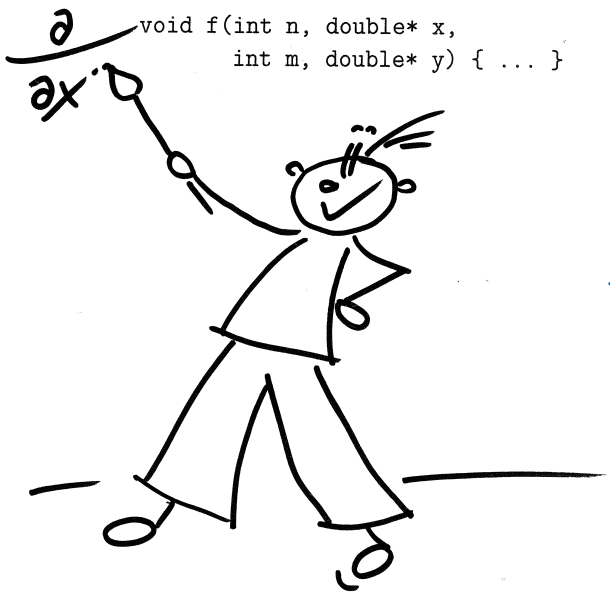
Software and Tools for Computational Engineering
RWTH AACHEN UNIVERSITY



Algorithmic Differentiation for Machine Learning

... beyond backpropagation

Uwe Naumann

RWTH Aachen University, Aachen, Germany

Disclaimer

1. I am not a user of ML. I work with people who are.

2. I would not call myself "ML developer" either. My areas of interest and expertise include
   - AD
   - combinatorial problems in scientific computing.
   - program analysis, transformation and optimization
   - software development

   They overlap with modern ML development.

3. Building on the above, how can I contribute to progress in ML?

Outline

Outline

We aim to replace a differentiable (w.l.o.g. C++) target [sub-]program

$$y = F(x) : \boldsymbol{R}^n \to \boldsymbol{R}^m$$

with a (w.l.o.g.) neural network

$$y = f(x)$$

of depth $D > 0$ implemented with (w.l.o.g.) TensorFlow.

Not covered here: Our AD software dco/c++ (1) features an *external adjoint interface* for coupling (e.g.) TensorFlow with a C++ adjoint. A corresponding → adjoint code design pattern (2) is under development.

(1) K. Leppkes, J. Lotz, U. N.: dco/c++: Derivative Code by Overloading in C++. NAG TR2/20, 2020.

(2) U. N.: Adjoint Code Design Patterns. ACM Trans. Math. Softw. 45 (3), 1-32, 2019.

F is embedded in a context $\hat{y} = G(\hat{x}) : \boldsymbol{R}^{\hat{n}} \to \boldsymbol{R}^{\hat{m}}$ to be differentiated with dco/c++.

Tangents

$$\boldsymbol{R}^{\hat{n} \times \hat{n}^{(1)}} \ni \hat{Y}^{(1)} = G' \cdot \hat{X}^{(1)} \equiv \frac{dG}{d\hat{x}} \cdot \hat{X}^{(1)} ,$$

adjoints

$$\boldsymbol{R}^{\hat{m}_{(1)} \times \hat{m}} \ni \hat{X}_{(1)} = \hat{Y}_{(1)} \cdot G'$$

and higher-order tangents and/or adjoints,[1] e.g,

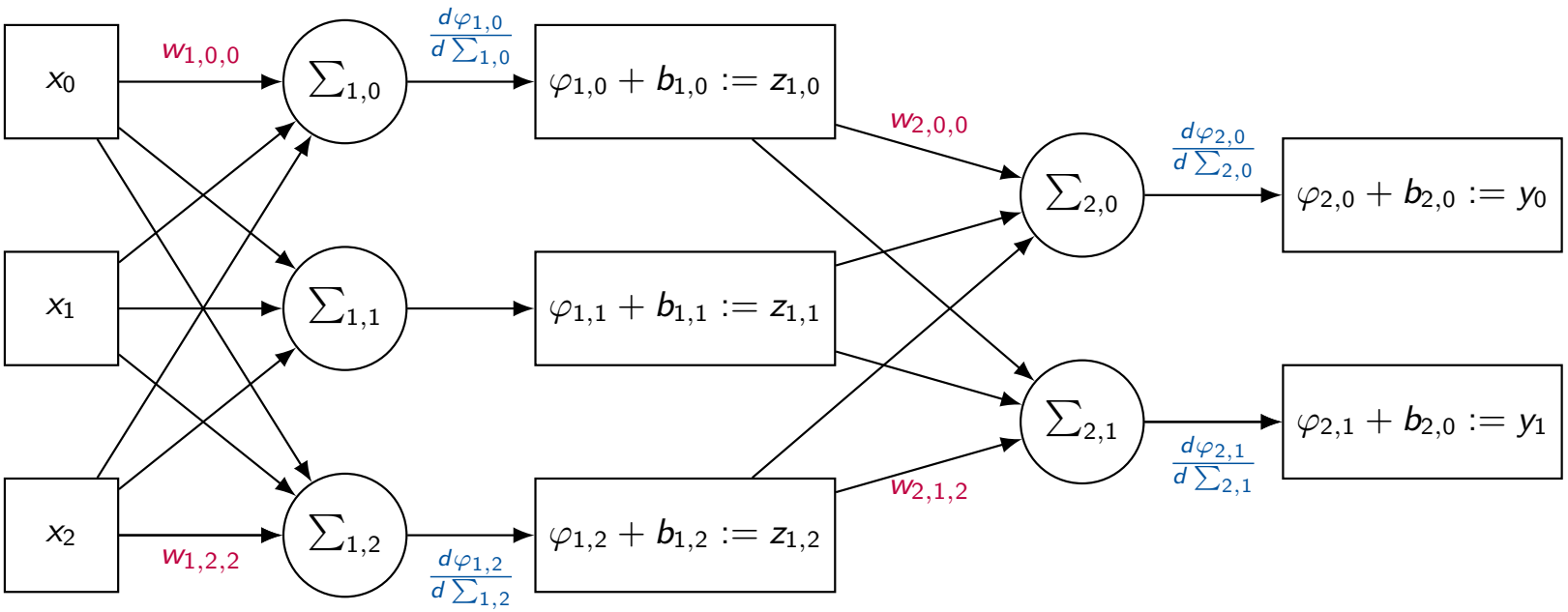
$$\boldsymbol{R}^{\hat{m}_{(1)} \times \hat{n} \times \hat{n}^{(2)}} \ni \left[\hat{X}^{(2)}_{(1)}\right]_{l,i_1,k} = \left[\hat{Y}_{(1)}\right]_{l,j} \cdot \left[\frac{d^2 G}{d\hat{x}^2}\right]_{j,i_1,i_2} \cdot \left[\hat{X}^{(2)}\right]_{i_2,k} ,$$

