Lab 3a. Bring your own Script (TensorFlow)

Prerequisites

The objective of this lab is to give you step by step instructions on how to bring your custom script to Amazon Sagemaker in your AWS account

- The following steps are an explanation on the cells you will be executing by pressing **Shift+Enter** in an Amazon SageMaker Notebook instance.
- Follow the instructions to launch Amazon SageMaker Studio
- Please ensure that you have git cloned the repository in your SageMaker Studio.

Overview

Amazon SageMaker provides both (1) built-in algorithms and (2) an easy path to train your own custom models. Although the built-in algorithms cover many domains (computer vision, natural language processing etc.) and are easy to use (just provide your data), sometimes training a custom model is the preferred approach. This notebook will focus on training a custom model using TensorFlow 2.

TensorFlow script mode training and serving

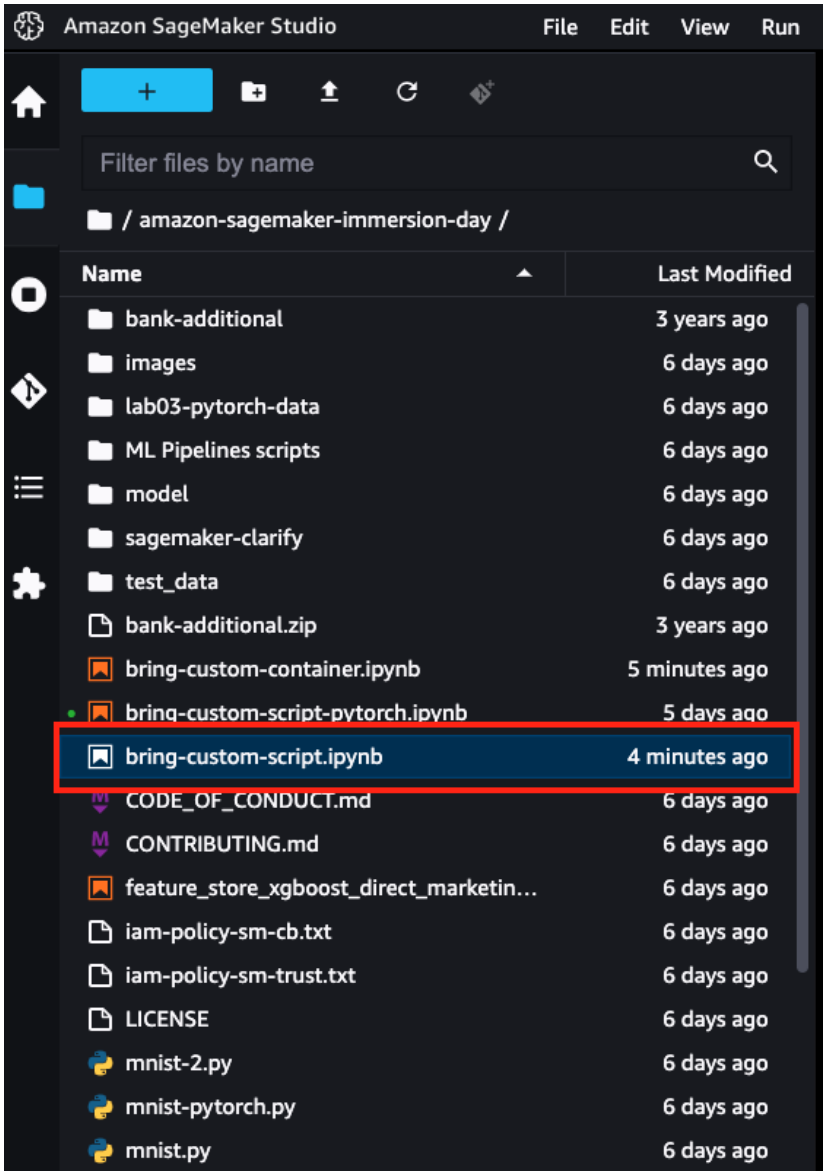
Script mode is a training script format for TensorFlow that lets you execute any TensorFlow training script in SageMaker with minimal modification. The SageMaker Python SDK handles transferring your script to a SageMaker training instance. On the training instance, SageMaker's native TensorFlow support sets up training-related environment variables and executes your training script. In this tutorial, we use the SageMaker Python SDK to launch a training job and deploy the trained model.

