10-301/601: Introduction to Machine Learning Lecture 13 – Backpropagation

Henry Chai

2/24/25

Front Matter

- Announcements
  - Exam 1 viewings this week, Tuesday – Thursday
    - See Piazza for complete details
  - Homework 4 released 2/17, due 2/26 at 11:59 PM

Recall: Automatic Differentiation (reverse mode)

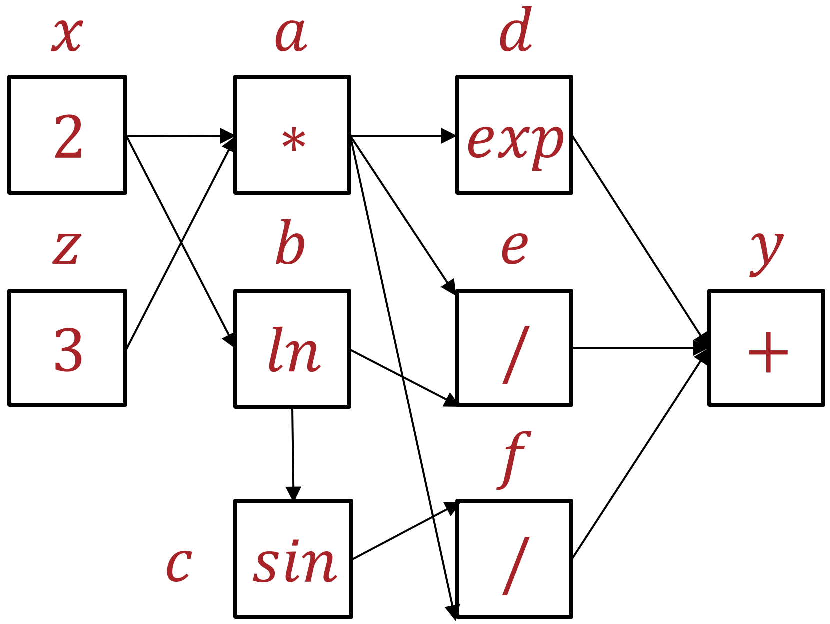
- Given

$$y = f(x,z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

what are $\partial y / \partial x$ and $\partial y / \partial z$ at $x = 2, z = 3$?

- First define some intermediate quantities, draw the computation graph and run the “forward” computation

$$a = xz$$
$$b = \ln(x)$$
$$c = \sin(b)$$
$$d = e^{a}$$
$$e = a/b$$
$$f = c/a$$
$$y = d + e + f$$



Example courtesy of Matt Gormley

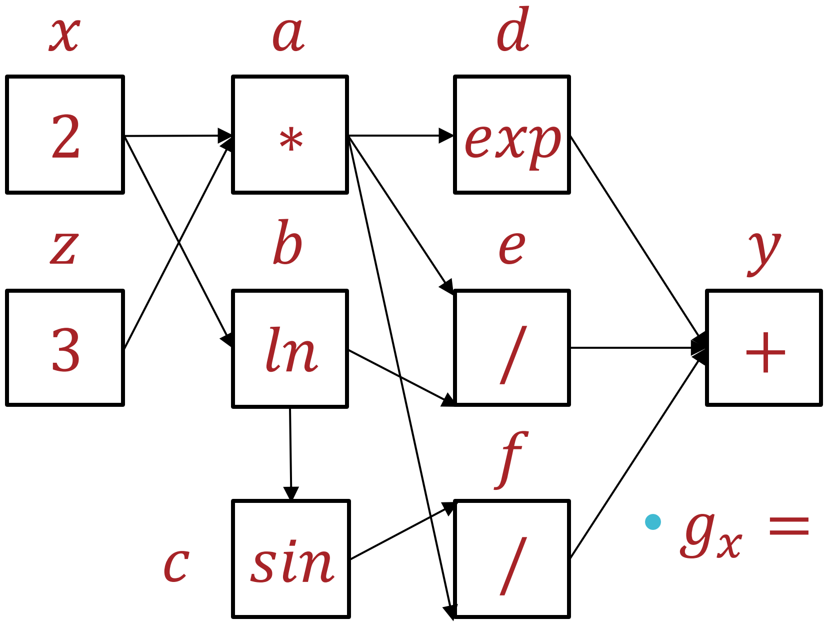
Recall: Automatic Differentiation (reverse mode)

- Given

$$y = f(x,z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

what are $\partial y / \partial x$ and $\partial y / \partial z$ at $x = 2, z = 3$?

