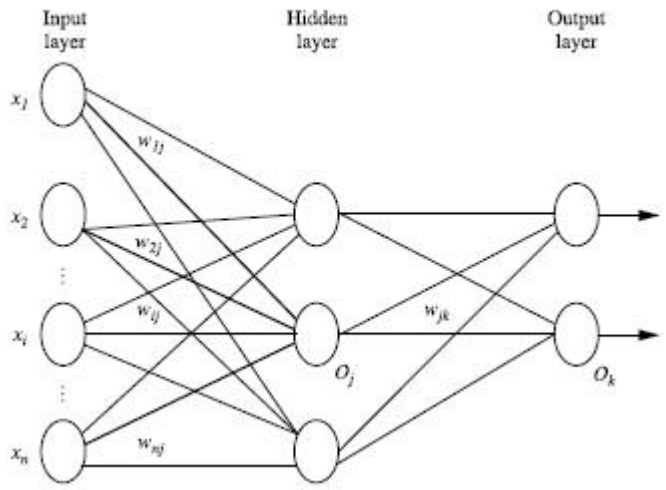
一、多层前向神经网络

多层前向神经网络由三部分组成：输出层、隐藏层、输出层，每层由单元组成；

输入层由训练集的实例特征向量传入，经过连接结点的权重传入下一层，前一层的输出是下一层的输入；隐藏层的个数是任意的，输入层只有一层，输出层也只有一层；

除去输入层之外，隐藏层和输出层的层数和为 n，则该神经网络称为 n 层神经网络，如下图为 2 层的神经网络；

一层中加权求和，根据非线性方程进行转化输出；理论上，如果有足够多的隐藏层和足够大的训练集，可以模拟出任何方程；



二、设计神经网络结构

使用神经网络之前，必须要确定神经网络的层数，以及每层单元的个数；

为了加速学习过程，特征向量在传入输入层前，通常需要标准化到 0 和 1 之间；

离散型变量可以被编码成每一个输入单元对应一个特征值可能赋的值

比如：特征值 A 可能去三个值（a0,a1,a2），那么可以使用 3 个输入单元来代表 A

如果 A=a0，则代表 a0 的单元值取 1，其余取 0；

如果 A=a1，则代表 a1 的单元值取 1，其余取 0；

如果 A=a2，则代表 a2 的单元值取 1，其余取 0；

A	a0	a1	a2
a0	1	0	0
a1	0	1	0
a2	0	0	1

神经网络既解决分类（classification）问题，也可以解决回归（regression）问题。对于分类问题，如果是两类，则可以用一个输出单元（0 和 1）分别表示两类；如果多余两类，则每一个类别用一个输出单元表示，所以输出层的单元数量通常等一类别的数量。没有明确的规则来设计最佳个数的隐藏层，一般根据实验测试误差和准确率来改进实验。

