单位代码：11414
学　　号：2015011510

中国石油大学（北京）
CHINA UNIVERSITY OF PETROLEUM , BEIJING

本科生毕业论文

题　　目　深度学习中的正则化方法研究

学 院 名 称　理学院

专 业 名 称　数学与应用数学

学 生 姓 名　胡汇丰

指 导 教 师　孙娜

起止时间：　**2018 年 12 月 25 日　至　2019 年 6 月 1 日**

摘　　要

深度学习随着数据集维度的增加而被广泛应用。我们在构建多层次网络模型的过程中，经常会遇到过拟合问题。如何防止过拟合成为深度学习的关键。本文主要研究 L^1、L^2 正则化和 Dropout，并通过 MNIST 手写体识别实验进行效果比对。

本文通过代数推导讨论了 L^1、L^2 正则化的特性，以及超参数对两者正则化效果的影响。得出 L^1 正则化通过稀疏化权重、L^2 正则化通过缩小权重，从而减小了模型拟合各种函数的能力来减弱过拟合的结论。本文还介绍了 Dropout 模型，并对其正则化特性进行了分析，得出其通过“模型的平均”和减小神经元之间的共适应减弱过拟合的结论。

最后，本门通过构建 784-1000-500-10 的全连接前馈神经网络，进行 MNIST 手写体识别实验，并分别运用 L^1、L^2 正则化和 Dropout 对该网络进行正则化处理。实验表明，虽然 L^1、L^2 正则化分别能将训练误差和测试误差的差距缩小到 0.8%、0.7%，但是由于其相对 Dropout 正则化效果较为振荡，二者的综合效果不如 Dropout。

关键字：过拟合；L^1 正则化；L^2 正则化；Dropout；MNIST

Regularization in deep learning

ABSTRACT

Deep learning is widely used as the dimension of data sets increases. In the process of constructing the multi-level network model, we often encounter the problem of overfitting. How to prevent overfitting as the key to deep learning. This paper focuses on L^1 and L^2 regularization and Dropout, and compares their effects through MNIST handwriting recognition experiment.

This paper discusses the characteristics of L^1 and L^2 regularization and the influence of hyperparameters on them through algebraic derivation. We get that ,L^1 regularization reduces the ability of the model to fit various functions by thinning the weights, and L^2 regularization does it by reducing weight, which reduces overfitting. This paper also introduces the Dropout model and analyzes its regularization characteristics, and concludes that it reduces the overfitting through the "average of the model" and the reduction of co-adaptation between neurons.

Finally, the MNIST handwriting recognition experiment was carried out by constructing a 784-1000-500-10 fully connected feedforward neural network, and the network was regularized by L^1, L^2 regularization and Dropout. Experiments show that although the regularization of L^1 and L^2 can reduce the difference between training error and test error to 0.8% and 0.7%, respectively, the relative effect of Dropout is relatively unstable, and the combined effect of the two is not as good as Dropout.

Key Words：Overfitting；L^1 Regularization；L^2 Regularization；Dropout；MNIST

目 录

前　言

计算机通过多层次的网络结构，构建简单的“认知”来学习复杂的概念，这种方法被称为 AI 深度学习[1]。它的另外一种解释是一种以人工神经网络为架构，对数据进行表征学习的算法[2-6]。在当前人工智能的发展中，深度学习起到了中流砥柱的作用。

一般我们通过构建神经网络来进行深度学习。现如今已有很多种深度学习的框架，例如深度神经网络（DNN）、卷积神经网络（CNN）、置信神经网络（DBN）和递归神经网络(RNN)。它们被应用在计算机视觉、自然语言处理、语音识别与生物信息学等领域并获得极好的效果。

本文主要研究深度神经网络，它是一种具备至少一个隐藏层的神经网络。我们在构建多层次的网络模型时，经常会遇到过拟合的问题。什么是过拟合？举个例子，我们现在有一组猫的照片（作为训练集），用深度神经网络训练，识别准确率为 99%。现在又有一组新的猫的照片（两次的照片规格相同），运用上面训练好的模型识别，识别结果只有 94%，这个时候我们就称该模型存在过拟合。为了使达到预期的效果，如何防止过拟合成为深度学习的关键。减弱过拟合的方法有 L^1 正则化和 L^2 正则化，本文通过代数推导研究这两种方法的正则化特性，以及超参数对正则化效果的影响。2014 年，Hinton Geoffrey 在文献[7]，正式介绍 Dropout 模型---随机删除神经元，训练一个个子网络，通过模型平均和降低神经元之间的共适应达到减小过拟合的效果。本文对该模型做了简要分析，并将其运用在深度神经网络上。

最后，我们构建了 784-1000-500-10 的深度前馈全连接神经网络，进行 MNIST 手写体实验，并运用在 L^1 正则化（惩罚系数为 0.00005）、L^2 正则化（惩罚系数为 0.0005）和 Dropout（存活概率为 0.4）进行正则化处理比较三者正则化效果。

第1章　神经网络和深度学习理论基础

本章主要介绍神经网络和深度学习的基本概念。主要包括感知器，S 型神经元模型，神经网络的架构，目标函数和随机梯度下降法，反向传播，过拟合等。为下文讨论 L^1 正则化、L^2 正则化、Dropout、MNIST 手写体识别提供一些理论基础。

1.1　感知器

首先什么是神经网络？它实际上是一种计算模型，由大量的被称为“感知器”的人工神经元直接相互关联而构成。感知器在 20 世纪五、六十年代由科学家 Frank Rosenblatt 提出，其受到 Warren McCulloch 和 Walter Pitts 早期工作的影响[8]。下面，我们通过一张图片来理解一下感知器是如何工作的？

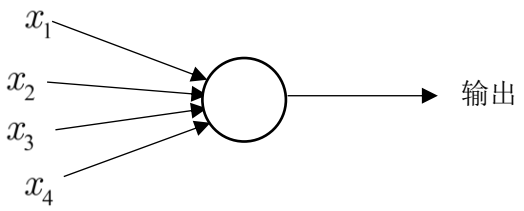

图 1.1 感知器示意图

Fig. 1.1 Perceptron schematic

如上图所示，感知器有四个输入，x_1,x_2,x_3,x_4。假设这些输入都是二进制的，并产生一个二进制的输出。神经网络用一种简单的规则来计算输出。由此引入**权重，**$w_1,w_2,\ldots$，表示相应的输入对于输出的重要程度。神经元的输出，0 或者 1，则由分配权重后的输入的总和$\sum_j w_j x_j$小于或者大于一些**阈值**P决定。权重和阈值都是神经元的参数。其代数形式如下：

$$\text{输出}=\begin{cases}0 & \text{如果}\sum_j w_j x_j \le P\\ 1 & \text{如果}\sum_j w_j x_j > P\end{cases} \tag{1.1}$$

对于式(1.1)，我们把阈值P移到不等式的左边，此时不等式变为（以其中一个为例）$\sum_j w_j x_j - P \le 0$，令$b=-P$,并称 b 为感知器的偏置。那么式(1.1)变为：

$$\text{输出}=\begin{cases}0 & \text{如果}\sum_j w_j x_j + b \le 0\\ 1 & \text{如果}\sum_j w_j x_j + b > 0\end{cases} \tag{1.2}$$

我们可以把偏置看作是让“感知器”输出 1 有多容易的估计。通过代数式，我们看到，对于具有一个非常大偏置的“感知器”来说，输出 1 是非常容易的。但当偏置是一个非常小的负数时，输出 1 就比较困难。由此我们看出，偏置是用来描述感知器的的变动。

通过上面的陈述，我们已经了解“感知器”是如何运行的。虽然我们通过“感知器”已经可以做出一些判断，但是它又略显单一。因为从代数形式我们可以看到其输出仅仅是输入线性相加，适用范围相对狭小。1.2 节将会介绍一种复杂的感知器---S 型神经元模型。

1.2 S 型神经元模型

首先，我们用描绘感知器的相同的方式来描绘 S 型神经元：

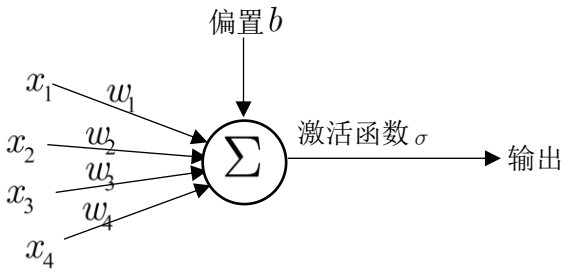

图 1.2 S 型神经元示意图

Fig. 1.2 Schematic diagram of s-type neurons

正如一个感知器，S 型神经元也有多个输入，$x_1, x_2, \ldots$。和感知器不同的是，这些输入可以取 0 和 1 中的任意值，而不仅仅是 0 或者 1[8]。对于每个输入，S 型神经元都有与之对应的权重 $w_1, w_2, \ldots$，和一个总的偏置 b。经过激活函数作用输出，这里的激活函数 σ 被称为 S 型函数（又称逻辑函数），定义如下：

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{1.3}$$

所以，一个具有输入 $x_1, x_2, \ldots$，权重 $w_1, w_2, \ldots$，和偏置 b 的 S 型神经元的输出是：

$$\frac{1}{1+e^{-(\sum_j w_j x_j + b)}} \tag{1.4}$$

从上面的代数式，我们发现其和感知器的代数式略有不同。下面我们分析一下它们的异同，假设 $z = \mathbf{w}\bullet\mathbf{x} + b$(其中 $\mathbf{w} = (w_1, w_2, \ldots, w_n)$，$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$，n 为输入的总个数，本文以下无特殊说明，均用此来表示权重和输入)。当 z 为正，且趋

于无穷的时候，$e^{-z}\to 0,\sigma(z)\to 1$，S 型神经元的输出近似为 1，正好和感知器的是一样；当z为负，且趋于无穷的时候，$e^{-z}\to\infty,\sigma(z)\to 0$，此时 S 型神经元的行为也非常近似一个感知器；只有在z取中间值时，S 神经元和感知器模型有比较大的偏差。下面是 MATLAB 画出的逻辑函数的图像，能更清晰的理解上面的阐述：

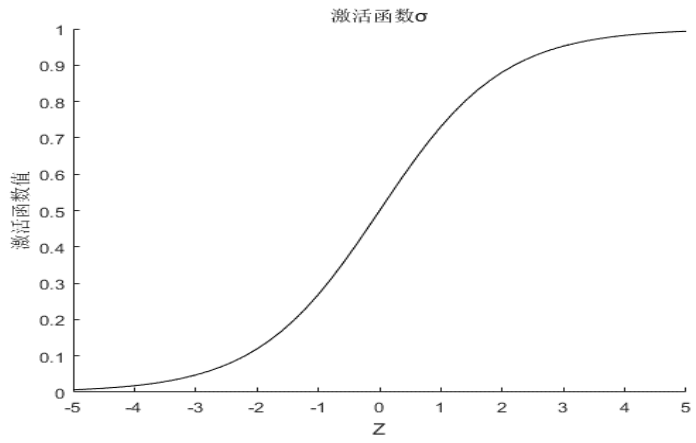

图 1.3 激活函数 σ 示意图

Fig. 1.3 Activation function diagram

假设我们正在识别一张图片是否为猫，用 S 型神经元该如何识别？不同于感知器输出是 0 或 1，S 型神经元的输出是0~1。在实际应用中，我们可以设定一个阈值来解决这个问题，例如，约定任何至少为 0.5 的输出来表示“图片是一只猫”，而其他小于 0.5 的输出则表示“图片不是一只猫”。这样的界定是根据实际需求而定，在此只是示例。由于激活函数和感知器的相似性，这也使得它常常被用于二分类问题。当然还有其他的“S 型函数”，例如双曲正切函数，其性质和逻辑函数类似，在此不过多分析，只是输出范围变为-1~1，下面只给出它的函数图像：

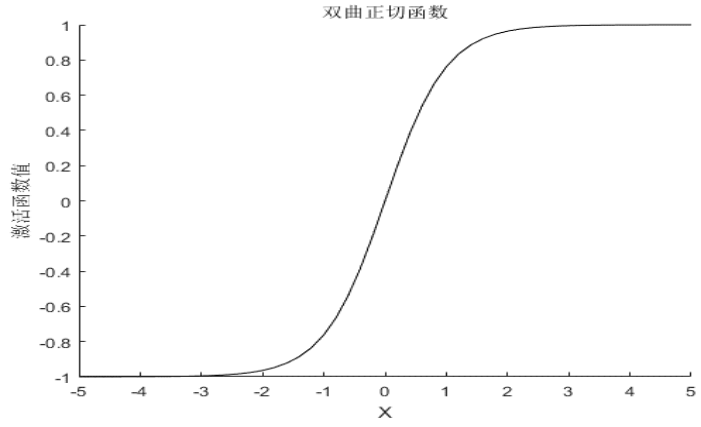

图 1.4 双曲正切函数示意图

Fig. 1.4 Plot of hyperbolic tangent function

1.3 神经网络的架构

在 1.1 节，我们提到神经网络是由“感知器”关联而成，根据关联的方式，神经网络可以分为：单层前馈网络、多层前馈网络、递归网络。本文主要运用的是多层前馈网络，如下图所示：

