One-and-a-Half Simple Differential Programming Languages

Gordon Plotkin

Calgary, 2019

$\sim$ Joint work at Google with Martín Abadi $\sim$

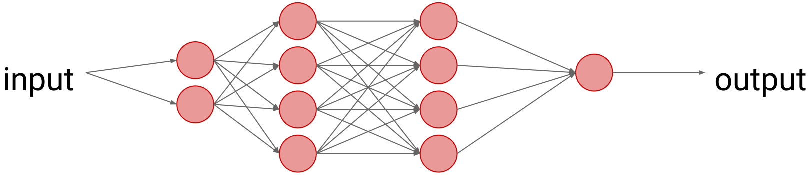
Talk Synopsis

- Review of neural nets
- Review of Differentiation
- A minilanguage
- Differentiating conditionals and loops
- Language semantics: operational and denotational
- Beyond powers of $\mathbb{R}$
- Conclusion and future work

Neural networks: a very brief introduction

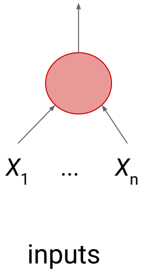
Deep learning is based on neural networks:

- loosely inspired by the brain;
- built from simple, *trainable* functions.



Primitives: the neuron

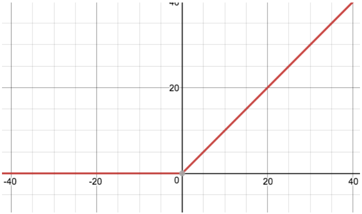
$$y = F(w_1 x_1 + \ldots + w_n x_n + b)$$



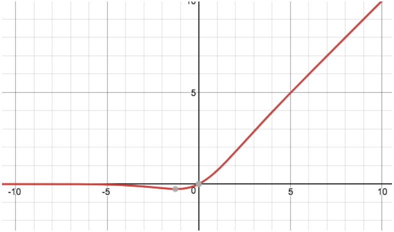
- $w_1 \ldots w_n$ are weights,
- b is a bias,
- weights and biases are parameters,
- F is a "differentiable" non-linear function, e.g., the "ReLU" $F(x) = \max(0, x)$

Two activation functions: ReLU and Swish

- ReLU $\max(x, 0)$

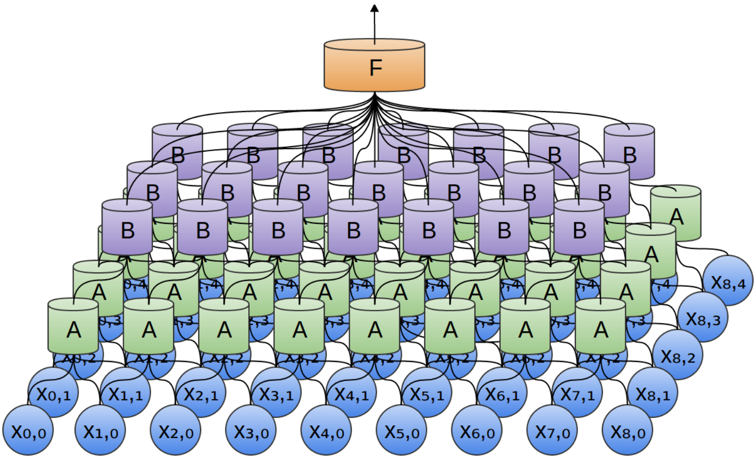


- Swish $x \cdot \sigma(\beta x)$, where $\sigma(z) = (1 + \mathrm{e}^{-z})^{-1}$



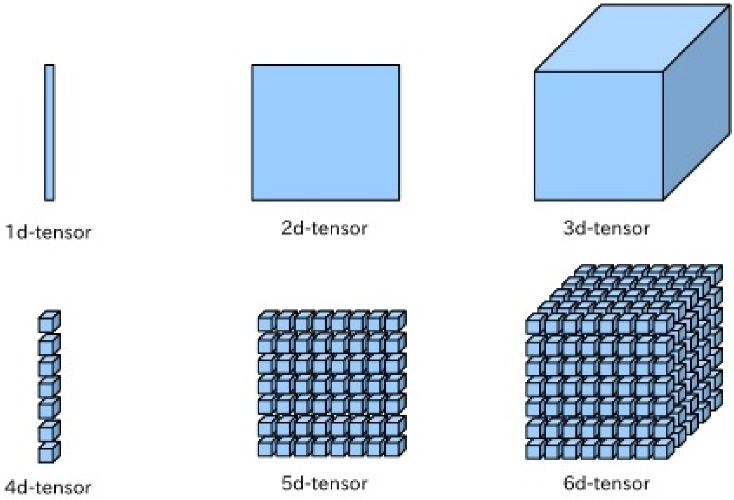
(Ramachandran, Zoph, and Le, 2017)

2d convolutional network



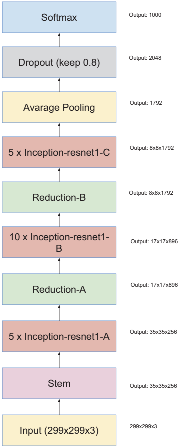
With thanks to C. Olah

Some tensors

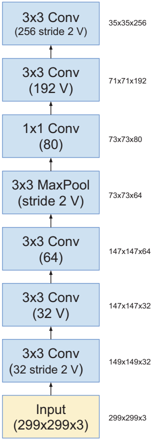


Convolutional image classification

Inception-Resnet-v1 achitecture



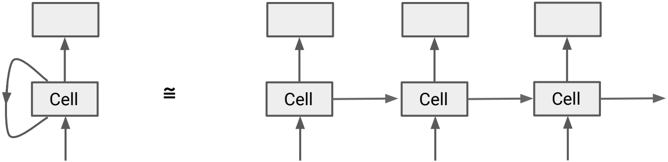
Schema



Stem

Szegedy et al, arxiv.org/abs/1602.07261

Recurrent neural networks (RNNs)

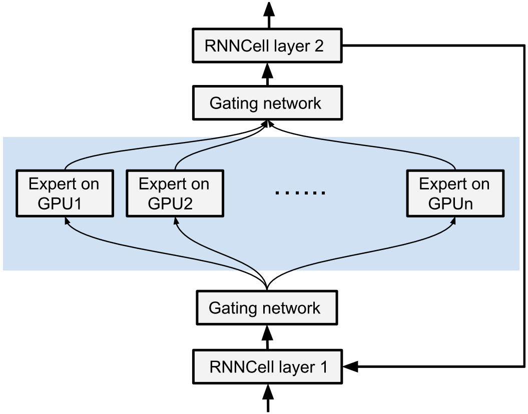


with shared parameters

There are many variants, e.g., LSTMs.

Mixture of experts

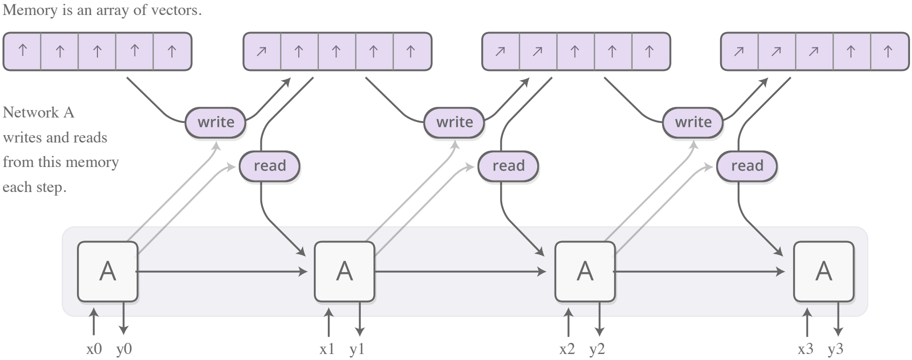
A model MoE architecture with a conditional and a loop:



With thanks to Yu et al.

