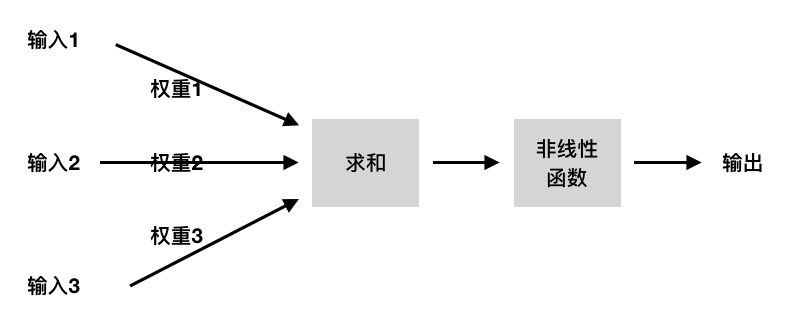
# 神经网络

# 概念：

神经网络：一种可以通过观测数据使计算机学习的仿生语言范例

神经元的结构：  
神经元的树突我们类比为多条输入，而轴突可以类比为最终的输出。

这里我们构造一个典型的神经元模型，该模型包含有3个输入，1个输出，以及中间的计算功能。



我们用数学的方式来表示一下神经元，我们定义 w为权重，x为输入

w=⎡⎣⎢w1...wm⎤⎦⎥,x=⎡⎣⎢x1...xm⎤⎦⎥

z=w1∗x1+...+wm∗xm

z输入的总和，也就是这两个矩阵的点乘，也叫内积。

感知器算法：

感知器（preceptron）算法，具体过程分为下面这么几个步骤：

1、首先将权重向量w进行初始化，可以为0或者是[0,1]之间的随机数；

2、将训练样本输入感知器（计算内积后输入激活函数得到最终结果），最后得到分类的结果（结果为1 或 -1）；

3、根据分类的结果再次更新权重向量w；

# 代码：python

#-\*- coding:utf-8 -\*-

# 简单神经网络 感知器

import numpy as np

reload(sys)

sys.setdefaultencoding("utf-8")

class Perception(object):

'''

eta: 学习率 η

time: 训练次数

w\_: 权重向量

'''

def \_\_init\_\_(self, eta = 0.01, time=10):

self.eta = eta

self.time = time

pass

'''

输入训练数据，X为输入样本向量，y对应样本分类

X:shape[n\_samples, n\_features]

X:[[1,2,3], [4,5,6]]

n\_samples : 2

n\_features: 3

y:[1, -1]

'''

def fit(self, X, y):

# 初始化权重向量为0，加一为w0，也就是损失函数的阈值

self.w\_ = np.zero[1 + X.shape[1]]

self.errors\_ = []

for \_ in range(self.time):

errors = 0

# x:[[1,2,3], [4,5,6]]

# y:[1, -1]

# zip(X,y) = [[1,2,3,1], [4,5,6.-1]]

for xi, target in zip(X, y):

# update = η \* ( y - y' )

update = self.eta \* (target - self.predict(xi))

# xi 为向量, 这里每个向量都会乘

self.w\_[1:] += update \* xi

self.w\_[0] += update;

errors += int(update != 0.0)

pass

# 损失函数

def predict(self, X):

# z = w1\*x1+...+wj\*xj + w0\*1

z = np.dot(X, self.w\_[1:]) + self.w\_[0]

# 损失函数

if z >= 0.0:

return 1

else:

return -1