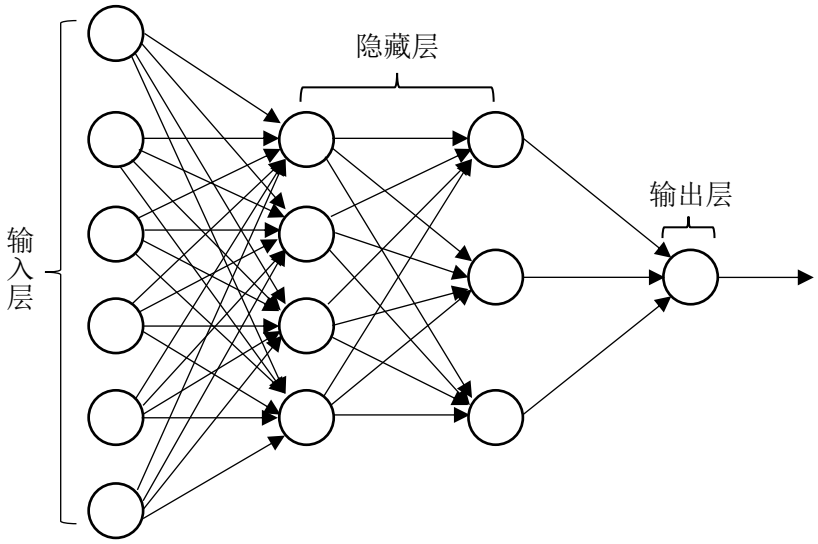

图 1.5 多层[illegible]神经网络

Fig. 1.5 Multilayer feedforward neural network

图 1.5 网络最左边的一层为**输入层**，对应的神经元被称为输入神经元。最右边的一层称为**输出层**，对应的神经元被称为输出神经元。而在输入层和输出层之间的这些网络层被称为**隐藏层**。网络输入层的神经元提供输入向量，组成第二层（第一隐藏层）神经元的输入信号。第二层的输出信号作为第三层的输入，这样一直传递下去，这种上一层的输出作为下一层的输入的网络被称为**前馈神经网络**[1][9]。最后的输出层给出相对于输入神经元的激活模式的网络输出。我们称该网络为 6-4-3-1 网络，四个数字分别对应输入层神经元的数量、第一隐藏层的神经元的数量、第二隐藏层的神经元的数量、输出层神经元的数量。并且这是一个**全连接网络**，即相邻层的任意一对节点都有连接，反之，则为部分连接网络。本文实验部分均采用的是多层前馈全连接神经网络。

计算机通过多层次的网络结构，构建简单的“认知”来学习复杂的概念。这种方法被称为 AI 深度学习[1]。现在神经网络的架构和神经元的运作方式，以及深度学习的概念已经介绍完毕。下面三小节，将对深度学习整体参数的更新以及过拟合现象进行研究。

1.4　目标函数和随机梯度下降法

1.4.1　目标函数

通过上面三小节，我们已经大致了解神经网络是怎样运作的，本节和下节将具体介绍神经网络如何更新权重 $\mathbf{W}$ 和偏置 b，以至于网络的输出（记为 $\hat{y}\ x$ ）能够拟合训练输入 x 对应的正确值 $y(x)$。所有的训练输入 x 组成**训练集**，以此来训练模型。相应的，用来评估模型的数据集被称为**测试集**。我们通过定义一个目标（或代价）函数 $J(\boldsymbol{\theta};\mathbf{X},\mathbf{y})$，来衡量模型的拟合程度。我们在此选用二次代价函数，以便更好地理解神经网络的运作，下面给出它的具体形式：

$$C(\boldsymbol{\theta};\mathbf{X},\mathbf{y})=\frac{1}{2n}\sum_{x}\left\|\hat{y}(x)-y(x)\right\|^2 \tag{1.5}$$

其中 $\boldsymbol{\theta}$ 代表神经网络中 $\mathbf{W}$ 和 b 的集合，$\mathbf{X}$ 表示所有的输入，y 表示所有的输出，n 是训练输入数据的个数。因为输出的一般为向量，所以用模进行运算。在深度学习中，二次代价函数也被称为均方误差。为了拟合训练输入 x 对应的正确值 $y(x)$，我们需要一种学习算法来找到合适的权重和偏置，使得目标函数尽可能的小。因为深度学习的目标函数一般都是凸函数，所以我们将采用梯度下降法来实现这一目的,下面介绍梯度下降法的相关内容。

梯度是相对于一个向量求导的导数：f 的导数是包含所有偏导数的向量，记为 $\nabla_x f(\mathbf{x})$。梯度的第 i 个元素是 f 关于 x_i 的偏导数。为了最小化 f，我们希望找到使 f 下降得最快的方向，即梯度的反方向，这种方法被称为**梯度下降法**[1][10]。其更新规则如下：

$$\mathbf{x}'=\mathbf{x}-\varepsilon\nabla_{\mathbf{x}}f(\mathbf{x}) \tag{1.6}$$

其中 ε 为学习率，是一个确定步长大小的正标量。一般是选择一个小常数。

因此对于二次代价函数 $\mathbf{W}$ 和 b 的更新方式如下：

$$\mathbf{w}\leftarrow\mathbf{w}-\varepsilon\nabla_{\mathbf{w}}C(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{1.7}$$

$$b\leftarrow b-\varepsilon\nabla_{b}C(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{1.8}$$

注意到为了计算二次代价函数 $C(\boldsymbol{\theta};\mathbf{X},\mathbf{y})$ 关于 $\mathbf{W}$ 和 b 的梯度，我们需要为每个训练输入 x 单独计算相应的梯度。当训练集或者测试集过大时会浪费很长时间，这样使得学习效率非常低。

1.4.2 随机梯度下降法

随机梯度下降法（SGD）能够解决 1.4.2 最后的问题。其核心思想是通过随机选取小批量的训练输入样本来近似估计整体梯度[8][10][11]。具体而言就是，在算法的每一步，我们从训练集中随机地抽出一小批样本$\mathbb{B}=\{x^{(1)},\ldots,x^{(m)}\}$。小批量的$m$通常是一个相对较小的数，从一到几百。重要的是，当训练集大小n增长时，m通常是固定的。

随机梯度下降法的梯度的估计可以表示成（以二次代价函数为例）：

$$\mathbf{g}=\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{m}C\left(\boldsymbol{\theta};\mathbf{X},\mathbf{y}\right)\approx\frac{1}{n}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{n}C\left(\boldsymbol{\theta};\mathbf{X},\mathbf{y}\right) \tag{1.9}$$

随机梯度下降算法的梯度下降为：

$$\boldsymbol{\theta}\leftarrow\boldsymbol{\theta}-\varepsilon\mathbf{g} \tag{1.10}$$

随机梯度下降法在深度学习之外也有很多应用。它是在大规模数据上训练大型线性模型的主要方法，在此不再详述。现在，我们已经大致了解$\mathbf{W}$和b的更新，下一小节将具体介绍具体如何更新这两个参数。

1.5 反向传播

在本节，我们依旧采用二次代价函数（式（1.5））作为研究对象。反向传播算法是，允许目标函数的信息通过网络向后流动，以便计算代价函数关于$\mathbf{W}$和b的梯度。计算梯度的解析表达式非常直观，但是数值化地求解这样的表达式的计算代价很大[1]。而反向传播算法则是使用“廉价”的步骤来实现这一过程。

1.5.1 基本定义

为了应用反向传播，我们需要对代价函数做出如下两个假设[8]：

1. 代价函数可以被写成一个在每个训练样本x上的代价函数C_x均值。
2. 代价函数可以写作关于神经网络输出$\hat{y}(x)$的函数。

做出假设 1 的原因是因为反向传播是对一个独立的训练样本计算$\frac{\partial C_x}{\partial\mathbf{W}}$和$\frac{\partial C_x}{\partial b}$。然后我们通过在所有训练样本上进行均值化从而得到$\frac{\partial C}{\partial\mathbf{W}}$和$\frac{\partial C}{\partial b}$。

做出假设 2 则是为求偏导$\frac{\partial C_x}{\partial\mathbf{W}}$和$\frac{\partial C_x}{\partial b}$，因为$\hat{y}(x)$是关于$\mathbf{W}$和$b$函数，而$y(x)$相当于固定的一个参数（对于每个$x$，$y(x)$不会随着$\mathbf{W}$和$b$的改变而改变）。

接下来，我们给出权重$\mathbf{W}$、b和激活值a的清晰定义，以便接下来的推导。

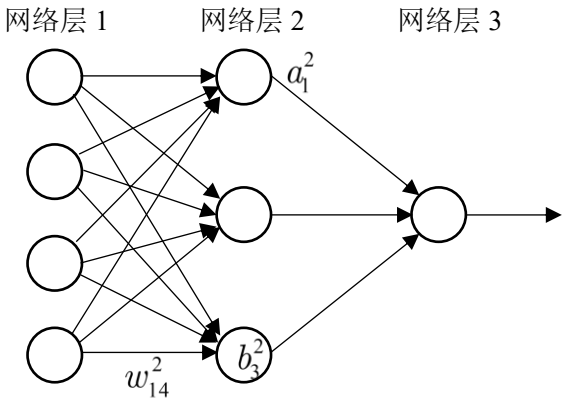

图 1.6 参数示意图

Fig. 1.6 Parameter diagram

如图 1.6 所示，w_{jk}^l 表示从 $(l-1)$ 层的第 k 个神经元到 l 层的第 j 个神经元的连接上的权重，b_j^l 表示在 l 层第 j 个神经元的偏置，a_j^l 表示 l 层第 j 个神经元的激活值。a_j^l 的具体形式如下：

$$a_j^l=\sigma(\sum_k w_{jk}^l a_k^{l-1}+b_j^l) \tag{1.11}$$

我们令 $z_j^l=\sum_k w_{jk}^l a_k^{l-1}+b_j^l$，其代表的含义是 l 层第 j 个神经元的带权输入，则式（1.11）可以写为：

$$a_j^l=\sigma(z_j^l) \tag{1.12}$$

接下来，我们引入一个中间变量 δ_j^l，以便计算 $\frac{\partial C}{\partial w_{jk}}$ 和 $\frac{\partial C}{\partial b_j^l}$，称之为 l 层第 j 个神经元的误差。它的具体形式如下：

$$\delta_j^l=\frac{\partial C}{\partial z_j^l} \tag{1.13}$$

如何理解这个定义？假设我们在 l 层第 j 个神经元的带权输入加入很小的扰动 Δz_j^l，那么此时神经元的输出由 $\sigma(z_j^l)$ 变为了 $\sigma(z_j^l+\Delta z_j^l)$。这个变化沿着网络向后传播，最终导致整个代价函数产生 $\frac{\partial C}{\partial z_j^l}\Delta z_j^l$ 的变化。如果 $\frac{\partial C}{\partial z_j^l}$ 接近于 0，那么扰动项就对代价函数的影响就会很小，此时我们认为神经元的参数已经接近最优。因此，我们把 $\frac{\partial C}{\partial z_j^l}$ 看作是神经元的误差的度量。

1.5.2 反向传播的基本方程

下面我们通过四个基本方程，来具体看一下，反向传播是如何更新参数的[12]。

首先是**输出层误差的方程**，定义如下：

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1.14}$$

其中 L 表示输出层，$\frac{\partial C}{\partial a_j^L}$ 表示代价随着第 j 个输出激活值的变化速度，$\sigma'(z_j^L)$ 表示了在 z_j^L 处激活函数 σ 变化的速度。回顾 δ_j^l 的定义，结合链式法则，其推导过程如下：

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

其次是**$l+1$层的误差导出l层的误差的方程**：

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l) \tag{1.15}$$

式（1.15）是反向传播的重要一步，即由 $l+1$ 层的误差导出 l 层的误差，以便计算 $\frac{\partial C}{\partial b_j^l}$ 和 $\frac{\partial C}{\partial w_{jk}^l}$。根据 δ_j^l 的定义，结合链式法则，其推导过程如下：

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l}\delta_k^{l+1} \tag{1.16}$$

而：

$$z_k^{l+1} = \sum_j w_{kj}^{l+1}a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1}\sigma(z_j^l) + b_k^{l+1} \tag{1.17}$$

则对于 z_k^{l+1} 求偏导：

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\partial(\sum_j w_{kj}^{l+1}\sigma(z_j^l) + b_k^{l+1})}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l) \tag{1.18}$$

将式（1.18）代入式（1.16）得式（1.15）。

接着是**目标函数关于偏置的变化率的方程**：

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{1.19}$$

通过式（1.19），我们知道该如何计算偏导数$\frac{\partial C}{\partial b_j^l}$。根据$\delta_j^l$的定义，结合链式法则，其推导过程如下：

$$\frac{\partial C}{\partial b_j^l}=\frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l}=\frac{\partial C}{\partial z_j^l}\frac{\partial(\sum_k w_{jk}^l a_k^{l-1}+b_j^l)}{\partial b_j^l}=\delta_j^l$$

最后是**目标函数关于权重的变化率的方程**：

$$\frac{\partial C}{\partial w_{jk}^l}=a_k^{l-1}\delta_j^l \tag{1.20}$$

通过式（1.20），我们知道该如何计算偏导数$\frac{\partial C}{\partial w_{jk}^l}$。根据$\delta_j^l$的定义，结合链式法则，其推导过程如下：

$$\frac{\partial C}{\partial b_j^l}=\frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l}=\frac{\partial C}{\partial z_j^l}\frac{\partial(\sum_k w_{jk}^l a_k^{l-1}+b_j^l)}{\partial w_{jk}^l}=a_k^{l-1}\delta_j^l$$

结合这四个方程，我们看到每个方程都直接或者间接包含了目标函数，而目标函数则是整个网络输出的结果，因此我们先计算最后一层的误差，然后沿着网络反向传播计算相应的误差，从而更新参数$\mathbf{W}$和b。

1.5.3　算法描述

在 1.5.2 节，我们通过四个基本方程了解了参数的更新。下面给出具有L层的神经网络的**反向传播算法描述**：

步骤一：将数据导入输入层，记输入层的神经元的激活值为向量$\mathbf{a}^1$($\mathbf{a}^l$表示第l层的激活值向量，其元素为l层每个神经元的激活值)。

步骤二：沿着网络前向传播，计算$l=2,3,\ldots,L$层的$z_j^l=\sum_k w_{jk}^l a_k^{l-1}+b_j^l$和$a_j^l=\sigma(z_j^l)$（在实际编程中，求和可以用向量点积，$w_{jk}^l$可以表示为矩阵的元素）。

