

Python深度学习

代码位置：

本地： D:\AI\deep-learning-with-python-notebooks-master

Git: <https://github.com/83286415/DeepLearningWithPythonKeras>

常用链接：

画图matplotlib: <https://www.meiwen.com.cn/subject/tiilsqtx.html>

mae, mse, rmse: <https://www.jianshu.com/p/9ee85fdad150> 回归问题（boston房价）常用loss

mae, mse的选择: <https://blog.csdn.net/ningyanggege/article/details/85229931>

LSTM, GRU: <https://blog.csdn.net/dqcfkyqdxym3f8rb0/article/details/82922386>

CNN和RNN区别: https://blog.csdn.net/weixin_42137700/article/details/83241774

常用范例：

CSV文件处理：P173 代码见6.3py

pyplot (plt) : P173 代码见6.3py

数学概念：

维度：

a是shape (3,3) 的二维数组，如下

a: [[1 2 3]

[4 5 6]

[7 8 9]]

a shape: (3, 3)

b也是shape (3,3) 的二维数组，如下

b: [[9 8 7]

[6 5 4]

[3 2 1]]

b shape: (3, 3)

c是c = np.stack((a, b), axis=0)，即沿着第一个维度拼接a和b数组成新的c数组

c: [[[1 2 3]

[4 5 6]

[7 8 9]]]

[[[9 8 7]

[6 5 4]

[3 2 1]]]]

c shape (2, 3, 3) 所以c变成了个三维数组，且第一维度是2，即2个3行3列数组

d是d = np.stack((a, b), axis=1)，沿第二个维度拼接

d: [[[1 2 3]

[9 8 7]]]

[[[4 5 6]

[6 5 4]]

[[7 8 9]

[3 2 1]]]

d shape (3, 2, 3) d变成了3个2行3列的数组

e = np.stack((a, b), axis=2) 沿着第三个维度拼接

e: [[[1 9]

[2 8]

[3 7]]

[[4 6]

[5 5]

[6 4]]

[[7 3]

[8 2]

[9 1]]]

e shape (3, 3, 2) e是3个3行2列的数组

第一部分 理论基础

Chapter 1 什么是深度学习

各种概念：

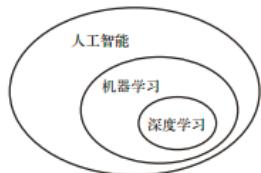


图 1-1 人工智能、机器学习与深度学习

1. 符号主义人工智能symbolic AI

e.g. 专家系统expert system

数据输入+程序规则=数据结果输出

2. 机器学习（浅层学习）

数据输入+预期结果输入=规则输出； 规则+新输入=预测结果输出

浅层学习：在预先定义好的可能性空间（假设空间hypothesisspace）中，利用反馈信号的指引来寻找输入数据的有用表示。

其中假设空间为一组预先设定好的操作（变换），机器通过遍历这组操作（变换）来寻找最优变换，根据任务将数据转化为更加有用的表示。如语音识别，自动驾驶都是机器学习的范畴。P5

3. 深度学习

数据模型的深度：几十甚至上百的连续的表示层

分层表示学习layered representations learning

层级表示学习hierarchical representations learning

深度学习：学习数据表示的多级方法。神经网络中每层对输入数据所做的具体操作保存在该层的权重中，每层的变换有其权重（权重见chapter2 2.4节P36）来参数化。模型在同一时间共同学习所有表示层，而不是依次连续学习。这也成为贪心学习。

损失函数lost funciton（目标函数objective function）：衡量输出与预期之间的距离（损失值）。优化器通过损失函数得到的距离进行反向传播来微调权重值，降低损失值。

机器学习简史

概率建模：

朴素贝叶斯算法，用来做机器学习的分类器。

逻辑回归算法 (logreg) ，同样用来做任务分类，也是分类算法。

早期神经网络：

卷积神经网络+反向传播

核方法：

支持向量机 (SVM, support vector machine)：通过在属于两个不同类别的两组数据点之间找到良好的决策边界 (decision boundary) 来解决分类问题。适用于简单分类问题，不适用于大型数据集

寻找决策边界的方法：先超平面再间隔最大化。难度大，需要用到核方法。P12

核方法：利用核函数 (kernel function) 计算新空间中的点对之间的距离。

决策树、随机森林和梯度提升机：

决策树：类似流程图，可视化

随机森林：构建很多决策树，然后所有输出集成在一起。

梯度提升机：类似随机森林，但会通过迭代训练新模型来解决旧模型的缺点，达到梯度提升。是目前处理非感知数据的最好算法之一。最适用于浅层学习。

深度神经卷积网络 (Deep ConvNet)：

视觉任务和所有感知任务的首选算法，取代了SVM和决策树。

现状：

梯度提升机适用于浅层学习，用XGBoost库处理结构化数据问题。

Keras适用于深度学习，处理感知问题（如视觉任务）。

Chapter 2 神经网络的数学基础

2.1 用Keras分类mnist数据集的示例

Keras中自带mnist数据集，不需下载：from keras.datasets import mnist

具体步骤会在第三章讲解

2.2 神经网络的数据表示

张量 (tensorflow)：

数据（数字）容器。张量是矩阵想任意维度的推广。（维度也叫轴axis）

矩阵是二维张量，有类似于x轴（行）和y轴（列）。

标量 (scalar, 0D张量, 0维张量)：

仅包含一个数字。一个float32或者float64

向量 (vector, 1D张量)：

np.array([12, 3, 6, 14, 7])是一个1D张量（即向量），只有一个轴。

也是5D向量，因为这个轴上有5个维度。

矩阵 (matrix, 2D张量)：

矩阵两个轴，行和列。行row是横着的，列column是竖着的。

3D张量和更高维张量：

多个矩阵组成。见2.2py，3个矩阵组成的3D张量，ndim=3

将多个3D张量合成一个4D张量，而不是在3D张量内多加个矩阵。

深度学习通常处理4D张量，视频数据处理5D张量，不会再高了。

概念解释：

轴 (axis, 阶)：张量维度。3D张量有3个轴，矩阵两个轴。ndim

形状 (shape)：标量形状 ()，向量形状 (5,)，矩阵形状 (3,5)，

3D张量形状 (3,3,5) , **4D张量** (2,3,3,5) 两个3D合成4D张量

数据类型 (dtype) : 通常是float32, float64, unit8。不要用char!

float32, 单精度float, 占4字节, 7位有效数字。

float64, 双精度double, 占8字节, 16位有效数字。

unit8, 无符号整数, 取值范围0~255, 图像处理常用。

张量切片 (tensor slicing) : 类似于python中列表的片取值操作, 索引可用负数。

样本轴 (sample axis) : 张量的第一个轴 (0轴), 也叫批量轴 (batch axis), 或

批量维度 (batch dimension) 。即形状元组的第一个数字。

张量例举:

向量数据: 2D张量, 形状 (样本, 特征features) 。**注:** 向量数据不是向量! P28

时间序列数据或序列数据: 3D张量, 形状 (样本, timesteps, features)

注: timestep始终是形状的第二个元素 (索引为1), 第一个是样本。

图像: 4D张量, 形状 (样本, height, width, channels) 或者

(样本, channels, height, width) TF将channels放在最后。

其中channels可以是1, 表示灰度; 也可以是3, 表示RGB三色深度。

视频: 5D张量, 形状 (样本, frames, height, width, channels) 或者

(样本, frames, channels, height, width) frames为帧, 每一帧就是一副图片

2.3 张量运算tensor operation

keras.layers.Dense(512, activation='relu')

这层可以理解为output = relu(dot(W, input) + b) 输入一个2D张量input, 输出一个2D的output

其中relu是线性整流函数: 此处relu (x) 相当于max (x, 0) 。relu的具体解释链接为

<https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E6%95%B4%E6%B5%81%E5%87%BD%E6%95%B0/20263760?fr=aladdin> 即, 将所有负值归零, 将大于零的值取出。

逐元素运算element-wise:

运算符号: *

numpy中已经实现的, shape相同的张量中, 相对元素间逐个运算。relu就是逐元素运算。

广播:

逐元素运算适用于shape相同的张量间。若shape不同, 则需要对较小的张量进行广播。

广播是将较小的张量添加轴, 然后在此轴维度重复原张量, 达到和大张量shape相同。

x.shape是 (32,10) ; y.shape是 (10,) , 对y广播后, y.shape是 (32,10)

广播后, x和y可以进行逐元素运算。见2.3py

张量点积 (tensor product, 张量积) :

运算符号: dot ()

两个向量的点积dot是标量, 即一个数字。元素数相同的向量间才可以dot运算。

一个矩阵dot一个向量, 返回一个向量; 且dot (x, y) **并不一定等于** dot (y, x)

矩阵之间, 只有当x.shape[1]==y.shape[0]时, 可做点积, 返回形状 (x.shape[0], y.shape[1])

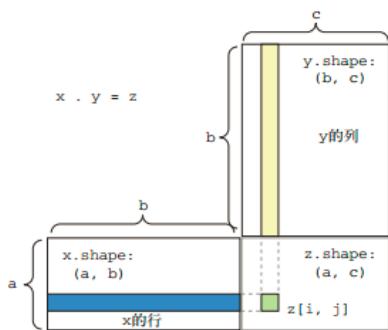


图 2-5 图解矩阵点积

如上图, dot((a, b), (b, c))返回(a,c)。推广到高维张量中:

`dot((a, b, c, d), (d,))` 返回(a, b, c); `dot((a, b, c, d), (d, e))`返回(a, b, c, e)

两张量点积dot, 前面张量shape最后一位与后面张量shape第0位应相同才可以, 返回如上。

张量变形 (tensor reshaping) :

改变张量的行和列, 得到想要的形状shape, 但张量内元素总数不变。

转置 (transpose) 是种特殊的张量变形, 用`new_tensor = np.transpose(tensor)`。见py2.3

几何解释:

元素可以理解为空间内点的坐标, 张量的运算则可以转化为几何运算。

深度学习的几何解释见P35-36 星标

2.4 基于梯度的优化

`output = relu(dot(W, input) + b)` #公式解释见3.4 构建网络

其中W和b都是张量, W是本层的权重 (可训练参数trainable parameters) 矩阵。

W是kernel属性张量 (权重矩阵), b是bias属性张量;

训练循环training loop:

1. 输入训练样本x和对应标签y的数据集——data input
2. 在x上向前传播训练, 得到y_pred预测值——张量运算
3. 计算数据损失, 衡量y_pred和y之间的距离——张量运算
4. 更新网络权重, 使损失略微下降——梯度优化

梯度gradient (张量运算的导数) :

梯度是导数这个概念向多元函数导数的推广。 (多元函数的导数, 是个张量)

`gradient(f)(W0)`是函数f (W) 在值W0附近的曲率的张量

随机梯度下降:

在以上训练循环第4步中, 先计算损失相对于网络参数的梯度, 然后将参数沿着梯度反向移动一点, 使损失减少一点。 $W = step * gradient$ 其中step是学习率learning rate, 取值要大小适中, 太大容易错过最值点, 太小容易陷入极值点。

以上操作在随机一个样本上时, 为随机梯度下降 (SGD,stochastic gradient descent)

在所有样本上时为批量SGD, 在小批量样本 (如128个样本) 上时为小批量SGD。

带动量的SGD被称为优化方法optimization method或优化器optimizer。 RMSProp也是优化器。

其中动量的解释见P40星标, 使用动量可以避免参数落入极小值而错过最小值。

反向传播backpropagation (反向微分reverse-mode differentiation) :

链式chain求导是对 $f(g(x))$ 函数的x求导, 需要先对f求导, 然后在乘以 $g(x)$ 的导数。

Chapter 3 神经网络

神经网络核心组件: 层, 网络, 目标函数, 优化器

层, 多个层组成网络or模型

损失函数: 用于学习的反馈信号

优化器: 决定学习过程如何进行 (这几者的关系, 见P44星标)

3.1 神经网络剖析

层:

有状态的层包含权重; 无权重的层则无状态。

权重:

利用梯度下降学到的张量, 包含网络的知识。

2D张量-----密集连接层densely connected layer(全连接层fully connected layer)-----Keras.Dense类
序列数据3D张量-----循环层recurrent layer-----Keras.LSTM类
图像数据4D张量-----二维卷积层-----Keras.Conv2D类
注：用Keras类处理的每层，只能接受一种形式的张量输入

例：

```
# 作为 Sequential 模型的第一层
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# 现在模型就会以尺寸为 (*, 16) 的数组作为输入, input_shape里不包含批量维度, 即样本量
# 其输出数组的尺寸为 (*, 32)

# 在第一层之后, 你就不再需要指定输入的尺寸了:
model.add(Dense(32)) # 32的含义见 3.4 构建网络
```

模型：

层构成的有向无环图，层为图中的节点。最简单的模型是层的线性堆叠（一个节点连另一个）

线性堆叠用Sequential类，然后一个一个add层即可。用函数式API可以任意有向无环图模型。

除了线性堆叠外，函数式API需要定义层输入，层和层输出，然后再定义网络模型。P48星标。

神经网络架构师：

具有嗅觉直觉，明白对于特定问题用哪些网络架构。

损失函数（目标函数）：

训练过程中让其最小化。

神经网络会包含多个输出，每个输出只对应一个损失函数，所以对多个损失函数的网络取损失函数均值，将其变为标量。

对二分类问题，损失函数用“二元交叉熵” binary crossentropy

对多分类问题，损失函数用“分类交叉熵” categorical crossentropy

对于回归问题，损失函数用“均方误差” mean-squared error

对于全新问题，损失函数需要自行研发。

注：直接优化损失函数不一定总是成功，损失函数必须是可微的。如ROC, AUC不可以直接被优化，可以优化其代替值，如交叉熵。（交叉熵越小，ROC, AUC越大）

优化器：

根据梯度下降来更新网络模型，使损失函数减小。

3.2 Keras简介

Keras后端引擎分别是TensorFlow（谷歌），CNTK（微软）和Theano（蒙特利尔MILA Lab）

建议用TensorFlow作为默认后端引擎，它在CPU上用Eigen运算张量，在GPU上用NVIDIA CUDA的cnDNN来运算张量，而且它可以在CPU和GPU上无缝切换。

Keras工作流：

定义input和output

定义网络（模型）model；其中需要定义层layers,如layers.Dense(隐藏单元数, 激活函数)

配置学习过程，选择损失函数，优化器和监控目标。即编译。

```
model.compile(优化器optimizer, 损失函数loss, 监控指标metrics)
```

训练迭代。model.fit()

3.3 工作站

Ubuntun + GPU + Jupyter notebook 附录A 安装 P283

除了以上方式外，可以在AWS上租GPU，很贵。

3.4 二分类问题

code url:

<https://github.com/83286415/DeepLearningWithPythonKeras/blob/master/3.4-binary-classification.py>

IMDB data

见py: D:\AI\deep-learning-with-python-notebooks-master\3.4-binary-classification.py

数据准备: # 在工作流中的定义input, output

将列表转化为 (sample, feature) 型的整数张量。

将列表进行one hot encode (见hands_on P63)，实例见py3.4.2

构建网络: # 在工作流中的定义网络模型model

Dense(16, activation='relu')

Dense层，16是隐藏单元个数，也是内部学习自由度，也是输出维度。隐藏单元数越多，越可以学到更加复杂的表示，但计算代价也越高，且有可能学到无用的表示模式。

<https://blog.csdn.net/u011311291/article/details/79820073>

output = relu(dot(W, input包含在隐藏单元中) + b)

隐藏单元所含的本层输入input与2.4中提到的权重矩阵W（卷积窗口）做点积，然后与偏差矩阵相加，应用激活函数relu得到output，即本层输出，也是下一层的输入input

激活函数: P55 # 在工作流中的定义网络模型model

中间层通常使用relu作为激活函数，将负值都归0；

最后层通常使用sigmoid作为激活函数，将输出值压缩在0-1之间，作为概率。

最后层的隐藏单元个数是1.

优化器: # 在工作流中的配置学习过程，编译

除了Keras自带的优化器用字符串作为参数传入compile之外，若需要自定义优化器，则：

```
from keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy']) # 在compile参数optimizer中传入一个optimizers的类实例
```

此RMSprop就是自定义在类optimizers模块下的类，lr是其初始化参数。

损失函数和监控指标: # 在工作流中的配置学习过程，编译

如以上优化器，自定义的损失函数和监控指标都是同种方法传入compile里

```
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy]) # 注意在列表中
```

方法验证: # 在工作流中的训练迭代

将训练集分为验证validation集和训练子集

model.fit 已经配置好的model，将15000个训练子集数据输入，每次输入512个，训练迭代20次
fit完毕后，返回一个history字典，字典包含了训练loss，训练acc，验证validation loss和其acc

画图，代码见py：3.4-binary-classification.py 中的3.4.4

在图中寻找loss目标函数在validation loss曲线中最低点（py中出现在第四轮迭代）

证明第四轮后的迭代存在过度拟合，防止此overfit，所以新建model，然后fit 4轮迭代

新model.fit后，model.evaluate(x_test, y_test)来评估新模型，此函数返回一个loss和acc的列表

预测：

model.predict(x_test) 返回一个数组，里面是每个512batch的预测结果，大于0.99的为正面评价，小于0.01的负面评价，其他居中的则为不确定结果。

进一步：

整个实验过程中可以改进的地方：

隐藏层的层数

隐藏单元的个数

损失函数用其他的

激活函数用其他的

注：

rmsprop优化器足够用！RMSProp学习率较小，适合模型微调（见P127）

不要过拟合，一定要控制过拟合，关注validation loss图线的最低点在第几轮迭代。

过拟合：随着训练的进行，模型在训练数据上的性能始终在提高，但在前所未见的数据上的性能则不再变化或者开始下降。

P77

3.5 多分类问题

code：

D:\AI\deep-learning-with-python-notebooks-master\3.5-multi-classification.py

路透社数据集：

同MNIST一样内嵌在Keras里，load_data时，参数num_words=10000为最常见的10000个单词

用法同MNIST，其中labels共46个，及46个news topics

数据准备：

上一步骤load了数据后，此步骤需要将train_data向量化，用自编的vectorize_sequence()

其中用到了numpy.zeros，具体见3.5-multi-classification.py

处理好train_data后，用Keras自带的to_categorical函数将labels变成one hot向量array

网络构建：

多分类问题的输出大于2，（示例中是46输出），所以每层再用16个隐藏单元容易丢失信息，所以每层的隐藏单元数应大于输出维度，本次选择64。若是选择的隐藏单元数小于46，则会造成网络模型**瓶颈flask**，使存储信息部分丢失，造成预测精度下降。见py 3.5.7部分。

另外最后一层针对多分类问题，使用softmax函数来替代二分类问题里的sigmoid函数。

softmax： <https://baike.baidu.com/item/Softmax%E5%87%BD%E6%95%BF/22772270?fr=aladdin> 以下是softmax含义及解释：

