神经网络
——应用案例与调优

神经网络用案例与调优

- 二分类问题：案例1电影评论情绪分类
- 多分类问题：案例2新闻分类
- 标量回归问题：案例3房价预测
- 神经网络建模与训练优化

案例1：二分类问题_电影评论分类

案例1：划分电影评论文本为正面或负面

- IMDB 数据集，来自互联网电影数据库(Internet Movie Database)，专门用于情绪分析。
  - **数据量：**50 000 条，50% 正面和 50% 负面。
  - **数据(review)：**评论文本，严重两极分化的评论。
  - **类别(sentiment)：**共2类，1代表正面、0代表负面

- Keras内置IMDB，对原始数据进行了处理:
  - 统计数据集中出现的不重复的所有单词，并为每个单词编号索引形成“字典”。
  - 每条评论根据文本内容查“字典”，被编码为一个词索引的整数序列。

统计IMDB数据集中出现的不重复的所有单词，并编号形成字典。

imdb.vocab

1	the
2	and
3	a
4	of
5	to
6	is
7	it
8	in
9	i
10	this
11	that
12	was
13	as
14	for
15	with
16	movie
17	but
18	film
19	on
20	not
21	you
22	he
23	are
24	his
25	have
26	be
27	one
28	!
29	all
30	at
31	by
32	an
33	who
34	they
35	from
36	so
37	like
38	there

字典

电影评论文本

id	review
1	I like this movie!
2	This is an example of why the majority of action films are the same …

编码后的整数序列

id	review
1	9,37,10,16,28
2	10,6,32.....

提示：Keras内置IMDB，运行程序时，可自动下载但慢。可下载IMDB.npz，放到 ~/.keras/datasets文件夹下。

第一步 读入和观察数据

意思是仅保留训练数据中前 10000个最常出现的单词，低频词舍弃。

```
#读入数据
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

#查看第1条评论数据和标签，第1、2条数据长度
print(train_data[0])
print(train_labels[0])      #标签值1
print(len(train_data[0]),len(train_data[1]))      # 第1条长度218，第2条长度189
```

第一条评论的值

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100,
43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172,
4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4,
22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62,
386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25,
124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107,
117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029,
13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134,
476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88,
12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
```

第一条评论对应文本内容

```
"? this film was just brilliant casting location scenery story direction everyone's really suited the part they
played and you could just imagine being there robert ? is an amazing actor and now the same being director ? fat
her came from the same scottish island as myself so i loved the fact there was a real connection with this film
the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as
it was released for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried a
t the end it was so sad and you know what they say if you cry at a film it must have been good and this definite
ly was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children are
often left out of the ? list i think because the stars that play them all grown up are such a big profile for th
e whole film but these children are amazing and should be praised for what they have done don't you think the wh
ole story was so lovely because it was true and was someone's life after all that was shared with us all"
```

第二步 准备数据：将数据向量化为 Dense 层可以处理的浮点数向量

将**数据序列向量化**，进行二进制编码，将其转换为 0 和 1 组成的10000 维向量。例：[3,5] 将被转换为只有索引为 3,5的元素是1，其余元素都是 0。

```
import numpy as np
#本函数转换序列集合到(样本数, 10000)维数组，每行存一条评论的编码，有词的位置为1，其他为0
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

#数据向量化。调用函数vectorize_sequences()
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
#查看转换后的结果
print(x_train[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100,
43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172,
4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4,
22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62,
386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25,
124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107,
117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029,
13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134,
476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88,
12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
```

```
[ 0.  1.  1. ...,  0.  0.  0.]
```

将**标签数据向量化**，直接转换为浮点数组即可。

```
#标签向量化
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
print(y_train[0])          #值为1.0
```

第三步 构建神经网络

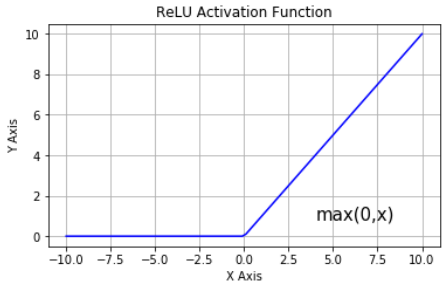
本问题输入数据是向量、输出结果是标量（0，1）。带Relu激活的全连接层Dense的堆叠，在这类问题上表现很好！

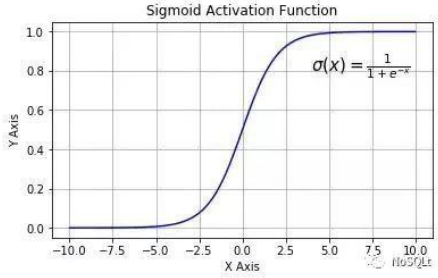
网络结构：

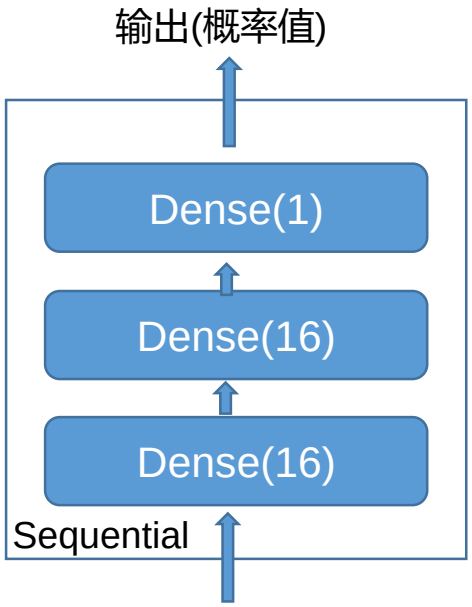
- 两个中间隐藏层，每层都有 16 个单元
- 输出层输出一个标量，预测评论的情感为0或1

```
#构建神经网络
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))  #隐层1
model.add(layers.Dense(16, activation='relu'))        #隐层2
model.add(layers.Dense(1, activation='sigmoid'))    #输出层
```







第四步 编译模型

优化器：一般选择rmsprop 优化器，它可以接收参数，例如:
optimizer=optimizers.RMSprop(lr=0.001)， lr是学习率。
损失函数：对于输出概率值的模型，交叉熵（crossentropy）往往是最好的选择，二元分类最好用binary_crossentropy（二元交叉熵）损失函数。
指标函数：分类一般观察准确率accuracy，也可以传入参数。

```
#编译网络模型（直接传入字符串参数）
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

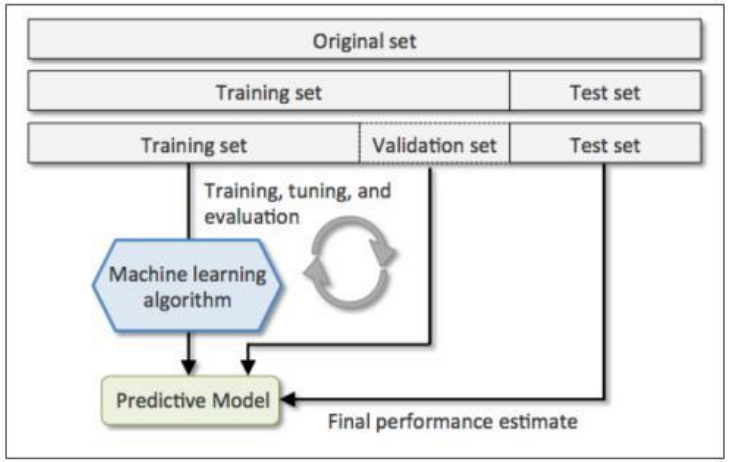
```
#编译网络模型（配置自定义参数）
from keras import optimizers
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                    loss=losses.binary_crossentropy,
                    metrics=[metrics.binary_accuracy])
```

accuracy：按类别值和预测值直接对比，计算在所有预测值上的平均正确率。
如6个样本，其y_true为[0, 0, 0, 1, 1, 0]，y_pred为[0, 0, 0,1, 1,0]，则accuracy= =预测正确数/总数=6/6=100%。
binary_accuracy：只适用于二分类，按阈值threshold将预测值转换为0或1，再计算在所有预测值上的平均正确率。
如6个样本，其y_true为[0, 0, 0, 1, 1, 0]，y_pred为[0.2, 0.3, 0.6, 0.7, 0.8, 0.1]，若阈值为0.5，得到y_pred_new=[0, 0, 1, 1, 1, 0]； 则binary_accuracy=预测正确数/总数=5/6=87.5%。

第五步 训练模型。注意训练之前留出验证集。

神经网络建模一般将数据划分为三个集合：

- **训练集**：用于模型训练
- **验证集**：在训练过程中验证和评估模型性能，开发者可根据反馈信息调节模型配置和超参数
- **测试集**：最终评估模型性能




```
#从训练集中留出验证集
x_val = x_train[:10000]              #10000个样本作验证集Validation
partial_x_train = x_train[10000:]  #其余样本作训练集
y_val = y_train[:10000]              #10000个验证集样本标签
partial_y_train = y_train[10000:]  #测试集样本标签
```

```
#训练模型
history = model.fit(partial_x_train,
      partial_y_train, epochs=20,
      batch_size=512,
      validation_data=(x_val, y_val))
```

history 对象是一个字典，存储训练过程数据:
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
训练损失, 训练精度，验证损失，验证精度

