Parte 2

Tensorflow 2

Profesores: Javier López

Daniel Cano

Datasets

Antes

- Implementación ad-hoc para cada problema
- Dificil de optimizar
- No hace un uso óptimo de los recursos

Ahora

- Forma estandarizada de definir
- Capaz de utilizar todos los recursos

Datasets

Como se crea un dataset desde pandas

```
import pandas as pd
import tensorflow as tf

csv_file = tf.keras.utils.get_file('file.csv', 'https://url.com/file.csv')
df = pd.read_csv(csv_file)
target = df.pop('target')


dataset = tf.data.Dataset.from_tensor_slices((df.values, target.values))
```

Datasets

Como se crea un dataset para imágenes

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

TRAIN_PATH = './dataset/train'
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16
CLASSES = ['car', 'cycle', 'plane', 'bus', 'truck']

train_data = ImageDataGenerator(
    horizontal_flip=True,
    rotation_range=40,
    fill_mode='nearest'
).flow_from_directory(TRAIN_PATH, target_size=IMAGE_SIZE, classes=CLASSES,
                      batch_size=BATCH_SIZE,shuffle=True)
```

Datasets

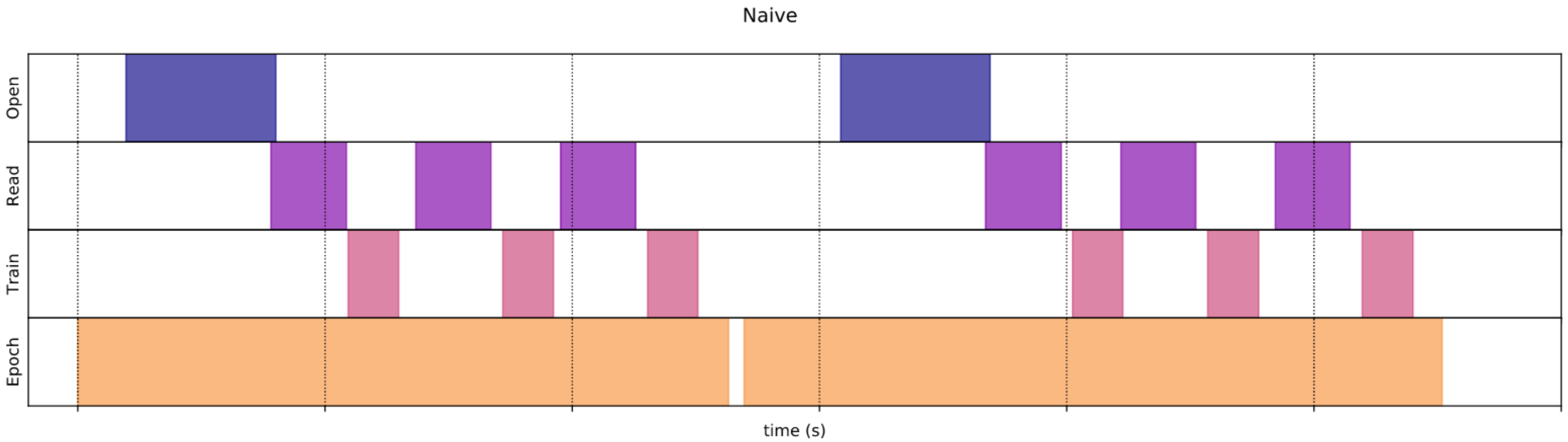
Como se crea un dataset para imágenes 2

```
BATCH_SIZE = 64

X = ["dataset/X/" + name for name in os.listdir("dataset/X")]
Y = ["dataset/Y/" + name for name in os.listdir("dataset/Y")]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,train_size=0.8,random_state=42)

train_dataset = tf.data.Dataset.from_tensor_slices(
  (tf.constant(X_train), tf.constant(Y_train))
)
test_dataset = tf.data.Dataset.from_tensor_slices(
  (tf.constant(X_test), tf.constant(Y_test))
)

def _load_image(filename_a, filename_b):
    img_a = tf.io.read_file(filename_a)
    img_a = tf.image.decode_jpeg(img_a)
    img_b = tf.io.read_file(filename_b)
    img_b = tf.image.decode_jpeg(img_b)
    img_a = tf.cast(img_a, tf.float32)/255
    img_b = tf.cast(img_b, tf.float32)/255
    #img = tf.image.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
    return img_a, img_b