Script mode supports training with a Python script, a Python module, or a shell script. In this example, we use a Python script to train a classification model on the MNIST dataset. In this example, we will show how easily you can train a SageMaker using TensorFlow 1.x and TensorFlow 2.0 scripts with SageMaker Python SDK. In addition, this notebook demonstrates how to perform real time inference with the SageMaker TensorFlow Serving container. The TensorFlow Serving container is the default inference method for script mode. For full documentation on the TensorFlow Serving container, please visit here.

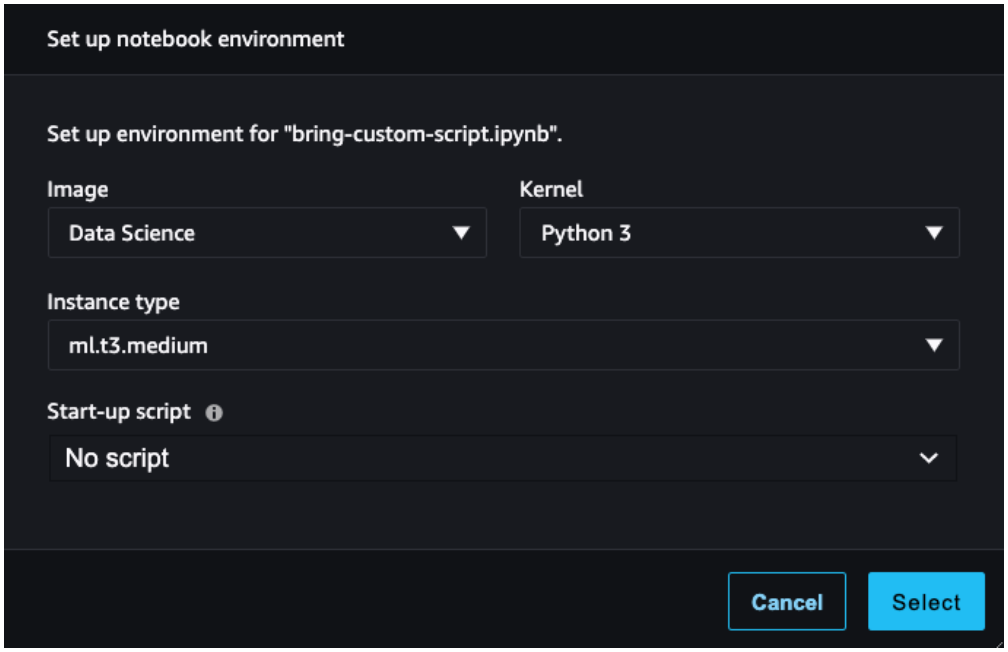
> The notebook contains detailed explanation of each step. This guide helps by providing steps to execute and expected outcomes of each step.

Launch the notebook instance

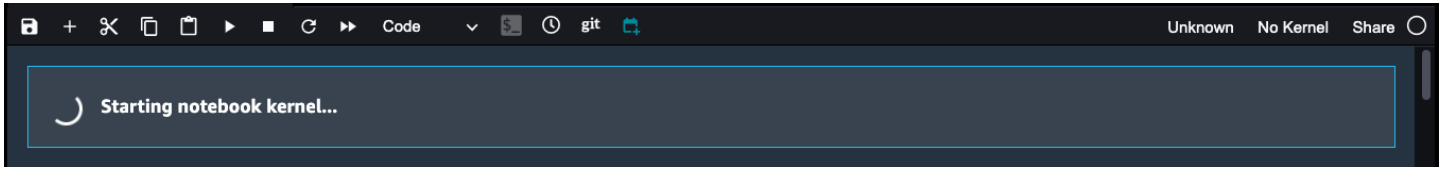
In your SageMaker Studio, in the "File Browser" pane on the left hand side, click on the file "amazon-sagemaker-immersion-day/bring-custom-script.ipynb"



You will be prompted to choose a kernel. Choose **Python 3** Kernel **Data Science** Image.



