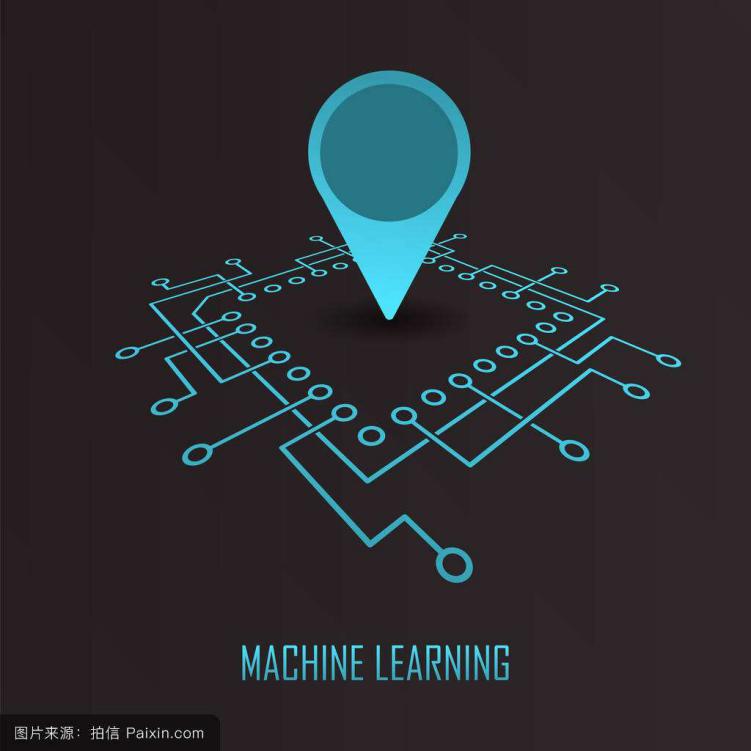
机器学习



目录

[第一章TensorFlow 3](#_Toc14336)

[1.1 Estimator 3](#_Toc17922)

[1.2 神经网络输入数据的规格要求 3](#_Toc15212)

[1.3 特征向量的设计 4](#_Toc3099)

[第二章 常用的函数 5](#_Toc4794)

[2.1将数据集划分为训练集和验证集 5](#_Toc12939)

[2.2 二维卷积函数 5](#_Toc7937)

[2.3 压缩纬数 7](#_Toc275)

[2.4 获取当前工作路径 7](#_Toc750)

[D:\pywork\gitWork 8](#_Toc27097)

[2.5 保存python中的序列数据--使用pickle模块 8](#_Toc406)

[第三章 细节 8](#_Toc24344)

[3.1 一维数组转置无效 8](#_Toc31190)

[a = np.random.normal(0, 1, 4) 8](#_Toc27522)

[a: 9](#_Toc32185)

[第四章 NLP 9](#_Toc19088)

[4.1 单词嵌入（Word Embedding） 9](#_Toc28172)

[4.2 数据预处理 9](#_Toc26692)

[4.2.1 VocabularyProcessor函数 9](#_Toc900)

[专业单词 10](#_Toc11322)

