Natural Language Processing with Deep Learning

Lecture 3 — Mathematical foundations of deep learning

Prof. Dr. Ivan Habernal

October 31, 2024

`www.trusthlt.org`
Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center Trustworthy Data Science and Security

Motivation

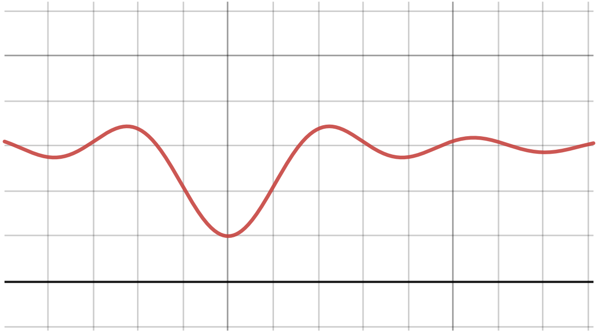
Why finding a minimum of a function matters?

In supervised machine learning ...

- We have some training data (e.g., for classification)
- We have a learning algorithm
- We want to minimize some kind of error (e.g., misclassification) of the learning algorithm on training data

Problem 1: Minimize functions

Problem: Find minimum of any function



- For "easy" functions, closed-form solution (high school math)
- For complicated functions not trivial and cumbersome

Function of single variable

We typically use Euler's notation with arbitrary but somehow standard naming conventions (x, y, f)

$y = f(x) \qquad f : \mathbb{R} \to \mathbb{R}$

$f : A \to B$ where A is domain, B is co-domain

Function composition

$f : \mathbb{R} \to \mathbb{R} \quad g : \mathbb{R} \to \mathbb{R}$

$h = g \circ f$

$h(x) = g(f(x))$ or $(g \circ f)(x) = g(f(x))$

Lines in two dimensions

Lines in a Cartesian plane are characterized by linear equations.

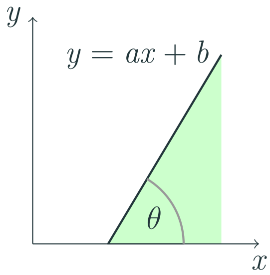
Every line L (including vertical lines) is the set of all points whose coordinates (x, y) satisfy a linear equation:

$$L = \{(x, y) \mid w_1 x + w_2 y = w_3\}$$

where w_1, w_2 and w_3 are fixed real numbers (called coefficients) such that w_1 and w_2 are not both zero.

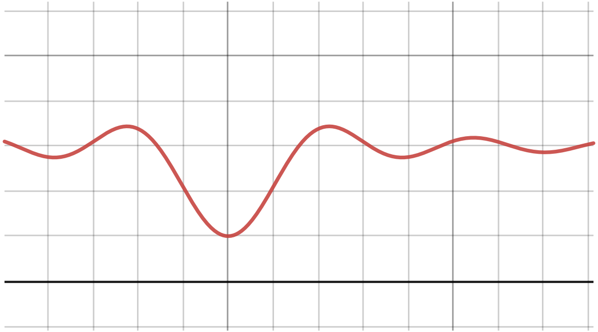
Linear function in two dimensions

Usually we use **slope-intercept** form $y = ax + b$



$\theta = \arctan(a) \qquad a = \tan(\theta)$

Approximate function by a line at point



"Steepness" at c?

$$f'(c) = \lim_{x \to c} \frac{f(x) - f(c)}{x - c}$$

The derivative of f at c

Derivative-computing function

We want a function D which, when given a differentiable function $f : \mathbb{R} \to \mathbb{R}$ as input, produces another function $g : \mathbb{R} \to \mathbb{R}$ output, such that $g(c) = f'(c)$ for every c.