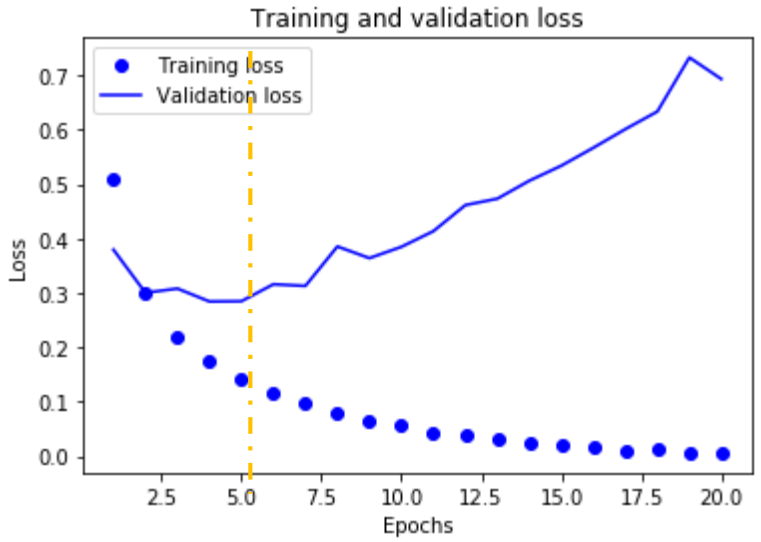
第六步 观察训练过程，调整超参数，防止过拟合。

可以直接观察训练过程中每次迭代的输出，观察训练损失、训练精度、验证损失、验证精度等，评估模型是否有效，决定迭代是否继续。但不直观！

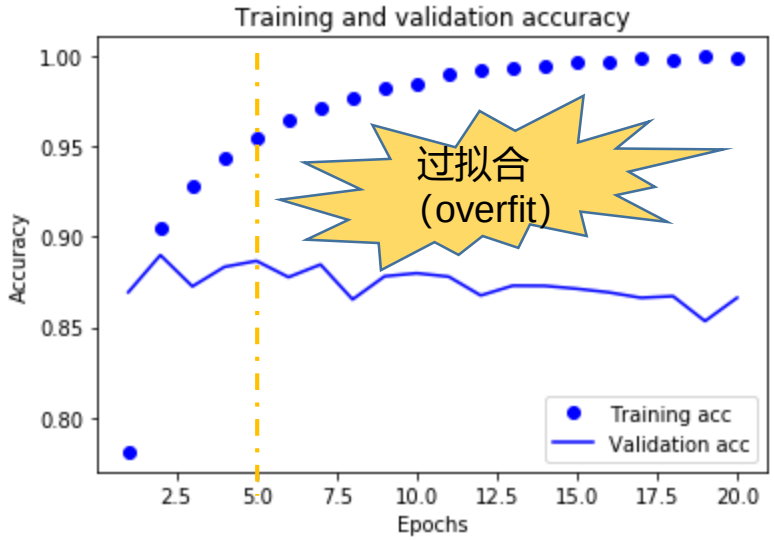
```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 190us/step - loss: 0.5090 - acc: 0.7810 - val_loss: 0.3796 - val_acc: 0.8693
Epoch 2/20
15000/15000 [==============================] - 2s 140us/step - loss: 0.3006 - acc: 0.9049 - val_loss: 0.3002 - val_acc: 0.8898
Epoch 3/20
15000/15000 [==============================] - 2s 144us/step - loss: 0.2179 - acc: 0.9281 - val_loss: 0.3079 - val_acc: 0.8725
Epoch 4/20
15000/15000 [==============================] - 2s 142us/step - loss: 0.1750 - acc: 0.9437 - val_loss: 0.2842 - val_acc: 0.8833
Epoch 5/20
15000/15000 [==============================] - 2s 139us/step - loss: 0.1425 - acc: 0.9543 - val_loss: 0.2847 - val_acc: 0.8866
Epoch 6/20
15000/15000 [==============================] - 2s 140us/step - loss: 0.1148 - acc: 0.9649 - val_loss: 0.3159 - val_acc: 0.8776
Epoch 7/20
15000/15000 [==============================] - 2s 142us/step - loss: 0.0979 - acc: 0.9707 - val_loss: 0.3128 - val_acc: 0.8846
Epoch 8/20
15000/15000 [==============================] - 2s 140us/step - loss: 0.0806 - acc: 0.9765 - val_loss: 0.3853 - val_acc: 0.8653
Epoch 9/20
15000/15000 [==============================] - 2s 139us/step - loss: 0.0659 - acc: 0.9819 - val_loss: 0.3639 - val_acc: 0.8781
Epoch 10/20
15000/15000 [==============================] - 2s 162us/step - loss: 0.0565 - acc: 0.9850 - val_loss: 0.3846 - val_acc: 0.8798
Epoch 11/20
15000/15000 [==============================] - 2s 143us/step - loss: 0.0426 - acc: 0.9903 - val_loss: 0.4135 - val_acc: 0.8779
Epoch 12/20
15000/15000 [==============================] - 2s 141us/step - loss: 0.0374 - acc: 0.9923 - val_loss: 0.4614 - val_acc: 0.8675
Epoch 13/20
15000/15000 [==============================] - 2s 147us/step - loss: 0.0298 - acc: 0.9928 - val_loss: 0.4732 - val_acc: 0.8729
```

对history中的保存的训练过程数据进行可视化，可以更好地评估。

```
#绘制训练损失和验证损失随迭代次数变化图
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
#绘制训练精度和验证精度随迭代次数变化图
plt.clf()
acc = history_dict['acc']
val_acc = history_dict['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



第七步 使用全部训练集，重新训练新的模型。

为了防止过拟合，可以调整各种超参数，也可在某轮之后或达到某一精度停止训练，待形成一个可接受的网络后。使用训练集和验证集的所有数据从头开始训练一个新的网络。

训练 4 轮，使用训练集和验证集的所有数据参加训练。

```
#训练新的模型
model.fit(x_train, y_train, epochs=4, batch_size=512)

#评估模型性能
loss,accuracy  = model.evaluate(x_test, y_test)
print('loss =', loss, 'accuracy =', accuracy)

#查看预测结果，属于各类的概率
classes = model.predict(x_test)
print('测试样本数：',len(classes))
print("分类概率:\n",classes)
```

使用测试集评价模型

```
loss = 0.4251979340744019,accuracy = 0.86284
测试样本数：  25000
分类概率:
 [[ 0.04614712]
 [ 0.99997544]
 [ 0.92470616]
 ...,
 [ 0.26491112]
 [ 0.0486741 ]
 [ 0.91364491]]
```

案例1 二元分类 总结

- **数据预处理：**通常需要对原始数据进行预处理，以便将其转换为张量输入到神经网络中。单词序列可以编码为二进制浮点向量，但也有其他编码方式。
- **隐藏层：**带有 relu 激活的 Dense 层(全连接层)堆叠很常用，可解决很多种问题(包括情感分类)。
- **输出层：**二分类问题，网络的最后一层应该是只有一个单元并使用 sigmoid 激活的 Dense 层，网络输出是 0~1 范围内的标量，表示概率值。
- **损失函数：**二分类问题的 sigmoid 标量输出，应该使用 binary_crossentropy 损失函数。
- **优化器：**无论你的问题是什么，rmsprop 优化器通常都是足够好的选择。
- **训练过程：**一定要一直监控模型在训练集之外的数据上的性能。随着神经网络在训练数据上的表现越来越好，模型最终会过拟合，并在前所未见的数据上得到越来越差的结果。

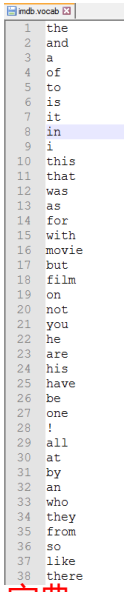
案例2：多分类问题_新闻分类

案例2：新闻分类：单标签、多分类

- Reuters 路透社数据集，1986年发布，短新闻及主题。
  - **数据量：**11,228 条。
  - **数据：**新闻文本。
  - **类别：**共46类，每个新闻只属于一类(单标签)

- Keras内置Reuters，对原始数据进行了处理:
  - 统计数据集中出现的不重复的所有单词，并为每个单词编号索引形成“字典”。
  - 每条新闻根据文本内容查“字典”，被编码为一个词索引的整数序列。

统计Reuters数据集中出现的不重复的所有单词，并编号形成字典。



字典

新闻文本

id	news
1	I like this movie!
2	This is an example of why the majority of action films are the same …

编码后的整数序列

id	review
1	9,37,10,16,28
2	10,6,32.....

提示：Keras内置Reuters，运行程序时，可自动下载但慢。可下载Reuters.npz，放到 ~/.keras/datasets文件夹下。

第一步 读入和观察数据

仅保留训练数据中前 10000个最常出现的单词，低频词舍弃。

```
#读入数据
from keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)

#查看训练集和测试集数据量，第1条评论数据和标签，第1、2条数据长度
print(len(train_data),len(test_data))    #训练集8982条，测试集2246条
print(train_data[0])
print(train_labels[0])    #标签值3
print(len(train_data[0]),len(train_data[1]))      # 第1条长度87，第2条长度56
```

```
#好奇的话，将第一条的词索引解码为单词，执行时会下载reuters_word_index.json比对查找
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
# 注 意，查找时索引减去了 3，因为 0、1、2 是 为 “padding”（ 填 充 ）、 “start of sequence”（序列开始）、 “unknown”（未知词）分别保留的索引
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
print(decoded_newswire)
```

```
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186, 90, 67, 7,
89, 5, 19, 102, 6, 19, 124, 15, 90, 67, 84, 22, 482, 26, 7, 48, 4, 49, 8, 864,
39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 15, 7, 48, 9, 4579, 1005, 504,
6, 258, 6, 272, 11, 15, 22, 134, 44, 11, 15, 16, 8, 197, 1245, 90, 67, 52, 29,
209, 30, 32, 132, 6, 109, 15, 17, 12]
```

? ? ? said as a result of its december acquisition of space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share this year should be 2 50 to three dlrs reuter 3'

#标签值3

第二步 准备数据：将数据向量化为 Dense 层可以处理的浮点数向量数据。

(1)将**数据序列向量化，**进行二级制编码，将其转换为 0 和 1 组成的向量。例：[9,37,10,16,28] 将被转换为10000 维向量，只有索引为 9,37,10,16,28 的元素是1，其余元素都是 0。

```
import numpy as np
#本函数转换序列到10000维数组，对应词的位置为1，其他为0
def vectorize_sequences(sequences, dimension=10000):
   results = np.zeros((len(sequences), dimension))
   for i, sequence in enumerate(sequences):
       results[i, sequence] = 1.
   return results
#调用函数将训练、测试序列向量化
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
#查看转换后的结果
print(x_train[0])
```

```
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186, 90, 67, 7,
89, 5, 19, 102, 6, 19, 124, 15, 90, 67, 84, 22, 482, 26, 7, 48, 4, 49, 8, 864,
39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 15, 7, 48, 9, 4579, 1005, 504,
6, 258, 6, 272, 11, 15, 22, 134, 44, 11, 15, 16, 8, 197, 1245, 90, 67, 52, 29,
209, 30, 32, 132, 6, 109, 15, 17, 12]
```

```
[ 0.  1.  1. ...,  0.  0.  0.]
```

(2)将标签向量化，转换为one-hot分类编码。即将每个标签表示为全零向量，只有标签索引对应的位置为1。

例：3对应的one-hot为[0,0,0,1,0,0,…..0]，只有第4个数时1，其他都是0。

```
#Keras 内置方法to_categorical可以实现这个操作
from keras.utils.np_utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
print(one_hot_train_labels[0]) #查看第1个标签的one-hot向量
```

或用以下自定义函数实现

```
#本函数将标签集合转换为(样本数，46)形式的one-hot向量
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
#调用函数将训练、测试序列向量化
one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
print(one_hot_train_labels[0])       #查看第1个标签的one-hot向量
```