步骤三：计算最后一层的误差$\delta_j^L=\frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$（在实际编程中，用矩阵的 Hadamard 乘积求解此项）。

步骤四：运用$\delta_j^l=\sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l)$计算$l=L-1,L-2,\ldots,2$层误差。

步骤五：根据$\frac{\partial C}{\partial w_{jk}^l}=a_k^{l-1}\delta_j^l$和$\frac{\partial C}{\partial b_j^l}=\delta_j^l$返回$\frac{\partial C}{\partial w_{jk}^l}$和$\frac{\partial C}{\partial b_j^l}$。

在 1.4.2 节，我们介绍了随机梯度下降法，而在本文实验部分，会将反向传播算法和随机梯度下降法结合使用，首先将大的输入数据集分成若干个小数据集，然后进行若干周期（迭代周期自主设定）的如下操作，假设小批量数据集的大小为 m，那么结合后的算法如下：

步骤一：导入小批量数据集。

步骤二：对于每次的训练样本 x,记其输入层的激活向量为 $\mathbf{a}^{x,1}$，执行下列操作：

1. 前向传播，计算 $l=2,3,\ldots,L$ 层的 $z_j^{x,l}=\sum_k w_{jk}^l a_k^{x,l-1}+b_j^l$ 和 $a_j^{x,l}=\sigma(z_j^{x,l})$
2. 计算最后一层的误差 $\delta_j^{x,L}=\frac{\partial C}{\partial a_j^L}\sigma'(z_j^{x,L})$
3. 运用 $\delta_j^{x,l}=\sum_k w_{kj}^{l+1}\delta_k^{x,l+1}\sigma'(z_j^{x,l})$ 计算 $l=L-1,L-2,\ldots,2$ 层误差。

步骤三：更新梯度，对 $l=L-1,L-2,\ldots,2$ 层，运用 $w_{jk}^l \leftarrow w_{jk}^l-\frac{\varepsilon}{m}\sum_x a_k^{x,l-1}\delta_j^{x,l}$ 和 $b_{jk}^l \leftarrow b_{jk}^l-\frac{\varepsilon}{m}\sum_x \delta_j^{x,l}$ 更新权重和偏置。

1.6 过拟合和欠拟合

在前面五小节，我们已经了解神经网络的基本架构和参数的更新方式。并且在 1.4.2 节，提到了训练集和测试集的概念。机器学习的主要挑战是学习算法在先前未观测的测试集上表现良好，而不仅仅是在训练集上表现良好。称在先前未观测的输入集上的拟合良好的能力为**泛化**[1][13]。

通常称模型在训练集上的误差称为**训练误差**（也叫**经验误差**），而称在测试集上的误差为**测试误差**（也叫**泛化误差**）[14]。在机器学习过程中我们不是提前固定参数，然后从数据集中抽取数据，来确定其匹配程度。而是，我们通过训练集的数据不断更新参数，然后用测试集来评估训练出来的模型的泛化能力。这样出现的结果就会是测试误差会大于或等于训练误差期望。而决定机器学习算法效果的因素有一下两点[1]：

1. 训练误差是否足够小。
2. 训练误差和测试误差的差距是否足够小。

这两个因素分别对应机器学习模型拟合过程中出现的两种现象：**欠拟合**和**拟合**。欠拟合是指网络训练出来的模型不能在训练集上获得足够小的误差。过拟合是指经验误差和泛化误差之间的差距过大。

第2章　L[1] 正则化和 L[2] 正则化

在本章我们将讨论 L^1 参数正则化和 L^2 参数正则化对于模型的影响。

2.1　L[1] 正则化

一般正则化方法都是通过对目标函数 J 添加一个参数惩罚项 $\Omega(\theta)$，来限制神经网络模型的学习能力[15]。我们将正则化的目标函数记为 $\tilde{J}$：

$$\tilde{J}(\mathbf{\theta};\mathbf{X},\mathbf{y}) = J(\mathbf{\theta};\mathbf{X},\mathbf{y}) + \alpha\Omega(\mathbf{\theta}) \tag{2.1}$$

其中 $\alpha \in [0,\infty)$ 是平衡范数惩罚系数项 Ω 和标准目标函数 $J(\mathbf{\theta};\mathbf{X},\mathbf{y})$ 相对贡献的超参数。

当我们的训练算法最小化正则化后的目标函数 $\tilde{J}$ 时，它会减少标准目标函数 $J(\mathbf{\theta};\mathbf{X},\mathbf{y})$ 关于训练数据的误差并同时减少参数 $\mathbf{\theta}$ 的规模[1]。选择不同的参数范数 Ω 会偏好不同的解法（我们仅对 $\mathbf{W}$ 正则化，即 $\mathbf{\theta} \Leftrightarrow \mathbf{w}$）。

L^1 参数正则化是通过向目标函数函数添加正则项 $\Omega(\theta) = \|\mathbf{w}\|_1$，使权重更加靠近坐标轴。

我们可以通过研究 L^1 参数正则化后目标函数的梯度，来量化权重衰减的正则化特征。通过上面的陈述，我们得到以下总的目标函数：

$$\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y}) = \alpha\|\mathbf{w}\|_1 + J(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{2.2}$$

与之对应的梯度为：

$$\nabla_{\mathbf{w}}\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y}) = \alpha sign(\mathbf{w}) + \nabla_{\mathbf{w}}J(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{2.3}$$

其中 $sign(\mathbf{w})$ 是取 $\mathbf{W}$ 各个元素的正负号，设 w 为 $\mathbf{W}$ 的一个元素，若 $w>0$，则 $sign(w)=1$；若 $w<0$，则 $sign(w)=-1$；若 $w=0$，则 $sign(w)=0$。

假设我们得到最优的目标解是 $\mathbf{w}^*$，用泰勒公式在 $\mathbf{w}^*$ 处展开可以得到一个逼近目标函数的截断泰勒级数：

$$J(\mathbf{w};\mathbf{X},\mathbf{y}) = J(\mathbf{w}^*;\mathbf{X},\mathbf{y}) + \nabla_{\mathbf{w}}J(\mathbf{w}^*;\mathbf{X},\mathbf{y})^T(\mathbf{w}-\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^T H(\mathbf{w}-\mathbf{w}^*)$$

其中 H 是海森矩阵，其形式如下（在接下来的推导的式子中用 H 来代替 $H(f)$）：

$$H(f)=\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

因为$\mathbf{w}^*$是最优解，所以$\nabla_{\mathbf{w}} J(\mathbf{w}^*;\mathbf{X},\mathbf{y})=\mathbf{0}$，上式可写为：

$$J(\mathbf{w};\mathbf{X},\mathbf{y})=J(\mathbf{w}^*;\mathbf{X},\mathbf{y})+\frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^T H(\mathbf{w}-\mathbf{w}^*) \tag{2.4}$$

因为在一般化的海森矩阵的条件下，L^1参数正则化无法得到清晰的代数表达式。在此我们输入数据已经被预处理（例如主成成分分析法处理），消除了输入特征之间的相关性。那么海森矩阵此时就是对角矩阵。即：

$$H=diag([H_{1,1},\ldots,H_{n,n}])$$

其中每个$H_{i,i}\geq 0, i=1,\ldots,n$。

此时，我们可以将L^1参数正则化目标函数的二次近似分解成关于参数的求和：

$$\hat{J}(\mathbf{w};\mathbf{X},\mathbf{y})=J(\mathbf{w}^*;\mathbf{X},\mathbf{y})+\sum_i[\frac{1}{2}H_{i,i}(w_i-w_i^*)^2+\alpha|w_i|] \tag{2.5}$$

因为H此时为对角阵，所以$\mathbf{w}$各个方向的导数是不相关的。若使$\hat{J}(\mathbf{w};\mathbf{X},\mathbf{y})$值达到最小，只需$\sum_i[\frac{1}{2}H_{i,i}(w_i-w_i^*)^2+\alpha|w_i|]$值达到最小，即对于每个$\frac{1}{2}H_{i,i}(w_i-w_i^*)^2+\alpha|w_i|$都取得最小值，等价于$\frac{1}{2}H_{i,i}(w_i-w_i^*)^2+\alpha|w_i|$关于$w_i$的导数为0，即：

$$H_{i,i}(w_i-w_i^*)+\alpha sign(w_i)=0 \tag{2.6}$$

通过式(2.5)，式(2.6)分析得到以下两点：

1. 式(2.5)的二次函数是关于w_i^*对称的，若使式(2.5)最小，那么必有$|w_i|<|w_i^*|$。因为在二次函数值不变的情况下，这样可以使$\alpha|w_i|$更小。
2. 在$\alpha|w_i|$不变的情况下，$sign(w_i)=sign(w_i^*)$或$w_i=0$，会使式(2.5)的值更小。

下面对$w_i^*>0$的情况进行讨论。

当 $w_i^* > 0$ 时，若 $sign(w_i) = sign(w_i^*)$，则式(2.6)可以写为：

$$H_{i,i}(w_i - w_i^*) + \alpha sign(w_i^*) = 0 \tag{2.7}$$

由此得：

$$w_i = w_i^* - \frac{\alpha}{H_{i,i}} sign(w_i^*) \tag{2.8}$$

因为 $w_i^* = sign(w_i^*)|w_i^*|$，替换上式的 w_i^* 得：

$$w_i = sign(w_i^*)(|w_i^*| - \frac{\alpha}{H_{i,i}}) \tag{2.9}$$

对于 $sign(w_i) = sign(w_i^*)$ 或 $w_i = 0$，若 $|w_i^*| \le \frac{\alpha}{H_{i,i}}$，那么 $sign(w_i) \ne sign(w_i^*)$，所以这时有 $w_i = 0$；若 $|w_i^*| > \frac{\alpha}{H_{i,i}}$，此时 $|w_i^*| - \frac{\alpha}{H_{i,i}} > 0$，$w_i = |w_i^*| - \frac{\alpha}{H_{i,i}}$。

综上，当 $w_i^* > 0$，最小化近似代价函数的解析解是：

$$w_i = sign(w_i^*)\max\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\} \tag{2.10}$$

从这个解可以得到两种结果：

1. 若 $w_i^* \le \frac{\alpha}{H_{i,i}}$，正则化后的目标函数的 w_i 的最优解是 $w_i = 0$。造成这种现象的原因是 w_i^* 所在方向上 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 对 $\hat{J}(\mathbf{w};\mathbf{X},\mathbf{y})$ 的影响受到抑制，L^1 正则化使得 w_i 推向 0。
2. 若 $w_i^* > \frac{\alpha}{H_{i,i}}$，正则化后的目标函数的 w_i 的最优解是 $w_i = w_i^* - \frac{\alpha}{H_{i,i}}$。此时。正则化不会使 w_i 推向 0，而是在 w_i^* 所在的方向上向原点移动了 $\frac{\alpha}{H_{i,i}}$ 长的距离。

同理，当 $w_i^* < 0$ 时，得到的结果与之类似。若 $|w_i^*| \le \frac{\alpha}{H_{i,i}}$，$L^1$ 正则化使得 w_i 推向 0；若 $|w_i^*| > \frac{\alpha}{H_{i,i}}$，$L^1$ 正则化使得 w_i^* 增加了 $\frac{\alpha}{H_{i,i}}$。

下面借助一张图来更形象的理解一下 L^1 正则化的效果：

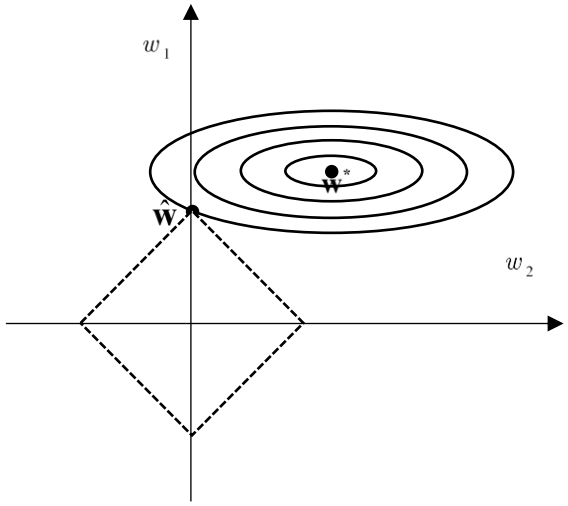

图 2.1 L^1 正则化效果示意图

Fig. 2.1 L^1 Regularization effect diagram

如图 2.1，坐标轴右上方的同心椭圆表示原始目标函数 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 的等值线，中心点 $\mathbf{w}^*$ 是没有正则化的原始最优解。图中（虚线）菱形表示 L^1 正则化项的等值线。最小化新的目标函数 $\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y})$，需要让 $\alpha\|\mathbf{w}\|_1$ 和 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 都尽可能小。在 $\hat{\mathbf{w}}$ 点处，这两个竞争目标达到平衡（$\hat{\mathbf{w}}$ 点为新的目标函数的最优解）。α 很大时，$\hat{\mathbf{w}}$ 直接等于 0；α 较小时，$\hat{\mathbf{w}}$ 被拉向 0。并且由于 L^1 正则化项图像的特殊性，$\hat{\mathbf{w}}$ 很容易就会出现在坐标轴上，即 L^1 正则化会让权重矩阵变得稀疏，使得网络复杂度降低，这也是为什么 L^1 **正则化能够防止过拟合**的原因。

2.2 L^2 正则化

L^2 参数正则化是通过向代价函数添加正则项 $\Omega(\boldsymbol{\theta})=\frac{1}{2}\|\mathbf{w}\|_2^2$，使权重更加接近原点[1]。$\|\mathbf{w}\|_2^2$ 写成向量的形式为 $\mathbf{w}^T\mathbf{w}$。

通过研究 L^2 参数正则化后代价函数的梯度，我们可以来量化权重衰减的正则化特征。通过上面的陈述，我们得到以下总的代价函数：

$$\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y})=\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}+J(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{2.11}$$

