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

Our Work

AD in functional programs.

Our Work

AD in functional programs.

AD is easier in functional programs.

How it all started

Date: Thu, 17 May 2001 18:22:55 -0600 (MDT)
From: Barak Pearlmutter <bap@cs.unm.edu>
To: Jeffrey Mark Siskind <qobi@research.nj.nec.com>
Subject: QobiScheme.sc
Reply-to: bap@cs.unm.edu

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.

How it all started

Date: Fri, 18 May 2001 09:39:28 -0400
From: Jeffrey Mark Siskind <qobi@research.nj.nec.com>
To: bap@cs.unm.edu
Subject: Re: QobiScheme.sc
Reply-to: Qobi@research.nj.nec.com

Barak - [...]

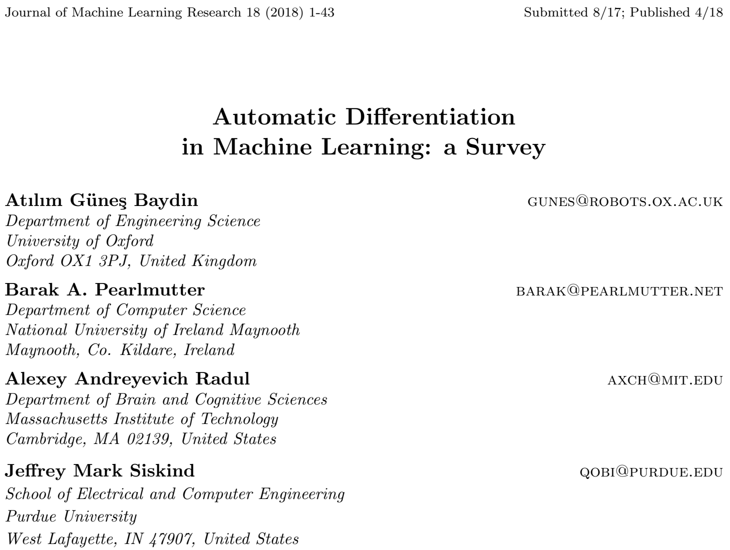
Stalingrad is a cool name!

[...]

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)



Journal of Machine Learning Research 18 (2018) 1-43 Submitted 8/17; Published 4/18

Automatic Differentiation in Machine Learning: a Survey

Atılım Güneş Baydin GUNES@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ, United Kingdom

Barak A. Pearlmutter BARAK@PEARLMUTTER.NET
Department of Computer Science
National University of Ireland Maynooth
Maynooth, Co. Kildare, Ireland

Alexey Andreyevich Radul AXCH@MIT.EDU
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, United States

Jeffrey Mark Siskind QOBI@PURDUE.EDU
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, United States

Rejected without review 3 years earlier

From: Tom Fawcett <tom.fawcett@gmail.com>
Subject: MACH-D-15-00174 "Automatic differentiation in machine learning: a survey"
Date: Thu, 11 Jun 2015 13:20:42 -0700
Cc: Peter Flach <Peter.Flach@bristol.ac.uk>
To: barak@cs.nuim.ie, axch@mit.edu, qobi@purdue.edu

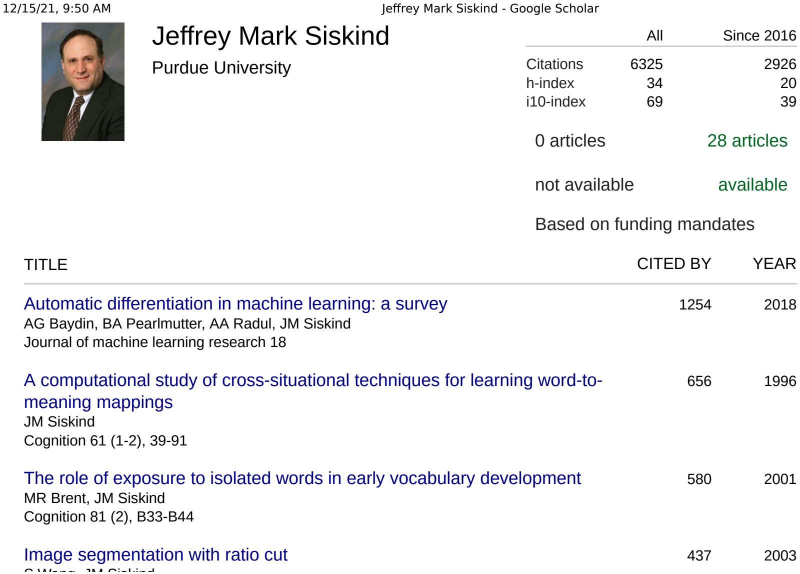
Hi. I am the survey/review editor for the Machine Learning journal. Peter assigned me your paper MACH-D-15-00174 "Automatic differentiation in machine learning: a survey".

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



12/15/21, 9:50 AM Jeffrey Mark Siskind - Google Scholar

Jeffrey Mark Siskind

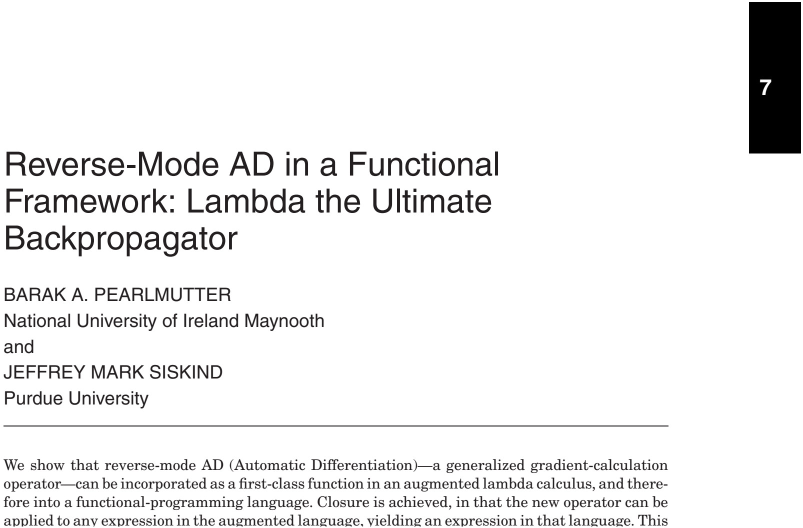
Purdue University

	All	Since 2016
Citations	6325	2926
h-index	34	20
i10-index	69	39

0 articles	28 articles
not available	available

Based on funding mandates

TITLE	CITED BY	YEAR
Automatic differentiation in machine learning: a survey AG Baydin, BA Pearlmutter, AA Radul, JM Siskind Journal of machine learning research 18	1254	2018
A computational study of cross-situational techniques for learning word-to-meaning mappings JM Siskind Cognition 61 (1-2), 39-91	656	1996
The role of exposure to isolated words in early vocabulary development MR Brent, JM Siskind Cognition 81 (2), B33-B44	580	2001
Image segmentation with ratio cut	437	2003



7

Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator

BARAK A. PEARLMUTTER
National University of Ireland Maynooth
and
JEFFREY MARK SISKIND
Purdue University

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

Date: Sun, 15 Apr 2007 20:51:05 -0400 (EDT)
From: cytron@cs.wustl.edu
To: qobi@purdue.edu
Subject: TOPLAS-00010-2005 Decision
Cc: bcpierce@cis.upenn.edu

[...]

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.

[...]

The Differentiable Curry

Abadi, Wei, Plotkin, Vytiniotis, & Belov (2019)

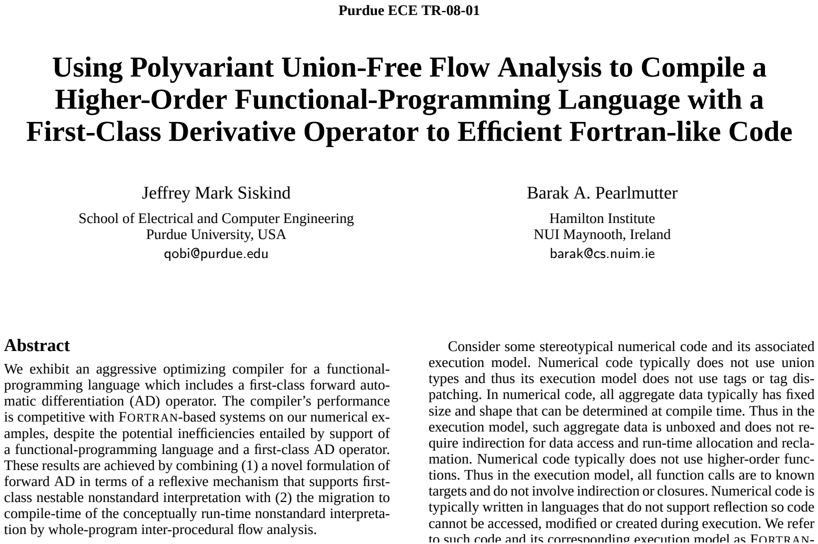
Recent projects, as Swift for TensorFlow (S4TF) (`www.tensorflow.org/swift`) and Julia Zygote [15], in the spirit of the seminal “Lambda the Ultimate Backpropagator” (LTUB) [21], advocate AD as a *first-class* construct in a general-purpose programming language, and aim to take advantage of traditional *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].



Purdue ECE TR-08-01

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-

Where you can read about this work

- POPL 2008, 2009, 2012
- PLDI 2008, 2009
- ICFP 2008

Where you can read about this work

- POPL 2008, 2009, 2012
- PLDI 2008, 200
- ICFP 2008

POPL 2009

```
                  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.
```

POPL 2008

Review #2

> [...]
>
> I would like:
>
> [...]
>
> 2. Comparison with ADIFOR, and

[...]

ADIFOR should be included and compared in Table 1.

[...]

Review #3

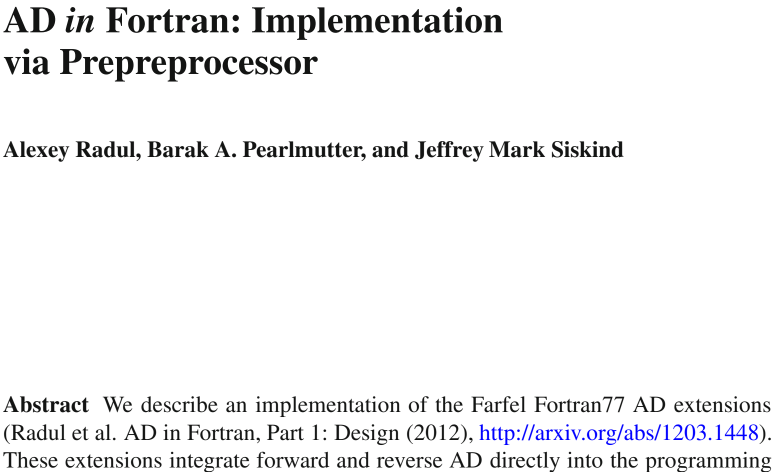
[...]

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

[...]

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

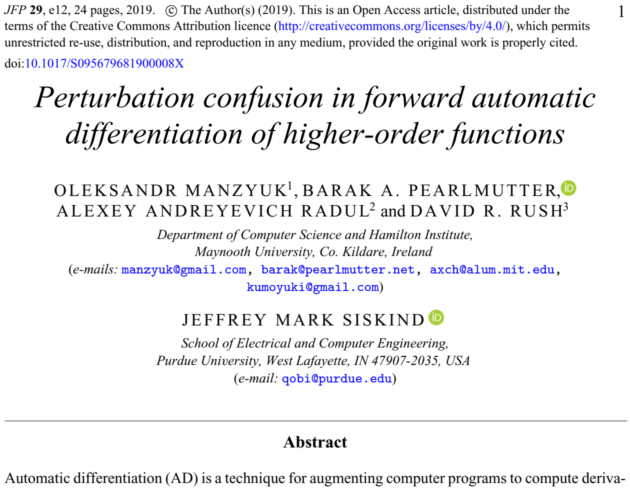
[...]



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



JFP **29**, e12, 24 pages, 2019.

doi:10.1017/S095679681900008X

Perturbation confusion in forward automatic differentiation of higher-order functions

OLEKSANDR MANZYUK[1], BARAK A. PEARLMUTTER,
ALEXEY ANDREYEVICH RADUL[2] and DAVID R. RUSH[3]
Department of Computer Science and Hamilton Institute,
Maynooth University, Co. Kildare, Ireland
(*e-mails:* `manzyuk@gmail.com`, `barak@pearlmutter.net`, `axch@alum.mit.edu`,
`kumoyuki@gmail.com`)

JEFFREY MARK SISKIND
School of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN 47907-2035, USA
(*e-mail:* `qobi@purdue.edu`)

Abstract

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

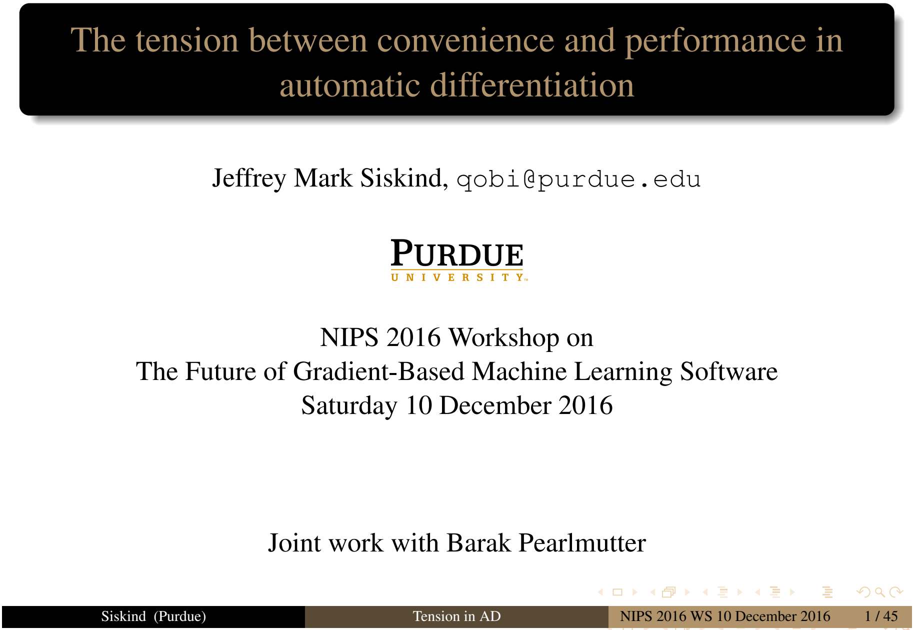
AD for Probabilistic Programming

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.



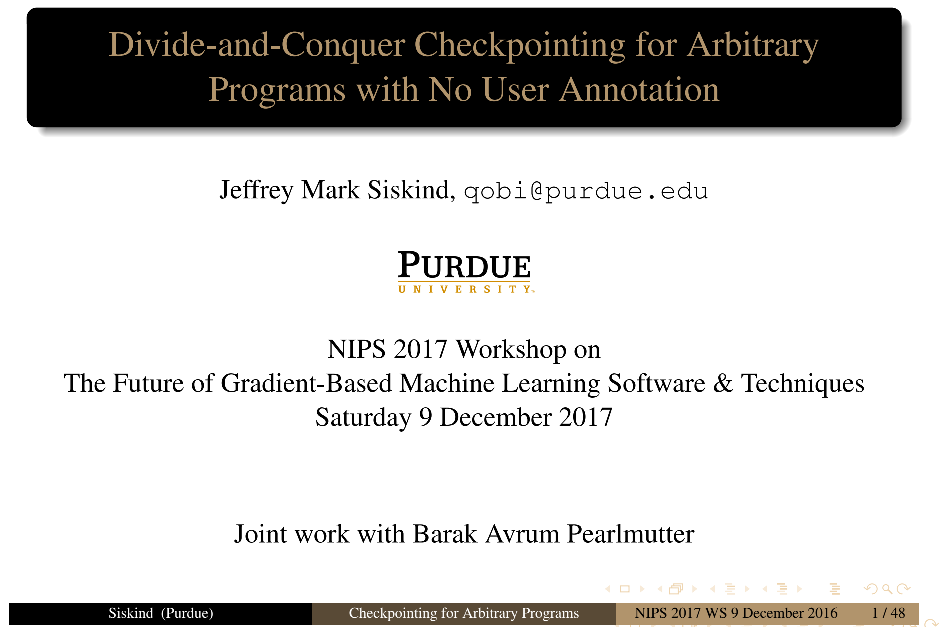
The tension between convenience and performance in automatic differentiation

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

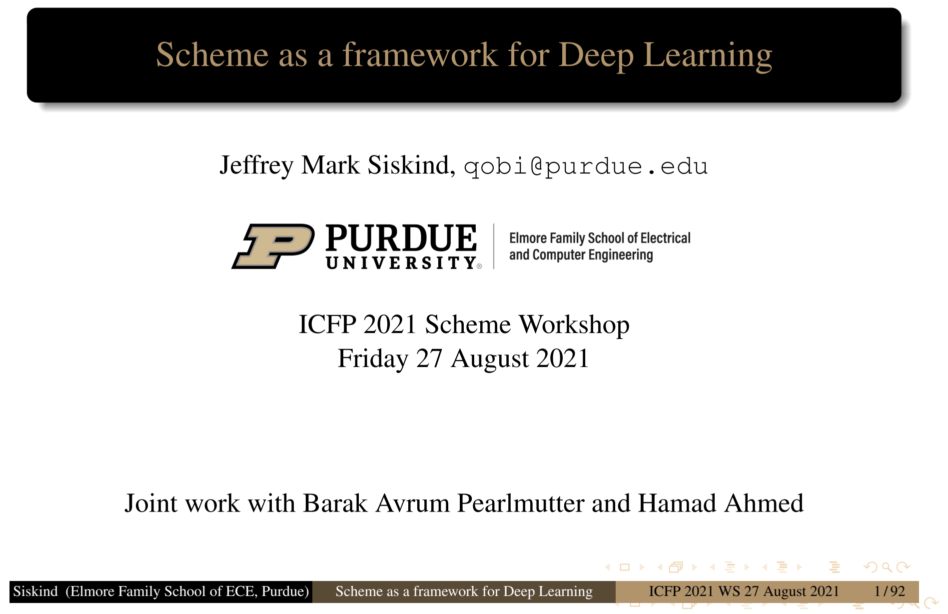


Divide-and-Conquer Checkpointing for Arbitrary Programs with No User Annotation

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

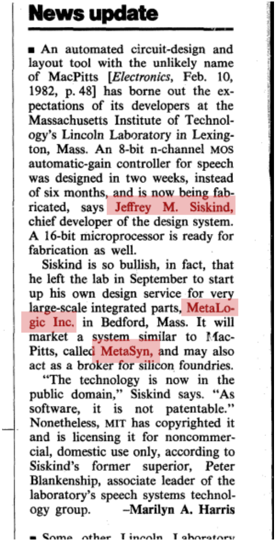
Jeffrey Mark Siskind, `qobi@purdue.edu`

PURDUE UNIVERSITY | Elmore Family School of Electrical and Computer Engineering

ICFP 2021 Scheme Workshop
Friday 27 August 2021

Joint work with Barak Avrum Pearlmutter and Hamad Ahmed

Electronics (13 January 1983, p. 32)



