A Functional-Programming Framework for Deep Learning

Jeffrey Mark Siskind, qobi@purdue.edu

Purdue University
Elmore Family School of Electrical and Computer Engineering

Meta
Thursday 16 December 2021

Joint work with Barak Avrum Pearlmutter
Our Work

AD is easier in functional programs.

Siskind (Elmore Family School of ECE, Purdue)
Our Work

AD in functional programs.
AD in functional programs.

AD is easier in functional programs.
Jeff, I was poking around Stalin seeing if it might make sense for the higher-order algorithmic differentiation stuff we’ve been doing. Which I want to do if at all possible because of the natural name of the resultant system: Stalin∇, pronounced Stalingrad.

[...]

--Barak.
Barak - [...]

Stalingrad is a cool name!

[...]

I think that adding AD to Scheme/Stalin is very important. So I’d like to help out on that any way that I can.

[...]

Jeff (http://www.neci.nj.nec.com/homepages/qobi)
Automatic Differentiation in Machine Learning: a Survey

Atılım Güneş Baydin
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ, United Kingdom

Barak A. Pearlmutter
Department of Computer Science
National University of Ireland Maynooth
Maynooth, Co. Kildare, Ireland

Alexey Andreyevich Radul
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, United States

Jeffrey Mark Siskind
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, United States

Abstract

Derivatives, mostly in the form of gradients and Hessians, are ubiquitous in machine learning. Automatic differentiation (AD), also called algorithmic differentiation or simply “autodiff”, is a family of techniques similar to but more general than backpropagation for efficiently and accurately evaluating derivatives of numeric functions expressed as computer programs. AD is a small but established field with applications in areas including computational fluid dynamics, atmospheric sciences, and engineering design optimization. Until very recently, the fields of machine learning and AD have largely been unaware of each other and, in some cases, have independently discovered each other’s results. Despite its relevance, general-purpose AD has been missing from the machine learning toolbox, a situation slowly changing with its ongoing adoption under the names “dynamic computational graphs” and “differentiable programming”. We survey the intersection of AD and machine learning, cover applications where AD has direct relevance, and address the main implementation techniques. By precisely defining the main differentiation techniques and their interrelationships, we aim to bring clarity to the usage of the terms “autodiff”, “automatic differentiation”, and “symbolic differentiation” as these are encountered more and more in machine learning settings.

Keywords:
Backpropagation, Differentiable Programming

© 2018 Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/ Attribution requirements are provided at http://jmlr.org/papers/v18/17-468.html.
Hi. I am the survey/review editor for the Machine Learning journal. Peter assigned me your paper MACH-D-15-00174 "Automatic differentiation in machine learning: a survey".

This is more of an advocacy paper than a survey paper. That’s not a problem, but it does mean the criteria for acceptance should be different (more of a case has to be made) than with a survey. I’m not familiar with automatic differentiation so I wanted a preliminary expert opinion before I officially assigned it to three reviewers. The comments I got were interesting:

I went through this paper --- while AD is interesting, and the authors try to build background, context, etc., and provide connections to various areas, the concrete examples they have illustrated involve a trivially small number of variables, and are thus going to have the opposite effect intended: i.e., after reading this paper, people will be turned away from being excited about AD --- hence, exactly the opposite of the desired effect will happen. Ultimately, AD methods and techniques deserve to be better and more widely known in machine learning, but I think this particular paper does not deliver on that goal in a context sensitive way. Much more work is needed to make it very sharply focused towards machine learning.

Hence, it its present form, this paper is not yet ready for review (though, I maintain, AD for ML is a relevant topic).

I realize this isn’t a formal review but I think it’s important to pass these comments back for your consideration. My suggestion is to retract the paper, address the concerns, and re-submit it. But if you have a strong disagreement with the reviewer’s comments I’m willing to take this into consideration and go ahead with the assignment. Please let me know how you want to proceed.

Regards,
-Tom Fawcett
Jeffrey Mark Siskind
Purdue University

Based on funding mandates

<table>
<thead>
<tr>
<th>TITLE</th>
<th>CITED BY</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic differentiation in machine learning: a survey</td>
<td>1254</td>
<td>2018</td>
</tr>
<tr>
<td>AG Baydin, BA Pearlmutter, AA Radul, JM Siskind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Journal of machine learning research 18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A computational study of cross-situational techniques for learning word-to-meaning mappings</td>
<td>656</td>
<td>1996</td>
</tr>
<tr>
<td>JM Siskind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cognition 61 (1-2), 39-91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The role of exposure to isolated words in early vocabulary development</td>
<td>580</td>
<td>2001</td>
</tr>
<tr>
<td>MR Brent, JM Siskind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cognition 81 (2), B33-B44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image segmentation with ratio cut</td>
<td>437</td>
<td>2003</td>
</tr>
<tr>
<td>S Wang, JM Siskind</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We show that reverse-mode AD (Automatic Differentiation)—a generalized gradient-calculation operator—can be incorporated as a first-class function in an augmented lambda calculus, and therefore into a functional-programming language. Closure is achieved, in that the new operator can be applied to any expression in the augmented language, yielding an expression in that language. This
this is really a quasi-accept! -- please upload as a revision your final PDF, and then I’ll move to accept it.

I was really hoping for a third review from an expert in the mathematical aspects of the topic (to balance the first reviewer, who expresses enthusiasm but warns that he is not equipped to follow all the details), but I think it is time to admit defeat on this point and move the article forward regardless. Both of the reviews that I have judge the work as excellent and recommend accepting the paper with only minor notational changes and expositional improvements. I am therefore very pleased to recommend acceptance for publication in TOPLAS.

Please pay particular attention to Reviewer 1’s request to give the reader more help, throughout, with mathematical background and concepts -- this will greatly increase the article’s potential readership.

This is a paper that could be published either in TOMS or TOPLAS, I reckon. Given that it has been submitted to TOPLAS, I recommend that the authors provide a lot more background material on the mathematical aspects of their work.

I got confused trying to read your lambda expressions---and I have been reading them for over 30 years. The fact that you don’t use dots after the variable names is one hurdle

All in all, a very interesting piece of highly cross-disciplinary work. If this were published in TOMS, you would have to spend a lot more time explaining lambda calculus to that audience. Here, for TOPLAS, you need to spend more time leading the reader through the heavy-duty mathematics.
Recent projects, as Swift for TensorFlow (S4TF) (www.tensorflow.org/swift) and Julia Zygote [15], in the spirit of the seminal “Lambda the Ultimate Backpropagator” (LTUB) [21], advocate AD as a first-class construct in a general-purpose programming language, and aim to take advantage of traditional compiler optimizations for efficient code generation.

The second is a dependently typed version of an idea behind “Lambda the Ultimate Backpropagator” [21], itself inspired by the view of functions as closures whose tangents are the tangents of their environments.

A key insight from the LTUB work [21], leading to an efficient solution, has been this:

LTUB originally presented an AD system for a dynamically typed language.

We presented the marriage of ideas behind Elliot’s categorical presentation of AD [11] and the seminal LTUB work [21].
Using Polyvariant Union-Free Flow Analysis to Compile a Higher-Order Functional-Programming Language with a First-Class Derivative Operator to Efficient Fortran-like Code

Jeffrey Mark Siskind
School of Electrical and Computer Engineering
Purdue University, USA
qobi@purdue.edu

Barak A. Pearlmutter
Hamilton Institute
NUI Maynooth, Ireland
barak@cs.nuim.ie

Abstract
We exhibit an aggressive optimizing compiler for a functional-programming language which includes a first-class forward automatic differentiation (AD) operator. The compiler’s performance is competitive with FORTRAN-based systems on our numerical examples, despite the potential inefficiencies entailed by support of a functional-programming language and a first-class AD operator. These results are achieved by combining (1) a novel formulation of forward AD in terms of a reflexive mechanism that supports first-class nestable nonstandard interpretation with (2) the migration to compile-time of the conceptually run-time nonstandard interpretation by whole-program inter-procedural flow analysis.

Consider some stereotypical numerical code and its associated execution model. Numerical code typically does not use union types and thus its execution model does not use tags or tag dispatching. In numerical code, all aggregate data typically has fixed size and shape that can be determined at compile time. Thus in the execution model, such aggregate data is unboxed and does not require indirection for data access and run-time allocation and reclamation. Numerical code typically does not use higher-order functions. Thus in the execution model, all function calls are to known targets and do not involve indirection or closures. Numerical code is typically written in languages that do not support reflection so code cannot be accessed, modified or created during execution. We refer to such code and its corresponding execution model as FORTRAN-like.

Where you can read about this work

- PLDI 2008, 2009
- ICFP 2008
Where you can read about this work

- PLDI 2008, 2009
- ICFP 2008
Evaluation: C. Weak paper, but it will not be an embarrassment to have it in POPL.
Confidence: Z. I am an informed outsider and tried my best to understand the paper.

==== Summary ====

Shows how to optimize a functional language with a built-in automatic differentiation operator.

==== Detailed Comments ====

The results look useful, but I wonder whether POPL is the right place to present them. Yes, the development involves functional programming. But it also involves a lot of concepts from scientific computing that may be unfamiliar to many and that are explained only minimally or not at all.
Review #2

[...]

I would like:

[...]

2. Comparison with ADIFOR, and

[...]

ADIFOR should be included and compared in Table 1.

[...]

Review #3

[...]

It would be nice to have some Fortran code to compare to as well.

[...]

It would also be interesting to know, as a reference, how does the approach compare with hand-written Fortran code?

[...]
AD in Fortran: Implementation via Prepreprocessor

Alexey Radul, Barak A. Pearlmutter, and Jeffrey Mark Siskind

Abstract  We describe an implementation of the Farfel Fortran77 AD extensions (Radul et al. AD in Fortran, Part 1: Design (2012), http://arxiv.org/abs/1203.1448). These extensions integrate forward and reverse AD directly into the programming
Perturbation confusion in forward automatic differentiation of higher-order functions

OLEKSANDR MANZYUK\textsuperscript{1}, BARAK A. PEARLMUTTER\textsuperscript{1, 10}, ALEXEY ANDREYEVICH RADUL\textsuperscript{2} and DAVID R. RUSH\textsuperscript{3}

Department of Computer Science and Hamilton Institute, Maynooth University, Co. Kildare, Ireland
\textup{(e-mails: manzyuk@gmail.com, barak@pearlmutter.net, axch@alum.mit.edu, kumoyuki@gmail.com)}

JEFFREY MARK SISKIND\textsuperscript{10}

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-2035, USA
\textup{(e-mail: qobi@purdue.edu)}

Abstract

Automatic differentiation (AD) is a technique for augmenting computer programs to compute deriva-
AD for Probabilistic Programming

Jeffrey Mark Siskind
qobi@purdue.edu

School of Electrical and Computer Engineering
Purdue University

Probabilistic Programming
Universal Languages, Systems and Applicationss
NIPS
13 December 2008

Joint work with Barak A. Pearlmutter.
The tension between convenience and performance in automatic differentiation

Jeffrey Mark Siskind, qobi@purdue.edu

Purdue University

NIPS 2016 Workshop on
The Future of Gradient-Based Machine Learning Software
Saturday 10 December 2016

Joint work with Barak Pearlmutter
Divide-and-Conquer Checkpointing for Arbitrary Programs with No User Annotation

Jeffrey Mark Siskind, qobi@purdue.edu

NIPS 2017 Workshop on
The Future of Gradient-Based Machine Learning Software & Techniques
Saturday 9 December 2017

Joint work with Barak Avrum Pearlmutter
Scheme as a framework for Deep Learning

Jeffrey Mark Siskind, qobi@purdue.edu

ICFP 2021 Scheme Workshop
Friday 27 August 2021

Joint work with Barak Avrum Pearlmutter and Hamad Ahmed
News update

An automated circuit-design and layout tool with the unlikely name of MacPitts [Electronics, Feb. 10, 1982, p. 48] has borne out the expectations of its developers at the Massachusetts Institute of Technology's Lincoln Laboratory in Lexington, Mass. An 8-bit n-channel MOS automatic-gain controller for speech was designed in two weeks, instead of six months, and is now being fabricated, says Jeffrey M. Siskind, chief developer of the design system. A 16-bit microprocessor is ready for fabrication as well.

Siskind is so bullish, in fact, that he left the lab in September to start up his own design service for very large-scale integrated parts, MetaLogic Inc., in Bedford, Mass. It will market a system similar to MacPitts, called MetaSyn, and may also act as a broker for silicon foundries.

"The technology is now in the public domain," Siskind says. "As software, it is not patentable." Nonetheless, MIT has copyrighted it and is licensing it for noncommercial, domestic use only, according to Siskind's former superior, Peter Blankenship, associate leader of the laboratory's speech systems technology group.

—Marilyn A. Harris
A Neural Network

<table>
<thead>
<tr>
<th>Layer</th>
<th>θ</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_0$</td>
<td>$\frac{\partial x_8}{\partial x_0}$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$\frac{\partial x_8}{\partial x_1}$</td>
</tr>
<tr>
<td>2</td>
<td>$x_2$</td>
<td>$\frac{\partial x_8}{\partial x_2}$</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$\frac{\partial x_8}{\partial x_3}$</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$\frac{\partial x_8}{\partial x_4}$</td>
</tr>
<tr>
<td>5</td>
<td>$x_5$</td>
<td>$\frac{\partial x_8}{\partial x_5}$</td>
</tr>
<tr>
<td>6</td>
<td>$x_6$</td>
<td>$\frac{\partial x_8}{\partial x_6}$</td>
</tr>
<tr>
<td>7</td>
<td>$x_7$</td>
<td>$\frac{\partial x_8}{\partial x_7}$</td>
</tr>
<tr>
<td>8</td>
<td>$x_8$</td>
<td>$\frac{\partial x_8}{\partial x_8}$</td>
</tr>
</tbody>
</table>
A Neural Network is a (Functional) Program

\[
\text{net} \left[ \theta_0, \ldots, \theta_7 \right] \left[ w_0, \ldots, w_7 \right] x_0 \triangleq \\
\text{let } x_1 = \text{layer}_0 \theta_0 w_0 x_0 \\\nx_2 = \text{layer}_1 \theta_1 w_1 x_1 \\\nx_3 = \text{layer}_2 \theta_2 w_2 x_2 \\\nx_4 = \text{layer}_3 \theta_3 w_3 x_3 \\\nx_5 = \text{layer}_4 \theta_4 w_4 x_4 \\\nx_6 = \text{layer}_5 \theta_5 w_5 x_5 \\\nx_7 = \text{layer}_6 \theta_6 w_6 x_6 \\\nx_8 = \text{layer}_7 \theta_7 w_7 x_7 \\
\text{in } x_8
\]
A (Functional) Program

\[ f \begin{bmatrix} w_0, w_1 \end{bmatrix} \begin{bmatrix} x_0, x_1 \end{bmatrix} \triangleq \]

\begin{align*}
\text{let} & \quad t_0 = w_0 \times x_0 \\
& \quad t_1 = w_1 \times x_1 \\
& \quad y = t_0 + t_1 \\
\text{in} & \quad y
\end{align*}
A (Functional) Program is a (Neural) Network

\[
f \left[ w_0, w_1 \right] \left[ x_0, x_1 \right] \triangleq \\
\text{let } t_0 = w_0 \times x_0 \\
\text{      } t_1 = w_1 \times x_1 \\
\text{      } y = t_0 + t_1 \\
\text{in } y
\]
A (Functional) Program is a (Neural) Network

$$f \ [w_0, w_1] \ [x_0, x_1] \triangleq$$

let

$t_0 = w_0 \times x_0$

$t_1 = w_1 \times x_1$

$y = t_0 + t_1$

in $y$
A (Functional) Program is a (Neural) Network

\[ f \left[ w_0, w_1 \right] \left[ x_0, x_1 \right] \triangleq \]
\[
\text{let } t_0 = w_0 \times x_0 \\
t_1 = w_1 \times x_1 \\
y = t_0 + t_1 \\
\text{in } y \]
A (Functional) Program is a (Neural) Network

$$f \ [w_0, w_1] \ [x_0, x_1] \triangleq$$

let \( t_0 = w_0 \times x_0 \) \\
\( t_1 = w_1 \times x_1 \) \\
\( y = t_0 + t_1 \) \\

in \( y \)
\( f \left[ w_0, w_1 \right] \left[ x_0, x_1 \right] \triangleq \)

```latex
\textbf{let} \quad t_0 &= w_0 \times x_0 \\
            t_1 &= w_1 \times x_1 \\
            y &= t_0 + t_1 \\
\textbf{in} \quad y
```

Siskind (Elmore Family School of ECE, Purdue)
Some Observations

Deep learning network 'frameworks' are domain specific (functional) programming languages.

A deep neural network is a long running (functional) program.

Can perform backpropagation on (functional) programs by having an execution of the program generate a network. This is called reverse-mode automatic differentiation (AD).
Some Observations

- Deep learning network ‘frameworks’ are domain specific (functional) programming languages.
Some Observations

- Deep learning network ‘frameworks’ are domain specific (functional) programming languages.
- A deep neural network is a long running (functional) program.
Some Observations

- Deep learning network ‘frameworks’ are domain specific (functional) programming languages.
- A deep neural network is a long running (functional) program.
- Can perform backpropagation on (functional) programs
Some Observations

- Deep learning network ‘frameworks’ are domain specific (functional) programming languages.
- A deep neural network is a long running (functional) program.
- Can perform backpropagation on (functional) programs by having an execution of the program generate a network.
Some Observations

- Deep learning network ‘frameworks’ are domain specific (functional) programming languages.
- A deep neural network is a long running (functional) program.
- Can perform backpropagation on (functional) programs by having an execution of the program generate a network. This is called reverse-mode automatic differentiation (AD).
1. Migrate reflective AD through partial evaluation

2. Implementing checkpointing reverse mode through CPS
1. Migrate reflective AD through partial evaluation

2. Implementing checkpointing reverse mode through CPS
Migrate reflective source-to-source transformation from run time to compile time with abstract interpretation
function \texttt{g(x)}
    \texttt{return } x+1
end

function \texttt{f(x)}
    \texttt{return } 2 \times \texttt{g(x)}
end

\ldots \texttt{derivative(f, 3)} \ldots
Traditional AD by Source-to-Source Transformation
Preprocessor at Compile Time

```plaintext
function g(x)
    return x + 1
end

function f(x)
    return 2 * g(x)
end

local y, y_tangent = f_forward(3, 1)
... y_tangent ...
```
function g(x)
    return x+1
end

function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end

local y, y_tangent = f_forward(3, 1)
... y_tangent ...
function g_forward(x, x_tangent)
    local y, y_tangent = x, x_tangent
    return x+1, x_tangent
end

function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end

local y, y_tangent = f_forward(3, 1)
... y_tangent ...

... y_tangent ...

... y_tangent ...
Source-to-Source Transformation at Run Time

Reflection

```plaintext
function f(x)
    return 2*g(x)
end
```

Siskind (Elmore Family School of ECE, Purdue) An FP Framework for Deep Learning Meta 16 December 2021 29 / 108
function f(x)
    return 2 * g(x)
end

code(f)
function f(x)
    return 2*g(x)
end

code(f) ==>
    "function f(x)
        return 2*g(x)
    end"
function f(x)
    return 2*g(x)
end

code(f) ==> "function f(x)
    return 2*g(x)
end"

transform("function f(x)
    return 2*g(x)
end")
Source-to-Source Transformation at Run Time

Reflection

```lua
function f(x)
    return 2*g(x)
end

code(f) ==>
    "function f(x)
        return 2*g(x)
    end"

transform("function f(x)
        return 2*g(x)
    end") ==>
    "function f_forward(x, x_tangent)
        local y, y_tangent = g_forward(x, x_tangent)
        return 2*y, 2*y_tangent
    end"
```

--

Siskind (Elmore Family School of ECE, Purdue)
An FP Framework for Deep Learning
Meta 16 December 2021
Source-to-Source Transformation at Run Time

Reflection

```plaintext
function f(x)
    return 2*g(x)
end

code(f) ==> "function f(x)
    return 2*g(x)
end"

transform("function f(x)
    return 2*g(x)
end") ==> "function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end"

compile("function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end")
```
function f(x)
    return 2*g(x)
end

code(f) ==> "function f(x)
    return 2*g(x)
end"

transform("function f(x)
    return 2*g(x)
end") ==> "function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end"

compile("function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end") ==> f_forward
function f(x)
    return 2*g(x)
end

code(f) ==> "function f(x)
    return 2*g(x)
end"

transform("function f(x)
    return 2*g(x)
end") ==> "function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end"

compile("function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end") ==> f_forward

called_by(f)
function f(x)
    return 2*g(x)
end

code(f) ==> "function f(x)
    return 2*g(x)
end"

transform("function f(x)
    return 2*g(x)
end") ==> "function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end"

compile("function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end") ==> f_forward

called_by(f) ==> {g}
Source-to-Source Transformation at Run Time

Reflection

```plaintext
function f(x)
    return 2*g(x)
end

code(f) ==> "function f(x)
    return 2*g(x)
end"

transform("function f(x)
    return 2*g(x)
end") ==> "function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end"

compile("function f_forward(x, x_tangent)
    local y, y_tangent = g_forward(x, x_tangent)
    return 2*y, 2*y_tangent
end") ==> f_forward

called_by(f) ==> {g}

function derivative(f, x)
    for g in called_by(f) do compile(transform(code(g))) end
    local y, y_tangent = compile(transform(code(f)))(x, 1)
    return y_tangent
end
```
But How Can We Make This Efficient?

