Compile and Train the GPT2 Model using the Transformers Trainer API with the SST2 Dataset for Single-Node Multi-GPU Training

SageMaker Training Compiler Overview

SageMaker Training Compiler is a capability of SageMaker that makes these hard-to-implement optimizations to reduce training time on GPU instances. The compiler optimizes DL models to accelerate training by more efficiently using SageMaker machine learning (ML) GPU instances. SageMaker Training Compiler is available at no additional charge within SageMaker and can help reduce total billable time as it accelerates training.

SageMaker Training Compiler is integrated into the AWS Deep Learning Containers (DLCs). Using the SageMaker Training Compiler enabled AWS DLCs, you can compile and optimize training jobs on GPU instances with minimal changes to your code. Bring your deep learning models to SageMaker and enable SageMaker Training Compiler to accelerate the speed of your training job on SageMaker ML instances for accelerated computing.

For more information, see SageMaker Training Compiler (https://docs.aws.amazon.com/sagemaker/latest/dg/training-compiler.html) in the *Amazon SageMaker Developer Guide*.

Introduction

In this demo, you'll use Hugging Face's `transformers` and `datasets` libraries with Amazon SageMaker Training Compiler to train the `gpt-2` model on the `Stanford Sentiment Treebank v2 (SST2)` dataset. To get started, we need to set up the environment with a few prerequisite steps, for permissions, configurations, and so on.

NOTE: You can run this demo in SageMaker Studio, SageMaker notebook instances, or your local machine with AWS CLI set up. If using SageMaker Studio or SageMaker notebook instances, make sure you choose one of the PyTorch-based kernels, `Python 3 (PyTorch x.y Python 3.x CPU Optimized)` or `conda_pytorch_p38` respectively.

NOTE: This notebook uses 2 `ml.g4dn.12xlarge` instances that have multiple GPUs. If you don't have enough quota, see Request a service quota increase for SageMaker resources (https://docs.aws.amazon.com/sagemaker/latest/dg/regions-quotas.html#service-limit-increase-request-procedure).

Development Environment

Installation

This example notebook requires the **SageMaker Python SDK v2.108.0**.