- Then compute partial derivatives, starting from y and working back



- $g_y = \frac{\partial y}{\partial y} = 1$
- $g_d = g_e = g_f = 1$
- $g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial c} = g_f\left(\frac{1}{a}\right)$
- $g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial e}\frac{\partial e}{\partial b} + \frac{\partial y}{\partial c}\frac{\partial c}{\partial b}$
  $= g_e\left(-\frac{a}{b^2}\right) + g_c(\cos(b))$
- $g_a = \frac{\partial y}{\partial a} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial a} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial a} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial a}$
  $= g_f\left(\frac{-c}{a^2}\right) + g_e\left(\frac{1}{b}\right) + g_d(e^a)$
- $g_x = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial b}\frac{\partial b}{\partial x} + \frac{\partial y}{\partial a}\frac{\partial a}{\partial x} = g_b\left(\frac{1}{x}\right) + g_a(z)$
- $g_z = \frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = g_a(x)$

Example courtesy of Matt Gormley

Computation Graph 10-301/601 Conventions

- The diagram represents *an algorithm*
- Nodes are rectangles with one node per intermediate variable in the algorithm
- Each node is labeled with the function that it computes (inside the box) and the variable name (outside the box)
- Edges are directed and do not have labels
- For neural networks:
  - Each weight, feature value, label and *bias term* appears as a node
  - We *can* include the loss function

Neural Network Diagram Conventions

- The diagram represents a *neural network*
- Nodes are circles with one node per hidden unit
- Each node is labeled with the variable corresponding to the hidden unit
- Edges are directed and each edge is labeled with its weight
- The diagram typically does *not* include any nodes related to the loss computation

Matrix Calculus

Numerator

Denominator

Types of Derivatives	**scalar**	**vector**	**matrix**
scalar	$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{Y}}{\partial x}$
vector	$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$
matrix	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Table courtesy of Matt Gormley

Matrix Calculus: Denominator Layout

- Derivatives of a scalar always have the *same* *shape* as the entity that the derivative is being taken with respect to.

Types of Derivatives	**scalar**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$
matrix	$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix}$

Table courtesy of Matt Gormley

Matrix Calculus: Denominator Layout

Types of Derivatives	**scalar**	**vector**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$	$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_N}{\partial x} \end{bmatrix}$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \cdots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$

Table courtesy of Matt Gormley

$$\frac{\partial y}{\partial \vec{x}} \in \mathbb{R}^{6\times 1}$$

$$\frac{\partial y}{\partial \vec{u}} \in \mathbb{R}^{8\times 1}, \quad \frac{\partial \vec{u}}{\partial \vec{x}} \in \mathbb{R}^{6\times 8}$$

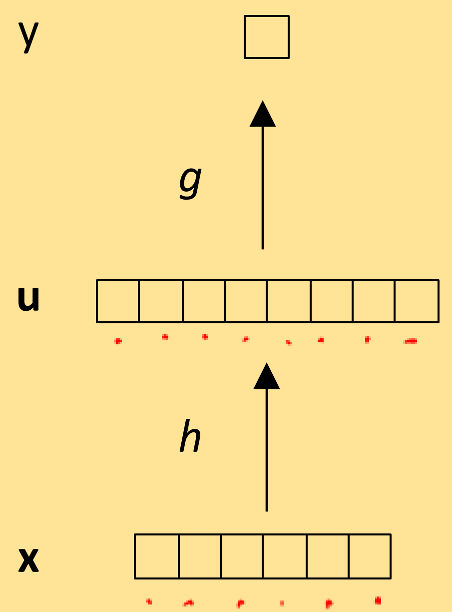
Matrix Calculus

$$\frac{\partial y}{\partial \vec{x}} = \frac{\partial \vec{u}}{\partial \vec{x}} \frac{\partial y}{\partial \vec{u}}$$

$$(\mathbb{R}^{a\times b})(\mathbb{R}^{b\times c}) \in \mathbb{R}^{a\times c}$$

Poll Question 1:

Suppose y = g(**u**) and **u** = h(**x**)



Which of the following is the correct definition of the chain rule?