\[\text{while not converged()} \ do \]
\[x = x - \eta \cdot \text{derivative}(f, x)\]
\[\text{end}\]
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add(x, y)
    if x:type()=='torch.Tensor' then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... add(x, y) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add(x, y)
    if x:type()=='torch.Tensor' then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = DOUBLE, y = DOUBLE
... add(x, y) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add(x, y)
    if x:type() == "torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = DOUBLE, y = DOUBLE
... add(DOUBLE, DOUBLE) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
function scalar_add(x, y)
    return x + y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i] + y[i]
    end
    return z
end

function add_1(DOUBLE, DOUBLE)
    if x:type() == "torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = DOUBLE, y = DOUBLE
... add_1(DOUBLE, DOUBLE) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_1(DOUBLE, DOUBLE)
    if DOUBLE=="torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = DOUBLE, y = DOUBLE
... add_1(DOUBLE, DOUBLE) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_1(DOUBLE, DOUBLE)
    if false then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = DOUBLE, y = DOUBLE... add_1(DOUBLE, DOUBLE) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()... add(x, y) ...
Abstract Interpretation aka (Polyvariant) Flow Analysis

```lua
function scalar_add(x, y)
    return x + y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i] + y[i]
    end
    return z
end

function add_1(DOUBLE, DOUBLE)
    return scalar_add(x, y)
end

local x = DOUBLE, y = DOUBLE... add_1(DOUBLE, DOUBLE) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()... add(x, y) ...
```
Abstract Interpretation aka (Polyvariant) Flow Analysis

```lua
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_1(DOUBLE, DOUBLE)
    return scalar_add(x, y)
end

local x = 3, y = 4
... scalar_add(x, y) ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
```
Abstract Interpretation aka (Polyvariant) Flow Analysis

```lua
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_1(DOUBLE, DOUBLE)

    return scalar_add(x, y)
end

local x = 3, y = 4
... x+y ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... add(x, y) ...
```
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add(x, y)
    if x:type()=='torch.Tensor' then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... x+y ...
local x = ARRAY, y = ARRAY
... add(x, y) ...
Abstract Interpretation aka (Polyvariant) Flow Analysis

```lua
function scalar_add(x, y)
    return x + y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i] + y[i]
    end
    return z
end

function add(x, y)
    if x:type() == "torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... x + y ...
local x = ARRAY, y = ARRAY
... add(ARRAY, ARRAY) ...
```
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_2(ARRAY, ARRAY)
    if x:type()=="torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... x+y ...
local x = ARRAY, y = ARRAY
... add_2(ARRAY, ARRAY) ...
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_2(ARRAY, ARRAY)
    if ARRAY=="torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... x+y ...
local x = ARRAY, y = ARRAY
... add_2(ARRAY, ARRAY) ...
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_2(ARRAY, ARRAY)
    if true then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... x+y ...
local x = ARRAY, y = ARRAY
... add_2(ARRAY, ARRAY) ...
Abstract Interpretation aka (Polyvariant) Flow Analysis

```lisp
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i]+y[i]
    end
    return z
end

function add_2(ARRAY, ARRAY)
    return vector_add(x, y)
end

local x = 3, y = 4
... x+y ...
local x = ARRAY, y = ARRAY
... add_2(ARRAY, ARRAY) ...
```
Abstract Interpretation aka (Polyvariant) Flow Analysis

```lua
function scalar_add(x, y)
    return x+y
end

function vector_add(x, y)
    local n = x:size(1)
    local z = torch.Tensor(n)
    for i = 1, n do
        z[i] = x[i] + y[i]
    end
    return z
end

function add(x, y)
    if x:type() == "torch.Tensor" then
        return vector_add(x, y)
    else
        return scalar_add(x, y)
    end
end

local x = 3, y = 4
... x+y ...
local x = torch.Tensor(5):zeros(), y = torch.Tensor(5):zeros()
... vector_add(x, y) ...
```
A Single Powerful Optimization

\{x = e_1, y = e_2\}.x
A Single Powerful Optimization

\{x = e_1, y = e_2\}.x \sim e_1
\{x = e1, y = e2\}.x \sim e1

- can eliminate storage allocation
A Single Powerful Optimization

{x = e1, y = e2}.x \sim e1

- can eliminate storage allocation
- can eliminate storage reclamation
A Single Powerful Optimization

\[
\{x = e_1, \ y = e_2\}.x \sim e_1
\]

- can eliminate storage allocation
- can eliminate storage reclamation
- can eliminate storage writes
A Single Powerful Optimization

\{x = e_1, y = e_2\}.x \sim e_1

- can eliminate storage allocation
- can eliminate storage reclamation
- can eliminate storage writes
- can eliminate storage reads
A Single Powerful Optimization

\{x = e_1, \ y = e_2\} \cdot x \leadsto e_1

- can eliminate storage allocation
- can eliminate storage reclamation
- can eliminate storage writes
- can eliminate storage reads
- can eliminate dead code
function map(f, x)
    y = torch.Tensor(x:size(1))
    for i = 1, x:size(1) do
        y[i] = f(x[i])
    end
    return y
end

function reduce(g, i, x)
    y = i
    for i = 1, x:size(1) do
        y = g(y, x[i])
    end
    return y
end

reduce(function(x, y) return x+y end, 
        0, 
        map(function(x) return x*x end, torch.Tensor({u, v, w, x, y})))
function map(f, x)
    y = torch.Tensor(x:size(1))
    for i = 1, x:size(1) do
        y[i] = f(x[i])
    end
    return y
end

function reduce(g, i, x)
    y = i
    for i = 1, x:size(1) do
        y = g(y, x[i])
    end
    return y
end

reduce(function(x, y) return x+y end, 0, map(function(x) return x*x end, torch.Tensor({u, v, w, x, y}))))

u*u + v*v + w*w + x*x + y*y

--
You need this anyway
to compile dynamic languages efficiently
Same mechanism can support AD
```plaintext
function f(x)
    return 2*x
end

function derivative(g, x)
    local y, y_tangent = compile(transform(code(g)))(x, 1)

    return y_tangent
end

... derivative(f, 3) ...
```
function f(x)
    return 2*x
end

function derivative_1(g, x)
    local y, y_tangent = compile(transform(code(g)))(x, 1)
    return y_tangent
end

... derivative_1(FUNCTION_F, 3) ...

function f(x)
    return 2*x
end

function derivative_1(FUNCTION_F, x)
    local y, y_tangent = compile(transform(code(FUNCTION_F)))(x, 1)

    return y_tangent
end

... derivative_1(FUNCTION_F, 3) ...

Siskind (Elmore Family School of ECE, Purdue) An FP Framework for Deep Learning Meta 16 December 2021 36/108
function f(x)
    return 2*x
end

function derivative_1(FUNCTION_F, x)
    local y, y_tangent = compile(transform("function f(x)
                                            return 2*x
                                            end")(x, 1))
    return y_tangent
end

... derivative_1(FUNCTION_F, 3) ...

Siskind (Elmore Family School of ECE, Purdue) An FP Framework for Deep Learning Meta 16 December 2021 36 / 108
function f(x)
    return 2*x
end

function derivative_1(FUNCTION_F, x)
    local y, y_tangent = compile("function f_forward(x, x_tangent)
        local y, y_tangent = 2*x, 2*x_tangent
        return y, y_tangent
    end") (x, 1)
    return y_tangent
end

... derivative_1(FUNCTION_F, 3) ...
function f(x)
    return 2*x
end

function f_forward(x, x_tangent)
    local y, y_tangent = 2*x, 2*x_tangent
    return y, y_tangent
end

function derivative_1(FUNCTION_F, x)
    local y, y_tangent = f_forward(x, 1)
    return y_tangent
end

... derivative_1(FUNCTION_F, 3) ...

Siskind (Elmore Family School of ECE, Purdue) An FP Framework for Deep Learning
function f(x)
    return 2*x
end

function f_forward(x, x_tangent)
    local y, y_tangent = 2*x, 2*x_tangent
    return y, y_tangent
end

function derivative(g, x)
    local y, y_tangent = compile(transform(code(g)))(x, 1)
    return y_tangent
end

local y, y_tangent = f_forward(x, 1) ...
    y_tangent ...

A Single Powerful Optimization

separates AD from optimization
allows simple formulation of AD transforms
(tape is a data structure in the language)
many AD optimizations (like TBR) fall out
makes it easier to get it right
makes it easier to get it to nest

Siskind (Elmore Family School of ECE, Purdue)
A Single Powerful Optimization

- separates AD from optimization
A Single Powerful Optimization

- separates AD from optimization
- allows simple formulation of AD transforms
A Single Powerful Optimization

- separates AD from optimization
- allows simple formulation of AD transforms
  (forward mode is 28 lines; reverse mode is 155 lines)
A Single Powerful Optimization

- separates AD from optimization
- allows simple formulation of AD transforms
  (forward mode is 28 lines; reverse mode is 155 lines)
- tape is a data structure (in the language)
A Single Powerful Optimization

- separates AD from optimization
- allows simple formulation of AD transforms
  (forward mode is 28 lines; reverse mode is 155 lines)
- tape is a data structure (in the language)
- many AD optimizations (like TBR) fall out
A Single Powerful Optimization

- separates AD from optimization
- allows simple formulation of AD transforms
  (forward mode is 28 lines; reverse mode is 155 lines)
- tape is a data structure (in the language)
- many AD optimizations (like TBR) fall out
- makes it easier to get it right
A Single Powerful Optimization

- separates AD from optimization
- allows simple formulation of AD transforms
  (forward mode is 28 lines; reverse mode is 155 lines)
- tape is a data structure (in the language)
- many AD optimizations (like TBR) fall out
- makes it easier to get it right
- makes it easier to get it to nest
Essence of Forward Transform

\[
\begin{align*}
\vec{c} &\sim \vec{J} \ c \\
\vec{\lambda x.e} &\sim \vec{\lambda x. e} \\
\vec{e_1 e_2} &\sim \vec{e_1 e_2}
\end{align*}
\]

letrec \( x_1 = e_1; \ldots; x_n = e_n \) in \( e \) \( \sim \) letrec \( \vec{x_1} = \vec{e_1}; \ldots; \vec{x_n} = \vec{e_n} \) in \( \vec{e} \)
Essence of Reverse Transform

\[
\begin{align*}
\vec{x} &= c \\
\vec{x_1} &= \vec{x_2} \\
\vec{x} &= \lambda x . e \\
\vec{x} &= \vec{x_1} \cdot \vec{x_2} \\
\vec{x} &= \vec{x_1} , \vec{x_2} \\
\vec{x} &= \vec{x_1} , \vec{x_2} \\
\vec{x_1} &= \vec{x_2} \\
\vec{x} &= \lambda x . e \\
\vec{x} &= \vec{x_1} \cdot \vec{x_2} \\
\vec{x} &= \vec{x_1} , \vec{x_2} + = \vec{x}
\end{align*}
\]

\[
\vec{\lambda x . \text{let } b_1 ; \ldots ; b_n \text{ in } y} \quad \Rightarrow \quad \lambda \vec{x} \cdot \text{let } \vec{b_1} ; \ldots ; \vec{b_n} \text{ in } \vec{y} , \lambda \vec{y} \cdot \text{let } \vec{b_n} ; \ldots ; \vec{b_1} \text{ in } \vec{x}
\]
Game Theory

<table>
<thead>
<tr>
<th></th>
<th>$b_1$</th>
<th>...</th>
<th>$b_j$</th>
<th>...</th>
<th>$b_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A$</td>
<td>$a_i$</td>
<td>...</td>
<td>PAYOFF($a_i, b_j$)</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>.</td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>$a_m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\max_{a \in A} \min_{b \in B} \text{PAYOFF}(a, b)$$

\[
\begin{array}{r|ccc}
\mathbb{R}^n & \cdots & \mathbf{b} & \cdots \\
\mathbb{R}^m & \mathbf{a} & \cdots & \text{PAYOFF}(\mathbf{a}, \mathbf{b}) & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}
\]

\[
\max_{\mathbf{a} \in \mathbb{R}^m} \min_{\mathbf{b} \in \mathbb{R}^n} \text{PAYOFF}(\mathbf{a}, \mathbf{b})
\]

(letrec ((loop
    (lambda (i r)
      (if (zero? i)
          r
        (loop (- i 1)
          (let* ((start (list (real 1) (real 1)))
            (f (lambda (x1 y1 x2 y2)
                (- (+ (sqr x1) (sqr y1))
                (+ (sqr x2) (sqr y2))))))
          ((list x1* y1*)
            (multivariate-argmin-F
              (lambda ((list x1 y1))
                (multivariate-max-F
                  (lambda ((list x2 y2)) (f x1 y1 x2 y2))
                  start)))
            start))
          ((list x2* y2*)
            (multivariate-argmax-F
              (lambda ((list x2 y2)) (f x1* y1* x2 y2))
              start)))
        (list (list (write-real x1*) (write-real y1*))
          (list (write-real x2*) (write-real y2*)))))
    (loop (real 1000) (list (list (real 0) (real 0)) (list (real 0) (real 0))))))
potential: \( p(x; w) = \| x - (10, 10 - w) \|^{-1} + \| x - (10, 0) \|^{-1} \)

\[
\begin{align*}
\ddot{x}(t) &= -\nabla_x p(x)|_{x=x(t)} \\
\dot{x}(t + \Delta t) &= \dot{x}(t) + \Delta t \ddot{x}(t) \\
x(t + \Delta t) &= x(t) + \Delta t \dot{x}(t)
\end{align*}
\]

When: \( x_1(t + \Delta t) \leq 0 \)

let: \( \Delta t_f = -x_1(t)/\dot{x}_1(t) \)

\[
\begin{align*}
t_f &= t + \Delta t_f \\
x(t_f) &= x(t) + \Delta t_f \dot{x}(t)
\end{align*}
\]

Error: \( E(w) = x_0(t_f)^2 \)

Find: \( \arg\min_w E(w) \)


(define (naive-euler w)
  (let* ((charges
              (list (list (real 10) (- (real 10) w)) (list (real 10) (real 0))))
         (x-initial (list (real 0) (real 8)))
         (xdot-initial (list (real 0.75) (real 0)))
         (delta-t (real 1e-1))
         (p (lambda (x)
              (reduce + (real 0))
              (map (lambda (c) (/ (real 1) (distance x c))) charges))))
    (letrec ((loop (lambda (x xdot)
                    (let* ((xdot (k*v (real -1) ((gradient-F p) x)))
                           (x-new (v+ x (k*v delta-t xdot)))
                           (if (positive? (list-ref x-new 1))
                               (loop x-new xdot (k*v delta-t xddot))
                               (let* ((delta-t-f (/ (- (real 0) (list-ref x 1))
                                               (list-ref xdot 1)))
                                      (x-t-f (v+ x (k*v delta-t-f xdot)))
                                      (sqr (list-ref x-t-f 0)))))))
               (loop x-initial xdot-initial))))

(letrec ((loop (lambda (i r)
                (if (zero? i)
                    r
                    (loop (- i 1)
                          (let* ((w0 (real 0))
                                 (list w*)
                                 (multivariate-argmin-F
                                  (lambda ((list w)) (naive-euler w)) (list w0)))
                                 (write-real w*)))))
               (loop (real 1000) (real 0)))
\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]
Probabilistic Lambda Calculus

\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \leftrightarrow \text{true}) &= p_0 \\
\Pr(x_0 \leftrightarrow \text{false}) &= 1 - p_0 \\
\Pr(x_1 \leftrightarrow \text{true}) &= p_1 \\
\Pr(x_1 \leftrightarrow \text{false}) &= 1 - p_1
\end{align*}
\]


Siskind (Elmore Family School of ECE, Purdue) An FP Framework for Deep Learning Meta 16 December 2021 44/108
$P = \textbf{if } x_0 \ \textbf{then } 0 \ \textbf{else if } x_1 \ \textbf{then } 1 \ \textbf{else } 2$

\[
\begin{align*}
\Pr(x_0 \mapsto \textbf{true}) &= p_0 \\
\Pr(x_1 \mapsto \textbf{true}) &= p_1 \\
\Pr(x_0 \mapsto \textbf{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \textbf{false}) &= 1 - p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(\mathcal{E}(P) = 0|p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1|p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2|p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

Probabilistic Lambda Calculus

\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 \\
\Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1 \\
\Pr(x_1 \mapsto \text{false}) &= 1 - p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(\mathcal{E}(P) = 0 | p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1 | p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2 | p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

\[
\prod_{v \in \{0, 1, 2, 2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = p_0(1 - p_0)^3p_1(1 - p_1)^2
\]

\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \leftrightarrow \text{true}) &= p_0 \\
\Pr(x_0 \leftrightarrow \text{false}) &= 1 - p_0 \\
\Pr(x_1 \leftrightarrow \text{true}) &= p_1 \\
\Pr(x_1 \leftrightarrow \text{false}) &= 1 - p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(E(P) = 0|p_0, p_1) &= p_0 \\
\Pr(E(P) = 1|p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(E(P) = 2|p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

\[
\prod_{v \in \{0,1,2,2\}} \Pr(E(P) = v|p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2
\]

\[
\arg\max_{p_0, p_1} \prod_{v \in \{0,1,2,2\}} \Pr(E(P) = v|p_0, p_1) = \left\{ \frac{1}{4}, \frac{1}{3} \right\}
\]

Probabilistic Prolog

\( p(0). \)
\( p(X) \leftarrow \neg q(X). \)
\( q(1). \)
\( q(2). \)
\[ Pr(p(0).) = p_0 \]
\[ Pr(p(X) :- q(X).) = 1 - p_0 \]
\[ Pr(q(1).) = p_1 \]
\[ Pr(q(2).) = 1 - p_1 \]
\[
\begin{align*}
\Pr(p(0)) &= p_0 \\
\Pr(p(x) \leftarrow \neg q(x)) &= 1 - p_0 \\
\Pr(q(1)) &= p_1 \\
\Pr(q(2)) &= 1 - p_1 \\
\Pr(?-p(0)) &= p_0 \\
\Pr(?-p(1)) &= (1 - p_0)p_1 \\
\Pr(?-p(2)) &= (1 - p_0)(1 - p_1)
\end{align*}
\]
\[ \Pr(p(0).) = p_0 \]
\[ \Pr(p(X) : -q(X).) = 1 - p_0 \]
\[ \Pr(q(1).) = p_1 \]
\[ \Pr(q(2).) = 1 - p_1 \]

\[ \Pr(?-p(0).) = p_0 \]
\[ \Pr(?-p(1).) = (1 - p_0)p_1 \]
\[ \Pr(?-p(2).) = (1 - p_0)(1 - p_1) \]

\[ \prod_{q \in \{p(0), p(1), p(2), p(2)\}} \Pr(?-q.) = p_0(1 - p_0)^3 p_1(1 - p_1)^2 \]
Probabilistic Prolog

\[
\begin{align*}
\Pr(p(0).) &= p_0 \\
\Pr(p(X) :- q(X).) &= 1 - p_0 \\
\Pr(q(1).) &= p_1 \\
\Pr(q(2).) &= 1 - p_1 \\
\Pr(?-p(0).) &= p_0 \\
\Pr(?-p(1).) &= (1 - p_0)p_1 \\
\Pr(?-p(2).) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

\[
\prod_{q \in \{p(0), p(1), p(2), p(3)\}} \Pr(?-q.) = p_0(1 - p_0)^3 p_1(1 - p_1)^2
\]

\[
\arg\max_{p_0, p_1} \prod_{q \in \{p(0), p(1), p(2), p(3)\}} \Pr(?-q.) = \left(\frac{1}{4}, \frac{1}{3}\right)
\]
Probabilistic Lambda Calculus

(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate
           (if x0 then 0 else if x1 then 1 else 2
           (list Pr(x0 \rightarrow true) = p0  Pr(x0 \rightarrow false) = 1 - p0
                  Pr(x1 \rightarrow true) = p1  Pr(x1 \rightarrow false) = 1 - p1
                  ...)))))
   (map-reduce
    * 1.0
    (lambda (value)
      (likelihood value tagged-distribution))
    '(0 1 2 2)))
  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
     (evaluate if x0 then 0 else if x1 then 1 else 2
       (list Pr(x0 \(\mapsto\) true) = p0  \(\Pr(x_0 \mapsto \text{true}) = p_0\)
         Pr(x0 \(\mapsto\) false) = 1 \(-\) p0
         \(\Pr(x_0 \mapsto \text{false}) = 1 - p_0\)
         Pr(x1 \(\mapsto\) true) = p1  \(\Pr(x_1 \mapsto \text{true}) = p_1\)
         Pr(x1 \(\mapsto\) false) = 1 \(-\) p1
         \(\Pr(x_1 \mapsto \text{false}) = 1 - p_1\)
         ...
       )))
     (map-reduce
      \ast
      1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '('(0 1 2 2)))
      '('(0.5 0.5)
      1000.0
      0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
           (evaluate if x0 then 0 else if x1 then 1 else 2
             (list Pr(x0 \rightarrow \text{true}) = p_0 \quad Pr(x0 \rightarrow \text{false}) = 1 - p_0
                Pr(x_1 \rightarrow \text{true}) = p_1 \quad Pr(x_1 \rightarrow \text{false}) = 1 - p_1
                \ldots)))
     (map-reduce*
       1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '(0.5 0.5)
       1000.0
       0.1))
  Siskind (Elmore Family School of ECE, Purdue)
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
       (evaluate (if x0 then 0 else if x1 then 1 else 2
             (list Pr(x0 \rightarrow true) = p0  Pr(x0 \rightarrow false) = 1 - p0
                  Pr(x1 \rightarrow true) = p1  Pr(x1 \rightarrow false) = 1 - p1
             ...
             ))))

     (map-reduce *
       1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '(0 1 2 2)))

     '(0.5 0.5)
     1000.0
     0.1))
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
       (evaluate if x0 then 0 else if x1 then 1 else 2
       (list Pr(x0 \rightarrow true) = p0  Pr(x0 \rightarrow false) = 1 - p0
            Pr(x1 \rightarrow true) = p1  Pr(x1 \rightarrow false) = 1 - p1
            ...)))))

   (map-reduce
    * 1.0
    (lambda (value)
      (likelihood value tagged-distribution))
    '(0 1 2 2)))))

  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
      (evaluate if x_0 then 0 else if x_1 then 1 else 2
        (list Pr(x_0 \rightarrow \text{true}) = p_0 \quad Pr(x_0 \rightarrow \text{false}) = 1 - p_0
          Pr(x_1 \rightarrow \text{true}) = p_1 \quad Pr(x_1 \rightarrow \text{false}) = 1 - p_1
          \ldots))))
      (map-reduce
        * 1.0
        (lambda (value)
          (likelihood value tagged-distribution))
        '(0 1 2 2)))))
    '(0.5 0.5)
    1000.0
    0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
           (evaluate
               if x0 then 0 else if x1 then 1 else 2
               (list
                Pr(x0 \rightarrow \text{true}) = p_0 \quad Pr(x0 \rightarrow \text{false}) = 1 - p_0
                Pr(x1 \rightarrow \text{true}) = p_1 \quad Pr(x1 \rightarrow \text{false}) = 1 - p_1
                ...)))))
    (map-reduce
      * 1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
            (evaluate
              if x₀ then 0 else if x₁ then 1 else 2
              (list
                Pr(x₀ \to \text{true}) = p₀ \quad Pr(x₀ \to \text{false}) = 1 - p₀
                Pr(x₁ \to \text{true}) = p₁ \quad Pr(x₁ \to \text{false}) = 1 - p₁
                ...
              )))
     (map-reduce
      * 1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1)
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
     (evaluate
       (if x_0 then 0 else if x_1 then 1 else 2
       (list
         Pr(x_0 \mapsto true) = p_0
         Pr(x_0 \mapsto false) = 1 \cdot p_0
         Pr(x_1 \mapsto true) = p_1
         Pr(x_1 \mapsto false) = 1 \cdot p_1
         ...
       ))
     (map-reduce
       *
       1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '((0.5 0.5)
         1000.0
         0.1))
   )))
)
(gradient-ascent
(lambda (p)
  (let ((tagged-distribution
    (evaluate
      (if x0 then 0 else if x1 then 1 else 2
        (list
          Pr(x0 -> true) = p0  Pr(x0 -> false) = 1 - p0
          Pr(x1 -> true) = p1  Pr(x1 -> false) = 1 - p1
          ...))))

