Kotlin∇

Differentiable functional programming with algebraic data types

Breandan Considine
University of Montréal
breandan.considine@umontreal.ca

ABSTRACT

Kotlin is a multi-platform programming language with compiler support for JVM, JS and native targets. The language emphasizes static typing, null-safety and interoperability with Java and JavaScript. In this work, we present an algebraically grounded implementation of forward and reverse mode automatic differentiation written in pure Kotlin and a property-based test suite for soundness checking. Our approach enables users to target multiple platforms through a single codebase and receive compile-time static analysis. A working prototype is provided at: https://github.com/breandan/kotlingrad.

1 INTRODUCTION

Differentiable programming has a long history among dynamic languages like Python, Lua and JavaScript, with early implementations including projects like Theano, Torch, and TensorFlow. More recently, their innovations have spread into statically typed, functional languages, such as Haskell (e.g. Stalin∇ [1]) and F# (e.g. DiffSharp [2]). However the majority of existing automatic differentiation (AD) frameworks still require using an embedded DSL, and comparatively few offer native compiler support in a general-purpose, statically typed programming language.

Prior AD implementations for the JVM include COJAC [6], ADiJaC [7] and DeepLearning4j [8], however these implementations either rely on a custom compiler toolchain, are limited to a fixed computation graph, or are intended for a domain specific audience. In contrast, our approach can be applied to native Kotlin, is not tied to the JVM and supports a broad set of use cases, enabling a greater degree of safety, flexibility and expressiveness.

2 BACKGROUND

To our present knowledge, Kotlin has no native AD implementation. However it does have a number of desirable features for implementing such a framework, including tooling and multi-platform support. In addition, Kotlin∇ relies on three primary language features:

(1) **Operator overloading** enables a concise notation for arithmetical operations on algebraic types.
(2) **Algebraic data types** supports the type-safe construction of computation graphs, via sealed classes.
(3) **Extensions with receivers** allows extending standard library types with new methods and exposing them to users without requiring sub-classing or inheritance.

Currently, we support common numerical operations over vector fields, and are working to add further support for subgradient and subderivative methods for non-differentiable functions. We use a property based testing technique adapted from jAlgebra to perform simple property based tests for algebraic soundness checking.

3 USAGE

```
import co.ndan.kotlingrad.math.types.Double
import co.ndan.kotlingrad.math.algebra.DoublePrototype
import co.ndan.kotlingrad.math.calculus.RealFunctor
import co.ndan.kotlingrad.math.types.*

val rft = RealFunctor(DoublePrototype)
val x = rft.variable("x", Double(0))
val y = rft.variable("y", Double(1))

val z = 2 * x * (-sin(x * y) + y)
val dz_dz = d(z) / d(x) // Leibniz notation
val dz_dy = d(z) / d(y) // Multiple variables
val d2z_dx2 = d(dz_dx) / d(x)  // Higher order and
val d2z_dxdy = d(dz_dx) / d(y) // partial derivatives
```

4 WORK IN PROGRESS

Kotlin∇ borrows heavily from prior work in functional [1], type-safe [2] differentiable programming [5]. Through a combination of JVM bytecode and source code analysis, we are able to partially recover symbolic expressions for the gradient as in [6]. Adapting from Breuleux et al. [4], we can perform reverse-mode AD on a universal abstract syntax tree (UAST). And applying Elliot's framework of generalized linear maps [3] to Kotlin's type system, we can support shape-inference and make simple variable substitutions.

While this project is currently still in its infancy, our goal is to utilize Kotlin's compiler plugin framework and existing multi-platform infrastructure to implement truly native, differentiable programming in a type-safe environment. This will allow users to perform gradient-based computation using a single codebase on multiple architectures and platforms, while leveraging the existing tools and libraries from the JVM ecosystem.

REFERENCES

[1] Barak A. Pearlmutter and Jeffrey Mark Siskind, Reverse-Mode AD in a functional framework: Lambda the ultimate backpropagator. TOPLAS 30(2):1-36, Mar 2008, doi:10.1145/1330017.1330018.
[2] Atilim Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind (2015) Automatic differentiation and machine learning: a survey. arXiv preprint. arXiv:1502.05767
[3] Conal Elliott. The Simple Essence of Automatic Differentiation. Proc. ACM Program. Lang., 2(ICFP):70:1âĂŞ70:29, July 2018.
[4] Olivier Breuleux and Bart van MerriÃńnboer. Automatic differentiation in Myia. In NIPS AutoDiff Workshop, 2017.
[5] Fei Wang, Xilun Wu, Gregory Essertel, James Decker, and Tiark Rompf. Demystifying Differentiable Programming: Shift/Reset the Penultimate Backpropagator.
[6] Monnard, Romain. COJAC Enriched Numbers. Diss. 2014.
[7] Slusanschi, Emil Dumitrel, Vlad. (2016). ADiJaC – Automatic Differentiation of Java Classfiles. ACM Transactions on Mathematical Software. 43. 1-33.
[8] Eclipse Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the JVM. http://deeplearning4j.org
[9] George, M.D. jAlgebra, An abstract algebra library for Java. GitHub repository. https://github.com/mdgeorge4153/jalgebra