News update

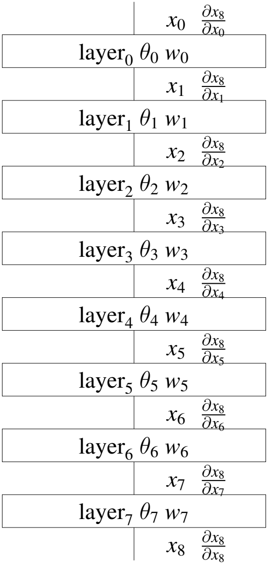
■ An automated circuit-design and layout tool with the unlikely name of MacPitts [*Electronics*, Feb. 10, 1982, p. 48] has borne out the expectations of its developers at the Massachusetts Institute of Technology's Lincoln Laboratory in Lexington, Mass. An 8-bit n-channel MOS automatic-gain controller 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**

■ Some other Lincoln Laboratory

A Neural Network



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

layer_0 θ_0 w_0

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

layer_1 θ_1 w_1

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

layer_2 θ_2 w_2

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

layer_3 θ_3 w_3

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

layer_4 θ_4 w_4

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

layer_5 θ_5 w_5

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

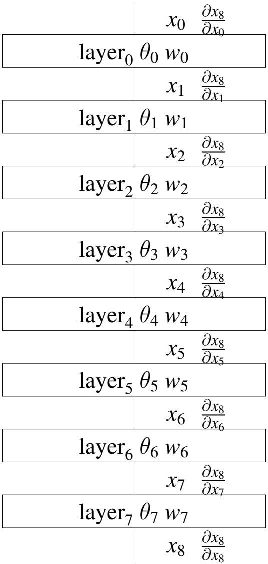
layer_6 θ_6 w_6

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

layer_7 θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

A Neural Network is a (Functional) Program



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

$$
\begin{array}{l}
\text{net}\ [\theta_0,\ldots,\theta_7]\ [w_0,\ldots,w_7]\ x_0 \triangleq\\
\quad \textbf{let}\ \begin{array}{rcl}
x_1 & = & \text{layer}_0\ \theta_0\ w_0\ x_0\\
x_2 & = & \text{layer}_1\ \theta_1\ w_1\ x_1\\
x_3 & = & \text{layer}_2\ \theta_2\ w_2\ x_2\\
x_4 & = & \text{layer}_3\ \theta_3\ w_3\ x_3\\
x_5 & = & \text{layer}_4\ \theta_4\ w_4\ x_4\\
x_6 & = & \text{layer}_5\ \theta_5\ w_5\ x_5\\
x_7 & = & \text{layer}_6\ \theta_6\ w_6\ x_6\\
x_8 & = & \text{layer}_7\ \theta_7\ w_7\ x_7
\end{array}\\
\quad \textbf{in}\ x_8
\end{array}
$$

A (Functional) Program

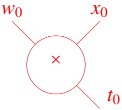
$$
\begin{array}{l}
f\ [w_0, w_1]\ [x_0, x_1] \triangleq \\
\quad \textbf{let} \begin{array}{rcl} t_0 & = & w_0 \times x_0 \\ t_1 & = & w_1 \times x_1 \\ y & = & t_0 + t_1 \end{array} \\
\quad \textbf{in}\ y
\end{array}
$$

A (Functional) Program is a (Neural) Network

$$
\begin{array}{l}
f\ [w_0, w_1]\ [x_0, x_1] \triangleq \\
\quad \textbf{let} \begin{array}{rcl} t_0 & = & w_0 \times x_0 \\ t_1 & = & w_1 \times x_1 \\ y & = & t_0 + t_1 \end{array} \\
\quad \textbf{in}\ y
\end{array}
$$

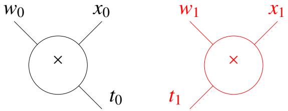
A (Functional) Program is a (Neural) Network

$$\begin{array}{l} f\ [w_0, w_1]\ [x_0, x_1] \triangleq \\ \quad \textbf{let}\ \begin{array}{rcl} t_0 & = & w_0 \times x_0 \\ t_1 & = & w_1 \times x_1 \\ y & = & t_0 + t_1 \end{array} \\ \quad \textbf{in}\ y \end{array}$$



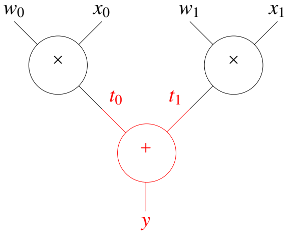
A (Functional) Program is a (Neural) Network

$$
\begin{array}{llcl}
\multicolumn{4}{l}{f\ [w_0, w_1]\ [x_0, x_1] \triangleq} \\
\textbf{let} & t_0 & = & w_0 \times x_0 \\
 & t_1 & = & w_1 \times x_1 \\
 & y & = & t_0 + t_1 \\
\multicolumn{4}{l}{\textbf{in}\ y}
\end{array}
$$



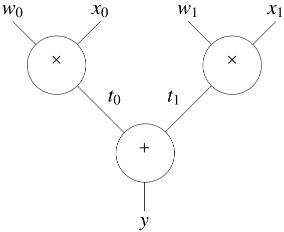
A (Functional) Program is a (Neural) Network

$$\begin{array}{l} f\ [w_0, w_1]\ [x_0, x_1] \triangleq \\ \quad \textbf{let}\ \begin{array}{rcl} t_0 & = & w_0 \times x_0 \\ t_1 & = & w_1 \times x_1 \\ y & = & t_0 + t_1 \end{array} \\ \quad \textbf{in}\ y \end{array}$$



A (Functional) Program is a (Neural) Network

$$
\begin{array}{l}
f\ [w_0, w_1]\ [x_0, x_1] \triangleq \\
\quad \textbf{let}\ \begin{array}{rcl} t_0 & = & w_0 \times x_0 \\ t_1 & = & w_1 \times x_1 \\ y & = & t_0 + t_1 \end{array} \\
\quad \textbf{in}\ y
\end{array}
$$



Some Observations

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).

Outline

1. Migrate reflective AD through partial evaluation

2. Implementing checkpointing reverse mode through CPS

Outline

1. Migrate reflective AD through partial evaluation

2. Implementing checkpointing reverse mode through CPS

Key Idea

Migrate reflective source-to-source transformation
from run time to compile time
with abstract interpretation

Traditional AD by Source-to-Source Transformation

Preprocessor at Compile Time

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

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

```
... derivative(f, 3) ...
```

Traditional AD by Source-to-Source Transformation

Preprocessor at Compile Time

```
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 ...
```

Traditional AD by Source-to-Source Transformation

Preprocessor at Compile Time

```
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 ...
```

Traditional AD by Source-to-Source Transformation

Preprocessor at Compile Time

```
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 ...
```

Source-to-Source Transformation at Run Time

Reflection

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

--

Source-to-Source Transformation at Run Time

Reflection

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

code(f)
```

--

Source-to-Source Transformation at Run Time

Reflection

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

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

--

Source-to-Source Transformation at Run Time

Reflection

```
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

```
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"
```

--

Source-to-Source Transformation at Run Time

Reflection

```
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")
```

--

Source-to-Source Transformation at Run Time

Reflection

```
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
```

--

Source-to-Source Transformation at Run Time

Reflection

```
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)
```

--

Source-to-Source Transformation at Run Time

Reflection

```
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

```
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?

```
while not converged() do
   x = x-eta⋆derivative(f, x)
end
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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

```
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

```
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

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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) ...
```

Abstract Interpretation aka (Polyvariant) Flow Analysis

```
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

```
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

```
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 = e1, y = e2}.x
```

A Single Powerful Optimization

$\texttt{\{x = e1, y = e2\}.x} \rightsquigarrow \texttt{e1}$

A Single Powerful Optimization

$$\texttt{\{x = e1, y = e2\}.x} \leadsto \texttt{e1}$$

- can eliminate storage allocation

A Single Powerful Optimization

$$\texttt{\{x = e1, y = e2\}.x} \rightsquigarrow \texttt{e1}$$

- can eliminate storage allocation
- can eliminate storage reclamation

A Single Powerful Optimization

$$\texttt{\{x = e1, y = e2\}.x} \rightsquigarrow \texttt{e1}$$

- can eliminate storage allocation
- can eliminate storage reclamation
- can eliminate storage writes

A Single Powerful Optimization

`{x = e1, y = e2}.x` ⇝ `e1`

- can eliminate storage allocation
- can eliminate storage reclamation
- can eliminate storage writes
- can eliminate storage reads

A Single Powerful Optimization

`{x = e1, y = e2}.x` $\leadsto$ `e1`

- can eliminate storage allocation
- can eliminate storage reclamation
- can eliminate storage writes
- can eliminate storage reads
- can eliminate dead code

The Kind of Code People Write in Dynamic Languages

```
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})))
```

--

The Kind of Code People Write in Dynamic Languages

```
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

--
```

Key Idea

You need this anyway
to compile dynamic languages efficiently

Key Idea

Same mechanism can support AD

Migrating Reflective AD from Run Time to Compile Time

```
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) ...
```

Migrating Reflective AD from Run Time to Compile Time

```
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) ...
```

Migrating Reflective AD from Run Time to Compile Time

```
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) ...
```

Migrating Reflective AD from Run Time to Compile Time

```
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) ...
```

Migrating Reflective AD from Run Time to Compile Time

```
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) ...
```

Migrating Reflective AD from Run Time to Compile Time

```
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) ...
```

Migrating Reflective AD from Run Time to Compile Time

```
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

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

$$\begin{aligned}
\overrightarrow{c} &\rightsquigarrow \overrightarrow{\mathcal{J}}\ c\\
\overrightarrow{\lambda x.e} &\rightsquigarrow \lambda\,\overrightarrow{x}\,.\,\overrightarrow{e}\\
\overrightarrow{e_1\ e_2} &\rightsquigarrow \overrightarrow{e_1}\ \overrightarrow{e_2}\\
\overrightarrow{\textbf{letrec}\ x_1 = e_1; \ldots; x_n = e_n\ \textbf{in}\ e} &\rightsquigarrow \textbf{letrec}\ \overrightarrow{x_1} = \overrightarrow{e_1}; \ldots; \overrightarrow{x_n} = \overrightarrow{e_n}\ \textbf{in}\ \overrightarrow{e}\\
\overrightarrow{e_1, e_2} &\rightsquigarrow \overrightarrow{e_1}\overrightarrow{,}\overrightarrow{e_2}
\end{aligned}$$

Essence of Reverse Transform

$$
\begin{aligned}
\overleftharpoonup{x = c} &\leadsto \overleftharpoonup{x} = \overleftarrow{\mathcal{J}}\, c\\
\overleftharpoonup{x_1 = x_2} &\leadsto \overleftharpoonup{x_1} = \overleftharpoonup{x_2}\\
\overleftharpoonup{x = \lambda x.e} &\leadsto \overleftharpoonup{x} = \overleftharpoonup{\lambda x.e}\\
\overleftharpoonup{x = x_1\; x_2} &\leadsto \overleftharpoonup{x}, \bar{x} = \overleftharpoonup{x_1}\; \overleftharpoonup{x_2}\\
\overleftharpoonup{x = x_1, x_2} &\leadsto \overleftharpoonup{x} = \overleftharpoonup{x_1} \overleftharpoonup{,} \overleftharpoonup{x_2}
\end{aligned}
$$

$$
\begin{aligned}
\overline{x_1 = x_2} &\leadsto \overleftharpoondown{x_2} \mathrel{+}= \overleftharpoondown{x_1}\\
\overline{x = \lambda x.e} &\leadsto \overleftharpoondown{\lambda x.e} \mathrel{+}= \overleftharpoonup{x}\\
\overline{x = x_1\; x_2} &\leadsto \overleftharpoondown{x_1} \overleftharpoondown{,} \overleftharpoondown{x_2} \mathrel{+}= \bar{x}\; \overleftharpoondown{x}\\
\overline{x = x_1, x_2} &\leadsto \overleftharpoondown{x_1} \overleftharpoondown{,} \overleftharpoondown{x_2} \mathrel{+}= \overleftharpoondown{x}
\end{aligned}
$$

$$
\overleftharpoonup{\lambda x.\mathbf{let}\; b_1;\ldots;b_n\,\mathbf{in}\; y} \leadsto \lambda \overleftharpoonup{x}.\mathbf{let}\; \overleftharpoonup{b_1};\ldots;\overleftharpoonup{b_n}\,\mathbf{in}\; \overleftharpoonup{y}, \lambda \overleftharpoondown{y}.\mathbf{let}\; \overline{b_n};\ldots;\overline{b_1}\,\mathbf{in}\; \overleftharpoondown{x}
$$

Game Theory

$$
\begin{array}{cc|ccccc}
 & & & & B & & \\
 & & b_1 & \dots & b_j & \dots & b_n \\
\hline
 & a_1 & & & & & \\
 & \vdots & & \ddots & \vdots & & \\
A & a_i & & \dots & \textsc{Payoff}(a_i, b_j) & \dots & \\
 & \vdots & & & \vdots & \ddots & \\
 & a_m & & & & &
\end{array}
$$

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ.

Game Theory

$$\begin{array}{cc|ccccc} & & & & B & & \\ & & b_1 & \dots & b_j & \dots & b_n \\ \hline & a_1 & & & & & \\ & \vdots & & \ddots & \vdots & & \\ A & a_i & & \dots & \textsc{Payoff}(a_i, b_j) & \dots & \\ & \vdots & & & \vdots & \ddots & \\ & a_m & & & & & \end{array}$$

$$\max_{a \in A} \min_{b \in B} \textsc{Payoff}(a, b)$$

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ.

Game Theory

$$\begin{array}{cc|ccc}
 & & & \mathbb{R}^n & \\
 & & \dots & \mathbf{b} & \dots \\
\hline
 & \vdots & \ddots & \vdots & \\
\mathbb{R}^m & \mathbf{a} & \dots & \textsc{Payoff}(\mathbf{a},\mathbf{b}) & \dots \\
 & \vdots & & \vdots & \ddots
\end{array}$$

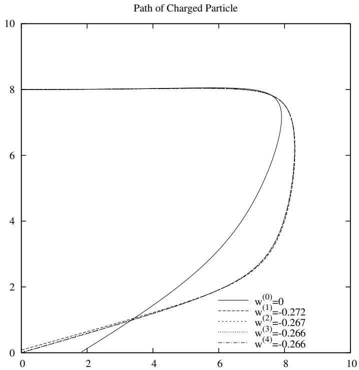
$$\max_{\mathbf{a}\in\mathbb{R}^m}\min_{\mathbf{b}\in\mathbb{R}^n}\textsc{Payoff}(\mathbf{a},\mathbf{b})$$

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ.

Code

```
(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)))))
```

Cathode Ray Tubes



$$\begin{aligned}
\text{potential: } p(\mathbf{x}; w) &= \|\mathbf{x} - (10, 10 - w)\|^{-1} + \|\mathbf{x} - (10, 0)\|^{-1} \\
\ddot{\mathbf{x}}(t) &= -\nabla_{\mathbf{x}}\, p(\mathbf{x})|_{\mathbf{x}=\mathbf{x}(t)} \\
\dot{\mathbf{x}}(t + \Delta t) &= \dot{\mathbf{x}}(t) + \Delta t\, \ddot{\mathbf{x}}(t) \\
\mathbf{x}(t + \Delta t) &= \mathbf{x}(t) + \Delta t\, \dot{\mathbf{x}}(t) \\
\text{When: } x_1(t + \Delta t) &\leq 0 \\
\text{let: } \Delta t_f &= -x_1(t)/\dot{x}_1(t) \\
t_f &= t + \Delta t_f \\
\mathbf{x}(t_f) &= \mathbf{x}(t) + \Delta t_f \dot{\mathbf{x}}(t) \\
\text{Error: } E(w) &= x_0(t_f)^2 \\
\text{Find: } & \operatorname*{argmin}_{w} E(w)
\end{aligned}$$

Sprague, C. S. and George, R. H. (1939). *Cathode Ray Deflecting Electrode*. US Patent 2,161,437.

George, R. H. (1940). *Cathode Ray Tube*. US Patent 2,222,942.

Code

```
(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* ((xddot (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 (v+ 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)))
```

Probabilistic Lambda Calculus

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$$\begin{array}{ll} \Pr(x_0 \mapsto \textbf{true}) = p_0 & \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0 \\ \Pr(x_1 \mapsto \textbf{true}) = p_1 & \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1 \end{array}$$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$

$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$

$$\Pr(\mathcal{E}(P) = 0|p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1|p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2|p_0, p_1) = (1 - p_0)(1 - p_1)$$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P = \textbf{if}\ x_0\ \textbf{then}\ 0\ \textbf{else}\ \textbf{if}\ x_1\ \textbf{then}\ 1\ \textbf{else}\ 2$

$$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$$
$$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$$

$$\Pr(\mathcal{E}(P) = 0|p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1|p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2|p_0, p_1) = (1 - p_0)(1 - p_1)$$

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

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$$
$$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$$

$$\Pr(\mathcal{E}(P) = 0|p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1|p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2|p_0, p_1) = (1 - p_0)(1 - p_1)$$

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

$$\operatorname*{argmax}_{p_0, p_1} \prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v|p_0, p_1) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle$$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Prolog

```
p(0).
p(X):-q(X).
q(1).
q(2).
```

Probabilistic Prolog

$\mathrm{Pr}(\texttt{p(0).}) = p_0$
$\mathrm{Pr}(\texttt{p(X):-q(X).}) = 1 - p_0$
$\mathrm{Pr}(\texttt{q(1).}) = p_1$
$\mathrm{Pr}(\texttt{q(2).}) = 1 - p_1$

Probabilistic Prolog

$$
\begin{aligned}
&\Pr(\texttt{p(0).}) = p_0\\
&\Pr(\texttt{p(X):-q(X).}) = 1 - p_0\\
&\Pr(\texttt{q(1).}) = p_1\\
&\Pr(\texttt{q(2).}) = 1 - p_1
\end{aligned}
$$

$$
\begin{aligned}
&\Pr(\texttt{?-p(0).}) = p_0\\
&\Pr(\texttt{?-p(1).}) = (1 - p_0)p_1\\
&\Pr(\texttt{?-p(2).}) = (1 - p_0)(1 - p_1)
\end{aligned}
$$

Probabilistic Prolog

$$\Pr(\texttt{p(0).}) = p_0$$
$$\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$$
$$\Pr(\texttt{q(1).}) = p_1$$
$$\Pr(\texttt{q(2).}) = 1 - p_1$$

$$\Pr(\texttt{?-p(0).}) = p_0$$
$$\Pr(\texttt{?-p(1).}) = (1 - p_0)p_1$$
$$\Pr(\texttt{?-p(2).}) = (1 - p_0)(1 - p_1)$$

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

Probabilistic Prolog

$$\begin{aligned}
&\Pr(\texttt{p(0).}) = p_0\\
&\Pr(\texttt{p(X):-q(X).}) = 1 - p_0\\
&\Pr(\texttt{q(1).}) = p_1\\
&\Pr(\texttt{q(2).}) = 1 - p_1
\end{aligned}$$

$$\begin{aligned}
&\Pr(\texttt{?-p(0).}) = p_0\\
&\Pr(\texttt{?-p(1).}) = (1 - p_0)p_1\\
&\Pr(\texttt{?-p(2).}) = (1 - p_0)(1 - p_1)
\end{aligned}$$

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

$$\operatorname*{argmax}_{p_0,p_1} \prod_{q\in\{\texttt{p(0)},\texttt{p(1)},\texttt{p(2)},\texttt{p(2)}\}} \Pr(\texttt{?-}q\texttt{.}) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle$$

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '($0$ $1$ $2$ $2$)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '($0$ $1$ $2$ $2$))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then 0 else if $x_1$ then 1 else 2
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...)))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...)))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...)))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if $x_0$ then $0$ else if $x_1$ then $1$ else $2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate $\textbf{if}\ x_0\ \textbf{then}\ 0\ \textbf{else}\ \textbf{if}\ x_1\ \textbf{then}\ 1\ \textbf{else}\ 2$
                   (list $\Pr(x_0 \mapsto \textbf{true}) = p_0$   $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
                         $\Pr(x_1 \mapsto \textbf{true}) = p_1$   $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list $\Pr(\texttt{p(0).}) = p_0$
                       $\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
                       $\Pr(\texttt{q(1).}) = p_1$
                       $\Pr(\texttt{q(2).}) = 1 - p_1$)))
   (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)
```