```
[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

#标签值3

第三步 构建神经网络

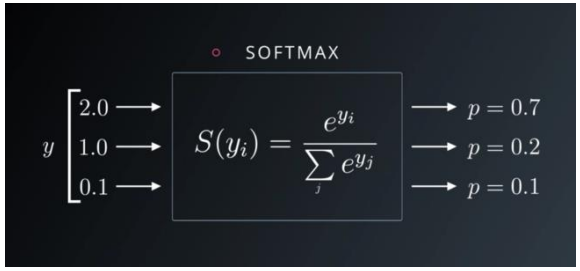
本问题输入数据是向量、输出结果是46位的向量。带Relu激活的全连接层Dense的堆叠，在这类问题上表现很好！

注意：隐藏层单元数不应该小于分类数，否则变换时会丢失信息，分类精度下降。

网络结构：

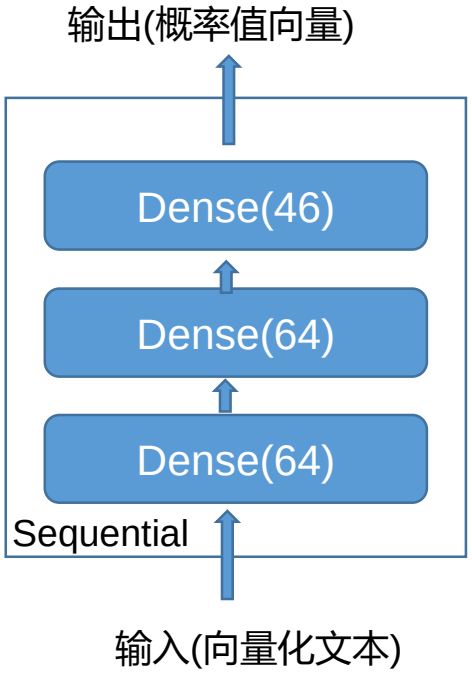
- 两个中间层，每层都有 64 个隐藏单元，Relu激活函数
- 第三层46个单元，softmax激活函数，输出46维向量，每个元素代表不同的输出类别，可以预测46个类别

```
#构建神经网络
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```



概率分布：

$1 > S(y_i) > 0,\ \sum_i S(y_i) = 1$



第四步 编译模型

优化器：一般选择rmsprop 优化器，可以接收参数，例如:
optimizer=optimizers.RMSprop(lr=0.001)， lr是学习率。

损失函数：对于输出概率值的模型，多元分类最好用categorical_ crossentropy（分类交叉熵）损失函数，衡量两个概率分布之间的距离。

指标函数：分类一般观察准确率accuracy，也可以传入参数。

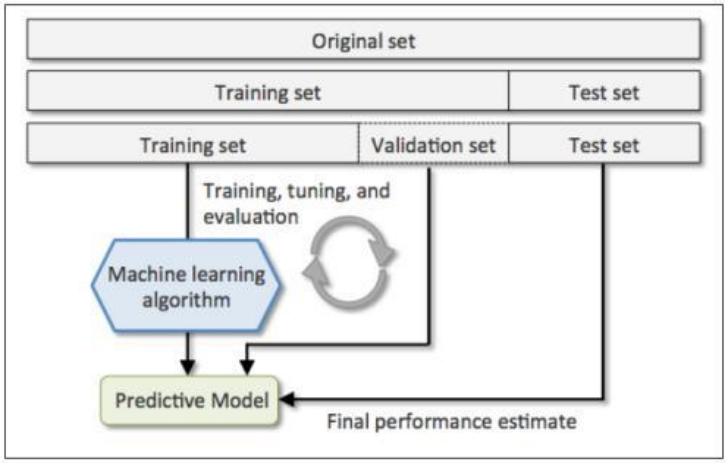
```
#编译网络模型（直接传入字符串参数）
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
#编译网络模型（配置自定义参数）
from keras import optimizers
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses. categorical_crossentropy,
              metrics=[metrics. categorical_accuracy])
```

第五步 训练模型。注意训练之前留出验证集。

神经网络建模一般将数据划分为三个集合：

- **训练集**：用于模型训练
- **验证集**：在训练过程中验证和评估模型性能，开发者可根据反馈信息调节模型配置和超参数
- **测试集**：最终评估模型性能




```
#从训练集中留出验证集
x_val = x_train[:1000]                  #1000个样本作验证集Validation
partial_x_train = x_train[1000:]    #其余样本作训练集
y_val = one_hot_train_labels[:1000]                 #1000个验证集样本标签
partial_y_train = one_hot_train_labels[1000:]   #测集样本标签
```

```
#训练模型
history = model.fit(partial_x_train,
partial_y_train,
 epochs=20,
 batch_size=512,
 validation_data=(x_val, y_val))
```

history 对象是一个字典，存储训练过程数据：
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
训练损失, 训练精度，验证损失，验证精度

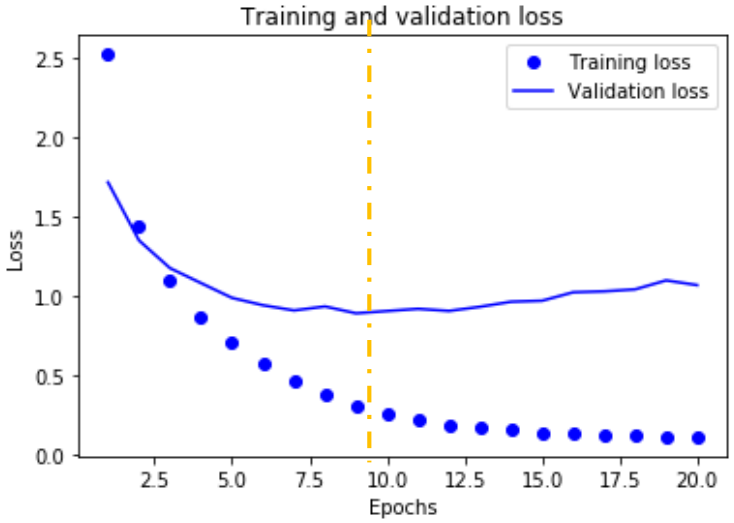
第六步 观察训练过程，调整超参数，防止过拟合。

可以直接观察训练过程中每次迭代的输出，观察训练损失、训练精度、验证损失、验证精度等， 评估模型是否有效，决定迭代是否继续。但不直观！

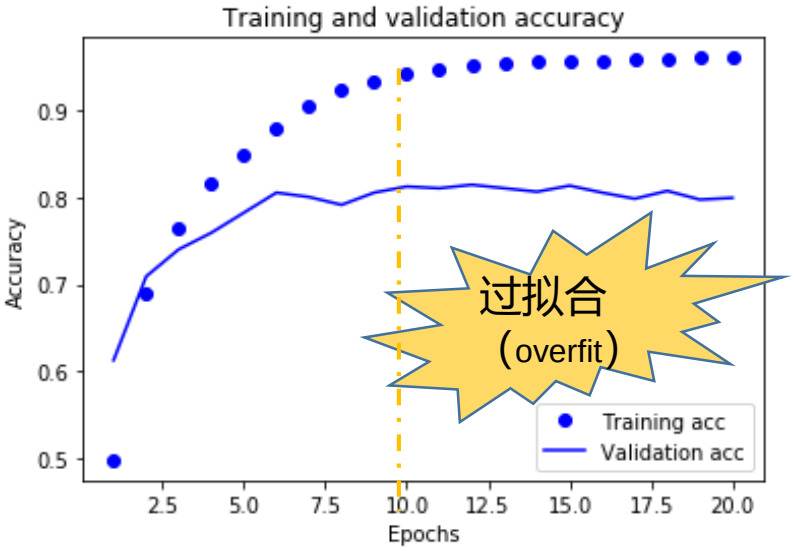
```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 2s 227us/step - loss: 2.5237 - acc: 0.4987 - val_loss: 1.7171 - val_acc: 0.6130
Epoch 2/20
7982/7982 [==============================] - 1s 144us/step - loss: 1.4430 - acc: 0.6891 - val_loss: 1.3503 - val_acc: 0.7090
Epoch 3/20
7982/7982 [==============================] - 1s 137us/step - loss: 1.0983 - acc: 0.7645 - val_loss: 1.1756 - val_acc: 0.7400
Epoch 4/20
7982/7982 [==============================] - 1s 141us/step - loss: 0.8725 - acc: 0.8167 - val_loss: 1.0814 - val_acc: 0.7590
Epoch 5/20
7982/7982 [==============================] - 1s 140us/step - loss: 0.7060 - acc: 0.8485 - val_loss: 0.9881 - val_acc: 0.7820
Epoch 6/20
7982/7982 [==============================] - 1s 151us/step - loss: 0.5694 - acc: 0.8796 - val_loss: 0.9416 - val_acc: 0.8050
Epoch 7/20
7982/7982 [==============================] - 1s 147us/step - loss: 0.4627 - acc: 0.9034 - val_loss: 0.9091 - val_acc: 0.8000
Epoch 8/20
7982/7982 [==============================] - 1s 144us/step - loss: 0.3728 - acc: 0.9223 - val_loss: 0.9340 - val_acc: 0.7910
Epoch 9/20
7982/7982 [==============================] - 1s 152us/step - loss: 0.3052 - acc: 0.9312 - val_loss: 0.8909 - val_acc: 0.8050
Epoch 10/20
7982/7982 [==============================] - 1s 152us/step - loss: 0.2548 - acc: 0.9414 - val_loss: 0.9053 - val_acc: 0.8120
Epoch 11/20
7982/7982 [==============================] - 1s 160us/step - loss: 0.2193 - acc: 0.9470 - val_loss: 0.9190 - val_acc: 0.8100
Epoch 12/20
7982/7982 [==============================] - 1s 150us/step - loss: 0.1878 - acc: 0.9509 - val_loss: 0.9060 - val_acc: 0.8140
Epoch 13/20
7982/7982 [==============================] - 1s 146us/step - loss: 0.1706 - acc: 0.9528 - val_loss: 0.9321 - val_acc: 0.8100
Epoch 14/20
7982/7982 [==============================] - 1s 147us/step - loss: 0.1535 - acc: 0.9557 - val_loss: 0.9639 - val_acc: 0.8060
Epoch 15/20
7982/7982 [==============================] - 1s 141us/step - loss: 0.1389 - acc: 0.9559 - val_loss: 0.9700 - val_acc: 0.8130
Epoch 16/20
7982/7982 [==============================] - 1s 145us/step - loss: 0.1314 - acc: 0.9555 - val_loss: 1.0234 - val_acc: 0.8050
Epoch 17/20
7982/7982 [==============================] - 1s 156us/step - loss: 0.1218 - acc: 0.9578 - val_loss: 1.0293 - val_acc: 0.7980
Epoch 18/20
7982/7982 [==============================] - 1s 152us/step - loss: 0.1197 - acc: 0.9570 - val_loss: 1.0419 - val_acc: 0.8070
Epoch 19/20
7982/7982 [==============================] - 1s 152us/step - loss: 0.1136 - acc: 0.9592 - val_loss: 1.0985 - val_acc: 0.7970
Epoch 20/20
7982/7982 [==============================] - 1s 150us/step - loss: 0.1111 - acc: 0.9595 - val_loss: 1.0687 - val_acc: 0.7990
```

对history中的保存的训练过程数据进行可视化，可以更好地评估。