Neural Turing Machines

Neural Turing Machines combine a RNN with an external memory bank:



With thanks to C. Olah

Supervised learning

Given a training dataset of (input, output) pairs, e.g., a set of images with labels:

While not done:

- Pick a pair (x, y)
- Run the neural network on x to get $\mathrm{Net}(x, b, \ldots)$
- Compare this to y to calculate the loss ($=$ error $=$ cost)

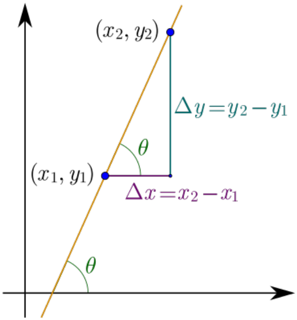
$$\mathrm{Loss}(b, \ldots) = |y - \mathrm{Net}(x, b, \ldots)|$$

- Adjust parameters $b, \ldots$ to reduce the loss

More generally, pick a “mini-batch" $(x_1, y_1), \ldots, (x_n, y_n)$ and minimise the loss

$$\mathrm{Loss}(b, \ldots) \ = \ \sqrt{\sum_{i=1}^{n} (y_i - Net(x_i, b, \ldots))^2}$$

Slope of a line



$$\text{slope} = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x}$$

So

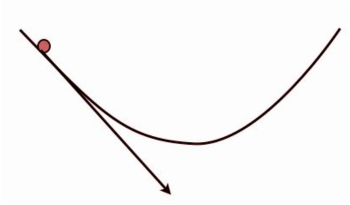
$$\Delta y = \text{slope} \times \Delta x$$

So

$$x' = x - r\,\text{slope} \implies y' = y - r\,\text{slope}^2$$

Gradient descent

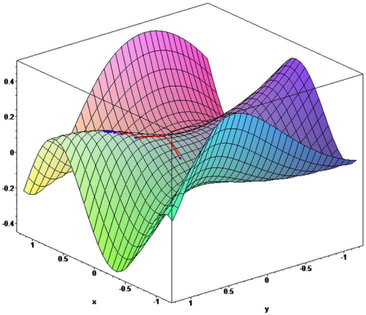
Follow the gradient of the loss function

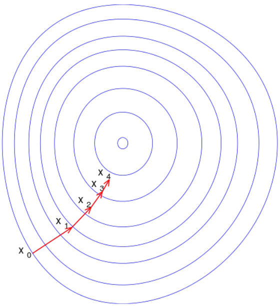


Thus:

$$x' := x - r(\text{slope of Loss at } x) = r\frac{\mathrm{d}\text{Loss(x)}}{\mathrm{d}x}$$

Multi-dimensional gradient descent





$$x' := x - r\frac{\partial \mathrm{L(x,y)}}{\partial x} \qquad \text{and} \qquad y' := y - r\frac{\partial \mathrm{L(x,y)}}{\partial y}$$

$$(x', y') := (x, y) - r\left(\frac{\partial \mathrm{L(x,y)}}{\partial x}, \frac{\partial \mathrm{L(x,y)}}{\partial y}\right)$$

$$\mathbf{v}' := \mathbf{v} - r\nabla L$$

Looking at differentiation

- Expressions with several variables:

$$\left.\frac{\partial e[x,y]}{\partial x}\right|_{x,y=a,b}$$

- Gradient of functions $f : \mathbb{R}^2 \to \mathbb{R}$ of two arguments:

$$\nabla(f) : \mathbb{R}^2 \to \mathbb{R}^2$$

$$\nabla(f)(u,v) \quad = \quad \left\langle \frac{\partial f(u,v)}{\partial u}, \frac{\partial f(u,v)}{\partial v} \right\rangle$$

- Chain rule

$$\frac{\partial f(g(x,y,z),h(x,y,z))}{\partial x} \quad = \quad \frac{\partial f(u,v)}{\partial u} \cdot \frac{\partial g(x,y,z)}{\partial x} + \frac{\partial f(u,v)}{\partial v} \cdot \frac{\partial h(x,y,z)}{\partial x}$$

where $u, v = g(x,y,z), h(x,y,z)$.

A matrix view of the multiargument chain rule.

- We have:

$$\begin{aligned}
\frac{\partial f(g(x,y,z),h(x,y,z))}{\partial x} &= \frac{\partial f(u,v)}{\partial u} \cdot \frac{\partial g(x,y,z)}{\partial x} + \frac{\partial f(u,v)}{\partial v} \cdot \frac{\partial h(x,y,z)}{\partial x} \\
\frac{\partial f(g(x,y,z),h(x,y,z))}{\partial y} &= \frac{\partial f(u,v)}{\partial u} \cdot \frac{\partial g(x,y,z)}{\partial y} + \frac{\partial f(u,v)}{\partial v} \cdot \frac{\partial h(x,y,z)}{\partial y} \\
\frac{\partial f(g(x,y,z),h(x,y,z))}{\partial z} &= \frac{\partial f(u,v)}{\partial u} \cdot \frac{\partial g(x,y,z)}{\partial z} + \frac{\partial f(u,v)}{\partial v} \cdot \frac{\partial h(x,y,z)}{\partial z}
\end{aligned}$$

- Set $k = \langle g, h \rangle : \mathbb{R}^3 \to \mathbb{R}^2$ and define its Jacobian to be the 2×3 matrix:

$$\mathrm{J}k = \begin{bmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial z} \\ \frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & \frac{\partial h}{\partial z} \end{bmatrix}$$

- Then the gradient of the composition $f \circ k$ is given by the vector-Jacobian product:

$$\nabla f(g(x,y,z), h(x,y,z)) \;=\; \nabla f(u,v) \cdot \mathrm{J}k(x,y,z)$$

Differentials: A functional view of differentiation

- Jacobians For $f : \mathbb{R}^m \to \mathbb{R}^n$ we have:

$$\mathrm{J}f : \mathbb{R}^m \to \mathrm{Mat}_{n,m}$$

- Chain rule for Jacobians For $\mathbb{R}^l \xrightarrow{f} \mathbb{R}^m \xrightarrow{g} \mathbb{R}^n$ we have:

$$\mathrm{J}_{\mathbf{x}}(g \circ f) = \mathrm{J}_{f(\mathbf{x})}(g) \cdot \mathrm{J}_{\mathbf{x}}(f)$$

- Differentials aka (forward) derivatives For $f : \mathbb{R}^m \to \mathbb{R}^n$ we define:

$$\mathrm{d}f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$$

by:

$$(\mathrm{d}_{\mathbf{x}}f)\mathbf{y} = (\mathrm{J}_{\mathbf{x}}f) \cdot \mathbf{y}$$

- Chain rule for differentials For $\mathbb{R}^l \xrightarrow{f} \mathbb{R}^m \xrightarrow{g} \mathbb{R}^n$ we have:

$$\mathrm{d}_{\mathbf{x}}(g \circ f) = \mathrm{d}_{f(\mathbf{x})}(g) \circ \mathrm{d}_{\mathbf{x}}(f)$$

Reverse derivatives

- Reverse derivatives For $f : \mathbb{R}^m \to \mathbb{R}^n$ we have:

$$\mathrm{d}^R f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$$

where:

$$(\mathrm{d}^R_{\mathbf{x}} f)\mathbf{y} \quad = \quad \mathbf{y} \cdot (\mathrm{J}_{\mathbf{x}} f) \quad (= \quad (\mathrm{d}_{\mathbf{x}} f)^\dagger \mathbf{y})$$

- Chain rule For $\mathbb{R}^l \xrightarrow{f} \mathbb{R}^m \xrightarrow{g} \mathbb{R}^n$ we have:

$$\mathrm{d}^R_{\mathbf{x}}(g \circ f) \quad = \quad \mathrm{d}^R_{\mathbf{x}}(f) \circ \mathrm{d}^R_{f(\mathbf{x})}(g)$$

as:

$$\begin{array}{rcl}
\mathrm{d}^R_{\mathbf{x}}(g \circ f)(\mathbf{z}) & = & \mathbf{z} \cdot \mathrm{J}_{\mathbf{x}}(g \circ f) \\
& = & \mathbf{z} \cdot (\mathrm{J}_{f(\mathbf{x})}(g) \cdot \mathrm{J}_{\mathbf{x}}(f)) \ = (\mathbf{z} \cdot \mathrm{J}_{f(\mathbf{x})}(g)) \cdot \mathrm{J}_{\mathbf{x}}(f) \\
& = & \mathrm{d}^R_{\mathbf{x}}(f)(\mathrm{d}^R_{f(\mathbf{x})}(g)(\mathbf{z}))
\end{array}$$

