Neural Networks and Deep Learning
Automatic Differentiation

Nicholas Dronen

Department of Computer Science
dronen@colorado.edu

January 30, 2019

University of Colorado **Boulder**

Questions?

Background

Some Methods of Differentiation

Automatic Differentiation (AD)
- Forward-Mode AD
- Reverse-Mode AD
- Implementations

Computing Derivatives of Composite Expressions

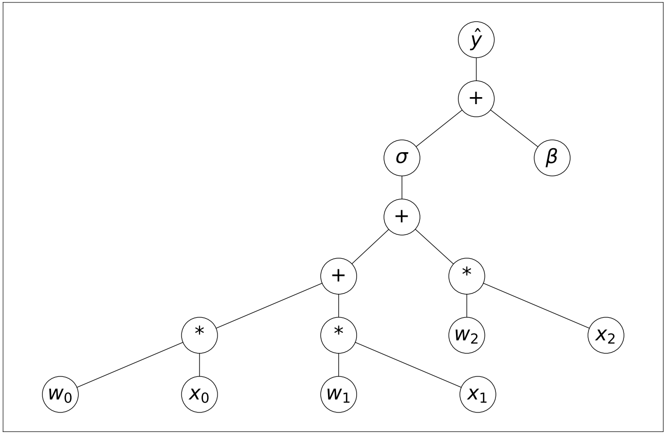
Recall from the previous lecture that we defined an artificial neuron that computes

$$\hat{y} = \sigma\left(\sum_{i=0}^{k} w_i x_i\right) + b.$$

To optimize neurons or networks of neurons, we must compute derivatives (gradients, if multidimensional) of the constituent terms of expressions like this to optimize neural networks.

Spot the error in this slide.

Computation Graphs



Chain Rule

Given functions f and g, the derivative of their composition, $f \circ g$, is $f \circ g$ is

$$(f' \circ g) \cdot g'.$$

Equivalently, given variables z, y, and x, where z depends on y and y on x,

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}.$$

A computational graph is a directed acyclic graph (DAG) of function compositions.

Jacobian

- A vector-valued function is a mapping from $f : R^n \rightarrow R^m$.
- The Jacobian operator is a generalization of the derivative operator to vector-valued functions.
- Jacobian matrix J captures the rate of change of each component of y with respect to each component of input variable x.

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & . & . & \frac{\partial y_m}{\partial x_1} \\ . & . & . & . \\ . & . & . & . \\ \frac{\partial y_1}{\partial x_n} & . & . & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Finite Differences

- Numerical differentiation is the finite difference approximation of derivatives using original function evaluated at some sample points.

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x + he_i) - f(x)}{h}$$

- Pros - Not complicated to implement
- Cons -
  - » Numerical approximations of derivatives are inherently ill-conditioned and unstable, due to introduction of -
    - Truncation errors (caused by chosen value of x).
    - Round-off errors (caused by limited precision of computations).
  - » $O(n)$ computation complexity for a gradient in n dimensions.

Symbolic Differentiation

- Symbolic differentiation is the automatic manipulation of expressions for obtaining derivative expressions, carried out by applying transformations representing rules of differentiation such as -

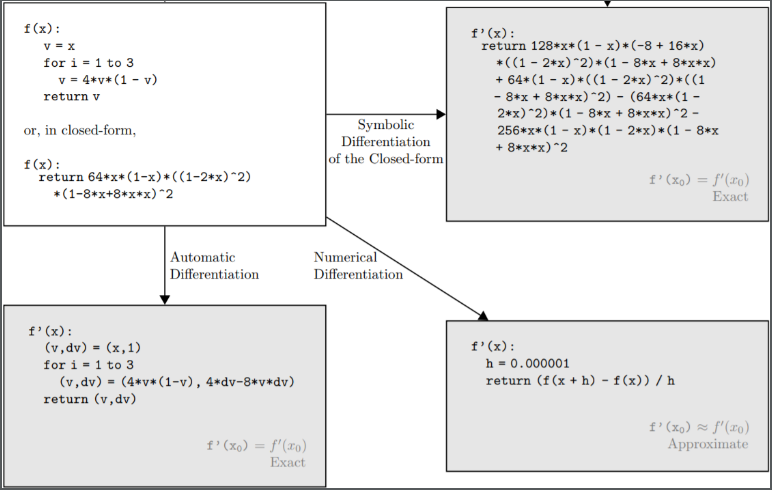
$$\frac{\partial}{\partial x}(f(x)+g(x)) = \frac{\partial}{\partial x}f(x) + \frac{\partial}{\partial x}g(x)$$

