Language Bindings for TensorFlow

Hermmy Wang
704978214

University of California, Los Angeles

Abstract

Language bindings are useful for programmers to export libraries written other languages to a language of their choice, so that these libraries are more versatile and portable while retaining enough efficiency and performance. TensorFlow, one of the most powerful machine learning libraries available in the industry, can be bind to other languages as well. To run TensorFlow on platforms other than Python or C++ and to avoid executing Python code being the bottleneck of the program's performance, we introduce language binding. The paper investigates three languages for TensorFlow bindings, Java, OCaml, and Kotlin. Specifically, the prototype using TensorFlow that we would like to support is a server proxy herd. Instead of writing the application in Python which takes too much time setting up the model, we use language binding allows implementing the application in other languages that outperform Python, meanwhile still remaining the access to the TensorFlow library. After exploring the advantages and disadvantages of binding to these three programming languages, I conclude that Kotlin is the most efficient choice for language binding of TensorFlow under this prototype due to its support of both imperative and functional language styles and many other features.

Introduction

Language binding refers to the service of providing application programming interface (API) from a given programming language to a library written in another language. Many useful libraries are written in C, C++, or some other low-level languages, while programmers often want to import these libraries to other languages such as Python, Java, OCaml, Kotlin, and many other high-level languages. Therefore, language bindings are needed for a more efficient, general, and flexible use of libraries.

In this paper, I will specifically look into a C++ or CUDA based library TensorFlow and investigate how other high-level languages such as Java, OCaml, and Kotlin can be bind to this library. Developed in 2015 by Google, TensorFlow is a powerful tool in the field of machine learning, deep neural networks and artificial intelligence. With TensorFlow, software engineers have developed millions of applications ranging from image processing to music producing. Besides its implementation in C++, TensorFlow offers extensible APIs in Python. The fundamental underlying data structure behind TensorFlow is a data flow graph in which the data flows through mathematical computations during a session. Since a large majority of data scientists and machine learning expert in the industry are using Python primarily, the developers of TensorFlow decides to choose Python as its most compatible language. In the following sections, I will introduce how to bind Java, OCaml, and Kotlin to TensorFlow and evaluate the language binds regarding their special features, support of event-driven server herd, and ease of use, flexibility, generality, performance, and reliability.

Java

TensorFlow can also be imported to Java via language binding. Java is an object-oriented, imperative language widely used in various platforms from embedded to mobile devices. Binding Java to TensorFlow needs to support at least the following functionality: running a predefined graph, constructing a graph, handling constants and optional parameters, computing gradients, implementing functions and control flow, and supporting a neural network library.
use

To use TensorFlow models on Java, we need a Python environment, Maven, TensorFlow source code, TensorFlow library jar file, and TensorFlow Java Native Interface (JNI) file. First, in order to build the source code, we need to utilize 'bazel'. Afterwards, we can build TensorFlow as a pip package by configuring and establishing dependencies on it; then, build the bindings via a similar process. JNI file is a so library file; Java runtime only needs to know the location of the library in order to run the application. Therefore, running the application in Maven can further simplify the dependencies. The installation instruction of using TensorFlow is simpler with fewer step. The library is essentially installed with pip command and is ready to use on the pre-installed Python environment. The high-level APIs of TensorFlow in Java are included the org.tensorflow package. The interfaces, classes, enums, and exceptions provided by this package enable Java to easily implement the functionalities such as TensorFlow graph construction and model loading with a few statements. Therefore, binding TensorFlow to Java retains enough readability and writability of code.

