Releasing PyTorch

General Overview

Releasing a new version of PyTorch generally entails 3 major steps:

0. Cutting a release branch preparations
1. Cutting a release branch and making release branch specific changes
2. Drafting RCs (Release Candidates), and merging cherry picks
3. Promoting RCs to stable and performing release day tasks

Cutting a release branch preparations

Following Requirements needs to be met prior to final RC Cut:

- Resolve all outstanding issues in the milestones(for example 1.11.0)before first RC cut is completed. After RC cut is completed following script should be executed from builder repo in order to validate the presence of the fixes in the release branch :

```
python github_analyze.py --repo-path ~/local/pytorch --remote upstream --branch release/1.11 --milestone-id 26 --missing-in-branch
```

- Validate that all new workflows have been created in the PyTorch and domain libraries included in the release. Validate it against all dimensions of release matrix, including operating systems(Linux, MacOS, Windows), Python versions as well as CPU architectures(x86 and arm) and accelerator versions(CUDA, ROCm).
- All the nighly jobs for pytorch and domain libraries should be green. Validate this using following HUD links:

- Pytorch
- TorchVision
- TorchAudio
- TorchText

Cutting release branches

`pytorch/pytorch`

Release branches are typically cut from the branch `viable/strict` as to ensure that tests are passing on the release branch.

There's a convenience script to create release branches from current `viable/strict` (from root `pytorch/pytorch`):

```
DRY_RUN=disabled scripts/release/cut-release-branch.sh
```

This script should create 2 branches:

- `release/{MAJOR}.{MINOR}`
- `orig/release/{MAJOR}.{MINOR}`

`pytorch/builder` / PyTorch domain libraries

Convenience script can also be used domains as well as `pytorch/builder`

> *NOTE: RELEASE_VERSION only needs to be specified if version.txt is not available in root directory*

```
DRY_RUN=disabled GIT_BRANCH_TO_CUT_FROM=main RELEASE_VERSION=1.11
scripts/release/cut-release-branch.sh
```

Making release branch specific changes

These are examples of changes that should be made to release branches so that CI / tooling can function normally on them:

- Update backwards compatibility tests to use RC binaries instead of nightlies
  - Example: https://github.com/pytorch/pytorch/pull/40706
- A release branches should also be created in `pytorch/xla` and `pytorch/builder` repos and pinned in `pytorch/pytorch`
  - Example PR (CircleCI, to be removed): https://github.com/pytorch/pytorch/pull/65433
  - Example PR (GHA): https://github.com/pytorch/pytorch/pull/72739

These are examples of changes that should be made to the *default* branch after a release branch is cut

- Nightly versions should be updated in all version files to the next MINOR release (i.e. 0.9.0 -> 0.10.0) in the default branch:
  - Example: https://github.com/pytorch/pytorch/pull/65435

Getting CI signal on release branches:

Create a PR from `release/{MAJOR}.{MINOR}` to `orig/release/{MAJOR}.{MINOR}` in order to start CI testing for cherry-picks into release branch.

Example:

- https://github.com/pytorch/pytorch/pull/51995

Drafting RCs (Release Candidates)

To draft RCs, a user with the necessary permissions can push a git tag to the main `pytorch/pytorch` git repository.

The git tag for a release candidate must follow the following format:

```
v{MAJOR}.{MINOR}.{PATCH}-rc{RC_NUMBER}
```

An example of this would look like:

```
v1.8.1-rc1
```

Pushing a release candidate should trigger the `binary_builds` workflow within CircleCI using `pytorch/pytorch-probot` 's `trigger-circleci-workflows` functionality.

This trigger functionality is configured here: `pytorch-circleci-labels.yml`

Release Candidate Storage

Release candidates are currently stored in the following places:

- Wheels: https://download.pytorch.org/whl/test/
- Conda: https://anaconda.org/pytorch-test
- Libtorch: https://download.pytorch.org/libtorch/test

Backups are stored in a non-public S3 bucket at `s3://pytorch-backup`

Cherry Picking Fixes

Typically within a release cycle fixes are necessary for regressions, test fixes, etc.

For fixes that are to go into a release after the release branch has been cut we typically employ the use of a cherry pick tracker.

An example of this would look like:

- https://github.com/pytorch/pytorch/issues/51886

Please also make sure to add milestone target to the PR/issue, especially if it needs to be considered for inclusion into the dot release.

NOTE: The cherry pick process is not an invitation to add new features, it is mainly there to fix regressions

Promoting RCs to Stable

Promotion of RCs to stable is done with this script: `pytorch/builder:release/promote.sh`

Users of that script should take care to update the versions necessary for the specific packages you are attempting to promote.

Promotion should occur in two steps:

- Promote S3 artifacts (wheels, libtorch) and Conda packages
- Promote S3 wheels to PyPI

NOTE: The promotion of wheels to PyPI can only be done once so take caution when attempting to promote wheels to PyPI, (see https://github.com/pypa/warehouse/issues/726 for a discussion on potential draft releases within PyPI)

Additonal Steps to prepare for release day

The following should be prepared for the release day

Modify release matrix

Need to modify release matrix for get started page. See following PR as reference.

