

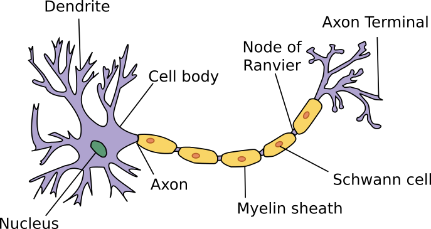
图表 1 https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a

# 深度学习基础

## 回归问题

### 神经元模型

人脑中包含了1000多一个神经元，每个神经元通过树突获取输入信号，通过轴突产生输出信号，巨大的神经网络相互连接，形成了人脑的感知和意识。人工智能的终极目标就是通过探索人脑的工作机制，在机器上制造出一种类似于人类的思考机器。在1943年，心理学家沃伦·麦卡洛克和数理逻辑学家沃尔特·皮茨通过对生物神经元的研究，提出了第一个模拟生物神经元机制的人工神经网络的数学模型，这一成果被美国神经学家弗兰克·罗森布拉特进一步发展成感知机，这也是今天深度学习的基石，我们在后面会详细介绍感知机模型 (尼克, 2017)。这里我们先重温当年科学家的探索之路，重新探索生物神经元的数学模型。



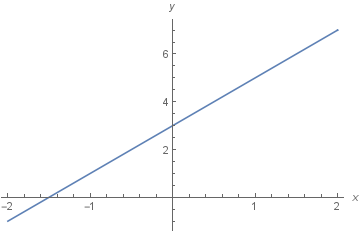
我们把生物神经元(neuron)的模型抽象为如下的数学结构，如下图所示，一个神经元输入信号表示为,经过某种处理机制：后得到输出信号。我们先简化一下，将个输入简化为单输入，或者更简便地记为, 也就是如图所示。

生物学家通过给神经元的树突施加不同的电平信号来研究神经元的输出与输入的关系，他们已经发现在某段有效区间内，输出与输入近似成线性关系。借助于这一条研究成果，我们可以把单输入的神经元数学模型抽象为：

也就是说，神经元的处理机制被我们简化为线性关系，随着输入电平

的增加，输出电平也会像下图一样增加，其实w是直线的斜率(weight/slope)，b是直线的偏置(bias)。



对于某个具体的神经元，x和y的关系是确定的，即w,b的值是固定的。二点确定一条直线，为了找出w,b的值，我们只需要找出图中的2点数据(data)即可。

为了获取任意的2点坐标数据，我们给神经元任意的2个输入值,然后测量对应的输入,

为了方便计算，我们给出2组虚构的数据：, 带入上式中

这也是我们初中学习的二元一次方程组，我们可以轻松地消元计算出。

到这里为止，我们还没有接触人工人工智能的算法，问题就被我们解决了，那么上述解法到底有什么问题呢？首先很简单地想到，对于任何测量，都是有测量误差的。我们假设测量误差属于某个均值为0，方差为的正态分布(也叫高斯分布，normal distribution/Gaussian distribution)：。一旦引入实际的测量误差后，即使是简单如线性模型，仅仅测量2个数据点，可能会带来较大地偏差，一般地做法是尽可能地多测量多组数据，然后找出一条最好的直线，它应尽可能地让所有点的误差(loss, 或者error)之和最小，如下图所示：

只测量2个点

测量多个点的示意图

也就是说，当我们采集了多个数据点时，由于测量误差的存在，有可能没有一条完美的直线经过所有的点，而是存在多条比较好的直线覆盖在数据群中间。那么怎么衡量那条直线更好呢？一个很自然的想法就是求出所有点的误差的平方和，然后求出误差的平方和最小的直线，也就是我们要寻找的直线：

### 怎么优化

现在我们来小结一下我们的问题：我们需要找出未知参数(parameters) w和b, 使得输入和输出（近似）满足线性关系。但是由于误差的存在，我们拿到了测量的多组数据组成的一个数据集(dataset):，我们希望找个一组最好的参数使得最小。

对于2个点的情况，我们可以通过消元法方便地求除解析解(closed-form solution)。但是对于多个数据()的情况，这个时候很有可能不存在解析解，我们只能借助于数值方法去优化(optimize)出一个最近似的数值解(numerical solution)。为什么叫做优化？这是因为计算机计算速度非常快，我们可以借助于强大的计算能力去多次“试错”，最简单的优化方法就是暴力搜索，比如我们要找出最合适的,我们就可以随机地测试任意的下面的误差值，最好从测试过的中挑出最好的,它所对应的就可以作为我们要找的最优。

暴力搜索算法固然简单直接，但是对于大规模高纬度的优化问题计算效率极低，基本行不通。我们这里讲非常简单地介绍梯度下降算法，用于解决前面的神经元参数预测的问题。由于梯度下降算法是现代深度学习的核心算法，我们在后面一节会非常详尽地推导梯度下降算法的应用，这里先给读者第一印象。