TensorFlow is flexible in the sense that programmers can easily utilize the multiple cores on the machine with a single API even they are using Python. Also, in TensorFlow 2.0, it has extended its flexibility by further supporting the high-level API of Keras, which is a used for prototyping, research, and production in machine learning. As the development team of TensorFlow introduces, the enhancements of TensorFlow 2.0 include "support for eager execution for intuitive debugging and fast iteration, support for the TensorFlow SavedModel model exchange format, and integrated support for distributed training, including training on TPUs". [9] In Java, TensorFlow is used with the JDK via JNI. Java is highly flexible in the sense that it is not limited by platforms. Whenever TensorFlow detects an error in executing TensorFlow graphs, it handles the error through throwing a TensorFlowException. Also, Java is a statically typed language, where type errors can be detected during compile time which saves more debugging time for Java programmers compared to Python programmers. The reliability of using TensorFlow in Java is a drawback of this type of language binding, as the documentation of building TensorFlow in Java clearly states that the package does not guarantee the reliability of these APIs.

Java is one of the first choices of programming languages to implement a client-server application especially for mobile devices largely due to its full support of mobile operating systems. Because of its great network capability, distributed computing is easier on Java compared to that on Python. Also, Java is more secure in the sense that the code is transformed into bytecodes before executing in the Java Virtual Machine.

Ocaml

Unlike Python or Java, OCaml is a functional language in which everything is bind to pure side-effect-free high-order functions and iterations are implemented via recursion. Besides, variables are immutable in functional language; while the feature maintains the stability of the program, it is a shortcoming with respect to memory and time performance.

Writing code in the form of pure functions greatly reduces the readability and writability of code. Although functional language often requires less lines of code compared to imperative language to achieve the same functionality, most programmers are more used to reading and writing in imperative language style due to its intuitiveness. Also, integrating functions with the rest of application and I/O operations is often a difficult task [12].
In order to bind OCaml with TensorFlow, we need to transform the C++-based implementation of TensorFlow to functional style.

The generality of OCaml is not as mature as imperative languages. Since OCaml is not a widely-used language in the field of machine learning, TensorFlow's support for it is not nearly as maintainable and pervasive as in Python. Also, debugging is painful for OCaml. Unlike Java or Python which can throw exceptions to tell where the code encounters a problem when applying TensorFlow library, OCaml does not provide a description of where the program goes wrong.

There are some advantages of writing programs in OCaml. Python is a dynamically typed language where type checking occurs during runtime. While providing more flexibilities of changing variable types and values during a program, dynamic type checking sacrifices run time performance as type errors might occur during run time and terminate the program. OCaml does type checking completely during compile time, which ensures that the program will not terminate abnormally during run time due to type errors.

Because Python code is interpreted during running time instead of being compiled to bytecode like Java, the performance is generally slower for Python. The situation is similar in the case of OCaml. OCaml is also transformed to native code during compile time. This nature of using bytecode interpreter can boost this language's simplicity, portability, and compilation speed [15].
implementation of the server herd

Kotlin

Kotlin, often described as an advanced version of Java, is an imperative language that is open source, runs on Java virtual machine, and is widely used in development Android applications. Besides similar syntax and style, Kotlin also provides more functionalities, improvements, and unique features that are not present in Java. These enhancements include lambda expressions and inline functions which is the essential feature of functional languages such as OCaml, smart casts, coroutines, type inference and many more. Since Kotlin possess both features of imperative language and functional language, it is outstanding in code conciseness, security, and easy debugging.

Language binding for TensorFlow to Kotlin is not currently available, but Kotlin/Native allows compilation in non-virtual-machine platforms such as embedded devices or iOS. Kotlin/Native supports interoperability to use static/dynamic C libraries and frameworks including TensorFlow.

To use TensorFlow in Kotlin, the installation instruction is similar but slightly easier to that in Java. We first need to install the TensorFlow binary source and then call Kotlin/Native to build bindings to the TensorFlow library file. Afterwards, we can directly use the compiler of Kotlin/Native to build an executable file. High-level APIs in Kotlin/Native TensorFlow library include functions that take status argument and are friendly for error checking. Just like in Java, there are classes available for building Tensor graphs, computing gradients or other operations, generating messages, and creating sessions, and so on. The APIs are concise and easy to use. Since Kotlin is a statically typed language, it adds type safety to TensorFlow and enhances run time performance for programmers. Also, Kotlin supports operator overloading and therefore reinforces the flexibility of TensorFlow.

