TensorFlow

Курс “Практическое применение по TensorFlow”

Шигапова Фирюза Зинатуллаевна

1-й семестр, 2019 г.

https://github.com/Firyuza/TensorFlowPractice

Data Input Pipeline

Tensorflow

TF QUEUE

- FIFOQueue
- PriorityQueue
- PaddingFIFOQueue
- RandomShuffleQueue

tf.data

- map
- batch
- prefetch
- shuffle
- cache
- repeat
- ...

Enable tf.data in Tensorflow 1.x

```
tf.enable_eager_execution()
```

tf.data

1. Create a **Dataset** through loading objects (data) stored in memory or in files:

   ```
   NumPy:
   dataset = tf.data.Dataset.from_tensor_slices(...),
   dataset = tf.data.Dataset.from_tensors(...)
   ```

   ```
   TFRecord:
   filenames = [filename]
   raw_dataset = tf.data.TFRecordDataset(filenames)
   ```

   ```
   etc.
   ```

2. Set up **Transformation** functions for dataset:

   ```
   dataset.map(...),
   dataset.batch(...) etc.
   ```

tf.data. Load data

1. CSV
2. **NumPy**
3. Text
4. Images
5. pandas.DataFrame
6. TF.Text
7. Unicode
8. **TFRecord and tf.Example**

tf.data. NumPy

```
dataset = tf.data.Dataset.from_tensor_slices((IMAGE_PATHS, IMAGE_LABELS))
```

NumPy arrays

```
for image, label in dataset:
    print(label.numpy())
```

tf.data. Transformation

```
dataset = dataset.map(map_func=preprocess)
```

Transformation function

- **tf.py_function** can also be called within *preprocess* function

tf.data. Transformation

```
dataset = dataset.shuffle(buffer_size=len(IMAGE_PATHS))
dataset = dataset.batch(batch_size=BATCH_SIZE)
```

tf.data. *map_func*

```
dataset = tf.data.Dataset.from_tensor_slices((IMAGE_PATHS, IMAGE_LABELS))

dataset = dataset.map(map_func=read_and_augmment_image,
                      num_parallel_calls=tf.data.experimental.AUTOTUNE)

def read_and_augmment_image(filename, label):
    file_contents = tf.io.read_file(filename)

    img_tensor = tf.image.decode_jpeg(file_contents, channels=IMAGE_CHANNEL_SIZE)
    img_tensor.set_shape((None, None, IMAGE_CHANNEL_SIZE))

    img_final = tf.image.resize(img_tensor, [IMAGE_SIZE, IMAGE_SIZE])
    img_final = img_final / 255.0

    return img_final, label
```

