菜菜的机器学习sklearn第十二期

sklearn中的神经网络

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂第十二期，今晚是最后一期啦~！是我们的大魔王神经网络哈哈~

我的开发环境是**Jupyter lab**，所用的库和版本大家参考：

Python 3.7.1（你的版本至少要3.4以上

Scikit-learn 0.20.1 （你的版本至少要0.20

Numpy 1.15.4, **Pandas** 0.23.4, **Matplotlib** 3.0.2, **SciPy** 1.1.0

请扫码进群领取课件和代码源文件，**扫描二维码后回复"K"就可以进群哦~**

菜菜的机器学习sklearn第十二期

sklearn中的神经网络

1 打开深度学习的大门：神经网络概述
2 单层神经网络
2.1 回归单层神经网络：线性回归
2.2 二分类单层神经网络：sigmoid与阶跃函数
2.3 多分类单层神经网络：softmax回归
2.4 回归vs二分类vs多分类

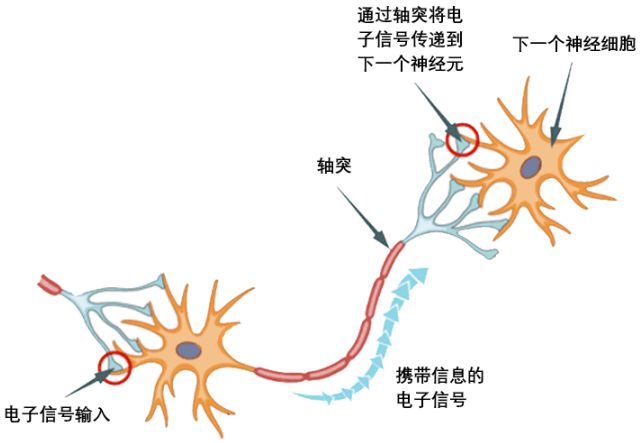
==========【ÍêÕû°æÄÚÈÝ¡¿==========

3 多层神经网络
3.1 异或门问题
3.2 多层神经网络的数学表示
3.3 探索多层神经网络：层 vs h(z)
3.4 激活函数
3.4.1 h(z) vs g(z)
3.4.2 ReLU & tanh
3.5 neural_network.MLPClassifier
3.5.1 隐藏层与神经元：重要参数hidden_layer_sizes
3.5.2 激活函数：重要参数activation
4 神经网络的学习
4.1 神经网络学习的基本思想
4.1.1 小批量随机梯度下降：向全局最优前进
4.1.2 关于mini-batch SGD提升模型速度的讨论
4.2 神经网络的学习流程
4.2.1 损失函数们与梯度
4.2.2 第一轮迭代：确定模型参数的初始值
4.2.3 第一轮迭代：全数据，小批量，epochs
4.2.4 第一轮迭代：学习率与学习率的调整
4.2.5 从第一轮迭代到第t轮迭代
4.2.6 收敛、停止神经网络的迭代
4.3 梯度的求解：正向传播与反向传播
4.3.1 多层神经网络的正向传播
4.3.2 反向传播
5 通往深度学习的大门

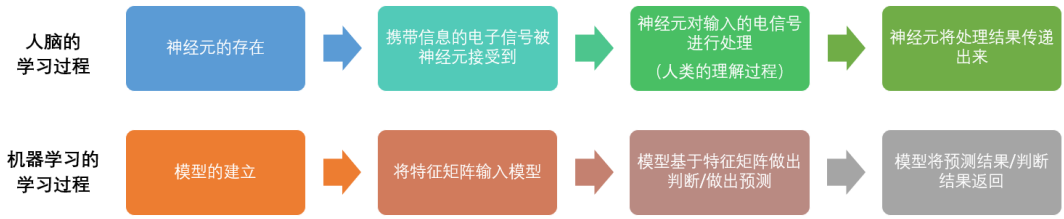
1 打开深度学习的大门：神经网络概述

人工神经网络（Artificial Neural Network，ANN），通常简称为神经网络，它是机器学习当中独树一帜的，最强大的强学习器没有之一。机器学习是研究如何让计算机学习的学科，是研究如何赋予计算机学习能力的学科。在过去的十一周中，我们介绍了各种各样能够让计算机“学习”到知识并自行做出判断的手段（算法），他们之中大多数是从“如何做出判断”或者“如何提取规则”的角度去探求出让计算机学习的方式，但却没有任何一种算法使用了和人类相似的学习方式。

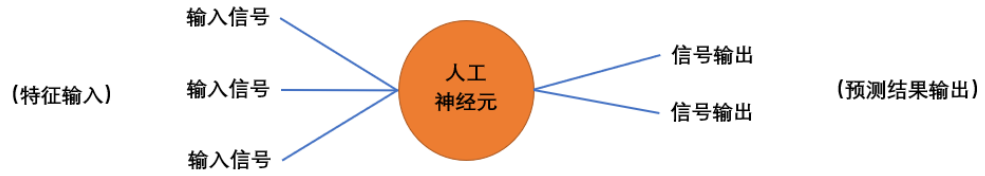
二十世纪后半，神经学家们发现人脑主要通过”生物神经网络“来进行学习。人类大脑主要由称为神经元的神经细胞组成，通过名为轴突的纤维束与其他神经元连接在一起。每当神经元接受到信号，神经元便会受到刺激，此时纤维束会将信号从一个神经元传递到另一个神经元上，**而人类正是通过在神经元上传入、传出信息来进行学习。**



