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

1.1 打开深度学习的大门

1.2 神经网络的基本原理

1.3 sklearn中的神经网络

2 neural_network.MLPClassifier

2.1 隐藏层与神经元：重要参数hidden_layer_sizes

2.2 激活函数：重要参数activation

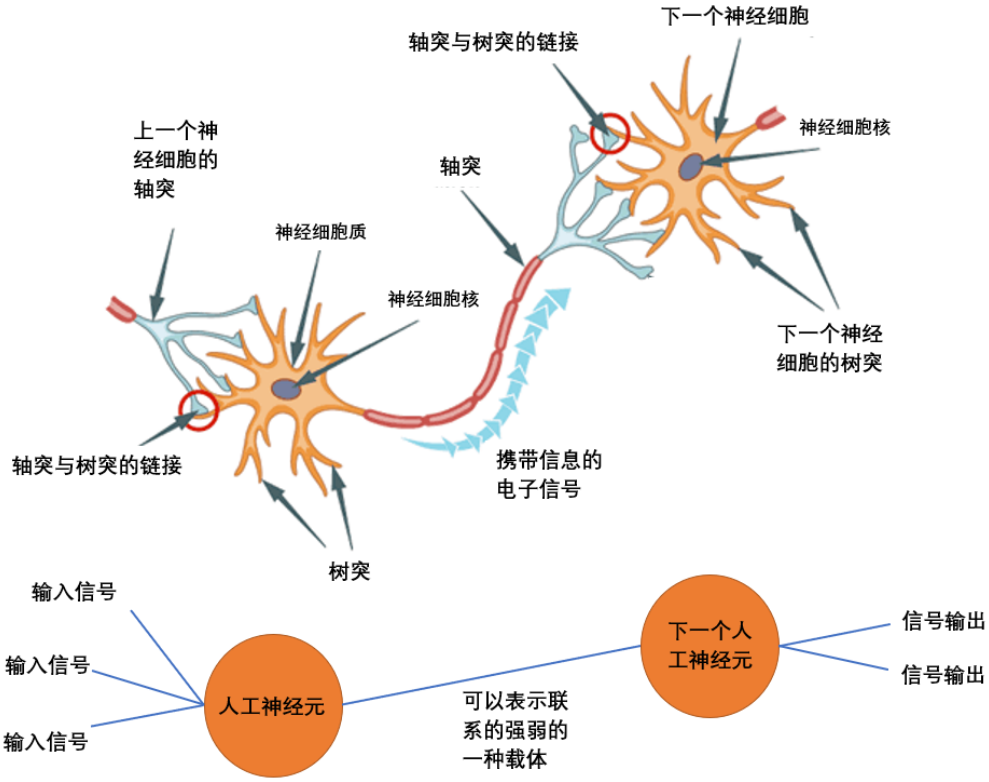
2.3 反向传播与梯度下降

3 三个月来的总结

1 打开深度学习的大门：神经网络概述

1.1 打开深度学习的大门

人工神经网络（Artificial Neural Network，ANN），通常简称为神经网络，是深度学习的基础，它是受到人类大脑结构启发而诞生的一种算法。神经学家们发现，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。每当神经元从其他的神经元接受到信号，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。人类正是通过相同的冲动反复地刺激神经元，改变神经元之间的链接的强度来进行学习。

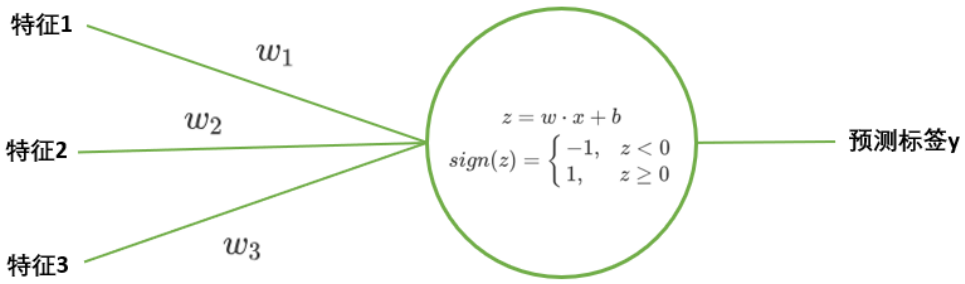


这其实和我们在过去十二周内经历的机器学习过程非常类似。在机器学习中，我们建模，将特征矩阵输入模型中，然后算法为我们输出预测结果。只不过在人脑中，数以亿计的神经细胞相互链接来构建一个生物神经网络（一个神经细胞当然可以和众多个神经细胞相连），我们的机器学习中，往往只有一个模型或者一种算法在运行。人脑通过构建复杂的网络可以进行逻辑，语言，情感的学习，相信模拟这种结构的机器也可以有很强大的学习能力，于是人工神经网络应运而生。