Generated Code

```
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;}
```

Benchmarks

		backprop		
		Fs	Fv	R
VLAD	STALIN∇	1.00	🟩	1.00
FORTRAN	ADIFOR	15.51	3.35	🟦
	TAPENADE	14.97	5.97	6.86
C	ADIC	22.75	5.61	🟦
C++	ADOL-C	12.16	5.79	32.77
	CPPAD	54.74	🟦	29.24
	FADBAD++	132.31	46.01	60.71
ML	MLTON	95.20	🟩	39.90
	OCAML	202.01	🟩	156.93
	SML/NJ	181.93	🟩	102.89
HASKELL	GHC	🟩	🟩	🟩
SCHEME	BIGLOO	743.26	🟩	360.07
	CHICKEN	1626.73	🟩	1125.24
	GAMBIT	671.54	🟩	379.63
	IKARUS	279.59	🟩	165.16
	LARCENY	1203.34	🟩	511.54
	MIT SCHEME	2446.33	🟩	1113.09
	MZC	1318.60	🟩	754.47
	MZSCHEME	1364.14	🟩	772.10
	SCHEME->C	597.67	🟩	280.93
	SCMUTILS	5889.26	🟦	🟦
	STALIN	435.82	🟩	281.27

Damned Benchmarks

		`particle`				`saddle`			
		`FF`	`FR`	`RF`	`RR`	`FF`	`FR`	`RF`	`RR`
VLAD	Stalin∇	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Fortran	ADIFOR	2.05	🟦	🟦	🟦	5.44	🟦	🟦	🟦
	Tapenade	5.51	🟥	🟩	🟥	8.09	🟥	🟩	🟥
C	ADIC	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
C++	ADOL-C	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	CppAD	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	FADBAD++	93.32	🟩	🟩	🟩	60.67	🟩	🟩	🟩
ML	MLton	78.13	111.27	45.95	32.57	114.07	146.28	12.27	10.58
	OCaml	217.03	415.64	352.06	261.38	291.26	407.67	42.39	50.21
	SML/NJ	153.01	226.84	270.63	192.13	271.84	299.76	25.66	23.89
Haskell	GHC	209.44	🟩	🟩	🟩	247.57	🟩	🟩	🟩
Scheme	Bigloo	627.78	855.70	275.63	187.39	1004.85	1076.73	105.24	89.23
	Chicken	1453.06	2501.07	821.37	1360.00	2276.69	2964.02	225.73	252.87
	Gambit	578.94	879.39	356.47	260.98	958.73	1112.70	89.99	89.23
	Ikarus	266.54	386.21	158.63	116.85	424.75	527.57	41.27	42.34
	Larceny	964.18	1308.68	360.68	272.96	1565.53	1508.39	126.44	112.82
	MIT Scheme	2025.23	3074.30	790.99	609.63	3501.21	3896.88	315.17	295.67
	MzC	1243.08	1944.00	740.31	557.45	2135.92	2434.05	194.49	187.53
	MzScheme	1309.82	1926.77	712.97	555.28	2371.35	2690.64	224.61	219.29
	Scheme->C	582.20	743.00	270.83	208.38	910.19	913.66	82.93	69.87
	SCMUTILS	4462.83	🟦	🟦	🟦	7651.69	🟦	🟦	🟦
	Stalin	364.08	547.73	399.39	295.00	543.68	690.64	63.96	52.93

Statistics

		probabilistic-lambda-calculus		probabilistic-prolog	
		F	R	F	R
VLAD	STALIN∇	1.00	1.00	1.00	1.00
FORTRAN	ADIFOR	🟩	🟦	🟩	🟦
	TAPENADE	🟩	🟩	🟩	🟩
C	ADIC	🟩	🟦	🟩	🟦
C++	ADOL-C	🟩	🟩	🟩	🟩
	CPPAD	🟩	🟩	🟩	🟩
	FADBAD++	🟩	🟩	🟩	🟩
ML	MLTON	129.11	114.88	848.45	507.21
	OCAML	249.40	499.43	1260.83	1542.47
	SML/NJ	234.62	258.53	2505.59	1501.17
HASKELL	GHC	🟩	🟩	🟩	🟩
SCHEME	BIGLOO	983.12	1016.50	12832.92	7918.21
	CHICKEN	2324.54	3040.44	44891.04	24634.44
	GAMBIT	1033.46	1107.26	26077.48	14262.70
	IKARUS	497.48	517.89	8474.57	4845.10
	LARCENY	1658.27	1606.44	25411.62	14386.61
	MIT SCHEME	4130.88	3817.57	87772.39	49814.12
	MZC	2294.93	2346.13	57472.76	31784.38
	MZSCHEME	2721.35	2625.21	60269.37	33135.06
	SCHEME->C	811.37	803.22	10605.32	5935.56
	SCMUTILS	7699.14	🟦	83656.17	🟦
	STALIN	956.47	1994.44	15048.42	16939.28

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

Outline

1. Migrate reflective AD through partial evaluation

2. Implementing checkpointing reverse mode through CPS

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

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323:533–536, 1986.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323:533–536, 1986.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323:533–536, 1986.

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

P.J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, 1974.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323:533–536, 1986.

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

P.J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, 1974.

G.M. Ostrovskii, Y.M. Volin, and W.W. Borisov, *Über die Berechnung von Ableitungen, Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg*, 13(4):382–384, 1971.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323:533–536, 1986.

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

P.J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, 1974.

G.M. Ostrovskii, Y.M. Volin, and W.W. Borisov, *Über die Berechnung von Ableitungen, Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg*, 13(4):382–384, 1971.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323:533–536, 1986.

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

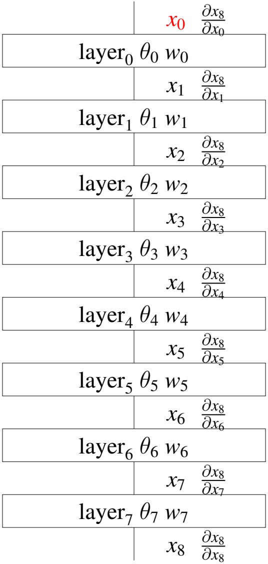
P.J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, 1974.

G.M. Ostrovskii, Y.M. Volin, and W.W. Borisov, *Über die Berechnung von Ableitungen, Wissenschaftliche Zeitschrift der Technischen Hochschule für Chemie, Leuna-Merseburg*, 13(4):382–384, 1971.

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.

Evaluating a Neural Network



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

layer_0 θ_0 w_0

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

layer_1 θ_1 w_1

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

layer_2 θ_2 w_2

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

layer_3 θ_3 w_3

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

layer_4 θ_4 w_4

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

layer_5 θ_5 w_5

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

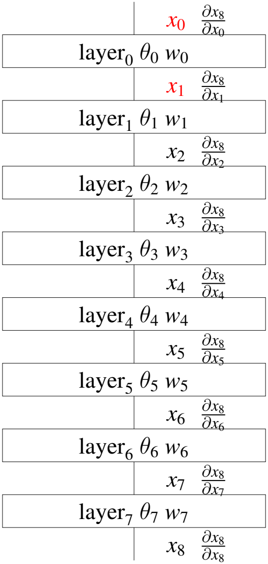
layer_6 θ_6 w_6

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

layer_7 θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

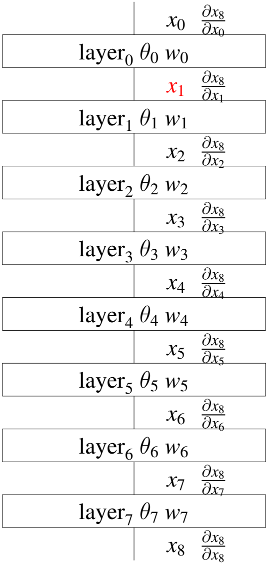
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

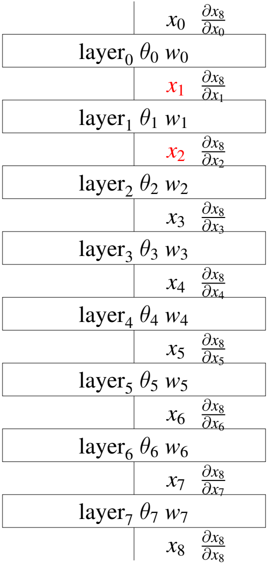
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

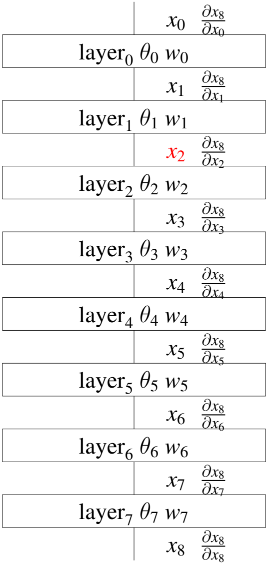
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

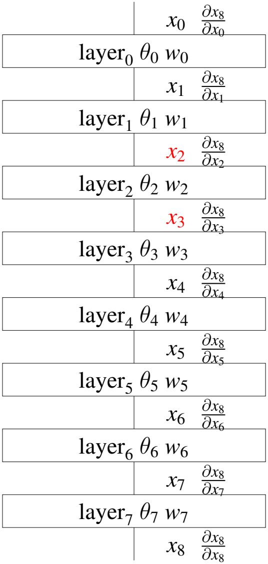
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

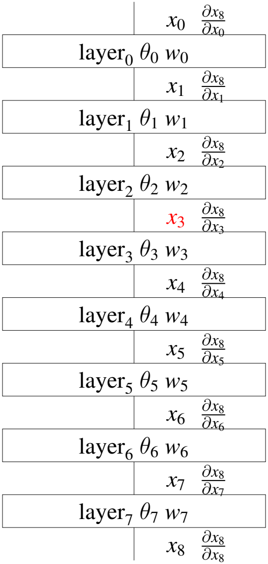
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

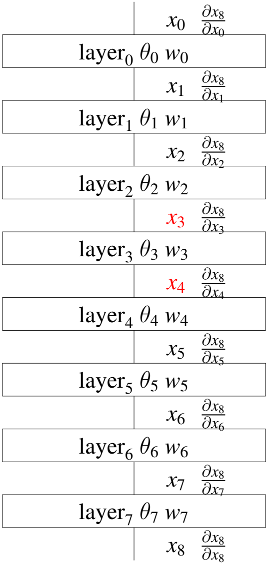
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer_0 θ_0 w_0

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

layer_1 θ_1 w_1

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

layer_2 θ_2 w_2

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

layer_3 θ_3 w_3

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

layer_4 θ_4 w_4

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

layer_5 θ_5 w_5

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

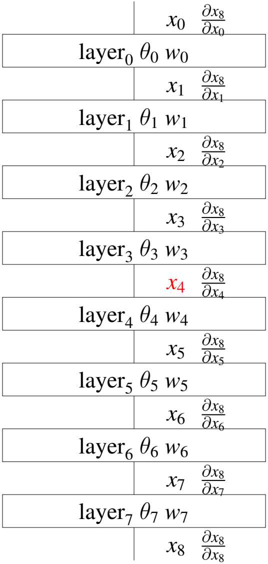
layer_6 θ_6 w_6

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

layer_7 θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

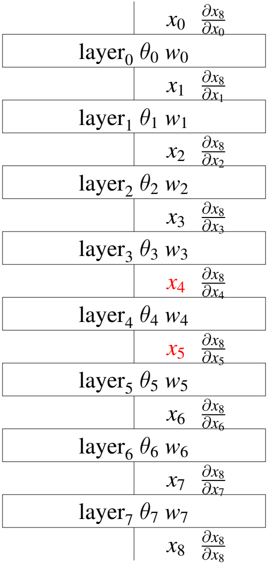
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

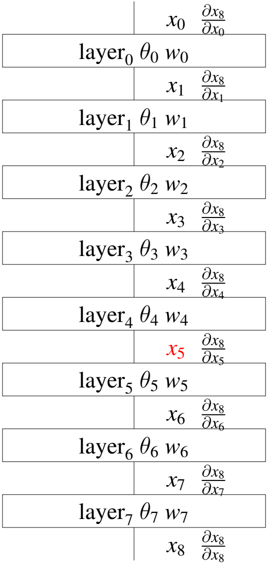
$\text{layer}_6\ \theta_6\ w_6$

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

$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

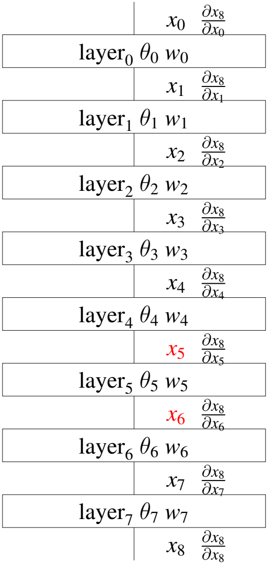
$\text{layer}_6\ \theta_6\ w_6$

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

$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

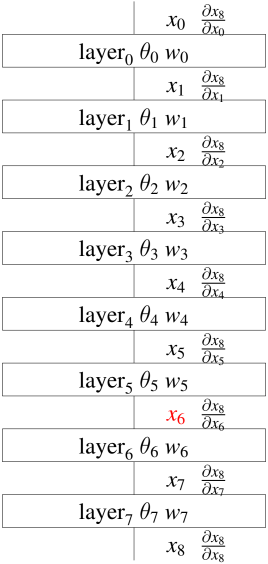
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

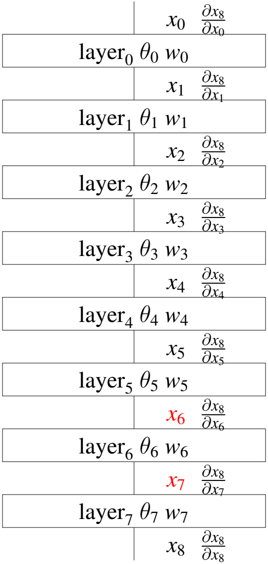
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

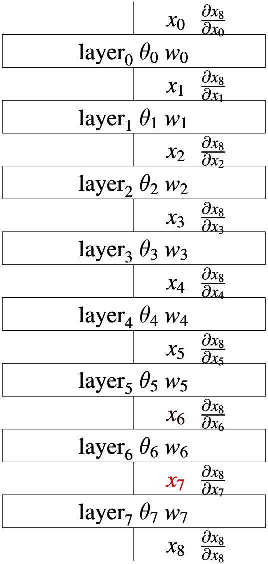
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

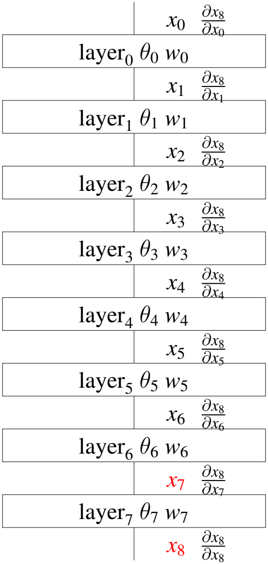
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

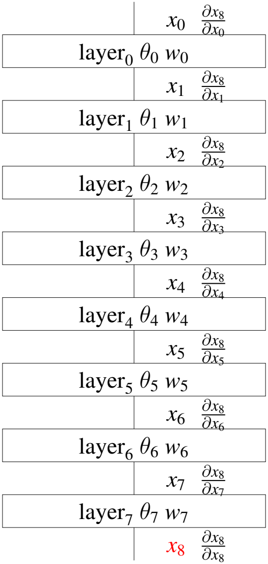
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

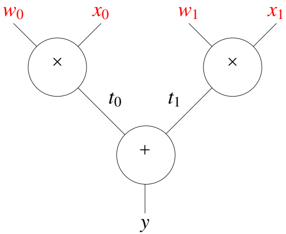
layer$_6$ θ_6 w_6

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

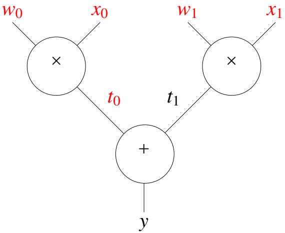
layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

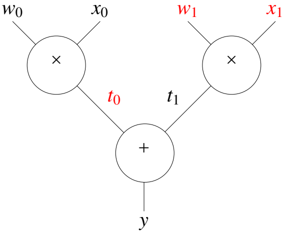
Evaluating a Network



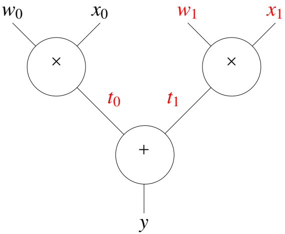
Evaluating a Network



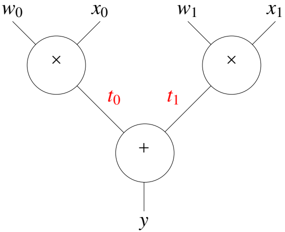
Evaluating a Network



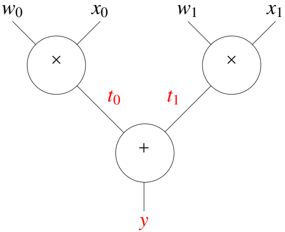
Evaluating a Network



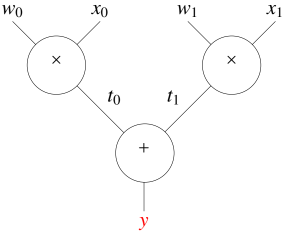
Evaluating a Network



Evaluating a Network



Evaluating a Network



Some Observations

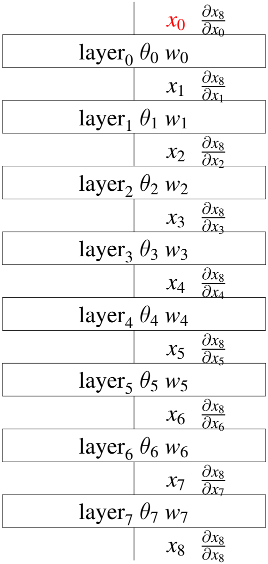
Some Observations

- 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



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

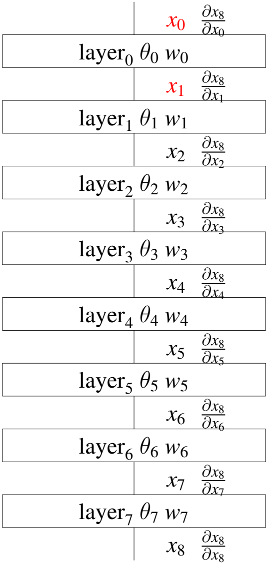
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

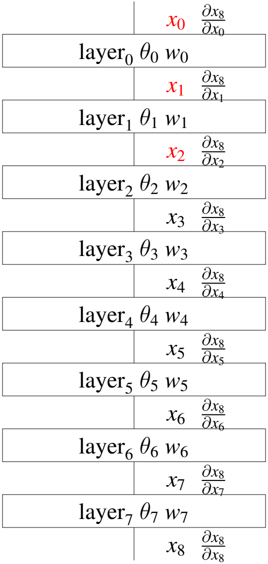
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

