

Keras深度学习笔记

一 基本概念

1.1 机器学习基础

1.1.1 机器学习分类

机器学习根据训练数据和学习目标的关系可分为四类：

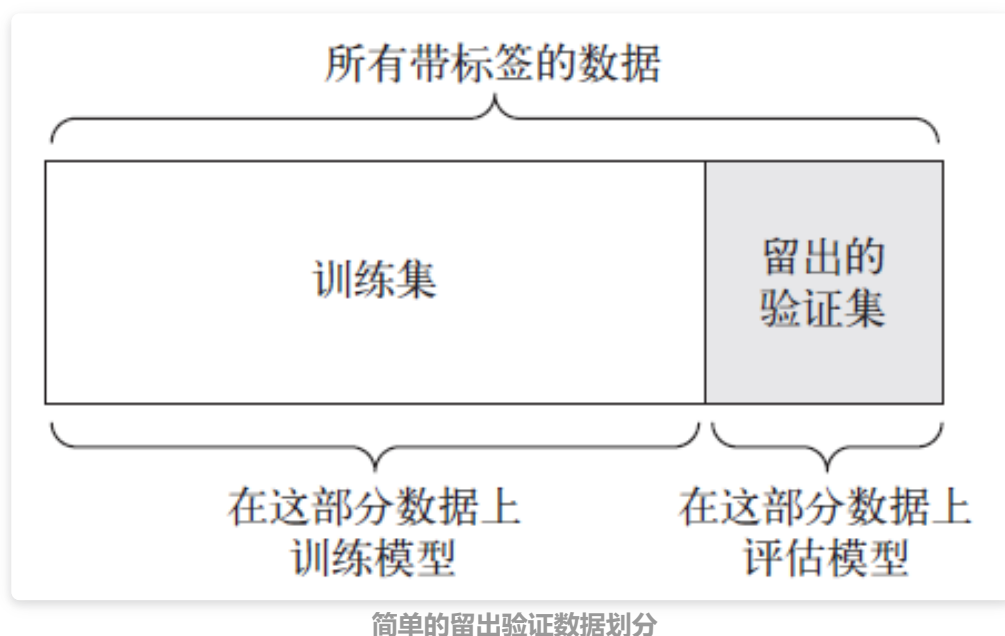
- **监督学习**：监督学习是目前最常见的机器学习类型，给定一组标注过的样本，模型将通过输入数据映射到已知目标。常用的是分类和回归（这两个都是预测，前者对离散数据预测，后者对连续数据预测）。
- **无监督学习**：无监督学习是指在没有目标的情况下寻找输入数据的有趣变换，其目的在于数据可视化、数据压缩、数据去噪或更好地理解数据中的相关性。常用的是降维和聚类。
- **自监督学习**：自监督学习是没有人工标注的标签的监督学习，可将它看作没有人类参与的监督学习，它是监督学习和无监督学习的过渡阶段。
- **强化学习**：使用奖惩机制，模型接收有关其环境的信息，并学会选择使某种奖励最大化。

1.1.2 训练集、验证集和测试集

开发模型时总是需要调节模型配置，即调节超参数，所以需要额外使用一部分数据（验证集）来支持该工作的进行。如果直接将数据划为训练集和测试集两部分，前者用于训练模型，后者用于评价模型并调整参数，这样将会导致模型过拟合。造成这一现象的关键在于信息泄露（*information leak*），每次基于模型在验证集上的性能来调节模型超参数，都会有一些关于验证数据的信息泄露到模型中。训练集、验证集和测试集的配比通常6:2:2，如数据量很大（过百万）可以调整为98:1:1。数据量小时，可使用简单的留出验证、K折验证以及带有打乱数据的重复K折验证。

1. 简单的留出验证（*hold-out validation*）

留出一定比例的数据作为测试集。在剩余的数据上训练模型，然后在测试集上评估模型。这是最简单的评估方法，但有一个缺点：如果可用的数据很少，那么可能验证集和测试集包含的样本就太少，从而无法在统计学上代表数据。