of G (as well as F, and, hence, f) may be required, for example, for differential learning, sensitivity analysis, model calibration or uncertainty quantification.

[1] index notation for tensor arithmetic

A (w.l.o.g. deep feed forward) neural network $y = f(x) : \mathbb{R}^n \to \mathbb{R}^m$ of depth $D > 0$ is parameterized by weights $W = (w_{i,j,k}, w_{D,j,l}) \in \mathbb{R}^{D-1 \times n \times n} \times \mathbb{R}^{n \times m}$ and biases $B = (b_{i,j}, b_{D,l}) \in \mathbb{R}^{D-1 \times n} \times \mathbb{R}^m$.

It induces a (directed acyclic) computational graph (V, E) with $|V| \geq 0$ vertices $V = \{0, \ldots, |V| - 1\}$ and $|E|$ edges $E \subset V \times V$, e.g, $n = 3$, $m = D = 2$:



Ultimately, ML amounts to the calibration of highly parameterized differentiable programs for certain loss functions $L(W)$.

A large-scale nonconvex optimization problem featuring a potentially large number of local stationary points needs to be solved.

W.l.o.g, we consider differentiable programs (e.g, neural networks) given as evolutions

$$\begin{aligned} z_0 &= x \\ z_i &= \Phi_i(z_{i-1}), \quad i = 1, \ldots, D \\ y &= z_D \end{aligned} \tag{1}$$

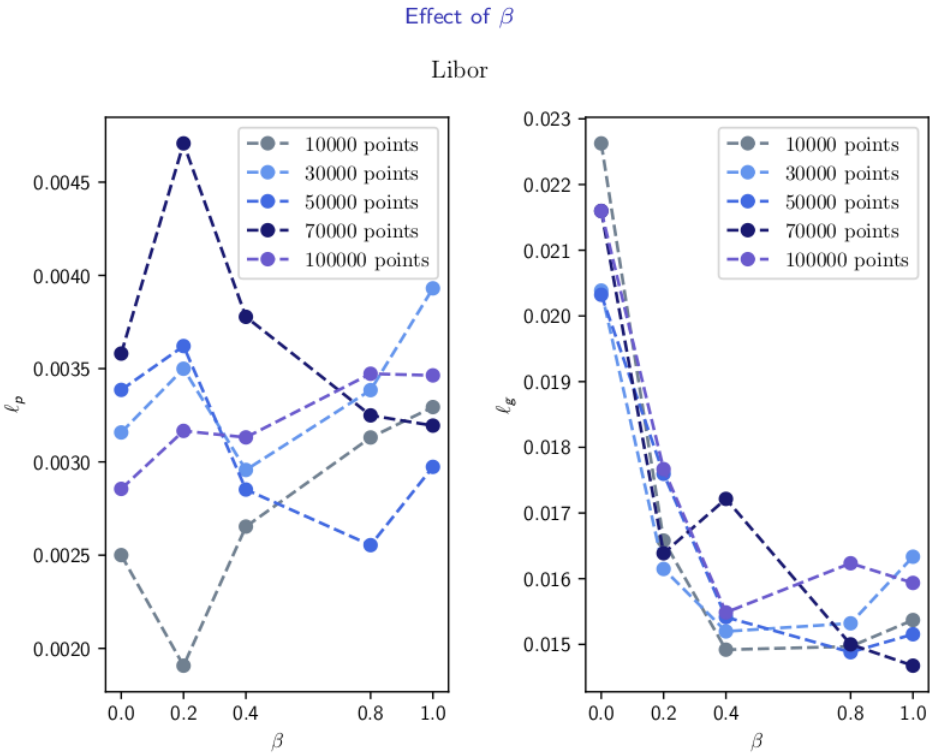
with sufficiently often differentiable transitions Φ_i and corresponding Jacobians

$$\Phi_i' \equiv \frac{d\Phi_i}{dz_{i-1}} .$$

Building on the capabilities of dco/c++ we consider loss functions with weighted value and first (and higher) derivative components

$$L(W) = \alpha \cdot l_p(W) + \beta \cdot l_g(W)$$

(e.g, results from (4) for LIBOR model code from (3)).



(3) M. Giles, P. Glasserman: Smoking Adjoints. Risk Magazine 2006.

(4) S. Christodoulou: Approximation of Expensive Functions through Neural Networks. BSc Thesis, STCE, RWTH Aachen, 2020. → second-order differential learning

(5) B. Huge, A. Savine: Differential Machine Learning: The Shape of Things to Come. risk.net, 2020.

Building on our main area of interest / expertise ($\rightarrow$ AD) we aim to reduce the

1. size of the network
   $\Rightarrow$ pruning

2. cost of differentiation of the network
   $\Rightarrow$ (generalized) Jacobian chain product.

Prior work is applied to the ML scenario. The exploitation of special structure allows for potentially stronger results.

Outline

Initially oversized neural networks have a regularizing effect on the training.