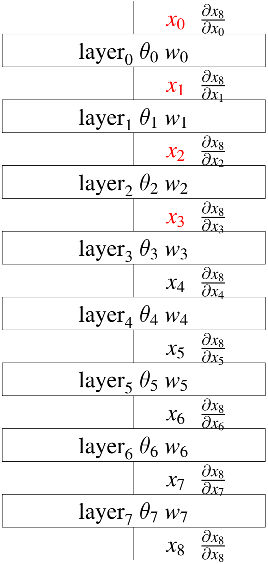
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

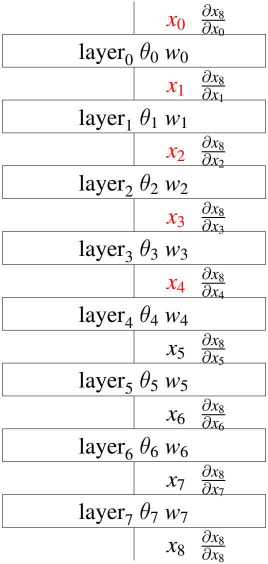
$\text{layer}_6\ \theta_6\ w_6$

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

$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

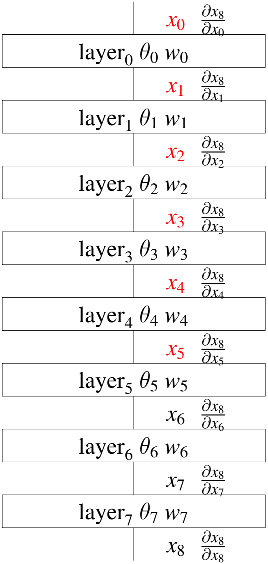
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

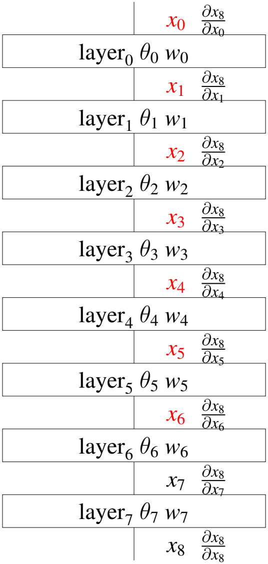
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

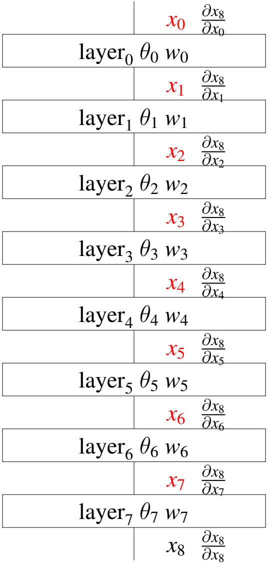
$\text{layer}_6\ \theta_6\ w_6$

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

$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

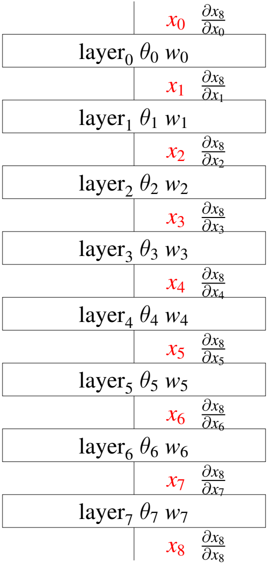
$\text{layer}_6\ \theta_6\ w_6$

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

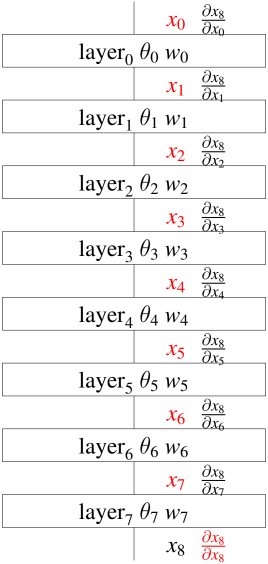
$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



Evaluating a Neural Network to Allow Subsequent Backpropagation



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

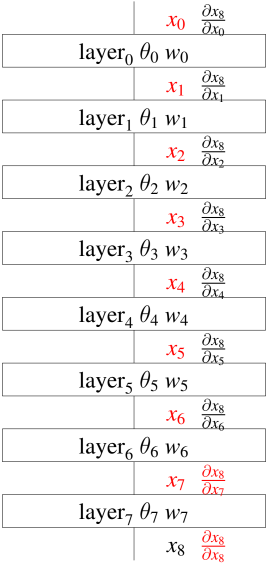
$\text{layer}_6\ \theta_6\ w_6$

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

$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

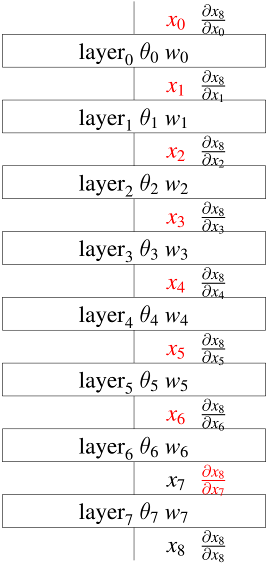
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

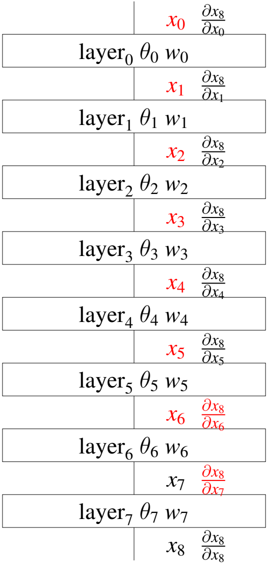
$\text{layer}_6\ \theta_6\ w_6$

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

$\text{layer}_7\ \theta_7\ w_7$

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

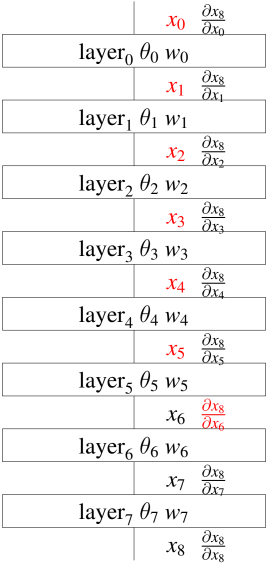
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

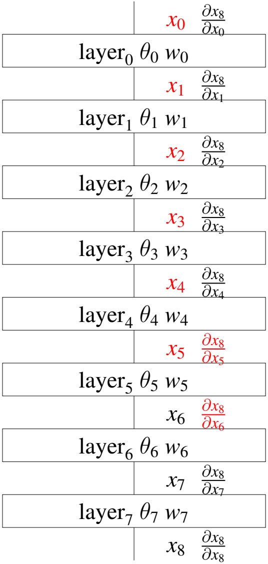
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

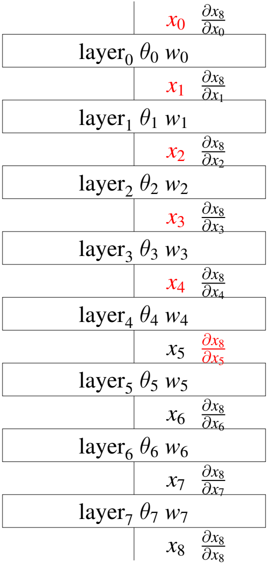
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

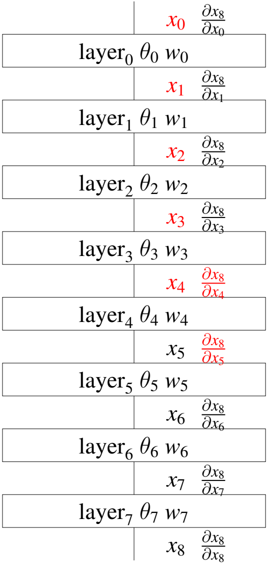
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

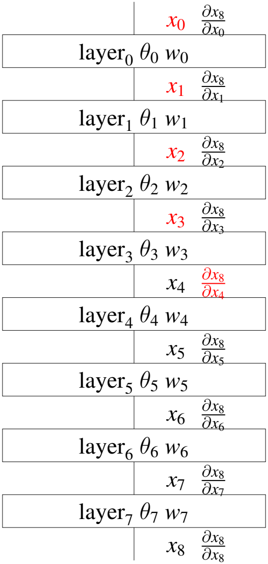
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

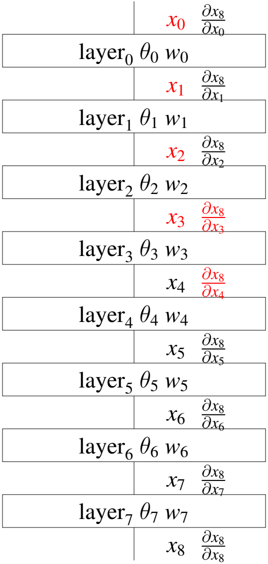
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

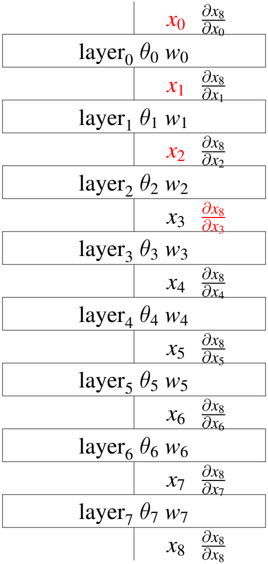
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

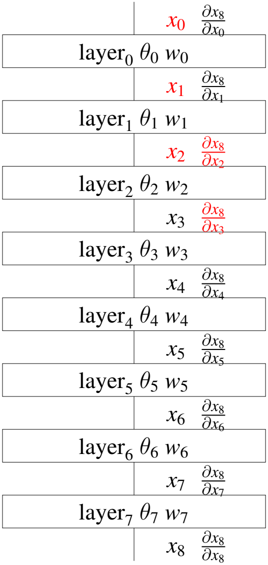
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

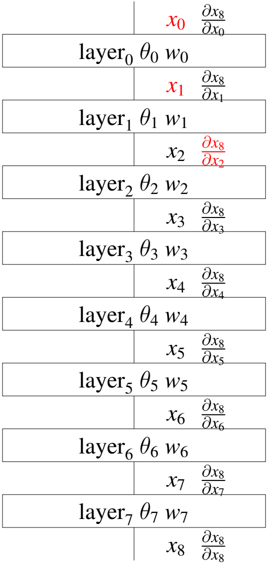
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

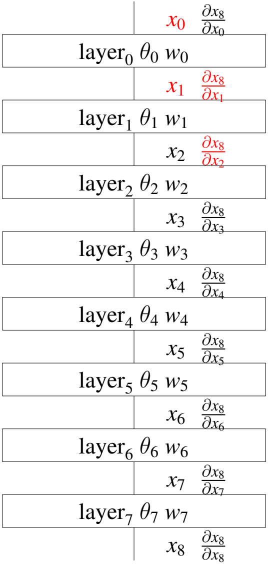
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

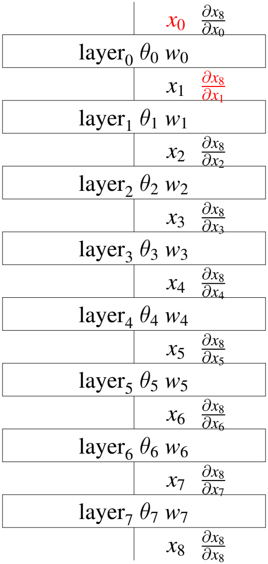
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

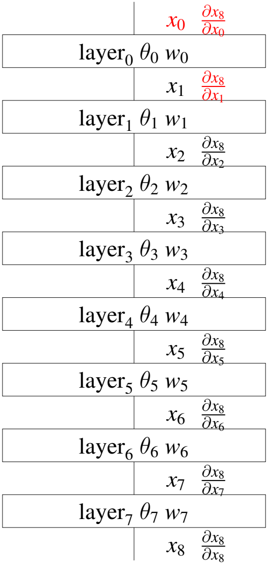
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

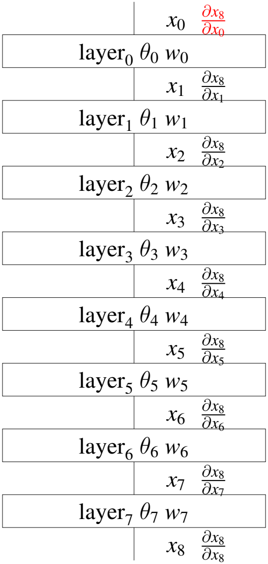
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Evaluating a Neural Network to Allow Subsequent Backpropagation



x_0 $\color{red}{\frac{\partial x_8}{\partial x_0}}$

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Some Observations

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

Complexity of Reverse-Mode AD

- 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 $O(w)$

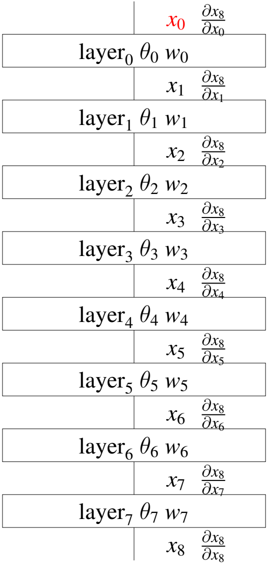
Complexity of Reverse-Mode AD

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

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.

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

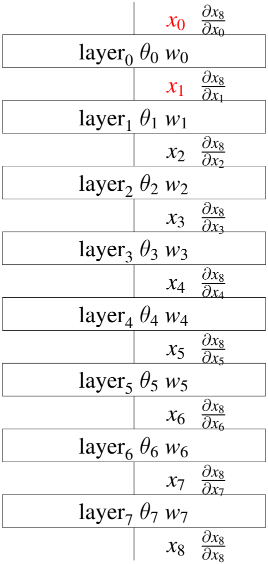
layer$_6$ θ_6 w_6

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

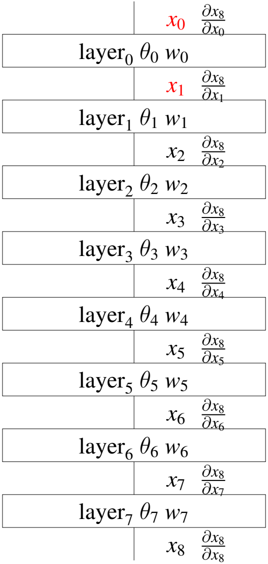
layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

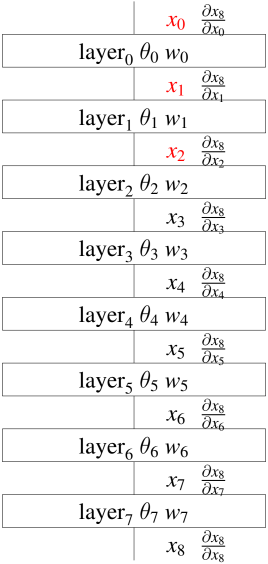
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

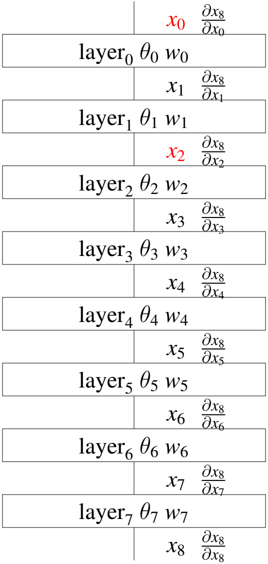
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

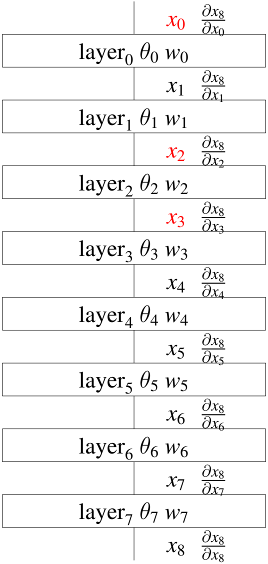
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

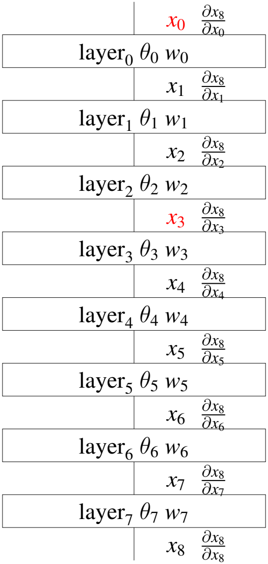
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

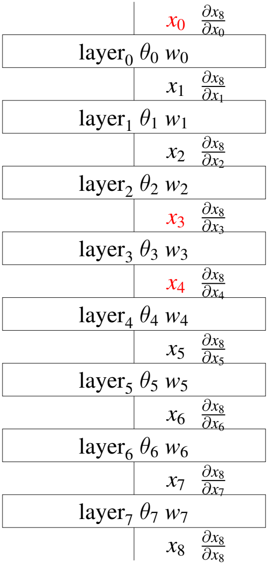
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

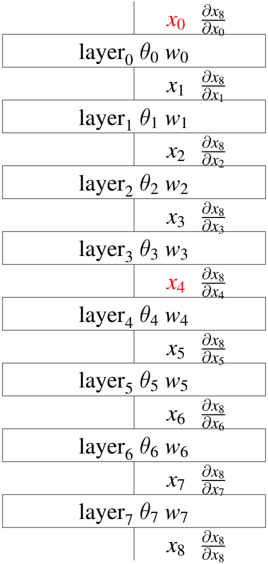
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

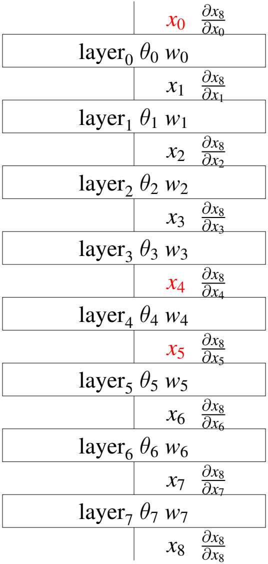
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

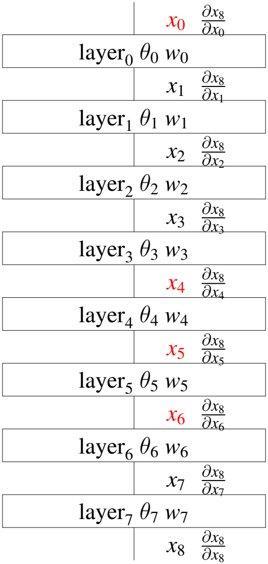
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

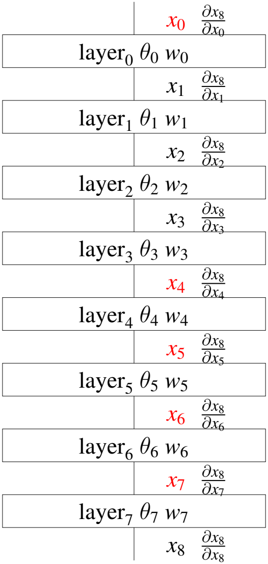
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

