Second Order Reverse Mode of AD
A Live Variable Approach for Evaluating Hessian

Mu Wang, Assefaw Gebremedhin and Alex Pothen

Computer Science, Purdue University
EECS, Washington State University

September 30, 2016
Second order reverse mode:
- More efficient in evaluating Hessian in both complexity and memory usage in many applications.
- Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient.

High order reverse mode:
- High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode.
- Implementation: ReverseAD

Applications:
- Uncertainty quantification
- Chemistry: exchange-correlation (XC) energy functional
Research Overview

- Second order reverse mode:
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient
- High order reverse mode:
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode
  - Implementation: ReverseAD
- Applications:
  - Uncertainty quantification
  - Chemistry: exchange-correlation (XC) energy functional
Research Overview

- Second order reverse mode:
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient.

- High order reverse mode:
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode.
  - Implementation: ReverseAD.

- Applications:
  - Uncertainty quantification.
  - Chemistry: exchange-correlation (XC) energy functional.
Research Overview

- **Second order reverse mode:**
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient.

- **High order reverse mode:**
  - High order reverse mode: evaluating derivative tensor $\nabla^d f$ up to any order in reverse mode.
  - Implementation: ReverseAD

- **Applications:**
  - Uncertainty quantification
  - Chemistry: exchange-correlation (XC) energy functional
Research Overview

- Second order reverse mode: \(\leftarrow (\text{this talk})\)
  - More efficient in evaluating Hessian in both complexity and memory usage in many applications.
  - Proved to be equivalent to an variance of vertex elimination on the computational graph of the gradient
- High order reverse mode:
  - High order reverse mode: evaluating derivative tensor \(\nabla^d f\) up to any order in reverse mode
  - Implementation: ReverseAD
- Applications:
  - Uncertainty quantification
  - Chemistry: exchange-correlation (XC) energy functional
First Order AD

- **Objective Function**: $f : \mathcal{R}^n \rightarrow \mathcal{R}^1$ as $f(x) = y$
  - Sufficiently differentiable.
  - Decomposed into SAC sequence $v_i = \varphi_i(v_j)_{\{v_j: v_j \prec v_i\}}, 1 \leq i \leq l$

- Consider AD as a functional operator which maps the objective function to another function

- **Forward Mode**: first order Taylor coefficients
  - $[\mathcal{F}_1 \circ f] : (\mathcal{R}^n, \mathcal{R}^n) \rightarrow \mathcal{R}^1$
  - $[\mathcal{F}_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$

- **Reverse Mode**: adjoints
  - $[\mathcal{R}_1 \circ f] : (\mathcal{R}_n, \mathcal{R}) \rightarrow \mathcal{R}_n$
  - $[\mathcal{R}_1 \circ f](x, \bar{y}) = (\bar{y} \frac{\partial f}{\partial x_1}, \cdots, \bar{y} \frac{\partial f}{\partial x_1}) = \bar{y} \nabla f$

- Fix $\bar{y}$ to be 1
  - $[\mathcal{R}_1 \circ f] : \mathcal{R}^n \rightarrow \mathcal{R}^n$
  - $[\mathcal{R}_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1}) = \nabla f$
First Order AD

- Objective Function: $f : \mathbb{R}^n \rightarrow \mathbb{R}^1$ as $f(x) = y$
  - Sufficiently differentiable.
  - Decomposed into SAC sequence $v_i = \varphi_i(v_j)_{v_j \prec v_i}, 1 \leq i \leq l$
- Consider AD as a functional operator which maps the objective function to another function
- Forward Mode: first order Taylor coefficients
  - $[\mathcal{F}_1 \circ f] : (\mathbb{R}^n, \mathbb{R}^n) \rightarrow \mathbb{R}^1$
  - $[\mathcal{F}_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$
- Reverse Mode: adjoints
  - $[\mathcal{R}_1 \circ f] : (\mathbb{R}^n, \mathbb{R}) \rightarrow \mathbb{R}^n$
  - $[\mathcal{R}_1 \circ f](x, \bar{y}) = (\bar{y} \frac{\partial f}{\partial x_1}, \ldots, \bar{y} \frac{\partial f}{\partial x_l}) = \bar{y} \nabla f$
- Fix $\bar{y}$ to be 1
  - $[\mathcal{R}_1 \circ f] : \mathbb{R}^n \rightarrow \mathbb{R}^n$
  - $[\mathcal{R}_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_l}) = \nabla f$
First Order AD

- **Objective Function**: \( f : \mathbb{R}^n \to \mathbb{R}^1 \) as \( f(\mathbf{x}) = y \)
  - Sufficiently differentiable.
  - Decomposed into SAC sequence \( v_i = \varphi_i(v_j)_{v_j \prec v_i}, 1 \leq i \leq l \)
- Consider AD as a functional operator which maps the objective function to another function
- **Forward Mode**: first order Taylor coefficients
  - \([\mathcal{F}_1 \circ f] : (\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R}^1\)
  - \([\mathcal{F}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{\mathbf{x}}\)
- **Reverse Mode**: adjoints
  - \([\mathcal{R}_1 \circ f] : (\mathbb{R}^n, \mathbb{R}) \to \mathbb{R}^n\)
  - \([\mathcal{R}_1 \circ f](\mathbf{x}, \bar{y}) = (\bar{y} \frac{\partial f}{\partial x_1}, \cdots, \bar{y} \frac{\partial f}{\partial x_1}) = \bar{y} \nabla f\)
- Fix \( \bar{y} \) to be 1
  - \([\mathcal{R}_1 \circ f] : \mathbb{R}^n \to \mathbb{R}^n\)
  - \([\mathcal{R}_1 \circ f](\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1}) = \nabla f\)
Objective Function: $f : \mathcal{R}^n \to \mathcal{R}^1$ as $f(x) = y$