- Gradients For the case $n = 1$ where $f : \mathbb{R}^m \to \mathbb{R}$, we have:

$$\mathrm{d}^{\mathbb{R}}_{\mathbf{x}} f : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$$

and then:

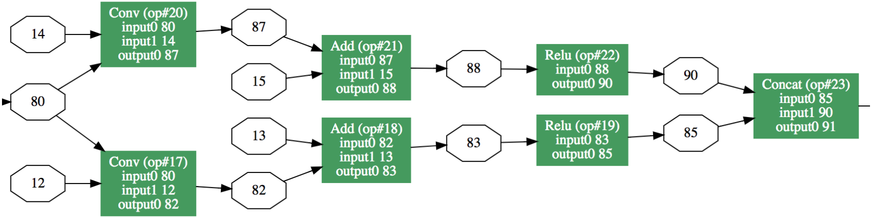
$$\nabla_{\mathbf{x}} f = (\mathrm{d}^R_{\mathbf{x}} f)1$$

Takeaway on differentiation

- For
$$f : \mathbb{R}^m \to \mathbb{R}^n$$
we have
  - $$\mathrm{d}f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$$
  - $$\mathrm{d}^R f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$$
- For
$$f : \mathbb{R}^m \to \mathbb{R}$$
we have:
$$\nabla_{\mathbf{x}} f = (\mathrm{d}^R_{\mathbf{x}} f)1$$

ONNX: Open Neural Network Exchange

ONNX is an open exchange format to represent deep learning models. Here is some of an ONXX graph:



Deep Learning: Differentiable Programming Languages

- Deep Learning Graphical Frameworks
  - *Caffe*,*CNTK*, *MXNet*, *Theano*, *TensorFlow*, ...

  TF graphs can have conditionals, iterations, and function calls.
- Automatic Differentiation (Dates back to 1965!)
  - *Autograd* which works as a Python package, adding a first-class gradient operation.
  - Similarly: *Python/TF Eager mode*, *Gluon*, *PyTorch*. Also *F#/Diffsharp*.
  - *VLAD*, a functional language with first-class forward and reverse differentiation (Pearlmutter, Siskind).
- Foundational studies
  - *Differential Lambda Calculi* (Ehrhard, Regnier,../ Manzyuk);
  - *Language for Diff. Functions* (Edalat, Gianantonio).
  - Differential/Tangent Categories (Blute, Cockett,...).
- Functional Programming
  - *Efficient Differentiable Programming in a Functional Array-Processing Language* (Shaikhha, Fitzgibbon et al)
  - *Demystifying Differentiable Programming: Shift/Reset the Penultimate Backpropagator* (Wang et al)

Core differentiable programming language desiderata

- As many programmable functions $f : T \to U$ differentiable as possible, for as many types T, U as possible.
- A gradient operation, more generally, a reverse derivative one; even higher-order ($=$ iterated) derivatives.
- Tensors (aka multidimensional arrays). These have ranks k and shapes $\langle d_0, \ldots, d_{k-1} \rangle$. The set of such real tensors is:

$$\mathbb{R}^{[d_0] \times \ldots \times [d_{k-1}]}$$

- Execution:
  - Learning: optimising neural net parameters against data.
  - Inference: using optimised neural nets.

How are we going to do prog language theory?

- Study a small functional programming language with relevant features:
  - Products of reals as datatypes, but:
  - No tensor datatypes ($\exists$ APL + 21 other array languages; functional programming: Steuwer et al; Gibbons; Haskell).
  - Reverse differentiation as a language primitive.
  - Control structures: conditionals/loops/recursion.
  - More, later.
- Give it a semantics.
- Use the semantics to justify an operational semantics including the differentiation constructs.
- We also have source code transformations eliminating all differentiation constructs, not given here, but summarised.

Previous foundational work

Erhard and Regnier's differential lambda calculus.

- I originally thought this was the way to go.
- It is a typed lambda calculus with product and function types and (forward) differentiation as a primitive.
- It is based on a general notion of a differential category (which has linear features - tensors).
- Example: convenient vector spaces of Frölicher (other examples exist too).
- Main issue: does not support partial functions
- There is, however, a non-higher order notion of a differential restriction category (Cockett et al) which has smooth partial functions over powers of the reals as a model.

Previous work

Automatic (aka algorithmic) differentiation

- Given a program, produce a program that calculates its derivative.
- Originally for scientific computing, not machine learning.
- Huge literature + large community: `www.autodiff.org`
- Very concerned with efficiency.
- As far as I could find out, largely not focused on semantics and its associated language theory — the focus of this talk — though there is functional programming work (VLAD).

References

A simple automatic derivative evaluation program, Wengert, 1964.

Compiling fast partial derivatives of functions given by Algorithms, Speelpenning, 1980.

Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation Griewank & Walther,

Automatic Differentiation in Machine Learning, Baydin et al, 2017.

A minilanguage: syntax

- Types

$$T \quad ::= \quad \mathrm{real} \mid \mathrm{unit} \mid T \times U$$

- Terms

$$\begin{array}{lll} M & ::= & x \mid r \;\; (r \in \mathbb{R}) \mid M + N \mid \mathrm{op}(M) \mid \\ & & M.\mathtt{rd}_L(x : T.\, N) \mid \\ & & \mathtt{let}\; x : T \;=\; M \;\mathtt{in}\; N \mid \\ & & * \mid \langle M, N \rangle \mid \mathrm{fst}(M) \mid \mathrm{snd}(M) \mid \\ & & \mathtt{if}\; B \;\mathtt{then}\; M \;\mathtt{else}\; N \mid \\ & & \mathtt{letrec}\; f(x : T) : U \;=\; M \;\mathtt{in}\; N \mid f(M) \end{array}$$

- Boolean terms

$$B \quad ::= \quad \mathrm{true} \mid \mathrm{false} \mid \mathrm{P}(M)$$

Minilanguage: Typing

- (Ordinary) Environments

$$\Gamma = x_0 : T_0, \ldots, x_{n-1} : T_{n-1}$$

- Function Environments

$$\Phi = f_0 : T_0 \to U_0, \ldots, f_{n-1} : T_{n-1} \to U_{n-1}$$

- Judgements

$$\Phi \mid \Gamma \vdash M : T \qquad \Phi \mid \Gamma \vdash B$$

Typing (cntnd)

- Operations

$$\frac{\Phi \mid \Gamma \vdash M : T}{\Phi \mid \Gamma \vdash \mathrm{op}(M) : U} \quad (\mathrm{op} : T \to U)$$

- Reverse derivatives

$$\frac{\Phi \mid \Gamma \vdash L : T \qquad \Phi \mid \Gamma[x : T] \vdash N : U \qquad \Phi \mid \Gamma \vdash M : U}{\Phi \mid \Gamma \vdash M.\mathtt{rd}_L(x : T.\, N) : T}$$

Differentiating sequences of operations

Consider differentiating $k(x) = h(g(f(x)))$ at $x = a$.

- Trace (or tape) method
  1. Compute the trace, the list
  $$[a, b, c] = [a, f(a), g(f(a))]$$
  2. Using the trace, play the tape $h(g(f(x))$ by applying the chain rule:
  $$k'(c) = h'(c) \cdot g'(b) \cdot f'(a)$$
- Source code transformation (SCT)
  1. Using the chain rule, transform the code to
  $$M = \begin{array}{l} \textbf{let } y = f(x) \textbf{ in} \\ \textbf{let } z = g(y) \textbf{ in } h'(z) \cdot g'(y) \cdot f'(x) \end{array}$$
  2. Evaluate the transformed code with $x = a$.

