torch.hub

Pytorch Hub is a pre-trained model repository designed to facilitate research reproducibility.

Publishing models

Pytorch Hub supports publishing pre-trained models(model definitions and pre-trained weights) to a github repository by adding a simple `hubconf.py` file;

`hubconf.py` can have multiple entrypoints. Each entrypoint is defined as a python function (example: a pre-trained model you want to publish).

```
def entrypoint_name(*args, **kwargs):
    # args & kwargs are optional, for models which take positional/keyword arguments.
    ...
```

How to implement an entrypoint?

Here is a code snippet specifies an entrypoint for `resnet18` model if we expand the implementation in `pytorch/vision/hubconf.py`. In most case importing the right function in `hubconf.py` is sufficient. Here we just want to use the expanded version as an example to show how it works. You can see the full script in pytorch/vision repo

```
dependencies = ['torch']
from torchvision.models.resnet import resnet18 as _resnet18

# resnet18 is the name of entrypoint
def resnet18(pretrained=False, **kwargs):
    """ # This docstring shows up in hub.help()
    Resnet18 model
    pretrained (bool): kwargs, load pretrained weights into the model
    """
    # Call the model, load pretrained weights
    model = _resnet18(pretrained=pretrained, **kwargs)
    return model
```

- `dependencies` variable is a **list** of package names required to **load** the model. Note this might be slightly different from dependencies required for training a model.
- `args` and `kwargs` are passed along to the real callable function.
- Docstring of the function works as a help message. It explains what does the model do and what are the allowed positional/keyword arguments. It's highly recommended to add a few examples here.
- Entrypoint function can either return a model(nn.module), or auxiliary tools to make the user workflow smoother, e.g. tokenizers.
- Callables prefixed with underscore are considered as helper functions which won't show up in :func:`torch.hub.list()`.

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 51); *backlink***
  >
  > Unknown interpreted text role "func".

- Pretrained weights can either be stored locally in the github repo, or loadable by :func:`torch.hub.load_state_dict_from_url()`. If less than 2GB, it's recommended to attach it to a project release and use the url from the release. In the example above `torchvision.models.resnet.resnet18` handles `pretrained`, alternatively you can put the following logic in the entrypoint definition.

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 52); *backlink***
  >
  > Unknown interpreted text role "func".

```
if pretrained:
    # For checkpoint saved in local github repo, e.g. <RELATIVE_PATH_TO_CHECKPOINT>=weights/save.pth
    dirname = os.path.dirname(__file__)
    checkpoint = os.path.join(dirname, <RELATIVE_PATH_TO_CHECKPOINT>)
    state_dict = torch.load(checkpoint)
    model.load_state_dict(state_dict)
```

```
# For checkpoint saved elsewhere
checkpoint = 'https://download.pytorch.org/models/resnet18-5c106cde.pth'
model.load_state_dict(torch.hub.load_state_dict_from_url(checkpoint, progress=False))
```

Important Notice

- The published models should be at least in a branch/tag. It can't be a random commit.

Loading models from Hub

Pytorch Hub provides convenient APIs to explore all available models in hub through :func:`torch.hub.list()`, show docstring and examples through :func:`torch.hub.help()` and load the pre-trained models using :func:`torch.hub.load()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 80); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 80); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 80); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 86)

Unknown directive type "automodule".

```
.. automodule:: torch.hub
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 88)

Unknown directive type "autofunction".

```
.. autofunction:: list
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 90)

Unknown directive type "autofunction".

```
.. autofunction:: help
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 92)

Unknown directive type "autofunction".

```
.. autofunction:: load
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 94)

Unknown directive type "autofunction".

```
.. autofunction:: download_url_to_file
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 96)

Unknown directive type "autofunction".

```
.. autofunction:: load_state_dict_from_url
```

Running a loaded model:

Note that `*args` and `**kwargs` in :func:`torch.hub.load()` are used to **instantiate** a model. After you have loaded a model, how can you find out what you can do with the model? A suggested workflow is

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 101); *backlink*

Unknown interpreted text role "func".

- `dir(model)` to see all available methods of the model.
- `help(model.foo)` to check what arguments `model.foo` takes to run

To help users explore without referring to documentation back and forth, we strongly recommend repo owners make function help messages clear and succinct. It's also helpful to include a minimal working example.

Where are my downloaded models saved?

The locations are used in the order of

- Calling `hub.set_dir(<PATH_TO_HUB_DIR>)`
- `$TORCH_HOME/hub`, if environment variable `TORCH_HOME` is set.
- `$XDG_CACHE_HOME/torch/hub`, if environment variable `XDG_CACHE_HOME` is set.
- `~/.cache/torch/hub`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 123)

Unknown directive type "autofunction".

```
.. autofunction:: get_dir
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 125)

Unknown directive type "autofunction".

```
.. autofunction:: set_dir
```

Caching logic

By default, we don't clean up files after loading it. Hub uses the cache by default if it already exists in the directory returned by :func:`~torch.hub.get_dir()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]hub.rst, line 130); *backlink*

Unknown interpreted text role "func".

Users can force a reload by calling `hub.load(..., force_reload=True)`. This will delete the existing github folder and downloaded weights, reinitialize a fresh download. This is useful when updates are published to the same branch, users can keep up with the latest release.

Known limitations:

Torch hub works by importing the package as if it was installed. There are some side effects introduced by importing in Python. For example, you can see new items in Python caches `sys.modules` and `sys.path_importer_cache` which is normal Python behavior. This also means that you may have import errors when importing different models from different repos, if the repos have the same sub-package names (typically, a `model` subpackage). A workaround for these kinds of import errors is to remove the offending sub-package from the `sys.modules` dict; more details can be found in this github issue.

A known limitation that is worth mentioning here: users **CANNOT** load two different branches of the same repo in the **same python**

process. It's just like installing two packages with the same name in Python, which is not good. Cache might join the party and give you surprises if you actually try that. Of course it's totally fine to load them in separate processes.