NVTX Plugins for Deep Learning

Release 0.1.8

unknown

Jun 09, 2023

USER GUIDE

1 Installation **3**
- 1.1 Installing NVTX-Plugins . . . 3
- 1.2 Installing from github . . . 3
- 1.3 Installing from source . . . 3
- 1.4 Build the documentation . . . 4
- 1.5 Nsight Systems . . . 4

2 API **5**
- 2.1 TensorFlow ops . . . 5
- 2.2 Session hooks . . . 7
- 2.3 Keras Layers . . . 7
- 2.4 Keras Callbacks . . . 8

3 Examples **9**
- 3.1 Keras example . . . 9
- 3.2 TensorFlow session example . . . 11

4 Frequently Asked Questions **15**
- 4.1 How is NVTX Plugins different from the built in markers in the NGC TensorFlow container ? . . . 15
- 4.2 Can NVTX Plugins be used with eager execution ? . . . 15
- 4.3 Is there an overhead to using NVTX Plugins ? . . . 15
- 4.4 In the example scripts, what does environment variable CUDA_LAUNCH_BLOCKING do ? . . . 15

5 Changelog **17**
- 5.1 Version 0.1.3 . . . 17
- 5.2 Version 0.1.2 . . . 17
- 5.3 Version 0.1.1 . . . 17
- 5.4 Version 0.1.0 . . . 17

Index **19**

Documentation Version: 0.1.8

NVTX Plugins allows users to add their own NVIDIA Tools Extension (NVTX) events and time ranges to a TensorFlow graph. Applications which integrate NVTX can use NVIDIA Nsight Systems, Nsight Compute, and Visual Profiler to capture and visualize these events and time ranges.

CHAPTER

ONE

INSTALLATION

1.1 Installing NVTX-Plugins

The package can be installed from PyPI:

```
# Stable release
pip install nvtx-plugins

# Pre-release (may present bugs)
pip install nvtx-plugins --pre
```

The package is also available for download on github: https://github.com/NVIDIA/nvtx-plugins/releases

```
pip install nvtx-plugins*.tar.gz
```

1.2 Installing from github

You can build and install the package from the github repository:

```
# Install Master Branch
pip install git+https://github.com/NVIDIA/nvtx-plugins

# Install Specific Commit (In this case commit 7d46c3a)
pip install git+https://github.com/NVIDIA/nvtx-plugins@7d46c3a

# Install Specific Branch (In this case branch master)
pip install git+https://github.com/NVIDIA/nvtx-plugins@master

# Install Specific Release (In this case 0.1.7)
pip install git+https://github.com/NVIDIA/nvtx-plugins@0.1.7
```

1.3 Installing from source

You can build and install the package from source:

```
python setup.py sdist
pip install dist/nvtx-plugins*.tar.gz
```

For development objectives, you can install the package directly from source with:

```
python setup.py install
```

We recommend building the package inside NVIDIA's NGC TensorFlow container: https://ngc.nvidia.com/catalog/containers/nvidia:tensorflow

For more information about how to get started with NVIDIA's NGC containers, see the following sections from the NVIDIA GPU Cloud Documentation and the Deep Learning DGX Documentation: Getting Started Using NVIDIA GPU Cloud, Accessing And Pulling From The NGC container registry and Running TensorFlow.

1.4 Build the documentation

The documentation is built by running:

```
cd docs
pip install -r requirements.txt
make html
```

The documentation files will be generated in *docs/build/html*

Building the documentation does not require NVTX Plugins to be installed. Nonetheless, due to an issue in Sphinx **only Python 3.7 is supported** to build the documentation.

1.5 Nsight Systems

NVIDIA Nsight Systems and can be downloaded and from the NVIDIA's Developer Website. *nsys* is preinstalled in our NGC TensorFlow container.

More details about nsys and Nsight Systems can be found here.

CHAPTER

TWO

API

2.1 TensorFlow ops

nvtx.plugins.tf.ops.**start**(*inputs*, *message*, *domain_name=None*, *grad_message=None*, *grad_domain_name=None*, *trainable=False*, *enabled=True*, *name=None*)

An identity operation with a side effect of opening an NVTX marker.