与之对应的梯度为：

$$\nabla_{\mathbf{w}}\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y})=\alpha\mathbf{w}+\nabla_{\mathbf{w}}J(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{2.12}$$

使用梯度下降法更新权重，过程如下：

$$\mathbf{w}\leftarrow\mathbf{w}-\varepsilon(\alpha\mathbf{w}+\nabla_{\mathbf{w}}J(\mathbf{w};\mathbf{X},\mathbf{y})) \tag{2.13}$$

同类项结合可得：

$$\mathbf{w} \leftarrow (1-\varepsilon\alpha)\mathbf{w} - \varepsilon\nabla_{\mathbf{w}} J(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{2.14}$$

和没有 L^2 正则化项的权重方式对比：

$$\mathbf{w} \leftarrow \mathbf{w} - \varepsilon\nabla_{\mathbf{w}} J(\mathbf{w};\mathbf{X},\mathbf{y}) \tag{2.15}$$

由式（2.14）知，在加入 L^2 正则化项之后，每次更新梯度前，都会先对权重向量乘于一个小于 1 的常数因子，这也就是 L^2 正则化称被为权重衰减的原因。

接下来，我们进一步分析这种权重衰减对整个模型产生的影响。我们假设在没有 L^2 正则化的情况下，使得代价函数取得最小训练误差时的权重向量为 $\mathbf{w}^*$，即 $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$。利用泰勒级数对原目标函数 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 在 $\mathbf{w} = \mathbf{w}^*$ 邻域内做二次近似，有：

$$J(\mathbf{w};\mathbf{X},\mathbf{y}) = J(\mathbf{w}^*;\mathbf{X},\mathbf{y}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^T H(\mathbf{w}-\mathbf{w}^*) \tag{2.16}$$

其中 H 是目标函数在 $\mathbf{w}^*$ 处的海森矩阵，因为 $\mathbf{w}^*$ 是最优解，所以目标函数在该点的梯度为 0，这也是该二次近似没有一阶项的原因，并且海森矩阵 H 是半正定的。

$J(\mathbf{w};\mathbf{X},\mathbf{y})$ 的梯度为：

$$\nabla_{\mathbf{w}} J(\mathbf{w};\mathbf{X},\mathbf{y}) = H(\mathbf{w}-\mathbf{w}^*) \tag{2.17}$$

根据式（2.12），加入 L^2 正则化项的总的目标函数的梯度为：

$$\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y}) = \alpha\mathbf{w} + H(\mathbf{w}-\mathbf{w}^*) \tag{2.18}$$

记加入正则化项之后的最优解为 $\hat{\mathbf{w}}$，有：

$$\nabla_{\mathbf{w}} \tilde{J}(\hat{\mathbf{w}};\mathbf{X},\mathbf{y}) = \alpha\hat{\mathbf{w}} + H(\hat{\mathbf{w}}-\mathbf{w}^*) = 0 \tag{2.19}$$

移项得：

$$(H+\alpha I)\hat{\mathbf{w}} = H\mathbf{w}^* \tag{2.20}$$

解得：

$$\hat{\mathbf{w}} = (H+\alpha I)^{-1} H\mathbf{w}^* \tag{2.21}$$

因为海森矩阵 H 是对称矩阵，可以通过特征分解将海森矩阵 H 分解成一个对角阵 Λ 和一组特征向量的标准正交基 Q，即 $H = Q\Lambda Q^T$。

因此式（2.21）可以写为：

$$\begin{aligned}\hat{\mathbf{w}} &= (Q\Lambda Q^T + \alpha I)^{-1} Q\Lambda Q^T \mathbf{w}^* \\ &= [Q(\Lambda + \alpha I)Q^T]^{-1} Q\Lambda Q^T \mathbf{w}^* \\ &= Q(\Lambda + \alpha I)^{-1} \Lambda Q^T \mathbf{w}^*\end{aligned}$$

由上式可以看出，权重衰减的效果是沿着海森矩阵的特征向量所对应的轴缩放$\mathbf{w}^*$。会根据缩放因子$\gamma=\frac{\lambda_i}{\lambda_i+\alpha}$，缩放与海森矩阵第$i$个特征向量同向的$\mathbf{w}^*$分量（$\lambda_i$是$H$的第$i$个特征值）。

由λ_i与α的关系以及α的取值可以分为以下四种情况：

1. 当α=0时，γ=1,正则化项为 0 时，不会进行缩放。
2. 当$\alpha \gg \lambda_i$时，$\gamma \approx 0$,即α越大缩放的幅度越大，即参数惩罚的也就越严重，最后得到的权重$\hat{\mathbf{w}}$就趋于 0。
3. 当α确定，$\lambda_i \ll \alpha$时，$\gamma \approx 0$，对应的分量w_i^*会被缩放为 0。
4. 当$\lambda_i \gg \alpha$时，$\gamma \approx 1$，对应的分量w_i^*保持不变。

由上面的情况可以看出，海森矩阵的特征值大小决定这权重的缩放程度。而海森矩阵的特征值表示的意义是该点附近特征向量方向上的凹凸性，特征值越大，对应的凸性越强。目标函数下降快的方向对应于训练样本的通用的特征方向，而下降的慢的方向则是会造成过拟合的特征方向[16]。

下面借助一张图来更形象的理解一下L^2正则化的效果。

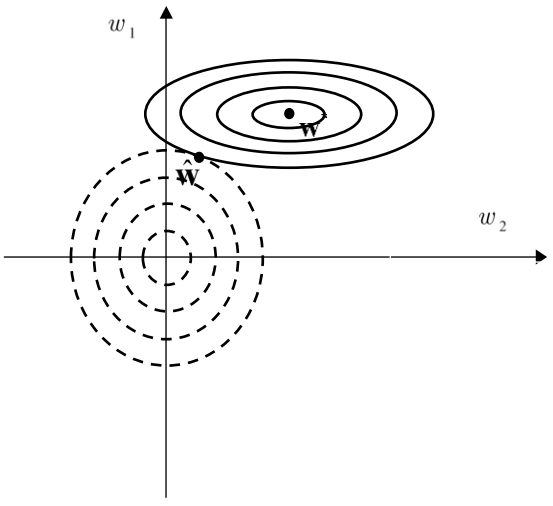

图 2.2 L^2 正则化效果示意图

Fig. 2.2 L^2 Regularization effect diagram

如图 2.2，坐标系右上方的同心椭圆表示原始目标函数 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 的等值线，中心点 $\mathbf{w}^*$ 是没有正则化的原始最优解。图中虚线的同心圆表示 L^2 正则化项的等值线。最小化新的目标函数 $\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y})$，需要让 $\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$ 和 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 都足够的小。我们发现，在 $\hat{\mathbf{w}}$ 点处，两者达到平衡。$\hat{\mathbf{w}}$ 点为新的目标函数的最优解。当正则化系数 α 越大时，$\hat{\mathbf{w}}$ 越接近零点；α 越小时，$\hat{\mathbf{w}}$ 越接近 $\mathbf{w}^*$。

我们看到，目标函数 $J(\mathbf{w};\mathbf{X},\mathbf{y})$ 的海森矩阵的 w_2 方向上的特征值很小。当 $\mathbf{w}^*$ 水平移动时，目标函数的值变化不会太大，表明目标函数对这个方向没有强烈的偏好。相反，正则化项则对此有较大的反应。由图 2.2，我们看到正则化项将 w_2 拉向零。再看，代价函数对于沿着 w_1 所在方向的移动较为敏感。其原因是因为对应海森矩阵的特征值比较大，表现为高曲率。因此，权重衰减对 w_1 所在方向影响较小。

通过上面的分析我们发现，保留的相对完整往往是有助于减小目标函数方向上的参数 w_i。而无助于目标函数减小的方向上的参数会在训练中逐渐的衰减掉。这也就是说，假设我们对于某一特征加一个扰动，目标函数变化很大，那么我们就可以认为该特征具有普适性，所以倾向于保留它。反之，是一些可能造成过拟合的特征，就通过权重衰减削弱它对整体的贡献。

那么 L^2 **正则化如何防止了过拟合**呢？

在目标函数添加 L^2 正则化项，通过代数推导，我们得到，L^2 正则化会使模型的参数倾向于比较小的值，针对参数减小了模型拟合各种函数的能力，从而减弱模型的过拟合现象。

2.3　L[1] 正则化和 L[2] 正则化比较

- L^1 参数正则化会趋于生成少量的权重，而其他权重都变为 0。如图 2.1，因为参数的最优值很大概率会出现在坐标轴上，这样就会导致 $\mathbf{w}$ 的某一维变为 0，产生稀疏的权重矩阵。
- L^2 参数正则化会保留更多的权重，但是这些权重都会趋近于 0。如图 2.2，L^2 参数正则化后最优的权重很小概率会出现在坐标轴上，因此 $\mathbf{w}$ 的每一维基本都不会是 0，而是在正则化下趋于 0。

第3章　Dropout

前面我们已经介绍了引入惩罚权重来削弱过拟合的影响，例如 L1 和 L2 正则化。本章主要讲述 **Dropout** 如何实现正则化。

3.1　简介

下面通过两张图来简单了解一下标准神经网络和应用 Dropout 之后的差异[7]：

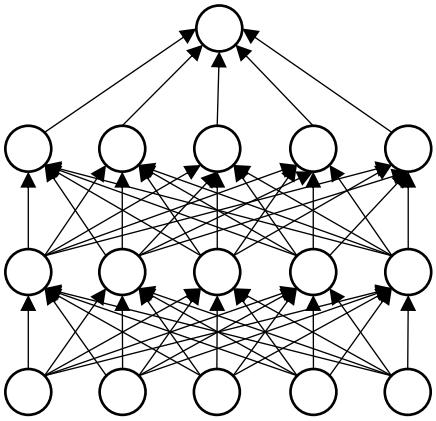

图 3.1 标准神经网络

Fig. 3.1 Standard neural network

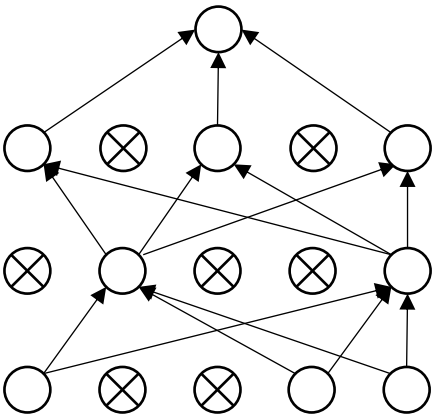

图 3.2 应用 Dropout 后

Fig. 3.2 After the application of dropout

图 3.1 为一个含有两个隐藏层的标准神经网络。而图 3.2 则是图 3.1 的神经网络应用 Dropout 之后的产生的稀疏网络，其中带叉的神经元已经被剔除。

下面介绍 Dropout 的具体工作流程，假设我们要训练如下神经网络：

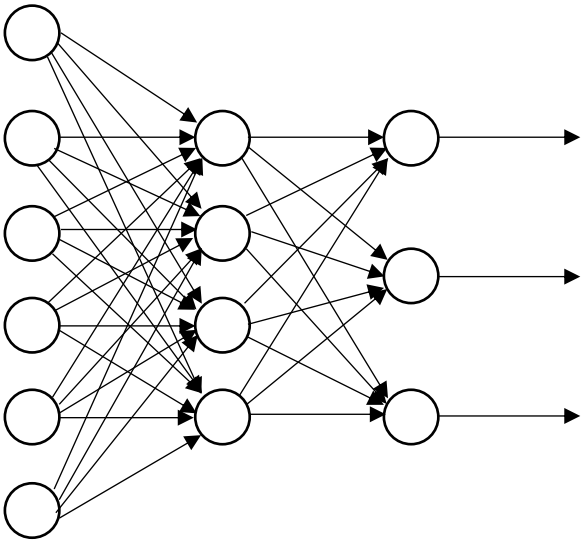

图 3.3 神经网络示意图

Fig. 3.3 Schematic diagram of neural network

步骤一：遍历网络所有的隐藏层，随机删除掉网络中隐藏层的部分神经元，输入层和输出层保持不变，如下图所示：

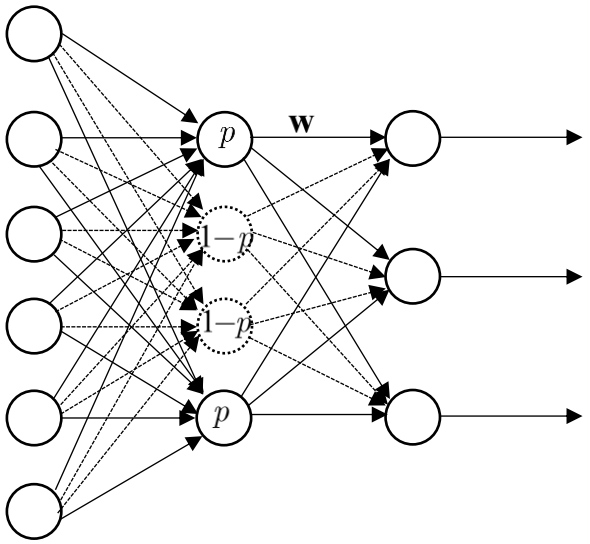

图 3.4 应用 Dropout 后

Fig. 3.4 After the application of Dropout

在最简单的情况下，每个单元都以固定的概率 p 保留，独立于其他单元，其中 p 可以使用验证集选择，也可以简单地设置成 0.1,0.2,…,0.9。这对于大型的网络和任务来说 0.5 似乎接近于最优值。然而，对于输入单元保留的最佳概率通常更接近于 1 而不是 0.5[7]。

步骤二：接着，输入 x 通过图 3.4 所示的神经网络传播，然后将得到的误差通过修改后的网络反向传播。被划分的小的训练集在执行完这个操做之后，按照随机梯度下降法更新没有被删除的神经元对应的参数 $\mathbf{w}$ 和 b。

