Comparative Analysis of Deep Learning Frameworks

Pravar Mahajan
mahajan.89@osu.edu

There has been a rapid surge in the number of Deep Learning frameworks available to a developer. These frameworks not only allow researchers to rapidly build, deploy and test prototype models, but many of them have also been designed and optimized to be production-efficient. As a result, for a new user venturing into the area of deep learning, it often becomes a hard to choose among the various options available.

In this work, we perform a comparative analysis of three popular Deep Learning frameworks - Tensorflow, PyTorch and Keras (with Theano backend). We look at several factors which would be of interest for a new user to start working in Deep Learning, for example, performance on benchmark datasets, ease of development, computational efficiency, documentation quality, community support and compatibility with other scientific computing frameworks like NumPy. We hope this work serves as a useful guide for those looking for the right framework to suit their tasks.

Introduction

A rapid surge in the number of Deep Learning frameworks has led to the democratization of deep learning based models. Whereas previously building models was considered to be the job of experts who are trained in advanced Mathematics as well as highly skilled in Computational Programming, today every user with some familiarity with topics in Deep Learning can build models which give state of the art performance in various Machine Learning tasks, like image recognition. The availability of various choices of Deep Learning frameworks allow developers to choose the framework best suited tasks at hand. Not all frameworks are created equally, therefore making an informed decision is often a difficult task. With this work, we hope to simplify the task by providing a comparative analysis of three of the most popular frameworks - TensorFlow, PyTorch and Keras.

Before going into the analysis, we start with providing a brief description of each of the frameworks compared

Tensorflow

Originally developed by The Google Brain Team [1], TensorFlow is arguably the most popular deep learning framework available today. Based on the idea of computational graphs, TensorFlow works by allowing the user to symbolically describe the structure, which is then compiled and optimized by the framework. A lot of optimization occurs via calls to low-level code written in C [1]. It allows smooth integration with GPU by allowing the same code to work in CPU as well as GPU, depending on the version of TensorFlow installed. It is actively maintained by a big community of 1000+ developers. [2]

PyTorch

PyTorch started off as a port of Torch (written in Lua) to Python, today it's an independent open source project maintained by about 300 developers [3]. PyTorch is more than a deep learning framework; it can be used as a Tensor computation library (like NumPy) with advanced GPU acceleration[3]. Like TensorFlow, it also allows user to specify a Computational Graph which is then compiled for performance optimization.

Keras

Keras is a “meta framework” owned and maintained primarily by François Chollet (there are 500+ contributers otherwise), an AI researcher at Google. It allows specification of Deep Learning models at a very high level, which then gets chosen to one of the backend frameworks configurable by user. Currently Tensorflow, CNTK and Theano are supported as backends[4]. The ease of development in Keras has made it the second fastest growing Deep Learning Framework [5]. In 2017, Google started supporting Keras in it’s own Tensorflow library.

Criteria for Evaluation

We are evaluating the frameworks on several factors, divided into two broad categories:

(1) Experimental: Under this category, we evaluate the performance of the three frameworks based on the speed of training and accuracy of the trained model on two open and popular datasets: MNIST handwritten digits and MNIST Fashion. More details on the architecture used, the experimental setup and results have been described under the section “Experimental Evaluation”.

(2) Non-Experimental Evaluation: Under this category, we evaluate the frameworks on qualitative parameters, like ease of development, documentation quality, size of the development and support community and integration with other scientific computing packages. Due to subjective nature of the parameters, it is possible that some of the qualitative results are subject to personal opinions and biases, even though we have tried our best to make the evaluation as objective as possible. Please refer to the section “Non-Experimental Evaluation”

Experimental Evaluation

In this section, we describe and evaluate the performance of the frameworks under consideration, on standard benchmark datasets. For evaluation, we train a standard CNN model (described next) built on each of the three frameworks, and calculate the test accuracy. Time to train and test accuracy have been described in Table 1. We begin by describing the datasets and the CNN model used.