This derivative-computing function D is often written as

$$\frac{d}{dx}$$

but this causes inconsistent notation like

$$\frac{d}{dx}(f), \qquad \frac{df}{dx}, \qquad \frac{dy}{dx}$$

and forces one to choose a variable name x or y

Derivative of nested functions: The chain rule hammer

Variant 1 (Lagrange's notation)

Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions which have derivatives.
Then the derivative of $g(f(x))$ is $g'(f(x)) \cdot f'(x)$

Variant 2 (Function composition operator $\circ$)

Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions which have derivatives.
Let $h = g \circ f$. The derivative of h is $h' = (g \circ f)' = (g' \circ f) \cdot f'$

Variant 3 (Leibniz's notation)

Call $h(x) = g(f(x))$. Then using $\frac{dh}{dx}$ for the derivative of h,
the chain rule for this would be $\frac{dh}{dx} = \frac{dh}{df}\frac{df}{dx}$

Chain rule example

Consider $y = e^{\sin(x^2)}$. Composite of three functions:

$$y = f(u) = e^u$$
$$u = g(v) = \sin v = \sin(x^2)$$
$$v = h(x) = x^2$$

Their derivatives are

$$\frac{dy}{du} = f'(u) = e^u = e^{\sin(x^2)}$$
$$\frac{du}{dv} = g'(v) = \cos v = \cos(x^2)$$
$$\frac{dv}{dx} = h'(x) = 2x$$

Chain rule example (cont.)

Consider $y = e^{\sin(x^2)}$. Composite of three functions:

$$y = f(u) = e^u, u = g(v) = \sin v = \sin\left(x^2\right), v = h(x) = x^2$$

Their derivatives are

$$\frac{dy}{du} = e^{\sin(x^2)}, \frac{du}{dv} = \cos\left(x^2\right), \frac{dv}{dx} = 2x$$

Derivative of their composite at the point $x = a$ is (in Leibniz notation)

$$\frac{dy}{dx} = \left.\frac{dy}{du}\right|_{u=g(h(a))} \cdot \left.\frac{du}{dv}\right|_{v=h(a)} \cdot \left.\frac{dv}{dx}\right|_{x=a}$$

Gradient-based optimization: Find minimum of a function

We want $\hat{x} = \operatorname{argmin}_x f(x)$

Pre-requisites:

- We can evaluate $y = f(x)$ for any x
- We can evaluate its derivative $f'(c)$ (or $\frac{dy}{dx}(c)$) for any c

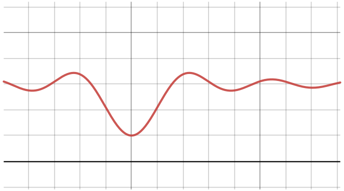
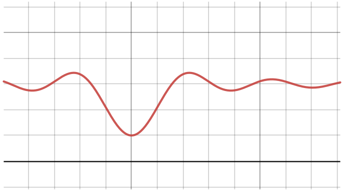


Figure 1: $3 - \frac{\sin(2x)}{x}$

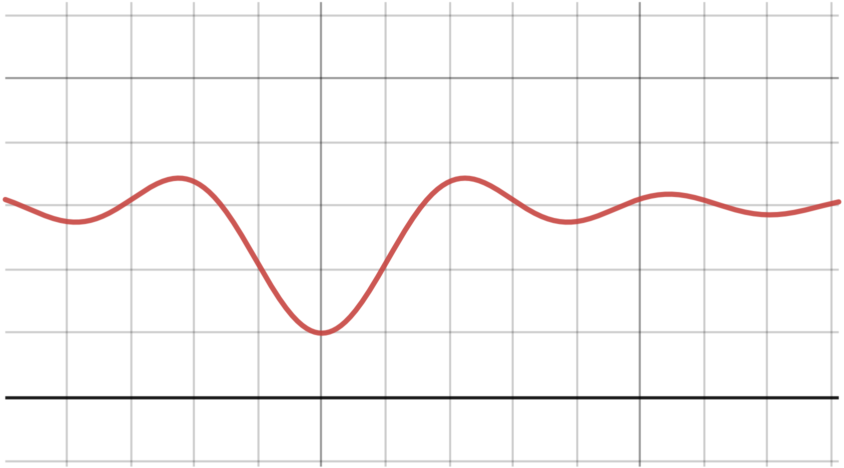
Gradient-based optimization: Find minimum of a function



1. Start with initial random value x_i
2. $u = f'(x_i)$ — direction and strength of change at x_i
3. Next value $x_{i+1} \leftarrow x_i - \eta \cdot u$
4. With small enough η (eta), $f(x_{i+1}) < f(x_i)$

Repeating 2 + 3 (with properly decreasing values of η) will find minimum point x_i

