**第4章**

# Keras简介

|  |  |
| --- | --- |
|  | 本章内容 |
|  | ◎ Keras简介  ◎ 安装Keras  ◎ Keras的两种模式  ◎ Keras常用API |

## 4.1 keras简介

Keras是一个基于Python的深度学习框架，其包含了一些高层API，Keras不处理张量的操作与计算，而是一个依赖于一些底层的专业处理张量计算的后端库，目前支持的后端库包括Tensorflow、Theano、CNTK，官方表示后续会对Pytorch进行支持。Keras的最大优势在于开发快捷，其极简的API能让开发人员以最简单的方式得到需要的模型，相同的代码可以在GPU与CPU之间无缝切换运行。目前Keras从Python2.7到Python3.6都可兼容，并且其也是目前最为热门的深度学习框架，图4.1-1展示了2018年底深度学习框架在工业场景中的使用占比情况

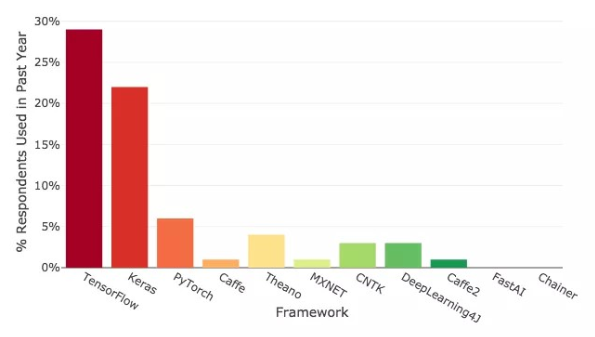


图4.1-1 深度学习框架使用占比

Keras的设计原则包括以下几点。

* 用户友好：Keras是为人类而不是天顶星人设计的API。用户的使用体验始终是Keras考虑的首要和中心内容。Keras遵循减少认知困难的最佳实践：Keras提供一致而简洁的API， 能够极大减少一般应用下用户的工作量，同时，Keras提供清晰和具有实践意义的bug反馈。
* 模块性：模型可理解为一个层的序列或数据的运算图，完全可配置的模块可以用最少的代价自由组合在一起。具体而言，网络层、损失函数、优化器、初始化策略、激活函数、正则化方法都是独立的模块，开发者可以使用它们来构建自己的模型。
* 易扩展性：添加新模块超级容易，只需要仿照现有的模块编写新的类或函数即可。创建新模块的便利性使得Keras更适合于先进的研究工作。
* 与Python协作：Keras没有单独的模型配置文件类型（作为对比，caffe有），模型由Python代码描述，使其更紧凑和更易debug，并提供了扩展的便利性。

Tensorflow、Theano与CNTK是目前深度学习领域常用的一些底层框架，Tensorflow是由Google开发，也是目前使用人数最多的框架，Theano由蒙特利尔大学开发，CNTK由微软开发。Keras可以在这三个平台上进行运行，无需修改代码。Keras的文件中有一个keras.json的文件，该文件的内容如下

{

"image\_data\_format": "channels\_last",

"epsilon": 1e-07,

"floatx": "float32",

"backend": "tensorflow"

}

其中backend对应的即是调用的后端库，TensorFlow是其默认的后端，主要原因在于其应用广泛扩张性强，如果想修改后端库，只需要将backend对应的内容改为想要使用的库即可。

## 4.2 安装Keras

**1.安装Anaconda**

Anaconda是一个基于Python的科学计算发行版，其包含了数以百计的常用的Python库以及机器学习与数据分析常会用到的库例如numpy、pandas、scikit-learn等。Keras需要对后端库的依赖，而这些后端库又依赖于一些其它的Python库，如果手动安装的话极其麻烦并且存在兼容性的问题。而Anaconda则已经提供了一个这样的包，简化了环境的搭建步骤。简单来说，Anaconda是目前深度学习领域最实用、最简单的环境安装包。在接下来的内容中，也将采用Anaconda中提供的一些三方库就行讲解。

Anaconda的具体安装步骤如下。

在Anacona的官网<https://www.anaconda.com/distribution/>选择自己计算机系统对应的包并点击Download，其中Windows平台与macOS平台提供了包含可视化的界面的安装包。