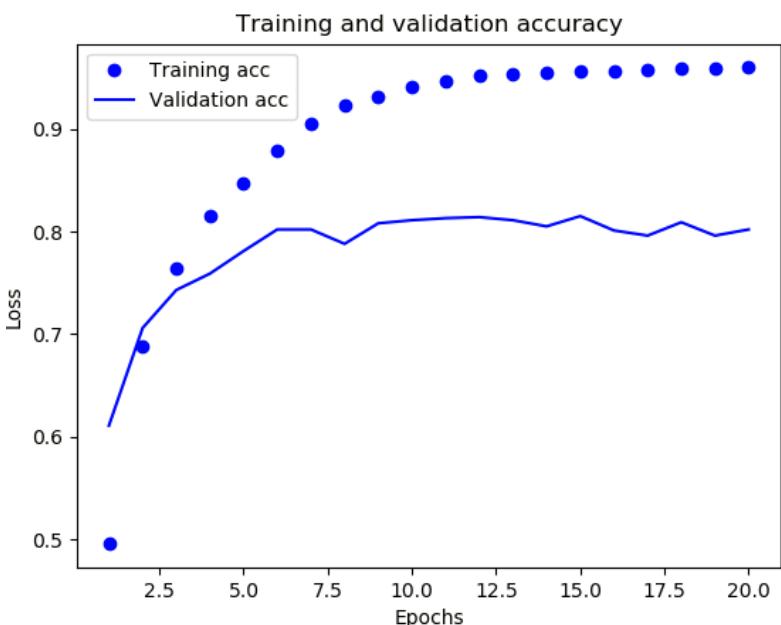
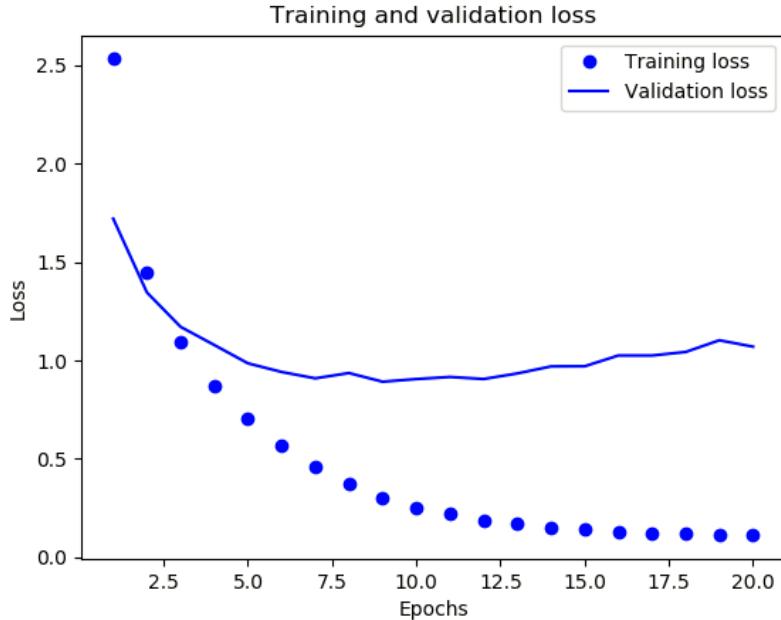
输入向量[1,2,3,4,1,2,3]对应的Softmax函数值是[0.024,0.064,0.175,0.475,0.024,0.064,0.175]。输出向量中拥有最大权重的项对应着输入向量中的最大值“4”。这也显示了这个函数通常的意义：对向量进行归一化，凸显其中最大的值并抑制远低于最大值的其他分量。而且softmax输出的是一个概率分布，此概率分布向量可以通过下面的损失函数求出预测值（的概率分布）与真实值（的概率分布）的距离。

损失函数使用categorical_crossentropy分类交叉熵，此损失函数计算softmax传过来的预测值的概率分布（及概率总和为1的概率向量）与真实值（one hot）概率分布的距离。

注：

对0/1标签（one hot）用损失函数categorical_crossentropy；但对整数标签（1,2,3,4...）用另外—个损失函数sparse_categorical_crossentropy，代码其他部分完全相同

方法验证：



如上两图所示，在loss图的validation线中，第九轮（第九个点）的loss值是在最低点，且在精度图中，第九轮validation线是平稳后的高点，推测在九轮之后出现过拟合overfit问题，所以选择重新训练8, 9和10轮三个网络模型。

重新添加layer, compile, fit之后，得到result，他返回一个二维列表[loss, metrics]

```
model.fit(partial_x_train,  
          partial_y_train,  
          epochs=8,  
          batch_size=512,  
          validation_data=(x_val, y_val))  
results = model.evaluate(x_test, one_hot_test_labels)
```

```
# 8 epochs: [0.9895354098543352, 0.7791629563934126] 列表里分别是交叉熵和精度值  
# 9 epochs: [1.0231352194228134, 0.7751558326443496]  
# 10 epochs: [1.040993108137951, 0.7827248442204849]  
第一个值交叉熵越小越好，第二个值精确度越大越好，所以选择8轮网络模型更合适。
```

之后3.5.4代码中还用完全随机法测试了随机分类精度，只为18%，很低。见py

预测：

在x_test数据集上做predict，得到的每个元素都是长度46的向量，向量中所有元素和是1
prediction[0].shape 看向量有多少个元素
np.sum(prediction[0])看向量所有元素总和，此处肯定是1，因为是概率
np.argmax(prediction[0])看向量中哪个元素值最大，即分在哪一类

瓶颈flask：

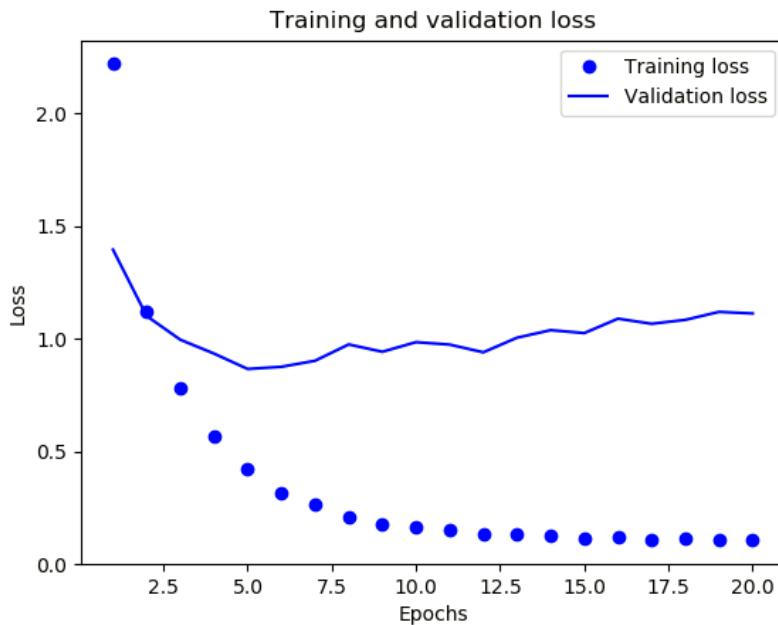
创建网络模型时，layer的Dense中隐藏单元数目应大于最后一层的输出数目46，否则会造成瓶颈

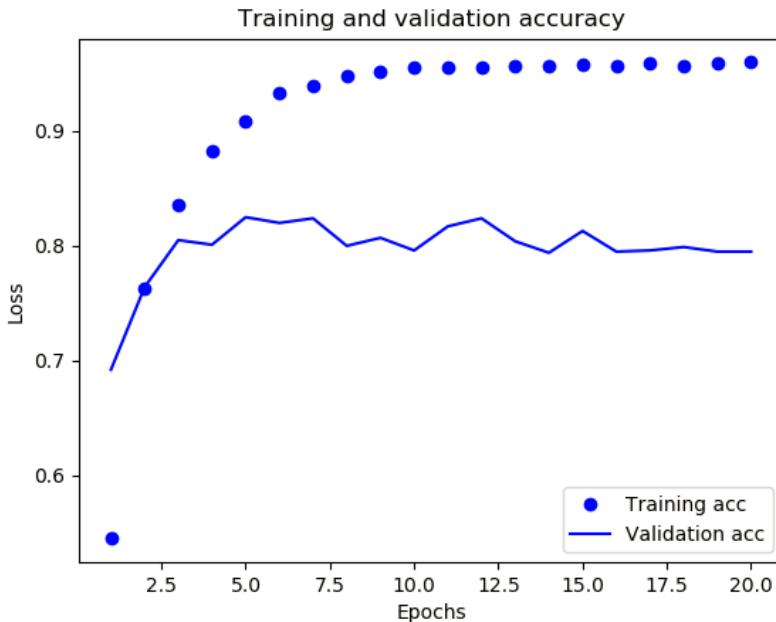
进一步测试：

测试1：原本每层64个隐藏单元，测试32个和128个的结果：

32小于输出维度，会造成瓶颈从而影响精度。

128:





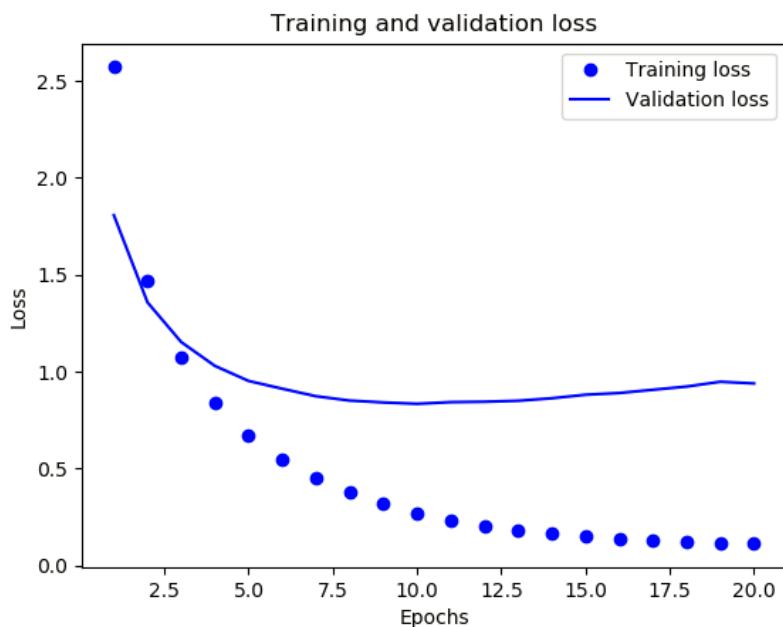
loss validation线在第五轮最低，精度validation线在第四，六轮最高。

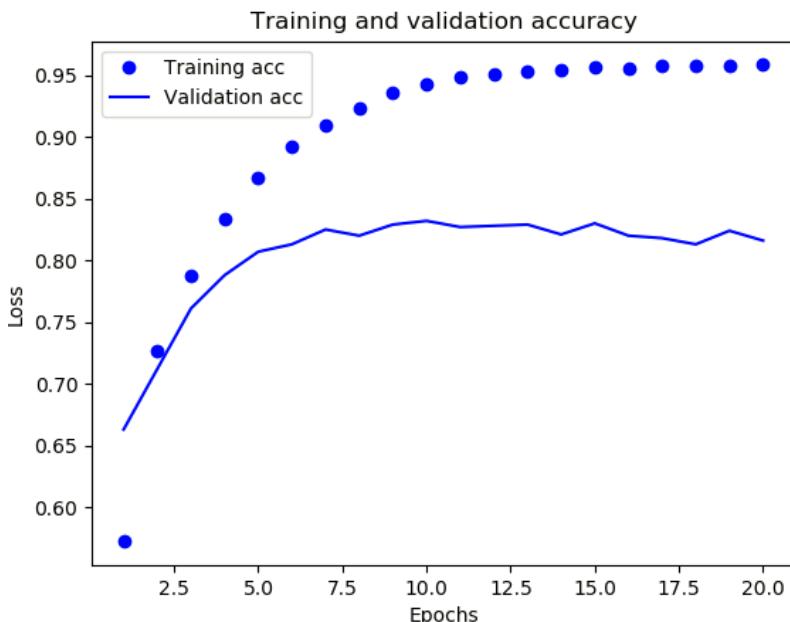
所以重新训练四，五，六轮模型，看交叉熵和精度，结果如下：

```
# Dense(128)
# epochs = 4, 5, 6
# 4: [1.0079761674546388, 0.7698130009435482]
# 5: [0.9829678087497755, 0.7827248442204849]
# 6: [0.9697435776887997, 0.7911843276936776] this is the best one and even better than (Dense(64) and 8
epochs)'s
```

测试2：

由原来2隐藏层改为1层或3层



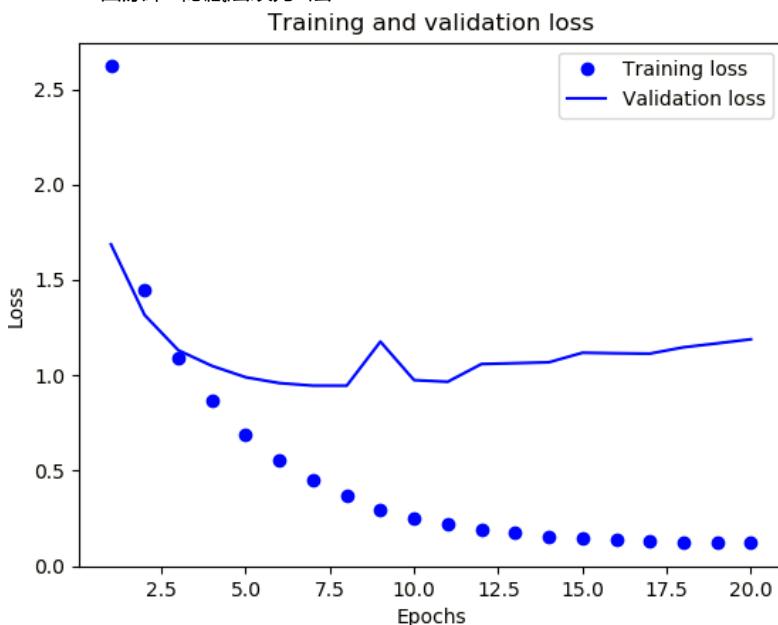


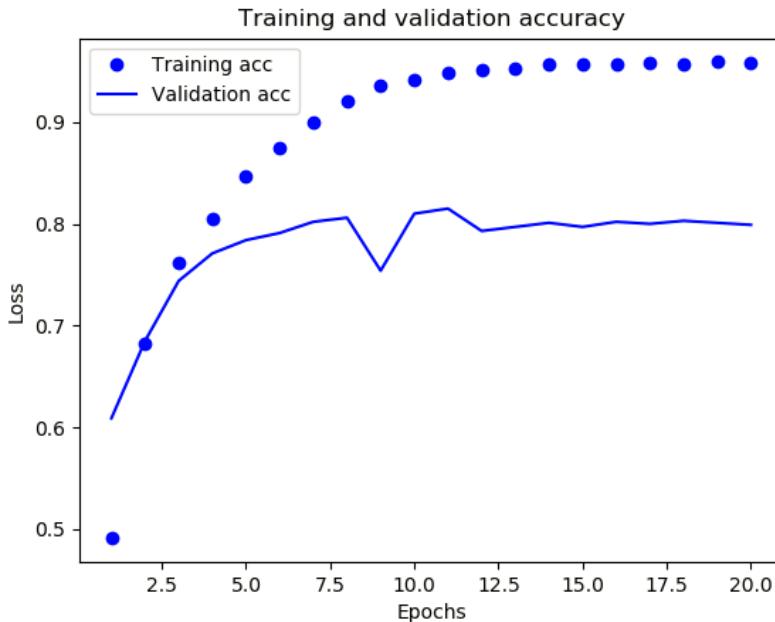
从图上看第十轮的loss最低，acc最高，重新训练10轮模型：

```
# only 1 layer
# epochs = 10
# 10: [0.9018554657777292, 0.7956366874974218] this is the best one ever!!!
```

测试3：

由原来2隐藏层改为3层





从图上看第八、十一轮的loss最低，acc最高，重新训练8,11轮模型：

```
# 3 layers
# epochs = 8, 11
# 8: [1.0612086290137002, 0.7720391807658059]
# 11: [1.1325280534722182, 0.7813891362422084] 结果都不好
```

3.6 回归问题：预测房价

code:

```
D:\AI\deep-learning-with-python-notebooks-master\3.6-regression.py
```

回归问题：

比如根据之前的天气数据预测明天天气。

logistic回归并不是回归问题，而是分类问题。如癌症组和非癌症组对于饮食习惯的分类。

回归问题的loss用MSE均方误差，metrics用MAE绝对平均误差，见书P72的3.6.5小节

波士顿房价数据集：

404个训练数据，103个测试数据。每条数据13个features，具体见py 3.6.1

target是房价中位数。

通过13个影响房价的因素，及404个训练样本数据和其房价label进行训练，对每种因素分配权重，然后预测测试数据集里的房价。

数据准备：

标准化：

因为13中因素的数据各不相同，取值范围也不相同，录入神经网络后无法计算，所以对13features进行标准化，即将其值缩放至0~1之间。

使用np进行标准化standardization：

train_data.mean(axis=0)得均值 train_data.std(axis=0)得标准差

然后train_data和test_data都减去mean后在除以std得到标准化数据

标准差mse（均方差）概念及计算公式：

<https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E5%B7%AE/1415772?fr=aladdin> 用来表示数据离散程度。

注： mean和std值都是train计算出来的，不是test！！

网络构建：

训练数据越少（比如几百个）越容易过拟合overfit，**减少隐藏层可以降低过拟合**（2层）

loss='mse' 均方差，见上红

metric='mae' 平均绝对误差mean absolute error

平均绝对误差MAE：

[https://baike.baidu.com/item/%E5%B9%B3%E5%9D%87%E7%BB%9D%E5%AF%B9%E8%AF%AF%E5%B7%AE/9383373?
fr=aladdin](https://baike.baidu.com/item/%E5%B9%B3%E5%9D%87%E7%BB%9D%E5%AF%B9%E8%AF%AF%E5%B7%AE/9383373?fr=aladdin)

K折验证：

用于训练数据很少时（只有几百个），将训练数据分为K份，在K-1份上训练，最后一份validation，重复K次（在每份上都validation），然后validation的结果取K份的均值。

通常每次fit后的model.evaluate会返回mse和mae两个结果，用np.mean取所有结果的均值

fit函数中的verbose，值为0时是静默模式，其他模式见py中的fit定义

fit函数中的batch_size设定：

<https://blog.csdn.net/wangmengmeng99/article/details/82460437>

<https://www.cnblogs.com/gengyi/p/9853664.html>

fit函数返回的history.history是个字典，每一epoch的数据都会写在对应键的值里。

键分别是：dict_keys(['val_loss', 'val_mean_absolute_error', 'loss', 'mean_absolute_error'])

键分别是（验证集损失，验证集在建模时的metrics，训练集loss，训练集metrics）

画图：

画图时纵轴范围大，可以从图中观测，将取值范围与其他点不同的点剔除后再重新画图

图中每个点起伏不定，曲线不光滑时，可以取点的**指数移动平均值EMA**再重新画图

指数移动平均值EMA：<https://baike.baidu.com/item/EMA/12646151?fr=aladdin>

EMA通常用在观测均值的趋势快慢时。

在EMA图上看到曲线最低点，即为收敛点，之后升高的点说明过拟合over-fit，以最低点轮次再重新训练模型（fit时用全部train data，不在k折），evaluate时用test data

小结：

二分类，多分类，回归

数据预处理，如标准化

头次训练会过拟合，重新定义epoch后再训练，避免过拟合。（过拟合概念P77）

数据少时，用1-2层隐藏层，避免过拟合

每层隐藏单元个数应大于output，避免信息瓶颈

回归和分类用不同的loss，metrics

数据量少时，用k折validation

Chapter 4 机器学习基础

4.1 机器学习四大分支

监督学习：**重点**

序列生成：图像识别，识别图像中的文字

语法树检测：在句子中识别其语法树

目标检测：在图像中识别头像，将头像画框

图像分割：在图像中固定物体上加掩模（mask，即马赛克，打码）

无监督学习：

目的是数据可视化，数据压缩，数据去噪

用于监督学习之前的步骤，来处理数据

主要是降维dimensionality reduction和聚类clustering

自监督学习：

它的标签label是在输入数据中用启发式算法生成，不是输入数据时的dataset中自带的。

例子：自编码器。给定视频中预测下一帧图像，给定文本中预测下一单词。

强化学习：**未来的趋势和主流**

谷歌的DeepMind就是强化学习，造就了阿尔法狗。

智能体agent接受环境信息，学习做出选择，使奖励最大化。

4.2 评估机器学习模型

过拟合概念：

P77

机器学习的目的：是得到**泛化**的模型

训练集：训练模型。

验证集：在验证集上的性能作为反馈信号，调节模型超参（层数，每层大小），参数（权重）。这个调节过程会造成验证集的信息泄露（P77）至模型中，所以验证集不可以同时作为测试集。

测试集：评估模型。**不可以**用测试集的反馈信号来调剂模型！

验证集三法：

1. 简单留出验证

下图所有带标签的数据应该是分成训练集和测试集，然后训练集中再分出一个验证集。

若数据量小，随机划分多次验证集后得到的模型性能差异会很大，此时说明此方法**不适用**。

相关代码参考：<https://github.com/83286415/DeepLearningWithPythonKeras/blob/master/3.5-multi-classification.py>

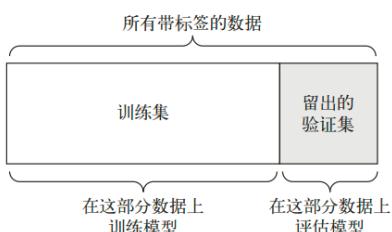


图 4-1 简单的留出验证数据划分

2. K折验证 K-fold validation

在分类（regression）问题“新闻话题”时用过。P79

相关代码参考：<https://github.com/83286415/DeepLearningWithPythonKeras/blob/master/3.6-regression.py>

3. 打乱数据的重复K折验证（推荐使用）

适用：数据量少，且需要精确评估模型时

做法：多次K折验证，每次分区前都打乱数据，最终分数是每次K折的均值。

缺点：计算代价大，每次需训练P*K个模型，其中P是重复验证次数。

注意点：

1. 划分验证集前，先打乱数据
2. 预测未来趋势的问题中，不应该打乱数据，以防时间泄露 (P80)
3. 确保每轮的训练集和验证集数据没有交集 (冗余)

4.3 数据预处理，特征工程，特征学习

数据预处理：

1. 向量化

训练数据数字化，如imdb关键词，新闻话题等转化为数字，其label用one-hot或者多分类

2. 标准化 (0-1化)

应对各个特征取值范围不同，此时不可直接输入模型，需要标准化。

在章节3.6房价预测中，特征标准化有涉及：使数据均值为0，标准差为1.