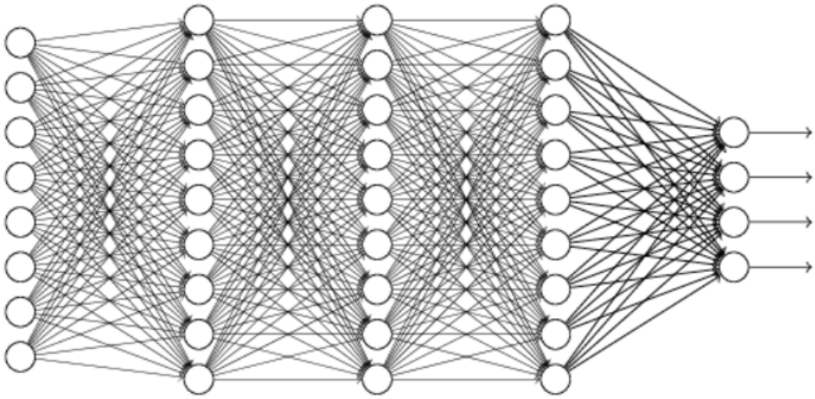
不难发现，这个过程其实可以和机器学习类比起来：在传统机器学习中，我们建立模型，对模型输入特征矩阵，模型将特征矩阵转化为判断结果后为我们输出，如果将模型看作是单个神经元，特征矩阵看作是神经元接受的”信号“，模型对特征矩阵的处理过程看作神经元受到的刺激，最后输出的结果看成是神经元通过轴突传递出的信号，那人类学习的模式是可以一定程度上被算法模仿的。



人脑通过构建复杂的网络可以进行逻辑，语言，图像的学习，而传统机器学习算法不具备和人类相似的学习能力。机器学习研究者们相信，模拟大脑结构可以让机器的学习能力更上一层楼，于是**人工神经网络**算法应运而生，现在基本都简称为”神经网络“。有了神经网络，基于其算法延申出来的机器学习分枝学科——深度学习也从此走入了人们的视野，成为所有让世人惊叹的人工智能技术的根基。在深度学习中，我们使用圆来表示神经元，使用线表示数据流动的方向。我们从神经元左侧输入特征，让神经元处理数据，并从右侧输出预测结果，以此来模拟人脑的学习过程。



在人脑中，大约存在8-10兆个如上所述的神经元，一次学习中调用的神经元越多，人类可以处理和学习的信息也越多。神经网络算法也是如此，神经元越多，算法对数据的处理能力和学习能力越强。所以大家经常会看见的神经网络结构可能是这样的：



到这里，许多人可能会好奇：神经元是怎样处理数据的呢？数据在众多神经元中是怎么传递的呢？如果神经元的数量影响神经网络的学习能力，那我们该如何去确认这个数量呢？这些问题都是神经网络算法十分核心的问题，你可以在今天的课程中读到所有的答案。

> **思考**
>
> 神经元越多，神经网络对数据的学习能力越强。如果神经元太多，可能会发生什么机器学习中普遍存在的现象呢？

现在，机器学习能够在逻辑运算上比人类表现得更好已经是一个通识了——AlphaGo战胜世界围棋冠军，视频通话中进行实时字幕和翻译，火车站飞机场的人脸识别系统，都是人工智能技术落地的经典例子。这些技术应用的背后全部都是以神经网络为核心的深度学习技术。因此，神经网络作为机器学习课程中的最后一个算法，是链接了机器学习和深度学习之间的桥梁。

在这里不得不提到几个要点。传统机器学习领域有三大追求：算法效果，运算速度和可解释性，而这三个追求都是为了业务而服务。其中算法效果与剩下的两大特征：运算速度、可解释性是一定程度上相互矛盾的。神经网络作为追求算法效果到极致的算法，几乎完全放弃了可解释性和运算速度，因此神经网络算法落地时的核心与其他我们已经学过的机器学习算法都不同：除了理解原理，了解提高识别精度的参数调优过程之外，还需要了解如何使用GPU将神经网络高速化，如何处理非结构化数据等等。然而sklearn是专注于机器学习的库，它不是专用于深度学习的平台，也不具备处理大型数据的能力，所以sklearn中的神经网络和其他机器学习算法比起来显得有些不

受重视，它不具备做真正的深度学习的功能，甚至对一些在深度学习中非常关键的点缺乏关注。因此，我们的课程也将**不会包括**：

1. Pytorch，Caffe，TensorFlow，Keras等深度学习框架的使用方法
2. 为了实现深度学习的高速化而进行的GPU、dropout相关的实现
3. 自然语言处理，图像识别，语音识别，视频识别的例子
4. 基本神经网络之外的，如卷积神经网络，循环神经网络，LSTM等更高级的深度学习算法

