Computational Graphs

Sargur N. Srihari
srihari@buffalo.edu

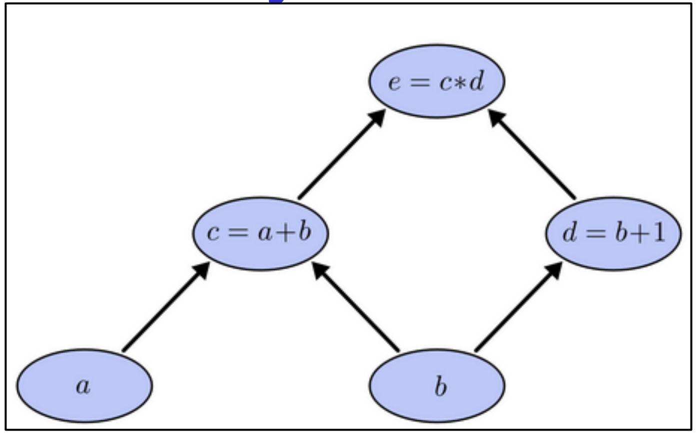
Topics

- Graph Language
  - Nodes as operations and inputs
  - Edges as values used in operations
- Composite functions as chains
- Derivatives in computational graphs
- Factoring paths
- Forward and Reverse Differentiation

Acknowledgements: colah.github, outlace.com/Computational-Graphs

Graph of a math expression

- Computational graphs are a nice way to:
  - Think about math expressions



- Consider the expression

  $$e=(a+b)*(b+1)$$

  - It has two adds, one multiply
  - Introduce a variable for result of each operation:

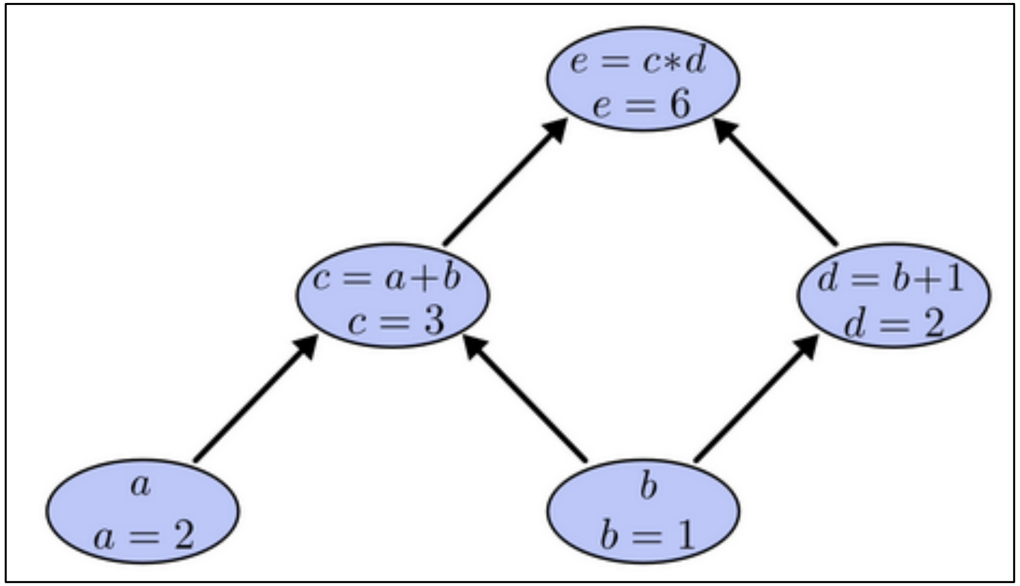
  $c=a+b$, $d=b+1$ and $e=c*d$

Such graphs are useful in CS especially functional programs. Core abstraction in deep learning using Theano

- To make a computational graph
  - Operations and inputs are nodes
  - Values used in operations are directed edges

Evaluating the expression

- Set the input variables to values and compute nodes up through the graph
- For a=2 and b=1