```
#绘制训练损失和验证损失随迭代次数变化图
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
#绘制训练精度和验证精度随迭代次数变化图
plt.clf()
acc = history.history['acc']
val_acc = history.history['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



第七步 使用全部训练集，重新训练新的模型。

为了防止过拟合，可以调整各种超参数，也可在某轮之后或达到某一精度停止训练，待形成一个可接受的网络后。使用训练集和验证集的所有数据从头开始训练一个新的网络。

训练 9 轮，使用训练集和验证集的所有数据参加训练。

```
#训练新的模型
model.fit(x_train, one_hot_train_labels, epochs=9, batch_size=512)

#评估模型性能
loss,accuracy  = model.evaluate(x_test, one_hot_test_labels)
print('loss =', loss, '  accuracy =', accuracy)

#查看预测结果，属于各类的概率
classes = model.predict(x_test)
print('测试样本数：',len(classes))
print("分类概率:\n",classes)
```

使用测试集评价模型

```
loss = 1.3320837316   accuracy = 0.777382012493
测试样本数：  2246
分类概率:
[[  1.39098429e-05   1.19523193e-05   5.83113611e-08 ...,   2.53943995e-08
   5.93961824e-11   3.16321680e-09]
 [  1.08462940e-04   7.43408455e-04   5.37263345e-09 ...,   2.73136153e-10
   2.57819590e-13   1.66662858e-06]
 [  7.65884668e-03   9.34565365e-01   1.90337651e-05 ...,   3.14045735e-07
   4.45593393e-08   2.23187249e-06]
 ...,
 [  2.27631517e-07   6.43660087e-06   1.51881832e-08 ...,   2.35794162e-10
   3.37686101e-10   3.34803782e-11]
 [  3.70288594e-03   2.63625085e-01   4.38473144e-05 ...,   9.98269570e-06
   1.80066682e-08   1.76851756e-07]
 [  1.33323570e-04   3.19045097e-01   1.79701596e-02 ...,   7.16720265e-08
   1.48703307e-11   1.29487341e-06]]
```

本例得到约 77.8% 的分类精度。性能还是不错的。

对于平衡的二分类问题，完全随机的分类器能够得到50%的精度。下面使用一段测试程序进行随机类别设定，完全随机的精度约为19%。

```
#随机分类
import copy
test_labels_copy = copy.copy(test_labels)      #复制测试集标签
np.random.shuffle(test_labels_copy)             #随机打乱复制标签
#比较原标签和打乱标签数组相同索引位置的值是否相等，相等置1
hits_array = np.array(test_labels) == np.array(test_labels_copy)
#分类正确率等于相等数/总标签数
print('随机分类正确率：', float(np.sum(hits_array)) / len(test_labels))
```

标签向量化的另一种方法

直接将标签转换为整数张量。

```
#将标签向量化
y_train = np.array(train_labels)
y_test = np.array(test_labels)
print('标签：',y_train,' 标签长度：', len(y_train))
```

```
标签： [ 3  4  3 ..., 25  3 25]   标签长度： 8982
```

此时，模型编译的损失函数必须选择sparse_categorical_crossentropy。

```
#编译模型
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy',metrics=['acc'])
```

- One-hot标签使用categorical_crossentropy损失函数
- 整数标签使用sparse_categorical_crossentropy损失函数

案例2 单标签多分类 总结

- **输出层：**网络的输出层采用 Dense 层，单元个数等于分类数 N ，采用 softmax 激活，这样可以输出在 N个输出类别上的概率分布。
- **损失函数：**应该使用交叉熵损失函数，它将网络输出的概率分布与目标的真实分布之间的距离最小化。处理多分类问题的标签有两种方法：
  - 对标签进行one-hot编码，使用 categorical_crossentropy 作为损失函数。
  - 将标签编码为整数，使用 sparse_categorical_crossentropy 损失函数。
- **网络结构：**多分类问题，隐层节点数要大于分类数，避免使用太小的中间层，以免在网络中造成信息瓶颈。

案例3：标量回归问题_预测房价

案例3：波士顿房价预测

- boston_housing 数据集，20世纪70年代美国波士顿郊区房屋价格的中位数。可用于回归预测。
  - **数据量**：506 条，404个训练样本，102个测试样本。
  - **数据**：13 个列特征。每个特征都有不同的取值范围。例如：犯罪率取值范围为 0~1；住宅房间数取值范围为 1~12；到波士顿城区距离取值范围为 0~1000等。

    CRIM：城镇人均犯罪率
    ZN：住宅用地超过 25000 sq.ft. 的比例
    INDUS：城镇非零售商用土地的比例
    CHAS：查理斯河空变量（如果边界是河流，则为1；否则为0）
    NOX：一氧化氮浓度
    RM：住宅平均房间数
    AGE：1940 年之前建成的自用房屋比例
    DIS：到波士顿五个中心区域的加权距离
    RAD：辐射性公路的接近指数
    TAX：每 10000 美元的全值财产税率
    PTRATIO：城镇师生比例
    B：1000（Bk-0.63）^ 2，其中 Bk 指代城镇中黑人的比例
    LSTAT：人口中地位低下者的比例
  - **标签**：MEDV平均房价，以千美元计，价格分布在1万~5万美元。

提示：Keras内置IMDB，运行程序时，可自动下载但慢。
可下载boston_housing.npz，放到 ~/.keras/datasets文件夹下。

第一步 读入和准备数据

```
#读入数据
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

#查看训练集和测试集型形状，第1条数据
print(train_data.shape,test_data.shape)
print(train_data[0])
```

```
(404, 13) (102, 13)
[   1.23247    0.         8.14       0.         0.538      6.142     91.7
    3.9769     4.       307.        21.       396.9       18.72   ]
```

```
#数据标准化
mean = train_data.mean(axis=0)
std = train_data.std(axis=0)
train_data = (train_data-mean)/std
test_data = (test_data-mean)/std
print(train_data[0])
```

因为数据值量纲有差异，映射为均值为0，标准差为1的正态分布。
注意：用训练集的规则计算！

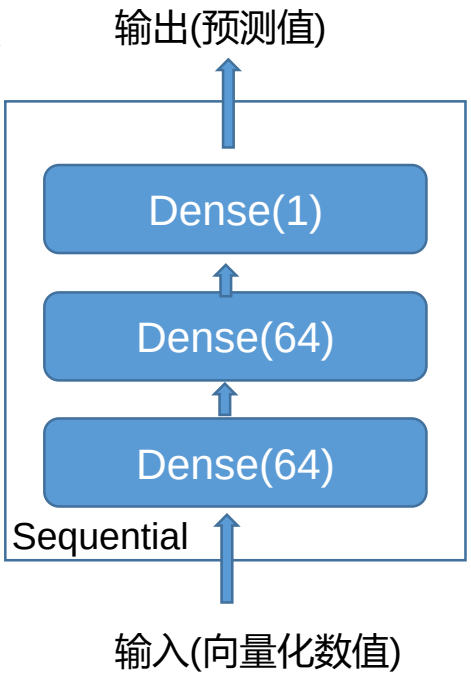
```
[-0.27224633 -0.48361547 -0.43576161 -0.25683275 -0.1652266  -0.1764426
  0.81306188  0.1166983  -0.62624905 -0.59517003  1.14850044  0.44807713
  0.8252202 ]
```

第三步 构建神经网络、编译神经网络

本案例样本数量很少。训练数据越少，过拟合会越严重，较小的网络可以降低过拟合。

- **网络结构：**
  - 两个中间层，每层 64 个隐藏单元，使用Relu激活函数
  - 第三层输出一个标量，预测房价值，不需要激活函数

```
#构建神经网络、编译模型
from keras import models
from keras import layers
def build_model():       #因为后续要建立多个模型，所以定义了一个函数
    model = models.Sequential()
    model.add(layers.Dense(64,activation='relu',input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64,activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

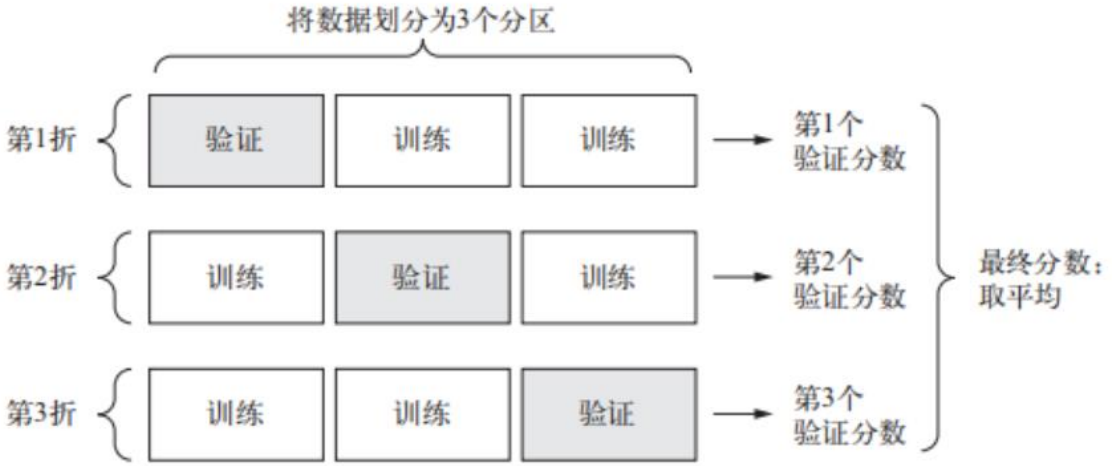


- **网络参数：**
- **损失函数：**回归问题常用均方误差（MSE，mean squared error），即预测值与目标值之差的平方。
- **监测指标：**平均绝对误差（MAE，mean absolute error）。它是预测值与目标值之差的绝对值。如MAE 等于 0.5，表示预测与实际价平均差 500 美元。

第三步 训练模型。注意使用K折交叉验证。

神经网络建模一般将数据划分为三个集合：训练集、验证集和测试集。本例由于数据量较小，如果划分出验证集会非常小（如100个），那么验证分数会有很大波动，无法对模型进行可靠评估。

- **K折交叉验证：**
  - 将数据划分为 K个分区（K 通常取4或5），实例化 K 个相同的模型，将每个模型在 K-1个分区上训练，并在剩下的1个分区上进行评估。
  - 模型的验证分数等于 K 个验证分数的平均值。



3折交叉验证

用K次循环建立K个模型，分别用第i折数据进行训练。

