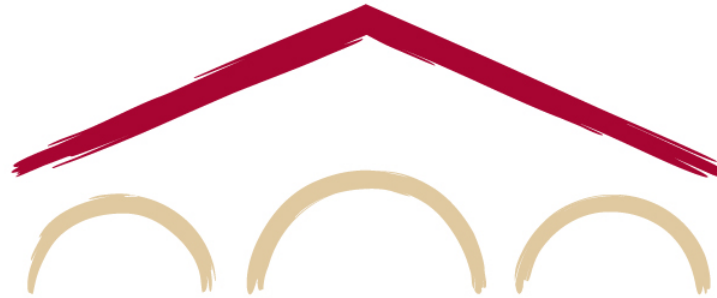


# Natural Language Processing with Deep Learning

## CS224N/Ling284



Christopher Manning

Lecture 3: Neural net learning: Gradients by hand (matrix calculus)  
and algorithmically (the backpropagation algorithm)

# 1. Introduction

Assignment 2 is all about making sure you really understand the math of neural networks ... then we'll let the software do it!

We'll go through it all quickly today, but this is the week of quarter to most work through the readings!

This will be a tough week for some! →

Make sure to get help if you need it

- Visit office hours

- Read tutorial materials given in the syllabus

Thursday will be mainly linguistics! Some people find that tough too 😊

# Named Entity Recognition (NER)

- The task: **find** and **classify** names in text, for example:

Last night , Paris Hilton wowed in a sequin gown .

PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .

PER PER LOC LOC LOC DATE DATE

- Possible uses:
  - Tracking mentions of particular entities in documents
  - For question answering, answers are usually named entities
- Often followed by Named Entity Linking/Canonicalization into Knowledge Base

# Simple NER: Window classification using binary logistic classifier

- **Idea:** classify each word in its context window of neighboring words
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors in a window
  - Really, we usually use multi-class softmax, but trying to keep it simple 😊
- **Example:** Classify “Paris” as +/- location in context of sentence with window length 2:

the museums in Paris are amazing to see .

$$X_{\text{window}} = [ x_{\text{museums}} \quad x_{\text{in}} \quad x_{\text{Paris}} \quad x_{\text{are}} \quad x_{\text{amazing}} ]^T$$

- Resulting vector  $x_{\text{window}} = \boxed{x \in \mathbb{R}^{5d}}$ , a column vector!
- To classify all words: run classifier for each class on the vector centered on each word in the sentence

# NER: Binary classification for center word being location

- We do supervised training and want high score if it's a location

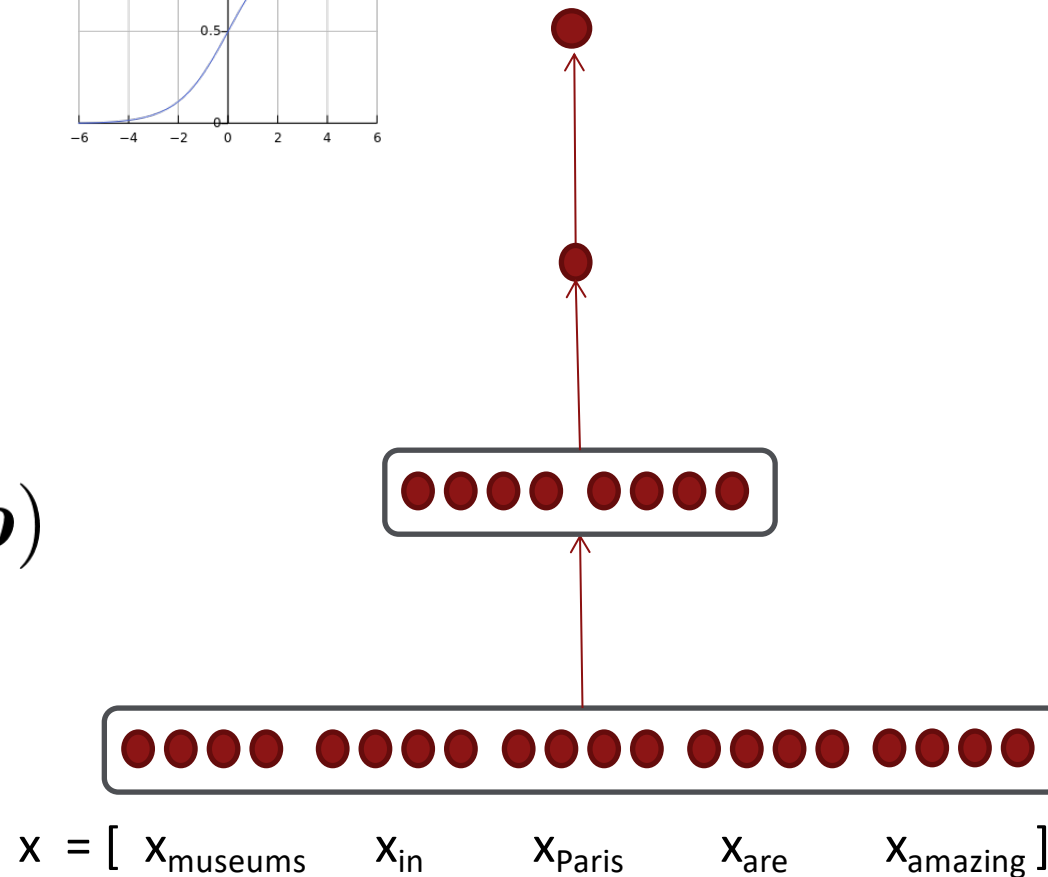
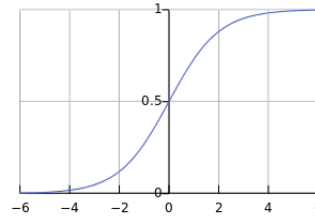
$$J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

predicted model probability of class

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$\mathbf{x}$  (input)



# Remember: Stochastic Gradient Descent

Update equation:

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

$\alpha = \textit{step size or learning rate}$

i.e., for each parameter:  $\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial J(\theta)}{\partial \theta_j^{old}}$

In deep learning, we update the data representation (e.g., word vectors) too!

How can we compute  $\nabla_{\theta} J(\theta)$ ?

1. By hand
2. Algorithmically: the backpropagation algorithm

# Lecture Plan

## Lecture 4: Gradients by hand and algorithmically

1. Introduction (5 mins)
2. Matrix calculus (40 mins)
3. Backpropagation (35 mins)

# Computing Gradients by Hand

- **Matrix calculus:** Fully vectorized gradients
  - “Multivariable calculus is just like single-variable calculus if you use matrices”
  - Much faster and more useful than non-vectorized gradients
  - But doing a non-vectorized gradient can be good for intuition; recall the first lecture for an example
  - **Lecture notes and matrix calculus notes cover this material in more detail**
  - **You might also review Math 51, which has a new online textbook:**  
<http://web.stanford.edu/class/math51/textbook.html>  
**or maybe you're luckier if you did Engr 108**



# Gradients

- Given a function with 1 output and 1 input

$$f(x) = x^3$$

- It's gradient (slope) is its derivative

$$\frac{df}{dx} = 3x^2$$

“How much will the output change if we change the input a bit?”

At  $x = 1$  it changes about 3 times as much:  $1.01^3 = 1.03$

At  $x = 4$  it changes about 48 times as much:  $4.01^3 = 64.48$

# Gradients

- Given a function with 1 output and  $n$  inputs

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$$

- Its gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \mathbf{x}} = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$$

# Jacobian Matrix: Generalization of the Gradient

- Given a function with  **$m$  outputs** and  $n$  inputs

$$\mathbf{f}(\mathbf{x}) = [f_1(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)]$$

- It's Jacobian is an  **$m \times n$  matrix** of partial derivatives

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\left( \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

# Chain Rule

- For composition of one-variable functions: **multiply derivatives**

$$z = 3y$$

$$y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = (3)(2x) = 6x$$

- For multiple variables at once: **multiply Jacobians**

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \dots$$

## Example Jacobian: Elementwise activation Function

$$\mathbf{h} = f(\mathbf{z}), \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}}? \quad \mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$
$$h_i = f(z_i)$$

## Example Jacobian: Elementwise activation Function

$$\mathbf{h} = f(\mathbf{z}), \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}}? \quad \mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$
$$h_i = f(z_i)$$

Function has  $n$  outputs and  $n$  inputs  $\rightarrow n$  by  $n$  Jacobian

## Example Jacobian: Elementwise activation Function

$$\mathbf{h} = f(\mathbf{z}), \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}}? \quad \mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$
$$h_i = f(z_i)$$

$$\left( \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \quad \text{definition of Jacobian}$$

## Example Jacobian: Elementwise activation Function

$$\mathbf{h} = f(\mathbf{z}), \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}}?$$

$$h_i = f(z_i)$$

$$\mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$

$$\begin{aligned} \left( \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \right)_{ij} &= \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \\ &= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \end{aligned}$$

definition of Jacobian

regular 1-variable derivative



## Example Jacobian: Elementwise activation Function

$$\mathbf{h} = f(\mathbf{z}), \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}}? \quad \mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$
$$h_i = f(z_i)$$

$$\left( \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \quad \text{definition of Jacobian}$$
$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \quad \text{regular 1-variable derivative}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\mathbf{f}'(\mathbf{z}))$$

## Other Jacobians

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{W}$$

## Other Jacobians

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{W}$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I} \text{ (Identity matrix)}$$

## Other Jacobians

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{W}$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

Fine print: This is the correct Jacobian.  
Later we discuss the “shape convention”;  
using it the answer would be  $\mathbf{h}$ .