Much of the automatic differentiation literature considers how to do reverse-mode differentiation efficiently, eg first translating to A-normal form, produces PL versions of the backprop algorithm (see: Griewank, *Who Invented the Reverse Mode of Differentiation?*, 2012)

Differentiating conditionals

Consider:

$$h(x) \;=\; \text{if } b(x) \text{ then } f(x) \text{ else } g(x)$$

The rule people use:

$$\frac{\mathrm{d}h}{\mathrm{d}x} \;=\; \text{if } b(x) \text{ then } \frac{\mathrm{d}f}{\mathrm{d}x} \text{ else } \frac{\mathrm{d}g}{\mathrm{d}x}$$

However consider:

$$h(x) \;=\; \text{if } x = 0 \text{ then } -x \text{ else } x$$

Have $h(x) \;=\; x$, so

$$\frac{\mathrm{d}h}{\mathrm{d}x} \;=\; 1$$

But rule gives

$$\frac{\mathrm{d}h}{\mathrm{d}x} \;=\; \text{if } x = 0 \text{ then } -1 \text{ else } 1$$

Another example:

$$\mathrm{ReLU}(x) \;=\; \text{if } x \le 0 \text{ then } 0 \text{ else } x$$

A way around the difficulty

- Note $b : \mathbb{R} \to \mathbb{T}$,
- Switch to *continuous partial* $b : \mathbb{R} \rightharpoonup \mathbb{T}$, meaning that $b^{-1}(tt)$ and $b^{-1}(ff)$ are open (eg $(-\infty, 0)$ and $(0, \infty)$).
- Write $f : \mathbb{R} \rightharpoonup \mathbb{R}$ to mean that f is partial, with *open domain of definition*.

Proposition

For continuous $b : \mathbb{R} \rightharpoonup \mathbb{T}$ *and differentiable* $f, g : \mathbb{R} \rightharpoonup \mathbb{R}$ *the conditional*

$$h(x) \quad \simeq \quad \mathrm{if}\ b(x)\ \mathrm{then}\ f(x)\ \mathrm{else}\ g(x)$$

is differentiable and, for all $x \in \mathbb{R}$ *we have:*

$$\frac{\mathrm{d}h}{\mathrm{d}x} \quad \simeq \quad \mathrm{if}\ b(x)\ \mathrm{then}\ \frac{\mathrm{d}f}{\mathrm{d}x}\ \mathrm{else}\ \frac{\mathrm{d}g}{\mathrm{d}x}$$

Reference Thomas Beck, Herbert Fischer, *The if-problem in automatic differentiation*, 1994.

Proof of proposition

Proposition

For continuous $b : \mathbb{R} \rightharpoonup \mathbb{T}$ *and differentiable* $f, g : \mathbb{R} \rightharpoonup \mathbb{R}$ *the conditional*

$$h(x) \quad \simeq \quad \mathrm{if}\ b(x)\ \mathrm{then}\ f(x)\ \mathrm{else}\ g(x)$$

is differentiable and, for all $x \in \mathbb{R}$ *we have:*

$$\frac{\mathrm{d}h}{\mathrm{dx}} \quad \simeq \quad \mathrm{if}\ b(x)\ \mathrm{then}\ \frac{\mathrm{d}f}{\mathrm{dx}}\ \mathrm{else}\ \frac{\mathrm{d}g}{\mathrm{dx}}$$

Proof.

Suppose $b(x) = tt$.

- Then there is an open interval (a, b) containing x such that $b(x') = tt$ for all x' in (a, b).
- So $h(x') \simeq f(x')$ for all $x' \in (a, b)$ (not just x!)
- So h and f have the same derivative at x, if any.

$\square$

Another example: swapping

Consider

$$\mathrm{swap}(x, y) = \mathrm{if}\ x > y\ \mathrm{then}\ (x, y)\ \mathrm{else}\ (y, x)$$

When is

$$\frac{\partial \mathrm{swap}}{\partial x} = \mathrm{if}\ x > y\ \mathrm{then}\ (1, 0)\ \mathrm{else}\ (0, 1)$$

OK?

By which I mean: at what points is $>$ continuous?

Equivalently, what is the maximum continuous restriction of $>$?

How While loops work

- We wish to compute the derivative at x of

$$h(x) \simeq \text{while } b(x) \text{ do } f(x)$$

- Suppose $h(x)\downarrow$, and the computation goes round the loop n times. Then

$$h(x) = f^n(x)$$

  and the rule for this x is:

$$\frac{\mathrm{d}h}{\mathrm{d}x} = \frac{\mathrm{d}f^n}{\mathrm{d}x}$$

- Potential proof assuming b continuous, and f differentiable, even have:

$$h(x') = f^n(x')$$

  for all x' in an open interval containing x.

Computing reverse derivatives of while loops

- Trace method
  - Run the loop (interpreter or compiler) till it terminates, producing a trace, being a sequence of intermediate values.
  - Evaluate the reverse derivative along the tape, here the corresponding iterated loop body, using the chain rule.
- Source code transformation Translate the code to code which consists of two while loops in sequence:
  - The first is the original while loop, but it also keeps copies of "checkpoint" intermediate values, and maintains a loop counter.
  - The second counts down from the final value of the loop counter, computing individual reverse derivatives on the way using the relevant intermediate values.

Vector-valued differentiable functions

- A function $f : \mathbb{R}^m \rightharpoonup \mathbb{R}$ is continuously differentiable if its gradient $\nabla_{\mathbf{x}} f$ exists and is continuous at every $\mathbf{x} \in \mathrm{Dom}(f)$.
- A function $f : \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ is continuously differentiable iff each component $\mathbb{R}^m \rightharpoonup \mathbb{R}$ is.

- Equivalently: A function $f : \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ is continuously differentiable if its Jacobian $\mathrm{J} : \mathbb{R}^m \rightharpoonup \mathrm{Mat}_{m,n}$ exists and is continuous at every point in $\mathrm{Dom}(f)$.
- Equivalently: A function $f : \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ is continuously differentiable if its differential $\mathrm{d} : \mathbb{R}^m \times \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ exists and is continuous at every point in its domain, which is $\mathrm{Dom}(f) \times \mathbb{R}$.

Need continuity to make chain rule work

Ordering partial functions

- Partial functions $\mathbb{R}^m \rightharpoonup \mathbb{R}^n$ with open domain are partially ordered by their graphs:

$$f \leq g \iff f \subseteq g$$

  equivalently:

$$f \leq g \iff \forall \mathbf{x} \in \mathbb{R}^m.\, f(\mathbf{x}) \preceq g(\mathbf{x})$$

- This makes $\mathbb{R}^m \rightharpoonup \mathbb{R}^n$ a cppo with $\bot = \emptyset$ and union of graphs as sup:

$$\bigvee f_n = \bigcup_n f_n$$

- This makes the conditional construction:

$$\mathrm{if} - \mathrm{then} - \mathrm{else} - : (\mathbb{R}^m \rightharpoonup \mathbb{T}) \times (\mathbb{R}^m \rightharpoonup \mathbb{R}^n) \times (\mathbb{R}^m \rightharpoonup \mathbb{R}^n) \to (\mathbb{R}^m \rightharpoonup \mathbb{R}^n)$$

  continuous.

Differentiable functions and ordering and conditionals

- Monotonicity Suppose $f, g : \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ are continuously differentiable. Then:

$$f \leq g \implies \mathrm{d}^R f \leq \mathrm{d}^R g$$

- Continuity Suppose f_n is an increasing sequence of continuously differentiable functions. Then so is its sup, and we have:

$$\mathrm{d}^R \left(\bigvee f_n\right) = \bigvee \mathrm{d}^R(f_n)$$

  so:

$$\mathrm{d}^R_x \left(\bigvee f_n\right)(y) = x' \iff \exists n.\, \mathrm{d}^R_x (f_n)(y) = x'$$

- Conditionals Suppose $b : \mathbb{R}^m \rightharpoonup \mathbb{T}$ is continuous, and $f, g : \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ are continuously differentiable. Then so is their conditional and we have:

$$\mathrm{d}^R(\mathrm{if}\ b\ \mathrm{then}\ f\ \mathrm{else}\ g) = \mathrm{if}\ b\ \mathrm{then}\ \mathrm{d}^R(f)\ \mathrm{else}\ \mathrm{d}^R(g)$$

