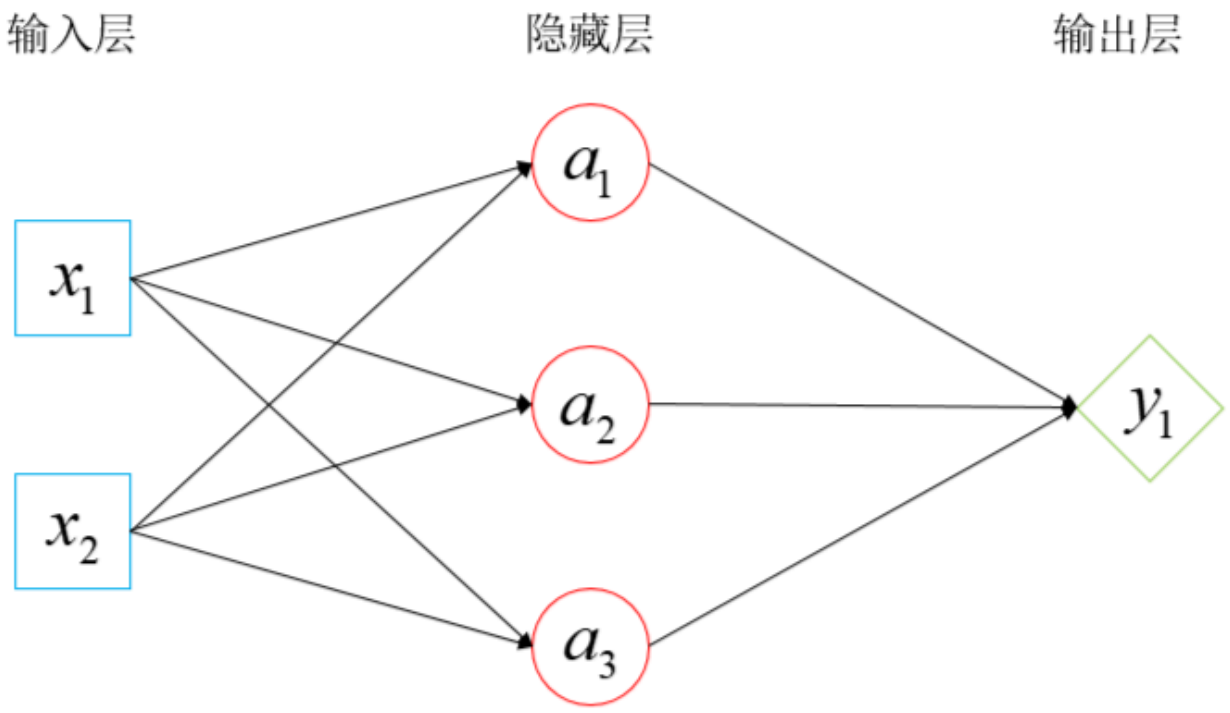
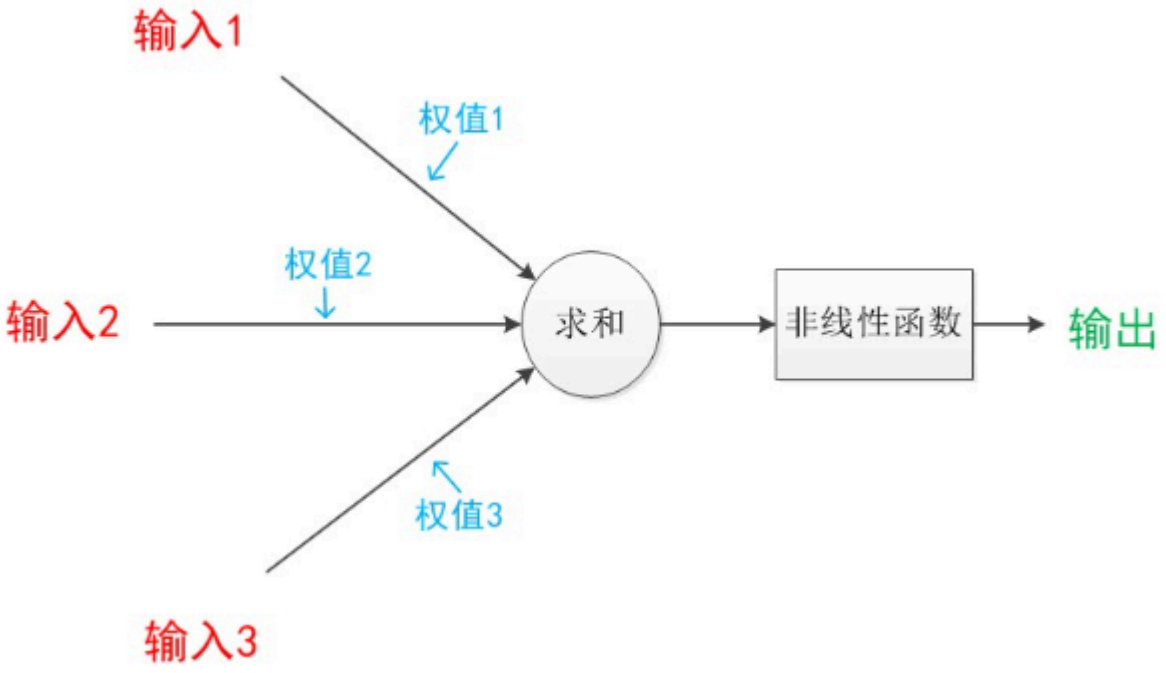
全连接神经网络

