

C2_W4_Lab_1_basic-mirrored-strategy

February 5, 2025

1 Mirrored Strategy: Basic

In this ungraded lab, you'll go through some of the basics of applying [Mirrored Strategy](#).

1.1 Imports

```
[1]: # Import TensorFlow and TensorFlow Datasets

import tensorflow_datasets as tfds
import tensorflow as tf
tfds.disable_progress_bar()

import os
```

Load the MNIST dataset and split it into training and test chunks.

```
[2]: # Load the dataset we'll use for this lab
datasets, info = tfds.load(name='mnist', with_info=True, as_supervised=True,
    ↪data_dir='./data')

mnist_train, mnist_test = datasets['train'], datasets['test']
```

Next, you define `strategy` using the `MirroredStrategy()` class. Print to see the number of devices available.

Note: - If you are running this on Coursera, you'll see it gives a warning about no presence of GPU devices. - If you are running this in Colab, make sure you have selected your Runtime to be GPU for it to detect it. - In both these cases, you'll see there's only 1 device that is available. - One device is sufficient for helping you understand these distribution strategies.

```
[3]: # Define the strategy to use and print the number of devices found
strategy = tf.distribute.MirroredStrategy()
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))
```

WARNING:tensorflow:There are non-GPU devices in `tf.distribute.Strategy`, not using nccl allreduce.

WARNING:tensorflow:There are non-GPU devices in `tf.distribute.Strategy`, not using nccl allreduce.

INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:CPU:0',)

INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:CPU:0',)

Number of devices: 1

Next, you create your training and test examples, define your batch size and also define `BATCH_SIZE_PER_REPLICA` which is the distribution you are making for each available device.

```
[4]: # Get the number of examples in the train and test sets
num_train_examples = info.splits['train'].num_examples
num_test_examples = info.splits['test'].num_examples

BUFFER_SIZE = 10000

BATCH_SIZE_PER_REPLICA = 64
# Use for Mirrored Strategy
BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.num_replicas_in_sync
# Use for No Strategy
# BATCH_SIZE = BATCH_SIZE_PER_REPLICA * 1
```

A mapping function which normalizes your images:

```
[5]: # Function for normalizing the image
def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255

    return image, label
```

Next, you create your training and evaluation datasets in the batch size you want by shuffling through your buffer size.

```
[6]: # Set up the train and eval data set
train_dataset = mnist_train.map(scale).cache().shuffle(BUFFER_SIZE).
    ↪ batch(BATCH_SIZE)
eval_dataset = mnist_test.map(scale).batch(BATCH_SIZE)
```

For your model to follow the strategy, define your model within the strategy's scope. - Run all the cells below and notice the results. - Afterwards comment out `with strategy.scope():` and run everything again, without the strategy. Then you can compare the results. The important thing to notice and compare is the time taken for each epoch to complete. As mentioned in the lecture, doing a mirrored strategy on a single device (which our lab environment has) might take longer to train because of the overhead in implementing the strategy. With that, the advantages of using this strategy is more evident if you will use it on multiple devices.

```
[7]: # Use for Mirrored Strategy -- comment out `with strategy.scope():` and
      ↪deindent for no strategy
      with strategy.scope():
          model = tf.keras.Sequential([
              tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
              tf.keras.layers.MaxPooling2D(),
              tf.keras.layers.Flatten(),
              tf.keras.layers.Dense(64, activation='relu'),
              tf.keras.layers.Dense(10)
          ])
```

```
[8]: model.compile(loss=tf.keras.losses.
      ↪SparseCategoricalCrossentropy(from_logits=True),
      optimizer=tf.keras.optimizers.Adam(),
      metrics=['accuracy'])
```

```
[9]: model.fit(train_dataset, epochs=12)
```

Epoch 1/12

WARNING:tensorflow:From /opt/conda/lib/python3.7/site-packages/tensorflow/python/data/ops/multi_device_iterator_ops.py:601: get_next_as_optional (from tensorflow.python.data.ops.iterator_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.data.Iterator.get_next_as_optional()` instead.

WARNING:tensorflow:From /opt/conda/lib/python3.7/site-packages/tensorflow/python/data/ops/multi_device_iterator_ops.py:601: get_next_as_optional (from tensorflow.python.data.ops.iterator_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.data.Iterator.get_next_as_optional()` instead.

938/938 [=====] - 32s 35ms/step - loss: 0.2017 - accuracy: 0.9414

Epoch 2/12

938/938 [=====] - 24s 25ms/step - loss: 0.0664 - accuracy: 0.9808

Epoch 3/12

938/938 [=====] - 24s 26ms/step - loss: 0.0473 - accuracy: 0.9856

Epoch 4/12

938/938 [=====] - 24s 25ms/step - loss: 0.0338 - accuracy: 0.9896

Epoch 5/12

938/938 [=====] - 26s 28ms/step - loss: 0.0252 - accuracy: 0.9927

Epoch 6/12

```
938/938 [=====] - 24s 25ms/step - loss: 0.0194 -  
accuracy: 0.9941  
Epoch 7/12  
938/938 [=====] - 24s 26ms/step - loss: 0.0150 -  
accuracy: 0.9953  
Epoch 8/12  
938/938 [=====] - 24s 25ms/step - loss: 0.0108 -  
accuracy: 0.9966  
Epoch 9/12  
938/938 [=====] - 23s 25ms/step - loss: 0.0085 -  
accuracy: 0.9973  
Epoch 10/12  
938/938 [=====] - 24s 25ms/step - loss: 0.0072 -  
accuracy: 0.9976  
Epoch 11/12  
938/938 [=====] - 24s 25ms/step - loss: 0.0069 -  
accuracy: 0.9976  
Epoch 12/12  
938/938 [=====] - 24s 25ms/step - loss: 0.0051 -  
accuracy: 0.9984
```

```
[9]: <tensorflow.python.keras.callbacks.History at 0x7a9d782b2a90>
```

```
[ ]:
```