爆炸问题仍会在训练过程中出现，并且 ReLU 激活函数对负梯度的抑制也会带来神经元死亡的问题，即负梯度经过 ReLU 函数后被抑制为 0，并且以后也不被任何函数激活，梯度永远为 0，不对任何数据响应。如果学习率设置不当，会导致部分神经元不可逆的死亡，从而导致训练失败。

为解决此问题，相关学者提出了梯度截断[36]，正则以及更换激活函数等方法[37]，但是这些方法往往缺乏泛用性并在实际训练中需要神经网络设计者人工干预调参。2015 年，Batch Normalization（BN）方法的提出使得由梯度问题导致的训练困难得到了一定缓解[37]。该方法通过在层与层之间依据一批数据（batch）训练一 BN 层，对每一层的特征进行归一化使其服从标准正态分布，同时 BN 层在训练过程中又引入线性变换使其能够对归一化的特征进行还原从而尽可能避免对网络精度的影响。假设为训练 BN 层输入的一批数据为从全部训练数据$\mathrm{X} = \{x_i, \mathrm{i} = 1,2,3 \ldots \mathrm{n}\}$中随机选取一批（batch）$k$个数据$\mathrm{X_k} = \{x_k | x_k \in X, 1 < k < n\}$，BN 层将学习一个线性变换的两个参数$\gamma$和$\beta$用于对归一化后的特征进行恢复即$\mathrm{BN}(\hat{x}_k) = \hat{y}_k$，其具体过程如下：

$$\mu = \frac{1}{k}\sum_{i=1}^{k} x_k \tag{2-30}$$

$$\sigma^2 = \frac{1}{k}\sum_{i=1}^{k}(x_k - \mu)^2 \tag{2-31}$$

$$\hat{x}_k = \frac{x_k - \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{2-32}$$

$$\hat{y}_k = \gamma \hat{x}_k + \beta \tag{2-33}$$

通过使用 Batch Normalization 方法，每一层网络输入数据的均值与方差均服从标准正态分布，使得后一层网络不再需要适应上层网络特征的尺度变化，从而使得梯度信息能过较好的进行传递。尽管相关研究表明通过引入 BN 层可以有助于改善损失函数得光滑性，但是由于 BN 层对于训练和测试时所进行的操作不同，使用 Batch Normalization 方法也可能带来模型精度降低的风险[39] 。

残差结构是解决深层网络训练困难的另一个方法，通过引入残差函数与跨连操作构成残差块对梯度信息进行补充以防止由梯度消失带来的误差[40] 。设输入x时第l层网络学习到的特征为$\mathrm{H}(x)$,当网络较深时，直接学习$\mathrm{H}(x)$会比较困难，由此构造残差函数$\mathrm{F}(x) = \mathrm{H}(x) - x$并希望该层网络学习残差$\mathrm{F}(x)$。由此可得原始特征$\mathrm{H}(x) = \mathrm{F}(x) + x$。

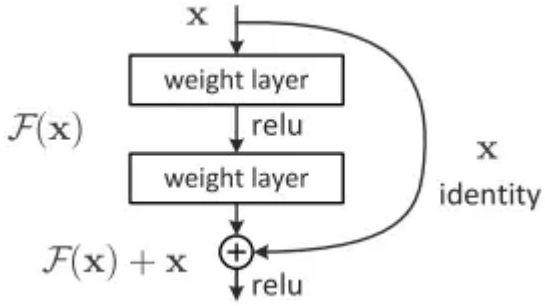


图 2-8　残差块结构[40]

Figure 2-8　　The structure of Residual block[40]

设x^l和x^{l+1}分别为第l个残差块的输入和输出，f为激活函数，$h(x^l) = x^l$为恒等映射，W^l为第l个残差块神经元权值由此可得

$$x^{l+1} = f(h(x^l) + F(x^l, W^l)) \tag{2-34}$$

而从第l层到第L层的特征为：

$$x^L = x^l + \sum_{i=l}^{l-1} F(x^i, W^i) \tag{2-35}$$

当反向传播时，第l个残差块神经元梯度为：

$$\frac{\partial \mathrm{Loss}}{\partial x^l} = \frac{\partial \mathrm{Loss}}{\partial x^L} \cdot \frac{\partial x^L}{\partial x^l} = \frac{\partial \mathrm{Loss}}{\partial x^L}\left(1 + \frac{\partial \sum_{i=l}^{l-1} F(x^i, W^i)}{\partial x^l}\right)$$

由于$\frac{\partial \sum_{i=l}^{l-1} F(x^i, W^i)}{\partial x^l}$不会全为-1，因此即使出现梯度消失问题，剩余梯度信息仍能通过$\frac{\partial \mathrm{Loss}}{\partial x^L}$进行回传。同时，当残差为 0 时，网络也仅仅做了恒等映射，从而并不会

导致性能的下降。

2.3 非反向传播优化算法

2.3.1 目标传播算法

在使用反向传播算法时，深层网络训练困难的问题源于梯度误差的积累，而由于梯度的连乘计算，误差会随着连乘放大。目标传播算法（Target Propagation）提供了一种不使用反向传播算法，不依赖链式法则以连乘形式传递误差信息的优化方法。该方法的误差值不通过梯度链形式传播，而是通过每一层的目标值（target）独立更新每一层网络的神经元权值[29] 。目标传播算法的核心在于目标值的获取，对于 L 层神经网络，记W_i为第i层网络神经元权值，h_i为第i层隐藏层输出值，f_i为第i层激活函数。对于第i层神经网络，目标传播算法通过训练一逆自动编码器g_i，满足：

$$g_i\left(V_i; f_i(W_i; h_{i-1})\right) \approx f_{i-1}(W_{i-1}; h_{i-2}) = h_{i-1} \tag{2-36}$$

用于获取误差对每一层目标值的影响情况。当每一层的逆自动编码器训练完成后，损失函数对每一层目标值的误差信息可以通过该层的逆自动编码器逐层计算。之后通过优化逐层损失函数

$$\underset{W_i}{argmin} L_i\left(\hat{h_i}, h_i\right) = \left\|\hat{h_i} - h_i\left(x; \theta_W^{0,i}\right)\right\|_2 \tag{2-37}$$

完成对每一层神经元的权值更新。

具体算法如下：

算法 目标传播算法

前向传播，对于 L 层神经网络:

for $i = 1$ to L do

$h_i \leftarrow f_i(W_i; h_{i-1})$ 其中$h_0 = X; h_L = \hat{Y}$

end for

计算全局损失函数 Loss($\hat{Y}, Y$)

计算末层目标值: $\hat{h}_L \leftarrow argmin_{h_L} Loss(h_L)$

逐层计算目标值:

for $i = L$ to 2 do

$\hat{h}_{i-1} \leftarrow h_{i-1} + g_i(\hat{h}_i) - g_i(h_i)$

end for

训练逆自动编码器:

for $i = L$ to 2 do

对隐藏层输出附加噪声 $\tilde{h}_{i-1} = h_{i-1} + \varepsilon$，$\varepsilon \sim N(0, \sigma)$

使用 SGD 更新逆自动编码器权值 V_i $\underset{V_i}{\operatorname{argmin}} \mathrm{L}_{g_i} = \left\| g_i\left(V_i; f_i\left(W_i; \tilde{h}_{i-1}\right)\right) - \tilde{h}_{i-1} \right\|_2$

end for

更新前馈传播神经元权值:

for $i = 1$ to L do

使用 SGD 更新神经网络权值 W_i $\underset{W_i}{\operatorname{argmin}} \mathrm{L}_i\left(\hat{h}_i, h_i\right) = \left\| \hat{h}_i - h_i\left(x; \theta_W^{0,i}\right) \right\|_2$ if $i < \mathrm{L}$, else

$\mathrm{L}_i = \mathrm{Loss}(y_i, t_i)$.

end for

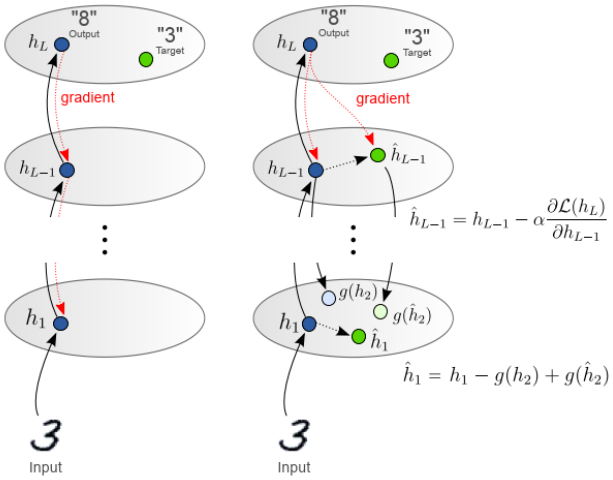


图 2-9　a)反向传播算法示意图。b)目标传播算法示意图。

Figure 2-9　a) Back propagation process　b) Target propagation process

2.3.2 多层梯度提升决策树

与目标传播算法类似，多层梯度提升决策树采用构造伪逆自动编码器的方式在不使用反向传播算法的前提下对误差信息进行传递。通过结合梯度提升决策树，模型可以允许存在一些不可微组件，使得中间层输出可以被视为有意义的分布式表示。其具体算法如下：

算法　多层梯度提升决策树

输入：层数 M，每层维度d_i，训练数据 X，Y，损失函数 L，$\alpha, \gamma, \mathrm{K}_1, \mathrm{K}_2$，批数 E 以及噪声$\sigma^2$

输出：多层梯度提升决策树

初始化映射 F 与 G；$\mathrm{o}_0 \leftarrow X; o_j \leftarrow F_j^0\left(o_{j-1}\right)$ for $j = 1,2 \ldots M$

for $t = 1$ to E do

$z_M^t \leftarrow o_M - \alpha \frac{\partial L(o_M, Y)}{\partial o_M}$ 计算末层目标值
for $j = M$ to 2 do
$G_j^t \leftarrow G_j^{t-1}$
$\mathrm{o}_{\mathrm{j}-1}^{noise} \leftarrow \mathrm{o}_{\mathrm{j}-1} + \varepsilon, \varepsilon \sim N(0, \sigma^2)$
for $k = 1$ to K_1 do
$L_j^{inv} \leftarrow \left\| G_j^t\left(F_j^{t-1}\left(\mathrm{o}_{\mathrm{j}-1}^{noise}\right)\right) - \mathrm{o}_{\mathrm{j}-1}^{noise} \right\|$
$\mathrm{r_k} \leftarrow -[\frac{\partial \mathrm{L_j^{inv}}}{\partial G_j^t\left(F_j^{t-1}\left(\mathrm{o}_{\mathrm{j}-1}^{noise}\right)\right)}]$
使用回归树拟合h_k为$\mathrm{r_k}$
$G_j^t \leftarrow G_j^t + \gamma \mathrm{r_k}$
end for
$z_{j-1}^t \leftarrow G_j^t(z_j^t)$ 计算$j-1$层目标值
end for
for $j = 1$ to M do
$F_j^t \leftarrow F_j^{t-1}$
使用目标值z_j^t更新$F_j^t$$\mathrm{K_2}$次
for $k = 1$ to $\mathrm{K_2}$ do
$L_j \leftarrow \left\| F_j^{t-1}(o_{j-1}) - z_j^t \right\|$
$\mathrm{r_k} \leftarrow -[\frac{\partial \mathrm{L_j^{inv}}}{\partial F_j^{t-1}(o_{j-1})}]$
使用回归树拟合h_k为$\mathrm{r_k}$
$F_j^t \leftarrow F_j^t + \gamma \mathrm{r_k}$
end for
$o_j \leftarrow F_j^t(o_{j-1})$
end for
end for