## Other Jacobians

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{W}$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

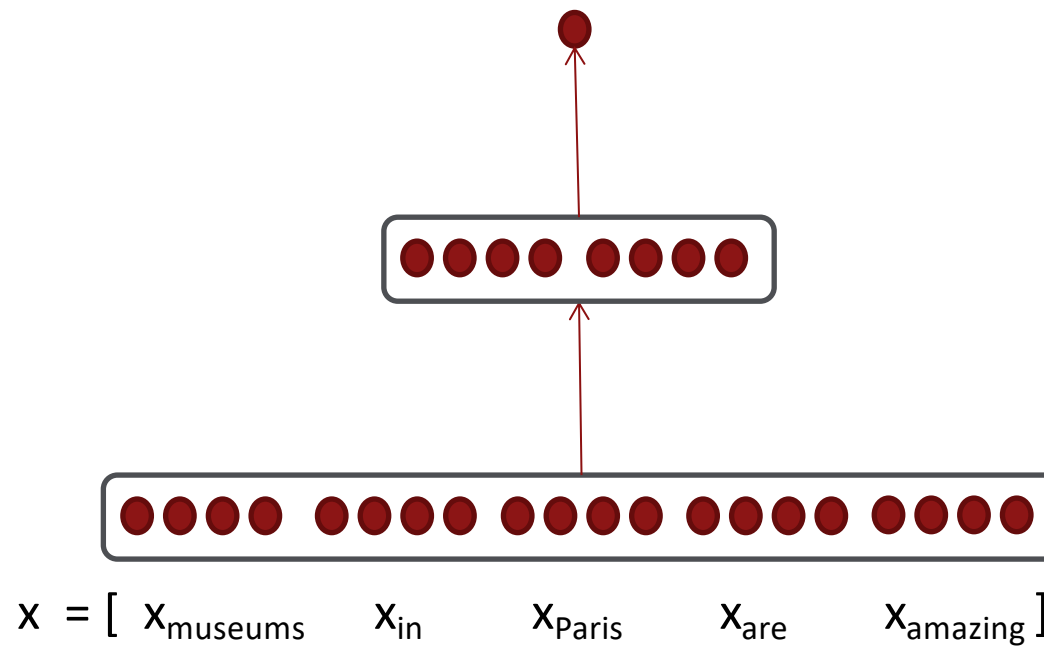
- Compute these at home for practice!
  - Check your answers with the lecture notes

# Back to our Neural Net!

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$\mathbf{x}$  (input)



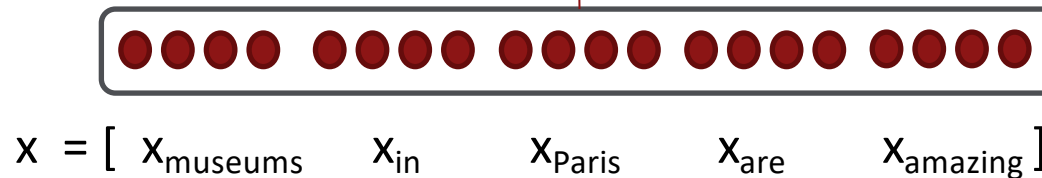
# Back to our Neural Net!

- Let's find  $\frac{\partial s}{\partial \mathbf{b}}$ 
  - Really, we care about the gradient of the loss  $J_t$  but we will compute the gradient of the score for simplicity

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$\mathbf{x}$  (input)



# 1. Break up equations into simple pieces

$$s = \mathbf{u}^T \mathbf{h}$$

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad \rightarrow \quad \mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \quad (\text{input})$$

$$\mathbf{x} \quad (\text{input})$$

Carefully define your variables and keep track of their dimensionality!



## 2. Apply the chain rule

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

## 2. Apply the chain rule

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

## 2. Apply the chain rule

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$\mathbf{x}$  (input)

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

## 2. Apply the chain rule

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$\mathbf{x}$  (input)

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

### 3. Write out the Jacobians

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \text{ (input)}$$

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{z}} (f(\mathbf{z})) = \text{diag}(f'(\mathbf{z}))$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$

### 3. Write out the Jacobians

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \text{ (input)}$$

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

$\downarrow$   
 $\mathbf{u}^T$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{z}} (f(\mathbf{z})) = \text{diag}(f'(\mathbf{z}))$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$

### 3. Write out the Jacobians

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$\mathbf{x}$  (input)

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \quad \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \quad \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

$\downarrow$                        $\downarrow$

$$\mathbf{u}^T \text{diag}(f'(\mathbf{z}))$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{z}} (f(\mathbf{z})) = \text{diag}(f'(\mathbf{z}))$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$

### 3. Write out the Jacobians

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$\mathbf{x}$  (input)

$$\begin{aligned} \frac{\partial s}{\partial \mathbf{b}} &= \frac{\partial s}{\partial \mathbf{h}} \quad \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \quad \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \\ &\quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\ &= \mathbf{u}^T \text{diag}(f'(\mathbf{z})) \mathbf{I} \end{aligned}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{z}} (f(\mathbf{z})) = \text{diag}(f'(\mathbf{z}))$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$



### 3. Write out the Jacobians

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \text{ (input)}$$

$$\begin{aligned} \frac{\partial s}{\partial \mathbf{b}} &= \frac{\partial s}{\partial \mathbf{h}} \quad \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \quad \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \\ &\quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\ &= \mathbf{u}^T \text{diag}(f'(\mathbf{z})) \mathbf{I} \\ &= \mathbf{u}^T \circ f'(\mathbf{z}) \end{aligned}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \mathbf{u}} (\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{z}} (f(\mathbf{z})) = \text{diag}(f'(\mathbf{z}))$$

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$

# Re-using Computation

- Suppose we now want to compute  $\frac{\partial s}{\partial \mathbf{W}}$ 
  - Using the chain rule again:

$$\frac{\partial s}{\partial \mathbf{W}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

# Re-using Computation

- Suppose we now want to compute  $\frac{\partial s}{\partial \mathbf{W}}$ 
  - Using the chain rule again:

$$\frac{\partial s}{\partial \mathbf{W}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$
$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

The same! Let's avoid duplicated computation ...

# Re-using Computation

- Suppose we now want to compute  $\frac{\partial s}{\partial \mathbf{W}}$ 
  - Using the chain rule again:

$$\frac{\partial s}{\partial \mathbf{W}} = \delta \frac{\partial z}{\partial \mathbf{W}}$$

$$\frac{\partial s}{\partial \mathbf{b}} = \delta \frac{\partial z}{\partial \mathbf{b}} = \delta$$

$$\delta = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} = \mathbf{u}^T \circ f'(z)$$

$\delta$  is the local error signal

## Derivative with respect to Matrix: Output shape

- What does  $\frac{\partial s}{\partial \mathbf{W}}$  look like?  $\mathbf{W} \in \mathbb{R}^{n \times m}$
- 1 output,  $nm$  inputs: 1 by  $nm$  Jacobian?
- Inconvenient to then do  $\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$

# Derivative with respect to Matrix: Output shape

- What does  $\frac{\partial s}{\partial \mathbf{W}}$  look like?  $\mathbf{W} \in \mathbb{R}^{n \times m}$
- 1 output,  $nm$  inputs: 1 by  $nm$  Jacobian?
  - Inconvenient to then do  $\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$
- Instead, we leave pure math and use the **shape convention**: the shape of the gradient is the shape of the parameters!

