

TPU训练

导师: GAUSS

目录

- 1/ TPU简介
- 2/ Keras方式TPU训练
- 3/ 自定义方式TPU训练
- 4/ 实战7：使用TPU进行花卉分类

TPU简介

TPU简介

TPU 代表 Tensor Processing Unit (张量处理单元)，是由谷歌在 2016 年 5 月发布的为机器学习而构建的定制集成电路 (ASIC)，并为 TensorFlow 量身定制。

早在 2015 年，谷歌大脑团队就成立了第一个 TPU 中心，为 Google Translation, Photos 和 Gmail 等产品提供支持。为了使所有数据科学家和开发人员能够访问此技术，不久之后就发布了易于使用，可扩展且功能强大的基于云的 TPU，以便在 Google Cloud 上运行 TensorFlow 模型。

TPU 由多个计算核心 (Tensor Core) 组成，其中包括标量，矢量和矩阵单元 (MXU)。TPU (张量处理单元) 与 CPU (中央处理单元) 和 GPU (图形处理单元) 最重要的区别是：TPU 的硬件专为线性代数而设计，线性代数是深度学习的基石。在过去几年中，Google TPU 已经发布了 v1, v2, v3, v2 Pod, v3 Pod, Edge 等多个版本：



版本	图片	性能	内存
TPU (v1, 2015)		92 TeraFLOPS	8 GB HBM
Cloud TPU (v2, 2017)		180 TeraFLOPS	64 GB HBM
Cloud TPU (v3, 2018)		420 TeraFLOPS	128 GB HBM
Cloud TPU Pod (v2, 2017)		11,500 TeraFLOPS	4,096 GB HBM
Cloud TPU Pod (v3, 2018)		100,000+ TeraFLOPS	32,768 GB HBM
Edge TPU (Coral, 2019)		4 TeraFLOPS	

为什么使用 TPU

通过使用 Cloud TPU，我们可以大大提升 TensorFlow 进行机器学习训练和预测的性能，并能够灵活的帮助研究人员，开发人员和企业 TensorFlow 计算群集。


根据 Google 提供的数据显示，在 Google Cloud TPU Pod 上可以仅用 8 分钟就能够完成 ResNet-50 模型的训练。

ResNet-50

	TPU	TPU Pod
训练速度（每秒图像数）	4000+	200,000+
最终精度	93%	93%
训练时长	7h 47m	8m 45s

TPU 性能

根据研究显示，TPU 比现代 GPU 和 CPU 快 15 到 30 倍。同时，TPU 还实现了比传统芯片更好的能耗效率，算力能耗比值提高了 30 倍至 80 倍。

每个周期的操作次数 

CPU	10
GPU	10,000
TPU	100,000



Source: [An in-depth look at Google's first Tensor Processing Unit \(TPU\)](#)

TPU使用

在 TPU 上进行 TensorFlow 分布式训练的核心 API 是 `tf.distribute.TPUStrategy`，可以简单几行代码就实现在 TPU 上的分布式训练，同时也可以很容易的迁移到 GPU 单机多卡、多机多卡的环境。以下是如何实例化 `TPUStrategy`：

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='grpc://' +  
os.environ['COLAB_TPU_ADDR'])  
tf.config.experimental_connect_to_cluster(resolver)  
# This is the TPU initialization code that has to be at the beginning.  
tf.tpu.experimental.initialize_tpu_system(resolver)  
print("All devices: ", tf.config.list_logical_devices('TPU'))
```

在上面的代码中，首先我们实例化 `TPUClusterResolver`；然后，我们连接 TPU Cluster，并对其进行初始化；最后，完成实例化 `TPUStrategy`。

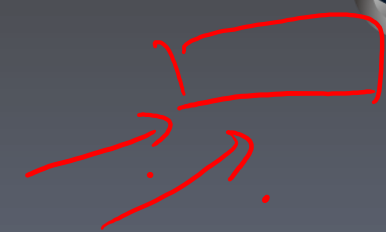
Keras方式TPU训练

输入数据集



深度之眼
deepshare.net

gcs
colab



tf.data.Dataset在使用Cloud TPU时，有效使用API至关重要，因为除非您可以足够快地提供数据，否则无法使用Cloud TPU。有关数据集性能的详细信息，请参见《输入管道性能指南》。

对于除最简单的实验（使用tf.data.Dataset.from_tensor_slices或其他图形数据）以外的所有实验，您都需要将数据集读取的所有数据文件存储在Google Cloud Storage (GCS) 存储桶中。



输入数据集

对于大多数用例，建议将数据转换为TFRecord格式并使用tf.data.TFRecordDataset读取。有关如何执行此操作的详细信息，请参见TFRecord和tf.Example教程。但是，这并不是硬性要求，如果愿意，可以使用其他数据集读取器（FixedLengthRecordDataset或TextLineDataset）。

小型数据集可以使用完全加载到内存中tf.data.Dataset.cache。

无论使用哪种数据格式，强烈建议您使用100MB左右的大文件。这在此网络设置中尤其重要，因为打开文件的开销明显更高。



输入数据集

在这里，您应该使用该~~tensorflow_datasets~~模块来获取MNIST训练数据的副本。
请注意，try_gcs已指定使用公共GCS存储桶中可用的副本。如果未指定，则TPU
将无法访问下载的数据。

gcs
try_gcs = True