神经网络算法试图模拟生物神经系统的学习过程，以此实现强大的预测性能。不过由于是模仿人类大脑，所以神经网络的模型复杂度很高也是众所周知。在现实应用中，神经网络可以说是解释性最差的模型之一，商业环境中很少使用神经网络。然而出了商业分析，还有许多算法应用的部分，其中最重要的是深度学习和人工智能的领域，现在大部分已经成熟的人工智能技术：图像识别，语音识别等等，背后都是基于神经网络的深度学习算法。因此，作为机器学习中（可能是）最复杂的，深度学习中基础的算法，神经网络的了解和学习是很有必要的。

1.2 神经网络的基本原理

神经网络原理最开始是基于感知机提出——感知机是最古老的机器学习分类算法之一，在1957年就已经被提出了。和今天的大部分模型比较起来，感知机的泛化能力比较弱，但支持向量机和神经网络都基于它的原理来建立。感知机的原理就是我们在支持向量机中详细介绍过的，使用一条线性决策边界$z = w \cdot x + b$来划分数据集，决策边界的上方是一类数据（$z \geq 0$），决策边界的下方是另一类数据（$z > 0$）的决策过程。使用神经元，我们可以把它表示为：

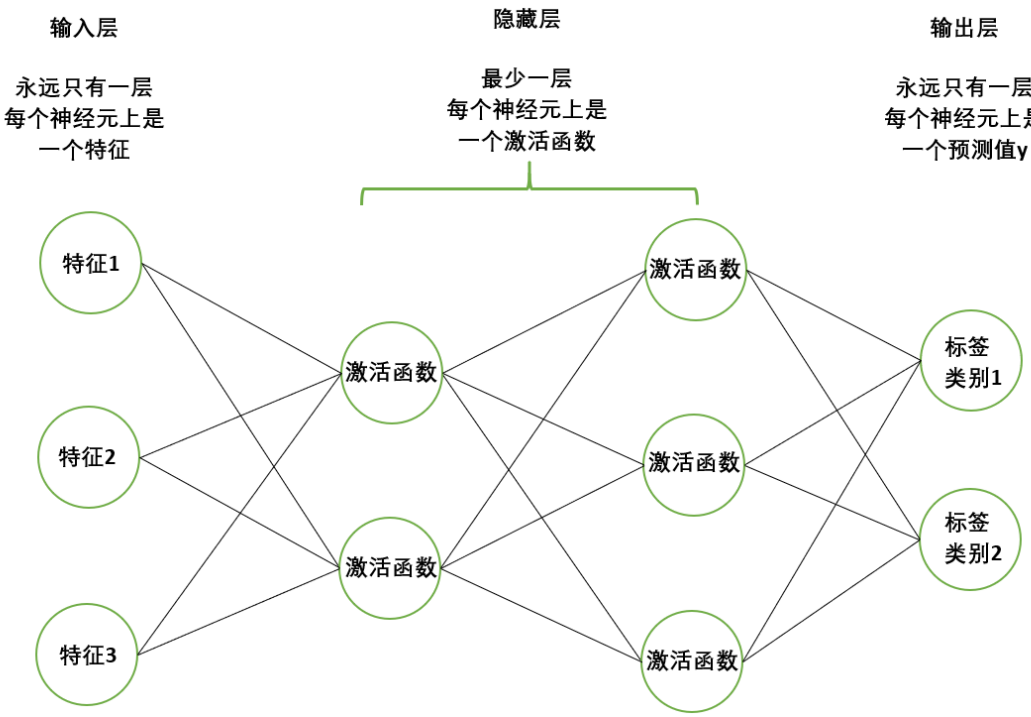


不同的特征数据被输入后，我们通过神经键将它输入我们的神经元。每条神经键上对应不同的参数w, b，因此特征数据会经由神经键被匹配到一个参数向量w, b，基于参数向量w算法可以求解出决策边界$z = w \cdot x + b$，然后由决策函数$sign(z)$进行判断，最终预测出标签y并且将结果输出。其中函数$sign(z)$被称为“激活函数”，这是模拟人类的大脑激活神经元的过程所命名的，其实本质就是决定了预测标签的输出会是什么内容的预测函数。