- So  $\frac{\partial s}{\partial \mathbf{W}}$  is  $n$  by  $m$ :
$$\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

# Derivative with respect to Matrix

- What is  $\frac{\partial s}{\partial \mathbf{W}} = \delta \frac{\partial z}{\partial \mathbf{W}}$ 
  - $\delta$  is going to be in our answer
  - The other term should be  $\mathbf{x}$  because  $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$

- Answer is:  $\frac{\partial s}{\partial \mathbf{W}} = \delta^T \mathbf{x}^T$

$\delta$  is local error signal at  $z$   
 $\mathbf{x}$  is local input signal

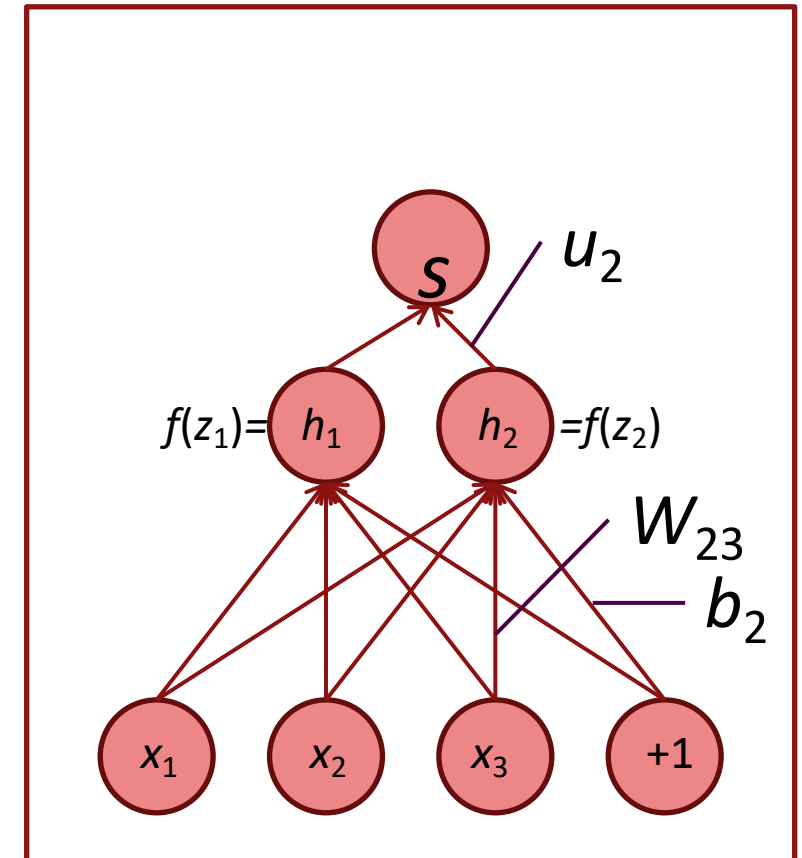
# Deriving local input gradient in backprop

- For  $\frac{\partial z}{\partial W}$  in our equation:

$$\frac{\partial s}{\partial W} = \delta \frac{\partial z}{\partial W} = \delta \frac{\partial}{\partial W} (Wx + b)$$

- Let's consider the derivative of a single weight  $W_{ij}$
- $W_{ij}$  only contributes to  $z_i$ 
  - For example:  $W_{23}$  is only used to compute  $z_2$  not  $z_1$

$$\begin{aligned} \frac{\partial z_i}{\partial W_{ij}} &= \frac{\partial}{\partial W_{ij}} W_{i \cdot} x + b_i \\ &= \frac{\partial}{\partial W_{ij}} \sum_{k=1}^d W_{ik} x_k = x_j \end{aligned}$$





# Why the Transposes?

$$\begin{aligned} \frac{\partial s}{\partial \mathbf{W}} &= \boldsymbol{\delta}^T \mathbf{x}^T \\ [n \times m] \quad [n \times 1][1 \times m] & \\ &= \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} [x_1, \dots, x_m] = \begin{bmatrix} \delta_1 x_1 & \dots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \dots & \delta_n x_m \end{bmatrix} \end{aligned}$$

- Hacky answer: this makes the dimensions work out!
  - Useful trick for checking your work!
- Full explanation in the lecture notes
  - Each input goes to each output – you want to get outer product

# What shape should derivatives be?

- Similarly,  $\frac{\partial s}{\partial \mathbf{b}} = \mathbf{h}^T \circ f'(z)$  is a row vector
  - But shape convention says our gradient should be a column vector because  $\mathbf{b}$  is a column vector ...
- Disagreement between Jacobian form (which makes the chain rule easy) and the shape convention (which makes implementing SGD easy)
  - We expect answers in the assignment to follow the **shape convention**
  - But Jacobian form is useful for computing the answers

# What shape should derivatives be?

Two options:

1. Use Jacobian form as much as possible, reshape to follow the shape convention at the end:
  - What we just did. But at the end transpose  $\frac{\partial s}{\partial \mathbf{b}}$  to make the derivative a column vector, resulting in  $\delta^T$
2. Always follow the shape convention
  - Look at dimensions to figure out when to transpose and/or reorder terms
  - The error message  $\delta$  that arrives at a hidden layer has the same dimensionality as that hidden layer

# 3. Backpropagation

We've almost shown you backpropagation

It's taking derivatives and using the (generalized, multivariate, or matrix) chain rule

Other trick:

We **re-use** derivatives computed for higher layers in computing derivatives for lower layers to minimize computation

# Computation Graphs and Backpropagation

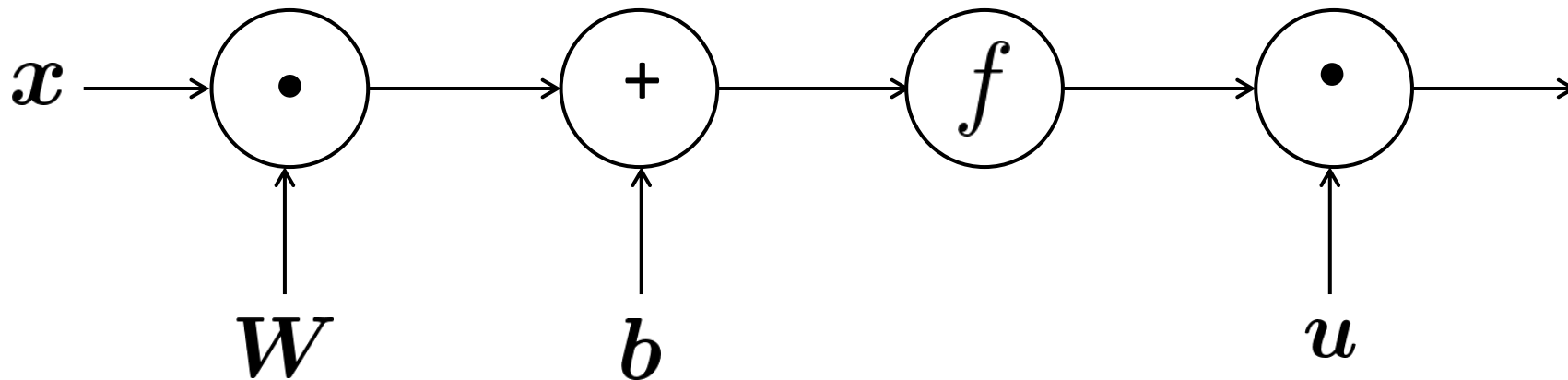
- Software represents our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations

$$s = u^T h$$

$$h = f(z)$$

$$z = \mathbf{W}x + b$$

$$x \text{ (input)}$$



# Computation Graphs and Backpropagation

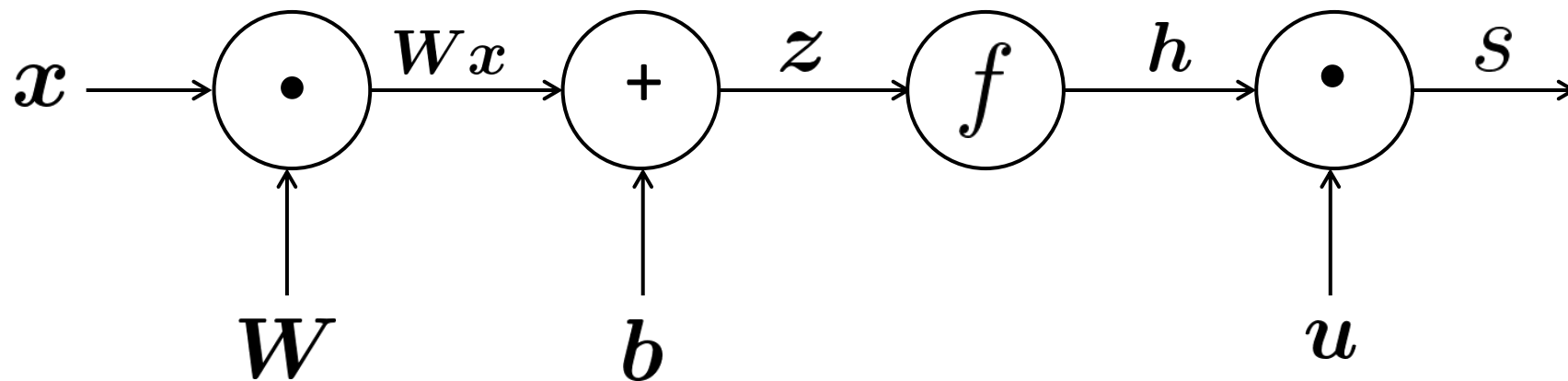
- Software represents our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