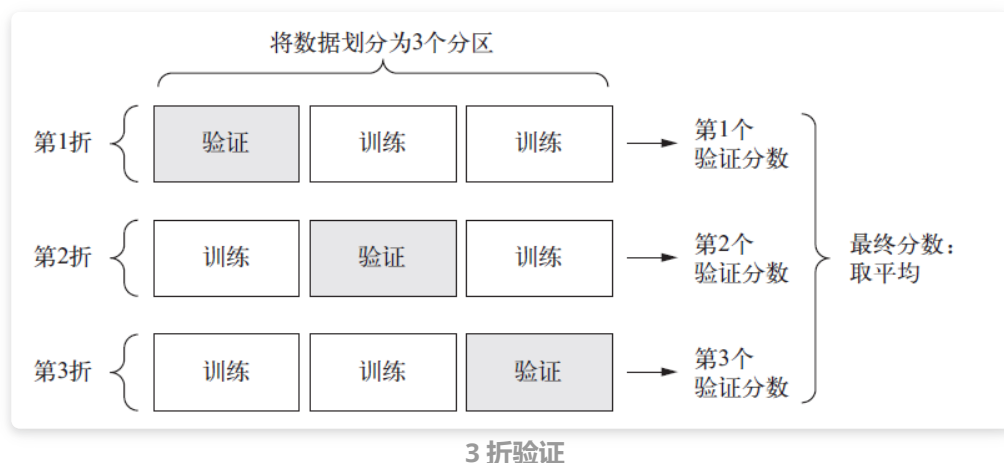
注：上图是在测试集划分后，对训练集和测试集进行划分

代码模板：

```
num_validation_samples = 10000
# 打乱数据
np.random.shuffle(data)
validation_data = data[:num_validation_samples]
data = data[num_validation_samples:]
training_data = data[:]
model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)
# 现在你可以调节模型、重新训练、评估，然后再次调节
model = get_model()
model.train(np.concatenate([training_data, validation_data]))
test_score = model.evaluate(test_data)
```

2. K折验证 (K-fold validation)

将数据划分为大小相同的K个分区。对于每个分区i，在剩余的K-1个分区上训练模型，然后在分区i上评估模型。最终分数等于K个分数的平均值。对于不同的训练集-测试集划分，如果模型性能的变化很大，那么这种方法很有用。



代码模板：

```
k = 4
num_validation_samples = len(data) // k

np.random.shuffle(data)

validation_scores = []
for fold in range(k):
    validation_data = data[num_validation_samples * fold:
                           num_validation_samples * (fold + 1)]
    training_data = np.concatenate([data[:num_validation_samples * fold],
                                     data[num_validation_samples * (fold + 1):]],
                                    axis=0)

    model = get_model()
    model.train(training_data)
    validation_score = model.evaluate(validation_data)
    validation_scores.append(validation_score)
```

```
validation_score = np.average(validation_scores)

model = get_model()
model.train(data)
test_score = model.evaluate(test_data)
```

3. 带有打乱数据的重复K 折验证 (*iterated K-fold validation with shuffling*)

具体做法是多次使用K 折验证，在每次将数据划分为K 个分区之前都先将数据打乱。最终分数是每次K 折验证分数的平均值。注意，这种方法一共要训练和评估 $P \times K$ 个模型（P是重复次数），计算代价很大。（Kaggle常用）

注：

1. 在将数据划分为训练集和测试集 之前，通常应该随机打乱数据。
2. 如果是时序数据，那么在划分数据前不应该随机打乱数据，因为这么做会造成时间泄露（temporal leak）。
3. 一定要确保训练集和验证集之间没有交集。

1.1.3 数据预处理

- **向量化**：神经网络的所有输入和目标都必须是浮点数张量（在特定情况下可以是整数张量）。
- **值标准化**：一般来说，将取值相对较大的数据（比如多位整数，比网络权重的初始值大很多）或异质数据（*heterogeneous data*，比如数据的一个特征在0~1 范围内，另一个特征在100~200 范围内）输入到神经网络中是不安全的。这么做可能导致较大的梯度更新，进而导致网络无法收敛。

输入数据应有以下特征。

- **取值较小**：大部分值都应该在 0~1 范围内。
- **同质性 (*homogenous*)**：所有特征的取值都应该在大致相同的范围内。
- **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。（注：不能训练集无缺失值，而测试集有缺失值，在这种情况下，网络不可能学会忽略缺失值。）

