Language Bindings for TensorFlow

Arnav Garg
304911796
University of California, Los Angeles

Abstract

TensorFlow is an open-source deep learning software library created by the Google Brain team for research and production that is written in Python, C++ and CUDA [1]. It is one of the most popular machine learning libraries and is used extensively to set up deep neural networks for classification and regression tasks.

Our application needs to handle many small queries quickly that create and/or execute small machine learning models. However, profiling and benchmarking our application reveals that most of the time is spent executing Python code to setup the models during these queries. This prompts the exploration of other bindings for TensorFlow.

Currently, it integrates best with Python, but also provides language bindings for other programming languages. This paper will explore whether Java, OCaml or Kotlin would be better than Python for our given use case, both in terms of handling event driven servers and their ability to support the TensorFlow machine-learning framework.

1. Introduction

TensorFlow applications, are usually bottle-necked within C++ or CUDA [2] for large applications. Further profiling reveals that a majority of the time is spent in performing matrix computations as compared to preprocessing and setting up the model itself. On the other hand, smaller applications with small queries are bottlenecked within Python, indicating that Python's model creation time is expensive and slows down the overall computation [2]. This prompts an exploration into TensorFlow's support for other languages such as OCaml and Java, and see whether they would be better suited for our application that runs many small queries on a server herd architecture.

We explore Java, OCaml and Kotlin as alternatives to Python for our use case, and weigh their potential pros and cons against each other and to Python. We also explore each language's performance, usability, reliability, generality and range of platforms that support it since we also need to handle client side code and only use the server to when absolutely necessary.

2. OCaml

OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles [2]. It is strongly typed language with built in garbage collection for efficient memory management. The OCaml garbage collector is a modern hybrid generational/incremental collector which outperforms hand-allocation in most cases [3] and is fast for immutable objects because memory is allocated by moving pointers. Although the language uses static type-checking, it is never actually written because OCaml checks its variables using type inference. Therefore, during development, a developer is made aware of each functions' input and return types without actually explicitly stating either of these attributes. This creates a steep learning curve since Python developers will need to keep track of and understand each function's return type within the TensorFlow API. As seen here, OCaml falls in between Java and Python because the developer isn't required to specify types as the OCaml compiler relies on type inference during static type checking and also provides the 'a type, which essentially means that the function can accept different types of variables.

OCaml and Python are both single threaded and don't support multithreading. Like Python's asyncio module, OCaml does support asynchronous code through frameworks such as Ocsigen [5] and uses locks to prevent multiple threads from running concurrently. Therefore, OCaml and Python are similar in providing asynchronous functionality since they both use modules and single threading for their asynchronous tasks. However, Python's asyncio appears to have a much more extensive and well-explained documentation than OCaml's Ocsigen, making it easier to design a server herd implementation in Python than in OCaml for client-server communication. Python's built in error handling in the form of catching exceptions is a major advantage, because developers would have to create their own try and catch functions in OCaml. Therefore, it would make sense to pick Python over OCaml for an application that requires a proxy server herd.

OCaml is compiled before running while Python programs are interpreted. The main difference lies in the fact that an interpreter produces a result from a program, while a compiler produces a program written in assembly language [6]. This allows OCaml to enforce type-checking and catch errors at compile time, speeding up the program. Additionally, OCaml can produce executables that can run with an ANSI C compiler on POSIX compliant systems. This increases its portability since these executables can be run on many machines, however, at the expense of optimization. On the other hand, since Python is interpreted, it is slower since the interpreter has to keep reprocessing lines and would require a Python interpreter to be installed another machine for a Python program to execute successfully.

It is also important to note that TensorFlow developers don't formally support the OCaml language binding. This poses two problems - the first is that implementations may not be reliable in terms of functionality and may be several updates behind the latest functions and features of Tensorflow. This can be seen if we inspect LaurentMazare's GitHub repository: github.com/LaurentMazare/tensorflow-ocaml/, where the last updates were made anywhere between 7 days to 3 years ago. The second is the limited community for TensorFlow-OCaml support that is restricted almost entirely to the developers of the bindings itself. If TensorFlow developers supported OCaml, all OCaml code would essentially serve as a wrapper around TensorFlow functionality, making it a bit slower and cumbersome because function types would require generalisation.