train_data = (train_dataset.map(_load_image).shuffle(buffer_size=3500).batch(BATCH_SIZE))
test_data = (test_dataset.map(_load_image).shuffle(buffer_size=3500).batch(BATCH_SIZE))
```

Datasets

Optimización de carga

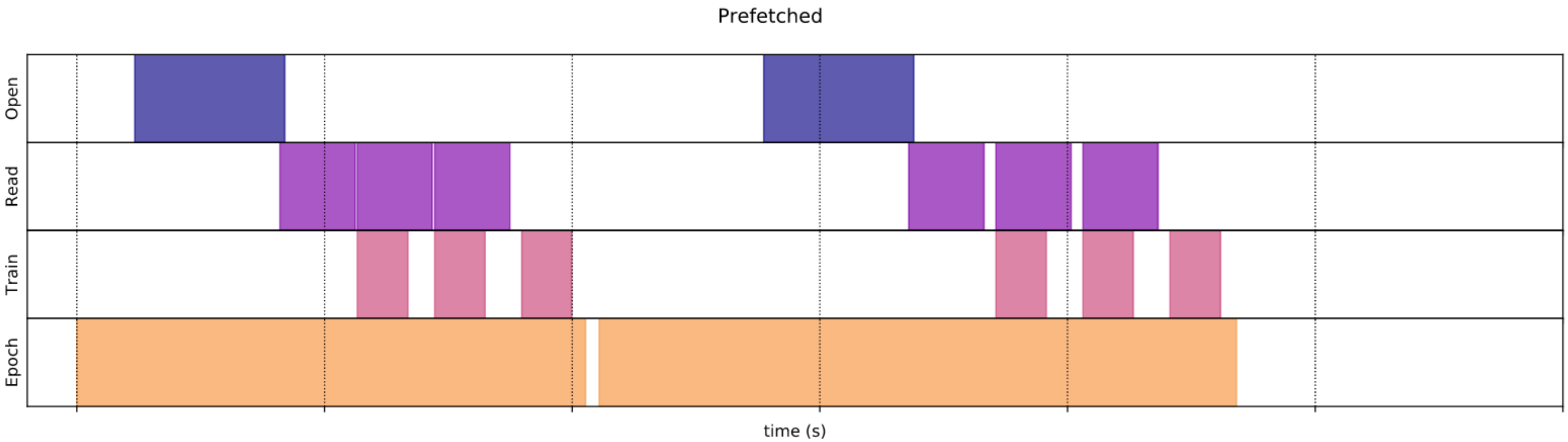
- Necesitamos reducir los tiempos