- Expression evaluates to 6

Computational Graph Language

- To describe backpropagation more precisely computational graph language is helpful
- Each node is either
  - a variable
    - Scalar, vector, matrix, tensor, or other type
  - Or an Operation
    - Simple function of one or more variables
    - Functions more complex than operations are obtained by composing operations
  - If variable y is computed by applying operation to variable x then draw directed edge from x to y

Composite Function

- Consider a composite function $f\left(g\left(h\left(x\right)\right)\right)$
  - We have an outer function f, an inner function f and a final inner function $h(x)$
- Say $f\left(x\right)= e^{\sin(x**2)}$ we can decompose it as:

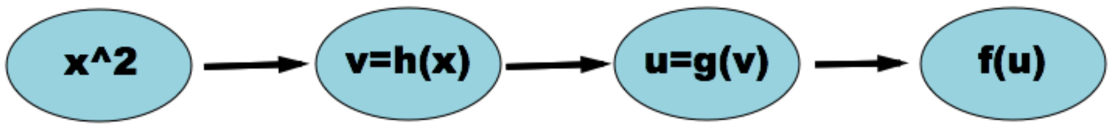
$$f\left(x\right)=e^{x}$$

$g(x)=sin\ x$ and

$h(x)=x^{2}$ or

$$f\left(g(h(x))\right)=e^{g(h(x))}$$

  - Its computational graph is



    - Every connection is an input, every node is a function or operation

Chain Rule for Composites

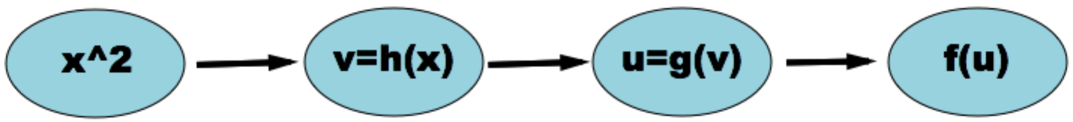
- Chain rule is the process we can use to analytically compute derivatives of composite functions.
- For example, $f\,(g\,(h\,(x)))$ is a composite function
  - We have an outer function f, an inner function f and a final inner function $h(x)$
  - Say $f\,(x) = e^{\sin(x{*}{*}2)}$ we can decompose it as:

    $f\,(x) = e^x,\ g(x) = sin\ x$ and $h(x) = x^2$ or

    $f\,(g(h(x))) = e^{g(h(x))}$

Derivatives of Composite function

- To get derivatives of $f(g(h(x))) = e^{g(h(x))}$ wrt x
  1. We use the chain rule $\boxed{\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}}$ where
     - $\boxed{\frac{df}{dg} = e^{g(h(x))}}$ since $f(g(h(x))) = e^{g(h(x))}$ & derivative of e^x is e
     - $\boxed{\frac{dg}{dh} = \cos(h(x))}$ since $g(h(x)) = sin\ h(x)$ & derivative $\sin$ is $\cos$
     - $\boxed{\frac{dh}{dx} = 2x}$ because $h(x) = x^2$ & its derivative is $2x$
     - Therefore $\boxed{\frac{df}{dx} = e^{g(h(x))} \cdot \cos\ h(x) \cdot 2x = e^{\sin x**2} \cdot \cos x^2 \cdot 2x}$
     - In each of these cases we pretend that the inner function is a single variable and derive it as such
  2. Another way to view it $f(x) = e^{\sin(x**2)}$
     - Create temp variables $u = \sin\ v$, $v = x^2$, then $f(u) = e^u$ with computational graph



Derivative using Computational Graph

- All we need to do is get the derivative of each node wrt each of its inputs

With $u = \sin v$, $v = x^2$, $f(u) = e^u$

$$\frac{dh}{dx} = 2x \qquad \frac{dg}{dh} = \cos(h(x)) \qquad \frac{df}{dg} = e^{g(h(x))}$$

- We can get whichever derivative we want by multiplying the 'connection' derivatives