步骤三：最后重复如下过程：

- 恢复被删除的神经元，随机删除的神经元的参数不会更新，没有被删除的神经元的参数得到更新。
- 再从隐藏层随机删除一部分神经元，并备份被删除神经元的参数。
- 在划分的小的训练集执行完这个操做之后，按照随机梯度下降法更新没有被删除的神经元对应的参数 $\mathbf{w}$ 和 b。被删除的神经元的参数保持原来的结果。

上述过程是在训练测试集的步骤，那么在用测试集验证的时候会有什么不同呢？我们注意到在训练的时候，神经元以概率 p 出现，并与下一层以权重 $\mathbf{w}$ 相连。

而在测试的时候，网络的神经单元一直存在，而权值要乘于 p。这样做是为了保证测试时的输出与训练时的输出期望相同。

下面是这两种情况的示意图：

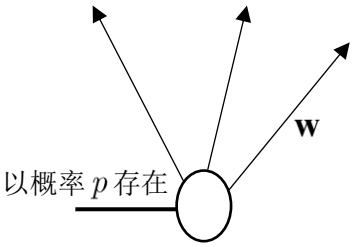

图 3.5 在训练时

Fig. 3.5 At training time

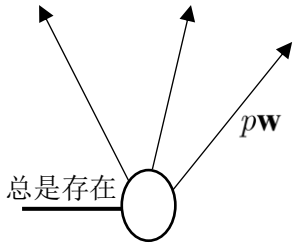

图 3.6 在测试时

Fig. 3.6 At test time

将 Dropout 应用到神经网络中相当于从神经网络抽取一个稀疏网络。稀疏网络由所有 Dropout 存活的单元组成（见图 3.4）。一个有 n 个单元（隐藏层单元）的神经网络，可以看作是有 2^n 个可能的神经网络的集合。这些网络共享权重，因此参数的总数仍然是 $O(n^2)$ 或更少。对于每次训练集的输出，将对一个新的稀疏网络重新训练。因此，训练一个 Dropout 神经网络可以看做是训练 2^n 个具有大量权重共享的稀疏网络的集合，每个稀疏网络很少接受训练，如果这种情况发生的话（训练单独的稀疏网络）[7]。

在测试时，以指数方式对许多稀疏网络模型的预测进行平均是不可行的。然而，一个非常简单的近似平均法在实践中的效果很好。该方法的思想是在测试时使用单一的神经网络而不 Dropout。该网络的权值是训练权值的“缩减版”，这样就类似于 L^2 正则化的权重缩减。如果一个单元在训练过程中以概率 p 保留，则该单元输出的权值在测试时乘于 p，如图 3.6 所示。这可以确保对于任何隐藏单元，预期输出（在训练时用于删除单元的分布下）与测试时的实际输出相同。在测试使用时，经过这样的交织，2^n 个共享权值的网络可以组成一个单独的神经网络。

3.2 模型描述

本节主要介绍 Dropout 神经网络模型。考虑一个具有 L 个隐含层的神经网络。让 $l \in 1,...,L$ 索引网络的隐藏层。设 z^l 为层的输入向量，y^l 为层 l 的输出向量（输入 $\mathrm{y}^0 = x$）。$\mathbf{w}^l$ 和 b^l 分别是 l 层的权重和偏差。标准神经网络的前馈操作（见图 3.7）可以被描述为（对于 $l \in \{0,....,L-1\}$ 的任意隐藏单元 i）

$$z_i^{l+1} = \mathrm{w}_i^{l+1}\ \mathrm{y}^l + b_i^{(l+1)} \tag{3.1}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{3.2}$$

其中 f 是任何激活函数，例如，$f(x) = \frac{1}{1+e^{-x}}$,也就是逻辑函数。

在 Dropout 后，前馈操作变为（见图 3.8）：

$$r_j^{(l)} \sim \text{Bernoulli}(p) \tag{3.3}$$

$$\tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)} \tag{3.4}$$

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)} \tag{3.5}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{3.6}$$

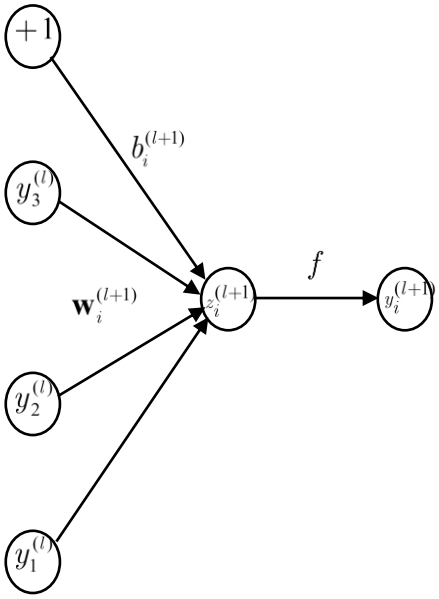

图 3.7 标准前馈操作

Fig. 3.7 Standard feedforward operation

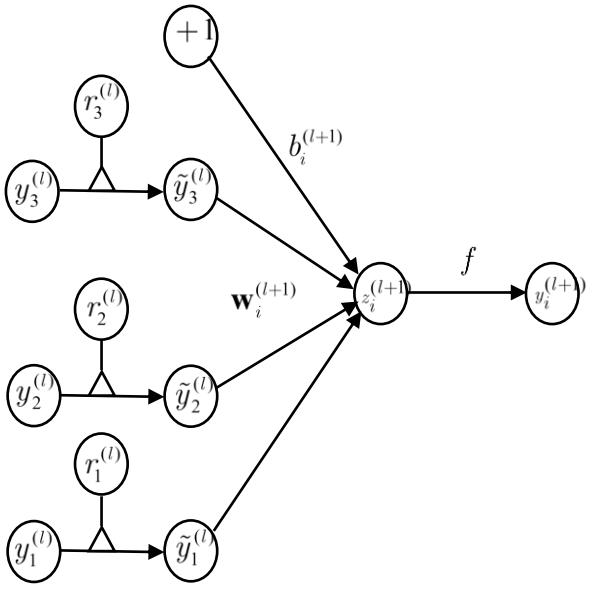

图 3.8　Dropout 前馈操作

Fig 3.8 Dropout feedforward operation

这里*表示 Hadamard。对于任意层 l，$\mathbf{r}^{(l)}$ 是一个独立伯努利随机变量的向量，每个变量以概率 p 为 1。用这个向量与该层的输出 $\mathbf{y}^{(l)}$ 相乘，从而得到稀疏后的输出 $\tilde{\mathbf{y}}^{(l)}$。然后将稀疏后的输出作为下一层的输入。这个过程应用于隐藏层的每一层。这相当于从一个更大的网络中抽取一个子网络。为了学习，损失函数的偏导数通过子网络反向传播。在测试时，将权重按比例 p 缩放为 $\mathbf{w}_{test}^{(l)} = p\mathbf{w}^{(l)}$，如图 3.6 所示。测试时的神经网络并没有使用 Dropout。

3.3 动机

Dropout 的灵感来自于性别在进化中的作用的理论[7]。有性生殖优越性的一种可能解释是，从长期来看，自然选择的标准可能不是个体的适应性，而是基因的混合程度。一组基因能够很好地与另一组随机基因一起工作的能力使它们更加健壮。由于一个基因不能依赖于其他基因每时每刻都在场，它必须学会自己独立做好一些事情，或者与少量的其他基因合作。根据这一理论，有性生殖的作用不仅是让有用的新基因在群体间传播，而且通过减少基因间复杂的共适应来促进这一过程，这种共适应会降低新基因改善个体适应性的可能。类比可知，在 Dropout 训练的网络中，每个隐藏层的神经元都必须学会使用随机选择的其他神经元样本。这将使每个隐藏单元更加健壮，降低自身创建有用的特性的可能，而不需要借助其他隐藏单元来纠正错误。然而，隐藏在一个层中的单元仍然要学会彼此做不同的事情。有人可能会认为，通过对每个隐藏单元进行多次复制，网络将变得健壮，从而防止 Dropout，但这是一个糟糕的解决方案，其原因与复制代码是处理噪声信道的糟糕方法完全相同。

Dropout 的一个密切相关但略有不同的动机来自于对计划的思考。20 个计划，每个计划涉及 5 个人，可能比一个大的计划，需要 100 个人都正确发挥自己的作用，能容易成功。假设条件不变，有足够的训练时间，大的计划也可以顺利进行，但复杂环境下，计划越小，继续运作的概率越大。复杂的共适应可以再训练集上训练得很好，但是在新的测试数据上，他们比实现相同目的的多个简单的共适应更容易失败[7]。

3.4 Dropout 如何防止过拟合

在 3.3 节，我们引用了欣顿教授形象的比喻解释了 Dropout 的“动机”。那么 Dropout 是如何防止过拟合，从而实现正则化的呢？我觉得有以下两点：

平均的作用：在标准的神经网络模型（没有用任何正则化方法）中，我们利用相同的训练集去训练 m 个不同的神经网络，一般会得到 m 种结果。如果我们采用取均值的方式来决定最终的模型，那么这种综合多个模型取均值的策略可以有效的减小过拟合。因为不同的网络可能会产生不同程度的拟合效果，有的欠拟合，有的过拟合，有的适中，取平均值会在一定程度上使过拟合和欠拟合相互抵消。Dropout 在训练时忽略部分神经元，训练不同的稀疏网络，并让这些网络共享权重，这样比多个模型组合付出的代价要低。在测试的时候，恢复所有的神经元，即所有的稀疏网络交织在一起，相应的权重乘于概率 p,相当于很多不同的神经网络取平均。这样在一定程度上使过拟合和欠拟合相互抵消达到整体上减小过拟合。

降低了神经元之间的适应性：正如在 3.3 节最后提到的，一个 100 个人一起完成的计划，要比 100 个人均分为 20 组完成 20 个任务要困难的多。因为前者需要 50 个人的默契配合，后者显然提高了这方面的容错率。回到 Dropout,在训练过程中，两个神经元不一定每次都会在一个 dropout 网络中出现。这样权值的更新不再依赖于有固定关系的隐藏层神经元的共同作用，一定程度上避免了一些特征只有在特定特征下才有效果的情况，迫使网络学习更加鲁棒（指系统的健壮性）的特征，从而降低了神经元之间的适应性，达到减小过拟合的效果[17]。

第4章　正则化在 MNIST 手写体识别的应用

在本章，我们将运用神经网络进行手写体识别，并对比标准神经网络和正则化下神经网络的 MNIST 手写体识别的效果。

4.1　MNIST 手写体

首先，什么是 MNIST 手写体[18]。它是 NIST 提供的用来进行神经网络实验的数据集，包含 60000 张手写数字的二进制数据组成的训练集和 10000 张相同规格的二进制图像组成的测试集。下面给出其中 100 张的可视化样例---灰度图像。（转换代码见附录 A-实验用代码）：

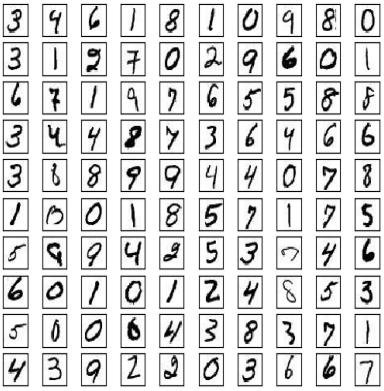

图 4.1(a) MNIST 手写体样例 1

Fig. 4.1(a) MNIST sample script-1

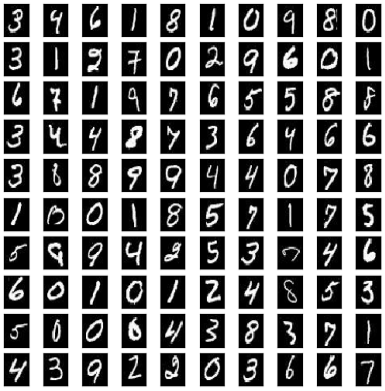

图 4.1(b) MNIST 手写体样例 2

Fig. 4.1(b) MNIST sample script-2

通过上面的图 4.1(a)和图 4.1(b)，我们注意到两张图的对应位置的图片相同，但是底色和数字的颜色恰巧相反。在本文实验中，我们训练和测试用的数据，用的是 MNIST 数据集的 784 维的二进制数据，其转换成可视化的数据集对应的图 4.1(b)，而图 4.1(a)是图 4.1(b)通过反转颜色得到。在这里强调一下，训练和测试的样例一定要相同，否则其准确率将会大幅下降。

4.2　MNIST 手写体标准网络下的识别

4.2.1　构建神经网络

由 4.1 节我们知道输入为 784 维的向量，输入的像素是灰度级的，值为 0 表示白色，值为 1.0 表示黑色，中间值表示逐渐暗淡的灰色。因此我们构建的全连接前馈神经网络的输入层的神经元的个数为 784 个。为了实现更好的拟合效果，我们

把第一个隐藏层的神经元的个数设置为 1000，第二个隐藏层的神经元的个数设置为 500。最后是输出层，因为手写体上包含 0~9 十个数字，因此我们的输出层的神经元的个数设置为 10。网络的示意图如下：