另外，同样也非常重要的是，**我假设在观看课程的你已经具备了扎实的机器学习基础，了解机器学习中的基本概念，并对机器学习中常见算法比较熟悉**（或者，当你不熟悉或忘记一些内容的时候，你知道从哪里能够获得这些机器学习算法的详细资料）。

sklearn中的神经网络课程将会注重向熟练使用机器学习算法、但对于深度学习尚不了解的技术人员讲解神经网络核心的工作原理。在sklearn中，神经网络只包含了如下的三个类：最初的神经网络玻尔兹曼机，以及已经成熟的以多层感知机为基础的神经网络分类和神经网络回归。我们的重点将会放在后两个类：MLPClassifier和MLPRegressor上。

类	含义
neural_network.BernoulliRBM	伯努利限制玻尔兹曼机器（RBM），这是一种随机递归神经网络。
neural_network.MLPClassifier	多层感知器分类器。
neural_network.MLPRegressor	多层感知器回归器。

2 单层神经网络

2.1 回归单层神经网络：线性回归

许多人都以为神经网络是一个非常复杂的算法，其实它最初的数学原理并不比机器学习中的算法如SVM，XGBoost等难多少，相反，它很多时候都比机器学习算法简单。了解神经网络，可以从**线性回归**算法开始。线性回归算法是机器学习中最简单的回归类算法，多元线性回归指的就是一个样本有多个特征的线性回归问题。对于一个有n个特征的样本i而言，它的回归结果可以写作一个几乎人人熟悉的方程：

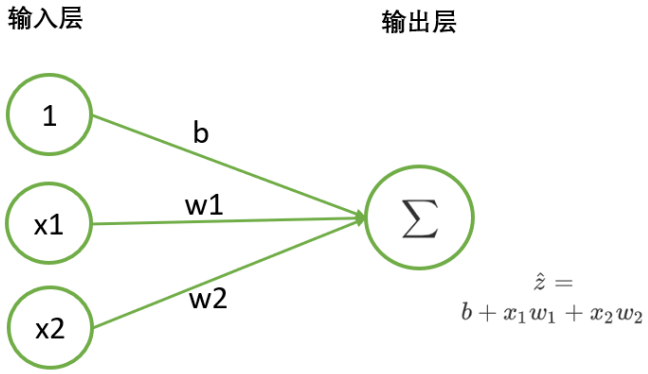
$$\hat{z}_i = b + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_n x_{in}$$

w和b被统称为模型的参数，其中b被称为截距(intercept)，也叫做偏差(bias)，w_1~w_n被称为回归系数(regression coefficient)，也叫作权重(weights)。这个表达式，其实就和我们小学时就无比熟悉的$y = ax + b$是同样的性质。其中y是我们的目标变量，也就是标签。

现在假设，我们的数据只有2个特征，则线性回归方程可以写作如下结构：

$$\hat{z} = b + x_1 w_1 + x_2 w_2$$

此时，我们只要对模型输入特征x_1, x_2的取值，就可以得出对应的预测值$\hat{z}$。在上一节中我们提到，神经网络的预测过程是从神经元左侧输入特征，让神经元处理数据，并从右侧输出预测结果。这个过程和我们刚才说到的线性回归输出预测值的过程是一致的。如果我们使用一个神经网络来表达线性回归上的过程，则可以有：



这就是一个最简单的神经网络的表示图。

在神经网络中，竖着排列在一起的一组神经元叫做“一层网络”，所以线性回归的网络直观看起来有两层，两层神经网络通过写有参数的线条相连。我们从左侧输入常数1和特征取值x_1, x_2，再让它们与相对应的参数**相乘**，就可以得到b，$x_1 w_1$，$x_2 w_2$三个结果。这三个结果通过连接到下一层神经元的直线，被输入下一层神经元。我们在第二层的神经元中将三个乘积进行**加和**（使用符号$\sum$表示），就可以得到加和结果$\hat{z}$，即$b + x_1 w_1 + x_2 w_2$，这个值正是我们的预测值。**可见，线性回归方程与上面的神经网络图达到的效果是一模一样的。**

在上述过程中，左侧的是神经网络的**输入层**（input layer）。输入层由众多承载数据用的神经元组成，数据从这里输入，并流入处理数据的神经元中。在所有神经网络中，输入层永远只有一层，且每个神经元上只能承载一个特征或一个常量。现在的二元线性回归只有两个特征，所以输入层上只需要三个神经元，包括两个特征和一个常量，其中这里的常量仅仅是被用来乘以偏差b用的。对于没有偏差的线性回归来说，我们可以不设置常量1。

右侧的是**输出层**（output layer）。输出层由大于等于一个神经元组成，我们总是从这一层来获取预测结果。输出层的每个神经元上都承载着单个或多个功能，可以处理被输入神经元的数据。在线性回归中，这个功能就是“加和”，当我们把加和替换成其他的功能，就能够形成各种不同的神经网络。