# 第一章TensorFlow

## Estimator

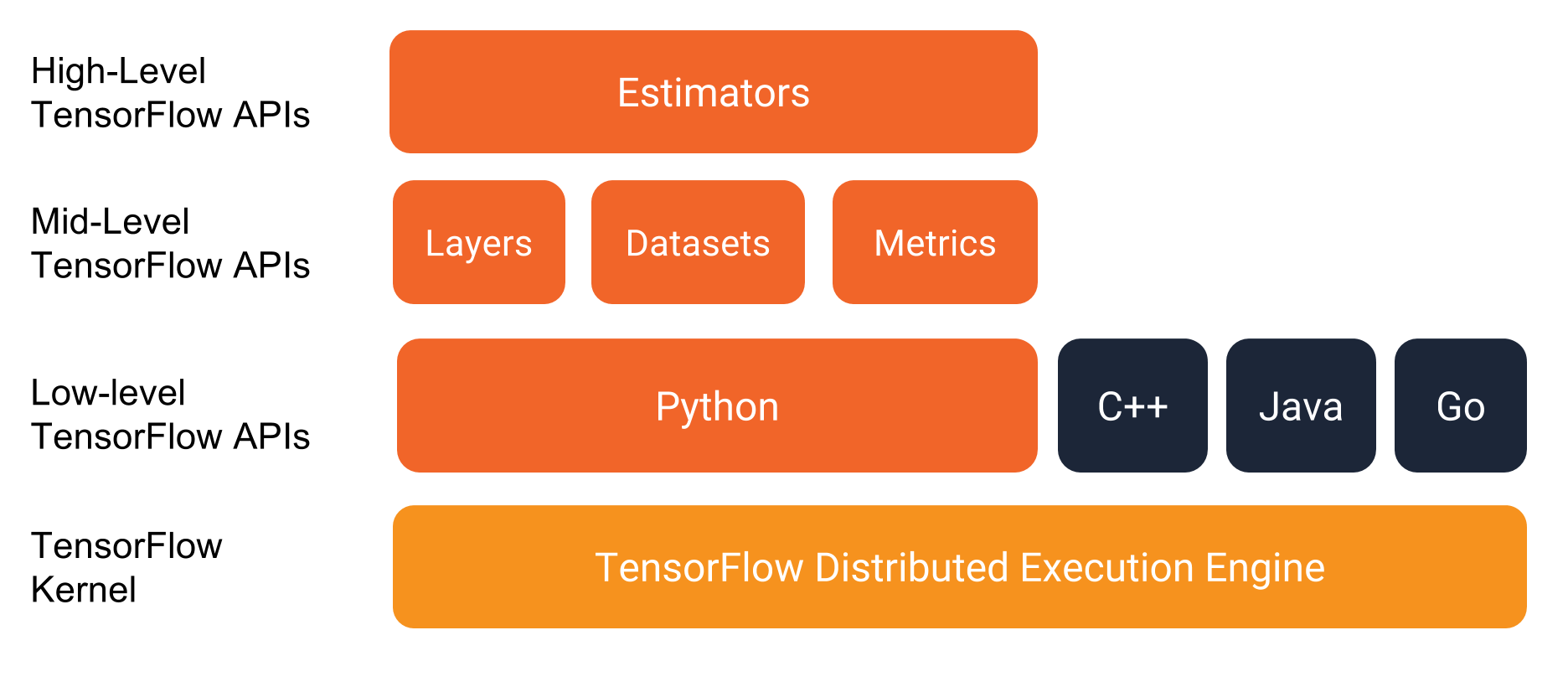


图1.1 TensorFlow的编程结构图

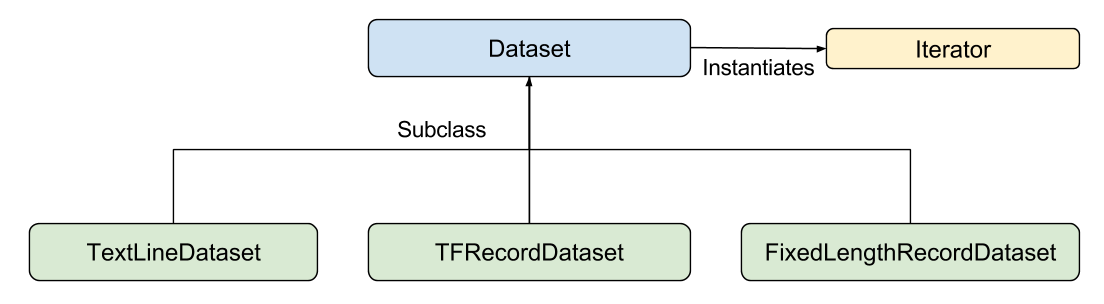


图1.2 Datasets API包含的类

## 神经网络输入数据的规格要求

Note that many of TensorFlow's layer functions are designed to deal with 4D data (4D = [batch size, width, height, channels]). We will need to modify our input data and the output data to extend or collapse the extra dimensions needed. For our example data, we have a batch size of 1, a width of 1, a height of 25, and a channel size of 1. To expand dimensions, we use the tf.expand\_dims() function, and to collapse dimensions, we use the tf.squeeze() function. Also note that we can calculate the output dimensions of convolutional layers from the formula: output\_size=(W-F+2P)/S+1（注意：这里的计算类似于numpy中所谓数组的运算，即对应元素相运算）, where W is the input size, F is the filter size, P is the padding size, and S is the stride size.