注意，在这个过程中，有两个非常重要的核心要点：

1. **每个输入的特征会被匹配到一个参数**w，我们都知道参数向量$\boldsymbol{w}$中含有的参数数量与我们的特征数目是一致的，在感知机中也是如此。也就是说，任何基于感知机的算法，必须至少要有参数向量$\boldsymbol{w}$可求。
2. **一个线性关系**z，z是由参数和输入的数据共同决定的。这个线性关系，往往就是我们的决策边界，或者它也可以是多元线性回归，逻辑回归等算法的线性表达式
3. 激活函数的结果，是基于激活函数本身，参数向量w和输入的数据一同计算出来的。也就是说，**任何基于感知机的算法，必须要存在一个激活函数。**

神经网络就相当于众多感知机的集成，因此，**确定激活函数，并找出参数向量**w**也是神经网络的计算核心。**只不过对于只运行一次激活函数的感知机来说，神经网络大大增加了的模型的复杂度，激活函数在这个过程中可能被激活非常多次，参数向量的数量也呈指数级增长。我们来看看神经网络的基本结构：



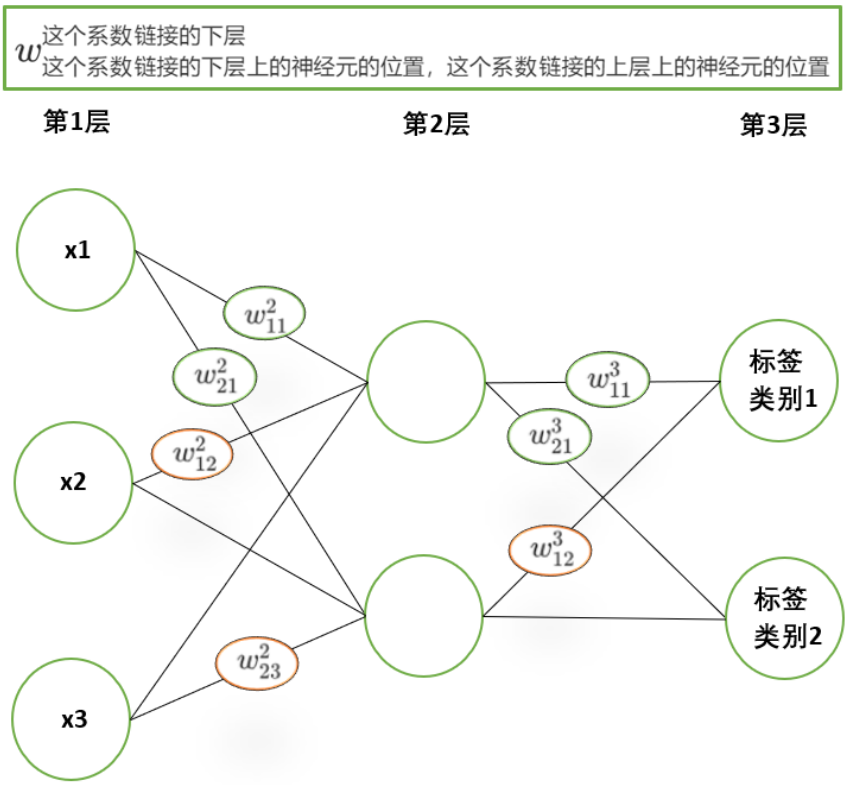
首先，神经网络有三层。第一层叫做**输入层**（Input layer），输入特征矩阵用，因此每个神经元上都是一个特征向量。极端情况下，如果一个神经网络只训练一个样本，则每个输入层的神经元上都是这个样本的一个特征取值。

最后一层叫做**输出层**（output layer），输出预测标签用。如果是回归类，一般输出层只有一个神经元，回归的是所有输入的样本的标签向量。如果是分类，可能会有多个神经元。二分类有两个神经元，多分类有多个神经元，分别输出所有输入的样本对应的每个标签分类下的概率。但无论如何，输出层只有一层，是用于输出预测结果用。

