Language Bindings for TensorFlow

Jason Lai
ID: 204-995-126

University of California Los Angeles

1 Introduction

For this assignment, we consider running an application server proxy herd on a set of virtual machines; the application that this herd is executing uses machine learning algorithms reliant on TensorFlow. TensorFlow is an open source library designed for machine learning that provides a collection of workflows to develop and train models. However, due to the complex nature of TensorFlow computations, we see that a majority of the application's runtime on the server herd is spent using Python code to set up models. This results in a bottleneck in performance, so we consider other languages as alternatives to Python in order to achieve a more efficient implementation of our machine learning application.

Three languages that we consider to implement the application are Java, OCaml, and Kotlin. When we consider which language is best suited to TensorFlow models, we look at a language's performance, reliability, flexibility, generality, and ease of use. To gauge each aspect of these languages, we research their respective documentation and supported software, as well as consider which language is best suited to replacing the implementation of the event-driven servers written in Python.

2 Potential Languages

While Python does offer many reasons to use it for TensorFlow, we see that one of its biggest drawbacks is its performance, so we consider three other languages: Java, OCaml, and Kotlin, and weigh them against each other and Python to choose which language is the best candidate for TensorFlow, while still maintaining the best reliability, flexibility, generality, and ease of use possible. In order for a language to work well with TensorFlow, it must be able to interface with the core of TensorFlow, which is written in C++. [1] This requires that languages use a foreign function interface in order to provide TensorFlow functionality.

2.1 Java

Unlike Python, Java is statically typed, supports multithreading, uses a distinct mode of compilation known as the Java Virtual Machine, and uses a different method of garbage collection. Java's static typing offers greater reliability against runtime bugs and executes code faster as the Just-In-Time (JIT) compiler does not need to check the types of variables at runtime like Python's interpreter does. Furthermore, Java's support for multithreading gives Java an edge over Python in performance. The Java Virtual Machine further optimizes Java programs by loading a program's code and either interpreting it to byte code or using a JIT compiler to compile it to machine code, depending on the spatial and temporal locality of the program. The Java Memory Model also allows the compiler to rearrange code in a manner that makes the program more efficient without changing the result. And while Python's reference count garbage collector is faster than Java's method of mark and sweep and generational garbage collection, both are very effective, so they're about even in this regard. From all of this, we see that Java has some obvious performance and reliability advantages over Python at the cost of flexibility.

In respect to TensorFlow, Java's static type checking would impose constraints on the TensorFlow functions, since developers must explicitly define the types of all variables and functions. This results in less flexibility when building a model and makes the language harder to use. However, this helps to mitigate the probability of runtime errors that could result if we used a dynamically typed language like Python. On the other hand, functions for Java can be parallelized using multithreading, which gives Java an edge in performance over Python. This can significantly improve the state of the bottleneck when setting up models for our application. And since Java is somewhere between an interpreted and compiled language, we retain the portability that Python has

and achieve a performance gain with the Java Virtual Machine's optimizations.

Beyond the inherent characteristics of Java, Java also has libraries which support TensorFlow implementations such as Maven. [2] For Java to interface with the C++ core of TensorFlow, the Java Development Kit has a way to bridge the interpreted byte code produced by the Java Virtual Machine and a different, native language such as C++. This bridge is known as the Java Native Interface (JNI), which allows us to both call and be called by native applications written in languages like C++ through the Java Virtual Machine. [3] This provides us with the requisite foreign function interface to work with TensorFlow's C++ core.

Furthermore, Java has a variety of supported libraries like Python and even has its own library for asynchronous event-driven server applications called `netty`, which could be used in place of Python's `asyncio` library to implement the server herd. [4] Due to Java's range of libraries, it has a good amount of generality for the number of applications the language can be used for.

2.2 OCaml

In contrast to the other languages we consider, OCaml is a functional programming language based on the original ML functional language, but with the addition of object conventions characteristic of object oriented programming. Like Java, OCaml is a statically typed language with its own garbage collector and a compiler. However, OCaml's type checking is very strict and has strong type inference, imposing even greater constraints on the developer than Java's type checking method. As a result, OCaml is much more difficult to work with, however, we gain some reliability. Unlike Java, OCaml lacks support for multithreading. Other features of OCaml include automatic storage management and a simple syntax with shortcuts in the form of syntactic sugar.

While OCaml has the advantage in reliability, the greatest drawback for OCaml is its ease of use, or lack thereof. Functional languages rely on recursion to iterate through data structures and discourage the use of loops and implementations that have side effects i.e. changing some global state inside of a function. Due to the sheer number of constraints that OCaml places on developers, this makes OCaml inflexible. And while OCaml is very fast at iterating through arrays, making it potentially useful for processing matrices quickly for TensorFlow, it's apparent that implementing our application with OCaml would be very challenging. Furthermore, without multithreading, OCaml falls behind Java in performance.

Despite the difficulty of implementing with OCaml, OCaml does have a tensorflow library known as the tensorflow-ocaml project. [5] This library provides some OCaml language bindings for TensorFlow, contributing to some ease of use. Additionally. OCaml's compiler can link with external system libraries via C code or produce native object files to invoke functions in other languages such as C++. [6] This gives OCaml a way to implement a foreign function interface that is compatible with TensorFlow's C++ core.