    (map-reduce
      * 1.0
      (lambda (value)
        (likelihood value tagged-distribution))
        '(0 1 2 2)))
  '(0.5 0.5)
  1000.0
  0.1))
(gradient-ascent
 (lambda (p)
   (let ((clauses (list
     Pr(p(0).) = p0
     Pr(p(X) :- q(X).) = 1 - p0
     Pr(q(1).) = p1
     Pr(q(2).) = 1 - p1)))
   (map-reduce
    * 1.0
    (lambda (query)
      (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2))))
  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
      Pr(p(0).) = p_0
      Pr(p(X) :- q(X).) = 1 - p_0
      Pr(q(1).) = p_1
      Pr(q(2).) = 1 - p_1))
     (map-reduce
      * 1.0
      (lambda (query)
        (likelihood (proof-distribution query clauses)))
      'p(0) p(1) p(2) p(2)))
    '0.5 0.5
    1000.0
    0.1)
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
      Pr(p(0).) = p₀
      Pr(p(X) :- q(X).) = 1 − p₀
      Pr(q(1).) = p₁
      Pr(q(2).) = 1 − p₁))))

    (map-reduce *
      1.0
      (lambda (query)
        (likelihood (proof-distribution query clauses)))
      '(p(0) p(1) p(2) p(2)))

      '(0.5 0.5)
      1000.0
      0.1))
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
                   Pr(p(0).) = p0
                   Pr(p(X) :- q(X).) = 1 − p0
                   Pr(q(1).) = p1
                   Pr(q(2).) = 1 − p1)))
      (map-reduce
       * 1.0
       (lambda (query)
         (likelihood (proof-distribution query clauses))))
       '(p(0) p(1) p(2) p(2)))
       '(0.5 0.5)
       1000.0
       0.1)
  )
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
                     (Pr(p(0).) = p0
                     Pr(p(X) :- q(X).) = 1 - p0
                     Pr(q(1).) = p1
                     Pr(q(2).) = 1 - p1))))
      (map-reduce
       *
       1.0
       (lambda (query)
         (likelihood (proof-distribution query clauses)))
       '(p(0) p(1) p(2) p(2))))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
                    Pr(p(0).) = p0
                    Pr(p(X) :- q(X).) = 1 - p0
                    Pr(q(1).) = p1
                    Pr(q(2).) = 1 - p1)))))

  (map-reduce
    *
    1.0
    (lambda (query)
      (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))
  '(0.5 0.5)
  1000.0
  0.1)
Probabilistic Prolog

(gradient-ascent
 (lambda (p)
   (let ((clauses (list
     Pr(p(0).) = p0
     Pr(p(X) :- q(X).) = 1 - p0
     Pr(q(1).) = p1
     Pr(q(2).) = 1 - p1))
     (map-reduce *
       1.0
       (lambda (query)
         (likelihood (proof-distribution query clauses)))
       '(p(0) p(1) p(2) p(2)))
   )))

'(0.5 0.5)
1000.0
0.1)
static void f2679(double a_f2679_0, double a_f2679_1, double a_f2679_2, double a_f2679_3) {
    int t272381 = ((a_f2679_2 == 0.) ? 0 : 1);
    double t272406;
    double t272405;
    double t272404;
    double t272403;
    double t272402;
    if ((t272381 == 0)) {
        double t272480 = (1. - a_f2679_0);
        double t272572 = (1. - a_f2679_1);
        double t273043 = (a_f2679_0 + 0.);
        double t274185 = (t272480 * a_f2679_1);
        double t274426 = (t274185 + 0.);
        double t275653 = (t272480 * t272572);
        double t275894 = (t275653 + 0.);
        double t277121 = (t272480 * t272572);
        double t277362 = (t277121 + 0.);
        double t277431 = (t277362 + 1.);
        double t277436 = (t275894 * t277431);
        double t277441 = (t274426 * t277436);
        double t277446 = (t273043 * t277441);
    ...
    double t1777107 = (t1774696 + t1715394);
    double t1777194 = (0. - t1745420);
    double t1778533 = (t1777194 + t1419700);
    t272406 = a_f2679_0;
    t272405 = a_f2679_1;
    t272404 = t277446;
    t272403 = t1778533;
    t272402 = t1777107;
    } else {...
    r_f2679_0 = t272406;
    r_f2679_1 = t272405;
    r_f2679_2 = t272404;
    r_f2679_3 = t272403;
    r_f2679_4 = t272402;"}
<table>
<thead>
<tr>
<th>Language</th>
<th>Tools</th>
<th>backprop</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fs</td>
<td>Fv</td>
</tr>
<tr>
<td>VLAD</td>
<td>1.00</td>
<td>□</td>
</tr>
<tr>
<td>Fortran</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADIFOR</td>
<td>15.51</td>
<td>3.35 □</td>
</tr>
<tr>
<td>TAPENADE</td>
<td>14.97</td>
<td>5.97 □</td>
</tr>
<tr>
<td>C</td>
<td>ADIC</td>
<td>22.75 □</td>
</tr>
<tr>
<td>C++</td>
<td>ADOL-C</td>
<td>12.16 □</td>
</tr>
<tr>
<td>Cppad</td>
<td>54.74</td>
<td>□</td>
</tr>
<tr>
<td>FADBAD++</td>
<td>132.31</td>
<td>46.01 □</td>
</tr>
<tr>
<td>ML</td>
<td>MLTON</td>
<td>95.20 □</td>
</tr>
<tr>
<td>OCAML</td>
<td>202.01</td>
<td>□</td>
</tr>
<tr>
<td>SML/NJ</td>
<td>181.93</td>
<td>□</td>
</tr>
<tr>
<td>Haskell</td>
<td>GHC</td>
<td>□</td>
</tr>
<tr>
<td>Scheme</td>
<td>Bigloo</td>
<td>743.26 □</td>
</tr>
<tr>
<td>Chicken</td>
<td>1626.73</td>
<td>□</td>
</tr>
<tr>
<td>Gambit</td>
<td>671.54</td>
<td>□</td>
</tr>
<tr>
<td>Ikarus</td>
<td>279.59</td>
<td>□</td>
</tr>
<tr>
<td>Larceny</td>
<td>1203.34</td>
<td>□</td>
</tr>
<tr>
<td>MIT Scheme</td>
<td>2446.33</td>
<td>□</td>
</tr>
<tr>
<td>MzC</td>
<td>1318.60</td>
<td>□</td>
</tr>
<tr>
<td>MzScheme</td>
<td>1364.14</td>
<td>□</td>
</tr>
<tr>
<td>Scheme-&gt;C</td>
<td>597.67</td>
<td>□</td>
</tr>
<tr>
<td>Scmutils</td>
<td>5889.26</td>
<td>□</td>
</tr>
<tr>
<td>Stalin</td>
<td>435.82</td>
<td>□</td>
</tr>
<tr>
<td></td>
<td>particle</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>FF</td>
<td>FR</td>
</tr>
<tr>
<td>VLAD</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>STALIN(\n)</td>
<td>2.05</td>
<td>5.44</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>ADIFOR</td>
<td>5.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADIC</td>
<td>93.32</td>
</tr>
<tr>
<td>C</td>
<td>ADOL-C</td>
<td>78.13</td>
</tr>
<tr>
<td>ML</td>
<td>OCAML</td>
<td>217.03</td>
</tr>
<tr>
<td></td>
<td>MLTON</td>
<td>153.01</td>
</tr>
<tr>
<td>C++</td>
<td>ADOL-C</td>
<td>209.44</td>
</tr>
<tr>
<td>SCHEME</td>
<td>BIGLOO</td>
<td>627.78</td>
</tr>
<tr>
<td></td>
<td>CHICKEN</td>
<td>1453.06</td>
</tr>
<tr>
<td></td>
<td>GAMBIT</td>
<td>578.94</td>
</tr>
<tr>
<td></td>
<td>IKARUS</td>
<td>266.54</td>
</tr>
<tr>
<td></td>
<td>LARCENY</td>
<td>964.18</td>
</tr>
<tr>
<td></td>
<td>MIT SCHEME</td>
<td>2025.23</td>
</tr>
<tr>
<td></td>
<td>SCMUTILS</td>
<td>1243.08</td>
</tr>
<tr>
<td></td>
<td>MZC</td>
<td>1309.82</td>
</tr>
<tr>
<td></td>
<td>MZSCHEME</td>
<td>582.20</td>
</tr>
<tr>
<td></td>
<td>SCHEME-(-&gt;C)</td>
<td>4462.83</td>
</tr>
</tbody>
</table>

Siskind (Elmore Family School of ECE, Purdue)
<table>
<thead>
<tr>
<th></th>
<th>Statistics</th>
<th>probabilistic-lambda-calculus</th>
<th>probabilistic-prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>R</td>
</tr>
<tr>
<td>VLAD</td>
<td>STALIN\n</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>ADIFOR</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td></td>
<td>TAPENADE</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>C</td>
<td>ADIC</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>C++</td>
<td>ADOL-C</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td></td>
<td>CppAD</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td></td>
<td>FADBAD++</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>ML</td>
<td>MLTON</td>
<td>129.11</td>
<td>114.88</td>
</tr>
<tr>
<td></td>
<td>OCAML</td>
<td>249.40</td>
<td>499.43</td>
</tr>
<tr>
<td></td>
<td>SML/NJ</td>
<td>234.62</td>
<td>258.53</td>
</tr>
<tr>
<td>HASKELL</td>
<td>GHC</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>SCHEME</td>
<td>BIGLOO</td>
<td>983.12</td>
<td>1016.50</td>
</tr>
<tr>
<td></td>
<td>CHICKEN</td>
<td>2324.54</td>
<td>3040.44</td>
</tr>
<tr>
<td></td>
<td>GAMBIT</td>
<td>1033.46</td>
<td>1107.26</td>
</tr>
<tr>
<td></td>
<td>IKARUS</td>
<td>497.48</td>
<td>517.89</td>
</tr>
<tr>
<td></td>
<td>LARCENY</td>
<td>1658.27</td>
<td>1606.44</td>
</tr>
<tr>
<td></td>
<td>MIT SCHEME</td>
<td>4130.88</td>
<td>3817.57</td>
</tr>
<tr>
<td></td>
<td>MzC</td>
<td>2294.93</td>
<td>2346.13</td>
</tr>
<tr>
<td></td>
<td>MzSCHEME</td>
<td>2721.35</td>
<td>2625.21</td>
</tr>
<tr>
<td></td>
<td>SCHEME-&gt;C</td>
<td>811.37</td>
<td>803.22</td>
</tr>
<tr>
<td></td>
<td>SCMUTILS</td>
<td>7699.14</td>
<td>83656.17</td>
</tr>
<tr>
<td></td>
<td>STALIN</td>
<td>956.47</td>
<td>1994.44</td>
</tr>
</tbody>
</table>
Take-Home Message

Powerful and efficient AD can be attained by:

- integrating AD into compiler
- formulating AD as one of many compiler transformations
- using abstract interpretation to migrate AD transformation from run time to compile time
1. Migrate reflective AD through partial evaluation

2. Implementing checkpointing reverse mode through CPS
A (Brief) History of Backpropagation aka Reverse-Mode AD


B. Speelpenning, Compiling Fast Partial Derivatives of Functions Given by Algorithms, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.


S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (in Finnish), Department of Computer Science, University of Helsinki, 1970.

A.E. Bryson, Jr. and Y.-C. Ho, Applied optimal control, Blaisdell, 1969.

B. Speelpenning, *Compiling Fast Partial Derivatives of Functions Given by Algorithms*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.

B. Speelpenning, *Compiling Fast Partial Derivatives of Functions Given by Algorithms*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.

A (Brief) History of Backpropagation aka Reverse-Mode AD


B. Speelpenning, *Compiling Fast Partial Derivatives of Functions Given by Algorithms*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.


A (Brief) History of Backpropagation aka Reverse-Mode AD


B. Speelpenning, *Compiling Fast Partial Derivatives of Functions Given by Algorithms*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.


S. Linnainmaa, *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors* (in Finnish), Department of Computer Science, University of Helsinki, 1970.
A (Brief) History of Backpropagation aka Reverse-Mode AD


B. Speelpenning, *Compiling Fast Partial Derivatives of Functions Given by Algorithms*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.


S. Linnainmaa, *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors* (in Finnish), Department of Computer Science, University of Helsinki, 1970.

Evaluating a Neural Network

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 w_0 \\
\hline
\text{x}_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
\text{layer}_1 & \theta_1 w_1 \\
\hline
\text{x}_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
\text{layer}_2 & \theta_2 w_2 \\
\hline
\text{x}_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
\text{layer}_3 & \theta_3 w_3 \\
\hline
\text{x}_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
\text{layer}_4 & \theta_4 w_4 \\
\hline
\text{x}_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
\text{layer}_5 & \theta_5 w_5 \\
\hline
\text{x}_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
\text{layer}_6 & \theta_6 w_6 \\
\hline
\text{x}_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
\text{layer}_7 & \theta_7 w_7 \\
\hline
\text{x}_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
\text{layer}_8 & \theta_8 w_8 \\
\hline
\text{x}_8 & \frac{\partial x_8}{\partial x_8}
\end{array}
\]
Evaluating a Neural Network

\[
\begin{array}{c|cc}
\text{layer}_0 & \theta_0 & w_0 \\
\hline
x_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
\text{layer}_1 & \theta_1 & w_1 \\
\hline
x_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
\text{layer}_2 & \theta_2 & w_2 \\
\hline
x_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
\text{layer}_3 & \theta_3 & w_3 \\
\hline
x_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
\text{layer}_4 & \theta_4 & w_4 \\
\hline
x_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
\text{layer}_5 & \theta_5 & w_5 \\
\hline
x_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
\text{layer}_6 & \theta_6 & w_6 \\
\hline
x_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
\text{layer}_7 & \theta_7 & w_7 \\
\hline
x_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
\end{array}
\]
Evaluating a Neural Network

\[
\begin{array}{c|c}
\text{layer} & x_0 \quad \frac{\partial x_8}{\partial x_0} \\
0 & \theta_0 \quad w_0 \\
1 & x_1 \quad \frac{\partial x_8}{\partial x_1} \\
2 & \theta_1 \quad w_1 \\
3 & x_2 \quad \frac{\partial x_8}{\partial x_2} \\
4 & \theta_2 \quad w_2 \\
5 & x_3 \quad \frac{\partial x_8}{\partial x_3} \\
6 & \theta_3 \quad w_3 \\
7 & x_4 \quad \frac{\partial x_8}{\partial x_4} \\
8 & \theta_4 \quad w_4 \\
9 & x_5 \quad \frac{\partial x_8}{\partial x_5} \\
10 & \theta_5 \quad w_5 \\
11 & x_6 \quad \frac{\partial x_8}{\partial x_6} \\
12 & \theta_6 \quad w_6 \\
13 & x_7 \quad \frac{\partial x_8}{\partial x_7} \\
14 & \theta_7 \quad w_7 \\
15 & x_8 \quad \frac{\partial x_8}{\partial x_8}
\end{array}
\]
Evaluating a Neural Network

\[
\begin{align*}
\text{layer}_0 \quad \theta_0 \quad w_0 \\
x_0 \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 \quad \theta_1 \quad w_1 \\
x_1 \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 \quad \theta_2 \quad w_2 \\
x_2 \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 \quad \theta_3 \quad w_3 \\
x_3 \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 \quad \theta_4 \quad w_4 \\
x_4 \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 \quad \theta_5 \quad w_5 \\
x_5 \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 \quad \theta_6 \quad w_6 \\
x_6 \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 \quad \theta_7 \quad w_7 \\
x_7 \quad \frac{\partial x_8}{\partial x_7} \\
\end{align*}
\]
Evaluating a Neural Network

\[
\begin{array}{c|c}
\text{layer}_0 & x_0 \frac{\partial x_8}{\partial x_0} \\
\theta_0 & w_0 \\
\hline
\text{layer}_1 & x_1 \frac{\partial x_8}{\partial x_1} \\
\theta_1 & w_1 \\
\hline
\text{layer}_2 & x_2 \frac{\partial x_8}{\partial x_2} \\
\theta_2 & w_2 \\
\hline
\text{layer}_3 & x_3 \frac{\partial x_8}{\partial x_3} \\
\theta_3 & w_3 \\
\hline
\text{layer}_4 & x_4 \frac{\partial x_8}{\partial x_4} \\
\theta_4 & w_4 \\
\hline
\text{layer}_5 & x_5 \frac{\partial x_8}{\partial x_5} \\
\theta_5 & w_5 \\
\hline
\text{layer}_6 & x_6 \frac{\partial x_8}{\partial x_6} \\
\theta_6 & w_6 \\
\hline
\text{layer}_7 & x_7 \frac{\partial x_8}{\partial x_7} \\
\theta_7 & w_7 \\
\hline
\text{layer}_8 & x_8 \frac{\partial x_8}{\partial x_8} \\
\end{array}
\]
Evaluating a Neural Network

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 \ w_0 \\
& x_0 \ \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \theta_1 \ w_1 \\
& x_1 \ \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \theta_2 \ w_2 \\
& x_2 \ \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \theta_3 \ w_3 \\
& x_3 \ \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \theta_4 \ w_4 \\
& x_4 \ \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \theta_5 \ w_5 \\
& x_5 \ \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \theta_6 \ w_6 \\
& x_6 \ \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \theta_7 \ w_7 \\
& x_7 \ \frac{\partial x_8}{\partial x_7} \\
\end{array}
\]
Evaluating a Neural Network
Evaluating a Neural Network

\[
\begin{array}{c|c}
\text{layer}_0 & x_0 \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & x_1 \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & x_2 \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & x_3 \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & x_4 \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & x_5 \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & x_6 \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & x_7 \frac{\partial x_8}{\partial x_7} \\
\end{array}
\]
Evaluating a Neural Network

<table>
<thead>
<tr>
<th>Layer</th>
<th>θ</th>
<th>w</th>
<th>( \frac{\partial x_8}{\partial x_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_0 )</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
</tr>
<tr>
<td>( l_1 )</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>( l_2 )</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
</tr>
<tr>
<td>( l_3 )</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
</tr>
<tr>
<td>( l_4 )</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
</tr>
<tr>
<td>( l_5 )</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>( l_6 )</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
</tr>
<tr>
<td>( l_7 )</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
</tr>
</tbody>
</table>

\( x_8 \) = \( \frac{\partial x_8}{\partial x_8} \)
Evaluating a Neural Network

<table>
<thead>
<tr>
<th></th>
<th>$x_0$</th>
<th>$\frac{\partial x_8}{\partial x_0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 0</td>
<td>$\theta_0$</td>
<td>$w_0$</td>
</tr>
<tr>
<td></td>
<td>$x_1$</td>
<td>$\frac{\partial x_8}{\partial x_1}$</td>
</tr>
<tr>
<td>Layer 1</td>
<td>$\theta_1$</td>
<td>$w_1$</td>
</tr>
<tr>
<td></td>
<td>$x_2$</td>
<td>$\frac{\partial x_8}{\partial x_2}$</td>
</tr>
<tr>
<td>Layer 2</td>
<td>$\theta_2$</td>
<td>$w_2$</td>
</tr>
<tr>
<td></td>
<td>$x_3$</td>
<td>$\frac{\partial x_8}{\partial x_3}$</td>
</tr>
<tr>
<td>Layer 3</td>
<td>$\theta_3$</td>
<td>$w_3$</td>
</tr>
<tr>
<td></td>
<td>$x_4$</td>
<td>$\frac{\partial x_8}{\partial x_4}$</td>
</tr>
<tr>
<td>Layer 4</td>
<td>$\theta_4$</td>
<td>$w_4$</td>
</tr>
<tr>
<td></td>
<td>$x_5$</td>
<td>$\frac{\partial x_8}{\partial x_5}$</td>
</tr>
<tr>
<td>Layer 5</td>
<td>$\theta_5$</td>
<td>$w_5$</td>
</tr>
<tr>
<td></td>
<td>$x_6$</td>
<td>$\frac{\partial x_8}{\partial x_6}$</td>
</tr>
<tr>
<td>Layer 6</td>
<td>$\theta_6$</td>
<td>$w_6$</td>
</tr>
<tr>
<td></td>
<td>$x_7$</td>
<td>$\frac{\partial x_8}{\partial x_7}$</td>
</tr>
<tr>
<td>Layer 7</td>
<td>$\theta_7$</td>
<td>$w_7$</td>
</tr>
<tr>
<td></td>
<td>$x_8$</td>
<td>$\frac{\partial x_8}{\partial x_8}$</td>
</tr>
</tbody>
</table>
### Evaluating a Neural Network

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta_i )</th>
<th>( w_i )</th>
<th>( x_i )</th>
<th>( \frac{\partial x_i}{\partial x_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_8} )</td>
</tr>
</tbody>
</table>
### Evaluating a Neural Network

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
<th>( x )</th>
<th>( \frac{\partial x_i}{\partial x_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
</tbody>
</table>

\( x_8 \)
Evaluating a Neural Network

\[
\begin{array}{c|c}
\text{layer}_0 & x_0 \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & x_1 \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & x_2 \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & x_3 \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & x_4 \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & x_5 \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & x_6 \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & x_7 \frac{\partial x_8}{\partial x_7} \\
\end{array}
\]
Evaluating a Neural Network

\[
\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
\text{layer}_7 & \quad \theta_7 \quad w_7
\end{align*}
\]
Evaluating a Neural Network

layer_0 \( \theta_0 \ w_0 \)

layer_1 \( \theta_1 \ w_1 \)

layer_2 \( \theta_2 \ w_2 \)

layer_3 \( \theta_3 \ w_3 \)

layer_4 \( \theta_4 \ w_4 \)

layer_5 \( \theta_5 \ w_5 \)

layer_6 \( \theta_6 \ w_6 \)

layer_7 \( \theta_7 \ w_7 \)
### Evaluating a Neural Network

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( layer_0 )</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
</tr>
<tr>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
<td></td>
</tr>
<tr>
<td>( layer_1 )</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
<td></td>
</tr>
<tr>
<td>( layer_2 )</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
<td></td>
</tr>
<tr>
<td>( layer_3 )</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
<td></td>
</tr>
<tr>
<td>( layer_4 )</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
<td></td>
</tr>
<tr>
<td>( layer_5 )</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
<td></td>
</tr>
<tr>
<td>( layer_6 )</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
</tr>
<tr>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
<td></td>
</tr>
<tr>
<td>( layer_7 )</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
</tr>
<tr>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
<td></td>
</tr>
<tr>
<td>( layer_8 )</td>
<td>( \theta_8 )</td>
<td>( w_8 )</td>
</tr>
<tr>
<td>( x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_8} )</td>
<td></td>
</tr>
</tbody>
</table>
Evaluating a Neural Network

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_0$</td>
<td>$w_0$</td>
<td>$x_0$</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>$w_1$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>$\theta_2$</td>
<td>$w_2$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>$\theta_3$</td>
<td>$w_3$</td>
<td>$x_3$</td>
</tr>
<tr>
<td>$\theta_4$</td>
<td>$w_4$</td>
<td>$x_4$</td>
</tr>
<tr>
<td>$\theta_5$</td>
<td>$w_5$</td>
<td>$x_5$</td>
</tr>
<tr>
<td>$\theta_6$</td>
<td>$w_6$</td>
<td>$x_6$</td>
</tr>
<tr>
<td>$\theta_7$</td>
<td>$w_7$</td>
<td>$x_7$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_8$</td>
</tr>
</tbody>
</table>
Evaluating a Network
Evaluating a Network

\[ y = x_0 w_0 + x_1 w_1 \]

Meta 16 December 2021 56 / 108
Evaluating a Network

\[ y = w_0 \times x_0 + w_1 \times x_1 \]
Evaluating a Network

\[ w_0 y + x_0 w_1 x_1 t_0 + t_1 = y \]
Evaluating a Network

\[
\begin{align*}
    w_0 & \times x_0 \\
    w_1 & \times x_1 \\
    t_0 & \times t_1 \\
    + & \\
    y & 
\end{align*}
\]
Evaluating a Network

\[ w_0 x_0 + w_1 x_1 \]

\[ t_0 + t_1 \]

\[ y \]
Evaluating a Network

\[ w_0 x_0 + w_1 x_1 = y \]

Siskind (Elmore Family School of ECE, Purdue)
Some Observations

Only need to store live variables.

