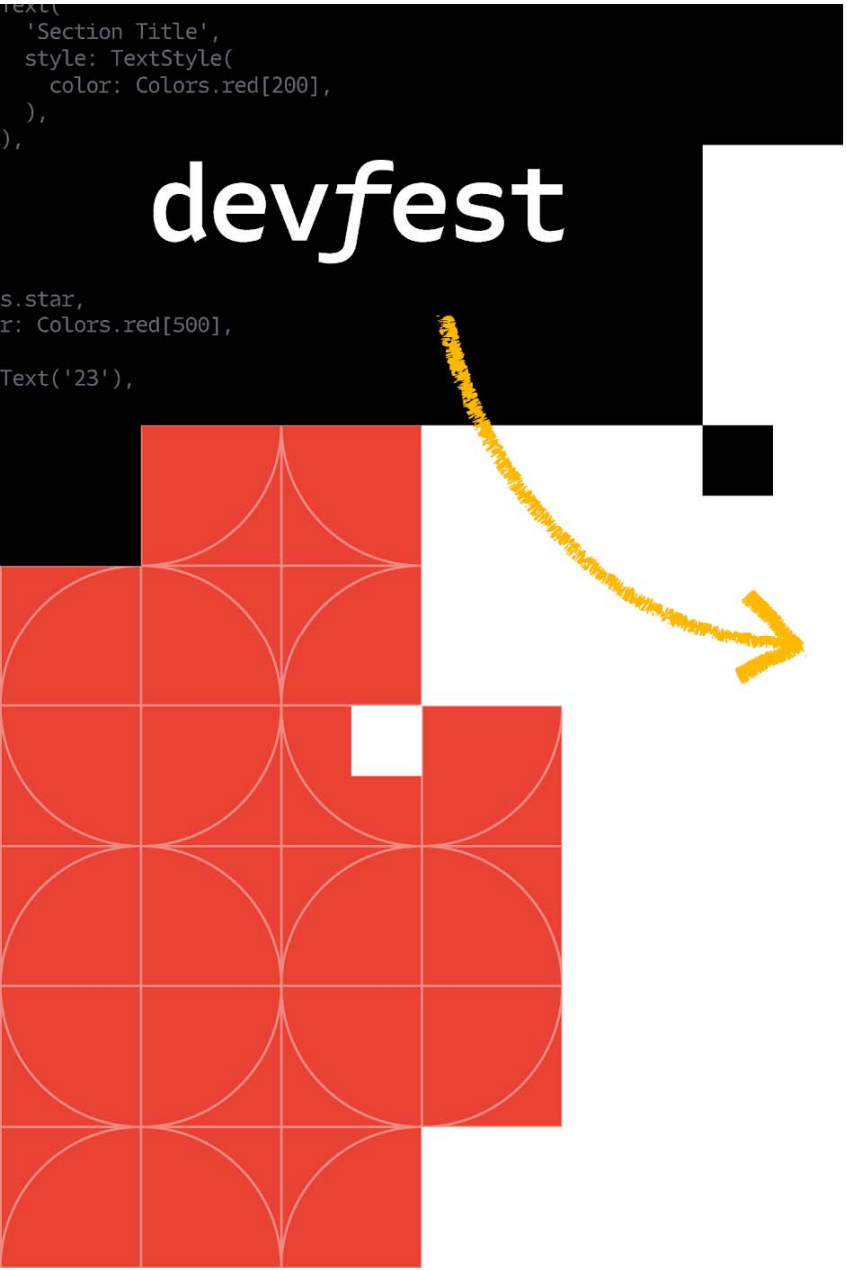
Keras Reloaded

Google Developer Groups

Beijing



Google Developer Groups

Beijing

分享内容

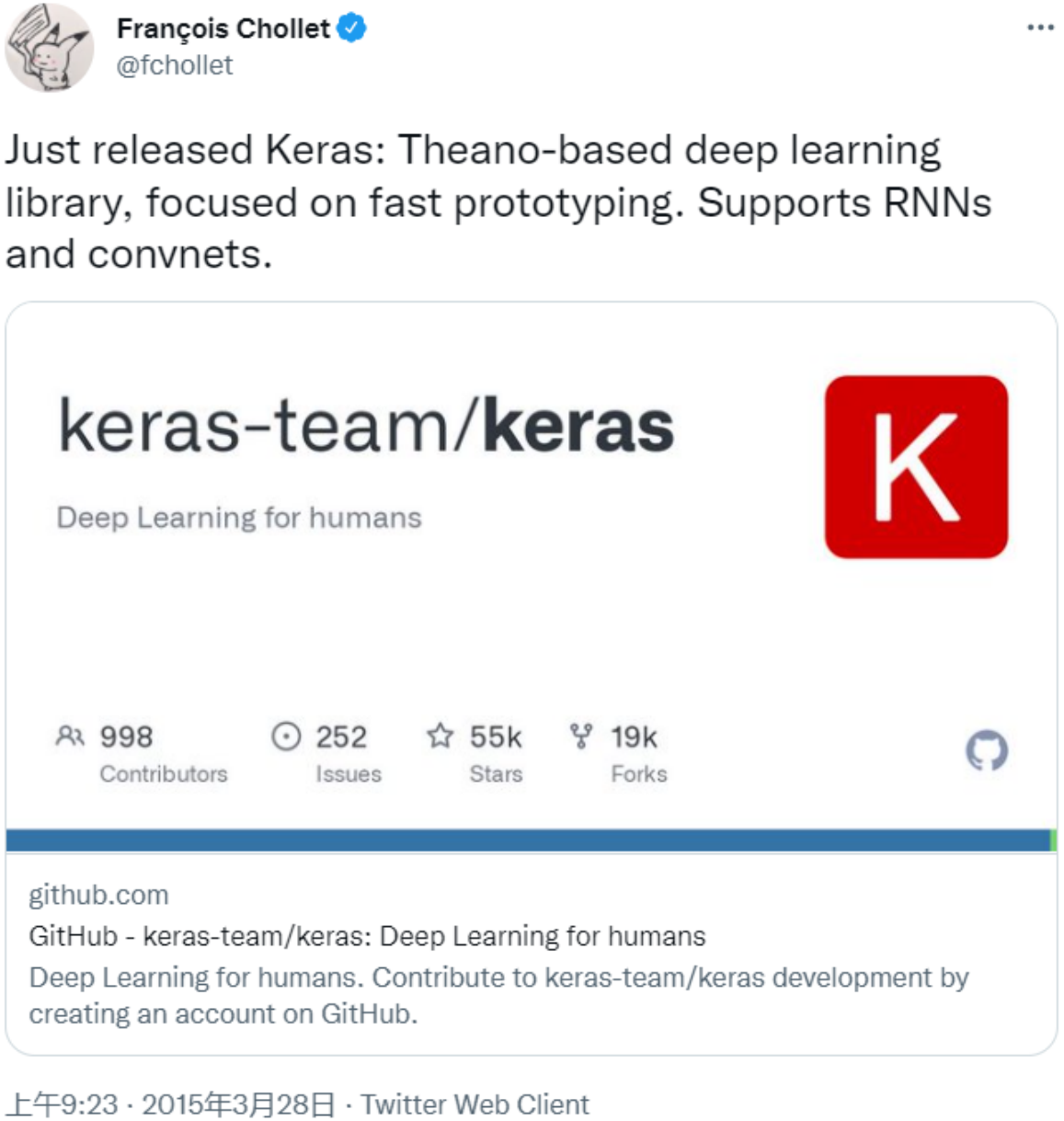
- 发展历程
- 新特性
- 已知问题提示
- 前景展望

Keras 时间简史

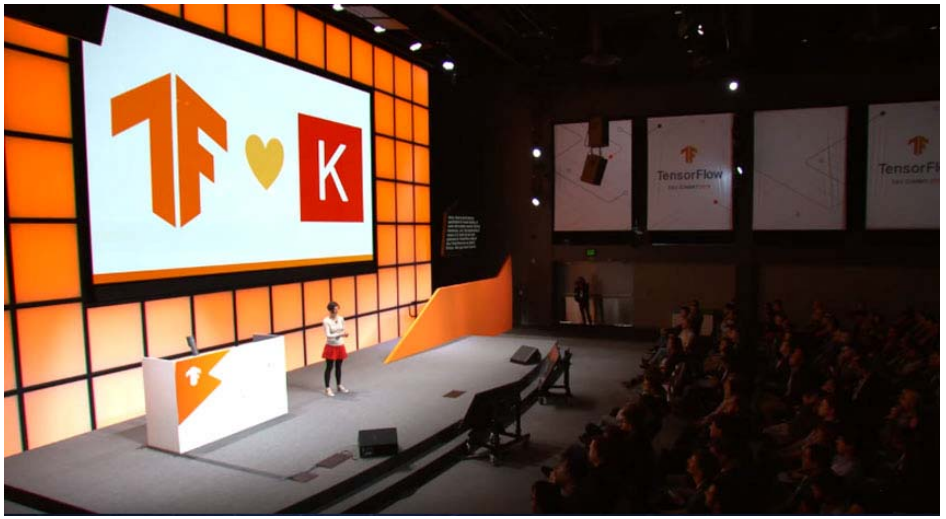
基于Theano的
Keras发布

Kera

201



Keras 时间简史



基于Theano的
Keras发布

Theano不再
更新

Kera
s

201
5

201
7

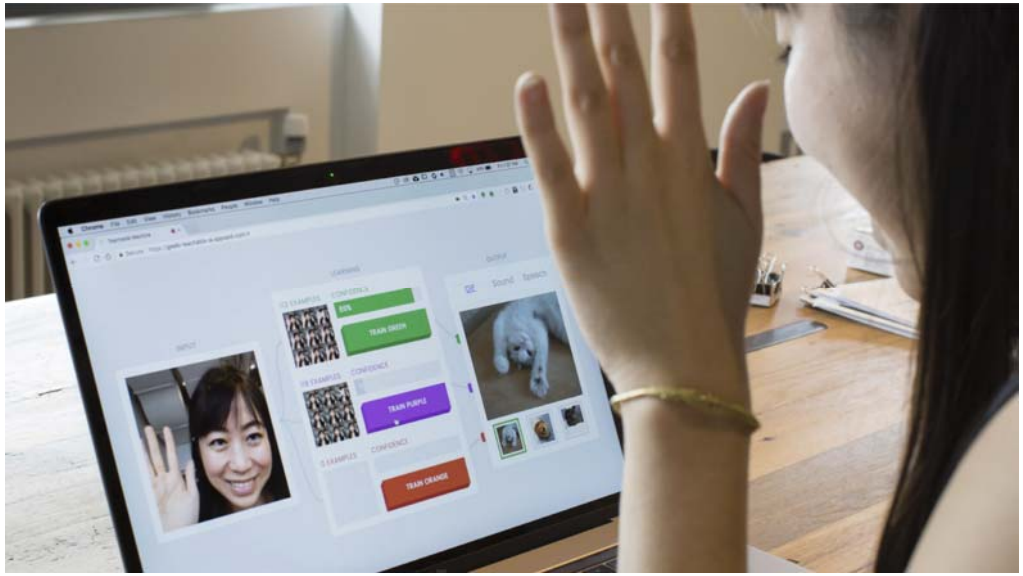
201
9

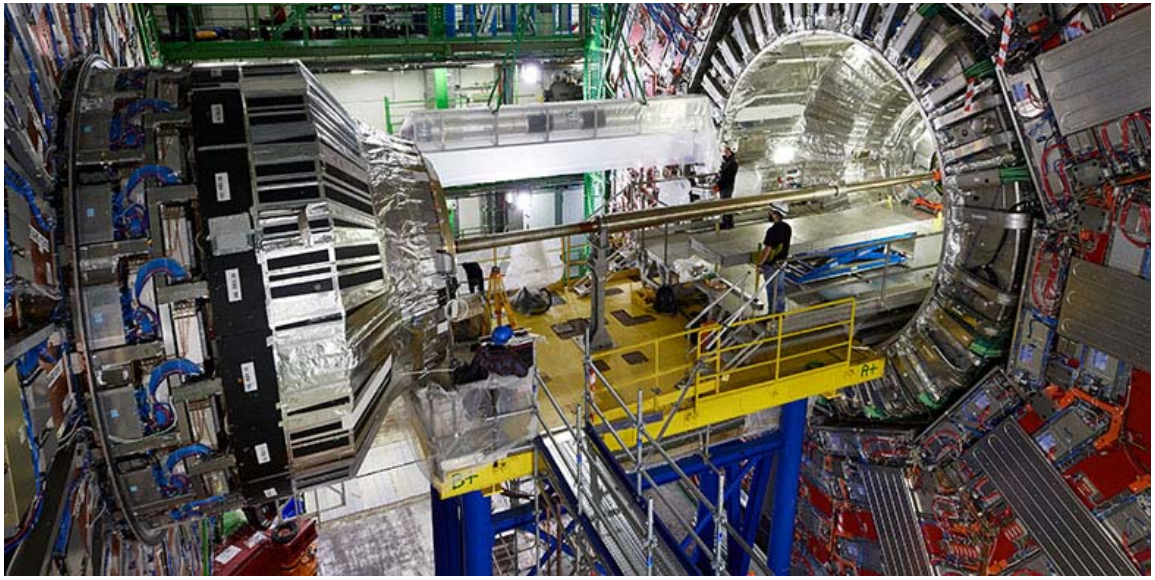
加入TensorFlow
支持

放弃多后端，并入
TensorFlow

Keras+
TensorFlo
w

TensorFlow 1.4
加入tf.keras


TRAIN GREEN
TRAIN PURPLE
TRAIN ORANGE



Keras 时间简史

基于Theano的Keras发布

Theano不再更新