Gradient-based optimization: Workout example



Problem 2: Minimize multivariate functions

Multivariate functions $f : \mathbb{R}^n \to \mathbb{R}$

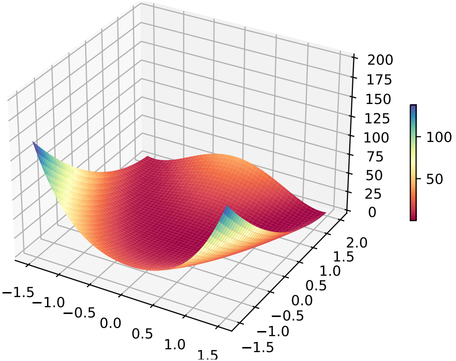


Figure 2: $f(x, y) = (a - x)^2 + b(y - x^2)^2$, $a = 1$, $b = 100$

`https://colab.research.google.com/drive/1mlZtxPXuk3mls56CQArmDzjdp5bLbrJC`

Partial derivatives

Partial derivative: the directional derivative wrt. a single variable

$\frac{\partial f}{\partial x_2}$ — "the partial derivative of f with respect to x_2"

Example: $f(x_1, x_2, x_3) = (x_1)^2 x_2 + \cos(x_3)$

$$\frac{\partial f}{\partial x_1} = 2x_2x_1 \qquad \frac{\partial f}{\partial x_2} = (x_1)^2 \qquad \frac{\partial f}{\partial x_3} = -\sin(x_3)$$

Gradient

Example: $f(x_1, x_2, x_3) = (x_1)^2 x_2 + \cos(x_3)$

$$\frac{\partial f}{\partial x_1} = 2x_2x_1 \qquad \frac{\partial f}{\partial x_2} = (x_1)^2 \qquad \frac{\partial f}{\partial x_3} = -\sin(x_3)$$

The resulting total derivative matrix Df is called the **gradient** of f, denoted ∇f

Example: $f(x_1, x_2, x_3) = (x_1)^2 x_2 + \cos(x_3)$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{pmatrix} = \begin{pmatrix} 2x_2x_1 & (x_1)^2 & -\sin(x_3) \end{pmatrix}$$

Gradient properties

Example: $f(x_1, x_2, x_3) = (x_1)^2 x_2 + \cos(x_3)$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{pmatrix} = \begin{pmatrix} 2x_2 x_1 & (x_1)^2 & -\sin(x_3) \end{pmatrix}$$

J. Kun (2020). *A Programmer's Introduction to Mathematics*. 2nd ed., p. 252

For every differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and every point $\boldsymbol{x} \in \mathbb{R}^n$, the gradient $\nabla f(\boldsymbol{x})$ points in the direction of steepest ascent of f at $\boldsymbol{x}$.

Warning!

Sometimes we call gradient the **function** for computing values for a given input (as above), sometimes the **vector of concrete numbers** computed for the given input

Gradient descent for minimizing multivariate functions

Given $f : \mathbb{R}^n \to \mathbb{R}$ we want to find

$$\hat{\boldsymbol{x}} = \operatorname*{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$$

1. Start at some random position with a random value vector $\boldsymbol{x}_i = (x_1, \ldots, x_n)$
2. Compute the gradient and update the position

$$\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i - \eta \cdot \nabla f(\boldsymbol{x}_i)$$

3. After enough iterations or some stopping criterion we have $\hat{\boldsymbol{x}}$

Gradient descent for minimizing multivariate functions

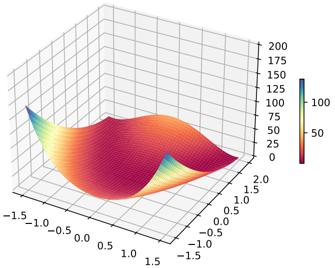
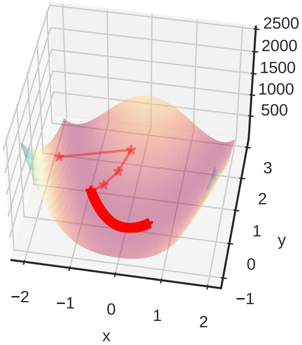
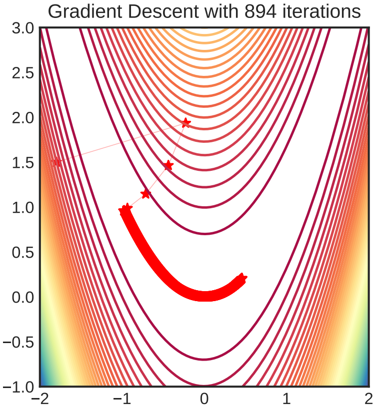