OCaml is widely used in analytical applications and academia than it is in server applications and industry. This means that it can be adapted to TensorFlow because it can be used to model to computation graphs efficiently since these are also used in academia and analytical applications. It is faster than Python because it is compiled rather than interpreted, but is still restricted by its ability to only support single threaded programming just like Python and has a steep learning curve because it uses functional programming rather than the very popular imperative style.

3. Kotlin

Kotlin is an object oriented and functional general-purpose programming language that supports cross-platform execution and is statically typed, with type inference [7]. Kotlin is designed to interoperate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise [7]. Similar to Python, Java and OCaml, Kotlin features an in built garbage collector that uses the mark-and-sweep function for garbage collection since it can run on the JVM. Therefore, Kotlin may have a similar learning curve to Java and pose some problems when trying to create a proxy server herd, but is significantly easier than OCaml because it offers the flexibility of imperative and functional programming. This also provides increased generality and better support across different types of applications and computation tasks.

If we look into Kotlin's performance, we see that Kotlin is a compiled language with static type checking, increasing reliability and speed. Kotlin features the ability to create coroutines and multithread, thereby possessing efficiency similar to Java. It can also support asynchronous functionality. This makes it a lot more powerful than Python or Java in terms of performance because it can utilize multiple coroutines, each of which can handle computationally intensive tasks concurrently and at high speed due to Kotlin's ability to write multithreaded code.

Since Kotlin runs on the JVM, its programs can use all Java libraries and the programs can be converted to bytecode just like Java programs. This increases Kotlin's ability to be flexible due to increased portability. Kotlin also supports code written on a browser, therefore giving it the highest generality of cross platform support in comparison to OCaml, Java and Python.

TensorFlow doesn't support Kotlin directly, but we see wrappers and bindings for TensorFlow written by the open-source community. Since TensorFlow was written using the C API, it would make it harder to write a language binding for Kotlin due to the lack of support (https://juliuskunze.com/tensorflow-in-kotlin-native.html). For TensorFlow, a developer would still need to go through the learning curve posed by functional programming to understand return types for various functions, but has increased flexibility due to the choice of programming style and the <Any> type checking, which can accept different types of variables defined in Kotlin.

4. Java

Oracle's Java first came out in 1995. It is a statically typed language with an in built garbage collector for automatic garbage collection and memory management. Since Python is an interpreted language, it tends to be a lot slower since it is processed at runtime. However, this also makes it easier to use since it has dynamic typing. On the other Java is compiled into bytecode by the Java Virtual Machine (JVM) and therefore tends to be faster than Python.

It is imperative in nature, which makes it easy to learn, read and understand. However, one big contrast from Python is that Java does not have type inference - instead, a developer needs to specify each function's return type during assignment and creating function definitions in Java. This may steepen the learning curve while switching from Python to Java. Additionally, a developer will be forced to learn the return types for all of the functions in the TensorFlow API and for effective client server communication, which can add to the difficulty of adapting to Java as a Python developer. Java is currently supported by TensorFlow, which indicates Java's success for TensorFlow applications, its widespread popularity and that it is typically a sought after language binding.

Both Java and Python have a strong developer community and extensive documentation for its core functionality and extended libraries. The languages are also maintained and constantly updated, allowing them to have constant improvements and bug fixes with increased support for documented issues. This also attracts a wider developer community.