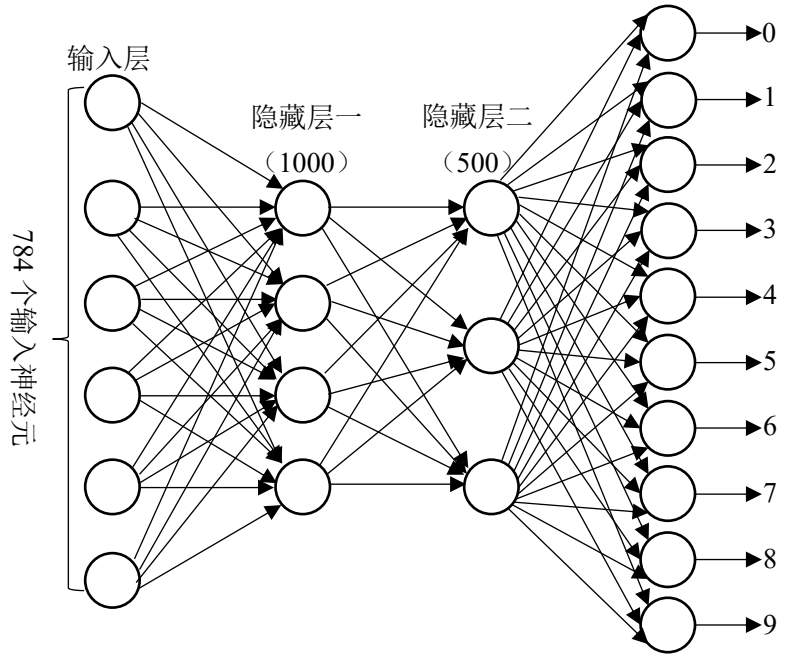

图 4.2 784-1000-500-10 网络

Fig. 4.2 784-1000-500-10 Net

4.2.2　激活函数、目标函数和超参数

在本文实验中，隐藏层激活函数我们选用双曲正切函数[19]（函数图像见图 1.4），其具体函数形式如下：

$$f(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \tag{4.1}$$

由于 MNIST 手写体识别是一个多分类问题，因此在输出层的激活函数我们选用 $Soft\max$ 函数，它的主要作用是是将多个神经元的输出，映射到 $0\sim1$ 的区间上，按照概率的高低进行分类，各个输出神经元的概率之和为 1[20]。则输出层激活函数的形式如下：

$$a_j^L=\frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \tag{4.2}$$

式中各部分的含义参见 1.5.1 节,1.5.2 节的相关定义。

对于学习而言，我们希望神经网络能够在犯错的情况学习地更快，这在平常的实际学习深有感触。所以在本实验中目标函数，我们采用交叉熵函数，其具体形式如下[21][22][23][24]：

$$C(\mathbf{\theta};\mathbf{X},\mathbf{y})=-\frac{1}{n}\sum_{x}[y(x)\ln\hat{y}(x)+(1-y(x))\ln(1-\hat{y}(x))] \tag{4.3}$$

式中各部分的含义参见 1.4.1 节的相关定义。

交叉熵函数之所以在犯错的情况下能够学习的更快，是因为我们用链式法则算式（4.3）关于权重的偏导数[25]。我们会发现，它是和 $\hat{y}(x)-y(x)$ 相关，$\hat{y}(x)-y(x)$ 刚好是误差大小。

在本文实验中，我们采用的的学习速率 $\varepsilon=0.5$，dropout 的存活概率为 0.5，L^1，L^2 正则化的惩罚系数 α 在 4.3.1 节和 4.3.2 节分别给出。

4.2.3 标准网络下的结果

本文实验的平台是 Ubuntu，运用的 python 版本是 3.6.7，TensorFlow 的版本是 1.13.1[26]。下面是在应用 4.2.2 的运行结果（下同；代码见附录 A-实验用代码）：

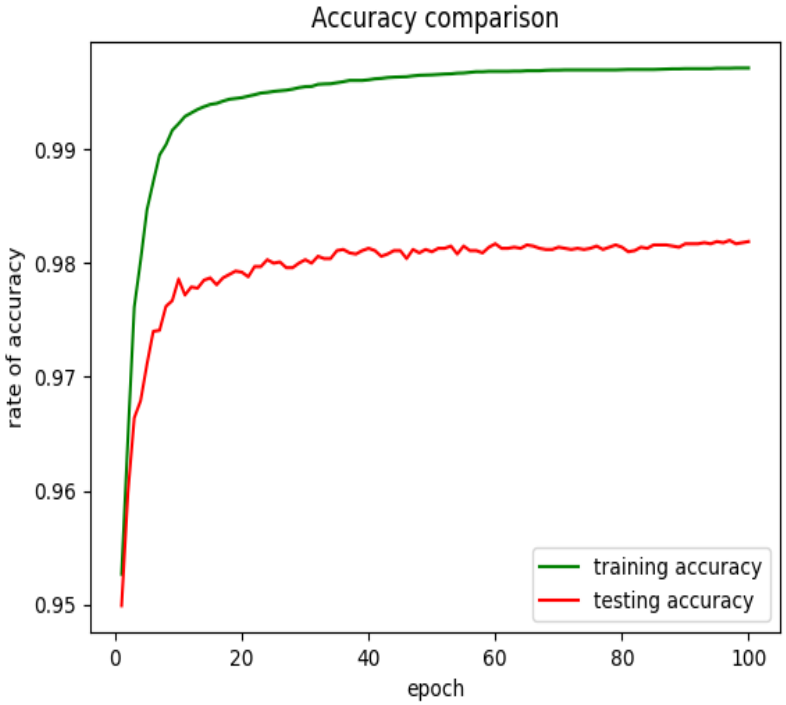

图 4.3 标准网络训练准确率和测试准确率对比图

Fig. 4.3 Standard network training accuracy and test accuracy comparison chart

我们节选了一部分训练周期的数据，如下表：

表 4.1 标准网络部分训练周期结果

Table 4.1 Standard network part training cycle results

	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
训练准确率(%)	99.45	99.61	99.69	99.70	99.71
测试准确率(%)	97.92	98.13	98.17	98.14	98.19
两者的差值(%)	1.53	1.48	1.52	1.56	1.52

由图 4.3，我们发现训练准确率要比测试准确率高出很多，再观察表 4.1 节选的五个时间点的两者的具体数值以及他们的差值，两者的差值最终稳定在 1.54%左右。由此我们得出神经网络存在拟合问题，接下来 4.3 节我们将用第二章和第三章介绍的正则化方法对该神经网络进行处理。

4.3　MNIST 手写体在正则化网络下的识别

4.3.1　L^1 正则化结果

我们取 α=0.5, 0.05, 0.005, 0.0005, 0.00005，编写程序比对结果（其他结果见附录 B-图），发现 0.00005 效果最好，运行结果如下（代码见附录 A-实验用代码）：

图 4.4　L^1 正则化网络训练准确率和测试准确率对比图

Fig. 4.4 Standard network training accuracy and test accuracy comparison chart of L^1

我们节选了一部分训练周期的数据，如下表：

表 4.2 L^1 正则化网络部分训练周期结果

Table 4.2 Standard network part training cycle results of L^1

	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
训练准确率(%)	97.26	97.55	97.95	98.02	97.88
测试准确率(%)	96.93	96.97	97.11	97.25	97.04
两者的差值(%)	0.33	0.58	0.84	0.77	0.84

由图 4.4 我们发现，在运用了 L^1 正则化后，训练准确率和测试准确率都有所下降，并且训练准确率要比测试准确率差距缩小，再观察表 4.2 节选的五个时间点的两者的具体数值以及他们的差值，两者的差值最终稳定在 0.8%左右，说明 L^1 正则化能在一定程度上减弱过拟合现象。

4.3.2 L^2 正则化结果

我们取 α=0.5, 0.05, 0.005, 0.0005，编写程序比对结果（其他结果见附录 B-图），发现 0.0005 效果最好，运行结果如下（代码见附录 A-实验用代码）：

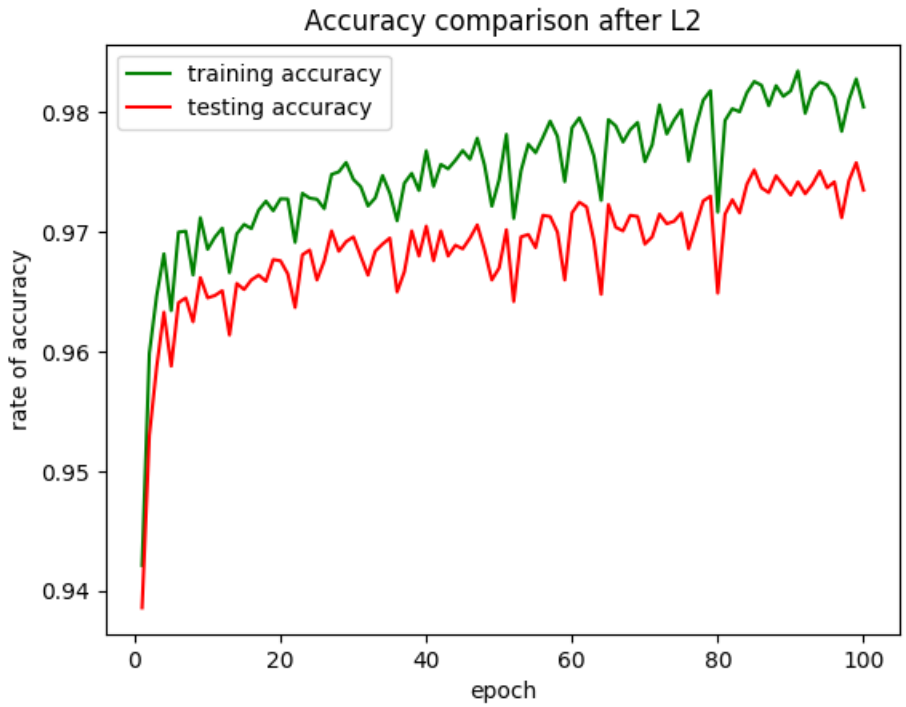

图 4.5 L^2 正则化网络训练准确率和测试准确率对比图

Fig. 4.5 Standard network training accuracy and test accuracy comparison chart of L^2

我们节选了一部分训练周期的数据，如下表：

表 4.3　L^2 正则化网络部分训练周期结果

Table 4.3 Standard network part training cycle results of L^2

	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
训练准确率(%)	97.27	97.68	97.87	97.16	98.04
测试准确率(%)	96.76	97.05	97.16	96.49	97.35
两者的差值(%)	0.51	0.63	0.71	0.67	0.69

由图 4.5 我们发现，在运用了 L^2 正则化后，训练准确率和测试准确率都有所下降，并且训练准确率要比测试准确率差距缩小，再观察表 4.3 节选的五个时间点的两者的具体数值以及他们的差值，两者的差值最终稳定在 0.7%左右。说明 L^2 正则化能在一定程度上减弱过拟合现象。

4.3.3　Dropout 结果

我们取 p=0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9，编写程序比对结果（其他结果见附录 B-图），发现 0.4 效果最好，运行结果如下（代码见附录 A-实验用代码）：

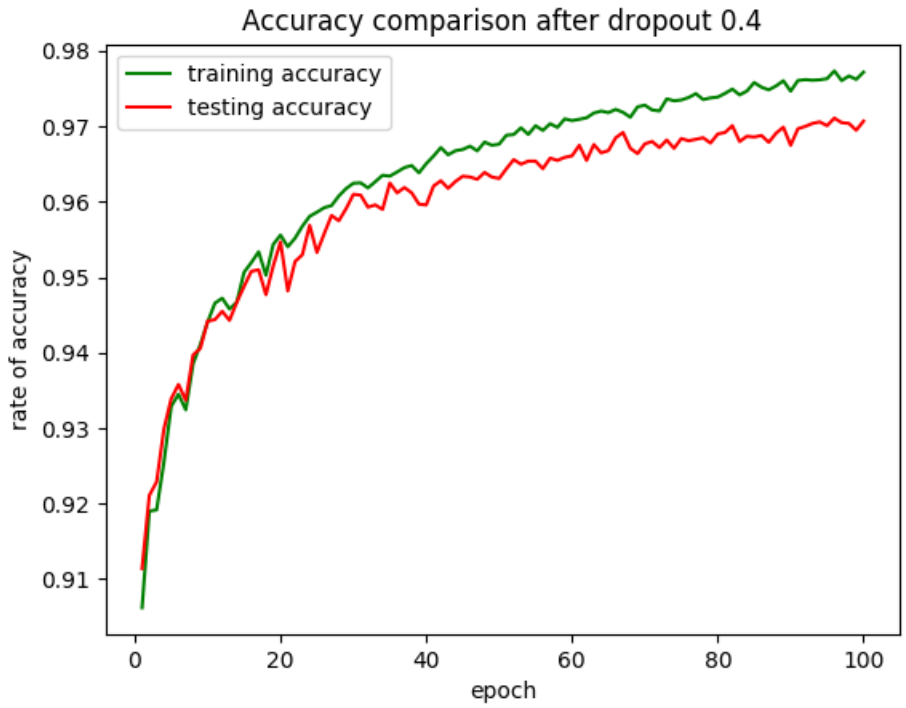

图 4.6 Dropout 后网络训练准确率和测试准确率对比图

Fig. 4.6 Standard network training accuracy and test accuracy comparison chart of Dropout

我们节选了一部分训练周期的数据，如下表：

表 4.4 Dropout 后网络部分训练周期结果

Table 4.4 Standard network part training cycle results of Dropout

	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
训练准确率(%)	95.56	96.51	97.07	97.46	97.72
测试准确率(%)	95.47	95.96	96.61	96.75	97.07
两者的差值(%)	0.09	0.55	0.46	0.71	0.65

由图 4.6 我们发现，在运用了 *Dropout* 正则化后，训练准确率和测试准确率都有所下降，并且训练准确率与测试准确率差距缩小，再观察表 4.4 节选的五个时间点的两者的具体数值以及他们的差值，两者的差值最终稳定在 0.68%左右。说明 *Dropout* 正则化能在一定程度上减弱过拟合现象。