Figure 3: $f(x, y) = (a - x)^2 + b(y - x^2)^2$, $a = 1$, $b = 100$

$$\nabla f = \left(-400xy + 400x^3 + 2x - 2; \quad 200y - 200x^2\right)$$

Gradient for minimizing multivariate functions





Random starting point $(-1.8; 1.5)$, minimum at $(1; 1)$

`https://colab.research.google.com/drive/1pTGjtbiQg3q08NGNkA7XgPMIQXf7uT76`

Problem 3: When functions become heavily nested

In reality we work with deeply composed functions

Example

Minimize function e wrt. $w_0, w_1, \ldots, w_K$

$$e = -\frac{1}{N} \sum_{i=1}^{N} y_{[i]} \log \left(\frac{1}{1 + \exp\left(w_0 + \sum_{j=1}^{K} w_k \cdot \boldsymbol{x}_{[i][k]} \right)} \right)$$

Where $\boldsymbol{x}_{[1]}, \ldots, \boldsymbol{x}_{[N]}$, and $y_{[1]}, \ldots, y_{[N]}$ are constants

$$\nabla f = \left(\frac{\partial e}{\partial w_0}; \frac{\partial e}{\partial w_1}; \ldots; \frac{\partial e}{\partial w_K} \right)$$

$\frac{\partial e}{\partial w_1} = \ldots$ Good luck!

Chain rule for multivariable functions (two independent variables)

Suppose $x = g(u, v)$ and $y = h(u, v)$ are differentiable functions of u and v, and $z = f(x, y)$ is a differentiable function of x and y. Then, $z = f(g(u, v), h(u, v))$ is a differentiable function of u and v, and

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}$$

$$\frac{\partial z}{\partial v} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}$$

Problem 3: When functions become heavily nested

Efficient computation of gradient

Working example

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

Compute gradient wrt. a and b

This one is easy by hand, but that's not the point

$$e = (a + b)(b + 1) = ab + a + b^2 + b$$

$$\frac{\partial e}{\partial a} = b + 1 \qquad \frac{\partial e}{\partial b} = a + 2b + 1$$

Add some intermediate variables and function names

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

$e = f_1(r, s) := r \cdot s$
$r = f_2(a, b) := a + b$
$s = f_3(b) := b + 1$

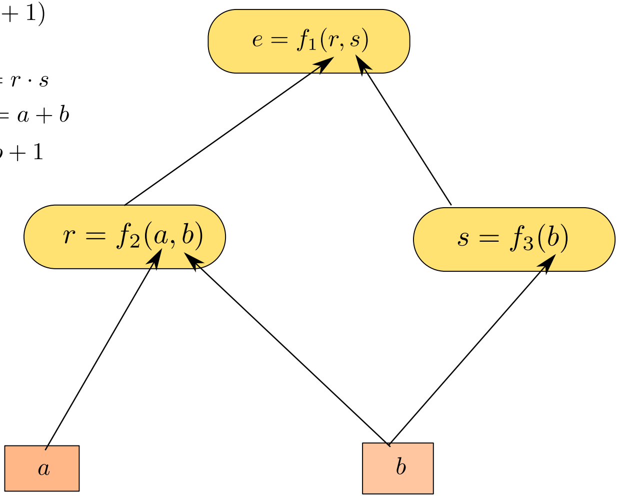
Build computational graph and evaluate (forward step)

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

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



Important: a, b will be some concrete real numbers, therefore r, s, e will be concrete real numbers too!