- Usar de la mejor forma todos los recursos



Datasets

Prefetch

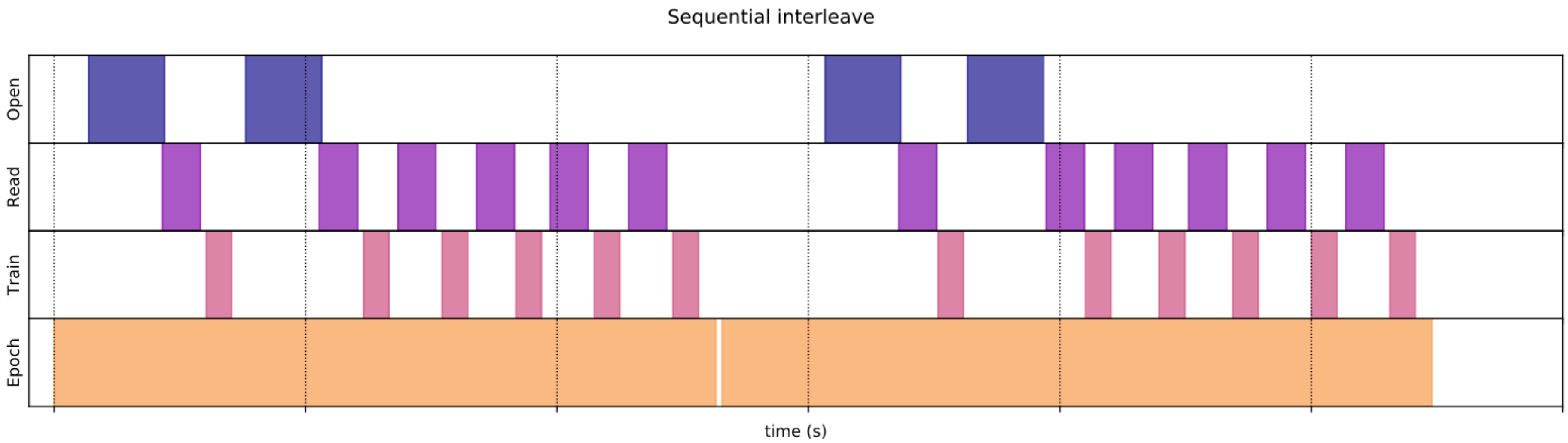
```
Dataset().prefetch(tf.data.experimental.AUTOTUNE)
```



Datasets

Interleave

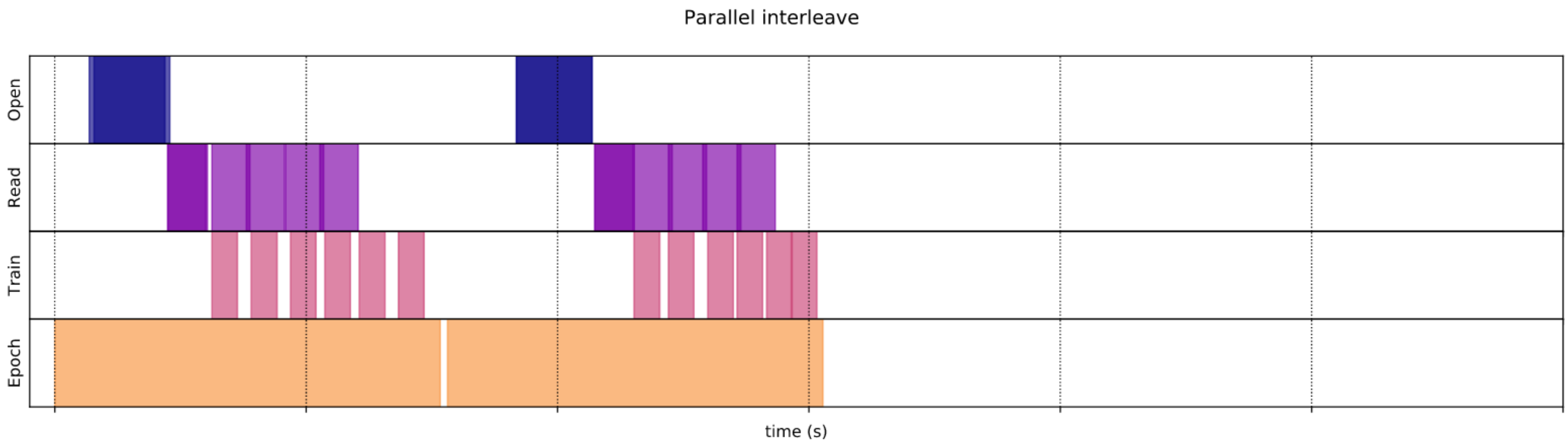
```
1 tf.data.Dataset.range(2).interleave(Dataset)