输入层和输出层中间的所有层，叫做**隐藏层**（Hidden layers），最少一层。**也就是说整个神经网络是最少三层。**隐藏层是我们用来让算法学习的网络层级，从更靠近输入层的地方开始叫做"上层"，相对而言，更靠近输出层的一层，叫做"下层"。在隐藏层上，每个神经元中都存在一个激活函数，我们的数据被逐层传递，**每个下层的神经元都必须处理上层的神经元中的激活函数处理完毕的数据**，本质是一个感知器嵌套的过程。隐藏层中上层的每个神经元，都与下层中的每个神经元相连，因此隐藏层的结构随着神经元的变多可以变得非常非常复杂。神经网络的两个要点：参数w和激活函数，也都在这一层发挥作用，因此理解隐藏层是神经网络原理中最难的部分。

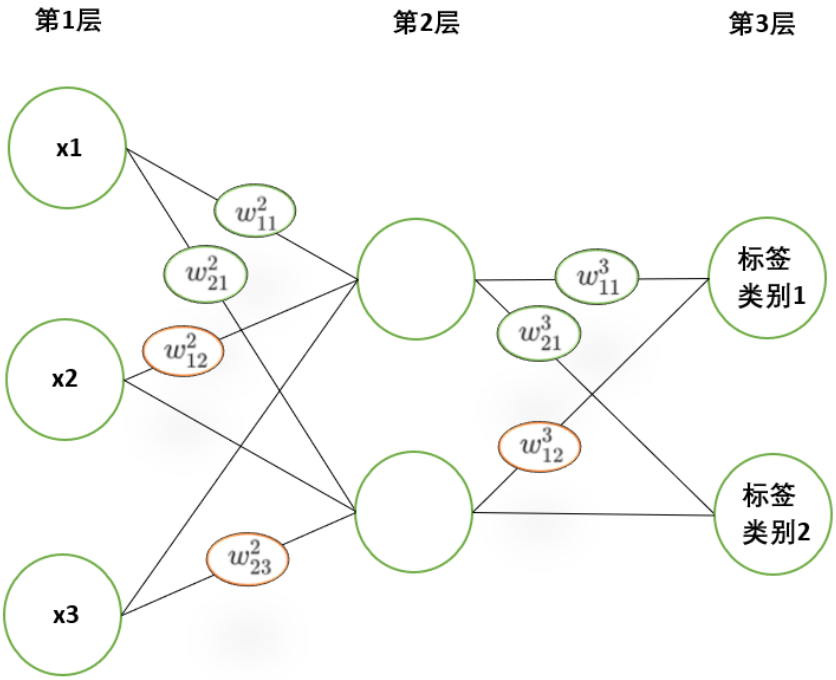
- 参数w

我们接下来，就来举一个非常简单的例子，为大家梳理一下神经网络的过程。假设我们的线性关系$z = \boldsymbol{w} \cdot \boldsymbol{x}$，我们将截距$b$包括在了$\boldsymbol{w}$当中。我们现在有一个最简单的三层神经网络中，带大家来了解一下我们的参数w。感知机上每条神经键上都有一个参数w，对神经网络也是如此，只不过神经网络中神经元众多，每个神经元又和相邻层的神经元全部相连，因此参数w的数量也众多。来看下面的三层神经网络，我们总共有3*2*2条神经键，因此总共有12个参数w。这12个参数w的表示法非常复杂，具体如下：