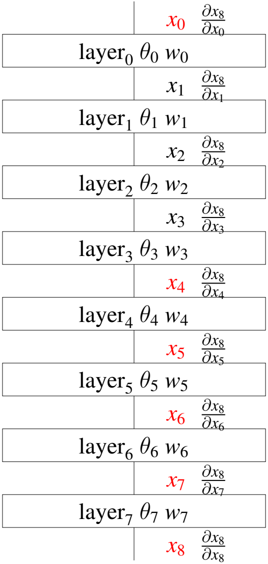
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer_0 θ_0 w_0

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

layer_1 θ_1 w_1

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

layer_2 θ_2 w_2

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

layer_3 θ_3 w_3

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

layer_4 θ_4 w_4

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

layer_5 θ_5 w_5

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

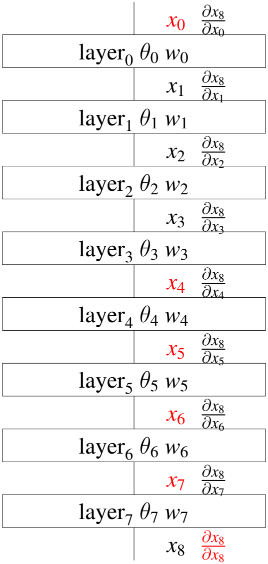
layer_6 θ_6 w_6

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

layer_7 θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

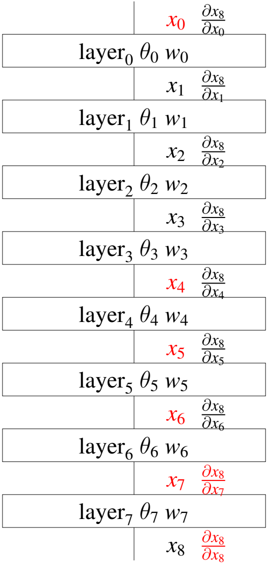
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

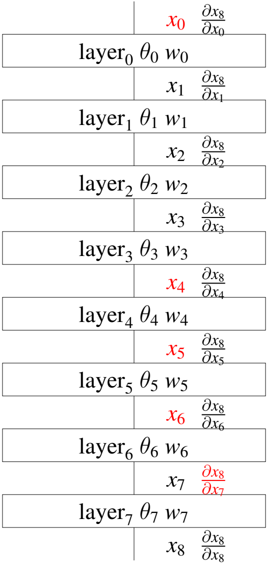
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

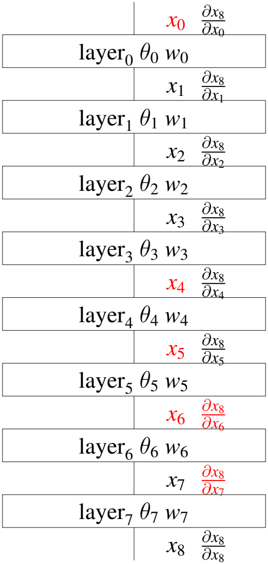
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

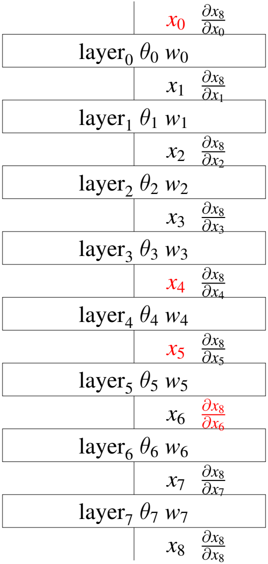
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

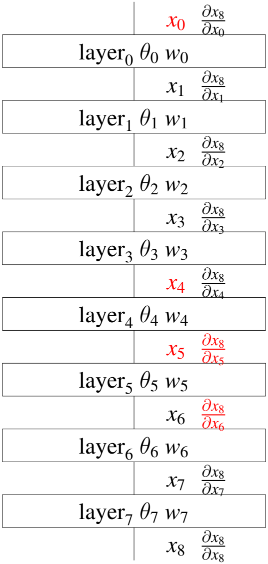
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

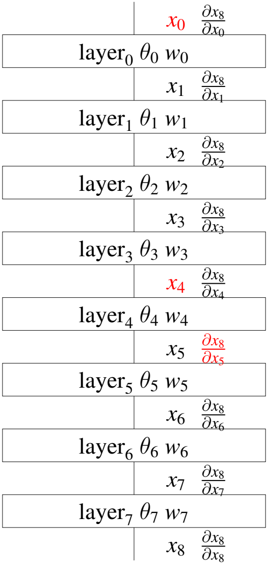
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

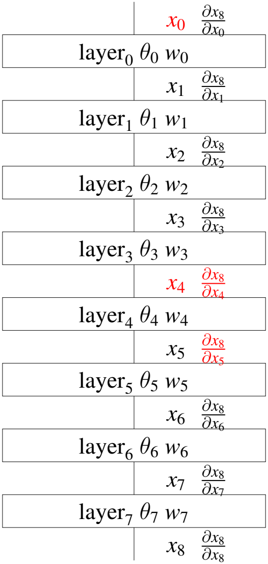
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

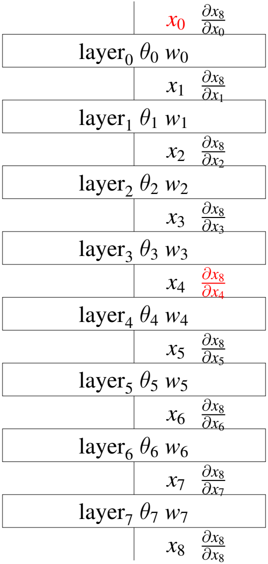
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

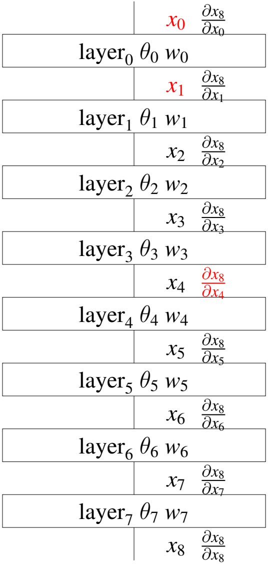
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

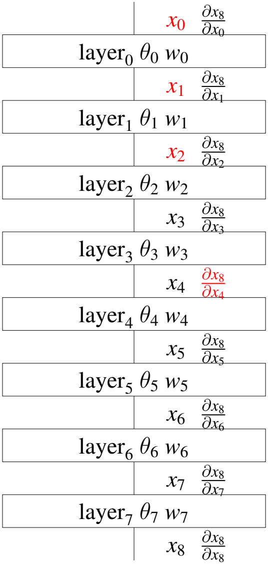
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

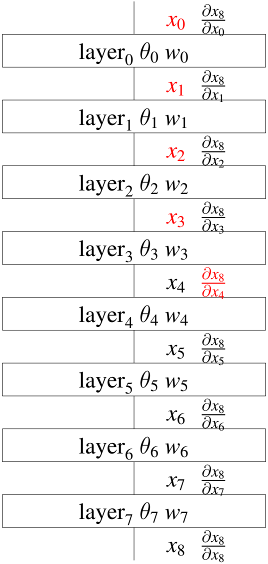
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

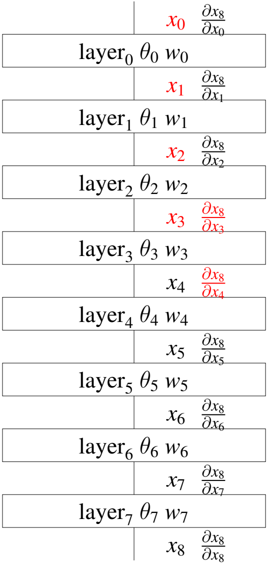
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

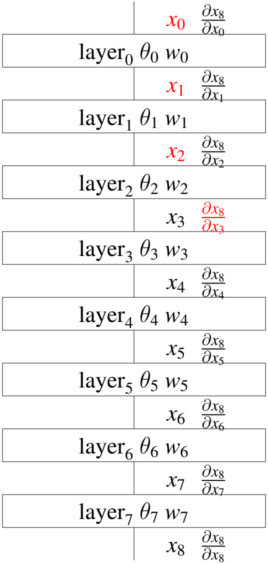
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

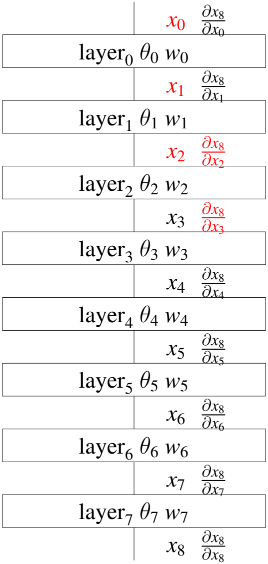
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

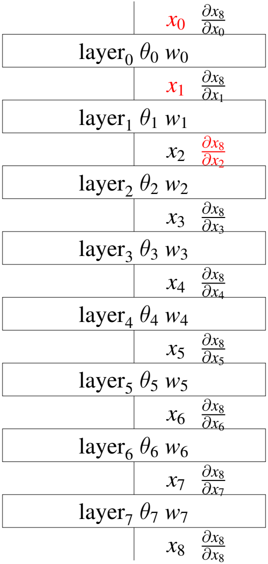
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

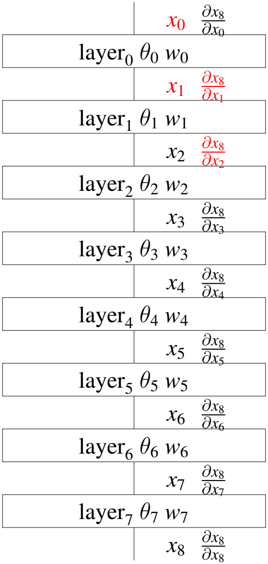
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

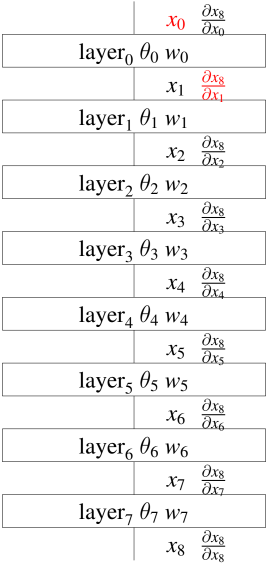
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

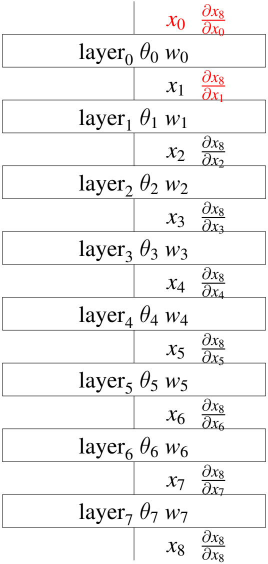
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

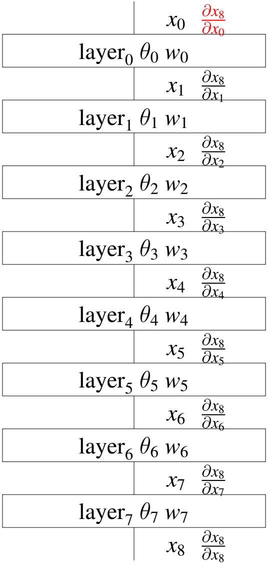
layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Backpropagation in a Neural Network with Checkpointing



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

layer$_0$ θ_0 w_0

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

layer$_1$ θ_1 w_1

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

layer$_2$ θ_2 w_2

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

layer$_3$ θ_3 w_3

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

layer$_4$ θ_4 w_4

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

layer$_5$ θ_5 w_5

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

layer$_6$ θ_6 w_6

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

layer$_7$ θ_7 w_7

x_8 $\frac{\partial x_8}{\partial x_8}$

Checkpointing

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

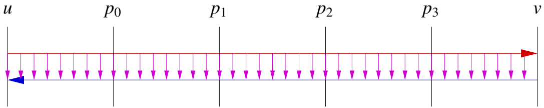
- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice.
  Once without saving intermediate variables.
  Once with saving intermediate variables.
- Backpropagation done in stages.
  Interleaved with (re)running forward pass.
  Only need saved intermediate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

Checkpointing

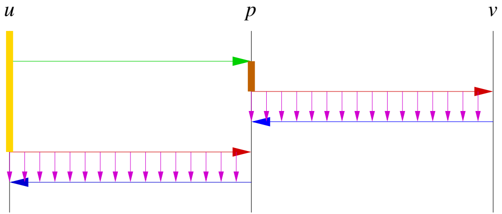
- Trades off extra running time for reduction in space.
- Forward pass of first half performed twice.
  Once without saving intermediate variables.
  Once with saving intermediate variables.
- Backpropagation done in stages.
  Interleaved with (re)running forward pass.
  Only need saved intermediate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

Divide-and-Conquer Checkpointing

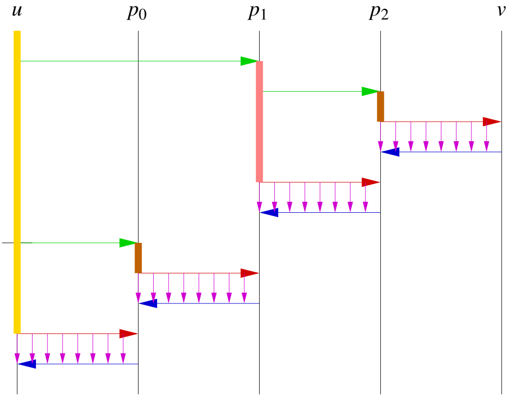
Divide-and-Conquer Checkpointing



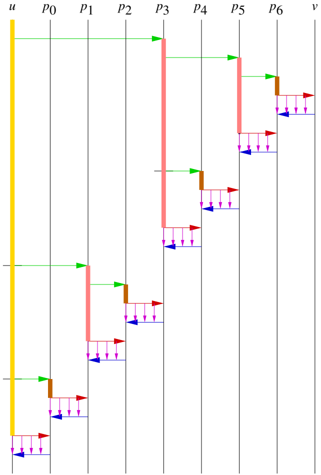
Divide-and-Conquer Checkpointing



Divide-and-Conquer Checkpointing



Divide-and-Conquer Checkpointing



Complexity of Divide-and-Conquer Checkpointing

Complexity of Divide-and-Conquer Checkpointing

- If running time of primal is $O(t)$

Complexity of Divide-and-Conquer Checkpointing

- 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

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 and $O(t \log t)$ time.

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

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

T. Chen, B. Xu, Z. Zhang, and C. Guestrin, *Training deep nets with sublinear memory cost*, arXiv 1604.06174, 2016.

A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, *Memory-Efficient Backpropagation Through Time*, NIPS, 2016.

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

T. Chen, B. Xu, Z. Zhang, and C. Guestrin, *Training deep nets with sublinear memory cost*, arXiv 1604.06174, 2016.

A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, *Memory-Efficient Backpropagation Through Time*, NIPS, 2016.

A. Griewank, *Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation*, Optimization Methods and Software, 1:35-54, 1992.

Implemented for DO Loops

L. Hascoët and V. Pascual, *TAPENADE 2.1 User's Guide*, Rapport technique 300, INRIA, 2004.

Implemented for DO Loops

L. Hascoët and V. Pascual, *TAPENADE 2.1 User's Guide*, Rapport technique 300, INRIA, 2004.

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

Implemented for DO Loops

L. Hascoët and V. Pascual, *TAPENADE 2.1 User's Guide*, Rapport technique 300, INRIA, 2004.

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

$\rightsquigarrow$

```
c$ad binomial-ckp n+1 30 1
      do 10 i=1, n
         ...
10    continue
```

Implemented for DO Loops

L. Hascoët and V. Pascual, *TAPENADE 2.1 User's Guide*, Rapport technique 300, INRIA, 2004.

$$\left.\begin{array}{ll} & \texttt{do 10 i=1, n} \\ & \quad\texttt{...} \\ \texttt{10} & \texttt{continue} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{ll} \multicolumn{2}{l}{\texttt{c\$ad binomial-ckp n+1 30 1}} \\ & \texttt{do 10 i=1, n} \\ & \quad\texttt{...} \\ \texttt{10} & \texttt{continue} \end{array}\right.$$

`https://www-sop.inria.fr/tropics/tapenade/faq.html`

> *Assuming that the final number of iterations N is known, and assuming that each iteration has the same runtime cost,*

Desiderata

Desiderata

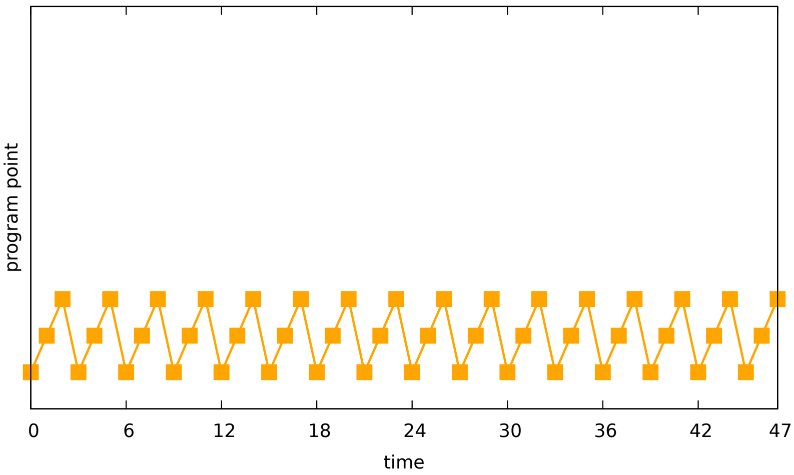
A (deep) neural network has no loops (except inside primitives).

Desiderata

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

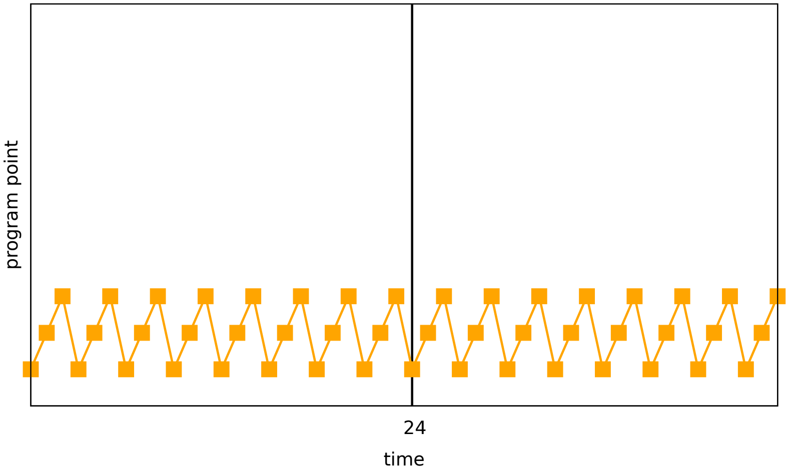
Want to implement for arbitrary code (not just a single DO loop).