$$s = u^T h$$

$$h = f(z)$$

$$z = \mathbf{W}x + b$$

$$x \text{ (input)}$$



# Computation Graphs and Backpropagation

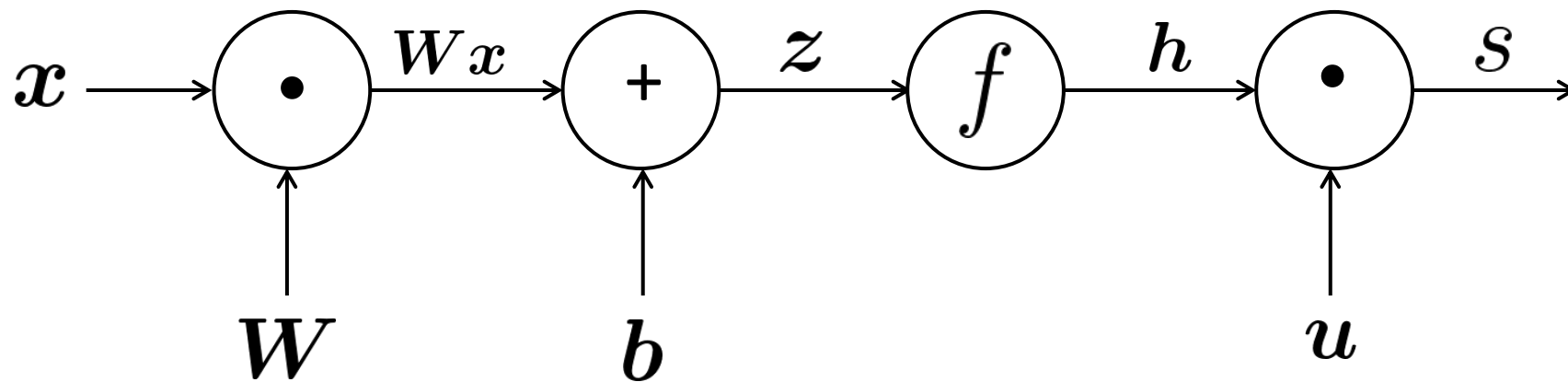
- Software represents our neural net equations as a graph

$$s = u^T h$$

$$h = f(z)$$

“Forward Propagation”

operation



# Backpropagation

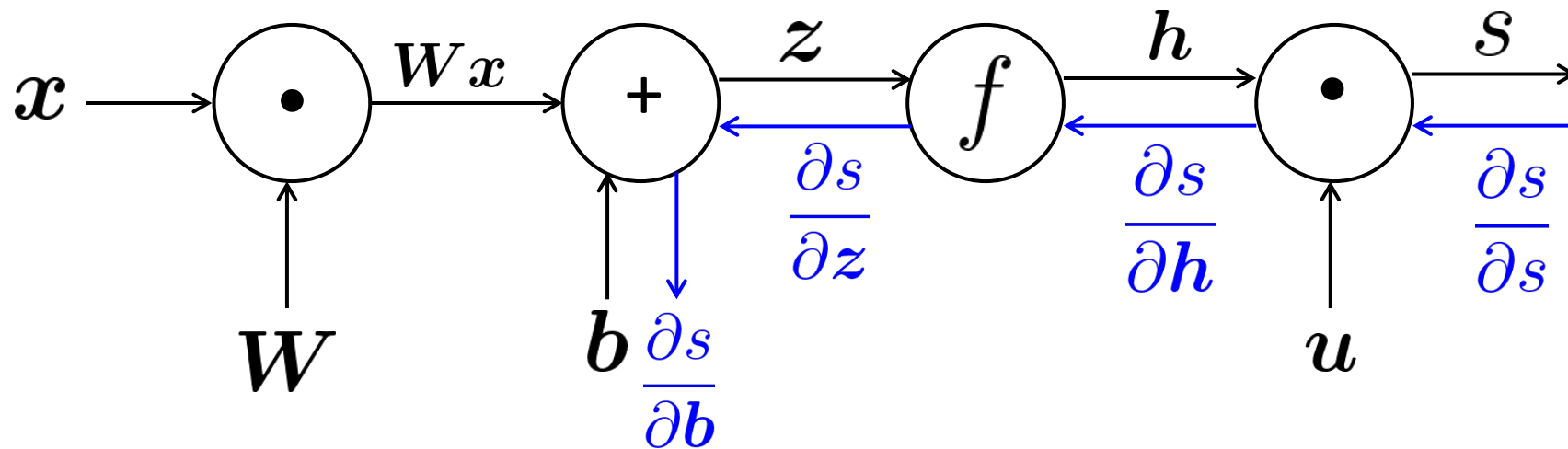
- Then go backwards along edges
  - Pass along **gradients**

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \text{ (input)}$$

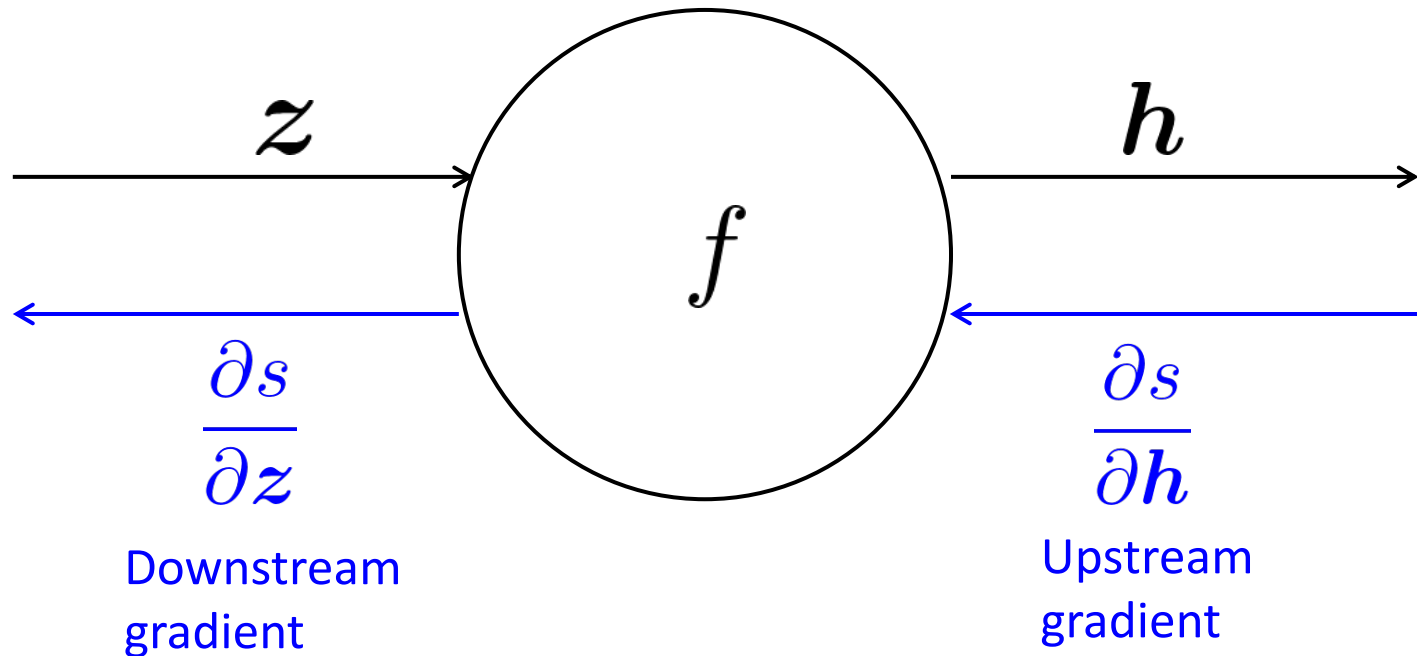




# Backpropagation: Single Node

- Node receives an “upstream gradient”
- Goal is to pass on the correct “downstream gradient”