1.1.4 特征工程

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。

但对于现代深度学习，大部分特征工程都是不需要的，因为神经网络能够从原始数据中自动提取有用的特征。但也需要注重特征工程，因为：

- 良好的特征仍然可以让你用更少的资源更优雅地解决问题。
- 良好的特征可以让你用更少的数据解决问题。

1.1.5 过拟合和欠拟合

训练数据上的损失越小，测试数据上的损失也越小。这时的模型是欠拟合（*underfit*）的，即仍有改进的空间，网络还没有对训练数据中所有相关模式建模，解决方案是继续学习或增加模型复杂度。但在训练数据上迭代一定次数之后，泛化不再提高，验证指标先是不变，然后开始变差，即模型开始过拟合，最优解决方法是获取更多的训练数据或者使用正则化。

过拟合的解决方法：

- 获得更多数据
 - 手工获得
 - 数据增强
- 减少网络容量

- 网络深度
- 隐藏神经元数量
- 权重正则化
 - L1 || L2正则化

```
from keras import regularizers
regularizers.l1(0.001) # L1正则化
regularizers.l1_l2(l1=0.001, l2=0.001) # L2正则化

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

- dropout正则化

```
model.add(layers.Dropout(0.5))
```

1.1.6 机器学习一般流程

- 定义问题与要训练的数据。收集这些数据，有需要的话用标签来标注数据。
- 选择衡量问题成功的指标。你要在验证数据上监控哪些指标？
- 确定评估方法：留出验证？K折验证？你应该将哪一部分数据用于验证？
- 开发第一个比基准更好的模型，即一个具有统计功效的模型。
- 开发过拟合的模型。
- 基于模型在验证数据上的性能来进行模型正则化与调节超参数。

1.2 Keras基础

1.2.1 层的概念

简单的向量数据保存在形状为 (samples, features) 的2D 张量中，通常用密集连接层 [densely connected layer，也叫全连接层 (fully connected layer) 或密集层 (dense layer)，对应于 keras 的 Dense 类] 来处理。序列数据保存在形状为 (samples, timesteps, features) 的3D 张量中，通常用循环层 (recurrent layer，比如 keras 的 LSTM 层) 来处理。图像数据保存在4D 张量中，通常用二维卷积层 (keras 的 Conv2D) 来处理。

```
from keras import models
from keras import layers
# 模型中添加的层都会自动匹配输入层的形状
model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,)))
model.add(layers.Dense(32)) # 输入神经元数量为上一层的输出数量
```

1.2.2 损失值与优化器

对于二分类问题，可以使用二元交叉熵 (binary_crossentropy) 损失函数；对于多分类问题，可以用分类交叉熵 (categorical_crossentropy) 损失函数；对于回归问题，可以用均方误差 (mean-squared error) 损失函数；对于序列学习问题，可以用联结主义时序分类 (CTC, connectionist temporal classification) 损失函数，等等。

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

1.2.3 Keras开发流程

1. 定义训练数据：输入张量和目标张量。
2. 定义层组成的网络（或模型），将输入映射到目标。
3. 配置学习过程：选择损失函数、优化器和需要监控的指标。
4. 调用模型的 `fit` 方法在训练数据上进行迭代。

注：

1. 定义模型有两种方法，一种是使用 `Sequential` 类（仅用于层的线性堆叠，这是目前最常见的网络架构），另一种是函数式API（functional API，用于层组成的有向无环图，让你可以构建任意形式的架构）；

2. 编译模型中的优化器、损失和指标可自定义，通过 `optimizer`、`loss` 和 `metrics` 类设置。

代码模板：

```
from keras import models
from keras import layers
from keras import optimizers

# Sequential 类
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))

# 函数API定义
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = models.Model(inputs=input_tensor, outputs=output_tensor)