After modifying published_versions.json you will need to regenerate regenerate the quick-start-module.js file run following command

```
python3 scripts/gen_quick_start_module.py >assets/quick-start-module.js
```

Please note: This PR needs to be merged on the release day and hence it should be absolutely free of any failures. To test this PR, open another test PR but pointing to to the Release candidate location as above Release Candidate Storage

Open Google Colab issue

This is normally done right after the release is completed. We would need to create Google Colab Issue see following PR

Patch Releases

A patch release is a maintenance release of PyTorch that includes fixes for regressions found in a previous minor release. Patch releases typically will bump the `patch` version from semver (i.e. `[major].[minor].[patch]`

Patch Release Criteria

Patch releases should be considered if a regression meets the following criteria:

1. Does the regression break core functionality (stable / beta features) including functionality in first party domain libraries?
    - First party domain libraries:
        - pytorch/vision
        - pytorch/audio
        - pytorch/text
2. Is there not a viable workaround?
    - Can the regression be solved simply or is it not overcomable?

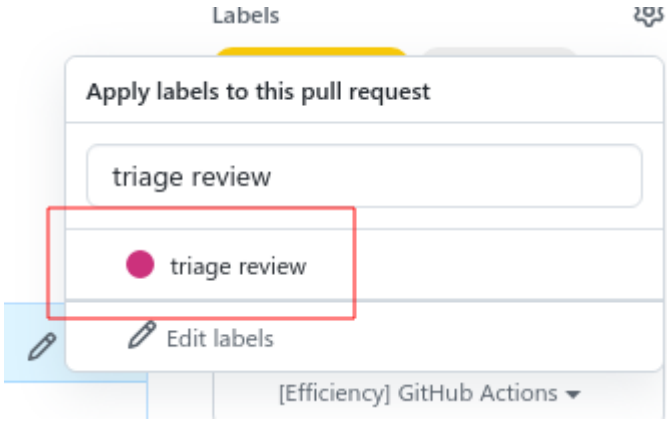
> *NOTE: Patch releases should only be considered when functionality is broken, documentation does not typically fall within this category*

Patch Release Process

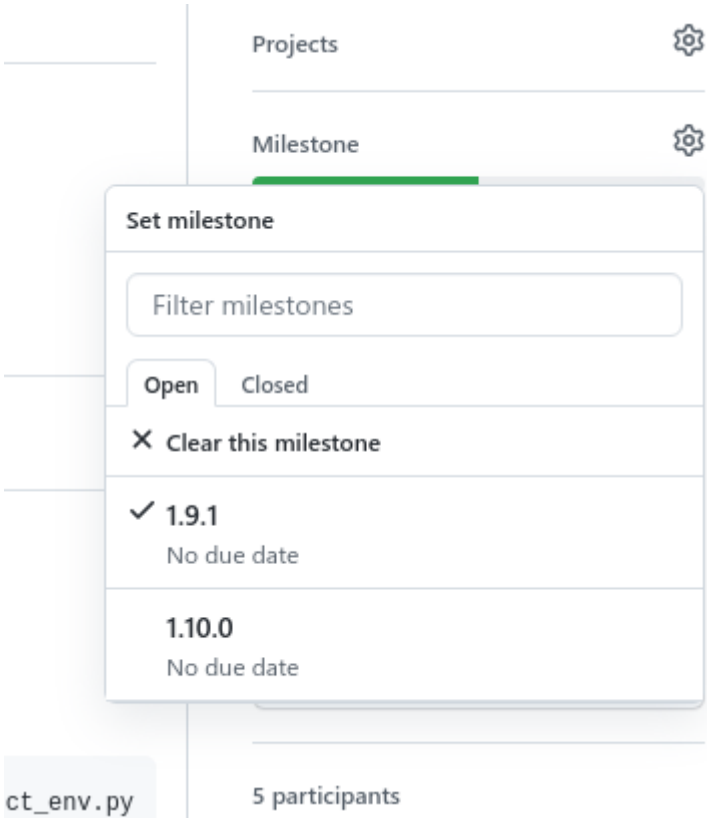
Triage

> *Main POC: Triage Reviewers*

1. Tag issues / pull requests that are candidates for a potential patch release with `triage review`



2. Triage reviewers will then check if the regression / fix identified fits within above mentioned Patch Release Criteria
3. Triage reviewers will then add the issue / pull request to the related milestone (i.e. `1.9.1`) if the regressions if found to be within the Patch Release Criteria



Building a release schedule / cherry picking

> *Main POC: Patch Release Managers*

1. After regressions / fixes have been triaged Patch Release Managers will work together and build /announce a schedule for the patch release
   - *NOTE*: Ideally this should be ~2-3 weeks after a regression has been identified to allow other regressions to be identified
2. Patch Release Managers will work with the authors of the regressions / fixes to cherry pick their change into the related release branch (i.e. `release/1.9` for `1.9.1`)

Building Binaries / Promotion to Stable

> *Main POC: Patch Release managers*

1. Patch Release Managers will follow the process of Drafting RCs (Release Candidates)
2. Patch Release Managers will follow the process of Promoting RCs to Stable

Special Topics

Updating submodules for a release

In the event a submodule cannot be fast forwarded and a patch must be applied we can take two different approaches:

- (preferred) Fork the said repository under the pytorch Github organization, apply the patches we need there, and then switch our submodule to accept our fork.
- Get the dependencies maintainers to support a release branch for us

Editing submodule remotes can be easily done with: (running from the root of the git repository)

```
git config --file=.gitmodules -e
```

An example of this process can be found here:

- https://github.com/pytorch/pytorch/pull/48312