```
#使用K折交叉验证训练模型
import numpy as np
k = 4    #4折
num_val_samples = len(train_data) // k   #每折样本数
num_epochs = 100
all_scores = []      #保存多个模型的mae验证评价
for i in range(k):
    print('processing fold #', i)
    #准备验证数据：第i分区的数据
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    #准备训练数据：其他所有分区的数据
    partial_train_data = np.concatenate([train_data[:i * num_val_samples],
    train_data[(i + 1) * num_val_samples:]], axis=0)
    partial_train_targets = np.concatenate([train_targets[:i * num_val_samples],
    train_targets[(i + 1) * num_val_samples:]], axis=0)

    #实例化已经编译好的模型，训练模型，保存mae验证评价
    model = build_model()
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose
    all_scores.append(val_mae)
    print(val_mae)      #输出mae评价
#输出mae平均值
avg_all_scores=np.mean(all_scores)
print('mae均值：', avg_all_scores)
```

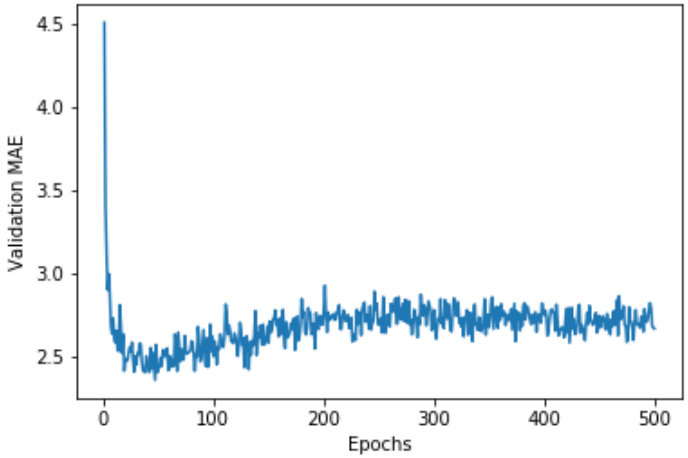
4个模型的评分有一定差异，mae平均值2.41，说明预测值与实际值平均差2.4千美元。实际价1~5万之间。

```
processing fold # 0
2.0567166345
processing fold # 1
2.29677486656
processing fold # 2
2.90970076193
processing fold # 3
2.35735268286
mae均值：  2.40513623646
```

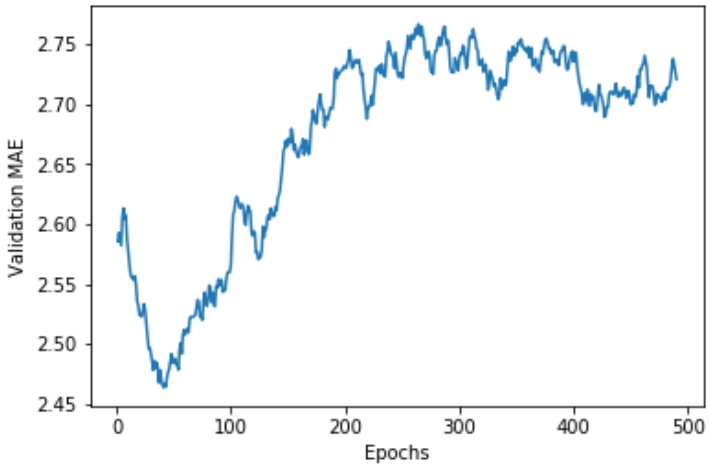
第四步 观察训练过程，调整超参数，防止过拟合。

可以直接观察训练过程中每次迭代的输出，观察Mae的值变化，评估模型是否有效，迭代是否继续。但模型多、迭代次数多，很难直接观察。

记录每个模型在每轮的表现，得到验证分数，进行可视化。



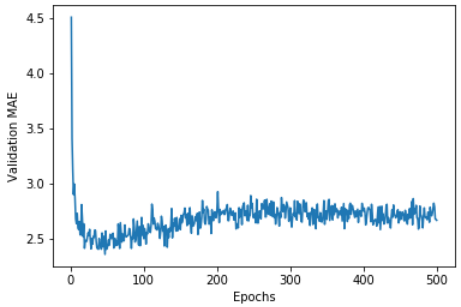
纵轴的范围较大，且数据方差相对较大，所以难以看清规律。



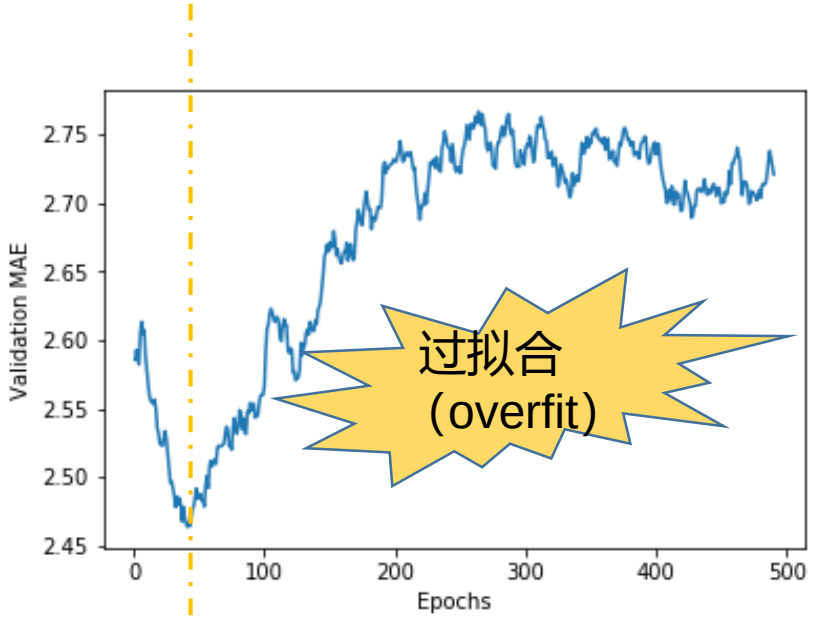
去掉前10个数据点，并将每个点替换为前面数据点的指数移动平均值，得到光滑曲线。
某一point的值替换为与其前一点值（已经替换过）相关，若factor=0.9，有
previous * factor + point * (1 - factor)
例：[3,4,2,5]替换为[3,3.1,2.99,3.19]
第1个值：3不变
第2个值：3*0.9+4*0.1=3.1
依次类推……

```
#使用K折交叉验证训练模型500轮次，保存每轮次的验证结果
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    #准备验证数据：第i分区的数据
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    #准备训练数据：其他所有分区的数据
    partial_train_data = np.concatenate([train_data[:i * num_val_samples],
    train_data[(i + 1) * num_val_samples:]],axis=0)
    partial_train_targets = np.concatenate([train_targets[:i * num_val_samples],
    train_targets[(i + 1) * num_val_samples:]],axis=0)
    #实例化已经编译好的模型
    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets, validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    all_mae_histories.append(mae_history)
```

```
#绘制训练精度和验证精度随迭代次数变化图
#计算所有轮次中的 K 折验证分数平均值
average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
#绘制验证分数
import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



```
#绘制验证分数的平滑曲线（删除前 10 个数据点）
def smooth_curve(points, factor=0.9):
 smoothed_points = []
 for point in points:
     if smoothed_points:
         previous = smoothed_points[-1]
         smoothed_points.append(previous * factor + point *
(1 - factor))
     else:
         smoothed_points.append(point)
 return smoothed_points
smooth_mae_history =
smooth_curve(average_mae_history[10:])
plt.plot(range(1, len(smooth_mae_history) + 1),
smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



观察平滑曲线，可以看到大概在40轮左右，平均绝对误差Mae值开始升高，说明开始过拟合。

第五步 使用全部训练集，重新训练新的模型。

为了防止过拟合，可以调整各种超参数，也可在某轮之后或达到某一精度停止训练，待形成一个可接受的网络后。使用训练集和验证集的所有数据从头开始训练一个新的网络。

训练 40 轮，所有数据参加训练。

```
#训练新的模型
model = build_model()
model.fit(train_data, train_targets, epochs=40, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)

#评估模型性能
test_mse_score, test_mae_score = model.evaluate(test_data,test_targets)
print('测试集mae值：', test_mae_score)

