从零开始搭建井字棋AI

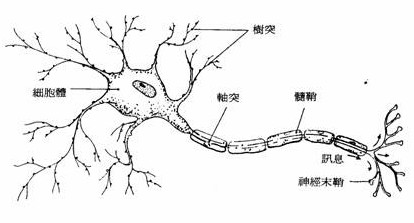
前言：尽管深度学习已经有大量成熟的库可以使用，但对于初学者而言，从底层学起可加深对深度学习的理解，为未来深入的学习打下基础。这篇文章将介绍深度学习的基础知识，并利用井字棋来进行实践。

深度学习的简介

机器学习是指从已知数据中获得规律，并利用规律对未知数据进行预测的方法。传统的机器学习需要人为设计特征值，而深度学习则无需人为设计特征值，可自动提取特征值。因此深度学习能自动获取一些人类难以想象的特征信息，识别效果比传统机器学习好。

深度学习的起源包括感知器和玻尔兹曼机，本文只涉及感知器的相关部分。感知器最早于1957年由美国学者罗森布拉特（Frank Rosenblatt）提出，但由于无法解决线性不可分的问题，使得后续的研究工作停滞了十多年。这个问题被后来提出的多层感知器解决。到了1985年，鲁梅尔哈特（Rumelhart）等人提出了误差反向传播算法（back－propagation，BP）,使得神经网络可以自动更新参数。如今，深度学习的模型和算法已经非常多，上面提到的模型和算法已经足以搭建一个简单的井字棋AI.

M-P模型

 在介绍感知器前，先介绍感知器的基础——M-P模型（The McCulloch-Pitts neural model）。这是一种模仿神经元的模型。神经元也就是单个神经细胞，它的细胞体周围有树突与轴突。树突接收到周围神经元的信号输入，对信号进行处理后，通过轴突将信号输出。

神经元模型

M-P模型类似于神经元，其基本组成为：输入 、权重 、 阈值 、输出 ，类似于神经元的树突-轴突结构。输入输出的关系为：若输入的加权和大于阈值，则输出1，否则输出0。利用函数可以表示为：

式中，w为权值（weight）,x为输出，h为阈值，y为输出，Sgn(x)为符号函数。利用M-P模型，我们可以将与、或等逻辑运算表述出来。以逻辑或为例，取 w1=1、w2=1、h=0.5，当输入x1、x2中至少有一个为1时，输出1，当输入全为0时，输出0。即：

但M-P模型只能表示线性可分问题，无法表示线性不可分问题。这是因为对于含有n个未知量的线性方程，其基础解系为n-1​维，只能将​维空间线性划分成两部分。以二维为例，若取​为x1横坐标，​x2为纵坐标，则只能表示一条直线，即只能将二维空间线性划分成两个部分。

M-P模型的另一个问题是：需要人为设定参数，无法通过输出与期望值的差异来调整参数。

感知器

感知器在M-P模型的基础上，通过有监督学习，即通过训练样本和期望输出，得到实际输出的误差，并利用误差调整参数。用公式表示为：

其中，α为调整的幅度，称为学习率。α增大则误差修正的速度增加，反之则速度降低。

通过不断重复上述过程，可以使得参数不断地趋近于目标参数，最终得到所需地模型。

多层感知器与误差反向传播算法

为了解决线性不可分问题，人们提出了多层感知器模型（multilayer perceptron），最简单的多层感知器采用三层结构：输入层、中间层、输出层，各层之间的输入输出与M-P模型相同。为了让各层之间的连接权重能够自动调整，我们采用的是误差反向传播算法。

误差反向传播算法的目标是：寻找使误差达到最小值的权重。其基本步骤为：

1. 正向计算出输出；
2. 根据输出与期望的误差，反向逐层调整连接权重；
3. 重复1. 2.步骤，直到误差缩小到合适的范围内。

权重的调整主要使用梯度下降法，即通过向梯度的反方向调整权重，让误差逐步下降到极小值。其基本的数学推导如下：

误差与权重构成一个多元函数，要让误差函数取得最小值，需要沿着梯度的反方向调整权重，故权重的调整值. 注意，是由两个函数复合而成：

首先，误差直接取决于输出：

而输出由取决于输入和权重：，但由于无法Sgn()函数求导，我们采用sigmoid函数：

综上，我们可以求出，从而得到调整后的权重：

上面只是梯度下降法对应单层感知器的公式，我们可与类比推导出对应多层感知器的情况。在此就不赘述。

利用Python的实现

Python作为一种解释性语言，其语法较为灵活，实现起来更方便。我们还需要用到Numpy，这个库有一些有用的数学函数。根据上面的原理，我们可以写出一个简单的两层网络。具体代码在<https://github.com/ToddZhouFeng/tic-tac-toe-AI> ，下面我们只是列出它的框架：