- Sufficiently differentiable.
- Decomposed into SAC sequence $v_i = \varphi_i(v_j)_{\{v_j : v_j \prec v_i\}}$, $1 \leq i \leq l$

Consider AD as a functional operator which maps the objective function to another function

Forward Mode: first order Taylor coefficients

- $[\mathcal{F}_1 \circ f] : (\mathcal{R}^n, \mathcal{R}^n) \to \mathcal{R}^1$
- $[\mathcal{F}_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x}$

Reverse Mode: adjoints

- $[\mathcal{R}_1 \circ f] : (\mathcal{R}^n, \mathcal{R}) \to \mathcal{R}^n$
- $[\mathcal{R}_1 \circ f](x, \bar{y}) = (\bar{y} \frac{\partial f}{\partial x_1}, \cdots, \bar{y} \frac{\partial f}{\partial x_1}) = \bar{y} \nabla f$

Fix $\bar{y}$ to be 1

- $[\mathcal{R}_1 \circ f] : \mathcal{R}^n \to \mathcal{R}^n$
- $[\mathcal{R}_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_1}) = \nabla f$
Objective Function: \( f : \mathbb{R}^n \rightarrow \mathbb{R}^1 \) as \( f(x) = y \)

- Sufficiently differentiable.
- Decomposed into SAC sequence \( v_i = \varphi_i(\{v_j : v_j \prec v_i\}), 1 \leq i \leq l \)

Consider AD as a functional operator which maps the objective function to another function

Forward Mode: first order Taylor coefficients

- \( [F_1 \circ f] : (\mathbb{R}^n, \mathbb{R}^n) \rightarrow \mathbb{R}^1 \)
- \( [F_1 \circ f](x, \dot{x}) = \sum_i \frac{\partial f}{\partial x_i} \dot{x}_i = \nabla f^T \cdot \dot{x} \)

Reverse Mode: adjoints

- \( [R_1 \circ f] : (\mathbb{R}^n, \mathbb{R}) \rightarrow \mathbb{R}^n \)
- \( [R_1 \circ f](x, \bar{y}) = (\bar{y} \frac{\partial f}{\partial x_1}, \cdots, \bar{y} \frac{\partial f}{\partial x_l}) = \bar{y} \nabla f \)

Fix \( \bar{y} \) to be 1

- \( [R_1 \circ f] : \mathbb{R}^n \rightarrow \mathbb{R}^n \)
- \( [R_1 \circ f](x) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_l}) = \nabla f \)
Forward-over-reverse mode:

\[[\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}, \text{ a Hessian-vector product}\]

When the Hessian is sparse: four step procedure\(^1\)

Complexity determined by sparsity properties

Second Order : Compression and Recovery

- **Forward-over-reverse mode:**
  
  \[ [F_1 \circ R_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}, \text{ a Hessian-vector product} \]

- **When the Hessian is sparse:** four step procedure \(^1\)

  - Complexity determined by sparsity properties

---

Second Order : Compression and Recovery

- Forward-over-reverse mode:
  \[ (F_1 \circ R_1 \circ f)(x, \dot{x}) = \nabla^2 f \cdot \dot{x}, \text{ a Hessian-vector product} \]
- When the Hessian is sparse: four step procedure\(^1\)
  - Sparsity pattern detection
  - Compute seed matrix (star/acyclic coloring)
  - Evaluate compressed Hessian
  - Recover sparse Hessian (direct/indirect)
- Complexity determined by sparsity properties

---

Second Order: Compression and Recovery

- Forward-over-reverse mode:
  - $[\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}$, a Hessian-vector product

- When the Hessian is sparse: four step procedure\textsuperscript{1}
  - Sparsity pattern detection
  - Compute seed matrix (star/acyclic coloring)
  - Evaluate compressed Hessian
  - Recover sparse Hessian (direct/indirect)

- Complexity determined by sparsity properties

Forward-over-reverse mode:
\[ [F_1 \circ R_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}, \text{ a Hessian-vector product} \]

When the Hessian is sparse: four step procedure\(^1\)
- Sparsity pattern detection
- Compute seed matrix (star/acyclic coloring)
- Evaluate compressed Hessian
- Recover sparse Hessian (direct/indirect)

Complexity determined by sparsity properties

---

Forward-over-reverse mode:

\[ [\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}, \text{ a Hessian-vector product} \]

When the Hessian is sparse: four step procedure\(^1\)

- Sparsity pattern detection
- Compute seed matrix (star/acyclic coloring)
- Evaluate compressed Hessian
- Recover sparse Hessian (direct/indirect)

Complexity determined by sparsity properties

---

Forward-over-reverse mode:
\[ [\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}, \text{ a Hessian-vector product} \]