Most deep learning frameworks store all intermediate variables to allow subsequent backpropagation.
Some Observations

- Only need to store live variables.
Some Observations

- Only need to store live variables.
- Most deep learning frameworks store all intermediate variables to allow subsequent backpropagation.
Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
<th>( x )</th>
<th>( \frac{\partial x_8}{\partial x_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>\frac{\partial x_8}{\partial x_0}</td>
</tr>
<tr>
<td>1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>\frac{\partial x_8}{\partial x_1}</td>
</tr>
<tr>
<td>2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>\frac{\partial x_8}{\partial x_2}</td>
</tr>
<tr>
<td>3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>\frac{\partial x_8}{\partial x_3}</td>
</tr>
<tr>
<td>4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>\frac{\partial x_8}{\partial x_4}</td>
</tr>
<tr>
<td>5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>\frac{\partial x_8}{\partial x_5}</td>
</tr>
<tr>
<td>6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>\frac{\partial x_8}{\partial x_6}</td>
</tr>
<tr>
<td>7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>\frac{\partial x_8}{\partial x_7}</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>( x_8 )</td>
<td>\frac{\partial x_8}{\partial x_8}</td>
</tr>
</tbody>
</table>
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[ \begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
x_0 & \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
x_1 & \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
x_2 & \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
x_3 & \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
x_4 & \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
x_5 & \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
x_6 & \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
x_7 & \quad \frac{\partial x_8}{\partial x_7} \\
x_8 & \quad \frac{\partial x_8}{\partial x_8}
\end{align*} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[ \begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
\text{layer}_7 & \quad \theta_7 \quad w_7
\end{align*} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer</th>
<th>θ</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>θ₀</td>
<td>w₀</td>
</tr>
<tr>
<td>1</td>
<td>θ₁</td>
<td>w₁</td>
</tr>
<tr>
<td>2</td>
<td>θ₂</td>
<td>w₂</td>
</tr>
<tr>
<td>3</td>
<td>θ₃</td>
<td>w₃</td>
</tr>
<tr>
<td>4</td>
<td>θ₄</td>
<td>w₄</td>
</tr>
<tr>
<td>5</td>
<td>θ₅</td>
<td>w₅</td>
</tr>
<tr>
<td>6</td>
<td>θ₆</td>
<td>w₆</td>
</tr>
<tr>
<td>7</td>
<td>θ₇</td>
<td>w₇</td>
</tr>
</tbody>
</table>

\[ \frac{\partial x_8}{\partial x_0}, \frac{\partial x_8}{\partial x_1}, \ldots, \frac{\partial x_8}{\partial x_7}, \frac{\partial x_8}{\partial x_8} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer} \; 0 & x_0 \frac{\partial x_8}{\partial x_0} \\
\theta_0 & w_0 \\
\text{layer} \; 1 & x_1 \frac{\partial x_8}{\partial x_1} \\
\theta_1 & w_1 \\
\text{layer} \; 2 & x_2 \frac{\partial x_8}{\partial x_2} \\
\theta_2 & w_2 \\
\text{layer} \; 3 & x_3 \frac{\partial x_8}{\partial x_3} \\
\theta_3 & w_3 \\
\text{layer} \; 4 & x_4 \frac{\partial x_8}{\partial x_4} \\
\theta_4 & w_4 \\
\text{layer} \; 5 & x_5 \frac{\partial x_8}{\partial x_5} \\
\theta_5 & w_5 \\
\text{layer} \; 6 & x_6 \frac{\partial x_8}{\partial x_6} \\
\theta_6 & w_6 \\
\text{layer} \; 7 & x_7 \frac{\partial x_8}{\partial x_7} \\
\theta_7 & w_7 \\
\text{layer} \; 8 & x_8 \frac{\partial x_8}{\partial x_8} \\
\end{array}
\]
### Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$\frac{\partial x_8}{\partial x_0}$</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>$\theta_0 w_0$</td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>$\theta_1 w_1$</td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\theta_2 w_2$</td>
<td></td>
</tr>
<tr>
<td>$x_4$</td>
<td>$\theta_3 w_3$</td>
<td></td>
</tr>
<tr>
<td>$x_5$</td>
<td>$\theta_4 w_4$</td>
<td></td>
</tr>
<tr>
<td>$x_6$</td>
<td>$\theta_5 w_5$</td>
<td></td>
</tr>
<tr>
<td>$x_7$</td>
<td>$\theta_6 w_6$</td>
<td></td>
</tr>
<tr>
<td>$x_8$</td>
<td>$\theta_7 w_7$</td>
<td></td>
</tr>
</tbody>
</table>

---

Siskind (Elmore Family School of ECE, Purdue)
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[ \frac{\partial x_8}{\partial x_0} \]

\[ \frac{\partial x_8}{\partial x_1} \]

\[ \frac{\partial x_8}{\partial x_2} \]

\[ \frac{\partial x_8}{\partial x_3} \]

\[ \frac{\partial x_8}{\partial x_4} \]

\[ \frac{\partial x_8}{\partial x_5} \]

\[ \frac{\partial x_8}{\partial x_6} \]

\[ \frac{\partial x_8}{\partial x_7} \]

\[ \frac{\partial x_8}{\partial x_8} \]
<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{layer}_0 )</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
</tr>
<tr>
<td>( \text{layer}_1 )</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>( \text{layer}_2 )</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
</tr>
<tr>
<td>( \text{layer}_3 )</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
</tr>
<tr>
<td>( \text{layer}_4 )</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
</tr>
<tr>
<td>( \text{layer}_5 )</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
</tr>
<tr>
<td>( \text{layer}_6 )</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
</tr>
<tr>
<td>( \text{layer}_7 )</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
</tr>
</tbody>
</table>

\( x_0 \) \( \frac{\partial x_8}{\partial x_0} \)

\( x_1 \) \( \frac{\partial x_8}{\partial x_1} \)

\( x_2 \) \( \frac{\partial x_8}{\partial x_2} \)

\( x_3 \) \( \frac{\partial x_8}{\partial x_3} \)

\( x_4 \) \( \frac{\partial x_8}{\partial x_4} \)

\( x_5 \) \( \frac{\partial x_8}{\partial x_5} \)

\( x_6 \) \( \frac{\partial x_8}{\partial x_6} \)

\( x_7 \) \( \frac{\partial x_8}{\partial x_7} \)

\( x_8 \) \( \frac{\partial x_8}{\partial x_8} \)
## Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer i</th>
<th>( \theta_i )</th>
<th>( w_i )</th>
<th>( x_{i+1} )</th>
<th>( \frac{\partial x_{i+1}}{\partial x_i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer 0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_0}{\partial x_0} )</td>
</tr>
<tr>
<td>layer 1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_1}{\partial x_1} )</td>
</tr>
<tr>
<td>layer 2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_2}{\partial x_2} )</td>
</tr>
<tr>
<td>layer 3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_3}{\partial x_3} )</td>
</tr>
<tr>
<td>layer 4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_4}{\partial x_4} )</td>
</tr>
<tr>
<td>layer 5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_5}{\partial x_5} )</td>
</tr>
<tr>
<td>layer 6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_6}{\partial x_6} )</td>
</tr>
<tr>
<td>layer 7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_7}{\partial x_7} )</td>
</tr>
</tbody>
</table>

\[ \frac{\partial x_8}{\partial x_8} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

layer_0 \( \theta_0 w_0 \)

layer_1 \( \theta_1 w_1 \)

layer_2 \( \theta_2 w_2 \)

layer_3 \( \theta_3 w_3 \)

layer_4 \( \theta_4 w_4 \)

layer_5 \( \theta_5 w_5 \)

layer_6 \( \theta_6 w_6 \)

layer_7 \( \theta_7 w_7 \)
Evaluating a Neural Network to Allow Subsequent Backpropagation

layer_0 \theta_0 w_0

layer_1 \theta_1 w_1

layer_2 \theta_2 w_2

layer_3 \theta_3 w_3

layer_4 \theta_4 w_4

layer_5 \theta_5 w_5

layer_6 \theta_6 w_6

layer_7 \theta_7 w_7

x_0 \frac{\partial x_8}{\partial x_0}

x_1 \frac{\partial x_8}{\partial x_1}

x_2 \frac{\partial x_8}{\partial x_2}

x_3 \frac{\partial x_8}{\partial x_3}

x_4 \frac{\partial x_8}{\partial x_4}

x_5 \frac{\partial x_8}{\partial x_5}

x_6 \frac{\partial x_8}{\partial x_6}

x_7 \frac{\partial x_8}{\partial x_7}

x_8 \frac{\partial x_8}{\partial x_8}
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 \ w_0 \\
\hline
\text{layer}_1 & \theta_1 \ w_1 \\
\hline
\text{layer}_2 & \theta_2 \ w_2 \\
\hline
\text{layer}_3 & \theta_3 \ w_3 \\
\hline
\text{layer}_4 & \theta_4 \ w_4 \\
\hline
\text{layer}_5 & \theta_5 \ w_5 \\
\hline
\text{layer}_6 & \theta_6 \ w_6 \\
\hline
\text{layer}_7 & \theta_7 \ w_7 \\
\hline
x_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
x_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
x_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
x_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
x_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
x_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
x_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
x_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
x_8 & \frac{\partial x_8}{\partial x_8}
\end{array}
\]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 w_0 \\
\hline
\text{x}_0 & \partial_{x_0} x_0 \\
\hline
\text{layer}_1 & \theta_1 w_1 \\
\hline
\text{x}_1 & \partial_{x_1} x_1 \\
\hline
\text{layer}_2 & \theta_2 w_2 \\
\hline
\text{x}_2 & \partial_{x_2} x_2 \\
\hline
\text{layer}_3 & \theta_3 w_3 \\
\hline
\text{x}_3 & \partial_{x_3} x_3 \\
\hline
\text{layer}_4 & \theta_4 w_4 \\
\hline
\text{x}_4 & \partial_{x_4} x_4 \\
\hline
\text{layer}_5 & \theta_5 w_5 \\
\hline
\text{x}_5 & \partial_{x_5} x_5 \\
\hline
\text{layer}_6 & \theta_6 w_6 \\
\hline
\text{x}_6 & \partial_{x_6} x_6 \\
\hline
\text{layer}_7 & \theta_7 w_7 \\
\hline
\text{x}_7 & \partial_{x_7} x_7 \\
\hline
\text{layer}_8 & \theta_8 w_8 \\
\hline
\text{x}_8 & \partial_{x_8} x_8 \\
\end{array}
\]
### Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
<th>( \frac{\partial x_8}{\partial x} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_0 )</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
</tr>
<tr>
<td>1</td>
<td>( x_1 )</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>2</td>
<td>( x_2 )</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
</tr>
<tr>
<td>3</td>
<td>( x_3 )</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
</tr>
<tr>
<td>4</td>
<td>( x_4 )</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
</tr>
<tr>
<td>5</td>
<td>( x_5 )</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
</tr>
<tr>
<td>6</td>
<td>( x_6 )</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
</tr>
<tr>
<td>7</td>
<td>( x_7 )</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
</tr>
<tr>
<td>8</td>
<td>( x_8 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluating a Neural Network to Allow Subsequent Backpropagation

layer_0 \theta_0 w_0
\n\nlayer_1 \theta_1 w_1
\n\nlayer_2 \theta_2 w_2
\n\nlayer_3 \theta_3 w_3
\n\nlayer_4 \theta_4 w_4
\n\nlayer_5 \theta_5 w_5
\n\nlayer_6 \theta_6 w_6
\n\nlayer_7 \theta_7 w_7
\n\n\frac{\partial x_8}{\partial x_0}
\frac{\partial x_8}{\partial x_1}
\frac{\partial x_8}{\partial x_2}
\frac{\partial x_8}{\partial x_3}
\frac{\partial x_8}{\partial x_4}
\frac{\partial x_8}{\partial x_5}
\frac{\partial x_8}{\partial x_6}
\frac{\partial x_8}{\partial x_7}
\frac{\partial x_8}{\partial x_8}
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer}_0 & x_0 \\
\theta_0 & \frac{\partial x_8}{\partial x_0} \\
w_0 & \\
\hline
\text{layer}_1 & x_1 \\
\theta_1 & \frac{\partial x_8}{\partial x_1} \\
w_1 & \\
\hline
\text{layer}_2 & x_2 \\
\theta_2 & \frac{\partial x_8}{\partial x_2} \\
w_2 & \\
\hline
\text{layer}_3 & x_3 \\
\theta_3 & \frac{\partial x_8}{\partial x_3} \\
w_3 & \\
\hline
\text{layer}_4 & x_4 \\
\theta_4 & \frac{\partial x_8}{\partial x_4} \\
w_4 & \\
\hline
\text{layer}_5 & x_5 \\
\theta_5 & \frac{\partial x_8}{\partial x_5} \\
w_5 & \\
\hline
\text{layer}_6 & x_6 \\
\theta_6 & \frac{\partial x_8}{\partial x_6} \\
w_6 & \\
\hline
\text{layer}_7 & x_7 \\
\theta_7 & \frac{\partial x_8}{\partial x_7} \\
w_7 & \\
\hline
\text{layer}_8 & x_8 \\
\theta_8 & \frac{\partial x_8}{\partial x_8} \\
w_8 & \\
\end{array}
\]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[ \begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
& \quad x_0 \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
& \quad x_1 \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
& \quad x_2 \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
& \quad x_3 \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
& \quad x_4 \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
& \quad x_5 \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
& \quad x_6 \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
& \quad x_7 \quad \frac{\partial x_8}{\partial x_7} \\
\text{layer}_8 & \quad \theta_8 \quad w_8 \\
& \quad x_8 \quad \frac{\partial x_8}{\partial x_8}
\end{align*} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 \ w_0 \\
\hline
x_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
\text{layer}_1 & \theta_1 \ w_1 \\
\hline
x_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
\text{layer}_2 & \theta_2 \ w_2 \\
\hline
x_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
\text{layer}_3 & \theta_3 \ w_3 \\
\hline
x_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
\text{layer}_4 & \theta_4 \ w_4 \\
\hline
x_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
\text{layer}_5 & \theta_5 \ w_5 \\
\hline
x_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
\text{layer}_6 & \theta_6 \ w_6 \\
\hline
x_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
\text{layer}_7 & \theta_7 \ w_7 \\
\hline
x_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
\text{layer}_8 & \theta_8 \ w_8 \\
\hline
x_8 & \frac{\partial x_8}{\partial x_8}
\end{array}
\]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 \ w_0 \\
\hline
\text{layer}_1 & \theta_1 \ w_1 \\
\hline
\text{layer}_2 & \theta_2 \ w_2 \\
\hline
\text{layer}_3 & \theta_3 \ w_3 \\
\hline
\text{layer}_4 & \theta_4 \ w_4 \\
\hline
\text{layer}_5 & \theta_5 \ w_5 \\
\hline
\text{layer}_6 & \theta_6 \ w_6 \\
\hline
\text{layer}_7 & \theta_7 \ w_7 \\
\end{array}
\]

\[
\begin{array}{c|c}
\hline
x_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
x_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
x_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
x_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
x_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
x_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
x_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
x_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
x_8 & \frac{\partial x_8}{\partial x_8} \\
\end{array}
\]
### Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer</th>
<th>$\theta$</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 0</td>
<td>$\theta_0$</td>
<td>$w_0$</td>
</tr>
<tr>
<td>Layer 1</td>
<td>$\theta_1$</td>
<td>$w_1$</td>
</tr>
<tr>
<td>Layer 2</td>
<td>$\theta_2$</td>
<td>$w_2$</td>
</tr>
<tr>
<td>Layer 3</td>
<td>$\theta_3$</td>
<td>$w_3$</td>
</tr>
<tr>
<td>Layer 4</td>
<td>$\theta_4$</td>
<td>$w_4$</td>
</tr>
<tr>
<td>Layer 5</td>
<td>$\theta_5$</td>
<td>$w_5$</td>
</tr>
<tr>
<td>Layer 6</td>
<td>$\theta_6$</td>
<td>$w_6$</td>
</tr>
<tr>
<td>Layer 7</td>
<td>$\theta_7$</td>
<td>$w_7$</td>
</tr>
</tbody>
</table>

- $x_0$ \(\frac{\partial x_8}{\partial x_0}\)
- $x_1$ \(\frac{\partial x_8}{\partial x_1}\)
- $x_2$ \(\frac{\partial x_8}{\partial x_2}\)
- $x_3$ \(\frac{\partial x_8}{\partial x_3}\)
- $x_4$ \(\frac{\partial x_8}{\partial x_4}\)
- $x_5$ \(\frac{\partial x_8}{\partial x_5}\)
- $x_6$ \(\frac{\partial x_8}{\partial x_6}\)
- $x_7$ \(\frac{\partial x_8}{\partial x_7}\)
- $x_8$ \(\frac{\partial x_8}{\partial x_8}\)
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[ \begin{align*}
\text{layer}_0 & : \theta_0 \ w_0 \\
\text{layer}_1 & : \theta_1 \ w_1 \\
\text{layer}_2 & : \theta_2 \ w_2 \\
\text{layer}_3 & : \theta_3 \ w_3 \\
\text{layer}_4 & : \theta_4 \ w_4 \\
\text{layer}_5 & : \theta_5 \ w_5 \\
\text{layer}_6 & : \theta_6 \ w_6 \\
\text{layer}_7 & : \theta_7 \ w_7 \\
\end{align*} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer}_0 & x_0 \\
\hline
\text{layer}_1 & x_1 \\
\hline
\text{layer}_2 & x_2 \\
\hline
\text{layer}_3 & x_3 \\
\hline
\text{layer}_4 & x_4 \\
\hline
\text{layer}_5 & x_5 \\
\hline
\text{layer}_6 & x_6 \\
\hline
\text{layer}_7 & x_7 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 \\
\hline
\text{layer}_1 & \theta_1 \\
\hline
\text{layer}_2 & \theta_2 \\
\hline
\text{layer}_3 & \theta_3 \\
\hline
\text{layer}_4 & \theta_4 \\
\hline
\text{layer}_5 & \theta_5 \\
\hline
\text{layer}_6 & \theta_6 \\
\hline
\text{layer}_7 & \theta_7 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c}
\text{layer}_0 & w_0 \\
\hline
\text{layer}_1 & w_1 \\
\hline
\text{layer}_2 & w_2 \\
\hline
\text{layer}_3 & w_3 \\
\hline
\text{layer}_4 & w_4 \\
\hline
\text{layer}_5 & w_5 \\
\hline
\text{layer}_6 & w_6 \\
\hline
\text{layer}_7 & w_7 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c}
\text{layer}_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
\text{layer}_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
\text{layer}_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
\text{layer}_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
\text{layer}_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
\text{layer}_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
\text{layer}_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
\text{layer}_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
\end{array}
\]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[ \begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
\text{layer}_7 & \quad \theta_7 \quad w_7
\end{align*} \]
Evaluating a Neural Network to Allow Subsequent Backpropagation

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{layer}_0 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>( \text{layer}_1 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>( \text{layer}_2 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>( \text{layer}_3 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>( \text{layer}_4 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>( \text{layer}_5 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>( \text{layer}_6 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>( \text{layer}_7 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
<tr>
<td>( x_8 )</td>
<td>( \partial x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_8} )</td>
</tr>
</tbody>
</table>
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{array}{c|c}
\text{layer} & \text{ Equations } \\
\hline
0 & x_0 \ \theta_0 \ \omega_0 \\
1 & x_1 \ \theta_1 \ \omega_1 \\
2 & x_2 \ \theta_2 \ \omega_2 \\
3 & x_3 \ \theta_3 \ \omega_3 \\
4 & x_4 \ \theta_4 \ \omega_4 \\
5 & x_5 \ \theta_5 \ \omega_5 \\
6 & x_6 \ \theta_6 \ \omega_6 \\
7 & x_7 \ \theta_7 \ \omega_7 \\
8 & x_8 \ \frac{\partial x_8}{\partial x_0} \\
\end{array}
\]
Evaluating a Neural Network to Allow Subsequent Backpropagation

\[
\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
\end{align*}
\]
Some Observations

Only need to store live variables from forward pass until they are used in reverse pass.

Only need to store live variables during reverse pass.

Most deep learning frameworks store all intermediate forward and reverse pass variables for simplicity of implementation.

It doesn't matter because storage use is dominated by maximal use.