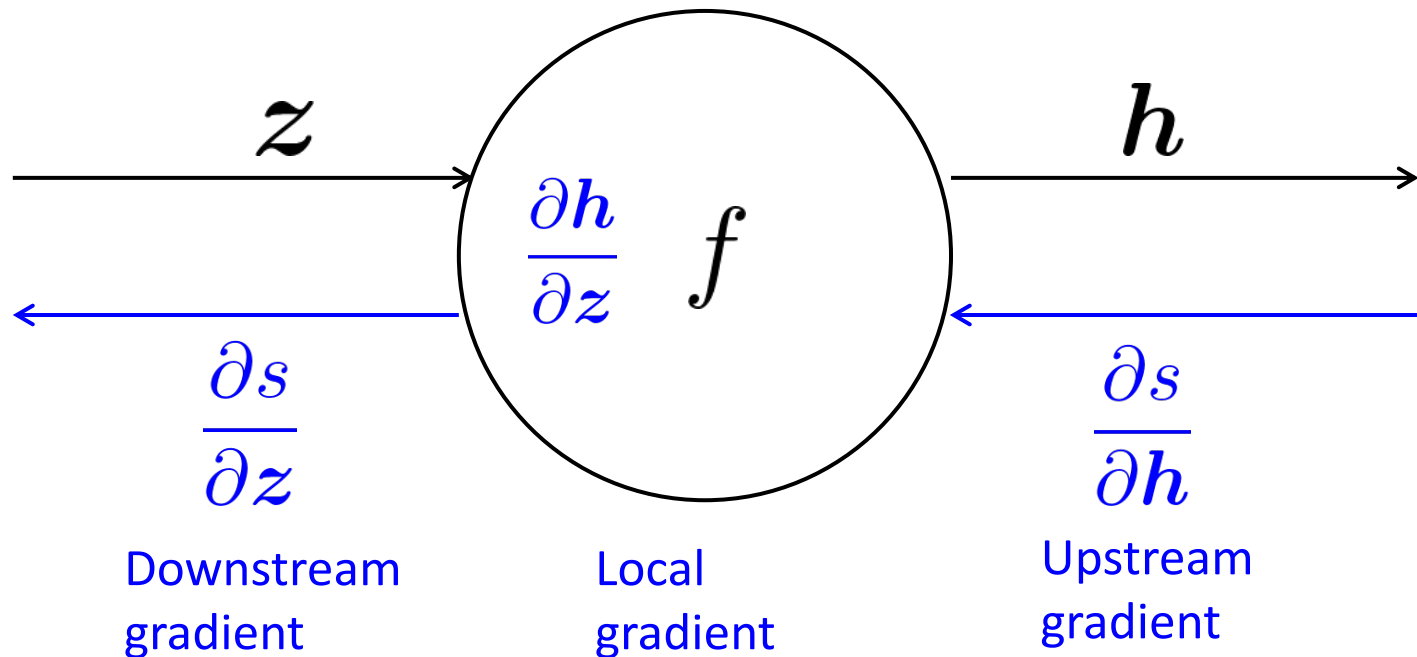
$$h = f(z)$$



# Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

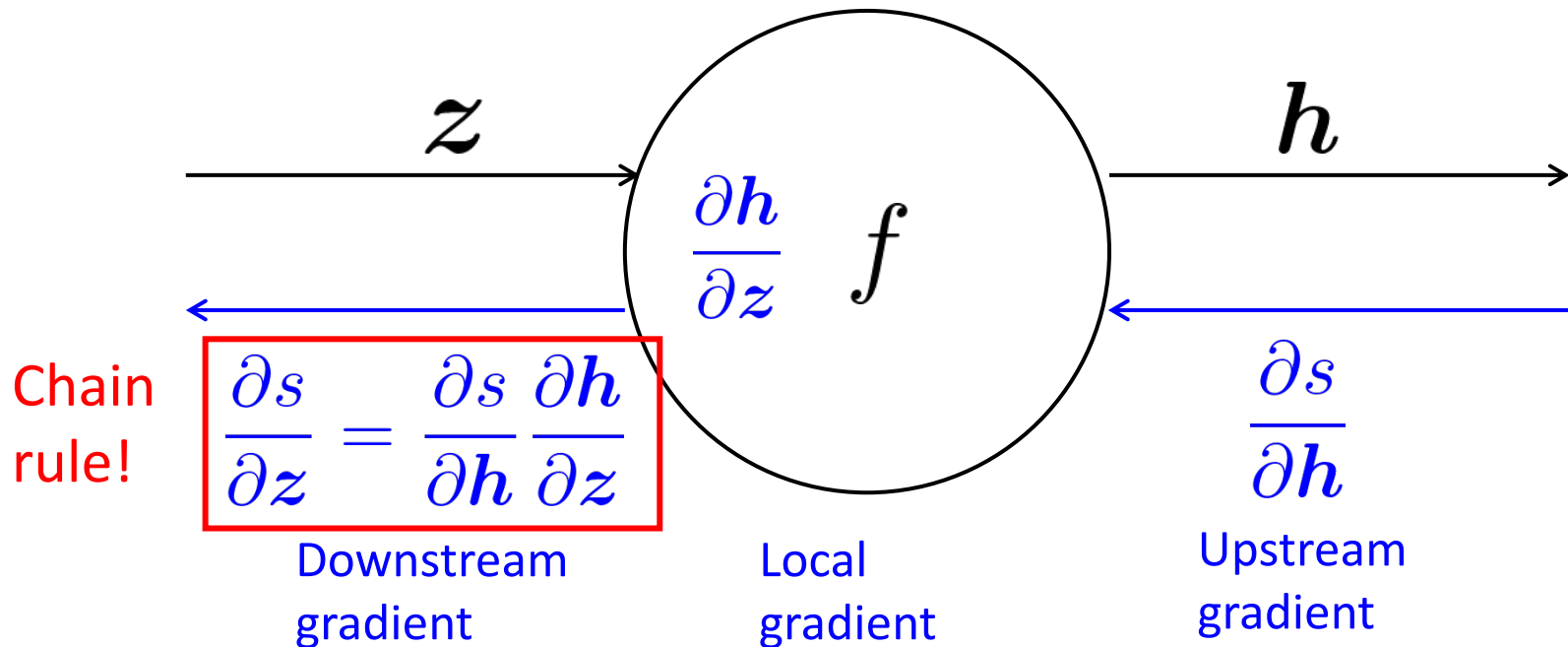
$$h = f(z)$$



# Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

$$h = f(z)$$

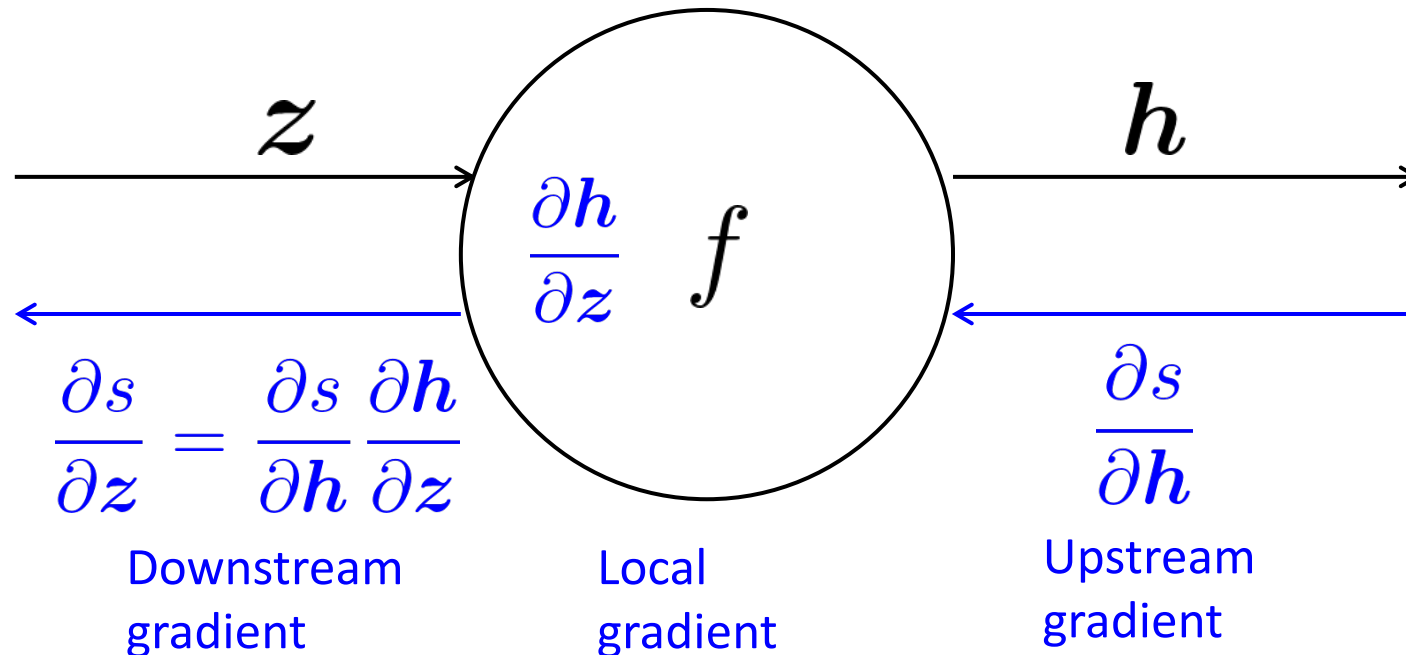


# Backpropagation: Single Node

- Each node has a **local gradient**
- The gradient of its output with respect to its input

