PyTorch vs Keras

Originally published at https://www.iflexion.com/blog/pytorch-vs-keras

Published: October 8, 2020 | By Martin Anderson

Over the last five years, the uptake and industry interest in two of the libraries in our recent machine learning frameworks overview suggests that they merit a more detailed comparison to each other: Facebook's PyTorch and the French-originated open-source library Keras, now backed by Google[1].

Although there is a great deal of ongoing absorption and consolidation in the machine learning research space, with frameworks rising, falling, merging and being usurped, the PyTorch vs Keras comparison is an interesting study for AI developers, in that it in fact represents the growing contention between TensorFlow and PyTorch — the former, with greater industry support in terms of links to manufacturing and deployment of both abstract systems and single-purpose products (such as CUDA-based GPU acceleration in the industrial and commercial spheres); the latter, more self-integrated, arguably easier to develop with, and to a certain extent better documented and less encumbered by technical debt.

Nonetheless, here we will consider the Keras API rather than TensorFlow itself, since Keras has evolved in recent years from its early ideals as a multi-backend API[2] into a committed stance as the 'accessible face' of TensorFlow, whereas PyTorch was conceived to provide both low-level computational resources and a relatively high level of user accessibility in a single product.

The PyTorch vs Keras comparison is an interesting study for AI developers, in that it in fact represents the growing contention between TensorFlow and PyTorch.

PyTorch

Written in Python, the PyTorch project is an evolution of Torch, a C-based tensor library with a Lua wrapper. Facebook's 2017 release of PyTorch brought GPU acceleration, the implementation of Chainer's ability to modify a neural network on the fly. 2018 heralded the incorporation of Caffe2[3], which at that time was a strong contender for market share against TensorFlow, especially in the field of computer vision, as we have shown in our review of Caffe vs TensorFlow.

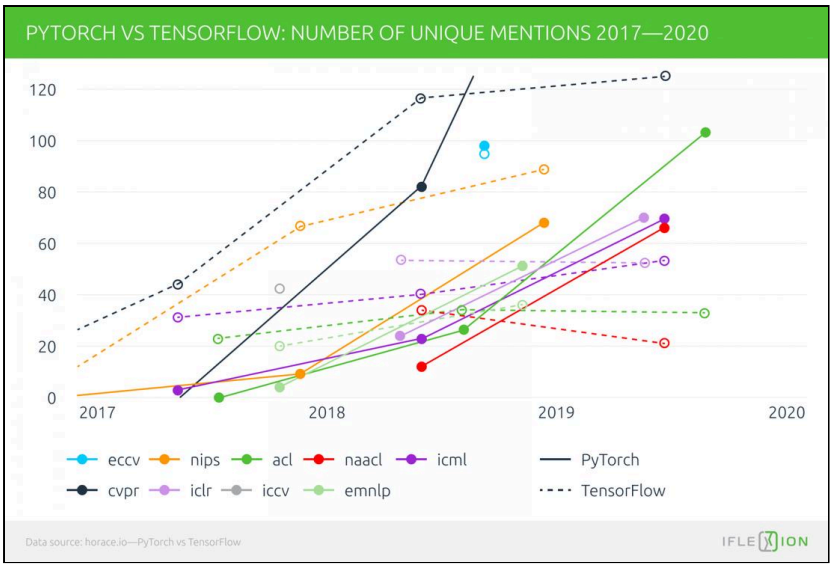
Capabilities

Featuring scalable distributed training[4], automated neural network optimization[5] and ad-hoc gradient generation via AutoGrad[6], PyTorch is also imperative[7], allowing operations to query code in the absence of a full model build, facilitating experimentation and rapid prototyping[8].