Computational graph

- DAG — directed acyclic graph (not necessarily a tree!)
- Each node — a differentiable function with arguments
- Leaves — variables (e.g., a, b) or constants
- Arrows — Function composition

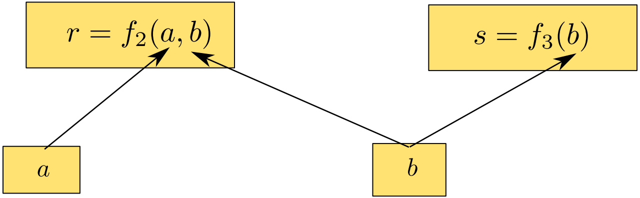


Figure 4: r, s are parents of b; a, b are children (arguments) of r

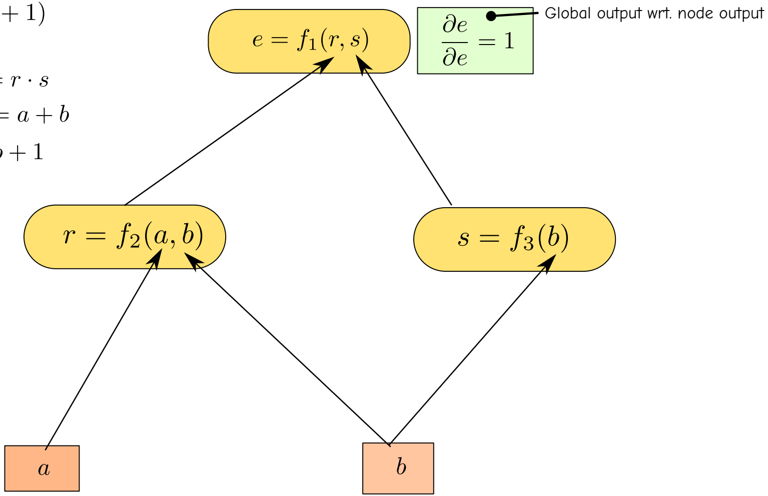
Goal: $\frac{\partial e}{\partial a}$ and $\frac{\partial e}{\partial b}$ (gradient), but let's do $\frac{\partial e}{\partial \star}$ for every node

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

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



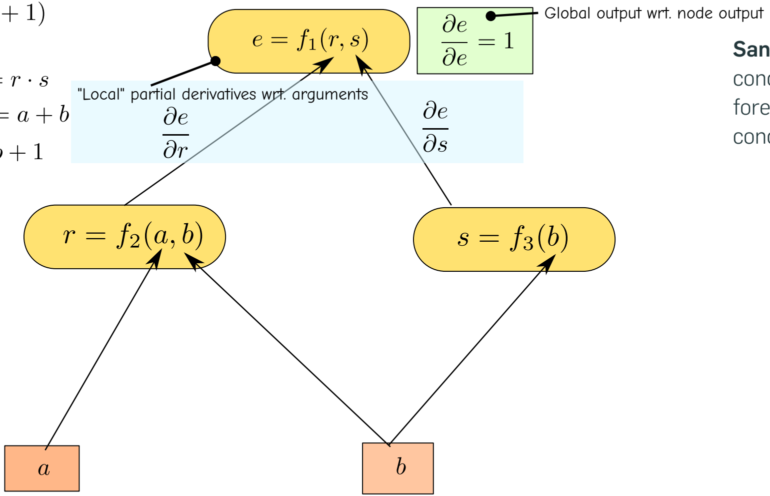
Since $e = r \cdot s$, partial derivatives are easy: $\frac{\partial e}{\partial r} = s$ and $\frac{\partial e}{\partial s} = r$

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

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



Sanity check: r, s are some concrete real numbers, therefore $\frac{\partial e}{\partial r}$ and $\frac{\partial e}{\partial s}$ will be concrete real numbers too!

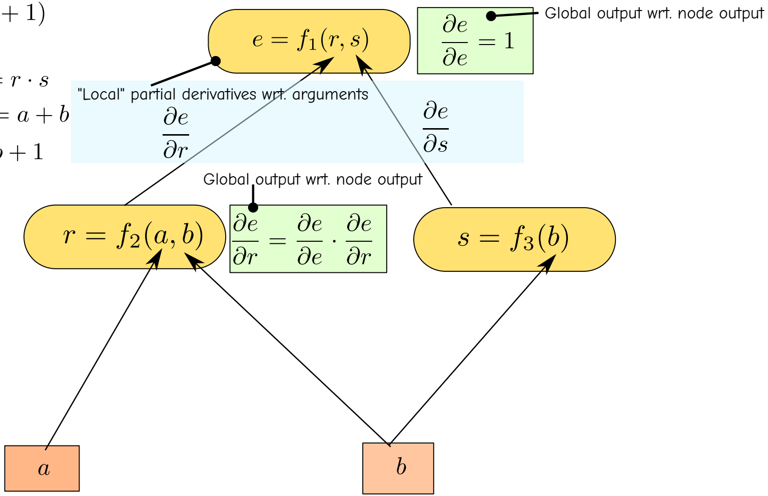
Proceed to next child r and compute $\frac{\partial e}{\partial r}$ – use chain rule!