In [137]:
```
!pip install "sagemaker>=2.108.0" botocore boto3 awscli --upgrade
```

```
Looking in indexes: https://pypi.org/simple, https://pip.repos.neuro
n.amazonaws.com
Requirement already satisfied: sagemaker>=2.108.0 in /home/ec2-user/
anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (2.108.0)
Requirement already satisfied: botocore in /home/ec2-user/anaconda3/
envs/pytorch_p38/lib/python3.8/site-packages (1.27.68)
Requirement already satisfied: boto3 in /home/ec2-user/anaconda3/env
s/pytorch_p38/lib/python3.8/site-packages (1.24.68)
Requirement already satisfied: awscli in /home/ec2-user/anaconda3/en
vs/pytorch_p38/lib/python3.8/site-packages (1.25.69)
Requirement already satisfied: importlib-metadata<5.0,>=1.4.0 in /ho
me/ec2-user/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages
(from sagemaker>=2.108.0) (4.8.2)
Requirement already satisfied: packaging>=20.0 in /home/ec2-user/ana
conda3/envs/pytorch_p38/lib/python3.8/site-packages (from sagemaker>
=2.108.0) (21.3)
Requirement already satisfied: protobuf<4.0,>=3.1 in /home/ec2-user/
anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from sagemak
er>=2.108.0) (3.19.4)
Requirement already satisfied: numpy<2.0,>=1.9.0 in /home/ec2-user/a
naconda3/envs/pytorch_p38/lib/python3.8/site-packages (from sagemake
r>=2.108.0) (1.21.2)
Requirement already satisfied: attrs<22,>=20.3.0 in /home/ec2-user/a
naconda3/envs/pytorch_p38/lib/python3.8/site-packages (from sagemake
r>=2.108.0) (21.2.0)
Requirement already satisfied: protobuf3-to-dict<1.0,>=0.1.5 in /hom
e/ec2-user/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (f
rom sagemaker>=2.108.0) (0.1.5)
Requirement already satisfied: smdebug-rulesconfig==1.0.1 in /home/e
c2-user/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from
sagemaker>=2.108.0) (1.0.1)
Requirement already satisfied: pandas in /home/ec2-user/anaconda3/en
vs/pytorch_p38/lib/python3.8/site-packages (from sagemaker>=2.108.0)
(1.3.4)
Requirement already satisfied: google-pasta in /home/ec2-user/anacon
da3/envs/pytorch_p38/lib/python3.8/site-packages (from sagemaker>=2.
108.0) (0.2.0)
Requirement already satisfied: pathos in /home/ec2-user/anaconda3/en
vs/pytorch_p38/lib/python3.8/site-packages (from sagemaker>=2.108.0)
(0.2.8)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-us
er/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from boto
core) (1.26.8)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/
ec2-user/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (fro
m botocore) (2.8.2)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /home/ec2-u
ser/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from bot
ocore) (0.10.0)
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in /home/ec2
-user/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from b
oto3) (0.6.0)
Requirement already satisfied: colorama<0.4.5,>=0.2.5 in /home/ec2-u
ser/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from aws
```

```
cli) (0.4.3)
Requirement already satisfied: rsa<4.8,>=3.1.2 in /home/ec2-user/ana
conda3/envs/pytorch_p38/lib/python3.8/site-packages (from awscli)
(4.7.2)
Requirement already satisfied: docutils<0.17,>=0.10 in /home/ec2-use
r/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from awscl
i) (0.15.2)
Requirement already satisfied: PyYAML<5.5,>=3.10 in /home/ec2-user/a
naconda3/envs/pytorch_p38/lib/python3.8/site-packages (from awscli)
(5.4.1)
Requirement already satisfied: zipp>=0.5 in /home/ec2-user/anaconda3
/envs/pytorch_p38/lib/python3.8/site-packages (from importlib-metada
ta<5.0,>=1.4.0->sagemaker>=2.108.0) (3.6.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/ec2
-user/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from p
ackaging>=20.0->sagemaker>=2.108.0) (3.0.6)
Requirement already satisfied: six in /home/ec2-user/anaconda3/envs/
pytorch_p38/lib/python3.8/site-packages (from protobuf3-to-dict<1.0,
>=0.1.5->sagemaker>=2.108.0) (1.16.0)
Requirement already satisfied: pyasn1>=0.1.3 in /home/ec2-user/anaco
nda3/envs/pytorch_p38/lib/python3.8/site-packages (from rsa<4.8,>=3.
1.2->awscli) (0.4.8)
Requirement already satisfied: pytz>=2017.3 in /home/ec2-user/anacon
da3/envs/pytorch_p38/lib/python3.8/site-packages (from pandas->sagem
aker>=2.108.0) (2021.3)
Requirement already satisfied: ppft>=1.6.6.4 in /home/ec2-user/anaco
nda3/envs/pytorch_p38/lib/python3.8/site-packages (from pathos->sage
maker>=2.108.0) (1.6.6.4)
Requirement already satisfied: multiprocess>=0.70.12 in /home/ec2-us
er/anaconda3/envs/pytorch_p38/lib/python3.8/site-packages (from path
os->sagemaker>=2.108.0) (0.70.12.2)
Requirement already satisfied: pox>=0.3.0 in /home/ec2-user/anaconda
3/envs/pytorch_p38/lib/python3.8/site-packages (from pathos->sagemak
er>=2.108.0) (0.3.0)
Requirement already satisfied: dill>=0.3.4 in /home/ec2-user/anacond
a3/envs/pytorch_p38/lib/python3.8/site-packages (from pathos->sagema
ker>=2.108.0) (0.3.4)
WARNING: You are using pip version 22.0.4; however, version 22.2.2 i
s available.
You should consider upgrading via the '/home/ec2-user/anaconda3/envs
/pytorch_p38/bin/python -m pip install --upgrade pip' command.
```

In [138]:
```
import botocore
import boto3
import sagemaker
import pandas as pd

print(f"sagemaker: {sagemaker.__version__}")
print(f"boto3: {boto3.__version__}")
print(f"botocore: {botocore.__version__}")
```

```
sagemaker: 2.108.0
boto3: 1.24.68
botocore: 1.27.68
```

Copy and run the following code if you need to upgrade IPython widgets for `datasets` library and restart kernel. This is only needed when preprocessing is done in the notebook.

```
%%capture
import IPython
!conda install -c conda-forge ipywidgets -y
# has to restart kernel for the updates to be applied
IPython.Application.instance().kernel.do_shutdown(True)
```

