УДК 004

Профилирование TPU в Colab

Бородулин Андрей Вадимович
Приамурский государственный университет им. Шолом-Алейхема
Студент

Аннотация
В данной статье исследуется процесс профилирования TensorProcessingUnit (TPU) в среде Colab. TPU - это специализированный аппаратный ускоритель, разработанный Google для обработки операций с тензорами. Colab, в свою очередь, предоставляет платформу для облачных вычислений, позволяющую пользователям запускать код Python и использовать аппаратное обеспечение Google, включая TPU. Профилирование TPU в Colab имеет важное значение для оптимизации производительности и эффективности вычислений. В статье рассматриваются методы и инструменты профилирования, позволяющие идентифицировать узкие места в коде, оптимизировать использование ресурсов TPU и снизить время выполнения задач. Результаты исследования показывают, что профилирование TPU в Colab может значительно повысить производительность и эффективность вычислений, особенно при работе с большими наборами данных и сложными моделями машинного обучения.
Ключевыеслова: TPU, TensorProcessingUnit, Colab, профилирование, производительность, эффективность, оптимизация, вычисления, машинное обучение.

TPUprofilinginColab

Borodulin Andrei Vadimovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article examines the Tensor Processing Unit (TPU) profiling process in the Colab environment. TPU is a specialized hardware accelerator developed by Google for processing tensor operations. Colab, in turn, provides a cloud computing platform that allows users to run Python code and use Google hardware, including TPU. TPU profiling in Colab is essential to optimize performance and computing efficiency. The article discusses profiling methods and tools that allow you to identify bottlenecks in the code, optimize the use of TPU resources and reduce task execution time. The results of the study show that TPU profiling in Colab can significantly improve computing performance and efficiency, especially when working with large datasets and complex machine learning models.
Keywords: TPU, Tensor Processing Unit, Colab, profiling, performance, efficiency, optimization, computing, machine learning.

1. Введение

1.1 Актуальность исследования

Рост популярности TPU: Специализированные процессоры TPU (TensorProcessingUnits) становятся все более распространенными в сфере машинного обучения и глубокого обучения. Исследование, посвященное профилированию TPU в Colab, актуально, так как оно поможет исследователям и разработчикам эффективно использовать и оптимизировать работу с TPU в данной среде.

1.2 Обзор исследований

Для создания данной статьи были рассмотрены онлайн ресурсы. Данная тема на сегодняшний день плохо изучена Российскими специалистам. Информация по данной теме была взята с зарубежных источников. MaëlFabien в своей статье описал что такое TPU [1]. Для изучения была взята статья с сайта MLCENTRE [2]. Так же был использован сайт TensorFlowBlog [3].

1.3 Цель исследования

Целью исследования является профилирование TPU в Colab

2. Материалы и методы

Среда выполнения Colab: Для проведения исследования использовалась облачная среда выполнения Colab, предоставляемая Google. Colab предоставляет возможность запуска кода Python в облачной среде с доступом к вычислительным ресурсам, включая TPU.

TensorFlow и TPU: В исследовании использовалась библиотека машинного обучения TensorFlow, которая поддерживает работу с TPU. TensorFlow предоставляет высокоуровневые API для создания и обучения моделей машинного обучения, а также специальные функции для работы с TPU.

Профилирование TPU: Для профилирования TPU в Colab использовались различные инструменты и методы. Это включало анализ времени выполнения операций, измерение использования ресурсов TPU, отслеживание памяти и другие аспекты производительности. Конкретные инструменты и методы описываются подробно в разделе "Инструменты профилирования".

Эксперименты и задачи: Для оценки производительности и эффективности профилирования TPU в Colab были проведены различные эксперименты на реальных задачах машинного обучения. Это могли быть задачи классификации, регрессии, обработки естественного языка и другие. Для каждой задачи описываются входные данные, архитектура модели и параметры обучения.

Анализ и интерпретация результатов: Полученные результаты профилирования TPU в Colab были анализированы и интерпретированы.

В целом, исследование использовало среду выполнения Colab, библиотеку TensorFlow и различные инструменты профилирования TPU для

проведения экспериментов на реальных задачах машинного обучения. Полученные результаты были анализированы и интерпретированы с использованием соответствующих визуализаций и сравнений с другими платформами.

