An Overview of High Order Reverse Mode

Mu Wang[*]
wang970@purdue.edu

Alex Pothen
Department of Computer Science
Purdue University, West Lafayette, IN 47906
apothen@purdue.edu

Abstract

Automatic Differentiation (AD) is increasingly an important component of Machine Learning (ML) packages. For evaluating the gradient, the first order reverse mode also known as back-propagation, is optimal and is widely used. However, the functionalities of current mainstream ML packages for evaluating second and higher order derivatives are limited. One reason is that high order derivatives are computed using an overlay of first order forward and reverse modes. Here we describe an algorithm and code that directly implements the high order reverse mode, called ReverseAD. It is more efficient than previous methods, especially when evaluating sparse derivatives.

1 High Order Reverse Mode of Automatic Differentiation

1.1 Background

Solving machine learning (ML) problems often requires evaluating derivatives for finding the optimal parameters of the model. Back-propagation, which is the Reverse Mode Automatic Differentiation algorithm for computing first order derivatives, evaluates the gradient with theoretically optimal cost, and it has been widely implemented [6]. However, although in many ML problems ([4, 5]) one would get faster convergence to a local optimum with second order methods, few ML codes compute Hessians directly; and if Hessians are evaluated, they are computed indirectly as Hessian vector products. Although Hessian vector evaluation suffices to solve linear systems w.r.t the Hessian matrix, it would be more efficient to compute Hessian matrices directly if a large number of iterations are required in the optimization. If the Hessian is sparse, then computing and storing the Hessian is often feasible, even for large problems.

In this abstract, we describe a new reverse mode algorithm for computing second and higher order derivatives, which is a generalization of the back-propagation algorithm. Restricting the algorithm to second order gives an algorithm which evaluates the Hessian matrix directly. More intriguingly, as we will show in the next section, the high order reverse mode can detect the sparsity structure of the derivatives automatically, and use it to obtain efficient algorithms. We have developed a prototype code `ReverseAD` in C++ that is efficient for computing gradients, sparse Hessians, sparse Hessian vector products, and higher order derivative tensors as well as tensor vector products, and it is available at `https://github.com/wangmu0701/ReverseAD`.

[*]Graduated with Ph.D in August, 2017 under supervisor Alex Pothen.

1.2 Notations

Let $y = f(x)$, $x \in \mathbb{R}^n$ and $y \in \mathbb{R}$ denote a scalar objective function of real values. Following standard notation in AD, let $v_{1-n}, \cdots, v_0$ denote the *independent variables* (x); the execution of the objective function can be decomposed as:

$$\textbf{for } k = 1, 2, \cdots, l \qquad v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}},$$

where each $v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$ represents an elementary function, and $v_i \prec v_k$ denotes that variable v_k directly *depends* on variable v_i [2]. The evaluation of the objective function is equivalent to evaluating the sequence of elementary functions $\varphi_k, k = 1, \cdots, l$.

During the function evaluation, we call a variable as *live* if and only if it holds a value that *will be* used in the future. The live variable set S_k before the evaluation of $v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$ is:

$$\begin{aligned} S_k &= \{v_i : \exists v_j,\ j \geq k \text{ and } v_i \prec v_j\} \\ &= \{v_{1-n}, \cdots, v_{k-1}\} \bigcap \Big(\cup_{j \geq k} \{v_i : v_i \prec v_j\} \Big). \end{aligned}$$

Here S_k is the live variable set at the point that φ_k is about to be evaluated, i.e., when φ_{k-1} is the last evaluated elementary function. At the beginning of the function evaluation, the live variable set S_1 consists of all the independent variables. At the end of the evaluation, the live variable set S_{l+1} consists only of the dependent variable.