一、全连接神经网络结构

全连接神经网络的整体结构如下所示：其包括输入层、隐藏层和输出层，其特点是**每一层的每一个节点都与上下层节点全部连接**，其中隐藏层可以有很多层。



对于其中的每一个节点（即单独的一个神经元结构）如下图所示：



这个结构实际上就是前面章节的逻辑回归模型结构，求和后需要经过一个激活函数，**如果网络中没有使用激活函数，每一层的节点的输入都是上层输出的线性函数，无论神经网络中的隐含层有多少，最后的输出结果都是网络输入的线性拟合，即隐含层没有发挥其作用。**

二、常见的激活函数特点

1.sigmoid函数（S型函数）

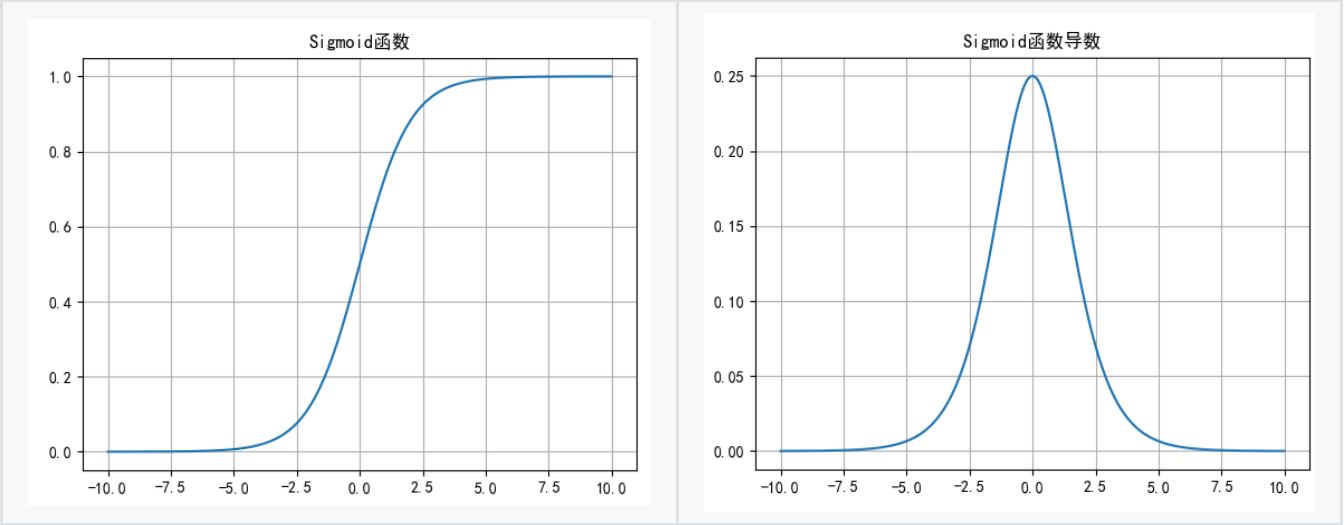
sigmoid函数的定义如下：

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

由于在使用梯度下降法求解最优参数时会对损失函数求导，而损失函数中会包含激活函数，因此需要研究激活函数的导数形式：

$$g^{'} = g(1-g) \tag{2}$$

其函数图像和导数图像如下所示：



观察图像可以看出，其输出在[0-1]之间，非常适合**二分类任务**；但是其导数值域在[0-0.25],因此**会出现梯度消失的问题**（隐藏层数过多时，参数更新过程中导数叠乘导致值趋于0，参数不再更新），同时由于其输出关于0不对称，会导致参数更新速度较慢。