2.4 本章小结

通过本章的内容介绍，我们基本了解了多层神经网络的组成结构以及反向传播算法和基于反向传播算法的神经网络常用优化算法，包括梯度下降法，随机梯度下降，动量法，RMSProp，以及一系列自适应算法如 AdaGrad 以及 Adam。并对可能导致训练失败的原因做出介绍。同时，对一些非反向传播算法训练方式进行详细介绍，包括目标传播算法和多层梯度提升决策树。

第 3 章　基于 ReLU 激活函数的神经网络自适应优化算法

尽管深度神经网络在在一些智能任务中取得了优秀的表现，但是训练一个具有较好结果的神经网络往往需要不断手动调试，以寻找合适的训练超参数。其中学习率的选择对模型精度影响很大。如果损失函数位于局部极小附近，较大的学习率会导致震荡无法收敛。因此，传统的训练方法往往只能采用较小的学习率不断“小步试探”，这种训练策略会导致训练花费较多时间和算力，从而增大训练成本。

为了解决上述问题，本文通过分析使用 ReLU 网络的损失函数的形状与性质，从而提出以近似逆映射获取的目标值作为误差传递的载体进行网络拆解分层更新。此更新方式可结合凸优化方法解析求解每一层神经网络的自适应最优学习率，使得损失函数可以快速下降，提高收敛速度，降低人工调参成本。

本章将详细介绍基于 ReLU 激活函数的神经网络自适应优化算法，包括网络辅助结构，在线、 批（batch）、与线搜索三种更新策略以及其对应的最优学习率计算方法。在本章最后，我们将会给出基于 ReLU 激活函数的神经网络自适应优化算法的复杂度分析。

3.1　ReLU 网络基本结构及分解更新

3.1.1 神经网络的结构以及非凸性

对于一l层深度神经网络，给定拥有m个样本的训练数据集$\mathrm{D}=\{\mathrm{x_i},\mathrm{t_i}\}_{i=1}^{m}$，其中$\mathrm{x_i}\in\mathrm{R^n}$为输入数据而$\mathrm{t_i}$为该数据对应的标签值。设第 d 层网络拥有$\mathrm{n_d}$个神经元，对于第一层输入层来说记$\mathrm{n_0}=n$，我们希望学习的是一组神经元权值$\mathrm{W}=\{\mathrm{W_d}:\mathrm{W_d}\in\mathrm{R^{n_d*n_{d-1}}}\}_{d=1}^{l}$使得网络误差最小。在考虑网络每一层的激活函数都为 ReLU 函数的情况下，网络的输出可以表示为：

$$y_i=f(x_i;W)=W_l(ReLU(W_{l-1}(ReLU(W_{l-2}(\dots W_2(ReLU(W_1x_i)))))))\qquad(3\text{-}1)$$

我们定义第 d 层隐藏层的输出为$\mathrm{h_d}=\mathrm{RELU}(\mathrm{W_d}\cdot\mathrm{h_{d-1}})$，以及第$i$层和第$j$层之间

$(0 \le i < j \le \mathrm{l})$神经元权值集合$\theta_W^{i,j} = \{W_k, k = i+1, \ldots, j\}$，为简化记载，我们约定偏置$\mathrm{b_i}$以增广矩阵形式包含在$W_k$中。通过使用此种表示方法，网络的输出可以表示为：

$$y = f(x; W) = h_l(h_i(x; \theta_W^{0,i}); \theta_W^{i,l}) \tag{3-2}$$

我们的目标是训练一组合适的权值集合，使得全局损失函数$L(\mathrm{Y}, \mathrm{T}) = \sum_{i=1}^{m} L(y_i, t_i)$最小。其中$L$表示损失函数，Y表示神经网络输出集，T为对应的标签集。

引理 1：$f: R^n \to R$是凸的，当且仅当$g: R \to R, g(t) = f(x+tv), dom\ g = \{x + tv \in dom\ f\}$对任意$x \in dom\ f, v \in R^n$关于$t$是凸的。

由引理 1 可知，一个高维凸函数可以等价于若干个一维凸函数的叠加。现预证损失函数L非凸，利用引理 1，只要找到一点$(x, y) \in R^n \times R$，和一个方向向量v，构造g非凸，即可证明L非凸。实际上，在固定某一神经元的情况下，容易得到神经元权值的某一个维度上两个点w_{ia}和w_{ib}对应的损失函数值$L(w_{ia})$和$L(w_{ib})$的平均值大于这两点均值所对应的损失函数值，即$\frac{1}{2}\left(L(w_{ia}) + L(w_{ib})\right) > L\left(\frac{1}{2}(w_{ia} + w_{ib})\right)$。令$g = L(w_i)$，由此构造出一维函数$g$非凸，利用引理 1 可得损失函数$L$非凸。实际上，由于网络的全局损失函数是由多层带有 ReLU 激活函数的神经网络复合而成，尽管 ReLU 函数以及内积函数等函数是凸的，但是凸函数的复合函数并不一定是凸的。我们很容易在给定的样本空间 X 中找到一个样本使得损失函数对神经元权值的 Hessian 阵非正定，因此可得全局损失函数对于各神经元权值是非凸的。

3.1.2 ReLU 网络的分解更新

为获取自适应最优学习率，神经元权值的更新不能依赖于非凸的全局损失函数，借助目标传播算法思想，本文对深度神经网络构造辅助近似逆映射结构用于对网络进行分层，使其可以不使用全局损失函数直接对各层神经元进行独立更新，由此使用凸优化方法即可求出符合神经元当前状态的最优学习率。

神经网络的更新是通过将误差信息传递回每一层网络，从而计算每一层神经

网络各自的更新方向。目标传播算法提出误差信息的传播不仅仅限于梯度信息，而也可以是目标值信息，即该层神经网络输出值受误差影响后的估计值。误差以目标值为载体在神经网络中逐层传递，并使每一层神经网络可以利用目标值构造损失函数对该层神经元进行更新[29] 。如果我们能获得隐藏层h_i受损失函数误差传递影响后的目标值$\hat{h}_i$，那么可以通过构造 MSE 逐层损失函数计算该层神经元的更新方向以及步长并完成神经元更新。具体方式为通过构造逐层损失函数：

$$L_i(\hat{h}_i, h_i) = MSE(\hat{h}_i, h_i) = \left\| h_i(x; \theta_W^{0,i}) - \hat{h}_i \right\|_2 \tag{3-3}$$

并使用该损失函数对神经元求梯度，即：

$$\frac{\partial L_i(\hat{h}_i, h_i)}{\partial W_i} = \frac{\partial \left\| h_i - \hat{h}_i \right\|_2}{\partial W_i} = 2(h_i - \hat{h}_i)\frac{\partial h_i}{\partial W_i} \tag{3-4}$$

即可获得当前神经元的更新方向。由于损失函数$L_i(\hat{h}_i, h_i)$中，目标值$\hat{h}_i$可视为一个常数，因此通过此法获取的神经元更新方向并不依赖全局损失和梯度链。图 3-1 将以 3 层神经网络为例展示神经网络分解更新的流程。

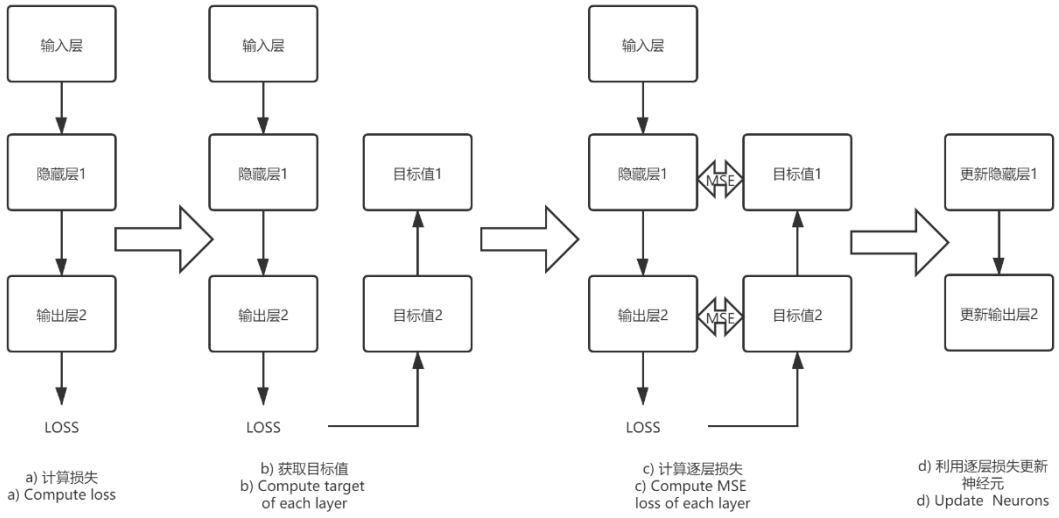


图 3-1　神经网络分解更新流程

Figure 3-1　The process of neural network decomposition and update

3.1.3 目标值的获取

隐藏层h_i所对应的目标值$\hat{h}_i$包含了损失函数的误差信息对当前隐藏层的影响，因此为获取每一层隐藏层对应的目标值，必须寻找一种能够完成误差“值传递”的方法。

现考虑第$i+1$层神经网络，其输入值为第i层神经网络的输出值h_i，经过本层

神经元权值W_{i+1}加工并完成激活后，输出值为$h_{i+1}=f_{i+1}(W_{i+1};h_i)$。此层神经网络的功能为加工第$i$层神经网络的输出结果$h_i$并将其传递至第$i+2$层神经网络作为输入数据。如若能够训练一种近似逆映射g_i，使得g_i能够将该层神经网络的输出值h_{i+1}尽可能还原为输入值h_i，那么对于第$i+1$层神经网络的目标值$\hat{h}_{i+1}$同样可以由g_i还原为第i层神经网络的目标值$\hat{h}_i$，由此便可完成目标值的逐层传递。具体来说，我们希望g_i能够使该层神经网络的输出值h_{i+1}尽可能还原为输入值h_i，即：

$$h_{i+1}=f_{i+1}(W_{i+1};h_i)\approx f_{i+1}(W_{i+1};g_i(V_i;h_{i+1})) \tag{3-5}$$

或

$$t_i=g_i\big(V_i;f_{i+1}(W_{i+1};h_i)\big)\approx f_i(W_i;h_{i-1})=h_i \tag{3-6}$$

