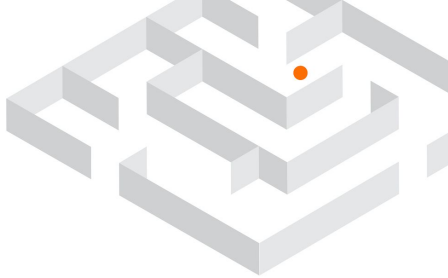
如何高效地使用TensorFlow

叶俊鹏



关于我

喜欢唱, 跳, Rap, 篮球

我是美图科技的NLP工程师, 机器学习谷歌开发者专家, 也是TFUG深圳的组织者.

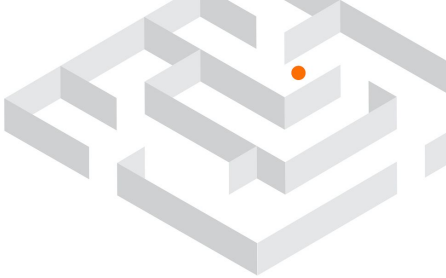
主要工作方向和兴趣是所有机器学习, 深度学习, NLP相关的技术

最近正在练习写作, 分享我的知识和感想给更多的人

欢迎关注我的Github和知乎专栏
https://github.com/JayYip
https://zhuanlan.zhihu.com/c_1116453746221309952

我也的确是喜欢Rap和篮球的

联系方式: junpang.yip@gmail.com



Agenda

- TensorFlow简介
- 动手构建TF模型
- 高效的数据处理流
- 动态图VS静态图
- 高效地预测
- 高层API使用建议

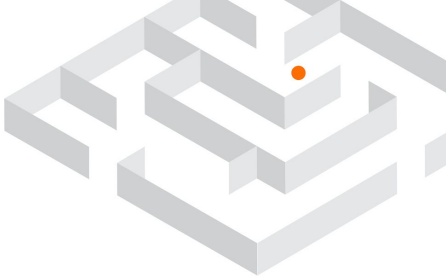
TensorFlow

TensorFlow(TF)简介

Mission Impossible

TensorFlow is an open source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them.

- From TF Github Repo

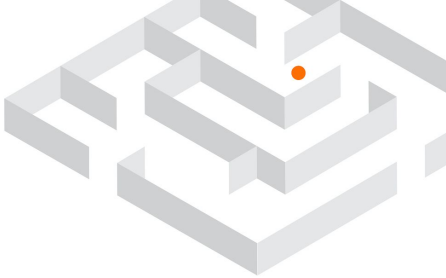


什么是TF?

一个计算软件

TF通过用户传给它的指令, 去构建计算图, 然后在不同的计算设备上面去执行这个计算图. 这个计算图的节点是计算操作, 边是流动方向.

Tensor中文叫张量也就是多维矩阵, Flow是流, 因此TensorFlow就是以张量计算以及流动为基础的一个计算软件

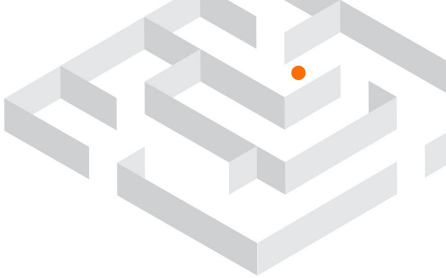


计算图

一个不恰当的比喻

想象计算图是一个手机工厂的流水线
节点是工人操作
边是货物流动方向
进入流水线的是一堆零件(原始数据)
流水线输出的是一部手机(预测结果)

作为构造流水线的工程师, 你需要有真正零件来之前构造好流水线, 你需要在构造流水线的时候想象有真的零件的情况, 来合理规划流水线. 一旦流水线确定好了, 开始运作后, 流水线将不可更改(静态图)



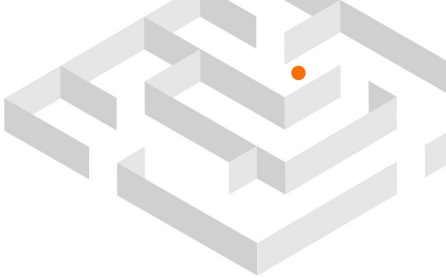
Operation, Tensor, Variable

一个不恰当的比喻

Operation, Tensor和Variable是TF中最重要的三个概念
Operation: 也就是对数据的操作, 回到比喻就是工人的操作, 比如安装电池
Tensor: 数据, 零件, 比如电池
Variable: 模型参数, 可能算是工人的大脑?