Recall:

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \cdots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$$

Answers:

$$\frac{\partial y}{\partial \mathbf{x}} = \ldots$$

A. $\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

B. $\frac{\partial y}{\partial \mathbf{u}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

C. $\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}^T$

D. $\frac{\partial y}{\partial \mathbf{u}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}^T$

E. $(\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}})^T$

F. None of the above

Gradient Descent for Neural Network Training

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While TERMINATION CRITERION is not satisfied
  - For $l = 1, \dots, L$
    - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$
    - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
  - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$

Computing Gradients

$$\ell_{\mathcal{D}}\left(W_{(t)}^{(1)},\ldots,W_{(t)}^{(L)}\right)=\sum_{n=1}^{N}\boxed{\ell^{(n)}}\left(W_{(t)}^{(1)},\ldots,W_{(t)}^{(L)}\right)$$

$$\nabla_{W^{(l)}}\ell_{\mathcal{D}}\left(W_{(t)}^{(1)},\ldots,W_{(t)}^{(L)}\right)$$

$$\star = \begin{bmatrix} \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,d^{(l-1)}}^{(l)}} \\ \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,d^{(l-1)}}^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},d^{(l-1)}}^{(l)}} \end{bmatrix}$$

$$\frac{\partial \ell_{\mathcal{D}}}{\partial w_{b,a}^{(l)}}=\sum_{n=1}^{N}\boxed{\frac{\partial \ell^{(n)}\left(W_{(t)}^{(1)},\ldots,W_{(t)}^{(L)}\right)}{\partial w_{b,a}^{(l)}}}$$

Computing Gradients: Intuition

- A weight affects the prediction of the network (and therefore the error) through downstream signals/outputs
  - Use the chain rule!
- Any weight going into the same node will affect the prediction through the same downstream path
  - Compute derivatives starting from the last layer and move “backwards”
  - Store computed derivatives and reuse for efficiency (automatic differentiation)

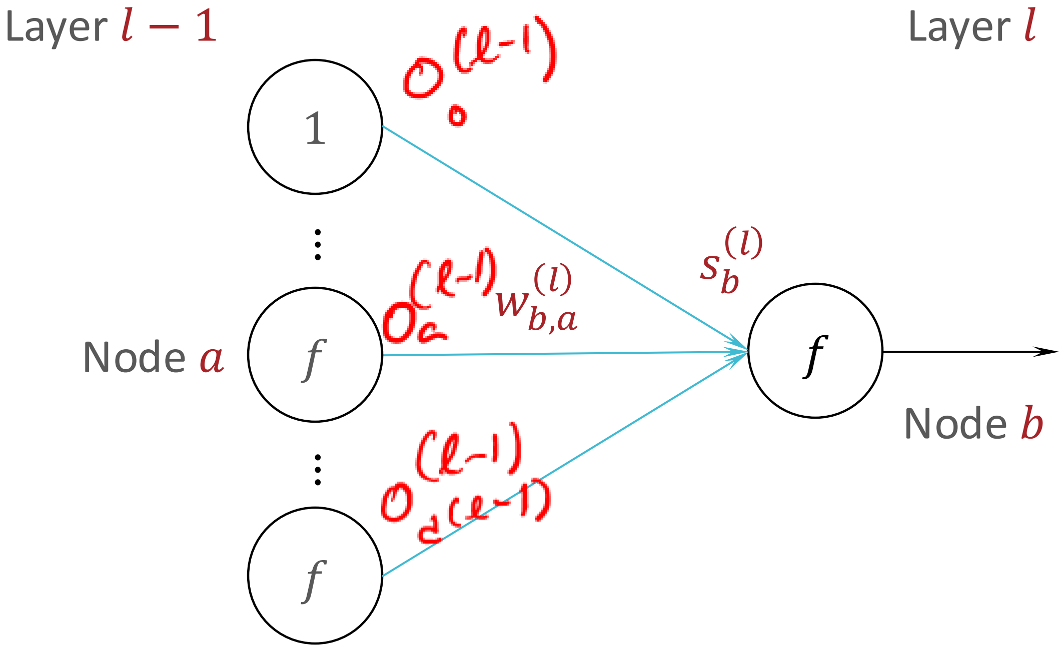
$(x^{(n)}, y^{(n)})$

Computing Partial Derivatives

Computing $\nabla_{W^{(l)}} \ell_{\mathcal{D}} \left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right)$ reduces to computing

$$\frac{\partial \ell^{(n)}}{\partial w_{b,a}^{(l)}}$$

Insight: $w_{b,a}^{(l)}$ *only* affects $\ell^{(n)}$ via $s_b^{(l)}$



Computing Partial Derivatives

Computing $\nabla_{W^{(l)}} \ell_{\mathcal{D}} \left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)} \right)$ reduces to computing

$$\frac{\partial \ell^{(n)}}{\partial w_{b,a}^{(l)}}$$

Insight: $w_{b,a}^{(l)}$ *only* affects $\ell^{(n)}$ via $s_b^{(l)}$

$$\frac{\partial \ell^{(n)}}{\partial w_{b,a}^{(l)}} = \frac{\partial \ell^{(n)}}{\partial s_b^{(l)}} \frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}}$$

