Taylor-made higher-order automatic differentiation

Matthew J. Johnson[1], Jesse Bettencourt[2],
Dougal Maclaurin[1], and David Duvenaud[2]

[1]Google Brain
[2]University of Toronto

Abstract

One way to implement higher-order automatic differentiation (AD) is to start with first-order AD and apply it repeatedly. That works, but for some specific cases, like evaluating a Taylor series, it can result in combinatorial amounts of redundant computation. This note describes a more efficient method, already known but with a new presentation, and its implementation in JAX.

1 Introduction

Consider the problem of evaluating the Kth-order Taylor approximation to a function f around a point x and with an offset v:

$$f(x+v) \approx f(x) + \partial f(x)[v] + \tfrac{1}{2!}\partial^2 f(x)[v,v] + \cdots + \tfrac{1}{K!}\partial^K f(x)[v,\ldots,v]. \tag{1}$$

For a multi-linear function like $\partial^k f(x)$ we use square brackets to denote its application to a list of k separate vectors, like $\partial^k f(x)[v_1,\ldots,v_k]$.

A formula like this one involves contracting against many copies of the same perturbation (tangent) vector v, with terms like

$$\partial^k f(x)[\underbrace{v,v,\ldots,v}_{k \text{ times}}] = \partial^k f(x)[\underbrace{v \otimes v \otimes \cdots \otimes v}_{k \text{ times}}] = \partial^k f(x)[v^{\otimes k}], \tag{2}$$

where we abuse notation to apply $\partial^k f(x)$ alternatively to a tensor product of vectors. Compare that to the more generic term

$$\partial^k f(x)[v_1, v_2, \ldots, v_k] = \partial^k f(x)\big[\textstyle\bigotimes_{i\in[k]} v_i\big], \tag{3}$$

where $v_1,\ldots,v_k$ may be distinct vectors and $[k] := \{1,2,\ldots,k\}$. (Since $\partial^k f(x)$ is symmetric under permuations of its arguments, the tensor product order doesn't matter.) Our goal is to find computational savings for the special case of (2) relative to the generic case of (3).

2 Function composition

Automatic differentiation is about function composition. Consider a composite function $f = g \circ h$. We'd like to understand how to express quantities like $\partial^k f(x)[v_1, \ldots, v_k]$ in terms of intermediate quantities $\partial^n h(x)[\cdots]$ and the functions $\partial^m g(x)$.

Start with $k = 1, 2, 3$ for concreteness:

$$\partial f(x)[v_1] = \partial g(h(x))\,[\partial h(x)[v_1]]\,. \tag{4}$$

$$\partial^2 f(x)[v_1, v_2] = \partial^2 g(h(x))\,[\partial h(x)[v_1], \partial h(x)[v_2]] + \partial g(h(x))\,\big[\partial^2 h(x)[v_1, v_2]\big]\,. \tag{5}$$

$$\begin{aligned}\partial^3 f(x)[v_1, v_2, v_3] = {}& \partial^3 g(h(x))\,[\partial h(x)[v_1],\, \partial h(x)[v_2],\, \partial h(x)[v_3]] \\ &+ \partial^2 g(h(x))\,\big[\partial^2 h(x)[v_1, v_3],\, \partial h(x)[v_2]\big] \\ &+ \partial^2 g(h(x))\,\big[\partial h(x)[v_1],\, \partial^2 h(x)[v_2, v_3]\big] \\ &+ \partial^2 g(h(x))\,\big[\partial^2 h(x)[v_1, v_2],\, \partial h(x)[v_3]\big] \\ &+ \partial g(h(x))\,\big[\partial^3 h(x)[v_1, v_2, v_3]\big]\end{aligned} \tag{6}$$

In Eq. (6) we can see a pattern emerging: each term corresponds to a partition of the set $\{1, 2, 3\}$, with the differentiation order m in each $\partial^m g(h(x))$ and n in each $\partial^n h(x)[\cdots]$ determined by the sizes of the partition and the sizes of the partition elements, respectively. By induction, in general we have

$$\partial^k f(x)[v_1, \ldots, v_k] = \sum_{\sigma \in \mathrm{part}([k])} \partial^{|\sigma|} g(h(x))\big[\textstyle\bigotimes_{\eta \in \sigma} \partial^{|\eta|} h(x)\big[\bigotimes_{\ell \in \eta} v_\ell\big]\big], \tag{7}$$

where $\mathrm{part}([k])$ denotes the set partitions of $\{1, 2, \ldots, k\}$ (so that σ is a set of sets), η ranges over the partition elements (each a set of integers), and ℓ ranges over integers.