Execution Trace of Loop



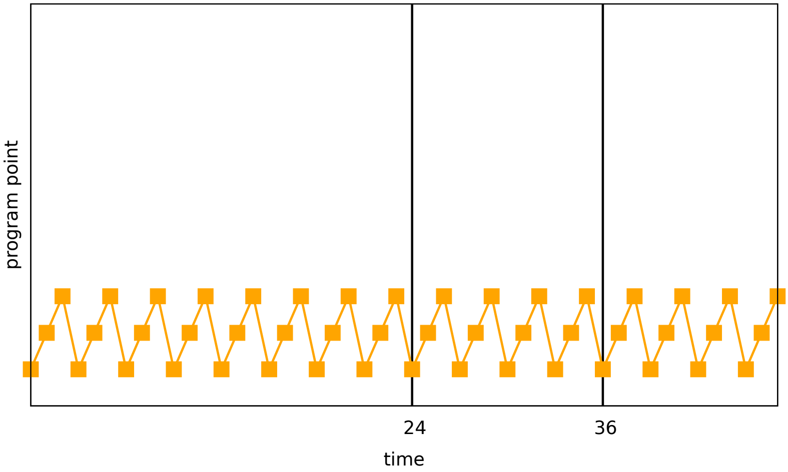
Execution Trace of Loop

Easy to make regular and uniform checkpoints



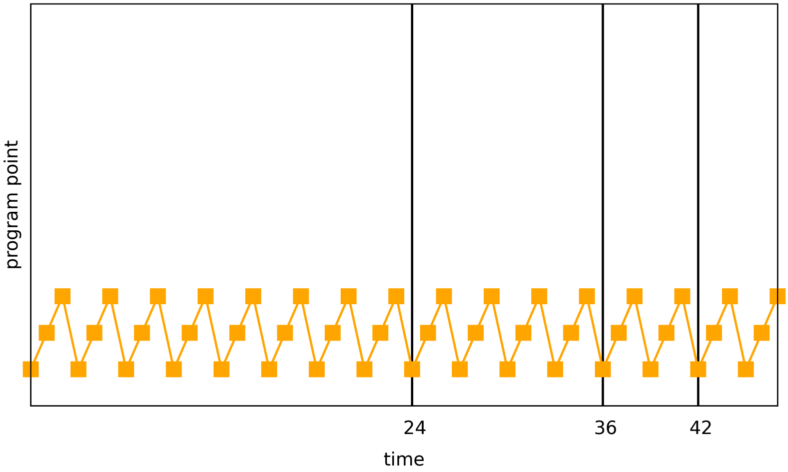
Execution Trace of Loop

Easy to make regular and uniform checkpoints



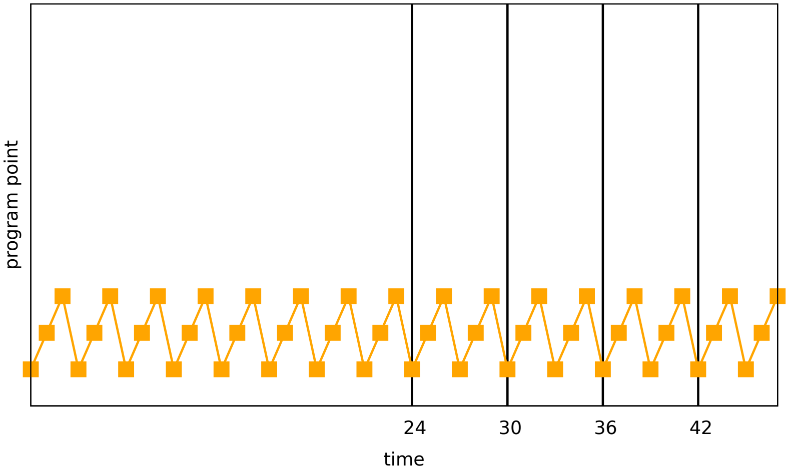
Execution Trace of Loop

Easy to make regular and uniform checkpoints



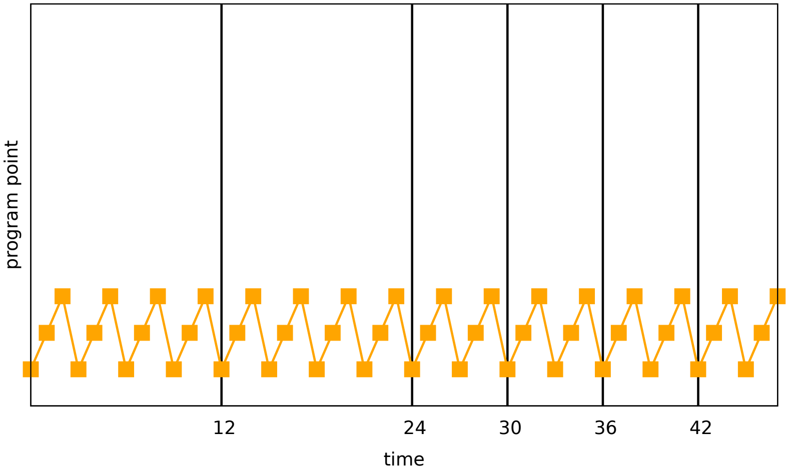
Execution Trace of Loop

Easy to make regular and uniform checkpoints



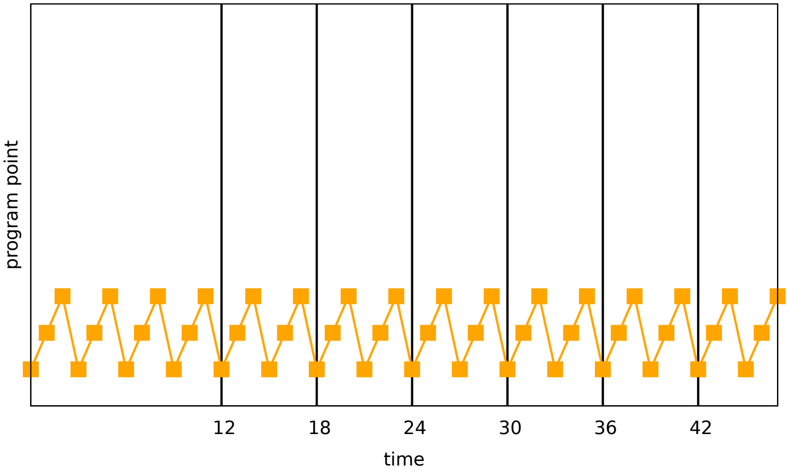
Execution Trace of Loop

Easy to make regular and uniform checkpoints



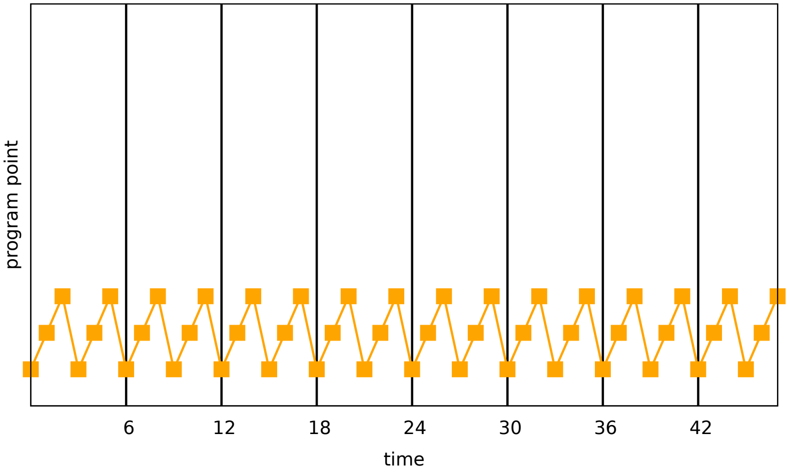
Execution Trace of Loop

Easy to make regular and uniform checkpoints

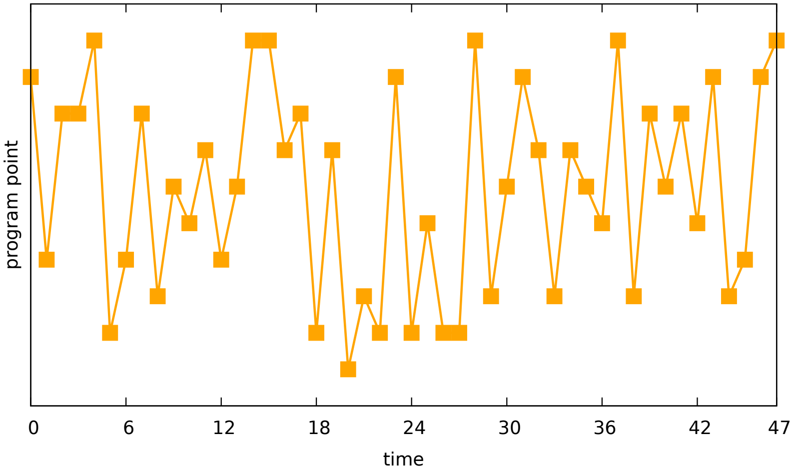


Execution Trace of Loop

Easy to make regular and uniform checkpoints

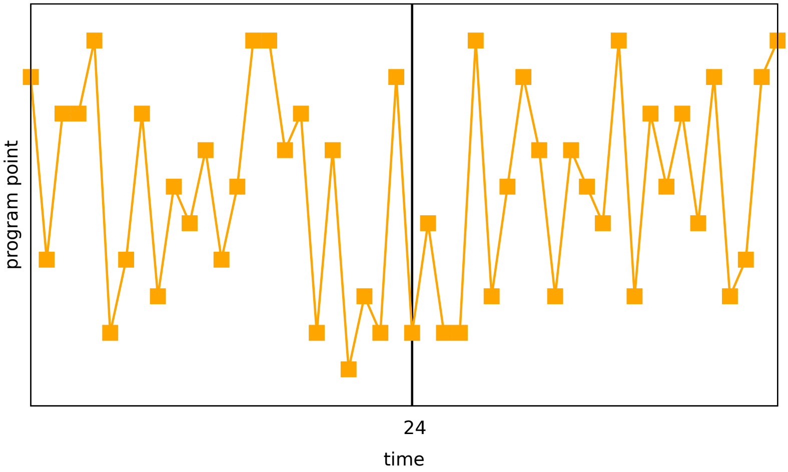


Execution Trace of Arbitrary Code



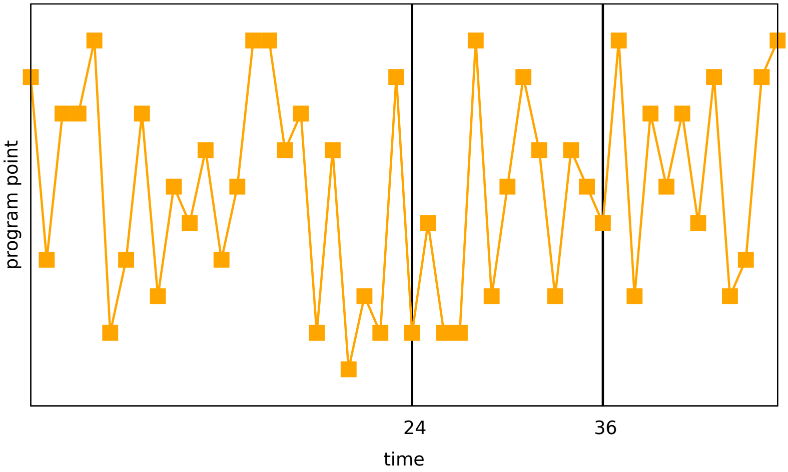
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



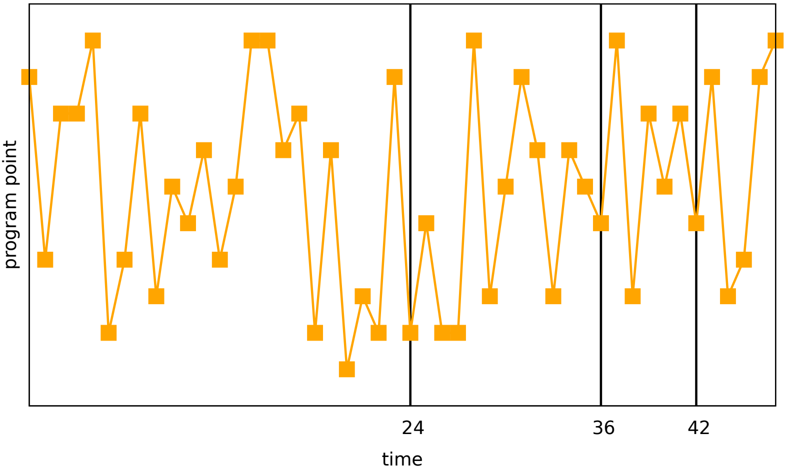
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



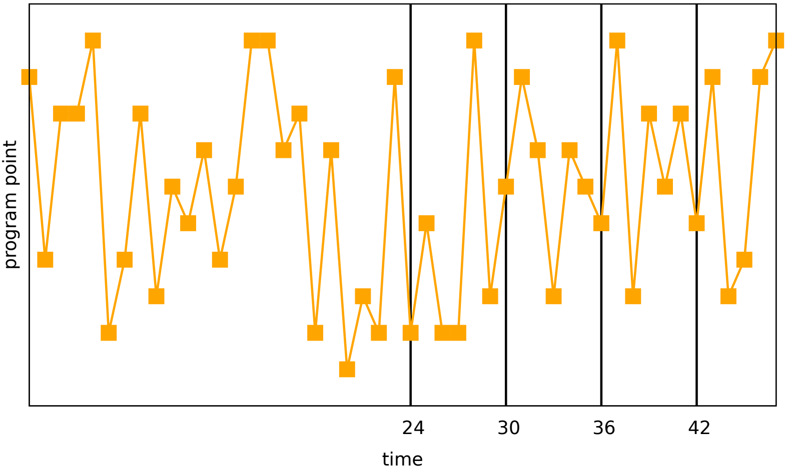
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



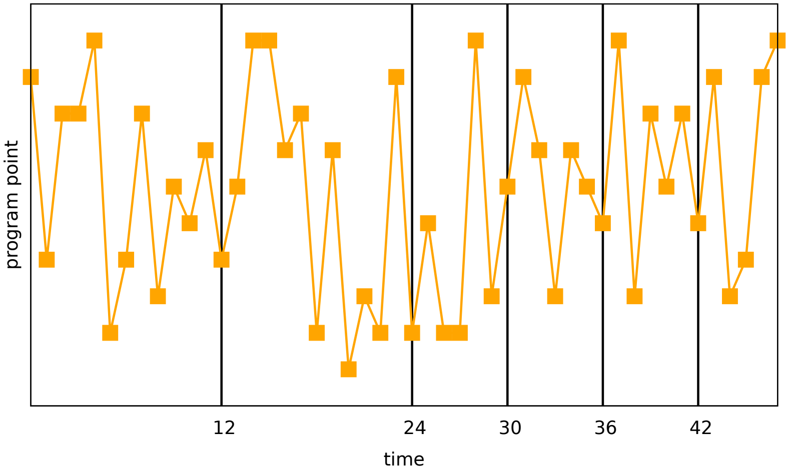
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



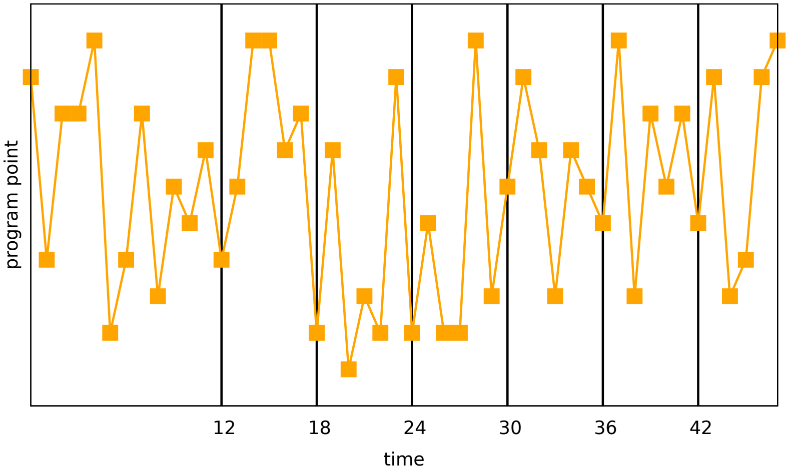
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



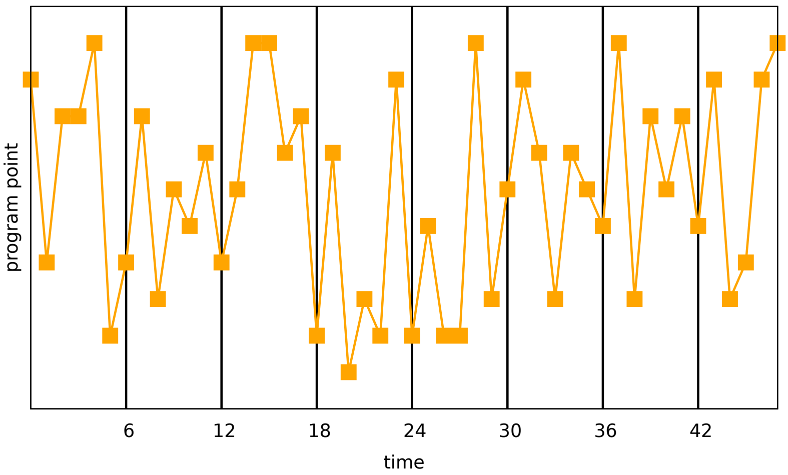
Execution Trace of Arbitrary Code

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.

Key Idea

```
function main(w)
   local x = f(w)
   local y = h(g(x))
   local z = p(y)
   return z
end
```

Key Idea

```
function main(w)
    local x = f(w)
    local y = h(g(x))
    local z = p(y)
    return z
end
```

$\rightsquigarrow$

```
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
```

Core Language

$$e ::= c \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 \mid \diamond e \mid e_1 \bullet e_2$$

Adding AD Operators to the Core Language

$$\overleftarrow{\mathcal{J}} : f\ x\ \grave{y} \mapsto (y, \grave{x}) \qquad\qquad \overset{\checkmark}{\mathcal{J}} : f\ x\ \grave{y} \mapsto (y, \grave{x})$$

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\; x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\; f\; x\; \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\; x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\; h\; z\; \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; g\; x\; \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case (f x fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\; f\; x\; \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\; x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\; h\; z\; \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; g\; x\; \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overset{\leftarrow}{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y,\grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y,\grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z,\grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overset{\leftarrow}{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

What is Needed to Implement the Algorithm?

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 $y = f(x)$.

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 $y = f(x)$.

General-Purpose Interruption and Resumption Interface

$\textsc{primops}\ f\ x \mapsto l$	Return the number l of evaluation steps needed to compute $y = f(x)$.
$\textsc{interrupt}\ f\ x\ l \mapsto z$	Run the first l steps of the computation of $f(x)$ and return a capsule z.
$\textsc{resume}\ z \mapsto y$	If $z = (\textsc{interrupt}\ f\ x\ l)$, 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 $y = f(x)$.

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

$\textsc{primops}\ f\ x \mapsto l$	Return the number l of evaluation steps needed to compute $y = f(x)$.
$\textsc{interrupt}\ f\ x\ l \mapsto z$	Run the first l steps of the computation of $f(x)$ and return a capsule z.
$\textsc{resume}\ z \mapsto y$	If $z = (\textsc{interrupt}\ f\ x\ l)$, 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 $y = f(x)$.

General-Purpose Interruption and Resumption Interface

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

General-Purpose Interruption and Resumption Interface

$\text{PRIMOPS}\, f\ x \mapsto l$	Return the number l of evaluation steps needed to compute $y = f(x)$.
$\text{INTERRUPT}\, f\ x\ l \mapsto z$	Run the first l steps of the computation of $f(x)$ and return a capsule z.
$\text{RESUME}\ z \mapsto y$	If $z = \left(\text{INTERRUPT}\, f\ x\ l\right)$, 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 $y = f(x)$.