大部分的Op都会产生Tensor, 但是这并不是一定的, 比如tf.Print, 它只有操作, 并不会产生Tensor

Variable和Tensor的本质都是矩阵, 但是Variable是可更新的(大脑会学习), Tensor是不可更新的(电池本身不会学习), Variable是多次使用的(工人一直用一个大脑), Tensor是单次使用的(一个电池不能装在两个手机上)



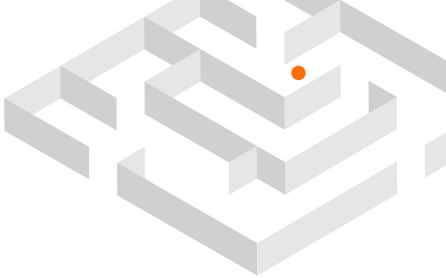
如何构建计算图

基础计算API

在TF中, 你可以使用基础的计算API来构建计算图, 比如加减乘除, 基本矩阵运算

你可以想象这些基础的计算是一个生活在水深火热中的底层工人, 他只会一个工作, 比如A工人只会装电池, B工人检测电池安装是否稳固.

只使用这些API能够构建任意的计算图, 但是这会让你的代码非常冗长



如何构建计算图

高级API

因此TF为构建深度学习模型提供了很多有用的高级API.
高级API可以想象成小组长, 可以帮你管理多个底层工人, 来达到更加复杂的操作. 而你只需要管理好小组长就可以了.

在TF1.X时代, 混乱的高级API是TF一直以来的一个痛处, 在TF2.0, 所有高级API都会放在tf.keras下

Talk is cheap. Show me the code.

- Linus Torvalds

TensorFlow

动手构建TF模型

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(input_shape=(28, 28)),
 tf.keras.layers.Dense(128, activation='relu'),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

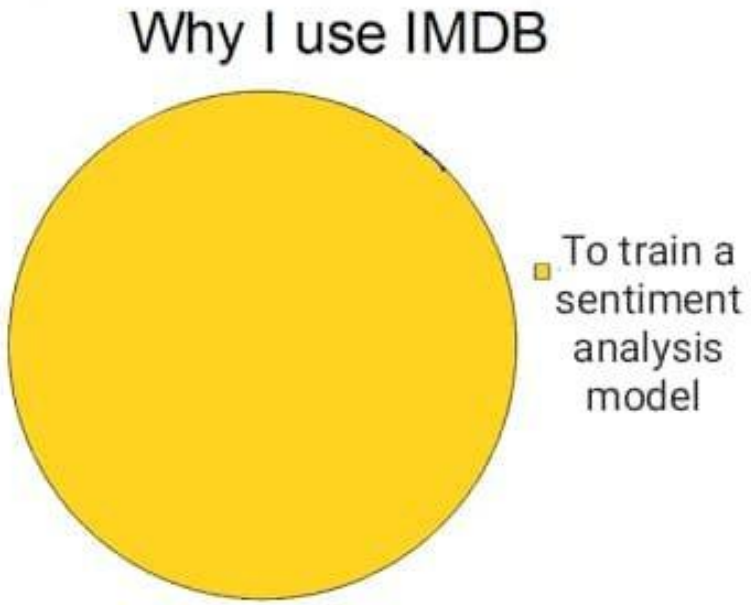
IMDB电影评论分类

1+1学完了, 是时候考微积分了

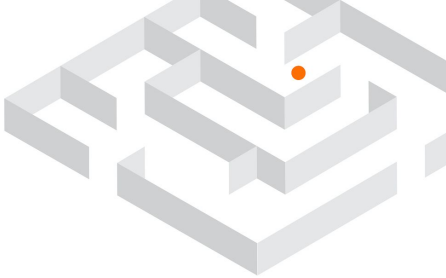
IMDB是blablabla… (who cares?)
根据用户对电影评论建立模型识别该评论是正面还是负面

输入: i love this movie!
输出: positive

输入: this movie is shit!
输出: negative



https://www.kaggle.com/yipjunpang/colab-practice



https://www.kaggle.com/yipjunpang/colab-practice

TensorFlow

高效的数据处理流

tf.data使用建议

高效的数据处理流

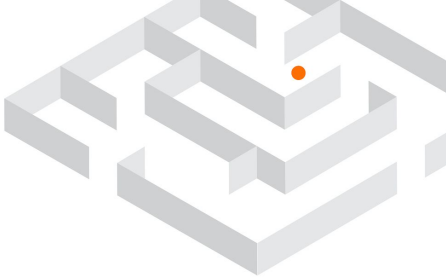
tf.data

tf.data是TF提供的高效的数据预处理工具, 利用它可以对数据进行高效的batching, shuffle, prefetch等功能.