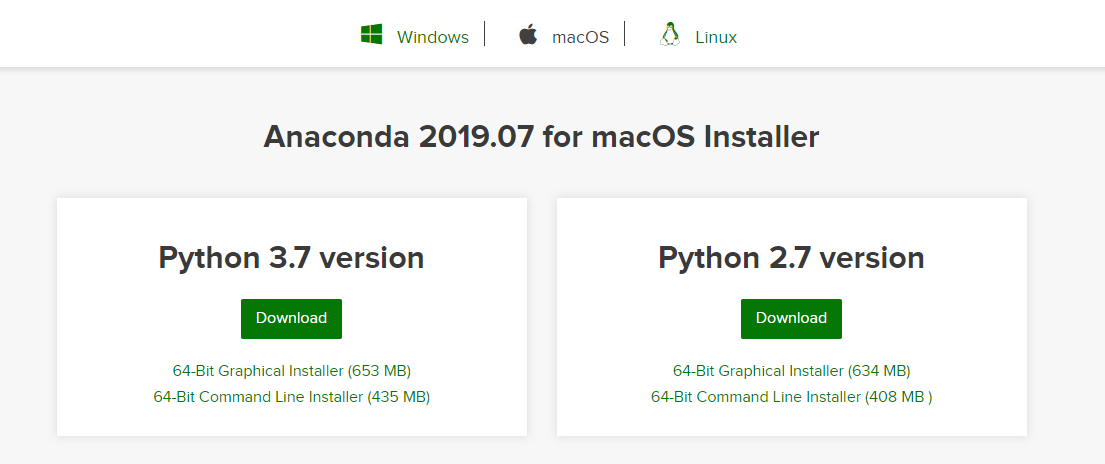


图4.2-1 Anaconda下载页面

如果是Windows平台或macOS平台下载完成后直接双击安装，如果是Linux平台，则执行以下命令。

bash Anaconda3-x.x.x-linux-x86\_64.sh

**2.安装Tensorflow**

后端库我们采用Tensorflow，Tensorflow包括CPU与GPU版本。其中CPU版本安装起来比较简单，只需要采用Python默认的包管理器pip来安装即可。pip可以理解为一个命令行形式的应用商店，其可对包进行安装、卸载、升级等，安装Tensorflow只需要一行命令。

pip install tensorflow

如果要安装Tensorflow的GPU版本，需要准备好一块NVIDIA的显卡，因为目前Tensorflow只支持CUDA。然后安装显卡驱动、CUDA与cuDNN。

其中CUDA是用于GPU运算时的SDK，cuDNN是对于深度学习提出来的高度优化实现，其采用了最先进的技术来提高CNN与RNN的运行速度。

首先前往<https://www.nvidia.com/Download/index.aspx?lang=en-us>确认自己计算机的显卡驱动对应的版本并下载安装。

然后前往<https://developer.nvidia.com/cuda-toolkit-archive>下载显卡对应的CUDA。

最后前往<https://developer.nvidia.com/rdp/cudnn-download>下载对应的cuDNN安装即可。

上述步骤准备完成后，在命令行中执行以下命令即可安装Tensorflow的GPU版本

pip install tensorflow-gpu

Tensorflow的GPU版本的安装方法十分复杂，并且不同的显卡与不同的显卡驱动需要安装包不同版本的CUDA与cuDNN，这里再推荐另一种安装方法，该方法官方表示使用时速度也会比上述提到的安装方式快，并且不需要安装额外的包，只需要采用Anaconda提供的包管理器conda执行以下命令即可。

conda install tensorflow-gpu

**3.安装Keras**

Keras的安装方法很简单，可以采用pip安装也可以采用conda安装，笔者建议采用conda安装，执行以下命令即可。

conda install keras-gpu

安装完成后在命令行中输入Python即可进入Python的ide，分别键入以下代码，如果没有报错即安装成功。

import tensorflow  
import keras

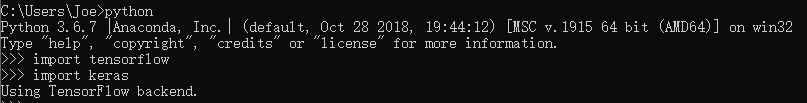


图4.3-1 测试Keras

## 4.3 Keras的两种模式

Keras的模型搭建提供了两种模式，分别是Sequential model与Functional model。对于简单的模型，建议采用Sequential的形式进行搭建，而对于复杂的模型，例如多输入多输出等情况，则建议采用Functional的模式。接下来分别介绍两种模式的使用方法。

1. **Sequential model**

Sequential model只需通过向Sequential传递一个layer的list来构造该模型。

|  |  |
| --- | --- |
| 1 | from keras.models import Sequential |
| 2 | model = Sequential([  Dense(32, input\_dim =784),  Activation('relu'),  Dense(10),  Activation('softmax'),  ]) |

也可以实例化一个Sequential对象，然后调用Add()方法添加Layer。

|  |  |
| --- | --- |
| 1 | from keras.models import Sequential |
| 2 | model = Sequential() |
| 3 | model.add(Dense(32, input\_shape=(784,))) |
| 4 | model.add(Activation('relu')) |
| 5 | model.add(Dense (10)) |
| 6 | model.add(Activation("softmax")) |