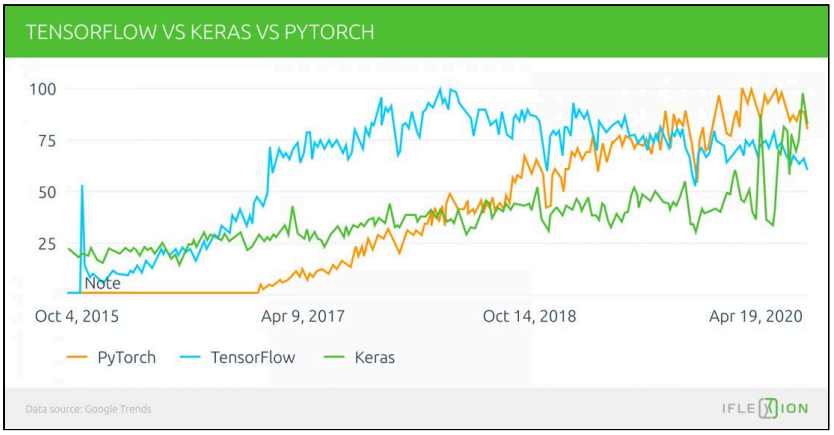
PyTorch has proved friendly to developers migrating from NumPy[9], as it offers a practically identical set of abstractions combined with GPU acceleration and a growing wealth of third-party extensions, ancillary systems and integrations (including NumPy itself)[10].

Reach

Although one 2020 study of keywords for TensorFlow and PyTorch across four major job listings sites found that PyTorch is still mentioned only half as often as TensorFlow[11], this differential is dropping rapidly year on year. Additionally, a study of citations of PyTorch in academic papers[12] reports a new insurgence of PyTorch mentions, with the Facebook-backed framework commanding a comfortable majority in 2019:



A look at Google Trends for TensorFlow, PyTorch, and Keras over the last five years reaffirms the rise of interest in PyTorch relative to Keras/TensorFlow:



However, TensorFlow is the incumbent, if not already the industry standard: it currently has 82,000 forks on GitHub[13], with 11,000 for PyTorch[14] and 18,600 for Keras[15].

Google's recently announced improved cloud support for PyTorch[16] adds to the library's growing accessibility for large-scale implementations, while PyTorch Mobile addresses the long-standing advantage that TensorFlow Lite (among other resource-limited TensorFlow device solutions) has held over it.

Tesla is probably the most famous of current PyTorch industry adherents[17].

Interest in PyTorch is rising. Here are some more reasons to consider the framework for your machine learning project.

Keras and tf.keras

Keras, originated by AI researcher François Chollet[18] in 2015, shortly before he joined Google as a deep learning researcher and engineer, was originally intended as a 'human-friendly' high-level API for complex but powerful machine learning frameworks.

Written in Python and available under an MIT license[19] with Python, R and now JavaScript[20] interfaces, Keras supported a variety of machine learning frameworks including the now-discontinued Theano[21] as well as PlaidML, a cross-platform Tensor compiler noted for opening up machine learning systems to non-NVIDIA GPUs.

However, as of June 2020, the latest version of Keras (2.3.0) has been announced as the last that would support secondary platforms beyond TensorFlow[22], with users (including PlaidML users) advised to switch their code to the integrated tf.keras module in TensorFlow.

Capabilities

Keras supports recurrent and convolutional neural networks. It provides a highly abstracted API for the low-level functionality of TensorFlow, with the capability to create six types of core layers: input object, dense layer, activation layer, embedding layer, masking layer, and lambda layer[23]. Further functionality is available with TensorFlow add-ons[24].

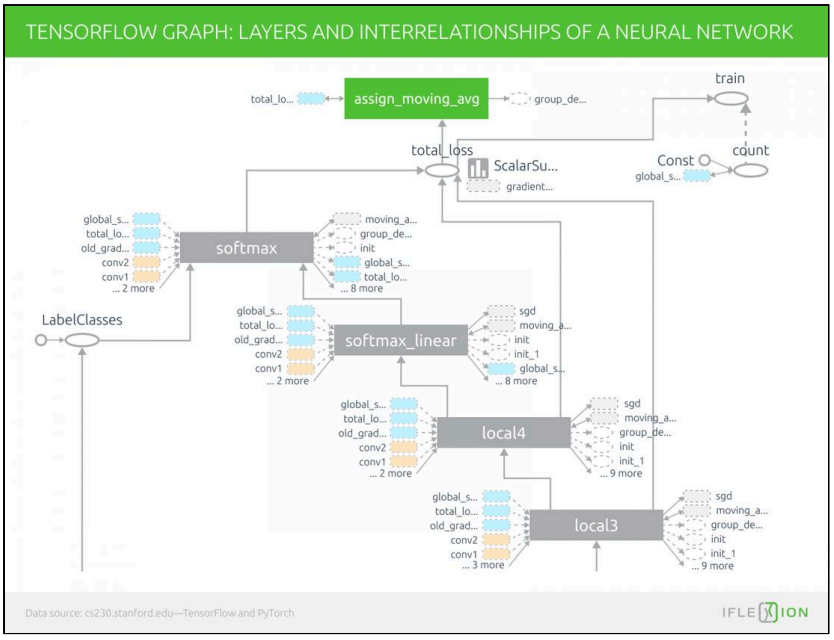
Keras' recent commitment to TensorFlow as of version 2.0 has also enabled resource optimization via mixed precision training on the NVIDIA GPU platform[25]. In addition, the AutoKeras package redresses some long-term concerns about matching PyTorch's easy prototyping capabilities, as well as offering an open source alternative to Google's commercial AutoML offering.