相关代码参考：

<https://github.com/83286415/DeepLearningWithPythonKeras/blob/master/3.6-regression.py> 中的prepare data

3. 处理缺失值

缺失的值用0代替，问题不大。

特征工程：

用计算机更容易接受的信息来表述问题。如：学习钟表图像来预测时间的问题。给模型输入时针分针的坐标 (x,y) 不如输入时针分针各自的角度更好。

4.4 过拟合和欠拟合

模型优化：调节模型，在训练数据集上获得最佳性能。

模型泛化：训练好的模型在前所未见的数据集上（测试集）的性能好坏。

机器学习的根本问题是模型**优化与泛化**的对立问题。

欠拟合：模型性能仍有改善空间。

过拟合：泛化不再提高，验证指标先是不变，然后变差，**模型开始学习仅和训练集相关的模式**。

记忆容量：模型中学习参数的个数。

太大的容量让拟合更佳，但泛化太差。太小的容量容易欠拟合，导致损失大。应适中。

正常流程：

先选择较少的层和参数，然后逐渐增加层的大小或新层，直到这种增加对损失的影响很小。

见4.5 P89

防止过拟合：

1. 加大训练集数据量（优先）
2. 减少模型允许存储的信息量，强迫模型只存重要模式。（见以下“减小网络”）
3. 添加权重正则化参数（层内添加）
4. 添加dropout正则化（添加层）

1. 增加训练集数据量

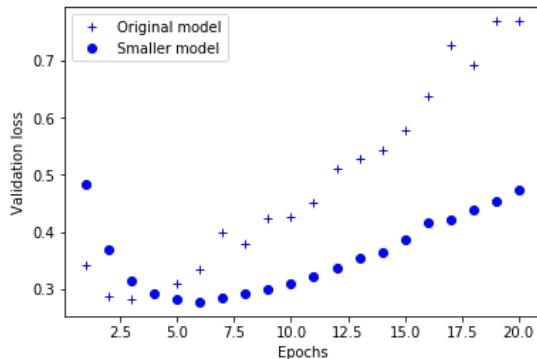
2. 减小网络：

原网络：

```
original_model = models.Sequential()  
original_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))  
original_model.add(layers.Dense(16, activation='relu'))  
original_model.add(layers.Dense(1, activation='sigmoid'))
```

减小网络：

```
smaller_model = models.Sequential()  
smaller_model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))  
smaller_model.add(layers.Dense(4, activation='relu'))  
smaller_model.add(layers.Dense(1, activation='sigmoid'))
```



减小后的网络（原点）比原网络（十字）过拟合更晚（第六轮：第四轮），损失更小。P84
如前所述，大网络更容易（较早）开始学习只和训练集相关的模式映射，及过早过拟合。

3. 添加权重正则化：只在模型训练时添加

权重：模型内W矩阵和b矩阵

权重正则化：使权重内取较小的值

权重正则化的方法：向网络损失函数中添加与较大权重值相关的成本，L1和L2正则化。

L1正则化：添加的成本与权重系数的绝对值成正比。

L2正则化：添加的成本与权重系数的平方成正比，也叫权重衰减weight decay

权重正则化代码：

```
# 完整代码参考4.4-overfitting-and-underfitting.py
```

```
from keras import regularizers
```

```
# L1 regularization
```

```
regularizers.l1(0.001)
```

```
# L2 regularization
```

```
regularizers.l2(0.001)
```

```
# L1 and L2 regularization at the same time
```

```
regularizers.l1_l2(l1=0.001, l2=0.001)
```

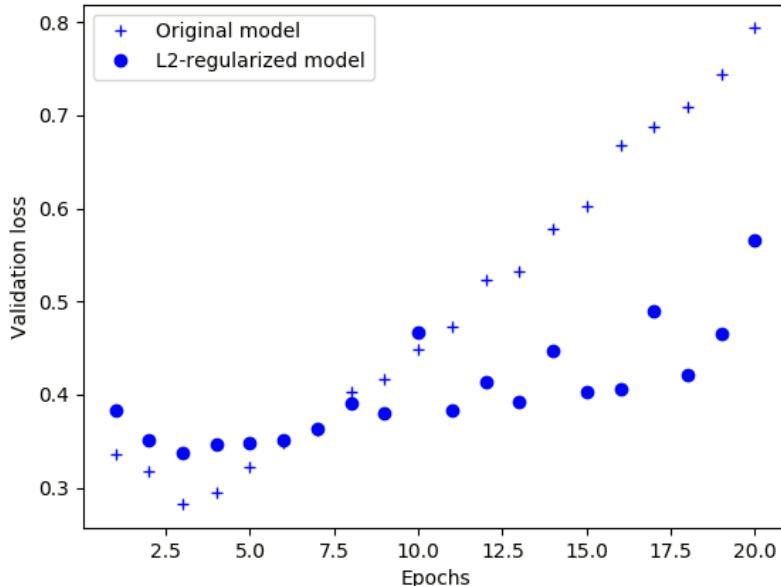
创建模型时的用法，以L2为例（注意：只在训练时添加L2，测试时不加！）

```
l2_model = models.Sequential()  
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),  
activation='relu', input_shape=(10000,)))
```

```

l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                         activation='relu'))
l2_model.add(layers.Dense(1, activation='sigmoid'))
# 完整代码参考4.4-overfitting-and-underfitting.py

```



添加权重正则化（蓝色原点），模型不容易过拟合，且过拟合后损失更小。

4. 添加dropout正则化：

降低过拟合的常用方法之一。

dropout正则化：将某一层输出的向量中是随机元素变为0。按照dropout正则化，改层的输出值会减小，所以为了弥补此损失，该层的输出需要除以该dropout rate后再进行下一层的计算。

dropout rate：通常在0.2-0.5之间，即将所有元素中的20%-50%的元素变为0

dropout思想：在层中引入噪声，打破层内偶然模式，降低过拟合。

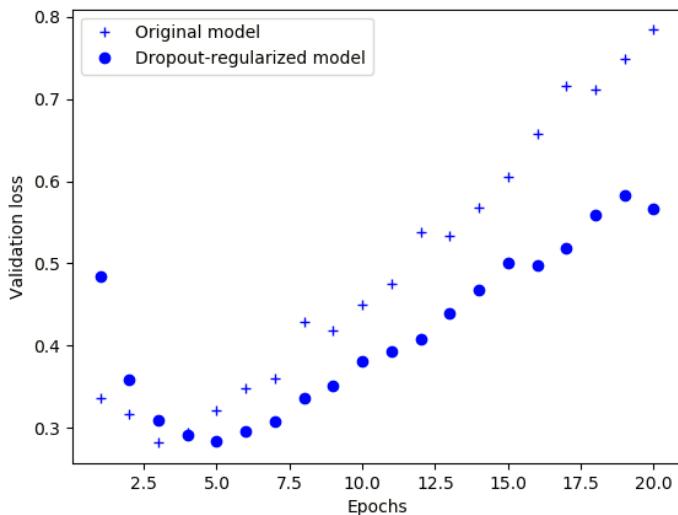
dropout正则化代码：

```

dpt_model = models.Sequential()
dpt_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
dpt_model.add(layers.Dropout(0.5)) # dropout 单独一层
dpt_model.add(layers.Dense(16, activation='relu'))
dpt_model.add(layers.Dropout(0.5)) # dropout 单独一层
dpt_model.add(layers.Dense(1, activation='sigmoid'))

dpt_model.compile(optimizer='rmsprop',
                   loss='binary_crossentropy',
                   metrics=['acc'])

```



dropout正则化添加层的模型（原点）更不易过拟合，且损失更低。

4.5 机器学习的通用工作流程

定义问题，收集数据：

1. 训练数据需要包含：x和y label
2. 问题类型：二分类，多分类，标量回归，向量回归，聚类，生成，强化学习etc

评估成功的指标：

1. 平衡分类问题（每个分类可能性相同）：精度和ROC, AUC
2. 不平衡分类问题：准确率，召回率
3. 排序问题，多标签分类：平均准确率均值
4. 自定义指标见Kaggle竞赛的问题和评估指标

模型评估方法：

1. 留出验证集：训练集大的时候用
2. K折交叉验证：样本量少
3. 重复K折交叉验证：样本量少，但要求模型的评估准确

数据准备：

1. 格式化张量：数字化，使其可输入至矩阵
2. 张量元素的值缩小：通常缩小到[-1, 1]或者[0, 1]之间
3. 若不同特征的取值范围不同，做标准化（4.3节note搜标准化）
4. 特征工程：找出描述问题的且计算机更容易接受的信息来作为输入

模型的关键参数：

1. 最后一层的激活函数：如sigmoid
2. 损失函数：binary_crossentropy, mse
3. 优化器：rmsprop

注：直接优化损失函数不一定总是成功，损失函数必须是可微的。如ROC, AUC不可以直接被优化，可以优化其代替值，如交叉熵。（交叉熵越小，ROC, AUC越大）

表 4-1 为模型选择正确的最后一层激活和损失函数

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

扩大模型规模：

理想的模型是在欠拟合和过拟合的分界线上。**所以一定要创建过拟合模型。**

创建过拟合模型：

1. 添加更多层
2. 把每一层变大
3. 训练更多轮

监视训练损失和验证损失，性能下降说明开始过拟合。

模型正则化和调节超参：**最重要步骤**

出现**过拟合**之后，开始进行正则化和调节超参，是模型得到理想。方法：

1. 添加dropout正则化（见4.4节）
2. 增加或者减少层
3. 添加L1,L2正则化
4. 调节超参：如每层单元个数，优化器的学习率
5. 特征工程
6. 下一章处理图像时的降低过拟合方法：数据增强P111

第二部分 学习实践

Chapter 5 深度学习用于计算机视觉

5.1 卷积神经网络ConvNet简介

密集Dense连接分类器网络层：

`layers.Dense(通道数channels, activation, kernel_regularizer, input_shape=(x, y, z...),)`

卷进神经网络层：

`layers.Conv2D(通道数channels, (a, a), activation, input_shape=(x, y, z...))`

`layers.MaxPooling2D((a, a))` 其中a是从输入图像中提取的图块尺寸a*a

卷积神经网络性质：

平移不变性translation invariant：在图像中某角落学习到的模型，在图像任何角落都可辨认出模式的空间层次结构：

第一层（顶层，后定义）卷积层提取的局部，高通用性特征图，如视觉边缘，颜色纹理等；

第N层（底层，先定义的层）会用来组成更大更抽象的模式，如猫耳朵，狗眼睛。

卷积计算：

```

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
# layers.Conv2D(输出通道数, (输出宽, 输出高), 激活函数, 输入特征图(宽, 高, 深))
# 输入特征图: 第一层卷积层的输入是(28, 28, 1)的3D张量。28是输入特征图的宽和高, 1是深度轴, 表示灰度等级。
若图像是RGB图像, 则此处1变为3, 因为RGB图像有三个颜色通道红绿蓝。

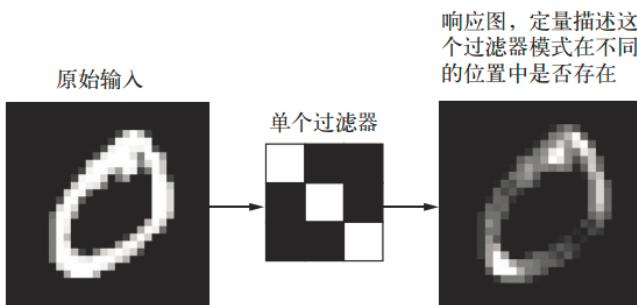
```

第一, 二层结果

Layer (type)	Output Shape	Param #
<hr/>		
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)	0	
<hr/>		
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320

输出特征图: 第一层卷积层的输出是(26, 26, 32)的3D张量。其中32不是深度轴, 而是输出通道channel(过滤器), 在定义卷积层时的[第一个参数](#)。每个输出通道(过滤器, 即[卷积核](#))都包含一个26*26的数值网格。此26计算得来, 不是定义。定义的是3*3, 最终卷积层会变为3*3网格

响应图: 每个通道包含的26*26的网格是此通道对整个输入特征图的响应图, 表示这个通道(过滤器)所学到的模式在输入中不同位置的响应。下图过滤器的网格为3*3



响应图的概念: 某个模式在输入中的不同位置是否存在的二维图

最终卷积层过滤器的网格通常是3*3或者5*5

卷积原理: P99

类似上图, 定义的卷积层参数(3, 3)或(5, 5)的单个过滤器的图块在输入特征图上滑动。每个图块与[卷积核](#)(卷积核是训练得来的权重矩阵, 也称卷积窗口)做张量积后成为1D向量。所有这些1D向量重组得到本卷积层的3D输出特征图。

其中[卷积核](#)也被成为前文的过滤器(也叫[特征图的深度](#)), 是因为卷积计算中会将像素(离散), 时间(一维), 图像(二维), 视频(三维)进行傅里叶变换。将时域和空域上复杂的卷积变为频域上的简单乘积, 然后根据卷积核矩阵对以上变换后的结果进行过滤。

注:

整个卷积计算过程中, 特征图的尺寸在逐渐缩小, 而卷积核(特征图深度)在逐渐增大。

边界效应和填充padding:

若想输出特征图尺寸和输入相同, 上例输入28*28, 输出26*26, 在此卷积层使用参数padding。

padding默认valid, 即不用填充功能; padding=same, 则用填充。

卷积步幅:

默认是1, 即3*3的图块每次移动一列or一排。不要修改为2或3!

最大池化:

以上卷积计算中第二层layers.MaxPooling2D((2, 2))

作用: 特征图的下采样(间隔采样)。使用2*2图块窗口, 步幅是2

意义:

1. 每个卷积层后进行最大池化的特征图下采样, 会让观察窗口越来越大, 越趋近于原图

2. 步幅是2，所以减少了模型中需要处理的特征图元素个数。

最合理的子采样策略：

1. 生成密集窗口，步幅1

2. 观察每个图块的最大激活（最大池化），而不是平均池化（取平均值）

5.2 在小型数据集上训练一个卷积神经网络

猫狗分类的数据下载：<https://www.kaggle.com/c/dogs-vs-cats/data>

代码：<https://github.com/83286415/DeepLearningWithPythonKeras/blob/master/5.2-using-convnets-with-small-datasets.py> 注意：此py代码已经过修改，可多次运行，不会报错。

构建网络：P107有详解

(Conv2D + MaxPooling2D) * 4 + Flatten + Dense + Dense

输入是150*150图像，通过四层卷积来缩减到7*7的特征图，用Flatten层将RGB的3D张量变1D后，用512个隐藏单元的全连接层Dense来学习，最后用带激活sigmoid的Dense输出0或1

模型编译时，损失函数用binary_crossentropy，优化器用RMSprop，衡量用acc

模型保存：（良好习惯，以后要常用！）

在model.fit_generator(各种参数)后用model.save保存模型为h5文件，方便以后使用。如下：

model.save('cats_and_dogs_small_2.h5') # 括号内是保存路径及保存文件名

默认，模型将会保存在工作根目录下D:\AI\deep-learning-with-python-notebooks-master

数据预处理：P108-110有详解

用Keras专门处理图像的类from Keras.preprocessing.image import ImageDataGenerator

将每个图像像素点的RGB三值从0~255缩减到0~1之间，形成训练和测试的ImageDataGenerator的实例。

在这两个实例中导入响应的训练和测试文件夹路径，调整图像大小，每次训练图像数等

训练model.fit_generator，输入每轮fit的step，轮数epochs和validation相关，保存model

画图，观察过拟合点出现在第几轮。（书上是第五轮过拟合，因为validation loss最低）

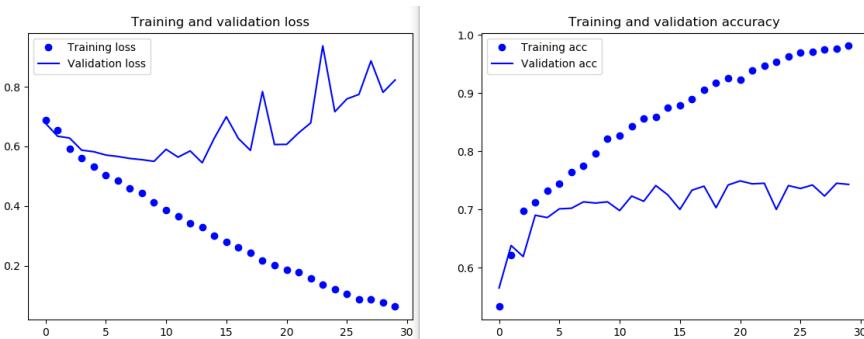
处理图像时用的降低过拟合方法：**数据增强**。

注：

由于代码运行耗时较长，所以此处未添加数据增强功能的model.fit会被注释掉。

之后添加了**数据增强**和dropout层的代码可以完整运行，并输出结果。

下图是未添加数据增强的模型输出，validation loss从第十轮（最低点）开始上升（过拟合）



数据增强data augmentation：

代码：<https://github.com/83286415/DeepLearningWithPythonKeras/blob/master/5.2.5-data-augmentation.py>

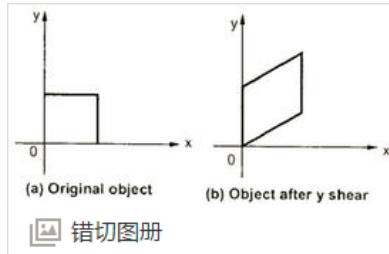
此py为单独验证数据增强功能用，所以未添加在5.2.py内

含义：

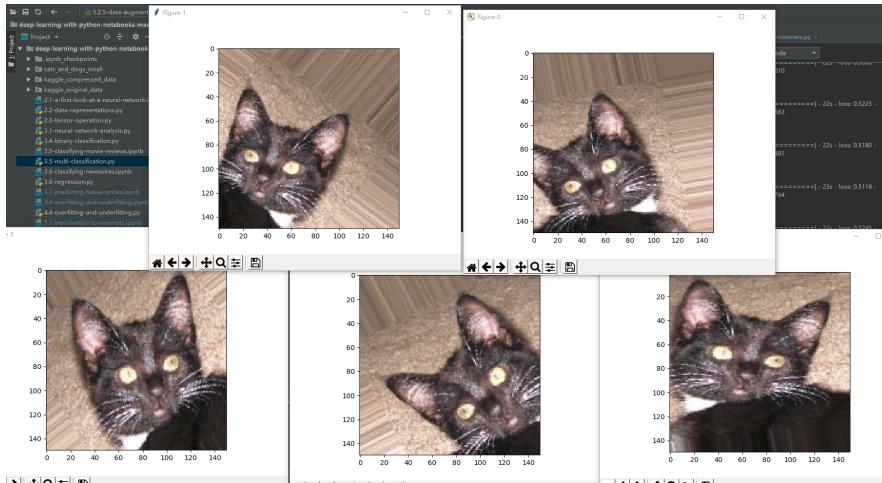
从现有训练样本中生成更多训练数据，利用多种能够生成可信图像的随机变换来增加样本数。但是**不可以增强validation数据集！**

