神经网络和机器学习之广告预测

案例说明：Advertising（广告预测），使用全连接神经网络层。

案例选择了keras框架，需要先安装keras和tensorflow。虚谷号教育版已经预装必要的库，可以直接使用。

本案例已经提供了训练好的模型，放在 `model` 文件夹中，文件名称为： `1-model-vv.h5` 。如果想直接测试模型，请跳到“导入模型”或者“应用模型”环节，输入数据开始识别。

1.环境搭建

下面是安装命令：

pip install keras

pip install -i https://pypi.tuna.tsinghua.edu.cn/simple (https://pypi.tuna.tsinghua.edu.cn/simple) tensorflow

建议选择清华源，速度将快很多。参考命令如下：

pip install -i https://pypi.tuna.tsinghua.edu.cn/simple (https://pypi.tuna.tsinghua.edu.cn/simple) tensorflow

2.数据说明

企业为了提高产品销售额，往往会在各种媒体投入资金展示自己的广告，常见的媒体有电视、广播以及报纸。作为决策者，需要确定在不同媒体的最佳广告投入，以期待最好的产品收益。为了寻求各媒体广告投入与收益之间的关系，还收集到了之前两百个不同产品在各个媒体中的广告投入资金与最终的销售业绩数据，这些数据被制成表格保存在“Advertising.csv”文件中。

数据总共有5列，第1列表示数据的行号，没有列名；第2列到第4列分别是在电视、广播、报纸上的广告投入，最后一列则是对应的销售额。我们让机器学习这一组数据，然后再输入新的数据，让机器来预测可能的销售额。

开始导入数据集吧，数据文件在 `data` 文件夹中。

`Advertising.csv` 文件是以纯文本形式存储表格数据（数字和文本）,文件一共201行，第一行为列名。文本内容如下：

,TV,radio,newspaper,sales

1,230.1,37.8,69.2,22.1

2,44.5,39.3,45.1,10.4

3,17.2,45.9,69.3,9.3

……

In [1]:

```
import pandas as pd
data=pd.read_csv('./data/Advertising.csv')
```

用pandas的read_csv功能，读入csv文件中的数据。data是pandas中的DataFrame对象，是一个二维数组。head

和 tail 方法可以分别查看最前面几行和最后面几行的数据（默认为5）。

In [2]:

```
data.head()
```

Out[2]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [3]:

```
data.tail()
```

Out[3]:

	Unnamed: 0	TV	radio	newspaper	sales
195	196	38.2	3.7	13.8	7.6
196	197	94.2	4.9	8.1	9.7
197	198	177.0	9.3	6.4	12.8
198	199	283.6	42.0	66.2	25.5
199	200	232.1	8.6	8.7	13.4

接下来用pandas的iloc，对数据进行切片。 `data.iloc[: , 1:-1]` 表示x等于，y=data.iloc[: , -1]

In [4]:

```
x=data.iloc[: , 1:-1]
y=data.iloc[: , -1]
```

现在，x与y的形状分别是(200,3)和(200,1)，即x具有200行、3列，y具有200行、1列。其中x是输入的数据(电视、广播、报纸上的广告投入)，y是输出的结果(销售额)。可以输出其中某一行看看。

In [5]:

```
print(x.values[3])
print(y.values[3])
```

```
[151.5  41.3  58.5]
18.5
```

为了让模型可以更好的用于对未知数据的预测，一般会把已有的数据分割成三个数据集，分别是训练集、验证集、测试集。

对于这里的200组数据，可以将其先后顺序随机打乱，然后取前160组作为训练集数据，之后的20组为验证集数据，最后的20组为测试集数据。

In [6]:

```
data=data.sample(frac=1).reset_index(drop=True)    #打乱数据的先后顺序
x=data.iloc[:,1:-1]
y=data.iloc[:,-1]
x_train,y_train=x[:160],y[:160]
x_val,y_val=x[160:180],y[160:180]
x_test,y_test=x[180:],y[180:]
```

3.建立模型

In [7]:

```
import keras
from keras import layers
```

```
Using TensorFlow backend.
```

为了调用方便，直接导入了keras的layers子集。keras支持建立序惯模型与函数式模型，在一般情况下，建立一个序贯模型就可以了。接着，为模型添加层，keras支持很多类型的神经网络层，这里使用add方法添加2个全连接神经网络层（Dense层）。