三、交叉验证方法

如何计算准确率？最简单的方法是通过一组训练集和测试集，训练集通过训练得到模型，将测试集输入模型得到测试结果，将测试结果和测试集的真实标签进行比较，得到准确率。

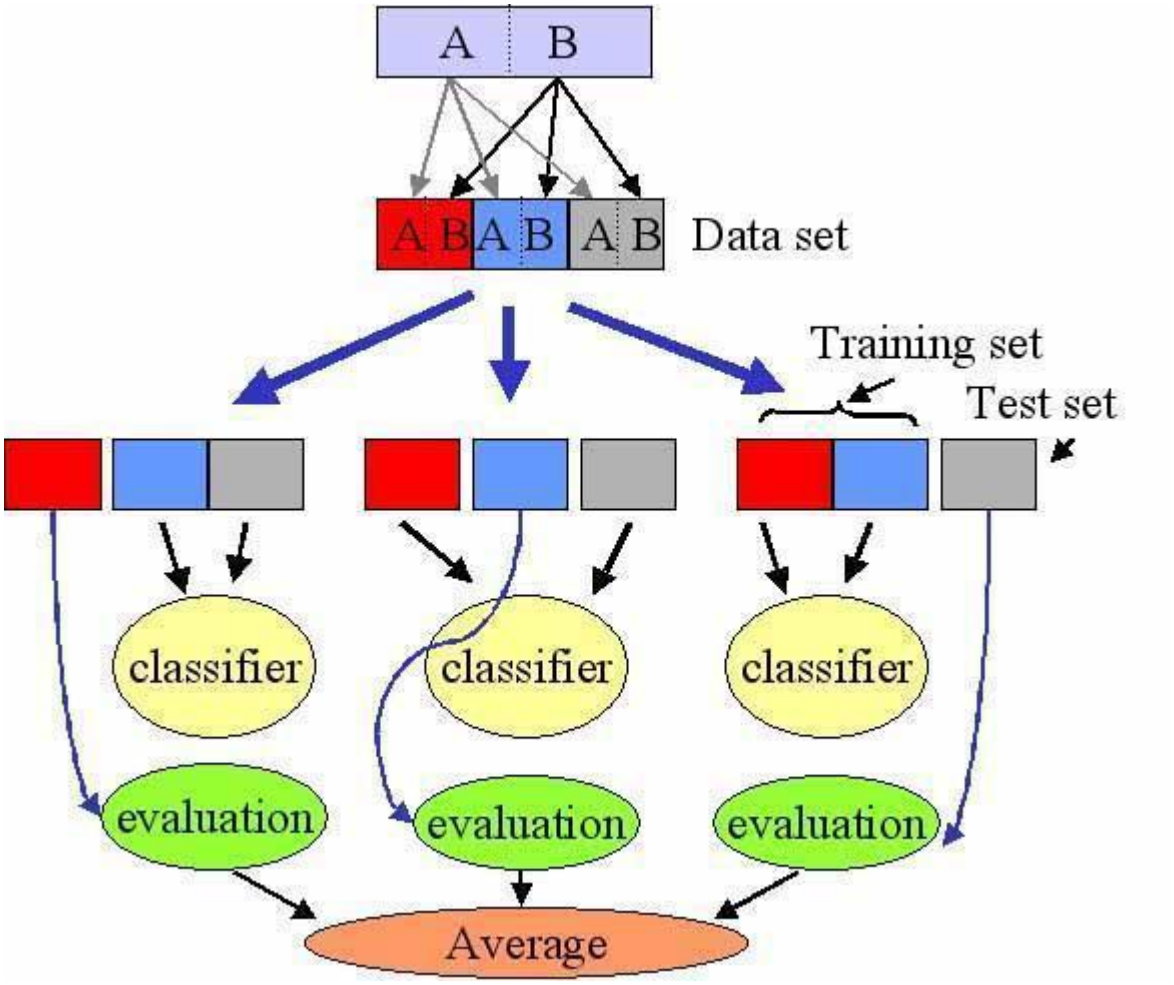
在机器学习领域一个常用的方法是交叉验证方法。一组数据不分成 2 份，可能分为 10 份，

第 1 次：第 1 份作为测试集，剩余 9 份作为训练集；

第 2 次：第 2 份作为测试集，剩余 9 份作为训练集；

……

这样经过 10 次训练，得到 10 组准确率，将这 10 组数据求平均值得到平均准确率的结果。这里 10 是特例。一般意义上将数据分为 k 份，称该算法为 K-fold cross validation，即每一次选择 k 份中的一份作为测试集，剩余 k-1 份作为训练集，重复 k 次，最终得到平均准确率，是一种比较科学准确的方法。