When the Hessian is sparse: four step procedure

1. Sparsity pattern detection
2. Compute seed matrix (star/acyclic coloring)
3. Evaluate compressed Hessian
4. Recover sparse Hessian (direct/indirect)

Complexity determined by sparsity properties

---

Second Order : Taylor Coefficients

- Second order forward mode: Taylor coefficients

\[ [\mathcal{F}_2 \circ f](x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x} \]

\[ [\nabla^2 f]_{ij} = [\mathcal{F}_2 \circ f](x, e_i + e_j) - [\mathcal{F}_2 \circ f](x, e_i) - [\mathcal{F}_2 \circ f](x, e_j) \]

- When the Hessian is sparse, the complexity is proportional to the number of nonzero entries in the Hessian matrix.

---

Second Order : Taylor Coefficients

- Second order forward mode : Taylor coefficients²
  \[ \mathcal{F}_2 \circ f (x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x} \]
  \[ [\nabla^2 f]_{ij} = [\mathcal{F}_2 \circ f](x, e_i + e_j) - [\mathcal{F}_2 \circ f](x, e_i) - [\mathcal{F}_2 \circ f](x, e_j) \]

- When the Hessian is sparse, the complexity is proportional to the number of nonzero entries in the Hessian matrix.

---

Second Order : Taylor Coefficients

- Second order forward mode : Taylor coefficients\(^2\)
  - \([\mathcal{F}_2 \circ f](\mathbf{x}, \mathbf{\dot{x}}) = \frac{1}{2} \mathbf{\dot{x}}^T \cdot \nabla^2 f \cdot \mathbf{\dot{x}}\)
  - \([\nabla^2 f]_{ij} = [\mathcal{F}_2 \circ f](\mathbf{x}, e_i + e_j) - [\mathcal{F}_2 \circ f](\mathbf{x}, e_i) - [\mathcal{F}_2 \circ f](\mathbf{x}, e_j)\)

- When the Hessian is sparse, the complexity is proportional to the number of nonzero entries in the Hessian matrix.

Second Order : Taylor Coefficients

- Second order forward mode: Taylor coefficients\(^2\)
  - \([\mathcal{F}_2 \circ f](\mathbf{x}, \mathbf{\dot{x}}) = \frac{1}{2} \mathbf{\dot{x}}^T \cdot \nabla^2 f \cdot \mathbf{\dot{x}}\)
  - \([\nabla^2 f]_{ij} = [\mathcal{F}_2 \circ f](\mathbf{x}, e_i + e_j) - [\mathcal{F}_2 \circ f](\mathbf{x}, e_i) - [\mathcal{F}_2 \circ f](\mathbf{x}, e_j)\)

- When the Hessian is sparse, the complexity is proportional to the number of nonzero entries in the Hessian matrix.

---

Second Order Reverse Mode: Introduction

- First Proposed by Gower and Mello
  - Called Edge Pushing initially
  - From the closed form of second order derivative for composite functions
- The proof can be simplified by adopting live variable analysis
  - Called LivarH in the paper
- Should better be called: second order reverse mode
  - $\mathcal{R}_2 \circ f : \mathcal{R}^n \rightarrow \mathcal{R}^{\frac{n(n+1)}{2}}$
  - $[\mathcal{R}_2 \circ f](x) = \nabla^2 f$

---


Second Order Reverse Mode: Introduction

- First Proposed by Gower and Mello\(^3\)
  - Called Edge Pushing initially
  - From the closed form of second order derivative for composite functions

- The proof can be simplified by adopting live variable analysis\(^4\)
  - Called LivarH in the paper

- Should better be called: second order reverse mode
  - \([R_2 \circ f] : \mathbb{R}^n \rightarrow \mathbb{R}^{\frac{n(n+1)}{2}}\)
  - \([R_2 \circ f](x) = \nabla^2 f\)

---


Second Order Reverse Mode: Definitions

**Definition**

In reverse mode of AD, since the SAC sequence is pre-determined, we say a variable is *live* if it holds a value that *will be* used in the future.

- During function evaluation: \( \{v_{1-n}, \cdots, v_0\} \rightarrow \{v_f\} \)
- During reverse mode: \( \{v_f\} \rightarrow \{v_{1-n}, \cdots, v_0\} \)

**Definition**

After processing SAC \( v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\} \) in reverse mode, SACs processed thus far define an equivalent function \( f_i(S_i) \). The objective function is the composition of \( f_i \) and the remaining SACs, and \( S_i \) is the current live variable set.
Definition

In reverse mode of AD, since the SAC sequence is pre-determined, we say a variable is *live* if it holds a value that *will be* used in the future.

- During function evaluation: \( \{v_{1-n}, \cdots, v_0\} \rightarrow \{v_l\} \)
- During reverse mode: \( \{v_l\} \rightarrow \{v_{1-n}, \cdots, v_0\} \)

Definition

After processing SAC \( \nu_i = \varphi_i(v_j)_{\{v_j : v_j \prec v_i\}} \) in reverse mode, SACs processed thus far define an equivalent function \( f_i(S_i) \). The objective function is the composition of \( f_i \) and the remaining SACs, and \( S_i \) is the current live variable set.
Second Order Reverse Mode: Definitions

Definition

In reverse mode of AD, since the SAC sequence is pre-determined, we say a variable is *live* if it holds a value that *will be* used in the future.