- Pros - Can give valuable insight into structure of problem domain.
- Cons -
  - » Careless symbolic differentiation can produce exponentially large symbolic expressions which take correspondingly long time to evaluate - *expression swell*
  - » Limited expressivity

Intuition

- The insight behind AD is to apply symbolic differentiation at the elementary operation level and keep intermediate numerical results.
- AD can differentiate not only closed-form expressions, but also algorithms making use of control flow such as branching, loops, recursion, and procedure calls.

Overview



```
f(x):
  v = x
  for i = 1 to 3
    v = 4*v*(1 - v)
  return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

```
f'(x):
  return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

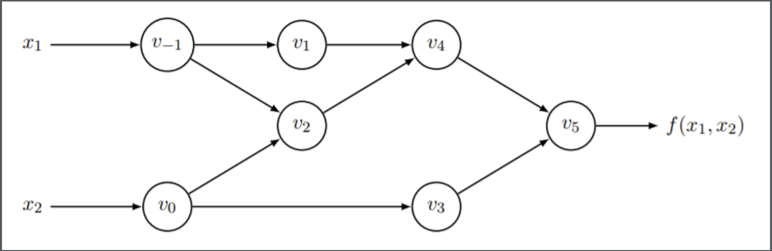
```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

Symbolic Differentiation of the Closed-form
Automatic Differentiation
Numerical Differentiation
f'(x₀) = f′(x₀) Exact
f'(x₀) = f′(x₀) Exact
f'(x₀) ≈ f′(x₀) Approximate

Figure taken from Automatic differentiation in machine learning: A survey

Example

$$f(x_1, x_2) = ln(x_1) + x_1 * x_2 - sin(x2)$$



Computational graph for $f(x_1, x_2)$, Figure taken from Automatic differentiation in machine learning: A survey

$$v_{-1} = x_1 \qquad v_0 = x_2$$

$$v_1 = ln(v_{-1}) \qquad v_2 = v_{-1}v_0$$

$$v_3 = sin(v_0) \qquad v_4 = v_1 + v_2$$

$$v_5 = v_4 - v_3 \qquad y = v_5$$

Intuition

- AD in forward mode is conceptually easy.
- Method
  - » Apply chain rule to each elementary operation in the forward primal trace and generate corresponding tangent (derivative) trace.
  - » Evaluating primals in lockstep with corresponding tangents gives the required derivative in the final variable.
- For cases $f : R^n \rightarrow R^m$, where $n \gg m$, Forward-mode becomes computationally expensive.

Example

Forward Primal Trace

$$\begin{aligned}
v_{-1} &= x_1 && = 2 \\
v_0 &= x_2 && = 5 \\
\hline
v_1 &= ln(v_{-1}) && = ln2 \\
v_2 &= v_{-1} && = 10 \\
v_3 &= sin(v_0) && = sin5 \\
v_4 &= v_1 + v_2 && = 10.693 \\
v_5 &= v_4 - v_3 && = 11.652 \\
\hline
y &= v_5 && = 11.652
\end{aligned}$$

Forward Tangent Trace

$$\begin{aligned}
\dot{v_{-1}} &= \dot{x_1} && = 1 \\
\dot{v_0} &= x_2 && = 0 \\
\hline
\dot{v_1} &= \dot{v_{-1}}/v_{-1} && = 1/2 \\
\dot{v_2} &= \dot{v_{-1}}v_0 + \dot{v_0}v_{-1} && = 5 \\
\dot{v_3} &= \dot{v_0}cos(v_0) && = 0 \\
\dot{v_4} &= \dot{v_1} + \dot{v_2} && = 5.5 \\
\dot{v_5} &= \dot{v_4} - \dot{v_3} && = 5.5 \\
\hline
\dot{y} &= \dot{v_5} && = 5.5
\end{aligned}$$

Forward mode AD example to compute $\frac{\partial y}{\partial x_1}$.The original evaluation of primals on the left is augmented by tangent operations on the right.

Complexity

- Forward mode is efficient and straightforward for functions $f : R \to R^m$, as all derivatives of $\frac{\partial y_i}{\partial x}$ can be computed in one forward pass.
- For $f : R^n \to R$, forward mode requires n evaluations to compute the gradient

$$\nabla f = \left(\frac{\partial y}{\partial x_1}, ..., \frac{\partial y}{\partial x_n} \right)$$

  which corresponds to a $1 \times n$ Jacobian matrix built one column at a time.