# 编译
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='mse',
              metrics=['accuracy'])

# 拟合
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)

# 评估
model.evaluate(x_test, y_test)

# 预测
```

```
model.predict(x_test) # 模型精度合理，可用于实践
```

二 常见问题基本模板

2.1 二分类问题

问题描述：预测电影评论的情绪，即电影评论分类

数据：IMDB数据集

损失函数：二分交叉熵 (`binary_crossentropy`)

评估指标：精度 (`accuracy`)

代码模板：

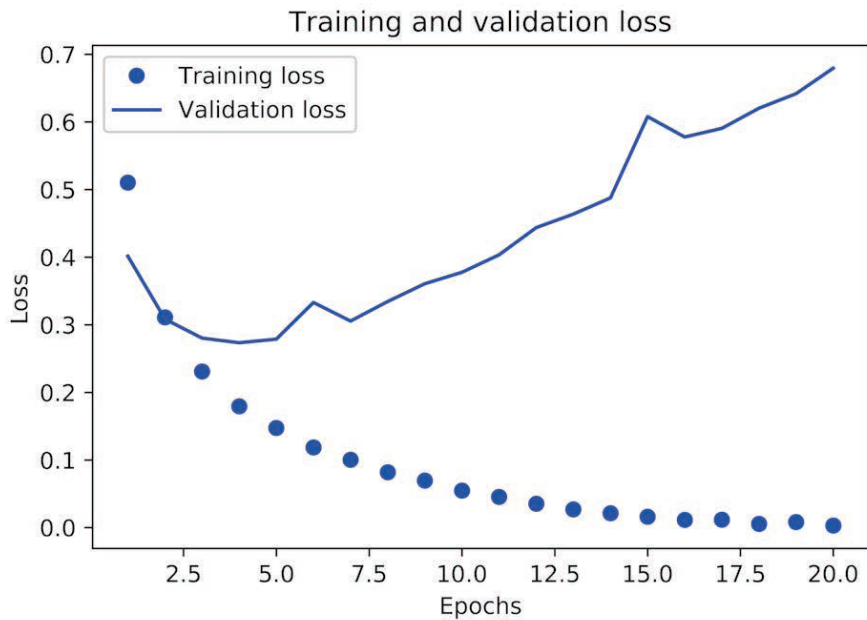
- 模型构建与测试

```
model = models.Sequential()
model.add(layers.Dense(16, activation="relu", input_shape=(10000,)))
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))
# 使用验证集，该验证集非test，是development数据集，
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
# fit方法返回一个History对象
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
# 该对象有loss和accuracy的字典
history_dict = history.history
```

- 绘制学习曲线

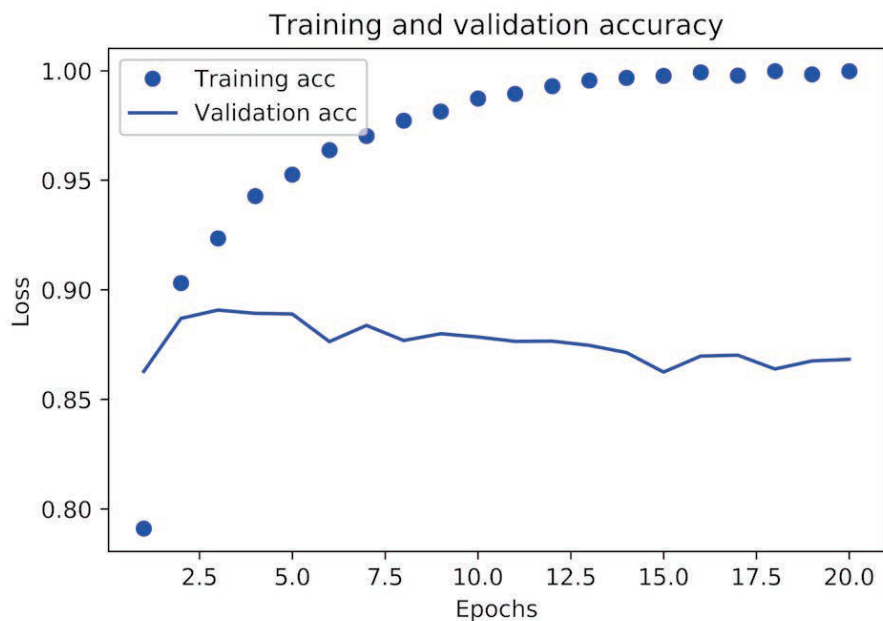
```
#使用matplotlib绘制训练成本图
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```



loss曲线

- 绘制精度曲线

```
#使用matplotlib绘制训练精度图
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()
```



accuracy曲线

分析：根据两幅图，可知发生了过拟合，在第3轮时应该停止训练（或者使用其他降低过拟合的方法）

- 模型调整

```

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
model.predict(x_test) # 模型精度合理，可用于实践