第一层通过input_dim参数指定接收输入数据的维度为3（电视、广播、报纸3列），units=32表示将这个3维数据全连接到32个神经元，并通过relu激活函数进行激活，当然，这一层中的神经元个数并不一定需要设置为32个，但是较多的神经元个数使得模型具有更强的拟合能力；从第二层开始，输入数据维度默认为前一层的输出维度，因此不再需要指定输入数据的维度，只需要指定神经元个数即可，在上述代码中，第一层的32维输出再次全连接到第二层的1个神经元中，最后这1个神经元的输出就是模型的预测结果了。

代码如下：

In [8]:

```
model=keras.models.Sequential()
model.add(layers.Dense(units=32,input_dim=3,activation='relu'))
model.add(layers.Dense(units=1))
```

定义好模型的层之后，需要对模型进行编译，同时指定训练模型所需要的优化器以及损失的估算方法。在keras中，可以通过optimizer参数来指定优化器。经验证明，adam优化器具有非常良好的表现。loss='mse'表示使用均方误差（mse）作为损失函数。

In [9]:

```
# 编译模型
model.compile(optimizer='adam',loss='mse')
```

最后对模型进行训练，一下代码利用现有数据x和y对模型进行训练1000次，epochs表示训练轮次，batch_size表示每次有多少行数据参与训练，最后把整个训练过程记录到history中。程序运行后，在控制台会打印出每轮次的训练情况。

In [10]:

```
history=model.fit(x_train,y_train,batch_size=80,epochs=1000,validation_data=(x_val,y
```

```
Epoch 20/1000
160/160 [==============================] - 0s 215us/step - loss: 326.
1151 - val_loss: 298.0489
Epoch 21/1000
160/160 [==============================] - 0s 181us/step - loss: 308.
0205 - val_loss: 279.4639
Epoch 22/1000
160/160 [==============================] - 0s 249us/step - loss: 295.
4550 - val_loss: 264.3066
Epoch 23/1000
160/160 [==============================] - 0s 215us/step - loss: 282.
4314 - val_loss: 252.0886
Epoch 24/1000
160/160 [==============================] - 0s 193us/step - loss: 272.
6535 - val_loss: 242.1785
Epoch 25/1000
160/160 [==============================] - 0s 186us/step - loss: 264.
2041 - val_loss: 233.9742
Epoch 26/1000
160/160 [==============================] - 0s 198us/step - loss: 255.
```

从输出的数据可以看出：一开始，loss的值非常大，而随着训练不断的进行，loss在逐渐减小。loss是我们预先设定的损失函数计算得到的损失值,val_loss是测试集的损失值，数字越小，说明模型的识别精度越好。

知识链接

history是keras的一个专用对象——History类。History类对象包含两个属性，分别为epoch和history。epoch为训练轮数，而history并不固定，由编译模型（compile）时的参数决定。

In [11]:

```
print(history.history)
```

```
246824264526, 2.115688681602478, 2.1173689365386963, 2.1096567511558
33, 2.11083722114563, 2.1032156944274902, 2.0963600873947144, 2.09162
95647621155, 2.0888951420783997, 2.0848525762557983, 2.08248174190521
24, 2.0792579650878906, 2.0729100704193115, 2.073031425476074, 2.0670
047998428345, 2.062166392803192, 2.0600074529647827, 2.05513763427734
38, 2.0502803325653076, 2.0524749159812927, 2.041222095489502, 2.0349
71594810486, 2.030910313129425, 2.027759611606598, 2.025147795677185,
2.019705355167389, 2.0199210047721863, 2.0124824047088623, 2.00821161
2701416, 2.0077051520347595, 1.9995449781417847, 1.994641125202179,
1.9880762696266174, 1.982707142829895, 1.9785178899765015, 1.97036319
9710846, 1.9677865505218506, 1.9606024026870728, 1.955645203590393,
1.9500704407691956, 1.9488518238067627, 1.9486325979232788, 1.9405600
428581238, 1.9355000257492065, 1.9323023557662964, 1.933681011199951
2, 1.9256304502487183, 1.9219188690185547, 1.9185079336166382, 1.9160
537719726562, 1.9132426381111145, 1.9123388528823853, 1.9035007357597
35, 1.9017903804779053, 1.8960450887680054, 1.8955784440040588, 1.892
2927379608154, 1.8871567845344543, 1.8838707208633423, 1.882677674293
518, 1.8782787322998047, 1.8743590116500854, 1.871442973613739, 1.871
152937412262, 1.8669732809066772, 1.8637937307357788, 1.8594226241111
755, 1.856656014919281, 1.8537089824676514, 1.852191686630249, 1.8478
```

输出history，我们可以发现history的值是字典类型（dict），key为`loss`和`val_loss`，`loss`的值为列表。用`history.history['loss'][0]`得到其中的某一个值。