```
def get_dataset(batch_size, is_training=True):  
    split = 'train' if is_training else 'test'  
    dataset, info = tfds.load(name='mnist', split=split, with_info=True,  
                             as_supervised=True, try_gcs=True)
```

```
def scale(image, label):  
    image = tf.cast(image, tf.float32)  
    image /= 255.0  
  
    return image, label
```

```
dataset = dataset.map(scale)
```

Only shuffle and repeat the dataset in training. The advantage to have a
infinite dataset for training is to avoid the potential last partial batch
in each epoch, so users don't need to think about scaling the gradients
based on the actual batch size.

```
if is_training:  
    dataset = dataset.shuffle(10000)  
    dataset = dataset.repeat()  
  
dataset = dataset.batch(batch_size)  
  
return dataset
```

+ corruption datasets

dataset

TPV



使用Keras高级API训练模型

#首先, 创建TPUStrategy对象。

```
strategy = tf.distribute.experimental.TPUStrategy(resolver)
```

```
with strategy.scope():
```

```
    model = create_model()
```

```
    model.compile(optimizer='adam')
```

```
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
        metrics=['sparse_categorical_accuracy'])
```

```
batch_size = 200
```

```
steps_per_epoch = 60000 // batch_size
```

```
train_dataset = get_dataset(batch_size, is_training=True)
```

```
test_dataset = get_dataset(batch_size, is_training=False)
```

```
model.fit(train_dataset,
```

```
        epochs=5,
```

```
        steps_per_epoch=steps_per_epoch,
```

```
        validation_data=test_dataset)
```

gcs!

TPU.

自定义方式TPU训练

数据集分发

分布式输入

- `experimental_distribute_datasets_from_function`
(更好的可伸缩性和性能。)
- `experimental_distribute_dataset`

数据集分发

- `experimental_distribute_datasets_from_function`

该API接受输入函数并返回分布式数据集实例。用户传入的输入函数有一个`tf.distribute.InputContext`参数，应该返回一个`tf.data.Dataset`实例。使用此API，`tf.distribute`不会对`tf.data.Dataset`从输入函数返回的用户实例进行任何进一步的更改。

- `experimental_distribute_dataset`

将`tf.data.Dataset`实例作为输入，并返回分布式数据集实例。你应该使用等于**全局批处理大小的值对输入数据集进行批处理**。此全局批处理大小是你要在所有设备中一步处理的样本数。

```
def get_dataset(batch_size, is_training=True):  
    split = 'train' if is_training else 'test'  
    dataset, info = tfds.load(name='mnist', split=split, with_info=True,  
                             as_supervised=True, try_gcs=True)
```

```
def scale(image, label):  
    image = tf.cast(image, tf.float32)  
    image /= 255.0
```

```
    return image, label
```

```
dataset = dataset.map(scale)
```

```
# Only shuffle and repeat the dataset in training. The advantage to have a  
# infinite dataset for training is to avoid the potential last partial batch  
# in each epoch, so users don't need to think about scaling the gradients  
# based on the actual batch size.
```

```
if is_training:  
    dataset = dataset.shuffle(10000)  
    dataset = dataset.repeat()
```

```
dataset = dataset.batch(batch_size)
```

```
return dataset
```

```
# Create the model, optimizer and metrics inside strategy scope, so that the  
# variables can be mirrored on each device.
```