Note: The *ops.start* and *ops.end* operations must be used in pairs.

Example

```
x, nvtx_context = nvtx.plugins.tf.ops.start(x, message='Dense 1-3',
    domain_name='Forward', grad_domain_name='Gradient')
x = tf.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_1')
x = tf.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_2')
x = tf.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_3')
x = nvtx.plugins.tf.ops.end(x, nvtx_context)
```

Parameters

- **inputs** – A `Tensor` object that is passed to `output`.
- **message** – A `string` message to be associated with this marker.
- **domain_name** – An optional `string` domain name to be associated with this marker. If not provided the default NVTX domain will be used.
- **grad_message** – An optional `string` message to be associated with the op gradient. If not provided `message` will be used.
- **grad_domain_name** – An optional `string` domain name to be associated with this marker gradient. If not provided `domain_name` will be used.
- **trainable** – `bool`, if `True` will make this op trainable. Used when this is the first operation in the graph to prevent an open ended marker during gradient calculation.
- **enabled** – `bool`, if `False` the nvtx marker will be disabled.
- **name** – An optional *string* name for the operation.

Returns

- output: The inputs `Tensor`.

- nvtx_context: `list`, NVTX context associated with this op and passed to *ops.end*. `None` if `enabled=False`.

Return type `tuple`

`nvtx.plugins.tf.ops.`**`end`**(*inputs*, *nvtx_context*, *name=None*)
An identity operation with a side effect of closing an NVTX marker.

Note: The *ops.start* and *ops.end* operations must be used in pairs.

Example

```
x, nvtx_context = nvtx.plugins.tf.ops.start(x, message='Dense 1-3',
    domain_name='Forward', grad_domain_name='Gradient')
x = tf.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_1')
x = tf.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_2')
x = tf.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_3')
x = nvtx.plugins.tf.ops.end(x, nvtx_context)
```

Parameters

- **`inputs`** – A `Tensor` object that will be passed to `output`.
- **`nvtx_context`** – `list`, NVTX context received from *ops.start* If *None* the marker will be disabled.
- **`name`** – An optional `string` name for the operation.

Returns The inputs `Tensor`.

`@nvtx.plugins.tf.ops.`**`trace`**(*message*, *domain_name=None*, *grad_message=None*, *grad_domain_name=None*, *trainable=False*, *enabled=True*, *name=None*)
An identity function decorator with a side effect of adding NVTX marker.

Note: The decorator expects the wrapped function to take the input `Tensor` as the first argument or to be named `inputs`, and to return a single `Tensor`.

Parameters

- **`message`** – A `string` message to be associated with this marker.
- **`domain_name`** – An optional `string` domain name to be associated with this marker. If not provided the default NVTX domain will be used.
- **`grad_message`** – An optional `string` message to be associated with the op gradient. If not provided *message* will be used.
- **`grad_domain_name`** – An optional `string` domain name to be associated with this marker gradient. If not provided `domain_name` will be used.
- **`trainable`** – `bool`, if `True` will make this op trainable. Used when this is the first operation in the graph to prevent an open ended marker during gradient calculation.
- **`enabled`** – `bool`, if `False` the nvtx marker will be disabled.

- **name** – An optional `string` name for the operation.

2.2 Session hooks

class nvtx.plugins.tf.estimator.**NVTXHook**(*skip_n_steps=0*, *name=None*)
Hook that adds NVTX markers to a TensorFlow session.

Parameters

- **skip_n_steps** – `int`, skips adding markers for the first N `session.run()` calls.
- **name** – `string`, a marker name for the session.

2.3 Keras Layers

class nvtx.plugins.tf.keras.layers.**NVTXStart**(*message*, *domain_name=None*, *trainable=False*, ***kwargs*)
An identity layer with a side effect of opening an NVTX marker.

Note: The *NVTXStart* and *NVTXEnd* layers must be used in pairs.

Example

```
x, marker_id, domain_id = NVTXStart(message='Dense',
                                    domain_name='forward')(x)
x = Dense(1024, activation='relu')(x)
x = NVTXEnd(grad_message='Dense grad',
            grad_domain_name='backwards')([x, marker_id, domain_id])
```

Parameters

- **message** – A `string` message to be associated with this layer.
- **domain_name** – An optional `string` domain name to be associated with this layer. If not provided the default NVTX domain will be used.
- **trainable** – `bool`, if `True` will make this layer trainable. Used when this is the first layer in the graph to prevent an open ended marker during gradient calculation.
- **name** – An optional `string` name for the layer.