代码解释：

```
datagen = ImageDataGenerator(  
    rotation_range=40, # 角度值0~180°，随机旋转角度  
    width_shift_range=0.2, # 随机水平方向平移范围，总长度的0~0.2之间  
    height_shift_range=0.2, # 随机垂直方向平移范围，总长度的0~0.2之间  
    shear_range=0.2, # 随机错切变换角度，见下示例图  
    zoom_range=0.2, # 随机缩放范围  
    horizontal_flip=True, # 水平翻转  
    fill_mode='nearest') # 平移或旋转后，空缺图像由就近像素填充  
    # 错切变换示例图：
```



数据增强示例图：

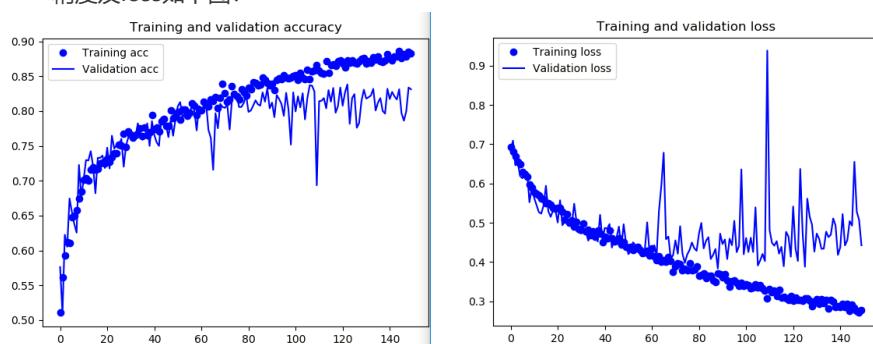


数据增强后，添加Dropout层降低输入相关性：P113

数据增强不可以完全消除过拟合，且输入的相关性增大，在卷积层与全连层间再添加一层dropout层，随机将部分元素变为0，降低过拟合程度。

此部分代码仍在5.2py内，可以完整运行，并输出结果。

精度及loss如下图：



精度在80%左右无法再提高，因为样本数太少。

所以需要使用下一节介绍的预训练模型。

5.3 使用预训练模型的卷积神经网络pretrained network

已经保存好的网络模型，已经在大型数据集上训练好。然后只需要**特征提取**or**模型微调**，即可应用自己的机器学习中。
code: D:\AI\deep-learning-with-python-notebooks-master\5.3-using-a-pretrained-convnet.py

VGG16架构：旧模型，2014年研发。

特征提取feature extraction:

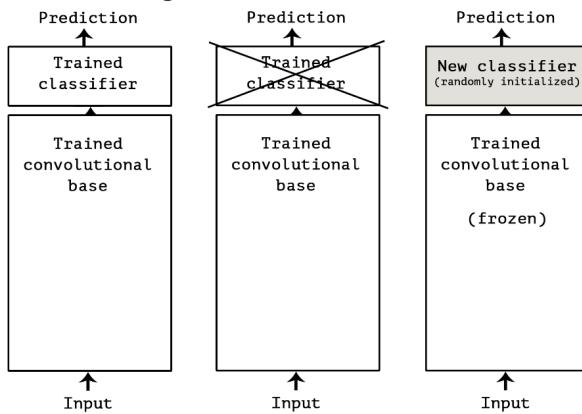
概念：

提取出预训练网络的**卷积基**，给新输入，然后在此卷积基的输出上再训练新的分类器。

注意，只用卷积基部分，而不用密集连接器层，因为卷积基特征图保存了图像中物体的位置，而密集连接器包含了分类器，可能已存在的分类器不适用于现在的分类需求。

卷积神经网络：(卷积层 + 池化层) + 密集连接器层

卷积基：卷积层Conv2D + 池化层MaxPooling2D



可import包：

from keras.applications import VGG16, Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet 这些训练好的卷积基都可以被import直接来用

1. 不使用数据增强的快速特征提取

code: 5.3py内#5.3.1.1

步骤：

通过以上引入VGG16的实例后，开始特征提取。

特征提取的方法是：

创建特征和label的zero矩阵，然后在已创建好的generator中循环；

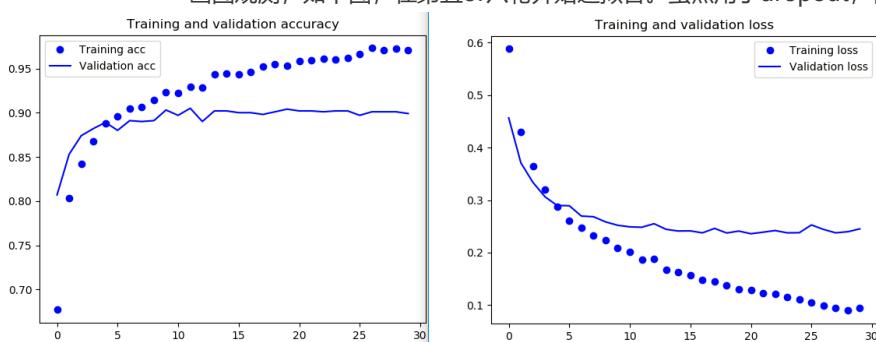
利用VGG16实例生成特征来填充zero矩阵；

创建网络模型，只创建Dense层即可；

为模型添加优化器，损失函数和metrics后编译compile；

model.fit，其中参数注入特征及label的矩阵和验证数据的矩阵；

画图观测，如下图，在第五or六轮开始过拟合。虽然用了dropout，但数据太少。



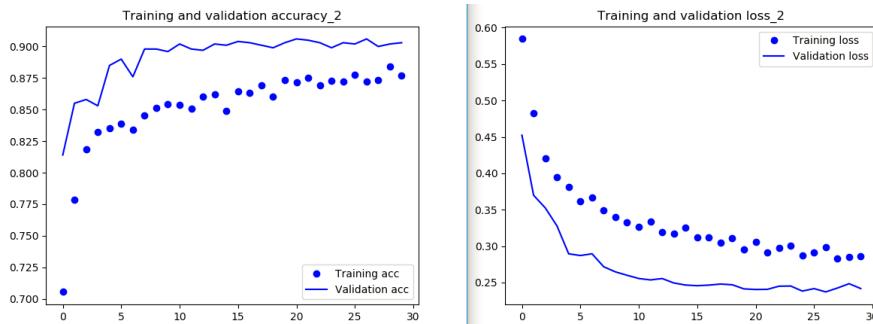
2. 使用数据增强的快速特征提取

注：需要GPU支持，若无GPU，不要尝试这种方法，计算代价太高！

步骤：

像添加一个层一样将预训练网络模型conv_base添加进model中；
后再添加Flatten和Dense层；
冻结conv_base.trainable = False 防止其权重在Dense层中被修改；
创建训练和validation生成器；
编译模型model.compile; 这一步是要让冻结的修改生效！

训练，画图：



这应该是目前位置最好看的图了，精确到90%左右，这是数据增强加预训练网络的威力！

但是这个程序跑了近24个小时，太耗时间了，真的需要GPU的支持啊，否则效率太低。

模型微调fine tuning: 微调猫耳朵

此章节用到了数据增强，特征提取和模型微调三法！

将之前冻结的conv_base的顶部卷积层（后定义的卷积层，包含猫耳朵，狗眼睛）解冻，微调这些预训练网络模型，使其底部逆反馈来调节的特征图的抽象表示更贴近于接下来的Dense层的分类器。见书P125图字，图的蓝色为底部，黄色为顶部。**顶部（猫耳朵）微调后更容易贴近需要解决的问题，而底部（颜色，纹理）微调的回报太小。**

code:

D:\AI\deep-learning-with-python-notebooks-master\5.3.2-using-a-pretrained-convnet-with-fine-tuning.py 以下步骤可参考本py，运行此py需要耗时约24小时（无GPU支持）

步骤：

将用于特征提取的模型基的顶部block解冻，因为顶部是提取猫耳朵特征图，微调顶部使其特征图更贴近于要解决的问题。

创建网络模型。并先add模型基后，在add Flatten和Dense层

定义训练数据生成器和测试数据生成器，参数里rescale和数据加强（只在训练数据上）

配置训练数据生成器和测试数据生成器，参数：目标文件夹，文件尺寸，batch size，二/多分类

compile模型，参数：损失函数，优化器用RMSprop，metrics

模型训练fit_generator

模型保存model.save('save_path')

绘图：若图波折（噪音）过多，使用smooth_curve指数移动平均值来使曲线顺滑

模型评估：

配置test数据生成器，参数：test_dir, 文件尺寸，batch size，二/多分类mode

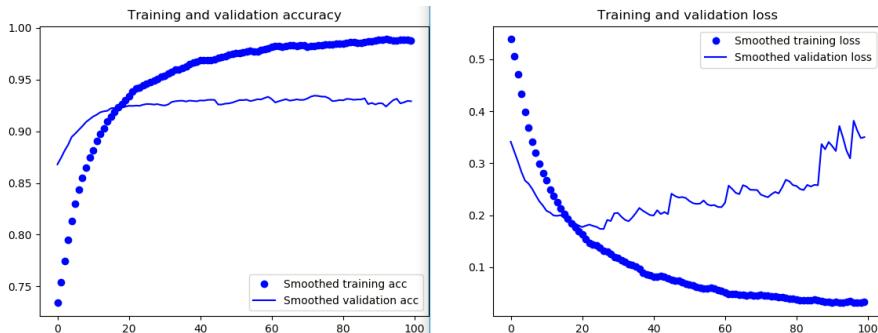
model.evaluate_generator(test_generator, steps=50)返回测试集上的loss和精度acc

注：指数移动平均值：

new point value = previous point value * factor + point value * (1 - factor)

图：

24小时（无GPU支持）运行出结果，精确度稳定在92.8%左右



小结：

卷积神经网络适用于小数据量（几千）上的训练。主要矛盾是处理过拟合问题，三法：
数据增强data augmentation, 特征提取feature extraction, 模型微调fine tuning

5.4 卷积神经网络的可视化

三法：

可视化中间激活，可视化过滤器，可视化类激活的热力图

1. 可视化中间激活：

将卷积神经网络（非dropout层，非Flatten层，非Dense层，只包含[卷积层和池化层](#)）每层的输出（即激活）可视化。

code:

D:\AI\deep-learning-with-python-notebooks-master\5.4-visualizing-what-convnets-learn.py

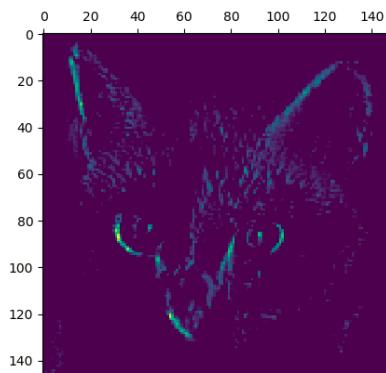
原图像可视化：

```
image.load_img(原图像路径, target_size=(150, 150))
img_to_array
pyplot.imshow(img's tensor) 详细code见5.4py的show the image
```

第4层输出可视化：

设定类models.Model的参数outputs为前8层的列表
将类models.Model实例化为activation_model,然后用此实例predict (image tensor)
predict的结果是如前定义的8个layers的输出列表，取其中一层进行matshow
注意matshow的input的最后一个参数来指定具体show哪个通道的图像。见5.4py

图：类似边缘探测器



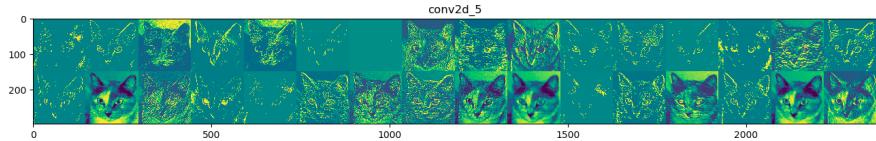
前8层所有通道可视化：

code见5.4py的'show all channels: picture of 8 layers

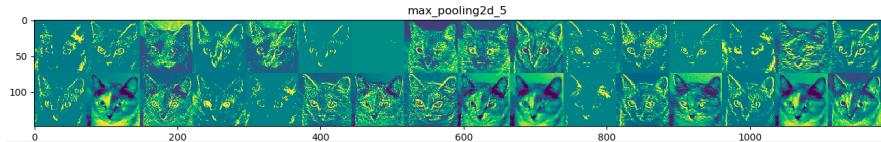
图：

第一层：conv2d_5

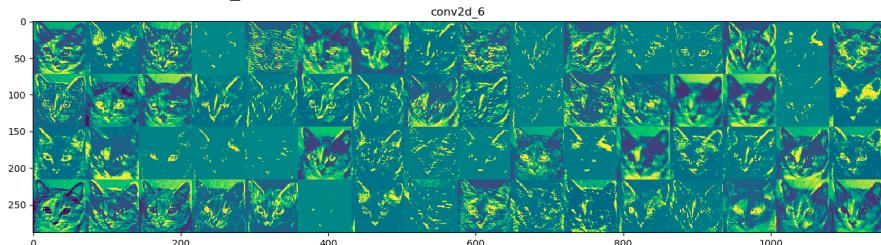
各种边缘探测器的通道集合，保留了大部分原图像信息。



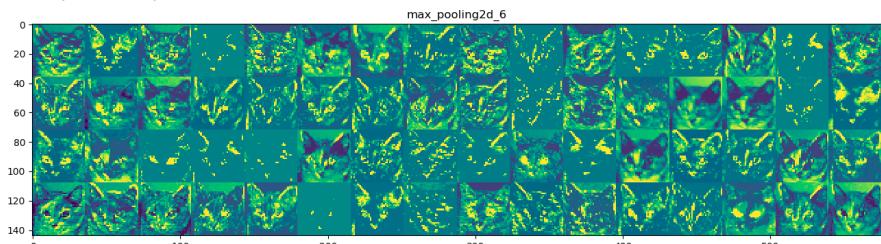
第二层：第一层的池化层



第三层： conv2d_6

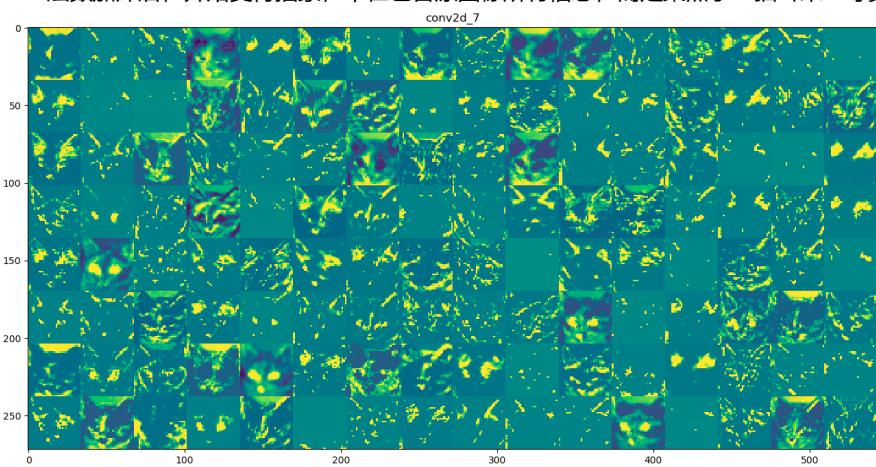


第四层：第三层的池化层

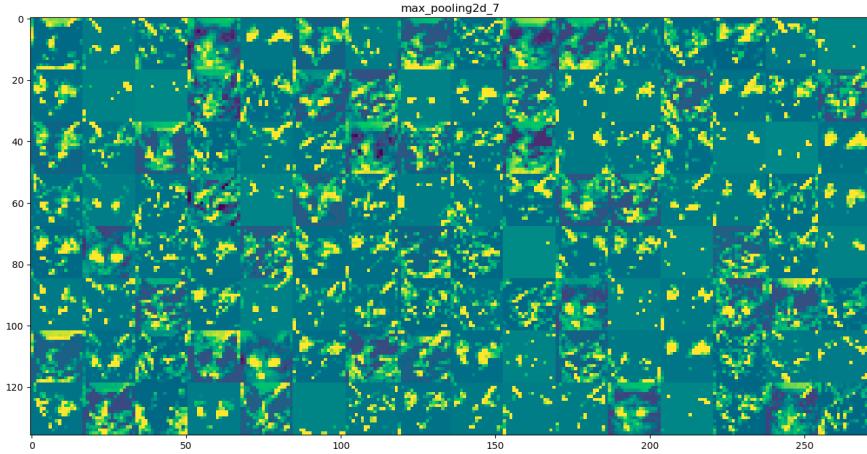


第五层： conv2d_7

层数加深后，开始变得抽象，不在包含原图像所有信息，而是聚焦于“猫耳朵”等更具体的概念

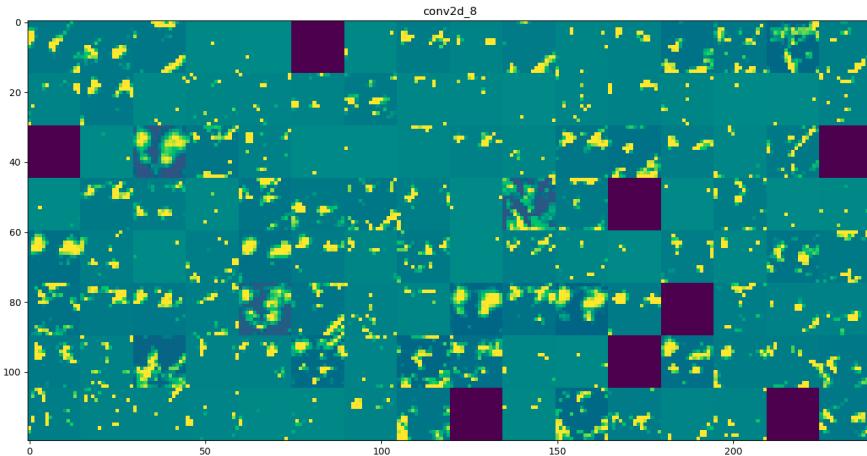


第六层：第五层的池化层

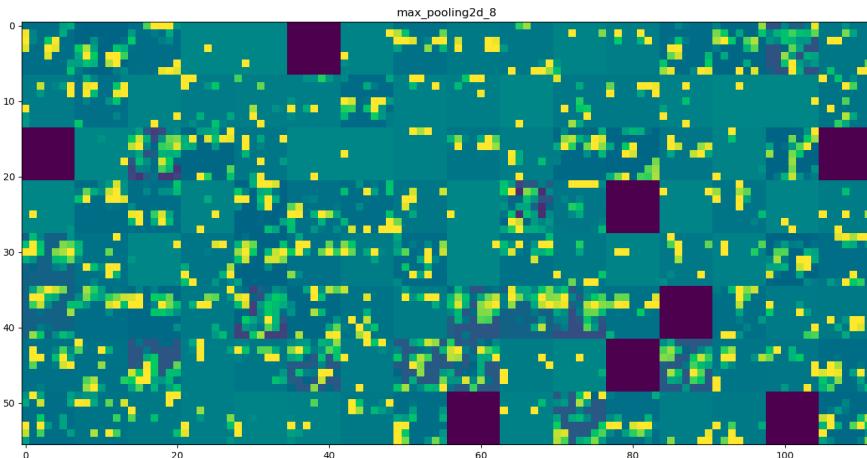


第七层：conv2d_8

grid里有很多空白图（全黑）说明输入的图像中找不到过滤器（通道）所编码的模式



第八层：第七层的池化层



以上中间激活的可视化展示完全符合P125卷积神经网络各层的分工：

code中在前面（先）定义的是卷积底层，它负责边界探测，分辨颜色，纹理等，层内存储了原图像的大部分信息。之后（晚）定义的是卷积顶层，也是我们模型微调会解冻的层，它存储了更具体的图像特征，如猫耳朵，这些更抽象信息用于目标的分类，预测等（如分辨猫狗），而无用的信息则在过程中的通道中被滤掉。这也是解冻顶层，来微调模型使其更贴近于目标的原因。

