CP8321-A6: A Comparison of Deep Learning Libraries for Convolutional Neural Networks

Christopher Indris
Department of Computer Science
Toronto Metropolitan University
Toronto, Canada
cindris@cs.ryerson.ca

***Abstract*—This paper compares three of the leading Python deep learning libraries—PyTorch, TensorFlow, Keras—in the context of convolutional neural networks (CNNs). Areas of assessment include each library's requirements, their strengths and weaknesses, and their user experience during the creation of a CNN to classify MNIST digits.**

***Index Terms*—PyTorch, TensorFlow, Keras, DL (Deep Learning), CNN (Convolutional Neural Network), API (Application Programming Interface), ML (Machine Learning)**

I. Introduction

Machine learning has been a leading source of excitement among the various subfields of technology during the past decade. Researchers and hobbyists regard it as tremendously promising for many disciplines. Prior to the rise of frameworks and APIs that abstract away most of the technical details, the difficulty of building, training and deploying a machine learning model prohibited those without considerable computer science or engineering experience from involvement in the machine learning pipeline. There are now several libraries that can make working with neural networks far more concise than they otherwise would be, though it may not be initially clear which one is the best choice for a particular application or skill set. This paper will attempt to provide some clarifications in this respect.

II. Overview

The key neural network architecture data structures that users of these libraries encounter are layers (for creating the model) and the models themselves [9]. They may provide finer control over the architecture; there is the Keras Functional API for creating models that are more complex than a sequence of layers, and researchers can even subclass existing classes to build entirely new layers and models [9].

PyTorch is a framework that was developed by Meta AI and pushed to open-source in 2017 [1]. The principal functionalities provided by PyTorch are GPU-accelerated tensor computation (`torch`) and deep neural networks (`torch.nn`) with autograd (`torch.autograd`) [13].

TensorFlow is a machine learning platform and community hub to simplify research and the creation of applications. Google Brain launched TensorFlow 1.0 in 2015 [7] with powerful and flexible low-level functionality. The low-level nature of TensorFlow 1.0 meant that it was difficult and tedious to work with, so in 2019 Google released TensorFlow 2.0. It now came paired with the high-level API of Keras so that writing a script using it became more straightforward, and did away with the need for a session environment [14]. TensorFlow now supports ragged tensors (arrays with non-uniform shapes) [16] and maintains its mission to make ML accessible by providing Trax, an end-to-end library focused on speed and clear code [18].

Keras is a high-level neural network API frontend with the aim of being accessible and modular. Created in 2015, it was adopted and integrated into TensorFlow in 2017 [7] and would later become part of TensorFlow 2.0. Currently, TensorFlow is the only backend that Keras supports [8], and Keras will generally be used as part of TensorFlow.

III. Hardware and Software Requirements

PyTorch can be run on the macOS Catalina and Windows 7 operating systems and their successors, as well as Linux-based systems with GNU C Library (glibc) version 2.17 or later (for Ubuntu, this means 13.04 at the earliest). At an early phase of release is the end-to-end workflow for mobile deployment with PyTorch 1.3, so that PyTorch could be used on iOS or Android devices. PyTorch can be used for the C++, Java and Python languages; for Windows, Python 3.7-3.9 are supported and Anaconda environments are highly recommended due to their sandboxed nature (by default, Windows does not come with Python), though pip is a common package manager alternative. PyTorch can also be installed from source code, with a BLAS library (MKL) for optimized mathematic routines in an Anaconda environment. The NVIDIA CUDA toolkit and cuDNN library will be critical for CUDA GPU support for the deep learning code. Binary files on macOS do not support CUDA, so to have CUDA the user should install from source. PyTorch can run on ARM-based Macs [12].

TensorFlow can be used by machines operating with macOS Sierra, Windows 7 and Ubuntu 16.04 and their successors. It is available for Python 3.7 through to 3.10. For GPU support (NVIDIA CUDA architecture), the NVIDIA CUDA toolkit, cuDNN SDK and appropriate drivers are needed, with TensorRT as a useful optional extra for better latency and throughput for ML inference. On macOS, GPUs are not supported by default, but a Miniconda environment is a recommended workaround. On Windows, the last version supporting a GPU is TensorFlow 2.10; for later versions, the user is advised to use WSL2 (Windows Subsystem for Linux). For macOS devices with an M1 chip, the `tensorflow-metal` plug-in is used, though this version cannot handle working with multiple GPUs [15]. Most users will use Keras with TensorFlow (`tensorflow.keras`), and so the hardware and software requirements follow from TensorFlow.