The notebook kernel will take a while to start:



After that, will be able to verify that the right kernel is selected on the top right of the screen:

Set up the environment

Execute the contents of cell 01 (click on the cell and then key Shift+Enter to execute). Please note, there will be no output and it will execute immediately

```
[2]: # cell 01
     import os
     import sagemaker
     from sagemaker import get_execution_role

     sagemaker_session = sagemaker.Session()

     role = get_execution_role()
     region = sagemaker_session.boto_session.region_name
```

Training Data

Execute the contents of cell 02. Please note, there will be no output and it will execute immediately

```
[3]: # cell 02
     training_data_uri = 's3://sagemaker-sample-data-{}/tensorflow/mnist'.format(region)
```

Construct a script for distributed training

Execute the contents of cell 03. It will take about a second to execute and then it will output the contents of the files `mnist.py` and `mnist-2.py`. These files are in the root of the folder "amazon-sagemaker-immersion-day/"

```
[4]: # cell 03

     !pygmentize 'mnist.py'

     # TensorFlow 2.1 script
     !pygmentize 'mnist-2.py'
     # Copyright 2018-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
     #
     # Licensed under the Apache License, Version 2.0 (the "License"). You
     # may not use this file except in compliance with the License. A copy of
     # the License is located at
     #
     #     http://aws.amazon.com/apache2.0/
     #
     # or in the "license" file accompanying this file. This file is
     # distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
     # ANY KIND, either express or implied. See the License for the specific
     # language governing permissions and limitations under the License.
     """Convolutional Neural Network Estimator for MNIST, built with tf.layers."""

     from __future__ import absolute_import
     from __future__ import division
     from __future__ import print_function
```

Create a training job using the TensorFlow estimator

Execute the contents of cell 04. It will take about a second to execute. The `sagemaker.tensorflow.TensorFlow` estimator handles locating the script mode container, uploading your script to a S3 location and creating a SageMaker training job. Let's call out a couple important parameters here:

- `py_version` is set to `'py3'` to indicate that we are using script mode since legacy mode supports only Python 2. Though Python 2 will be deprecated soon, you can use script mode with Python 2 by setting `py_version` to `py2` and `script_mode` to True.
- `distribution` is used to configure the distributed training setup. It's required only if you are doing distributed training either across a cluster of instances or across multiple GPUs. Here we are using parameter servers as the distributed training schema. SageMaker training jobs run on homogeneous clusters. To make parameter server more performant in the SageMaker setup, we run a parameter server on every instance in the cluster, so there is no need to specify the number of parameter servers to launch. Script mode also supports distributed training with Horovod. You can find the full documentation on how to configure distributions here.

```
[5]: # cell 04
     from sagemaker.tensorflow import TensorFlow


     mnist_estimator = TensorFlow(entry_point='mnist.py',
                                  role=role,
                                  instance_count=2,
                                  instance_type='ml.c5.xlarge',
                                  framework_version='1.15.2',
                                  py_version='py3',
                                  distribution={'parameter_server': {'enabled': True}})
```

You can also initiate an estimator to train with TensorFlow 2.1 script. The only things that you will need to change are the script `name` and `framework_version`

```
[6]: # cell 05
     mnist_estimator2 = TensorFlow(entry_point='mnist-2.py',
                                  role=role,
                                  instance_count=2,
                                  instance_type='ml.c5.xlarge',
                                  framework_version='2.1.0',
                                  py_version='py3',
                                  distribution={'parameter_server': {'enabled': True}})
```

Calling fit

To start a training job, we call `estimator.fit(training_data_uri)`. Execute the contents of cell 06. It will take several minutes to execute.

```
[*]: # cell 06
     mnist_estimator.fit(training_data_uri)

     INFO:sagemaker.image_uris:image_uri is not presented, retrieving image_uri based on instance_type, framework etc.
     INFO:sagemaker:Creating training-job with name: tensorflow-training-2023-03-25-01-44-48-570
     2023-03-25 01:44:52 Starting - Starting the training job...
```

It will spit out multiple lines of log with some depracation warnings. At the end, it will show an output similar to the following:

```
  https://sagemaker.readthedocs.io/en/stable/using_tf.html#adapting-your-local-tensorflow-script
  2023-03-25 02:01:42,020 sagemaker-containers INFO     Reporting training SUCCESS

  2023-03-25 02:01:58 Uploading - Uploading generated training model
  2023-03-25 02:01:58 Completed - Training job completed
  Training seconds: 1934
  Billable seconds: 1934
```