$$h = f(z)$$

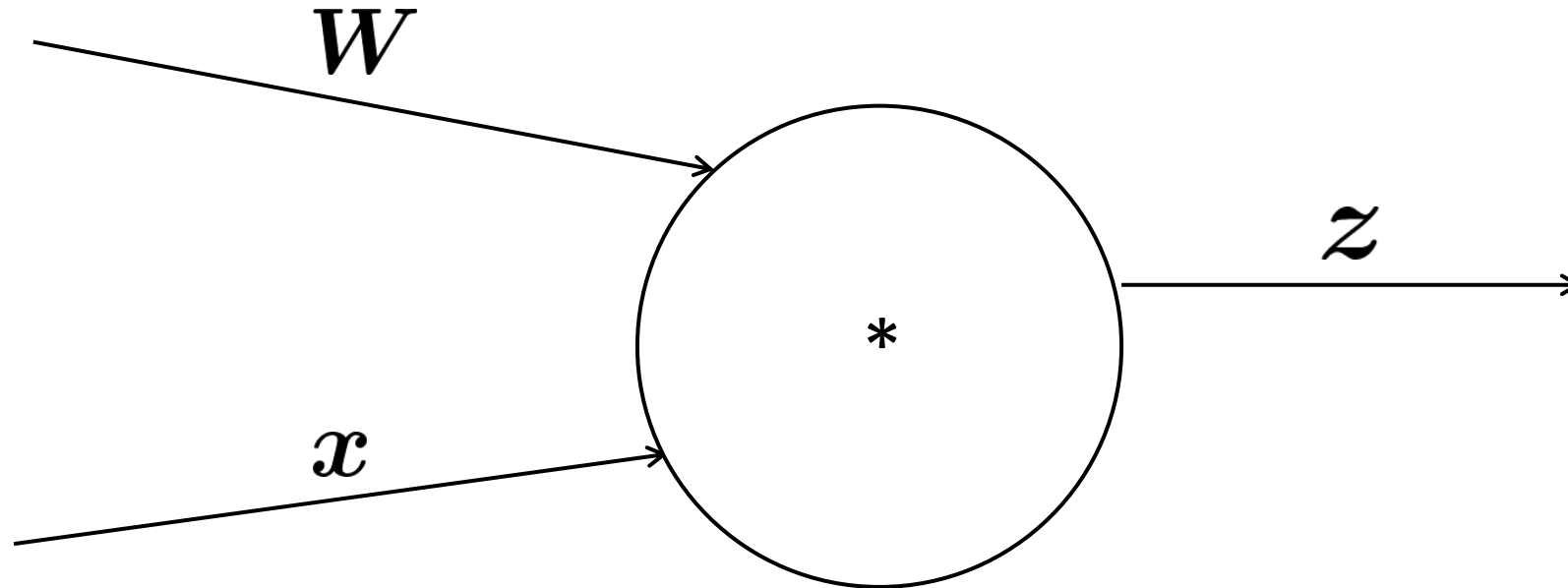
- [downstream gradient] = [upstream gradient] x [local gradient]



# Backpropagation: Single Node

- What about nodes with multiple inputs?

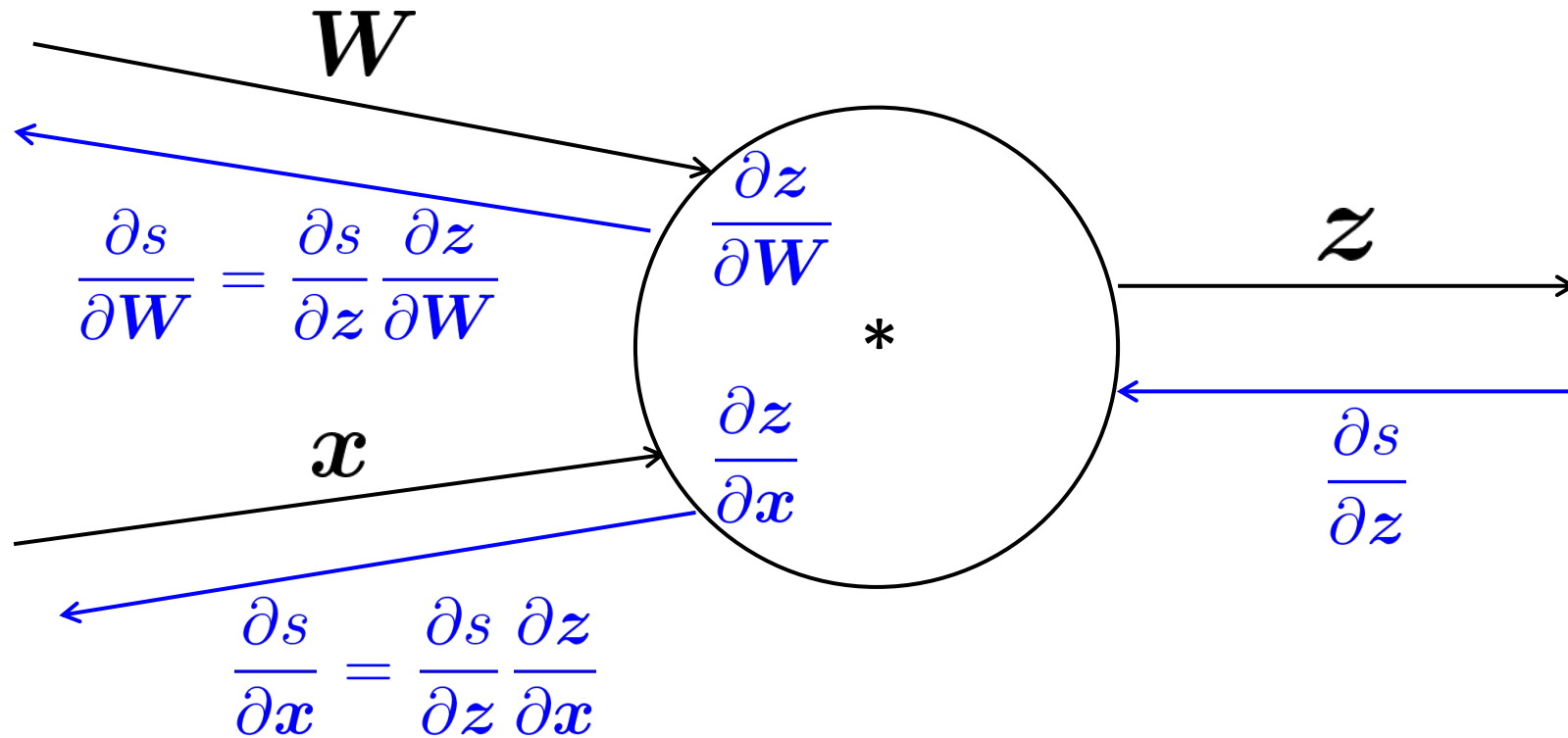
$$z = Wx$$



# Backpropagation: Single Node

- Multiple inputs  $\rightarrow$  multiple local gradients

$$z = Wx$$



Downstream  
gradients

Local  
gradients

Upstream  
gradient

## An Example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

# An Example

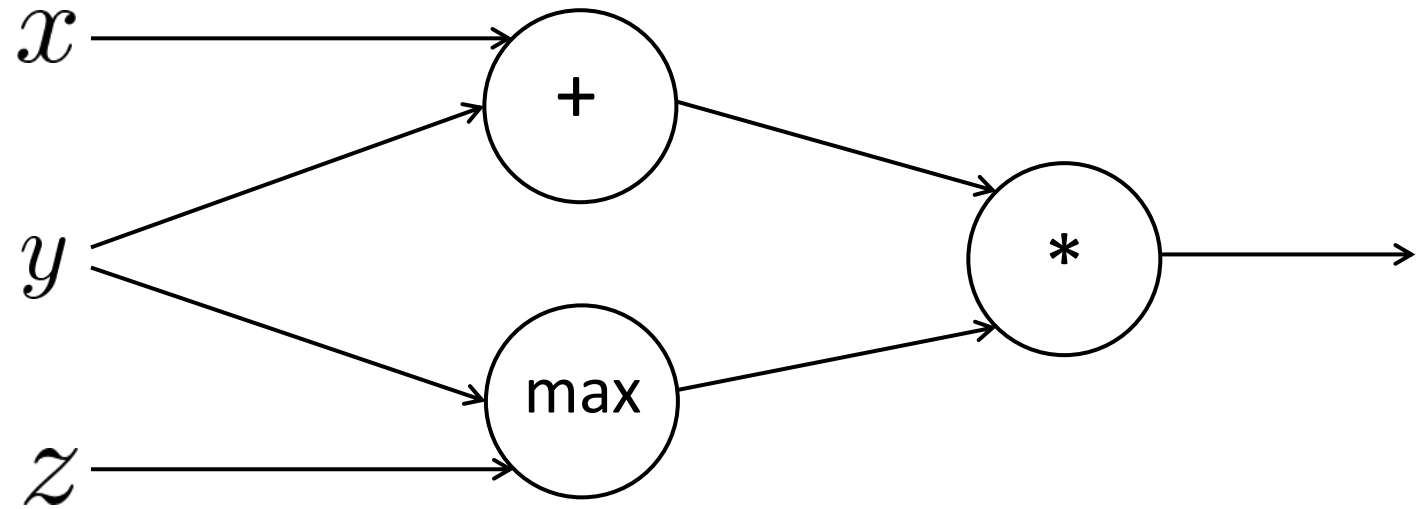
$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$