```



Datasets

Interleave + Parallel call

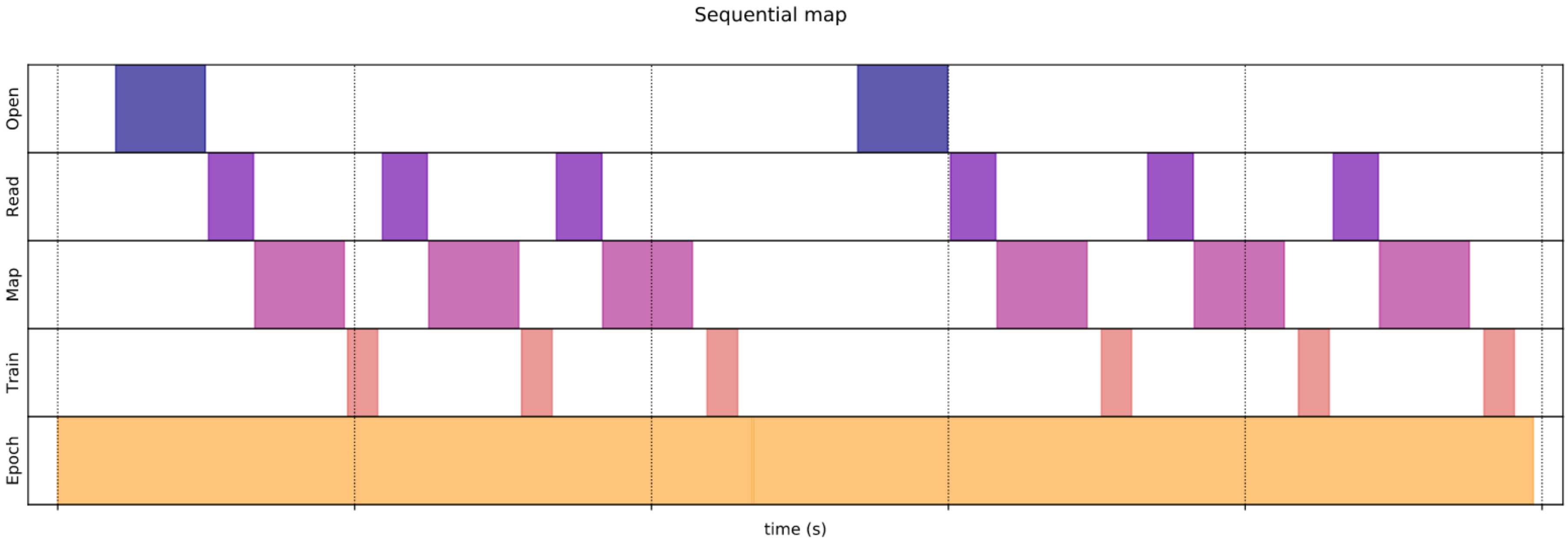
```
tf.data.Dataset.range(2)
    .interleave(
        Dataset,
        num_parallel_calls=tf.data.experimental.AUTOTUNE
    )
```



Datasets

Map Operation Naive

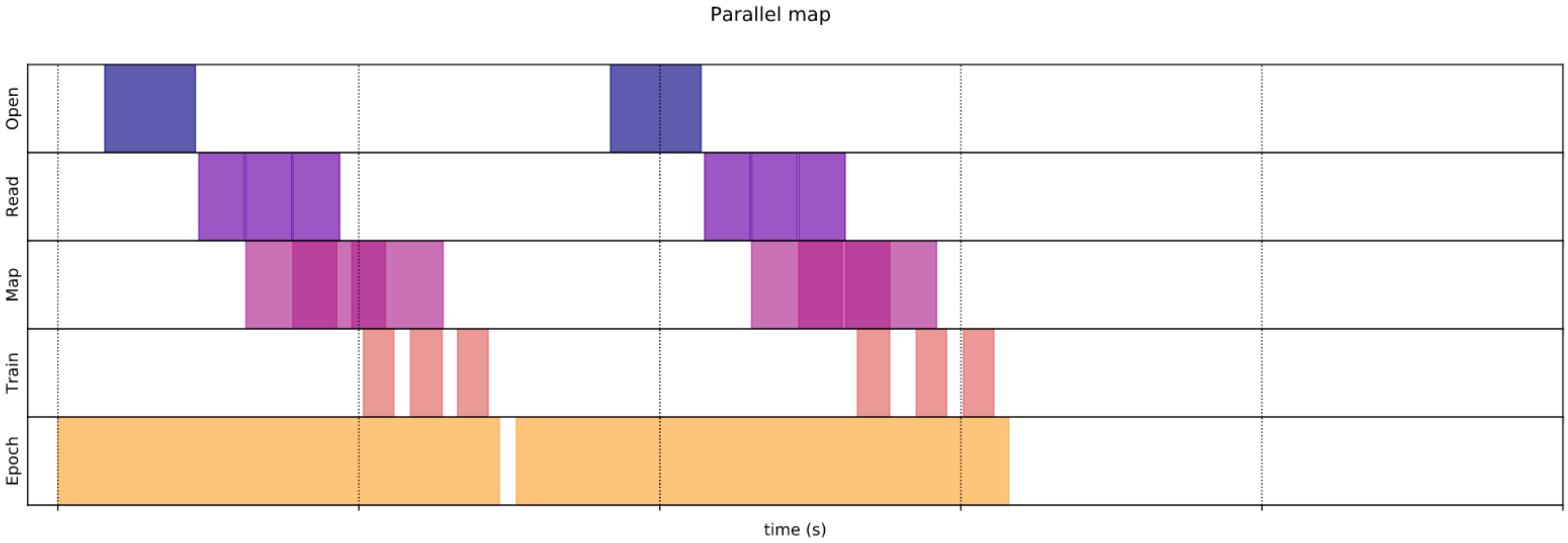