Execute the contents of cell 07 to train a model with TensorFlow 2.1 script. It will take several minutes to execute.

```
Calling fit to train a model with TensorFlow 2.1 script.

[*]: # cell 07
     mnist_estimator2.fit(training_data_uri)

     2023-03-25 02:52:22 Starting - Starting the training job...
```

At the end, it will show an output similar to the following:

```
INFO:tensorflow:Assets written to: /opt/ml/model/000000001/assets
INFO:tensorflow:Assets written to: /opt/ml/model/000000001/assets
2023-03-25 02:54:12,383 sagemaker-containers INFO     Reporting training SUCCESS

2023-03-25 02:54:47 Completed - Training job completed
Training seconds: 124
Billable seconds: 124
```

Deploy the trained model to an endpoint

Execute the contents of cell 08 to deploy the trained model with TensorFlow 1.15. It will take several minutes to execute.

```
[12]: # cell 08
      predictor = mnist_estimator.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')

      update_endpoint is a no-op in sagemaker>=2.
      See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
      ------!
```

Execute the contents of cell 09 to deploy the trained model with TensorFlow 2.1. It will take several minutes to execute.

```
[13]: # cell 09
      predictor2 = mnist_estimator2.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')

      update_endpoint is a no-op in sagemaker>=2.
      See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
      ------!
```

Invoke the endpoint

Execute the contents of cell 10 to download training data in SageMaker Studio. It will take a few seconds to execute.

```
[14]: # cell 10
      import numpy as np

      !aws --region {region} s3 cp s3://sagemaker-sample-data-{region}/tensorflow/mnist/train_data.npy train_data.npy
      !aws --region {region} s3 cp s3://sagemaker-sample-data-{region}/tensorflow/mnist/train_labels.npy train_labels.npy

      train_data = np.load('train_data.npy')
      train_labels = np.load('train_labels.npy')
```

Now, run the predictions by executing cell 11 which takes about a second to execute

```
[15]: # cell 11
      predictions = predictor.predict(train_data[:50])
      for i in range(0, 50):
          prediction = predictions['predictions'][i]['classes']
          label = train_labels[i]
          print('prediction is {}, label is {}, matched: {}'.format(prediction, label, prediction == label))

      prediction is 7, label is 7, matched: True
      prediction is 3, label is 3, matched: True
      prediction is 4, label is 4, matched: True
      prediction is 6, label is 6, matched: True
      prediction is 1, label is 1, matched: True
      prediction is 8, label is 8, matched: True
      prediction is 1, label is 1, matched: True
      prediction is 0, label is 0, matched: True
      prediction is 9, label is 9, matched: True
```

Now, run the predictions using TensorFlow 2.1 by executing cell 12 which takes about a second to execute

```
[16]: # cell 12
      predictions2 = predictor2.predict(train_data[:50])
      for i in range(0, 50):
          prediction = np.argmax(predictions2['predictions'][i])
          label = train_labels[i]
          print('prediction is {}, label is {}, matched: {}'.format(prediction, label, prediction == label))

      prediction is 7, label is 7, matched: True
      prediction is 3, label is 3, matched: True
      prediction is 9, label is 4, matched: False
      prediction is 6, label is 6, matched: True
      prediction is 1, label is 1, matched: True
      prediction is 8, label is 8, matched: True
      prediction is 1, label is 1, matched: True
      prediction is 0, label is 0, matched: True
      prediction is 9, label is 9, matched: True
      prediction is 8, label is 8, matched: True
      prediction is 0, label is 0, matched: True
```

Delete the endpoint

After analyzing the results, you can terminate the endpoints by executing cells 13 and 14. Optionally, you can use AWS console to verify that the endpoints are deleted.

```
[17]: # cell 13
      predictor.delete_endpoint()

[18]: # cell 14
      predictor2.delete_endpoint()
```

Conclusion

In this tutorial, we use the SageMaker Python SDK to launch a training job and deploy the trained model. On the training instance, SageMaker's native TensorFlow support sets up training-related environment variables and executes your training script in files `mnist.py` and `mnist-2.py`