General-Purpose Interruption and Resumption Interface

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

General-Purpose Interruption and Resumption Interface

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

General-Purpose Interruption and Resumption Interface

$\text{PRIMOPS}\ f\ x \mapsto l$	Return the number l of evaluation steps needed to compute $y = f(x)$.
$\text{INTERRUPT}\ f\ x\ l \mapsto z$	Run the first l steps of the computation of $f(x)$ and return a capsule z.
$\text{RESUME}\ z \mapsto y$	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 $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $l = \text{PRIMOPS}\ f\ x$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $l = \text{PRIMOPS}\ f\ x$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $l = \text{PRIMOPS}\ f\ x$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overset{\leftarrow}{\mathcal{J}}\ f\ x\ \grave{y}$ (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)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

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

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (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)
$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)
$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

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

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda z.\text{RESUME}\ z)\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

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

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda z.\text{RESUME}\ z)\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\;x$ fast): $(y,\grave{x})=\overleftarrow{\mathcal{J}}\;f\;x\;\grave{y}$ (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)

$(y,\grave{z})=\overset{\checkmark}{\mathcal{J}}\;(\lambda z.\text{RESUME}\;z)\;z\;\grave{y}$ (step 3)

$(z,\grave{x})=\overset{\checkmark}{\mathcal{J}}\;g\;x\;\grave{z}$ (step 4)

Algorithm for Divide-and-Conquer Checkpointing

via General-Purpose Interruption and Resumption Interface

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

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $l = \text{PRIMOPS}\ f\ x$ (step 1)

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

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda z.\text{RESUME}\ z)\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda x.\text{INTERRUPT}\ f\ x\ \lfloor \frac{l}{2} \rfloor)\ x\ \grave{z}$ (step 4)

Example of CPS Conversion

```
function f(x)
   return q(p(g(x), h(x)))
end
```

⇝

```
function f(c, x)
   return g(function(t1)
               return h(function(t2)
                           return p(function(t3)
                                       return q(c, t3)
                                    end, t1, t2)
                        end, x)
            end, x)
end
```

Implementation

Implementation

1. Convert source program to CPS.

Implementation

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

Implementation

1. Convert source program to CPS.
2. Thread step count and limit.
3. Translate CPS to 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 $\overset{\checkmark}{\mathcal{J}}$ 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 $\sqrt{\mathcal{J}}$ 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 $\sqrt{\mathcal{J}}$ written in C.
5. Compile to machine code.

CPS Conversion as a Program Transformation

$$\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x) \urcorner
\end{aligned}$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
{}^{\ulcorner}_{\llcorner}{\color{red}x}|k{}^{\urcorner}_{\lrcorner} &\leadsto k\;x\\
{}^{\ulcorner}_{\llcorner}(\lambda x.e)|k{}^{\urcorner}_{\lrcorner} &\leadsto k\;(\lambda k'\;x.{}^{\ulcorner}_{\llcorner}e|k'{}^{\urcorner}_{\lrcorner})\\
{}^{\ulcorner}_{\llcorner}(e_1\;e_2)|k{}^{\urcorner}_{\lrcorner} &\leadsto {}^{\ulcorner}_{\llcorner}e_1|(\lambda x_1.{}^{\ulcorner}_{\llcorner}e_2|(\lambda x_2.(x_1\;k\;x_2)){}^{\urcorner}_{\lrcorner}){}^{\urcorner}_{\lrcorner}\\
e_0 &\leadsto {}^{\ulcorner}_{\llcorner}e_0|(\lambda x.x){}^{\urcorner}_{\lrcorner}
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$\begin{aligned}
\ulcorner x|{\color{red}k}\urcorner &\leadsto k\;x\\
\ulcorner(\lambda x.e)|k\urcorner &\leadsto k\;(\lambda k'\;x.\ulcorner e|k'\urcorner)\\
\ulcorner(e_1\;e_2)|k\urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\;k\;x_2))\urcorner)\urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$\begin{aligned}
\ulcorner {\color{red}x}|k \urcorner &\leadsto k\ {\color{red}x}\\
\ulcorner(\lambda x.e)|k\urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner e|k'\urcorner)\\
\ulcorner(e_1\ e_2)|k\urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\ x \\
\ulcorner (\lambda x.e)|k \urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner e|k' \urcorner) \\
\ulcorner (e_1\ e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner \\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\ x\\
\ulcorner (\lambda x.e)|{\color{red}k} \urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x | k \urcorner &\leadsto k\ x \\
\ulcorner (\lambda x.e) | k \urcorner &\leadsto k\ (\lambda k'\ x. \ulcorner e | k' \urcorner) \\
\ulcorner (e_1\ e_2) | k \urcorner &\leadsto \ulcorner e_1 | (\lambda x_1. \ulcorner e_2 | (\lambda x_2. (x_1\ k\ x_2)) \urcorner) \urcorner \\
e_0 &\leadsto \ulcorner e_0 | (\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
{}^{\ulcorner}_{\llcorner} x|k {}^{\urcorner}_{\lrcorner} &\rightsquigarrow k\;x\\
{}^{\ulcorner}_{\llcorner} ({\color{red}\lambda x}.e)|k {}^{\urcorner}_{\lrcorner} &\rightsquigarrow k\;({\color{red}\lambda} k'\;{\color{red}x}.{}^{\ulcorner}_{\llcorner} e|k' {}^{\urcorner}_{\lrcorner})\\
{}^{\ulcorner}_{\llcorner} (e_1\;e_2)|k {}^{\urcorner}_{\lrcorner} &\rightsquigarrow {}^{\ulcorner}_{\llcorner} e_1|(\lambda x_1.{}^{\ulcorner}_{\llcorner} e_2|(\lambda x_2.(x_1\;k\;x_2)) {}^{\urcorner}_{\lrcorner}) {}^{\urcorner}_{\lrcorner}\\
e_0 &\rightsquigarrow {}^{\ulcorner}_{\llcorner} e_0|(\lambda x.x) {}^{\urcorner}_{\lrcorner}
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\leadsto k\ (\lambda {\color{red}k'}\ x.\ulcorner e|{\color{red}k'} \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner ({\color{red}e_1}\ {\color{red}e_2})|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x | k \urcorner &\leadsto k\ x \\
\ulcorner (\lambda x.e) | k \urcorner &\leadsto k\ (\lambda k'\ x. \ulcorner e | k' \urcorner) \\
\ulcorner (e_1\ e_2) | {\color{red}k} \urcorner &\leadsto \ulcorner e_1 | (\lambda x_1. \ulcorner e_2 | (\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner \\
e_0 &\leadsto \ulcorner e_0 | (\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\ x\\
\ulcorner(\lambda x.e)|k\urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner e|k'\urcorner)\\
\ulcorner(e_1\ e_2)|k\urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|({\color{red}\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner})\urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner\!\llcorner x|k \urcorner\!\lrcorner &\rightsquigarrow k\ x \\
\ulcorner\!\llcorner (\lambda x.e)|k \urcorner\!\lrcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner\!\llcorner e|k' \urcorner\!\lrcorner) \\
\ulcorner\!\llcorner (e_1\ e_2)|k \urcorner\!\lrcorner &\rightsquigarrow \ulcorner\!\llcorner e_1|(\lambda \textcolor{red}{x_1}.\ulcorner\!\llcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner\!\lrcorner) \urcorner\!\lrcorner \\
e_0 &\rightsquigarrow \ulcorner\!\llcorner e_0|(\lambda x.x) \urcorner\!\lrcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
{}^{\ulcorner}_{\llcorner} x | k {}^{\urcorner}_{\lrcorner} &\leadsto k\; x\\
{}^{\ulcorner}_{\llcorner} (\lambda x.e) | k {}^{\urcorner}_{\lrcorner} &\leadsto k\; (\lambda k'\; x. {}^{\ulcorner}_{\llcorner} e | k' {}^{\urcorner}_{\lrcorner})\\
{}^{\ulcorner}_{\llcorner} (e_1\; e_2) | k {}^{\urcorner}_{\lrcorner} &\leadsto {}^{\ulcorner}_{\llcorner} e_1 | (\lambda x_1. {}^{\ulcorner}_{\llcorner} e_2 | (\lambda x_2.(x_1\; k\; x_2)) {}^{\urcorner}_{\lrcorner}) {}^{\urcorner}_{\lrcorner}\\
e_0 &\leadsto {}^{\ulcorner}_{\llcorner} e_0 | (\lambda x.x) {}^{\urcorner}_{\lrcorner}
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$\begin{aligned}
\ulcorner\!\!\llcorner x|k \urcorner\!\!\lrcorner &\leadsto k\ x\\
\ulcorner\!\!\llcorner (\lambda x.e)|k \urcorner\!\!\lrcorner &\leadsto k\ (\lambda k'\ x.\ulcorner\!\!\llcorner e|k' \urcorner\!\!\lrcorner)\\
\ulcorner\!\!\llcorner (e_1\ e_2)|k \urcorner\!\!\lrcorner &\leadsto \ulcorner\!\!\llcorner e_1|(\lambda x_1.\ulcorner\!\!\llcorner e_2|({\color{red}\lambda x_2.(x_1\ k\ x_2)})\urcorner\!\!\lrcorner)\urcorner\!\!\lrcorner\\
e_0 &\leadsto \ulcorner\!\!\llcorner e_0|(\lambda x.x)\urcorner\!\!\lrcorner
\end{aligned}$$

CPS Conversion as a Program Transformation

$$\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda {\color{red}x_2}.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\;x\\
\ulcorner(\lambda x.e)|k \urcorner &\leadsto k\;(\lambda k'\;x.\ulcorner e|k' \urcorner)\\
\ulcorner(e_1\;e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\;k\;x_2))\urcorner)\urcorner\\
e_0 &\leadsto \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\; x \\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\; (\lambda k'\; x.\ulcorner e|k' \urcorner) \\
\ulcorner (e_1\; e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\; k\; x_2)) \urcorner) \urcorner \\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\rightsquigarrow k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\urcorner)\urcorner\\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x)\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner\!\llcorner x|k \lrcorner\!\urcorner &\leadsto k\ x\\
\ulcorner\!\llcorner (\lambda x.e)|k \lrcorner\!\urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner\!\llcorner e|k' \lrcorner\!\urcorner)\\
\ulcorner\!\llcorner (e_1\ e_2)|k \lrcorner\!\urcorner &\leadsto \ulcorner\!\llcorner e_1|(\lambda x_1.\ulcorner\!\llcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \lrcorner\!\urcorner) \lrcorner\!\urcorner\\
{\color{red}e_0} &\leadsto \ulcorner\!\llcorner e_0|(\lambda x.x) \lrcorner\!\urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$
\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k\ x\\
\ulcorner (\lambda x.e)|k \urcorner &\leadsto k\ (\lambda k'\ x.\ulcorner e|k' \urcorner)\\
\ulcorner (e_1\ e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2)) \urcorner) \urcorner\\
e_0 &\leadsto \ulcorner {\color{red}e_0}|(\lambda x.x) \urcorner
\end{aligned}
$$

CPS Conversion as a Program Transformation

$$\begin{aligned}
\ulcorner x|k \lrcorner &\rightsquigarrow k\ x \\
\ulcorner (\lambda x.e)|k \lrcorner &\rightsquigarrow k\ (\lambda k'\ x.\ulcorner e|k' \lrcorner) \\
\ulcorner (e_1\ e_2)|k \lrcorner &\rightsquigarrow \ulcorner e_1|(\lambda x_1.\ulcorner e_2|(\lambda x_2.(x_1\ k\ x_2))\lrcorner)\lrcorner \\
e_0 &\rightsquigarrow \ulcorner e_0|(\lambda x.x)\lrcorner
\end{aligned}$$

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 $\sqrt{\mathcal{J}}$ written in C.
5. Compile to machine code.

Treading Step Counts and Limits in CPS Conversion

$$\begin{aligned}
\ulcorner x|k \urcorner &\leadsto k \quad x\\
\ulcorner(\lambda x.e)|k \urcorner &\leadsto k \quad (\lambda k \quad x.\ulcorner e|k \urcorner)\\
\ulcorner(e_1\ e_2)|k \urcorner &\leadsto \ulcorner e_1|(\lambda \quad x_1.\\
&\qquad\qquad \ulcorner e_2|(\lambda \quad x_2.\\
&\qquad\qquad\qquad (x_1\ k \quad x_2)),\\
&\qquad\qquad\qquad \urcorner),\\
&\qquad\qquad\qquad \urcorner
\end{aligned}$$

Treading Step Counts and Limits in CPS Conversion

$$
\begin{aligned}
\ulcorner x|k \qquad \urcorner &\leadsto k \qquad\qquad x\\
\ulcorner(\lambda x.e)|k \qquad \urcorner &\leadsto k \qquad\qquad (\lambda k \quad x.\ulcorner e|k \qquad \urcorner)\\
\ulcorner(e_1\ e_2)|k \qquad \urcorner &\leadsto \ulcorner e_1|(\lambda \quad x_1.\\
&\qquad\qquad \ulcorner e_2|(\lambda \quad x_2.\\
&\qquad\qquad\qquad (x_1\ k \quad x_2)),\\
&\qquad\qquad\qquad \urcorner),\\
&\qquad\qquad\qquad \urcorner
\end{aligned}
$$

Treading Step Counts and Limits in CPS Conversion

$$\begin{aligned}
\ulcorner x|k,n \urcorner &\leadsto k\ (n+1)\ x\\
\ulcorner (\lambda x.e)|k,n \urcorner &\leadsto k\ (n+1)\ (\lambda k\ n\ x.\ulcorner e|k,n \urcorner)\\
\ulcorner (e_1\ e_2)|k,n \urcorner &\leadsto \ulcorner e_1|(\lambda n\ x_1.\\
&\qquad\qquad \ulcorner e_2|(\lambda n\ x_2.\\
&\qquad\qquad\qquad (x_1\ k\ n\ x_2)),\\
&\qquad\qquad\quad n \urcorner),\\
&\qquad\quad (n+1) \urcorner
\end{aligned}$$

Treading Step Counts and Limits in CPS Conversion

$$\begin{aligned}
\ulcorner x|k,n,l\urcorner &\rightsquigarrow k\ (n+1)\ l\ x\\
\ulcorner(\lambda x.e)|k,n,l\urcorner &\rightsquigarrow k\ (n+1)\ l\ (\lambda k\ n\ l\ x.\ulcorner e|k,n,l\urcorner)\\
\ulcorner(e_1\ e_2)|k,n,l\urcorner &\rightsquigarrow \ulcorner e_1|(\lambda n\ l\ x_1.\\
&\qquad\qquad \ulcorner e_2|(\lambda n\ l\ x_2.\\
&\qquad\qquad\qquad (x_1\ k\ n\ l\ x_2)),\\
&\qquad\qquad\quad n,l\urcorner),\\
&\qquad\quad (n+1),l\urcorner
\end{aligned}$$

Treading Step Counts and Limits in CPS Conversion

$$
\begin{aligned}
\ulcorner x|k,n,l\urcorner &\rightsquigarrow k\;(n+1)\;l\;x\\
\ulcorner(\lambda x.e)|k,n,l\urcorner &\rightsquigarrow k\;(n+1)\;l\;(\lambda k\;n\;l\;x.\ulcorner e|k,n,l\urcorner)\\
\ulcorner(e_1\;e_2)|k,n,l\urcorner &\rightsquigarrow \ulcorner e_1|(\lambda n\;l\;x_1.\\
&\qquad\qquad \ulcorner e_2|(\lambda n\;l\;x_2.\\
&\qquad\qquad\qquad\qquad (x_1\;k\;n\;l\;x_2)),\\
&\qquad\qquad\qquad n,l\urcorner),\\
&\qquad\qquad (n+1),l\urcorner
\end{aligned}
$$

$$\langle\!\langle e\rangle\!\rangle_{k,n,l}\rightsquigarrow \textbf{if } n=l \textbf{ then } [\![k,\lambda k\;n\;l\;\_.e]\!] \textbf{ else } e$$

Treading Step Counts and Limits in CPS Conversion

$$\begin{aligned}
\ulcorner\!\!\llcorner x|k,n,l\urcorner\!\!\lrcorner &\rightsquigarrow {\color{red}\langle\!\langle} k\ (n+1)\ l\ x {\color{red}\rangle\!\rangle_{k,n,l}}\\
\ulcorner\!\!\llcorner(\lambda x.e)|k,n,l\urcorner\!\!\lrcorner &\rightsquigarrow {\color{red}\langle\!\langle} k\ (n+1)\ l\ (\lambda k\ n\ l\ x.\ulcorner\!\!\llcorner e|k,n,l\urcorner\!\!\lrcorner) {\color{red}\rangle\!\rangle_{k,n,l}}\\
\ulcorner\!\!\llcorner(e_1\ e_2)|k,n,l\urcorner\!\!\lrcorner &\rightsquigarrow {\color{red}\langle\!\langle} \ulcorner\!\!\llcorner e_1|(\lambda n\ l\ x_1.\\
&\qquad\qquad\qquad \ulcorner\!\!\llcorner e_2|(\lambda n\ l\ x_2.\\
&\qquad\qquad\qquad\qquad\qquad (x_1\ k\ n\ l\ x_2)),\\
&\qquad\qquad\qquad\qquad n,l\urcorner\!\!\lrcorner),\\
&\qquad\qquad\qquad (n+1),l\urcorner\!\!\lrcorner {\color{red}\rangle\!\rangle_{k,n,l}}
\end{aligned}$$

$$\langle\!\langle e\rangle\!\rangle_{k,n,l} \rightsquigarrow \textbf{if } n=l \textbf{ then } [\![k,\lambda k\ n\ l\ \_.e]\!] \textbf{ else } e$$

Treading Step Counts and Limits in CPS Conversion

$$
\begin{aligned}
\ulcorner x|k,n,l\urcorner &\rightsquigarrow \langle\!\langle k\ (n+1)\ l\ x\rangle\!\rangle_{k,n,l}\\
\ulcorner(\lambda x.e)|k,n,l\urcorner &\rightsquigarrow \langle\!\langle k\ (n+1)\ l\ (\lambda k\ n\ l\ x.\ulcorner e|k,n,l\urcorner)\rangle\!\rangle_{k,n,l}\\
\ulcorner(e_1\ e_2)|k,n,l\urcorner &\rightsquigarrow \langle\!\langle \ulcorner e_1|(\lambda n\ l\ x_1.\\
&\qquad\qquad\quad \ulcorner e_2|(\lambda n\ l\ x_2.\\
&\qquad\qquad\qquad\qquad (x_1\ k\ n\ l\ x_2)),\\
&\qquad\qquad\qquad\quad n,l\urcorner),\\
&\qquad\qquad\quad (n+1),l\urcorner\rangle\!\rangle_{k,n,l}\\
&\vdots\\
\langle\!\langle e\rangle\!\rangle_{k,n,l} &\rightsquigarrow \mathbf{if}\ n=l\ \mathbf{then}\ [\![k,\lambda k\ n\ l\ \_.e]\!]\ \mathbf{else}\ e
\end{aligned}
$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

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

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

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

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$
\begin{aligned}
\text{PRIMOPS}\, f\ \textcolor{red}{x} &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ \textcolor{red}{x}\\
\text{INTERRUPT}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\ f\ x\\
\text{RESUME}\ [\![k, f]\!] &= \mathcal{A}\ k\ 0\ \infty\ f\ \bot
\end{aligned}
$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