class TwoLayerNet:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, weight\_init\_std=0.01)

def predict(self, x)#利用输入x得到输出

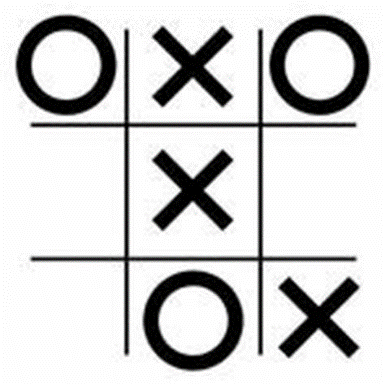
def gradient(self, x, t) # x:输入数据, t:监督数据, 返回更新后的参数

使用该框架时，先初始化输入层的结点数、中间层的结点数和输出的结点数。最开始的权重由随机数产生，之后利用gradient()函数求梯度，不断地更新权重，使得误差达到最小。

在此补充一下，我们的输出将利用softmax()函数，使得输出值实际上表示的是“概率”，在井字棋中对应着“下那个位置能赢的概率”。

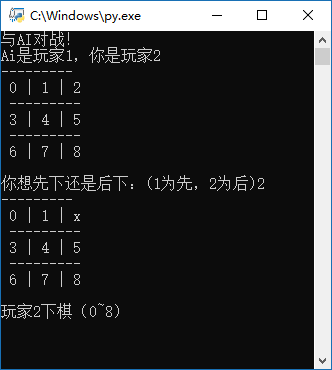
将神经网络用于井字棋

井字棋（Tic-Tac-Toe），是一种在3\*3格子上进行的连珠游戏，和五子棋类似，最先连成3子的一方获胜。

我们将棋盘分为三部分：未下的，甲下的，乙下的，我们分别用三个数组表示（每个数组包含九个元素，对应九个格），1表示这个位置在数组中，0表示不在。比如我们用下面三个数组表示右图的情况：

empty=[0, 0, 0, 1, 0, 1, 1, 0, 0]  
X=[0, 1, 0, 0, 1, 0, 0, 0, 1]  
O=[1, 0, 1, 0, 0, 0, 0, 1, 0]

那么对于整个棋盘，我们可以把它当成一个3\*9的矩阵

board=[  
  [0, 0, 0, 1, 0, 1, 1, 0, 0],  
  [0, 1, 0, 0, 1, 0, 0, 0, 1,  
  [1, 0, 1, 0, 0, 0, 0, 1, 0] ]

当然，我们输入的时候是输入一个1\*27的数组。

我们先编写“人-人对战”的程序，将赢的一方的下棋位置作为监督数据，棋盘上的棋子分布作为输入数据。然后利用这些数据去更新神经网络的连接权重。更新的过程可能需要上千次的循环。最终，我们可以将AI用于“人-机对战”的程序。

关于中间层的结点数量

在井字棋神经网络中，输入和输出层是根据游戏的需要规定的，而中间层的结点数量是可以任意规定的。那么到底应该取多少才能使得输出的误差最小？ 通过尝试不同的中间层结点数，我们得到如下折线图，横轴为训练次数，纵轴为误差。

可以发现，结点数小于20时，中间层的结点数越多，其误差减小的速度越快，最终的误差也越小。但结点数多于20后，继续增加结点数对改善误差的帮助并不大，反而会加重训练负担。因此，中间层的结点数在20~30之间即可。

结语

通过模拟神经元，逐步搭建了一个神经网络，并通过误差反向传播算法使该网络能够达到我们得目标——下井字棋。在这个过程中，我们学习并实际了深度学习的基本算法。但这只是一个十分简单的任务，如果需要完成一些复杂的任务（比如图像识别），我们还需要更加深度的学习。希望读者以本文为敲门砖，逐步领略深度学习的奥秘。

参考文献：

斋藤康毅.深度学习入门：基于Python的理论与实现[M].北京：人民邮电出版社.2018