- AD in reverse-mode corresponds to a generalized back propagation algorithm, in that it propagates derivatives backward from a given output.
- This is done by complementing each intermediate variable v_i with an adjoint $\frac{\partial y_j}{\partial v_i}$, which represents the sensitivity of a considered output y_j w.r.t changes in v_i.
- Method -
  - » Original function code is run forward, populating intermediate variables v_i and recording dependencies in the computational graph in a book-keeping procedure
  - » Derivatives are calculated by propagating adjoints in reverse, from outputs to inputs
- For cases $f : R^n \rightarrow R^m$, where $n \gg m$, reverse mode AD performs well computationally.

Example

Forward Primal Trace

$$
\begin{array}{lll}
v_{-1} & = x_1 & = 2 \\
v_0 & = x_2 & = 5 \\
\hline
v_1 & = ln(v_{-1}) & = ln2 \\
v_2 & = v_{-1} & = 10 \\
v_3 & = sin(v_0) & = sin5 \\
v_4 & = v_1 + v_2 & = 10.693 \\
v_5 & = v_4 - v_3 & = 11.652 \\
\hline
y & = v_5 & = 11.652
\end{array}
$$

Reverse Adjoint Trace

$$
\begin{array}{lll}
\bar{x_1} & = \bar{v_{-1}} & = 5.5 \\
\bar{x_2} & = \bar{v_0} & = 1.716 \\
\hline
\bar{v_{-1}} & = \bar{v_{-1}} + \bar{v_1}\frac{\partial v_1}{\partial v_{-1}} & = 5.5 \\
\bar{v_0} & = \bar{v_0} + \bar{v_2}\frac{\partial v_2}{\partial v_0} & = 1.716 \\
\bar{v_{-1}} & = \bar{v_2}\frac{\partial v_2}{\partial v_{-1}} & = 5 \\
\bar{v_0} & = \bar{v_3}\frac{\partial v_3}{\partial v_0} & = -0.284 \\
\bar{v_2} & = \bar{v_4}\frac{\partial v_4}{\partial v_2} & = 1 \\
\bar{v_1} & = \bar{v_4}\frac{\partial v_4}{\partial v_1} & = 1 \\
\bar{v_3} & = \bar{v_5}\frac{\partial v_5}{\partial v_3} & = -1 \\
\bar{v_4} & = \bar{v_5}\frac{\partial v_5}{\partial v_4} & = 1 \\
\hline
\bar{v_5} & = \bar{y} & = 1
\end{array}
$$

Computational

- For $f : R^n \rightarrow R$, one application of reverse mode is sufficient to calculate the full gradient.

$$\nabla f = \left(\frac{\partial y}{\partial x_1}, ..., \frac{\partial y}{\partial x_n} \right)$$

- Machine learning principally involves gradient of a scalar-valued objective w.r.t a large number of parameters, making reverse AD the mainstay technique in form of back propagation.
- In general, for $f : R^n \rightarrow R^m$, if operation count to evaluate original function is $ops(f)$, the time taken to calculate an $m \times n$ Jacobian for forward mode is $n\ c\ ops(f)$, whereas reverse AD takes $m\ c\ ops(f)$ (Typically, $m \gg n$).
- However, the advantage comes at the cost of increased storage.

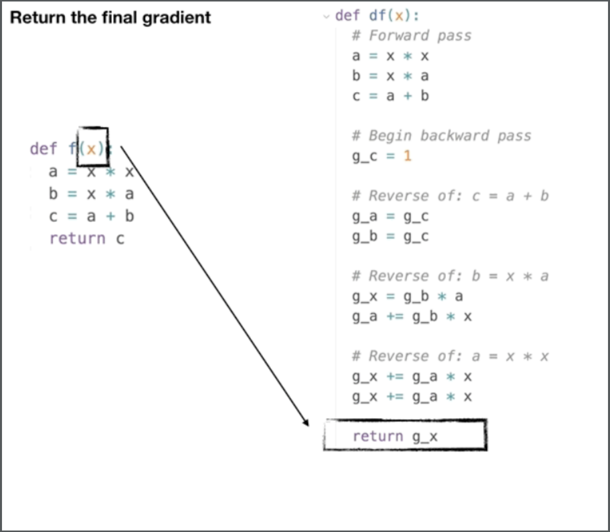
Memory

Pop quiz: given what you've learned so far, what – if anything – about feed-forward neural networks and back propagation may consume more memory than when using a network after training?

Source Code Transformation

- This type of implementation provides extensions to programming languages that automate the decomposition of algorithms into AD-enabled elementary operations.
- They are typically executed as preprocessors to transform input in the extended language into the original language.
- Tangent (Python), ADiMat (MATLAB)

Source Code Transformation with Tangent



Return the final gradient