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

$e = f_1(r,s) := r \cdot s$

$r = f_2(a,b) := a + b$

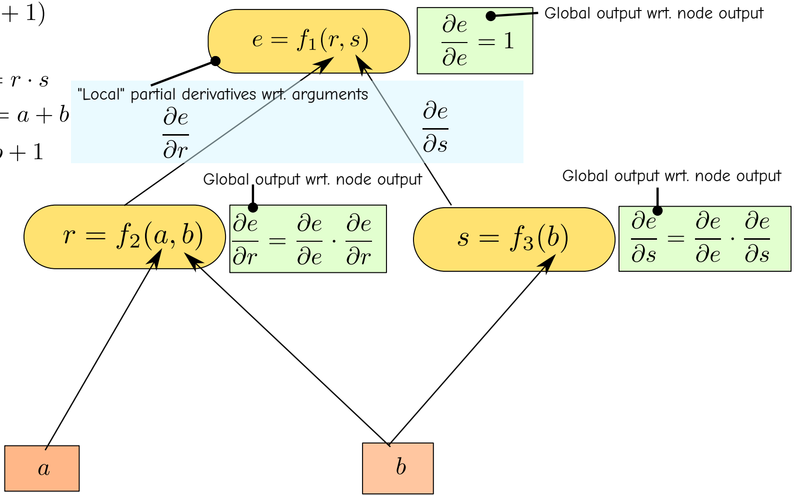
$s = f_3(b) := b + 1$



Proceed to next child s and compute $\frac{\partial e}{\partial s}$ – use chain rule!

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

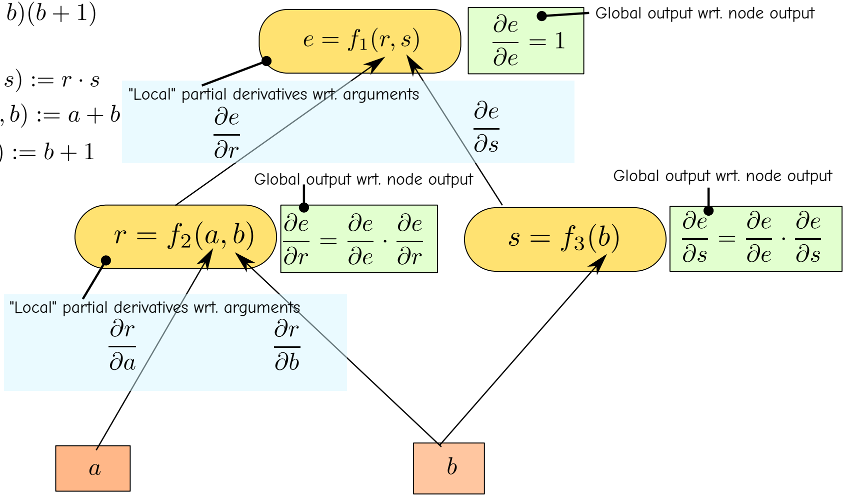
$e = f_1(r,s) := r \cdot s$
$r = f_2(a,b) := a + b$
$s = f_3(b) := b + 1$



Since $r = a + b$, partial derivatives are easy: $\frac{\partial r}{\partial a} = 1$ and $\frac{\partial r}{\partial b} = 1$

