Performance Comparison of Pytorch Python and C++ API.

Introduction to High Performance Machine Learning (ECE-GY 9143)

Ghulam Mujtaba

gm2667

Executive summary

- Pytorch is primarily used through its python interface although most of the underlying high-performance code is written in C++.
- A C++ interface called LibTorch for Pytorch is also available that exposes the underlying codebase. There are many cases where a need arises for the use of C++ instead of the primary python API to meet project needs, such as in Low latency operation for robotics, stock trading etc.
- The goal of this project is to compare the performance of the two API and provide a quantitative comparison that can be used to make choices on when and where to use the different API's.

Reasons for using Pytorch with C++

- **Low Latency Systems:** For real time applications such as self-driving cars, game AI and other real-time projects the need lower latency, pure C++ is a much better fit here as compared to the latency limitations of the python interpreter.
- **Highly Multithreaded Environments:** The Global Interpreter Lock (GIL) in python prevents it from running more than one thread at a time. Multiprocessing is available but it is not as scalable and has some shortcomings. C++ is not limited by such constraints so for application where the models require heavy penalization such as Deep Neuroevolution, this can benefit.
- **Existing C++ Codebases:** Many existing projects have existing C++ code bases, and it is more convenient to keep everything in on language as supposed to binding C++ and python. Using C++ throughout the project is a more elegant solution.

Approach

- ResNet18 and ResNet34 image classification models were trained for 50 epochs and benchmarked using the CIFAR dataset.
- Care was taken to ensure that both the python and C++ implementations are as close to each other as possible, e.g. the batch size, the learning rates, number of workers, LibTorch does not have a learning rate scheduler so had to remove that from the python implementation as well etc.
- Added logging to a file capability to log the epoch number, Per epoch time, the training/testing loss and top 1% accuracy.
- Added latency measurement functionality to the python code.
- This was all run in a docker container, and a bash script was written to run the entire experiment with a single command.

Main Results

- The results from the experiment are as following:

Language and Model	Total excecution time (s)	Time per Epoch (s)	Inference Latency image/ms	Training Loss	Training Accuracy %	Test Loss	Test Accuracy %
C++ ResNet18	361.814	7.235071992	3.31	0.0948172	96.684	1.34854	70.44
C++ ResNet34	553.106	11.06131683	4.23	0.105326	96.406	1.25164	72.06
Python ResNet18	1289.118005	25.77832627	2.30788256	0.107002093	96.238	0.449515	88.41
Python ResNet34	2327.167102	45.23919889	3.968456321	0.119861529	95.83	0.4285611	88.1

- This experiment was conducted in a docker container running on an AMD Ryzen 9 5900HS (16 CPUs)~3.3GHz, 16 GB RAM and NVIDIA GeForce RTX 3070 Laptop GPU.

Hyper Parameters	# Epochs	Learning Rate	Momentum	# Workers
SGD optimizer	50	0.1	0.0001	4

Observations

- The LibTorch C++ program has more than **3-4x speedup in training and test time** as compared to the Pytorch implementation. This is possibly because it is compiled beforehand as well as there is no Global Interpreter Lock (GIL) in C++.
- The latency counter intuitively is better with the python code, I am not sure why this is, possibly because latency is measured with a single image, and it only needs 1 thread.
- The Training accuracy for all the models in very similar at around 96% but the test accuracy for the C++ ones is around 71% as opposed to 88% for the Pytorch version. Not sure why this is going on, possible due to non identical data augmentation even though I checked the code for anything like that and could not find it.

Challenges/Conclusion

- A lot of functionality present in Pytorch such as data augmentations implementations, such as flip, random crop, etc., are missing in the LibTorch C++ library. Learning rate scheduler is also not present.
- Programing using the LibTorch API is not straight forward, and the documentation is not as well developed as the Pytorch version.
- It only makes sense to use LibTorch if the compute model being trained is HUGE or GPU budget is very low, this is because the slowness of rapid development and experimentations with C++ as opposed to Pytorch has to be taken into consideration and for ML rapid and flexible experimentation is a key for successful ML projects.

Links and References

- My GitHub link with Code, Documentation, Presentation, Results, etc. https://github.com/GM223/pytorch_python_cpp_benchmark_comparison
- Lei Mao blog and git repository on LibTorch resnet https://leimao.github.io/blog/LibTorch-ResNet-CIFAR/ https://github.com/leimao/LibTorch-ResNet-CIFAR
- Pytorch CIFAR10 https://github.com/kuangliu/pytorch-cifar
- Pytorch NN latency https://deci.ai/blog/measure-inference-time-deep-neural-networks/