## 特征向量的设计

将不同的单词转化为特征向量通常使用one-hot向量，例如词库中有三个单词，线将单词排好序，然后用3个长度等于词库数量的一维向量表示这三个单词，排序后的第i个单词对应的向量为：第i列的值为1，其余位置的值为0，例如第一单词对应的向量是（1,0,0），这样表示的一个好处是他们是稀疏向量。

## tf.tensor的常用的数学计算函数

### tf.reduce\_?

Reduction is an operation that applies an operation across one of the tensor's dimensions, leaving it with one less dimension.

The supported operations include (with the same parameters) product, minimum, maximum,mean, all, any, and accumulate\_n).

# 第二章 常用的函数

## 2.1将数据集划分为训练集和验证集

获得任意打乱的数据集的下标

train\_index = np.random.choice(32, round(32\*0.8), replace=False)

print(list(train\_index))

print(len(train\_index))

test\_index = list(set(range(32))-set(train\_index))

print(test\_index)

print(len(test\_index))

输出：

[0, 15, 30, 21, 18, 6, 9, 29, 28, 27, 7, 14, 23, 31, 3, 24, 10, 13, 11, 5, 20, 1, 22, 8, 12,19]

26

[2, 4, 16, 17, 25, 26]

6

## 2.2 二维卷积函数

import tensorflow as tf

# tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, name=None)

# 除去name参数用以指定该操作的name，与方法有关的一共五个参数：

#

# 第一个参数input：指需要做卷积的输入图像，它要求是一个Tensor，具有[batch, in\_height, in\_width, in\_channels]这样的shape，具体含义是[训练时一个batch的图片数量, 图片高度, 图片宽度, 图像通道数]，注意这是一个4维的Tensor，要求类型为float32和float64其中之一

#

# 第二个参数filter：相当于CNN中的卷积核，它要求是一个Tensor，具有[filter\_height, filter\_width, in\_channels, out\_channels]这样的shape，具体含义是[卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求类型与参数input相同，有一个地方需要注意，第三维in\_channels，就是参数input的第四维

#

# 第三个参数strides：卷积时在图像每一维的步长，这是一个一维的向量，长度4

#

# 第四个参数padding：string类型的量，只能是"SAME","VALID"其中之一，这个值决定了不同的卷积方式（SAME表示卷积核可以停靠在图像的边缘，露出的部分用值填充的方式填补）

#

# 第五个参数：use\_cudnn\_on\_gpu:bool类型，是否使用cudnn加速，默认为true

#

# 结果返回一个Tensor，这个输出，就是我们常说的feature map

oplist=[]

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([1, 3, 3, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([1 ,1 , 5 ,1]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,1,1,1], use\_cudnn\_on\_gpu=False, padding='VALID')

oplist.append([op2, "case 2"])

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([1, 3, 3, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([3 ,3 , 5 ,1]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,1,1,1], use\_cudnn\_on\_gpu=False, padding='VALID')

oplist.append([op2, "case 3"])

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([1, 5, 5, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([3 ,3 , 5 ,1]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,1,1,1], use\_cudnn\_on\_gpu=False, padding='VALID')

oplist.append([op2, "case 4"])

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([1, 5, 5, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([3 ,3 , 5 ,1]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,1,1,1], use\_cudnn\_on\_gpu=False, padding='SAME')

oplist.append([op2, "case 5"])

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([1, 5, 5, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([3 ,3 , 5 ,7]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,1,1,1], use\_cudnn\_on\_gpu=False, padding='SAME')

oplist.append([op2, "case 6"])

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([1, 5, 5, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([3 ,3 , 5 ,7]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,2,2,1], use\_cudnn\_on\_gpu=False, padding='SAME')

oplist.append([op2, "case 7"])

# [batch, in\_height, in\_width, in\_channels]

input\_arg = tf.Variable(tf.ones([4, 5, 5, 5]))

# [filter\_height, filter\_width, in\_channels, out\_channels]

filter\_arg = tf.Variable(tf.ones([3 ,3 , 5 ,7]))

op2 = tf.nn.conv2d(input\_arg, filter\_arg, strides=[1,2,2,1], use\_cudnn\_on\_gpu=False, padding='SAME')

oplist.append([op2, "case 8"])

with tf.Session() as a\_sess:

a\_sess.run(tf.global\_variables\_initializer())

for aop in oplist:

print("----------{}---------".format(aop[1]))

print(a\_sess.run(aop[0]))

print('---------------------\n\n')

