PyTorch vs TensorFlow - The State of Art in Machine Learning Frameworks
CSE 598 Data Intensive Systems for Machine Learning, Spring 2020

Parth Doshi
Arizona State University
Tempe, AZ
pdoshi4@asu.edu

Abstract

This paper deals with the comparison of the two most popular Machine Learning frameworks - PyTorch and TensorFlow, used in the domain of Deep Learning. The paper highlights the similarities as well as the key differences between these frameworks while establishing the fact that both these frameworks are widely adopted, working extremely well with the different set of problems in industry, academia and research.

1. Motivation

The advent of deep learning in the early 2010s gave rise to the development of many machine learning frameworks, vying to become the new favorite amongst researchers and industry folks. We have seen the development of early academic frameworks like Caffe and Theano to the massive industry-backed PyTorch and TensorFlow. It is difficult to keep track of the most frequently used, popular frameworks due to the deluge of the options available in this previously untapped domain. [5]

In 2020, we see that in this battle for Machine Learning framework supremacy, we have two main contenders namely, PyTorch designed by Facebook [10] and TensorFlow developed by Google [1]. Using the number of mentions in published research papers at top research conferences as metric, we try to understand the sentiment of researchers and industry practitioners alike1. The recent trend suggests that PyTorch is gaining popularity in the research while TensorFlow still retains the backing of the industry at the moment. [5] In order to understand this sentiment, it is important to examine the two frameworks to understand their strengths and weaknesses while also exploring their similarities and dissimilarities.

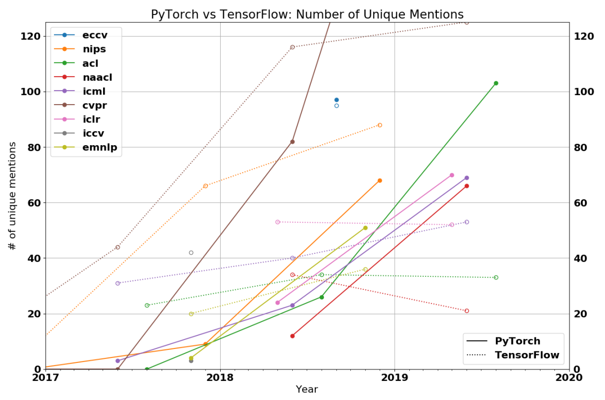


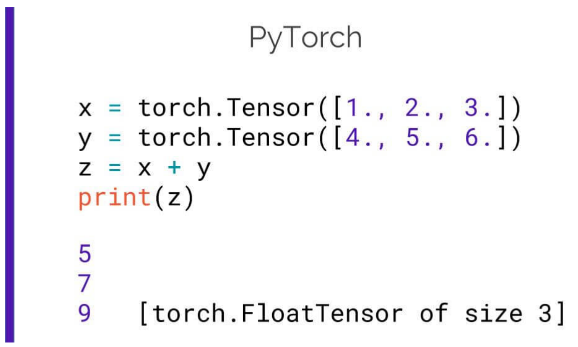
Figure (1) Count of papers that use either Tensorflow or PyTorch at each of the top research conferences over time [5]

2. PyTorch

Often usability and speed are two main factors for the adaptability for a Deep learning frameworks. However most of them don't seem to have both of them in unison. PyTorch aims to show that these two goals are in fact compatible. It is a machine learning library providing an imperative and Pythonic programming style to support code as a model. It helps make debugging easy while being consistent with other popular scientific computing libraries, thereby retaining its support of efficient and supporting hardware accelerators such as GPUs. [10]. After its introduction in the late 2017, its popularity has been steadily growing, with its dynamic computation graphs helping researchers process variable length inputs and outputs, a trait particularly useful with RNNs.

2.1. Strengths [4] [9]

- Python-like coding. 2
- Dynamic graph.
- Easy & quick editing to write your own layer types and



```
x = torch.Tensor([1., 2., 3.])
y = torch.Tensor([4., 5., 6.])
z = x + y
print(z)

5
7
9   [torch.FloatTensor of size 3]
```

Figure (2) PyTorch Sample Code [7]

run on GPU.

- Good documentation and community support.
- Available pre-trained models.
- Open source.
- Modular pieces that are easy to combine.
- Plenty of projects out there using PyTorch.