```

注：

1. 无论你的问题是什么，`rmsprop` 优化器通常都是足够好的选
2. 当曲线不平滑时，可使用如下代码：

```

def smooth_curve(points, factor=0.8):
    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

```

2.2 多分类问题

问题描述：预测新闻的主题，即对新闻进行多分类

数据：路透社数据集

损失函数：分类交叉熵 (`categorical_crossentropy`)

评估指标：精度 (`accuracy`)

代码模板：

- **数据清洗—独热编码**

极大可能会用到，由于这是 `Keras` 笔记，故不介绍 `sklearn` 等其他实现方法

```
one_hot_train_labels = to_categorical(train_labels)
```

- **模型构建与测试**

曲线和分析什么的同上，故不介绍，直接贴模型

```

model = models.Sequential()
model.add(layers.Dense(64, activation="relu", input_shape=(10000,)))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(46, activation="softmax")) #输出层维数的等于独热编码的维
数
model.compile(optimizer='rmsprop',

```



```

loss='categorical_crossentropy',
metrics=['accuracy'])

x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]

model.fit(partial_x_train,
          partial_y_train,
          epochs=20,
          batch_size=512,
          validation_data=(x_val, y_val))
model.evaluate(x_test, one_hot_test_labels)

```

注：

- 1.如果要对 N 个类别的数据点进行分类，网络的最后一层应该是大小为 N 的 `Dense` 层。一种合理的解释为，试图将大量信息（这些信息足够恢复 N 个类别的分割超平面）压缩到维度很小的中间空间，从而导致信息丢失、精度下降。
2. 对于单标签、多分类问题，网络的最后一层应该使用 `softmax` 激活，这样可以输出在 N 个输出类别上的概率分布。
3. 这种问题的损失函数几乎总是应该使用 **分类交叉熵**。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。
4. 处理多分类问题的标签有两种方法：
 - 通过分类编码（也叫 `one-hot` 编码）对标签进行编码，然后使用 `categorical_crossentropy` 作为损失函数。
 - *将标签编码为整数，然后使用 `sparse_categorical_crossentropy` 损失函数。

2.3 回归问题

问题描述：预测房价

数据：波士顿房价数据集

损失函数：均方误差（`MSE`, `mean squared error`）

评估指标：平均绝对误差（`MAE`, `mean absolute error`）

代码模板：

- **数据清洗—标准化**

建模时基本会用到，这也不是回归问题的专属，前者分类问题也需用到

这里只介绍如何实现，也可使用 `skleran.preprocessing.StandardScaler` 实现

```

mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

# 虽然已知测试集，但也要用训练集的均值和标准差来计算
test_data -= mean
test_data /= std

```

- 模型构建与测试

```
def build_model():  
    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
```

- K折交叉验证（取决于数据量）



曲线和分析什么的同上，故不介绍，直接贴模型

```
k = 4  
num_val_samples = len(train_data) // k  
num_epochs = 100  
all_scores = []  
  
for i in range(k):  
    print("processing fold # ", i)  
    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]  
  
    # 合并剩下的数据  
    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)  
  
    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=0)  
    all_scores.append(val_mae)
```