4.3.4 综合比对

我们分别取上述四种情况的测试准确率进行比对，如下图：

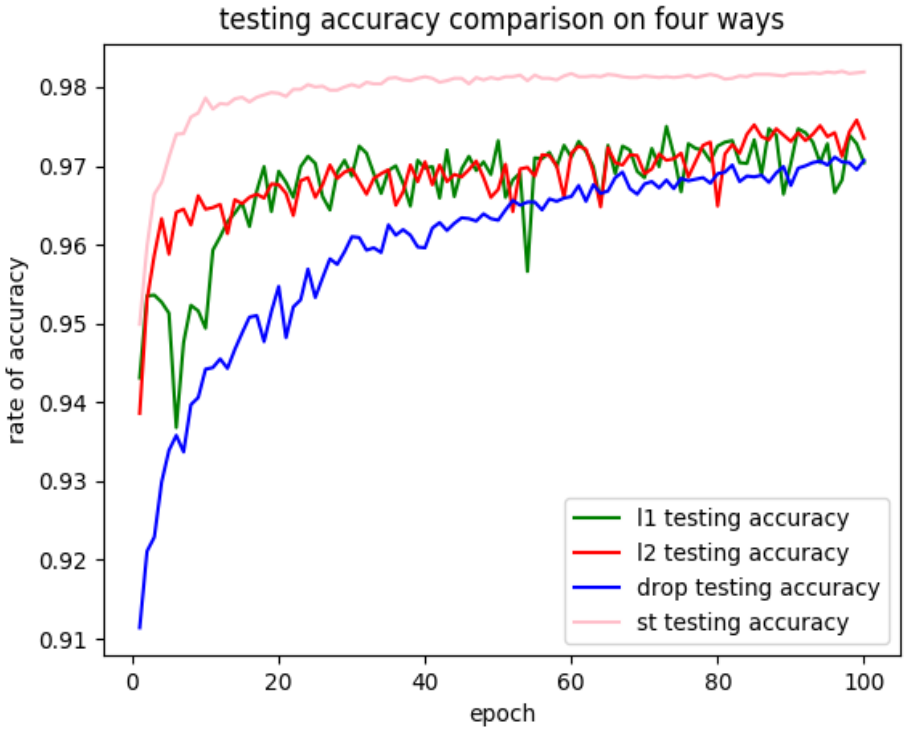

图 4.7 四种情况下的测试准确率对比

Fig. 4.7 Comparison of test accuracy in four cases

对应的表格如下：

表 4.5 四种情况测试准确率部分训练周期结果

Table 4.5 Four cases test accuracy rate partial training cycle results

测试准确率(%)	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
标准网络	97.92	98.13	98.17	98.14	98.19
L^1 正则化	96.93	96.97	97.11	97.25	97.04
L^2 正则化	96.76	97.05	97.16	96.49	97.35
Dropout	95.47	95.96	96.61	96.75	97.07

由图 4.7，我们看到 L^1、L^2 正则化测试准确率上下震荡比较厉害，而 Dropout 的只有校小幅度的震荡，并且由表 4.5，在 80~100 个 epoch，三者的测试 准确率都在 97%左右，由此看出 Dropout 作用后拟合效果（相对于 L^1、L^2 ）比较好。

我们分别取上述四种情况的训练准确率进行比对，如下图：

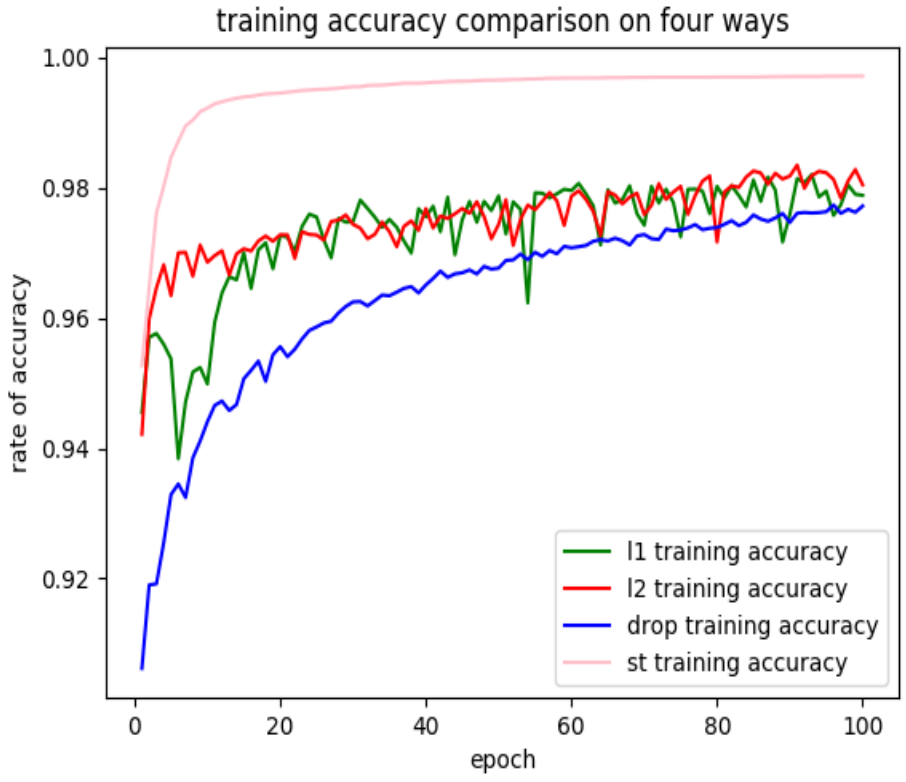

图 4.8 四种情况下的训练准确率对比

Fig. 4.8 Comparison of train accuracy in four cases

对应的表格如下：

表 4.6 四种情况训练准确率部分训练周期结果

Table 4.6 Four cases train accuracy rate partial training cycle results

测试准确率(%)	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
标准网络	99.45	99.61	99.69	99.70	99.71
L^1 正则化	97.26	97.55	97.95	98.02	97.88
L^2 正则化	97.27	97.68	97.87	97.16	98.04
Dropout	95.56	96.51	97.07	97.46	97.72

由图 4.8，我们看到 L^1、L^2 正则化测试准确率上下震荡比较厉害，而 Dropout 的只有校小幅度的震荡，并且由表 4.6，在 80~100 个 epoch，三者的测试 准确率都在 97.9%左右，由此看出 Dropout 作用后拟合效果（相对于 L^1、L^2 ）比较好。

下面我们对比部分周期四种情况下的训练误差和测试误差的差值。

表 4.7 四种情况准确率差值部分训练周期结果

Table 4.7 Four cases of accuracy rate difference part of the training cycle results

准确率差值(%)	Epoch 20	Epoch 40	Epoch 60	Epoch 80	Epoch 100
标准网络	1.53	1.48	1.52	1.56	1.52
L^1 正则化	0.33	0.58	0.84	0.77	0.84
L^2 正则化	0.51	0.63	0.71	0.67	0.69
Dropout	0.09	0.55	0.46	0.71	0.65

由表 4.7，我们发现 dropout 最终将准确率的差值缩小到 0.65%，相比 L^1、L^2 正则化的 0.69%，0.84%效果较好。综上对于 784-1000-500-10 的深度神经网络，Dropout 的正则化效果相对较好。

第5章 结 论

经过第二章和第三章的理论分析以及第五章的实验结果，我们得到关于 L^1、L^2 正则化和 Dropout 的如下结论：

（**1**）L^1 参数正则化会趋于生成少量的权重，而其他权重都变为 0。如图 2.1，由于 L^1 正则项函数的特性，参数的最优值很大概率会出现在坐标轴上。这样就会导致 **w** 的某一维变为 0，使得权重矩阵变得稀疏，并且网络复杂度降低，从而一定程度上减小过拟合。

（**2**）L^2 参数正则化则会保留更多（相对于 L^1 参数正则化）的权重，但是这些权重都会在不同程度上逼近于 0。如图 2.2，L^2 参数正则化的最优的参数只有很小概率会出现在坐标轴上，因此 **w** 的每一维基本都不会是 0，而是在正则化下逼近于 0。它通过衰减参数减小了模型拟合各种函数的能力，从而减弱模型的过拟合现象。

（**3**）Dropout 则是在训练时忽略部分神经元，训练不同的稀疏网络，并让这些网络共享权重。在测试的时候，恢复所有的神经元，即所有的稀疏网络交织在一起，相应的权重乘于概率 p,相当于很多不同的神经网络取平均。这样在一定程度上使过拟合和欠拟合相互抵消达到整体上减小过拟合。并且在训练不同稀疏网络时，两个神经元不一定每次都会在一个 dropout 网络中出现。这样权重的更新不再依赖于有“逻辑关系”的隐藏层的神经元的共同作用，一定程度上避免了一些特征只有在特定特征下才有效果的情况，迫使网络学习更加鲁棒（指系统的健壮性）的特征，从而降低了神经元之间的适应性，达到减小过拟合的效果。

（**4**）在 MNIST 手写体实验中，我们构建 784-1000-500-10 的深度神经网络，并进行正则化处理。结果表明 dropout 的正则化效果会更好，最终的准确率稳定在 97%，训练误差与测试误差的差值为 0.65%。

参 考 文 献

[1] Bengio , Y et al. Deep learning.人民邮电出版社.2017:87-133,196-205.

[2] Deng, L.; Yu, D. Deep Learning: Methods and Applications . Foundations and Trends in Signal Processing. 2014, 7: 3–4.

[3] Bengio, Yoshua. Learning Deep Architectures for AI . Foundations and Trends in Machine Learning. 2009, 2 (1): 1–127.

[4] Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013, 35 (8): 1798–1828.

[5] Schmidhuber, J. Deep Learning in Neural Networks: An Overview. Neural Networks. 2015, 61: 85–117.

[6] Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey. Deep Learning. Nature. 2015, 521: 436–444.

[7] Srivastava N , Hinton G , Krizhevsky A , et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting[J]. Journal of Machine Learning Research, 2014, 15(1):1929-1958.

[8] Michael A. Nielsen, Neural Networks and Deep Learning,Determination Press, 2015.

[9] Kurt Hornik,Maxwell Stinchcombe,Halbert White. Multilayer feedforward networks are universal approximators.1989.

[10] G. E. Hinton., "A Practical Guide to Training Restricted Boltzmann Machines," Tech. Rep. UTML TR 2010-003, Dept. CS., Univ. of Toronto, 2010.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner,Gradient-Based Learning Applied to Document Recognition,(1998).

[12] Rumelhart D E . Learning Representations by Back-Propagating Errors[J]. Nature, 1986, 23.

[13] 李航. 统计学习方法[M].北京：清华大学出版社, 2012: 10-17.

[14] Scott Fortmann-Roe. http://scott.fortmann-roe.com/docs/BiasVariance.html. 2012.

[15] Hinton G E , Srivastava N , Krizhevsky A , et al. Improving neural networks by preventing co-adaptation of feature detectors[J]. Computer Science, 2012.

[16] Bengio Y, Boulanger-Lewandowski N, Pascanu R. Advances in optimizing recurrent networks[C]// IEEE International Conference on Acoustics. 2013.

[17] Krizhevsky A , Sutskever I , Hinton G . ImageNet Classification with Deep Convolutional Neural Networks[C]// NIPS. Curran Associates Inc. 2012.

[18] Yann LeCun,Corinna Cortes ,Christopher J. C. Burges. MNIST Dataset. http://yann.lecun.com/exdb/mnist/. 2013.

[19] Lecun Y A , Bottou L , Orr G B , et al. Efficient BackProp[J]. 1998.

[20] G. E. Hinton et al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups," IEEE Signal Processing Magazine, pp. 82–97, November 2012.

[21] Thomas M. Cover, Joy A. Thomas. Elements of Information Theory.2012: 150-216.

[22] Haykin S . 神经网络与机器学习[M]. 机械工业出版社, 2011:1-17.

[23] 周志华, 王珏. 机器学习及其应用 2009[M]. 清华大学出版社, 2009.

[24] 雷明. 机器学习与应用. 北京：清华大学出版社, 2018：138-179.

[25] Bengio Y . Practical recommendations for gradient-based training of deep architectures[J]. 2012.

[26] Tom Hope et al.,TensorFlow 学习指南：深度学习系统构建详解.机械工业出版社,2018:1-52.

附录 A　实验用代码

MNIST 手写体可视化代码

```
#coding: utf-8
from tensorflow.examples.tutorials.mnist import input_data
import scipy.misc
import os

# 读取 MNIST 数据集。如果不存在会事先下载。
mnist = input_data.read_data_sets("/home/huifeng-hu/桌面/MNIST_data/",
one_hot=True)

# 我们把原始图片保存在 MNIST_data/raw/文件夹下
# 如果没有这个文件夹会自动创建
save_dir = '/home/huifeng-hu/桌面/MNIST_data/picture/'
if os.path.exists(save_dir) is False:
    os.makedirs(save_dir)

# 保存前 2000 张图片
for i in range(500):
    # 请注意，mnist.train.images[i, :]就表示第 i 张图片（序号从 0 开始）
    image_array = mnist.train.images[i, :]
    # TensorFlow 中的 MNIST 图片是一个 784 维的向量，我们重新把它还原为
28x28 维的图像。
    image_array = image_array.reshape(28, 28)
    # 保存文件的格式为 mnist_train_0.jpg,
mnist_train_1.jpg, ... ,mnist_train_19.jpg
    filename = save_dir + '%d.jpg' % i
    # 将 image_array 保存为图片
    # 先用 scipy.misc.toimage 转换为图像，再调用 save 直接保存。
    scipy.misc.toimage(image_array, cmin=0.0, cmax=1.0).save(filename)

print('Please check: %s ' % save_dir)

#**********转换图片黑白色代码**********
import cv2
```

```
import matplotlib.pyplot as plt
save_dir = '/home/huifeng-hu/MNIST_data/picture/'
for i in range(500):
    filename = save_dir + '%d.jpg' % i
    img = cv2.imread(filename)
    print(img)
    img = 255-img
    filename = save_dir + '%d.jpg' % i
    cv2.imwrite(filename,img)

plt.figure()
for i in range(1,101):
    filename = save_dir + '%d.jpg' % i
    img = cv2.imread(filename)
    plt.subplot(10,10,i)
    plt.imshow(img)
    plt.xticks([])
    plt.yticks([])
plt.show()
#**********************************end**********************************
```

