Alternative Programming Languages for TensorFlow Bindings

Jonathan Chu
University of California, Los Angeles

Abstract

In traditional machine learning applications, the performance bottleneck comes in training models on massive sets of data, a task that can require tremendous computing power. TensorFlow, one of the most popular machine learning libraries of today, has a Python API where variable bindings are executed using Python before training models in C++ or other languages. In an application where models are fairly simple and data sets are not too large, one could imagine that the Python bindings could become the bottleneck, as there could be more efficient bindings in other languages. This paper explores the advantages and disadvantages of choosing alternative languages to Python for TensorFlow Bindings: Java, OCaml, and Crystal. From this investigation, Java seems to be best choice for stability and efficiency, providing the performance of a compiled language, syntax of an imperative language, and maturity of a well-established, heavily utilized programming language.

1. Introduction

Extending the application proxy server herd model discussed previously, our new application must accept queries to train machine learning models using TensorFlow. It runs efficiently in Python for large queries that need to train fairly large models but bottlenecks running Python code on small queries, suggesting there could be ways to improve the efficiency of bindings. Perhaps the client-side performance could be improved by choosing a different programming language which would handle instructions and bindings from TensorFlow more efficiently.

Java, an object-oriented, imperative, language known for platform independence and multithreaded concurrency will be the first candidate for consideration. Next, OCaml, particularly the functional side of the language, will be the next language to be considered. Finally, Crystal, a new language featuring Ruby-like syntax at greater performance will be the last language considered.

2. TensorFlow and Python

TensorFlow is an open source library for numerical computation, including support for machine learning and deep learning[1]. Python, as it is known to developers today, supports a wide variety of applications and provides libraries and language features for web development, machine learning, scientific computing, game development, and much more. Python was the first client language supported by TensorFlow and currently has the most features supported[2].

2.1. Advantages of Python

While most of TensorFlow's computationally heavy code is written in C++ and CUDA, it follows intuitively that the client-side API was written for Python, given Python's established reputation in the computing community, especially among data scientists and machine learning practitioners. Python has a number of useful libraries for data preprocessing and visualization, including NumPy and Pandas, which along with TenslorFlow, would allow developers to conveniently package their code for every step of the pipeline into a single language.

There is extensive community support and documentation for TensorFlow in Python that would not be available if a new language were introduced. In addition, Python's built-in memory manager would keep track of TensorFlow objects and garbage collect automatically. Its dynamic typing would also allow for easy creation and handling of TensorFlow objects. Overall, Python's central advantage is its usability – its familiarity among programmers, straightforward syntax, and built-in libraries.

2.2. Disadvantages of Python

The disadvantages of using Python for any application exist in its TensorFlow client. High-level abstraction and simplified syntax come at the expense of performance, where Python clearly loses to its low-level counterparts C and C++ because of runtime type-checking and type-inference, garbage collection, and other runtime operations that produce additional overhead. As an interpreted language, Python is unable to perform compile-time optimizations and error-checking that other languages can, giving it a measured disadvantage in performance[3].

3. Java

Java is a multi-purpose programming language and computing platform known for its portability. A TensorFlow binding implementation does exist, as evident on the homework spec, but we will consider advantages and disadvantages of using Java as such.

3.1. Advantages of Java

Java code is typically translated into bytecode, from which it can be compiled into other formats. It is statically typed and runs mark-and-sweep garbage collection on objects. These aspects of Java already provide a performance advantage over Python which has overhead resulting from its interpreted nature. With this advantage, Java is inherently one step ahead of Python at achieving our goal of increasing binding speed.

Another notable feature of Java is its support for multithreading, as opposed to Python which does not support multithreaded execution as smoothly due to the Global Interpreter Lock[4]. If our application server herd receives many small requests from which it then has to generate machine learning models, multithreading could make binding faster by delegating tasks to different threads. With larger models, multithreading on the client-side would not improve performance, since the performance bottleneck on the backend would still remain untouched.

Although perhaps not as high-level as Python, Java is an imperative language with relatively straightforward syntax that is not too far off from the similarly imperative style of Python, so the transition between the two languages would not be too difficult for most developers that are already familiar with languages like Python and C++.

3.1. Disadvantages of Java

As it includes less abstractions than Python, a Java implementation would require more work and care on the developers' end keeping track of objects, their types, and the memory on which they live. Python abstracted these concerns away and took care of them with dynamic type-checking and a built-in memory manager. However, these disadvantages are features of the language that anyone with experience in C, C++, or another such language would have familiarity with and be able to handle.

4. OCaml

OCaml is a general-purpose programming language that supports object-orientation, functional programming, and an imperative style[5]. However, this section will focus more on the functional side of the language since that is what we explored in past lectures and projects.