Now consider when all the perturbation vectors are the same. Going back to the example in (6), if we set $v_1 = v_2 = v_3 = v$ we can collect the middle three terms:

$$\begin{aligned}\partial^3 f(x)[v, v, v] = {}& \partial^3 g(h(x))\,[\partial h(x)[v],\, \partial h(x)[v],\, \partial h(x)[v]] \\ &+ 3\partial^2 g(h(x))\,\big[\partial h(x)[v],\, \partial^2 h(x)[v, v]\big] \\ &+ \partial g(h(x))\,\big[\partial^3 h(x)[v, v, v]\big]\,.\end{aligned} \tag{8}$$

In general, how many like terms can we collect? It corresponds to this counting problem: given a set of k elements (the perturbation vectors), in how many ways can we collect them into m groups (for the arguments to $\partial^m g(h(x))$) using $n_1 \geq 0$ groups of size 1 (where n_1 is the number of $\partial^1 h(x)[v]$ terms), n_2 groups of size 2 (n_2 the number of $\partial^2 h(x)[v, v]$ terms), n_3 groups of size 3, etc.? The answer is

$$\mathrm{sym}(\nu) = \binom{k}{\underbrace{1, \ldots, 1}_{n_1 \text{ times}},\, \underbrace{2, \ldots, 2}_{n_2 \text{ times}},\, \ldots} \frac{1}{n_1!\, n_2! \cdots n_k!}, \tag{9}$$

where $\nu = (\nu_1, \nu_2, \ldots, \nu_m)$ here represents an integer partition of k as a (sorted) tuple of positive integers with $\sum_i \nu_i = k$ and with n_i denoting the number of times i appears in ν. The first term in (9) is a multinomial coefficient, counting how many ways to collect k

distinct items into the given numbers and sizes of distinct bins, and the second term corrects for treating bins of the same size as distinct (thus counting un-distinguished groups).

Putting these pieces together, we have the Faà di Bruno formula,

$$\partial^k f(x)[v, \ldots, v] = \sum_{\nu \in \text{part}(k)} \text{sym}(\nu) \partial^{|\nu|} g(h(x)) \big[\bigotimes_{n \in \nu} \partial^n h(x)[v^{\otimes n}] \big], \tag{10}$$

where $\text{part}(k)$ denotes the integer partitions of the integer k (rather than set partitions as in (7)).

3 Composition rule for Taylor series terms

First-order automatic differentiation solves a composition problem: for a function $f = g \circ h$, given a pair $(z_0, z_1) = (h(x), \partial h(x)[v])$, compute the pair $(f(x), \partial f(x)[v])$. We solve this problem using the composition rule

$$f(x) = g(h(x)) = g(z_0), \tag{11}$$
$$\partial f(x)[v] = \partial g(h(x))[\partial h(x)] = \partial g(z_0)[z_1]. \tag{12}$$

We implement this rule for every primitive function g and thus can differentiate composite functions f.

The higher-order analogue is: given a tuple

$$(z_0, z_1, z_2, \ldots, z_K) = \big(h(x), \quad \partial h(x)[v], \quad \tfrac{1}{2}\partial^2 h(x)[v, v], \quad \ldots \quad \tfrac{1}{K!}\partial^K h(x)[v, \ldots, v]\big), \tag{13}$$

compute the tuple

$$(w_0, w_1, w_2, \ldots, W_K) = \big(f(x), \quad \partial f(x)[v], \quad \tfrac{1}{2}\partial^2 f(x)[v, v], \quad \ldots \quad \tfrac{1}{K!}\partial^K f(x)[v, \ldots, v]\big). \tag{14}$$

We can solve this problem using (10), expressing each component as

$$w_k = \tfrac{1}{k!}\partial^k f(x)[v, \ldots, v] = \sum_{\nu \in \text{part}(k)} \text{sym}'(\nu) \partial^{|\nu|} g(z_0) \big[\bigotimes_{n \in \nu} z_n \big], \tag{15}$$

$$\text{sym}'(\nu) = \frac{1}{n_1! \, n_2! \cdots n_k!}. \tag{16}$$

Some of the factorials cancel by including them in the definition of the tuples in (13) and (14).