注：

1.回归常用的损失函数为均方误差（MSE），常用的评估指标为平均绝对误差（MAE）。

2.如果可用的数据很少，可使用K折验证评估模型，此时也应使用小的网络，即隐藏层较少，否则将造成过拟合。

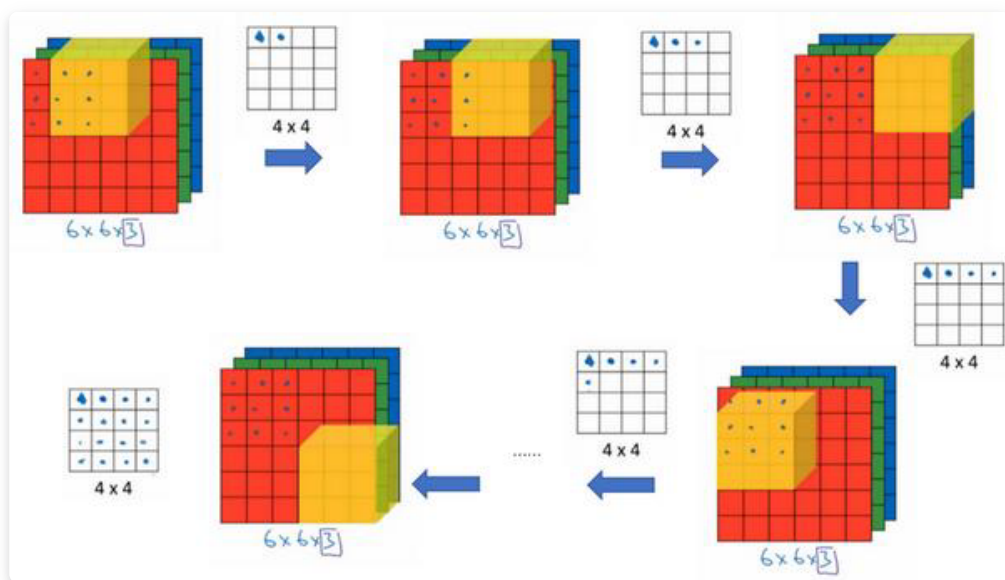
三 计算机视觉&卷积神经网络

3.1 基本概念

- **图像组成**：包含两个空间轴（**高度和宽度**）和一个深度轴（也叫作**通道**，*Channel*）的3D张量。对于黑白图（如MNIST数字图像），深度等于1，RGB图深度等于3，。
- **卷积运算（Convolution）**：通过卷积学到图像的局部模式，具有平移不变性（可以高效地学到局部模式）以及模式的空间层次结构（卷积从前往后能学到局部到宏观的特征）。

对应于Keras的 `Conv2D` 层：`Conv2D(output_depth, (window_height, window_width))`

- **尺寸**：过滤器（*Filter*）的尺寸为 $n * n * c_{l-1}$ ，其中 n 一般为奇数，常用3、5、7等， c_{l-1} 为上一层的通道数量。卷积窗口的数量作为下一层特征图的通道数。
- **Padding**：将特征图进行填充，根据 `padding_size`，分为 *Valid* 卷积和 *Same* 卷积，前者意味着不填充，后者意味着填充以及卷积后输出的特征图尺寸和原来一样。
- **步长（Stride）**：过滤器每次滑动的步幅。



卷积示意图

卷积核运算后输出的特征图尺寸为：

$$\text{times}(\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) \times m_{l-1}$$

其中， n 为图像的宽和高， p 为padding size， f 为过滤器的宽和高， s 为步长， m_{l-1} 前一层的过滤器的数量。

- **池化（Pooling）**：一般用**最大池化（MaxPooling2D）**，最大池化与卷积的最大不同之处在于，最大池化通常使用2x2的窗口和步幅2，其目的是**将特征图下采样2倍，即缩减一半**。用池化的原因主要有两点：1、让后面的卷积层包含更复杂的信息（学到宏观知识）；2、减少参数，防止过拟合。
- **全连接（Dense）**：同普通的神经网络，一般将该层放到卷积的最末，以学习宏观知识。

基本模板

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

```

model = models.Sequential()
# 卷积1
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
# 卷积2
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# 卷积3
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
#model.summary() #查看模型的架构

# 拉直
model.add(layers.Flatten())
# 全连接
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# 编译、训练、评价等操作同基本的神经网络

```

注：网络中特征图的深度在逐渐增大（如从32 增大到128），而特征图的尺寸在逐渐减小（如从150×150 减小到7×7），这几乎是所有卷积神经网络的模式。

3.2 CNN技巧

3.2.1 数据增强

Keras中有一个图像处理辅助工具的模块，位于 `keras.preprocessing.image`，它包含了一个 `ImageDataGenerator` 类，可以快速创建Python生成器，能够将硬盘上的图像文件自动转换为预处理好的张量批量，可完成一下处理：