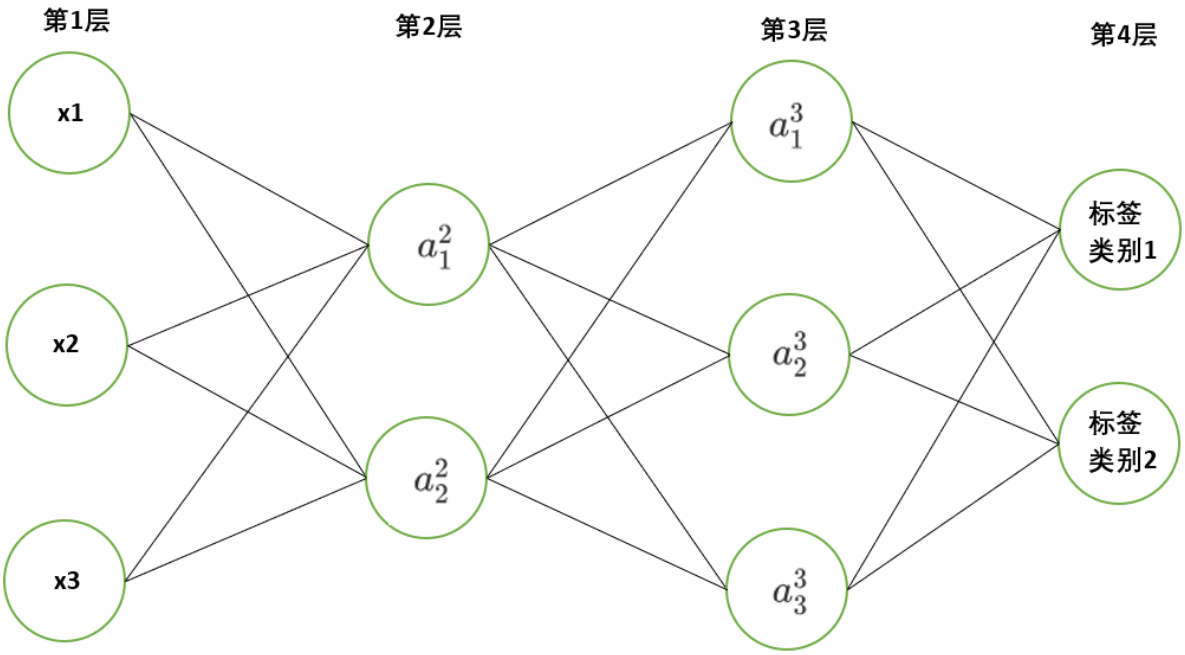
现在请判断一下，我写作w_{12}^3，w_{23}^2，w_{12}^2，w_{31}^2分别是哪三条神经键上的参数呢？

来看正确答案：



而w_{31}^2是不存在的，因为第二层根本就没有第三个神经元。有了这些神经元的编号，我们就可以来演示一下神经网络上一组数据是如何在网络中变化，最终变化成标签的了。为了解释隐藏层之间的关系，我们现在将神经网络增加到四层，并假设我们的激活函数就是$\sigma(z)$，别忘记我们的$z = \boldsymbol{w} \cdot \boldsymbol{x}$，其实也就是：

$$z = w_1 x_1 + w_2 x_2 + \ldots w_n x_n$$



看如上的神经网络，我们有三个输入的特征x_1, x_2, x_3，这三个特征首先要被匹配到自己所对应的w，然后放入第二层的神经元中去求出线性函数的取值z和激活函数的取值a。对于a和z来说，上标表示所在的层数，下表表示这个层数上所在的神经元的位置，比如说a_2^3就是第三层的从上往下数的第二个神经元中的激活函数的取值。

注意，这一层的每个神经元上会有几个参数对应，是由上层有几个神经元确定的。那对于现在我们的第二层来说，上层有三个神经元，因此本层的每个神经元会对应三个x，即三个w，则我们有：

$$a_1^2 = \sigma(z_1^2) = \sigma(w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$$
$$a_2^2 = \sigma(z_2^2) = \sigma(w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$$

则对于第三层来说，被输入第三层的就不再是三个x了，而是我们经过第二层计算出的a_1^2和a_2^2了。第三层我们的每个输入结构都会对应两个w，因此在第三层，我们有：

$$a_1^3 = \sigma(z_1^3) = \sigma(w_{11}^3 a_1^2 + w_{12}^3 a_2^2)$$
$$a_2^3 = \sigma(z_2^3) = \sigma(w_{21}^3 a_1^2 + w_{22}^3 a_2^2)$$
$$a_3^3 = \sigma(z_3^3) = \sigma(w_{31}^3 a_1^2 + w_{32}^3 a_2^2)$$

对于最后一层，就是我们的输出层来说，我们的上一层是有三个神经元，则这一层的每个神经元中就要对应三个w，和之前的过程一样，我们可以有：

$$a_1^4 = \sigma(z_1^4) = \sigma(w_{11}^4 a_1^3 + w_{12}^4 a_2^3 + w_{13}^4 a_3^3)$$
$$a_2^4 = \sigma(z_2^4) = \sigma(w_{21}^4 a_1^3 + w_{22}^4 a_2^3 + w_{23}^4 a_3^3)$$