```
with strategy.scope():  
    model = create_model()  
    optimizer = tf.keras.optimizers.Adam()  
    training_loss = tf.keras.metrics.Mean('training_loss', dtype=tf.float32)  
    training_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(  
        'training_accuracy', dtype=tf.float32)
```

```
# Calculate per replica batch size, and distribute the datasets on each TPU  
# worker.
```

```
per_replica_batch_size = batch_size // strategy.num_replicas_in_sync
```

```
#数据集分发
```

```
train_dataset = strategy.experimental_distribute_datasets_from_function(  
    lambda _: get_dataset(per_replica_batch_size, is_training=True))
```


损失函数计算

为什么要使用`tf.nn.compute_average_loss`?

您应该将每个示例损失相加并将总和除以 `GLOBAL_BATCH_SIZE` : `scale_loss = tf.reduce_sum(loss) * (1. / GLOBAL_BATCH_SIZE)` 或者你可以使用 `tf.nn.compute_average_loss` 来获取每个示例的损失, 可选的样本权重, 将 `GLOBAL_BATCH_SIZE` 作为参数, 并返回缩放的损失。

@tf.function

def train_step(iterator):

"""The step function for one training step"""

def step_fn(inputs):

"""The computation to run on each TPU device."""

images, labels = inputs

with tf.GradientTape() as tape:

logits = model(images, training=True)

loss = tf.keras.losses.sparse_categorical_crossentropy(

labels, logits, from_logits=True)

loss = tf.nn.compute_average_loss(loss, global_batch_size=batch_size)

grads = tape.gradient(loss, model.trainable_variables)

optimizer.apply_gradients(list(zip(grads, model.trainable_variables)))

training_loss.update_state(loss * strategy.num_replicas_in_sync)

training_accuracy.update_state(labels, logits)

#要复制计算以使其可以在所有TPU内核中运行，只需将其传递给strategy.runAPI

strategy.run(step_fn, args=(next(iterator),))

