Homework 6. Language bindings for TensorFlow

Boyuan He
Computer Science 131, Programming Language
University of California, Los Angeles
boyuan_heboyuan@live.com

Abstract

The purpose of this study is to look into different languages that can serve as a replacement for Python regarding using TensorFlow machine learning library. Specifically, I will investigate TensorFlow architectures and TensorFlow in other languages such as Java and Ocaml. By comparing the pros and cons of those languages as the platform for TensorFlow, I will give suggestions for picking languages as replacement for Python.

Introduction

TensorFlow is an open-source software library built and maintained by Google. It is very suitable for machine learning tasks such as building neural networks. When TensorFlow is built using C++, the first client language it supported is Python. Although most times Python is good for the task, sometimes, it can be the bottleneck for performance. Thus, this paper will look in different replacement of Python and client language for TensorFlow. I will start by analyzing TensorFlow's functionality that a language need to support and the architecture of TensorFlow library. Then I'll move on to discuss the pros and cons of using other client languages for TensorFlow.

1. TensorFlow

1.1. Architecture

The general architecture of TensorFlow is shown in the figure below, on the top level is the client code written in different languages, then there is a C API that separates user code in different from the core runtime. Below that is the Distributed Master that deal with graphs and the kernel implementation that does the calculation.

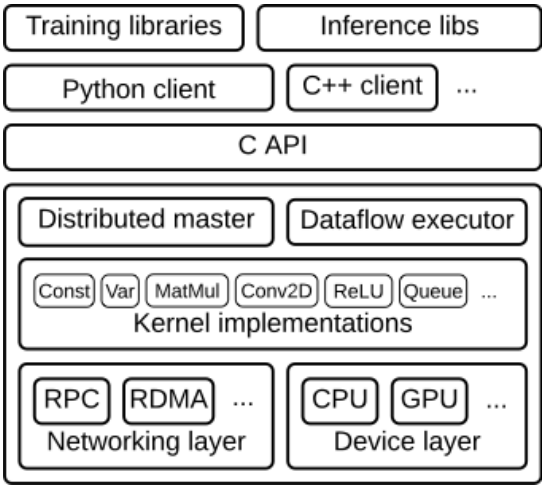

Figure1. TensorFlow architecture [1]

User of TensorFlow writes client programs that builds the computation graph, and when they create session to run the graph, the graph will be sent to distribute master. The distribute master will then break do the user graph into small and distribute them to workers to do the work.

1.2. Functionality

The core features required of a binding for TensorFlow is listed below. [2]

Run a predefined graph – Given a GraphDef protocol message, need to be able to create a session, run queries, and get tensor result

Graph construction – At least on function per defined TensorFlow op htat adds an operation to the graph

Gradient – about to do partial derivatives

Function – able to define functions that can be called many time

Neural Network library – components that can support creating neural networks. This currently only supported for Python.

Control Flow.

2. Possible client languages for TensorFlow

In this section, I analyzes different languages as the possible client languages for TensorFlow.

2.1. Java

Comparing to Python, Java is harder to use for various tasks of Machine Learning. This is due to the fact that Python and R are mostly used by data scientist even before TensorFlow come out. So, there are many libraries that make it really easy to process the large amount of data for machine learning tasks. For instance, there is Numpy to deal with matrix, Matplotlib to visualize data, Pandas to load and process data, sk-

learn to preprocess data. So, from the perspective of easiness, Python definitely wins. Also the fact that there are so many data science tools for python ensures a good generality for Python doing machine learning tasks.

Moreover, when we consider flexibility, Python's dynamically type checking definitely gives it an advantage over Java, which is statically typed. Thus, on can argue that Python is flexible. However, the fact that Java is more low level and allows programmer to manage more low-level things make it more flexible in the sense of providing more control over a program

As for performance, it really depends on the implementation. For instance, the original CPython would run much slower than Java, however, the PYPY, which is very similar to Java's Hotspot just in time compiling, could run approximately the same or even faster than Java in some cases [3]. However, one should be aware that both of these languages were not good for performance, that is, their performance different is not significant. It will be very rare that one finds Python not enough for performance while Java okay.

As for event driven server support, Java have Grizzly and Deft library based on the Java New I/O functionality added in Java 1.4, so there shouldn't be much problem with building even driving server.