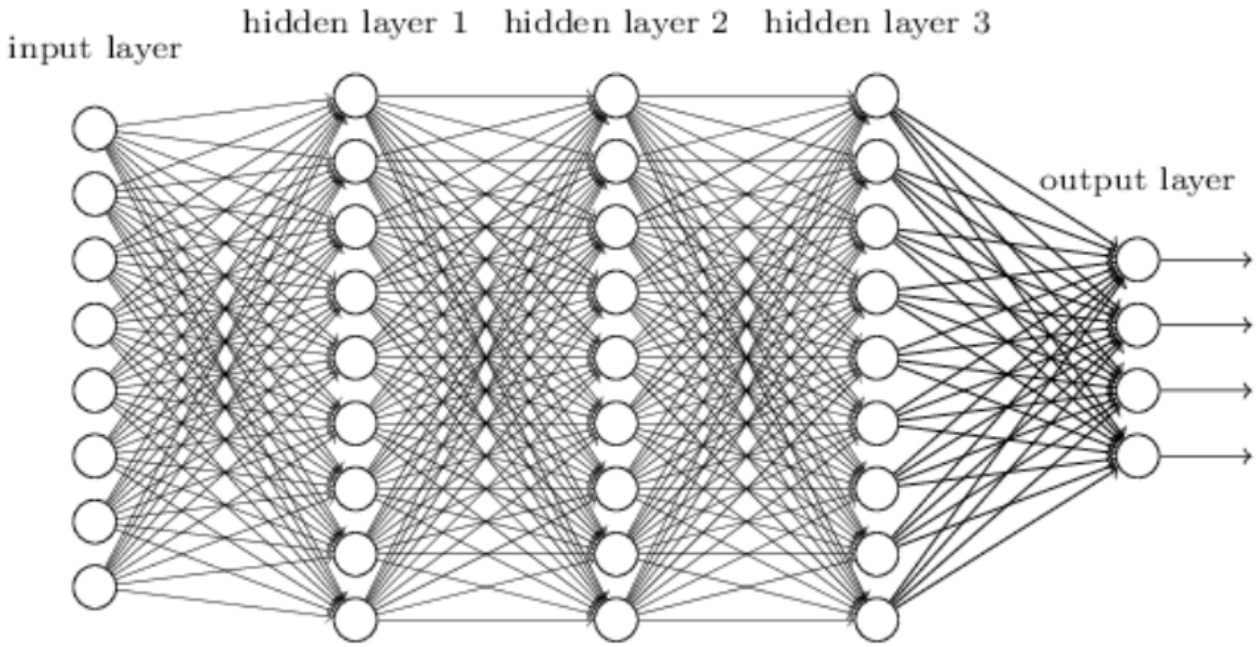
如果我们现在执行的是分类算法，我们的a_1^4和a_2^4就是我们的二分类下每个标签类别对应的概率，我们会取两个a中数值较大的那一个，作为我们的分类结果。这个计算过程，其实就是神经网络的基本原理。在这个过程中，有两个非常容易混淆的地方不得不给大家提到：

1. **我们神经网络的每一层的结果之间的关系是嵌套，不是迭代。**

   回忆一下我们在逻辑回归中如何迭代参数？我们执行梯度下降，每下降一步，上一步的参数就被下一步的参数所替换掉，最终我们使用来求解预测结果y的只是我们最后一次迭代出的参数组合。但在我们的神经网络中，上一层的结果和参数，会被放到下一层中去求解新的结果，但是上一层的参数还会被保留，没有被覆盖每次求解都还需要执行整个嵌套过程，需要每一层上的每个参数。

2. 由于我们执行的是嵌套，不是迭代，所以**我们的每一个系数之间是相互独立的，每一层的系数之间也是相互独立的**，我们不是在执行使用上一层或者上一个神经元的参数来求解下一层或者下一个神经元的参数的过程。我们不断求解的，是激活函数的结果a，不是参数w，**重要的事情说三遍，在一次神经网络计算中，我们没有迭代参数w，没有迭代参数w，没有迭代参数w。**