注意，Sequential模式需要知道输入数据的shape，因此，Sequential的第一层需要接受一个关于输入数据shape的参数，后面的各个层则可以自动的推导出中间数据的shape，因此不需要为每个层都指定这个参数。有两种方法来为第一层指定输入数据的shape，如果是二维的层例如Dence层，则可采用input\_dim如果是三维或者更高维度的层则需要采用input\_shape，当然input\_shape也可用于二维的层。

1. **Functional model**

事实上，Functional模式才是Keras中最常用的，而Sequential其实是Functional的一种特殊情况，只有当你的网络结构像VGG那样一条路走到头，那可以使用Sequential模式，但是如果模型结果比较复杂，具有多输入、多输出、循环结构、共享层等，这时候Sequential是无法满足需求的，Functional相比于Sequential来说会更加自由一些，依旧以一个全连接网络举例子。

|  |  |
| --- | --- |
| 1 | from keras.models import Model |
| 2 | Input1 = Input(shape=(784,)) |
| 3 | Input2 = Input(shape=(784,)) |
| 4 |  |
| 5 | x = Dense(64, activation='relu')(inputs) |
| 6 | x = Dense(64, activation='relu')(x) |
| 7 | predictions = Dense(10, activation='softmax')(x) |
| 8 | model = Model(inputs=[input1,input2], outputs=predictions) |

可以看到，Functional模式需要提前定义好输入，经过不同的网络结构后得到输出，最后实例化一个Model对象，传递输入值与输出值，对于多输入或多输出，则将输入输出的值修改为一个list即可。

## 4.4 Keras常用API

Keras封装了许多常用的API包括张量的变换模型的训练预测等，本节将会给大家介绍一些在训练过程中常用到的一些API

通常完成模型的搭建后，我们需要使用compile()方法来编译模型，compile()方法包括四个比较重要的参数，分别是

* loss即损失函数，该模型应该采用什么样的损失函数来作为优化函数。常用的有mean\_squared\_error、binary\_crossentropy、categorical\_crossentropy等。
* optimizer即优化器，采用什么优化器来计算梯度，常用的包括SGD、Adam等，
* metrics即训练过程中打印的评判指标，通常包括准确率，召回率等等。
* loss\_weights该参数主要用于多个输出的情况，不同的输出对应不同的损失函数的时候，可以给不同的损失函数不同的权重。

具体的调用方法如下。

model.compile(optimizer='rmsprop',

loss='binary\_crossentropy',

metric=['accuracy']

loss\_weights=[1., 0.2])

完成模型编译后，即可开始训练模型，训练模型的方法如下，其中epochs表示训练多少轮次。

model.fit(x\_train, y\_train, epochs=5, batch\_size=32)

在数据量较大的时候，例如CV与NLP领域，此时内存中没有办法加载大量的数据，此时可采用生成器的方式来生成数据，并采用fit\_generator()来进行训练。注意fit\_generator()的第一个参数是生成器生成的数据，每次应该生成batch\_size大小的数据。

如果要使用模型对新数据进行测试则调用以下方法。

classes = model.predict(x\_test, batch\_size=128)

fit方法中有一个参数叫做callbacks，该参数需要传递一个数组，数组中是CallBack对象，Keras官方提供了一个常用的CallBack叫做EarlyStopping，EarlyStopping可以在检测到某个评判值逆增长时停止训练，并保留上一轮训练好的模型，使用方法如下，其中monitor表示的是监控的评判值，patience表示的是待监控值如果逆增长了多少轮即停止。

|  |  |
| --- | --- |
| 1 | early\_stopping=EarlyStopping(monitor='acc',patience=3) |
| 2 | model.fit(x\_train,y\_train,batch\_size=64,epochs=200, callbacks=[early\_stopping]) |

对于callbacks来说，读者也可以自行定制，只需要继承CallBack类，并重写其中的on\_epoch\_begin、on\_epoch\_end、on\_batch\_end等方法。

在Keras中还有一个很有意义的函数，该函数不会对模型造成影响，但是其作用很大，这就是summary()函数，该函数会打印出编写好的model的所有层对应的输入输出维度，参数个数信息等，在搭建完模型后，通常都会调用该函数，打印出模型结构，从该输出的结果中能有效的对模型结构进行二次检查，以防模型结构错误，其次也能了解到当前模型的参数量，对模型的剪枝等优化操作都有一定的指导性作用，使用方法如下。

model.summary()

以上即是Keras训练阶段常会用到的API，可以看到其简洁的设计相比于Tensorflow来说能节约大量的开发时间。在之后的章节中，我们会为读者介绍更多关于模型搭建的API，并从输入数据的处理、训练到预测提供一体化的代码讲解。