Pruning aims to reduce the size of the trained network.

We consider two methods:

Vertex Pruning

Elimination of insignificant vertices incl. all incident edges from the network.

Edge Pruning

Elimination of insignificant edges from the network. This may lead to the elimination of an incident node in case of complete isolation.

... as well as combinations thereof.

In approximate ("green") computing a (intermediate) value v computed by a differentiable program $y = f(\mathrm{x})$ is insignificant over the given interval (sub-)domain $[\mathrm{x}] \subseteq \boldsymbol{R}^n$ if

$$\sigma(v) = w([v]) \cdot \max \left| \frac{dy}{dv}([\mathrm{x}]) \right| \leq \bar{\sigma}$$

for context-sensitive $\bar{\sigma} \in \boldsymbol{R}$.

Switching v from variable to constant (e.g, midpoint of $[v]$) allows for more aggressive compiler optimization (constant propagation and dead code elimination).

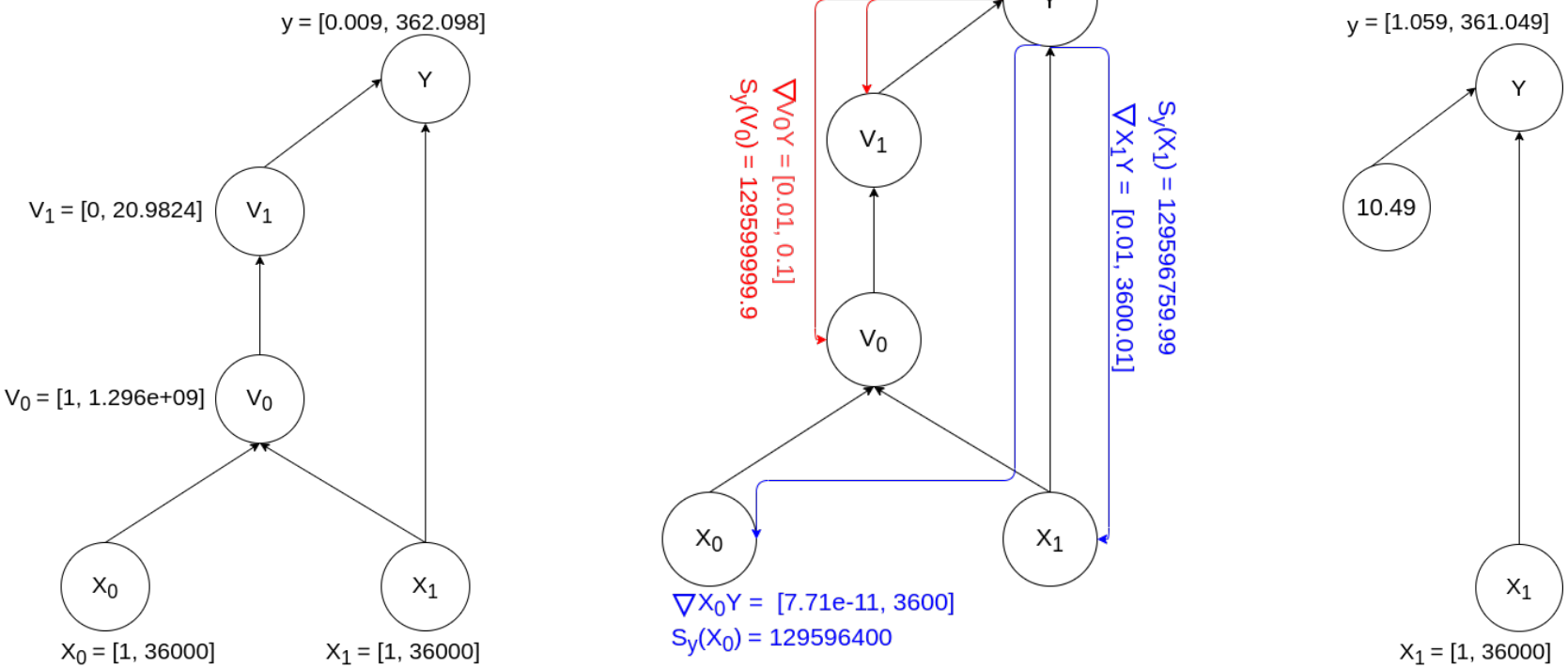
The interval adjoint computes $\frac{dy}{dv}([\mathrm{x}])$ efficiently.

(6) V. Vassiliadis, U.N. et al.: Towards automatic significance analysis for approximate computing. IEEE/ACM CGO, 2016.

Interval Adjoint Significance Analysis

Example

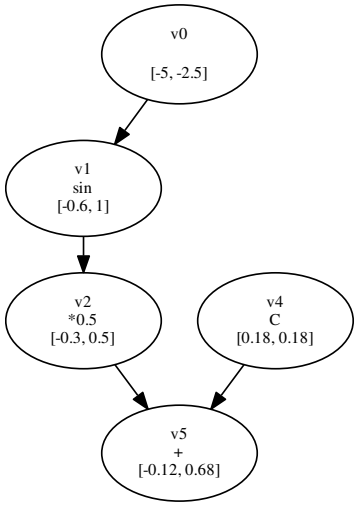
$$y = \frac{\log(x_0 \cdot x_1)}{10} + \frac{x_1}{100}, \quad x_i \in [1, 36000]$$



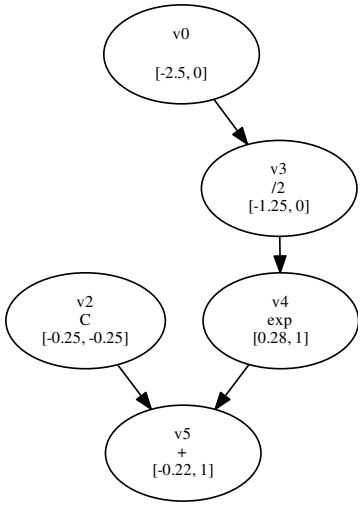
Note: efficient interval gradient wrt. all intermediates in adjoint mode