```
Dataset().map(mapped_function)
```



Datasets

Map Operation + Parallel

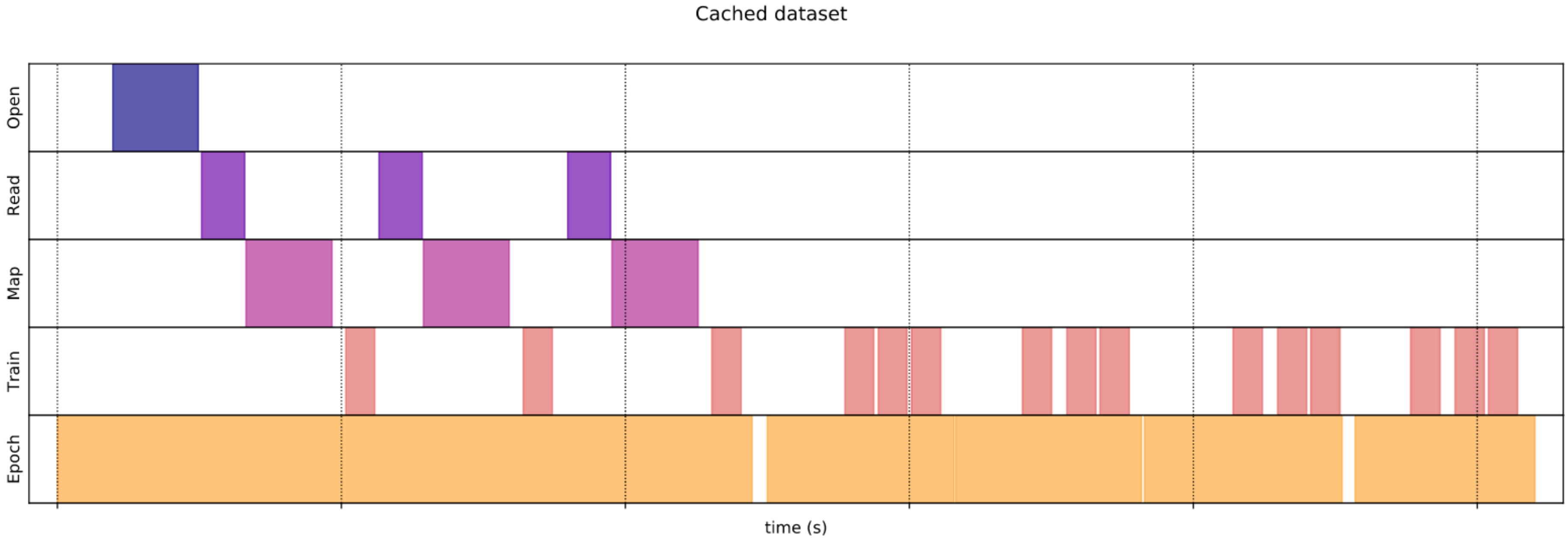
```
Dataset()
    .map(
        mapped_function,
        num_parallel_calls=tf.data.experimental.AUTOTUNE
    )
```



Datasets

Map Operation Naive + Cache