SageMaker environment

In [139]:
```
import sagemaker

sess = sagemaker.Session()

# SageMaker session bucket -> used for uploading data, models and l
ogs
# SageMaker will automatically create this bucket if it does not ex
ist
sagemaker_session_bucket = None
if sagemaker_session_bucket is None and sess is not None:
    # set to default bucket if a bucket name is not given
    sagemaker_session_bucket = sess.default_bucket()

role = sagemaker.get_execution_role()
sess = sagemaker.Session(default_bucket=sagemaker_session_bucket)

print(f"sagemaker role arn: {role}")
print(f"sagemaker bucket: {sess.default_bucket()}")
print(f"sagemaker session region: {sess.boto_region_name}")
```

```
sagemaker role arn: arn:aws:iam::875423407011:role/SageMakerRole
sagemaker bucket: sagemaker-us-west-2-875423407011
sagemaker session region: us-west-2
```

SageMaker Training Job

To create a SageMaker training job, we use an estimator. We use a `PyTorch` estimator for native PyTorch and a `HuggingFace` estimator for SageMaker Training Compiler. Using the estimator, you can define which training script should SageMaker use through `entry_point`, which `instance_type` to use for training, which `hyperparameters` to pass, and so on.

When a SageMaker training job starts, SageMaker takes care of starting and managing all the required machine learning instances, picks up the `PyTorch` or `HuggingFace` Deep Learning Container, uploads your training script, and downloads the data from `sagemaker_session_bucket` into the container at `/opt/ml/input/data`.

First, we define some basic parameters common to all estimators.

Note: We recommend you to turn the SageMaker Debugger's profiling and debugging tools off to avoid additional overheads.

In [140]:
```
estimator_args = dict(
    source_dir="scripts",
    entry_point="run_clm.py",
    instance_type='ml.g4dn.12xlarge',
    instance_count=1,
    role=role,
    py_version="py38",
    volume_size=100,
    disable_profiler=True,  # Disabling SageMaker Profiler to avoid overheads during benchmarking
    debugger_hook_config=False,  # Disabling SageMaker Debugger to avoid overheads during benchmarking
    base_job_name='trcomp-pt-example',
    metric_definitions=[
        {"Name": "summary_train_runtime", "Regex": "'train_runtime': ([0-9.]*)"},
        {"Name": "summary_train_samples_per_second", "Regex": "'train_samples_per_second': ([0-9.]*)"},
        {"Name": "summary_train_steps_per_second", "Regex": "'train_steps_per_second': ([0-9.]*)"},
        {"Name": "summary_train_loss", "Regex": "'train_loss': ([0-9.]*)"},
        {"Name": "epoch", "Regex": "'epoch': ([0-9.]*)"},
        {"Name": "train_loss", "Regex": "'loss': ([0-9.]*)"},
        {"Name": "learning_rate", "Regex": "'learning_rate': ([0-9.]*)"},
    ],
)

# Since ml.g4dn.12xlarge instance has 4 GPUs, we set num_gpus_per_instance to 4
num_gpus_per_instance=4
```

Next, we define some basic arguments to be passed to the training script.

In [141]:
```
# Hyperparameters are passed to the training script as arguments.

hyperparameters = {
    "model_type": "gpt2",
    "tokenizer_name": "gpt2",
    "dataset_name": "glue",
    "dataset_config_name": "sst2",
    "do_train": True,
    "do_eval": False,
    "fp16": True,
    "per_device_eval_batch_size": 8,
    "num_train_epochs": 100,
    "block_size": 512,
    "overwrite_output_dir": True,
    "save_strategy": "no",
    "evaluation_strategy": "no",
    "logging_strategy": "epoch",
    "output_dir": "/opt/ml/model",
    "dataloader_drop_last": True,
}
```

In the following sections, we will create estimators and start training.

Training with Native PyTorch

In the following sections, we will create estimators and start training.

The `per_device_train_batch_size` below is the largest batch we could fit into the memory of a `ml.g4dn.12xlarge` instance. If you change the model, instance type, sequence length, or other parameters that affect memory consumption, you need to find the corresponding largest batch size.

This example uses HuggingFace training script `run_clm.py`, which you can find it inside the `scripts` folder.

