Gradient-based Hyperparameter Optimization through Reversible Learning: Appendix

Dougal Maclaurin[†] MACLAURIN@PHYSICS.HARVARD.EDU
Harvard University, Cambridge, Massachusetts

David Duvenaud[†] DDUVENAUD@SEAS.HARVARD.EDU
Harvard University, Cambridge, Massachusetts

Ryan P. Adams RPA@SEAS.HARVARD.EDU
Harvard University, Cambridge, Massachusetts

Forward vs. reverse-mode differentiation

By the chain rule, the gradient of a set of nested functions is given by the product of the individual derivatives of each function:

$$\frac{\partial f_4(f_3(f_2(f_1(x))))}{\partial x} = \frac{\partial f_4}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial x}$$

If each function has multivariate inputs and outputs, the gradients are Jacobian matrices.

Forward and reverse mode differentiation differ only by the order in which they evaluate this product. Forward-mode differentiation works by multiplying gradients in the same order as the functions are evaluated:

$$\frac{\partial f_4(f_3(f_2(f_1(x))))}{\partial x} = \frac{\partial f_4}{\partial f_3} \cdot \left(\frac{\partial f_3}{\partial f_2} \cdot \left(\frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial x} \right) \right)$$

Reverse-mode multiplies the gradients in the opposite order, starting from the final result:

$$\frac{\partial f_4(f_3(f_2(f_1(x))))}{\partial x} = \left(\left(\frac{\partial f_4}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_2} \right) \cdot \frac{\partial f_2}{\partial f_1} \right) \cdot \frac{\partial f_1}{\partial x}$$

In an optimization setting, the final result of the nested functions, f_4, is a scalar, while the input x and intermediate values, $f_1 - f_3$, can be vectors. In this scenario the advantage of reverse-mode differentiation is very clear. Let's imagine that the dimensionality of all the intermediate vectors is D. In reverse mode, we start from the (scalar) output, and multiply by the next $D \times D$ Jacobian at each step. The value we accumulate is just a D-dimensional vector. In forward mode, however, we must accumulate an entire $D \times D$ matrix at each step. But do we have still have to compute and instantiate the $D \times D$ Jacobian matrices themselves either way? In general, yes. But in the (common) case that the vector-to-vector functions are either elementwise operations or (reshaped) matrix multiplications, the Jacobian matrices can actually be very sparse, and multiplication by the Jacobian can be performed efficiently without instantiation (Pearlmutter & Siskind, 2008).

The main drawback of reverse-mode differentiation is that intermediate values must be maintained in memory during the forward pass. In sections 2.1 and 2.3, we show how to drastically reduce the memory requirements of reverse-mode differentiation when differentiating through the entire learning procedure.

References

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