2.Tanh函数（双曲正切函数）

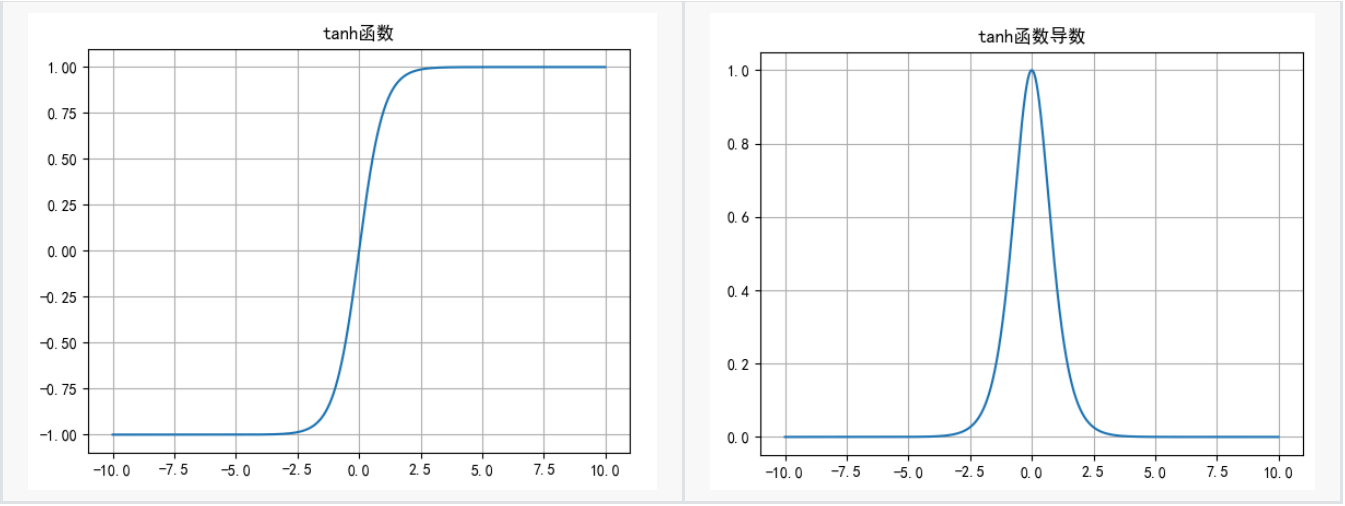
Tanh函数的定义如下：

$$g(Z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \tag{3}$$

导数公式如下：

$$g^{'} = 1-g^{2} \tag{4}$$

其函数图像和导数图像如下所示：



其与sigmoid函数非常相似，**输出范围在[-1,1]关于0对称，训练时收敛速度更快**，其同样存在梯度消失的问题。

3.ReLU函数

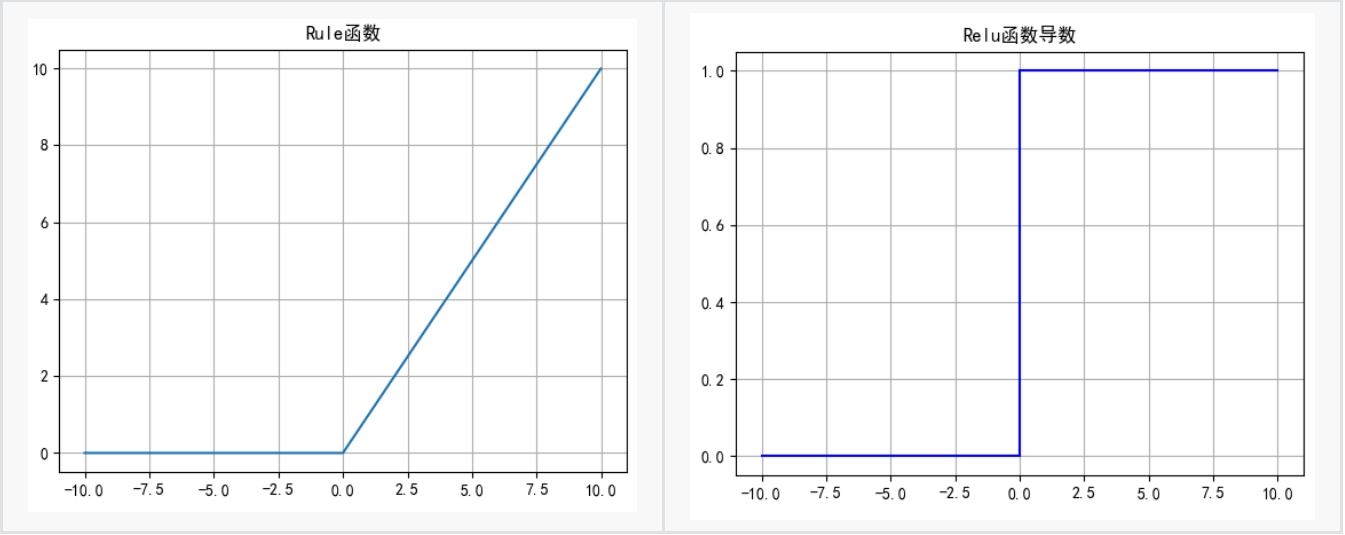
Tanh函数的定义如下：

$$g(Z) = \begin{cases} Z & if \quad Z > 0 \\ 0 & if \quad Z <= 0 \end{cases} \tag{5}$$

导数公式如下：

$$g^{'} = \begin{cases} 1 & if \quad Z > 0 \\ 0 & if \quad Z <= 0 \end{cases} \tag{6}$$

其函数图像和导数图像如下所示：



其解决了梯度消失的问题，但是**当输入为负时，梯度为0。这个神经元及之后的神经元梯度永远为0，不再对任何数据有所响应，导致相应参数永远不会被更新**，即可能出现神经元死亡。

4.Softmax函数

Softmax函数的定义如下：

$$g_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{7}$$