```
steps_per_eval = 10000 // batch_size
```

```
train_iterator = iter(train_dataset)
```

```
for epoch in range(5):
```

```
    print('Epoch: {}/5'.format(epoch))
```

```
    for step in range(steps_per_epoch):
```

```
        train_step(train_iterator)
```

```
    print('Current step: {}, training loss: {}, accuracy: {}'.format(
```

```
        optimizer.iterations.numpy(),
```

```
        round(float(training_loss.result()), 4),
```

```
        round(float(training_accuracy.result()) * 100, 2)))
```

```
    training_loss.reset_states()
```

```
    training_accuracy.reset_states()
```



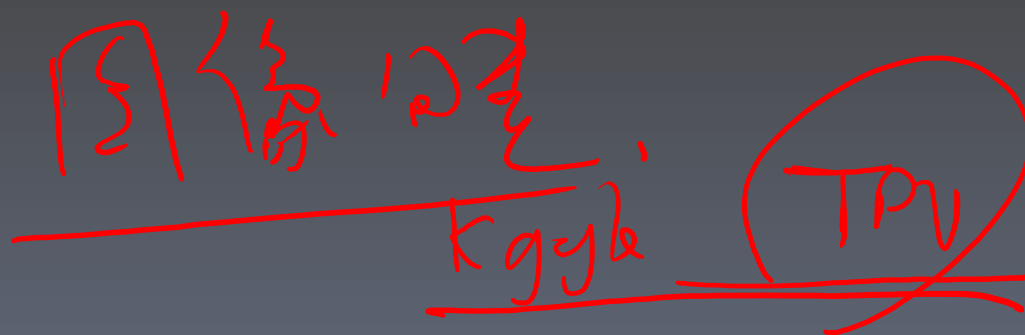

官方讲解

分布式训练: <https://www.tensorflow.org/tutorials/distribute/input>
https://www.tensorflow.org/tutorials/distribute/custom_training

实战7：使用TPU进行花卉分类



项目简介



赛题网址: <https://www.kaggle.com/c/flower-classification-with-tpus>

很难理解我们自然世界的广阔和多样性。这里有5,000多种哺乳动物, 10,000多种鸟类, 30,000多种鱼类, 而且惊人地有400,000多种不同类型的花。

在这场比赛中, 您面临建立一个机器学习模型的挑战, 该模型可识别图像数据集中的花朵类型 (为简单起见, 我们仅使用100多种类型的花朵)。



评价指标

评价指标为macro-F1:

F1 is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

数据集

我们将根据从五个不同的公共数据集中提取的图像对104种花朵进行分类

The screenshot displays the Google Cloud Data Explorer interface. On the left, a sidebar titled 'Data Explorer' shows a tree view of the dataset '4.8 GB'. The tree includes folders for 'tfrecords-jpeg-192x192', 'tfrecords-jpeg-224x224', 'tfrecords-jpeg-331x331', and 'tfrecords-jpeg-512x512', each containing 'test', 'train', and 'val' subfolders. A red box highlights the 'sample_submission.csv' file at the bottom of the list. The main panel shows the details for 'sample_submission.csv' (86.52 KB), including a 'Competition Rules' section with a laptop icon and a 'Download All' button. A red arrow points from the text '104种花朵' in the preceding paragraph to the 'sample_submission.csv' file in the sidebar.

Data Explorer
4.8 GB

- tfrecords-jpeg-192x192
 - test
 - train
 - val
- tfrecords-jpeg-224x224
- tfrecords-jpeg-331x331
- tfrecords-jpeg-512x512
- sample_submission.csv

sample_submission.csv (86.52 KB)

Competition Rules

To see this data you need to agree to the [competition rules](#).
By clicking "I understand and accept" you agree to be bound to these rules.

[I understand and agree](#)

Summary

- 193 files
- 2 columns

[Download All](#)

Colab

IT Reward

数据读取

tf.data.Dataset和TFRecords:

由于TPU的速度非常快，因此许多移植到TPU的型号最终都会出现数据瓶颈。TPU处于空闲状态，等待每个训练时期的大部分时间的数据。TPU仅从GCS（Google云存储）读取训练数据。如果GCS从多个文件并行并行流式传输，则可以保持相当大的吞吐量。遵循几个最佳实践将优化吞吐量：

对于TPU训练，请在GCS中以合理数量（10s至100s）的相当大的文件（10s至100s MB）组织数据。

TFRecords

数据读取

如果文件太少，GCS将没有足够的流来获得最大吞吐量。如果文件过多，访问每个文件都将浪费时间。

用于TPU训练的数据通常会分摊到适当数量的较大文件中。通常的容器格式是TFRecords。您可以通过编写以下内容从TFRecords文件加载数据集：

```
# On Kaggle you can also use KaggleDatasets().get_gcs_path() to obtain the GCS
path of a Kaggle dataset
filenames = tf.io.gfile.glob("gs://flowers-public/tfrecords-jpeg-
512x512/* tfrec") # list files on GCS
dataset = tf.data.TFRecordDataset(filenames)
dataset = dataset.map(...) # TFRecord decoding here...
```

image
512x512
192x192

数据读取

要启用来自多个TFRecord文件的并行流，请修改如下代码：

```
AUTO = tf.data.experimental.AUTOTUNE
ignore_order = tf.data.Options()
ignore_order.experimental_deterministic = False

# On Kaggle you can also use KaggleDatasets().get_gcs_path() to obtain the GCS
# path of a Kaggle dataset
filenames = tf.io.gfile.glob("gs://flowers-public/tfrecords-jpeg-
512x512/*.tfrec") # list files on GCS
dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=AUTO)
dataset = dataset.with_options(ignore_order)
dataset = dataset.map(...) # TFRecord decoding here...
```

这里有两个设置：

- ✓ `num_parallel_reads=AUTO`指示API从多个文件读取（如果有）。它会自动计算出多少。
- ✓ `experimental_deterministic = False`禁用数据顺序强制。无论如何，我们将对数据进行混洗，因此顺序并不重要。通过此设置，API可以在流入后立即使用任何TFRecord。

代码实战

以下部分均在<https://colab.research.google.com/>实战!

参考



深度之眼
deepshare.net

https://tf.wiki/zh_hans/appendix/tfhub.html#

<https://www.kaggle.com/c/flower-classification-with-tpus>

总结

本节小结

Summary

GPU分布式训练&TPU训练	TPU简介	
	Keras方式TPU训练	
	自定义方式TPU训练	
	实战7：使用TPU进行花卉分类	

结语

——我 说——

看过千万代码，不如实践一把！





深度之眼
deepshare.net

联系我们：

电话：18001992849

邮箱：service@deepshare.net

QQ：2677693114



公众号



客服微信