IV. Installation Process and Related Requirements

To install PyTorch, simplest method is typically to run the `pip install torch` command. PyTorch can also be built from source code, or run with a pre-built Docker image. The image provides virtualization in the form of a container. Arguably the easiest method of all is to use Google Colaboratory, which runs in a browser and comes pre-installed with PyTorch. The user can then run `import torch` in their Python script or console. PyTorch lists NumPy and Cmake as dependencies; the former since tensors and NumPy ndarrays are closely related, the latter since PyTorch was written mainly in C++. [13]

TensorFlow is also pre-installed on Google Colaboratory, can be installed from source code on Linux and macOS, and can be installed on the command line using `pip install tensorflow` for the current stable CPU/GPU version or `tf-nightly` for the potentially unstable preview build. In my Anaconda environment, some key dependencies are recognizable: NumPy, for arrays; TensorBoard for visualization; Keras, the API; Bazel, for building (needed when TensorFlow is being installed from source code). [15] I was able to view these dependencies using the `pipdeptree` command line utility.

Keras depends on Python's suite of scientific tools (NumPy and SciPy), BLAS for the CPU, and the CUDA drivers and cuDNN so that the GPU can manage DNN code [2] [3].

V. Strengths and Shortcomings

The fast tape system for Autograd is a selling point for PyTorch. When a neural network is built in PyTorch, a dynamic graph is built. When the graph (the network behaviour) is changed, PyTorch can quickly adapt by using the tape recording and reverse-mode Autodiff so that there is no need to start the graph from scratch [13]. It is open-source with a large community. For shortcomings, the included visualization features are not as sophisticated as those in TensorFlow [7] and an API server is needed when putting a program into production [19].

TensorFlow visualizes training well with TensorBoard and is production-ready. Again, there is a large open-source community. Unfortunately, the graphs are static and the debugging methods are not as good [19]; together, these make it harder to implement quick fixes to the users' programs [13].

Keras supports multiple GPUs [5]. The combination of modularity, available pre-trained models and fast deployment make it user-friendly. However, it is slow compared to the backend and working with the low-level TensorFlow functionality may lead to errors that Keras does not catch [4].

VI. My Implementations of a CNN

To experience the different platforms, I opted for the purpose of this paper to see about implementing a CNN using the different libraries. Though I made separate Keras and TensorFlow files, I learned while doing this assignment that Keras requires the TensorFlow backend and TensorFlow has fully adopted Keras as a frontend; hence, the architecture and training for these two are virtually identical. This exercise was primarily to assess the user experience. As I used Google Colaboratory, I could bypass almost all of the installation steps. Both PyTorch and Keras include the MNIST dataset, and so these can be imported using `torchvision.datasets.MNIST()` and `tfds.load()` respectively, where `torchvision` is a PyTorch-related library for computer vision work (also pre-installed in Colaboratory) and `tfds` (TensorFlow Data Sets) is a TensorFlow-related library for data loading, which can easily be installed using `!pip install tfds-nightly`. For all libraries, it was easy to plot a few example images. In Keras/TensorFlow, building the model architecture could be done by adding instances of the `Layers` module to an instance of the `Sequential` class. In PyTorch, I created a class inheriting from the `torch.nn.Module` class to create a neural network. The class's __init__ method was where variables were assigned (using the `Conv2D`, `Dropout2d` and `Linear` classes for representing the layers), and the `forward()` method specified the activation functions (from `torch.nn.Functional`). Though Keras required that the input shape of the data be specified, the `forward()` function made the building of the architecture in PyTorch a bit more complicated (specifying activation functions in Keras amounts to simply passing a string argument into each layer). Training in Keras is incredibly easy; the `fit()` function takes the training and test sets, batch size and epoch count as inputs and does the rest, while returning a simple callback (a brief history of the loss over the epochs, for graphing purposes). For both TensorFlow/Keras and PyTorch, I was able to get and plot the learned weights. I was also able to see a particular case and make a prediction on it (the `predict()` method from Keras is used for this), although since this involves selecting a case from the datasets I suspect that the complexity in doing this varies more based on the method for data loading (various methods exist for all libraries) than the libraries themselves.

In terms of performance, the PyTorch model attained a test accuracy of 94% after one epoch and 97% after three epochs, whereas the Keras/TensorFlow files attained 70% and 72% after one epoch, 94% after three epochs and 96% after six epochs. This difference between the Keras and TensorFlow files seemed to decrease as the epochs progressed; this seems reasonable, as the only substantive difference between them would be the random initialization of weights. The PyTorch model may differ from the other two as a result of slightly different versions of losses and metrics. Existing studies have shown that there is no significant difference in the system performance during training between the libraries [6] [11].

TABLE I
SIMILARITIES AND DIFFERENCES

Library	Open Source	Platform	CUDA Support	Autodiff	Speed	API level	Debugging	Visualization
PyTorch	Yes	Linux/macOS/Windows	Yes	Yes	Fast	Low	Easy	Limited
TensorFlow	Yes	Linux/macOS/Windows	Yes	Yes	Fast	Both	Hard	Good
Keras	Yes	Linux/macOS/Windows	Yes	Yes	Slow	High	Easy	Good

My implementations can be found in the Python notebooks at https://github.com/chrisindris/CP8321A6.