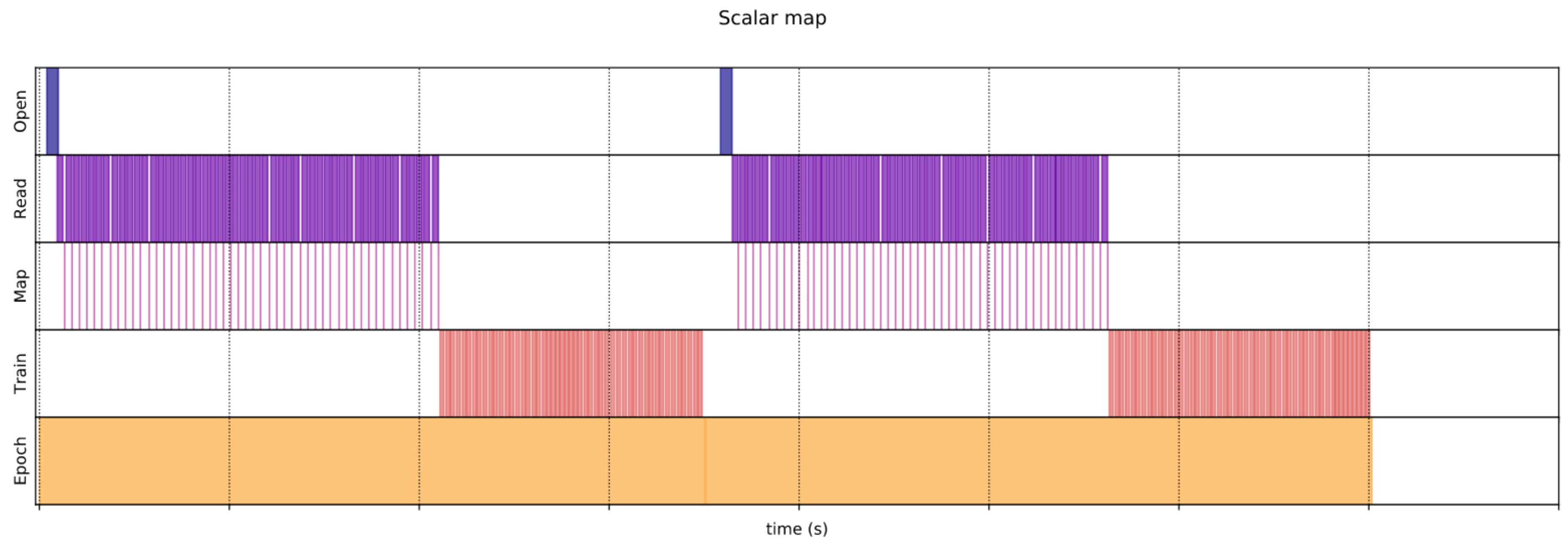
```
Dataset()
    .map(
        mapped_function
    ).cache()
```



Datasets

Scalar Mapping

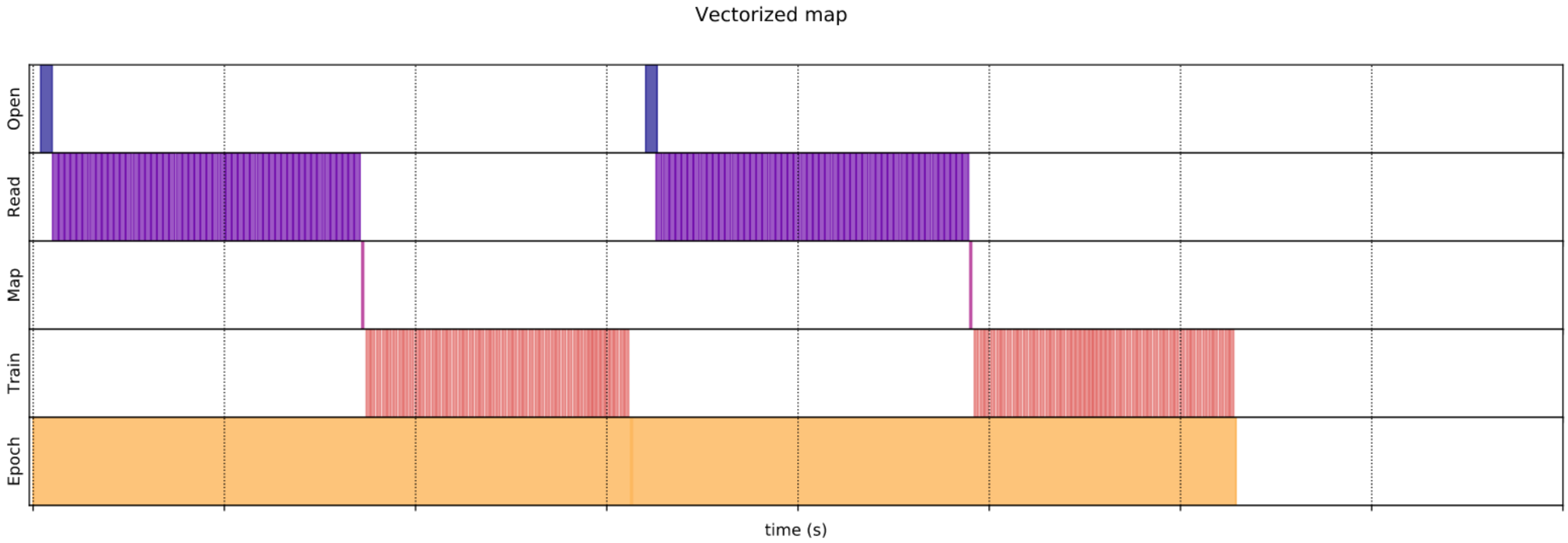
```
dataset.map(function_to_apply).batch(256)
```



Datasets

Scalar Mapping (pre. batch)