While loops

- Iterates

$$\begin{aligned} \text{while}_{n+1}\ b \text{ do } f &= \perp \\ \text{while}_{n+1}\ b \text{ do } f &= \text{if } b \text{ then } (\text{while}_n\ b \text{ do } f) \circ f \text{ else id} \end{aligned}$$

- Loops

$$\text{while } b \text{ do } f = \bigvee_n \text{while}_n\ b \text{ do } f$$

Theorem

$$(\text{while}_n\ b \text{ do } f)\mathbf{x}\downarrow \implies \mathrm{d}^R_{\mathbf{x}}(\text{while } b \text{ do } f) = \mathrm{d}^R_{\mathbf{x}}(f^n(x))$$

Loop source code transformation

- A while loop
$$w = \text{while } b \text{ do } f : \mathbb{R}^m \rightharpoonup \mathbb{R}^m$$
has a recursive definition:
$$w \quad = \quad \text{if } b \text{ then } w \circ f \text{ else id}$$
equivalently:
$$w(\mathbf{x}) \simeq \text{if } b(\mathbf{x}) \text{ then } w(f(\mathbf{x})) \text{ else } \mathbf{x}$$
- Reverse differentiating we get:
$$\mathrm{d}^R_{\mathbf{x}} w(\mathbf{y}) \quad \simeq \quad \text{if } b(\mathbf{x}) \text{ then } \mathrm{d}^R_{\mathbf{x}} f(\mathrm{d}^R_{f(\mathbf{x})} w(\mathbf{y})) \text{ else } \mathbf{y}$$
- which suggests making the recursive definition of a function $g : \mathbb{R}^m \times \mathbb{R}^m \rightharpoonup \mathbb{R}^m$ by:
$$g(\mathbf{x}, \mathbf{y}) \quad \simeq \quad \text{if } b(\mathbf{x}) \text{ then } (\mathrm{d}^R f)(\mathbf{x}, g(f(\mathbf{x}), \mathbf{y})) \text{ else } \mathbf{y}$$

Loop source code transformation (cntnd)

Theorem

Suppose $w = \text{while } b \text{ do } f$. *Then* $\mathrm{d}^R w$ *is the least function*

$$g : \mathbb{R}^m \times \mathbb{R}^m \rightharpoonup \mathbb{R}^m$$

st.:

$$g(\mathbf{x}, \mathbf{y}) \quad \simeq \quad \text{if } b(\mathbf{x}) \text{ then } (\mathrm{d}^R f)(\mathbf{x}, g(f(\mathbf{x}), \mathbf{y})) \text{ else } \mathbf{y}$$

Proof.

By induction we have:

$$\mathrm{d}^R(f^{(n)}) = g^{(n)}$$

So:

$$\mathrm{d}^R(f) = \mathrm{d}^R(\bigvee f^{(n)}) = \bigvee \mathrm{d}^R(f^{(n)}) = \bigvee g^{(n)} = g$$

$\square$

Minilanguage reminder

- Types

$$T \quad ::= \quad \mathrm{real} \mid \mathrm{unit} \mid T \times U$$

- Terms

$$\begin{aligned} M \quad ::= \quad & x \mid r \ \ (r \in \mathbb{R}) \mid M + N \mid \mathrm{op}(M) \mid \\ & M.\mathtt{rd}_L(x : T.\, N) \mid \\ & \mathtt{let}\ x : T \ = \ M\ \mathtt{in}\ N \mid \\ & * \mid \langle M, N \rangle \mid \mathrm{fst}(M) \mid \mathrm{snd}(M) \mid \\ & \mathtt{if}\ B\ \mathtt{then}\ M\ \mathtt{else}\ N \mid \\ & \mathtt{letrec}\ f(x : T) : U \ = \ M\ \mathtt{in}\ N \mid f(M) \end{aligned}$$

- Boolean terms

$$B \quad ::= \quad \mathrm{true} \mid \mathrm{false} \mid \mathrm{P}(M)$$

Minilanguage semantics: types

$$
\begin{aligned}
[\![\mathrm{real}]\!] &= \mathbb{R} \\
[\![\mathrm{unit}]\!] &= \mathbb{1} \\
[\![T \times U]\!] &= [\![T]\!] \times [\![U]\!]
\end{aligned}
$$

Flattening types and functions

- Flattening Types

$$\varphi : [\![T]\!] \cong \mathbb{R}^{|T|}$$

where:

$$\begin{array}{lcl} |\mathrm{real}| & = & 1 \\ |\mathrm{unit}| & = & 0 \\ |T \times U| & = & |T| + |U| \end{array}$$

- Flattening Functions

$$\begin{tikzcd} {[\![T]\!]} \arrow[r, "f"] & {[\![U]\!]} \arrow[d, "\varphi"] \\ \mathbb{R}^{|T|} \arrow[u, "\varphi^{-1}"] \arrow[r, "\varphi(f)"'] & \mathbb{R}^{|U|} \end{tikzcd}$$

Smooth functions

- A function $f : \mathbb{R}^m \rightharpoonup \mathbb{R}$ is *smooth* (or of class C^∞) if all its m partial derivatives $\frac{\partial f}{\partial x_i} : \mathbb{R}^m \rightharpoonup \mathbb{R}$ $(i = 1, m)$ are defined on its domain and they too are all smooth.
  (It is of class C^0 if it is continuous, and of class C^{k+1} if all its partial derivatives are defined on its domain and are of class C^k.)
- (Equivalently) A function $f : \mathbb{R}^m \rightharpoonup \mathbb{R}^n$ is *smooth* if, for all $\mathbf{y} \in \mathbb{R}^m$, $\mathrm{d}f(-, \mathbf{y})$ exists and is smooth.
- A function $f : [\![T]\!] \rightharpoonup [\![U]\!]$ is *smooth* if $\varphi(f)$ is smooth.
- We write $\mathcal{S}[[\![T]\!], [\![U]\!]]$ for the collection of all such functions. It forms a subcppo of the continuous such functions.

Semantics of the language

- Operations

$$[\![\mathrm{op}]\!] \in \mathcal{S}[[\![T]\!], [\![U]\!]] \quad (\mathrm{op} : S \to T)$$

- Environments

$$[\![x_0 : T_0, \ldots, x_{n-1} : T_{n-1}]\!] = [\![T_0]\!] \times \ldots [\![T_{n-1}]\!]$$

- Function environments

$$[\![f_0 : T_0 \to U_0, \ldots, f_{n-1} : T_{n-1} \to U_{n-1}]\!] = \mathcal{S}[[\![T_0]\!], [\![U_0]\!]] \times \ldots \times \mathcal{S}[[\![T_{n-1}]\!], [\![U_{n-1}]\!]]$$

- Terms

$$\frac{\Phi \mid \Gamma \vdash M : T}{[\![M]\!] : [\![\Phi]\!] \longrightarrow \mathcal{S}[[\![\Gamma]\!], [\![T]\!]]} \qquad \frac{\Phi \mid \Gamma \vdash B}{[\![B]\!] : [\![\Phi]\!] \longrightarrow \mathcal{C}[[\![\Gamma]\!], \mathbb{T}]}$$

Example denotational semantics

Operations

$$[\![\mathrm{op}(M)]\!](\varphi,\gamma) \simeq [\![\mathrm{op}]\!]([\![M]\!](\varphi,\gamma))$$

Reverse derivatives

$$[\![M.\mathtt{rd}_L(x : T.\, N)]\!](\varphi,\gamma) \quad \simeq \quad \mathrm{d}^R_{[\![L]\!](\varphi,\gamma)}(\lambda a : [\![T]\!].\, [\![N]\!](\varphi,\gamma[a/x]))([\![M]\!](\varphi,\gamma))$$

where for any differentiable $f : [\![T]\!] \rightharpoonup [\![U]\!]$ we set:

$$\mathrm{d}^R(f) = \varphi^{-1}_{T\times U,T}(\mathrm{d}^R(\varphi_{T,U}(f)))$$

Example denotational semantics

Operations

$$[\![\mathrm{op}(M)]\!](\varphi, \gamma) \simeq [\![\mathrm{op}]\!]([\![M]\!](\varphi, \gamma))$$