The liveness analysis in AD is a simplified version of the "backward may" analysis in compilers. The algorithm begins with $S_{l+1} = \{v_l\}$, and computes S_k in the order $k = l, l-1, \cdots, 1$ according to the equation:

$$S_k = S_{k+1} \setminus \{v_k\} \cup \{v_i : v_i \prec v_k\}. \tag{1}$$

Based on the concept of live variables, the objective function $f(\mathbf{x})$ can be written as a composite function:

$$f = \Phi_l \circ \Phi_{l-1} \circ \cdots \circ \Phi_1, \quad \text{where } \Phi_k : S_k \to S_{k+1}. \tag{2}$$

In the decomposition, each Φ_k is a mapping from S_k to S_{k+1} in which the variable v_k is replaced by $\varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$, and all other variables are unchanged.

1.3 Family of Basic AD Algorithms

By introducing live variables, we define a model that unifies both the forward mode and reverse mode AD into a single framework. Figure 1 gives a high-level overview of the unified framework of the basic forward and reverse AD algorithms.

The first series of functions is generated by $g_k(S_1) = \Phi_k \circ g_{k-1}(S_1), k = 1, 2, \cdots, l$, and the initial function g_0 is defined as an identity function on all independent variables S_1. In each step, a forward mode algorithm evaluates the derivatives of $g_k(S_1)$ based on the derivatives of $g_{k-1}(S_1)$, which is the result of the previous step, and the derivatives of $\varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$. The loop invariant of forward mode AD is that the intermediate results are the derivatives of S_{k+1} w.r.t S_1 for the function $g_k : S_1 \to S_{k+1}$, after step $k = 1, 2, \cdots, l$. Since each mapping Φ_k is the identity except for the variables in $v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$, in each step only the derivatives of $v_k(S_1)$ need to be computed.

The second series of functions is generated by $f_k(S_k) = f_{k+1} \circ \Phi_k(S_k), k = l, l-1, \cdots, 1$, where the initial function f_{l+1} is defined as the identity function on the dependent variable $S_{l+1} = \{y\}$. In each step, a reverse mode algorithm evaluates the derivatives of $f_k(S_k)$ based on the derivatives of $f_{k+1}(S_k + 1)$, which is the result of the previous step, and the derivatives of $\varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$. The loop invariant of reverse mode is that the intermediate results are the derivatives of f_k w.r.t the live variables S_k, after each step $k = l \cdots, 1$. The first order reverse mode evaluates and stores the first order derivatives of $f_k(S_k)$ in each step according to the first order chain rule. As we will show, the high order reverse mode evaluates and stores the high order derivatives of $f_k(S_k)$ in each step according to the high order chain rule.

We call each f_k an *equivalent function*, because the objective function can be considered as a composite function of f_k with the unprocessed $\Phi_{k-1} \circ \cdots \Phi_1$. The relationship between two

[2]For simplicity, we focus on scalar objective functions here, the techniques also apply to vector functions.

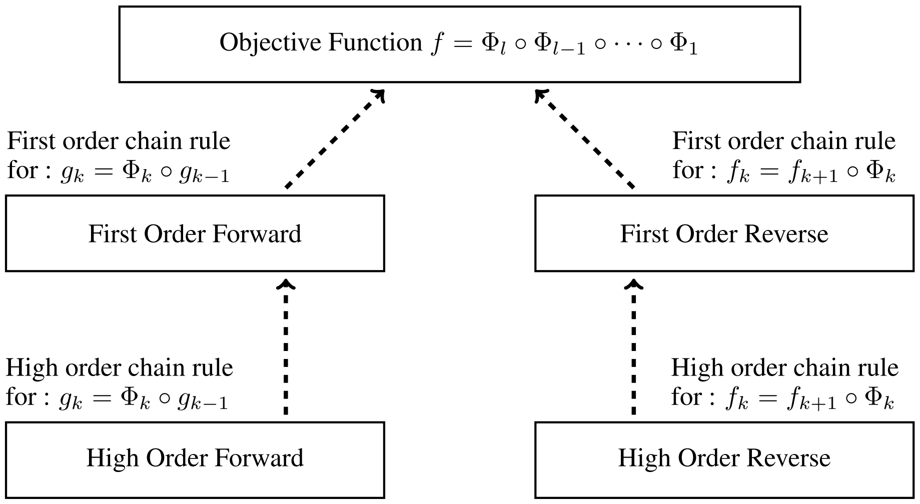
Objective Function $f = \Phi_l \circ \Phi_{l-1} \circ \cdots \circ \Phi_1$
First order chain rule
for : $g_k = \Phi_k \circ g_{k-1}$
First order chain rule
for : $f_k = f_{k+1} \circ \Phi_k$
First Order Forward
First Order Reverse
High order chain rule
for : $g_k = \Phi_k \circ g_{k-1}$
High order chain rule
for : $f_k = f_{k+1} \circ \Phi_k$
High Order Forward
High Order Reverse