四、BP 算法

通过迭代来处理训练集中的实例；

对比经过神经网络后预测值与真实值之间的差；

反方向（从输出层=>隐藏层=>输入层）来最小化误差，来更新每个连接的权重；

4.1、算法详细介绍

输入：数据集、学习率、一个多层神经网络构架；

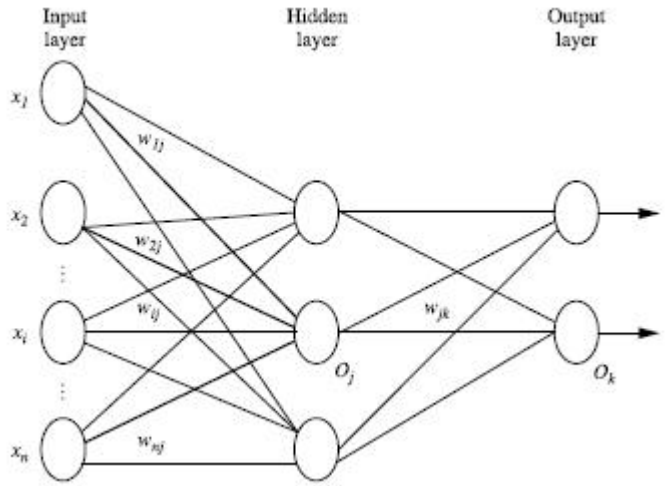
输出：一个训练好的神经网络；

初始化权重和偏向：随机初始化在-1 到 1 之间（或者其他），每个单元有一个偏向；

对于每一个训练实例 X，执行以下步骤：

1、由输入层向前传送：

结合神经网络示意图进行分析：



由输入层到隐藏层：

$$O_j = \sum_i w_{ij} x_i + \theta_j$$

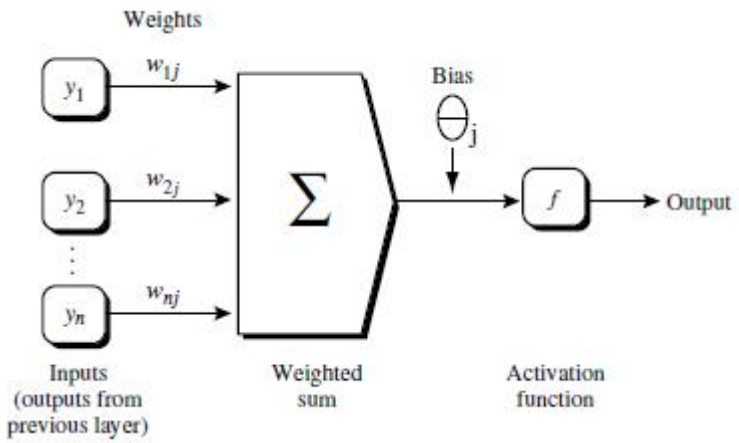
由隐藏层到输出层：

$$O_k = \sum_j w_{jk} O_j + \theta_k$$

两个公式进行总结，可以得到：

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Ij 为当前层单元值，Oi 为上一层的单元值，wij 为两层之间，连接两个单元值的权重值，sitaj 为每一层的偏向值。我们要对每一层的输出进行非线性的转换，示意图如下：



当前层输出为 Ij，f 为非线性转化函数，又称为激活函数，定义如下：

$$f(x)=\frac{1}{1+e^{-x}}$$

即每一层的输出为：

$$O_j=\frac{1}{1+e^{-I_j}}$$

这样就可以通过输入值正向得到每一层的输出值。

2、根据误差反向传送 对于输出层：其中 Tk 是真实值，Ok 是预测值

$$Err_k=O_k(1-O_k)(T_k-O_k)$$

对于隐藏层：

$$Err_j=O_j(1-O_j)\sum_k Err_k w_{jk}$$

权重更新：其中 l 为学习率

$$\Delta w_{ij}=(l)Err_j O_i$$
$$w_{ij}=w_{ij}+\Delta w_{ij}$$

偏向更新：

$$\Delta\theta_j = (l)Err_j$$
$$\theta_j = \theta_j + \Delta\theta_j$$

3、终止条件

偏重的更新低于某个阈值；

预测的错误率低于某个阈值；

达到预设一定的循环次数；

4、非线性转化函数

上面提到的非线性转化函数 f，一般情况下可以用两种函数：

（1）tanh(x)函数：

tanh(x)=sinh(x)/cosh(x)

sinh(x)=(exp(x)-exp(-x))/2

cosh(x)=(exp(x)+exp(-x))/2

（2）逻辑函数，本文上面用的就是逻辑函数

五、BP 神经网络的 python 实现

需要先导入 numpy 模块

```
import numpy as np
```

定义非线性转化函数，由于还需要用到给函数的导数形式，因此一起定义

```
def tanh(x):
    return np.tanh(x)
def tanh_deriv(x):
    return 1.0 - np.tanh(x)*np.tanh(x)
def logistic(x):
    return 1/(1 + np.exp(-x))
def logistic_derivative(x):
    return logistic(x)*(1-logistic(x))
```

设计 BP 神经网络的形式（几层，每层多少单元个数），用到了面向对象，主要是选择哪种非线性函数，以及初始化权重。layers 是一个 list，里面包含每一层的单元个数。

```
class NeuralNetwork:
    def __init__(self, layers, activation='tanh'):
        """
        :param layers: A list containing the number of units in each layer.
        Should be at least two values
        :param activation: The activation function to be used. Can be
        "logistic" or "tanh"
        """
        if activation == 'logistic':
            self.activation = logistic
            self.activation_deriv = logistic_derivative
        elif activation == 'tanh':
            self.activation = tanh
            self.activation_deriv = tanh_deriv

        self.weights = []
        for i in range(1, len(layers) - 1):
            self.weights.append((2*np.random.random((layers[i - 1] + 1,
layers[i] + 1))-1)*0.25)
            self.weights.append((2*np.random.random((layers[i] + 1,
 layers[i + 1]))-1)*0.25)
```

实现算法

```
    def fit(self, X, y, learning_rate=0.2, epochs=10000):
        X = np.atleast_2d(X)
        temp = np.ones([X.shape[0], X.shape[1]+1])
        temp[:, 0:-1] = X
        X = temp
        y = np.array(y)

        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [X[i]]

            for l in range(len(self.weights)):
                a.append(self.activation(np.dot(a[l],
self.weights[l])))
            error = y[i] - a[-1]
```

```
            deltas = [error * self.activation_deriv(a[-1])]

            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_deriv(a[l]))
            deltas.reverse()

            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)
```

实现预测

```
    def predict(self, x):
        x = np.array(x)
        temp = np.ones(x.shape[0]+1)
        temp[0:-1] = x
        a = temp
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
```

我们给出一组数进行预测，我们上面的程序文件保存名称为 BP

```
from BP import NeuralNetwork
import numpy as np

nn = NeuralNetwork([2,2,1], 'tanh')
x = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([1,0,0,1])
```

```
nn.fit(x,y,0.1,10000)
for i in [[0,0], [0,1], [1,0], [1,1]]:
    print(i, nn.predict(i))
```

结果如下：

```
([0, 0], array([ 0.99738862]))
([0, 1], array([ 0.00091329]))
([1, 0], array([ 0.00086846]))
([1, 1], array([ 0.99751259]))
```

来源：http://python.jobbole.com/87016/