Maximal use is proportional to the depth of the network i.e., the running time of the program.
Some Observations

- Only need to store live variables from forward pass until they are used in reverse pass.
Some Observations

- Only need to store live variables from forward pass until they are used in reverse pass.
- Only need to store live variables during reverse pass.
Some Observations

- Only need to store live variables from forward pass until they are used in reverse pass.
- Only need to store live variables during reverse pass.
- Most deep learning frameworks store all intermediate forward and reverse pass variables for simplicity of implementation.
Some Observations

- Only need to store live variables from forward pass until they are used in reverse pass.
- Only need to store live variables during reverse pass.
- Most deep learning frameworks store all intermediate forward and reverse pass variables for simplicity of implementation.
- It doesn’t matter because storage use is dominated by maximal use.
Some Observations

- Only need to store live variables from forward pass until they are used in reverse pass.
- Only need to store live variables during reverse pass.
- Most deep learning frameworks store all intermediate forward and reverse pass variables for simplicity of implementation.
- It doesn’t matter because storage use is dominated by maximal use.
- Maximal use is proportional to the depth of the network *i.e.*, the running time of the program.
Complexity of Reverse-Mode AD

If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$ then reverse mode takes $O(wt)$ space and $O(t)$ time.
If running time of primal is $O(t)$
 Complexity of Reverse-Mode AD

- If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$
If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$ then reverse mode takes $O(wt)$ space.
If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$ then reverse mode takes $O(wt)$ space and $O(t)$ time.
Backpropagation in a Neural Network with Checkpointing

\[ x_0 \frac{\partial x_8}{\partial x_0} \]

layer_0 \( \theta_0 \) \( w_0 \)

\[ x_1 \frac{\partial x_8}{\partial x_1} \]

layer_1 \( \theta_1 \) \( w_1 \)

\[ x_2 \frac{\partial x_8}{\partial x_2} \]

layer_2 \( \theta_2 \) \( w_2 \)

\[ x_3 \frac{\partial x_8}{\partial x_3} \]

layer_3 \( \theta_3 \) \( w_3 \)

\[ x_4 \frac{\partial x_8}{\partial x_4} \]

layer_4 \( \theta_4 \) \( w_4 \)

\[ x_5 \frac{\partial x_8}{\partial x_5} \]

layer_5 \( \theta_5 \) \( w_5 \)

\[ x_6 \frac{\partial x_8}{\partial x_6} \]

layer_6 \( \theta_6 \) \( w_6 \)

\[ x_7 \frac{\partial x_8}{\partial x_7} \]

layer_7 \( \theta_7 \) \( w_7 \)

\[ x_8 \frac{\partial x_8}{\partial x_8} \]
Backpropagation in a Neural Network with Checkpointing

\[ x_0 \quad \frac{\partial x_8}{\partial x_0} \]

layer_0 \( \theta_0 \) \( w_0 \)

\[ x_1 \quad \frac{\partial x_8}{\partial x_1} \]

layer_1 \( \theta_1 \) \( w_1 \)

\[ x_2 \quad \frac{\partial x_8}{\partial x_2} \]

layer_2 \( \theta_2 \) \( w_2 \)

\[ x_3 \quad \frac{\partial x_8}{\partial x_3} \]

layer_3 \( \theta_3 \) \( w_3 \)

\[ x_4 \quad \frac{\partial x_8}{\partial x_4} \]

layer_4 \( \theta_4 \) \( w_4 \)

\[ x_5 \quad \frac{\partial x_8}{\partial x_5} \]

layer_5 \( \theta_5 \) \( w_5 \)

\[ x_6 \quad \frac{\partial x_8}{\partial x_6} \]

layer_6 \( \theta_6 \) \( w_6 \)

\[ x_7 \quad \frac{\partial x_8}{\partial x_7} \]

layer_7 \( \theta_7 \) \( w_7 \)

\[ x_8 \quad \frac{\partial x_8}{\partial x_8} \]
Backpropagation in a Neural Network with Checkpointing

\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
& \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
& \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
& \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
& \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
& \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
& \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
& \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
& \quad \frac{\partial x_8}{\partial x_7} \\
\end{align*}
### Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
<th>( x )</th>
<th>( \frac{\partial x}{\partial x} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>( x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_8} )</td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

\[ x_0 \frac{\partial x_8}{\partial x_0} \]

layer_0 \( \theta_0 \ w_0 \)

\[ x_1 \frac{\partial x_8}{\partial x_1} \]

layer_1 \( \theta_1 \ w_1 \)

\[ x_2 \frac{\partial x_8}{\partial x_2} \]

layer_2 \( \theta_2 \ w_2 \)

\[ x_3 \frac{\partial x_8}{\partial x_3} \]

layer_3 \( \theta_3 \ w_3 \)

\[ x_4 \frac{\partial x_8}{\partial x_4} \]

layer_4 \( \theta_4 \ w_4 \)

\[ x_5 \frac{\partial x_8}{\partial x_5} \]

layer_5 \( \theta_5 \ w_5 \)

\[ x_6 \frac{\partial x_8}{\partial x_6} \]

layer_6 \( \theta_6 \ w_6 \)

\[ x_7 \frac{\partial x_8}{\partial x_7} \]

layer_7 \( \theta_7 \ w_7 \)

\[ x_8 \frac{\partial x_8}{\partial x_8} \]
Backpropagation in a Neural Network with Checkpointing

\[ x_0 \frac{\partial x_8}{\partial x_0} \]

\[ x_1 \frac{\partial x_8}{\partial x_1} \]

\[ \text{layer}_0 \theta_0 w_0 \]

\[ x_2 \frac{\partial x_8}{\partial x_2} \]

\[ \text{layer}_1 \theta_1 w_1 \]

\[ x_3 \frac{\partial x_8}{\partial x_3} \]

\[ \text{layer}_2 \theta_2 w_2 \]

\[ x_4 \frac{\partial x_8}{\partial x_4} \]

\[ \text{layer}_3 \theta_3 w_3 \]

\[ x_5 \frac{\partial x_8}{\partial x_5} \]

\[ \text{layer}_4 \theta_4 w_4 \]

\[ x_6 \frac{\partial x_8}{\partial x_6} \]

\[ \text{layer}_5 \theta_5 w_5 \]

\[ x_7 \frac{\partial x_8}{\partial x_7} \]

\[ \text{layer}_6 \theta_6 w_6 \]

\[ x_8 \frac{\partial x_8}{\partial x_8} \]

\[ \text{layer}_7 \theta_7 w_7 \]
Backpropagation in a Neural Network with Checkpointing

\[ x_0 \frac{\partial x_8}{\partial x_0} \]
\[ \text{layer}_0 \theta_0 w_0 \]
\[ x_1 \frac{\partial x_8}{\partial x_1} \]
\[ \text{layer}_1 \theta_1 w_1 \]
\[ x_2 \frac{\partial x_8}{\partial x_2} \]
\[ \text{layer}_2 \theta_2 w_2 \]
\[ x_3 \frac{\partial x_8}{\partial x_3} \]
\[ \text{layer}_3 \theta_3 w_3 \]
\[ x_4 \frac{\partial x_8}{\partial x_4} \]
\[ \text{layer}_4 \theta_4 w_4 \]
\[ x_5 \frac{\partial x_8}{\partial x_5} \]
\[ \text{layer}_5 \theta_5 w_5 \]
\[ x_6 \frac{\partial x_8}{\partial x_6} \]
\[ \text{layer}_6 \theta_6 w_6 \]
\[ x_7 \frac{\partial x_8}{\partial x_7} \]
\[ \text{layer}_7 \theta_7 w_7 \]
\[ x_8 \frac{\partial x_8}{\partial x_8} \]
Backpropagation in a Neural Network with Checkpointing

Layer 0 \( \theta_0 w_0 \)

\[ x_0 \quad \frac{\partial x_8}{\partial x_0} \]

Layer 1 \( \theta_1 w_1 \)

\[ x_1 \quad \frac{\partial x_8}{\partial x_1} \]

Layer 2 \( \theta_2 w_2 \)

\[ x_2 \quad \frac{\partial x_8}{\partial x_2} \]

Layer 3 \( \theta_3 w_3 \)

\[ x_3 \quad \frac{\partial x_8}{\partial x_3} \]

Layer 4 \( \theta_4 w_4 \)

\[ x_4 \quad \frac{\partial x_8}{\partial x_4} \]

Layer 5 \( \theta_5 w_5 \)

\[ x_5 \quad \frac{\partial x_8}{\partial x_5} \]

Layer 6 \( \theta_6 w_6 \)

\[ x_6 \quad \frac{\partial x_8}{\partial x_6} \]

Layer 7 \( \theta_7 w_7 \)

\[ x_7 \quad \frac{\partial x_8}{\partial x_7} \]

Layer 8 \( \theta_8 w_8 \)

\[ x_8 \quad \frac{\partial x_8}{\partial x_8} \]
Backpropagation in a Neural Network with Checkpointing

\[
\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
& \quad x_0 \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
& \quad x_1 \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
& \quad x_2 \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
& \quad x_3 \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
& \quad x_4 \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
& \quad x_5 \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
& \quad x_6 \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
& \quad x_7 \quad \frac{\partial x_8}{\partial x_7} \\
\end{align*}
\]
Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>$\theta$</th>
<th>$w$</th>
<th>$x$</th>
<th>$\frac{\partial x_8}{\partial x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer 0</td>
<td>$\theta_0$</td>
<td>$w_0$</td>
<td>$x_0$</td>
<td>$\frac{\partial x_8}{\partial x_0}$</td>
</tr>
<tr>
<td>layer 1</td>
<td>$\theta_1$</td>
<td>$w_1$</td>
<td>$x_1$</td>
<td>$\frac{\partial x_8}{\partial x_1}$</td>
</tr>
<tr>
<td>layer 2</td>
<td>$\theta_2$</td>
<td>$w_2$</td>
<td>$x_2$</td>
<td>$\frac{\partial x_8}{\partial x_2}$</td>
</tr>
<tr>
<td>layer 3</td>
<td>$\theta_3$</td>
<td>$w_3$</td>
<td>$x_3$</td>
<td>$\frac{\partial x_8}{\partial x_3}$</td>
</tr>
<tr>
<td>layer 4</td>
<td>$\theta_4$</td>
<td>$w_4$</td>
<td>$x_4$</td>
<td>$\frac{\partial x_8}{\partial x_4}$</td>
</tr>
<tr>
<td>layer 5</td>
<td>$\theta_5$</td>
<td>$w_5$</td>
<td>$x_5$</td>
<td>$\frac{\partial x_8}{\partial x_5}$</td>
</tr>
<tr>
<td>layer 6</td>
<td>$\theta_6$</td>
<td>$w_6$</td>
<td>$x_6$</td>
<td>$\frac{\partial x_8}{\partial x_6}$</td>
</tr>
<tr>
<td>layer 7</td>
<td>$\theta_7$</td>
<td>$w_7$</td>
<td>$x_7$</td>
<td>$\frac{\partial x_8}{\partial x_7}$</td>
</tr>
<tr>
<td>layer 8</td>
<td>$\theta_8$</td>
<td>$w_8$</td>
<td>$x_8$</td>
<td>$\frac{\partial x_8}{\partial x_8}$</td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

\[
\begin{array}{c|cc}
\text{layer} & \theta & w \\
0 & \theta_0 & w_0 \\
1 & \theta_1 & w_1 \\
2 & \theta_2 & w_2 \\
3 & \theta_3 & w_3 \\
4 & \theta_4 & w_4 \\
5 & \theta_5 & w_5 \\
6 & \theta_6 & w_6 \\
7 & \theta_7 & w_7 \\
\end{array}
\]

\[
\begin{array}{c|c}
x_0 & \frac{\partial x_8}{\partial x_0} \\
x_1 & \frac{\partial x_8}{\partial x_1} \\
x_2 & \frac{\partial x_8}{\partial x_2} \\
x_3 & \frac{\partial x_8}{\partial x_3} \\
x_4 & \frac{\partial x_8}{\partial x_4} \\
x_5 & \frac{\partial x_8}{\partial x_5} \\
x_6 & \frac{\partial x_8}{\partial x_6} \\
x_7 & \frac{\partial x_8}{\partial x_7} \\
x_8 & \frac{\partial x_8}{\partial x_8} \\
\end{array}
\]
Backpropagation in a Neural Network with Checkpointing

\[
\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
\text{layer}_7 & \quad \theta_7 \quad w_7
\end{align*}
\]

\[
\begin{align*}
x_0 & \quad \frac{\partial x_8}{\partial x_0} \\
x_1 & \quad \frac{\partial x_8}{\partial x_1} \\
x_2 & \quad \frac{\partial x_8}{\partial x_2} \\
x_3 & \quad \frac{\partial x_8}{\partial x_3} \\
x_4 & \quad \frac{\partial x_8}{\partial x_4} \\
x_5 & \quad \frac{\partial x_8}{\partial x_5} \\
x_6 & \quad \frac{\partial x_8}{\partial x_6} \\
x_7 & \quad \frac{\partial x_8}{\partial x_7} \\
x_8 & \quad \frac{\partial x_8}{\partial x_8}
\end{align*}
\]
Backpropagation in a Neural Network with Checkpointing

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 w_0 \\
\text{layer}_1 & \theta_1 w_1 \\
\text{layer}_2 & \theta_2 w_2 \\
\text{layer}_3 & \theta_3 w_3 \\
\text{layer}_4 & \theta_4 w_4 \\
\text{layer}_5 & \theta_5 w_5 \\
\text{layer}_6 & \theta_6 w_6 \\
\text{layer}_7 & \theta_7 w_7 \\
x_0 & \frac{\partial x_8}{\partial x_0} \\
x_1 & \frac{\partial x_8}{\partial x_1} \\
x_2 & \frac{\partial x_8}{\partial x_2} \\
x_3 & \frac{\partial x_8}{\partial x_3} \\
x_4 & \frac{\partial x_8}{\partial x_4} \\
x_5 & \frac{\partial x_8}{\partial x_5} \\
x_6 & \frac{\partial x_8}{\partial x_6} \\
x_7 & \frac{\partial x_8}{\partial x_7} \\
x_8 & \frac{\partial x_8}{\partial x_8}
\end{array}
\]
### Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>Parameters</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer0</td>
<td>$\theta_0$, $w_0$</td>
<td>$x_0$, $\frac{\partial x_8}{\partial x_0}$</td>
</tr>
<tr>
<td>layer1</td>
<td>$\theta_1$, $w_1$</td>
<td>$x_1$, $\frac{\partial x_8}{\partial x_1}$</td>
</tr>
<tr>
<td>layer2</td>
<td>$\theta_2$, $w_2$</td>
<td>$x_2$, $\frac{\partial x_8}{\partial x_2}$</td>
</tr>
<tr>
<td>layer3</td>
<td>$\theta_3$, $w_3$</td>
<td>$x_3$, $\frac{\partial x_8}{\partial x_3}$</td>
</tr>
<tr>
<td>layer4</td>
<td>$\theta_4$, $w_4$</td>
<td>$x_4$, $\frac{\partial x_8}{\partial x_4}$</td>
</tr>
<tr>
<td>layer5</td>
<td>$\theta_5$, $w_5$</td>
<td>$x_5$, $\frac{\partial x_8}{\partial x_5}$</td>
</tr>
<tr>
<td>layer6</td>
<td>$\theta_6$, $w_6$</td>
<td>$x_6$, $\frac{\partial x_8}{\partial x_6}$</td>
</tr>
<tr>
<td>layer7</td>
<td>$\theta_7$, $w_7$</td>
<td>$x_7$, $\frac{\partial x_8}{\partial x_7}$</td>
</tr>
<tr>
<td>layer8</td>
<td></td>
<td>$x_8$, $\frac{\partial x_8}{\partial x_8}$</td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

\[
\begin{align*}
\text{layer}_0 & \; \theta_0 \; w_0 \\
x_0 \; \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \; \theta_1 \; w_1 \\
x_1 \; \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \; \theta_2 \; w_2 \\
x_2 \; \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \; \theta_3 \; w_3 \\
x_3 \; \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \; \theta_4 \; w_4 \\
x_4 \; \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \; \theta_5 \; w_5 \\
x_5 \; \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \; \theta_6 \; w_6 \\
x_6 \; \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \; \theta_7 \; w_7 \\
x_7 \; \frac{\partial x_8}{\partial x_7} \\
\text{layer}_8 & \\
x_8 \; \frac{\partial x_8}{\partial x_8}
\end{align*}
\]
Backpropagation in a Neural Network with Checkpointing

\[ x_0 \frac{\partial x_8}{\partial x_0} \]

layer \( \theta_0 w_0 \)

\[ x_1 \frac{\partial x_8}{\partial x_1} \]

layer \( \theta_1 w_1 \)

\[ x_2 \frac{\partial x_8}{\partial x_2} \]

layer \( \theta_2 w_2 \)

\[ x_3 \frac{\partial x_8}{\partial x_3} \]

layer \( \theta_3 w_3 \)

\[ x_4 \frac{\partial x_8}{\partial x_4} \]

layer \( \theta_4 w_4 \)

\[ x_5 \frac{\partial x_8}{\partial x_5} \]

layer \( \theta_5 w_5 \)

\[ x_6 \frac{\partial x_8}{\partial x_6} \]

layer \( \theta_6 w_6 \)

\[ x_7 \frac{\partial x_8}{\partial x_7} \]

layer \( \theta_7 w_7 \)

\[ x_8 \frac{\partial x_8}{\partial x_8} \]
Backpropagation in a Neural Network with Checkpointing

\[
\begin{array}{c|cc}
\text{layer}_0 & x_0 & \frac{\partial x_8}{\partial x_0} \\
\theta_0 & w_0 & \\
\text{layer}_1 & x_1 & \frac{\partial x_8}{\partial x_1} \\
\theta_1 & w_1 & \\
\text{layer}_2 & x_2 & \frac{\partial x_8}{\partial x_2} \\
\theta_2 & w_2 & \\
\text{layer}_3 & x_3 & \frac{\partial x_8}{\partial x_3} \\
\theta_3 & w_3 & \\
\text{layer}_4 & x_4 & \frac{\partial x_8}{\partial x_4} \\
\theta_4 & w_4 & \\
\text{layer}_5 & x_5 & \frac{\partial x_8}{\partial x_5} \\
\theta_5 & w_5 & \\
\text{layer}_6 & x_6 & \frac{\partial x_8}{\partial x_6} \\
\theta_6 & w_6 & \\
\text{layer}_7 & x_7 & \frac{\partial x_8}{\partial x_7} \\
\theta_7 & w_7 & \\
\text{layer}_8 & x_8 & \frac{\partial x_8}{\partial x_8} \\
\end{array}
\]
Backpropagation in a Neural Network with Checkpointing

\[
\begin{array}{c|cc}
\text{layer}_0 & \theta_0 & w_0 \\
\hline
x_0 & \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \theta_1 & w_1 \\
\hline
x_1 & \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \theta_2 & w_2 \\
\hline
x_2 & \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \theta_3 & w_3 \\
\hline
x_3 & \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \theta_4 & w_4 \\
\hline
x_4 & \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \theta_5 & w_5 \\
\hline
x_5 & \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \theta_6 & w_6 \\
\hline
x_6 & \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \theta_7 & w_7 \\
\hline
x_7 & \frac{\partial x_8}{\partial x_7} \\
\text{layer}_8 & & \\
\hline
x_8 & \frac{\partial x_8}{\partial x_8} \\
\end{array}
\]
<table>
<thead>
<tr>
<th>Layer</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer_0</td>
<td>$\theta_0, w_0$</td>
</tr>
<tr>
<td>$x_0$</td>
<td>$\frac{\partial x_8}{\partial x_0}$</td>
</tr>
<tr>
<td>layer_1</td>
<td>$\theta_1, w_1$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$\frac{\partial x_8}{\partial x_1}$</td>
</tr>
<tr>
<td>layer_2</td>
<td>$\theta_2, w_2$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$\frac{\partial x_8}{\partial x_2}$</td>
</tr>
<tr>
<td>layer_3</td>
<td>$\theta_3, w_3$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\frac{\partial x_8}{\partial x_3}$</td>
</tr>
<tr>
<td>layer_4</td>
<td>$\theta_4, w_4$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>$\frac{\partial x_8}{\partial x_4}$</td>
</tr>
<tr>
<td>layer_5</td>
<td>$\theta_5, w_5$</td>
</tr>
<tr>
<td>$x_5$</td>
<td>$\frac{\partial x_8}{\partial x_5}$</td>
</tr>
<tr>
<td>layer_6</td>
<td>$\theta_6, w_6$</td>
</tr>
<tr>
<td>$x_6$</td>
<td>$\frac{\partial x_8}{\partial x_6}$</td>
</tr>
<tr>
<td>layer_7</td>
<td>$\theta_7, w_7$</td>
</tr>
<tr>
<td>$x_7$</td>
<td>$\frac{\partial x_8}{\partial x_7}$</td>
</tr>
<tr>
<td>layer_8</td>
<td>$\theta_8, w_8$</td>
</tr>
<tr>
<td>$x_8$</td>
<td>$\frac{\partial x_8}{\partial x_8}$</td>
</tr>
</tbody>
</table>
### Backpropagation in a Neural Network with Checkpointing

A neural network can be diagrammed as follows:

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta_i )</th>
<th>( w_i )</th>
<th>( x_i )</th>
<th>( \frac{\partial x_8}{\partial x_i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer_0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>layer_1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>layer_2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>layer_3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>layer_4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>layer_5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>layer_6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>layer_7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
<tr>
<td>layer_8</td>
<td>( \theta_8 )</td>
<td>( w_8 )</td>
<td>( x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_8} )</td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

\[
\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
x_0 & \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
x_1 & \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
x_2 & \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
x_3 & \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
x_4 & \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
x_5 & \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
x_6 & \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
x_7 & \quad \frac{\partial x_8}{\partial x_7} \\
x_8 & \quad \frac{\partial x_8}{\partial x_8}
\end{align*}
\]
### Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>θ</th>
<th>w</th>
<th>( \frac{\partial x_8}{\partial x_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>θ₀</td>
<td>w₀</td>
<td>( x₀ )</td>
</tr>
<tr>
<td>1</td>
<td>θ₁</td>
<td>w₁</td>
<td>( x₁ )</td>
</tr>
<tr>
<td>2</td>
<td>θ₂</td>
<td>w₂</td>
<td>( x₂ )</td>
</tr>
<tr>
<td>3</td>
<td>θ₃</td>
<td>w₃</td>
<td>( x₃ )</td>
</tr>
<tr>
<td>4</td>
<td>θ₄</td>
<td>w₄</td>
<td>( x₄ )</td>
</tr>
<tr>
<td>5</td>
<td>θ₅</td>
<td>w₅</td>
<td>( x₅ )</td>
</tr>
<tr>
<td>6</td>
<td>θ₆</td>
<td>w₆</td>
<td>( x₆ )</td>
</tr>
<tr>
<td>7</td>
<td>θ₇</td>
<td>w₇</td>
<td>( x₇ )</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>( x₈ )</td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

layer_0 \theta_0 w_0

layer_1 \theta_1 w_1

layer_2 \theta_2 w_2

layer_3 \theta_3 w_3

layer_4 \theta_4 w_4

layer_5 \theta_5 w_5

layer_6 \theta_6 w_6

layer_7 \theta_7 w_7

x_0 \frac{\partial x_8}{\partial x_0}

x_1 \frac{\partial x_8}{\partial x_1}

x_2 \frac{\partial x_8}{\partial x_2}

x_3 \frac{\partial x_8}{\partial x_3}

x_4 \frac{\partial x_8}{\partial x_4}

x_5 \frac{\partial x_8}{\partial x_5}

x_6 \frac{\partial x_8}{\partial x_6}

x_7 \frac{\partial x_8}{\partial x_7}

x_8 \frac{\partial x_8}{\partial x_8}
Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer_0</td>
<td>$x_0$</td>
<td>$\frac{\partial x_8}{\partial x_0}$</td>
</tr>
<tr>
<td>layer_1</td>
<td>$x_1$</td>
<td>$\frac{\partial x_8}{\partial x_1}$</td>
</tr>
<tr>
<td>layer_2</td>
<td>$x_2$</td>
<td>$\frac{\partial x_8}{\partial x_2}$</td>
</tr>
<tr>
<td>layer_3</td>
<td>$x_3$</td>
<td>$\frac{\partial x_8}{\partial x_3}$</td>
</tr>
<tr>
<td>layer_4</td>
<td>$x_4$</td>
<td>$\frac{\partial x_8}{\partial x_4}$</td>
</tr>
<tr>
<td>layer_5</td>
<td>$x_5$</td>
<td>$\frac{\partial x_8}{\partial x_5}$</td>
</tr>
<tr>
<td>layer_6</td>
<td>$x_6$</td>
<td>$\frac{\partial x_8}{\partial x_6}$</td>
</tr>
<tr>
<td>layer_7</td>
<td>$x_7$</td>
<td>$\frac{\partial x_8}{\partial x_7}$</td>
</tr>
<tr>
<td>layer_8</td>
<td>$x_8$</td>
<td>$\frac{\partial x_8}{\partial x_8}$</td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta_i )</th>
<th>( w_i )</th>
<th>( x_i )</th>
<th>( \frac{\partial x_i}{\partial x_j} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>Layer 1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>Layer 2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>Layer 3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>Layer 4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>Layer 5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>Layer 6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>Layer 7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
</tbody>
</table>

\( x_8 \)
### Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>Weight</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\theta_0)</td>
<td>(w_0)</td>
<td>(x_0)</td>
</tr>
<tr>
<td>1</td>
<td>(\theta_1)</td>
<td>(w_1)</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>(\theta_2)</td>
<td>(w_2)</td>
<td>(x_2)</td>
</tr>
<tr>
<td>3</td>
<td>(\theta_3)</td>
<td>(w_3)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>4</td>
<td>(\theta_4)</td>
<td>(w_4)</td>
<td>(x_4)</td>
</tr>
<tr>
<td>5</td>
<td>(\theta_5)</td>
<td>(w_5)</td>
<td>(x_5)</td>
</tr>
<tr>
<td>6</td>
<td>(\theta_6)</td>
<td>(w_6)</td>
<td>(x_6)</td>
</tr>
<tr>
<td>7</td>
<td>(\theta_7)</td>
<td>(w_7)</td>
<td>(x_7)</td>
</tr>
</tbody>
</table>

\[
\frac{\partial x_8}{\partial x_0}, \frac{\partial x_8}{\partial x_1}, \frac{\partial x_8}{\partial x_2}, \frac{\partial x_8}{\partial x_3}, \frac{\partial x_8}{\partial x_4}, \frac{\partial x_8}{\partial x_5}, \frac{\partial x_8}{\partial x_6}, \frac{\partial x_8}{\partial x_7}, \frac{\partial x_8}{\partial x_8}
\]
Backpropagation in a Neural Network with Checkpointing