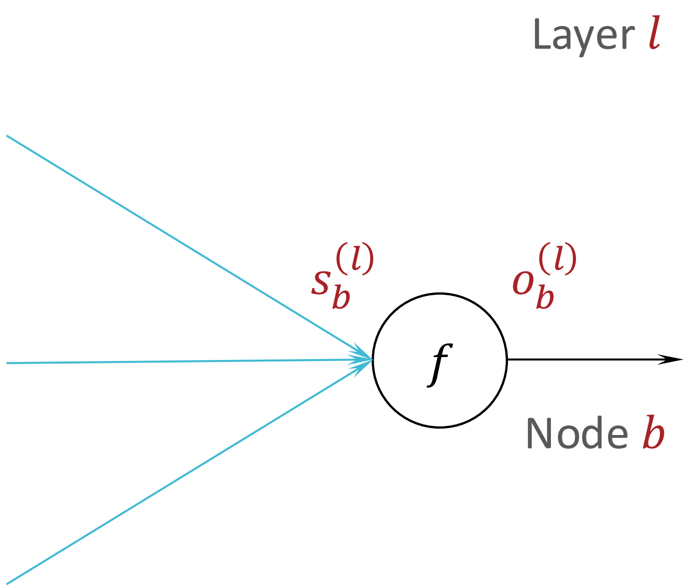
$$\delta_b^{(l)} = \text{"sensitivity"}$$

$$s_b^{(l)} = \sum_{\alpha=0}^{d^{(l-1)}} w_{b,\alpha}^{(l)} o_\alpha^{(l-1)}$$

$$\partial s_b^{(l)} / \partial w_{b,a}^{(l)} = o_a^{(l-1)}$$

Computing Partial Derivatives

Insight: $s_b^{(l)}$ *only* affects $\ell^{(n)}$ via $o_b^{(l)}$



Computing Partial Derivatives

Insight: $s_b^{(l)}$ *only* affects $\ell^{(n)}$ via $o_b^{(l)}$

"sensitivity" $= \delta_b^{(\ell)} = \dfrac{\partial \ell^{(n)}}{\partial o_b^{(\ell)}} \dfrac{\partial o_b^{(\ell)}}{\partial s_b^{(\ell)}}$

"harder"

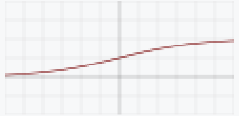
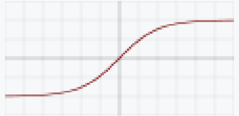
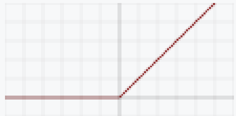
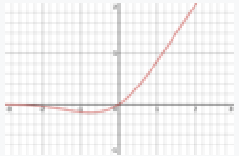
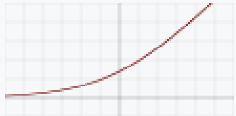
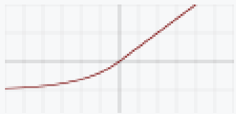
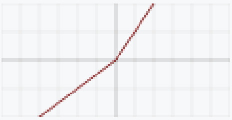
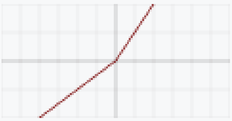
$o_b^{(\ell)} = f(s_b^{(\ell)})$

if $f(z) = \tanh(z)$

$\dfrac{\partial o_b^{(\ell)}}{\partial s_b^{(\ell)}} = (1 - o_b^{(\ell)2}) \leq 1$