3. Результат и обсуждение

3.1. Профилирование TPU

В данном примере модель обучается классификации изображений цветов в молниеносном облачном сервисе Google. модель принимает в качестве входных данных фотографию цветка и возвращает, является ли это маргариткой, одуванчиком, розой, подсолнухом или тюльпаном. Ключевая цель этойколлаборации - как настроить и запустить TensorBoard, программу, используемую для визуализации и анализа производительности программ в Cloud TPU.

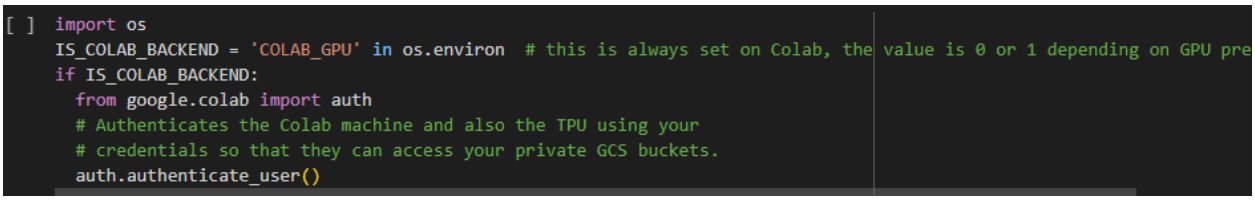

Рисунок 1. Аутентификация для подключения к корзине GCS для ведения журнала

Следующим шагом будет это включение и тестирование TPU:

Сначала нужно включить TPU для ноутбука:

Перейдите в меню Правка→ Настройки ноутбука

Следует выбрать TPU в раскрывающемся списке Аппаратный ускоритель

Далее проверим, можем ли подключиться к TPU(Рис. 2).

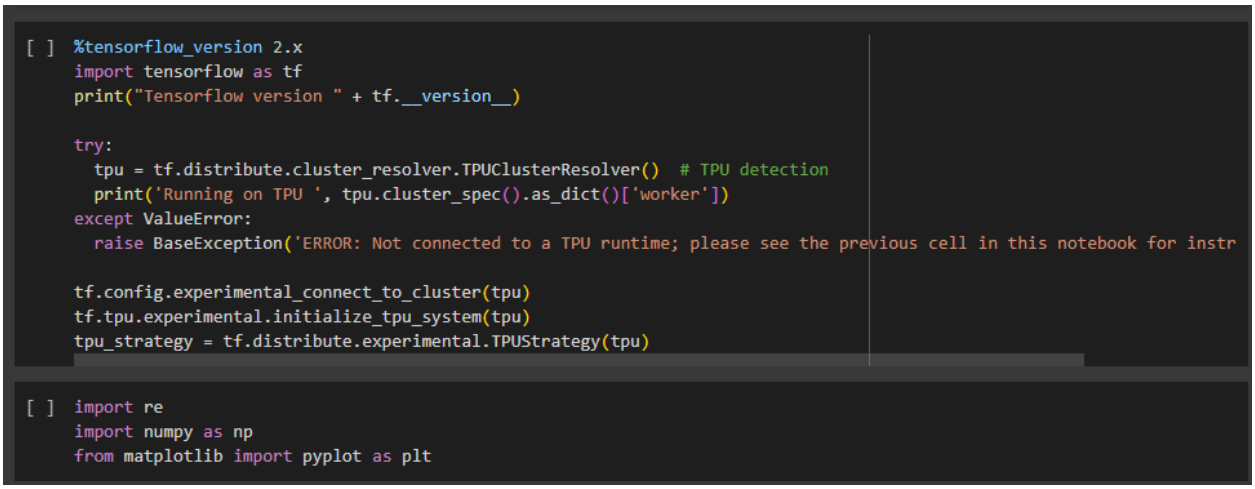

Рисунок 2. Включение и тестирование TPU

Входные данные хранятся в облачном хранилище Google. Чтобы более полно использовать возможности параллелизма, предлагаемые TPU, и