<table>
<thead>
<tr>
<th>Layer</th>
<th>( \theta )</th>
<th>( w )</th>
<th>( x )</th>
<th>( \frac{\partial x_8}{\partial x_i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer_0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 )</td>
<td>( \frac{\partial x_8}{\partial x_0} )</td>
</tr>
<tr>
<td>layer_1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 )</td>
<td>( \frac{\partial x_8}{\partial x_1} )</td>
</tr>
<tr>
<td>layer_2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 )</td>
<td>( \frac{\partial x_8}{\partial x_2} )</td>
</tr>
<tr>
<td>layer_3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 )</td>
<td>( \frac{\partial x_8}{\partial x_3} )</td>
</tr>
<tr>
<td>layer_4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 )</td>
<td>( \frac{\partial x_8}{\partial x_4} )</td>
</tr>
<tr>
<td>layer_5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 )</td>
<td>( \frac{\partial x_8}{\partial x_5} )</td>
</tr>
<tr>
<td>layer_6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 )</td>
<td>( \frac{\partial x_8}{\partial x_6} )</td>
</tr>
<tr>
<td>layer_7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 )</td>
<td>( \frac{\partial x_8}{\partial x_7} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( x_8 )</td>
<td>( \frac{\partial x_8}{\partial x_8} )</td>
</tr>
<tr>
<td>Layer</td>
<td>( \theta )</td>
<td>( w )</td>
<td>Equation</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( \theta_0 )</td>
<td>( w_0 )</td>
<td>( x_0 \frac{\partial x_8}{\partial x_0} )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( \theta_1 )</td>
<td>( w_1 )</td>
<td>( x_1 \frac{\partial x_8}{\partial x_1} )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( \theta_2 )</td>
<td>( w_2 )</td>
<td>( x_2 \frac{\partial x_8}{\partial x_2} )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( \theta_3 )</td>
<td>( w_3 )</td>
<td>( x_3 \frac{\partial x_8}{\partial x_3} )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( \theta_4 )</td>
<td>( w_4 )</td>
<td>( x_4 \frac{\partial x_8}{\partial x_4} )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( \theta_5 )</td>
<td>( w_5 )</td>
<td>( x_5 \frac{\partial x_8}{\partial x_5} )</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( \theta_6 )</td>
<td>( w_6 )</td>
<td>( x_6 \frac{\partial x_8}{\partial x_6} )</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>( \theta_7 )</td>
<td>( w_7 )</td>
<td>( x_7 \frac{\partial x_8}{\partial x_7} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( x_8 \frac{\partial x_8}{\partial x_8} )</td>
<td></td>
</tr>
</tbody>
</table>
Backpropagation in a Neural Network with Checkpointing

\[
\begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
\text{layer}_7 & \quad \theta_7 \quad w_7
\end{align*}
\]

\[
\begin{align*}
x_0 & \quad \frac{\partial x_8}{\partial x_0} \\
x_1 & \quad \frac{\partial x_8}{\partial x_1} \\
x_2 & \quad \frac{\partial x_8}{\partial x_2} \\
x_3 & \quad \frac{\partial x_8}{\partial x_3} \\
x_4 & \quad \frac{\partial x_8}{\partial x_4} \\
x_5 & \quad \frac{\partial x_8}{\partial x_5} \\
x_6 & \quad \frac{\partial x_8}{\partial x_6} \\
x_7 & \quad \frac{\partial x_8}{\partial x_7} \\
x_8 & \quad \frac{\partial x_8}{\partial x_8}
\end{align*}
\]
Backpropagation in a Neural Network with Checkpointing

\[ \begin{align*}
\text{layer}_0 & \quad \theta_0 \quad w_0 \\
x_0 & \quad \frac{\partial x_8}{\partial x_0} \\
\text{layer}_1 & \quad \theta_1 \quad w_1 \\
x_1 & \quad \frac{\partial x_8}{\partial x_1} \\
\text{layer}_2 & \quad \theta_2 \quad w_2 \\
x_2 & \quad \frac{\partial x_8}{\partial x_2} \\
\text{layer}_3 & \quad \theta_3 \quad w_3 \\
x_3 & \quad \frac{\partial x_8}{\partial x_3} \\
\text{layer}_4 & \quad \theta_4 \quad w_4 \\
x_4 & \quad \frac{\partial x_8}{\partial x_4} \\
\text{layer}_5 & \quad \theta_5 \quad w_5 \\
x_5 & \quad \frac{\partial x_8}{\partial x_5} \\
\text{layer}_6 & \quad \theta_6 \quad w_6 \\
x_6 & \quad \frac{\partial x_8}{\partial x_6} \\
\text{layer}_7 & \quad \theta_7 \quad w_7 \\
x_7 & \quad \frac{\partial x_8}{\partial x_7} \\
\end{align*} \]
Backpropagation in a Neural Network with Checkpointing

\[
\begin{array}{c|c}
\text{layer}_0 & \theta_0 \ w_0 \\
\hline
x_0 & \frac{\partial x_8}{\partial x_0} \\
\hline
\text{layer}_1 & \theta_1 \ w_1 \\
\hline
x_1 & \frac{\partial x_8}{\partial x_1} \\
\hline
\text{layer}_2 & \theta_2 \ w_2 \\
\hline
x_2 & \frac{\partial x_8}{\partial x_2} \\
\hline
\text{layer}_3 & \theta_3 \ w_3 \\
\hline
x_3 & \frac{\partial x_8}{\partial x_3} \\
\hline
\text{layer}_4 & \theta_4 \ w_4 \\
\hline
x_4 & \frac{\partial x_8}{\partial x_4} \\
\hline
\text{layer}_5 & \theta_5 \ w_5 \\
\hline
x_5 & \frac{\partial x_8}{\partial x_5} \\
\hline
\text{layer}_6 & \theta_6 \ w_6 \\
\hline
x_6 & \frac{\partial x_8}{\partial x_6} \\
\hline
\text{layer}_7 & \theta_7 \ w_7 \\
\hline
x_7 & \frac{\partial x_8}{\partial x_7} \\
\hline
\end{array}
\]
Checkpointing

Trades off extra running time for reduction in space.

Forward pass of first half performed twice. Once without saving intermediate variables. Once with saving intermediate variables.

Backpropagation done in stages. Interleaved with (re)running forward pass. Only need saved intermediate variables from forward pass for current stage.

Can perform divide-and-conquer.
Checkpointing

- Trades off extra running time for reduction in space.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice. Once without saving intermediate variables.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice.
  Once without saving intermediate variables.
  Once with saving intermediate variables.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice. Once without saving intermediate variables. Once with saving intermediate variables.
- Backpropagation done in stages.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice.
  Once without saving intermediate variables.
  Once with saving intermediate variables.
- Backpropagation done in stages.
  Interleaved with (re)running forward pass.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice. 
  Once without saving intermediate variables. 
  Once with saving intermediate variables.
- Backpropagation done in stages. 
  Interleaved with (re)running forward pass. 
  Only need saved intermediate variables from forward pass for current stage.
Checkpointing

- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice. Once without saving intermediate variables. Once with saving intermediate variables.
- Backpropagation done in stages. Interleaved with (re)running forward pass. Only need saved intermediate variables from forward pass for current stage.
- Can perform divide-and-conquer.
Divide-and-Conquer Checkpointing

\[ u \rightarrow p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow v \]
Divide-and-Conquer Checkpointing

\[ u \quad p_0 \quad p_1 \quad p_2 \quad p_3 \quad v \]
Divide-and-Conquer Checkpointing
Divide-and-Conquer Checkpointing
If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$ then reverse mode takes $O(w \log t)$ space and $O(t \log t)$ time.
- If running time of primal is $O(t)$
If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$
Complexity of Divide-and-Conquer Checkpointing

- If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$, then reverse mode takes $O(w \log t)$ space.
If running time of primal is $O(t)$ and primal has maximal live storage $O(w)$ then reverse mode takes $O(w \log t)$ space and $O(t \log t)$ time.
A (Brief) History of Divide-and-Conquer Checkpointing
A (Brief) History of Divide-and-Conquer Checkpointing


A (Brief) History of Divide-and-Conquer Checkpointing


```fortran
do 10 i=1, n
  ...
10 continue
```

\[\begin{align*}
\text{do 10 i=1, n} & \quad \sim \quad \text{do 10 i=1, n} \\
\text{...} & \quad \sim \quad \text{...} \\
10 \text{ continue} & \quad \sim \quad 10 \text{ continue}
\end{align*}\]
Implemented for DO Loops


```fortran
  do 10 i=1, n
    ...  
  10 continue
```

---

Assuming that the final number of iterations \( N \) is known, and assuming that each iteration has the same runtime cost,
Desiderata

A (deep) neural network has no loops (except inside primitives).

Want to implement for arbitrary code (not just a single DO loop).

Siskind (Elmore Family School of ECE, Purdue)

An FP Framework for Deep Learning

Meta 16 December 2021
A (deep) neural network has no loops (except inside primitives).
A (deep) neural network has no loops (except inside primitives).

Want to implement for arbitrary code (not just a single DO loop).
Execution Trace of Loop

Easy to make regular and uniform checkpoints
Execution Trace of Loop
Easy to make regular and uniform checkpoints
Execution Trace of Loop
Easy to make regular and uniform checkpoints
Execution Trace of Loop

Easy to make regular and uniform checkpoints

<table>
<thead>
<tr>
<th>Program point</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>42</td>
</tr>
</tbody>
</table>

Siskind (Elmore Family School of ECE, Purdue)  
An FP Framework for Deep Learning  
Meta 16 December 2021
Execution Trace of Loop

Easy to make regular and uniform checkpoints

<table>
<thead>
<tr>
<th>Program Point</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>42</td>
</tr>
</tbody>
</table>

Siskind (Elmore Family School of ECE, Purdue)

An FP Framework for Deep Learning

Meta 16 December 2021
Execution Trace of Loop
Easy to make regular and uniform checkpoints
Execution Trace of Loop
Easy to make regular and uniform checkpoints
Execution Trace of Arbitrary Code

![Graph showing execution trace of arbitrary code over time.](image)
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints
Execution Trace of Arbitrary Code
Difficult to make regular and uniform checkpoints
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints
Key Challenges

Need to interleave generation of the network with forward and backward passes through the network.
Key Challenges

Need to interleave generation of the network with forward and backward passes through the network.

Portions of the network need to be (re)generated, and (re)evaluated with forward and backward passes, multiple times and out of order.
function main(w)
    local x = f(w)
    local y = h(g(x))
    local z = p(y)
    return z
end
function main(w)
    local x = f(w)
    local y = h(g(x))
    local z = p(y)
    return z
end

function main(w)
    for i = 1, 5
        if i==1 then
            local x = f(w)
        elseif i==2 then
            local t = g(x)
        elseif i==3 then
            local y = h(t)
        elseif i==4 then
            local z = p(y)
        elseif i==5 then
            return z
        end
    end
end
Core Language

\[ e ::= c \mid x \mid \lambda x. e \mid e_1 \ e_2 \mid \text{if } e_1 \ \text{then } e_2 \ \text{else } e_3 \mid \diamond e \mid e_1 \bullet e_2 \]
Adding AD Operators to the Core Language

\[
\overleftarrow{J} : f \limp (y, \dot{x}) \quad \overrightarrow{J} : f \limp (y, \dot{x})
\]
Algorithm for Divide-and-Conquer Checkpointing

To compute \( (y, \dot{x}) = \mathcal{J} f x \dot{y} \):

**base case** \((f x \text{ fast})\):

\[
(y, \dot{x}) = \mathcal{J} f x \dot{y}
\]  
(step 0)

**inductive case:**

\[
h \circ g = f
\]  
(step 1)

\[
z = g x
\]  
(step 2)

\[
(y, \dot{z}) = \mathcal{J} h z \dot{y}
\]  
(step 3)

\[
(z, \dot{x}) = \mathcal{J} g x \dot{z}
\]  
(step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute $(y, \dot{x}) = \mathcal{J} f x \dot{y}$:

**base case ($f x$ fast):** $(y, \dot{x}) = \mathcal{J} f x \dot{y}$

**inductive case:** $h \circ g = f$

$z = g x$

$(y, \dot{z}) = \mathcal{J} h z \dot{y}$

$(z, \dot{x}) = \mathcal{J} g x \dot{z}$

(Step 0)

(Step 1)

(Step 2)

(Step 3)

(Step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \sqrt{J} f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast})\): \((y, \dot{x}) = \overrightarrow{J} f \ x \ \dot{y}\)  \hspace{1cm} (step 0)

**inductive case:**

\(h \circ g = f\)
\(z = g \ x\)

\((y, \dot{z}) = \sqrt{J} \ h \ z \ \dot{y}\)  \hspace{1cm} (step 3)

\((z, \dot{x}) = \sqrt{J} \ g \ x \ \dot{z}\)  \hspace{1cm} (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, x) = \mathcal{J} f x \dot{y}\):

**base case** \((f x \text{ fast})\):
\[(y, \dot{x}) = \mathcal{J} f x \dot{y}\] (step 0)

**inductive case**:
\[h \circ g = f\] (step 1)
\[z = g x\] (step 2)

\[(y, \dot{z}) = \mathcal{J} h z \dot{y}\] (step 3)

\[(z, \dot{x}) = \mathcal{J} g x \dot{z}\] (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \nabla f \ x \dot{y}\):

**base case** \(f \ x \text{ fast})): \((y, \dot{x}) = \nabla f \ x \dot{y}\)  

**inductive case:**  
\[h \circ g = f\]  
\[z = g \ x\]  
\[(y, \dot{z}) = \nabla h \ z \dot{y}\]  
\[(z, \dot{x}) = \nabla g \ x \dot{z}\]
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, x) = \overset{\checkmark}{\mathcal{J}} f x \dot{y}\):

**base case** \((f \ x \text{ fast}): \)  
\((y, \dot{x}) = \overset{\checkmark}{\mathcal{J}} f x \dot{y}\) \hspace{1cm} (step 0)

**inductive case:** \(h \circ g = f\)  
\(z = g x\) \hspace{1cm} (step 1) \hspace{1cm} (step 2)

\((y, \dot{z}) = \overset{\checkmark}{\mathcal{J}} h z \dot{y}\) \hspace{1cm} (step 3)

\((z, \dot{x}) = \overset{\checkmark}{\mathcal{J}} g x \dot{z}\) \hspace{1cm} (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \frac{\partial}{\partial x} f(x, \dot{y})\):

**base case** \((f \ x \ fast): \quad (y, \dot{x}) = \frac{\partial}{\partial x} f(x, \dot{y}) \quad (\text{step 0})

**inductive case:**

\[ h \circ g = f \quad (\text{step 1}) \]
\[ z = g(x) \quad (\text{step 2}) \]

\[ (y, \dot{z}) = \frac{\partial}{\partial z} h(x, \dot{z}) \quad (\text{step 3}) \]

\[ (z, \dot{x}) = \frac{\partial}{\partial x} g(x, \dot{z}) \quad (\text{step 4}) \]
Algorithm for Divide-and-Conquer Checkpointing

To compute \( (y, \dot{x}) = \nabla f \ x \dot{y} \):

**base case** (\( f \ x \) fast):
\[
(y, \dot{x}) = \nabla f \ x \dot{y} \quad \text{(step 0)}
\]

**inductive case:**
\[
h \circ g = f \quad \text{(step 1)}
\]
\[
z = g \ x \quad \text{(step 2)}
\]
\[
(y, \dot{z}) = \nabla h \ z \dot{y} \quad \text{(step 3)}
\]
\[
(z, \dot{x}) = \nabla g \ x \dot{z} \quad \text{(step 4)}
\]
To compute \((y, \dot{x}) = \sqrt[\uparrow]{\mathcal{J} f x \dot{y}}\):

**base case \((f x \text{ fast})\):** \((y, \dot{x}) = \mathcal{J} f x \dot{y}\) (step 0)

**inductive case:**

\[ h \circ g = f \] (step 1)
\[ z = g x \] (step 2)

\((y, \dot{z}) = \mathcal{J} h z \dot{y}\) (step 3)
\((z, \dot{x}) = \mathcal{J} g x \dot{z}\) (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \( (y, \dot{x}) = \mathcal{J} f x \dot{y} \):

**base case** \((f x \text{ fast})\):
\[
(y, \dot{x}) = \mathcal{J} f x \dot{y}
\]  
(step 0)

**inductive case:**
\[
\begin{align*}
 h \circ g &= f \\
 z &= g x \\
 (y, \dot{z}) &= \mathcal{J} h z \dot{y} \\
 (z, \dot{x}) &= \mathcal{J} g x \dot{z}
\end{align*}
\]  
(step 1)  
(step 2)  
(step 3)  
(step 4)
To compute $(y, \dot{x}) = \nabla f(x, \dot{y})$:

**base case** ($f(x)$ fast): $(y, \dot{x}) = \nabla f(x, \dot{y})$ (step 0)

**inductive case:**

$\begin{align*}
h \circ g &= f \quad \text{(step 1)} \\
z &= g(x) \quad \text{(step 2)} \\
(y, \dot{z}) &= \nabla h(z, \dot{y}) \quad \text{(step 3)} \\
(z, \dot{x}) &= \nabla g(x, z) \quad \text{(step 4)}
\end{align*}$
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, x) = \sqrt{J} f x \dot{y}\):

- **base case** (\(f x\) fast):
  \[(y, \dot{x}) = \sqrt{J} f x \dot{y}\]  
  (step 0)

- **inductive case**:
  \[h \circ g = f\]  
  (step 1)
  \[z = g x\]  
  (step 2)
  \[(y, \dot{z}) = \sqrt{J} h z \dot{y}\]  
  (step 3)
  \[(z, \dot{x}) = \sqrt{J} g x \dot{z}\]  
  (step 4)
To compute \((y, \dot{x}) = \sqrt{J} f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast}): \) \((y, \dot{x}) = \sqrt{J} f \ x \ \dot{y}\) (step 0)

**inductive case:**

- \(h \circ g = f\) (step 1)
- \(z = g \ x\) (step 2)
- \((y, \dot{z}) = \sqrt{J} h \ z \ \dot{y}\) (step 3)
- \((z, \dot{x}) = \sqrt{J} g \ x \ \dot{z}\) (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \sqrt{\mathcal{J}} f x \dot{y}\):

**base case** \((f \times \text{fast})\):

\((y, \dot{x}) = \sqrt{\mathcal{J}} f x \dot{y}\)  \hspace{1cm} (step 0)

**inductive case**:

\(h \circ g = f\)
\(z = g x\)

\((y, \dot{z}) = \sqrt{\mathcal{J}} h z \dot{y}\)  \hspace{1cm} (step 3)

\((z, \dot{x}) = \sqrt{\mathcal{J}} g x \dot{z}\)  \hspace{1cm} (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \nabla f x \dot{y}\):

**base case** \((f x \text{ fast})\):
\((y, \dot{x}) = \nabla f x \dot{y} \quad \text{(step 0)}\)

**inductive case**: \(h \circ g = f\)
\(z = g x \quad \text{(step 2)}\)
\((y, \dot{z}) = \nabla h z \dot{y} \quad \text{(step 3)}\)
\((z, \dot{x}) = \nabla g x \dot{z} \quad \text{(step 4)}\)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \leftarrow \frac{\partial}{\partial x} f(x \dot{y})\):

**base case** \((f \ x \ fast):\) \((y, \dot{x}) = \leftarrow \frac{\partial}{\partial x} f(x \dot{y})\)  

**inductive case:**  
\(h \circ g = f\)  
\(z = g(x)\)  
\((y, \dot{z}) = \leftarrow \frac{\partial}{\partial z} h(z \dot{y})\)  
\((z, \dot{x}) = \leftarrow \frac{\partial}{\partial x} g(x \dot{z})\)

(step 0)  
(step 1)  
(step 2)  
(step 3)  
(step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \sqrt{\mathcal{J}} f x \dot{y}\):

**base case** \((f \ x \ \text{fast}):\)
\[
(y, \dot{x}) = \sqrt{\mathcal{J}} f x \dot{y}
\]  
(step 0)

**inductive case:**
\[
h \circ g = f
\]
\[
z = g x
\]  
(step 1)

\[
(y, \dot{z}) = \sqrt{\mathcal{J}} h z \dot{y}
\]  
(step 2)

\[
(z, \dot{x}) = \sqrt{\mathcal{J}} g x \dot{z}
\]  
(step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \nabla \mathcal{J} f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast})\):
\[(y, \dot{x}) = \nabla \mathcal{J} f \ x \ \dot{y}\] (step 0)

**inductive case**:
\[h \circ g = f\] (step 1)
\[z = g \ x\] (step 2)
\[(y, \dot{z}) = \nabla \mathcal{J} h \ z \ \dot{y}\] (step 3)
\[(z, \dot{x}) = \nabla \mathcal{J} g \ x \ \dot{z}\] (step 4)
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = ∇ f \cdot x \cdot y\):

**base case** \((f \cdot x \text{ fast}):\) \((y, \dot{x}) = \overset{∇}{f} \cdot x \cdot \dot{y}\) \hfill \text{(step 0)}

**inductive case:** \(h \circ g = f\)
\(z = g \cdot x\)
\((y, \dot{z}) = ∇ h \cdot z \cdot \dot{y}\) \hfill \text{(step 3)}
\((z, \dot{x}) = ∇ g \cdot x \cdot \dot{z}\) \hfill \text{(step 4)}
Algorithm for Divide-and-Conquer Checkpointing

To compute \((y, \dot{x}) = \nabla f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast}):\)

\((y, \dot{x}) = \nabla f \ x \ \dot{y}\) \hspace{1cm} (step 0)

**inductive case:**

\(h \circ g = f\)

\(z = g \ x\) \hspace{1cm} (step 1)

\((y, \dot{z}) = \nabla h \ z \ \dot{y}\) \hspace{1cm} (step 2)

\((z, \dot{x}) = \nabla g \ x \ \dot{z}\) \hspace{1cm} (step 3)
What is Needed to Implement the Algorithm?

1. Measure the length of the primal computation.
2. Interrupt the primal computation at a portion of the measured length.
3. Save the state of the interrupted computation as a capsule.
4. Resume an interrupted computation from a capsule.
What is Needed to Implement the Algorithm?

1. measure the length of the primal computation
What is Needed to Implement the Algorithm?

1. measure the length of the primal computation
2. interrupt the primal computation at a portion of the measured length
What is Needed to Implement the Algorithm?

1. measure the length of the primal computation
2. interrupt the primal computation at a portion of the measured length
3. save the state of the interrupted computation as a capsule
What is Needed to Implement the Algorithm?