#查看预测结果，各个房屋的价格
price = model.predict(test_data)
print('预测价格:\n',price)
```

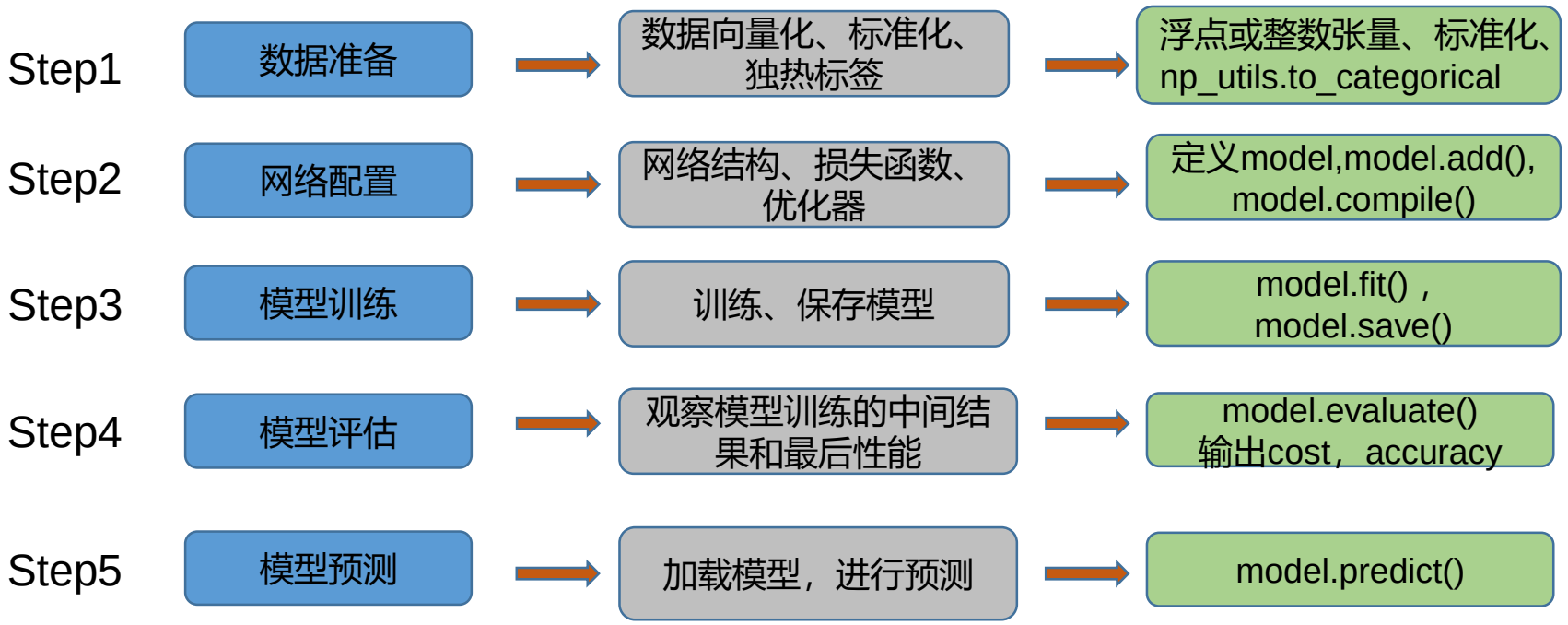
使用测试集评价模型

案例3 标量回归问题 总结

- **损失函数**：回归问题常用的损失函数是均方误差（MSE），即预测值与目标值之差的平方。
- **评价指标**：常用的回归指标是平均绝对误差（MAE），即预测值与目标值之差的绝对值。精度的概念不适用于回归问题。
- **数据预处理**：如果输入数据的特征具有不同的取值范围，应该先进行标准化或归一化，对每个特征单独进行缩放。
- **模型评估**：如果可用的数据很少，使用 K 折验证能可靠地评估模型。
- **网络结构**：如果训练数据很少，最好使用隐藏层较少（通常一到两个）的小型网络，以避免严重的过拟合。
- **最终训练**：一个全新的编译好的模型在所有训练数据上训练。

神经网络建模与训练优化

神经网络建模步骤


Step1
数据准备
数据向量化、标准化、独热标签
浮点或整数张量、标准化、np_utils.to_categorical
Step2
网络配置
网络结构、损失函数、优化器
定义model,model.add(), model.compile()
Step3
模型训练
训练、保存模型
model.fit()，model.save()
Step4
模型评估
观察模型训练的中间结果和最后性能
model.evaluate() 输出cost，accuracy
Step5
模型预测
加载模型，进行预测
model.predict()

Step1数据准备——定义问题

明确问题的类型、输入数据特征、预期输出结果，以及可用的数据集。

默认假设：

- 输出是可以根据输入进行预测的，即未来的规律和过去相同。
- 可用数据包含足够多的信息，足以学习输入和输出之间的关系。

注意：

- 不是所有问题都可以解决。
- 非平稳问题。例如服装推荐的季节性问题。解决方法1是不断利用最新数据重新训练模型；2是在平衡的时间尺度上搜集数据。

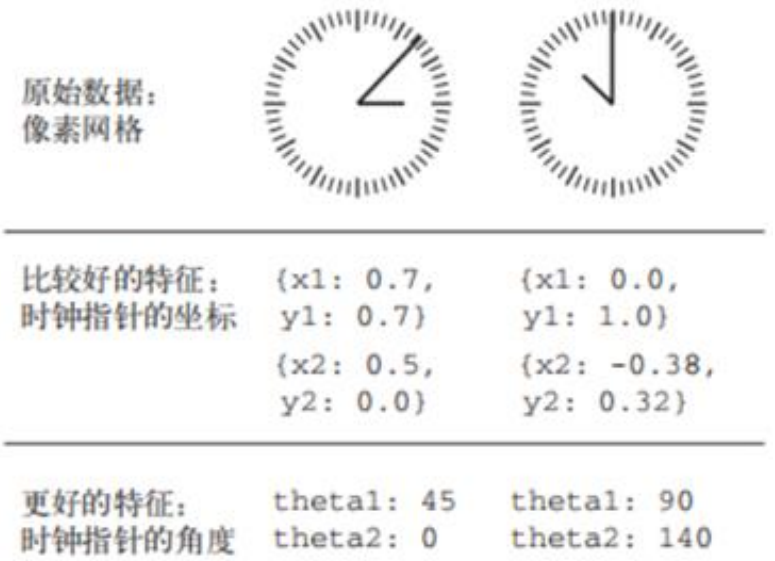
Step1数据准备——特征工程

利用数据和机器学习算法知识对数据进行变换，最大限度地从从原始数据中提取特征以供算法和模型使用。主要包括数据预处理、特征选择、数据降维等。

特征工程的本质：
用更简单的方式表述问题，
从而使问题变得更容易。它
通常需要深入理解问题。

良好的特征可以：

- 用更少的数据解决问题。
- 用更少的资源更优雅地解决问题。



从钟表读取时间的特征工程

Step1数据准备——数据预处理

数据预处理的目的是使原始数据更适于用神经网络处理，包括向量化、标准化、处理缺失值和特征提取。

1. 向量化

神经网络的所有输入和目标都必须是浮点数张量（在特定情况下整数张量）。声音、图像、文本都必须首先将其转换为张量。注意分类标签常用one_hot编码。

2. 值标准化

神经网络的所有输入值最好取值较小(大部分在0~1)、同质性（所有特征在大致相同范围）。

- min-max归一化，将数据映射到[0,1]区间 $x' = \frac{x - min}{max - min}$
- z-score 标准化，映射到标准正态分布 $x' = \frac{x-\mu}{\sigma}$，$\mu$是样本均值，$\sigma$是标准差

3. 处理缺失值

对于神经网络，将缺失值设置为 0 是安全的，只要 0 不是一个有意义的值。网络能够从数据中学到 0 意味着缺失数据，并会忽略它。注意：如果训练时无缺失值，那么模型不能处理缺失值，可在训练时人为增加一些有缺失值的样本。

Step2网络配置

根据任务类型，以及数据特点选择合适的网络层、节点数以及激活函数等。

1. 分类和回归问题都可选用全连接神经网络来建模。隐藏层一般用Relu激活函数。

2. 训练可以从小网络开始，逐步复杂化，避免严重的过拟合。多分类时中间层节点数不能小于类别数。

3. 输出层根据问题类型选择激活函数和损失函数。

问题类型	最后一层激活	损失函数
二分类问题	sigmoid	binary_crossentropy
多分类、单标签问题	softmax	categorical_crossentropy
多分类、多标签问题	sigmoid	binary_crossentropy
回归到任意值	无	mse
回归到 0~1 范围内的值	sigmoid	mse 或 binary_crossentropy

Step3模型训练

通过调整网络结构和各种超参数，监控模型在训练集和验证集上的性能，确定最佳网络模型。

第一步：开发比基准好的模型

开发一个能打败纯随机的小型模型做基准。二分类精度大于0.5，多分类大于随机分类精度。

第二步：开发过拟合的模型

理想的模型是刚好在欠拟合和过拟合的界线上，为了找到这条界线，必须穿过它。

调整模型规模和参数，训练更多轮次，监控各轮的训练损失和验证损失，以及评估指标。如果发现模型在验证数据上的性能开始下降，那么就出现了过拟合。

第三步：调节得到理想模型

进行正则化和调节模型，以便尽可能地接近理想模型，既不过拟合也不欠拟合。最后在所有可用数据（训练集+验证集）上训练最终的生产模型，在测试集上评估。

Step3模型训练——优化器选择

选择合适的参数更新方法， 即**优化器**及**学习率参数**，已经有很多有效方法，根据情况选择，例如：

- BGD（Batch Gradient Descent，批量梯度下降），每次参数更新都使用了所有样本，迭代次数少，易于得到最优解，但样本量大时，收敛速度慢，不适合在线系统。