Like Java and Python, OCaml also has a library called `Async` that can be used to implement an asynchronous, event-driven server herd. [7] This adds to OCaml's generality and ease of use, but ultimately, OCaml is much more difficult to implement with than Java.

2.3 Kotlin

Kotlin is like OCaml in that it combines aspects of object oriented programming with functional programming, but is designed to be Java compatible in that it is interoperable with the Java Virtual Machine and other Java-based platforms. [8] This allows any and all Java code to be translated to Kotlin and vice versa, without changes in a program's behavior.

For the most part, Kotlin and Java share much of the same features, barring the different syntax. However, Kotlin does provide many additional features such as extensions, which extend a class's functionality, coroutines, which help to avoid thread blocking, and more. [9] However, in regards to TensorFlow, we see that Kotlin has a compiler called Kotlin/Native which has the ability to compile Kotlin code to native binaries without the aid of a virtual machine. [10] These native binaries along with Java's JNI allow Kotlin to interoperate with other languages such as C++. In addition to this, there is an existing TensorFlow library for Kotlin produced by JetBrains called kotlin-native; it essentially is a TensorFlow client built atop the TensorFlow C API similar to how it is for Python. [11]

However, despite Kotlin's improvements upon Java's implementation and seeming ease of use from its support for TensorFlow, Kotlin's performance falls behind Java in some cases and matches Java in others. [8] For this reason, I believe Kotlin is not a good choice to implement TensorFlow as it is relatively new, so implementing with it may pose a challenge due to a lack of documentation and a new syntax compared to its predecessor. Furthermore, in order to interface with TensorFlow's C++ core, Kotlin may need to use a combination of its Kotlin/Native compiler and JNI, as Kotlin/Native lacks support for C++ libraries. This could prove to be harder to do that just using Java and JNI outright, which may allude that Kotlin is too inconvenient to implement the TensorFlow application and server with for so little performance gain compared to Java.

3 Conclusion

I believe Java is the best candidate to replace Python in setting up models for TensorFlow, as Java has a good balance between reliability, flexibility, performance, generality, and ease of use. Using OCaml would be too hard to implement our application and server, and Kotlin is newer than Java, so it may be harder to interface with it.

For our application, Java offers better performance than Python due to the Java Virtual Machine's optimizations, support for multithreading, and static type checking, but also retains many of the advantages that Python has. It's reliable due to its static type checking and is flexible, general, and easy to use due to the variety of useful Java libraries. More specifically, Java's Maven library is specifically made to facilitate a Java API for TensorFlow, and `netty`, which is an application server framework akin to Python's `asyncio`. The Java Native Interface also gives a way to establish conjoin a high-level TensorFlow interface written in Java with TensorFlows low-level core written in C++, making it well suited for TensorFlow. And since Java programs can be interpreted by the Java Virtual Machine, we retain the portability that Python has.

Furthermore, one of the reasons why Python was a good choice to interface with TensorFlow was because it was one of the most popular languages among the library's internal users. [12] Java could be said to be equally as popular as Python and just as well documented, making Java a good candidate for TensorFlow and the server herd.

References

1. TensorFlow. (n.d.). *TensorFlow in other languages*. [online] Available at: https://www.tensorflow.org/guide/extend/bindings [Accessed 8 Jun. 2019].
2. GitHub. (2018). *TensorFlow for Java using Maven*. [online] Available at: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/java/maven/README.md [Accessed 8 Jun. 2019].
3. Baeldung. (2018). *Guide to JNI (Java Native Interface)*. [online] Available at: https://www.baeldung.com/jni [Accessed 8 Jun. 2019].
4. Netty.io. (n.d.). *Netty: Home*. [online] Available at: https://netty.io/ [Accessed 8 Jun. 2019].
5. GitHub. (2019). *tensorflow-ocaml*. [online] Available at: https://github.com/LaurentMazare/tensorflow-ocaml/ [Accessed 8 Jun. 2019].
6. Minsky, Y., Hickey, J. and Madhavapeddy, A. (2014). *Real world OCaml*. Beijing: O'Reilly.
7. Jane Street Open Source. (n.d.). *async*. [online] Available at: https://opensource.janestreet.com/async/ [Accessed 8 Jun. 2019].
8. Dyvliash, V. (2018). *Java vs Kotlin: a Veteran against the Upstart Challenger*. [online] Belitsoft. Available at: https://belitsoft.com/apps-development-services/java-vs-kotlin [Accessed 8 Jun. 2019].
9. Kotlin. (n.d.). *Comparison to Java Programming Language*. [online] Available at: https://kotlinlang.org/docs/reference/comparison-to-java.html [Accessed 8 Jun. 2019].
10. Kotlin. (n.d.). *Kotlin/Native for Native*. [online] Available at: https://kotlinlang.org/docs/reference/native-overview.html [Accessed 8 Jun. 2019].
11. GitHub. (2019). *kotlin-native*. [online] Available at: https://github.com/JetBrains/kotlin-native/tree/master/samples/tensorflow [Accessed 8 Jun. 2019].
12. TensorFlow. (n.d.). *TensorFlow Architecture*. [online] Available at: https://www.tensorflow.org/guide/extend/architecture [Accessed 8 Jun. 2019].