2. 可视化过滤器：

本章节的可视化较为抽象，即便是最高层卷积块的过滤器可视化图，仍难以观测。

所以以后应该不会常用到此可视化方案。故此章节未做深入学习。详见书P137

注：

本章用到了keras后台tf相关计算，如mean, gradients, square, sqrt等

本章用到了梯度相关知识，详见书P137和handson笔记，搜“梯度”

code: D:\AI\deep-learning-with-python-notebooks-master\5.4.2-visualizing-convnets-filters.py

步骤：

卷积模型可通过名称得到具体某层layer的输出，型为 (1, size, size, filter_index)

可用keras.backend.mean定义loss

可用kerasbackend.gradients(loss, model.input)来计算loss在输入张量上的梯度

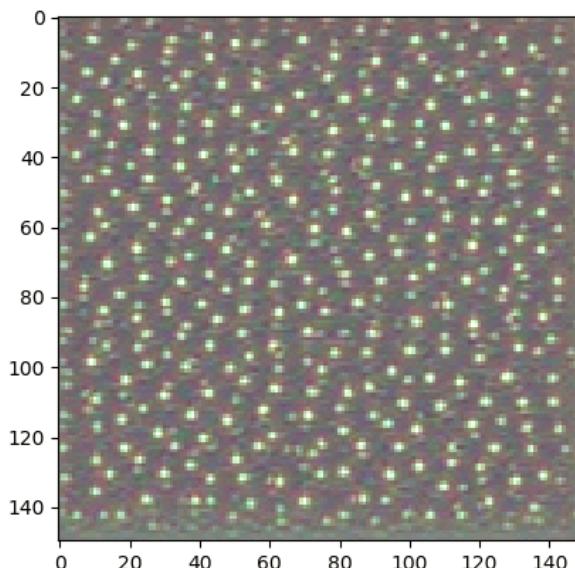
梯度标准化至[0, 1]之间

可用keras.backend.function(input, output)来实例化iterate

用iterate来计算输入是model.input时的output，即loss和grads

通过grads和步长step做40次梯度上升计算，得到绘图数据img_data

绘图如下：波尔卡点图（其实没啥用）



通常底层过滤器可视化得到的图是边缘和颜色，高层的图是羽毛，眼睛，树叶等具体纹理。

3. 可视化类激活的热力图

此可视化日后会常用到，主要用户观察每层特征图的热力图，通过热力图的热力度可以观测到图片分类和预测时用到特征图的那些部位。

code: D:\AI\deep-learning-with-python-notebooks-master\5.4.3-visualizing-heatmaps-of-class-activation.py

常用代码解释：

np.max和np.maximum: <https://blog.csdn.net/CSDN5529/article/details/79038544>

步骤：

load VGG16全模型，包括Dense层

load图片，将其转化为array

通过keras.application.vgg16的preprocess_input来预处理图片数组向量

model.predict(图片数组向量)

decode_prediction(预测结果)可得分类top5选项

利用Grad-CAM算法，得出最后一层卷积层的热力图

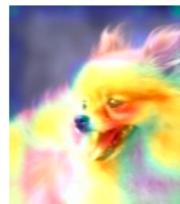
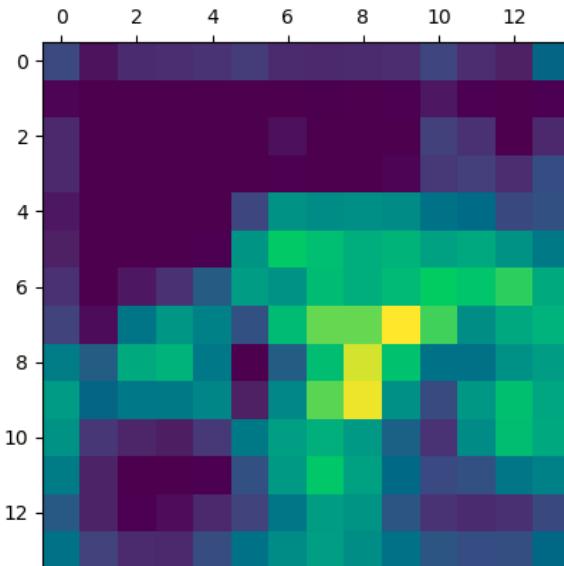
利用opencv将热力图和原图融合

绘图如下，第一张图是原图，第二张图是热力图，第三张图是合成图。

Grad-CAM算法refer to: <https://www.jianshu.com/p/1d7b5c4ecb93>

具体算法可见书P144，所有图片都可用此算法，写法固定

tf写法见： <https://github.com/Ankush96/grad-cam.tensorflow/blob/master/main.py>



Chapter 6 深度学习用于文本和序列

深度学习模型处理： 1. 文本 2. 时间序列 3. 一般序列

算法： 循环神经网络recurrent neural network 一维卷积神经网络1D convnet

用途： 书P147 主要用于语言翻译，文档相关性估测，天气预测，文档分类等

6.1 处理文本数据

处理文档时，深度学习模型也同样无法直接处理原始文本的输入，只能处理数值张量。 **(向量化)**

注：

关于词袋使用的n-gram是特征工程的工具，此处省略。见书P148-149

文本预处理：

将原始文本通过分词过程分解为单词or字符（被分解的单元叫标记token）。然后将标记token与向量相关联，最后将向量输入网络模型。

其中标记与向量相关联的方法有二： one-hot编码encoding 标记嵌入token embedding（词嵌入）

6.1.1 one-hot编码：

code：见下6.1.1py

优点：无需维护冗长的显式单词索引，节省内存

缺点：散列冲突（哈希冲突hash collision）：单词种类多余one-hot矩阵列长时发生不同单词具有相同的散列值。尽量用以下技巧：P151

```
index = abs(hash(word)) % dimensionality  
# hash: return a hash value;  
# abs: return a absolute active value;  
# %: return a remaind value in [0, max_coloumn)
```

单词和字符的one-hot编码：

code：

D:\AI\deep-learning-with-python-notebooks-master\6.1.1-one-hot-encoding-of-words-or-characters.py
和书P149-150代码解释

Keras实现单词级别的one-hot编码：

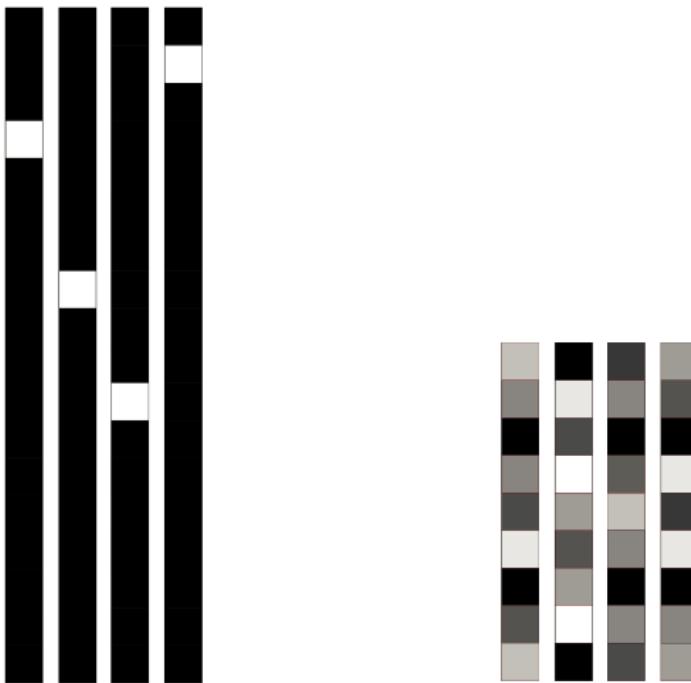
code：同上

Keras可在处理文本时将原始文本中的标点符号自动剔除，且能实现除one-hot以外的向量化模式

6.1.2 词嵌入word embedding

one-hot编码得到的是二进制，稀疏的，高维度（20000+）的向量。

词嵌入编码得到的是浮点数，密集的，低维度（256,512,1024）的向量。



One-hot word vectors:

- Sparse
- High-dimensional
- Hard-coded

词嵌入方法：

1. Embedding层学习词嵌入
2. 预训练学习词嵌入

注：

单Dense层词嵌入只能简单分类，无法学习单词间关系，所以要使用卷积模型来学习。

Word embeddings:

- Dense
- Lower-dimensional
- Learned from data

code:

D:\AI\deep-learning-with-python-notebooks-master\6.1.2-using-word-embeddings.py 以上代码只是单Dense层的Embedding词嵌入，未用到预训练模型和卷积。

需要注意的是，数据预处理时，用到的sequence.pad_sequence(训练集，评论最大长度)返回的是一个二维张量（训练集长度，评论最大长度） 书P154

6.1.3 整合：从原始文本到词嵌入

将IMDB正面及负面的review评价利用词嵌入层进行分类，为达到分类目的，使用GloVe的预训练模型（已经对每个词的weight设置好了）在embedding层设置权重weight。

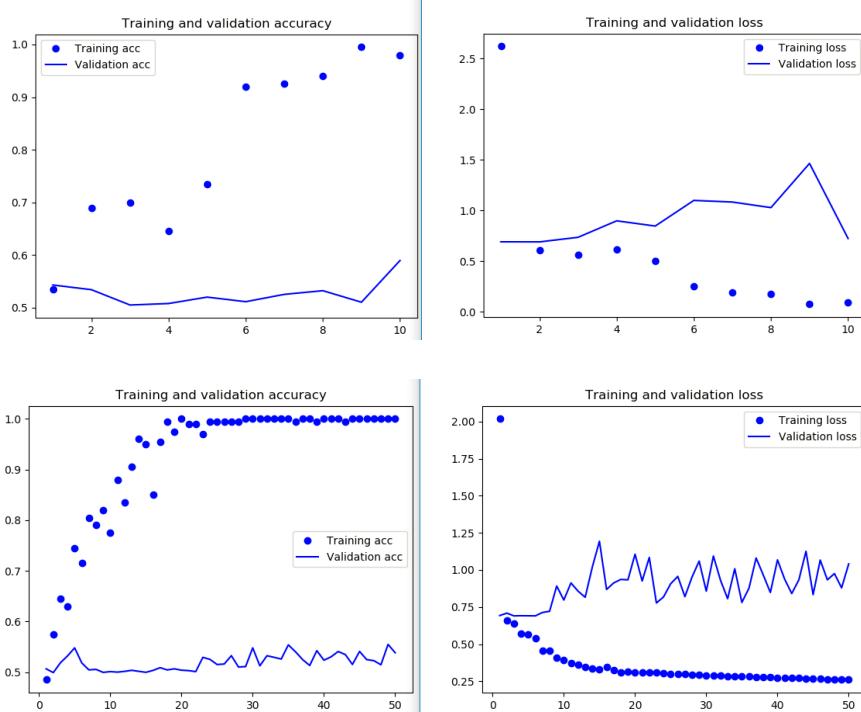
code:

D:\AI\deep-learning-with-python-notebooks-master\6.1.3-putting-it-all-together-from-raw-text-to-word-embeddings.py 代码解释参考书P156-158和py注释

步骤：

1. 将下载的IMDB文件中的正负面评论录入 texts list，其正负面label也录入label list
2. 创建分词器实例tokenizer，只保留使用最频繁的前10000个词num_words=10000
3. 使用分词器将250000条英文review转化为int形式的review sequence序列，其中评论总条数未变，但每条评论不是英文了，而是英文对应的int数字，且每条评论只保留了最频繁的前10000个词

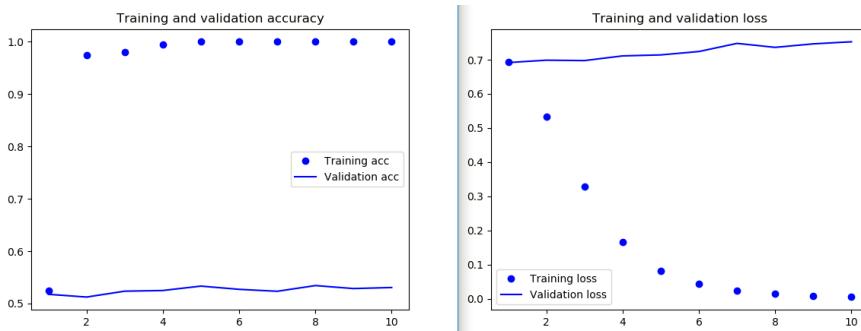
4. `pad_sequence`将评论序列截断，每条评论保留前100个单词的int，返回data二元数组array (25000, 100) 即25000条int序列，每条序列只有100个int元素。
5. `np.random.shuffle(np.arange(data.shape[0]))`来打乱所有25000条评论顺序，重新洗牌
6. 取前200个data做训练集`x_train`，接下来的10000个做validation集，`lable`也是这样。
7. 将下载的GloVe包括400000个单词的100维嵌入向量做成字典{词：100维向量array}
8. 遍历7中字典，拿出其前10000个单词的100维向量array组成一个权重矩阵
9. build network model: Embedding层，Flatten层，Dense层，Dense层
10. 在Embedding层用8的权重矩阵设置权重，并冻结此层
11. 编译compile (优化器rmsprop, 损失binary_crossentropy, acc)，训练fit，保存模型。
12. 绘图如下。上图训练10轮，下图训练50轮。



validation的精度acc只有50%+，这非常低。由于train训练集样本非常少，只有200个，所有一开始就过拟合了。

其他测试：

1. 在Embedding层不添加GloVe预训练模型的权重矩阵，直接用200个样本训练，10000个样本validation，得到的精度更低。如下图：



可见，使用预训练模型，可以提高精度！

2. 重新load使用了预训练模型加权的h5保存模型，然后用测试集test来评估`model.evaluate(x_test, y_test)`模型，acc达到56.7% 所以预训练模型的加权功不可没。

6.2 理解循环神经网络

前馈网络：

之前设计的所有卷积网络及其相关代码全部都是一次性处理完所有数据，层与层之间无循环。

循环神经网络RNN：

初始状态是0，遍历所有元素，保存状态作为下一时刻的输入，依次循环，并将每次状态都做保存。最后输出一个全序列的状态信息表。如图P164

6.2.1 Keras中的循环层SimpleRNN

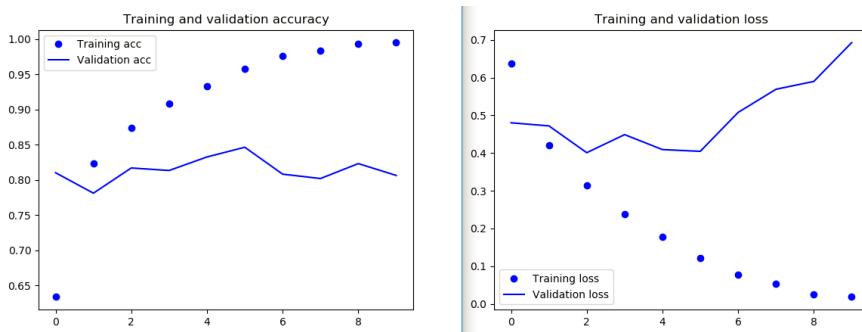
SimpleRNN不擅长处理长序列，比如文本。所以下面处理IMDB评论时，将每个评论单词在500时做了截断，最后导致了精确度只有80%。这个结果都不如P58用的简单模型做的预测结果88%

SimpleRNN通过参数return_sequence来控制是否输出每个循环的状态，还是只输出最终结果。P165

用SimpleRNN来分类IMDB评论：

code:

D:\AI\deep-learning-with-python-notebooks-master\6.2.1-a-first-recurrent-layer-in-keras.py



6.2.2 理解LSTM层和GRU层

LSTM: long short term memory 长短期记忆

通俗解释： P168

在RNN层中添加了新的“携带轨道C”，此轨道携带了各个时间点的状态信息 c_t ，（下一时间点的状态信息 c_{t+1} 需要通过公式计算P169）。此轨道贯穿整个LSTM层，允许过去的信息稍后重新注入，从而解决梯度消失问题。

6.2.3 Keras中的一个LSTM例子

code:

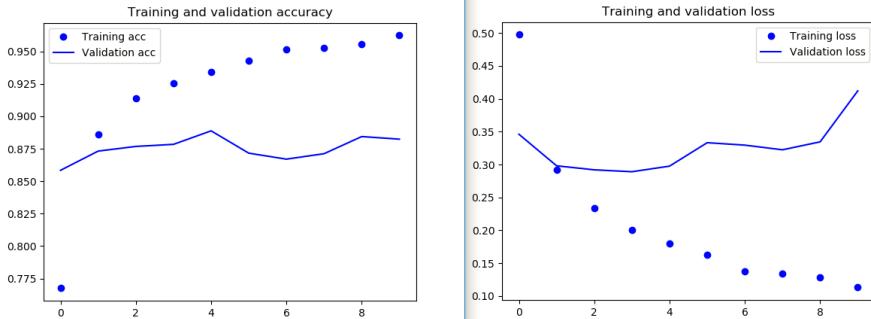
D:\AI\deep-learning-with-python-notebooks-master\6.2.3-a-concrete-LSTM-example-in-Keras.py

代码很简单，就是将之前的simpleRNN换成LSTM层。

书P171的fit中，用到了validation_split=0.2参数，此参数的含义是将导入fit的train数据，划分出20%用来做验证validation。

fit函数中还有个参数是shuffle，它的作用是打乱数据。**但注意**：fit先执行validation_split后再执行shuffle。所以若数据顺序很强，且顺序影响训练，这时需要将数据进行手动reshuffle，然后再将数据导入fit中，只用validation_split，不用shuffle了。

LSTM结果如图：



大概在第五轮后开始过拟合，过拟合之前精确度可以达到87.5%+
这个精度比simpleRNN和第三章P58要好，而且用的数据更少（在评论500单词处做了截断）

6.3 循环神经网络的高级用法

高级用法：

循环dropout (recurrent dropout)：内置

堆叠循环层 (stacking recurrent layers)：计算代价高

双向循环层 (bidirectional recurrent layers)：提高精度，缓解遗忘

6.3.1 温度预测

boston房价问题：(样本数, features) 2D, 向量数据

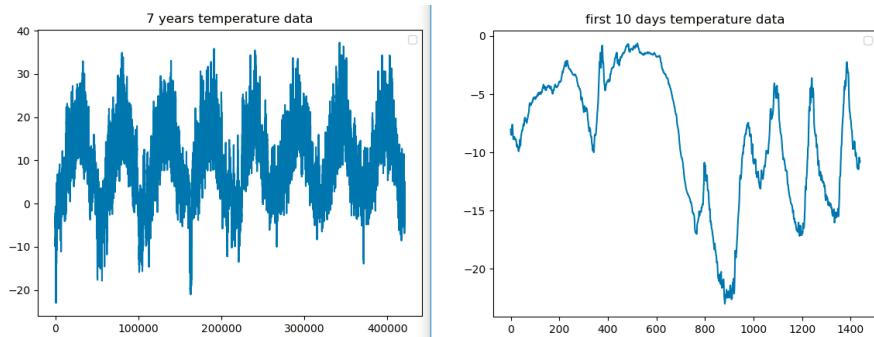
IMDB评论二分类，路透社新闻多分类：(样本数, [one hot二维array]) 3D, 序列数据