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

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}\textsc{primops}\, f\; x &= \mathcal{A}\;(\lambda n\; l\; v.n)\; 0\; {\color{red}\infty}\; f\; x\\ \textsc{interrupt}\, f\; x\; l &= \mathcal{A}\;(\lambda n\; l\; v.v)\; 0\; l\, f\; x\\ \textsc{resume}\; [\![k, f]\!] &= \mathcal{A}\; k\; 0\; \infty\; f\; \bot\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$
\begin{aligned}
\text{PRIMOPS}\,f\;x &= \mathcal{A}\;(\lambda n\;l\;v.n)\;0\;\infty\;f\;x\\
\text{INTERRUPT}\,f\;x\;l &= \mathcal{A}\;(\lambda n\;l\;v.v)\;0\;l\,f\;x\\
\text{RESUME}\;[\![k,f]\!] &= \mathcal{A}\;k\;0\;\infty\;f\;\perp
\end{aligned}
$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}
\text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\text{INTERRUPT}\, {\color{red}f}\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\ {\color{red}f}\ x\\
\text{RESUME}\ [\![k, f]\!] &= \mathcal{A}\ k\ 0\ \infty\ f\ \bot
\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned} \text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x \\ \text{INTERRUPT}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\, f\ x \\ \text{RESUME}\ \llbracket k, f \rrbracket &= \mathcal{A}\ k\ 0\ \infty\ f\ \bot \end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}
\text{PRIMOPS}\,f\;x &= \mathcal{A}\;(\lambda n\;l\;v.n)\;0\;\infty\;f\;x\\
\text{INTERRUPT}\,f\;x\;l &= \mathcal{A}\;(\lambda n\;l\;v.v)\;{\color{red}0}\;l\,f\;x\\
\text{RESUME}\;[\![k,f]\!] &= \mathcal{A}\;k\;0\;\infty\;f\;\bot
\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}
\text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\text{INTERRUPT}\, f\ x\ \textcolor{red}{l} &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ \textcolor{red}{l}\, f\ x\\
\text{RESUME}\ [\![k,f]\!] &= \mathcal{A}\ k\ 0\ \infty\ f\ \bot
\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}
\text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\text{INTERRUPT}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\, f\ x\\
\text{RESUME}\ \color{red}{[\![k, f]\!]} &= \mathcal{A}\ k\ 0\ \infty\ f\ \bot
\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$
\begin{aligned}
\text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\text{INTERRUPT}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\ f\ x\\
\text{RESUME}\ [\![k, {\color{red}f}]\!] &= \mathcal{A}\ k\ 0\ \infty\ {\color{red}f}\ \bot
\end{aligned}
$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

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

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$
\begin{aligned}
\textsc{primops}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\textsc{interrupt}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\, f\ x\\
\textsc{resume}\ [\![{\color{red}k}, f]\!] &= \mathcal{A}\ {\color{red}k}\ 0\ \infty\ f\ \bot
\end{aligned}
$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$
\begin{aligned}
\textsc{primops}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\textsc{interrupt}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\, f\ x\\
\textsc{resume}\ [\![k, f]\!] &= \mathcal{A}\ k\ \textcolor{red}{0}\ \infty\ f\ \bot
\end{aligned}
$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}
\text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\text{INTERRUPT}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ v.v)\ 0\ l\ f\ x\\
\text{RESUME}\ [\![k,f]\!] &= \mathcal{A}\ k\ 0\ {\color{red}\infty}\ f\ \bot
\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}
\text{PRIMOPS}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\
\text{INTERRUPT}\, f\ x\ l &= \mathcal{A}\ {\color{red}(\lambda n\ l\ v.v)}\ 0\ l\ f\ x\\
\text{RESUME}\ [\![k,f]\!] &= \mathcal{A}\ {\color{red}k}\ 0\ \infty\ f\ \bot
\end{aligned}$$

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

$$\begin{aligned}\textsc{primops}\, f\ x &= \mathcal{A}\ (\lambda n\ l\ v.n)\ 0\ \infty\ f\ x\\ \textsc{interrupt}\, f\ x\ l &= \mathcal{A}\ (\lambda n\ l\ {\color{red}v}.{\color{red}v})\ 0\ l\, f\ x\\ \textsc{resume}\ [\![k, f]\!] &= \mathcal{A}\ k\ 0\ \infty\ f\ \bot\end{aligned}$$

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 $\sqrt{\mathcal{J}}$ written in C.
5. Compile to machine code.

Code Generation

$$\begin{aligned}
\mathcal{S}\,\pi\,() &= \texttt{null\_constant}\\
\mathcal{S}\,\pi\,\textbf{true} &= \texttt{true\_constant}\\
\mathcal{S}\,\pi\,\textbf{false} &= \texttt{false\_constant}\\
\mathcal{S}\,\pi\,(c_1, c_2) &= \texttt{cons}\,\texttt{(}(\mathcal{S}\,\pi\,c_1)\texttt{,}\;\;(\mathcal{S}\,\pi\,c_1)\texttt{)}\\
\mathcal{S}\,\pi\,n &\\
\mathcal{S}\,\pi\,\texttt{`k'} &= \texttt{continuation}\\
\mathcal{S}\,\pi\,\texttt{`n'} &= \texttt{count}\\
\mathcal{S}\,\pi\,\texttt{`l'} &= \texttt{limit}\\
\mathcal{S}\,\pi\,\texttt{`x'} &= \texttt{argument}\\
\mathcal{S}\,\pi\,x &= \texttt{as\_closure(target)->environment[}\pi\,x\texttt{]}
\end{aligned}$$

Code Generation

```
$\mathcal{S}\;\pi\;(\lambda_3 n\;l\;x.e)$ = ({
               thing function(thing target,
                              thing count,
                              thing limit,
                              thing argument) {
               return $(\mathcal{S}\;(\phi\;e)\;e)$;
               }
               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;
               as_closure(lambda)->n = $|\phi\;e|$;
               as_closure(lambda)->environment[$0$] = $\mathcal{S}\;\pi\;(\phi\;e)_0$
               $\vdots$
               as_closure(lambda)->environment[$|\phi\;e|-1$] = $\mathcal{S}\;\pi\;(\phi\;e)_{|\phi\;e|-1}$
               lambda;
             })
```

Code Generation

```
$\mathcal{S}\;\pi\;(\lambda_4 k\;n\;l\;x.e)$ = ({
               thing function(thing target,
                              thing continuation,
                              thing count,
                              thing limit,
                              thing argument) {
               return ($\mathcal{S}\;(\phi\;e)\;e$);
               }
               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;
               as_closure(lambda)->n = $|\phi\;e|$;
               as_closure(lambda)->environment[0] = $\mathcal{S}\;\pi\;(\phi\;e)_0$
               ⋮
               as_closure(lambda)->environment[$|\phi\;e|-1$] = $\mathcal{S}\;\pi\;(\phi\;e)_{|\phi\;e|-1}$
               lambda;
             })
```

Code Generation

$$\begin{aligned}\mathcal{S}\;\pi\;(e_1\;e_2\;e_3\;e_4) = \texttt{continuation\_apply}(&(\mathcal{S}\;\pi\;e_1)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_2)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_3)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_4)\texttt{)}\end{aligned}$$

$$\begin{aligned}\mathcal{S}\;\pi\;(e_1\;e_2\;e_3\;e_4\;e_5) = \texttt{converted\_apply}(&(\mathcal{S}\;\pi\;e_1)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_2)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_3)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_4)\texttt{,}\\ &(\mathcal{S}\;\pi\;e_5)\texttt{)}\end{aligned}$$

$$\mathcal{S}\;\pi\;(\mathbf{if}\;e_1\;\mathbf{then}\;e_2\;\mathbf{else}\;e_3) = \texttt{(!is\_false(}(\mathcal{S}\;\pi\;e_1)\texttt{)?}(\mathcal{S}\;\pi\;e_2)\texttt{:}(\mathcal{S}\;\pi\;e_3)\texttt{)}$$

$$\mathcal{S}\;\pi\;(\diamond e) = (\mathcal{N}\;\diamond)\texttt{(}(\mathcal{S}\;\pi\;e)\texttt{)}$$

$$\mathcal{S}\;\pi\;(e_1 \bullet e_2) = (\mathcal{N}\;\bullet)\texttt{(}(\mathcal{S}\;\pi\;e_1)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_2)\texttt{)}$$

$$\mathcal{S}\;\pi\;(\overrightarrow{\mathcal{J}}\;e_1\;e_2\;e_3) = (\mathcal{N}\;\overrightarrow{\mathcal{J}})\texttt{(}(\mathcal{S}\;\pi\;e_1)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_2)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_3)\texttt{)}$$

$$\mathcal{S}\;\pi\;(\overleftarrow{\mathcal{J}}\;e_1\;e_2\;e_3) = (\mathcal{N}\;\overleftarrow{\mathcal{J}})\texttt{(}(\mathcal{S}\;\pi\;e_1)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_2)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_3)\texttt{)}$$

$$\mathcal{S}\;\pi\;(\overset{\checkmark}{\mathcal{J}}\;e_1\;e_2\;e_3) = (\mathcal{N}\;\overset{\checkmark}{\mathcal{J}})\texttt{(}(\mathcal{S}\;\pi\;e_1)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_2)\texttt{,}\;\;(\mathcal{S}\;\pi\;e_3)\texttt{)}$$

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 $\sqrt{\mathcal{I}}$ written in C.
5. Compile to machine code.

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

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

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

written in 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);
}
```

Implementation of the General-Purpose Interruption and Resumption Interface with Threaded CPS

```
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;
}
```

Algorithm for Divide-and-Conquer Checkpointing

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

base case (f x fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (step 0)

inductive case: $h \circ g = f$ (step 1)

$z = g\ x$ (step 2)

$(y, \grave{z}) = \overset{\checkmark}{\mathcal{J}}\ h\ z\ \grave{y}$ (step 3)

$(z, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{z}$ (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));
}
```

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 $\overset{\checkmark}{\mathcal{J}}$ written in C.
5. Compile to machine code.

Three Reference Implementations

Three Reference Implementations

1. Interpreter using CPS evaluator

Three Reference Implementations

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

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

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.

All three documented in detail in paper.
J.M. Siskind and B.A. Pearlmutter, 'Divide-and-Conquer Checkpointing for Arbitrary Programs with No User Annotation,' *Optimization Methods and Software (OMS)*, 33(4–6):1288–1330, September 2018.
`https://www.tandfonline.com/doi/full/10.1080/10556788.2018.1459621`

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.
J.M. Siskind and B.A. Pearlmutter, ‘Divide-and-Conquer Checkpointing for Arbitrary Programs with No User Annotation,’ *Optimization Methods and Software (OMS)*, 33(4–6):1288–1330, September 2018.
`https://www.tandfonline.com/doi/full/10.1080/10556788.2018.1459621`

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.
J.M. Siskind and B.A. Pearlmutter, 'Divide-and-Conquer Checkpointing for Arbitrary Programs with No User Annotation,' *Optimization Methods and Software (OMS)*, 33(4–6):1288–1330, September 2018.
`https://www.tandfonline.com/doi/full/10.1080/10556788.2018.1459621`

Could add FFI bindings to GPU Tensor library.

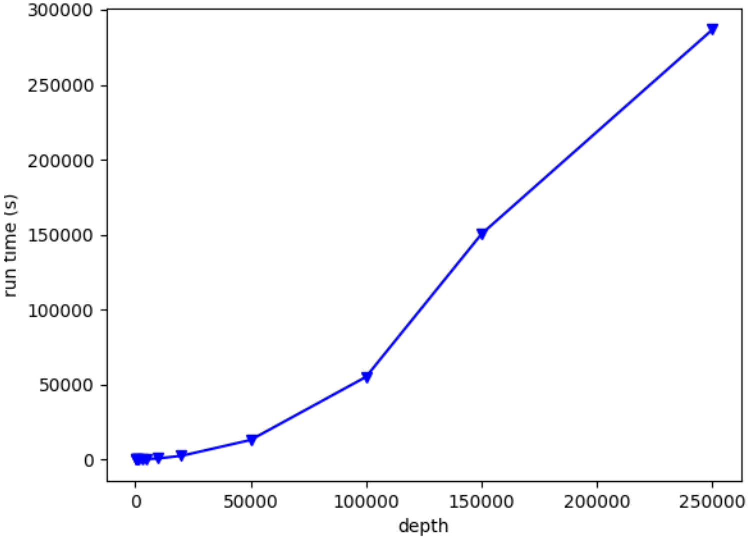
Scorch

list->⟨*residence*⟩-⟨*type*⟩-tensor	fill-⟨*residence*⟩-⟨*type*⟩	⟨*residence*⟩-⟨*type*⟩-tensor?
⟨*residence*⟩-⟨*type*⟩	randn-⟨*residence*⟩-⟨*floating type*⟩	normal-⟨*residence*⟩-⟨*floating type*⟩
size	tensor->list	view
transpose	narrow-tensor	expand-tensor
concat-tensors	dot	sumall
addmv	addmm	baddbmm
addr	resize	pad
crop	decimate	interpolate
interpolate-to-size	upsample-nearest	downsample-nearest
permute	ReLU	LeakyReLU
GeLU	sigmoid	convolve
transpose-convolve	batch-normalization-training	batch-normalization-test
initialize-batch-normalization	layer-normalization	convolve-add-tied
embedding	max-pool	average-pool
dropout	dropout-planewise	max-value
index-of-max	cross-entropy-loss	softmax
fused-scale-mask-softmax		

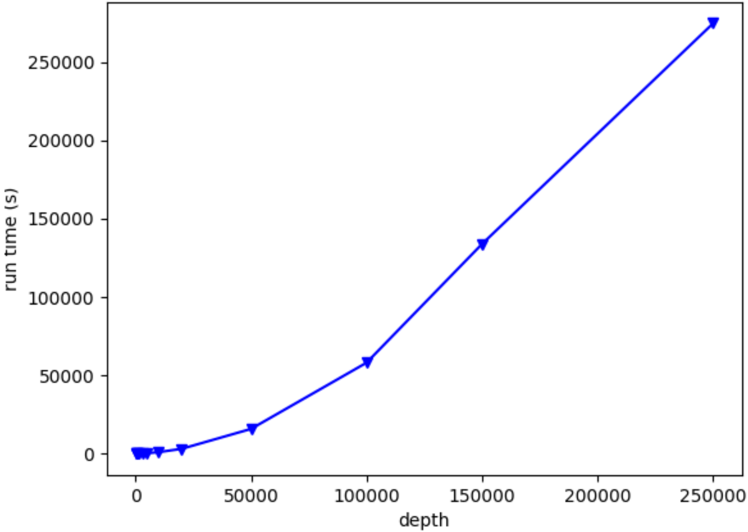
Beyond these, the standard unary basis procedures `sqrt`, `exp`, `log`, `sin`, `cos`, `zero?`, `positive?`, and `negative?` are extended pointwise to tensors and the standard binary basis procedures +, −, $\star$, /, `max`, `min`, `atan`, =, <, >, <=, and >= are extended to apply to a tensor and a scalar, or a scalar and a tensor, or two tensors of the same type and dimensions, in a pointwise fashion.

SCORCH

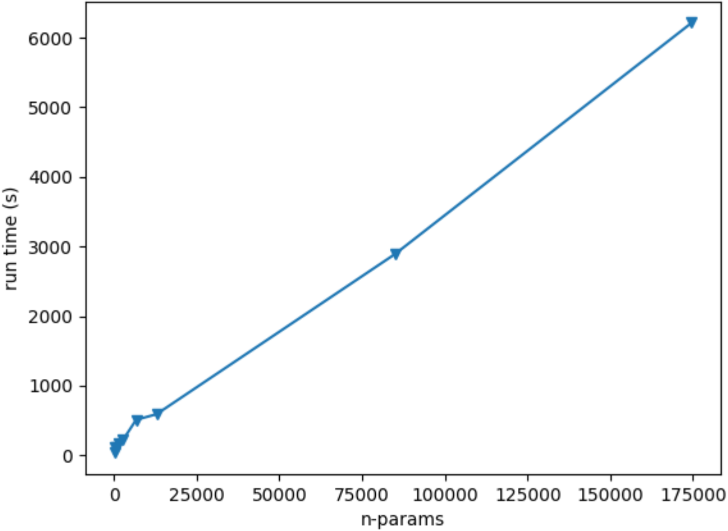
ResNet


300000
250000
200000
150000
100000
50000
0
run time (s)
0
50000
100000
150000
200000
250000
depth

DRANet


run time (s)
250000
200000
150000
100000
50000
0
0
50000
100000
150000
200000
250000
depth

GPT


run time (s)
6000
5000
4000
3000
2000
1000
0
0
25000
50000
75000
100000
125000
150000
175000
n-params

Ray Tracer

```
(map (lambda (y)
      (map (lambda (x)
             (shoot-ray (list y x))) (iota width)))
     (iota height))
```

Ray Tracer

```
(if antecedent consequent alternate)
```

Ray Tracer

```
(shoot-ray
 (map (lambda (y)
       (map (lambda (x) (list y x))
            (iota width)))
      (iota height)))
```

Ray Tracer

```
(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))))
```

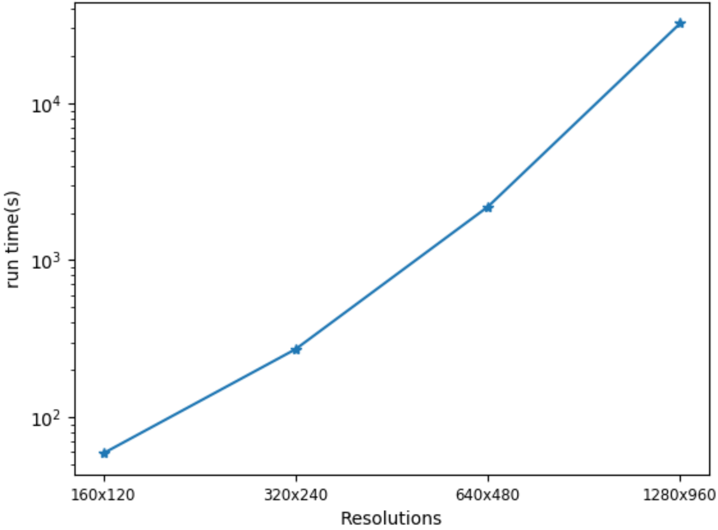
Ray Tracer

```
(map (lambda (y)
       (map (lambda (x) y)
            (iota width)))
      (iota height))
```

Ray Tracer

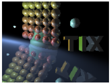
```
(define (if-function antecedent consequent alternate)
 (if antecedent consequent alternate))

(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))
```

Ray Tracer


run time(s)
10^4
10^3
10^2
160x120
320x240
640x480
1280x960
Resolutions

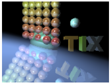
Target Scene

Light source at position `(−40 −15 60)`



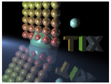
Initial Scene

Light source at position `(−20 −30 30)`



Final Scene after 111 iterations

Light source at position `(−42.7 −13.8 39.3)`



Comparison

	ResNet	GPT	Ray Tracer
TENSORFLOW (individual checkpointing)	608	4	280×210
PYTORCH (right-branching checkpointing)	760	48	320×240
DEEPSPEED	760	48	n/a
L2L	38,000	3,000	n/a
SCORCH	**250,000**	**14,000**	**1280×960**

Take-Home Message

Divide-and-Conquer checkpointing

Take-Home Message

Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations

Take-Home Message

Divide-and-Conquer checkpointing

- is traditionally formulated around loop iterations
- but can be extended to arbitrary code

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

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