Figure 1: Family of basic AD algorithms viewed as composite functions.

consecutive equivalent functions $f_k(S_k)$ and $f_{k+1}(S_{k+1})$ is $f_k(S_k) = f_{k+1} \circ \Phi_k$. Consider that Φ_k maps S_k to S_{k+1} with $v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$, and all other are variables unchanged, so v_k is an independent variable in $f_{k+1}(S_{k+1})$, but it is considered an inner function defined by $v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}$ in $f_k(S_k)$. So we can rewrite it as follows:

$$f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, \quad v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}}). \tag{3}$$

This matches the live variable Equation 1, as v_k is a live variable in S_{k+1} but not in S_k, and all predecessors $\{v_i, v_i \prec v_k\}$ are live variables in S_k. The initial state of the reverse mode is the derivative of the trivial equivalent function $f_{l+1}(S_{l+1})$ as $f_{l+1}(v_l) = v_l$.

1.4 High Order Reverse Mode

The generalized high order reverse mode is the implementation of the high order chain rule for $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}})$. Since we need to differentiate a function multiple times with respect to a variable, we need concepts involving multisets for describing it.

A *multiset* D is a generalization of the notion of a set in which elements are allowed to appear more than once. One way to represent a multiset is $D = \{v_1^{t_1}, \cdots, v_m^{t_m}\}$, which signifies that D has m distinct elements $\{v_1, \cdots, v_m\}$, and the multiplicity of each member v_i is t_i. We call this the *canonical representation* of a multiset. The *order (cardinality)* of a multiset $|D|$ is the sum of all the multiplicities of all its elements, i.e., $|D| = t_1 + \cdots + t_m$.

A multiset D can also be written in a *flat representation* as $D = \{v_{i_1}, v_{i_2}, \cdots, v_{i_{|D|}}\}$, where v_{i_j} and v_{i_k} may represent an identical element even if $j \neq k$. With the flat representation, we define the union ($D_1 \cup D_2$), subset ($D_I \subset D_J$), power set ($\mathcal{P}(D)$), and partition operations on a multiset as we do for a set by treating the index $I = \{i_1, \cdots, i_{|D|}\}$ as a set.

We use the symbol $\mathcal{D}_S$ to represent all multisets over a given set S:

$$\mathcal{D}_S = \Big\{\{v_{i_1}, v_{i_2}, \cdots, v_{i_d}\} : v_{i_k} \in S, 1 \leq k \leq d, d \geq 1\Big\}. \tag{4}$$

Further, for a multiset D, we define the operator $\frac{\partial}{\partial D}$ as: $\frac{\partial}{\partial D} = \frac{\partial^{|D|}}{\partial v_{i_1} \partial v_{i_2} \cdots \partial v_{i_{|D|}}}$.

