Documentation for developing the PyTorch-ONNX exporter (`torch.onnx`).

Table of Contents

Development process

Environment setup

We highly recommend using Linux. Other platforms are not tested in PyTorch CI and are generally not used by the torch.onnx developers.

Fork PyTorch

Fork github.com/pytorch/pytorch and clone your fork to your workstation.

Build PyTorch

CUDA is not required for most development tasks. If you use CUDA, building PyTorch will probably be slower.

Install Anaconda and activate a new environment.

Install direnv and initialize your envrc file in the root of your PyTorch git repo:

```
# Make the local package name built by `setup.py develop` the same
# as the one that's on conda.
echo "export TORCH_PACKAGE_NAME=pytorch" >> .envrc
# Let CMake find binaries and libs installed by conda.
echo 'export CMAKE_PREFIX_PATH=${CONDA_PREFIX:-"$(dirname $(which conda))/../"}' >>
.envrc
direnv allow
```

Then see the instructions in PyTorch's README.

Optional build tips

PyTorch C++ development tips.

Use direnv for Anaconda environment selection.

Set more environment variables in your .envrc file:

```
# Only if you're building without CUDA.
export USE_CUDA=0
# Only if you're building with ccache.
PATH_add /usr/lib/ccache
# Needed for older compilers or conda compilers
export LDFLAGS='-lrt'
# Build with debug symbols.
export DEBUG=1
```

Install additional dependencies

Install the dependencies required to run CI checks locally.

```
# flake8 version restriction due to https://github.com/pytorch/pytorch/issues/69500
conda install -c conda-forge expecttest pytest mypy=0.812 'flake8<4' hypothesis
```

ONNX Runtime

```
# install a version of protobuf compatible with ONNX submodule
conda install -c conda-forge protobuf=$(cat third_party/onnx/requirements-
release.txt | grep protobuf | awk '{print $3}') flatbuffers
pip install onnxruntime
```

Onnxruntime is also available from conda-forge, but it seems to demand a version of protobuf that's newer than what the ONNX submodule wants to use, which lead to seg-faults in my case. This may be resolved with future versions of ONNX or ONNX Runtime. If you find `conda install -c conda-forge onnxruntime` works, please update these instructions.

If you need a newer or different version of ONNX Runtime than what is available via conda, you can instead install it from source.

TorchVision

```
# cpuonly not needed if you're using CUDA
conda install -c pytorch-nightly torchvision cpuonly
conda uninstall --force pytorch
```

This first command installs PyTorch as a dependency, so we remove it with the second command so that you can use the locally-built pytorch rather than the one from conda. If you later run into issues with conda complaining about an inconsistent environment, you can temporarily reinstall PyTorch via conda:

```
conda update -c pytorch-nightly torchvision cpuonly
```

I hope there's a better way to deal with this. If you know of one please update these instructions and email the team!

Sanity check

You should be able to run these commands successfully:

```
python setup.py develop
python test/onnx/test_pytorch_onnx_onnxruntime.py
TestONNXRuntime_opset10.test_arithmetic_prim_long
```

And this should fail:

```
echo "assert False" >> torch/onnx/utils.py
python test/onnx/test_pytorch_onnx_onnxruntime.py
TestONNXRuntime_opset10.test_arithmetic_prim_long
git restore torch/onnx/utils.py
```

If the second command succeeds, then probably python is finding the PyTorch that was installed via `conda` or `pip`, not the one that was built from source by `python setup.py develop`.

Pull requests

PRs should be opened directly against master. PRs can be directly merged into master as long as it satisfies the ONNX merge rule:

- Approved by one of torch.onnx developers listed in `approved_by` section.
- All modified files fall under the `patterns` section.

Pay special attention to the following GitHub checks:

- Has "onnx" in the name, which runs ONNX related tests.
- Has "Lint" in the name, which does code format checks.

Regarding other failing GitHub checks, if you are certain the failure is unrelated to your change, try rebasing with master. Often times these kind of failures are caused by branch out of sync with master. For rare occasions, You can ignore the failing check if it is a regression in master. This can be verified by checking if master is also failing from CI HUD for PyTorch.

To merge your pull request, comment on the PR "@pytorchbot merge this".

If you make changes to non-ONNX related code, i.e. files out side of ONNX merge rule, please note the PR will require additional reviews from people outside of torch.onnx developers, and will take a longer process to merge into master. In this case, pytorchbot will not be able to merge the pull request. It will leave a comment like "Merge failed due to PR XXX does not match merge rules". Please label the pull request with `onnx-needs-import`.

See GitHub pull request workflow.

Adhere to Google's Code Review Developer Guide.

Tests

Running all the tests locally takes a very long time, so generally you should run a few tests locally and rely on GitHub CI checks for comprehensive testing. We highly recommend using pytest to run tests selectively. Note that you should use `python -m pytest` rather than calling `pytest` directly to make sure it uses your locally built version of PyTorch.

Most relevant tests are in test/onnx/.

The most used test file is test_pytorch_onnx_onnxruntime.py. The tests in this file generally:

- Define a subclass of `torch.nn.Module` .
- Define some inputs.
- Call `self.run_test()` with the instantiated module and inputs.

`run_test()` converts the module to ONNX and compares the output between PyTorch and ONNX Runtime.

Tests added to `TestONNXRuntime` are automatically defined for all supported opset versions. The class name with no suffix runs tests against opset version 9.

For example:

```
# run test for opset 11
python test/onnx/test_pytorch_onnx_onnxruntime.py
TestONNXRuntime_opset11.test_arithmetic_prim_bool
# run test for opset 9
python test/onnx/test_pytorch_onnx_onnxruntime.py
TestONNXRuntime_opset9.test_arithmetic_prim_bool
```

An example of adding unit tests for a new symbolic function: Add binary_cross_entropy_with_logits op

Links

- User-facing docs.

Relevant parts of PyTorch repo

- User-facing doc: docs/source/onnx.rst
- Python tests: test/onnx/
- More Python tests: test/jit/test_onnx_export.py
- Python code: torch/onnx/
- C++ code: torch/csrc/jit/passes/onnx/

Features

Quantized model export

To support quantized model export, we need to unpack the quantized tensor inputs and the PackedParam weights (https://github.com/pytorch/pytorch/pull/69232). We construct through `TupleConstruct` to have a 1-to-1 input mapping, so that we can use `replaceAllUsesWith` API for its successors. In addition, we support quantized namespace export, and the developers can add more symbolics for quantized operators conveniently in the current framework.