Keras Core 公开发布

Kera

Keras Core 仓库开始更新

201

201

201

202

202

加入TensorFlow支持

放弃多后端，并入TensorFlow

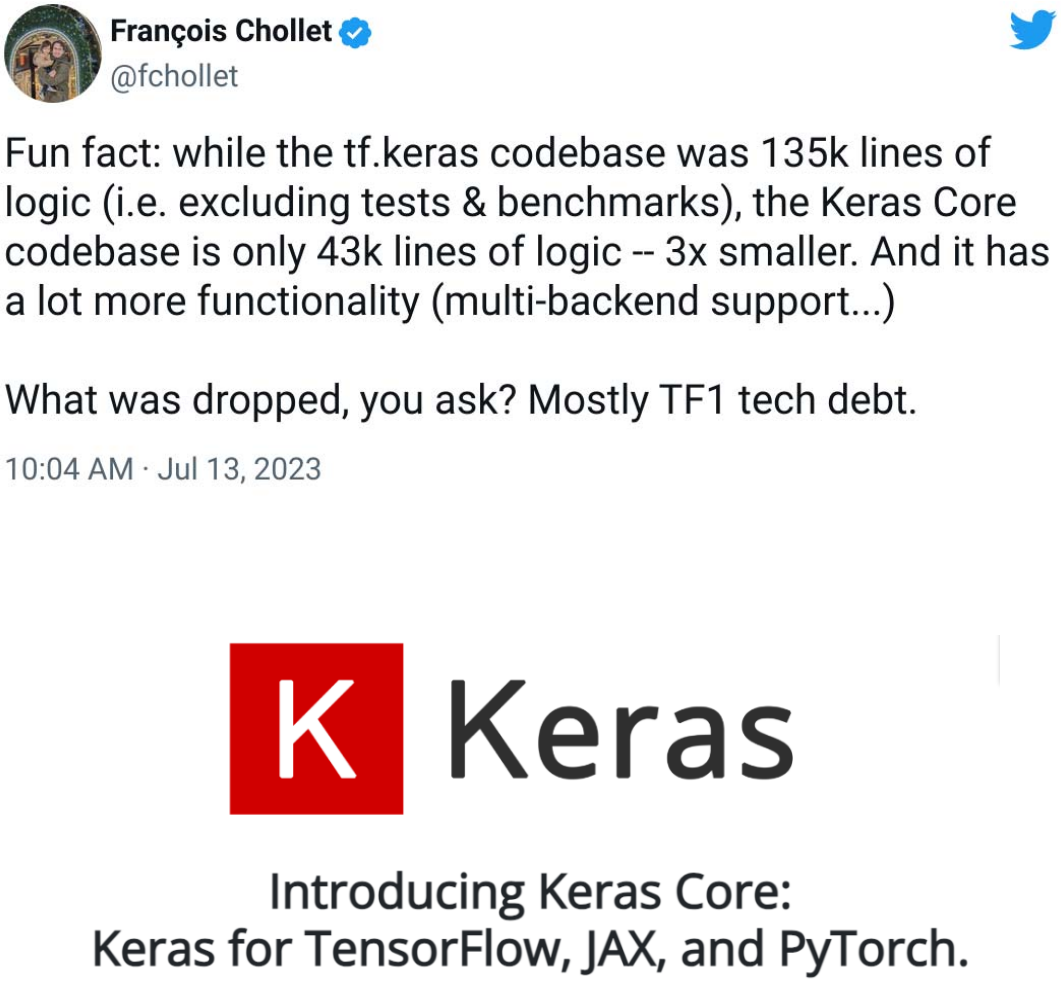
Keras+TensorFlow

TensorFlow 1.4加入tf.keras

Keras Core 里程碑

- 重写 Keras - loc 135k降到45k

- 支持 TensorFlow, JAX, PyTorch 等多个后端
  - 2023 StackOverflow Developer Survey
  - 2022 Kaggle Machine Learning & Data Science Survey
  - PyPI downloads
  - Conda downloads
  - Colab import statistics



Why Keras Core?

One API

- 用于每一个主流框架以避免生态的碎片化
- 从一种框架到另一种框架的无缝切换（Intuitive develop-Fastest deploy）

```
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras_core
import jax