In [142]:
```
from sagemaker.pytorch import PyTorch

# The original learning rate was set for a batch of 32. Here we sca
le learning rate linearly with an adjusted batch size
per_device_train_batch_size = 10
global_batch_size = per_device_train_batch_size * num_gpus_per_inst
ance * estimator_args['instance_count']
learning_rate = float("5e-5") / 32 * global_batch_size

# Configure the training job
native_estimator = PyTorch(
    framework_version="1.11",
    hyperparameters=dict(**hyperparameters, **{
        "per_device_train_batch_size": per_device_train_batch_size,
        "learning_rate": learning_rate,
    }),
    distribution={'pytorchddp': {'enabled': True}},
    **estimator_args,
)

# Start the training job
native_estimator.fit(wait=False)

native_estimator.latest_training_job.name
```

Out[142]:
```
'trcomp-pt-example-2022-09-08-18-56-29-423'
```

Training with Optimized PyTorch

Compilation through Training Compiler changes the memory footprint of the model. Most commonly, this manifests as a reduction in memory utilization and a consequent increase in the largest batch size that can fit on the GPU. Note that when you change the batch size, you must adjust the learning rate appropriately. Below, we have scaled the learning rate linearly with the increase in batch size.

Note: We are using distribution mechanism `pytorchxla` which is a compiler aware method of distributed training.

In [143]:
```
from sagemaker.huggingface import HuggingFace, TrainingCompilerConf
ig

# The original learning rate was set for a batch of 32. Here we sca
le learning rate linearly with an adjusted batch size
new_per_device_train_batch_size = 20
global_batch_size = new_per_device_train_batch_size * num_gpus_per_
instance * estimator_args['instance_count']
learning_rate = float("5e-5") / 32 * global_batch_size

# Configure the training job
optimized_estimator = HuggingFace(
    compiler_config=TrainingCompilerConfig(),
    transformers_version="4.21",
    pytorch_version="1.11",
    hyperparameters=dict(**hyperparameters, **{
        "per_device_train_batch_size": new_per_device_train_batch_s
ize,
        "learning_rate": learning_rate,
    }),
    distribution={'pytorchxla': {'enabled': True}},
    **estimator_args,
)

# Start the training job
optimized_estimator.fit(wait=False)

optimized_estimator.latest_training_job.name
```

Out[143]:
```
'trcomp-pt-example-2022-09-08-18-56-29-834'
```

Wait for training jobs to complete

In [145]:
```
waiter = native_estimator.sagemaker_session.sagemaker_client.get_wa
iter(
    "training_job_completed_or_stopped"
)
waiter.wait(TrainingJobName=native_estimator.latest_training_job.na
me)
waiter.wait(TrainingJobName=optimized_estimator.latest_training_jo
b.name)
```

Analysis

Note: If the estimator object is no longer available due to a kernel break or refresh, you need to directly use the training job name and manually attach the training job to a new HuggingFace estimator. For example:

```
native_estimator = PyTorch.attach("your_huggingface_training_job_name")
optimized_estimator = HuggingFace.attach("your_huggingface_training_job_na
me")
```

In [146]:
```
native_estimator = PyTorch.attach(native_estimator.latest_training_
job.name)
optimized_estimator = HuggingFace.attach(optimized_estimator.latest
_training_job.name)
```

```
2022-09-08 19:53:03 Starting - Preparing the instances for training
2022-09-08 19:53:03 Downloading - Downloading input data
2022-09-08 19:53:03 Training - Training image download completed. Tr
aining in progress.
2022-09-08 19:53:03 Uploading - Uploading generated training model
2022-09-08 19:53:03 Completed - Training job completed

2022-09-08 19:41:50 Starting - Preparing the instances for training
2022-09-08 19:41:50 Downloading - Downloading input data
2022-09-08 19:41:50 Training - Training image download completed. Tr
aining in progress.
2022-09-08 19:41:50 Uploading - Uploading generated training model
2022-09-08 19:41:50 Completed - Training job completed
```

Load logs of the training job *with* SageMaker Training Compiler

In [147]:
```
%%capture optimized

# access the logs of the optimized training job
optimized_estimator.sagemaker_session.logs_for_job(optimized_estima
tor.latest_training_job.name)
```

Load logs of the training job *without* SageMaker Training Compiler

In [148]:
```
%%capture native

# access the logs of the native training job
native_estimator.sagemaker_session.logs_for_job(native_estimator.la
test_training_job.name)
```