- During function evaluation: \( \{v_{1-n}, \ldots, v_0\} \rightarrow \{v_l\} \)
- During reverse mode: \( \{v_l\} \rightarrow \{v_{1-n}, \ldots, v_0\} \)

Definition

After processing SAC \( v_i = \varphi_i(v_j)_{v_j:v_j \prec v_i} \) in reverse mode, SACs processed thus far define an equivalent function \( f_i(S_i) \). The objective function is the composition of \( f_i \) and the remaining SACs, and \( S_i \) is the current live variable set.
Second Order Reverse Mode: Definitions

Definition

In reverse mode of AD, since the SAC sequence is pre-determined, we say a variable is **live** if it holds a value that will be used in the future.

- During function evaluation: \( \{ v_{1-n}, \ldots, v_0 \} \rightarrow \{ v_i \} \)
- During reverse mode: \( \{ v_i \} \rightarrow \{ v_{1-n}, \ldots, v_0 \} \)

Definition

After processing SAC \( v_i = \varphi_i(v_j)_{v_j:v_j \prec v_i} \) in reverse mode, SACs processed thus far define an equivalent function \( f_i(S_i) \). The objective function is the composition of \( f_i \) and the remaining SACs, and \( S_i \) is the current live variable set.
Second Order Reverse Mode: Definitions

**Definition**

In reverse mode of AD, since the SAC sequence is pre-determined, we say a variable is *live* if it holds a value that *will be* used in the future.

- During function evaluation: \( \{v_{1-n}, \ldots, v_0\} \rightarrow \{v_i\} \)
- During reverse mode: \( \{v_j\} \rightarrow \{v_{1-n}, \ldots, v_0\} \)

**Definition**

After processing SAC \( v_i = \varphi_i(v_j)_{v_j \prec v_i} \) in reverse mode, SACs processed thus far define an equivalent function \( f_i(S_i) \). The objective function is the composition of \( f_i \) and the remaining SACs, and \( S_i \) is the current live variable set.
Before we process $v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\}$:
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\}$:
- We have the equivalent function $f_i(S_i)$.

What has changed?

- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j)\{v_j: v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$
- $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j: v_j \prec v_i\}$

Observation:
First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j) \{v_j : v_j \prec v_i\}$:
  - We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j) \{v_j : v_j \prec v_i\}$:
  - We have the equivalent function $f_i(S_i)$.

What has changed?
  - $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j) \{v_j : v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$.
  - $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$

Observation
First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j) \{ v_j : v_j \prec v_i \}$:
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j) \{ v_j : v_j \prec v_i \}$:
- We have the equivalent function $f_i(S_i)$.

What has changed?

- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j) \{ v_j : v_j \prec v_i \}$ in $f_{i+1}(S_{i+1})$.
- $S_i = S_{i+1} \setminus \{ v_i \} \cup \{ v_j : v_j \prec v_i \}$

**Observation**

First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j)\{v_j : v_j \prec v_i\}$:
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j)\{v_j : v_j \prec v_i\}$:
- We have the equivalent function $f_i(S_i)$

What has changed?
- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j)\{v_j : v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$
- $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$

**Observation**

First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j) \{v_j : v_j \prec v_i\} :$
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j) \{v_j : v_j \prec v_i\} :$
- We have the equivalent function $f_i(S_i)$

What has changed?

- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j) \{v_j : v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$
- $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$

**Observation**

First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j)\{v_j : v_j \prec v_i\}$:
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j)\{v_j : v_j \prec v_i\}$:
- We have the equivalent function $f_i(S_i)$

What has changed?

- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j)\{v_j : v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$
- $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$

Observation

First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process \( v_i = \varphi_i(v_j) \{ v_j : v_j \prec v_i \} \):
- We have the equivalent function \( f_{i+1}(S_{i+1}) \).

After we have processed \( v_i = \varphi_i(v_j) \{ v_j : v_j \prec v_i \} \):
- We have the equivalent function \( f_i(S_i) \).

What has changed?

- \( f_i(S_i) \) is the composite function obtained by replacing \( v_i \) using \( \varphi_i(v_j) \{ v_j : v_j \prec v_i \} \) in \( f_{i+1}(S_{i+1}) \).
- \( S_i = S_{i+1} \setminus \{ v_i \} \cup \{ v_j : v_j \prec v_i \} \).

**Observation**

First order reverse mode computes the first order derivatives (adjoints) of \( f_i(S_i) \) in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j)\{v_j : v_j \prec v_i\}$:
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j)\{v_j : v_j \prec v_i\}$:
- We have the equivalent function $f_i(S_i)$

What has changed?
- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using $\varphi_i(v_j)\{v_j : v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$
- $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$

Observation
First order reverse mode computes the first order derivatives (adjoints) of $f_i(S_i)$ in each step by following the chain rule.
Before we process $v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\}$:
- We have the equivalent function $f_{i+1}(S_{i+1})$.

