mseithl Add dependencies in setup.py (#493) · Jebbie1 on Oct 31 · 518 commits

.github	Codeowner file (#521)	2 years ago
bin	Remove scripts to build containers (#213)	3 years ago
config	create buildspec for .net tests docker (#497)	8 months ago
docs	Introducing the profiling functionality. (#407)	3 years ago
examples	Introducing the profiling functionality. (#407)	3 years ago
smdebug	fix: remove hvdvarch pytorch import, preventing segfault with rdma (#...	8 months ago
tests	Fix for importing tensorflow data loader in pytorch hook issue (#604)	8 months ago
.coveragerc	add .coveragerc (#239)	3 years ago
.gitignore	fix protoc and gatherv2 failures in cd pipelines (#189)	3 years ago
pre-commit-config.yaml	fix protoc and gatherv2 failures in cd pipelines (#189)	3 years ago
CODE_OF_CONDUCT.md	merge required files	4 years ago
CONTRIBUTING.md	Remove references to tornasole (#492)	3 years ago
LICENSE	merge required files	4 years ago
LICENSE-THIRD-PARTY	Publishing changes from 0.6 into master (#79)	5 years ago
NOTICE	Update NOTICE	4 years ago
README.md	Fix broken link in README (#672)	2 years ago
THIRD-PARTY	Update THIRD-PARTY	4 years ago
codecov.yml	Create codecov.yml (#236)	3 years ago
compile_protobuf.py	Install Protoc With setup.py (#513)	2 years ago
setup.cfg	Rename package "tornasole" to "smdebug (#362)	3 years ago
setup.py	Add dependencies in setup.py (#493)	2 months ago

Amazon SageMaker Debugger

codecov 100% · pypi package 0.9.10

Table of Contents

- Overview
- Install the smdebug library
- Debugger-supported Frameworks
- How It Works
- Examples
- SageMaker Debugger in action
- Further Documentation and References

Overview

Amazon SageMaker Debugger automates the debugging process of machine learning training jobs. From training jobs, Debugger allows you to run your own training script (Zero Script Change experience) using Debugger built-in features—Hooks and Rules—to capture tensors, have flexibility to build customized Hooks and Rules for configuring tensors as you want, and make the tensors available for analysis by saving in an Amazon S3 bucket, all through a flexible and powerful API.

The smdebug library powers Debugger by calling the saved tensors from the S3 bucket during the training job. smdebug retrieves and filters the tensors generated from Debugger such as gradients, weights, and biases.

Debugger helps you develop better, faster, and cheaper models by minimally modifying estimator, training the tensors, catching anomalies while training models, and iterative model pruning.

Debugger supports TensorFlow, PyTorch, MXNet, and XGBoost frameworks.

The following list is a summary of the main functionalities of Debugger.

- Run and debug training jobs of your model on SageMaker when using supported containers
- No changes needed to your training script if using AWS Deep Learning Containers with Debugger fully integrated
- Minimal changes to your training script if using AWS containers with script mode or custom containers
- Full visibility into any tensor relevant from targeted parts of the training jobs
- Real-time training job monitoring through Rules
- Automated anomaly detection and state assertions through built-in and custom Rules on SageMaker
- Actions on your training jobs based on the status of Rules
- Interactive exploration of saved tensors
- Distributed training support
- TensorBoard support

See How it works for more details.

Install the smdebug library

The smdebug library runs on Python 3. Install using the following command:

```
pip install smdebug
```

Debugger-supported Frameworks

For a complete overview of Amazon SageMaker Debugger to learn how it works, go to the Use Debugger in AWS Containers developer guide.

AWS Deep Learning Containers with zero code change

Debugger is installed by default in AWS Deep Learning Containers with TensorFlow, PyTorch, MXNet, and XGBoost. The following framework versions enable you to use Debugger with no changes to your training script, by automatically adding SageMaker Debugger's hook.

The following frameworks are available AWS Deep Learning Containers with the deep learning frameworks for the zero script change experience.

Framework	Version
TensorFlow	1.15, 2.1.0, 2.2.0, 2.3.0, 2.3.1
MXNet	1.6, 1.7
PyTorch	1.4, 1.5, 1.6
XGBoost	0.90-2, 1.0-1 (As a built-in algorithm)

Note Debugger with zero script change is partially available for TensorFlow v2.1. The inputs, outputs, gradients, and layers built-in collections are currently not available for these TensorFlow versions.

AWS training containers with script mode

The smdebug library supports frameworks other than the ones listed above while using AWS containers with script mode. If you want to use SageMaker Debugger with one of the following framework versions, you need to make minimal changes to your training script.

Framework	Versions
TensorFlow	1.13, 1.14, 1.15, 2.1.0, 2.2.0, 2.3.0, 2.3.1
Keras (with TensorFlow backend)	2.3
MXNet	1.4, 1.5, 1.6, 1.7
PyTorch	1.2, 1.3, 1.4, 1.5, 1.6
XGBoost	0.90-2, 1.0-1 (As a framework)

Debugger on custom containers or local machines

You can also fully use the Debugger feature in custom containers with the SageMaker Python SDK. Furthermore, smdebug is an open source library, so you can install it on your local machine for any advanced use cases that cannot be run in the SageMaker environment and for constructing smdebug custom hooks and rules.

How It Works

Amazon SageMaker Debugger uses the construct of a Hook to save the values of requested tensors throughout the training process. You can then setup a Rule job which simultaneously monitors and validates these tensors to ensure that training is progressing as expected.

A Rule checks for vanishing gradients, exploding tensor values, or poor weight initialization. Rules are attached to Amazon CloudWatch events, so that when a rule is triggered it changes the state of the CloudWatch event. You can configure any action on the CloudWatch event, such as to stop the training job saving you time and money.