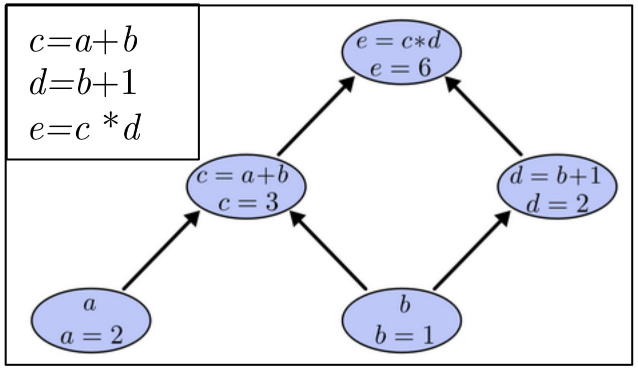
$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}$$

$$\frac{df}{dx} = e^{g(h(x))} \cdot \cos\ h(x) \cdot 2x$$
$$= e^{\sin x^2} \cdot \cos x^2 \cdot 2x$$

Since $f(x) = e^x$, $g(x) = sin\ x$ and $h(x) = x^2$

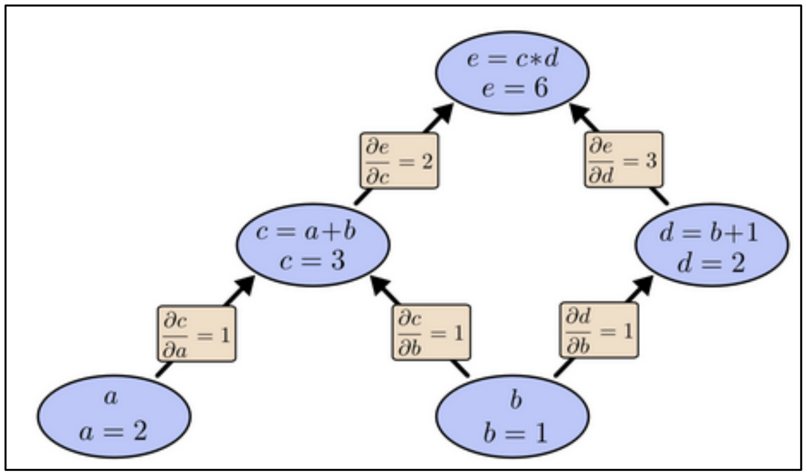
Derivatives for $e=(a+b)^{*}\ (b+1)$

- Computational graph
  - for $e=(a+b)^{*}\ (b+1)$



- Need derivatives on the edges
  - If a directly affects $c=a+b$, then we want to know how it affects c.
  - This is called *partial derivative* of c wrt a.
    - For partial derivatives of e we need sum & product rules of calculus

$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$
$$\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$