"vanishing gradient" $\Longrightarrow$

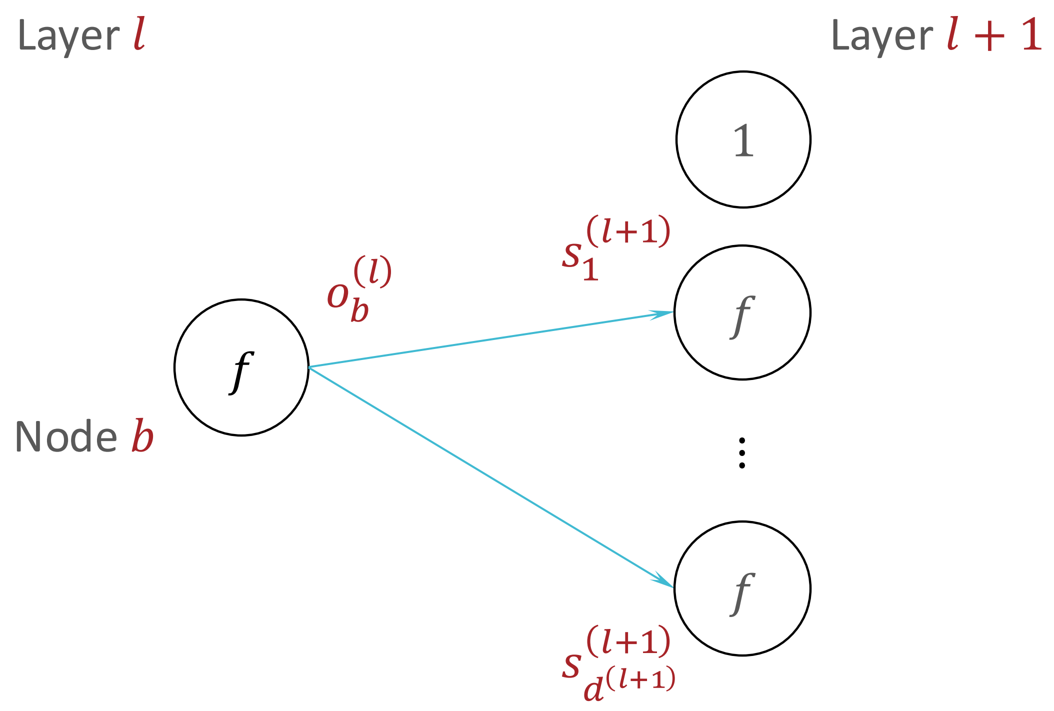
Recall: Other Activation Functions

Logistic, sigmoid, or soft step		$\sigma(x) = \dfrac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$
Softplus[8]		$\ln(1+e^x)$
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Parametric rectified linear unit (PReLU)[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α

Source: https://en.wikipedia.org/wiki/Activation_function

Computing Partial Derivatives

Insight: $o_b^{(l)}$ affects $\ell^{(n)}$ via $s_1^{(l+1)}, \ldots, s_{d^{(l+1)}}^{(l+1)}$



Computing Partial Derivatives

Insight: $o_b^{(l)}$ affects $\ell^{(n)}$ via $s_1^{(l+1)}, \ldots, s_{d^{(l+1)}}^{(l+1)}$

$$\frac{\partial \ell^{(n)}}{\partial o_b^{(\ell)}} = \sum_{c=1}^{d^{(\ell+1)}} \frac{\partial \ell^{(n)}}{\partial s_c^{(\ell+1)}} \frac{\partial s_c^{(\ell+1)}}{\partial o_b^{(\ell)}}$$

$$\delta_c^{(\ell+1)}$$

$$s_c^{(\ell+1)} = \sum_{\beta=0}^{d^{(\ell)}} w_{c,\beta}^{(\ell+1)} o_\beta^{(\ell)}$$

$$\frac{\partial s_c^{(\ell+1)}}{\partial o_b^{(\ell)}} = w_{c,b}^{(\ell+1)}$$

Computing Partial Derivatives

$$\boxed{\delta_b^{(l)}} = \frac{\partial \ell^{(n)}}{\partial o_b^{(l)}} \left(\frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$$

$$= \left(\sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left(w_{c,b}^{(l+1)} \right) \right) \left(1 - \left(o_b^{(l)} \right)^2 \right)$$

$$\boldsymbol{\delta}^{(l)} := \nabla_{\mathbf{s}^{(l)}} \ell^{(n)} \left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)} \right) \in \mathbb{R}^{(d^{(\ell)}+1) \times 1}$$

$$= W^{(\ell+1)^T} \vec{\delta}^{(\ell+1)} \odot \nabla_{\vec{s}^{(\ell)}} f \in \mathbb{R}^{(d^{(\ell)}+1) \times 1}$$