Debugger can be used inside or outside of SageMaker. However the built-in rules that AWS provides are only available for SageMaker training. Scenarios of usage can be classified into the following three cases.

Using SageMaker Debugger on AWS Deep Learning Containers with zero training script change

Use Debugger built-in hook configurations and rules while setting up the estimator and monitor your training job.

For a full guide and examples of using the built in rules, see Running a Rule with zero script change on AWS Deep Learning Containers.

To see a complete list of built-in rules and their functionalities, see List of Debugger Built-in Rules.

Using SageMaker Debugger on AWS training containers with script mode

You can use Debugger with your training script on your own container making only a minimal modification to your training script to add Debugger's Hook. For an example template of code to use Debugger on your own container on TensorFlow 2.x framework, see Run Debugger in custom container. See the following instruction pages to set up Debugger in your preferred framework.

- TensorFlow
- MXNet
- PyTorch
- XGBoost

Using SageMaker Debugger on custom containers

Debugger is available for any deep learning models that you bring to Amazon SageMaker. The AWS CLI, the SageMaker Estimator API, and the Debugger APIs enable you to use any Docker base images built and customize containers to train and debug your models. To use Debugger with customized containers, go to Use Debugger in Custom Training Containers.

Using SageMaker Debugger on a non-SageMaker environment

Using the smdebug library, you can create custom hooks and rules (or manually analyze the tensors) and modify your training script to enable tensor analysis on a non-SageMaker environment, such as your local machine. For an example of this, see Run Debugger locally.

Examples

SageMaker Notebook Examples

To find a collection of demonstrations using Debugger, see SageMaker Debugger Example Notebooks.

Run Debugger rules with zero script change

This example shows a how to use Debugger with Zero Script Change of your training script on a SageMaker DLC.

```
import sagemaker as sm
from sagemaker.debugger import rule_configs, Rule, CollectionConfig

# Choose a built-in rule to monitor your training job
rule = Rule.sagemaker(
    rule_configs.vanishing_tensor(),
    rule_to_invoke="VanishingTensor",
    rule_parameters={"tensor_regex": ".*"},
    # specify collections to save for processing your rule
    collections_to_save=[
        CollectionConfig(name="weights"),
        CollectionConfig(name="losses"),
    ]
)

# Pass the rule to the estimator
sagemaker_simple_estimator = sm.TensorFlow(
    entry_point="script.py", # regular script.py to your own training script
    role=sm.get_execution_role(),
    instance_count=1,
    instance_type="ml.c5.2xlarge",
    py_version="py3",
    # argument for smdebug below
    rules=[rule],
)

sagemaker_simple_estimator.fit()
tension_path = sagemaker_simple_estimator.latest_job_debugger_artifacts_path()

import smdebug.trials as smd
trial = smd.create_trial(out_dir=tensors_path)
print("Saved these tensors: {trial.tensor_names()}")
print("Loss values during evaluation are {trial.tensor('CrossEntropyLoss:0').values(mode=smd.modes
```

That's it! When you configure the sagemaker_simple_estimator, you simply specify the entry_point to your training script python file. When you run the sagemaker_simple_estimator.fit() API, SageMaker will automatically monitor your training job for you with the Rules specified and create a trial object that tracks the status of the Rule, so you can take any action based on them.

If you want additional configuration and control, see Running SageMaker jobs with Debugger for more information.

Run Debugger in custom container

The following example shows how to set Hook to set a training model using Debugger in your own container. This example is for containers in TensorFlow 2.x framework using GradientTape to configure the Hook.

```
import smdebug.tensorflow as smd
hook = smd.KerasHook(out_dir=sys.out_dir)

model = tf.keras.models.Sequential([ ... ])
for epoch in range(0, epochs):
    for step in range(0, steps):
        dataset_labels = labels
        # wrap the tape to capture tensors
        with hook.wrap_tape(tf.GradientTape(persistent=True)) as tape:
            logits = model(dataset_inputs, training = True)
            loss_value = cce(dataset_labels, logits)
        grads = tape.gradient(loss_value, model.variables)
        opt.apply_gradients(zip(grads, model.variables))
        acc = train_acc_metric(dataset_labels, logits)
        # manually record the values of the loss
        hook.record_tensor_value(tensor_name="accuracy", tensor_value=acc)
```

Run Debugger locally

This example shows how to use Debugger for the Keras model.fit() API.

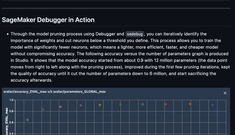
To use Debugger, simply add a callback hook.

```
import smdebug.tensorflow as smd
hook = smd.KerasHook(out_dir='/ood_outputs')

model = tf.keras.models.Sequential([ ... ])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
)

# Add the hook as a callback
model.fit(x_train, y_train, epochs=1, callbacks=[hook])
model.evaluate(x_test, y_test, callbacks=[hook])

# Create a trial to inspect the saved tensors
trial = smd.create_trial(path='/ood_outputs')
print("Saved these tensors: {trial.tensor_names()}")
print("Loss values during evaluation were {trial.tensor('CrossEntropyLoss:0').values(mode=smd.modes
```

SageMaker Debugger in Action

- Through the model pruning process using Debugger and smdebug, you can iteratively identify the importances of weights and cut neurons below a threshold you define. This process allows you to train the model with significantly fewer neurons, which means a lighter, more efficient, faster, and cheaper model without compromising accuracy. The following accuracy versus the number of parameters graph is produced in Studio. It shows that the model accuracy started from about 0.84 with 12 million parameters (the data point moves from right to left along with the pruning process), improved during the first few pruning iterations, kept the quality of accuracy until it cut the number of parameters down to 8 million, and start sacrificing the accuracy afterwards.