After we have processed $v_i = \varphi_i(v_j)\{v_j: v_j \prec v_i\}$:
- We have the equivalent function $f_i(S_i)$

What has changed?
- $f_i(S_i)$ is the composite function obtained by replacing $v_i$ using
  $\varphi_i(v_j)\{v_j: v_j \prec v_i\}$ in $f_{i+1}(S_{i+1})$
- $S_i = S_{i+1} \setminus \{v_i\} \cup \{v_j : v_j \prec v_i\}$

**Observation**

First **Second** order reverse mode computes the first and the second order derivatives of $f_i(S_i)$ in each step by following the first and second order chain rule.
Second Order Chain Rule

Definition

\( a_i : S_i \rightarrow \mathcal{R} \) as
\[ a_i(v) = \frac{\partial f_i}{\partial v} \]

\( h_i : S_i \times S_i \rightarrow \mathcal{R} \) as
\[ h_i(v, u) = \frac{\partial^2 f_i}{\partial v \partial u}, h_i(v, u) = h_i(u, v) \]

- \( f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\}, v_i = \varphi_i(v_j)_{v_j \prec v_i}) \)
- First order chain rule: \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i). \)
- Second order chain rule:

\[
\begin{align*}
  h_i(v, u) &= h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \\
  &\quad + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)
\end{align*}
\]

- That’s why we call it second order reverse mode
Second Order Chain Rule

**Definition**

\[ a_i : S_i \rightarrow \mathcal{R} \text{ as } a_i(v) = \frac{\partial f_i}{\partial v} \]

\[ h_i : S_i \times S_i \rightarrow \mathcal{R} \text{ as } h_i(v, u) = \frac{\partial^2 f_i}{\partial v \partial u}, h_i(v, u) = h_i(u, v) \]

\[ f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\}, v_i = \varphi_i(v_j)_{v_j : v_j < v_i}) \]

- First order chain rule: \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \).
- Second order chain rule:

\[ h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \]

\[ + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i) \]

- That’s why we call it second order reverse mode
Definition

\[ a_i : S_i \rightarrow \mathcal{R} \text{ as } a_i(v) = \frac{\partial f_i}{\partial v} \]
\[ h_i : S_i \times S_i \rightarrow \mathcal{R} \text{ as } h_i(v, u) = \frac{\partial^2 f_i}{\partial v \partial u}, h_i(v, u) = h_i(u, v) \]

\[ f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\}, v_i = \varphi_i(v_j)_{v_j < v_i}) \]

- First order chain rule: \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \).
- Second order chain rule:

\[ h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \]
\[ + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i) \]

- That’s why we call it second order reverse mode
Second Order Chain Rule

Definition

\[ a_i : S_i \to \mathcal{R} \text{ as } a_i(v) = \frac{\partial f_i}{\partial v} \]

\[ h_i : S_i \times S_i \to \mathcal{R} \text{ as } h_i(v, u) = \frac{\partial^2 f_i}{\partial v \partial u}, \quad h_i(v, u) = h_i(u, v) \]

\[ f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\}, v_i = \varphi_i(v_j)_{v_j \prec v_i}) \]

- First order chain rule: \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \).
- Second order chain rule:

\[
h_i(v, u) = h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)
\]

That’s why we call it second order reverse mode.
Second Order Chain Rule

**Definition**

\[ a_i : S_i \rightarrow \mathcal{R} \text{ as } a_i(v) = \frac{\partial f_i}{\partial v} \]

\[ h_i : S_i \times S_i \rightarrow \mathcal{R} \text{ as } h_i(v, u) = \frac{\partial^2 f_i}{\partial v \partial u}, h_i(v, u) = h_i(u, v) \]

▶ \( f_i(S_i) = f_{i+1}(S_{i+1} \setminus \{v_i\}, v_i = \varphi_i(v_j)_{v_j \prec v_i}) \)

▶ First order chain rule : \( a_i(v) = a_{i+1}(v) + \frac{\partial \varphi_i}{\partial v} a_{i+1}(v_i) \).

▶ Second order chain rule:

\[
\begin{align*}
    h_i(v, u) &= h_{i+1}(v, u) + \frac{\partial \varphi_i}{\partial v} h_{i+1}(v_i, u) + \frac{\partial \varphi_i}{\partial u} h_{i+1}(v, v_i) \\
    &\quad + \frac{\partial \varphi_i}{\partial v} \frac{\partial \varphi_i}{\partial u} h_{i+1}(v_i, v_i) + \frac{\partial^2 \varphi_i}{\partial v \partial u} a_{i+1}(v_i)
\end{align*}
\]

▶ That’s why we call it second order reverse mode
$h_{i+1}$:

\[ S_{i+1} \]

$S_{i+1}$
Retrieve the row corresponding to $v_i$ in $h_{i+1}(S_{i+1}, S_{i+1})$
Complexity and Implementation

- Retrieve the row corresponding to $v_i$ in $h_{i+1}(S_{i+1}, S_{i+1})$
- Compute $h_i(S_i, S_i)$:
Complexity and Implementation

$h_{i+1}$:

$h_i$:

- Retrieve the row corresponding to $v_i$ in $h_{i+1}(S_{i+1}, S_{i+1})$
- Compute $h_i(S_i, S_i)$:
  - Only update necessary entries
\[ h_{i+1} : \]

\[ S_{i+1} \]

\[ v_i \]

\[ \{ v_j : v_j \prec v_i \} \]

\[ S_i \]

\[ h_i : \]

\[ S_i \{ v_j : v_j \prec v_i \} \]

\[ \varphi_i(v_j)\{v_j : v_j \prec v_i\} \text{ is unary or binary} \]
$h_{i+1}: S_{i+1} \downarrow v_i$

$h_i:\ S_i\{v_j : v_j \prec v_i\}$

- $\varphi_i(v_j)_{\{v_j : v_j \prec v_i\}}$ is unary or binary
- The complexity in each step is $O(s_i)$, $s_i = |S_i|$
 Complexity and Implementation

\[ h_{i+1} : S_{i+1} \]

\[ h_i : S_i \{ v_j : v_j \prec v_i \} \]

- \( \varphi_i(v_j) \{ v_j : v_j \prec v_i \} \) is unary or binary
- The complexity in each step is \( O(s_i) \), \( s_i = |S_i| \)
- Overall complexity : \( O(l \cdot s) \), \( s = \max_i \{ s_i \} \), \( 1 \leq i \leq l \)

Mu, et.al (Purdue University) Second Order Reverse AD September 30, 2016 10 / 1
Maintain only one set of $a(S)$ and $h(S, S)$, incremental updates
Maintain only one set of \( a(S) \) and \( h(S, S) \), incremental updates

Only retrieve and update nonzero entries
Complexity and Implementation

$h_{i+1}$: $S_{i+1}$ $v_i$ $h_i$: $S_i$ $\{v_j : v_j \prec v_i\}$

- Maintain only one set of $a(S)$ and $h(S, S)$, incremental updates
- Only retrieve and update nonzero entries
- $O(d_i)$ updates in each step: number of nonzeros in row $v_i$ of $h_{i+1}(S_{i+1}, S_{i+1})$
Second Order AD : Different Scenarios

- When only some entries in Hessian are needed, e.g., the diagonal:
  Second order forward mode
  \[
  [\mathcal{F}_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \dot{\mathbf{x}}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}}
  \]
  \[
  [\nabla^2 f]_{ii} = 2[\mathcal{F}_2 \circ f](\mathbf{x}, e_i), \; 1 \leq i \leq n
  \]

- Hessian-vector product: forward-over-reverse mode
  \[
  [\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}
  \]

- Hessian matrix: second order reverse mode
  \[
  [\mathcal{R}_2 \circ f](\mathbf{x}) = \nabla^2 f
  \]

- Inverse of the Hessian: it depends
  - Solve linear system: \( H \cdot \mathbf{x} = \mathbf{b} \)
  - Directly evaluate \( H \) and then take the inverse
  - Indirectly evaluate \( H \cdot \mathbf{v} \) via forward-over-reverse mode
When only some entries in Hessian are needed, e.g., the diagonal:

- **Second order forward mode**
  
  \[
  [\mathcal{F}_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \dot{\mathbf{x}}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}}
  \]
  
  \[
  [\nabla^2 f]_{ii} = 2[\mathcal{F}_2 \circ f](\mathbf{x}, e_i), 1 \leq i \leq n
  \]

- **Hessian-vector product : forward-over-reverse mode**
  
  \[
  [\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}
  \]

- **Hessian matrix : second order reverse mode**
  
  \[
  [\mathcal{R}_2 \circ f](\mathbf{x}) = \nabla^2 f
  \]

- **Inverse of the Hessian : it depends**
  
  - Solve linear system : \( H \cdot \mathbf{x} = \mathbf{b} \)
  
  - Directly evaluate \( H \) and then take the inverse
  
  - Indirectly evaluate \( H \cdot \mathbf{v} \) via forward-over-reverse mode
When only some entries in Hessian are needed, e.g., the diagonal:

**Second order forward mode**

- $[\mathcal{F}_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \dot{\mathbf{x}}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}}$
- $[\nabla^2 f]_{ii} = 2[\mathcal{F}_2 \circ f](\mathbf{x}, e_i), 1 \leq i \leq n$

**Hessian-vector product: forward-over-reverse mode**

- $[\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}$

**Hessian matrix: second order reverse mode**

- $[\mathcal{R}_2 \circ f](\mathbf{x}) = \nabla^2 f$

**Inverse of the Hessian: it depends**

- Solve linear system: $H \cdot \mathbf{x} = \mathbf{b}$
- Directly evaluate $H$ and then take the inverse
- Indirectly evaluate $H \cdot \mathbf{v}$ via forward-over-reverse mode
Second Order AD: Different Scenarios

- When only some entries in Hessian are needed, e.g., the diagonal:
  - Second order forward mode
    - \([\mathcal{F}_2 \circ f](x, \dot{x}) = \frac{1}{2} \dot{x}^T \cdot \nabla^2 f \cdot \dot{x}\)
    - \([\nabla^2 f]_{ii} = 2[\mathcal{F}_2 \circ f](x, e_i), 1 \leq i \leq n\)

- Hessian-vector product: forward-over-reverse mode
  - \([\mathcal{F}_1 \circ \mathcal{R}_1 \circ f](x, \dot{x}) = \nabla^2 f \cdot \dot{x}\)

- Hessian matrix: second order reverse mode
  - \([\mathcal{R}_2 \circ f](x) = \nabla^2 f\)