Reverse derivatives

$$[\![M.\mathtt{rd}_L(x : T.\, N)]\!] \quad \simeq \quad \mathrm{d}^R_{[\![L]\!]}(\lambda a : [\![T]\!].\, [\![N]\!][a/x])[\![M]\!]$$

Operational semantics: basics

- Value Environments Any finite function

$$\gamma : \mathrm{Variables} \to_{\mathrm{fin}} \mathrm{ClosedValues}$$

- Values These are terms $V, W, \ldots$:

$$V \quad ::= \quad x \mid r \; (r \in \mathbb{R}) \mid * \mid \langle V, W \rangle$$

- Boolean values These are terms V_{bool}:

$$V_{\mathrm{bool}} \quad ::= \quad \mathrm{true} \mid \mathrm{false}$$

- Function Environments Any finite function

$$\varphi : \mathrm{FunctionVariables} \to_{\mathrm{fin}} \mathrm{Closures}$$

- Closures These are structures

$$\mathbf{clo}_{\rho,\varphi}(f(x : T) : U.\, M)$$

where:
(1) ρ is a value environment with $\mathrm{FV}(M)\backslash x \subseteq \mathrm{Dom}(\rho)$
(2) φ is a function environment with $\mathrm{FFV}(M)\backslash f \subseteq \mathrm{Dom}(\varphi)$

Evaluation relations

- (Ordinary) Evaluation Relation These relations have the form

$$\varphi \mid \rho \vdash M \Rightarrow V \qquad \varphi \mid \rho \vdash B \Rightarrow V_{\mathrm{bool}}$$

  with V closed.

- Symbolic Evaluation Relation These relations have the form

$$\varphi \mid \rho \vdash M \rightsquigarrow C$$

- Tape terms These are terms $C, D, \ldots$ with no control constructs. More specifically, they contain no: function variables; conditionals; function definitions; or function applications:

$$\begin{array}{rcl} C & ::= & x \mid r\ (r \in \mathbb{R}) \mid C + D \mid \mathrm{op}(C) \mid \\ & & \mathtt{let}\ x : T \ = \ C\ \mathtt{in}\ D \mid \\ & & * \mid \langle C, D \rangle \end{array}$$

Example ordinary evaluation rules

Operations

$$\frac{\varphi \mid \rho \vdash M \Rightarrow V}{\varphi \mid \rho \vdash \mathrm{op}(M) \Rightarrow W} \quad (\mathrm{ev}(\mathrm{op}, V) \simeq W)$$

Local Definitions

$$\frac{\varphi \mid \rho \vdash M \Rightarrow V \quad \varphi \mid \rho[V/x] \vdash N \Rightarrow W}{\varphi \mid \rho \vdash \mathtt{let}\ x : T \ = \ M\ \mathtt{in}\ N \Rightarrow W}$$

Conditionals

$$\frac{\varphi \mid \rho \vdash B \Rightarrow \mathrm{true} \quad \varphi \mid \rho \vdash M \Rightarrow W}{\varphi \mid \rho \vdash \mathtt{if}\ B\ \mathtt{then}\ M\ \mathtt{else}\ N \Rightarrow W}$$

Reverse Derivatives

$$\frac{\varphi \mid \rho \vdash M.\mathtt{rd}_L(x : T.\, N) \rightsquigarrow C \quad \varphi \mid \rho \vdash C \Rightarrow V}{\varphi \mid \rho \vdash M.\mathtt{rd}_L(x : T.\, N) \Rightarrow V}$$

Symbolic evaluation rules

Variables

$$\varphi \mid \rho \vdash x \rightsquigarrow x$$

Operations

$$\frac{\varphi \mid \rho \vdash M \rightsquigarrow C}{\varphi \mid \rho \vdash \mathrm{op}(M) \rightsquigarrow \mathrm{op}(C)}$$

Local Definitions

$$\frac{\varphi \mid \rho \vdash M \rightsquigarrow C \quad \varphi \mid \rho \vdash C \Rightarrow V \quad \varphi \mid \rho[V/x] \vdash N \rightsquigarrow D}{\varphi \mid \rho \vdash \texttt{let}\ x : T\ =\ M\ \texttt{in}\ N \rightsquigarrow \texttt{let}\ x : T\ =\ C\ \texttt{in}\ D}$$

Conditionals

$$\frac{\varphi \mid \rho \vdash B \Rightarrow \mathrm{true} \quad \varphi \mid \rho \vdash M \rightsquigarrow C}{\varphi \mid \rho \vdash \texttt{if}\ B\ \texttt{then}\ M\ \texttt{else}\ N \rightsquigarrow C}$$

Symbolic evaluation rules (cntnd)

Function Definition

$$\frac{\varphi[\mathbf{clo}_{\rho,\varphi}(f(x:T):U.\,M)/f] \mid \rho \vdash N \rightsquigarrow C}{\varphi \mid \rho \vdash \mathtt{letrec}\ f(x:T):U \ = \ M\ \mathtt{in}\ N \rightsquigarrow C}$$

Function Application

$$\frac{\varphi \mid \rho \vdash M \rightsquigarrow C \quad \varphi \mid \rho \vdash C \Rightarrow V \quad \varphi'[\varphi(f)/f] \mid \rho'[V/x] \vdash N \rightsquigarrow D}{\varphi \mid \rho \vdash f(M) \rightsquigarrow \mathtt{let}\ x:T \ = \ C\ \mathtt{in}\ D\rho'}$$

$$(\varphi(f) = \mathbf{clo}_{\rho',\varphi'}(f(x:T):U.\,N))$$

Reverse Derivatives

$$\frac{\varphi \mid \rho \vdash L \rightsquigarrow C \quad \varphi \mid \rho \vdash M \rightsquigarrow D \quad \varphi \mid \rho \vdash C \Rightarrow V \quad \varphi \mid \rho[V/x] \vdash N \rightsquigarrow E}{\varphi \mid \rho \vdash M.\mathtt{rd}_L(x:T.\,N) \rightsquigarrow \mathbf{let}\ \overline{x}:T, \overline{y}:U \ = \ C, D\ \mathbf{in}\ \overline{y}.\mathcal{R}_{\overline{x}}(x:T.\,E)}$$

$$(\overline{x}, \overline{y} \notin \mathrm{Dom}(\rho), \Gamma_\rho \vdash E:U)$$

Symbolic differentiation: $W.\mathcal{R}_V(x\colon T.\, C)$

$$W.\mathcal{R}_V(x\colon T.\, y) = \begin{cases} W & (y = x) \\ 0_T & (y \neq x) \end{cases}$$

$$W.\mathcal{R}_V(x\colon T.\, D + E) = W.\mathcal{R}_V(x\colon T.\, D) + W.\mathcal{R}_V(x\colon T.\, E)$$

$$W.\mathcal{R}_V(x\colon T.\, \mathrm{op}(D[x])) = W.\mathrm{op}^r(D[V]).\mathcal{R}_V(x\colon T.\, D)$$

$$W.\mathcal{R}_V(x\colon T.\, \begin{array}{l}\textbf{let } y\colon U \;=\; C[x] \\ \textbf{in } D[x, y])\end{array} = \begin{array}{l}\textbf{let } \overline{y}\colon U \;=\; C[V] \textbf{ in} \\ \textbf{let } \overline{z}\colon T \times U \;= \\ \quad W.\mathcal{R}_{\langle V, \overline{y}\rangle}(z\colon T \times U.\, D[\mathrm{fst}(z), \mathrm{snd}(z)]) \\ \textbf{in } \mathrm{fst}(\overline{z}) + \mathrm{snd}(\overline{z}).\mathcal{R}_V(x\colon X.\, C[x])\end{array}$$

$$W.\mathcal{R}_V(x\colon T.\, \langle D, E\rangle) = \begin{array}{l}\mathrm{fst}(W).\mathcal{R}_V(x\colon T.\, D) + \\ \quad \mathrm{snd}(W).\mathcal{R}_V(x\colon T.\, E)\end{array}$$

$$W.\mathcal{R}_V(x\colon T.\, \mathrm{fst}(D[x])) = \begin{array}{l}\textbf{let } \overline{x}\colon T \;=\; D[V] \textbf{ in} \\ \quad \langle W, 0\rangle\, .\mathcal{R}_V(x\colon T.\, D)\end{array} \quad (\overline{x} \notin \mathrm{FV}(D))$$