当我们获得了近似逆映射g_i后，即可将第$i+1$层网络的目标值$\hat{h}_{i+1}$还原为第i层网络的目标值$\hat{h}_i$，为了尽可能减小由于近似逆映射所带来的扰动，我们定义目标值传递方式为：

$$\hat{h}_i=h_i+g_i\big(\hat{h}_{i+1}\big)-g_i(h_{i+1}) \tag{3-7}$$

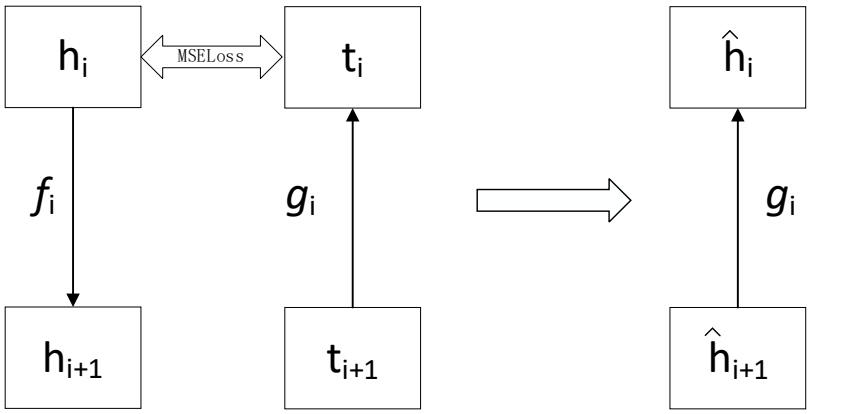
当我们由分类损失获得最末层目标值$\hat{\mathrm{h}}_l=argmin_{h_l}Loss(Y,T)=h_l-\frac{\partial Loss(Y,T)}{\partial h_l}$后，便可利用近似逆映射将包含误差信息的目标值逐层传递回网络的每一层当中。

3.1.4 近似逆映射的训练

由于神经网络具有强大的非线性拟合能力，因此我们可以构造神经网络用于拟合近似逆映射。我们定义近似逆映射g_i对应神经元为V_i，那么在训练近似逆映射g_i时，其输出值$t_i=g_i(h_{i+1};V_i)=ReLU(V_ih_{i+1})$应和第$i+1$层神经网络输入值$h_i$尽可能相似。由此我们可以构造 MSE 损失

$$L_{g_i}=\left\|g_i\big(V_i;f_{i+1}(W_{i+1};h_i+\varepsilon)\big)-(h_i+\varepsilon)\right\|_2 \tag{3-8}$$

其中ε为防止过拟合引入的一随机扰动。通过优化此损失函数，我们可以约束近似逆映射g_i的输出值t_i与h_i尽可能相似。从而使g_i有能力将h_{i+1}还原为h_i的近似值。当训练完成后，我们便可使用训练好的g_i将第$i+1$层目标值$\hat{h}_{i+1}$还原为$\hat{h}_i$。其训练及传递过程如图 3-2 所示：



a）近似逆映射的训练过程
a）The training process of approximate inverse mapping

b）目标值传递过程
b）The process of target propagation

图 3-2　近似逆映射

Figure 3-2　Approximate inverse mapping

由于近似逆映射训练过程中近似逆映射神经元V_i的训练依赖于第$i-1$层神经元权值W_{i-1}，因此在迭代更新时，V_i的训练将与W_i交替进行，在后续章节中将详述整体更新算法。

3.2　在线优化算法

当近似逆映射构造完成后，我们拥有神经网络每层目标值的估计值。考虑 MSE 逐层损失：

$$L_i(\hat{h}_i, h_i) = MSE(\hat{h}_i, h_i) = \frac{1}{N}\sum_{k=1}^{N}\left[f_{relu}(W_i h_{i-1;k}) - \hat{h}_{i;k}\right]^2 \qquad (3\text{-}9)$$

对于$\mathrm{L_i}(\hat{h}_i, h_i)$，$\hat{h}_i$可以视作一常数，与优化过程无关。记$f_{liner}(x) = Wx$，$ReLU(x)' = \begin{cases} 1 & , x > 0 \\ 0 & , x = 0 \end{cases}$，因此在忽略$x = 0$点情况下$ReLU(x)$为凸函数。$f_{liner}(x)$为仿射函数，因此其复合函数$ReLU\left(f_{liner}(x)\right) = \mathrm{ReLU}(Wx)$也为凸函数。令$g(x) = \mathrm{ReLU}\left(f_{liner}(x)\right)$，可得

$$\nabla g(x) = \begin{cases} x^T & , w_j x_k > 0 \\ 0 & , w_j x_k \le 0 \end{cases} \qquad (3\text{-}10)$$

因此有

$$\nabla L_i(\hat{h}_i, h_i) = 2\left[g(x) - \hat{h}_i\right]\nabla g(x) = \begin{cases} 2\left[g(x) - \hat{h}_i\right]x^T & , w_j x_k > 0 \\ 0 & , w_j x_k \le 0 \end{cases} \qquad (3\text{-}11)$$

$$\nabla^2 L_i(\hat{h}_i, h_i) = \begin{cases} 2xx^T, w_j x_k > 0 \\ 0 \quad , w_j x_k \le 0 \end{cases} \tag{3-12}$$

因为$\nabla^2 \mathrm{L_i}(\hat{h}_i, h_i) \geq 0$，可以判断逐层损失函数$\mathrm{L_i}(\hat{h}_i, h_i)$对神经元权值为凸函数，故我们可以使用一些凸优化的性质来分析最优学习率。

对于第i层神经网络，$W_i h_{i-1} = 0$构成其参数空间中的决策超平面，此超平面将参数空间分为两部分，我们定义 Pos 集为参数空间中满足$W_i h_{i-1} > 0$的部分，即：$Pos_{i;j,k} = \{h_{i-1;j,k} | w_{i;j} h_{i-1;k} > 0\}$。

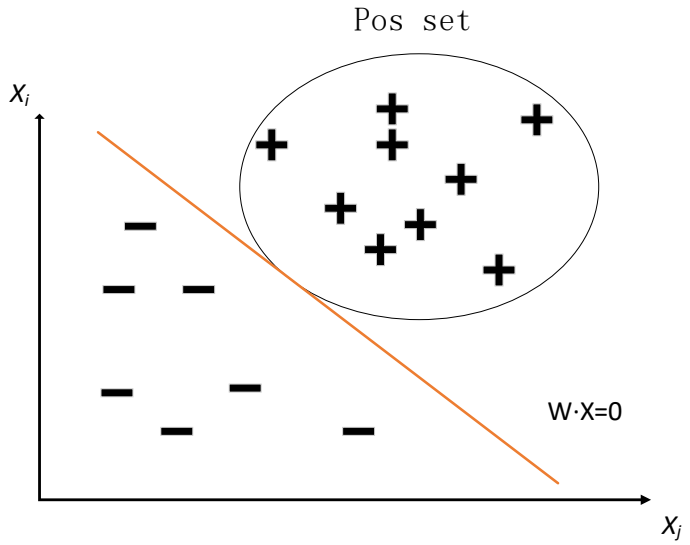


图 3-3　Pos 集

Figure 3-3 Pos Set

由于 ReLU 激活函数只会对 Pos 集的神经元进行激活并向后传值，并且只有 Pos 集的元素会对损失函数起贡献，因此如果 Pos 集不改变，此时损失函数形式也不变，由此可以采用凸优化方法中精确线性搜索法获得此时该层神经网络最优学习率的解析表达，即将损失函数分解为：

$$L_i = \sum_{j,k \in Pos} \left[\hat{h_i} - W_{i;j} h_{i-1;k}\right]^2 + \sum_{j,k \notin Pos} \hat{h_i}^2 \tag{3-13}$$

由于更新权值规则为$W_{i;j} \leftarrow W_{i;j} - \rho W_{i;j}'$，我们令$\mathrm{Loss}_i$对学习率ρ求导等于 0，得：

$$\frac{dLoss_i(\rho)}{d\rho} = \sum_{j,k \in Pos} -2[\hat{h_i} - (W_{i;j} - \rho W_{i;j}')h_{i-1;k}] \cdot W_{i;j}' h_{i-1;k} = 0 \tag{3-14}$$

可以解得此时最优学习率为：

$$\rho_{opt_{i;j}} = \frac{\sum_{j,k \in Pos}[\hat{h_i} - W_{i;j} h_{i-1;k}](W_{i;j}' h_{i-1;k})}{\sum_{j,k \in Pos}(W_{i;j}' h_{i-1;k})^2} \tag{3-15}$$

考虑计算效率，我们取j个神经元中最小的最优学习率为第i层神经网络最优学习率，即：$\rho_{opt_i} = \min_j \rho_{opt_{i;j}}$

我们已经分析了 Pos 集不改变时最优学习率的情况，但是当迭代使 Pos 集发生改变时，损失函数的形式也会发生改变，即激活的神经元发生改变，此时更新神经元权值可以视作决策超平面在参数空间上的移动，如果移动超过某一个 Pos 集中的样本点，那么该样本点便不再属于 Pos 集，此样本点对应的神经元也不再激活从而损失函数的形式也发生改变。因此使 Pos 集最先发生改变的步长即为最优步长，即当前决策超平面到离其最近的属于 Pos 集中的样本点的距离。

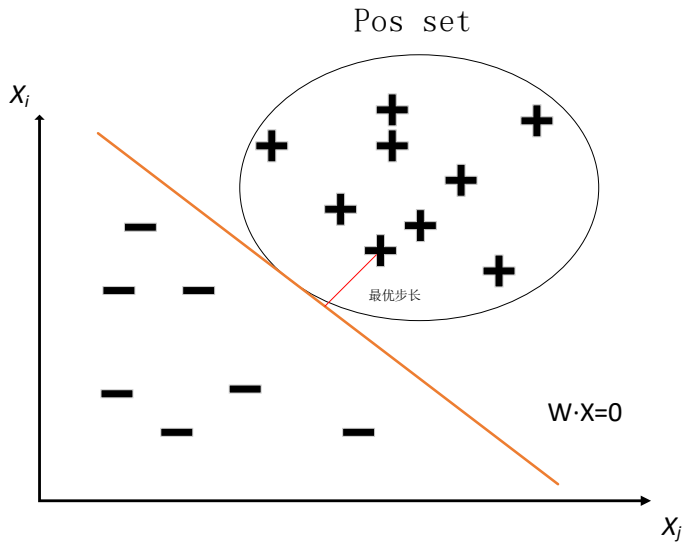


图 3-4　最优步长

Figure 3-4 Optimal step size

为获取使 Pos 集发生改变时的最优步长，我们通过解当前决策超平面到 Pos 集中所有点的距离的最小值来获得，即求解：

$$(W_{i;j} - \rho_j W_{i;j}')h_{i-1;k} = 0 \tag{3-16}$$

得第i层神经网络第j个神经元与第k个网络输入样本的距离为：

$$\rho_{i;j,k} = \frac{W_{i;j}h_{i-1;k}}{W_{i;j'}h_{i-1;k}} \tag{3-17}$$

第i层神经网络第j个神经元的最优学习率为：

$$\min_{\substack{k \\ j,k\in Pos}} \rho_{i;j,k} = \frac{W_{i;j}h_{i-1;k}}{W_{i;j'}h_{i-1;k}} \tag{3-18}$$

考虑计算效率，我们取当前j个神经元中最小的最有学习率为第i层神经网络神经元权值的最优学习率，即：

$$\rho_{online_i} \leftarrow \min_{j}\left(\min_{\substack{k \\ j,k\in Pos}} \frac{W_{i;j}h_{i-1;k}}{W_{i;j'}h_{i-1;k}}\right) \tag{3-19}$$