```
def f(x):
    a = x * x
    b = x * a
    c = a + b
    return c
```

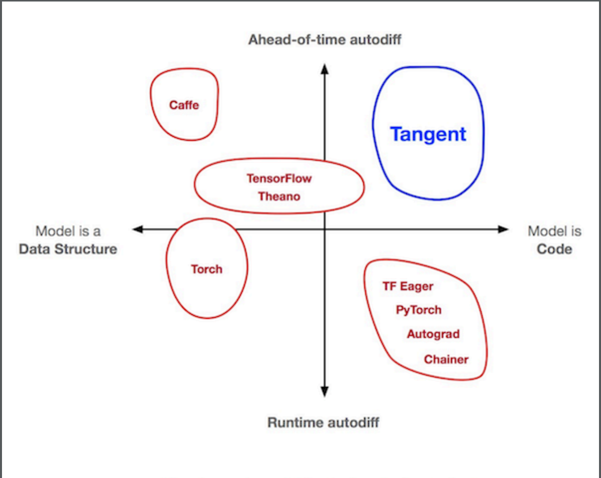
```
def df(x):
  # Forward pass
  a = x * x
  b = x * a
  c = a + b

  # Begin backward pass
  g_c = 1

  # Reverse of: c = a + b
  g_a = g_c
  g_b = g_c

  # Reverse of: b = x * a
  g_x = g_b * a
  g_a += g_b * x

  # Reverse of: a = x * x
  g_x += g_a * x
  g_x += g_a * x

  return g_x
```

Figure source: https://github.com/google/tangent

Tracing or Operator Overloading

- In modern programming languages with polymorphic features, operator overloading provides the most straightforward way of implementing AD.
- Primitives are overloaded so that each operation is logged onto a tape (a linear trace) at runtime, the chain rule can then be applied by walking this tape backward.
- Autograd (Python), Chainer (Python), PyTorch (Python, naturally), INTLab (MATLAB)
- TensorFlow 2.0 supports "eager mode" by default – closer to AD, but instructor does not know how close yet (still have to try it).

AD Spectrum



Different approaches and styles of modern deep learning libraries.
Not drawn to scale!

Figure source: https://github.com/google/tangent