温度预测：(样本数, time stamps, features) 3D, 时间序列

CSV文件处理：P173 代码见6.3py

pyplot (plt)：P173 代码见6.3py

读取csv文件后，展示七年和前十天温度数据如下图：



6.3.2 数据准备

标准化：每项数据在不同的取值范围内，需要减去平均值mean后再除以标准差，来做标准化。

生成器generator：输入多种数据，冗余度高，需要采样后输出采样样本和某种数据。P176

注：书上P176的生成器def代码在6.3py中已经优化，参考py为主。

6.3.3 非机器学习范畴的基准方法

假设每天温度都相同，则计算数据中前一天的温度均值和当天温度均值的绝对差，若温差较大，则此基准方法不准。

6.3.4 简单机器学习的方法Dense

根据温度的历史数据来预测之后的温度，此问题类似于boston房价预测，属于**回归问题regression**

回归问题可以见3.6.py

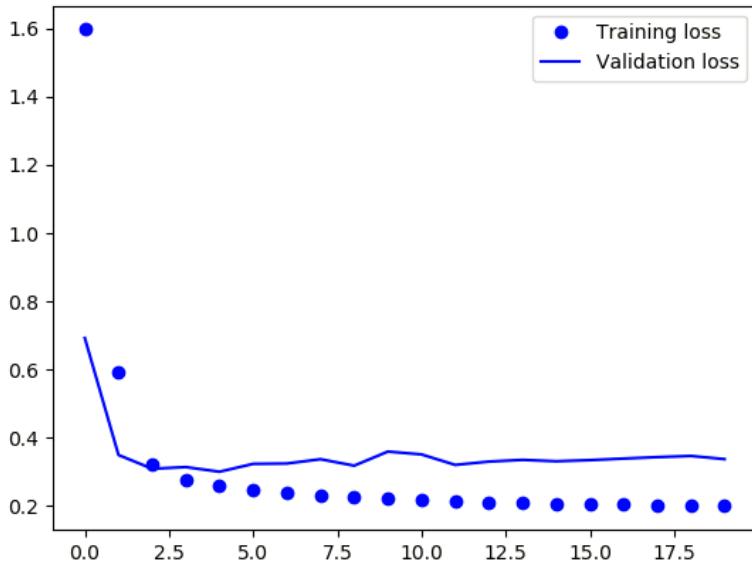
注意点：

密集连接层最后一层layer没有激活函数activation

编译model时用的loss函数可以选择mae或者mse，具体情况参考本笔记最前的“常用链接”

loss: 0.2004 - val_loss: 0.3380 第五轮开始出现过拟合，结果不可靠

Training and validation loss



6.3.5 循环网络

上一小节全连接将时间序列展平的同时消除了时间概念，本节GRU层可以利用数据点的时间顺序来处理循环序列。

门控循环单元GRU gated recurrent unit

计算代价低，但表示能力不如LSTM。本书并未讲解GRU原理，需自学

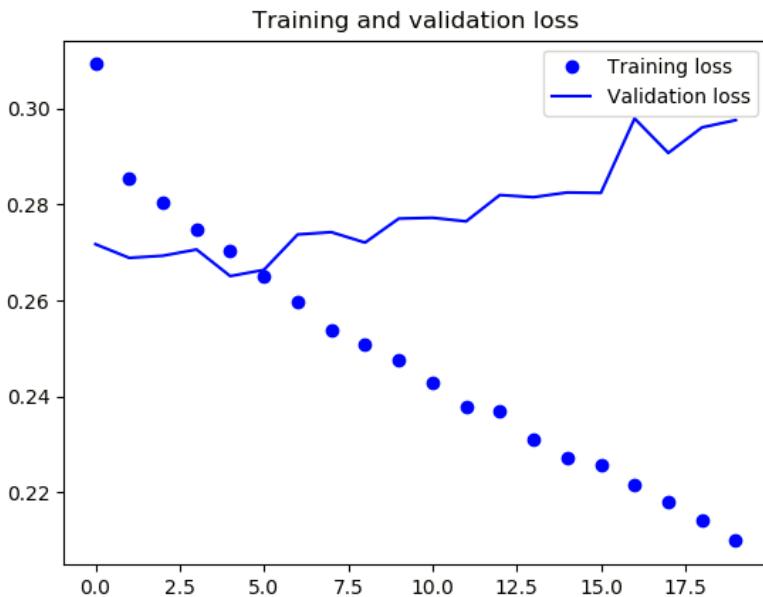
LSTM和GRU讲解：<https://blog.csdn.net/dqcfkyqdxym3f8rb0/article/details/82922386>

code: D:\AI\deep-learning-with-python-notebooks-master\6.3.5-a-first-recurrent-baseline.py

TODO: GRU层的input_shape如何解释？为什么是 (None, features)

结果如下图：第七轮开始出现过拟合，如何降低GRU循环层中过拟合，见下一小节。

loss: 0.2100 - val_loss: 0.2975 结果稍微好一点



6.3.6 用循环dropout降低过拟合

GRU层存在两个参数来降低过拟合：

dropout: float, 指定本层dropout比例

recurrent_dropout: 指定循环单元的dropout比例

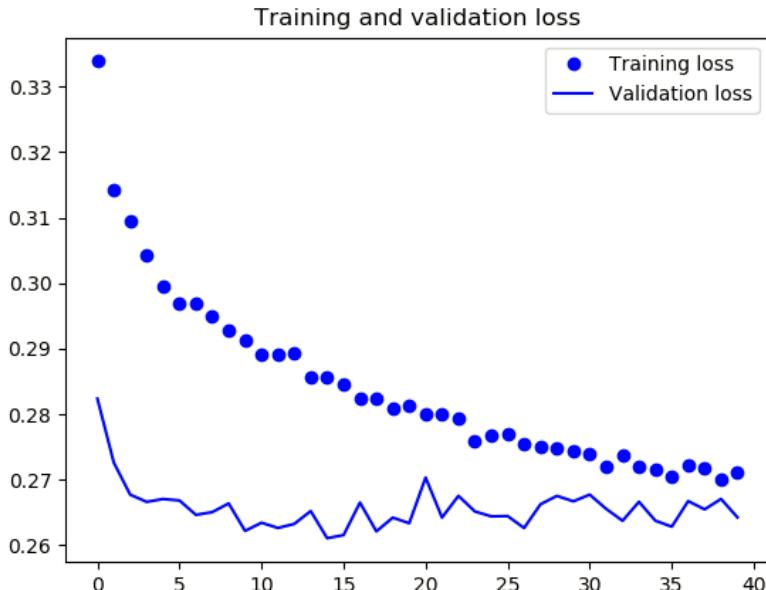
因为dropout正则化需要更久才收敛，所以训练轮次*2 (epochs*2=40)

code:

D:\AI\deep-learning-with-python-notebooks-master\6.3.6-recurrent-dropout-to-fight-overfitting.py

结果如图：40epochs, 过拟合（点和线分离）不再明显

loss: 0.2712 - val_loss: 0.2643, loss稍微降低



6.3.7 循环层堆叠 stacking recurrent layers

网络模型**性能不足**，若此时过拟合不再是问题，则需要考虑增加网络容量。

增加网络容量：

增加每层单元数： 32->64

增加层数： 添加了一层GRU

注：

堆叠循环层时，非最后GRU层应该返回3D张量，用到了GRU的参数return_sequence=True
最后一GRU层不用，只需要输出一个时间步，然后交给Dense处理。

code:

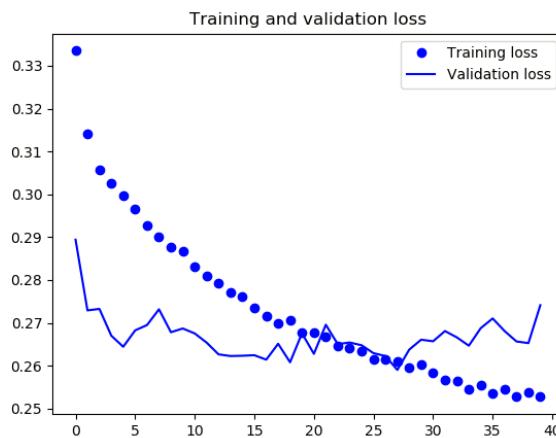
D:\AI\deep-learning-with-python-notebooks-master\6.3.7-stacking-recurrent-layers.py

结果如图：

随着层数和单元数的增加，网络性能增加，loss减少，但最后还是出现了点线分离的过拟合。

就结果看，针对这个case，这种解决方式并不理想，甚至不如6.3.6的循环dropout。

loss: 0.2554 - val_loss: 0.2792



6.3.8 双向RNN bidirectional RNNs

双向RNN：（见P187总结！）

常用于处理自然语言问题。

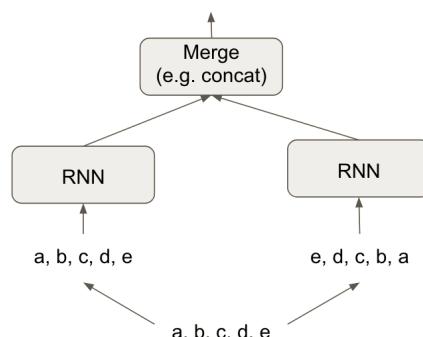
顺序很重要，对顺序依赖很强的问题，用双向RNN的结果很好。

正反（逆）两向可以捕捉到单向RNN忽略的模式。

判断一个问题是否对顺序依赖：

以正序和逆序进行了两次fit，来对比结果，若正序结果比逆序好很多，则问题对正序敏感！对逆序并不敏感。若只对正序敏感，则不用考虑双向RNN。反之亦然，只对逆序敏感，也不用RNN。如下文温度预测问题。

若正逆序结果都很好，则可考虑用双向RNN！因为双向RNN可以捕捉到单向循环难以捕捉的模式来集成（第七章）。如下列文本问题。

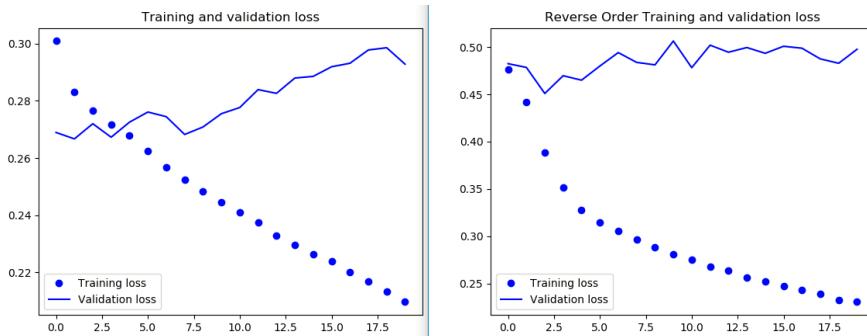


正逆序GRU对比：温度预测问题P184

code: 6.3.5.py

如下图所示，对比显示了之前用GRU循环层预测温度的损失。loss: 0.2304 - val_loss: 0.4977

图二逆序的loss要比图一正序高很多，可见时间顺序对温度预测的重要性。所以可以不考虑用双向RNN来解决此问题。



正逆序GRU对比：处理文本问题P185

此问题无py代码编写。

结果：

正序和逆序的处理结果同样好，并不是顺序对处理文本不重要，而是正序和逆序对处理文本问题同样重要。可以考虑用**双向RNN**。

双向RNN：处理文本问题P186

code: 6.2.3.py

py内新增了bi_direction_model模型训练双向RNN。模型区别仅一处，将LSTM层嵌入Bidirectional层中。

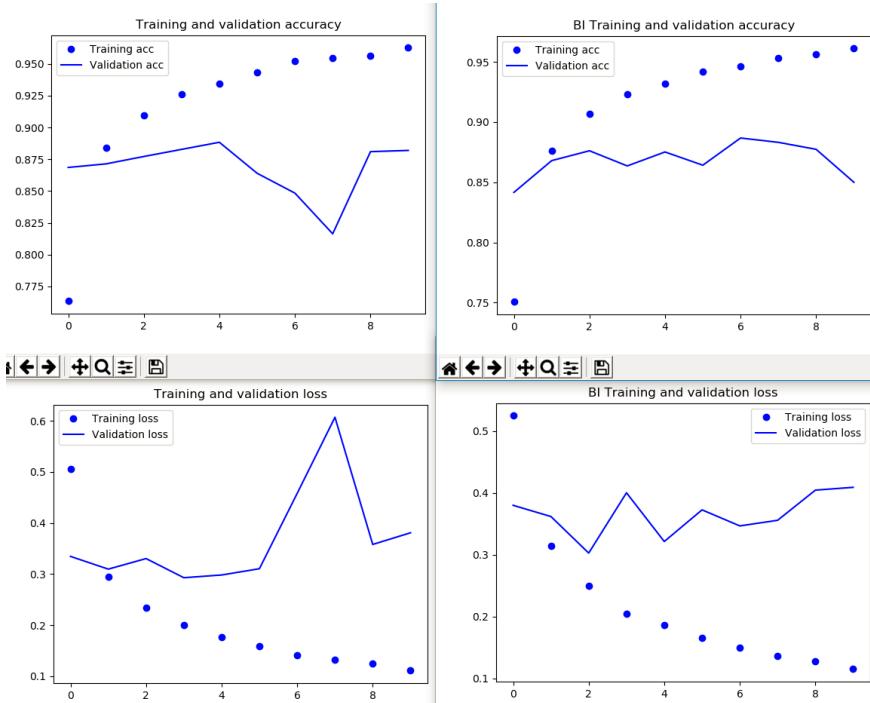
结果如图：

左侧单向LSTM结果： loss: 0.1253 - acc: 0.9567 - val_loss: 0.3582 - val_acc: 0.8810

右侧双向RNN结果： loss: 0.1144 - acc: 0.9590 - val_loss: 0.5002 - val_acc: 0.8712

两种解决方案都存在过早过拟合的状况发生（左第四轮，右第二轮）。导致结果不可信，但就过拟合之前的精度来看，正向LSTM和双向RNN精度差不多，都可达到88%左右。

所以**双向RNN网络模型更容易过拟合**，因为它的参数是单向的**2倍**。**正则化**后表现会好很多。



双向RNN：温度预测问题P187

code: 6.3.8.py

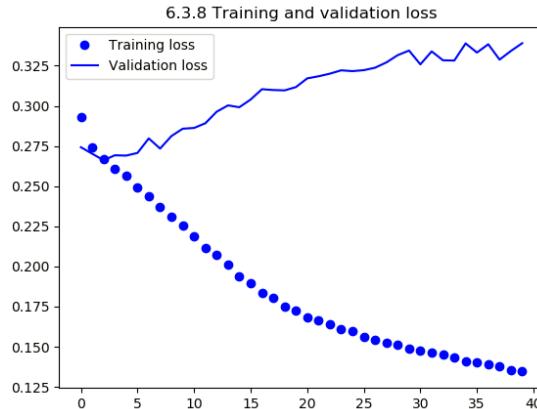
此py与6.3.5.py的区别就是将GRU层嵌入了bi_directional层中，使其变为双向RNN。且GRU层未使用循环dropout，未增加GRU层or隐藏单元。

结果如图：

loss: 0.1148 - acc: 0.9611 - val_loss: 0.4092 - val_acc: 0.8500

从结果看，此结果和6.3.5py结果类似，且过早的出现了过拟合。（2倍参数，逆向无有用模式）

双向RNN并不适用于正序敏感的温度预测问题。



6.3.9 模型优化方向

调整每层layer的隐藏单元个数。 6.3.7py已调整

调整RMSprop优化器的学习率。 P127 5.3.1py

LSTM替换GRU: <https://blog.csdn.net/dqcfkyqdxym3f8rb0/article/details/82922386>

调整Dense和堆叠Dense层： 4.4py

利用test数据集评估模型model.evaluate (x_test, y_test) : 6.1.3 py

注：

每个问题都是独一无二的，没用通用的模型，每个问题的最佳模型都是不断尝试和调整出来的！

TODO:

循环注意recurrent attention

序列掩码sequence masking

6.4 卷积神经网络处理序列

6.4.1 理解序列数据的一维卷积

和图像问题一样，一维卷积可以处理序列问题（文本）。

字符级别的一维卷积可以将构词法学习成为一种局部模式。（构词法见P189）

6.4.2 序列数据一维池化

和二维池化（最大池化，平均池化）相类似，下采样来降低输入长度。

常用的有mean-pooling, max-pooling和Stochastic-pooling三种。

mean-pooling，即对邻域内特征点只求平均，保留更多背景信息。

max-pooling，即对邻域内特征点取最大，保留更多纹理信息。

6.4.3 一维卷积神经网络

接受的input形状也是（样本量， time， features）三维张量。

卷积窗口：

卷积层的权重矩阵W，也称卷积核。

一维卷积神经网络架构：

Conv1D+Maxpooling1D: 提取特征

全局池化层 or Flatten层: 3D -> 2D

Dense层: 分类 or 回归

code:

6.4.3-implementing-a-1d-convnet.py

其中:

一维卷积层`layers.Conv1D(32, 7, activation='relu')`参数不再是(2,2)或(3,3)而是7, 表示层内权重矩阵包含7个特征向量。

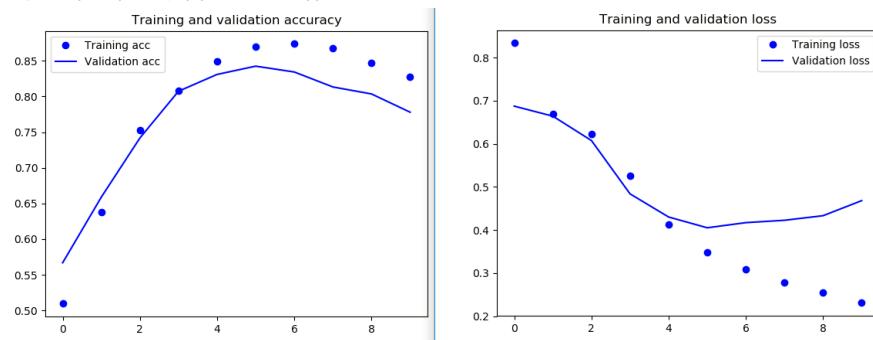
池化层`layers.MaxPooling1D(5)`参数也不再是(2,2)的half二选一, 而是5, 即: 5选1

结果如下图:

loss: 0.2309 - acc: 0.8273 - val_loss: 0.4682 - val_acc: 0.7780

精度低于LSTM的88%, 但计算速度更快, 计算代价更低。

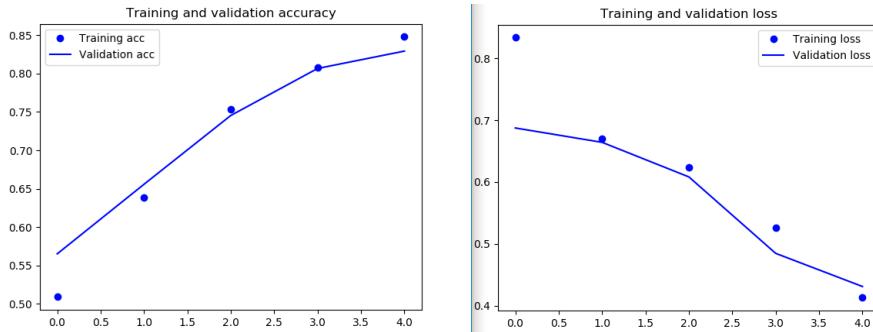
在情感分类任务上, 一维卷积可以替代循环网络。



大概在第5轮过拟合, 重新训练5轮模型, 结果如下:

loss: 0.4127 - acc: 0.8478 - val_loss: 0.4308 - val_acc: 0.8292

将一维卷积层7->9, 池化层5->3后, acc可提高至84%, 但计算速度下降。仍然不如LSTM的88%



6.4.4 结合CNN (卷积神经网络) 和RNN (循环神经网络) 来处理长序列

CNN和RNN区别: https://blog.csdn.net/weixin_42137700/article/details/83241774

一维卷积ConV1D和二维卷积ConV2D都属于CNN卷积神经网络。

书本内容提炼: P191-193

- 根据6.4.3的结果来看, 类似于对IMDB评论进行情感分类的问题, 对顺序并不敏感, 所以用一维卷积网络模型+Dense来处理比较合适。精确度虽略低于LSTM模型, 但速度快很多。
- 但一维卷积模型对处理“温度预测”这类的**时间顺序敏感问题**, 效果就差很多了, 还是用**RNN**好。
- 但对于**“长序列”**数据, 一维卷积对顺序不敏感, 效果不好; 但序列过长, RNN也无法处理, 所以使用**一维CNN+RNN**来处理, 其中CNN来做数据预处理, 提取特征, 剪短序列长度。如图P193