在神经元之间相互连接的线表示了数据流动的方向，就像人脑神经细胞之间相互联系的“轴突”。在人脑神经细胞中，轴突控制电子信号流过的强度，在人工神经网络中，神经元之间的连接线上的权重也代表了“信息可通过的强度。最简单的例子是，当w_1为0.5时，在特征x_1上的信息就只有0.5倍能够传递到下一层神经元中，因为被输入到下层神经元中去进行计算的实际值是$0.5x_1$。相对的，如果w_1是2.5，则会传递2.5倍的x_1上的信息。因此，有的深度学习课程会将权重w比喻成是电路中的”电压“，电压越大，则电信号越强烈，电压越小，信号也越弱，这都是在描述权重w会如何影响传入下一层神经元的信息/数据量的大小。

到此，我们已经了解了线性回归的网络是怎么一回事，它是最简单的回归神经网络，同时也是最简单的神经网络。类似于线性回归这样的神经网络，被称为**单层神经网络。**

> **单层神经网络**
>
> 从直观来看，线性回归的网络结构明明有两层，为什么线性回归被叫做“单层神经网络”呢?
> 实际上，在描述神经网络的层数的时候，**我们不考虑输入层。**
> 输入层是每个神经网络都必须存在的一层，任意两个神经网络之间的不同之处就在输入层之后的所有层。所以，我们把输入层之后只有一层的神经网络称为单层神经网络。当然了，在有的深度学习课程或教材中，也会直接将所有层都算入其中，将上述网络称为“两层神经网络”。因此，当出现“N层神经网络”的描述时，一定要注意原作者是否将输入层考虑进去了。

让我们使用简单的代码来实现回归神经网络。来看下面这组数据：

x1	x2	z
0	0	-0.2
1	0	-0.05
0	1	-0.05
1	1	0.1

我们将构造能够拟合出以上数据的单层回归神经网络：

```
#首先使用numpy来创建数据
import numpy as np

X = np.array([[0,0],[1,0],[0,1],[1,1]])
z_reg = np.array([-0.2, -0.05, -0.05, 0.1])

X

X.shape

z_reg

#定义实现简单线性回归的函数
def LinearR(x1,x2):
    w1, w2, b = 0.15, 0.15,-0.2 #给定一组系数w和b
    z = x1*w1 + x2*w2 + b #z是系数*特征后加和的结果
```

```
    return z

LinearR(X[:,0],X[:,1])
```

可以看到，只要能够给到适合的w和b，回归神经网络其实非常容易实现。从这样的一个简单回归神经网络，我们很容易就可以把它推广到分类模型上。

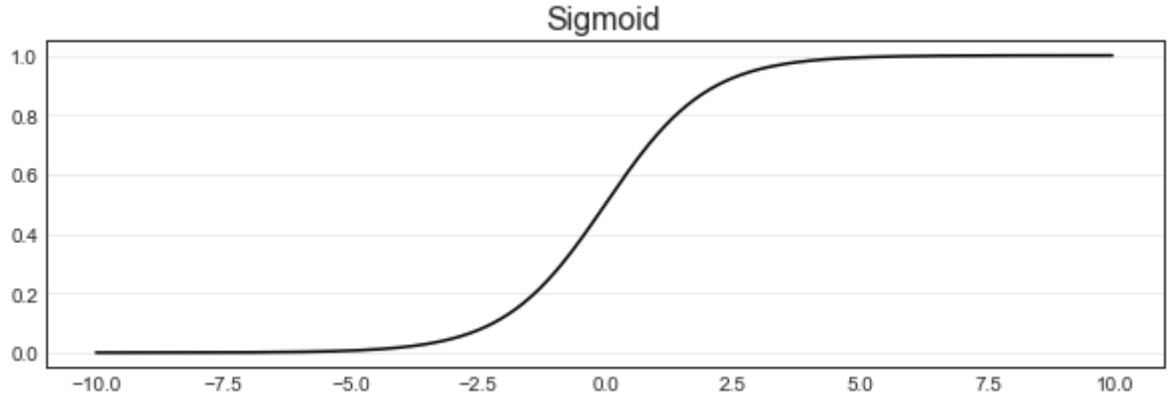
2.2 二分类单层神经网络：sigmoid与阶跃函数

在过去我们学习逻辑回归时，我们了解到sigmoid函数可以帮助我们将线性回归连续型的结果转化为0-1之间的概率值，从而帮助我们将回归类算法转变为分类算法逻辑回归。对于神经网络来说我们也可以使用相同的方法。首先先来复习一下Sigmoid函数的的公式和性质：