使用tf.data首先要生成一个tf.data.Dataset对象
我个人最常用的构建tf.data.Dataset的方式是以下三种:

1. 用python构建一个generator, 然后使用tf.data.Dataset.from_generator(最方便, 最慢或者一般快)
2. 生成TFRecords, 然后使用tf.data.Dataset.TFRecordDataset(最麻烦, 最快)
3. 用tf.data.TextLineDataset先构建原始数据dataset, 再用map方法处理, 可以方便地用多线程(一般方便, 一般)

偷懒的话也可以用from_tensor_slice直接读numpy



方法1: generator

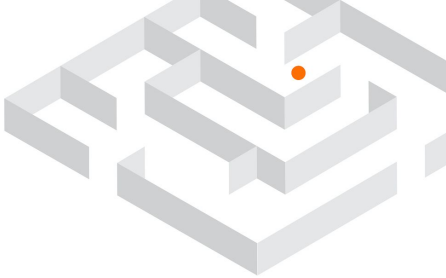
说好的高效呢?

1. 用python做好所有预处理, 然后用yield每一个数据, 构建generator(比如g)
2. 用一个没有输入参数的函数包起来: lambda: return g
3. 确定generator的output_type和output_shape, 它们和g返回的元素应有相同的结构
4. 构建dataset

Tips: 由于generator不便于并行化, 因此预处理逻辑都是单进程的. 如果性能是瓶颈的话建议用方法2或者将复杂逻辑用joblib等库并行化再返回generator. 如果内存是瓶颈的话可以使用pickler逐条写入逐条读入.

使用joblib并行:https://github.com/JayYip/bert-multitask-learning/blob/d4daa5423a42b9f624648dbe66ff40fd498751f8/bert_multitask_learning/create_generators.py#L283

```
def fake_gen():
   for i in range(1e4):
       yield {'test1': i, 'test2': [i, i+1], 'test3': {'test4': i+1}}

output_shapes = {'test1': [], 'test2': [2], 'test3': {'test4': []}}
output_types =  {'test1': tf.int32, 'test2': tf.int32, 'test3': {'test4':
tf.int32}}

dataset = tf.data.Dataset.from_generator(fake_gen,
output_types=output_types, output_shapes=output_shapes)
dataset = dataset.batch(32).shuffle(1000).prefetch(100)
```



方法2: TFRecord

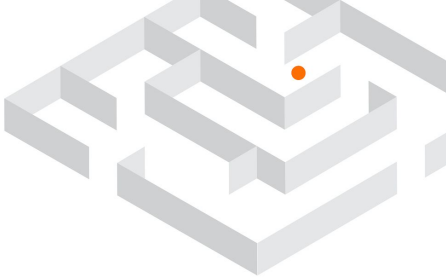
Binary file sucks, but you’re gonna love it

1. 用python做好所有预处理
2. 写serialize function
3. 将serialized feature写入到TFRecord中
4. 用tf.data.TFRecordDataset构建dataset
5. 写parsing函数, 并调用map方法来解析结果

例子:
Serialize并写入TFRecord:
https://github.com/re-search/DocProduct/blob/f68e926763b738d45fd513d55b2341a83efefe0c/docproduct/dataset.py#L67

Parsing函数:
https://github.com/re-search/DocProduct/blob/f68e926763b738d45fd513d55b2341a83efefe0c/docproduct/dataset.py#L337

```
with tf.io.TFRecordWriter(tf_record_file_path) as writer:
    for features in g:
        example = serialize_fn(features)
        writer.write(example)

dataset = tf.data.TFRecordDataset(tfrecord_file_list)

def _parse_example(example_proto):
    feature_description = ...
    feature_dict = tf.io.parse_single_example(
        example_proto, feature_description)
    return dense_feature_dict, feature_dict['labels']
dataset = dataset.map(_parse_example)

dataset = dataset.batch(32).shuffle(1000).prefetch(100)
```

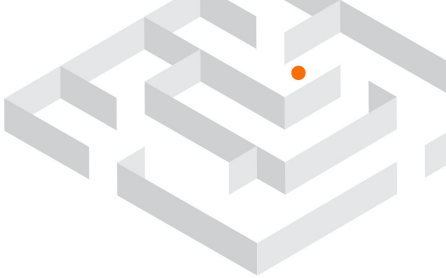


方法3: Map方法

自己摸索去吧

1. 用tf.data.Dataset.TextLineDataset读入原始文本
2. 用dataset.map(strange_fn, ...)