Interval Adjoint Significance Analysis

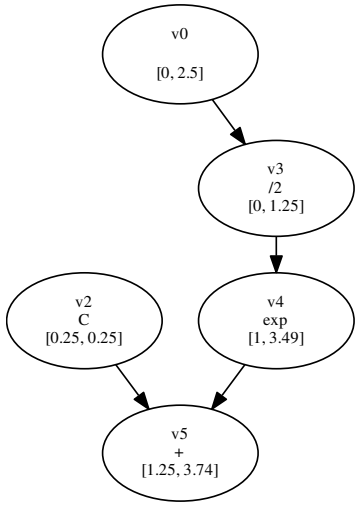
Exploitation through Specialization



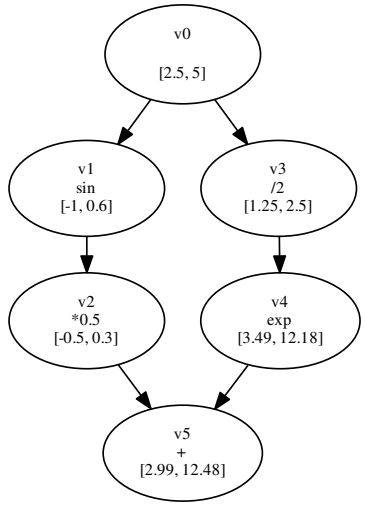
(a) $x \in [-5, -2.5]$



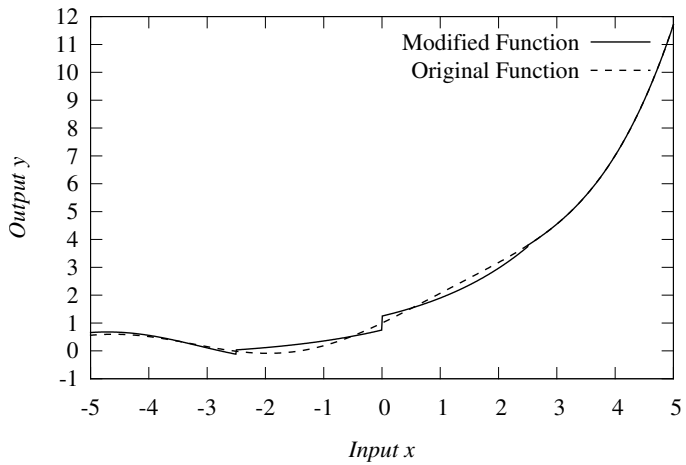
(b) $x \in [-2.5, 0]$



(c) $x \in [0, 2.5]$



(d) $x \in [2.5, 5]$

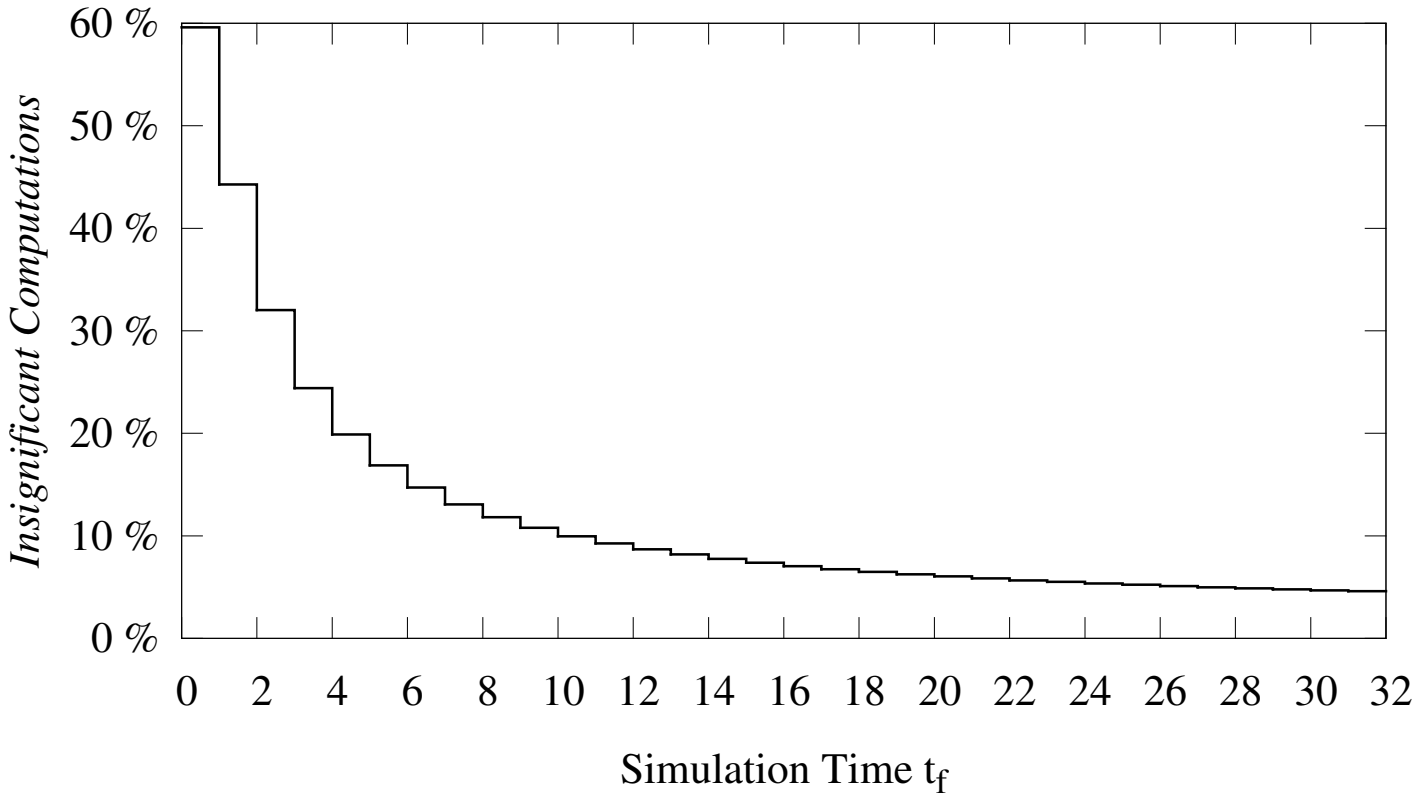


... running different code over different subdomains.