由于频繁验证 Pos 集是否发生改变将消耗较大算力，我们取

$$\rho_i = min(\rho_{opt_i}, \rho_{online_i}) \tag{3-20}$$

为第i层最优学习率。

由此我们获得了基于 ReLU 激活函数的神经网络的在线优化算法，其流程如下：

算法 在线优化算法

输入: 网络层数 L，每层网络维度d_i，训练数据 X，T，损失函数 Loss，K_1以及噪声σ^2

输出: 神经网络神经元权值集$\theta_W^{1,L}$

初始化各层神经元权值 W 与近似逆映射权值 V

前向传播，对于 L 层神经网络:

for $i = 1$ to L do

$h_i \leftarrow f_i(W_i; h_{i-1})$ 其中$h_0 = X; h_L = \hat{Y}$

end for

计算全局损失函数 Loss$(\hat{Y}, T)$

计算末层目标值: $\hat{h}_L \leftarrow argmin_{h_L} L(h_L)$

逐层计算目标值:

for $i = L$ to 2 do

$\hat{h}_{i-1} \leftarrow h_{i-1} + g_i(\hat{h}_i) - g_i(h_i)$

end for

训练近似逆映射:

for $i = L$ to 2 do

for *j=1 to K1*

计算近似逆映射损失:

$L_{g_i} = \left\| g_i(V_i; f_i(W_i; h_{i-1} + \varepsilon)) - (h_{i-1} + \varepsilon) \right\|_2$ ， $\varepsilon \sim N(0, \sigma^2)$

求 Pos 集，以及 Pos 集不改变时的最优学习率ρ_{optV_i}

求 Pos 集改变时的最优学习率$\rho_{onlineV_i}$

求V_i的最优学习率ρ_{V_i}，并使用此最优学习率对V_i进行更新，即:

$V_i \leftarrow V_i - \rho_{V_i} \nabla V_i$

end for

end for

更新前馈传播神经元权值:

for $i = 1$ to L do

计算逐层损失函数:

$L_i(\hat{h_i}, h_i) = \left\| \hat{h_i} - h_i(x; \theta_W^{0,i}) \right\|_2$ if $i < L$, else $L_i = \text{Loss}(\hat{Y}, T)$.

求 Pos 集，以及 Pos 集不改变时的最优学习率ρ_{optW_i}

求 Pos 集改变时的最优学习率$\rho_{onlineW_i}$

求W_i的最优学习率ρ_{W_i}，并使用此最优学习率对W_i进行更新，即:

$W_i \leftarrow W_i - \rho_{W_i} \nabla W_i$

end for

3.3　批优化算法

为改善优化速度，本文也提供了在线优化算法的批方法改进，即从全部训练数据$\mathrm{X}=\{x_i,\mathrm{i}=1,2,3\ldots\mathrm{n}\}$中随机选取一批$k$个数据$\mathrm{X_k}=\{x_k|x_k\in X,1<k<n\}$，每次更新使用这批数据的平均梯度作为优化方向用于更新权值和计算最优学习率。由于每次仅选用一小批数据，此算法能够较大提高优化算法的速度并降低算力需求。与随机梯度下降相似，尽管一小批数据的平均梯度并不一定是最优的下降方向，但是通过多次迭代该算法仍能达到较好的精度。算法详细流程如下：

算法 批优化算法

输入：网络层数 L，每层网络维度d_i，训练数据 X，T，损失函数 Loss，$\mathrm{K_1}$,批数 E 以及噪声σ^2

输出：神经网络神经元权值集$\theta_W^{1,\mathrm{L}}$

初始化各层神经元权值 W 与近似逆映射权值 V

for $t=1$ to E do

　前向传播，对于 L 层神经网络:

　for $i=1$ to L do

　　$\mathrm{h}_i\leftarrow f_i(W_i;h_{i-1})$　其中$h_0=X_t;h_L=\widehat{Y_t}$。$X_t$为由$X$中选取的一批 k 个数据

　end for

　计算全局损失函数 Loss($\widehat{Y_t},T_t$)

　计算末层目标值: $\hat{\mathrm{h}}_{\mathrm{L}}\leftarrow argmin_{h_L}L(h_L)$

　逐层计算目标值:

　for $i=L$ to 2 do

　$\hat{h}_{i-1}\leftarrow \mathrm{h}_{i-1}+g_i(\hat{h}_i)-g_i(h_i)$

　end for

　训练近似逆映射:

　for $i=L$ to 2 do

　　for *j=1 to K1*

　　计算近似逆映射损失:

　　$\mathrm{L}_{g_i}=\left\|g_i\left(V_i;f_i(W_i;h_{i-1}+\varepsilon)\right)-(h_{i-1}+\varepsilon)\right\|_2$，　　$\varepsilon\sim N(0,\sigma^2)$

　　求 Pos 集，以及 Pos 集不改变时的最优学习率$\rho_{\mathrm{opt}V_i}$

　　求 Pos 集改变时的最优学习率$\rho_{onlineV_i}$

　　求V_i的最优学习率ρ_{V_i}，并使用此最优学习率以及本批 k 个数据的平均梯度对V_i进行更新，即:

　　$V_i\leftarrow \mathrm{V}_i-\rho_{V_i}\frac{1}{k}\sum_{m=1}^{k}\nabla V_{i;m}$

　　end for

　end for

　更新前馈传播神经元权值:

　for $i=1$ to L do

计算逐层损失函数:

$\mathrm{L}_i(\widehat{h_i}, h_i) = \left\|\widehat{h_i} - h_i(x;\theta_W^{0,i})\right\|_2$ if $i < \mathrm{L}$, else $\mathrm{L}_i = \mathrm{Loss}(\widehat{Y_t}, T_t)$.

求 Pos 集，以及 Pos 集不改变时的最优学习率ρ_{optW_i}

求 Pos 集改变时的最优学习率$\rho_{onlineW_i}$

求W_i的最优学习率ρ_{W_i}，并使用此最优学习率以及本批 k 个数据的平均梯度对W_i进行更新，即:

$$W_i \leftarrow \mathrm{W}_i - \rho_{W_i}\frac{1}{k}\sum_{m=1}^{k}\nabla \mathrm{W}_{i;m}$$

end for

end for

3.4 线搜索优化算法

由于在线方法在计算最优学习率的过程中，需要计算 Pos 集以及$\frac{\sum_{j,k\in Pos}[\widehat{h_i}-W_{i;j}h_{i-1;k}](W_{i;j'}h_{i-1;k})}{\sum_{j,k\in Pos}(W_{i;j'}h_{i-1;k})^2}$，当矩阵较大时，可能会带来一定的计算成本，因此本文提供了一种通过线搜索方法试探最优学习率的近似方法。线搜索方法是凸优化中一种启发式算法，即当获取精准的搜索步长比较困难时，采用试探所得满足函数值下降的最大的非精确最优步长进行迭代更新。利用非精确线搜索思想，我们可以由较大步长开始试探寻找能够保证目标函数值下降的最大步长，并以此步长为近似最优学习率进行更新。由于逐层损失$\mathrm{L_i}(\hat{h}_i, h_i)$为凸函数，因此只要试探适当的学习率保持逐层损失下降即可保证算法的收敛性。当样本维度较大时，采用线搜索优化算法从一个较大的步长开始试探直至获得保证损失函数下降的步长可能能够获得相较于精确计算最优步长更快的表现。使用该算法不需要计算精确的最优步长，只需不断试探并计算损失函数直到找到满足逐层损失下降的学习率即可，算法流程如下：

算法 线搜索优化算法

输入: 网络层数 L，每层网络维度d_i，训练数据 X，T，损失函数 Loss，K_1以及噪声σ^2

输出: 神经网络神经元权值集$\theta_W^{1,\mathrm{L}}$

初始化各层神经元权值 W 与近似逆映射权值 V;初始化学习率ρ_{W_i}与ρ_{V_i}

前向传播，对于 L 层神经网络:

for $i = 1$ to L do

$\mathrm{h}_i \leftarrow f_i(W_i; h_{i-1})$ 其中$h_0 = X; h_L = \hat{Y}$

end for

计算全局损失函数 Loss($\hat{Y}, T$)

计算末层目标值: $\hat{\mathrm{h}}_{\mathrm{L}} \leftarrow argmin_{h_L} L(h_L)$

逐层计算目标值:

for $i = L$ to 2 do

$\hat{h}_{i-1} \leftarrow \mathrm{h}_{i-1} + g_i(\hat{h}_i) - g_i(h_i)$

end for

训练近似逆映射:

for $i = L$ to 2 do

for *j=1 to K1*

$\bar{V}_i \leftarrow V_i$

计算更新前近似逆映射损失:

$\mathrm{L}_{g_{i\ before}} = \left\| g_i\left(V_i; f_i(W_i; h_{i-1} + \varepsilon)\right) - (h_{i-1} + \varepsilon) \right\|_2$,　　$\varepsilon \sim N(0, \sigma^2)$

do

$\rho_{V_i} \leftarrow \frac{\rho_{V_i}}{2}$

使用ρ_{V_i}更新$\bar{V}_i \leftarrow \mathrm{V}_i - \rho_{V_i} \nabla V_i$

计算更新后的近似逆映射损失:

$\mathrm{L}_{g_{i\ after}} = \left\| g_i\left(\bar{V}_i; f_i(W_i; h_{i-1} + \varepsilon)\right) - (h_{i-1} + \varepsilon) \right\|_2$,　　$\varepsilon \sim N(0, \sigma^2)$

While $\mathrm{L}_{g_{i\ after}} > \mathrm{L}_{g_{i\ before}}$

接受更新$V_i \leftarrow \bar{V}_i$

end for

end for

更新前馈传播神经元权值:

for $i = 1$ to L do

$\bar{W}_i \leftarrow W_i$

计算更新前逐层损失函数:

$\mathrm{L}_{i\ before}(\hat{h}_\iota, h_i) = \left\| \hat{h}_\iota - h_i(x; \theta_W^{0,i}) \right\|_2$ if $i < \mathrm{L}$, else $\mathrm{L}_i = \mathrm{Loss}(\hat{Y}, T)$.

do

$\rho_{W_i} \leftarrow \frac{\rho_{W_i}}{2}$

使用ρ_{V_i}更新$\bar{W}_i \leftarrow \mathrm{W}_i - \rho_{W_i} \nabla \mathrm{W}_i$

计算更新后的近似逆映射损失:

$\mathrm{L}_{i\ after}(\hat{h}_\iota, h_i) = \left\| \hat{h}_\iota - h_i(x; \theta_W^{0,i}) \right\|_2$

While $\mathrm{L}_{i\ after}(\hat{h}_\iota, h_i) > \mathrm{L}_{i\ before}(\hat{h}_\iota, h_i)$

接受更新$W_i \leftarrow \bar{W}_i$

end for

3.5　复杂度分析

由于本章所提优化算法的收敛速度与参数空间中样本点的分布密度有关，为

了分析本章所提出的优化算法的复杂度，我们假设第i层隐藏层h_i拥有n_d个服从U(0,1)分布的神经元。假设输入m个样本，在h_i的参数空间中则有m个均匀分布于n_d维单位球中的样本点。在此假设下，平均最优学习率可以等价为求该单位球中任意两点之间的平均距离。为了计算该单位球中任意两点之间的平均距离，我们设其中一点为球心，此时单位球中任意两点之间的平均距离等于球心离最近点的距离。