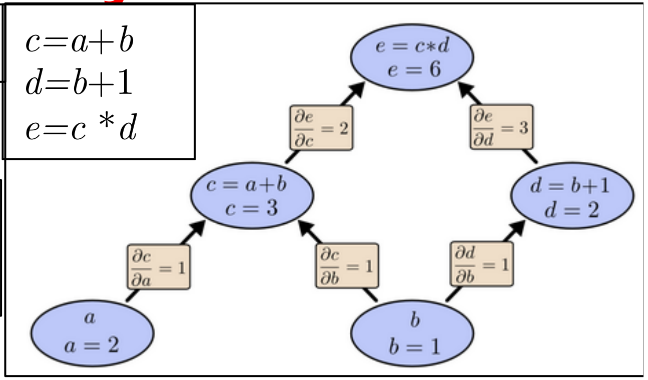
    - Derivative on edge: labeled

Derivative wrt variables indirectly connected

$e=(a+b)*(b+1)$

$c=a+b$
$d=b+1$
$e=c*d$

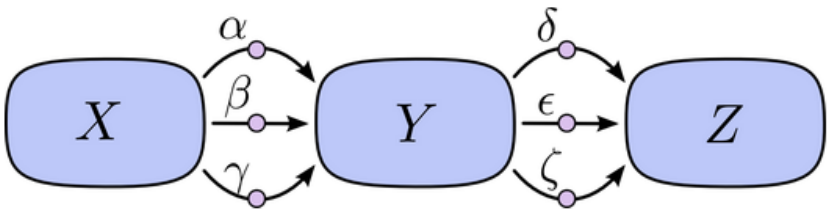
$a=2$ and $b=1$



- Effect of indirect connection:
  - How is e affected by a?
    - Since $\frac{\partial c}{\partial a}=\frac{\partial}{\partial a}(a+b)=1+0=1$
      - If we change a at a speed of 1, c changes by speed of 1
    - Since $\frac{\partial e}{\partial c}=\frac{\partial}{\partial c}(c*d)=d=b+1=1+1=2$
      - If we change c by a speed of 1, e changes by speed of 2
    - So e changes by a speed of $1*2=2$ wrt a
    - Equivalent to chain rule: $\frac{\partial e}{\partial a}=\frac{\partial c}{\partial a}\cdot\frac{\partial e}{\partial c}$
- The general rule (with multiple paths) is:
  - Sum over all possible paths from one node to the other while multiplying derivatives on each path
    - E.g., to get derivative of e wrt b $\frac{\partial e}{\partial b}=1*2+1*3=5$

Factoring Paths

- Summing over paths leads to combinatorial explosion



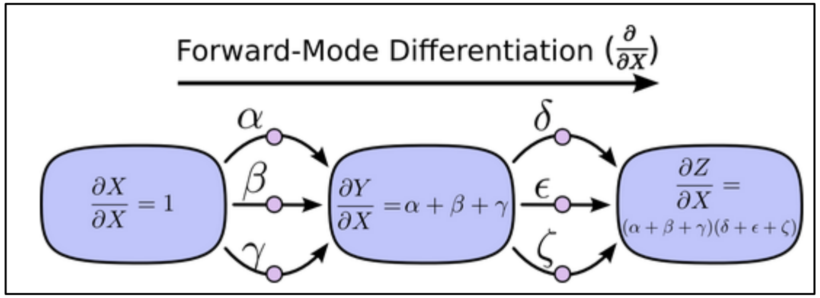
- If we want to get derivative $\boxed{\frac{\partial Z}{\partial X}}$ we need to sum over $3*3=9$ paths: $\boxed{\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\varepsilon + \alpha\zeta + \beta\delta + \beta\varepsilon + \beta\zeta + \gamma\delta + \gamma\varepsilon + \gamma\zeta}$
  - It will grow exponentially
  - Instead we could factor the paths as:

$$\boxed{\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \varepsilon + \zeta)}$$

- This is where *forward-mode* and *reverse-mode* differentiation come in

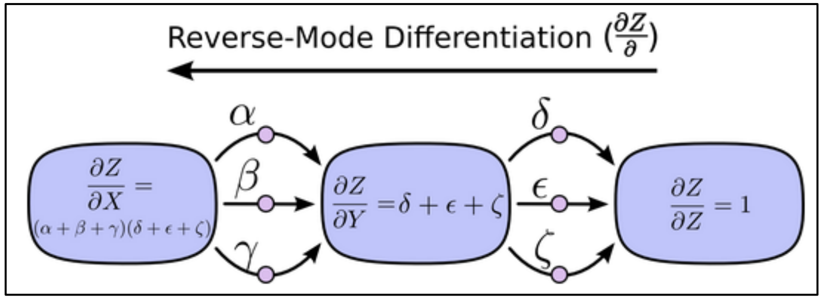
Forward- and Reverse-Mode Differentiation

- Forward mode differentiation tracks how one input affects every node
  - Applies $\boxed{\frac{\partial}{\partial X}}$ to every node



-

- Reverse mode differentiation tracks how every node affects one output
  - Applies $\boxed{\frac{\partial Z}{\partial}}$ to every node