Dataset

Two benchmark datasets have been used for evaluation - MNIST handwritten digits and MNIST fashion. MNIST handwritten digits[6] is a dataset consisting of 70,000 images of handwritten digits, each image being 28x28 in dimensions. The dataset is very popular in the Machine Learning and Computer Vision community, with the state of the art results in excess of 99%.
Because of nearly 100% accuracy of new models on MNIST handwritten digits, MNIST fashion[7] was released in August’ 17 as a drop-in replacement for the original MNIST dataset. The 28x28 images are grayscale pictures of fashion items, like shirts, trousers, sneakers etc. We demonstrate that MNIST Fashion is slightly harder dataset to solve and leaves quite a scope for improvement in terms of accuracy.

CNN Model
The CNN model used for the task of classification is a deep neural network model having same architecture as described in Tensorflow's MNIST tutorial[8]. The model consists of a 28x28 input layer followed by two alternating sets of 5x5 convolution layer (1x1 stride, 2 rows/columns of zero-padding on each side) and 2x2 max-pooling layer (2x2 stride, no padding). This is followed by 1 fully connected layer with 1024 units and rectifier-linear activation function. Finally, we have the output layer consisting of 10 neurons and softmax activation function. Dropout of 0.5 has been added between the two dense layers to prevent overfitting. We minimize the cross-entropy loss between model's output and the actual output in order to train the model, while using Adam[9] as the optimization algorithm.

Experimental Setup
The MNIST dataset has already been split into 60,000 training and 10,000 testing samples. We train the CNN model as described previously on 60,000 training samples. The accuracy reported is as seen on the testing samples. We ran our experiments on 12-core Intel Xeon E5-2670 processor, with Tesla K80 GPU. The time taken represents the total time taken to load dataset, compile model, train and then predict on test dataset. Each experiment was performed 5 times and the reported numbers are means with ± 1 std dev of uncertainty.

Results
The results have been succinctly summarized in Table 1 and Table 2. The first striking (and very worrying!) observation is the fact that some differences exist in the accuracies obtained using the same model on different frameworks. The differences are not consistent across different datasets, therefore we couldn't come up with a possible explanation for such observed differences. We have looked at some of the small default parameters which are provided by the APIs (like momentum operator in Adam) and tried to bring them to same value as much as possible. We hypothesize that such differences may be attributed to specific ways in which optimizers have been implemented in each of these frameworks, however we couldn't find a good way of testing our hypothesis.

Tensorflow's easily beats PyTorch and Keras (with Theano backend) when it comes to speed of training and testing. In fact, Keras takes almost thrice as much time as Tensorflow, which can be explained by the fact that Keras is a meta-framework and lots of overheads possibly exist in providing an API which can work with different frameworks working at the backend.

Finally, we observe that the same model which gives ~99% accuracy on MNIST handwritten digits, has a 10-fold increase in error rate when evaluated on MNIST fashion dataset. This shows that MNIST fashion dataset is slightly harder to solve, leaving a huge scope of improvement. This justifies the release of fashion MNIST dataset as a drop in replacement for handwriting MNIST dataset for future machine learning and computer vision work.

Table 1: Comparison of Accuracy across Different Frameworks

	Tensorflow	**PyTorch**	**Keras (Theano Backend)**
MNIST (handwritten)	$99.17 \pm 0.1\%$	$98.65 \pm 0.07\%$	$98.94 \pm 0.09\%$

MNIST (fashion)	90.93±0.26%	89.88±0.37%	91.23±0.17%

Table 2: Comparison of Time taken for different Frameworks

	Tensorflow	PyTorch	Keras (Theano Backend)
MNIST (handwritten)	3m 14s ±1s	6m 0s ±9s	9m 40 ±28s
MNIST (fashion)	3m 19s ±1s	6m 5s ±5s	9m 25s ±8s

We have released the code to run all the experiments on github.