(7) J. Deussen, J. Riehme, U. N.: Automation of Significance Analyses with Interval Splitting. ParCo 2015.

Interval Adjoint Significance Analysis

Application to Diffusion



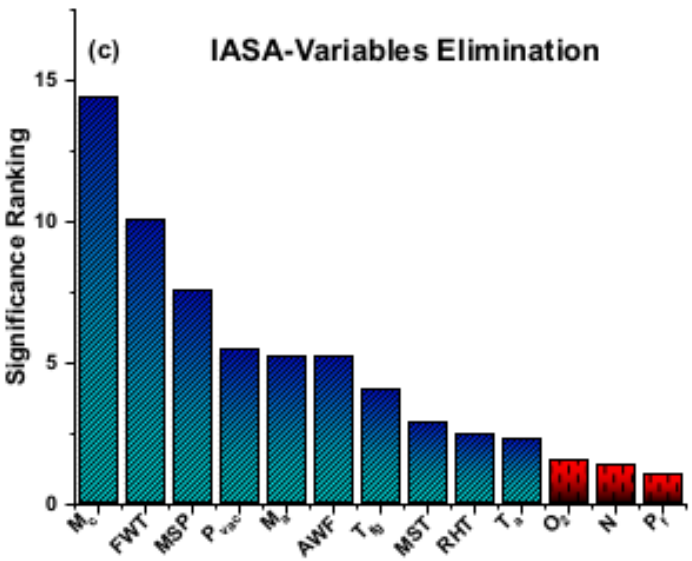
Accuracy can be relaxed in early stages of the simulation (intuitively clear).

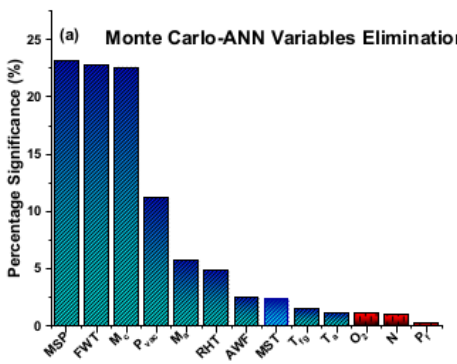
(8) J. Deussen, U.N. et al.: Interval-Adjoint Significance Analysis: A Case Study. WAPCO 2016.

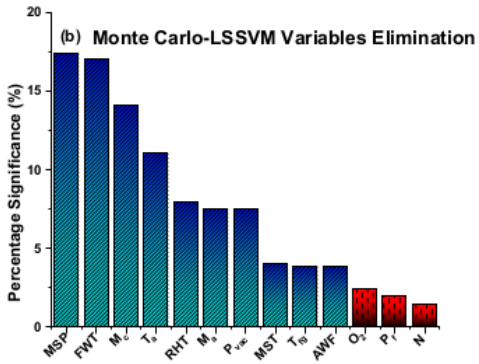
Objective: Efficiency of coal power plant.

Three out of thirteen control variables are insignificant.

Overall improvement of efficiency by $\approx 7\%$.