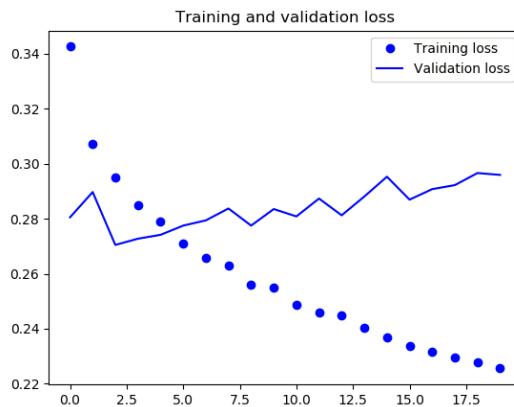
code:

6.4.4-combining-CNNs-and-RNNs-to-process-long-sequences.py

结果如下图：

loss: 0.2256 - val_loss: 0.2959

效果并不如单用GRU好，但是模型输入了两倍的数据量，且计算速度更快。



小结：

CNN一维卷积： ConV1D ---- 处理文本序列（顺序不敏感问题）

CNN二维卷积： ConV2D ---- 处理图像识别

RNN循环网络： LSTM, GRU, 双向RNN（正逆） ---- 处理顺序敏感序列（短序列），自然语言

CNN一维卷积+RNN ---- 处理长序列（上千时间步）序列

未来用途：

RNN: 顺序序列未来趋势预测，分类，序列异常检测，序列标记（找出句子中人名，日期）

CNN 一维： 机器翻译（序列to序列的卷积模型），文档分类，拼写校正。

Chapter 7 高级深度学习实践

7.1 不用Sequential模型解决问题： Keras函数式API

之前的模型都是Sequential模型实现，无法处理多输入，多输出，残差连接，内部多分支的类层图等问题。

可利用函数式API解决。

7.1.1 函数式API

之前一直用的Sequential()来build网络模型model，然后编译，训练……这样只能应付单输入，单输出的问题。

现在摆脱Sequential()，利用keras的Input，Dense，Model等提供的API，仍然可以将输入和输出张量所在的所有层组成一个类图的模型model。

code: P199-200

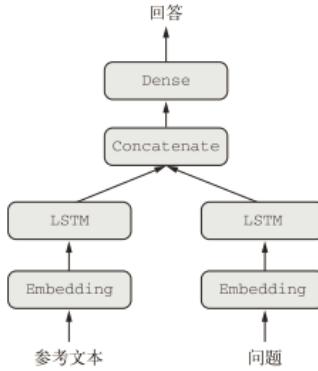
D:\AI\deep-learning-with-python-notebooks-master\7.1.1-functional-API-introduction.py

注：

若类Model()中的input张量和output张量不相关，则code会报错run time error. 详见7.1.1py

7.1.2 多输入模型

如下图，多输入（问题输入1，参考文本输入2），单输出（回答）



多输入和多输出模型用到了keras.layers.add（相加）和layers.concatenate（相连）方法。

code：书P201-202和code中都有详细解释

D:\AI\deep-learning-with-python-notebooks-master\7.1.2-multi-input-model.py

注：

layers.concatenate([input_tensor_1, input_tensor_2], axis=)可以将两个不同维度的张量相连

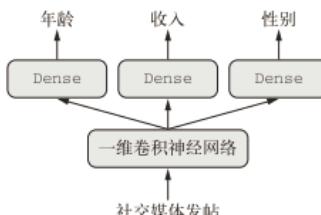
Model(input, output)中，若输入，输出张量不是单个张量，可以用list或tuple来做多输入，多输出

model.fit(train_data, train_label, ...)若同样是多输入张量，也可以用list或tuple

model.fit有两种方法，第一种之前常用（若multi，则list或tuple）。第二种在输入layer有unique名字时，可以映射成字典来输入。P202 fit({name1: tensor1, name2: tensor2}, lable, epochs, ...)

7.1.3 多输出模型

单输入（帖子信息输入），多输出（预测的发帖人年龄输出1，收入输出2，性别输出3）



code：代码中有详细解释

D:\AI\deep-learning-with-python-notebooks-master\7.1.3-multi-output-model.py

compile：

多输出模型，可以为每种输出指定不同的loss函数，在compile里将loss写成list或者dict形式。

注：

list中各输出loss的顺序要按照之前个输出output layer的定义顺序填写。

dict中各输出loss的name是之前output layer中定义的name，所以要unique

不同的loss，在compile中可以指定不同的loss weight，见以下fit。

fit：

最后model模型会将各loss相加，得到全局loss，fit的过程就是将这个全局loss最小化的过程。

严重的不平衡的loss会让model忽略其他loss的优化，所以分配loss权重解决此不平衡loss问题。

推测：compile中权重的分配值是此值乘以loss值后得1即可，而不是所有权重值相加得1。

即，每个weight值乘以相应的loss估计值后得1即可。

loss估计值参考：MSE: 3~5 Binary_crossentropy: 0.1 Categorical_crossentropy: 1

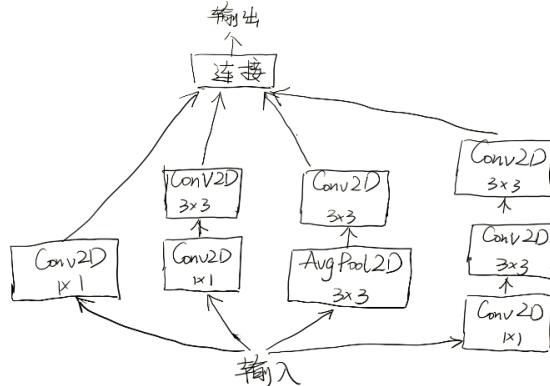
所以书中的weight list: [0.25, 1, 10] 以上红字为推测，需要后续核实。

7.1.4 层组成的有向无环图

无环： 图内没有循环，但循环层的内部循环除外。

Inception模块：

包含3~4个并行分支，其中可能会包含：1x1卷积（逐点卷积，即非空间卷积），空间卷积，池化运算



注：

其中逐点卷积不会将跨空间的信息混合，区分了通道特征学习和空间特征学习，让相关性不高的不同通道间信息隔离。

code:

D:\AI\deep-learning-with-python-notebooks-master\7.1.4-directed-acyclic-graph-of-layers.py

注：

Inception代码中共4个branches，每个分支的步幅strides要求一样，最后才可以concatenate

Xception：

Xception是Inception的极端，其参数个数和Inception V3大致相同，但性能更高效，精度更高。

残差相连：

类似于循环层LSTM引入携带轨道（平行于主轨道）来传播信息保存梯度。残差相连引入了纯线性轨道（特征图尺寸相同，相加）来携带信息，帮助任意深度的层来传播梯度，解决了梯度消失问题

由此，残差相连解决了**梯度消失 (P207)** 问题。同时也解决了表示**瓶颈问题(P207)**，表示瓶颈通常是由于中间某层太小，导致其激活所携带信息过少，在此层丢失的信息，下游的层无法恢复。利用残差相连可以将之前层的信息与后面层的激活相加，防止了信息丢失。

两种情况下的残差相连：

1. 前面层的输出（激活）的特征图尺寸与后面层的激活特征图尺寸相同，

则用恒等残差相连identity residual connection

2. 尺寸不同，用线性残差相连linear residual connection

code:

D:\AI\deep-learning-with-python-notebooks-master\7.1.4-directed-acyclic-graph-of-layers.py

代码解释见py和书P207

注：

表示瓶颈

梯度消失 P207 和此笔记上文7.1.4红字

7.1.5 共享层权重

实例化一个层，如lstm=layer.LSTM(32)。则之后调用这个实力lstm，就可以使用此LSTM层的权重
这个lstm实例成为连体LSTM或共享LSTM模型。

为这个共享LSTM模型输入不同的input，即共享了此模型的权重。此实例可多次调用！

code:

```
D:\AI\deep-learning-with-python-notebooks-master\7.1.5-shared-layer-weight.py
```

7.1.6 将模型作为层

把Model类的实例model作为层来使用，给其input，输出output。如
y1, y2 = model([x1, x2]) P209 多输入多输出model，作为层来用
和7.1.5相同，只是共享model的权重（学习表示），**只利用，并不修改**

code:

```
D:\AI\deep-learning-with-python-notebooks-master\7.1.6-model-as-layer.py
```

7.2 使用Keras回调函数和TensorBoard来检查并监控深度学习模型

7.2.1 训练过程中将回调函数作用于模型

在model.fit()中添加参数callbacks，此参数可以提前定义好，多参时用list作为参数input

目前常用的Keras自带回调函数：

EarlyStopping(monitor='acc', patience=1):

解释：fit在2轮(patience参数+1)内，若acc(monitor参数)没有改善，则终止训练。

<https://blog.csdn.net/zwqjoy/article/details/86677030> 解释的很详细，一定要看！

作用：根据监控项可提前终止fit过程。

ModelCheckpoint(filepath, monitor='val_loss', save_best_only=True):

解释：在fit中，若val_loss(monitor参数)有改善（提升），则保存当前训练模型h5文件。所以可以一直保存最佳模型
(save_best_only=True)

<https://blog.csdn.net/jieshaoxiansen/article/details/82762922>

作用：根据监控项来保存最佳模型h5文件

ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10):

解释：在fit中，若val_loss(monitor参数)在10轮(patience参数)内没有改善，则触发此回调，学习率乘以0.1(factor参数)

<https://www.cnblogs.com/jsxyhelu/p/9251101.html> 内含“降低学习率的作用”，很重要

作用：根据监控项来修改优化器内学习率lr参数的值

code:

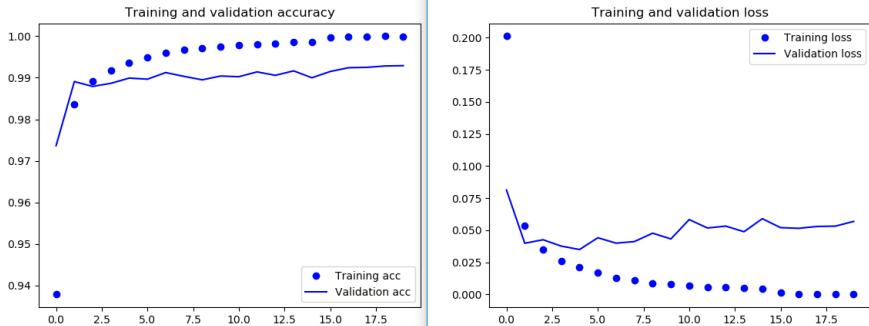
```
D:\AI\deep-learning-with-python-notebooks-master\7.2.1-model-callbacks-in-training-process.py
```

运行结果：

从如下图中看，acc为出现在2轮内不提升的情况，所以earlystopping未被触发。

val_loss在第4轮训练最佳，所以最佳模型h5文件被保存，modelcheckpoint被触发。

val_loss没有10轮内无改善的情况，所以ReduceLROnPlateau未被触发。



注:

回调函数内，凡是涉及到监控val数据的，在fit()内必须添加validation_data参数，当然用validation_split也可以。

编写自己的回调函数：

自己写回调类，需要继承keras.callbacks.Callback类

类中定义如on_epoch_begin等函数，fit会自动识别。书P212

如上fit自动识别的6个函数中都有log参数，字典，保存前一train, epoch, batch信息。

自己的回调类可以访问当前model的属性，如model.layers

自己的回调类可以访问当前fit中validation_data

code:

D:\AI\deep-learning-with-python-notebooks-master\7.2.1-model-callbacks-in-training-process.py 此py中在每epoch后写一个txt，已经可以被触发

7.2.2 可视化框架：TensorBoard

单纯的每次将训练过程和结果通过plot展示，太过于简单。

使用TensorBoard可以可视化：

1. 训练过程中的监控指标
2. 模型架构（层图，层shape）
3. 激活和梯度的直方图
4. 二维、三维形式展示嵌入内容

code:

D:\AI\deep-learning-with-python-notebooks-master\7.2.2-tensor-board.py

TensorBoard使用方法：

1. 在工作目录创建TensorBoard存放展示数据用的目录，如：
D:\AI\deep-learning-with-python-notebooks-master\my_log_dir
2. 在py中添加callbacks=keras.callbacks.TensorBoard(log_dir刚刚指定的存放目录, histogram_freq激活直方图/轮（频率）, embeddings_freq记录嵌入数据/轮（频率）)
3. model.fit()中添加此callbacks
4. 在工作目录下的cmd中输入：tensorboard --logdir=my_log_dir
5. 运行py，运行完毕后，在浏览器查看TensorBoard监控页面：<http://localhost:6006>

TensorBoard功能：

SCALARS页面展示监控指标图，如acc, loss曲线图，有这个，以后不用plot了

HISTOGRAMS页面展示每一层每一轮的激活值的直方图

EMBEDDINGS页面展示输入的文本的嵌入位置和空间关系，二维or三维展示

GRAPHS页面展示model模型底层TensorFlow的交互式层图，大多和梯度下降有关

除了TensorBoard外，可用plot_model来画简易层图：（并不好用，不如用TensorBoard）

先按照pip3 install pydot, pydot-ng, graphviz三个第三方库，其中graphviz的安装如下：

<https://www.cnblogs.com/shuodehaoa/p/8667045.html> 需要改系统path参数

然后from keras.utils import plot_model

plot_model(model, to_file='model.png', show_shapes=True) 画出模型层图 P218

模型层图：

7.3 让模型性能发挥到极致

本小节无code演示内容（前沿领域，参数调优or模型集成，代码庞大，无法演示）

7.3.1 高级架构模式

批标准化：

不仅在输入录入网络模型前需要做标准化，如下：

```
normalized_data = (data - np.mean) / np.std
```

在网络模型层与层之间也可以添加标准化层来帮助梯度传播。

通常在卷积层or密集连接层之后使用批标准化层。用法如下：

```
layer.BatchNormalization(axis=-1) 参数通常是-1，也有例外：P220
```

深度可分离卷积：

替代Conv2D，速度更快，性能更好，更适用于小数据集训练。

用法如下：

```
layer.SeparableConv2D(参数同Conv2D层) P221
```

深度可分离卷积是Xception架构的基础，内置Keras中。

7.3.2 超参数优化

架构在层面的参数都可以叫做超参，如层，层的dropout率，激活函数，隐藏单元个数等等。

为了使模型最佳，超参的调节和优化没有成文的规定，只有不停的尝试和改进模型。

超参的优化过程：

选择一组超参，建模，训练，验证集上验证模型性能，然后再选一组超参，重复上述步骤。

这个步骤实际上是在验证集上调整超参，所以一定要注意**验证集过拟合**问题！用callbacks。

超参的优化工具：

Hyperopt, Hyperas (此库已和Keras集成)

超参的自动优化：

前沿领域，现在还不成熟。因为超参是离散的，不能用梯度下降，且验证超参时的计算代价很大（需要重新训练模型）

7.3.3 模型集合

将不同模型的预测结果汇集到一起，会打败任何一个高性能模型。

因为：**多样性就是力量！**

模型集合示例：

分类器集成：

1. 将多种分类器的预测结果总和取平均值作为预测结果
2. 为每种分类器加权后，取其所有结果总和的平均值
3. 集成时不要用同种分类器，没有意义，要保持多样性。
4. 好的模型（分类器）就加权值高，不好的就低，保证多样性，效果更好！

新集成模型——宽且深：

深度学习结合浅层学习。训练一个深度神经网络和一个大型的线性模型，然后联合训练。

Chapter 8 生成式深度学习

8.1 使用LSTM生成文本

Google2016年的SmartReply使用了LSTM生成文本的技术

8.1.1 生成式循环网络的简史

最先用于生成音乐 -- 2002年

8.1.2 如何生成序列数据

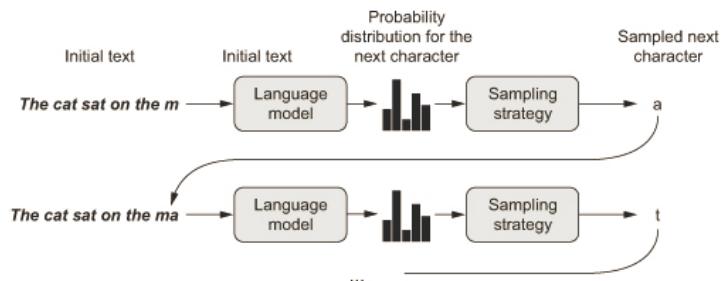
字符集神经语言模型：

训练一个网络模型，给定其一个输入，如 “the cat is on the ma” 让模型预测输入的最后一个字符是什么（应该是字母t）。

模型训练好后，给模型一个初始字符串（条件数据）input，然后要求模型输出下一个字符或者单词。然后将生成的字符或者单词重新加入到input中，使其再次输出。重复此过程。

此过程也叫采样过程：模型可以捕捉到语言结构的潜在空间，从而生成新的字符或单词。

输出过程是对所有字符or单词做softmax多分类，得到下一个字符或单词的概率分布。



8.1.3 采样策略的重要性

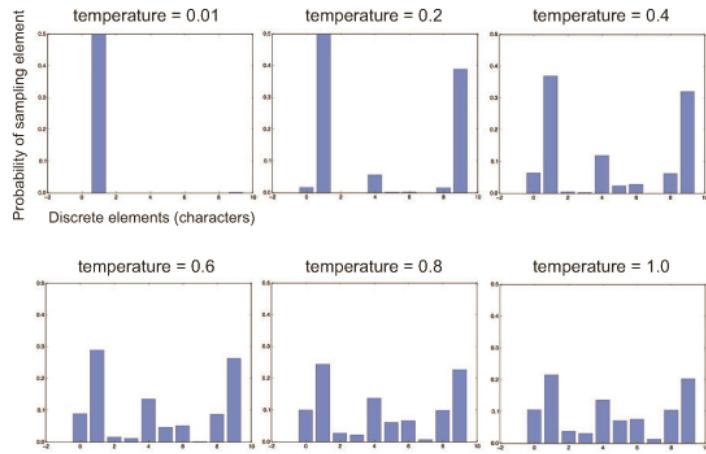
上小节提及的生成字符的过程（采样）有两种实现方法：

贪婪采样greedy sampling：始终选择可能性最大的输出，即0,1,0,0,0,0的选择

随机采样stochastic sampling：根据概率选择，即若此字符的概率有30%，则输出的时候，30%概率输出此字符。

softmax温度：

以上两种方法，贪婪的输出太固定，无创造性；随机法无法控制随机性大小。故引入softmax温度，作为概率分布熵值，温度越低，输出越固定（趋近贪婪）；温度越高，输出越随机。



使用温度重新对概率加权的方法: P230

```
re-weighting = exp(log(原概率数组)/temperature)/sum(exp(log(原概率数组)/temperature))
```

最后返回的是温度重新加权过后的概率数组。可在中间选择最大概率，并返回其index。P232

8.1.4 实现字符集的LSTM文本生成

录入一段文字，根据提前训练好的模型，尝试输出这段文字之后的下一个字符。

方法:

录入的文字以 (sequence, max_len, unique_char) 的三维数组形式。其中sequence是录入每次录入的句子，max_len是句子字符数长度，unique_char所有字符的可能性，比如共58种可能。

LSTM+Dense建模，训练完毕后随机输入一段文字，predict生成这段文字之后那个字符的概率数组，温度加权这个数组后，返回其最大概率的索引，利用索引将此字符找到后添加到这段文字末尾，重复此过程，文本渐渐生成。P231-234

其中温度加权，温度越低，单词重复性越大。温度越高，越可能创建新单词。

所有生成的文本无任何意义，只是概率的统计输出，无法传递任何有意义的信息。

code:

```
D:\AI\deep-learning-with-python-notebooks-master\8.1-text-generation-with-lstm.py
```

8.2 DeepDream

艺术性（在我看来比较恶心）的修改图像的技术。（不止图像，语音，音乐都可以用）

原理类似于：P136过滤器可视化

8.2.1 用Keras实现DeepDream

code:

```
D:\AI\deep-learning-with-python-notebooks-master\8.2-deep-dream.py
```

代码部分要点讲解：

1. 加载了预训练模型inception后，在本py内不会在次进行训练，所以用：

```
K.set_learning_phase(0) # 在py内禁止所有训练操作
```