Input shape: A `Tensor` object that is passed to `output`.

Output shape:

`list` of length 3:

- output: The inputs `Tensor`.
- marker_id: `int64 Tensor`, sent to *NVTXEnd*.
- domain_handle: `int64 Tensor`. sent to *NVTXEnd*.

class nvtx.plugins.tf.keras.layers.**NVTXEnd**(*grad_message=None*, *grad_domain_name=None*, ***kwargs*)

An identity layer with a side effect of closing an NVTX marker.

Note: The *NVTXStart* and *NVTXEnd* layers must be used in pairs.

Example

```
x, marker_id, domain_id = NVTXStart(message='Dense',
                                    domain_name='forward')(x)
x = Dense(1024, activation='relu')(x)
x = NVTXEnd(grad_message='Dense grad',
            grad_domain_name='backwards')([x, marker_id, domain_id])
```

Parameters

- **grad_message** – An optional `string` message to be associated with the op gradient. If not provided an empty message will be used.
- **grad_domain_name** – An optional `string` domain name to be associated with this marker gradient. If not provided the default domain name will be used.
- **name** – An optional `string` name for the layer.

Input shape:

`list` of length 3:

- inputs: The input `Tensor`.
- marker_id: `int64 Tensor` from *NVTXStart*.
- domain_handle: `int64 Tensor` from *NVTXStart*.

Output shape: A `Tensor` with `inputs` shape.

2.4 Keras Callbacks

class nvtx.plugins.tf.keras.callbacks.**NVTXCallback**(***kwargs*)

Callback that adds NVTX markers to a keras session.

CHAPTER

THREE

EXAMPLES

3.1 Keras example

```
# -*- coding: utf-8 -*-

# Copyright (c) 2019, NVIDIA CORPORATION. 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.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License 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.

import os
import numpy as np

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

from tensorflow.keras import optimizers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

from nvtx.plugins.tf.keras.layers import NVTXStart, NVTXEnd
from nvtx.plugins.tf.keras.callbacks import NVTXCallback

TRAINING_STEPS = 5000

# load pima indians dataset
dataset = np.loadtxt('examples/pima-indians-diabetes.data.csv', delimiter=',')
features = dataset[:, 0:8]
labels = dataset[:, 8]


def DenseBinaryClassificationNet(input_shape=(8,)):
    inputs = Input(input_shape)

    x = inputs
```

(continues on next page)

(continued from previous page)

```
    x, marker_id, domain_id = NVTXStart(message='Dense 1',
                                        domain_name='forward',
                                        trainable=True)(x)
    x = Dense(1024, activation='relu')(x)
    x = NVTXEnd(grad_message='Dense 1 grad',
                grad_domain_name='backwards')([x, marker_id, domain_id])

    x, marker_id, domain_id = NVTXStart(message='Dense 2',
                                        domain_name='forward')(x)
    x = Dense(1024, activation='relu')(x)
    x = NVTXEnd(grad_message='Dense 2 grad',
                grad_domain_name='backwards')([x, marker_id, domain_id])

    x, marker_id, domain_id = NVTXStart(message='Dense 3',
                                        domain_name='forward')(x)
    x = Dense(512, activation='relu')(x)
    x = NVTXEnd(grad_message='Dense 3 grad',
                grad_domain_name='backwards')([x, marker_id, domain_id])

    x, marker_id, domain_id = NVTXStart(message='Dense 4',
                                        domain_name='forward')(x)
    x = Dense(512, activation='relu')(x)
    x = NVTXEnd(grad_message='Dense 4 grad',
                grad_domain_name='backwards')([x, marker_id, domain_id])

    x, marker_id, domain_id = NVTXStart(message='Dense 5',
                                        domain_name='forward')(x)
    x = Dense(1, activation='sigmoid')(x)
    x = NVTXEnd(grad_message='Dense 5 grad',
                grad_domain_name='backwards')([x, marker_id, domain_id])

    predictions = x
    model = Model(inputs=inputs, outputs=predictions)
    return model


nvtx_callback = NVTXCallback()

model = DenseBinaryClassificationNet()
sgd = optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd,
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(
    features,
    labels,
    batch_size=128,
    callbacks=[nvtx_callback],
    epochs=1,
    steps_per_epoch=TRAINING_STEPS
)
```