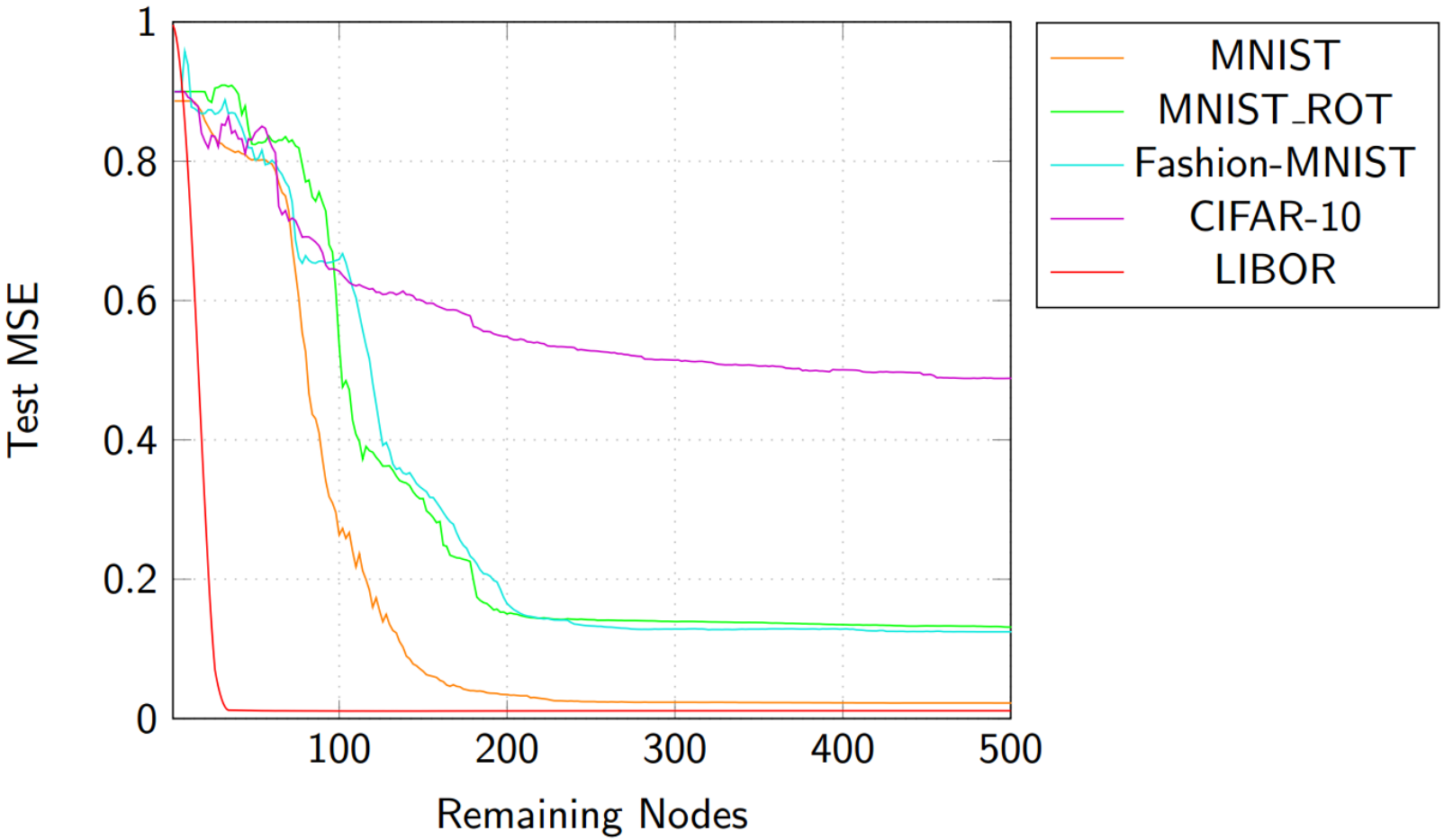




(9) W. Ashraf, S. Afghan, U.N. et al.: Optimization of a 660 MWe Supercritical Power Plant Performance—A Case of Industry 4.0 in the Data-Driven Operational Management. Part 1. Thermal Efficiency. Energies. MDPI, 2020.

Interval Adjoint Significance Analysis

Application to Neural Networks



(10) S. Afghan, U.N.: Interval Adjoint Significance Analysis for Neural Networks. ICCS 2020.

Application of interval adjoint significance analysis to edge pruning is work in progress → S. Afghan.

First test results with simple feed forward deep neural networks indicate superiority of standard value pruning. Currently, our favorite method combines vertex pruning based on interval adjoint significance analysis with edge pruning based on absolute values of weights.

We expect to see different behavior when considering alternative test cases and network types.

Outline

Differentiation of G (the context of F) requires differentiation of f (the surrogate for F).

W.l.o.g, we consider the accumulation of the Jacobian f' of f.

The chain rule yields

$$f' = \frac{dy}{dx} = \Phi'_D \cdot \Phi'_{D-1} \cdot \ldots \cdot \Phi'_1 \, .$$

Three scenarios will be discussed:

1. Dense Φ'_i are given with $\Phi'_i \in \boldsymbol{R}^{n \times n}$ for $i = 1, \ldots, D-1$ and $\Phi'_D \in \boldsymbol{R}^{m \times n}$. (clear!)
2. Vertex pruning yields dense Φ'_i with $\Phi'_i \in \boldsymbol{R}^{n_i \times n_{i-1}}$ for $i = 1, \ldots, D$ such that $n_0 = n$ and $m = n_D$.
3. Edge pruning (potentially combined with vertex pruning) yields sparse Φ'_i with $\Phi'_i \in \boldsymbol{R}^{n_i \times n_{i-1}}$ for $i = 1, \ldots, D$ such that $n_0 = n$ and $m = n_D$.

All factors in

$$f' = \Phi'_D \cdot \Phi'_{D-1} \cdot \ldots \cdot \Phi'_1 \, .$$

turn out to be dense.

We denote the computational cost of evaluating a subchain $\Phi'_j \cdot \ldots \cdot \Phi'_i$, $j > i$ as $\mathrm{fma}_{j,i}$ (fused multiply-adds).

All Φ'_i are assumed to be available, that is, $\mathrm{fma}_{i,i} = 0$.

Dynamic programming yields an optimal bracketing at a computational cost of $\mathcal{O}(D^3)$:

$$\mathrm{fma}_{j,i} = \begin{cases} 0 & j = i \\ \min_{i \le k < j} \left(\mathrm{fma}_{j,k+1} + \mathrm{fma}_{k,i} + \mathrm{fma}_{j,k,i} \right) & j > i \, . \end{cases} \tag{2}$$

Equation (2) extends to the sparse case due to edge pruning. Sparsity patterns of the subchains need to be computed explicitly. The computational complexity increases to $\mathcal{O}(D^3 \cdot \max(\max_{i=1,\dots,D} n_i, \bar{m})^3)$. It remains polynomial in the problem size.

The general Sparse Matrix Chain Product problem is known to be NP-complete. Reduction of Ensemble Computation to Diagonal Matrix Chain Product is at the heart of the formal proof. For example, the matrix product

$$\begin{pmatrix} 6 & 0 \\ 0 & 7 \end{pmatrix} \begin{pmatrix} 7 & 0 \\ 0 & 6 \end{pmatrix} = \begin{pmatrix} 42 & 0 \\ 0 & 42 \end{pmatrix}$$

can be evaluated at the expense of a single fma as opposed to two by exploiting commutativity of scalar multiplication.

(42) D. Adams: The Hitchhiker's Guide to the Galaxy. Pan Books, 1979.

(11) U. N.: On Sparse Matrix Chain Products. SIAM CSC 2020.

Realistically, the Φ_i' are not available as the $\Phi_i = \Phi_i(z_{i-1})$ can be nontrivial differentiable subprograms.