2.2. Weaknesses [4] [9]

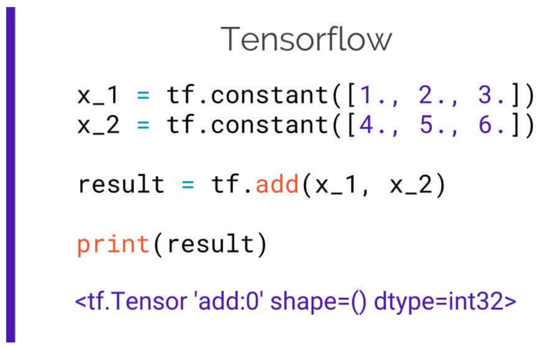
- Third-party needed for visualization.
- You need to write your own training code.
- No commercial support.
- API server needed for production.

3. TensorFlow

Scalability and the ability to operate in heterogeneous environments is what makes TensorFlow a prominent player in the industry setting. Designed by Google to replace Theano, TensorFlow is a machine learning system that uses data-flow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a data-flow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). [1]

3.1. Strengths [4] [9]

- Simple built-in high-level API. 3
- Data and model parallelism.
- Python + Numpy.
- Computational Graph Abstraction.



```
x_1 = tf.constant([1., 2., 3.])
x_2 = tf.constant([4., 5., 6.])

result = tf.add(x_1, x_2)

print(result)

<tf.Tensor 'add:0' shape=() dtype=int32>
```

Figure (3) TensorFlow Sample Code [7]

- Tensorboard visualizing training.
- Production-ready thanks to TensorFlow serving.
- Easy mobile support.
- Open source.
- Good documentation and community support.

3.2. Weaknesses [4] [9]

- Dynamic typing is error-prone.
- Computation Graph is pure Python, hence slow than other frameworks.
- Static Graph.
- Difficult debugging method.
- Not so easy to make quick changes.

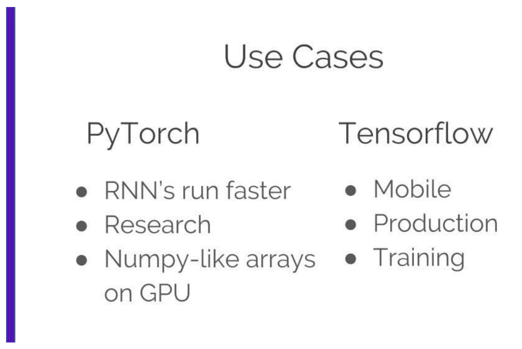
4. Common Features - PyTorch & TensorFlow [10] [1] [8]

- Open Source.
- Graph based Python API Frameworks.
- Layered based Architectures with distributed execution.
- Auto-differentiation frameworks.
- Data-flow and control-flow graphs.
- Efficient C++ Core Kernel.
- Great documentation and community support.
- Debugging functionality and device management ability.
- CUDA GPU supported libraries with serialization support.

- Simplified model saving and loading in PyTorch API and TensorFlow Saver Object.
- Custom Extensions written in C, C++ or CUDA available.

5. Main Differences - Pytorch v/s TensorFlow [8] [6]

- Tensorflow has a steep learning curve compared to PyTorch. PyTorch is more pythonic while for using Tensorflow, one will have to learn a bit more about it's working (sessions, placeholders etc.). [6]
- Tensorflow is a static computational graph framework while PyTorch relies on a dynamic computational graph methodology. Due to this, the entire computation graph of the model needs to be defined in TensorFlow in order to run the ML model. However, one can define/manipulate the graph on-the-go in PyTorch which is extremely useful when dealing variable length inputs in RNNs. [6]
- TensorBoard is a brilliant tool that enables visualizing ML models. PyTorch doesn't have such a tool, although there are libraries like matplotlib which can be used for similar purposes. [6]
- Tensorflow is well equipped for production models and scalability due to its production ready build. On the other hand, with its ease of learning and lightness to work with, PyTorch is well adopted in research and academia to build rapid prototypes. [6]
- In TensorFlow serialization, the entire graph can be saved as a protocol buffer, including parameters as well as operations. This enables the graph to be then loaded in other supported languages (C++, Java). Helpful in heterogeneous environments where Python is not the preferred option. [8]
- Data loading APIs are well designed in PyTorch with interfaces specified in dataset, a sampler and a data loader. This leads to ease of data parallelism. However, in order to achieve similar results in TensorFlow, we have to add all the preprocessing code you want to run in parallel into the TensorFlow graph which can be difficult to learn and much more verbose (e.g. computing a spectrogram). [8]
- Due to the dynamic computational process, PyTorch offers easy debugging process by offering Python debugging tools like pdb or ipdb, etc. In TensorFlow, you have to use the TensorFlow debugger tfdbg to view the internal structure and states of running TensorFlow graphs which can be a difficult proposition during training and inference. [2]