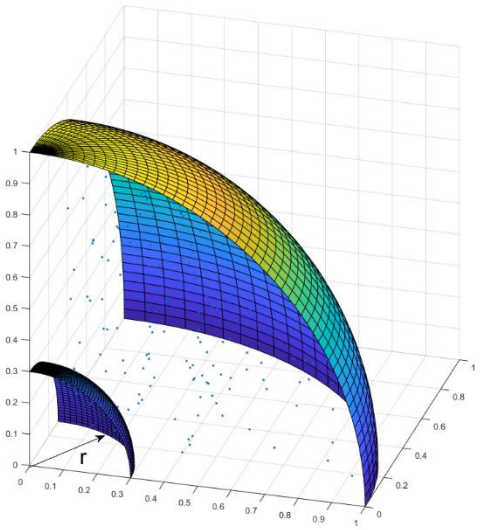


图 3-5　参数空间中最优步长示意图

Figure 3-5　Optimal step size in parameter space

记单位球 O 内离球心最近点到球心的距离为r，此时以r为半径的n_d维单位球的同心球 Q 体积为$V = \mathrm{C} \cdot r^{n_d}$，C为常数。可知一样本点落在单位球 O 内且落在球 Q 外的概率P为：

$$P = \frac{C \cdot [1^{n_d} - r^{n_d}]}{C \cdot 1^{n_d}} \tag{3-21}$$

则m个样本点全落入单位球 O 内且落在球 Q 外的概率P^m为：

$$P^m = \left[\frac{C \cdot [1^{n_d} - r^{n_d}]}{C \cdot 1^{n_d}}\right]^m = (1 - r^{n_d})^m \tag{3-22}$$

此时m个样本点任意两点之间的平均距离E(r)为：

$$E(r) = \int_0^1 r \cdot (1 - r^{n_d})^m \, dr \tag{3-23}$$

由于积分较难获得显示表达，我们使用中值距离作为平均距离的近似替代，即令

$$(1 - r^{n_d})^m = \frac{1}{2} \tag{3-24}$$

可解得：

$$r = \left(1 - 2^{-\frac{1}{m}}\right)^{\frac{1}{n_d}} \tag{3-25}$$

由此获得m个样本点任意两点之间的平均距离，即第i层隐藏层h_i平均最优学习率与样本个数m以及神经元个数n_d的关系，随着神经元个数n_d的增大，最优步长也会跟随迅速增大，由于凸优化收敛步数与步长成反比，由此可得此优化方法平均复杂度为$O(\frac{1}{r})$，即随着神经元个数n_d的增大，最优学习率也跟随增大，单次迭代所下降的距离增大最终使得算法更快收敛。由于神经元的更新可以视为决策超平面在参数空间上的移动，在最坏情况下，即使点分布不均匀，神经元完成更新也只需对参数空间内每个点进行试探，因此最坏情况的复杂度也不会超过$O(m)$。

3.6　本章小结

本章详细介绍了基于 ReLU 激活函数的神经网络自适应优化算法，包括网络结构，在线，批（batch），与线搜索三种更新方式以及其对应的最优学习率计算方法。通过训练近似逆映射，神经网络可以拆分并逐层独立优化，从而将复杂的非凸优化问题转化成求解若干凸优化的组合。通过分析 ReLU 网络损失函数性质，本章提出了最优学习率的计算方法，并给出了三种具体的神经元更新流程。同时，本章给出了基于 ReLU 激活函数的神经网络自适应优化算法的复杂度分析，通过推导最优步长与神经元个数和样本个数的关系得出了算法的复杂度，在样本服从均匀分布的情况下，本算法的平均复杂度为$O(\frac{1}{r})$，并且在极端情况下复杂度也不会超过$O(m)$，其中r为平均学习率而m为样本个数。

第 4 章　实验及结果分析

通过第三章对算法的复杂度分析，我们可以得知本文所提出的自适应优化算法对样本个数少，维度高的数据集应具有较好的优化效率，为验证本文第三章所提出的自适应优化算法，本章选取 UCI soybean、MNIST 数据集以及 CIFAR-10 数据集构造对比试验以验证所提出算法的性能，同时为了评估不同优化算法的效率和优势，本文设计了多组对比试验对梯度下降算法，随机梯度下降，目标差传播算法以及本文提出的自适应在线算法，批（batch）算法和线搜索算法进行比较。本章将详细描述了实验的数据集，实验环境，网络结构以及实现方式。并针对不同实验结果进行分析，得出实验结论。

4.1　数据集及网络结构

为验证本文所提出的三种基于 ReLU 激活函数的自适应优化算法的性能以及收敛速度，由于样本数量和神经网络结构均会对优化算法性能产生影响，本文使用了 UCI soybean 数据集、MNIST 数据集以及 CIFAR-10 数据集配合不同的网络结构以设计对比试验用于验证本文提出算法在不同数据集以及网络结构下的表现。本小节将会对数据集及网络结构做出详细介绍。

4.1.1 UCI soybean 数据集

UCI 数据集是加州大学欧文分校（University of CaliforniaIrvine）提出的一个著名的机器学习数据集[41]，其中 soybean 数据集是源自 R.S. Michalski 等人对大豆疾病专家系统研究所获取的实验数据[42]，UCI 对 soybean 数据提供 large 和 small 两个版本的数据集，由于 large 数据集包含一些不适用的属性，本次实验选用的 small 数据集包含清洗完成后的 47 个样本，其中每个样本拥有 35 个属性值用来描述大豆的属性，例如根茎叶的形状等。这些样本将会被分为 4 个类别，分别对应 4 种不同的大豆疾病。由于此数据集样本点较少，我们设计对比试验用于验证相对于其他算法，本文提出的在线方法以及线搜索方法的加速效果。

我们尽可能均匀随机选取 7 个样本用于测试模型表现，剩余 40 个样本用于

训练神经网络模型。由于模型较小，任务复杂性相对较低，因此用于预测的神经网络由带有 2 层分别具有 100 个神经元的隐藏层的神经网络构成。其网络结构如图 4-1 所示：

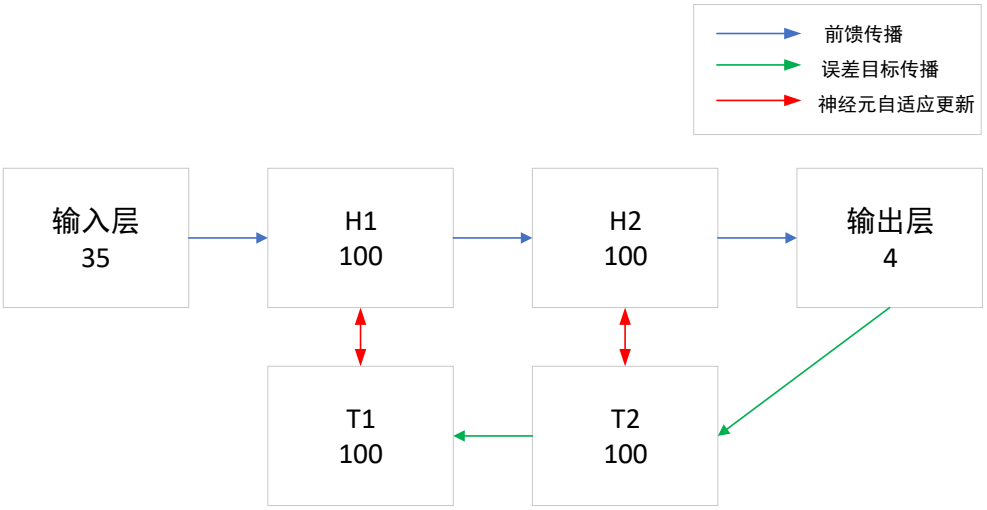


图 4-1　UCI soybean 数据集分类网络结构

Figure 4-1　The classification network for UCI soybean dataset

4.1.2 MNIST 手写数字数据集

MNIST 手写数字数据集是机器学习领域中的一个常见数据集，由 60000 个训练样本和 10000 个测试样本组成，每个样本都是一张 28×28 像素的灰度手写的 0 至 9 数字图片。其样例如图 4-2 所示：

图 4-2　MNSIT 数据集样例

Figure 4-2　Sample of MNIST dataset

对于单张样本图片，我们将 28×28 像素的灰度图转换成一 784 维向量作为输入向量。由于此数据集样本点较多，我们设计对比试验用于验证批（batch）方法的性能。对于此分类任务，本文设计一 3 层分别具有 500 个神经元作为隐藏层的神经网络用于训练分类模型，其神经网络结构如图 4-3 所示：

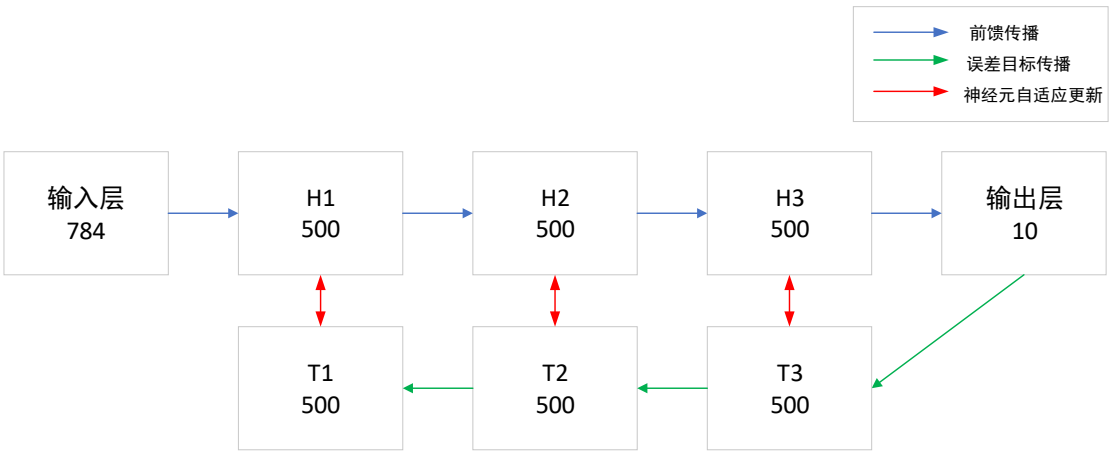


图 4-3　MNIST 数据集分类网络结构

Figure 4-3　The classification network for MNIST dataset

4.1.3 CIFAR-10 数据集

CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型图片数据集。一共包含 10 个类别的 RGB 彩色图片，分别为：飞机、汽车、鸟类、猫、鹿、狗、蛙类、马、船和卡车。与 MNIST 数据集不同，CIFAR-10 数据集每张图片为一张三通道分辨率为 32×32 的真实世界物体的彩色图片，不仅噪声很大，而且物体的比例、 特征都不尽相同，这也导致对这些图片进行分类具有较大的挑战性。数据集中一共有 50000 张训练图片和 10000 张测试图片。其样例如图 4-4 所示：