import tensorflow as tf
```

```
import os
os.environ["KERAS_BACKEND"] = "torch"
import keras_core
import torch
import tensorflow as tf
```

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import keras_core as keras
```

```
>>> Using TensorFlow backend
```

```
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras_core as keras
```

```
>>> Using JAX backend
```

```
import os
os.environ["KERAS_BACKEND"] = "torch"
import keras_core as keras
```

```
>>> Using PyTorch backend
```

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import keras_core as keras
keras.ops.numpy.arange(5)
```

```
>>> <tf.Tensor: shape=(5,), dtype=int32, numpy=array([0, 1, 2, 3, 4], dtype=int32)>
```

```
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras_core as keras
keras.ops.numpy.arange(5)
```

```
>>> Array([0, 1, 2, 3, 4], dtype=int32)
```

```
import os
os.environ["KERAS_BACKEND"] = "torch"
import keras_core as keras
keras.ops.numpy.arange(5)
```

```
>>> tensor([0., 1., 2., 3., 4.])
```

```python
import keras_core as keras

model = keras.Sequential([
    keras.layers.Input(shape=(num_features,)),
    keras.layers.Dense(512, activation="relu"),
    keras.layers.Dense(512, activation="relu"),
    keras.layers.Dense(num_classes, activation="softmax"),
])
model.summary()

model.compile(
    optimizer=keras.optimizers.AdamW(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=[
        keras.metrics.CategoricalAccuracy(),
        keras.metrics.AUC(),
    ],
)

history = model.fit(
    x_train, y_train, batch_size=64, epochs=8, validation_split=0.2
)
evaluation_scores = model.evaluate(x_val, y_val, return_dict=True)
predictions = model.predict(x_test)
```

```
$ python example.py
Using TensorFlow backend
```

```
$ python example.py
Using PyTorch backend
```

```
$ python example.py
Using JAX backend
```