- SGD（ Stochastic Gradient Descent，随机梯度下降），每次从训练集中随机选择1个或m个样本进行梯度计算，迭代次数可能增多，但对大样本量时适合。但注意学习率不变，并且容易陷入局部最小值。
- Momentum，在梯度上引入动量(不同时间的数据对预测值影响程度给出不同的权重)，加快收敛速度。
- RMSProp（Root Mean Square Prop），对梯度使用了微分平方加权平均数，解决摆动幅度过大问题，加快收敛速度。
- Adam（Adaptive Moment Estimation），将Momentum算法和RMSProp算法结合并通过自适应调节学习率 。

Step3模型训练——参数初始化

参数初始化，不同的初始化参数会导致不同的结果和收敛速度

- **随机初始化：**在0附近取随机值初始化W。
- **全零初始化：**指对偏置b全部初始化为0。W值不能完全相同，否则无法学习到有用信息。
- **Xavier初始化：**根据输入和输出神经元的数量自动决定初始化的范围，保持所有层的方差相似，减少梯度消失问题，使用Sigmoid或tanh激活函数时常用。
- **He初始化：**用Relu激活函数时常用该方法，通过加倍权重补偿负输入值时的方差损失，减少梯度消失问题。

Step3模型训练——避免过拟合

机器学习的根本问题是优化和泛化之间的对立。过拟合是指模型在训练集上性能好，但在测试集上性能差。

- 优化是指调节模型以在训练数据上得到最佳性能；
- 泛化是指训练好的模型在前所未见的数据上的性能好坏。

避免过拟合

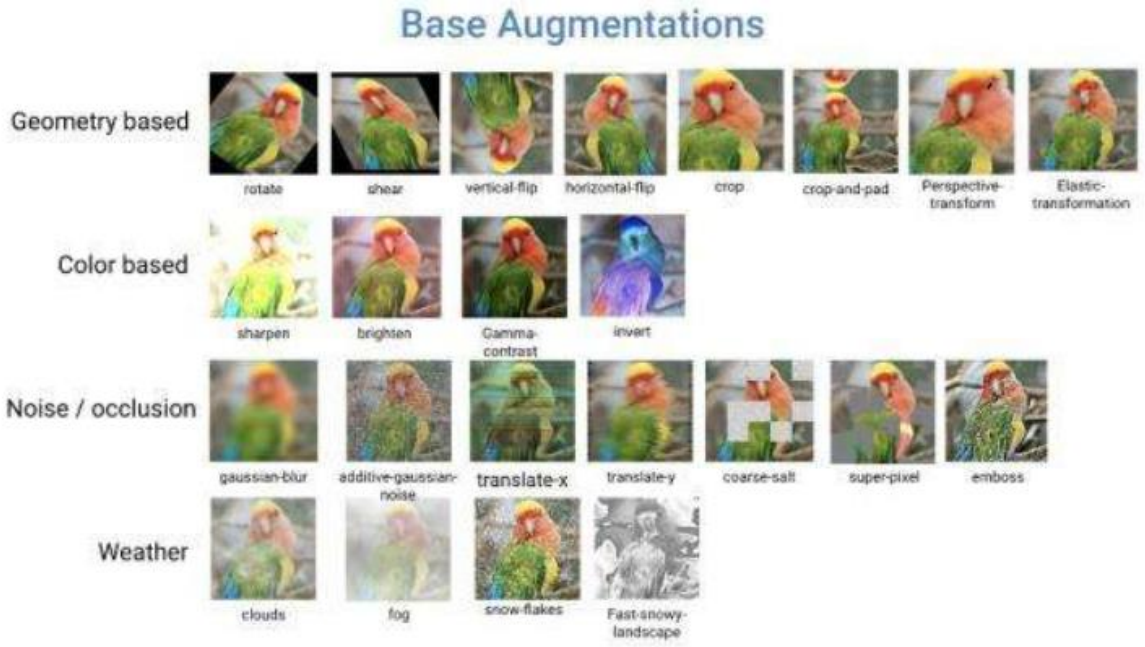
- **获取更多训练数据：**这是最优解决方法。或者通过向训练集添加转换或噪声来人工增加训练数据。例如对图像的翻转、裁剪、色彩变换、缩放和旋转等。
- **减小网络大小：**减少网络的层数和各层单元个数，就可减少模型中的参数，降低模型复杂度。使模型专注学习有预测能力的数据表示。但也要注意不能太小，引起欠拟合。
- **添加权重正则化：**通过对损失函数增加惩罚项，让权重只能取较小的值，使参数值分布的熵更小，模型更简单，使权重更规则。常用L2或L1正则化。
- **添加Dropout正则化：**训练时从网络结构中暂时随机丢弃一部分神经元及其连接。
- **提前停止：**限制迭代次数，通过跟踪迭代次数和对应的精度变化，来决定是否停止。

避免过拟合方法1——获取更多训练数据

收集更多训练数据是最优方案。训练数据多，模型泛化能力越强。如果无法获得，也可在现有数据集上人工进行数据增强。

数据增强：

通过向训练集添加转换或噪声来人工增加训练数据。例如对图像的翻转、裁剪、色彩变换、缩放和旋转等。



避免过拟合方法2——减小网络大小

例题：电影评论分类中选用不同大小的模型

+Original：Dense16-16-1
*Smaller：Dense4-4-1 更优

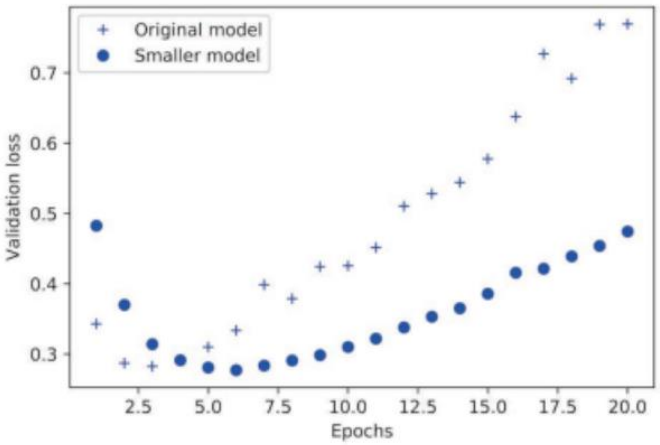


图 4-4　模型容量对验证损失的影响：换用更小的网络

小网络过拟合时间晚，验证损失更小，性能变差速度更慢。

+Original：Dense16-16-1
*Bigger：Dense512-512-1 更差

大网络过拟合时间早(第3轮)，验证损失波动大。

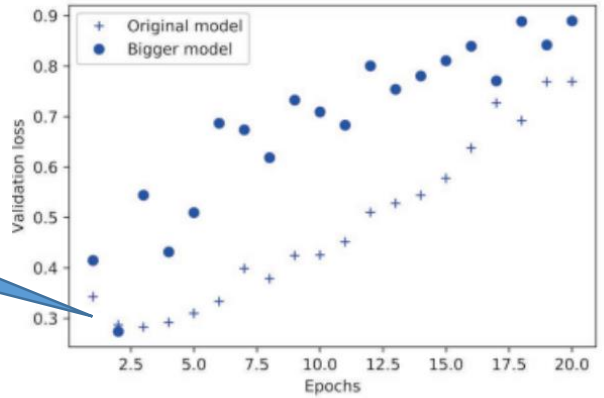
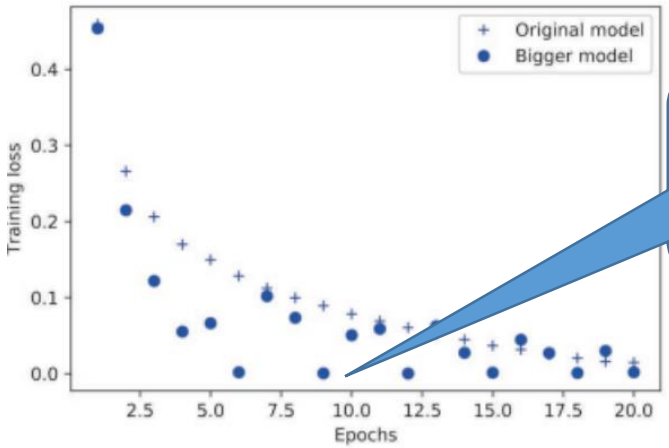


图 4-5　模型容量对验证损失的影响：换用更大的网络

+Original：Dense16-16-1
*Bigger：Dense512-512-1 更差



大网络拟合快，训练损失很快接近零，但验证损失很大，明显过拟合。

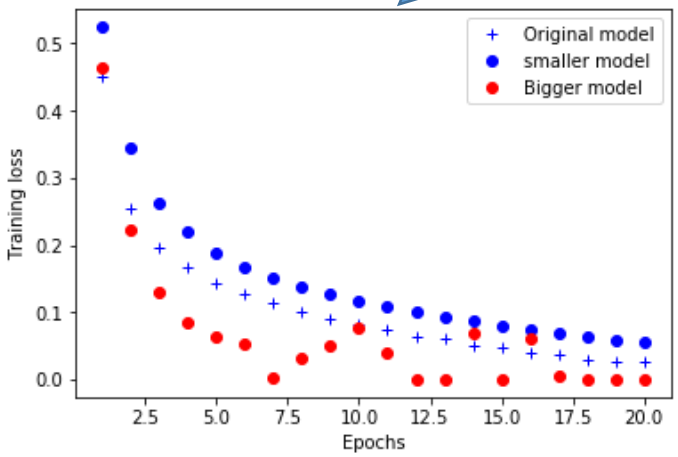
图 4-6　模型容量对训练损失的影响：换用更大的网络

避免过拟合方法2——减小网络大小

例如：电影评论分类中选用不同大小的模型

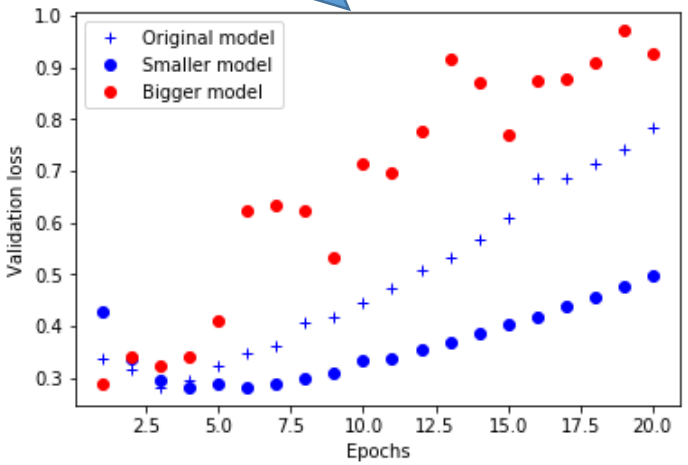
+Original：Dense16-16-1 一般
●Smaller：Dense4-4-1 更优
●Bigger：Dense64-64-1 更差

大网络拟合最快，训练损失很快接近零，



训练损失对比图

小网络过拟合时间晚，验证损失更小，性能变差速度更慢。



验证损失对比图

避免过拟合方法3——添加权重正则化

通过对损失函数增加惩罚项，让权重只能取较小的值，使权重更规则，分布的熵更小，模型更简单。

- L1 正则化： $\boldsymbol{Loss = Loss + \lambda \sum_{i=1}^{n} |\omega_i|}$
- L2 正则化： $\boldsymbol{Loss = +\lambda \sum_{i=1}^{n} \omega_i^2}$

$\boldsymbol{\omega_i}$ 是权重系数，$\boldsymbol{n}$是权重数；$\boldsymbol{\lambda}$称为**惩罚系数，在0-1，1-10之间尝试**

Keras实现：

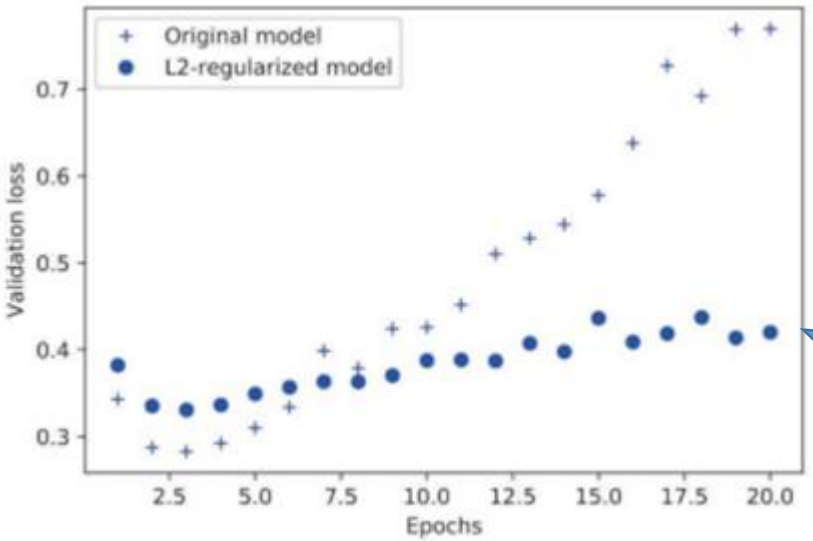
```
from keras import regularizers
from keras.layers import Dense
model.add(Dense(16, kernel_regularizer=regularizers.l2(0.001),activation='relu'))
```

向该层添加 L2 权重正则化，惩罚系数为0.001

注意：惩罚项只在训练时添加，所以网络的训练损失会比测试损失大很多。

例如：电影评论分类模型中第1层添加L2正则化，第2层同时作L1和L2正则化。

