
KOTLIN ∇

A SHAPE-SAFE DSL FOR DIFFERENTIABLE FUNCTIONAL PROGRAMMING

Breandan Considine¹ Liam Paull¹ Michalis Famelis²

ABSTRACT

Kotlin is a statically-typed programming language with support for embedded domain specific languages, asynchronous programming, and multi-platform compilation. In this work, we present an algebraically-grounded implementation of forward- and reverse- mode automatic differentiation (AD) with shape-safe tensor operations, written in pure Kotlin. Our approach differs from existing AD frameworks in that Kotlin ∇ is the first shape-safe AD library fully compatible with the Java type system, requiring no metaprogramming, reflection or compiler intervention to use. A working prototype is available at: <https://github.com/breandan/kotlingrad>.

1 INTRODUCTION

Prior work has shown it is possible to encode a deterministic context-free grammar as a *fluent interface* (?) in Java. This result was strengthened to prove Java’s type system is Turing complete (?). As a practical consequence, we can use the same technique to perform shape-safe automatic differentiation (AD) in Java, using type-level programming. A similar technique is feasible in any language with generic types. We use *Kotlin*, whose type system is less expressive, but fully compatible with Java.

Differentiable programming has a rich history among dynamic languages like Python, Lua and JavaScript, with early implementations including projects like Theano, Torch, and TensorFlow. Similar ideas have been implemented in statically typed, functional languages, such as Haskell’s Stalin ∇ (?), DiffSharp in F# (?) and recently Swift 2018tensorflow. However, the majority of existing automatic differentiation (AD) frameworks use a loosely-typed DSL, and few offer shape-safe tensor operations in a widely-used programming language.

Existing AD implementations for the JVM include Lantern (?), Nexus (?) and DeepLearning.scala (?), however these are Scala-based and do not interoperate with other JVM languages. Kotlin ∇ is fully interoperable with vanilla Java, enabling broader adoption in neighboring languages. To our knowledge, Kotlin has no prior AD implementation. However, the language contains a number of desirable features for implementing a native AD framework. In addition

to type-safety and interoperability, Kotlin ∇ primarily relies on the following language features:

1. **Operator overloading and infix functions** allow a concise notation for defining arithmetic operations on tensor-algebraic structures, i.e. groups, rings and fields.
2. **λ -functions and coroutines** support backpropagation with lambdas and shift-reset continuations, following ? and ?.
3. **Extension functions** support extending classes with new fields and methods and can be exposed to external callers without requiring sub-classing or inheritance.

2 USAGE

Kotlin ∇ allows users to implement differentiable programs by composing operations on fields to form algebraic expressions. Expressions are lazily evaluated inside a numerical context, which may be imported on a per-file basis or lexically scoped for finer-grained control over the runtime behavior.

Figure 1. A basic Kotlin ∇ program with two inputs and one output.

Above, we define a function with two variables and take a series of partial derivatives with respect to each variable. The function is numerically evaluated on the interval $(-1, 1)$ in each dimension and rendered in 3-space. We can also plot higher dimensional manifolds (e.g. the loss surface of a neural network), projected into four dimensions, and rendered in three, where one axis is represented by time. To demonstrate, a display is required for animation purposes. Kotlin ∇ treats mathematical functions and programming functions with the same underlying abstraction. Expressions are composed recursively to form a data-flow graph (DFG). An expression is simply a `Function`, which is only evaluated once invoked with numerical values, e.g. `z(0, 0)`. Kotlin ∇ supports shape-safe tensor operations by encoding tensor rank as a parameter of the operand’s type signature. By enumerating type-level integer literals, we can define

¹Mila, Université de Montréal ²Université de Montréal. Correspondence to: Breandan Considine <bre@ndan.co>.

$$z = \sin(10(x * x + y^2))/10, \text{plot}\left(\frac{\partial^3 z}{\partial x^2 \partial y}\right)$$

