EXXACT Blog

Deep Learning

Introduction to PyTorch Lightning

June 9, 2021 11 min read

An Introduction to PyTorch Lightning

Anyone who's been working with deep learning for more than a few years knows that it wasn't always as easy as it is today. The "kids these days" have no idea what it's like to roll their own back-propagation, implement numerical gradient checking, or even understand what it's like to use the clunky, boilerplate-heavy API of TensorFlow 1.0. It has gotten significantly easier to implement a state-of-the-art model and training on a new (version of an old) problem, especially in those categories that have had ample time to mature over the last decade, such as image tasks.

In many ways and on many tasks, deep learning has gone from an experimental science to an engineering problem, and finally to a technical task. In fact, for most applications in the wild, data and statistics skills are probably more important than deep learning and neural network engineering. We might call this the “data science-ification” of deep learning.

With the ease of use to build, train, and transfer model skills with high-level libraries like Keras, fast.ai, and now the topic of this article, PyTorch Lightning, you’ll certainly need more than a custom MNIST demo and a couple of MOOCs on your deep learning resume to stand out.

That’s not to say that there’s no longer any value in keeping up with the latest state-of-the-art research literature and rolling out a few of your own experiments to grow the field as well. Although a standard conv-net is no longer going to impress anyone on its own, there are plenty of areas of research that are at very early stages of development with ample room to grow. Plenty of people have been predicting the fall of deep learning and the next AI winter for years now, and yet we have graph neural networks gaining steam, especially in structural biology and computational chemistry, and many other exciting and nascent areas of active research (such as the lively world of differentiable cellular automata).

There are still edges to be honed and avenues to explore in deep learning research. That’s not even mentioning that the research side of things is often the most fun, a good reason to pursue a trajectory if we ever saw one.

So, where does PyTorch Lightning fit in?

PyTorch Lightning speeds up development by abstracting and automating many aspects of the deep learning workflow. There’s something of a cliche, that most of your time will typically be spent cleaning data and building pipelines. PyTorch Lightning is intended to remove some of this pain, so that developer/researcher time is not so heavily taxed by the mundane. At this point, readers with significant experience using PyTorch may be starting to see some similarities with another project. After all, didn’t PyTorch already have a high-level abstracted library with an emphasis on improved efficiency?

What is PyTorch Lightning?

PyTorch Lightning is a high-level programming layer built on top of PyTorch. It makes building and training models faster, easier, and more reliable. But isn’t that what Jeremy Howard and Rachel Thomas set out to provide when they launched the fast.ai library and online courses? In some respects yes, but the intended use cases and utility of PyTorch Lightning versus fast.ai are actually quite different.

Fast.ai was built on the idea that most of the programming effort (and computational expenditure) of deep learning was, in practice, wasted. That's because every other deep learner was still trying to "be a hero," training from scratch, building all their own model architectures (even when those architectures were more or less the same as every other deep learning practitioner was coding), and often introducing a few of their own idiosyncratic flavor along the way in the form of bugs.

When fast.ai was introduced, the basic premise focused on two things:

1. deep learning is not that hard, and,
2. 90-99% of deep learning engineering effort could be replaced by simply taking full advantage of transfer learning.

In addition to providing a high-level programming interface akin to Francois Chollet's Keras (that has been integrated into TensorFlow for a few years now), fast.ai makes it trivially easy to download some state-of-the art pre-trained weights and fine-tune them on your own problem with an optimized learning schedule.

That's a big part of why you'll find the library heavily represented on Kaggle, a data science competitions website.

The motivation behind PyTorch Lightning has a lot in common with the reasoning behind fast.ai. **However, PyTorch Lightning is much more geared toward research.** Where fast.ai makes it easy to forget about all the detailed inner workings of a deep learning model, focusing on the data in, data out, and optimizing the training curve, PyTorch Lightning likes to make it easy to try something new.

PyTorch Lightning aims to abstract away the "boring stuff" related to data hygiene, validation, etc. leaving experimenters with more cognitive cycles to apply to the "fun stuff" of wacky new architectures and proof-of-concept capabilities.

The emphasis on what to abstract away and streamline for PyTorch Lightning is essentially the opposite of fast.ai, which tends to abstract the details about models and architectures so practitioners can focus on the data science. Therefore, we can think of PyTorch Lightning as being for research what fast.ai is for data science.