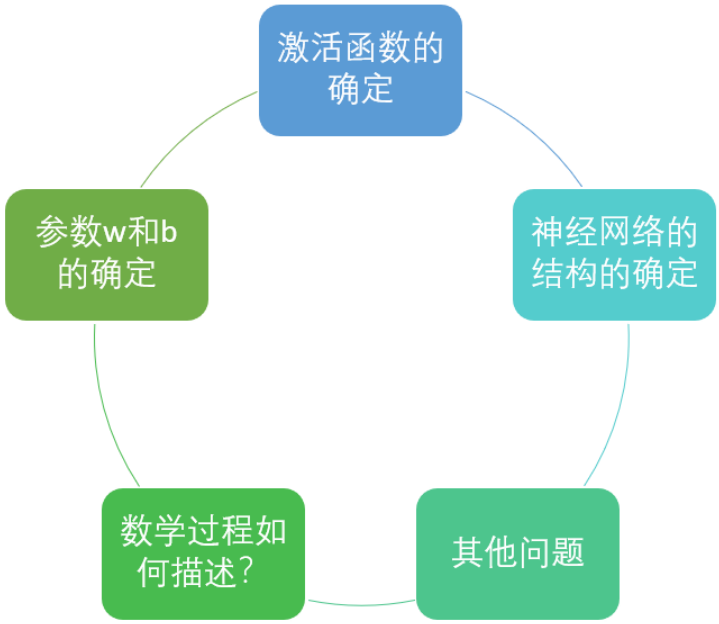
了解了这个过程，其实大家就知道神经网络是如何工作的了。大家可以看到，光是一个四层的神经网络，我们对于w的标号就已经有点一团乱麻的感觉了。试想一下我们的真实数据中，往往有多少特征需要在输入层被输入，那我们需要多少层，多少个神经元去输出我们的预测值呢？真实的神经网络，可能很多都长这样：



这张图还不算真实数据中特别复杂的情况，但已经含有总共 8*9*9*9*4 = 23,328个w，对于手算或者自行理解来说，已经是不可能了，更别说存在高维特征，比如说100，1000个特征的情况了。现实中神经网络常常被用于处理大型数据，所以最后神经网络究竟是怎样产出了我们的y，中间发生了这样的过程，产出了多少个系数，完全就是一个“黑箱”过程了——我们是无法理解的。

从上面的讲解来看，神经网络的计算本身其实并不晦涩，它所涉及到的概念：比如感知机，比如嵌套，比如之后我们要讲解的梯度下降，都是我们曾经在其他算法中见过的过程，它难在计算过程过于复杂，元素太多，不在原理本身的数学水平或者机器学习水平。

到这里，你就算是理解了神经网络的.....开头！我们仅仅讲解了神经网络的冰山一角。深度学习中的众多算法，都是基于我们刚才讲解的最简单的神经网络进行的一些变化和改进。基于这个简单的原理，我们其实已经可以预见我们神经网络的重点，以及我们的sklearn中都会有什么样的参数来供我们调整了：



1. 激活函数：就如同核函数一样，必然有各种各样的激活函数可以供我们选择
2. 神经网络的结构：隐藏层有多少，每层有多少神经元？必然是一个超参数，需要我们自己来进行调整

3. w怎么确定：之前一直说得非常理所当然，每个神经键上都有一个参数w，然而这些参数从哪里来的呢？在SVM中和逻辑回归中，我们写了十页数学计算来求解我们的w，在神经网络中，求解参数必然也不会轻松。是不是要最优化呢？如果要最优化的话，是不是有损失函数呢？有损失函数的话，是不是有梯度下降呢？如果我们使用的是类似于梯度下降，最小二乘这样的数学过程，则必然有众多的数学过程相关的参数在sklearn中等待着我们。
4. 除此之外，我们还想知道一些其他的问题：比如说，神经网络为什么要使用嵌套的方式进行？为什么需要多层，一层不行吗？

在今天的课中，我就将带着大家去一一解决这些问题。神经网络可能是我们遇到的最难调参的算法没有之一。接下来我们就来看看sklearn中的神经网络吧。

1.3 sklearn中的神经网络

神经网络对于机器学习来说，尤其是对于业务分析的机器学习来说，可以说是位置尴尬。虽然它效果很好，但它的可解释性太差，别说完全不理解机器学习的老板和同事们，就连把机器学习学了个遍，听了十二周课来到这里的你，可能都无法理解神经网络中究竟发什么了什么。再加上神经网络的学习能力超强，所以它是一个非常容易过拟合的模型，很多时候也许都不是一个好的选择。所以神经网络其实并不如大家想象得那么受欢迎，尤其是在业务分析的领域。当然，在深度学习的领域是完全另一番风景，老板也不会去问你，为什么你研究的程序能够识别出图像。

sklearn是专注于机器学习的库，它在神经网络的模块中特地标注：sklearn不是用于深度学习的平台，因此这个神经网络不具备做深度学习的功能，也不具备处理大型数据的能力，所以神经网络在sklearn中颇有被冷落的意思。原理讲解也非常简单，并没有详细的描述。但是使用神经网络的类还是有很多参数，写法详细。