Reach

As of early 2020, Keras claims a user base of 375,000 individual end users[26], as well as adoption by major names such as Netflix, Yelp, and Instacart. It further claims the number one position for mentions in scientific papers by large research bodies such as NASA and CERN (as opposed to the 'general' academic papers cited by PyTorch — see above).

PyTorch vs Keras: Static/Dynamic Graphs

Until the advent of TensorFlow 2.0, one of the main considerations with Keras was its use of static rather than dynamic graphs. All deep learning frameworks will assemble and run neural networks to a 'master mapping', or a computational graph. Variables such as loss functions, biases, and weight assignment and configuration are all allocated and configured by the graph:



Keras originally generated *static* graphs, wherein all the variables are assigned and locked at runtime.

On the plus side, this makes the model very efficient, and works well where the variability of the data and the parameters of its journey are relatively predictable.

Negatively, the neural network will inevitably be less flexible and less able to revise its goals mid-journey, which can be a deciding factor between productive and unproductive convergence (which we covered before in the overview of machine learning challenges).

PyTorch has always used a dynamic graph, which allows the variables to be reevaluated and changed according to ongoing factors in model training, albeit at the expense of additional computation.

Dynamic Graphs in tf.keras

However, since TensorFlow 2.0 added support for dynamic graphs, and since tf.keras represents a low-level integration of the Keras framework into TensorFlow, this issue is likely only to crop up when importing legacy structures into a deep learning pipeline. Even then, provision has been made[27] to accommodate dynamic graphs while retaining stability and functionality.

The use of machine learning for natural language processing (NLP) gives one example of why dynamic graphs are useful, since the length of input data (sentences, phrases, etc.) may require more space downstream than can be anticipated in a static graph, hindering the model's progress in cases where such variables must be pre-determined.

In principle, this can apply to most types of data in a machine learning model. In fact, the inventor of Keras has concededthat NLP is the prime driver for dynamic graphs[28].

It should also be considered that dynamic graphs can make a neural network easier to debug via static analysis or other means[29]. Here the transparency of PyTorch's native Python environment could give an advantage, though the addition of the NumPy-like Eager Execution library to TensorFlow v1.7 in 2017[30] has sought to redress this shortcoming and bring a more Pythonic approach to dynamic, state-defined execution. PyTorch also has an 'Eager Mode'[31].

Accessibility and Debugging

Keras has arguably gained parity with PyTorch[32] since its integration as the official high-level TensorFlow API, not least because its slightly easier learning curve[33] is no longer undermined by a dependence on static graphs, or some of its former limitations in accessing and exploiting GPU resources (such as poor support for distributed computing over multiple GPUs[34]), and it can now easily implement automatic differentiation[35] and model and layer sub-classing[36].

With greater functional similarity between PyTorch and Keras, the superior abstraction of the Keras API is arguably one compelling reason to consider it as the central framework for a machine learning project — as long as one considers that where problems inevitably do emerge, bug fixes and up-to-date help resources can sometimes be harder to obtain than with PyTorch (see below).

However, this applies only if the proposed architecture is reasonably complex: it has also been argued[37] that in terms of API nomenclature, conventions and general usability when defining and instantiating models, there is by now little practical difference in user experience between the two frameworks.

When the time comes to debug, the C++ roots of Keras begin to show. Not only is the underlying code more difficult to navigate than in PyTorch, but it can be harder to identify the point in the dependency chain at which the code is causing problems.

While PyTorch can seem more arcane than Keras at first glance, depending on the complexity of the project, there are a number of high-level APIs that can provide a similar quality of abstraction and facility to Keras, where needed. These include fast.ai, Flare, and Ignite.