2. 加载的inception模型，如之前所述，底层几何图案，顶层是形象图案（狗脸，鸟嘴）

3. 梯度标准化：

```
grads /= K.maximum(K.mean(K.abs(grads)), 1e-7) # 最后的1e-7防止除0
```

4. DeepDream的核心算法： P239

5. 图像与array相关的转换算法见8.2py中:
def preprocess_image和def deprocess_image
- 另外, resize方法见def resize_img
6. 梯度上升算法见
def gradient_ascent
7. 后端K的数学运算在此py内广泛运用

8.3 神经风格迁移

风格迁移: 目标内容 + 参考风格 = 生成图像

参考风格: 纹理, 颜色, 视觉图案

目标内容: 图像的高级宏观结构, 如建筑轮廓, 远近景构图等

生成图像: 包含了目标内容, 但具有参考风格

另外:

卷积底层捕捉图像的局部信息, 如纹理, 颜色等。

卷积顶层捕捉图像的抽象详细, 如猫耳朵, 狗眼睛等。P125

8.3.1 内容损失

为了保证生成图像 (具有原图像的内容, 参考图像的风格) 的内容损失尽量少, 所以内容损失的控制在于预训练卷积模型靠近顶部的某层在**目标图像**上的激活, 和此层在计算**生成图像**时的激活。这两个激活之间的L2范数可以控制内容损失。激活越相似, 保留内容越多。

8.3.2 风格损失

在较低层和较高层保持类似的相互关系, 保留风格。

8.3.3 用Keras实现神经风格迁移

code:

D:\AI\deep-learning-with-python-notebooks-master\8.3-neural-style-transfer.py

迁移过程:

1. 创建网络模型, 能够同时计算参考图像, 目标图像和生成图像的VGG19层激活
2. 使用以上三图的层激活定义损失函数, 是损失函数最小化。
3. 梯度下降是损失函数最小化。

损失函数: P242

$\text{loss} = \text{L2}(\text{style (参考)} - \text{style (生成)}) + \text{L2}(\text{content (目标)} - \text{content (生成)})$

利用预训练模型:

辅助函数:

图像to张量的预处理, 张量to图形的逆处理: P244 两个def

K.constant定义不变张量 K.placeholder定义可变张量占位符 K.concatenate张量合并

损失计算辅助函数:

K.sum K.square平方 K.batch_flatten多维张量变2D K.variable定义可变量loss

K.permute_dimensions张量维度互换 K.transpose矩阵转置 K.dot点积

风格损失计算:

gram矩阵 P245 两个def

梯度下降优化:

L-BFGS算法，在SciPy中实现有两个要求：

input的张量是2D，需展平：x.flatten()

损失loss和梯度grads要作为两个单独的函数传入算法中：建类Evaluator()

8.4 用变分自编码器生成图像

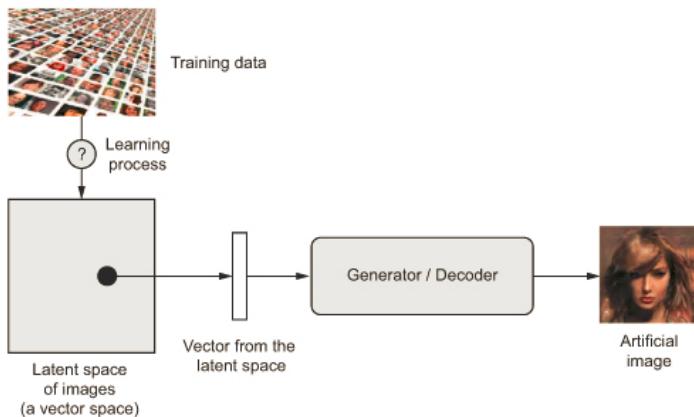
从图像(声音，音乐，文本)的潜在空间中采样，生成全新图像或者编辑现有图像。

技术：

变分自编码器VAE, variational auto encoder

生成式对抗网络GAN, generative adversarial network

8.4.1 从图像的潜在空间中采样



寻找图像潜在空间，在空间中采样，然后将其映射到图像空间生成新图像。如上图。

VAE:

用于具有良好结构的潜在空间，连续性好。适用于图像编辑。

GAN:

生成的图像逼真，但潜在空间没有良好结构。适用于学术研究。

8.4.2 图像编辑的概念向量

概念向量concept vector:

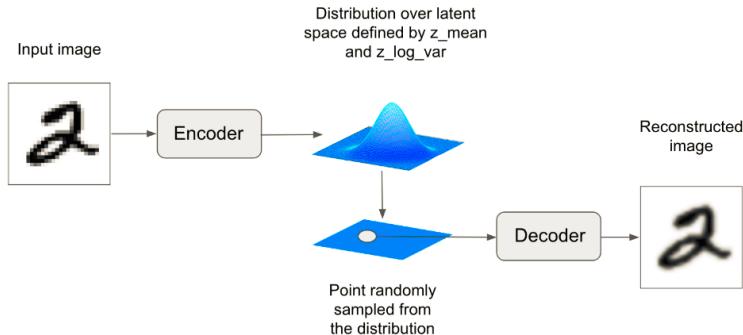
如人脸图片的微笑向量，墨镜向量，性别向量。墨镜向量可以决定图像的人脸是否佩戴墨镜。

8.4.3 变分自编码器VAE

VAE适用于利用概念向量进行图像编辑任务。

VAE原理：

如下图，Encoder处是编码器，将原始图像映射到潜在空间compressed representation中的两个参数mean和log_zariance。在正态分布随机采样一个点。解码器将潜在空间的这个点映射回原始输入图像。



注：上面提到的潜在空间的正态分布可以理解为书P250的图，是个符合正态分布的连续空间。所以靠近mean值的正态分布上的每个点都可被解码为与原始图像类似的图像。P252

潜在空间和概念向量：

VAE原理的“注”解释了潜在空间。潜在空间（上图蓝色）中的每个方向都表述数据中一个有意义的变化轴，所以概念向量非常适合与此变化轴配合来进行训练操作。

VAE的loss：

重构损失reconstruction loss：解码后的样本匹配原始输入的损失loss

正则化损失regularization loss：学习良好结构的潜在空间，防止过拟合。

code:

D:\AI\deep-learning-with-python-notebooks-master\8.4-generating-images-with-vaes.py

code解释：见8.4py

1. VAE 编码器模型：原始图像作为输入，模型层用Conv2D，padding=same，后连扁平层和全连接。最后模型实例化为两个潜在空间的向量mean和log_var

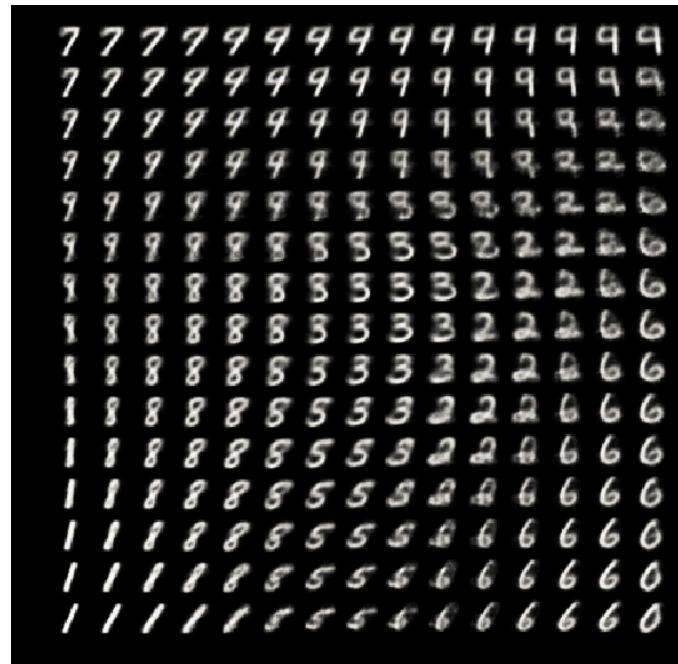
2. 采样函数：自定义层Lambda，内嵌采样函数，采样函数利用编码器输出的两个在潜在空间中的向量来采样一个点

3. VAE解码器模型：采样的点作为输入，模型层用Dense，Reshape，Conv2DTranspose，Conv2D，将采样点解码为与原始输入图相同尺寸的特征图，并做输出。

4. VAE损失模型：自定义层CustomVariationalLayer，输入是原始图像，输出是解码器模型。

5. 训练模型：Model的输入是原始图像输入，输出是VAE损失模型。由于损失模型自带损失，所以训练模型在编译和fit的validation时，loss全部指定为None。

6. 图形绘制：绘制的网格中显示了数字3在沿着不同方向会逐渐变为其他数字。所以网格展示了不同数字间的完全连续分布。
如下图：



小节：

深度学习来生成图像，就是通过读潜在空间进行学习来实现，潜在空间能捕捉到关于图像数据集的统计信息。通过对潜在空间中的点进行采样和解码，可以生成不同的图像。 P256

数据集CelebA有20w名人肖像，适用于VAE实验，比MNIST好很多。

8.5 生成式对抗网络GAN简介

生成式对抗网络GAN， generative adversarial network

潜在空间：低维向量空间，其中任意一点都可以由生成器映射为一张图像。

生成器网络generator network：以潜在空间随机一点作为输入，将其解码为一张合成图像。

判别器网络discriminator network：以一张真实or合成图像作为输入，预测其来自训练集还是生成器

生成器要在训练中不断生成越来越逼真的图像来欺骗判别器，让其无法判断人造图像。判别器不断适应生成器的图像，提高能力，为图像真实性设定更高的判别标准。此为对抗性网络，如下图。

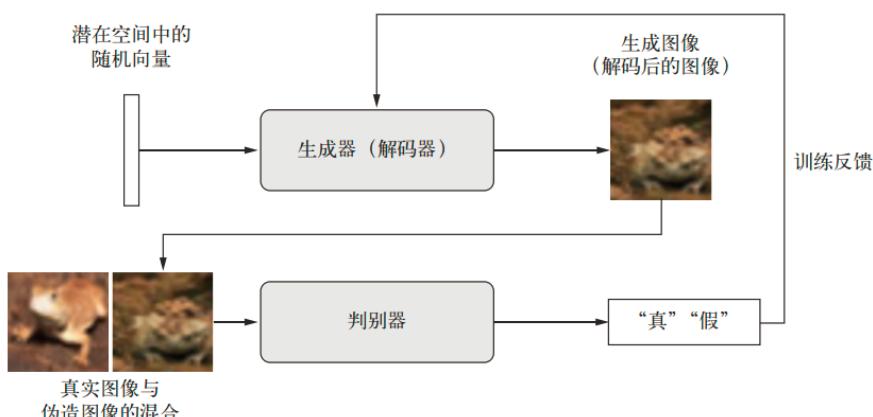


图 8-15 生成器将随机潜在向量转换成图像，判别器试图分辨真实图像与生成图像。
生成器的训练是为了欺骗判别器

注：

1. 对抗性网络的潜在空间没有良好的结构，且不连续。
2. 不想其他网络模型，优化loss即可。GAN的优化对象不是固定的，它是动态系统，优化过程不是找到某项的最小值，而是生成器和判别器之间的平衡。

8.5.1 GAN的简要实现过程

GAN实现流程：

1. 生成器generator将shape为(潜在空间维度,)的向量映射成(32, 32, 3)的图像
2. 判别器discriminator将此图像映射为二进制分数，评估图像为真的概率。
3. GAN将生成器和判别器连接， $gan(x)=\text{discriminator}(\text{generator}(x))$ 。即GAN网络模型的作用是将潜在空间一点映射为评估概率的结果。
4. 训练判别器：使用带有“真、假”标签的图像来训练判别器。
5. 训练生成器：gan模型的损失loss相对于生成器权重的梯度。梯度的移动方向是使判别器更相信生成器生成的图像为真。所以每步训练都需要移动生成器的权重。

8.5.2 GAN技巧

技巧：

1. 使用三角函数tan作为生成器最后一步的激活，而不是用sigmoid
2. 使用正态分布（高斯分布），而不是均匀分布进行采样。
3. 在判别器中引入dropout，或者向判别器的标签中添加随机噪音从而引入随机性。
4. 最大池化和ReLU激活会引起梯度稀疏，通过步进卷积代替最大池化进行下采样，使用LeakyReLU层代替ReLU激活。
(LeakyReLU允许较小的负激活值，放宽了稀疏限制)
5. 生成的图像中存在棋盘状伪影，可通过生成器，判别器在使用2D卷积时，让内核大小能够被步幅整除来消除伪影。伪影图见P260

8.5.3 生成器

生成器的常见问题是会卡壳在类似噪声的图像上，通过上述技巧2，引入dropout后解决此问题

code:

```
D:\AI\deep-learning-with-python-notebooks-master\8.5-gan.py
```

8.5.4 判别器

判别器主要用于将接收来的图像做分类：生成图像 or 来自训练集的真实图像

code:

如上

注：

同生成器，也在全连接层上面添加了dropout来排除掉噪声问题，从而使生成器和判别器达成平衡。另外也可以在判别器标签中添加噪音来解决此问题。P259 技巧3

另外，在优化器内添加了梯度剪裁clipvalue来限制梯度范围，使用了学习率来稳定训练过程。

8.5.5 GAN网络

GAN网络模型让其生成器往梯度方向移动，提高其欺骗判别器的能力。

GAN网络模型会冻结判别器，防止其训练，所以它的权重不会更新，而生成器的权重会得到更新。

code:

如上。

8.5.6 训练DCGAN

DCGAN: a GAN where the generator and discriminator are deep convnets. In particular, it leverages a Conv2DTranspose layer for image upsampling in the generator.

循环训练流程：

1. 在潜在空间latent space中，随机选取一点（亦为随机噪声）作为输入
2. 将此随机输入用generator映射为图像
3. 将生成图像与真实图像混合（不是图像融合，而是混放在一起组成训练集合）
4. 上一步组成的训练集合（含有真实图像，生成图像）及相应标签（真，假）来训练判别器
5. 在潜在空间中选取新的点
6. 使用新点生成的图像，混合真实图像的数据集来训练GAN网络模型，由于判别器被冻结（无法再训练）所以只更新生成器权重，使生成器可以更加欺骗判别器。

code:

同上。非常耗时，10000训练大概耗时>60h

代码解释：

循环训练中，先在潜在空间挑选多个（batch）随机点，然后generator.predict将其映射成图像，将此图像用np.concatenate混入其他真实图像的训练集中（标签也是）。在标签集中添加随机噪声。训练判别器。重新选取batch个新随机点。将其标签全部标为“真实图像”，冻结判别器模型，训练gan模型。每一百图，保存模型权重，打印gan的loss和判别器loss进行对比。若对抗gan的loss大幅增加而判别器损失接近0，则增大判别器dropout比率，减小判别器学习率（**因为此时判别器支配了gan网络模型，没有达成生成器和判别器的平衡！**）。

注：

实际应用时，用VAE。学术研究时，用GAN。

Chapter 9 总结

9.1 重点内容回顾

人工智能AI, artificial intelligence:

古老而宽泛的概念，将认知过程自动化的所有尝试。即，思想的自动化。

机器学习machine learning:

人工智能的分支之一，通过观察训练数据，用数据训练改善模型。

深度学习deep learning:

机器学习的分支之一，深度学习的模型是层组成的图，层由权重来参数化，模型所学到的知识存在权重中，训练的过程就是权重值趋于正确值的过程。

深度学习模型的每一层都是对数据的几何变换，这种几何变换必须是可微的，这样才可以通过梯度下降来学习参数。

深度学习将意义转换为向量，转换为几何空间，然后逐步学习将一个空间映射到另外一个空间的几何变换。意义来自于事物之间的成对关系，这些关系用距离函数来表示。

深度学习的模型是层组成的图，和神经网络没有任何关系，叫分层表示学习layered representations learning（或层级表示学习，深度可微模型，链式几何变换）更贴切些，它的核心是连续的几何空间操作。

深度学习的推动得益于：算法的进步，大量可用数据，GPU硬件和可用框架（tf, keras）

深度学习流程：

P268

网络模型network model:

向量数据：密集连接层Dense

图像数据：二维卷积神经网络Conv2D

声音数据：一维卷积神经网络Conv1D（首选） or 循环神经网络

文本数据：一维卷积神经网络Conv1D（首选） or 循环神经网络

时间序列数据：循环神经网络（首选） or 一维卷积神经网络Conv1D

其他类型数据：循环神经网络 or 一维卷积神经网络Conv1D

视频数据：三维卷积神经网络Conv3D 或者（帧级二维-提取特征+循环or一维卷积-处理序列）

立体数据：三维卷积神经网络Conv3D

密集连接层Dense：

层的每个单元都和其他所有单元相连，所以可以映射任意两个输入特征之间的关系。（二维卷积只能查看局部关系。）

通常用于网络最终的分类or回归阶段。

二分类：IMDB评论分类

激活：sigmoid loss: binary_crossentropy

单标签多分类：新闻话题分类

激活：softmax loss: categorical_crossentropy sparse_categorical_crossentropy 整数

多标签多分类：书中无具体案例

激活：sigmoid loss: binary_crossentropy

回归问题：boston房价预测

激活：无激活函数 loss: mean_squared_error mean_absolute_error

卷积神经网络：

卷积层可查看空间的局部模式。

卷积神经网络是卷积层（1D,2D,3D）和池化层（平均，最大）的堆叠。池化层下采样，保证特征图尺寸合理，最后用全局池化或Flatten层将特征转为向量，再练Dense层用于分类or回归。

深度可分离卷积：

效果和卷积相同，但速度更快，效率更高，推荐使用！如SeparableConv2D层

循环神经网络RNN：

对时间更敏感的序列，优先用循环神经网络。

分类：SimpleRNN, LSTM, GRU。其中LSTM更强大，但计算代价更高。

RNN堆叠时，最后一层之前的每一层都应该返回输出完整的序列。

深度学习可应用于行业：

P272 其中文本与视频间的相互映射，一听就很厉害！

9.2 深度学习的局限性

目前深度学习的局限性来源于很多问题无法建模，比如有了代码和代码说明，无法建模来自动生成新的代码。

对抗样本：

用于输入网络模型后，让模型对他们做出错误分类。比如一直大熊猫的照片，但tag是长臂猿。此对抗样本的输入，可能完全改变模型对其他图片的分类判断。

极端泛化：

人类具有根据很少数据就能够进行推理和抽象的能力，此能力称为极端泛化。比如我们只见过几种狗和猫（未见所有or大部分品种），但对于未见过品种的狗，我们可以立刻判断是狗，而不是猫。

局部泛化：

深度学习通过大量样本的训练后，可以捕捉样本的局部特征，从而进行判断分类或回归预测。即，我们需要提供输入空间的密集采样，以便学到输入空间到输出空间的可靠映射。

局限性总结：

由上可知，现实中存在对抗样本，深度学习具有局部泛化而不具备极端泛化，不能像人一样做出推理和抽象，所以很多问题无法通过建模来用深度学习解决。

9.3 深度学习的未来

P277

1. 深度学习需要转型，从目前纯模式识别的局部泛化-->能够进行推理和抽象的极端泛化。用RNN实现。P277
2. 程序合成：利用搜索算法或是遗传算法来探索程序的巨大空间，从而生成简单的程序。
3. 找到训练不可微系统的方法：遗传算法，进化策略，强化学习方法，交替方向乘子法ADMM等。
4. 自动化机器学习AutoML：自动调整网络模型架构和超参。Hyperopt库
5. 网络模型的复用和抽象：将网络模型model当做当前的代码模块，可以复用，可以抽象。

9.4 最新进展

Kaggle竞赛：

XGBoost库：用来浅层机器学习

Keras框架：用来深度学习

arXiv Sanity Preserver：最新论文推荐网站

Keras Slack频道：<https://kerasteam.slack.com> 寻求帮助和讨论keras

Keras博客：<https://blog.keras.io> keras教程和深度学习有关文章

推特：@fchollet

附录：

python及库： P284

GPU支持： P284 AWS GPU实例： P288

Theano： P285

Keras： P286

Jupyter notebook： P287

至此，Python深度学习全部结束

Python深度学习 共9章内容，至此结束

未涉及的机器学习及后端TF知识，详见：Hands on SK&TF笔记。