Typing environments

We give rules for judgments

$$\rho : \Gamma \qquad \mathrm{Cl} : T \to U \qquad \varphi : \Phi$$

as follows:

$$\frac{V_i : T_i \quad (i = 0, n-1)}{\{x_0 \mapsto V_0, \ldots, x_{n-1} \mapsto V_{n-1}\} : x_0 : T_0, \ldots, x_{n-1} : T_{n-1}}$$

$$\frac{\mathrm{Cl}_i : T_i \to U_i \quad (i = 0, n-1)}{\{f_0 \mapsto \mathrm{Cl}_0, \ldots, f_{n-1} \mapsto \mathrm{Cl}_{n-1}\} : f_0 : T_0 \to U_0, \ldots, f_{n-1} : T_{n-1} \to U_{n-1}}$$

$$\frac{\varphi' : \Phi \quad \rho' : \Gamma \quad \Phi, \Gamma[T/x] \vdash M : U}{\mathbf{clo}_{\rho,\varphi}(f(x : T) : U.\, M) : T \to U} \qquad \begin{array}{c}(\varphi' = \varphi \restriction \mathrm{FFV}(M)\backslash f, \\ \rho' = \rho \restriction \mathrm{FV}(M)\backslash x)\end{array}$$

Correctness theorems

Theorem (Formal reverse-mode differentiation correctness)

Suppose $\Gamma[x : T] \vdash E : U$, $\Gamma \vdash C : T$, *and* $\Gamma \vdash D : U$ *(and so* $\Gamma \vdash D.\mathtt{rd}_C(x : T.\, E) : T$*). Then, for any* $\gamma \in [\![\Gamma]\!]$*, we have:*

$$[\![D.\mathtt{rd}_C(x : T.\, E)]\!](\gamma) \simeq [\![D.\mathcal{R}_C(x : T.\, E)]\!](\gamma)$$

Correctness theorems (cntnd)

Two conditions:

- NGV No recursive function definitions in M have global free variables.
- NGFD No recursive function definitions in M contain the function variable within a derivative expression occurring within the function body.

Theorem (Operational correctness)

1. ***Operational semantics.*** *Suppose* $\Phi \mid \Gamma \vdash M : T$, $\varphi : \Phi$, *and* $\rho : \Gamma$. *Then:*

$$\varphi \mid \rho \vdash M \Rightarrow V \implies [\![M]\!][\![\varphi]\!][\![\rho]\!] = [\![V]\!]$$

2. ***Symbolic operational semantics.*** *Suppose* $\Phi \mid \Gamma \vdash M : T$, $\Phi \mid \Gamma \vdash C : T$, $\varphi : \Phi$, *and* $\rho : \Gamma$. *Then:*

$$\varphi \mid \rho \vdash M \rightsquigarrow C \implies \begin{array}{l} \exists\, O \subseteq_{open} [\![\Gamma]\!].\ [\![\rho]\!] \in O \ \wedge \\ \quad \forall \gamma \in O.\ [\![M]\!][\![\varphi]\!]\gamma \simeq [\![C]\!]\gamma \end{array}$$

Operational completeness

Theorem (Operational completeness)

The following hold:

1. ***Operational semantics.*** *Suppose* $\Phi \mid \Gamma \vdash M : T$, $\varphi{:}\Phi$, *and* $\rho{:}\Gamma$. *Then:*

$$\llbracket M \rrbracket \llbracket \varphi \rrbracket \llbracket \rho \rrbracket = \llbracket V \rrbracket \implies \varphi \mid \rho \vdash M \Rightarrow V$$

2. ***Symbolic operational semantics.*** *Suppose* $\Phi \mid \Gamma \vdash M : T$, $\varphi{:}\Phi$, *and* $\rho{:}\Gamma$. *Then:*

$$\varphi \mid \rho \vdash M \Rightarrow V \implies \exists C.\, \varphi \mid \rho \vdash M \rightsquigarrow C$$

Derivative Elimination Theorem

Theorem

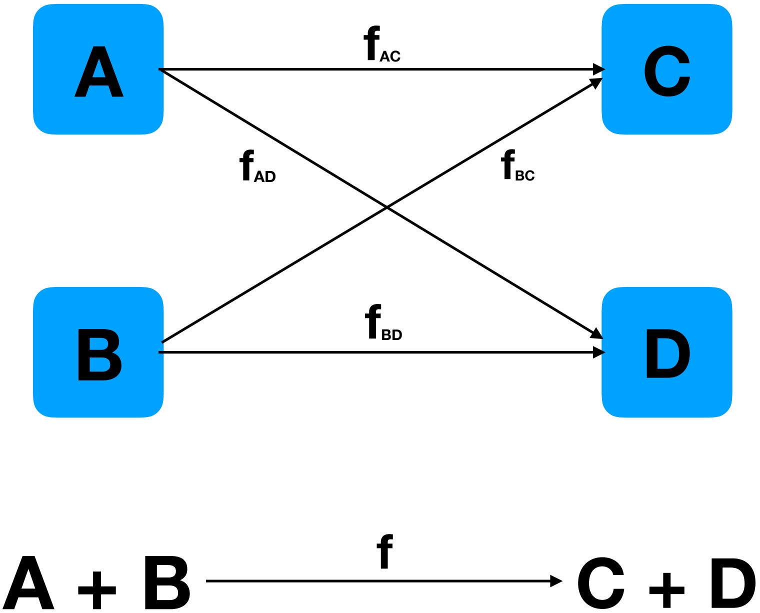
Let M be a closed well-typed NGV and NGFD term over a given alphabet of function variables.
Then there is a unique derivative-free term D, possibly containing additional primed function variables, such that:

$$M \triangleright D$$

Further:

$$[\![M]\!] = [\![D]\!]$$

Decomposing a partial function into its components



The reverse derivative of a decomposed function

- We want:

$$\mathrm{d}^R f : (A+B) \times (C+D) \rightharpoonup A+B$$

- Identifying f with its composition with the distributive expansion of its domain, look for

$$\mathrm{d}^R f : (A \times C) + (A \times D) + (B \times C) + (B \times D) \rightharpoonup A+B$$

- Given by:

$$(\mathrm{d}^R f)_{(A\times C)A} = \mathrm{d}^R f_{AC} \qquad (\mathrm{d}^R f)_{(A\times D)A} = \mathrm{d}^R f_{AD}$$

$$(\mathrm{d}^R f)_{(B\times C)B} = \mathrm{d}^R f_{BC} \qquad (\mathrm{d}^R f)_{(B\times D)B} = \mathrm{d}^R f_{BD}$$

  and taking the other components such as

$$(\mathrm{d}^R f)_{(A\times C)B}$$

  to be undefined.

Sums: injections

- For

$$\mathrm{inl} : A \rightharpoonup A + B$$

- Wish:

$$\mathrm{d}^R(\mathrm{inl}) : A \times (A + B) \rightharpoonup A$$

- Define

$$\mathrm{d}^R_x(\mathrm{inl})(z) \simeq \begin{cases} y & (z = \mathrm{inl}(y)) \\ \downarrow & (\text{otherwise}) \end{cases}$$

- Differentiation equivalence

$$Q.\mathtt{rd}_P(x : T.\,\mathtt{inl}(M)) \quad = \quad \begin{array}{l} \mathtt{let}\ x : T, y : U + V \ = \ P, Q\ \mathtt{in} \\ \mathtt{cases}\ y\ \mathtt{of} \\ \ \mathtt{inl}(u : U) \Rightarrow u.\mathtt{rd}_x(x : T.\,M) \mid \\ \ \mathtt{inr}(v : V) \Rightarrow \mathtt{UNDEF} \end{array}$$

Sums: cotupling

- For

$$\frac{f : A \rightharpoonup C \quad g : B \rightharpoonup C}{[f, g] : A + B \rightharpoonup C}$$