The superior abstraction of the Keras API is arguably one compelling reason to consider it as the central framework for a machine learning project.

Speed, Configuration and Ease of Deployment

Since the move to dynamic graphs for Keras under TensorFlow 2.0, both the functionality and performance of PyTorch and Keras (over TensorFlow 2.0+) have converged to what is now often considered a non-critical, even trivial level of difference[38]. The comparison between the systems has arguably evolved from *Coke vs Coffee* to *Coke vs Pepsi*, with user preference and technical debt among the less compelling considerations that might tip the balance.

In terms of speed, both frameworks are similarly outfitted and dependent on Python interpreters; both use some form of asynchronous execution[39] to queue jobs into CUDA to avoid a multitude of time-consuming read/write operations; both are now equally capable of distributed workloads over multiple GPUs, as well as resource-saving mixed precision computation[40]; and both have a workable foothold in the mobile space.

For multi-agent systems, where multiple networks need to collaborate and/or draw live data from external sources, the modularity and extensibility of PyTorch can prove an advantage, whereas Keras can perhaps more easily deploy a more 'templated' implementation of a machine learning task (i.e. classification, image recognition, segmentation, and NLP).

Though Keras arguably retains a more mature ecostructure of packages to speed deployment times, the very popular Flask can be used with both Keras[41] and PyTorch[42]. Additionally, Amazon Web Services (AWS) offers the TorchServe architecture for PyTorch, reducing the need for custom code in PyTorch model deployments[43].

Version Management, Bug-Fixes, and Documentation

In contrast to PyTorch, Keras has undergone so many transformations in intent, outlook, and execution since its inception that versions, bug-fixes, and accuracy of available documentation have become interrelated issues over the course of time. Keras' roots in pure machine learning research have left its online resources in a state that isn't necessarily congruent with its subsequent success in commercial deployments or its general industry reach.

Though the project has a well-updated learning wiki[44], Keras-related documentation can prove lacking in practical solutions for some common problems, or an adequate number of code examples to cope with the popularity of the framework, or the breadth and scope of developers' issues.

Due to the specious nature of the average machine learning project and the need for its developers to maintain 'real-time' group hubs to address new issues and respond to bug reports, some of the best Keras developer information is found in discrete communities on Discord or other gated channels, hidden from search indexes, obscured by 'infinite scroll' chat interfaces, and missing from the static documentation that the developers do not have the time or inclination to write. This is a vicious circle, since more accessible 'fixed' resources would reduce the need for such cloisters.

Open Issues

At the time of writing, there are nearly 31,000 Keras-related questions on Stack Overflow[45], as opposed to less than 9,000 for PyTorch[46]. Perhaps it Is for the individual to decide if this reflects on the popularity of Keras, or else on the fragmented nature of its support systems; or, perhaps, on the superior online resources for PyTorch developers.

Additionally there are over 3,000 open issues at the Keras GitHub at the time of writing[47]. Though there are more than 5,000 unclosed issues at the PyTorch GitHub right now[48], that repository has *closed* over 11,000 issues to date, versus the 7,000 closed at the Keras project. The increased concentration of developer resources at PyTorch means a reduced number of long-term bugs compared to the slower productivity of the smaller Keras developer group.

Though PyTorch maintains an even more exhaustive instructional wiki[49] than Keras, it must be considered that PyTorch is also a lower-level and less-abstracted API, with a greater level of explicit complexity for the coder and a greater need for a higher volume of documentation.

One could argue that there is an inverse relationship between reach and documentation between the two frameworks: though the insurgent PyTorch is the relative newcomer for those who want to 'move fast and break things', its Facebook-led communities and investment in outreach means that PyTorch is generally much better (and *usefully*) documented than Keras*.

The fragmented history of Keras has even led its creator to archive the documentation of previous versions for those users who are currently committed to older iterations of the framework[50].

In terms of abstract internet wisdom, PyTorch is less hindered by the outdated 'authority' posts that can plague the Keras initiate, since PyTorch's development has been more consistent since its inception.

PyTorch maintains an active and helpful user forum[51], while Keras defers to the Stack Overflow community.

**It should be noted that while PyTorch's documentation and help resources are currently in advance of those of Keras, it is a* relative *comparison — the breadth, accessibility and extent of PyTorch's resources still invite frequent complaints among end users[52].*