3.2 TensorFlow session example

```
# -*- coding: utf-8 -*-

# Copyright (c) 2019, NVIDIA CORPORATION. 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.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License 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.

import os
import numpy as np

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import tensorflow as tf
import nvtx.plugins.tf as nvtx_tf
from nvtx.plugins.tf.estimator import NVTXHook

ENABLE_NVTX = True
TRAINING_STEPS = 5000


def batch_generator(features, labels, batch_size, steps):
    dataset_len = len(labels)
    idxs = list(range(dataset_len))

    idxs_trunc = None

    steps_per_epoch = dataset_len // batch_size

    for step in range(steps):

        start_idx = batch_size * (step % steps_per_epoch)

        end_idx = batch_size * ((step + 1) % steps_per_epoch)
        end_idx = end_idx if end_idx != 0 else (steps_per_epoch * batch_size)

        if step % (steps_per_epoch) == 0:
            np.random.shuffle(idxs)
            idxs_trunc = idxs[0:batch_size * steps_per_epoch]

        x_batch = np.array([features[j] for j in idxs_trunc[start_idx:end_idx]])

        y_batch = np.array([labels[j] for j in idxs_trunc[start_idx:end_idx]])
        y_batch = np.expand_dims(y_batch, axis=1)

        yield x_batch, y_batch
```

(continues on next page)

(continued from previous page)

```
# Option 1: use decorators
@nvtx_tf.ops.trace(message='Dense Block', grad_message='Dense Block grad',
                   domain_name='Forward', grad_domain_name='Gradient',
                   enabled=ENABLE_NVTX, trainable=True)
def DenseBinaryClassificationNet(inputs):
    x = inputs
    x, nvtx_context = nvtx_tf.ops.start(x, message='Dense 1',
        grad_message='Dense 1 grad', domain_name='Forward',
        grad_domain_name='Gradient', trainable=True, enabled=ENABLE_NVTX)
    x = tf.compat.v1.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_1')
    x = nvtx_tf.ops.end(x, nvtx_context)

    x, nvtx_context = nvtx_tf.ops.start(x, message='Dense 2', grad_message='Dense 2␣
↪grad',
        domain_name='Forward', grad_domain_name='Gradient', enabled=ENABLE_NVTX)
    x = tf.compat.v1.layers.dense(x, 1024, activation=tf.nn.relu, name='dense_2')
    x = nvtx_tf.ops.end(x, nvtx_context)

    x, nvtx_context = nvtx_tf.ops.start(x, message='Dense 3', grad_message='Dense 3␣
↪grad',
        domain_name='Forward', grad_domain_name='Gradient', enabled=ENABLE_NVTX)
    x = tf.compat.v1.layers.dense(x, 512, activation=tf.nn.relu, name='dense_3')
    x = nvtx_tf.ops.end(x, nvtx_context)

    x, nvtx_context = nvtx_tf.ops.start(x, message='Dense 4', grad_message='Dense 4␣
↪grad',
        domain_name='Forward', grad_domain_name='Gradient', enabled=ENABLE_NVTX)
    x = tf.compat.v1.layers.dense(x, 512, activation=tf.nn.relu, name='dense_4')
    x = nvtx_tf.ops.end(x, nvtx_context)

    x, nvtx_context = nvtx_tf.ops.start(x, message='Dense 5', grad_message='Dense 5␣
↪grad',
        domain_name='Forward', grad_domain_name='Gradient', enabled=ENABLE_NVTX)
    x = tf.compat.v1.layers.dense(x, 1, activation=None, name='dense_5')
    x = nvtx_tf.ops.end(x, nvtx_context)

    predictions = x
    return predictions

tf.compat.v1.disable_eager_execution()

# Load Dataset
dataset = np.loadtxt('examples/pima-indians-diabetes.data.csv', delimiter=',')
features = dataset[:, 0:8]
labels = dataset[:, 8]

# tf Graph Inputs
features_plh = tf.compat.v1.placeholder('float', [None, 8])
labels_plh = tf.compat.v1.placeholder('float', [None, 1])

logits = DenseBinaryClassificationNet(inputs=features_plh)
loss = tf.math.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=logits,␣
↪labels=labels_plh))
acc = tf.math.reduce_mean(tf.compat.v1.metrics.accuracy(labels=labels_plh,␣
↪predictions=tf.round(tf.nn.sigmoid(logits))))
optimizer = tf.compat.v1.train.MomentumOptimizer(learning_rate=0.01, momentum=0.9,␣
↪use_nesterov=True).minimize(loss)
```