Tips:
用tf.py_func将复杂python预处理逻辑包起来



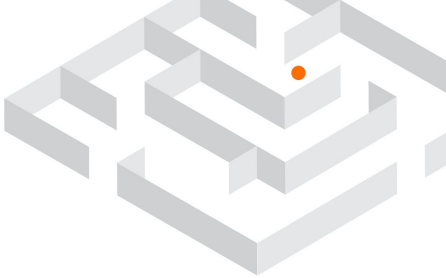
动手

将刚才的电影评论情感分析模型数据输入改写成使用tf.data API

TensorFlow

动态图VS静态图

灵活使用tf.function和AutoGraph

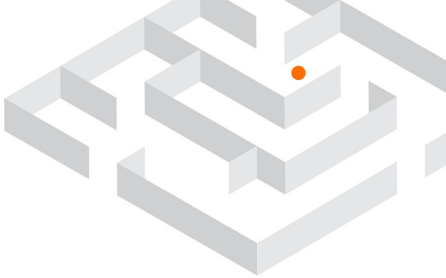


动态图

灵活, 用着爽

用回我们刚才流水线的比喻, 动态图相当于边生产边建生产线, 所以我们能够实时地看到每个工人(op)操作后的结果, 再决定下一个工人干什么.

- 在动态图中, 真正的计算操作是和python同步的, 因此可以看作在用一个支持GPU的功能更加丰富的numpy
- 但是, 在动态图下, 很多适用于数据流图的图优化就不能用了, 如果写模型的时候真的放飞自我的话, 运行效率比较捉急



静态图

高效, 稳定, 适合大规模部署

- 静态图相对开发难度会比动态图要大(符号式编程)
- 计算图建立好后, 可以进行多种图优化
- 更加利于大规模分布式训练和部署(本穷逼并没有试过)

动态图&静态图

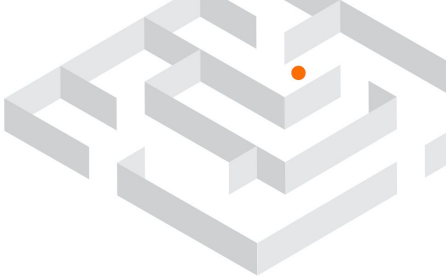
小孩子才做选择

Introducing tf.function and AutoGraph!

利用这两个工具, 我们能够在动态图下将代码转成图表示代码, 同时兼顾高效和灵活!

当然, 在官网看到的教程都是很美好很简单的, 但是在实际使用当中还是有许多暗雷

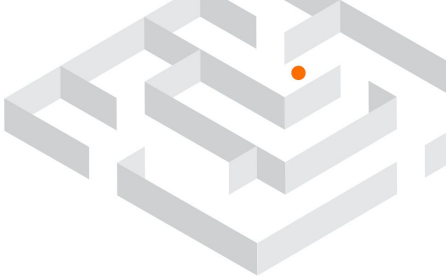
```
# A function is like an op

@tf.function
def add(a, b):
 return a + b

add(tf.ones([2, 2]), tf.ones([2, 2]))  #  [[2., 2.], [2., 2.]]
```



tf.function使用守则

欲知细节请移步知乎专栏

- 不要在被tf.function装饰的函数中初始化状态(tf.Variable), 可能会导致错误
- 尽可能使用tf.Tensor, 而不是python object, 可能会导致运行效率极大地下降
- 并不是所有运算都会被改写, 因此在可能的情况下, 使用tf的API而不是python的操作, 比如, 推荐使用tf.equal而不是==