导数公式如下：

$$g_i^{'} = g_i(1 - g_i) \tag{8}$$

Softmax激活函数主要针**对多分类问题**，其中N为分类的个数，g_i为属于对应类别的概率输出，Softmax函数满足每个输出值在[0-1],所有输出值和为1。

三、前向传播和反向传播

1.前向传播

神经网络中的前向传播实际上就是**通过输入的特征X和设定的参数w计算出预测输出Y值的过程。**

2.反向传播

神经网络中的反向传播实际上就是**利用前向传播计算出的值与真实值之间误差进行梯度更新参数的过程。**

进行具体的数学计算时只需要按照求导的链式法则，依次对每一层进行损失函数的求导即可。链式法则如下所示:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{9}$$

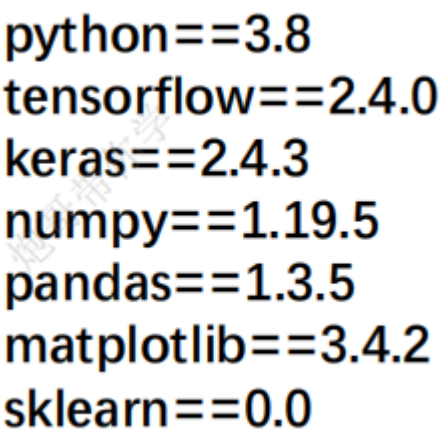
三、代码实践问题

1.Numpy和TensorFlow版本冲突：

TensorFlow ≤2.10 时配合 **NumPy ≥1.24** 会出充图问题，具体报错如下所示：

```
E:\Software\anaconda\envs\keras\lib\site-packages\tensorflow\python\framework\dtypes.py:513:
FutureWarning: In the future `np.object` will be defined as the corresponding NumPy scalar.
  np.object,
```