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

$e = f_1(r, s) := r \cdot s$
$r = f_2(a, b) := a + b$
$s = f_3(b) := b + 1$



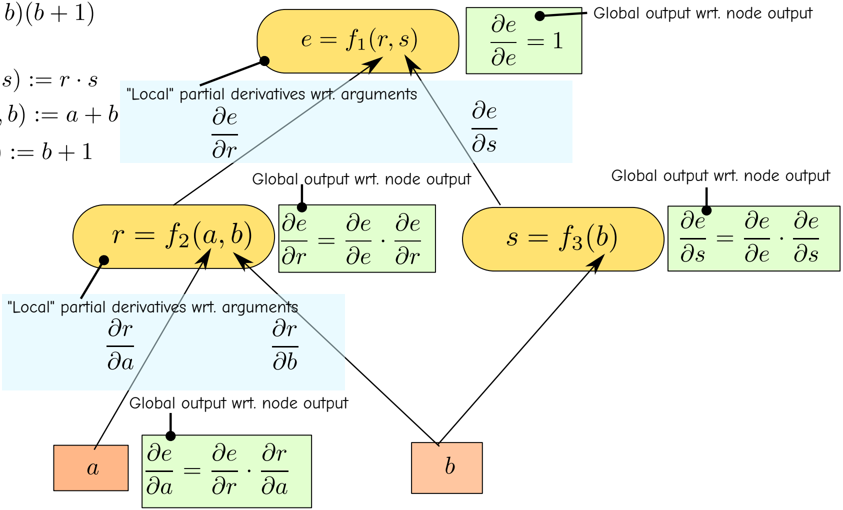
Proceed to next child a and compute $\frac{\partial e}{\partial a}$ – use chain rule!

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

$e = f_1(r,s) := r \cdot s$

$r = f_2(a,b) := a + b$

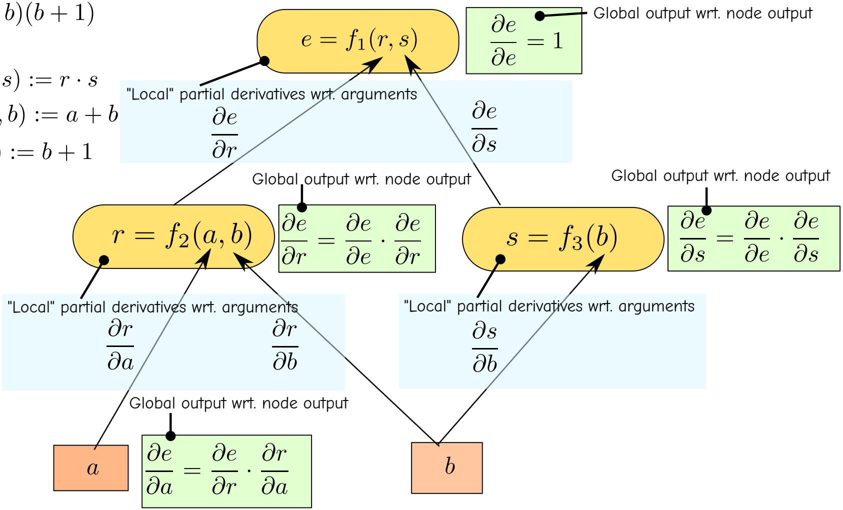
$s = f_3(b) := b + 1$



Since $s = b + 1$, partial derivatives are easy: $\frac{\partial s}{\partial b} = 1$

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

$$e = f_1(r, s) := r \cdot s$$
$$r = f_2(a, b) := a + b$$
$$s = f_3(b) := b + 1$$



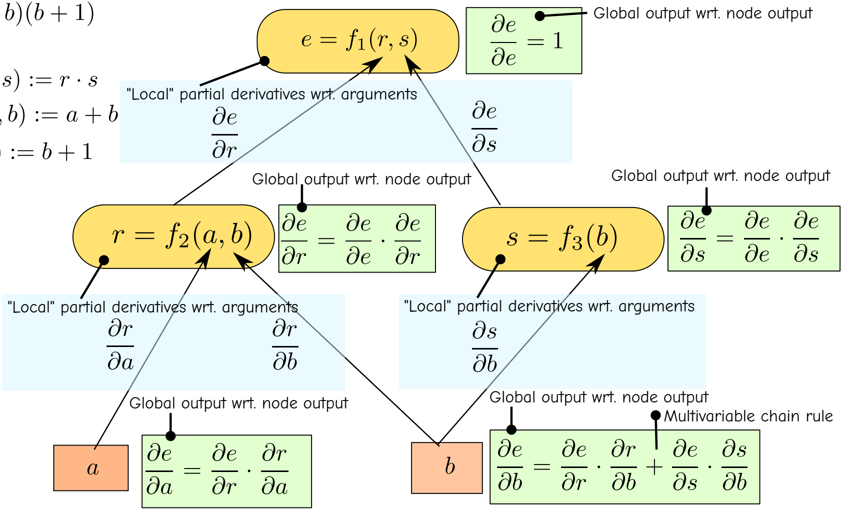
Proceed to b and compute $\frac{\partial e}{\partial b}$ – use multivariate chain rule!