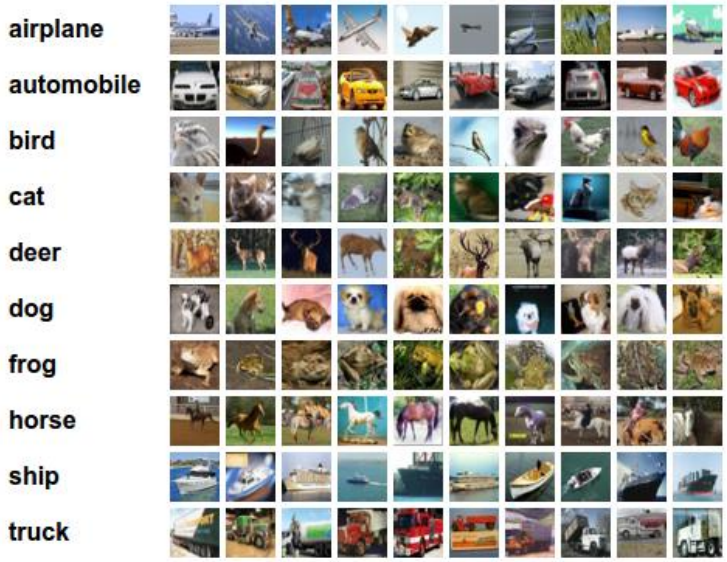


图 4-4　CIFAR-10 数据集样例

Figure 4-4　Sample of CIFAR-10 dataset

由于原始三通道图片难以直接输入全连接 ReLU 网络，我们将尺寸为 3×32×32 的原始图片转换成一 3072 维向量作为输入向量。由于此数据集样本点较多，我们设计对比试验用于验证批（batch）方法的性能。对于此分类任务，本文设计一 3 层分别具有 500 个神经元作为隐藏层的神经网络用于训练分类模型，其神经网络结构如图 4-5 所示：

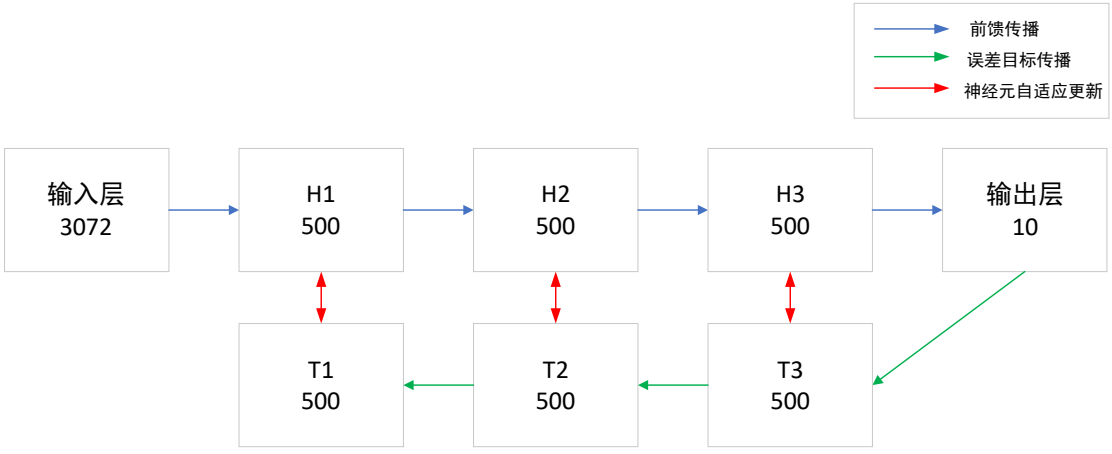


图 4-5　CIFAR-10 数据集分类网络结构

Figure 4-5　The classification network for CIFAR-10 dataset

4.2　实验环境

本文提出的基于 ReLU 激活函数的神经网络自适应优化算法的多种实现均依

赖于 Python 环境实现，其中梯度的计算依赖 Pytorch 框架的自动微分模块进行实现。Pytorch 是一款动态构建的深度学习框架，并对 Numpy 库具有良好的支持性。同时，Pytorch 自动微分模块提供了 GPU 并行张量计算的基本操作，这为我们的实验提供了工程上的便利。其余工程组件均由 Numpy、Theano 等 python 库构建。

本文所使用的实验设备如表 4-1 所示：

表 4-1　实验环境标准

Table 4-1　The standards of experimental environment

实验环境	配置
CPU	Intel i5-4590
GPU	Nvidia GTX-650
CUDA 版本	8.0
Python 版本	Python3.5
系统版本	Ubuntu16.04LTS

4.3　UCI soybean 数据集分类实验结果及对比

本实验在对图 4-1 所示的网络结构进行构建后，将训练数据输入神经网络。其中样本的标签以 one-hot 编码形式输入，即对于每一标签值t_i，构造一长度为标签种类数 4 的向量l，将此向量第t_i分量值设为 1，其余分量值设为 0。全局损失函数采用交叉熵损失，即$H(\mathrm{Y},\mathrm{T}) = -\sum_{i=1}^{40}\sum_{j=1}^{4} l_{ij}\, log\, y_{\mathrm{ij}}$。由于$l_{ij}$只有第$j = t_i$时才为 1，其余时为 0，因此我们只需计算交叉熵为$H(\mathrm{Y},\mathrm{T}) = -\sum_{i=1}^{40} log\, y_{it_j}$。

我们以精度（accuracy）和损失函数（loss）下降速度评判网络性能，其中精度的计算方式如下：

$$accuracy = \frac{分类正确样本数}{总样本数} \qquad (4\text{-}1)$$

在有限的迭代次数（epoch）内，精度越高，损失函数下降越快证明算法性能越好。

我们对比了梯度下降法、目标差传播算法以及本文提出的自适应在线优化算法以及线搜索算法的精度和收敛速度，通过 300 次迭代实验，不同算法的精度及损失下降情况如图 4-6 和 4-7 所示：

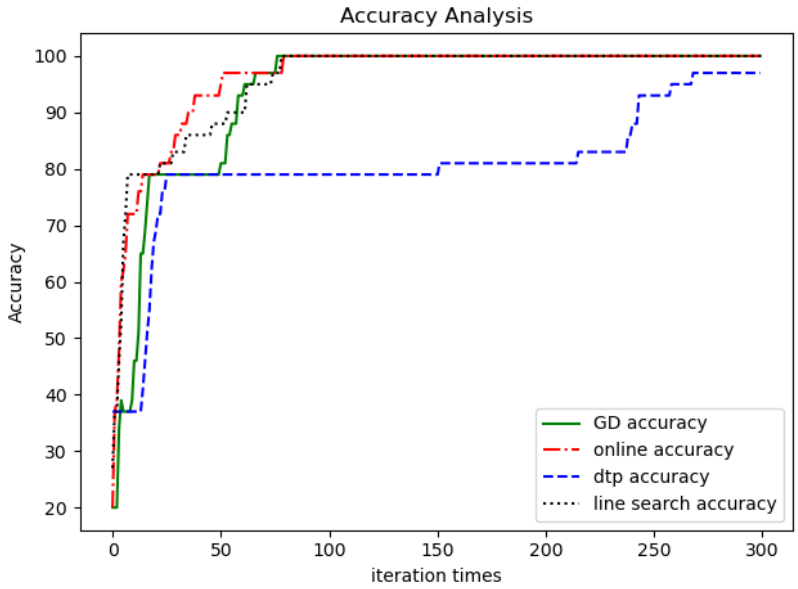


图 4-6　UCI soybean 数据集精度变化曲线

Figure 4-6　The accuracy curve of UCI soybean dataset

图中 GD accuracy 为采用梯度下降算法的精度变化曲线，online accuracy 为采用本文提出在线自适应优化算法的精度变化曲线，dtp accuracy 为采用目标差传播算法的精度变化曲线，line search accuracy 为采用本文提出线搜索自适应算法的精度变化曲线。可以看出，本文所提出的在线自适应优化算法与线搜索自适应优化算法在第 76 次迭代后均可以达到和梯度下降算法相同的精度结果，并在前 50 次迭代精度上升更快。

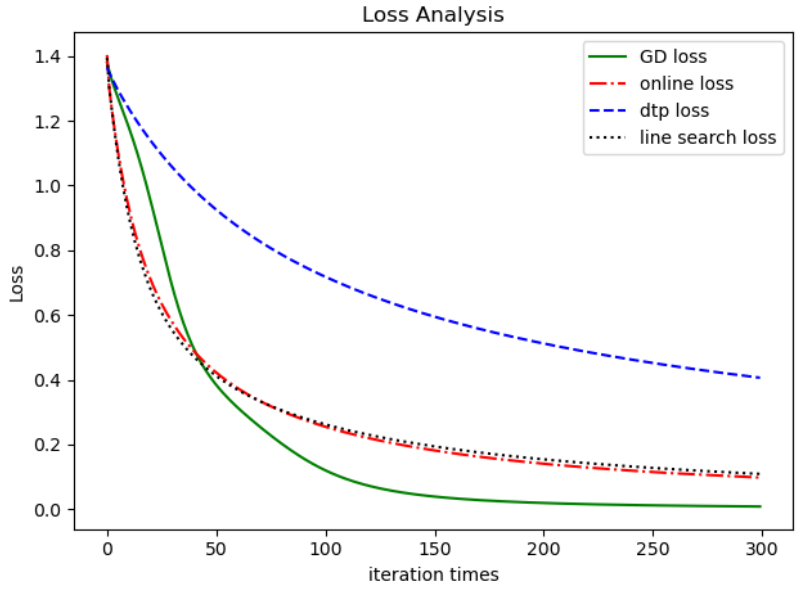


图 4-7　UCI soybean 数据集损失下降曲线

Figure 4-7　The loss curve of UCI soybean dataset

图中 GD loss 为采用梯度下降算法的损失函数变化曲线，online loss 为采用本文提出在线自适应优化算法的损失函数变化曲线，dtp loss 为采用目标差传播算法的损失函数变化曲线，line search loss 为采用本文提出线搜索自适应算法的损失函数变化曲线。可以看出，在前 50 次迭代本文提出在线自适应优化算法以及线搜索自适应优化算法损失函数下降速度更快。由于优化目标不同，在优化完成时，损失函数会有一定的数值差异，但结合精度变化曲线分析图中所示四种算法均已收敛。

训练完成后，使用测试集对所训练的模型进行验证时不同优化算法的精度表现如表 4-2 所示：

表 4-2 UCI soybean 数据集训练集及测试集精度结果

Table 4-2 The accuracy results of UCI soybean dataset

	梯度下降	目标差传播	自适应在线方法	自适应线搜索方法
训练集	100%	97%	**100%**	**100%**
测试集	100%	85.71%	**100%**	**100%**

4.4　MNIST 手写数字数据集分类实验结果及对比

本实验在对图 4-3 所示的网络结构进行构建后，将训练数据输入神经网络。其中样本的标签以 one-hot 编码形式输入，全局损失函数采用交叉熵损失，即 $H(\mathrm{Y},\mathrm{T}) = -\sum_{i=1}^{50000} log\, y_{it_j}$。由于 MNIST 数据集样本量较多，本次实验采用自适应批（batch）方法加速训练并验证不同批方法对于优化效果的影响。

我们对比了随机梯度下降法、目标差传播算法以及本文提出的自适应批（batch）优化算法的精度和收敛速度，在本实验中，batch-size 设置为 500，即每次更新使用 500 个样本进行训练，完成一轮使用全部训练集的训练总共需要迭代 100 次。训练轮次设定为 300 次，我们同样对比不同算法的精度和损失下降变化用于衡量算法性能，其变化情况如图 4-8 和 4-9 所示：