我们在高中就学过导数的概念，如果要求解一个函数的极大极小值，可以非常简单地令导数为0，并求出对应的值，然后检验一下极值类型。

如果我们分析一下函数在各处的导数趋势，如

举例一个函数极值问题，并配图

函数导数在各处的变化

就能比较直观地感受到，函数在各处的导数的方向总是指向函数值增大的方向，导数的反方向总是指向函数值减少的方向，利用这一性质，我们只需要按着

来更新,这样得到的处的函数值总是更有可能比在处的函数值更小。也就是说，如果要求解函数的极小值，我们只需要按着上述方法更新即可，经过多次迭代后的即是我们要找的最优数值解。

点到为止，这就是梯度下降算法。

现在我们将应用我们秒学到的梯度下降算法来求解参数。与上面设定不一样的是，我们这里要极小化的是误差函数，要优化的参数是,所以我们的算法可以写成

有2处需要注意，表示为偏微分，大家可以简单理解为对w的导数；lr是一个超参数(hyper-parameters)，一般设定为0.1~0.00001区间范围内的一个常数，比如就是一个非常常用的设定，通过在偏微分前面乘以一个较小的常数，可以将偏微分衰减到一个较小的值，这样一次更新的步长不至于过大而错过最小值。

### 实战

在介绍了如何去优化之后，我们来实战某个具体的神经元数学模型的问题。还记得我们上面通过虚构的2点数据来求出神经元的参数:。因此，我们可以简单地理解为，此神经元的真实模型为：

**采集数据** 那么如何获取N组样板数据呢？我们直接从真实的模型中间采样，为了能够很好地模拟真实的采样误差，我们给模型添加了误差变量，他采样自均值为0，方差为0.01的高斯分布

通过重复上面采样N次，来获得我们需要的训练数据：

data = []

for i in range(100):

x = np.random.uniform(3., 12.)

# mean=0, std=0.1

eps = np.random.normal(0., 0.1)

y = 1.477 \* x + 0.089 + eps

data.append([x, y])

data = np.array(data)

上面循环进行100次采样，每次从区间[0, 1]的均匀分布中随机采样一个数据x, 同时从均值为0，方差为的高斯分布中间随机采样一个误差变量eps，根据我们掌握的真实模型生成y的数据，并保存到numpy数组中间。

**计算误差** 循环计算在每个点处的差的平方，并累加。

**def compute\_error\_for\_line\_given\_points(b, w, points):**

**totalError = 0**

**for i in range(0, len(points)):**

**x = points[i, 0]**

**y = points[i, 1]**

**# computer mean-squared-error**

**totalError += (y - (w \* x + b)) \*\* 2**

**# average loss for each point**

**return totalError / float(len(points))**

最后的误差之后除以数据样本总数，从而得到每个样本的平均误差。

计算梯度 根据我们之前介绍的梯度下降算法，我们需要计算出在每一个点上的梯度信息：和。现在我们来推导一下这两个偏微分的计算公式：

这其中应用到了一个简单的求导法则

我们会在梯度下降算法介绍梯度相关性质。同样的方法，我们可以推导出

根据上面的推导，我们只需要计算在每一个点上面的和值，在累加起来即可得到和：

def step\_gradient(b\_current, w\_current, points, learningRate):

b\_gradient = 0

w\_gradient = 0

N = float(len(points))

for i in range(0, len(points)):

x = points[i, 0]

y = points[i, 1]

# grad\_b = 2(wx+b-y)

b\_gradient += (2/N) \* ((w\_current \* x + b\_current) - y)

# grad\_w = 2(wx+b-y)\*x

w\_gradient += (2/N) \* x \* ((w\_current \* x + b\_current) - y)

# update w'

new\_b = b\_current - (learningRate \* b\_gradient)

new\_w = w\_current - (learningRate \* w\_gradient)

return [new\_b, new\_w]

**梯度更新** 在计算出误差函数在w,b处的偏微分后，我们可以非常简单的根据上面的公式来更新 w和b的值

def gradient\_descent\_runner(points, starting\_b, starting\_w, learning\_rate, num\_iterations):

b = starting\_b

w = starting\_w

# update for several times

for i in range(num\_iterations):

b, w = step\_gradient(b, w, np.array(points), learning\_rate)

return [b, w]

在经过多次的七度更新后，我们会得到一个最后的 w 和 b 的值此时的 w 和 b 的值就是我们要找的。需要注意的是我们这里求解到的w 和 b 的值很有可能是一个全局极小值姐而不是一个全局最小值节但是我们在实践中发现即使是一个全职技巧肢解它的性能往往都能够满足我们的需求所以我们可以直接使用求解到的 w 和 b来作为最优解。到这里为止我们已经成功的应用梯度下降算法来研究生物神经元的内部数学模型。我们这里经过一千次的迭代以后小娜一个最优解 w 和 b，和我们之前的真实模型的 w和b几乎是完全一致的。因此因此这个问题得到了完美的解决。

# 引用

**尼克. 2017.** 人工智能简史. 无出版地 : 图灵教育, 2017. 9787115471604.