4.1. Advantages of OCaml

Like Python, OCaml has a memory manager that makes creating and deleting objects painless for the programmer[6]. This could lead to less errors related to memory in the development process. Additionally, OCaml compiles to bytecode and has a strong type-checking system as opposed to Python which is dynamically typed and interpreted. OCaml's type-checking allows for better performance once compiled to machine code. Compile-time optimizations will also lead to improved parallelism and performance overall.

4.2. Disadvantages of OCaml

Because TensorFlow has more commonly been used in imperative languages like Python and C++, it would be difficult for the average programmer or data scientist to transition to a functional programming language. Object creation and assignment would certainly not be as straightforward as in an imperative language. The great majority of programmers today don't work with functional programming languages regularly and would have to become familiar with OCaml before being able to effectively apply it in any real-world setting. This would drastically increase the development time of the application.

Having a much stricter type system, OCaml may pose difficulties for programmers coming from a language like Python where types are so often modified and inferred. This would lead to further difficulty in the development process through unexpected errors.

A purely functional language would present further difficulty through a lack of certain data structures that would simplify the job of the programmer. Lists are built-in, but many common list methods one might expect in an imperative language are not provided, and it requires more careful thought to work with lists overall. Many of the data structures provided in OCaml, graphs being a prime example, that would be useful for the TensorFlow client are only available or much more straightforward in the imperative side of the language.

5. Crystal

Crystal is a compiled, object-oriented language which has appeared recently, within the past few years. It is statically typed and has a syntax similar to Ruby.

5.1. Advantages of Crystal

Crystal is "fast as C, slick as Ruby", according to its own website[8]. With languages like C/C++, we sacrifice simple syntax for efficiency and robustness of a compiled language. With Python, we trade efficiency for the simplified scripting language syntax that is easy for any programmer to pick up. Crystal combines the advantages of both and provides great efficiency with a syntax like Ruby, another popular scripting language. Crystal's TensorFlow client would almost certainly run faster than Python's, alleviating our bottleneck. Being compiled and object-oriented, this language has many of the performance advantages a language like C++ would have, including static typing and compile-time optimizations that allow for improved efficiency.

Although there are some notable differences between the syntax of Ruby and that of Python, they are fairly similar, and a Python developer would certainly be able to transition to Crystal fairly easily.

5.2. Disadvantages of Crystal

Because it is very new, there isn't a lot of community support for Crystal developers and there's limited availability of Shards, libraries in Crystal. It is still under heavy development and may still experience breaking changes at any time, making development less stable[9]. There isn't a well-developed IDE for Crystal, although packages for editors like Sublime and Atom for Crystal do exist. For more heavy and frequent computations, Java's multithreading would provide greater efficiency than Crystal's use of green threads, which provide concurrency but on the user level, unable to make full use of a multicore machine.

Although its syntax and efficiency show considerable promise, Crystal is still in its own development process and would not be the most stable choice of language for the application, especially if it is to be released soon with the possibility of breaking updates to Crystal.

6. Conclusion

To address the performance issue of the TensorFlow client in Python, it would certainly be beneficial to move to a language that would allow faster bindings. Any of the three languages explored, Java, OCaml, and Crystal, would provide better performance than Python. However, because development would be much more straightforward with an imperative language, the functional subset of OCaml would not be an ideal choice. Crystal, while showing promise with its efficiency and syntax, is still in development, and it may be safer to use a better-established language for production. Java is a great balance, holding many of the advantages discussed, including performance, syntax, and usability. Between Python, Java, OCaml, and Crystal, Java is best suited for our application.

7. References

[1] "TensorFlow." TensorFlow, www.tensorflow.org/.
[2] "TensorFlow in Other Languages | TensorFlow." TensorFlow, www.tensorflow.org/extend/language_bindings.
[3] "Why is Python slower than the xxx language." ForLoop - Python Wiki, wiki.python.org/moin/Why%20is%20Python%20slower%20than%20the%20xxx%20language.
[4] "Global Interpreter Lock." ForLoop - Python Wiki, wiki.python.org/moin/GlobalInterpreterLock.
[5] "OCaml." OCaml, ocaml.org/.
[6] "What Is OCaml?" OCaml, ocaml.org/learn/description.html.
[7] "OCamlgraph." Ocamlgraph: an Ocaml Graph Library, ocamlgraph.lri.fr/index.en.html.
[8] "Crystal." The Crystal Programming Language, crystal-lang.org/.
[9]crystal-lang. "Crystal-Lang/Crystal." GitHub, github.com/crystal-lang/crystal.