Reverse Mode Differentiation

Reverse-mode differentiation from e down

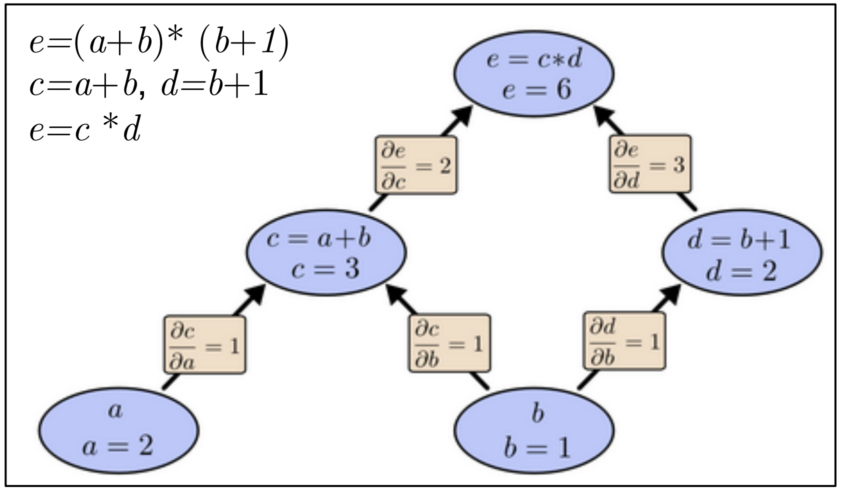
- Apply $\boxed{\frac{\partial e}{\partial}}$ to every node

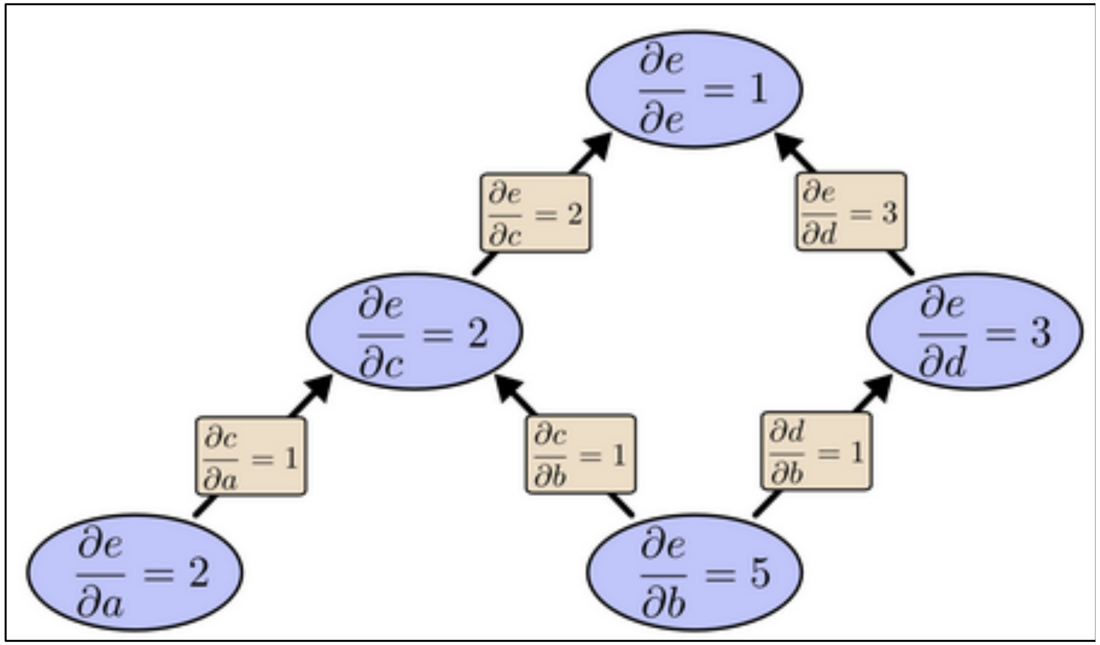
$$\boxed{\frac{\partial e}{\partial c} = \frac{\partial (c*d)}{\partial c} = d = b+1 = 1+1 = 2}$$

$$\boxed{\frac{\partial e}{\partial a} = \frac{\partial (c*d)}{\partial a} = \frac{\partial ((a+b)*(b+1))}{\partial a} = b+1 = 2}$$