1. measure the length of the primal computation
2. interrupt the primal computation at a portion of the measured length
3. save the state of the interrupted computation as a capsule
4. resume an interrupted computation from a capsule
General-Purpose Interruption and Resumption Interface

PRIMOPS $f \ x \mapsto l$  Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$  Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$  If $z = (\text{INTERRUPT } f \ x \ l)$, return $y = f(x)$. 
<table>
<thead>
<tr>
<th>PRIMOPS $f \ x \mapsto l$</th>
<th>Return the number $l$ of evaluation steps needed to compute $y = f(x)$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT $f \ x \ l \mapsto z$</td>
<td>Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.</td>
</tr>
<tr>
<td>RESUME $z \mapsto y$</td>
<td>If $z = (\text{INTERRUPT} \ f \ x \ l)$, return $y = f(x)$.</td>
</tr>
</tbody>
</table>
General-Purpose Interruption and Resumption Interface

**PRIMOPS** $f \ x \mapsto l$

Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

**INTERRUPT** $f \ x \ l \mapsto z$

Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

**RESUME** $z \mapsto y$

If $z = (\text{INTERRUPT} \ f \ x \ l)$, return $y = f(x)$.
General-Purpose Interruption and Resumption Interface

PRIMOPS $f \ x \mapsto l$  
Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$  
Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$  
If $z = (\text{INTERRUPT } f \ x \ l)$, return $y = f(x)$. 
PRIMOPS $f \ x \mapsto l$

Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$

Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$

If $z = (\text{INTERRUPT } f \ x \ l)$, return $y = f(x)$. 
PRIMOPS $f \ x \mapsto \ l$ \hspace{1em} Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto \ z$ \hspace{1em} Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto \ y$ \hspace{1em} If $z = (\text{INTERRUPT } f \ x \ l)$, return $y = f(x)$. 
PRIMOPS $f \ x \mapsto l$  
Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$  
Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$  
If $z = (\text{INTERRUPT} \ f \ x \ l)$, return $y = f(x)$.
<table>
<thead>
<tr>
<th>PRIMOPS $f \ x \leftrightarrow l$</th>
<th>Return the number $l$ of evaluation steps needed to compute $y = f(x)$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT $f \ x \ l \leftrightarrow z$</td>
<td>Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.</td>
</tr>
<tr>
<td>RESUME $z \leftrightarrow y$</td>
<td>If $z = \text{INTERRUPT} \ f \ x \ l$, return $y = f(x)$.</td>
</tr>
</tbody>
</table>
PRIMOPS $f \ x \mapsto l$  
Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$  
Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$  
If $z = (\text{INTERRUPT} \ f \ x \ l)$, return $y = f(x)$.
PRIMOPS $f \ x \mapsto l$ \quad Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$ \quad Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$ \quad If $z = (\text{INTERRUPT} \ f \ x \ l)$, return $y = f(x)$.
PRIMOPS $f \ x \mapsto l$  
Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$  
Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$  
If $z = (\text{INTERRUPT } f \ x \ l)$, return $y = f(x)$. 
PRIMOPS $f \ x \mapsto l$ Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto z$ Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto y$ If $z = \text{(INTERRUPT} \ f \ x \ l\text{)}$, return $y = f(x)$. 
PRIMOPS $f \ x \mapsto \ l$  
Return the number $l$ of evaluation steps needed to compute $y = f(x)$.

INTERRUPT $f \ x \ l \mapsto \ z$  
Run the first $l$ steps of the computation of $f(x)$ and return a capsule $z$.

RESUME $z \mapsto \ y$  
If $z = (\text{INTERRUPT} \ f \ x \ l)$, return $y = f(x)$. 
PRIMOPS \( f \ x \mapsto l \)  
Return the number \( l \) of evaluation steps needed to compute \( y = f(x) \).

INTERRUPT \( f \ x \ l \mapsto z \)  
Run the first \( l \) steps of the computation of \( f(x) \) and return a capsule \( z \).

RESUME \( z \mapsto y \)  
If \( z = (\text{INTERRUPT} \ f \ x \ l) \), return \( y = f(x) \).
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \nabla \mathcal{J} f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast})\):
\[
(y, \dot{x}) = \nabla \mathcal{J} f \ x \ \dot{y}
\]  
(step 0)

**inductive case:**
\[
h \circ g = f
\]
\[
z = g \ x
\]  
(step 1)

\[
(y, \dot{z}) = \nabla \mathcal{J} h \ z \ \dot{y}
\]  
(step 2)

\[
(z, \dot{x}) = \nabla \mathcal{J} g \ x \ \dot{z}
\]  
(step 3)
To compute \((y, \dot{x}) = \sqrt{J} f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast})\):
\[(y, \dot{x}) = \sqrt{J} f \ x \ \dot{y}\] (step 0)

**inductive case**:
\[h \circ g = f\] (step 1)
\[z = g \ x\] (step 2)
\[(y, \dot{z}) = \sqrt{J} h \ z \ \dot{y}\] (step 3)
\[(z, \dot{x}) = \sqrt{J} g \ x \ \dot{z}\] (step 4)
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, x) = \sqrt{J} f x \dot{y}\):

**base case** \((f x \text{ fast})\):
\((y, x) = \sqrt{J} f x \dot{y}\) \hspace{1cm} (step 0)

**inductive case**: \(h \circ g = f\)
\(z = g x\) \hspace{1cm} (step 1)
\(y, z = \sqrt{J} h z \dot{y}\) \hspace{1cm} (step 2)
\((z, x) = \sqrt{J} g x \dot{z}\) \hspace{1cm} (step 3)

Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \mathcal{J} f x \dot{y}\):

**base case** (\(f x\) fast): \((y, \dot{x}) = \mathcal{J} f x \dot{y}\)  \hspace{1cm} (step 0)

**inductive case:**

\[ l = \text{PRIMOPS} f x \] \hspace{1cm} (step 1)

\[ z = g x \] \hspace{1cm} (step 2)

\((y, \dot{z}) = \mathcal{J} h z \dot{y}\) \hspace{1cm} (step 3)

\((z, \dot{x}) = \mathcal{J} g x \dot{z}\) \hspace{1cm} (step 4)
To compute $(y, \dot{x}) = \mathcal{J} f x \dot{y}$:

**base case** $(f \ x$ fast): \[(y, \dot{x}) = \mathcal{J} f x \dot{y} \quad \text{(step 0)}\]

**inductive case:**
- $l = \text{PRIMOPS} f x$ \quad \text{(step 1)}
- $z = g x$ \quad \text{(step 2)}
- $(y, \dot{z}) = \mathcal{J} h z \dot{y}$ \quad \text{(step 3)}
- $(z, \dot{x}) = \mathcal{J} g x \dot{z}$ \quad \text{(step 4)}
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \( (y, x) = \mathcal{J} f \ x \dot{y} \):

**base case** \((f \ x \text{ fast})\): \( (y, \dot{x}) = \mathcal{J} f \ x \dot{y} \)  

**inductive case**: \( l = \text{PRIMOPS} f \ x \)  
\( z = g \ x \)  
\( (y, \dot{z}) = \mathcal{J} h \ z \dot{y} \)  
\( (z, \dot{x}) = \mathcal{J} g \ x \dot{z} \)
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \sqrt[J]{f(x \dot{y})}\):

**base case** \((f \ x \ fast)\):
\[(y, \dot{x}) = \sqrt[J]{f(x \dot{y})}\] (step 0)

**inductive case**:
\[l = \text{PRIMOPS } f(x)\] (step 1)
\[z = \text{INTERRUPT } f(x) \left[ \frac{l}{2} \right]\] (step 2)
\[(y, \dot{z}) = \sqrt[J]{h(z \dot{y})}\] (step 3)
\[(z, \dot{x}) = \sqrt[J]{g(x \dot{z})}\] (step 4)
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \check{\mathcal{J}} f x \dot{y}\):

**base case** \((f x \text{ fast}):\)
\[
(y, \dot{x}) = \check{\mathcal{J}} f x \dot{y} \quad \text{(step 0)}
\]

**inductive case:**
\[
l = \text{PRIMOPS} f x \quad \text{(step 1)}
\]
\[
z = \text{INTERRUPT} f x \left[ \frac{l}{2} \right] \quad \text{(step 2)}
\]
\[
(y, \dot{z}) = \check{\mathcal{J}} h z \dot{y} \quad \text{(step 3)}
\]
\[
(z, \dot{x}) = \check{\mathcal{J}} g x \dot{z} \quad \text{(step 4)}
\]
To compute $(y, \hat{x}) = \checkmark f x \hat{y}$:

**base case** $(f \ x \ fast)$: $(y, \hat{x}) = \checkmark f x \hat{y}$  

**inductive case:**

$l = \text{PRIMOPS} f x$  

$z = \text{INTERRUPT} f x \lfloor \frac{l}{2} \rfloor$  

$(y, \hat{z}) = \checkmark h z \hat{y}$  

$(z, \hat{x}) = \checkmark g x \hat{z}$
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \overleftarrow{\mathcal{J}} f x \dot{y}\):

- **base case** \((f x \text{ fast})\):
  \[(y, \dot{x}) = \overleftarrow{\mathcal{J}} f x \dot{y}\]  (step 0)

- **inductive case**:
  \[l = \text{PRIMOPS } f x\]  (step 1)
  \[z = \text{INTERRUPT } f x \left\lfloor \frac{l}{2} \right\rfloor\]  (step 2)
  \[(y, \dot{z}) = \overleftarrow{\mathcal{J}} (\lambda z.\text{RESUME } z) z \dot{y}\]  (step 3)
  \[(z, \dot{x}) = \overleftarrow{\mathcal{J}} g x \dot{z}\]  (step 4)
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \mathcal{J} f \ x \ \dot{y}\):

**base case** \((f \ x \ \text{fast})\): \((y, \dot{x}) = \mathcal{J} f \ x \ \dot{y}\) \hspace{1cm} (step 0)

**inductive case**: \(l = \text{PRIMOPS} f \ x\) \hspace{1cm} (step 1)
\(z = \text{INTERRUPT} f \ x \lfloor \frac{l}{2} \rfloor\) \hspace{1cm} (step 2)
\((y, \dot{z}) = \mathcal{J} (\lambda z.\text{RESUME} z) z \ \dot{y}\) \hspace{1cm} (step 3)
\((z, \dot{x}) = \mathcal{J} g \ x \ \dot{z}\) \hspace{1cm} (step 4)
Algorithm for Divide-and-Conquer Checkpointing
via General-Purpose Interruption and Resumption Interface

To compute \((y, \dot{x}) = \nabla f x \dot{y}\):

**base case** \((f \ x \ \text{fast})\):
\[(y, \dot{x}) = \nabla f x \dot{y} \quad \text{(step 0)}\]

**inductive case**:
\[l = \text{PRIMOPS } f x \quad \text{(step 1)}\]
\[z = \text{INTERRUPT } f x \left[ \frac{l}{2} \right] \quad \text{(step 2)}\]
\[(y, \dot{z}) = \nabla \left( \lambda z. \text{RESUME } z \right) z \dot{y} \quad \text{(step 3)}\]
\[(z, \dot{x}) = \nabla g x \dot{z} \quad \text{(step 4)}\]
To compute \((y, \dot{x}) = \overline{\mathcal{J}} f x \dot{y}\):

**base case** \((f \ x \text{ fast})\): \((y, \dot{x}) = \overline{\mathcal{J}} f x \dot{y}\) \hspace{1cm} (step 0)

**inductive case**: \(l = \text{PRIMOPS} f x\) \hspace{1cm} (step 1)
\(z = \text{INTERRUPT} f x \lfloor \frac{l}{2} \rfloor\) \hspace{1cm} (step 2)

\((y, \dot{z}) = \overline{\mathcal{J}} (\lambda z. \text{RESUME} z) \ z \dot{y}\) \hspace{1cm} (step 3)

\((z, \dot{x}) = \overline{\mathcal{J}} (\lambda x. \text{INTERRUPT} f x \lfloor \frac{l}{2} \rfloor) \ x \dot{z}\) \hspace{1cm} (step 4)
Example of CPS Conversion

```latex
\begin{align*}
\text{function } f(x) \\
& \quad \text{return } q(p(g(x), h(x))) \\
\end{align*}
\quad \rightarrow \quad \begin{cases}
\text{function } f(c, x) \\
& \quad \text{return } g(\text{function}(t1)) \\
& \quad \hspace{1cm} \text{return } h(\text{function}(t2)) \\
& \quad \hspace{2cm} \text{return } p(\text{function}(t3)) \\
& \quad \hspace{3cm} \text{return } q(c, t3) \\
& \hspace{4cm} \text{end, } t1, t2 \\
& \quad \hspace{5cm} \text{end, } x) \\
\end{cases}
\end{align*}
```
Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and written in C.
5. Compile to machine code.
1. Convert source program to CPS.
**Implementation**

1. Convert source program to CPS.
2. Thread step count and limit.

---

Siskind (Elmore Family School of ECE, Purdue)  
An FP Framework for Deep Learning  
Meta 16 December 2021
Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and ✓✓ written in C.
Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and ✓ written in C.
5. Compile to machine code.
Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and ✓ written in C.
5. Compile to machine code.
\[
\begin{align*}
[x|k] & \xrightarrow{ } k\ x \\
[(\lambda x.e)|k] & \xrightarrow{ } k\ (\lambda k\ x.[e|k']) \\
[(e_1\ e_2)|k] & \xrightarrow{ } [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1\ k\ x_2))])] \\
\ e_0 & \xrightarrow{ } [e_0|(\lambda x.x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \leadsto k \ x \\
(\lambda x. e)|k & \leadsto k \ (\lambda k' \ x. [e|k']) \\
(e_1 \ e_2)|k & \leadsto [e_1|(\lambda x_1. [e_2|(\lambda x_2. (x_1 \ k_2))])]) \\
e_0 & \leadsto [e_0|(\lambda x. x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \rightsquigarrow k \ x \\
[(\lambda x. e)|k] & \rightsquigarrow k \ (\lambda k' \ x.[e|k']) \\
[(e_1 \ e_2)|k] & \rightsquigarrow [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1 \ k \ x_2))])]
\end{align*}
\]

\[
e_0 \rightsquigarrow [e_0|(\lambda x.x)]
\]
\[
\begin{align*}
[x|k] & \rightsquigarrow k \ x \\
[(\lambda x. e)|k] & \rightsquigarrow k \ (\lambda k' \ x. [e|k']) \\
[(e_1 \ e_2)|k] & \rightsquigarrow [e_1|(\lambda x_1. [e_2|(\lambda x_2. (x_1 \ k \ x_2))])'] \\
 e_0 & \rightsquigarrow [e_0|(\lambda x. x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \rightarrow k \ x \\
[(\lambda x.e)|k] & \rightarrow k \ (\lambda k' \ x.[e|k']) \\
[(e_1 \ e_2)|k] & \rightarrow [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1 \ k \ x_2))|)] \\
e_0 & \rightarrow [e_0|(\lambda x.x)]
\end{align*}
\]
CPS Conversion as a Program Transformation

\[
\begin{align*}
[x|k] & \rightsquigarrow k \; x \\
[(\lambda x. e)|k] & \rightsquigarrow k \; (\lambda k' x. [e|k']) \\
[(e_1 \; e_2)|k] & \rightsquigarrow [e_1|(\lambda x_1. [e_2|(\lambda x_2. (x_1 \; k \; x_2))]|)] \\
e_0 & \rightsquigarrow [e_0|(\lambda x. x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \sim k \ x \\
[(\lambda x. \ e)|k] & \sim k \ (\lambda k' \ x. \ [e|k']) \\
[(e_1 \ e_2)|k] & \sim [e_1|(\lambda x_1. \ [e_2|\ (\lambda x_2. \ (x_1 \ k \ x_2)) ])|] \\
e_0 & \sim [e_0|(\lambda x. \ x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \leadsto k \ x \\
[(\lambda x. e)|k] & \leadsto k \ (\lambda k' \ x. [e|k']) \\
[(e_1 \ e_2)|k] & \leadsto [e_1|(\lambda x_1. [e_2|(\lambda x_2.(x_1 k x_2))])] \\
\ e_0 & \leadsto [e_0|(\lambda x.x)]
\end{align*}
\]
\[ \begin{align*}
[x|k] & \leadsto k \ x \\
[(\lambda x.e)|k] & \leadsto k \ (\lambda k' \ x.[e|k']) \\
[(e_1 e_2)|k] & \leadsto \left[ e_1 | (\lambda x_1.[e_2|(\lambda x_2.(x_1 \ k \ x_2))] \right) \\
e_0 & \leadsto \left[ e_0 | (\lambda x.x) \right]
\end{align*} \]
CPS Conversion as a Program Transformation

\[ [x|k] \leadsto k \ x \]
\[ [(\lambda x. e)|k] \leadsto k \ (\lambda k' \ x. [e|k']) \]
\[ [(e_1 e_2)|k] \leadsto [e_1|(\lambda x_1. [e_2|(\lambda x_2.(x_1 \ k \ x_2))])] \]
\[ e_0 \leadsto [e_0|(\lambda x.x)] \]
\[
\begin{align*}
[x|k] & \leadsto k\ x \\
[(\lambda x. e)|k] & \leadsto k\ (\lambda k'\ x.[e|k']) \\
[(e_1\ e_2)|k] & \leadsto [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1\ k\ x_2))])] \\
e_0 & \leadsto [e_0|(\lambda x.x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \leadsto k \ x \\
[(\lambda x.e)|k] & \leadsto k \ (\lambda k' x.[e|k']) \\
[(e_1 e_2)|k] & \leadsto [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1 k x_2))])] \\
 e_0 & \leadsto [e_0|(\lambda x.x)]
\end{align*}
\]
\[
\begin{align*}
\left[ x \mid k \right] & \leadsto k \ x \\
\left[ (\lambda x. e) \mid k \right] & \leadsto k \ (\lambda k' \ x. \left[ e \mid k' \right]) \\
\left[ (e_1 \ e_2) \mid k \right] & \leadsto \left[ e_1 \left( \lambda x_1. \left[ e_2 \left( \lambda x_2. (x_1 \ k \ x_2) \right) \right] \right) \right] \\
e_0 & \leadsto \left[ e_0 \left( \lambda x. x \right) \right]
\end{align*}
\]
\[
\begin{align*}
\left[ x | k \right] & \leadsto k \ x \\
\left[ (\lambda x. e) | k \right] & \leadsto k \ (\lambda k' \ x. \left[ e | k' \right] ) \\
\left[ (e_1 \ e_2) | k \right] & \leadsto \left[ e_1 | (\lambda x_1. \left[ e_2 | (\lambda x_2. (x_1 \ k \ x_2) ) \right] ) \right] \\
e_0 & \leadsto \left[ e_0 | (\lambda x. x) \right]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \sim k \ x \\
[(\lambda x. e)|k] & \sim k \ (\lambda k' \ x. [e|k']) \\
[(e_1 \ e_2)|k] & \sim [e_1|(\lambda x_1. [e_2|(\lambda x_2. (x_1 \ k \ x_2))|)]|] \\
\ e_0 & \sim [e_0|(\lambda x. x)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \leadsto k \; x \\
(\lambda x. e)|k & \leadsto k \; (\lambda k' \; x. [e|k']) \\
(e_1 \; e_2)|k & \leadsto [e_1](\lambda x_1. [e_2|(\lambda x_2. (x_1 \; k \; x_2))]) \\
e_0 & \leadsto [e_0|(\lambda x. x)]
\end{align*}
\]
\[
\begin{align*}
\lbrack x \mid k \rbrack & \leadsto k \ x \\
\lbrack (\lambda x. e) \mid k \rbrack & \leadsto k \ (\lambda k' \ x. \lbrack e \mid k' \rbrack) \\
\lbrack (e_1 \ e_2) \mid k \rbrack & \leadsto \lbrack e_1 \mid (\lambda x_1. \lbrack e_2 \mid (\lambda x_2. (x_1 \ k \ x_2)) \rbrack) \rbrack \\

\end{align*}
\]
\[\begin{align*}
\llbracket x | k \rrbracket & \rightsquigarrow k \ x \\
\llbracket (\lambda x. e) | k \rrbracket & \rightsquigarrow k \ (\lambda k' \ x. \llbracket e | k' \rrbracket) \\
\llbracket (e_1 \ e_2) | k \rrbracket & \rightsquigarrow \llbracket e_1 | (\lambda x_1. \llbracket e_2 | (\lambda x_2. (x_1 \ k \ x_2)) \rrbracket) \rrbracket \\
e_0 & \rightsquigarrow \llbracket e_0 | (\lambda x. x) \rrbracket
\end{align*}\]
\[
\begin{align*}
[x|k] & \rightarrow k \ x \\
[(\lambda x. e)|k] & \rightarrow k \ (\lambda k' \ x.[e|k']) \\
[(e_1 \ e_2)|k] & \rightarrow [e_1|(\lambda x_1. [e_2|(\lambda x_2.(x_1 \ k \ x_2))])'] \\
 e_0 & \rightarrow [e_0|(\lambda x.x)]
\end{align*}
\]
\[
\begin{align*}
\llbracket x|k \rrbracket & \leadsto k \cdot x \\
\llbracket (\lambda x.e)|k \rrbracket & \leadsto k \cdot (\lambda k' \cdot \llbracket e|k' \rrbracket) \\
\llbracket (e_1 \cdot e_2)|k \rrbracket & \leadsto \llbracket e_1|k \rrbracket \cdot (\lambda x_1. \llbracket e_2|k \rrbracket \cdot (\lambda x_2. (x_1 \cdot k \cdot x_2))) \\
\llbracket e_0 \rrbracket & \leadsto \llbracket e_0|k \rrbracket \cdot (\lambda x. x)
\end{align*}
\]
\[
\begin{align*}
[x|k] & \sim k \, x \\
[(\lambda x. e)|k] & \sim k \, (\lambda k' \, x. [e|k']) \\
[(e_1 \, e_2)|k] & \sim \big[e_1|\big(\lambda x_1. [e_2|\big(\lambda x_2. (x_1 \, k \, x_2)\big)\big]\big]\big] \\
& \quad e_0 \sim [e_0|\big(\lambda x. x\big)]
\end{align*}
\]
\[
\begin{align*}
[x|k] & \leadsto k \ x \\
[(\lambda x.e)|k] & \leadsto k \ (\lambda k' \ x.[e|k']) \\
[(e_1 \ e_2)|k] & \leadsto [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1 \ k \ x_2))])]
\end{align*}
\]
\[
e_0 \leadsto [e_0|(\lambda x.x)]
\]
\[
\begin{align*}
[x|k] & \Rightarrow k \ x \\
[(\lambda x.e)|k] & \Rightarrow k \ (\lambda k' \ x. [e|k']) \\
[(e_1 \ e_2)|k] & \Rightarrow [e_1|(\lambda x_1. [e_2|(\lambda x_2.(x_1 \ k \ x_2))])]
\end{align*}
\]