动手

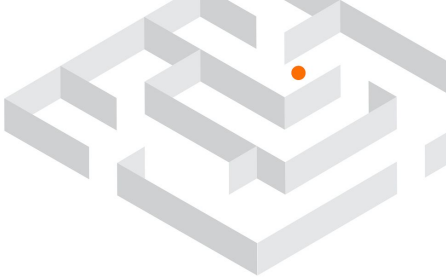
生死看淡, 不服就干

自己写一个Dense Layer, 如果权重矩阵的平均值大于零则在每次运行时打出, 最多打出10次, 并用tf.function转成图表示代码

TensorFlow

高效地预测

装着我很懂的样子



高效地预测

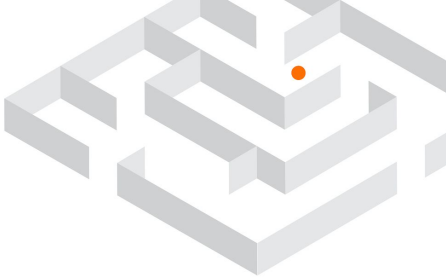
Graph Transforms

Ref:
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/graph_transforms

一般的优化操作:

- 计算节点排序, 方便构建最小inference图
- 去除无用节点
- 常数折叠, 用常数替换掉那些一直都是输出一个值的子图
- 合并重复节点
- Quantization, 压缩模型大小 **会对模型效果有影响
- And More! (我自己就用过这几种)

Example:
https://github.com/JayYip/bert-multitask-learning/blob/master/bert_multitask_learning/export_model.py#L61



高效地预测

Predictor & TF Serving

Predictor Ref:
https://www.tensorflow.org/api_docs/python/tf/contrib/predictor/from_estimator?hl=en

TF Serving:
https://www.tensorflow.org/tfx/serving/serving_basic?hl=en

Bert as Service:
https://github.com/hanxiao/bert-as-service
只需要少量改动, 就能serve任意的TF模型, 并且不需要模型构建代 码