Conclusion

The paper explores the features of three languages for TensorFlow bindings, Java, OCaml, and Kotlin. The prototype using TensorFlow that we would like to support is a server proxy herd. Instead of writing the application in Python which takes too much time setting up the model, we use language binding allows implementing the application in other languages that outperform Python, meanwhile still remaining the access to the TensorFlow library. After exploring the advantages and disadvantages of binding to these three programming languages, I conclude that Kotlin is the most efficient choice for language binding of TensorFlow under this prototype. Kotlin has extensible support for mobile devices as our server proxy want to accept messages from mobile clients as efficiently as possible. Kotlin also has features such as type checking during compile time and operator overloading that can further enhance the functionalities of TensorFlow. Although the binding is not directly available in Kotlin, Kotlin/Native can support TensorFlow in an alternative way. I believe it is promising for the development team of Kotlin and TensorFlow to support language binding for TensorFlow in Kotlin in the future.

Reference

[1] TensorFlow in other languages. https://www.tensorflow.org/guide/extend/bindings

[2] Language binding. https://en.wikipedia.org/wiki/Language_binding

[3] Brownlee, Jason. Introduction to the Python Deep Learning Library TensorFlow. https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/

[4] Singh Chaitanya. Introduction to Java programming. https://beginnersbook.com/2013/05/java-introduction/

[5] Chandrakant, Kumar. Introduction to TensorFlow for Java. https://www.baeldung.com/tensorflow-java

[6] Module: tf https://www.tensorflow.org/api_docs/java/reference/org/tensorflow/package-summary

[7] TensorFlow for Java. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/java

[8] Elizabeth, Jane. Top 5 machine learning frameworks for Java and Python. https://jaxenter.com/top-5-machine-learning-frameworks-140397.html

[9] Gupta, Sandeep. Standardizing on Keras: Guidance on High-level APIs in TensorFlow 2.0. https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a

[10] Java Advantages and Disadvantages. https://www.mindsmapped.com/java-advantages-and-disadvantages/

[11] Mazare, Laurent. Deep learning experiments in OCaml. https://blog.janestreet.com/deep-learning-experiments-in-ocaml/

[12] Functional Programming Paradigm. https://www.geeksforgeeks.org/functional-programming-paradigm/

[13] Why do hedge funds and financial services often use OCaml? https://stackoverflow.com/questions/1924367/why-do-hedge-funds-and-financial-services-often-use-ocaml

[14] Anil Madhavapeddy, Yaron Minsky. Hickey, Jason. Real World OCaml. https://v1.realworldocaml.org/v1/en/html/the-compiler-backend-byte-code-and-native-code.html

[15] Python, Scheme, and OCaml Speed Comparison. https://sixthhappiness.github.io/articles/python-scheme-and-ocaml-speed-comparison/index.html

[16] Kotlin vs. Python. https://www.slant.co/versus/110/1543/~python_vs_kotlin

[17] Kunze, Julius. TensorFlow in Kotlin/Native. https://juliuskunze.com/tensorflow-in-kotlin-native.html
[18] Wu, Teresa. TensorFlow Lite Image recognition: Android with Kotlin. https://medium.com/@teresa.wu/tensorflow-image-recognition-on-android-with-kotlin-cee8d977ae9
[19] Kotlin/Native for Native. https://kotlinlang.org/docs/reference/native-overview.html

Appendix

Source code for everyNth.kt

```
fun<T> everyNth(lst: List<T>, N:
Int) : List<T>{
    if (N <= 0 || N > lst.size) {
        println("Error: please
provide a valid N")
        return emptyList()
    }
    var ret = listOf<T>()
    for (i in 0..(lst.size-1)) {
        if ((i+1) % N == 0) {  //
every Nth element
            ret += lst[i]
        }
    }
    return ret
}
```