избежать узких мест при передаче данных, следует сохранить входные данные в файлах TFRecord, по 230 изображений на файл(Рис. 3).

```
IMAGE_SIZE = [331, 331]

batch_size = 16 * tpu_strategy.num_replicas_in_sync

gcs_pattern = 'gs://flowers-public/tfrecords-jpeg-331x331/*.tfrec'
validation_split = 0.19
filenames = tf.io.gfile.glob(gcs_pattern)
split = len(filenames) - int(len(filenames) * validation_split)
train_fns = filenames[:split]
validation_fns = filenames[split:]

def parse_tfrecord(example):
  features = {
    "image": tf.io.FixedLenFeature([], tf.string), # tf.string means bytestring
    "class": tf.io.FixedLenFeature([], tf.int64),  # shape [] means scalar
    "one_hot_class": tf.io.VarLenFeature(tf.float32),
  }
  example = tf.io.parse_single_example(example, features)
  decoded = tf.image.decode_jpeg(example['image'], channels=3)
  normalized = tf.cast(decoded, tf.float32) / 255.0 # convert each 0-255 value to floats in [0, 1] range
  image_tensor = tf.reshape(normalized, [*IMAGE_SIZE, 3])
  one_hot_class = tf.reshape(tf.sparse.to_dense(example['one_hot_class']), [5])
  return image_tensor, one_hot_class

def load_dataset(filenames):
  # Read from TFRecords. For optimal performance, we interleave reads from multiple files.
  records = tf.data.TFRecordDataset(filenames, num_parallel_reads=AUTO)
  return records.map(parse_tfrecord, num_parallel_calls=AUTO)

def get_training_dataset():
  dataset = load_dataset(train_fns)

  # Create some additional training images by randomly flipping and
  # increasing/decreasing the saturation of images in the training set.
  def data_augment(image, one_hot_class):
    modified = tf.image.random_flip_left_right(image)
    modified = tf.image.random_saturation(modified, 0, 2)
    return modified, one_hot_class
  augmented = dataset.map(data_augment, num_parallel_calls=AUTO)

  # Prefetch the next batch while training (autotune prefetch buffer size).
  return augmented.repeat().shuffle(2048).batch(batch_size).prefetch(AUTO)

training_dataset = get_training_dataset()
validation_dataset = load_dataset(validation_fns).batch(batch_size).prefetch(AUTO)
```

Рисунок 3. Входные данные

Чтобы получить максимальную точность, используем предварительно обученную модель распознавания изображений (здесь, Xception). Удаляем верхние слои, специфичные для ImageNet (include_top=false), и добавляем максимальное объединение и слой softmax, чтобы предсказать следующие 5 классов(Рис. 4).

```
[ ] def create_model():
      pretrained_model = tf.keras.applications.Xception(input_shape=[*IMAGE_SIZE, 3], include_top=False)
      pretrained_model.trainable = True
      model = tf.keras.Sequential([
        pretrained_model,
        tf.keras.layers.GlobalAveragePooling2D(),
        tf.keras.layers.Dense(5, activation='softmax')
      ])
      model.compile(
        optimizer='adam',
        loss = 'categorical_crossentropy',
        metrics=['accuracy']
      )
      return model

    with tpu_strategy.scope(): # creating the model in the TPUStrategy scope means we will train the model on the TPU
      model = create_model()
    model.summary()
```

Рисунок 4. Модель

Следующим щагом следует вычислить количество изображений в каждом наборе данных. Вместо того, чтобы загружать данные для этого используем подсказки в имени файла. Это используется для вычисления количества пакетов за эпоху (рис. 5).

```
[ ] def count_data_items(filenames):
      # the number of data items is written in the name of the .tfrec files, i.e. flowers00-230.tfrec = 230 data items
      n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for filename in filenames]
      return np.sum(n)

    n_train = count_data_items(train_fns)
    n_valid = count_data_items(validation_fns)
    train_steps = count_data_items(train_fns) // batch_size
    print("TRAINING IMAGES: ", n_train, ", STEPS PER EPOCH: ", train_steps)
    print("VALIDATION IMAGES: ", n_valid)
```