解决方法：使用视频中指定的版本



卸载安装对应版本numpy

```
pip uninstall numpy -y
pip install numpy==1.19.5
```

2.预测空气质量中数据归一化问题

整个归一化的流程如下所示：

1.在训练模型时将整个dataframe的数据进行归一化（包括特征X和目标值Y），然后进行数据的提取和划分。

对Y也要归一化的原因是：这里的预测空气质量是回归任务，对于前面的分类任务，由于目标会使用one-hot编码，每个标签的值没有实际量纲意义，因此不进行归一化也不会影响训练，但是**回归任务如果不归一化就会导致数值偏大的样本对训练的影响过大**，因此回归任务的目标Y也需要归一化。

2.在使用模型时也是先将整个dataframe的数据进行归一化，然后用这个实例对特征进行归一化，输出的预测值再进行反归一化。

这里有一个疑问就是：**实际进行模型使用时是输入一组特阵值预测一个目标值，那这个特征怎么归一化的呢**?

实际上看归一化内部的代码可以看到，对于视频中调用的MinMaxScaler这个包，实际上就是下面公式的一个归一化(假设归一化到[0-1])方式：

$$x_i^{'} = \frac{1-0}{max(X)-min(X)} * x_i + (0 - min(X)) \tag{10}$$

代码中将$\frac{1-0}{max(X)-min(X)}$定义为scale（缩放比例），将$0 - min(X)$定义为min（偏移量），因此就得到计算公式：

$$x_i^{'} = scale * x_i + min \tag{11}$$

关键计算代码如下：

```
data_min = np.nanmin(X, axis=0)
data_max = np.nanmax(X, axis=0)

if first_pass:
    self.n_samples_seen_ = X.shape[0]
else:
    data_min = np.minimum(self.data_min_, data_min)
    data_max = np.maximum(self.data_max_, data_max)
    self.n_samples_seen_ += X.shape[0]

data_range = data_max - data_min
self.scale_ = (feature_range[1] - feature_range[0]) / _handle_zeros_in_scale(
    data_range, copy=True
)
self.min_ = feature_range[0] - data_min * self.scale_
self.data_min_ = data_min
self.data_max_ = data_max
self.data_range_ = data_range
```

所以实际进行模型预测时，尽管只有一组数据，但是由于使用的是之前实例化的对象，因此其scale的值和min的值是传递过来了的，也就是**实际上使用的是训练数据的缩放尺度和偏移量。**

具体的关键使用代码如下（sc就实例化的那个对象）：

```
# 将数据进行归一化
sc = MinMaxScaler(feature_range=(0, 1))
scaled = sc.fit_transform(dataset)
```

```
# 进行预测值的反归一化
inv_yhat = concatenate((x_test, yhat), axis=1)
inv_yhat = sc.inverse_transform(inv_yhat)
```

3.视频中的归一化是对整个数据进行fit_transform归一化的，这里存在一定疑惑，这样测试集的尺度信息也包含进来了，或许应该对x_train进行fit_transform，对x_test只进行transform（缩放），下面是调整后的关键代码：

```
# 分别归一化
x_scaler = MinMaxScaler(feature_range=(0, 1))
y_scaler = MinMaxScaler(feature_range=(0, 1))
x_train = x_scaler.fit_transform(x_train)
x_test = x_scaler.transform(x_test)
y_train = y_scaler.fit_transform(y_train.values.reshape(-1,1))
y_test = y_scaler.transform(y_test.values.reshape(-1,1))
```

```
# 预测值反归一化
y_pred = concatenate((x_test, y_pred), axis=1)
y_pred = y_scaler.inverse_transform(y_pred)
y_pred = y_pred[:, -1]
print("反归一化后的预测值:", y_pred)
```