VII. SOFTWARE OF CHOICE

In general, TensorFlow (with Keras) is easier to use, while PyTorch is more complex but provides the programmer with a considerable amount of freedom and good debugging capabilities. For Assignment 5, my group and I put together a visualizer for the backpropagation of a CNN being trained on the MNIST dataset. We chose PyTorch for our implementation because the functionality we wanted for our program—the GUI updates in (just about) real-time as the network is training in a worker thread—seemed to be easier to do in PyTorch. In PyTorch, the `backward()` function can be called explicitly, giving us a good indication of where the backpropagation was happening in the code. In TensorFlow, the updating weights could be recorded using callback functionality (as the model is being fit, the process can be written to a "history" variable). This may have required custom callback code to save the weights, and it did not seem like the ideal interface for writing to the GUI in real time. Though TensorBoard makes TensorFlow's visualizations better than those in PyTorch, our project used `matplotlib` for visualization and only required the weight matrices from the model, so the libraries's built-in visualization features were not of utmost importance to us. [7] Perhaps this leads us to the main takeaway, which is that a user should choose between PyTorch and Keras/TensorFlow for a project based on which aspects of a project are key and which library more effectively provides support for those associated tasks.

REFERENCES

[1] A. Paszke, S. Gross, F. Massa, G. Chanan, J. Bradbury, A. Lerer, S. Chintala, J. Bai, L. Fang, B. Steiner, S. Chilamkurthy, A. Tejani, E. Y. Zach DeVito, A. Köpf, A. Desmaison, L. Antiga, N. Gimelshein, Z. Lin, and T. Killeen, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," NeurIPS, vol. 32, 2019.

[2] A. Sankaran, N. A. Alashti, C. Psarras, and P. Bientinesi, "Benchmarking the linear algebra awareness of tensorflow and pytorch," 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Feb. 2022.

[3] Chollet François, "Appendix A," in Deep Learning with Python, Shelter Island, NY: Manning Publications Co., 2018, pp. 340–345.

[4] D. F. Team, "Python keras advantages and limitations," DataFlair, 25-Aug-2021. [Online]. Available: https://data-flair.training/blogs/python-keras-advantages-and-limitations/. [Accessed: 15-Nov-2022].

[5] F. Chollet, "Keras Documentation: Multi-gpu and distributed training," Keras, 28-Apr-2020. [Online]. Available: https://keras.io/guides/distributed_training/. [Accessed: 17-Nov-2022].

[6] H. Dai, X. Peng, X. Shi, L. He, Q. Xiong, and H. Jin, "Reveal training performance mystery between tensorflow and Pytorch in the single GPU environment," Science China Information Sciences, vol. 65, no. 1, Dec. 2021.

[7] J. Terra, "Pytorch vs tensorflow vs keras: Here are the difference you should know," Simplilearn.com, 22-Sep-2022. [Online]. Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article. [Accessed: 15-Nov-2022].

[8] K. Team, "Keras Documentation: Keras FAQ," Keras. [Online]. Available: https://keras.io/getting_started/faq/. [Accessed: 17-Nov-2022].

[9] Keras-Team, "Keras-Team/Keras: Deep Learning for Humans," GitHub, 2015. [Online]. Available: https://github.com/keras-team/keras. [Accessed: 15-Nov-2022].

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16), pp. 265–283, 2016.

[11] M. C. Chirodea, O. C. Novac, C. M. Novac, N. Bizon, M. Oproescu, and C. E. Gordan, "Comparison of tensorflow and pytorch in convolutional neural network - based applications," 2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2021.

[12] PyTorch, "Pytorch," PyTorch. [Online]. Available: https://pytorch.org/get-started/locally/. [Accessed: 15-Nov-2022].

[13] Pytorch, "Pytorch/Pytorch: Tensors and dynamic neural networks in python with strong GPU acceleration," GitHub, 2017. [Online]. Available: https://github.com/pytorch/pytorch. [Accessed: 15-Nov-2022].

[14] S. Gupta, "Tensorflow 1.0 vs. tensorflow 2.0: What's the difference?," Springboard Blog, 20-Jul-2022. [Online]. Available: https://www.springboard.com/blog/data-science/tensorflow-1-0-vs-tensorflow-2-0/. [Accessed: 15-Nov-2022].

[15] TensorFlow, "Install tensorflow 2," TensorFlow. [Online]. Available: https://www.tensorflow.org/install. [Accessed: 15-Nov-2022].

[16] TensorFlow, "Ragged tensors, Tensorflow Core," TensorFlow. [Online]. Available: https://www.tensorflow.org/guide/ragged_tensor. [Accessed: 15-Nov-2022].

[17] Tensorflow, "Tensorflow/tensorflow: An open source machine learning framework for everyone," GitHub, 2015. [Online]. Available: https://github.com/tensorflow/tensorflow. [Accessed: 15-Nov-2022].

[18] TensorFlow, "Trax Quick Intro," Google Colab. [Online]. Available: https://colab.research.google.com/github/google/trax/blob/master/trax/intro.ipynb. [Accessed: 15-Nov-2022].

[19] V. Kurama, "Pytorch vs. tensorflow: Which framework is best for your deep learning project?," Built In, 2022. [Online]. Available: https://builtin.com/data-science/pytorch-vs-tensorflow. [Accessed: 15-Nov-2022].