复习：Sigmoid函数的公式和性质

Sigmoid函数是一个S型的函数，当自变量z趋近正无穷时，因变量g(z)趋近于1，而当z趋近负无穷时，g(z)趋近于0，因此它能够将任何实数映射到(0,1)区间，使其可用于将任意值函数转换为更适合二分类的函数。通常来说，**自变量z往往是回归类算法（如线性回归）的结果**。将回归类算法的连续型数值压缩到(0,1)之间后，我们使用阈值0.5来将其转化为分类。即当$g(z_i)$大于0.5时，我们认为样本z_i对应的分类结果为1类，反之则为0类。

$$g(z) = \frac{1}{1+e^{-z}}$$



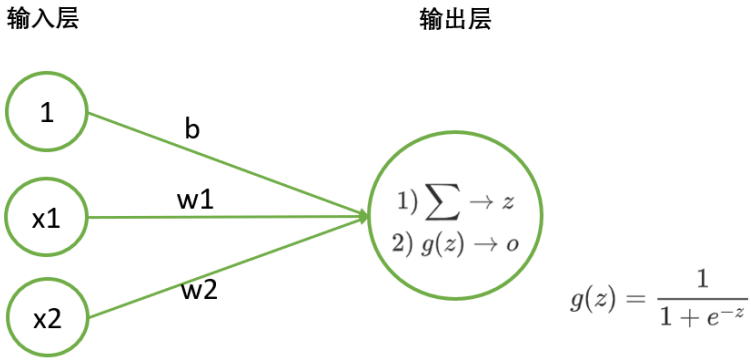
来看下面这组数据。很容易注意到，这组数据和上面的回归数据的特征（x_1, x_2）是完全一致的，只不过标签y由连续型结果转变为了分类型。这一组分类的规律是这样的：当两个特征都为1的时候标签就为1，否则标签就为0。这一组特殊的数据被我们称之为**“与门”（AND GATE）**，这里的“与”正是表示“特征一与特征二都是1”的含义。

x1	x2	y_and
0	0	0
1	0	0
0	1	0
1	1	1

要拟合这组数据，只需要在刚才我们写好的代码后面加上sigmoid函数以及阈值处理后的变化。

```
#重新定义数据中的标签
y_and = [0,0,0,1]

#根据sigmoid公式定义sigmoid函数
def sigmoid(z):
    return 1/(1 + np.exp(-z))

def AND_sigmoid(x1,x2):
    w1, w2, b = 0.15, 0.15,-0.2 #给定的系数w和b不变
    z = x1*w1 + x2*w2 + b
    o = sigmoid(z) #使用sigmoid函数将回归结果转换到(0,1)之间
    y = [int(x) for x in o >= 0.5] #根据阈值0.5，将(0,1)之间的概率转变为分类0和1
    return o, y

#o:sigmoid函数返回的概率结果
#y:对概率结果按阈值进行划分后，形成的0和1，也就是分类标签
o, y_sigm = AND_sigmoid(X[:,0],X[:,1])

o

y_sigm

y_sigm == y_and
```

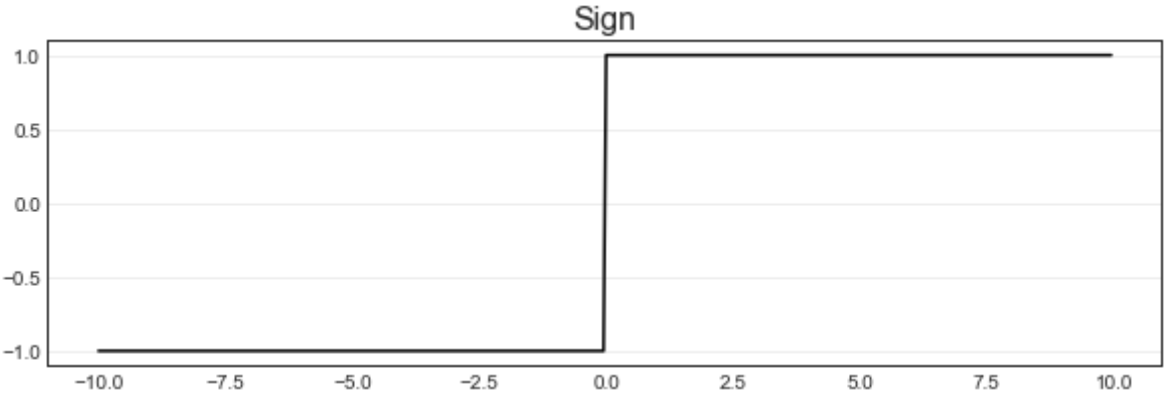
可见，这里得到了与我们期待的结果一致的结果，这就将回归算法转变为了二分类。这个过程在神经网络中的表示图如下：



可以看出，这个结构与线性回归的神经网络唯一不同的就是**输出层中多出了一个$\mathbf{g(z)}$**。当有了$g(z)$之后，只要了解阈值是0.5（或者任意我们自己设定的数值），就可以轻松地判断任意样本的预测标签$\hat{y}$。在二分类神经网络中，$g(z)$实现了将连续型数值转换为分类型数值的作用，理论上来说它可以是任何能够将连续型分割的函数，较为常用的就是sigmoid、著名的符号函数sign了。

- 符号函数sign

符号函数是图像如下所示的函数。



我们可以使用以下表达式来表示它：

$$g(z) = y = \begin{cases} 1 & if\ z > 0 \\ 0 & if\ z = 0 \\ -1 & if\ z < 0 \end{cases}$$