- Wish

$$\mathrm{d}^R([f, g]) : (A + B) \times C \rightharpoonup A + B$$

- Define

$$\mathrm{d}^R_z([f, g])(u) = \begin{cases} \mathrm{inl}((\mathrm{d}^R_x f)u) & (z = \mathrm{inl}(x)) \\ \mathrm{inr}((\mathrm{d}^R_y f)u) & (z = \mathrm{inr}(y)) \end{cases}$$

Sums: cotupling (cntnd)

Differentiation equivalence

$$
\begin{array}{l}
Q.\mathtt{rd}_P(x : T.\,\mathtt{cases}\ L[x]\ \mathtt{of} \\
\qquad\qquad\quad \mathtt{inl}(u : U) \Rightarrow M[x,u] \mid \\
\qquad\qquad\quad \mathtt{inr}(v : V) \Rightarrow N[x,v])
\end{array}
$$

$$=$$

$$
\begin{array}{l}
\mathtt{let}\ x : T, z : W \ = \ P, Q\ \mathtt{in} \\
\mathtt{cases}\ L[x]\ \mathtt{of} \\
\ \mathtt{inl}(u : U) \Rightarrow \mathtt{let}\ x' : T, u' : U \\
\qquad\qquad\qquad\quad = \ z.\mathtt{rd}_{x,u}(x : T, u : U.\,M[x,u]) \\
\qquad\qquad\qquad\quad \mathtt{in}\ x' + \mathrm{inl}(u').\mathtt{rd}_x(x : T.\,L[x]) \mid \\
\ \mathtt{inr}(v : V) \Rightarrow \ldots
\end{array}
$$

Symbolic operational semantics for sums

An abbreviation

$$\mathtt{castl}_{T,U}(M) \equiv \mathtt{cases}\ M\ \mathtt{of}\ x : T \Rightarrow x \mid y : U \Rightarrow \mathtt{UNDEF}$$

Redex

$$\frac{\varphi \mid \rho \vdash V \Rightarrow \mathtt{inl}_{T,U}(W) \qquad \varphi \mid \rho[W/x] \vdash M \rightsquigarrow C}{\begin{array}{r}\varphi \mid \rho \vdash \mathtt{cases}\ V\ \mathtt{of}\ x : T \Rightarrow M \mid y : U \Rightarrow N \rightsquigarrow \\ \mathbf{let}\ x : T \ = \ \mathtt{castl}_{T,U}(V)\ \mathbf{in}\ C\end{array}}$$

Differentiating functions on lists of reals

Is

$$\mathit{reverse} : \mathrm{List}(\mathbb{R}) \to \mathrm{List}(\mathbb{R})$$

differentiable?
It can be considered as a collection of functions

$$\mathit{reverse}_{n,n} : \mathrm{List}(\mathbb{R}, n) \to \mathrm{List}(\mathbb{R}, n)$$

As

$$\mathrm{List}(\mathbb{R}, n) = \mathbb{R}^n$$

we say it is differentiable everywhere as each of its *components* $\mathit{reverse}_{n,n}$ is.

Example

At which lists is

$$\mathrm{sort} : \mathrm{List}(\mathbb{R}) \to \mathrm{List}(\mathbb{R})$$

differentiable?

Differentiating functions between lists, in general

Any function:

$$f : \mathrm{List}(\mathbb{R}) \to \mathrm{List}(\mathbb{R})$$

decomposes into a collection of *components*

$$f_{nm} : \mathrm{List}(\mathbb{R}, n) \rightharpoonup \mathrm{List}(\mathbb{R}, m)$$

where

$$f_{nm}(l) \simeq \begin{cases} f(l) & (f(l) \text{ has length } m) \\ \downarrow & (\text{otherwise}) \end{cases} \qquad (l \in \mathrm{List}(\mathbb{R}, n))$$

- We say f is *differentiable at l* if f_{nm} is at l (where $n = |l|$, and $m = |f(l)|$).
- We say f is *differentiable with open domain* if, and only if, each of its components f_{nm} is.

Reverse derivatives of functions on lists

- As
$$f_{nm} : \mathrm{List}(\mathbb{R}, n) \rightharpoonup \mathrm{List}(\mathbb{R}, m)$$
have
$$\mathrm{d}^R f_{n,m} : \mathrm{List}(\mathbb{R}, n) \times \mathrm{List}(\mathbb{R}, m) \rightharpoonup \mathrm{List}(\mathbb{R}, n)$$
- So might expect a dependent type
$$\mathrm{d}^R f : \prod_{l \in \mathrm{List}(\mathbb{R}, n)} \mathrm{List}(\mathbb{R}, |f(l)|) \rightharpoonup \mathrm{List}(\mathbb{R}, n)$$
- but we instead use a simple type
$$\mathrm{d}^R f : \mathrm{List}(\mathbb{R}) \times \mathrm{List}(\mathbb{R}) \rightharpoonup \mathrm{List}(\mathbb{R})$$

Differentiable shapely datatypes

Given a container, viz:

- A set S of shapes.
- For each shape $s \in S$, a finite set P_s of places.

Shapely differentiable datatypes have the form

$$D_{S,P} = \sum_{s \in S} \mathbb{R}^{P_s} = \{\langle s, \mathbf{x} \rangle \mid s \in S,\ \mathbf{x} : P_s \to \mathbb{R}\}$$

Examples of differentiable shapely datatypes

- Sets
$$X \cong \sum_{s \,\in\, X} \mathbb{R}^{\emptyset}$$
- Finite products of $\mathbb{R}$:
$$\mathbb{R}^n \cong \sum_{s \,\in\, \{*\}} \mathbb{R}^{[n]}$$
- Lists of reals
$$\mathrm{List}(\mathbb{R}) \cong \sum_{n \,\in\, \mathbb{N}} \mathbb{R}^{[n]}$$
- Tensors of rank $k \geq 0$ of reals
$$\mathrm{Tensor}_k(\mathbb{R}) \cong \sum_{\langle d_0,\ldots,d_{k-1}\rangle \,\in\, \mathbb{N}^k_{>0}} \mathbb{R}^{[d_0]\times\ldots\times[d_{k-1}]}$$
- Binary trees of reals
$$\mathrm{BinaryTrees}(\mathbb{R}) \cong \sum_{s \,\in\, \mathsf{BinaryTrees}} \mathbb{R}^{\mathsf{Branches}(s)}$$

Shapely differentiable datatypes are manifolds

A differentiable manifold of varying dimension is:

- A Hausdorff topological space X, plus
- an atlas on X, ie a collection of open subsets U_i covering X and each with a specified homeomorphism $U_i \xrightarrow{\varphi_i} \mathbb{R}^n$ (a coordinate chart) to an open subset of some $\mathbb{R}^n$,
- subject to some axioms.

For shapely differentiable datatypes we have the charts:

$$U_s = \{s\} \times \mathbb{R}^{P_s} \longrightarrow \mathbb{R}^{|P_s|} \qquad (s \in S)$$

- This connects shapely differentiable datatypes with standard notions of differentiable functions.
- Manifolds figure commonly in learning theory. Pymanopt, Townsend et al, 2016
- Should (a suitable version of) manifolds be datatypes of differentiable programming languages? Pearlmutter, Automatic Differentiation: History and Headroom, NIPS Autodiff Workshop, 2016.

Future work

- More language features in either external or internal mode, according to whether they cannot or can be differentiated. Examples:
    - Higher-order functions. External: Autograd; Internal cf. convenient vector spaces.
    - Effects: exceptions, global state, I/O. All available with shapely differential datatypes.
    - Probability. Current work restricted to a graphical model with a mixture of discrete distributions and those with a density.
- Connect traditional semantic frameworks with differentiation:
    - Domain theory: could do streams and higher-order functions
    - Metric spaces: could relate ideal computation with reals with approximate computation; differentiation of iteration scheme computations.

Future work (cntnd)

- Make less sweeping, more realistic, assumptions about smoothness
  - Work with functions (and hence programs) in smoothness classes C^k.
  - Allow weaker forms of differentiability, eg some form of Clarke generalised derivative, as in work of Edalat and Gianantonio (who use domain theory).
- Look at theory of work in automatic differentiation, to establish correctness of their techniques for efficiency.
- Investigate use of dependent types to track shape analysis of tensor computations.
- Consider how to program with manifolds as types.