Instead we have access to (Jacobian-free!) tangents

$$\dot{Z}_i = \dot{\Phi}_i(z_{i-1}) \cdot \dot{Z}_{i-1} \equiv \Phi_i'(z_{i-1}) \cdot \dot{Z}_{i-1}$$

and adjoints

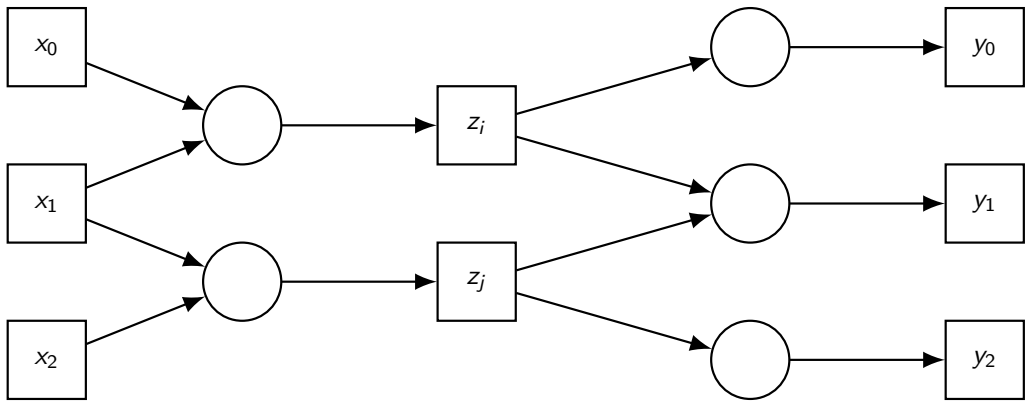
$$\bar{Z}_{i-1} = \bar{Z}_i \cdot \Phi_i'(z_{i-1}) \equiv \bar{Z}_i \cdot \bar{\Phi}_i(z_{i-1}) \ .$$

The Φ_i induce computational graphs (V_i, E_i) for $i = 1, \ldots, D$ such that

$$\mathrm{fma}(\dot{\Phi}_i(z_{i-1}) \cdot v) = \mathrm{fma}(w \cdot \bar{\Phi}_i(z_{i-1})) = |E_i|$$

for $v \in \boldsymbol{R}^{n_i}$ and $w \in \boldsymbol{R}^{1 \times m_i}$.

Let $y = f(x) = \Phi_2(\Phi_1(x))$ have the following structure:



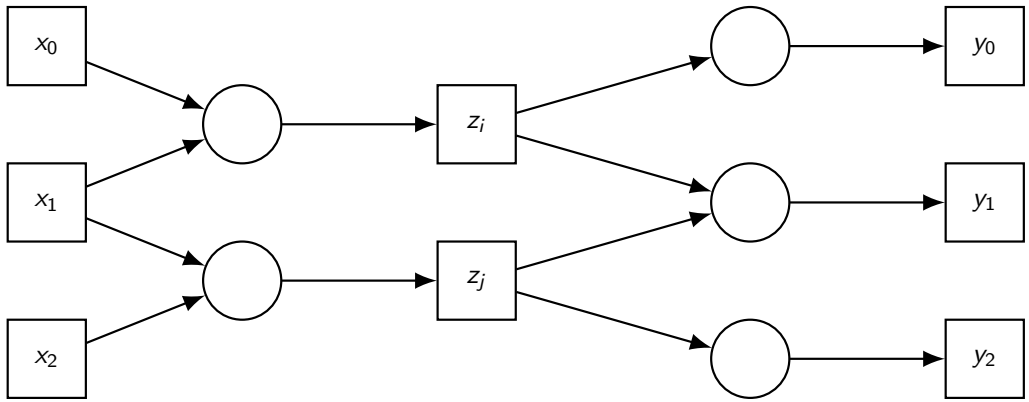
- $\dot{\Phi}_2 \cdot (\dot{\Phi}_1 \cdot I_n)$: $n \cdot (|E_1| + |E_2|) = 39$
- $(I_m \cdot \bar{\Phi}_2) \cdot \bar{\Phi}_1$: $m \cdot (|E_1| + |E_2|) = 39$
- $(\dot{\Phi}_2 \cdot I_{n_1}) \cdot (I_{n_1} \cdot \bar{\Phi}_1)$: $2 \cdot |E_1| + 2 \cdot |E_2| + 8 = 34$
- $(\dot{\Phi}_2 \cdot I_{n_1}) \cdot \bar{\Phi}_1$: $2 \cdot |E_2| + 3 \cdot |E_1| = 32$

Note: Column compression applied to Φ_1' yields optimal cost as

$\Phi_2' \cdot \Phi_1' \hat{=}$

$$\begin{pmatrix} x & \\ x & x \\ & x \end{pmatrix} \begin{pmatrix} x & x & \\ & x & x \end{pmatrix}$$

The generalized Jacobian chain product for



yields the following search space:

$$
\begin{aligned}
f' &= \dot{\Phi}_2 \cdot \Phi'_1 = \dot{\Phi}_2 \cdot (\dot{\Phi}_1 \cdot I_{n_0}) && = \dot{\Phi}_2 \cdot \Phi'_1 = \dot{\Phi}_2 \cdot (I_{n_1} \cdot \bar{\Phi}_1) \\
&= \Phi'_2 \cdot \bar{\Phi}_1 = (I_{n_2} \cdot \bar{\Phi}_2) \cdot \bar{\Phi}_1 && = \Phi'_2 \cdot \bar{\Phi}_1 = (\dot{\Phi}_2 \cdot I_{n_1}) \cdot \bar{\Phi}_1 \\
&= \Phi'_2 \cdot \Phi'_1 = (\dot{\Phi}_2 \cdot I_{n_1}) \cdot (I_{n_1} \cdot \bar{\Phi}_1) && = \Phi'_2 \cdot \Phi'_1 = (I_{n_2} \cdot \bar{\Phi}_2) \cdot (I_{n_1} \cdot \bar{\Phi}_1) \\
&= \Phi'_2 \cdot \Phi'_1 = (I_{n_2} \cdot \bar{\Phi}_2) \cdot (\dot{\Phi}_1 \cdot I_{n_0}) && = \Phi'_2 \cdot \Phi'_1 = (\dot{\Phi}_2 \cdot I_{n_1}) \cdot (\dot{\Phi}_1 \cdot I_{n_0}) \,.
\end{aligned}
$$