- Inverse of the Hessian: it depends
  - Solve linear system: \(H \cdot x = b\)
  - Directly evaluate \(H\) and then take the inverse
  - Indirectly evaluate \(H \cdot v\) via forward-over-reverse mode
When only some entries in Hessian are needed, e.g., the diagonal:
Second order forward mode
\[
[F_2 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2} \dot{\mathbf{x}}^T \cdot \nabla^2 f \cdot \dot{\mathbf{x}}
\]
\[
[\nabla^2 f]_{ii} = 2[F_2 \circ f](\mathbf{x}, e_i), 1 \leq i \leq n
\]

Hessian-vector product: forward-over-reverse mode
\[
[F_1 \circ R_1 \circ f](\mathbf{x}, \dot{\mathbf{x}}) = \nabla^2 f \cdot \dot{\mathbf{x}}
\]

Hessian matrix: second order reverse mode
\[
[R_2 \circ f](\mathbf{x}) = \nabla^2 f
\]

Inverse of the Hessian: it depends

- Solve linear system: \( H \cdot \mathbf{x} = \mathbf{b} \)
- Directly evaluate \( H \) and then take the inverse
- Indirectly evaluate \( H \cdot v \) via forward-over-reverse mode
ReverseAD: an operator overloading implementation of the second order reverse mode in C++11.

- (Almost) compatible interface with ADOL-C
- Monotonic indexing is required for efficiency
  \[ v_j < v_i \implies \text{index}(v_j) < \text{index}(v_i) \]
- https://github.com/wangmu0701/ReverseAD

Design choices:

- Store intermediate values also in the trace
  Needed anyway
  \[ O(\log s) \text{ value access cost instead of } O(1) \]
- In-memory / disk trace for small / large cases.
  No one cares about performance on trivial functions
  Potential of checkpointing, user defined functions, etc
ReverseAD: an operator overloading implementation of the second order reverse mode in C++11.

- (Almost) compatible interface with ADOL-C
- Monotonic indexing is required for efficiency
  \[ v_j < v_i \implies \text{index}(v_j) < \text{index}(v_i) \]
- https://github.com/wangmu0701/ReverseAD

Design choices:

- Store intermediate values also in the trace
  Needed anyway
  \[ O(\log s) \text{ value access cost instead of } O(1) \]
- In-memory / disk trace for small / large cases.
  No one cares about performance on trivial functions
  Potential of checkpointing, user defined functions, etc
A synthetic function designed with parameters:

- $n$: number of independent variables
- $s$: size of live variables during the function evaluation
- $l$: the complexity of the function
- $p$: average number of nonzeros per row in the final Hessian

$$y = \sum_{i=1}^{n} z_i * t_i$$

$$z_i = ID_k \circ \cdots \circ ID_1(x_i)$$

$$t_i = \sum_{j=1}^{\rho/2} x_{r_j} + \sum_{j=1}^{s} x_{ij}$$

$$ID(w) = \begin{cases} 
\sqrt{w * w}, \\
2.0 + w - 2.0, \\
w * 2.0 * 0.5, \\
\log(\exp(w)), \\
1.0/(1.0/w), \\
sin(asin(w)). 
\end{cases}$$
A synthetic function designed with parameters:

- $n$: number of independent variables
- $s$: size of live variables during the function evaluation
- $l$: the complexity of the function
- $p$: average number of nonzeros per row in the final Hessian

$$
y = \sum_{i=1}^{n} z_i \ast t_i
$$

$$
z_i = ID_k \circ \cdots \circ ID_1(x_i)
$$

$$
t_i = \sum_{j=1}^{\rho/2} x_{r_j} + \sum_{j=1}^{s} x_{i_j}
$$

$$
ID(w) = \begin{cases} 
\sqrt{w \ast w}, \\
2.0 + w - 2.0, \\
w \ast 2.0 \ast 0.5, \\
\log(\exp(w)), \\
1.0/(1.0/w), \\
\sin(\arcsin(w)).
\end{cases}
$$
Synthetic Function: Varying Size of Live Variables

- $n = 20,000$, $p = 6$, $l$ fixed, $s$ varies
Synthetic Function: Varying Size of Live Variables

- $n = 20,000$, $p = 6$, $l$ fixed, $s$ varies
Synthetic Function: Varying Sparsity Pattern

- \([n = 15,000, p = 4]\), \([n = 20,000, p = 3]\), and \([n = 30,000, p = 2]\).

<table>
<thead>
<tr>
<th></th>
<th>(n = 15,000)</th>
<th>20,000</th>
<th>30,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>139.8</td>
<td>25.7</td>
<td>12.8</td>
</tr>
<tr>
<td>Indirect</td>
<td>134.2</td>
<td>22.2</td>
<td>9.6</td>
</tr>
<tr>
<td>ReverseAD</td>
<td>7.6</td>
<td>8.6</td>
<td>9.7</td>
</tr>
</tbody>
</table>

Mu, et.al (Purdue University)
Synthetic Function : Varying Sparsity Pattern

- \([n = 15,000, p = 4]\), \([n = 20,000, p = 3]\), and \([n = 30,000, p = 2]\).