- 读取图像；
- 将JPEG文件解码为RGB像素网格；
- 将这些像素网格转换成浮点张量；
- 将像素值缩放到[0, 1]区间。

数据增强是从现有的训练样本中生成更多的训练数据，其方法是利用多种能够生成可信图像的随机变换来**增加 (augment)** 样本。

使用 `ImageDataGenerator` 进行预处理以及数据增强

```

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,      # （数据增强开始）， 旋转角度范围
    width_shift_range=0.2,  # 水平平移比例
    height_shift_range=0.2, # 垂直平移比例
    shear_range=0.2,       # 随机错切角度
    zoom_range=0.2,        # 随机缩放范围
    horizontal_flip=True,   # 随机将一半图像水平翻转
    fill_mode='nearest')   # 处理变换时超出边界的点
test_datagen = ImageDataGenerator(rescale=1./255) # 不能增强验证数据

train_generator = train_datagen.flow_from_directory(
    train_dir, # 目标目录

```

```

target_size=(150, 150), # 将所有图像大小调整为150*150
batch_size=32,
class_mode='binary') # 若inary_crossentropy损失, 则二进制标签
validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

#####
#model已经建好
#####

history = model.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50)
model.save('my_model.h5')

```

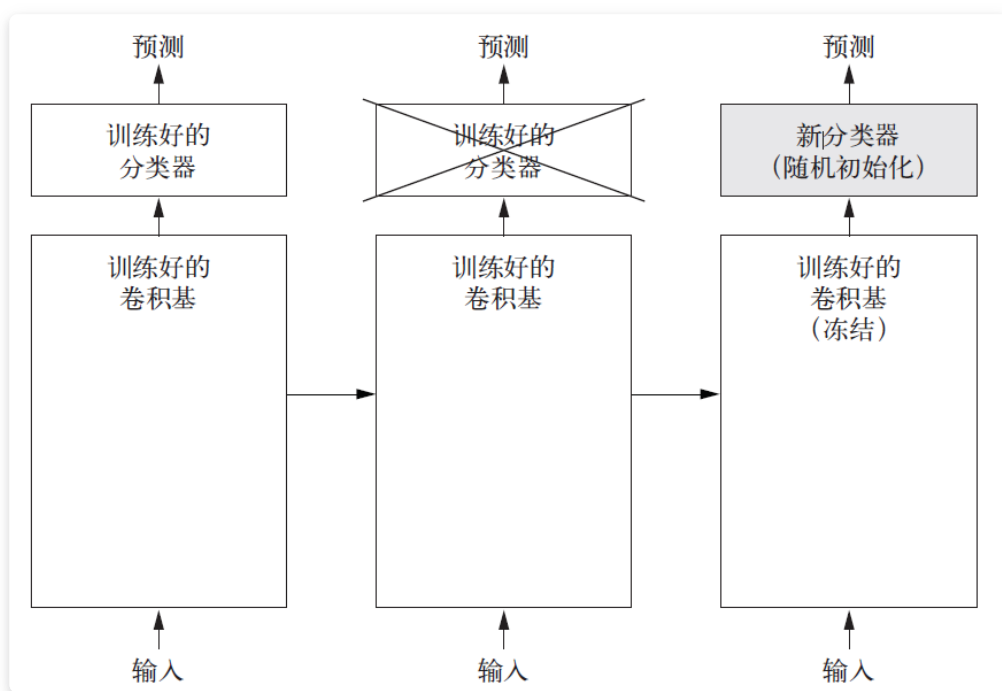
3.2.2 迁移学习 (预训练)

预训练网络 (pretrained network) 是一个保存好的网络, 之前已经在大型数据集上训练好。使用预训练网络有两种方法: 特征提取 (feature extraction) 和微调模型 (fine-tuning)

3.2.2.1 特征提取 (feature extraction)

特征提取是使用之前网络学到的表示来从新样本中提取出有趣的特征。然后将这些特征输入一个新的分类器, 从头开始训练。CNN由两部分组成: **卷积基 (convolutional base)** 和**密集连接分类器**。特征提取即使用原始模型的卷积基, 并根据现有数据重新训练分类器。

注: 如果新数据集与原始模型训练的数据集有很大差异, 那么最好只使用模型的前几层来做特征提取, 而不是使用整个卷积基。



特征提取示意图

1. 基于VGG16以及ImageNet数据集的特征提取 (无数据增强, 直接使用原始卷积基)

```

from keras.applications import VGG16