由于函数的取值是间断的，**符号函数也被称为“阶跃函数”**，表示在0的两端，函数的结果y是从-1直接阶跃到了1。在这里，我们使用y而不是$g(z)$来表示输出的结果，是因为输出结果直接是0、1、-1这样的类别。对于sigmoid函数而言，$g(z)$返回的是0~1之间的概率值，如果我们希望获取最终预测出的类别，还需要将概率转变成0或1这样的数字才可以。但符号函数可以直接返回类别，因此我们可以认为符号函数输出的结果就是最终的预测结果y。在二分类中，符号函数也可以忽略中间$z = 0$的时候，直接分为0和1两类，用如下式子表示：

$$y = \begin{cases} 1 & if\ z > 0 \\ 0 & if\ z \leq 0 \end{cases}$$

等号被并在上方或下方都可以。这个式子可以很容易被转化为下面的式子：

$$\because z = w_1x_1 + w_2x_2 + b$$

$$\therefore y = \begin{cases} 1 & if\ w_1x_1 + w_2x_2 + b > 0 \\ 0 & if\ w_1x_1 + w_2x_2 + b \leq 0 \end{cases}$$

$$\therefore y = \begin{cases} 1 & if\ w_1x_1 + w_2x_2 > -b \\ 0 & if\ w_1x_1 + w_2x_2 \leq -b \end{cases}$$

此时，$-b$就是一个阈值，我们可以使用任意字母来替代它，比较常见的是字母θ。当然，不把它当做阈值，依然保留$w_1x_1 + w_2x_2 + b$与0进行比较的关系也没有任何问题。和sigmoid一样，我们也可以使用阶跃函数来处理"与门"的数据：

```
def AND(x1,x2):
    w1, w2, b = 0.15, 0.15, -0.23 #和sigmoid相似的w和b
    z = x1*w1 + x2*w2 + b
    y = [int(x) for x in z >= 0]
    return y

AND(X[:,0],X[:,1])

y_and
```

阶跃函数和sigmoid都可以完成二分类的任务。在神经网络的二分类中，$g(z)$几乎默认是sigmoid函数，少用阶跃函数，这是由神经网络的解法决定的，我们将在第三部分来详细讲解。**需要注意的是**，在sklearn中，一旦标签以二分类的形式存在，sklearn就一定会采用sigmoid函数作为$g(z)$。除非调整源码，否则sigmoid在二分类时的使用属于调用者不可干涉和改变的范围。

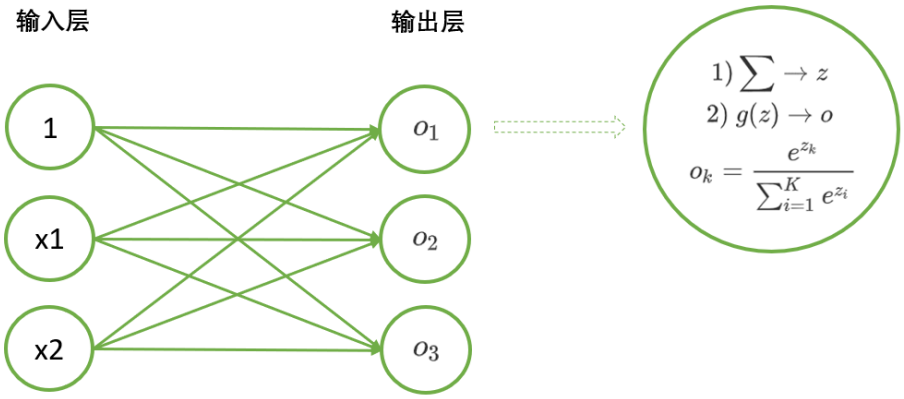
2.3 多分类单层神经网络：softmax回归

在了解二分类后，我们可以继续将神经网络推广到多分类。在sklearn中，我们曾经学习过逻辑回归做多分类的做法。逻辑回归通过Many-vs-Many（多对多）和One-vs-Rest（一对多）模式来进行多分类。其中，OvR是指将多个标签类别中的一类作为类别1，其他所有类别作为类别0，分别建立多个二分类模型，综合得出多分类结果的方法。MvM是指把好几个标签类作为1，剩下的几个标签类别作为0，同样分别建立多个二分类模型来得出多分类结果的方法。这两种方法非常有效，尤其是在逻辑回归做多分类的问题上能够解决很多问题，但是对于神经网络却不奏效。理由非常简单：

1. 逻辑回归是一个非常快速的算法，在使用OvR和MvM这样需要同时建立多个模型的方法时，运算速度不会成为太大的问题。但神经网络本身是一个计算量庞大的算法，建立一个模型就会耗费很多时间，因此必须建立很多个模型来求解的方法对神经网络来说就不够高效。

2. 我们有更好的方法来解决这个问题，那就是softmax回归。

Softmax函数是深度学习基础中的基础，它是神经网络进行多分类时，默认放在输出层中处理数据的函数。假设现在神经网络是用于三分类数据，且三个分类分别是苹果，柠檬和百香果，序号则分别是分类1、分类2和分类3。则使用softmax函数的神经网络的模型会如下所示：