Lemma 1 *For a composite function defined as $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i:v_i \prec v_k\}})$, the derivative $\frac{\partial f_k(S_k)}{\partial D}$, $D \in \mathcal{D}_{S_k}$ is given by:*

$$\frac{\partial f_k}{\partial D} = \frac{\partial f_{k+1}}{\partial D} + \sum_{\substack{Z \in \mathcal{P}(D) \\ Z \neq D \\ z = |Z|}} \left[\sum_{r=1}^{|D|-z} \Big(\sum_{\substack{D_1, D_2, \cdots, D_r \text{ is} \\ \text{an } r\text{-partition} \\ \text{of } D \setminus Z}} \frac{\partial \varphi_k}{\partial D_1} \cdots \frac{\partial \varphi_k}{\partial D_r} \Big) \frac{\partial f_{k+1}}{\partial Z \partial v_k^r} \right]. \tag{5}$$

The high order reverse mode is the implementation of the high order chain rule Eq 5. An efficient implementation should fully take advantage of the combinatorial properties of this equation. The details can be found in [1, 2, 3].

2 Complexity Analysis

An intriguing feature of the high order reverse mode is illustrated by the following lemma.

Lemma 2 *In step k of the high order reverse mode, let*

$$n_z^{(k)} = \Big|\{Z : Z \in \mathcal{D}_{S_{k+1}\setminus\{v_k\}}, \exists r \in \mathbb{N}^+,\ s.t.\ \frac{\partial f_{k+1}}{\partial Z \partial v_k^r} \neq 0\}\Big|, \tag{6}$$

which is the number of choices of a set Z such that there exists an $r \in \mathbb{N}^+$ and $\frac{\partial f_{k+1}}{\partial Z \partial v_k^r} \neq 0$. Then the complexity in this step is bounded by $O\Big(\binom{3d^*}{d^*} \cdot n_z^{(k)}\Big)$, where d^* is the highest order of derivative that the reverse mode evaluates.

Then by summing over the steps k, we obtain the complexity of the high order reverse mode. Notice that when $d^* = 1$, $n_z^{(k)}$ is a constant, and the lemma reduces to the Baur-Strassen theorem, i.e., the complexity of first order reverse mode is linearly proportional to the complexity of the objective function. The complexity of the high order reverse mode can be precisely expressed using the set

$$L_k = \big\{(Z, r) : \frac{\partial f_{k+1}}{\partial Z \partial v_k^r} \neq 0\big\}.$$

This set L_k corresponds to all the nonzero entries in the slice of the derivative tensor that corresponds to v_k. The number of operations in step k is bounded by $O(q_{d^*} \cdot |L_k|)$, where q_{d^*} is a constant that depends only on d^*. This number is bounded by the size of live variables ($|L_k| \leq d^* \cdot n_z^{(k)}$); but it depends only on the sparsity of the derivative tensor.

From the complexity analysis, we can conclude that the high order reverse mode can detect and exploit sparsity in the derivative tensor on-the-fly. This is an especially attractive feature of the reverse mode, since in many applications the derivative tensor is sparse but the sparsity patterns are not provided as inputs.

Table 1: The constant factor q_{d^*} in the complexity of the high order reverse mode.

Max Order $d^* =$	1	2	3	4	5	6	7	8	9
Constant Factor q_{d^*}=	2	5	9	23	45	86	178	335	591

References

[1] Wang, M. & Gebremedhin, A. & Pothen, A. (2016) Capitalizing on live variables: New algorithms for efficient Hessian computation via Automatic Differentiation. *Mathematical Programming Computation 8*, pp. 393–433.

[2] Wang, M. (2017) High Order Reverse Mode in Automatic Differentiation, PhD thesis, Purdue University.

[3]. Wang, M. & Pothen, A. (2016) Evaluating high order derivative tensors with reverse mode Automatic Differentiation, 41 pp., Submitted for publication.

[4] Martens, J. & Sutskever, I. & Swersky, K. (2012) Estimating the Hessian by back-propagating curvature. arXiv preprint arXiv:1206.6464.

[5] Byrd, R.H. & Chin, G.M. & Neveitt, W. & Nocedal, J. (2011) On the use of stochastic Hessian information in optimization methods for machine learning. SIAM Journal on Optimization, 21(3), pp.977-995.

[6] Goodfellow, I. & Bengio, Y. & Courville, A. (2016) Deep Learning. MIT Press.