MNIST 手写体识别代码

```
'''python 3.6.7 ---tensorflow 1.13.1'''
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
import matplotlib.pyplot as plt#用来作图
#载入数据集
mnist = input_data.read_data_sets("/home/huifeng-hu/桌面
/MNIST_data",one_hot=True)
#每个批次的大小
batch_size = 64
#计算一共有多少个批次
n_batch = mnist.train.num_examples // batch_size
#定义三个 placeholder
'''tf.placeholder(dtype,shape=None,name=None)
1.dtype：数据类型。常用的是 tf.float32,tf.float64 等数值类型
```

2.shape：数据形状。默认是 None，就是一维值，也可以是多维（比如[2,3], [None, 3]表示列是 3，行不定）
3.name：名称
'''

```
x = tf.placeholder(tf.float32,[None,784])
y = tf.placeholder(tf.float32,[None,10])
keep_prob=tf.placeholder(tf.float32)

#
# 定义神经网络结构：784-1000-500-10
W1 = tf.Variable(tf.truncated_normal([784,1000],stddev=0.1))
# 参数 stddev 用于设置正太分布被截断前的标准差  下同
b1 = tf.Variable(tf.zeros([1000])+0.1)
L1 = tf.nn.tanh(tf.matmul(x,W1)+b1)
# 双曲正切曲线  tanh(z) = (e^z-e^-z)/(e^z+e^-z)  f' = 1 - f^2 激活函数 下同
# 下面的激活函数也是如此 下同
L1_drop = tf.nn.dropout(L1,keep_prob)

W2 = tf.Variable(tf.truncated_normal([1000,500],stddev=0.1))
b2 = tf.Variable(tf.zeros([500])+0.1)
L2 = tf.nn.tanh(tf.matmul(L1_drop,W2)+b2)
L2_drop = tf.nn.dropout(L2,keep_prob)

W3 = tf.Variable(tf.truncated_normal([500,10],stddev=0.1))
b3 = tf.Variable(tf.zeros([10])+0.1)
prediction = tf.nn.softmax(tf.matmul(L2_drop,W3)+b3)
#reg = tf.contrib.layers.apply_regularization(tf.contrib.layers.l2_regularizer(5e-4),
tf.trainable_variables())  #L2 参数正则化  68 行 dropout 设置为 1
reg = tf.contrib.layers.apply_regularization(tf.contrib.layers.l1_regularizer(5e-5),
tf.trainable_variables())
# 输出层运用了柔性最大值层 其激活函数是 a(j,L) = exp(z(j,L))/Σk(exp(z(k,L)))
'''代价函数则是运用了交叉熵代价函数
（交叉熵函数可以使 程序 在“犯错的时候” 学习的更快 这是由其偏导数决定的）
```

```
量化的话就是  在误差大的时候    其对应的偏 w 和 偏 b 的导数也会变大  从而使得调整的速率会变快
而普通的函数（比如说二次代价函数） 则是在犯错的时候的学习速率 越来越慢
'''
loss = tf.losses.softmax_cross_entropy(y,prediction) + reg
#使用随机梯度下降法  学习率为 0.5
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(loss)

#初始化变量
init = tf.global_variables_initializer()

#结果存放在一个布尔型列表中
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(prediction,1))#argmax 返回一维张量中最大的值所在的位置
#求准确率
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))

#定义会话
test_acc_list = []
train_acc_list = []
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(100):
        for batch in range(n_batch):
            batch_xs,batch_ys =  mnist.train.next_batch(batch_size)
            sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys,keep_prob:1.0})
            # 在训练的时候  如果不使用 dropout 则 设置为 1.0 就是说删除元素的概率为 0
            # 如果使用 dropout  概率一般设置为 0.5

        test_acc = sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels,keep_prob:1.0})
        train_acc = sess.run(accuracy,feed_dict={x:mnist.train.images,y:mnist.train.labels,keep_prob:1.0})
```

```
            test_acc_list.append(test_acc)
            train_acc_list.append(train_acc)
            print("Iter " + str(epoch) + ",Testing Accuracy " + str(test_acc) +",Training
Accuracy " + str(train_acc))
x = np.linspace(1,100,100)
plt.title('Accuracy comparison after L1')
#l2 时  after L2    l1 时    after l1      dropout after dropout
plt.plot(x, train_acc_list, color='green', label='training accuracy')
plt.plot(x, test_acc_list, color='red', label='testing accuracy')
plt.legend()
plt.xlabel('epoch')
plt.ylabel('rate of accuracy')
plt.show()
```

附录 B 图

Dropout 结果图

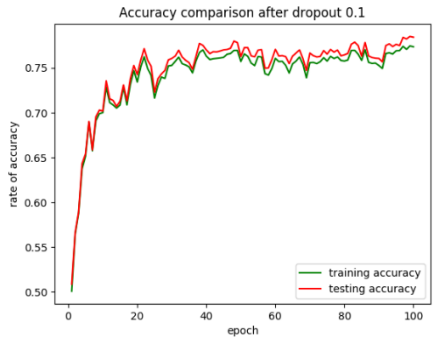

图 B.1 Dropout 存活 0.1 结果图

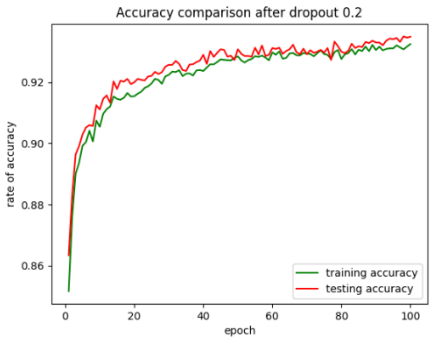

图 B.2 Dropout 存活 0.2 结果图

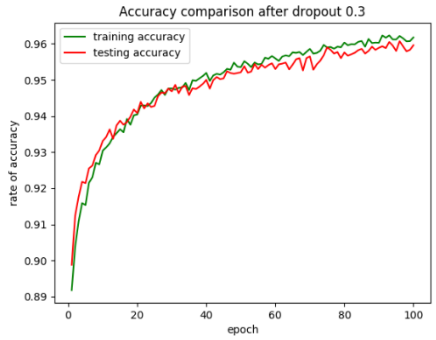

图 B.3 Dropout 存活 0.3 结果图

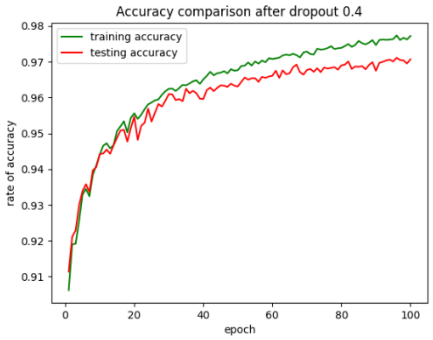

图 B.4 Dropout 存活 0.4 结果图

图 B.5 Dropout 存活 0.5 结果图

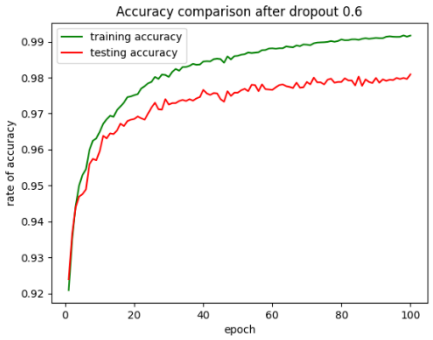

图 B.6 Dropout 存活 0.6 结果图

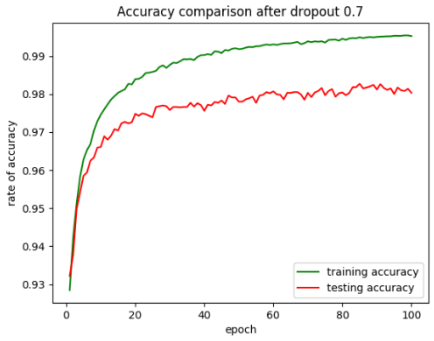

图 B.7 Dropout 存活 0.7 结果图

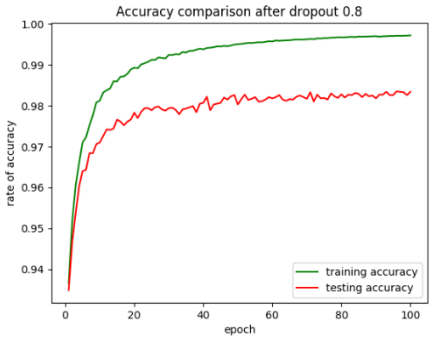

图 B.8 Dropout 存活 0.8 结果图

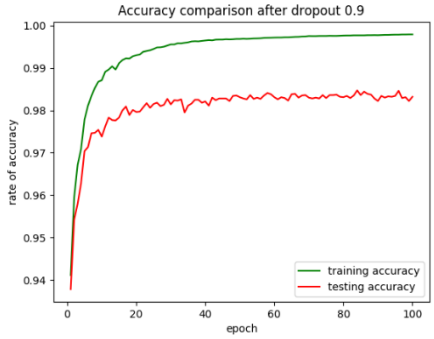

图 B.9 Dropout 存活 0.9 结果图

L^1 正则化结果图

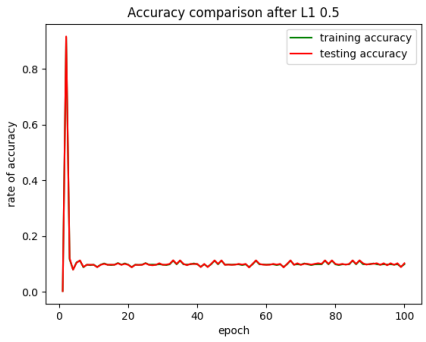

图 B.10 L^1 惩罚系数 0.5 结果图

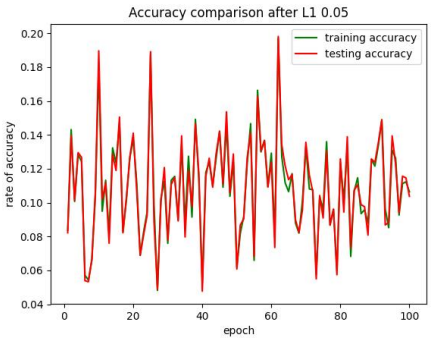

图 B.11 L^1 惩罚系数 0.05 结果图

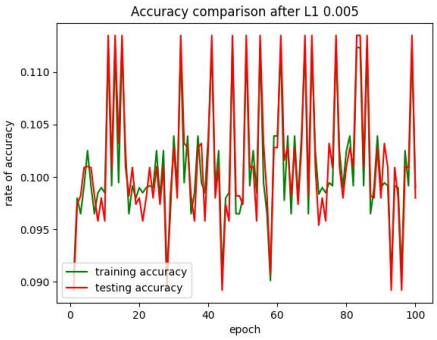

图 B.12 L^1 惩罚系数 0.005 结果图

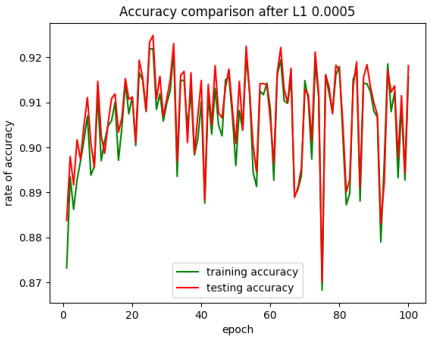

图 B.13 L^1 惩罚系数 0.0005 结果图

L^2 正则化结果图

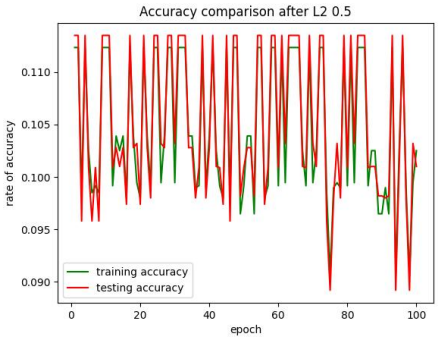

图 B.14 L^2 惩罚系数 0.5 结果图

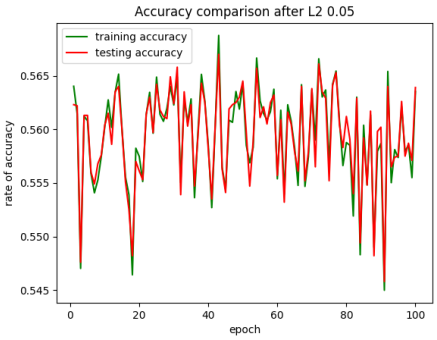

图 B.15 L^2 惩罚系数 0.05 结果图

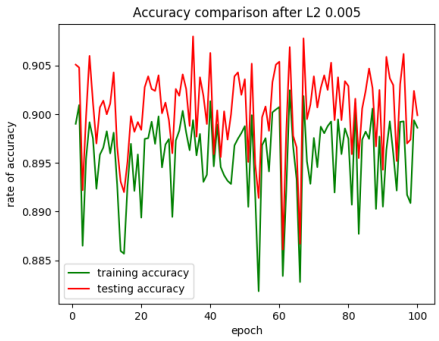

图 B.16 L^2 惩罚系数 0.005 结果图

致　　谢

衷心感谢我的导师孙娜老师。孙娜老师待人亲和，关心学生。在孙娜老师的悉心指导下，我对机器学习的兴趣日益浓厚，对当今人工智能的认识不断深入。在这六个月相处中，孙娜老师无论在学业还是在生活上，都为我提供了极大的帮助，我愿借此机会再次向孙娜老师表达最诚挚的感谢。

感谢中国石油大学（北京）理学院的武国宁、崔学慧、刘建军、荆宝坤、胡锐等诸位老师在学业和生活中给予我的无私帮助。

感谢刘劭、胡明禹和范智颖同学，和你们在学习和生活上的相处让我受益匪浅。

最后，感谢我的姥姥、姥爷和爸爸、妈妈，感谢你们在背后默默地支持和鼓励。