与二分类一样，我们从网络左侧输入特征，从右侧输出概率，且概率是通过在加和结果z外套上另一个函数$g(z)$来计算。在二分类时，输出层只有一个神经元，只输出样本对于正类别的概率（通常是标签为1的概率），而softmax的输出层有三个神经元，分别输出该样本的真实标签是苹果、柠檬或百香果的概率o_1, o_2, o_3。**在多分类中，神经元的个数与标签类别的个数是一致的**，如果是十分类，在输出层上就会存在十个神经元，分别输出十个不同的概率。此时，**样本的预测标签就是所有输出的概率o_1, o_2, o_3中最大的概率对应的标签类别。**

那每个概率是如何计算出来的呢？来看Softmax函数的公式：

$$g(z)_k = o_k = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}}$$

其中e为自然常数（约为2.71828），z与sigmoid函数中的z一样，表示回归类算法（如线性回归）的结果。K表示该数据的标签中总共有K个标签类别，如三分类时$K=3$，四分类时$K=4$。k表示标签类别k类，i表示总共K个类别中的第i个类别。很容易可以看出，Softmax函数的分子是多分类状况下某**一个**标签类别的回归结果的自然对数，分母是多分类状况下**所有**标签类别的回归结果的自然对数之和，因此**Softmax函数的结果代表了样本的结果为类别k的概率。**

有的小伙伴可能会被i和k所代表的含义混淆，来看下面两组标签：

组别	**类别1**	**类别2**	**类别3**
文字组	苹果	柠檬	百香果
数字组	1	2	3

这两组标签中，类别3所对应的softmax公式分别是：

$$g(z)_{百香果} = o_{百香果} = \frac{e^{z_{百香果}}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$g(z)_3 = o_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

发现奇怪的事情了吗？当标签类别由数字标记时，分子上的e^{z_1}与分母上的e^{z_1}不是同样的数字！因为分子实际上是$e^{z_{k=1}}$，而分母实际上是$e^{z_{i=1}}$，真正与$e^{z_{k=1}}$相等的应该是$e^{z_{i=3}}$。所以，在多分类中使用与标签类别序号不一致的数字，有在公式中让公式混淆的风险。正常来说，我们都对多分类的类别按整数[1,+∞]的数字进行编号，避免混淆。

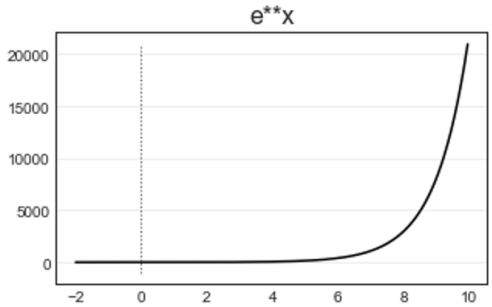
另一个问题是，既然有混淆的风险，为什么不直接统一分子分母上的表示法呢？因为分子必须与o所带的脚标保持一致，而分母必须保留加和符号Σ来应对标签类别很多的情况，而加和符号带有"以某个顺序遍历元素"的含义，因此必须使用数字。因此在进行数学推导时，千万注意对多分类模型使用softmax时的编号问题，避免出现计算错误。

思考

为什么二分类时我们只需要输出一个概率，而多分类却需要输出多个概率？

- softmax函数的缺陷

我们曾经提到过，神经网络是模型效果很好，但运算速度非常缓慢的算法。softmax函数也存在相同的问题——它可以将多分类的结果转变为概率（这是一个极大的优势），但它需要的计算量非常巨大。由于softmax的分子和分母中都带有e为底的指数函数，所以在计算中非常容易出现极大的数值。



如上图所示，e^{10}就已经等于20000了，而回归结果z完全可能是成千上万的数字。事实上e^{100}会变成一个后面有40多个0的超大值，e^{1000}则会直接返回无限大inf，这意味着这些数字已经超出了计算机处理数时要求的有限数据宽度，超大数值无法被计算机运算和表示。这种现象叫做"溢出"，当计算机返回"内存不足"或Python服务器直接断开连接的时候，可能就是发生了这个问题。来看看这个问题实际发生时的状况：

```
#假定一组巨大的z
z = np.array([1010,1000,990])
np.exp(z) / np.sum(np.exp(z)) # softmax函数的运算
```

```
C:\Python\lib\site-packages\ipykernel_launcher.py:3: RuntimeWarning: invalid value encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
array([nan, nan, nan])
```

为了解决这个问题，我们需要对softmax的公式进行如下更改。首先，在分子和分母上都乘上常数C。因为同时对分母和分子乘以相同的常数，所以计算结果不变。然后，把这个C移动到指数函数中，计作e^{logC}，在这里log指的是以自然底数e为底的对数函数。只要C大于0，logC就必然存在，此时我们可以把logC替换为另一个符号C'，它同样代表了一个常数。**即是说，其实我们只需要在softmax函数的两个指数函数自变量上都加上一个常数就可以了，这样做不会改变计算结果。**

$$
\begin{aligned}
o_k &= \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}} \\
&= \frac{Ce^{z_k}}{C\sum_{i=1}^{K} e^{z_i}} \\
&= \frac{e^{logC} \cdot e^{z_k}}{\sum_{i=1}^{K} e^{logC} \cdot e^{z_i}} \\
&= \frac{e^{(z_k+logC)}}{\sum_{i=1}^{K} e^{(z_i+logC)}} \\
&= \frac{e^{(z_k+C')}}{\sum_{i=1}^{K} e^{(z_i+C')}}
\end{aligned}
$$