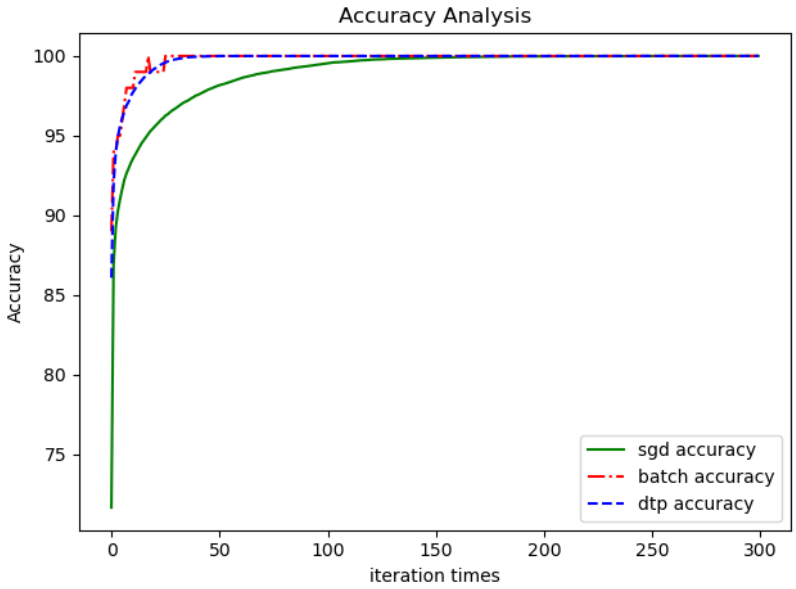


图 4-8　MNIST 数据集精度变化曲线

Figure 4-8　The accuracy curve of MNIST dataset

图中 sgd accuracy 为随机梯度下降算法的精度变化曲线，batch accuracy 为本文提出自适应批优化方法的精度变化曲线，dtp accuracy 为目标差传播算法的精度变化曲线。可以看出，三种算法在有限次迭代内均可达到近乎相同的精度水平，但是本文所提出的自适应批优化方法达到稳定精度水平所需的迭代次数更少。

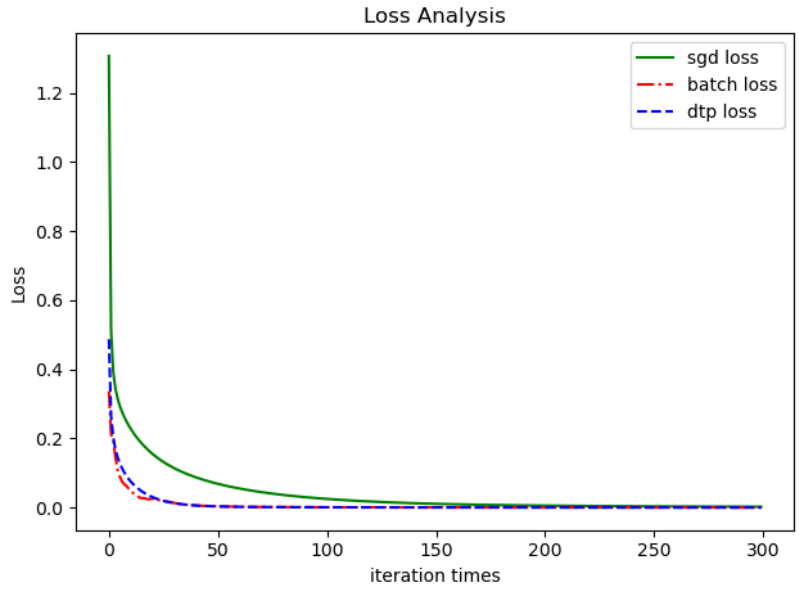


图 4-9　MNIST 数据集损失下降曲线

Figure 4-9　The loss curve of MNIST dataset

图中 sgd loss 为随机梯度下降算法的损失函数变化曲线，batch loss 为本文

提出自适应批优化方法的损失函数变化曲线，dtp loss 为目标差传播算法的损失函数变化曲线。可以看出，三种算法在有限次迭代内损失函数均下降到稳定数值，但是本文所提出的自适应批优化方法损失函数下降速度更快，所需迭代次数更少。

训练完成后，使用测试集对所训练的模型进行验证时不同优化算法的精度表现如表 4-3 所示：

表 4-3 MNIST 数据集训练集及测试集精度结果
Table 4-3 The accuracy results of MNIST dataset

	随机梯度下降	目标差传播	自适应批方法
训练集	100%	10%	**100%**
测试集	97.97%	97.97%	**98.06%**

4.5　CIFAR-10 数据集分类实验结果及对比

本实验在对图 4-5 所示的网络结构进行构建后，将训练数据输入神经网络。其中样本的标签以 one-hot 编码形式输入，全局损失函数采用交叉熵损失，即 $H(\mathrm{Y},\mathrm{T}) = -\sum_{i=1}^{50000} log\, y_{it_j}$。由于 CIFAR-10 数据集样本量较多，单个样本维度较高，本次实验将着重关注自适应批（batch）方法在相对复杂的数据集下的表现，并与随机梯度下降以及目标差传播算法的性能进行对比。

在本实验中，batch-size 设置为 500，即每次更新使用 500 个样本进行训练，完成一轮使用全部训练集的训练总共需要迭代 100 次。训练轮次设定为 300 次，由于本数据集较为复杂，单纯使用全连接构成的深度神经网络获得的精度有限，我们将着重关心在有限迭代次数内不同算法在精度以及损失函数下降速度上的变化用于衡量其性能。具体变化情况如图 4-10 和 4-11 所示：

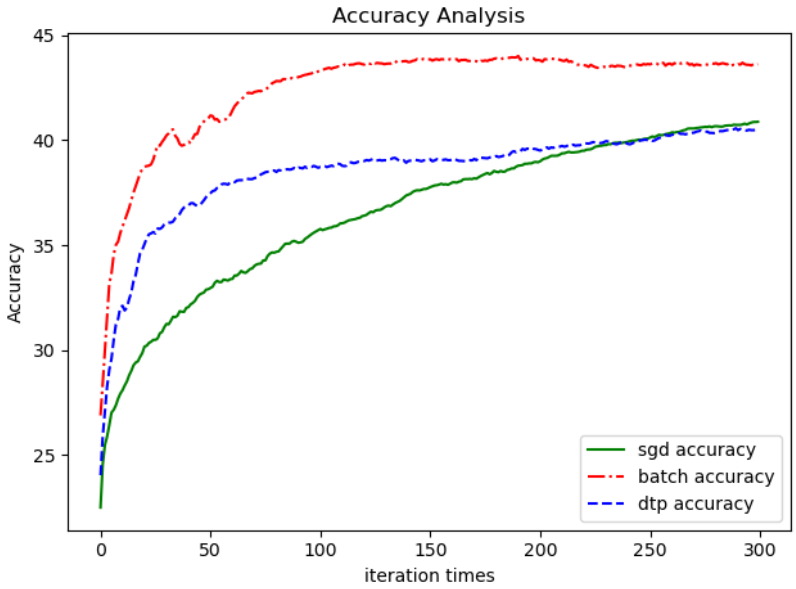


图 4-10　CIFAR-10 数据集精度变化曲线

Figure 4-10　The accuracy curve of CIFAR-10 dataset

图中 sgd accuracy 为随机梯度下降算法的精度变化曲线，batch accuracy 为本文提出自适应批优化方法的精度变化曲线，dtp accuracy 为目标差传播算法的精度变化曲线。可以看出，本文所提出的自适应批方法能够在较短迭代批次内获取相较于随机梯度下降以及目标差传播算法更高的精度。

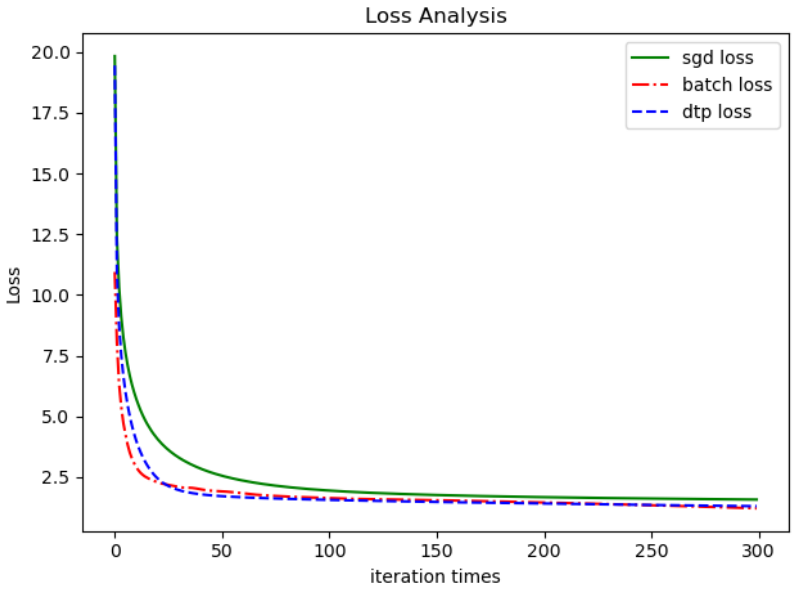


图 4-11　CIFAR-10 数据集损失下降曲线

Figure 4-11　The loss curve of CIFAR-10 dataset

图中 sgd loss 为随机梯度下降算法的损失函数变化曲线，batch loss 为本文提出自适应批优化方法的损失函数变化曲线，dtp loss 为目标差传播算法的损失

函数变化曲线。可以看出，三种算法在有限次迭代内损失函数均下降到稳定数值，但是本文所提出的自适应批优化方法损失函数下降速度更快，所需迭代次数更少,在训练初期尤为明显。

训练完成后，使用测试集对所训练的模型进行验证时不同优化算法的精度表现如表 4-4 所示：

表 4-4 CIFAR-10 数据集训练集及测试集精度结果
Table 4-4 The accuracy results of CIFAR-10 dataset

	随机梯度下降	目标差传播	自适应学习率 Batch 方法
训练集	50.44%	50.25%	**53.75%**
测试集	40.87%	40.56%	**44.00%**

受限于全连接网络性能限制，三种算法在 CIFAR-10 数据集上整体精度均较低。其中随机梯度下降与目标差传播性能大致相同，这与 Bengio 关于目标差传播算法的结论一致。本文所提出的自适应批方法所得结果在精度上优于随机梯度下降算法 3.13%以及优于目标差传播算法 3.44%，同时收敛速度也优于随机梯度下降算法以及目标差传播算法，这也验证了本文所提自适应优化算法在复杂数据集下也有优于随机梯度下降算法以及目标差传播算法的表现。

4.6　实验结果分析

上述实验可以验证本文提出的基于 ReLU 网络自适应优化算法能在不降低精度的前提下有效加快收敛速度，在部分情况下能够获得优于随机梯度下降的精度水平。通过观察精度曲线以及损失下降曲线可知，本文所提出的自适应优化算法在训练初期可使损失函数快速下降，模型精度快速上升。这与本文所提出“自适应最优步长”理论相符，由于模型在训练初期不再需要“小步试探”而可以每次精确以最优步长进行优化，这使得模型初期的训练加速效果十分明显。