<table>
<thead>
<tr>
<th></th>
<th>(n = 15,000)</th>
<th>20,000</th>
<th>30,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>139.8</td>
<td>25.7</td>
<td>12.8</td>
</tr>
<tr>
<td>Indirect</td>
<td>134.2</td>
<td>22.2</td>
<td>9.6</td>
</tr>
<tr>
<td>ReverseAD</td>
<td>7.6</td>
<td>8.6</td>
<td>9.7</td>
</tr>
</tbody>
</table>

Mu, et.al (Purdue University)
Performance: Airfoil

- From the AD-Suite: https://gitlab.com/mod0/AD-suite
- \( n = 43,566 \), Hessian becomes more dense with more iterations
Performance : GMM

- From Filip Srajer: https://github.com/awf/autodiff/.
- \( n = k \cdot (d + 1) \cdot (d + 2) / 2 \), parameterized on \( d \) and \( k \)
- Dense Hessian

<table>
<thead>
<tr>
<th>( d, k )</th>
<th>( n )</th>
<th>ReverseAD</th>
<th>HessVec</th>
<th>FullHess</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 10</td>
<td>60</td>
<td>1.096</td>
<td>0.009</td>
<td>0.313</td>
</tr>
<tr>
<td>2, 25</td>
<td>150</td>
<td>6.976</td>
<td>0.080</td>
<td>14.270</td>
</tr>
<tr>
<td>2, 50</td>
<td>300</td>
<td>30.213</td>
<td>0.163</td>
<td>55.511</td>
</tr>
<tr>
<td>2, 100</td>
<td>600</td>
<td>144.818</td>
<td>0.326</td>
<td>211.157</td>
</tr>
<tr>
<td>10, 5</td>
<td>330</td>
<td>31.594</td>
<td>0.135</td>
<td>48.104</td>
</tr>
<tr>
<td>10, 10</td>
<td>660</td>
<td>134.031</td>
<td>0.258</td>
<td>185.614</td>
</tr>
<tr>
<td>10, 25</td>
<td>1650</td>
<td>1057.216</td>
<td>0.617</td>
<td>1110.392</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

- Second order derivative tensor (Hessian) could (and probably should) be evaluated directly via reverse mode.
- Proved (with Paul Hovland) to be equivalent to performing vertex elimination on the computational graph of the gradient following a reverse topological order while preserving symmetry (To appear in SIAM Procs. CSC 2016)
- Extend to high orders *(14:30pm today)*
  - More related topics will be covered
Conclusion and Future Work

- Second order derivative tensor (Hessian) could (and probably should) be evaluated directly via reverse mode.
- Proved (with Paul Hovland) to be equivalent to performing vertex elimination on the computational graph of the gradient following a reverse topological order while preserving symmetry (To appear in SIAM Procs. CSC 2016)
- Extend to high orders (14:30pm today)
  - More related topics will be covered
Conclusion and Future Work

- Second order derivative tensor (Hessian) could (and probably should) be evaluated directly via reverse mode.
- Proved (with Paul Hovland) to be equivalent to performing vertex elimination on the computational graph of the gradient following a reverse topological order while preserving symmetry (To appear in SIAM Procs. CSC 2016)
- Extend to high orders (14:30pm today)
  - More related topics will be covered
Conclusion and Future Work

- Second order derivative tensor (Hessian) could (and probably should) be evaluated directly via reverse mode.
- Proved (with Paul Hovland) to be equivalent to performing vertex elimination on the computational graph of the gradient following a reverse topological order while preserving symmetry (To appear in SIAM Procs. CSC 2016)
- Extend to high orders (14:30pm today)
  - More related topics will be covered

Observation

Second order reverse mode computes the first and the second order derivatives of $f_i(S_i)$ in each step by following the first and second order chain rule.
Conclusion and Future Work

- Second order derivative tensor (Hessian) could (and probably should) be evaluated directly via reverse mode.
- Proved (with Paul Hovland) to be equivalent to performing vertex elimination on the computational graph of the gradient following a reverse topological order while preserving symmetry (To appear in SIAM Procs. CSC 2016)
- Extend to high orders (14:30pm today)
  - More related topics will be covered

Observation

Second order High order reverse mode computes the first and the second order derivatives up to order $d$ of $f_i(S_i)$ in each step by following the first and second high order chain rule.
Conclusion and Future Work

- Second order derivative tensor (Hessian) could (and probably should) be evaluated directly via reverse mode.
- Proved (with Paul Hovland) to be equivalent to performing vertex elimination on the computational graph of the gradient following a reverse topological order while preserving symmetry (To appear in SIAM Procs. CSC 2016)
- Extend to high orders (14:30pm today)
  - More related topics will be covered

\[
\mathcal{T}_{f_i}(D) = \mathcal{T}_{f_{i+1}}(D) + \sum_{D_L \not\subset D} \left[ \sum_{D_i \cap D_j = \emptyset} \mathcal{T}_{\varphi_i}(D_1) \cdots \mathcal{T}_{\varphi_i}(D_r) \mathcal{T}_{f_{i+1}}(D_L \cup \{v'_i\}) \right]
\]
References


- Wang, Mu, Alex Pothen and Paul Hovland. "Edge Pushing is Equivalent to Vertex Elimination for Computing Hessians". To be appear in SIAM CSC16.