Theorem

Generalized Jacobian Chain Product *can be solved by dynamic programming as follows:*

$$fma_{j,i} = \begin{cases} |E_j| \cdot \min\{n_j, m_j\} & j = i \\ \min_{i \le k < j} \left\{ \min \left\{ \begin{array}{l} fma_{j,k+1} + fma_{k,i} + m_j \cdot m_k \cdot n_i, \\ fma_{j,k+1} + m_j \cdot \sum_{\nu=i}^{k} |E_\nu|, \\ fma_{k,i} + n_i \cdot \sum_{\nu=k+1}^{j} |E_\nu| \end{array} \right. \right\} & j > i \,. \end{cases}$$

Proof: (12)

We consider randomly generated problem instances of growing size.

Tangent	Adjoint	Preaccumulation	Optimum
3,708	5,562	**2,618**	**1,344**
1,283,868	1,355,194	1,687,575	**71,668**
3,677,565	44,866,293	40,880,996	**1,471,636**
585,023,794	1,496,126,424	1,196,618,622	**9,600,070**
21,306,718,862	19,518,742,454	**1,027,696,225**	**149,147,898**

Factors of up to 60 are observed.

(12) U. N.: Toward Generalized Jacobian Chaining. Under review. (arXiv:2003.05755)

(12+) `https://github.com/un110076/ADMission`

Outline

Theorem

Limited Memory Generalized Jacobian Chain Product *is NP-complete.*

Proof: (13)

DAG Reversal (14) needs to be solved to minimize the fma cost while staying feasible with respect to memory requirement. The development of suitable heuristics is work in progress → N. Nguyen.

(13) U. N.: Generalized Jacobian Chaining. Under review.

(14) U. N.: DAG Reversal is NP-Complete. J. Discr. Alg. 7 (4), 402-410, 2009.

Theorem

Optimal Jacobian Accumulation *is NP-complete.*

Proof: (15)

Generalized elimination techniques on computational graphs (16) are the subject of ongoing research and development → E. Schneidereit.

(15) U. N.: Optimal Jacobian Accumulation is NP-Complete. Math. Prog. 112 (2), 427-441, 2008.

(16) U. N.: Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. Math. Prog. 99 (3), 399-421, 2004.

Objective:

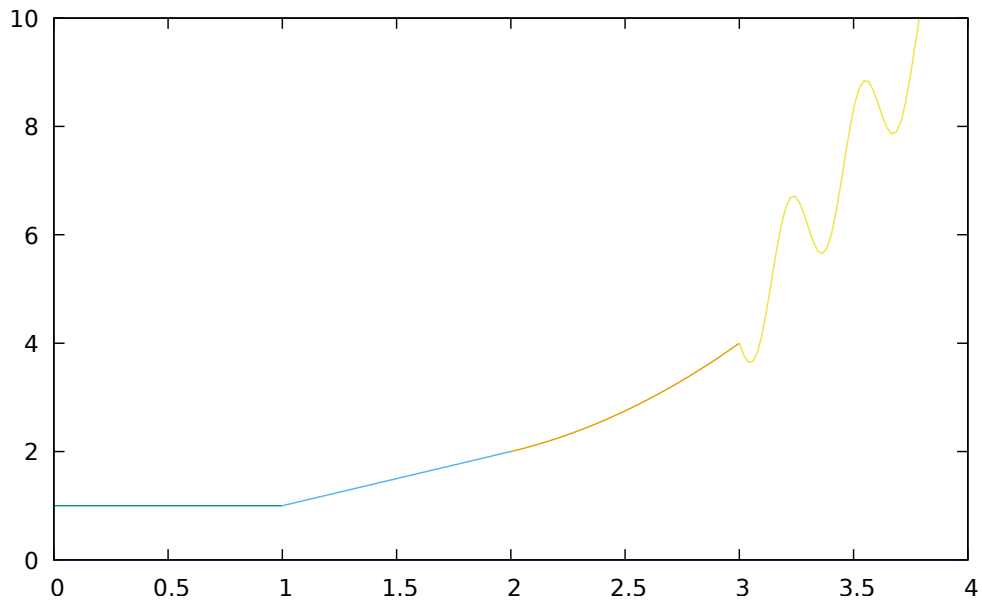
Avoid uniform random sampling across all dimensions by exploring special properties of F over subdomains.

Adaptive Samples:

Decrease sampling density for locally quasi-constant, quasi-linear, quasi-quadratic, etc. target programs.

Adaptive Surrogates:

Replace network locally by custom Taylor model.



F is quasi-linear over a (sub-)domain $[\mathrm{x}] = [(x_k)] \subseteq \boldsymbol{R}^n$ if

$$\max_{j=0,\ldots,m-1, i=0,\ldots,n-1} w\left(\frac{dF_j}{dx_i}([\mathrm{x}])\right) \leq \epsilon$$

for $0 < \epsilon \ll 1$.

F is quasi-linear with respect to x_i over the same (sub-)domain if

$$\max_{j=0,\ldots,m-1} w\left(\frac{dF_j}{dx_i}([\mathrm{x}])\right) \leq \epsilon\ .$$

Tests for quasi-linearity, -quadraticity, etc. are embedded into an exploratory domain decomposition method.

Application to practically relevant problems is work in progress → S. Afghan, S. Christodoulou.

Summary

Motivation and Problem Description

- Neural Networks as Surrogate Models
- AD

Reducing Size of Neural Networks

- Pruning
- Interval Adjoint Significance Analysis
- Results

Reducing Cost of Differentiation of Neural Networks

- Generalized Jacobian Chaining
- Dynamic Programming
- Results

Outlook

- Generalized Jacobian Chain Product with Limited Memory
- AD Mission Planning
- Adaptive Sampling / Adaptive Surrogates