The Python code implementing (15) may be more legible than the math notation:

```
def sym(nu):
  return prod(fact(count) for _, count in Counter(nu).items())

def prop(derivs, terms):
  return [sum(derivs[len(nu)-1]([terms[i-1] for i in nu]) / sym(nu)
              for nu in partitions(k))
          for k in range(1, len(terms) + 1)]
```

Adding and subtracting 1 in places just has to do with off-by-one indexing issues.

The number of integer partitions grows like $\Theta\big(\exp(\sqrt{k})\big)$, so direct evaluation of the formula (15) reduces the computational complexity of higher-order differentiation for computing all the terms in a Taylor series from exponential $O\big(2^K\big)$ to subexponential $O\big(\exp(\sqrt{K})\big)$, but not further. However, the combinatorial structure of the formula enables more efficient evaluation via a recurrence.

4 Reducing complexity for common primitives to $O(K^2)$

One simple case where we can reduce the complexity in (15) is when the function g has some derivatives that are zero. For example, when g is linear (or bilinear) so that $\partial^k g(x) \equiv 0$ for $k > 1$, then the complexity of evaluating (15) for $k = 1, 2, \ldots, K$ is $O\big(K\big)$. This case handles all the common bilinear operations on arrays (contracion, convolution, etc.).

What about element-wise unary nonlinear functions, like exp and sin, which have no zero derivatives? For these we can exploit the fact that they satisfy first-order linear ordinary differential equations (ODEs). The basic idea is that while Eq. (15) generates a system of equations, expressing each w_k in terms of $z_{j\leq k}$, the expressions in $z_{j\leq k}$ get large. However, if g satisfies an ODE, that produces additional constraints on the Taylor coefficients. We can use those constraints to write equations for w_k in terms of both $z_{j\leq k}$ and $w_{i<k}$.

For example, $g = \exp$ has no zero derivatives but satisfies $g'(y) = g(y)$, where we write $g'(y) = \partial g(y)[1]$ for the scalar (element-wise) case. To see the constraints among the Taylor series coefficients, we substitute the ODE in the chain rule statement

$$f'(x) = g'(h(x))h'(x) = g(h(x))h'(x) = f(x)h'(x). \tag{17}$$

Substituting the Taylor expansions

$$f(x) = \sum_{k=0}^{\infty} w_k x^k, \qquad g(x) = \sum_{k=0}^{\infty} z_k x^k, \tag{18}$$

we get

$$\sum_{k=1}^{\infty} k\, w_k\, x^{k-1} = \left(\sum_{k=0}^{\infty} w_k\, x^k\right)\left(\sum_{k=1}^{\infty} k\, z_k\, x^{k-1}\right), \tag{19}$$

and equating coefficients we get

$$k\, w_k = \sum_{\ell=1}^{k} \ell\, w_{k-\ell}\, z_\ell, \qquad k = 1, 2, \ldots, K. \tag{20}$$

With these equations we can solve for the w_k in time $O(K^2)$. These equations hold true element-wise for the vector case because we can use the expansions

$$f(x + tv) = \sum_{k=0}^{\infty} \tfrac{1}{k!}\partial^k f(x)[v^{\otimes k}]t^k = \sum_{k=0}^{\infty} w_k t^k, \quad g(x + tv) = \sum_{k=0}^{\infty} \tfrac{1}{k!}\partial^k g(x)[v^{\otimes k}]t^k = \sum_{k=0}^{\infty} z_k t^k.$$

Slightly more generally, Proposition 13.1 in Griewank and Walter follows this reasoning to provide the result that if g satisfies the first-order linear ODE

$$b(y)g'(y) = a(y)g(y) + c(y), \tag{21}$$

then the coefficients satisfy

$$k\, w_k = \frac{1}{b_0}\left[\sum_{j=1}^{k} j\,(c_{k-j}e_{k-j})\,z_j - \sum_{j=1}^{k-1} j\,b_{k-j}w_j\right], \quad k = 1, 2, \ldots, K, \tag{22}$$

$$e_k = \sum_{j=0}^{k} a_j w_{k-j}, \quad k = 1, 2, \ldots, K-1. \tag{23}$$