Review of “Pytorch: An imperative Style, High-Performance Deep Learning Library”

Nghia Dang

1. Paper summary

Before Pytorch was developed in Japan in 2015, deep learning frameworks could only achieve either usability or speed due to the complexity of maintaining both advantages on users. In fact, deep learning engineers had another good choice afterwards since Pytorch could satisfy both: models could be built-up in a Pythonic way, in which the model can connect with other external libraries and it is time-and-energy saving for the easy debugging; also, GPU could be made use in order to reduce time consumption from training models. Again, the users can fully control Pytorch since it is a regular Python program.

2. Introduction and background

With the backup of the vast contributions from open-source ecosystem, programming languages such as Python have received attention from scientists and researchers rather than closed software such as Mathlab. Specifically, Python users can import external packages that were built from the array-based ones such as Numpy, Scipy, Pandas, etc.due to their substantial usefulness in data preprocessing, statistical analysis, plotting, etc. Pytorch, used in Python basement not only took advantage of the forementioned one but also provided array-based programming models supported by GPU with automatic differentiation computing.

3. Highlights

According to this paper [1], the followings can generally summarize its advantages compared with other deep learning software / frameworks:

3.1 Pythonic:

- The user can use Python to run its models while utilizing external packages. For example, users can modify data classes inherited from Pytorch module, choose different optimizers and their parameters as much as they wish, and more. Generally, it’s the users' wish to copy, remove, or modify any Pytorch component in accordance with their wish.

- Pytorch enables users to come into use of their local machines' GPUs thanks to Pytorch's parallel training mechanism in Python, a programming language for general purposes.
- Pytorch can bidirectionally transform its structure with other libraries such as Numpy array. For instance, using command torch.from_numpy(array) or torch.numpy(array), we can easily transform Pytorch tensors to Numpy arrays and vice versa. The good thing of this is that the execution time is insensitive with the magnitude of the array content.

3.2 Automatic differentiation:

The differentiation will be taken care automatically that researchers can spend time on other work

3.3 Multiprocessing:

Pytorch has its own module of multiprocessing that other processors can share the same memory from tensors.

4. Criticism

4.1 Illustrative graphs

Graphs make the paper's points clear to readers. Also, the table comparing throughput among frameworks is easy to follow.

4.2 Speed

Pytorch can be furthermore improved as follows: as Python interpreter is on average slower than other software / programming languages, having it executed outside of Python will enhance its speed.

4.3 Multiprocessing

This can be furthermore improved so that Pytorch can receive more attention from Python researchers.

Reference

[1]A. Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.