```
from keras import regularizers
from keras.layers import Dense
model = models.Sequential()
model.add(Dense(16, kernel_regularizer=regularizers.l2(0.001), activation='relu',  input_shape=(10000,)))
model.add(Dense(16, kernel_regularizer=regularizers.l1_l2(l1=0.001,l2=0.001), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```



添加正则化后更不容易过拟合！

权重正则化对验证损失的影响

避免过拟合方法4——添加Dropout正则化

在训练过程中随机将某些网络层的一些输出特征舍弃(设置为 0)，随机丢弃一部分神经元及其连接。dropout 比率是被设为 0 的特征所占的比例，通常在 0.2~0.5范围内。

其核心思想是在层的输出值中引入噪声， 打破不显著的偶然模式。

例如：向电影评论分类模型中加dropout

```
from keras.layers import Dense
model = models.Sequential()
model.add(Dense(16, activation='relu', input_shape=(10000,)))
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

添加dropout之后验证损失下降，降低过拟合。

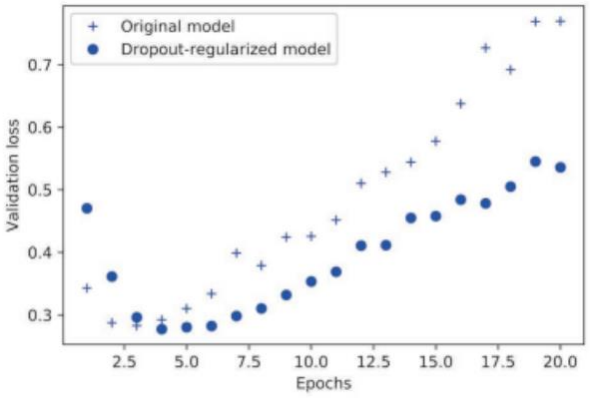
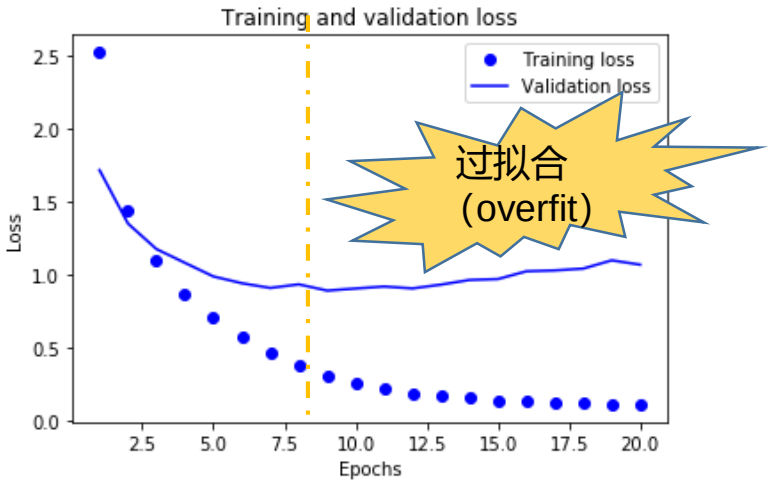


图 4-9 dropout 对验证损失的影响

避免过拟合方法5——提前停止

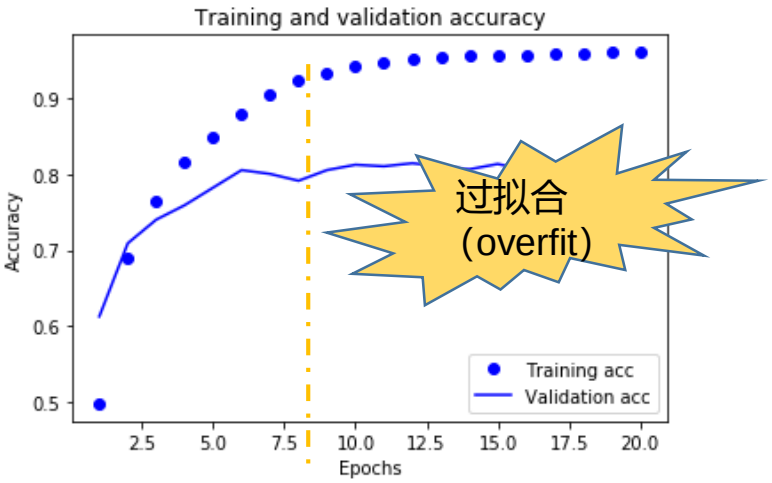
限制迭代次数。通过跟踪迭代次数和对应的精度变化，来决定是否训练停止。

训练损失和验证损失



在第8次迭代开始出现过拟合。

训练精度和验证精度



Step4模型评估——数据集分割

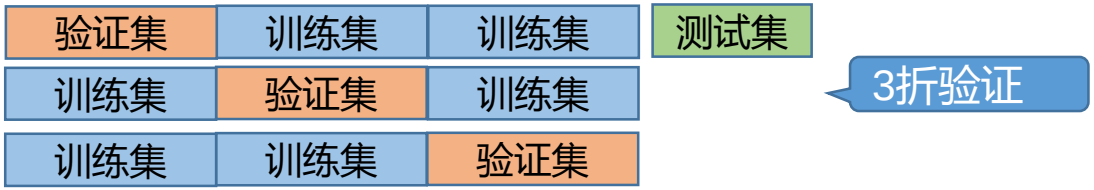
评估模型的重点是将数据划分为三个集合：

- 训练集：用于训练模型
- 验证集：用于评估模型
- 测试集：用于最终模型评估

} 一旦找到最佳参数，在训练集和验证集上最后训练一次模型

✓ **留出法**：直接将数据集划分为互斥的集合。数据量大时采用。

✓ **K折交叉验证法**：将数据集划分为k个大小相似的互斥子集，每次用k-1个作为训练集，余下的一个作验证集。



注意事项：

- **数据代表性**：保持划分后集合数据分布的一致性，避免划分过程中引入额外的偏差而对最终结果产生影响。通常在分割前随机打乱数据。
- **时间箭头**：如果数据有时间性，不能打乱数据，否则会造成时间泄露。并且测试集数据要晚于训练集。
- **数据冗余**：确保训练集和验证集之间没有交集，否则评估不准确。

Step4模型评估——评估指标和可视化观察

训练过程中可视化观察损失值和评价指标，并据此调整模型参数。

一旦找到了最佳参数，在测试数据上测试，得到该模型评估指标值。

主要指标：

- **平衡分类问题：**分类精度，ROC曲线面积
- **不平衡分类问题：**准确率、召回率
- **排序或多标签问题：**平均准确率均值
- **向量回归问题：**平均绝对损失Mae

Step5模型预测

加载模型，在未知数据上进行预测。分类模型会得到属于各类别的概率值，回归模型得到预测数值。

注意事项：

- 未知数据是指未知其标签值；
- 未知数据不一定是未来的数据；
- 验证模型要符合分析假定，否则预测可能不准确。
- 变量设置要在拟合模型的数据范围内，否则预测可能不准确。

实验

实验

1.电影评论的倾向性分类

(1)模仿案例1 用神经网络实现。

(2)调整网络结构和超参数，观察验证精度和测试精度。

- 尝试使用一个或三个隐藏层
- 尝试使用更多或更少的隐藏单元，比如 32 个、64 个等
- 尝试使用 mse 损失函数代替 binary_crossentropy
- 尝试使用 不同的batch_size
- 尝试设置达到一定的验证精度，停止训练（提示：用if语句）

2.新闻分类

(1)模仿案例2 用神经网络网络实现。

(2)调整网络结构和超参数，观察验证精度和测试精度

- 尝试使用一个或三个隐藏层
- 尝试使用更多或更少的隐藏单元，比如 32 个、128 个等
- 尝试使用 不同的batch_size
- 尝试设置达到一定的验证精度，停止训练（提示：用if语句）

3. 房价预测

（1）模仿案例3 用神经网络实现房价预测。

（2）调整网络结构和超参数，观察预测平均绝对误差的影响。

- 尝试使用一个或三个隐藏层
- 尝试使用更多或更少的隐藏单元，比如 32 个、128 个等
- 尝试使用 不同的batch_size
- 尝试设置达到一定的验证精度，停止训练（提示：用if语句）

4. 葡萄酒分类预测

葡萄酒数据集(winequality-red.csv、 winequality-white.csv)搜集了法国不同产区红葡萄酒、白葡萄酒的化学指标。试建立神经网络分类器模型，实现葡萄酒质量预测。

程序命名“学号_Dense2_n”.py，代码对应你的最高分类性能。
1）注意为各语句增加注释，2）代码最后用注释说明实验过程、模型性能。
打包压缩后提交在超星提交至少2题（第4题、1-3题中的至少1题）

葡萄酒数据集

- 葡萄酒是一种成分复杂的酒精饮料，一般通过感官评估判断质量。
- 机器学习算法可根据酒的物理化学性质预测质量。

样本：1599个红葡萄酒样本，4898个白葡萄酒样本
特征：11个。包括：固定酸度、挥发酸度、柠檬酸、残糖、氯化物、游离二氧化硫、总二氧化硫、密度、pH值、硫酸盐、酒精
标签：最后一列quality 。葡萄酒的质量（基于感觉），值为 0-10之间的整数

数据预处理

为了提高数据的质量，可以对离群点和极端值进行丢弃修正，可以指定3个标准差以外的数据为离群点，5个标准差以外的为极端值。经过处理后，红葡萄酒数据可以剔除100多个样本，白葡萄酒可以剔除400多个样本。

通过对质量和各特征的相关性分析，可以得到影响葡萄酒的质量的关键特征顺序：
影响红葡萄酒的质量依次为：酒精度->硫酸盐->挥发酸->总二氧化硫->PH值->残糖->游离二氧化硫->密度->氯化物->柠檬酸->固定酸度；
影响白葡萄酒的质量依次为：酒精度->挥发酸->游离二氧化硫->固定酸度->总二氧化硫->残糖->密度->PH值->柠檬酸->氯化物->硫酸盐。

THANK YOU!