Use Cases

PyTorch
- RNN's run faster
- Research
- Numpy-like arrays on GPU

Tensorflow
- Mobile
- Production
- Training

Figure (4) PyTorch vs TensorFlow use cases [7]

- As Tensorflow maps nearly all of the GPU memory of all GPUs visible to the process, it results in fair device management. Whereas, PyTorch performs custom allocation of tensors and keeps track of the currently selected GPU and all the CUDA tensors which will be allocated. [2]

6. Key Takeaways

Understanding the major strengths and weaknesses of the two most prominent ML frameworks was the key takeaway from the papers. Also the shared similarities between TensorFlow and PyTorch help us understand the key factors and design principles of a Machine Learning framework. Recently, PyTorch introduced the JIT compiler and "TorchScript", introducing graph based features while TensorFlow also introduced TensorFlow Eager by default in 2.0.

As we review TensorFlow, we see a widely used deep learning library with strong visualization tools. Due to its production-ready deployment options and support for mobile platforms,[3] TensorFlow is a good option 4 if you:

- Develop production models for heterogeneous environments,
- Develop models for mobile platform deployment,
- Would like to experience good community support
- Like thorough documentation,
- Enjoy rich learning resources and tutorials,
- Have visualization requirements - Tensorboard
- Need to use large-scale distributed model training.

PyTorch is still a young framework which is featuring prominently in research and academic setting. [3] You may find it a good fit 4 if you:

- Perform research,
- Don't have demanding non-functional requirements,

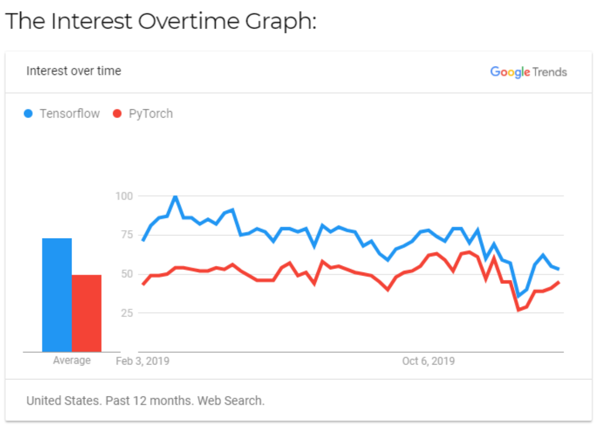


Figure (5) Google Trends for ML frameworks - Past 12 months. [2]

- Would like to experience better development and debugging,
- Love all things in the Pythonic way.

To sum it up, we see PyTorch currently being the #1 choice in research while TensorFlow is still preferred in the industry setting. With the release of PyTorch JIT and TensorFlow Eager, we look at the following questions in the near future [5]:

- Does research preference affect industry output?
- With the release of TensorFlow's Eager, will it finally be able to catch up PyTorch in terms of usability?
- Can PyTorch be production ready in heterogeneous environments and how quickly?

Machine Learning is a constantly evolving field and we need to see how these frameworks evolve in order to keep up with the evolving use cases and requirements.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 1, 2

[2] A. Choudhury. Tensorflow vs pytorch: Top 10 differences between the two ml libraries, 2019. 3, 4

[3] K. Dubovikov. Pytorch vs tensorflow — spotting the difference, 2017. 3

[4] A. Hannun. Pytorch vs. tensorflow: Which framework is best for your deep learning project?, 2019. 1, 2

[5] H. He. The state of machine learning frameworks in 2019, 2019. 1, 4

[6] Y. Jain. Tensorflow or pytorch : The force is strong with which one?, 2019. 3

[7] S. Kim. Exploring the deep learning framework pytorch, 2018. 2, 3

[8] V. Kurama. Pytorch or tensorflow?, 2017. 2, 3

[9] C. Nicholson. Comparison of ai frameworks, 2019. 1, 2

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 1, 2