Рисунок 5. Обучение

Следует рассчитать и показать график скорости обучения. Запускается с довольно низкой скорости, поскольку используем предварительно обученную модель в целях отсутствия потерь оптимизации проделанной ранее на ее обучение (рис. 6).

```
EPOCHS = 12

start_lr = 0.00001
min_lr = 0.00001
max_lr = 0.00005 * tpu_strategy.num_replicas_in_sync
rampup_epochs = 5
sustain_epochs = 0
exp_decay = .8

def lrfn(epoch):
  if epoch < rampup_epochs:
    return (max_lr - start_lr)/rampup_epochs * epoch + start_lr
  elif epoch < rampup_epochs + sustain_epochs:
    return max_lr
  else:
    return (max_lr - min_lr) * exp_decay**(epoch-rampup_epochs-sustain_epochs) + min_lr

lr_callback = tf.keras.callbacks.LearningRateScheduler(lambda epoch: lrfn(epoch), verbose=True)

rang = np.arange(EPOCHS)
y = [lrfn(x) for x in rang]
plt.plot(rang, y)
print('Learning rate per epoch:')
```

Рисунок 6. Скорость обучения

После выполнения команды по графику видно, что скорость обучения возрастает с количеством поставленных эпох (рис. 7).

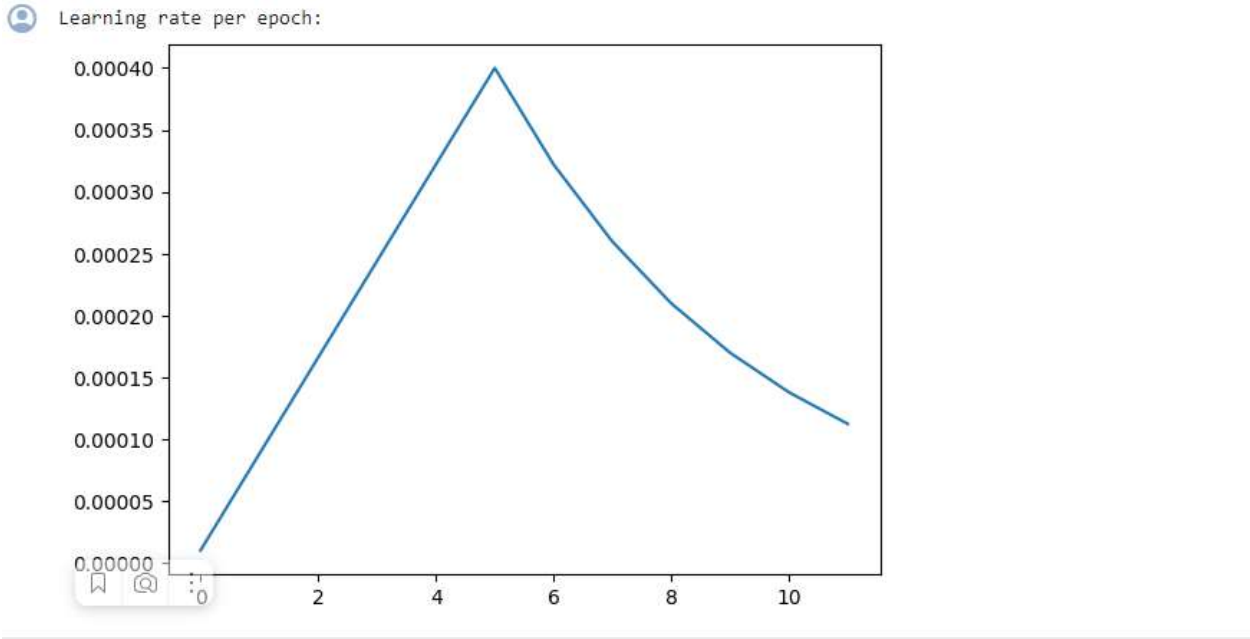

Рисунок 7. Скорость обучения за эпоху

Для получения информации о скорости обучения следует использовать следующие команды (рис. 8-9).

```
history = model.fit(training_dataset, validation_data=validation_dataset,
                    steps_per_epoch=train_steps, epochs=EPOCHS, callbacks=[lr_callback])

final_accuracy = history.history["val_accuracy"][-5:]
print("FINAL ACCURACY MEAN-5: ", np.mean(final_accuracy))
```

Рисунок 8. Команды для запуска тестирования скорости обучения

```
def display_training_curves(training, validation, title, subplot):
  ax = plt.subplot(subplot)
  ax.plot(training)
  ax.plot(validation)
  ax.set_title('model '+ title)
  ax.set_ylabel(title)
  ax.set_xlabel('epoch')
  ax.legend(['training', 'validation'])

plt.subplots(figsize=(10,10))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'], 'accuracy', 211)
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss', 212)
```

Рисунок 9. Вывод информации в виде графика по обучению

После проделанной работы будут предоставлены два графика, на которых показаны результаты оптимизации обучения модели в Google Colab (рис. 10).