element-wise product

$$\boxed{\frac{\partial f}{\partial s_b^{(\ell)}}} \text{ if } f(z) = \tanh(z)$$

$$W^{(\ell+1)} \in \mathbb{R}^{d^{(\ell+1)} \times (d^{(\ell)}+1)}$$

$$\vec{\delta}^{(\ell+1)} \in \mathbb{R}^{d^{(\ell+1)} \times 1}$$

$$\nabla_{\vec{s}^{(\ell)}} f \in \mathbb{R}^{(d^{(\ell)}+1) \times 1}$$

Computing Gradients

$$\boxed{\frac{\partial \ell^{(n)}}{\partial w_{b,a}^{(l)}}} = \delta_b^{(l)} \left(\frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}} \right) = \delta_b^{(l)} \left(o_a^{(l-1)} \right)$$

$$\nabla_{w^{(\ell)}} \ell^{(n)} \in \mathbb{R}^{d^{(\ell)} \times (d^{(\ell-1)} + 1)}$$

$$\nabla_{W^{(l)}} \ell^{(n)} = \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$$

$$\vec{\delta}^{(\ell)} \in \mathbb{R}^{d^{(\ell)} \times 1}$$

$$\vec{o}^{(\ell-1)} \in \mathbb{R}^{(d^{(\ell-1)} + 1) \times 1}$$

Computing Partial Derivatives

Can recursively compute $\boldsymbol{\delta}^{(l)}$ using $\boldsymbol{\delta}^{(l+1)}$; need to compute the base case: $\boldsymbol{\delta}^{(L)}$

- Assume the output layer is a single node and the error function is the squared error: $\boldsymbol{\delta}^{(L)} = \delta_1^{(L)}$, $\boldsymbol{o}^{(L)} = o_1^{(L)}$ and $\ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = \left(o_1^{(L)} - y^{(n)}\right)^2$

$$\delta_1^{(L)} = \frac{\partial \ell^{(n)}\left(o_1^{(L)}, y^{(n)}\right)}{\partial s_1^{(L)}} = \frac{\partial}{\partial s_1^{(L)}}\left(o_1^{(L)} - y^{(n)}\right)^2$$

$$= 2\left(o_1^{(L)} - y^{(n)}\right)\frac{\partial o_1^{(L)}}{\partial s_1^{(L)}}$$

$\hookrightarrow \frac{\partial \theta}{\partial s_1^{(L)}}$

Back-propagation

- Input: $W^{(1)}, \dots, W^{(L)}$ and $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
- Initialize: $\ell_{\mathcal{D}} = 0$ and $G^{(l)} = 0 \odot W^{(l)} \ \forall\, l = 1, \dots, L$
- ~~For $n = 1, \dots, N$~~ sample n from D
    - Run forward propagation with $\boldsymbol{x}^{(n)}$ to get $\boldsymbol{o}^{(1)}, \dots, \boldsymbol{o}^{(L)}$
    - (Optional) Increment $\ell_{\mathcal{D}}$: $\ell_{\mathcal{D}} = \ell_{\mathcal{D}} + \left(o^{(L)} - y^{(n)}\right)^2$
    - Initialize: $\boldsymbol{\delta}^{(L)} = 2\left(o_1^{(L)} - y^{(n)}\right)\frac{\partial o_1^{(L)}}{\partial s_1^{(L)}}$
    - For $l = L-1, \dots, 1$
        - Compute $\boldsymbol{\delta}^{(l)} = W^{(l+1)^T}\boldsymbol{\delta}^{(l+1)} \odot \underbrace{\left(1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)}\right)}_{\nabla_{s^{(l)}} f}$
        - Increment $G^{(l)}$: $G^{(l)} = G^{(l)} + \boldsymbol{\delta}^{(l)}\boldsymbol{o}^{(l-1)^T}$
- Output: $G^{(1)}, \dots, G^{(L)}$, the gradients of $\ell_{\mathcal{D}}$ w.r.t $W^{(1)}, \dots, W^{(L)}$

Backpropagation Learning Objectives

You should be able to…

- Differentiate between a neural network diagram and a computation graph
- Construct a computation graph for a function as specified by an algorithm
- Carry out the backpropagation on an arbitrary computation graph
- Construct a computation graph for a neural network, identifying all the given and intermediate quantities that are relevant
- Instantiate the backpropagation algorithm for a neural network
- Instantiate an optimization method (e.g. SGD) and a regularizer (e.g. L2) when the parameters of a model are comprised of several matrices corresponding to different layers of a neural network
- Apply the empirical risk minimization framework to learn a neural network
- Use the finite difference method to evaluate the gradient of a function
- Identify when the gradient of a function can be computed at all and when it can be computed efficiently
- Employ basic matrix calculus to compute vector/matrix/tensor derivatives.