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。

今天我们的重点是带大家来了解一下两个以多层感知机为基础的类：MLPClassifier和MLPRegressor。

2 neural_network.MLPClassifier

class `sklearn.neural_network.MLPClassifier`(*hidden_layer_sizes=(100,)*, *activation='relu'*, *solver='adam'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*, *n_iter_no_change=10*)

2.1 隐藏层与神经元：重要参数hidden_layer_sizes

神经网络算法中要考虑的第一件事情就是我们的隐藏层的结构，如果不设定结构，神经网络本身甚至无法构建，因此这是一个超参数。

参数	含义
hidden_layer_sizes	元组，长度= n_layers - 2，默认值(100,) 元祖中包含多少个元素，就表示设定多少隐藏层 元祖中的第i个元素表示第i个隐藏层中的神经元数量

先来建立一个神经网络吧。

1. **导入需要的数据和库，导入数据集**

```
import numpy as np
from sklearn.neural_network import MLPClassifier as DNN
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score as cv
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.model_selection import train_test_split as TTS
from time import time
import datetime

data = load_breast_cancer()
X = data.data
y = data.target

Xtrain, Xtest, Ytrain, Ytest = TTS(X,y,test_size=0.3,random_state=420)
```

2. **建模，使用交叉验证导出分数**

```
times = time()
dnn = DNN(hidden_layer_sizes=(100,),random_state=420)
print(cv(dnn,X,y,cv=5).mean())
print(time() - times)

#使用决策树进行一个对比
times = time()
clf = DTC(random_state=420)
print(cv(clf,X,y,cv=5).mean())
print(time() - times)
```

3. **查看如何使用参数hidden_layer_sizes**

```
dnn = DNN(hidden_layer_sizes=(100,),random_state=420).fit(Xtrain,Ytrain)
dnn.score(Xtest,Ytest)

#使用重要参数n_layers_
dnn.n_layers_
#可见，默认层数是三层，由于必须要有输入和输出层，所以默认其实就只有一层隐藏层

#如果增加一个隐藏层上的神经元个数，会发生什么呢?
dnn = DNN(hidden_layer_sizes=(200,),random_state=420)
```

```
dnn = dnn.fit(Xtrain,Ytrain)
dnn.score(Xtest,Ytest)
#看似结果会

#来试试看学习曲线
s = []
for i in range(100,2000,100):
    dnn = DNN(hidden_layer_sizes=(int(i),),random_state=420).fit(Xtrain,Ytrain)
    s.append(dnn.score(Xtest,Ytest))
print(i,max(s))
plt.figure(figsize=(20,5))
plt.plot(range(200,2000,100),s)
plt.show()

#那如果增加隐藏层，控制神经元个数，会发生什么呢?
s = []

layers = [(100,),(100,100),(100,100,100),(100,100,100,100),(100,100,100,100,100),
(100,100,100,100,100,100)]

for i in layers:
    dnn = DNN(hidden_layer_sizes=(i),random_state=420).fit(Xtrain,Ytrain)
    s.append(dnn.score(Xtest,Ytest))
print(i,max(s))
plt.figure(figsize=(20,5))
plt.plot(range(3,9),s)
plt.xticks([3,4,5,6,7,8])
plt.xlabel("Total number of layers")
plt.show()

#如果同时增加隐藏层和神经元个数，会发生什么呢?
s = []

layers = [(100,),(150,150),(200,200,200),(300,300,300,300)]

for i in layers:
    dnn = DNN(hidden_layer_sizes=(i),random_state=420).fit(Xtrain,Ytrain)
    s.append(dnn.score(Xtest,Ytest))
print(i,max(s))
plt.figure(figsize=(20,5))
plt.plot(range(3,7),s)
plt.xticks([3,4,5,6])
plt.xlabel("Total number of layers")
plt.show()
```

我们不可能无尽地画学习曲线画下去，并且，无论是什么样的学习曲线，看起来数据上的表现都没有明确的提升或下降，可见这不是一种有效的方法。有没有既定的规则可以帮助我们确定隐藏层到底需要多少层，而每个隐藏层上到底需要多少个神经元呢?