PyTorch Lightning has a few other tricks up its metaphorical sleeve, however, especially related to scaling and speeding up.

Interested in a deep learning workstation?

Learn more about Exxact AI workstations starting at $3,700

PyTorch Lightning at Speed

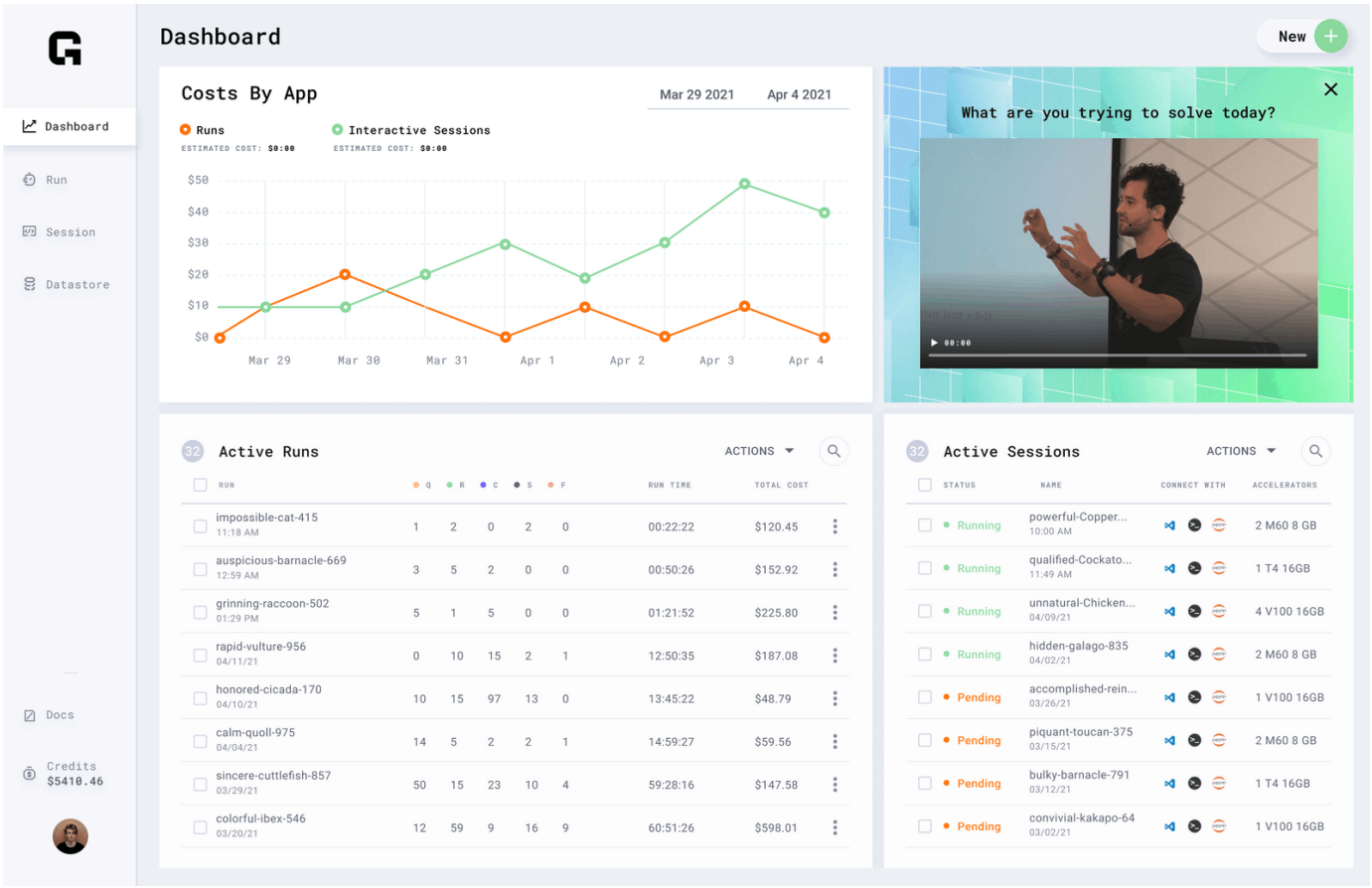
One of the primary must-have value-adds for any deep learning library is hardware acceleration. That's probably second only to automatic differentiation and back-propagation. We would even argue that hardware acceleration can be *more* important than automatic differentiation, as you always want fast matrix multiplies but sometimes you don't even care about gradients (e.g. for black box optimization or evolutionary computation).