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

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$



Goal: $\nabla e = \left(\frac{\partial e}{\partial a}; \frac{\partial e}{\partial b}\right)$ — we computed it for concrete a and b!

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

$e = f_1(r, s) := r \cdot s$

$r = f_2(a, b) := a + b$

$s = f_3(b) := b + 1$

$e = f_1(r, s)$

$\frac{\partial e}{\partial e} = 1$ Global output wrt. node output

"Local" partial derivatives wrt. arguments

$\frac{\partial e}{\partial r}$ $\frac{\partial e}{\partial s}$

$r = f_2(a, b)$

Global output wrt. node output

$\frac{\partial e}{\partial r} = \frac{\partial e}{\partial e} \cdot \frac{\partial e}{\partial r}$

$s = f_3(b)$

Global output wrt. node output

$\frac{\partial e}{\partial s} = \frac{\partial e}{\partial e} \cdot \frac{\partial e}{\partial s}$

"Local" partial derivatives wrt. arguments

$\frac{\partial r}{\partial a}$ $\frac{\partial r}{\partial b}$

"Local" partial derivatives wrt. arguments

$\frac{\partial s}{\partial b}$

a

Global output wrt. node output

$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial r} \cdot \frac{\partial r}{\partial a}$

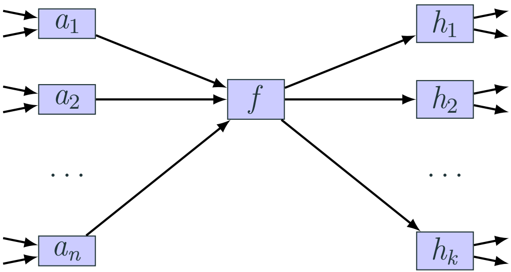
b

Global output wrt. node output

Multivariable chain rule

$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial r} \cdot \frac{\partial r}{\partial b} + \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial b}$

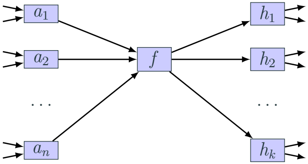
Generic node in a computational graph



Adapted from J. Kun (2020). *A Programmer's Introduction to Mathematics*. 2nd ed., p. 265

Figure 5: A generic node of a computation graph. Node f has many inputs, its output feeds into many nodes, and each of its inputs and outputs may also have many inputs and outputs.

Generic node in a computational graph $f(a_1, \ldots, a_n)$



Assuming the graph is a function $e = g(\ldots)$, we compute

$$\frac{\partial e}{\partial f} = \sum_{i=1}^{k} \frac{\partial e}{\partial h_i} \cdot \frac{\partial h_i}{\partial f}$$

and

$$\frac{\partial f}{\partial a_i} \quad \text{for } a_i, \ldots, a_n$$

What each node must implement?

For example a function $s = f(a, b, c, d)$

- How to compute the output value s (given the parameters a, b, c, d)
- How to compute partial derivatives wrt. the parameters, i.e. $\frac{\partial s}{\partial a}, \frac{\partial s}{\partial b}, \frac{\partial s}{\partial c}, \frac{\partial s}{\partial d}$

Backpropagation

- Forward computation: Compute all nodes' output (and cache it)
- Backward computation (Backprop): Compute the overall function's partial derivative with respect to each node

Ordering of the computations? Recursively or build a graph's topology upfront and iterate

Backpropagation: Recap

- We can express any arbitrarily complicated function $f : \mathbb{R}^n \to \mathbb{R}$ as a computational graph
- For computing the gradient ∇f at a concrete point $(x_1, x_2, \ldots, x_n)$ we run the forward pass and backprop
- When caching each node's intermediate output and partial derivatives, we avoid repeating computations $\to$ efficient algorithm

Recap

Take aways

- We can quite efficiently find a minimum of any differentiable nested multivariate function
  - Iterative gradient descent takes the most promising direction
  - Backpropagation utilizes computational graphs and caching $\rightarrow$ computes gradients efficiently
- We have not touched neural networks yet at all!

License and credits

Credits

Ivan Habernal

Content from ACL Anthology papers licensed under CC-BY
`https://www.aclweb.org/anthology`