- Gives derivative of e wrt every node
- We get both $\boxed{\frac{\partial e}{\partial a}}$ and $\boxed{\frac{\partial e}{\partial b}}$

a=2 and b=1





Combining the two modes

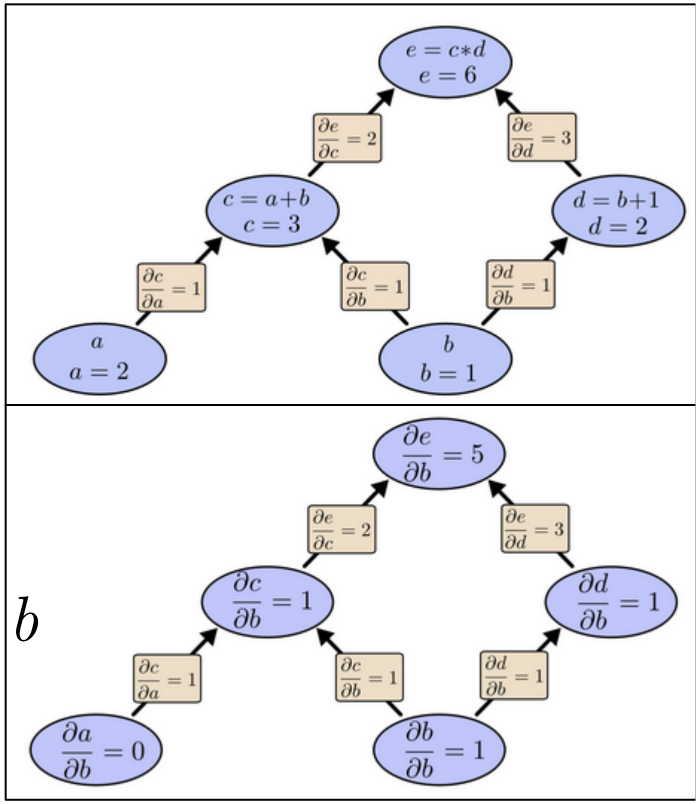
- Why reverse mode?

Consider Original example

$e=(a+b)*(b+1)$

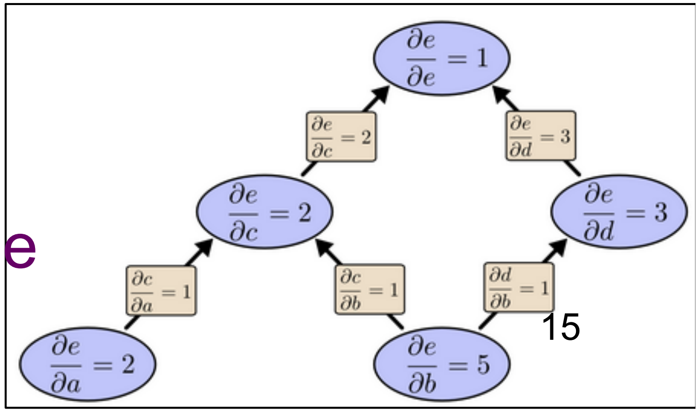
$c=a+b,\ d=b+1$

$e=c*d$



Forward differentiation from b up

- Gives derivative of every node wrt b
- *i.e.,* wrt a single input
- We get $\boxed{\frac{\partial e}{\partial b}}$



Reverse-mode diff from e down

- Gives derivative of e wrt every node
- We get both $\boxed{\frac{\partial e}{\partial a}}$ and $\boxed{\frac{\partial e}{\partial b}}$