It has thankfully been a long time since training a deep neural network on a GPU meant manually programming in CUDA. PyTorch Lightning takes this one (or maybe several) steps further by supporting TPUs out of the box and removing all barriers to training on multiple devices while taking full advantage of lower precision, faster, 16-bit training and inference.

PyTorch Lightning – Training at Scale, & Everywhere

PyTorch supports training on multiple devices (*i.e.* more than one GPU), and in most cases it is as simple as wrapping your model in a "nn.DataParallel" object. Anything more involved, however, and things get a little more tricky.

PyTorch Lightning does offer a few tools for streamlining multi-GPU training by following their programming tips, but where the library really offers some value is by making it much easier to perform distributed training, for example on an on-premise cluster. Another popular use case is in parallelizing experiments across multiple virtual machines in Google Cloud Platform or AWS. PyTorch Lightning facilitates distributed cloud training by using the grid.ai project.



You might expect from the name that Grid is essentially just a fancy grid search wrapper, and if so you may remember that grid search is one of the least efficient ways to tune hyperparameters. However, Grid also has plenty of options for sampling hyperparameter values from several different distributions and, according to the PyTorch Lightning documentation, many different schemes for hyperparameter search.

Is PyTorch Lightning Difficult to Learn?

PyTorch Lightning code looks like PyTorch code, but with less code. It moves the emphasis solidly away from the training loop and into the model.

In PyTorch you may be used to sub-classing the “nn.Module” class when building models. PyTorch Lightning uses a similar pattern with the pytorch_lightning.LightningModule.

A tremendous amount of functionality has been moved into the LightningModule. Whereas typically a programmer may spend a non-negligible amount of time writing training and validation loops, and, let’s face it, most of the code is copy and paste or boilerplate, PyTorch Lightning moves that functionality into the LightningModule while still retaining significant flexibility. In short, instead of re-writing the same code over and over again to take care of training, validation, logging, etc., you can instead just worry about over-writing those functions where more flexibility is needed in a

LightningModule model. This is similar to overwriting the “forward” function inherited from built-in forward function in “nn.Module” in PyTorch.

The team (and documentation) over at PyTorch Lightning likes to say that the library doesn’t abstract away PyTorch so much as it just organizes it.

PyTorch LIghtning provides a number of video and text tutorials for those looking to get started. Additionally they’ve put together “Lightning Bolts” which are part tutorial and part community contribution. Bolts are mostly community-sourced contributions of state-of-the-art implementations and pre-trained models, as well as datasets and a few other examples of support functionality.

It’s also readily apparent from the website and documentation that the PyTorch Lightning team prides themselves on putting out well-tested code, so you can be reasonably certain that Lightning (and Bolt) functionality has good support on multiple hardware devices. They’ve also had enough success and interest to form a company, so you can anticipate that they’ll be offering expanded paid services for industrial and institutional labs in the near future.

Applications of PyTorch Lightning

PyTorch Lightning isn’t the first library to add streamlined capabilities on top of PyTorch. See their own blog post comparing Lightning to PyTorch Ignite and fast.ai for an overview of the main differences from their perspective. From our point-of-view, fast.ai is geared more towards data scientists (especially Kagglers) adding deep learning to their toolbox. That is to say, fast.ai comes from a very practical mindset.

Ignite and Lightning, on the other hand, are aimed at the research community. Lightning differentiates itself from Ignite by making scale much easier to achieve, going beyond the typical multi-GPU support to enable training across many nodes and devices, as well as placing a significant focus on best practices. You know the engineer on your team that is always bugging everyone about testing everything from every angle? There’s a little bit of that mentality in the way that Lightning is developed and described.

While not nearly as popular as fast.ai, Lightning has significantly more interest on Github than Ignite as of this writing, which could be a good sign of continued support going forward. Unlike the utilitarian aims of fast.ai, Ignite and Lightning are both built with experimenters in mind, those who are more interested in exploring the space of what’s possible with deep learning than solving a particular analytics problem with a standard set of steps.

Related Posts

One way to put it is that PyTorch Lightning may be “more PyTorch than PyTorch,” which for a particular type of experimental enthusiast is a very good thing.