Java beats Python in performance because Java is statically checked, which increases reliability by decreasing the room for runtime errors. Static type checking also allows for code optimization and faster runtime. However, in terms of asynchronous capabilities for a proxy server herd implementation, we see that Python's asyncio module does well due to the presence of coroutines that allow for asynchronous function calls. However, Python is single threaded while Java supports multithreading, which may allow computationally intensive tasks to execute faster in Java than in Python. However, since our task only has small queries, the cost of creation of threads itself may counteract the performance gain of multithreading, preventing any real benefits from being seen. Java and Python both support error handling through methods such as try-catch blocks, which is required for server side code. Java can also be used to write web applets, which are small downloadable internet-based programs [8]. This makes it a good choice for a proxy server herd implementation.

Most computers come pre-installed with JDK and JRE. This coupled with the fact that Java code is compiled into bytecode by the JVM allows for increased portability regardless of architecture. However, Python is not as portable because it is an interpreted language and therefore requires a Python interpreter to be installed on a computer for it to execute. It is interesting to note that while Python does have bytecode, but it is still interpreted, thereby decreasing its portability.

Java and Python both support OOP through the means of classes, subclasses and classic OOP concepts such as inheritance and polymorphism. Therefore, both Python and Java are great at providing means of reusing code by creating objects with member functions.

One of the only downfalls is that language bindings are usually open-source and not made directly by the core TensorFlow community, but is instead created by the open-source community that would like to adapt the TensorFlow framework to Java. (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/java). Therefore, this implementation may not be up-to-date, may be buggy, may not perform the exact computations and may pose additional complexity because of the lack of an extensive community to support Java language bindings for TensorFlow.

Conclusion

We can see that each programming language has its own pros and cons because of its nature, community, ease of learning, type-checking, type of language (interpreted vs compiled), etc. Java and Python appear to have the most support. Kotlin, OCaml and Java have an advantage when it comes to type-checking since they can reduce the number of run-time errors by catching errors during compile time. Kotlin and Java appear to possess tremendous potential when it comes to performing computationally heavy tasks because they allow for multithreading and concurrency. Although Python and OCaml are single threaded, OCaml is faster because of its static type-checking, ability to create optimizations during compile time. All of these languages have built in garbage collectors for memory allocation and deallocation (memory management). Python is the best suited for our prototype because of its asyncio module and simple syntax, while Kotlin wins in terms of flexibility and generality as it supports web apps along with full stack web and mobile development.

Returning to our goal of developing an event driven server that uses a language binding, we want to maximize asynchronous performance while handling client and server operations. It is also important to ensure reliability and usability, along with having a good developer community and documentation for our choice of language binding. Given these constraints, we see that Kotlin or Java might better suit our application compared to Python. Java is already extensively supported by the TensorFlow developers, which means that there will be decent support, automatic garbage collection and multithreading using a similar imperative style of writing code, making the move to Java easier. Kotlin can handle small machine learning queries with reliability and high performance because of static type-checking, providing multithreaded and asynchronous functionality, functional and imperative styles of programming, and good support since Kotlin runs on the JVM and can use Java libraries.

Citations

[1] "TensorFlow in Other Languages | TensorFlow Core | TensorFlow." *TensorFlow*, www.tensorflow.org/guide/extend/bindings.
[2] *Homework 6. Language Bindings forTensorFlow*, web.cs.ucla.edu/classes/spring19/cs131/hw/hw6.html.
[3] "OCaml - OCaml" *OCaml*, ocaml.org/.
[4] "Garbage Collection." *OCaml*, ocaml.org/learn/tutorials/garbage_collection.html.
[5] "Ocsigen." Multi-Tier Programming for Web and Mobile Apps, ocsigen.org/home/intro.html.
[6] "Indiana University Indiana University Indiana University." What Is the Difference between a Compiled and an Interpreted Program?, kb.iu.edu/d/agsz.
[7] "Kotlin (Programming Language)." Wikipedia, Wikimedia Foundation, 25 May 2019, en.wikipedia.org/wiki/Kotlin_(programming_language).
[8] Staff, - Webopedia. "Java Applet." What Is Java Applet? Webopedia Definition, www.webopedia.com/TERM/J/Java_applet.html.