```

```

import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras import models
from keras import layers
from keras import optimizers

# 预训练网络
conv_base = VGG16(weights='imagenet', # 模型初始化的权重检查点
                    include_top=False, # 指定模型最后是否包含密集连接分类器，原始有
                    input_shape=(150, 150, 3)) # 输入到网络中的图像张量的形状，可选
1000个

# 图像预处理生成器
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

# 提取特征（利用预训练的卷积基将数据进行计算处理）
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
    # 生成器会一直循环，必须在读取完成所有图像后终止循环
    if i * batch_size >= sample_count:
        break
    return features, labels

# 对训练集、验证集和测试集进行特征提取
train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)

# 为了进行密集连接迅速按，将数据展平
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))

# 密集连接层
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5)) # 防止过拟合
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

```

```
history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features,
                                    validation_labels))
```

2. 同1，对原始卷积基进行冻结，可用数据增强

```
from keras import models
from keras import layers
from keras.applications import VGG16

# 预训练网络
conv_base = VGG16(weights='imagenet', # 模型初始化的权重检查点
                  include_top=False, # 指定模型最后是否包含密集连接分类器，原始有
                  input_shape=(150, 150, 3)) # 输入到网络中的图像张量的形状，可选

model = models.Sequential()
model.add(conv_base) # 原始卷积基代替卷积层
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

# 设置trainable进行冻结
conv_base.trainable = False

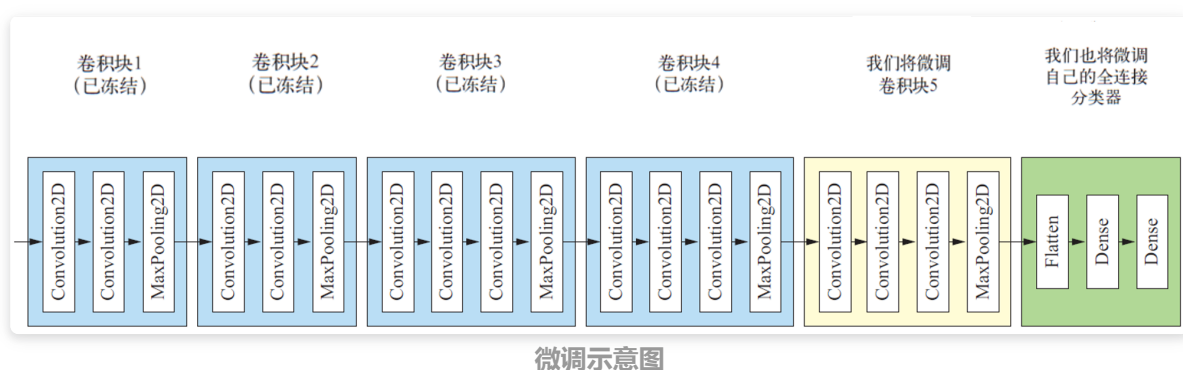
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc'])

# 数据增强见2.1
```

3.2.2.2 微调模型 (fine-tuning)

对于用于特征提取的冻结的模型基，**微调是指将其顶部的几层“解冻”，并将这解冻的几层和新增加的部分（如全连接分类器）联合训练。**微调步骤：

1. 在已经训练好的基网络 (base network) 上添加自定义网络。
2. 冻结基网络。
3. 训练所添加的部分。
4. 解冻基网络的一些层。
5. 联合训练解冻的这些层和添加的部分。



```

conv_base.trainable = True

# 一直冻结到某层
set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5), # 为了不影响冻结层，学习率较
              metrics=['acc'])
小

# 其他代码同3.2.2.1 2

```

注：

1.卷积基中更靠底部的层编码的是更加通用的可复用特征，而更靠顶部的层编码的是更专业化的特征。微调更靠底部的层，得到的回报会更少。

2.训练的参数越多，过拟合的风险越大。卷积基有 1500 万个参数，所以在小型数据集上训练这么多参数是有风险的。

Author：钱小z

Email：qz_gis@163.com

Bio：GISer，Spatiotemporal data mining

GitHub：QianXzhen