Keras Core 新特性

用于 TensorFlow, JAX, PyTorch 的整套 Keras API

- 只使用 built-in layers 的 Keras model 可以在所有支持的后端使用
- tf.keras model 可以在 JAX / PyTorch 运行，只需要把 import 从 keras 改成 keras_core

Keras Core 新特性

用于深度学习的跨框架的 low-level 语言

- 使用 keras_core.ops 的 namespace，支持各种后端，包括：
- near-full 实施的 NumPy API. 不只是 "NumPy-like" 的 NumPy API, 完全一样的函数和参数. 例如 ops.matmul, ops.sum, ops.stack, ops.einsum, 等.
- NumPy 之外的一系列神经网络函数, 例如 ops.softmax, ops.binary_crossentropy, ops.conv 等.
- keras_core.ops 来实现的定制化的 layers, losses, metrics, optimizers 支持各种后端，（只需要维护 model.py 和 checkpoint 文件）

Keras Core 新特性

无缝集成 JAX, PyTorch, TensorFlow 的 native workflows

- 过去定义 Keras model, Keras optimizer, Keras loss, Keras metrics, 然后调用 fit()/evaluate()/predict()
- 现在可以分别用 JAX/PyTorch/TensorFlow 的 training loop 去训练 Keras model
- 把Keras layer/model 作为 torch.nn.Module 的一部分
- Keras Core model 可以实例化为 PyTorch Module, 输出为 TensorFlow SavedModel, 或者实例化为 stateless JAX function.
- Keras Core models 可以利用 PyTorch packages, 也可以通过TensorFlow (TF-Serving, TF.js, TFLite) 部署, 还可以在JAX 大规模 TPU 训练.

Keras Core 新特性

支持跨所有后端框架的 data pipelines， Keras Core models 的 fit() / evaluate() / predict() 传统模式可以兼容不同 pipelines 来 loading 和 preprocessing，例如：

- tf.data.Dataset pipelines: 大规模 ML 的建议.
- torch.utils.data.DataLoader objects.
- NumPy arrays.
- Pandas dataframes.
- keras_core.utils.PyDataset objects.

- 使用 PyTorch DataLoader 训练 Keras Core + TensorFlow model 或 使用 tf.data.Dataset 训练 Keras Core + PyTorch model.

Keras Core 新特性

Progressive disclosure of complexity 渐进揭示复杂性

- Keras API 设计指导思想的核心
- 介于 fit() 和 GradientTape 之间，既想自定义训练算法，又调用 fit()，只需要重写 train_step 方法。
- 定制化 fit() / training loops / distribute training

Keras Core 新特性

新的 stateless API 用于 layers, models, metrics, optimizers

- 所有stateful objects 有了 stateless API 以用于 JAX functions （全部 stateless），
- layers/models: stateless_call() method 对应原来的 __call__().
- optimizers: stateless_apply() method 对应原来的 apply().
- metrics: stateless_update_state(), stateless_result() method 对应原来的 update_state(), result().