```
dataset.batch(256).map(function_to_apply)

```



Datasets

Resumen

1. Usa prefetch para solapar el trabajo de carga
2. Paraleliza la lectura de datos con Interleave
3. Paraleliza los 'map' con 'num_parallel_calls'
4. Utiliza la cache en memoria cuando puedas
5. Agrupa las operaciones
6. Reduce el uso de memoria aplicando Interleave, Prefetch y Shuffle.

Datasets

Naive
Open
Read
Map
Train
Epoch
Optimized
Open
Read
1st map
2nd map
Train
Epoch
time (s)

Callbacks

Necesidades

1. No tenemos control del proceso de train
2. No podemos realizar acciones durante el train

Usuales

1. ModelCheckpoint
2. Tensorboard
3. EarlyStopping
4. ReduceLROnPlateau
5. RemoteMonitor
6. Base Callback Class

Callbacks

ModelCheckpoint

```
tf.keras.callbacks.ModelCheckpoint(
    filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=False,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
    options=None,
    **kwargs
)
```

Callbacks

EarlyStopping

```
tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=0,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

Callbacks

ReduceLROnPlateau

```
tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=10,
    verbose=0,
    mode="auto",
    min_delta=0.0001,
    cooldown=0,
    min_lr=0,
    **kwargs
)
```