## 2.3 压缩纬数

tf.squeeze(x\_data)：假如x\_data的shape为[1, 1, 1, 4]，则压缩后shape为[4]

假如x\_data的shape为[2, 1, 1, 4]，则压缩后shape为[2, 4]

## 2.4 获取当前工作路径

import os

d1 = os.path.dirname(\_\_file\_\_) #\_\_file\_\_是该脚本的一个属性，这条命令就是获得该脚本所在目录的绝对路径

print(d1)

cwd = os.getcwd() #获得当前工作目录的绝对路径

print(cwd)

print(os.path.dirname(cwd)) #打印当前工作目录上一层目录的绝对路径

print(os.path.realpath(\_\_file\_\_)) #获得包含脚本名的绝对路径

输出：

D:/pywork/gitWork/tensorflow\_cookbook-master

D:\pywork\gitWork\tensorflow\_cookbook-master

D:\pywork\gitWork

D:\pywork\gitWork\tensorflow\_cookbook-master\test\_script.py

## 2.5 保存python中的序列数据--使用pickle模块

python的pickle模块实现了基本的数据序列和反序列化。通过pickle模块的序列化操作我们能够将程序中运行的对象信息保存到文件中去，永久存储；通过pickle模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

　　基本接口：

　　pickle.dump(obj, file, [,protocol])

　　注解：将对象obj保存到文件file中去。

　　　　　protocol为序列化使用的协议版本，0：ASCII协议，所序列化的对象使用可打印的ASCII码表示；1：老式的二进制协议；2：2.3版本引入的新二进制协议，较以前的更高效。其中协议0和1兼容老版本的python。protocol默认值为0。

　　　　　file：对象保存到的类文件对象。file必须有write()接口， file可以是一个以'w'方式打开的文件或者一个StringIO对象或者其他任何实现write()接口的对象。如果protocol>=1，文件对象需要是二进制模式打开的。

　　pickle.load(file)

　　注解：从file中读取一个字符串，并将它重构为原来的python对象。

　　file:类文件对象，有read()和readline()接口。

# 细节

## 3.1 一维数组转置无效

a = np.random.normal(0, 1, 4)

print('a:')

print(a)

print('transpose(a):')

print(np.transpose(a))

print(type(a))

print('transpose([a]):')

print(np.transpose([a]))

print(type([a]))

输出：

a:

[-0.27116264 -0.10124029 -2.29855456 -0.65075512]

transpose(a):

[-0.27116264 -0.10124029 -2.29855456 -0.65075512]

<class 'numpy.ndarray'>

transpose([a]):

[[-0.27116264]

[-0.10124029]

[-2.29855456]

[-0.65075512]]

<class 'list'>

# NLP

## 4.1 单词嵌入（Word Embedding）

词嵌入（Word embeddings）是一组广泛应用于预测NLP建模的特征工程技术，特别是在深度学习应用中的使用更为显著。词嵌入是将词的稀疏向量表示转换为密集、连续的向量空间，使你能够识别单词和短语之间的相似性，而这一点很大程度上依赖于它们的语境。

## 4.2 数据预处理

### 4.2.1 VocabularyProcessor函数

tf.contrib.learn.preprocessing.VocabularyProcessor(max\_sequence\_length, min\_frequency=min\_word\_frequency)

Lets say that you have just two documents I like pizza and I like Pasta. Your whole vocabulary consists of these words (I, like, pizza, pasta) For every word in the vocabulary, there is an index associated like so (1, 2, 3, 4). Now given a document like I like pasta it can be converted into a vector [1, 2, 4]. This is what the learn.preprocessing.VocabularyProcessor does. The parameter max\_document\_length makes sure that all the documents are represented by a vector of length max\_document\_length either by padding numbers if their length is shorter than max\_document\_length and clipping them if their length is greater than max\_document\_length Hope this helps you

# 专业单词

scalar：标量 parse：解析，从语法上解析 session：会议，学期

placeholder：占位符 truncate：截断 determinant：行列式

neural network：神经网络（nn） padding：填补