(continues on next page)

(continued from previous page)

```
# Initialize variables. local variables are needed to be initialized for tf.metrics.*
init_g = tf.compat.v1.global_variables_initializer()
init_l = tf.compat.v1.local_variables_initializer()

nvtx_callback = NVTXHook(skip_n_steps=1, name='Train')

# Start training
with tf.compat.v1.train.MonitoredSession(hooks=[nvtx_callback]) as sess:
    sess.run([init_g, init_l])

    # Run graph
    for step, (x, y) in enumerate(batch_generator(features, labels, batch_size=128,␣
↪steps=TRAINING_STEPS)):
        _, loss_, acc_ = sess.run(
            [optimizer, loss, acc],
            feed_dict={features_plh: x, labels_plh: y}
        )

        if step % 100 == 0:
            print('Step: %04d, loss=%f acc=%f' % (step, loss_, acc_))

    print('\nFinal loss=%f acc=%f' % (loss_, acc_))

print('Optimization Finished!')
```

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS

4.1 How is NVTX Plugins different from the built in markers in the NGC TensorFlow container ?

The NVTX markers in the NGC TensorFlow container wrap a single graph node call and don't modify the graph. NVTX plugins allows users to add their own markers to highlight specific layers or parts of their model by adding NVTX nodes to the graph. The built in TensorFlow markers can be disabled by setting the environment variable *TF_DISABLE_NVTX_RANGES*.

4.2 Can NVTX Plugins be used with eager execution ?

Yes, the Keras layers fully support eager execution. However, the nvtx markers are still added and executed at the graph level and not in python.

We plan to add python level calls in the future.

4.3 Is there an overhead to using NVTX Plugins ?

NVTX has a small overhead and when no NVTX logger is present this overhead amounts to an empty function call. However, NVTX Plugins works by adding nodes to the graph, and therefore has at least the overhead of initializing and calling an additional TensorFlow operation. This overhead is small and mostly negligible in large models.

In general, NVTX Plugins is intended for profiling and debugging, it is not recommended for use in deployed code.

4.4 In the example scripts, what does environment variable CUDA_LAUNCH_BLOCKING do ?

The environment variable *CUDA_LAUNCH_BLOCKING* disables asynchronously kernel launches and is useful for debugging.

More about asynchronous execution at: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#asynchronous-concurrent-execution

CHAPTER

FIVE

CHANGELOG

5.1 Version 0.1.3

Date August 21, 2019

- PyPI Description Added

5.2 Version 0.1.2

Date July 18, 2019

- URLs fix on PyPI
- Simplification of setup process on the backend
- No feature change

5.3 Version 0.1.1

Date July 17, 2019

- Release on PyPI
- Automatic build triggered during python extension installation: Extension
- Non Closed Ranges Exception raised to ease user-debug

5.4 Version 0.1.0

Date June 19, 2019

Initial Release (Beta)

INDEX

E

end() (*in module nvtx.plugins.tf.ops*), 6

N

NVTXCallback (*class in nvtx.plugins.tf.keras.callbacks*), 8
NVTXEnd (*class in nvtx.plugins.tf.keras.layers*), 7
NVTXHook (*class in nvtx.plugins.tf.estimator*), 7
NVTXStart (*class in nvtx.plugins.tf.keras.layers*), 7

S

start() (*in module nvtx.plugins.tf.ops*), 5

T

trace() (*in module nvtx.plugins.tf.ops*), 6