Tensorboard

Necesidades

1. Representar gráficamente información
2. Ver la evolución del modelo
3. Gestión de la información producida

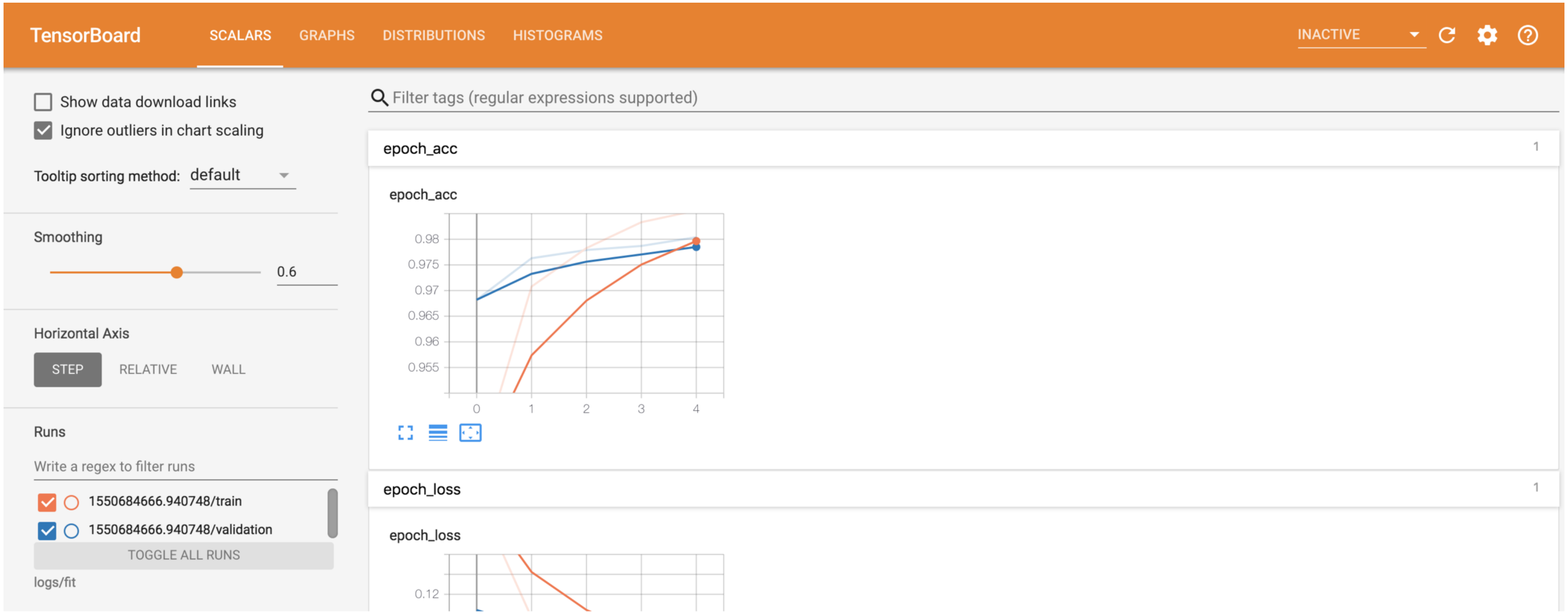
Cómo se utiliza

1. Declarar el callback
2. Lanzar la herramienta al finalizar el train

Tensorboard

Uso

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(
  log_dir=log_dir, histogram_freq=1)

model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback]) # <<<---------

tensorboard --logdir logs/fit
```



Entrenamiento Distribuido

Idea

- Utilizar todas las unidades de procesamiento disponibles
- Distribuir de la mejor forma los cálculos a realizar

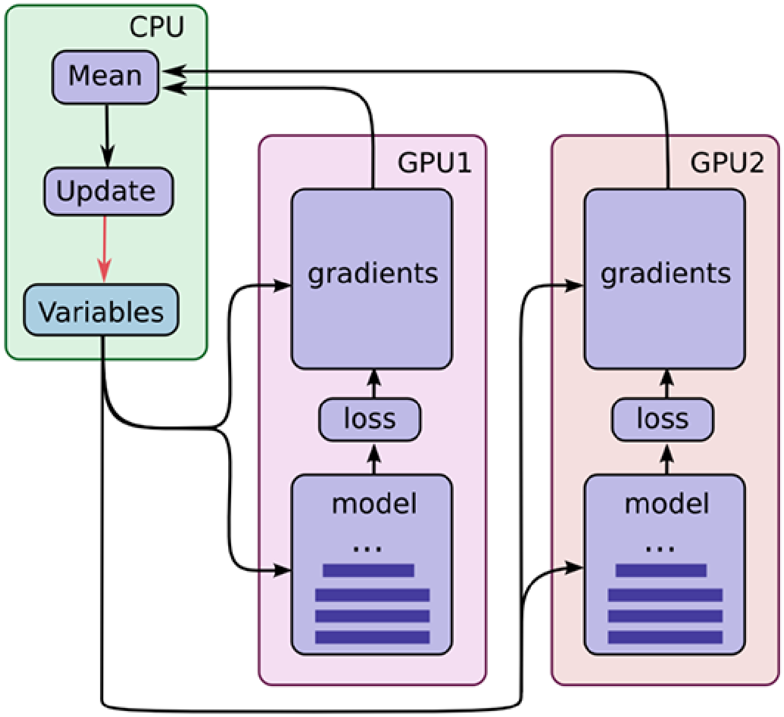
Opciones disponibles

- MirrorStrategy
- TPUStrategy
- MultiWorkerMirroredStrategy
- CentralStorageStrategy
- ParameterServerStrategy

Entrenamiento Distribuido

MirrorStrategy

```
mirrored_strategy = tf.distribute.MirroredStrategy(
    cross_device_ops=tf.distribute.HierarchicalCopyAllReduce())
```



Entrenamiento Distribuido

TPUStrategy

```
cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
    tpu=tpu_address)
tf.config.experimental_connect_to_cluster(cluster_resolver)
tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
tpu_strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
```