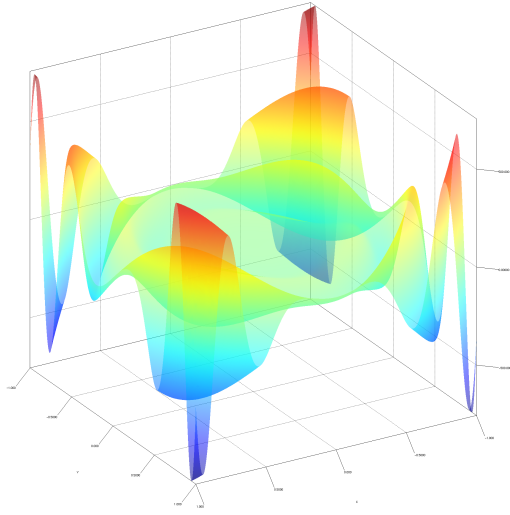


Figure 2. Output generated by the program shown in Figure 1.

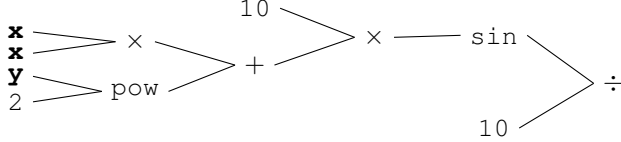


Figure 3. Implicit DFG constructed by the original expression, z .

tensor operations just once using the highest literal, and rely on Liskov substitution to preserve shape safety for subtypes.

Figure 4. Shape safe tensor addition for rank-1 tensors, $\forall L \leq 2$.

It is possible to enforce shape-safe vector construction as well as checked vector arithmetic up to a fixed L , but the full implementation is omitted for brevity. A similar pattern can be applied to matrices and higher rank tensors, where the type signature encodes the shape of the operand at runtime.

With these basic ingredients, we have almost all the features necessary to build an expressive shape-safe AD, but unlike prior implementations using Scala or Haskell, in a language that is fully interoperable with Java, while also capable of compiling to JVM bytecode, JavaScript, and native code.

In future work, we intend to implement a full grammar of differentiable primitives including matrix convolution, control flow and recursion. While Kotlin ∇ currently implements arithmetic manually, we plan to wrap a BLAS such as cuBLAS or native linear algebra library for performance.

REFERENCES

- Baydin, A. G., Pearlmutter, B. A., and Siskind, J. M. DiffSharp: Automatic differentiation library. *CoRR*, abs/1511.07727, 2015. URL <http://arxiv.org/abs/1511.07727>.
- Bo, Y. DeepLearning.scala: A simple library for

creating complex neural networks. 2018. URL <https://github.com/ThoughtWorksInc/DeepLearning.scala>.

Chen, T. Typesafe abstractions for tensor operations (short paper). pp. 45–50, 2017. doi: 10.1145/3136000.3136001. URL <http://doi.acm.org/10.1145/3136000.3136001>.

Gil, Y. and Levy, T. Formal Language Recognition with the Java Type Checker. 56:10:1–10:27, 2016. ISSN 1868-8969. doi: 10.4230/LIPIcs.ECOOP.2016.10. URL <http://drops.dagstuhl.de/opus/volltexte/2016/6104>.

Grigore, R. Java generics are Turing Complete. pp. 73–85, 2017. doi: 10.1145/3009837.3009871. URL <http://doi.acm.org/10.1145/3009837.3009871>.

Pearlmutter, B. A. and Siskind, J. M. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7, 2008a.

Pearlmutter, B. A. and Siskind, J. M. Using programming language theory to make automatic differentiation sound and efficient. pp. 79–90, 2008b. ISSN 1439-7358. doi: 10.1007/978-3-540-68942-3_8. URL <http://www.bcl.hamilton.ie/~barak/papers/sound-efficient-ad2008.pdf>.

Wang, F., Wu, X., Essertel, G. M., Decker, J. M., and Rompf, T. Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *CoRR*, abs/1803.10228, 2018. URL <http://arxiv.org/abs/1803.10228>.