尽管使用本方法需要额外的辅助结构，这意味着对内存或显存的额外占用以及更多的待训练参数，然而，通过本方法可以在不指定学习率的情况下完成深度神经网络的训练，这在网络训练的过程中将会降低训练的经验依赖和人工调试的成本。

综上所述，通过使用本文提出的自适应优化算法，训练神经网络不再需要不断尝试学习率。由于不使用反向传播算法，本文所提出的算法避免了梯度信息由于使用链式法则传递导致的一系列更新失败问题。本文提出的自适应更新算法较为新颖，能够有效自适应加速全连接 ReLU 网络的训练，实验结果与理论分析相符。

4.7 本章小结

本章详细介绍了本文所提出的基于ReLU网络自适应优化算法在UCI soybea数据集 MNIST 数据集以及 CIFAR-10 数据集下的实验过程和实验结果。本章节介绍了实验的具体细节，如网络结构、实验环境等，并最终对实验结果做出了对比分析。通过与其他主流优化算法对比，验证了本文所提出的三种自适应优化算法，即在线方法，批（batch）方法以及线搜索方法均能够获得不亚于随机梯度下降或梯度下降的优化精度，同时提高收敛速度。

结 论

本文提出了一种基于 ReLU 激活函数的神经网络自适应学习率更新方法。该方法不依赖反向传播算法，通过拆分网络独立更新每一层神经元权值，将复杂的非凸优化问题转化成求解若干凸优化问题的组合。对于每一层独立更新的神经网络，本文所提出的自适应优化算法通过分析损失函数形状，能够借助凸优化方法解析计算最优学习率从而使训练不再依赖学习率设置的经验。由于本文所提出自适应优化算法在误差传递过程中不再依赖链式法则，因此避免了由链式法则导致的梯度传递问题。同时，本文分析了每一个独立更新的网络层最优学习率的计算方法，并给出了两种改进加速算法。并通过实验验证本文所提出的算法在不需要手动设置学习率的前提下拥有不亚于随机梯度下降算法的精度，以及对于个数少维度高的样本具有优于随机梯度下降以及目标差传播算法的收敛速度。

对于本文所提出的基于 ReLU 激活函数的神经网络自适应优化算法，主要创新点如下：

（1）本文提出了一种不依赖反向传播算法的神经网络更新算法，这种算法在误差传递过程中不依赖梯度传递的链式法则，从而避免了由于链式法则带来的梯度消失与梯度爆炸等问题。

（2）本文所提出的更新算法能够独立更新每一层神经网络，并在独立更新过程中能够利用凸优化方法解析求得符合该层网络当前状态的最优学习率，从而使得训练神经网络不再需要手动调整学习率。通过使用最优学习率更新网络，可以达到加速收敛速度的效果。同时，本文给出了两种加速算法以改进收敛效果，并用实验验证了所提出算法的性能。

（3）本文针对所提出的更新算法给出了复杂度分析，从而给出了算法复杂度与输入样本个数以及神经元个数的关系。这有助于帮助算法使用者选取更适合的更新策略进行网络训练。

本方法仍有一定改善空间，在未来的研究中，将会尝试在本自适应更新算法框架下支持更多神经网络操作符，例如卷积操作等，以支持愈发复杂的深度神经网络结构。在今后的工作中，我们将持续开展相关研究以使得基于 ReLU 激活函数的神经网络自适应优化算法支持更多网络操作，如卷积操作等，这将使得本文提出的优化算法能够兼容更多网络结构。

参 考 文 献

[1] ROGERS K R, ROGERS M A. Principles of neural science[J]. American Journal of Psychiatry, 1987, 144(3): 370-370.

[2] Hubel D H, Wiesel T N. Ferrier lecture-Functional architecture of macaque monkey visual cortex[J]. Proceedings of the Royal Society of London. Series B. Biological Sciences, 1977, 198(1130): 1-59.

[3] Tanaka K. Columns for complex visual object features in the inferotemporal cortex: clustering of cells with similar but slightly different stimulus selectivities[J]. Cerebral cortex, 2003, 13(1): 90-99.

[4] 危辉. 视觉初级皮层区超柱结构的自组织适应模型[J]. 浙江大学学报: 工学版, 2001, 35(3): 258-263.

[5] LeCun Y, Boser B, Denker J S, et al. Backpropagation applied to handwritten zip code recognition[J]. Neural computation, 1989, 1(4): 541-551.

[6] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[7] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 580-587.

[8] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 3431-3440.

[9] Luo H, Fu J, Glass J. Bidirectional backpropagation: Towards biologically plausible error signal transmission in neural networks[J]. arXiv preprint arXiv:1702.07097, 2017.

[10] Ge R, Jin C, Zheng Y. No spurious local minima in nonconvex low rank problems: A unified geometric analysis[C]//Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017: 1233-1242.

[11] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors[J]. nature, 1986, 323(6088): 533-536.

[12] 陈建廷, 向阳. 深度神经网络训练中梯度不稳定现象研究综述[J]. 软件学报, 2018 (2018 年 07): 2071-2091.

[13] McCulloch W S, Pitts W. A logical calculus of the ideas immanent in nervous activity[J]. The bulletin of mathematical biophysics, 1943, 5(4): 115-133.

[14] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain[J]. Psychological review, 1958, 65(6): 386.

[15] Werbos P J. Beyond regression[M]. 1974:65-68.

[16] Rumelhart D E, Hinton G E, Williams R J. Learning internal representations by error propagation[R]. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[17] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines[C]//Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 807-814.

[18] Bottou L. Large-scale machine learning with stochastic gradient descent[M]//Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010: 177-186.

[19] Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent[J]. Cited on, 2012, 14(8).

[20] Arora S, Cohen N, Golowich N, et al. A convergence analysis of gradient descent for deep linear neural networks[J]. arXiv preprint arXiv:1810.02281, 2018.

[21] Qian N. On the momentum term in gradient descent learning algorithms[J]. Neural networks, 1999, 12(1): 145-151.

[22] Tieleman T, Hinton G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude[J]. COURSERA: Neural networks for machine learning, 2012, 4(2): 26-31.

[23] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization[J]. Journal of machine learning research, 2011, 12(Jul): 2121-2159.

[24] Zeiler M D. Adadelta: an adaptive learning rate method[J]. arXiv preprint arXiv:1212.5701, 2012.

[25] Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[26] Reddi S J, Kale S, Kumar S. On the convergence of adam and beyond[J]. arXiv preprint arXiv:1904.09237, 2019.

[27] Nguyen T T T, Armitage G. A survey of techniques for internet traffic classification using machine learning[J]. IEEE communications surveys & tutorials, 2008, 10(4): 56-76.

[28] 张斌, 武广号. 人工神经网络与改进遗传算法的协作求解[D]. , 2009.

[29] Bengio Y. How auto-encoders could provide credit assignment in deep networks via target propagation[J]. arXiv preprint arXiv:1407.7906, 2014.

[30] Lee D H, Zhang S, Fischer A, et al. Difference target propagation[C]//Joint european conference on machine learning and knowledge discovery in databases. Springer, Cham, 2015: 498-515.

[31] Bartunov S, Santoro A, Richards B, et al. Assessing the scalability of biologically-motivated deep learning algorithms and architectures[C]//Advances in Neural Information Processing Systems. 2018: 9368-9378.

[32] Lillicrap T P, Cownden D, Tweed D B, et al. Random synaptic feedback weights support error backpropagation for deep learning[J]. Nature communications, 2016, 7(1): 1-10.

[33] Feng J, Yu Y, Zhou Z H. Multi-layered gradient boosting decision trees[C]//Advances in neural information processing systems. 2018: 3551-3561.

[34] Friesen A L, Domingos P. Deep learning as a mixed convex-combinatorial optimization problem[J]. arXiv preprint arXiv:1710.11573, 2017.

[35] Nemirovski A, Juditsky A, Lan G, et al. Robust stochastic approximation approach to stochastic programming[J]. SIAM Journal on optimization, 2009, 19(4): 1574-1609.

[36] Langford J, Li L, Zhang T. Sparse online learning via truncated gradient[J]. Journal of Machine Learning Research, 2009, 10(Mar): 777-801.

[37] 郑书新. 针对深度学习模型的优化问题研究[D]. 中国科学技术大学, 2019.

[38] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[39] Santurkar S, Tsipras D, Ilyas A, et al. How does batch normalization help optimization?[C]//Advances in Neural Information Processing Systems. 2018: 2483-2493.

[40] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[41] Asuncion A, Newman D. UCI machine learning repository[J]. 2007.

[42] Michalski R S. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of development an expert system for soybean disease diagnosis[J]. International Journal of Policy Analysis and Information Systems, 1980, 4(2): 125-161.

攻读硕士学位期间所发表的学术论文及其他成果

（一）发表的学术论文

[1] Wang D, Wang J, Scaioni M, et al. Coarse-to-Fine Classification of Road Infrastructure Elements from Mobile Point Clouds Using Symmetric Ensemble Point Network and Euclidean Cluster Extraction[J]. Sensors, 2020, 20(1): 225. (SCI 检索号：000510493100225).

（二）申请的专利

[1] 刘波，王铎 一种基于目标传播和线搜索的深层神经网络优化及图像分类方法[P]. 国家发明专利. 已受理，专利号：202010252752.2

致 谢

本论文是作者在攻读硕士期间的主要工作总结，凝聚着我和我的导师刘波老师在三年的科研生活中所付出的时光与汗水。值此论文完成之际，首先感谢三年研究生生活中给予我帮助的老师和同学们，感谢大家在这三年时间内所给予我的帮助与指导。

首先，真挚感谢我的导师刘波老师，感谢他这三年对我学术上的指导，无论是从研究生课程中还是后续的科研课题中，刘老师都给予我很多指导与建议，使我从一个丝毫不懂得如何进行科研工作的门外汉转变为立志于从事研究工作的研究者。刘老师对科学的严谨态度，以及对科研工作的热情给了我极大的动力探寻未知。在这三年的研究生生活中，从刘老师的科研工作中学习到的如何思考成为了我最宝贵的精神财富。刘老师在面对陌生问题时独立思考的精神将不断勉励我如何做好学问。在此，再次感谢我学术生活的引路人，刘波老师。

感谢我的女友赵诗萌，感谢她在我科研和日常生活中的陪伴与鼓励。感谢她对我科研工作中给予的帮助。感谢她对我之后继续从事科研工作的支持与理解。

感谢我的实验室同学和师弟，他们在我的日常生活和科研生活中给予了我极大的帮助。感谢你们在我遇到困难时对我的关心与鼓舞。感谢你们在这三年生活中对我的帮助与陪伴。与你们共同度过美好的三年校园时光我将铭记于心。

感谢北京工业大学计算机学院的各位老师，感谢你们孜孜不倦的教导我们各门课程的知识，丰富了我的视野。同时感谢学校后勤相关的老师和工作人员，是你们辛苦的后勤保障给予了我稳定的科研和生活环境。

最后，感谢我的家人的支持。是他们一直在背后支持我完成学业，也是他们的不断支持才使得我有勇气走上学术科研的道路。在此感谢他们的默默付出，家人的关爱将会成为我坚强的源泉。

祝我的老师、同学和家人身体健康、工作顺利、学业有成。

谢谢！