现在来思考，加入一个怎样的C'才会对抑制计算溢出最有效呢？我们是希望避免十分巨大的z，因此我们只要让C'的符号为负号，在让其大小接近z_k中的最大值就可以了（在sklearn中实际上也是这么做的，所以sklearn中不存在任何可以调节C'的取值的参数）。来看看在代码中我们如何操作：

```
#定义softmax函数
def softmax(z):
    c = np.max(z)
    exp_z = np.exp(z - c) #溢出对策
    sum_exp_z = np.sum(exp_z)
    o = exp_z / sum_exp_z
    return o

#导入刚才定义的z
softmax(z)
```

来看看刚才求出的softmax函数的结果加和之后会显示怎样的效果：

```
sum(softmax(z))
```

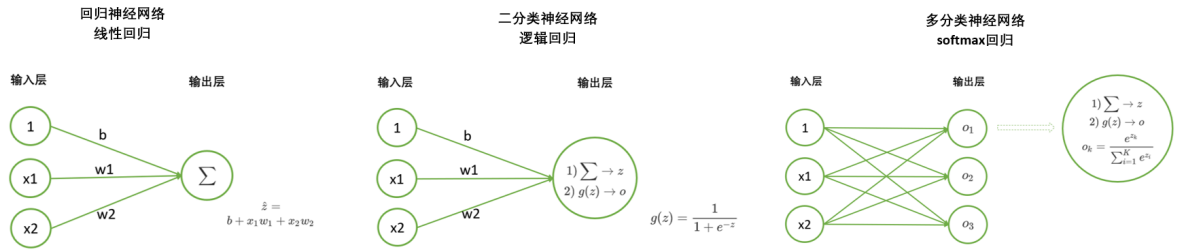
从上面的结果可以看出，softmax函数输出的是从0到1.0之间的实数，而且多个输出值的总和是1。因为有了这个性质，我们可以把softmax函数的输出解释为“概率”，这和我们使用sigmoid函数之后认为函数返回的结果是概率异曲同工。从结果来看，我们可以认为返回了我们设定的z（[1010,1000,990]）的这个样本的结果应该是类别1，因为类别1的概率是最大的（实际上几乎达到了99.9%）。

需要注意的是，使用了softmax函数之后，各个z_k之间的大小关系并不会随之改变，这是因为指数函数e^z是单调递增函数，也就是说，使用softmax之前的z如果比较大，那使用softmax之后返回的概率也依然比较大。这是说，**无论是否使用softmax，我们都可以判断出样本被预测为哪一类，我们只需要看z最大的那一类就可以了**。所以，在神经网络进行分类的时候，如果不需要了解具体分类问题每一类的概率是多少，而只需要知道最终的分类结果，我们可以省略输出层上的softmax函数。

在实际中，训练神经网络时往往会使用softmax函数，但在预测时就不再使用softmax函数，而是直接读取结果最大的z对应的类别了。训练时使用softmax的理由与神经网络的训练过程有关，之后我们会来详细讲到。

2.4 回归vs二分类vs多分类

到这里，我们已经见过了三个不同的神经网络：



注意到有什么相似和不同了吗？

首先可能会注意到的是，这三个神经网络都是单层神经网络，除了输入层，他们都有且只有一层网络。实际上，现实中使用的神经网络几乎99%都是多层的，但我们的网络也能够顺利进行预测，这说明单层神经网络其实已经能够实现基本的预测功能。同时，这也说明了一个问题，无论处理的是回归还是分类，神经网络的处理原理是一致的。实际上，就连算法的限制、优化方法和求解方法也都是一致的。回归和分类神经网络唯一的不同只有输出层上的$g(z)$。

回归看起来并没有$g(z)$的存在，但实际上我们可以认为回归中的$g(z)$是一个恒等函数（identity function），即是说$g(z)=z$（相当于$y=x$，或$f(x)=x$）。而多分类的时候也可以不采用任何函数，只观察z的大小，所以多分类也可以被认为是利用了恒等函数作为$g(z)$。总结来说，回归和分类对应的$g(z)$分别如下：

输出类型	**原理中可使用的g(z)**	**sklearn中的g(z)**
回归	恒等函数	恒等函数
二分类	sigmoid或sign函数（通常都是sigmoid）	sigmoid
多分类	softmax或恒等函数	softmax

第二个很容易发现的现象是，只有多分类的情况在输出层出现了超过一个神经元。实际上，当处理单标签问题时（即只有一个y的问题），回归神经网络和二分类神经网络的输出层永远只有一个神经元，而只有多分类的情况才会让输出层上超过一个神经元。

为了方便计算和演示，在之后的课程中我们都会使用二分类神经网络作为例子，**并且为了课程展示的目的，我们将使用的$g(z)$是阶跃函数。**