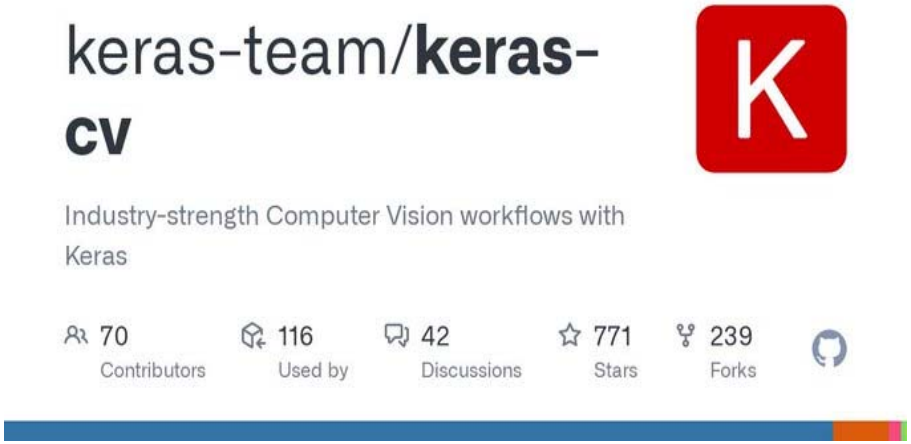
Keras Core 新特性

Pretrained models - 从发布即提供

- keras_core.applications namespace 中的 40+ Keras Applications models（PyTorch少一个）.
- KerasCV and KerasNLP (BERT, T5, YOLOv8, Whisper, etc.）.

配置方法

- KERAS_BACKEND 环境变量
- .keras/keras_cv.json 中 "multi_backend": True.



Quick look 典型使用方法

Classify text

```
model = BertClassifier.from_preset(
    "bert_base_en_uncased",
    num_classes=2,
)
model.compile(...)
model.fit(movie_review_dataset)
model.predict([
    "What an amazing movie!",
])
```

Text to image

```
model = StableDiffusion(
    img_width=512, img_height=512,
)
images = model.text_to_image(
    "photograph of an astronaut "
    "riding a horse",
    batch_size=3,
)
```

已知问题提示

和已有 tf.keras 的兼容性

- 由 tf.keras 开发的代码和预训练模型均可在使用 TensorFlow 后端的 Keras Core 中运行。
- 如果模型只使用 built-in 的 Keras layers，也可以在其它后端的 Keras Core 中运行。
- 目前的 Keras Core 会作为 keras package 发布在 PyPI，tf.keras 会 pointer 到它。

在一种后端保存的 Keras Core model，用另一种后端加载

- 只要保证 .keras 的文件格式（Keras v3）。
- 但是如果 model 里面包含自定义成分，那么要保证是使用 backend-agnostic（无关后端）APIs，如 keras.ops.

已知问题提示

import 顺序

- 在 TensorFlow 之前 import torch .
- 使用 torch 后端时, keras_core 需要 imports torch, keras_core 应该 import 先于 TensorFlow .

PyTorch Integer dtypes

- JAX 和 TensorFlow 的 dtypes uint16/32 使用 torch 后端时会转为 int32/64.

PyTorch 中的 Average pooling

- padding="same", 不同后端在最后行列结果不同.

已知问题提示

PyTorch 中的 Image layout 和 performance.

- 卷积网络的 image layout 在 cuDNN, TensorFlow, JAX中是 "channels_last" (NHWC), PyTorch 是 "channels_first". 可以通过 keras_core.config.set_image_data_format() 的 flag 来切换.
- 通过 torch.nn ops 直接传给 cuDNN 规避 channels 问题?

cuDNN 版本.

- 不同后端最新版本对 cuDNN 版本的要求不一，如果在GPU 上，同一个环境，不同后端，使用Keras Core ，需要保证同一个 cuDNN 版本对所有后端都有效。

已知问题提示

tf.data pipeline 中的 Keras layers/models.

- 使用 TensorFlow 后端， Keras layers/models 可以 .map() 到 tf.data pipeline，别的后端不可以.

稀疏 NN 的支持.

- 暂时缺乏 tf.keras 中的支持.

Cons & Pros

Cons

- 动态整合能力
- 为什么不直接学 TensorFlow/PyTorch?

Pros

- 增加开源模型 releases. 近似至少double?
- “喜新厌旧”的生态位

结论

Keras has no end?

参考

• Keras Core: Keras for TensorFlow, JAX, and PyTorch, https://keras.io/keras_core/

keras_core. thanks