\[
e_0 \Rightarrow [e_0|(\lambda x. x)]
\]
\[
\begin{align*}
\left[x \mid k\right] & \leadsto k \ x \\
\left[(\lambda x. e) \mid k\right] & \leadsto k \ (\lambda k' \ x. \left[e \mid k'\right]) \\
\left[(e_1 \ e_2) \mid k\right] & \leadsto \left[e_1 \mid (\lambda x_1. \left[e_2 \mid (\lambda x_2. (x_1 \ k \ x_2))\right])\right] \\
\ e_0 & \leadsto \left[e_0 \mid (\lambda x. x)\right]
\end{align*}
\]
\[
[x|k] \leadsto k \ x
\]
\[
[(\lambda x.e)|k] \leadsto k \ (\lambda k' \ x.[e|k'])
\]
\[
[(e_1 \ e_2)|k] \leadsto [e_1|(\lambda x_1.[e_2|(\lambda x_2.(x_1 \ k \ x_2))])]
\]
\[
e_0 \leadsto [e_0|(\lambda x.x)]
\]
\[
\begin{align*}
\llbracket x \mid k \rrbracket & \leadsto k \ x \\
\llbracket (\lambda x. e) \mid k \rrbracket & \leadsto k \ (\lambda k' \ x. \llbracket e \mid k' \rrbracket) \\
\llbracket (e_1 \ e_2) \mid k \rrbracket & \leadsto \llbracket e_1 \mid (\lambda x_1. \llbracket e_2 \mid (\lambda x_2. (x_1 \ k \ x_2) \rrbracket) \rrbracket \rrbracket \\
e_0 & \leadsto \llbracket e_0 \mid (\lambda x. x) \rrbracket
\end{align*}
\]
Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and √ written in C.
5. Compile to machine code.
\[ [x|k] \sim k \quad x \]
\[ [(\lambda x.e)|k] \sim k \quad (\lambda k \quad x.[e|k]) \]
\[ [(e_1 \ e_2)|k] \sim [e_1|(\lambda \quad x_1.
\quad [e_2|(\lambda \quad x_2.
\quad (x_1 \ k \quad x_2))], \]
\]
\[
\begin{align*}
\llbracket x \rrbracket_k & \leadsto k \quad x \\
\llbracket (\lambda x. e) \rrbracket_k & \leadsto k \quad (\lambda k \quad x. \llbracket e \rrbracket_k) \\
\llbracket (e_1 \ e_2) \rrbracket_k & \leadsto \llbracket e_1 \rrbracket(\lambda x_1. \\
& \quad \llbracket e_2 \rrbracket(\lambda x_2. \\
& \quad \quad (x_1 \ k \quad x_2)), \\
& \quad \quad ), \\
& \quad \quad )
\end{align*}
\]

\[
\begin{align*}
[x|k, n] & \leadsto k(n+1) x \\
(\lambda x.e)|k, n & \leadsto k(n+1)(\lambda k n x.[e|k, n]) \\
(e_1 e_2)|k, n & \leadsto e_1(\lambda n x_1.
\begin{aligned}
&[e_2(\lambda n x_2.
\begin{aligned}
&(x_1 k n x_2)), \\
&n), \\
&(n+1)
\end{aligned}
\end{aligned})
\end{align*}
\]
\[
\begin{align*}
\unlceil x \rceil_{k, n, l} & \Rightarrow k (n + 1) \; l \; x \\
\unlceil (\lambda x . e) \rceil_{k, n, l} & \Rightarrow k (n + 1) \; l \; (\lambda k \; n \; l \; x . \unlceil e \rceil_{k, n, l}) \\
\unlceil (e_1 \; e_2) \rceil_{k, n, l} & \Rightarrow \unlceil e_1 \rceil_{(\lambda n \; l \; x_1 . \\
& \quad \unlceil e_2 \rceil_{(\lambda n \; l \; x_2 . \)
& \quad (x_1 \; k \; n \; l \; x_2) ,
& \quad n, l') ,
& \quad (n + 1), l'}
\end{align*}
\]
\begin{align*}
\llbracket x \rrbracket_{k,n,l} & \leadsto k (n + 1) \ l \ x \\
\llbracket (\lambda x. e) \rrbracket_{k,n,l} & \leadsto k (n + 1) \ l \ (\lambda k \ n \ l \ x. \llbracket e \rrbracket_{k,n,l}) \\
\llbracket (e_1 \ e_2) \rrbracket_{k,n,l} & \leadsto \llbracket e_1 \rrbracket (\lambda n \ l \ x_1. \\
& \llbracket e_2 \rrbracket (\lambda n \ l \ x_2. \\
& \quad (x_1 \ k \ n \ l \ x_2)), \\
& \quad n, l') \\
& \quad (n + 1), l') \\
\end{align*}

\langle e \rangle_{k,n,l} \leadsto \text{if } n = l \text{ then } [k, \lambda k \ n \ l . e] \text{ else } e
\[
\begin{align*}
\llbracket x \rrbracket_{k,n,l} & \sim \llbracket k \ (n+1) \ l \ x \rrbracket_{k,n,l} \\
\llbracket (\lambda x. e) \rrbracket_{k,n,l} & \sim \llbracket k \ (n+1) \ l \ (\lambda x. \ llbracket e \rrbracket_{k,n,l} \ x) \rrbracket_{k,n,l} \\
\llbracket (e_1 \ e_2) \rrbracket_{k,n,l} & \sim \llbracket \llbracket e_1 \rrbracket (\lambda n \ l \ x_1. \llbracket e_2 \rrbracket (\lambda n \ l \ x_2. \ x_1 \ k \ n \ l \ x_2), \ n, l', (n+1), l'' \rrbracket)_{k,n,l} \rrbracket_{k,n,l}
\end{align*}
\]

\[
\llbracket e \rrbracket_{k,n,l} \sim \text{if } n = l \text{ then } \llbracket k, \lambda k \ n \ l \ . \ e \rrbracket \text{ else } e
\]
\[ \boxed{\text{Treading Step Counts and Limits in CPS Conversion}} \]

\[ \boxed{\boxed{[x|k,n,l] \rightsquigarrow \llangle k \ (n+1) \ l \ x \rrangle_{k,n,l}} \}

\[ \boxed{\boxed{[\lambda x.e]|k,n,l] \rightsquigarrow \llangle k \ (n+1) \ l \ \lambda k \ n \ l \ x.\llangle e|k,n,l'] \rrangle_{k,n,l}} \] \]

\[ \boxed{\boxed{[e_1 \ e_2]|k,n,l] \rightsquigarrow \llangle [e_1](\lambda n \ l \ x_1. \}} \]

\[ \boxed{\boxed{[e_2](\lambda n \ l \ x_2. \}} \]

\[ \boxed{\boxed{(x_1 \ k \ n \ l \ x_2)), \ \}} \]

\[ \boxed{\boxed{n, l_1')}, \ \}} \]

\[ \boxed{\boxed{(n+1), l_1']} \rrangle_{k,n,l}} \] \]

\[ \boxed{\vdots} \]

\[ \boxed{\llangle e\rrangle_{k,n,l} \rightsquigarrow \text{if } n = l \ \text{then } [k, \lambda k \ n \ l \ \_ \ e] \ \text{else } e} \]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = A (\lambda n l v.n) 0 \infty f x \\
\text{INTERRUPT } f \ x \ l = A (\lambda n l v.v) 0 l f x \\
\text{RESUME } [k,f] = A k 0 \infty f \bot
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\begin{align*}
\text{PRIMOPS } f \ x &= \mathcal{A} (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \\
\text{INTERRUPT } f \ x \ l &= \mathcal{A} (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \\
\text{RESUME } [k, f] &= \mathcal{A} \ k \ 0 \ \infty \ f \ \bot
\end{align*}
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = \mathcal{A} (\lambda n \ l \ v. n) \ 0 \ \infty f \ x
\]
\[
\text{INTERRUPT } f \ x \ l = \mathcal{A} (\lambda n \ l \ v. v) \ 0 \ l f \ x
\]
\[
\text{RESUME } [k, f] = \mathcal{A} k \ 0 \ \infty f \ \bot
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = A (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x
\]
\[
\text{INTERRUPT } f \ x \ l = A (\lambda n \ l \ v. v) \ 0 \ l \ f \ x
\]
\[
\text{RESUME } [k, f] = A k \ 0 \ \infty \ f \ \bot
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

PRIMOPS \( f \ x = A (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \)

INTERRUPT \( f \ x \ l = A (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \)

RESUME \([k,f] = A k 0 \ \infty \ f \ \perp \)
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\begin{align*}
\text{PRIMOPS } f x &= \mathcal{A} (\lambda n \; l \; v. n) \; 0 \; \infty \; f \; x \\
\text{INTERRUPT } f x l &= \mathcal{A} (\lambda n \; l \; v. v) \; 0 \; l \; f \; x \\
\text{RESUME } [k,f] &= \mathcal{A} k \; 0 \; \infty \; f \; \bot
\end{align*}
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[\text{PRIMOPS } f \ x = \mathcal{A} \ (\lambda n \ l \ v.n) \ 0 \ \infty f \ x\]
\[\text{INTERRUPT } f \ x \ l = \mathcal{A} \ (\lambda n \ l \ v.v) \ 0 \ l \ f \ x\]
\[\text{RESUME } [k,f] = \mathcal{A} \ k \ 0 \ \infty f \ \bot\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = A (\lambda n \ l \ v.n) \ 0 \ \infty \ f \ x \\
\text{INTERRUPT } f \ x \ l = A (\lambda n \ l \ v.v) \ 0 \ l \ f \ x \\
\text{RESUME } [k,f] = A k \ 0 \ \infty \ f \ \bot
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\begin{align*}
\text{PRIMOPS } f x &= \mathcal{A} (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \\
\text{INTERRUPT } f \ x \ l &= \mathcal{A} (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \\
\text{RESUME } [k, f] &= \mathcal{A} k \ 0 \ \infty \ f \ \bot
\end{align*}
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = \mathcal{A} (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \\
\text{INTERRUPT } f \ x \ l = \mathcal{A} (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \\
\text{RESUME } \lfloor k, f \rfloor = \mathcal{A} k 0 \ \infty \ f \ \bot
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

PRIMOPS \( f \ x = A (\lambda n \ l \ v . n) \ 0 \ \infty f \ x \)

INTERRUPT \( f \ x \ l = A (\lambda n \ l \ v . v) \ 0 \ l f \ x \)

RESUME \( [k, f] = A \ k \ 0 \ \infty f \ \perp \)
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

PRIMOPS \( f \ x = A (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \)

INTERRUPT \( f \ x \ l = A (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \)

RESUME \( [k,f] = A k 0 \ \infty \ f \ \perp \)
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = \mathcal{A} (\lambda n \ l \ v. n) \ 0 \ \infty f \ x \\
\text{INTERRUPT } f \ x \ l = \mathcal{A} (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \\
\text{RESUME } [k, f] = \mathcal{A} k 0 \ \infty f \ \bot
\]
PRIMOPS $f \ x = A \ (\lambda n \ l \ v \ . \ n) \ 0 \ \infty \ f \ x$

INTERRUPT $f \ x \ l = A \ (\lambda n \ l \ v \ . \ v) \ 0 \ l \ f \ x$

RESUME $[k,f] = A \ k \ 0 \ \infty \ f \ \perp$
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[ \text{PRIMOPS } f \ x = \text{A} (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \]

\[ \text{INTERRUPT } f \ x \ l = \text{A} (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \]

\[ \text{RESUME } [k,f] = \text{A} \ k \ 0 \ \infty \ f \ \bot \]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

PRIMOPS $f\ x = \mathcal{A} (\lambda\ n\ l\ v.n)\ 0\ \infty\ f\ x$

INTERRUPT $f\ x\ l = \mathcal{A} (\lambda\ n\ l\ v.v)\ 0\ l\ f\ x$

RESUME $[k,f] = \mathcal{A} k\ 0\ \infty\ f\ \bot$
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[
\text{PRIMOPS } f \ x = \mathcal{A} (\lambda n \ l \ v.n) \ 0 \ \infty \ f \ x
\]

\[
\text{INTERRUPT } f \ x \ l = \mathcal{A} (\lambda n \ l \ v.v) \ 0 \ l \ f \ x
\]

\[
\text{RESUME } [k,f] = \mathcal{A} k \ 0 \ \infty \ f \ \bot
\]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[ \text{PRIMOPS } f \ x = \ A \ (\lambda \ n \ l \ v. \ n) \ 0 \ \infty \ f \ x \]
\[ \text{INTERRUPT } f \ x \ l = \ A \ (\lambda \ n \ l \ v. \ v) \ 0 \ l \ f \ x \]
\[ \text{RESUME } [k, f] = \ A \ k \ 0 \ \infty \ f \ \bot \]
## Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. **Translate CPS to C.**
4. Combine with general-purpose interruption and resumption interface and \( \checkmark \) written in C.
5. Compile to machine code.
\[ S \pi() = \text{null\_constant} \]
\[ S \pi \text{true} = \text{true\_constant} \]
\[ S \pi \text{false} = \text{false\_constant} \]
\[ S \pi (c_1, c_2) = \text{cons}((S \pi c_1), (S \pi c_1)) \]
\[ S \pi n \]
\[ S \pi 'k' = \text{continuation} \]
\[ S \pi 'n' = \text{count} \]
\[ S \pi 'l' = \text{limit} \]
\[ S \pi 'x' = \text{argument} \]
\[ S \pi x = \text{as\_closure}(	ext{target}) -> \text{environment}[\pi x] \]
\[ S \, \pi \,(\lambda_3 n \, l \, x. \, e) = (\{ \]
\[
\text{thing function(thing target,} \\
\text{thing count,} \\
\text{thing limit,} \\
\text{thing argument}) \} \]
\[
\text{return } (S \,(\phi \, e) \, e); \\
\}
\]
\[
\text{thing lambda} \, = \, (\text{thing}) \, \text{GC\_malloc(sizeof(struct {} \\
\quad \text{enum tag tag;} \\
\quad \text{struct {} \\
\quad \quad \text{thing \,(\ast\text{function})();} \\
\quad \quad \text{unsigned n;} \\
\quad \quad \text{thing environment[|\phi \, e|];} \\
\quad \})} \\
\})
\}
\]
\[
\text{set\_closure(lambda);} \\
\text{as\_closure(lambda)->function = } \&\text{function;} \\
\text{as\_closure(lambda)->n} \, = \, |\phi \, e|; \\
\text{as\_closure(lambda)->environment[0]} \, = \, S \, \pi \,(\phi \, e)_0 \\
\text{;} \\
\text{as\_closure(lambda)->environment[|\phi \, e|\,-1]} \, = \, S \, \pi \,(\phi \, e)|_{\phi \, e|\,-1} \\
\text{lambda;}
\})
\]
\( S \pi (\lambda knlx.e) = \{ \}

\begin{verbatim}
    thing function(thing target,
        thing continuation,
        thing count,
        thing limit,
        thing argument) {

        return (S (\phi e) e);
    }

    thing lambda = (thing)GC_malloc(sizeof(struct {
        enum tag tag;
        struct {
            thing (*function)();
            unsigned n;
            thing environment[|\phi e|];
        }
    })

    set_closure(lambda);
    as_closure(lambda)->function = &function;
    asClosure(lambda)->n = |\phi e|;
    asClosure(lambda)->environment[0] = S \pi (\phi e)_{0}
    :;
    asClosure(lambda)->environment[|\phi e|-1] = S \pi (\phi e)|_{\phi e|-1}
    lambda;

    }
\end{verbatim}
\[ S \pi (e_1 e_2 e_3 e_4) = \text{continuation}_\text{apply}((S \pi e_1),
(S \pi e_2),
(S \pi e_3),
(S \pi e_4)) \]
\[ S \pi (e_1 e_2 e_3 e_4 e_5) = \text{converted}_\text{apply}((S \pi e_1),
(S \pi e_2),
(S \pi e_3),
(S \pi e_4),
(S \pi e_5)) \]
\[ S \pi (\text{if} \ e_1 \ \text{then} \ e_2 \ \text{else} \ e_3) = (!\text{is\_false}((S \pi e_1)) \ ? (S \pi e_2) : (S \pi e_3)) \]
\[ S \pi (\L e_1 e_2) = (N \ L)((S \pi e)) \]
\[ S \pi (e_1 \cdot e_2) = (N \cdot)((S \pi e_1), (S \pi e_2)) \]
\[ S \pi (\R e_1 e_2 e_3) = (N \ R)((S \pi e_1), (S \pi e_2), (S \pi e_3)) \]
\[ S \pi (\L e_1 e_2 e_3) = (N \ L)((S \pi e_1), (S \pi e_2), (S \pi e_3)) \]
\[ S \pi (\R e_1 e_2 e_3) = (N \ R)((S \pi e_1), (S \pi e_2), (S \pi e_3)) \]
**Implementation**

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and ✓ T written in C.
5. Compile to machine code.
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

\[ \text{PRIMOPS} \ f \ x = \mathcal{A} (\lambda n \ l \ v. n) \ 0 \ \infty \ f \ x \]

\[ \text{INTERRUPT} \ f \ x \ l = \mathcal{A} (\lambda n \ l \ v. v) \ 0 \ l \ f \ x \]

\[ \text{RESUME} \ [k,f] = \mathcal{A} \ k \ 0 \ \infty \ f \ \bot \]
Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS written in C

```c
static thing lambda_expression_that_returns_x (thing f, thing n, thing l, thing x) {
    return x;
}

static thing lambda_expression_that_returns_n (thing f, thing n, thing l, thing x) {
    return n;
}

static thing lambda_expression_that_resumes (thing f, thing continuation, thing n, thing l, thing x) {
    if (!is_interrupt(x)) internal_error();
    return converted_apply(as_interrupt(x)->closure,
                            as_interrupt(x)->continuation,
                            make_real(0.0),
                            l,
                            null_constant);
}
```
static unsigned long primops(thing f, thing x) {
    thing result = converted_apply(f,
        continuation_that_returns_n,
        make_real(0.0),
        make_real(HUGE_VAL),
        x);

    else if (is_real(result)) return (unsigned long)as_real(result);
}

static thing interrupt(thing f, thing x, thing l) {
    thing result = converted_apply(f,
        continuation_that_returns_x,
        make_real(0.0),
        l,
        x);

    if (!is_interrupt(result)) internal_error();
    return result;
}
To compute \((y, \dot{x}) = \nabla J f x \dot{y}\):

**base case** (\(f x\) fast): \((y, \dot{x}) = \nabla J f x \dot{y}\) (step 0)

**inductive case**: \(h \circ g = f\)
\[z = g x\] (step 2)

\((y, \dot{z}) = \nabla J h z \dot{y}\) (step 3)
\((z, \dot{x}) = \nabla J g x \dot{z}\) (step 4)
Algorithm for Divide-and-Conquer Checkpointing
written in C

```c
static thing checkpoint_starj(thing f, thing x, thing y_cotangent)
{
    thing loop(thing f, thing x, thing y_cotangent, unsigned long l) {
        if (l<=base_case_duration) return ternary_starj(f, x, y_cotangent);
        else {
            thing u = interrupt(f, x, make_real(l/2));
            thing y_u_cotangent = loop(closure_that_resumes, u, y_cotangent, l-l/2);
            if (!is_pair(y_u_cotangent)) internal_error();
            thing u_x_cotangent = loop(make_closure_for_interrupt(f, l/2), x,
                as_pair(y_u_cotangent)->cdr, l/2);
            if (!is_pair(u_x_cotangent)) internal_error();
            return cons(as_pair(y_u_cotangent)->car, as_pair(u_x_cotangent)->cdr);
        }
    }
    return loop(f, x, y_cotangent, primops(f, x));
}
```

Siskind (Elmore Family School of ECE, Purdue)
1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to C.
4. Combine with general-purpose interruption and resumption interface and ✓ written in C.
5. Compile to machine code.
Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator
3. Compiler using CPS conversion followed by translation to C

All three support the exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors. All three are documented in detail in the paper:


Could add FFI bindings to GPU Tensor library.
Three Reference Implementations

1. Interpreter using CPS evaluator


Could add FFI bindings to GPU Tensor library.

Siskind (Elmore Family School of ECE, Purdue)
Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator

All three support the exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors. All three documented in detail in paper.


Could add FFI bindings to GPU Tensor library.

Siskind (Elmore Family School of ECE, Purdue)
Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator
3. Compiler using CPS conversion followed by translation to C

All three support the exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors. All three documented in detail in paper.


Could add FFI bindings to GPU Tensor library.

Siskind (Elmore Family School of ECE, Purdue)
Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator
3. Compiler using CPS conversion followed by translation to C

All three support exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors.
Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator
3. Compiler using CPS conversion followed by translation to C

All three support exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors.


Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator
3. Compiler using CPS conversion followed by translation to C

All three support exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors.


Could add FFI bindings to GPU Tensor library.
Three Reference Implementations

1. Interpreter using CPS evaluator
2. Hybrid compiler/interpreter using CPS conversion followed by direct-style evaluator
3. Compiler using CPS conversion followed by translation to C

All three support exact same source language. No exceptions. Same space and time complexity. Differ only in constant factors.

All three documented in detail in paper.

Could add FFI bindings to GPU Tensor library.
**Beyond these, the standard unary basis procedures** \( \sqrt{x} \), \( e^x \), \( \log(x) \), \( \sin(x) \), \( \cos(x) \), \( \text{zero?} \), \( \text{positive?} \), and \( \text{negative?} \) **are extended pointwise to tensors** and the standard binary basis procedures \( +, - , \times, / , \text{max}, \text{min}, \text{atan}, =, <, >, \leq, \text{and} \geq \) **are extended to apply to a tensor and a scalar, or a scalar and a tensor, or two tensors of the same type and dimensions, in a pointwise fashion.**
SCORCH
ResNet

![Graph showing the relationship between run time and depth.](image)
(map (lambda (y)
      (map (lambda (x)
            (shoot-ray (list y x)))
       (iota width)))
    (iota height))
(if antecedent consequent alternate)
(shoot-ray
  (map (lambda (y)
          (map (lambda (x) (list y x))
               (iota width)))
       (iota height)))
(shoot-ray
  (list
    (map (lambda (y)
          (map (lambda (x) y) (iota width)))
         (iota height))
    (map (lambda (y)
           (map (lambda (x) x) (iota width)))
         (iota height))))
(map (lambda (y)
    (map (lambda (x) y)
        (iota width)))
    (iota height))
(define (if-function antecedent consequent alternate)
  (if antecedent consequent alternate))

(define (if-tensor antecedent consequent alternate)
  (map (lambda (antecedent consequent alternate)
      (map (lambda (antecedent consequent alternate)
        (if-function antecedent
        consequent
        alternate))
    antecedent consequent alternate))
  antecedent consequent alternate))
Target Scene
Light source at position \((-40, -15, 60)\)
Initial Scene

Light source at position \((-20 \ -30\ 30)\)
Final Scene after 111 iterations
Light source at position \((-42.7 \ -13.8 \ 39.3)\)
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>ResNet</th>
<th>GPT</th>
<th>Ray Tracer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TENSORFLOW (individual checkpointing)</strong></td>
<td>608</td>
<td>4</td>
<td>280 \times 210</td>
</tr>
<tr>
<td><strong>PYTORCH (right-branching checkpointing)</strong></td>
<td>760</td>
<td>48</td>
<td>320 \times 240</td>
</tr>
<tr>
<td><strong>DEEPSPEED</strong></td>
<td>760</td>
<td>48</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>L2L</strong></td>
<td>38,000</td>
<td>3,000</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>SCORCH</strong></td>
<td>250,000</td>
<td>14,000</td>
<td>1280 \times 960</td>
</tr>
</tbody>
</table>
Divide-and-Conquer checkpointing
Take-Home Message

Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations
Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations
- but can be extended to arbitrary code
Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations
- but can be extended to arbitrary code
- that doesn’t have same-size iterations of a single loop
Take-Home Message

Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations
- but can be extended to arbitrary code
- that doesn’t have same-size iterations of a single loop
- using CPS to make arbitrary code look like it does.
Take-Home Message

Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations
- but can be extended to arbitrary code
- that doesn’t have same-size iterations of a single loop
- using CPS to make arbitrary code look like it does.

metaphor: a CPU is an instruction-execution loop
Thank You