The main goal of this page is to describe how to improve the type annotations in the PyTorch code base, to get to the point where `mypy` can be used to typecheck code that uses PyTorch. The tasks that need tackling are listed, and the workflow for picking up a task is described.

Optional type checking with mypy

Mypy is an optional static typechecker that works with Python 3. To use it, install the following dependencies:

```
# Install dependencies (note: use pip - conda mypy likely won't work, see https://github.co
pip install mypy mypy-extensions

# Run type checker in the pytorch/ directory (after `setup.py develop`)
mypy
```

Note that the minimum version of mypy that is supported is 0.770

What we're aiming for

1. Complete type annotations for the whole code base, and shipping those in a PEP 561 compatible manner - adding a `py.typed` file so the installed package supports typechecking.
2. Inline type annotations for all Python code where possible, *except* if there are too many overloads for functions/methods - in that case a stub file should be preferred (what's too many is a bit of a judgement call, suggested is three per function is a reasonable threshold). Another reason we may have to stay with stub files is if there's a mypy or code limitation (see for example gh-35566 about `nn.Module`).
3. Stub files for the extension modules (e.g. `torch._C`).
4. Good type annotation test coverage, by using `check_untyped_defs=True` for the test suite (or adding type annotations to tests).

`mypy.ini` does not get installed, it's only meant to facility development. Therefore the end state of `mypy.ini` should be minimal and not include any ignores, something like:

```
[mypy]
warn_unused_configs = True
warn_redundant_casts = True
show_error_codes = True
check_untyped_defs = True

files =
    torch,
    test,
    aten/src/ATen/function_wrapper.py
```

Note that adding annotations to `caffe2/` is low-prio (it's a lot of work, and there's not much value in it), we recommend focusing on other tasks.

How to go about improving type annotations

For issues with the current state of type annotations as well as TODO items grouped by module, see issues for `typing`.

The tracking issue for making code using PyTorch type-checkable is https://github.com/pytorch/pytorch/issues/16574.

Setting up and checking mypy works

Before starting, install mypy (0.770 or newer), build PyTorch with `python setup.py develop`, and run `mypy` in the root of the repo. This should give output like:

```
Success: no issues found in 969 source files
```

In `mypy.ini` there's a long list of `ignore_missing_imports` and `ignore_errors` for specific modules or files. If you remove one and re-run `mypy`, then errors should appear. For example, deleting

```
[mypy-torch._C]
ignore_missing_imports = True
```

will show (currently):

```
...
torch/utils/data/_utils/signal_handling.py:39: error: Cannot find implementation or library
Found 14 errors in 14 files (checked 969 source files)
```

Stub files (i.e. `.pyi` files), if they exist for the `.py` files you're working on, need to be moved to inline annotations first. When you move the types inline, that will "turn on type checking"; if the file is not ignored already and running mypy shows new errors, add an `ignore_errors=True` entry for it in `mypy.ini`, and fix the type errors in a *separate* PR.

Note that mypy caching can be flaky, in particular removing `ignore_missing_imports` has a caching bug (https://github.com/python/mypy/issues/7777). If you don't see any errors appear, try `rm -rf .mypy_cache` and try again.

Picking a task

Once the above works, pick a task by assigning yourself to the relevant issue.

Script to give an overview of remaining tasks

```
import os

import torch
```

```
# Assumes an in-place build to find all .pyi files
rootdir = os.path.dirname(os.path.abspath(torch.__file__))

filepaths = []
for subdir, dirs, files in os.walk(rootdir):
    for f in files:
        if f.endswith(".pyi"):
            os.path.join(subdir, f)
            filepaths.append(subdir + os.sep + f)

for filepath in sorted(filepaths):
    print("- [ ] `" + filepath[len(rootdir)-5:] + "`")


with open('mypy.ini', 'r') as f:
    lines = f.readlines()


errors_list = []
imports_list = []

for ix, line in enumerate(lines):
    if line.startswith("ignore_missing_imports"):
        if lines[ix-1].startswith("[mypy-torch."):
            imports_list.append(lines[ix-1][6:-2])
    elif line.startswith("ignore_errors"):
        errors_list.append(lines[ix-1][6:-2])


print("\n\nFiles with ignored errors:\n")
for f in sorted(errors_list):
    print("- [ ] `" + f + "`")


print("\n\nFiles with missing stub files:\n")
for f in sorted(imports_list):
    print("- [ ] `" + f + "`")
```