Create helper functions for analysis

In [149]:
```
from ast import literal_eval
from collections import defaultdict
from matplotlib import pyplot as plt


def _summarize(captured):
    final = []
    for line in captured.stdout.split("\n"):
        cleaned = line.strip()
        if "{" in cleaned and "}" in cleaned:
            final.append(cleaned[cleaned.index("{") : cleaned.index
("}") + 1])
    return final


def make_sense(string):
    try:
        return literal_eval(string)
    except:
        pass


def summarize(summary):
    final = {"train": [], "eval": [], "summary": {}}
    for line in summary:
        interpretation = make_sense(line)
        if interpretation:
            if "loss" in interpretation:
                final["train"].append(interpretation)
            elif "eval_loss" in interpretation:
                final["eval"].append(interpretation)
            elif "train_runtime" in interpretation:
                final["summary"].update(interpretation)
    return final
```

Plot Optimized vs Native Training Throughput

Visualize average throughput as reported by HuggingFace and see potential savings.

In [150]:
```
# Average throughput for the native PyTorch training as reported by Trainer
n = summarize(_summarize(native))
native_throughput = n["summary"]["train_samples_per_second"]

# Average throughput for the optimized PyTorch training as reported by Trainer
o = summarize(_summarize(optimized))
optimized_throughput = o["summary"]["train_samples_per_second"]

# Calculate percentage speedup of optimized PyTorch over native PyTorch
avg_speedup = f"{round((optimized_throughput/native_throughput-1)*100)}%"
```

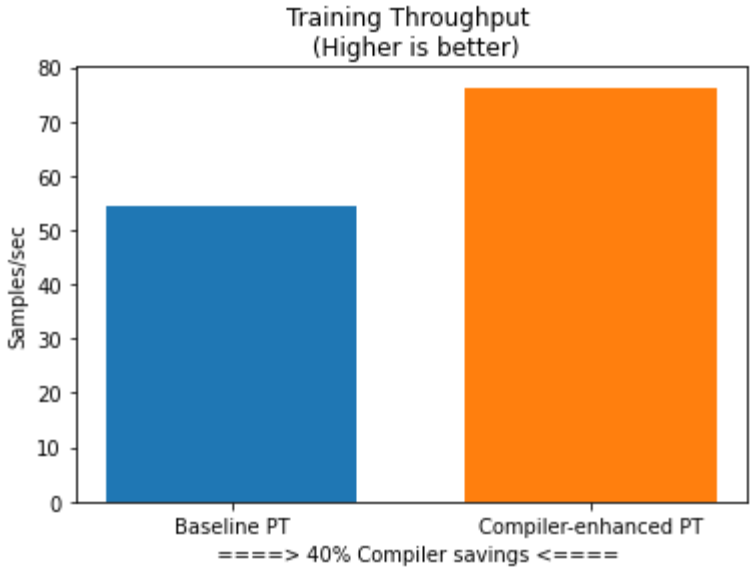
In [151]:
```
%matplotlib inline

plt.title("Training Throughput \n (Higher is better)")
plt.ylabel("Samples/sec")

plt.bar(x=[1], height=native_throughput, label="Baseline PT", width=0.35)
plt.bar(x=[1.5], height=optimized_throughput, label="Compiler-enhanced PT", width=0.35)

plt.xlabel("  ====> {} Compiler savings <====".format(avg_speedup))
plt.xticks(ticks=[1, 1.5], labels=["Baseline PT", "Compiler-enhanced PT"])
```

Out[151]:
```
([<matplotlib.axis.XTick at 0x7f3c818885b0>,
  <matplotlib.axis.XTick at 0x7f3c81888730>],
 [Text(1.0, 0, 'Baseline PT'), Text(1.5, 0, 'Compiler-enhanced PT')])
```