In [12]:

```
print(history.history['loss'][0])
```

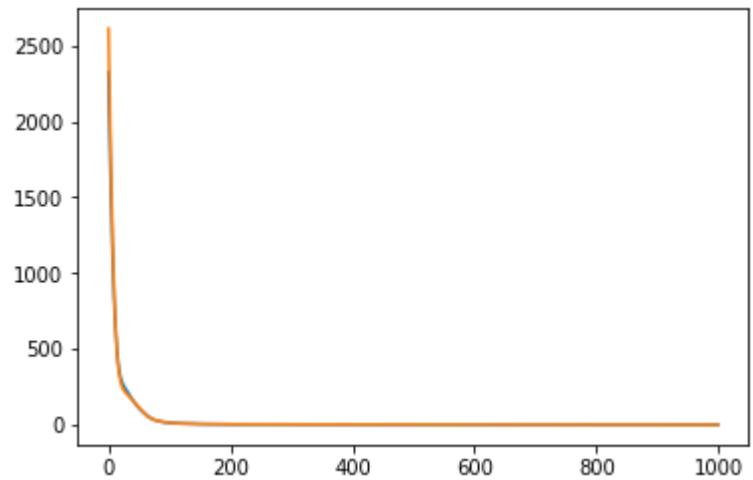
```
2323.8714599609375
```

将history中的数据通过matplotlib绘图表现出来，能够更加直观地看出loss的变化规律。因为在jupyter上调试，代码中加入%matplotlib inline命令，让图片输出在网页中。下面的图中，将训练过程中的训练集loss以及验证集loss全部输出，并打印出测试集的loss。

In [18]:

```
#将图片内嵌在交互窗口，而不是弹出一个图片窗口
%matplotlib inline
import matplotlib.pyplot as plt
t=len(history.history['loss'])
plt.plot(range(t),history.history['loss'])
plt.plot(range(t),history.history['val_loss'])
print("test_loss:",model.evaluate(x_test,y_test))
```

```
20/20 [==============================] - 0s 201us/step
test_loss: 1.0507547855377197
```



我们如何评价一个模型训练是否成功？首先，训练过程中训练集loss要下降到一个较小的值，表示模型收敛较好，没有欠拟合；其次，测试集loss最后与训练集loss要尽可能相似，差距越小越好小，说明该模型没有过拟合。

当一个神经网络模型成功训练出来后，便可以使用该模型进行预测了。通过pandas的DataFrame方法构造x_input，并使用模型的 `predict` 方法进行预测。这里的数据是根据 `Advertising.csv` 的前三条略加修改的，可以看看这个模型输出的结果与真实结果（sales列）是否一致。

In [19]:

```
#用字典生成的DataFrame，需要指定一下列的次序
d={'TV':[230,44,17],'radio':[37,39,45],'newspaper':[69,45,69]}
x_input=pd.DataFrame(d,columns=['TV','radio','newspaper'])
model.predict(x_input)
```

Out[19]:

```
array([[21.8746  ],
       [10.844945],
       [ 8.222065]], dtype=float32)
```

In [20]:

```
x_input
```

Out[20]:

	TV	radio	newspaper
0	230	37	69
1	44	39	45
2	17	45	69

用输出来的结果和前三条数据比较（原来数据中的 sales 分别为：22，10，9），看起来效果还是不错的。

4.保存模型

训练出来的模型，可以保存。下次使用的时候载入，还可以继续训练。一般保存为h5格式，需要先安装h5py。

命令如下：pip install h5py

In [22]:

```
model.save('./model/1-model-vv.h5')    # HDF5文件
```

5.应用模型

使用keras.models的load_model语句载入模型，就可以直接用这个模型来做预测了。

In [23]:

```
from keras.models import load_model
import pandas as pd
model = load_model('./model/1-model-vv.h5')
```

这段代码将在 data 文件夹中生成一个名为 test.csv 的文件。

In [24]:

```
%%writefile ./data/test.csv
TV,radio,newspaper
230.0,37.0,69.0
44,39,45
17,45,69
283.1,42.1,66.1
232.1,8.6,8.7
```

```
Overwriting ./data/test.csv
```

In [25]:

```
#读取数据，并且输出预测结果
x_input=pd.read_csv('./data/test.csv')
model.predict(x_input)
```

Out[25]:

```
array([[21.279892],
       [10.489032],
       [ 9.173268],
       [24.312817],
       [13.093816]], dtype=float32)
```

注：可以导入模型后，继续训练，直到loss不会继续变小。

In [ ]:

```

```