2.2. Ocaml

As for easiness to use, Ocaml will be harder for machine learning tasks. Although there are libraries for data science in Ocaml such as Owl, its' library support is much less that Python or Haskell, which will be discussed later. Ocaml has a relatively fragmented ecosystem dur to multiple standard libraries, which also prevented the development of easily useable libraries for Ocaml. However, Ocaml have a easily usable foreign function interface for C, which could make the implementation of TensorFlow easier.

When we consider flexibility, Ocaml is static type checking and it tries to infer the type for everything in the program. So it is less flexible that Python. However, the advantage of Ocaml's type inferring makes it easier for programmer to find bugs before they even run the code. This can be even more useful when we consider machine learning, during which programmers need to keep considering the shape of the tensors that are used in different layers.

When it comes to performance, although different implementation can have a large influence, Ocaml generally wins by a lot. The native compiler for Ocaml is fat better that the Python interpreter when we considering the performance [4]. Sometime, Ocaml can even generate code comparable to C and C++. However, one shouldn't keep in mind that similar to Python, Ocaml also have a GlobalInterpreterLock that requires programmers to do multiprocessing instead of multithreading.

As for implementing an event driven server, Ocaml probably is not the best choice. The fact that Ocaml's poor support for multithread limits is ability. Specifically, Ocaml runtime is not thread safe itself [6], so one need to use interface to other language for many of the applications. Admittedly, there are packages such as Lwt (Light Weight Threads) and Async, the lack of popularity of these libraries means less support from the community.

2.3. Haskell

Unlike any of the language discussed above, Haskell is a pure functional programming language, which makes it hard to use. It would be hard to create neural network models and use them. Moreover, Haskell is filled with both confusing jargon and confusing types, which make the situation even worse. Lastly, the due to the lazy evaluation of Haskell, it would be very hard for programmers who don't have a good understanding of programming language to take advantage of Haskell's performance.

Here, I should mention that although Haskell is harder to use than Python, it has a wider library ecosystem especially for data science comparing to Ocaml.

However, there is still upside of Haskell. For instance, Haskell allows that non-strict, meaning expressions to have a value even if their sub-expressions do not; this prevents it from failing or requiring everything to be evaluated [5]. Is gives Haskell great generality.

Furthermore, Haskell have a great performance. Haskell is not interpreted language like Python, or virtual machine system like Java, it compiles everything down to machine code, so it doesn't have many overheads of Python. Since it is statically type check, there is no overhead of checking types during run time either. The pure functional nature ensures that Haskell compiler can do very aggressive optimization without considering the influence of changing the value of a variable. One should also be aware that Haskell have a great production-ready parallel runtime, however, is would less likely to be influential in machine learning considering one can get much more performance form GPU than CPU and Nvidia had developed CUDA and cuDNN for machine learning.

Haskell is kind of special if we considering implementing a event driven server. According to Bryan O'Sullivan and Ho-han Tibell's paper, even and thread in Haskell are kind of unified. In fact, threads are actually implemented in term of event and run across multiple cores. Moreover, concurrency is not hard in Haskell because most code is pure and so is thread-safe. Thus, it would be easy to implement a server that take a different approach but is equally good as, if not better than, event driven server by Nodejs. (that is, numerous light weight thread each serve client)

3. Conclusion

After reviewing Java, Ocaml and Haskell. I would say that Haskell is the best solution despite the fact that its steep learning curve.

Admittedly, Java have a large community and it is easy to build an event driven server. However, The fact that we would not get much performance boost from Java comparing to Python make Java a less favored candidate for the task, since the original problem we are trying to solve is poor performance due to Python.

Ocaml will not be a good choice for both using as a client language for TensorFlow or build event driven servers. It's relative hard to learn, it has a bad library system and its poor performance improvement comparing to Haskell make it a unfavored candidate for our task

Although Haskell is hard to learn, it has great performance due to its pure functional nature. Moreover, its rich tools for data science make things every better. Furthermore, the fact that Haskell is great for building server make it even better for the application we are developing.

Reference

[1] TensorFlow Architecture,

https://www.tensorflow.org/extend/architecture

[2] TensorFlow in other languages,

https://www.tensorflow.org/extend/language_bindings

[3] Which one is faster, Python or Java,
https://www.quora.com/Which-one-is-faster-Python-or-Java

[4] Why OCaml, Why Now,
https://news.ycombinator.com/item?id=7416203

[5] Haskell for Data Science: the Good, Bad and Ugly,
https://www.linkedin.com/pulse/haskell-data-science-good-bad-ugly-tom-hutchins/