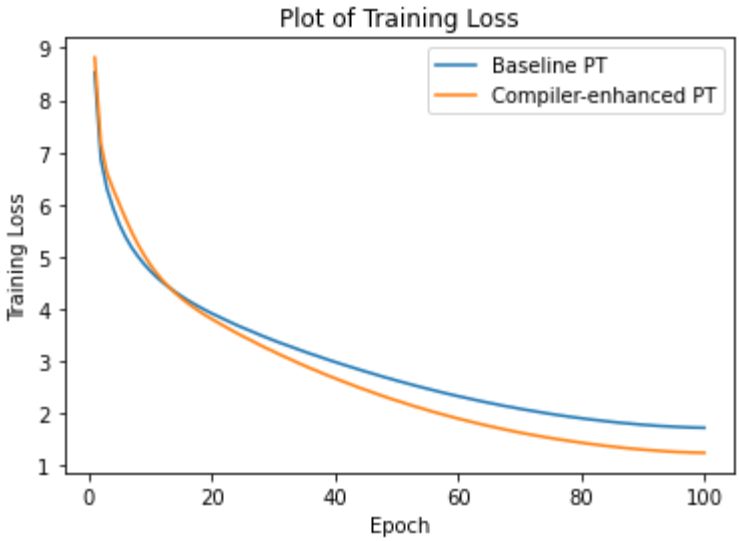
Convergence of Training Loss

SageMaker Training Compiler does not affect the model convergence behavior. Here, we see the loss converges faster than the baseline PyTorch training.

In [152]:
```
vanilla_loss = [i["loss"] for i in n["train"]]
vanilla_epochs = [i["epoch"] for i in n["train"]]
optimized_loss = [i["loss"] for i in o["train"]]
optimized_epochs = [i["epoch"] for i in o["train"]]

plt.title("Plot of Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.plot(vanilla_epochs, vanilla_loss, label="Baseline PT")
plt.plot(optimized_epochs, optimized_loss, label="Compiler-enhanced PT")
plt.legend()
```

Out[152]:
```
<matplotlib.legend.Legend at 0x7f3c80af7bb0>
```



Training Stats

Let's compare various training metrics with and without SageMaker Training Compiler. SageMaker Training Compiler provides an increase in training throughput which translates to a decrease in total training time.

In [153]:
```
import pandas as pd

pd.DataFrame([n["summary"], o["summary"]], index=["Native", "Optimi
zed"])
```

Out[153]:

	train_runtime	train_samples_per_second	train_steps_per_second	train_loss	epoch
Native	2924.4537	54.540	1.334	2.992272	100.0
Optimized	2091.4227	76.264	0.908	2.710749	100.0

In [154]:
```
# calculate percentage speedup from SageMaker Training Compiler in
terms of total training time reported by HF
speedup = (
    (n["summary"]["train_runtime"] - o["summary"]["train_runtime"])
    * 100
    / n["summary"]["train_runtime"]
)
print(
    f"SageMaker Training Compiler integrated PyTorch is about {int
(speedup)}% faster in terms of total training time as reported by H
F."
)
```

```
SageMaker Training Compiler integrated PyTorch is about 28% faster i
n terms of total training time as reported by HF.
```

Total Billable Time

Finally, the decrease in total training time results in a decrease in the billable seconds from SageMaker

In [155]:
```
def BillableTimeInSeconds(name):
    describe_training_job = (
        optimized_estimator.sagemaker_session.sagemaker_client.desc
ribe_training_job
    )
    details = describe_training_job(TrainingJobName=name)
    return details["BillableTimeInSeconds"]
```

In [156]:
```
Billable = {}
Billable["Native"] = BillableTimeInSeconds(native_estimator.latest_
training_job.name)
Billable["Optimized"] = BillableTimeInSeconds(optimized_estimator.l
atest_training_job.name)
pd.DataFrame(Billable, index=["BillableSecs"])
```

Out[156]:

	Native	Optimized
BillableSecs	3297	2608

In [157]:
```
speedup = (Billable["Native"] - Billable["Optimized"]) * 100 / Bill
able["Native"]
print(f"SageMaker Training Compiler integrated PyTorch was {int(spe
edup)}% faster in summary.")
```

```
SageMaker Training Compiler integrated PyTorch was 20% faster in sum
mary.
```

Clean up

Stop all training jobs launched if the jobs are still running.

In [158]:
```
import boto3

sm = boto3.client("sagemaker")


def stop_training_job(name):
    status = sm.describe_training_job(TrainingJobName=name)["Traini
ngJobStatus"]
    if status == "InProgress":
        sm.stop_training_job(TrainingJobName=name)


stop_training_job(native_estimator.latest_training_job.name)
stop_training_job(optimized_estimator.latest_training_job.name)
```

Also, to find instructions on cleaning up resources, see Clean Up (https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html) in the *Amazon SageMaker Developer Guide*.