tf.data. *map_func*

1. If **.map(...)** <u>before</u> **.batch(...)** *map_func*

   process **single** item

2. If **.map(...)** <u>after</u> **.batch(...)** *map_func*

   process **batch** of items

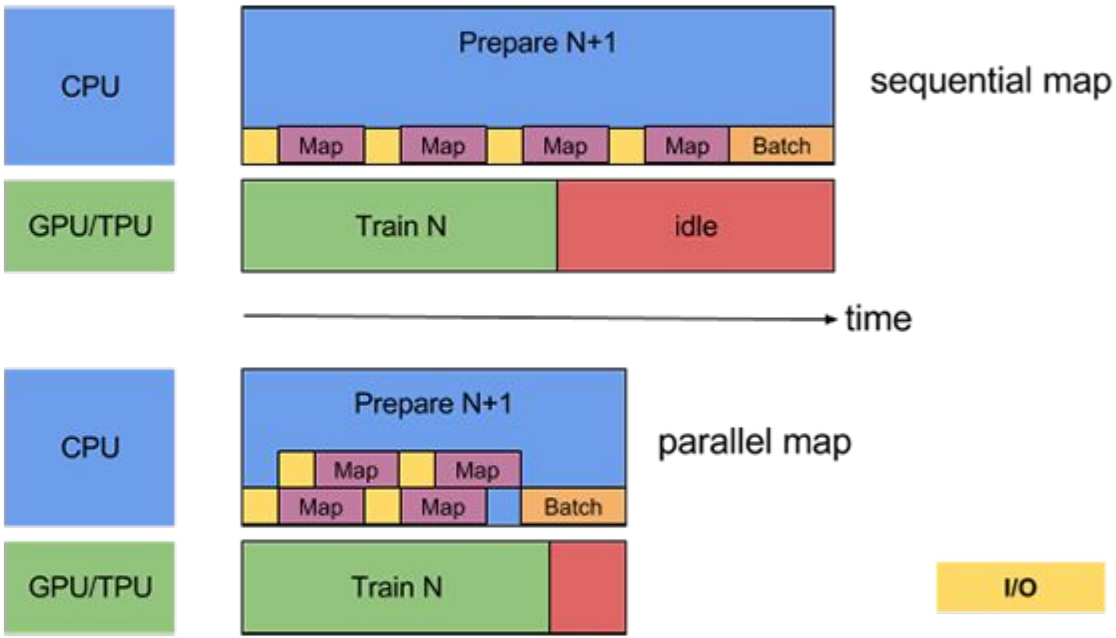
Performance

Without pipelining, the CPU and the GPU/TPU sit idle much of the time:

With pipelining, idle time diminishes significantly:

Source: https://www.tensorflow.org/guide/data_performance

Performance. map

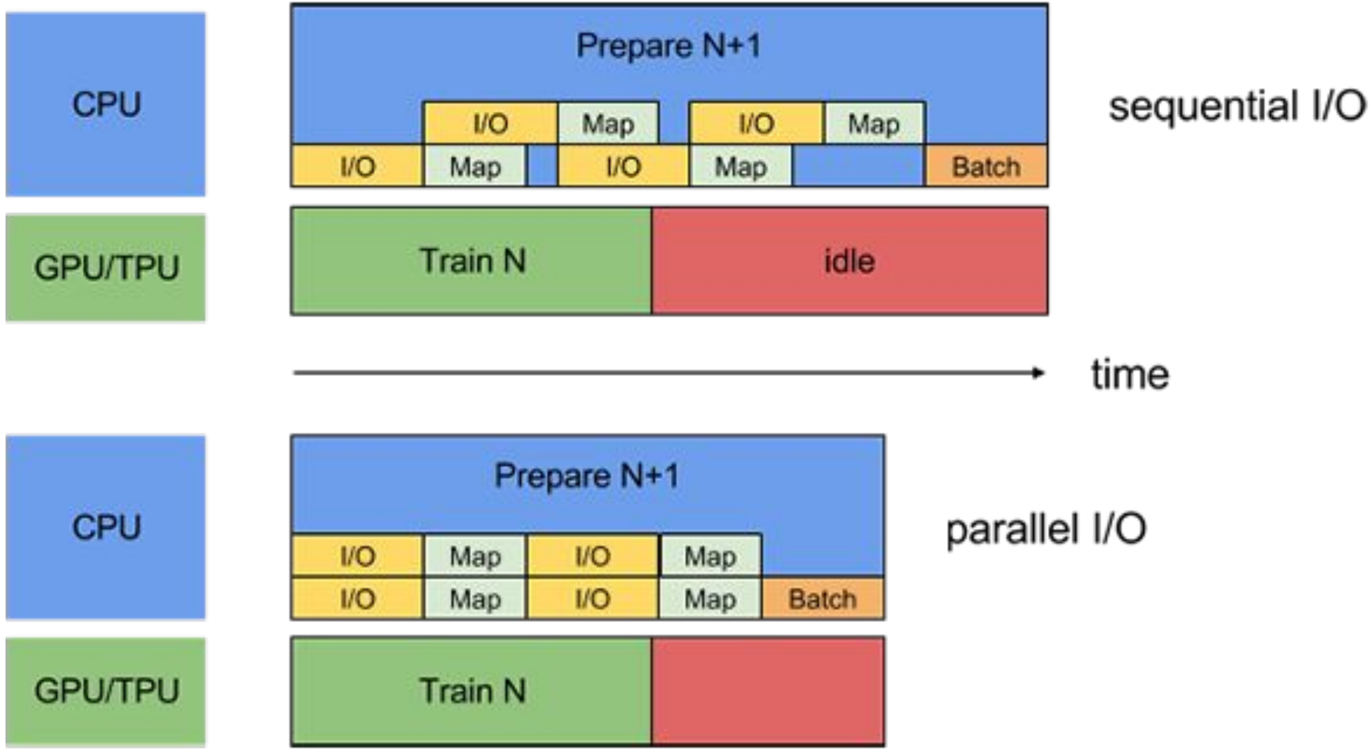


Source: https://www.tensorflow.org/guide/data_performance

Performance. map

```
dataset = dataset.map(map_func=read_and_augmment_image,
                      num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

Performance. interleave



Source: https://www.tensorflow.org/guide/data_performance

Performance. interleave

```
files = tf.data.Dataset.list_files("/path/to/dataset/train-*.tfrecord")
dataset = files.interleave(
    tf.data.TFRecordDataset, cycle_length=FLAGS.num_parallel_reads,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

Performance. prefetch

Source: https://www.tensorflow.org/guide/data_performance

Performance. prefetch

```
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

Best practices

1. **map(...) & batch(...):**
   - if *map_func* is **expensive**, call **map(...) first** and then batch(...);
   - if *map_func* does **little** work, call **batch(...) first** and then map(...) // should vectorize *map_func*
2. **prefetch(...)** at the **end** of data input pipeline
3. **cache(...)**: cache transformed data if *map_func* is expensive
4. **interleave(...)** to **parallelize** the reading from different files
5. Use **num_parallel_calls** (*tf.data.experimental.AUTOTUNE*)

tf.data.TFRecordDataset

Работает с TFRecord файлами (*.tfrecord)

```
__init__(
    filenames,
    compression_type=None,
    buffer_size=None,
    num_parallel_reads=None
)
```

```
filenames = [self.filename]
raw_dataset = tf.data.TFRecordDataset(filenames)
```

TFRecord file

- Serialize data
- Store a sequence of binary records
- Serialize data via **tf.train.Example**
- Save data using **tf.data.experimental.TFRecordWriter**

How to create TFRecord file

1. Wrap features of the data into **tf.train.Feature**

```
tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
```

2. Put features into dictionary:

```
feature = {
    'feature': tf.train.Feature(int64_list=tf.train.Int64List(value=[feature0]))
}
```

3. Create **tf.train Example** and **serialize** to string:

```
example_proto = tf.train.Example(features=tf.train.Features(feature=feature))

serialized_data = example_proto.SerializeToString()
```

How to create TFRecord file

4. Write it using **TFRecordWriter**

```
features_dataset = tf.data.Dataset.from_tensor_slices((feature0, feature1, feature2))

def generator():
    for features in features_dataset:
        yield self.serialize_example(*features)

serialized_features_dataset = tf.data.Dataset.from_generator(
    generator, output_types=tf.string, output_shapes=())

writer = tf.data.experimental.TFRecordWriter(self.filename)
writer.write(serialized_features_dataset)
```

Read TFRecord

```
raw_dataset = tf.data.TFRecordDataset(filenames)
parsed_dataset = raw_dataset.map(tf.io.parse_single_example(raw_dataset, self.feature_description))

for raw_record in parsed_dataset.take(1):
    print(raw_record)
    # {'feature0': < tf.Tensor: id = 95, shape = (), dtype = int64, numpy = 1 >,
    # 'feature1': < tf.Tensor: id = 96, shape = (), dtype = float32, numpy = 1.0 >,
    #  'feature2': < tf.Tensor: id = 97, shape = (), dtype = string, numpy = b'name1' >}
```