Non-Experimental Evaluation

In this section, we compare the frameworks based on some of the qualitative criteria. Many of these evaluations are subjective in nature and therefore subject to personal opinions and biases.

Ease of Usage

Keras is a lot easier to learn and much better suited for building prototype models and other research purposes. The model description is made at a very high level, making it easier to program and difficult to make mistakes in building the model. Even within keras, changing from one backend to other involves making one line change in a config file.
Comparing Tensorflow and PyTorch, one important difference was in terms of ease of porting the code to GPUs. No change is required in Tensorflow, the same code will run on CPU or exploit the GPUs based on the version of Tensorflow installed. However, same flexibility does not exist on PyTorch, we have to declare every variable using `cuda()` function for the code to utilize GPU instead of CPU.

Community Support Size

The number of contributors for each of the frameworks is as follows:

Framework	Contributors
Tensorflow	1021 [2]
PyTorch	295 [3]
Keras	521 [4]

Documentation Quality

Being open source, the documentation quality and coverage largely depend on size of the active contributors. We find that tensorflow has extensive documentation coverage, with lots of tutorials. The community is actively involved in updating the documentation as there are changes to the APIs. This is not the case with Keras and Tensorflow, where documentations are either incomplete, or not clear in many cases.
Scopes of improvements exist in each of the documentations, by providing suitable examples on how each and every function is run.

Datasets Available
All the 3 frameworks provide ways to easily download and install certain benchmark datasets. For example, support for MNIST handwriting dataset exists in all the three frameworks.

Integration With other Scientific Computing Libraries
Another useful aspect of keras is the ease with which it can be integrated with other scientific computing libraries, in particular NumPy and SciPy. Keras provides functions (for example [10]) which can directly run training and evaluation on NumPy's `ndarray` objects. To the best of our knowledge, there exists no direct support for integration with NumPy in TensorFlow and PyTorch and the numpy variables need to be cast into either Python's list or some other internal object types of these frameworks before they can be used for training/testing.

Conclusion

We performed a study on 3 different deep learning frameworks, and evaluated the on several quantitative and qualitative parameters. The results of our study may be summarized as follows:

(1) Each of the frameworks have different performances with respect to accuracy as well as speed, while using the same model on the same hardware and same dataset. Although the differences in accuracies are small, it is still a little worrying to see that such differences. These differences may manifest themselves in much bigger way on datasets like MNIST, where the error rate is already so small that a 0.1% improvement in accuracy makes a difference.
(2) Differences also exist in the way models are built and trained, with keras providing the easiest way to build large and complicated models. Depending on the size of the active development community, differences also exist in documentation quality.
(3) We note that Fashion MNIST is harder to solve than MNIST handwriting dataset, without changing the size and dimensions of the latter, making it an excellent drop in replacement for benchmarking of Machine Learning models. Minimal changes were need to switch the experiments from running on the handwriting dataset to the fashion dataset.

In the light of the above points, we therefore conclude the work by stating that not all deep learning frameworks are the same, with differences arising from not only the ease of usage and flexibility of integration with other libraries, but something as basic as accuracy score while using the same models. We also concur with Yann LeCun's arguments that the fashion MNIST can (and should) serve as a replacement for the original MNIST dataset, given that it's not easy to push the error rate below 10% on Fashion MNIST.

References

[1] https://www.tensorflow.org/
[2] https://github.com/tensorflow/tensorflow
[3] https://github.com/pytorch/pytorch
[4] https://github.com/fchollet/keras
[5] https://twitter.com/fchollet/status/776455778274250752
[6] http://yann.lecun.com/exdb/mnist/
[7] https://github.com/zalandoresearch/fashion-mnist

[8] https://www.tensorflow.org/get_started/mnist/pros
[9] https://arxiv.org/abs/1412.6980
[10] https://keras.io/models/sequential/