Differentiable Programming: Several Directions

Supervisor: Luke Ong

Optimisation problems are ubiquitous in Computer Science, with applications ranging from neural network learning to computer vision. A widely used algorithm for solving optimisation problems is *gradient descent*, which uses the gradient of a differentiable function at a point to determine how much the function's inputs should be adjusted.

Differentiable programming is about the development of a programming system with a built-in gradient operation for solving optimisation problems. Automatic differentiation systems, and more generally, differentiable programming, have become extremely important recently because they are the computational foundation of deep learning.

A key desideratum is that the constructible programs should denote differentiable functions. We focus on imperative programs, so as to benefit from their expressiveness and ease-of-use when creating models of complex systems that need to be optimised.

Unfortunately imperative programs are not always differentiable. For example, conditional statements, which are a key feature of imperative programming languages, introduce branch points – discontinuities where the branch taken changes – which are either ignored (as nondifferentiable) or whose gradients are given as intervals by traditional automated differentiation systems.

This project is concerned with approaches to the problem of optimising imperative programs by *transforming* them to differentiable functions on which gradient descent can safely operate.

Possible directions

- Formulation of a first-order imperative programming language with a reverse differentiation operator.
- Definition of its denotational semantics, and a source code transformation.
- Approximation by simulated annealing: by gradually creating stricter differentiable approximations of the program that condition on comparisons of values, apply gradient descent to each approximation. Design and implement an algorithmic solution.

References

[1] Swarat Chaudhuri and Armando Solar-Lezama. Smooth interpretation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 279–291, 2010.

[2] U. Naumann. *The art of differentiating computer programs: an introduction to algorithmic differentiation*. Society for Industrial and Applied Mathematics, 2011.

[3] Gordon D. Plotkin. Some principles of differentiable programming languages. ACM POPL Plenary Lecture, 2018.

[4] Stan Development Team. nomad: A high-peformance automatic differentiation package.