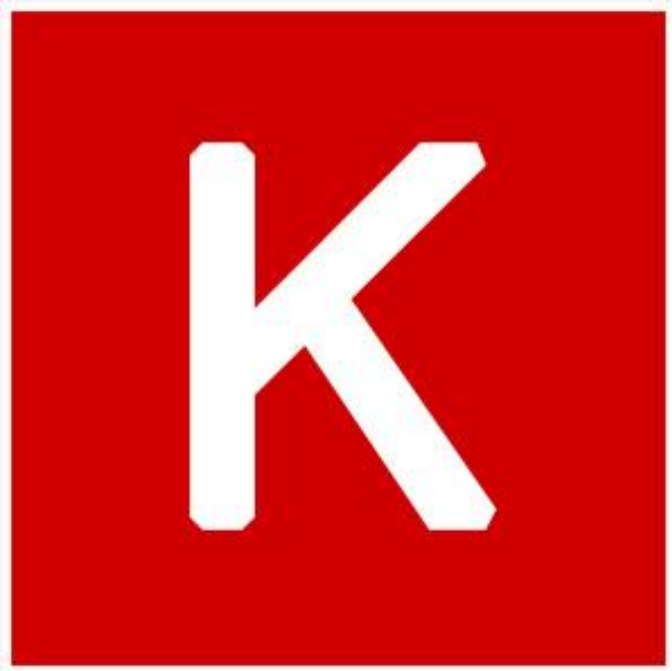
TensorFlow

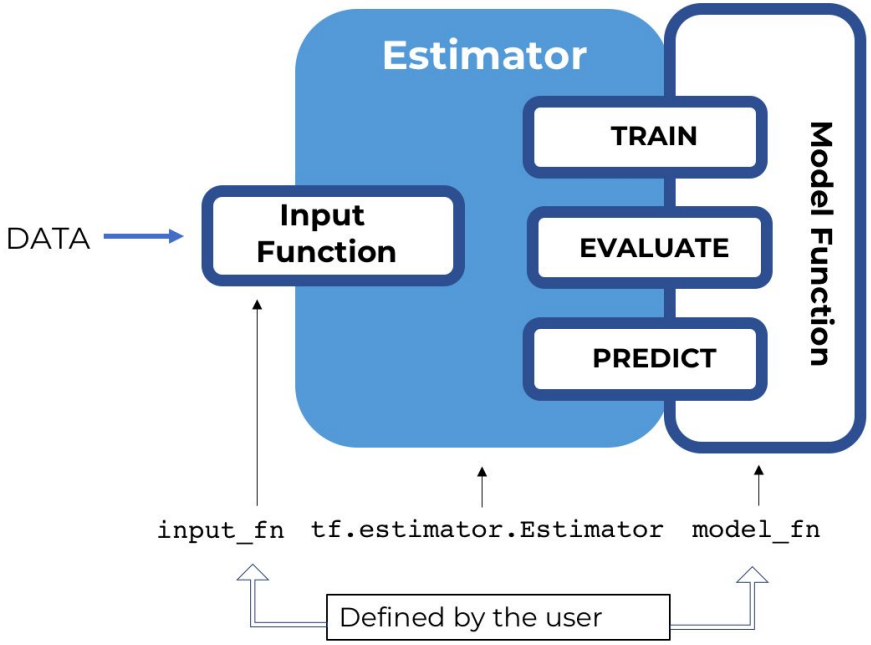
高层API使用建议

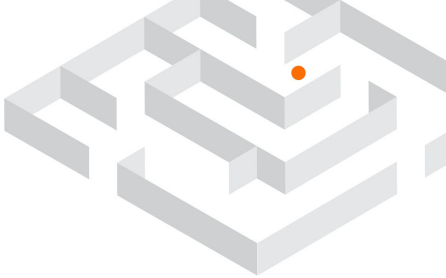
I LOVE ESTIMATOR!!!

Keras & Estimator

From keras import tensorflow







Estimator

Which is not recommended

你什么时候应该用Estimator?

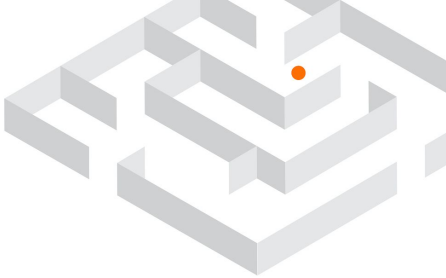
- 分布式, Mirror Strategy目前还是在Estimator上用比较顺滑
- 和我一样, 喜欢用dict/map做输入
- 已经用习惯

优点:

1. 灵活
2. 高效的分布式

缺点:

1. 相对比较难上手
2. 对epoch base的训练不友好
3. 似乎不能完全用上动态图的灵活性(待确认)



Keras

Functional API & Model Subclassing

TF 2.0官方推荐使用Keras来构建模型，扔掉小学生用的低级玩意，我们直接来看Keras里面的两种最灵活API: Functional API和Model Subclassing

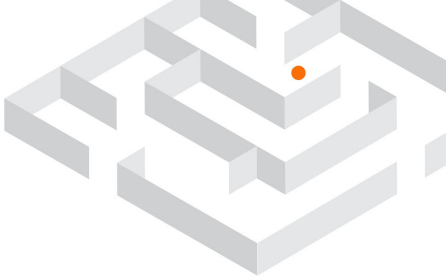
它们各自有各自的优缺点，我个人比较喜欢用Model Subclassing，因为它和PyTorch风格比较相近，更加符合面向对象编程的思想.

Key point: use the right API for the job. The Model subclassing API can provide you with greater flexbility for implementing complex models, but it comes at a cost (in addition to these missing features): it is more verbose, more complex, and has more opportunities for user errors. If possible, prefer using the functional API, which is more user-friendly.

- Keras Official Document

With great power comes great responsibility

- Uncle Ben



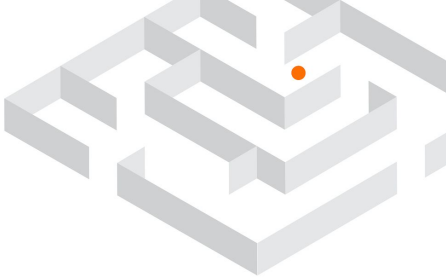
Keras

Functional API

你什么时候应该用Keras Functional API:

- 面向过程风格
- 在模型建好后不需要在 执行的时候有太灵活的操作
- 开箱即用各种Keras Model的API: fit, evaluate, predict, etc.

参考例子:
用Functional API实现的Keras BERT
https://github.com/CyberZHG/keras-bert



Keras

Model Subclassing

你什么时候应该用Keras Model Subclassing:

- 面向对象风格
- 将动态图的灵活性使用到极致
- 不需要或者愿意折腾着用各种Keras Model的API: fit, evaluate, predict, etc.
- 有一定的PyTorch经验并对其比较喜爱

参考例子:

用Model Subclassing实现的Keras BERT(注 该分支合并后链接可能失效)

https://github.com/re-search/DocProduct/blob/master/docproduct/bert.py

TensorFlow

Q&A

谢谢大家浪费时间听