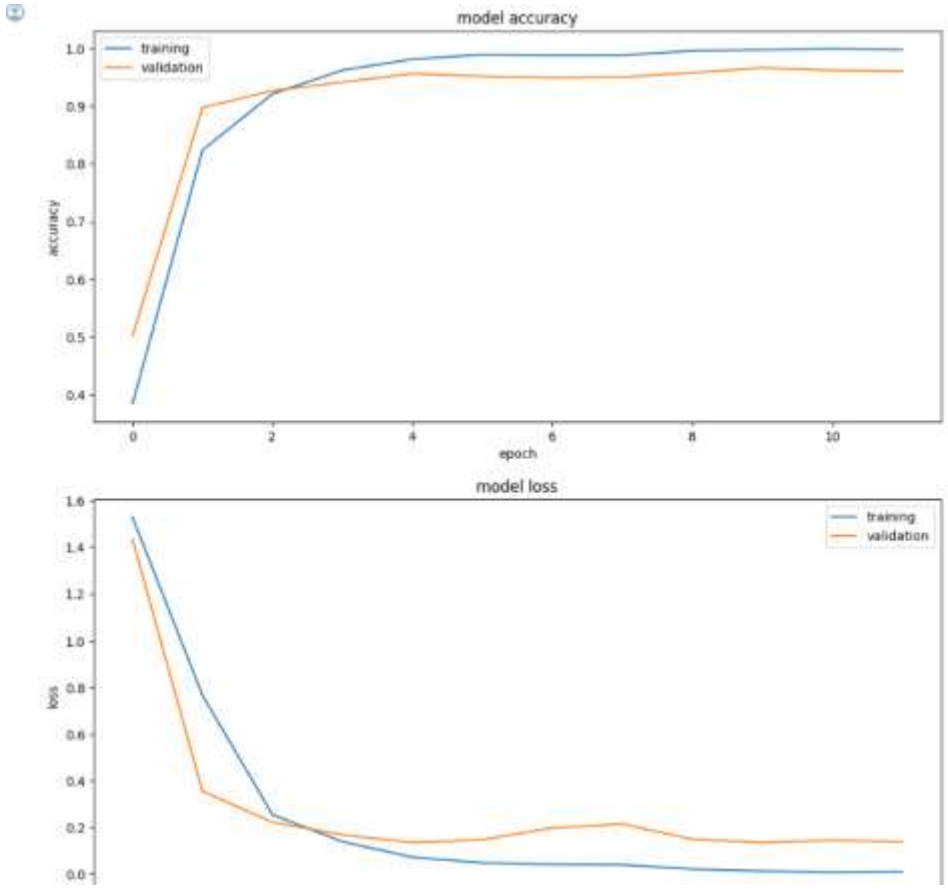

Рисунок 10. Результаты обучения

В результате видно, что скорость обучения возросла и вероятность потери данных приблизилась к нулю.

Вывод

В заключение, профилирование TPU в Colab является полезным инструментом для разработчиков и исследователей в области машинного обучения. Оно позволяет получить глубокое понимание производительности моделей, выявить узкие места и оптимизировать их. Рекомендуется использовать профилирование TPU в Colab для достижения максимальной производительности и эффективности в задачах машинного обучения.

Библиографический список

1. Maelfabien URL: https://maelfabien.github.io/bigdata/ColabTPU/#
2. Mlcentre URL: https://mlcentre.ru/articles/343867/
3. TensorFlowBlog URL: https://blog.tensorflow.org/2019/01/keras-on-tpus-in-colab.html
4. Googlecolab URL: https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/profiling_tpus_in_colab.ipynb#scrollTo=Zs2tsV8xebhU