# An Example

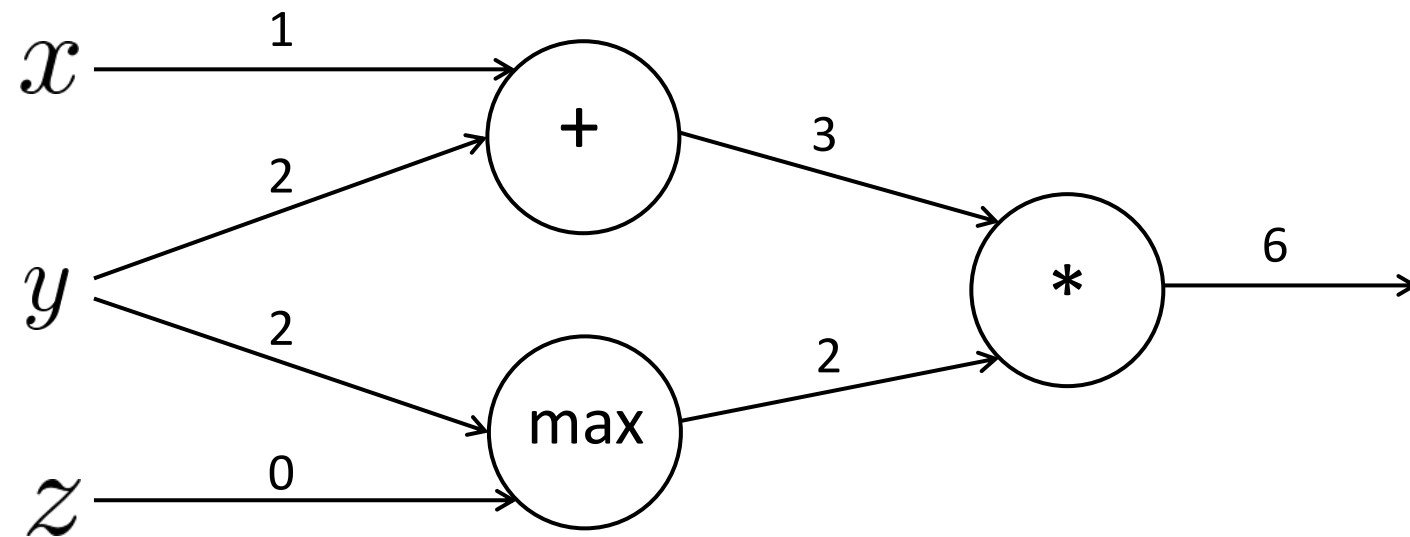
$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$



# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

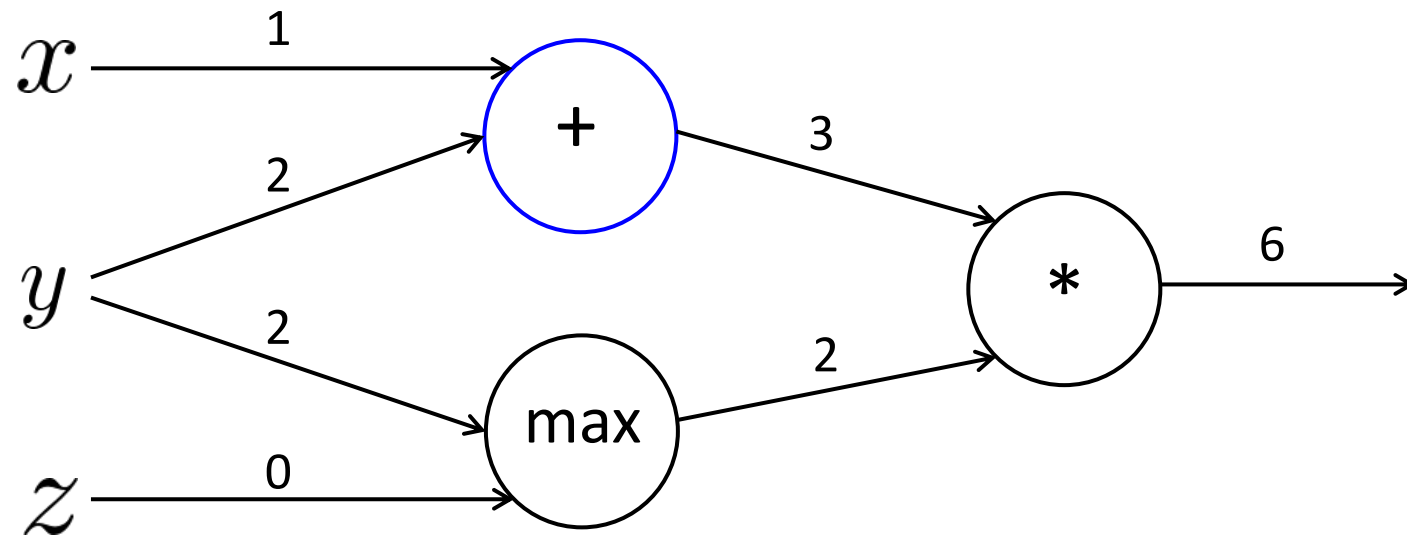
$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$



# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

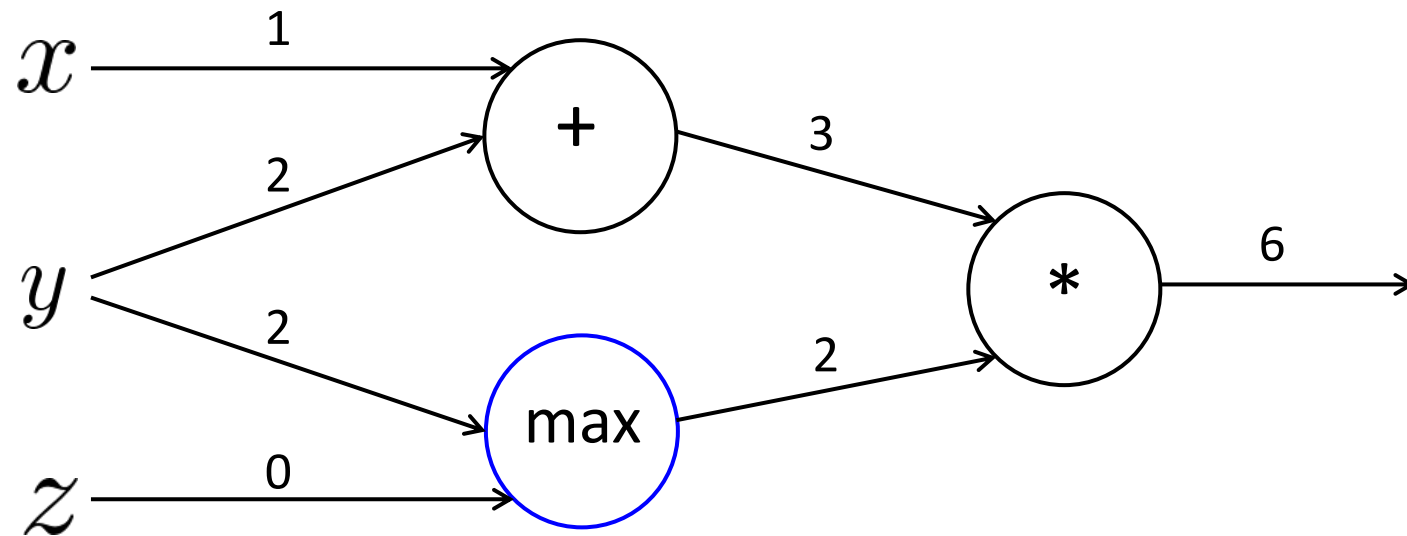
$$b = \max(y, z)$$

$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$



# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

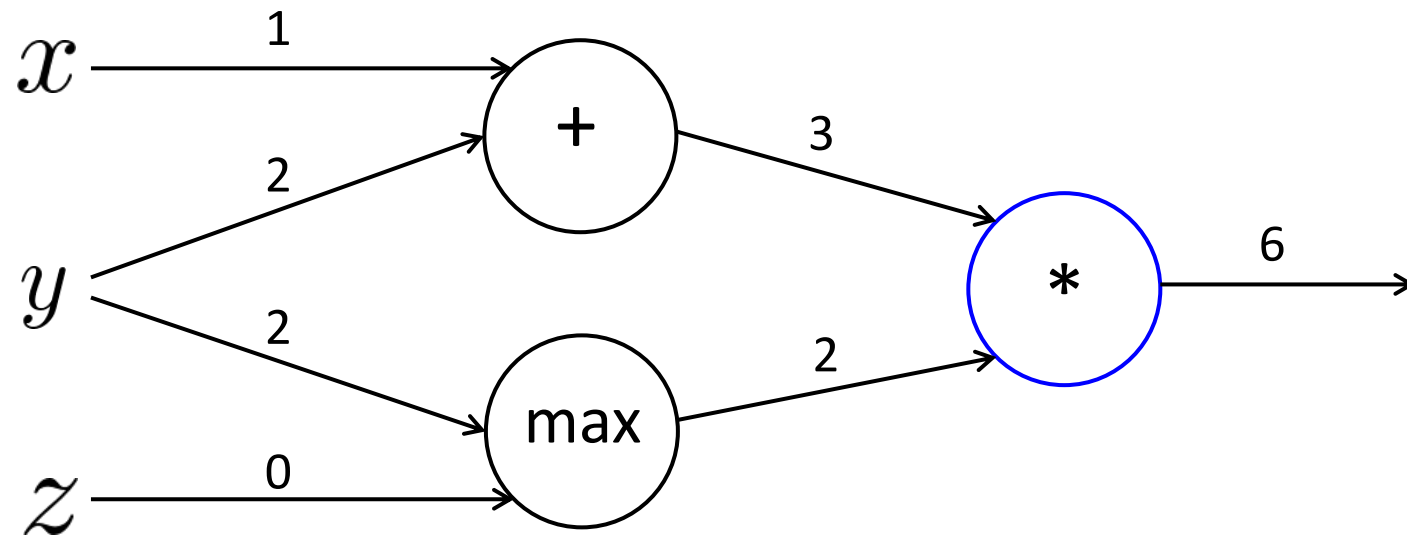
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

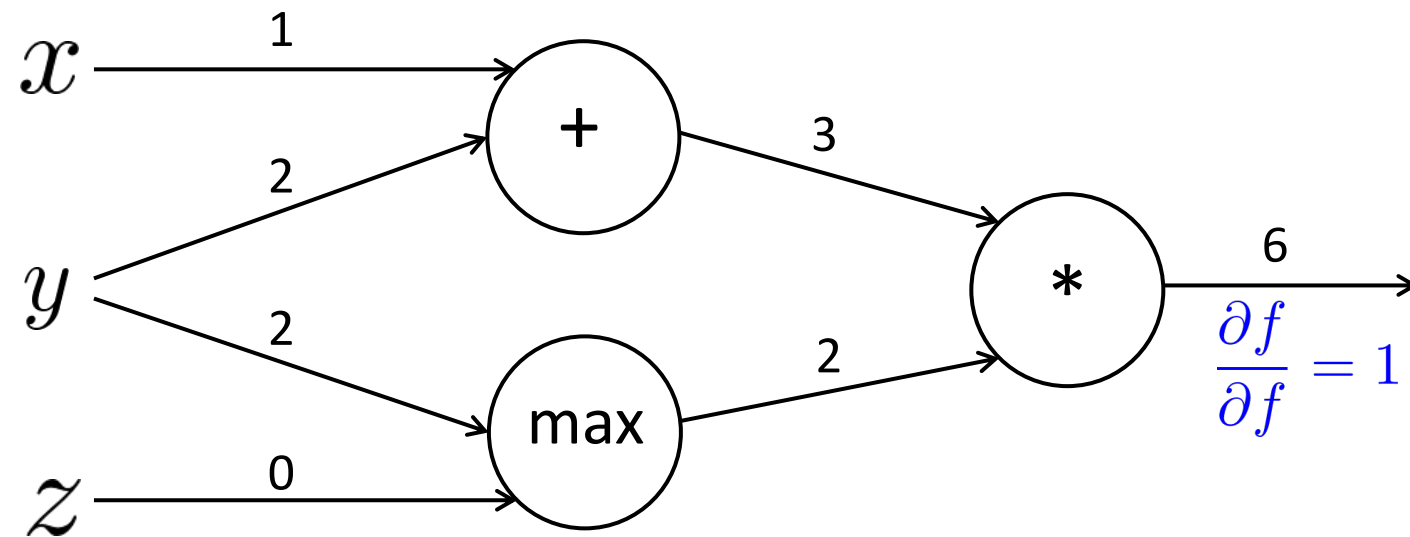
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

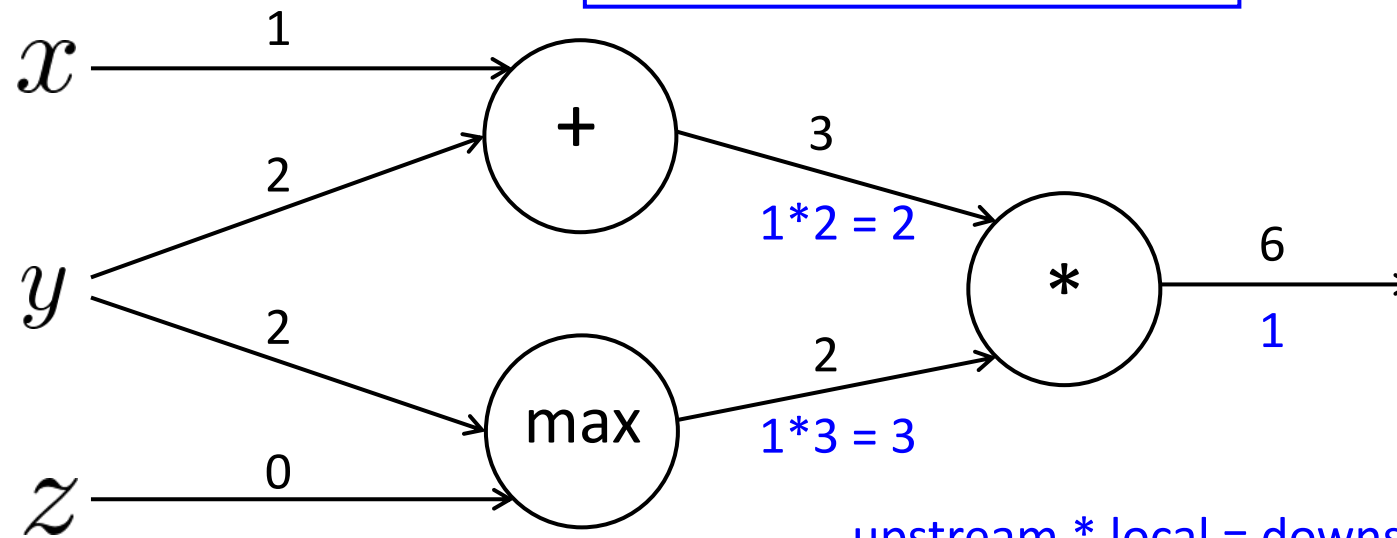
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



upstream \* local = downstream

# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$

$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

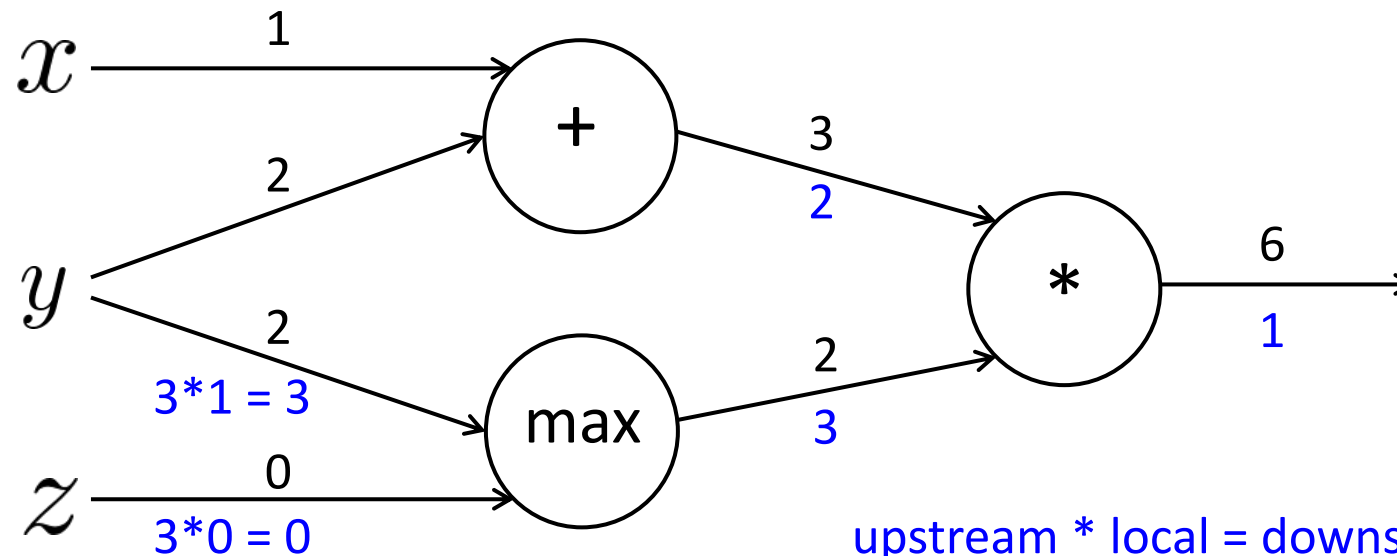
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$

$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

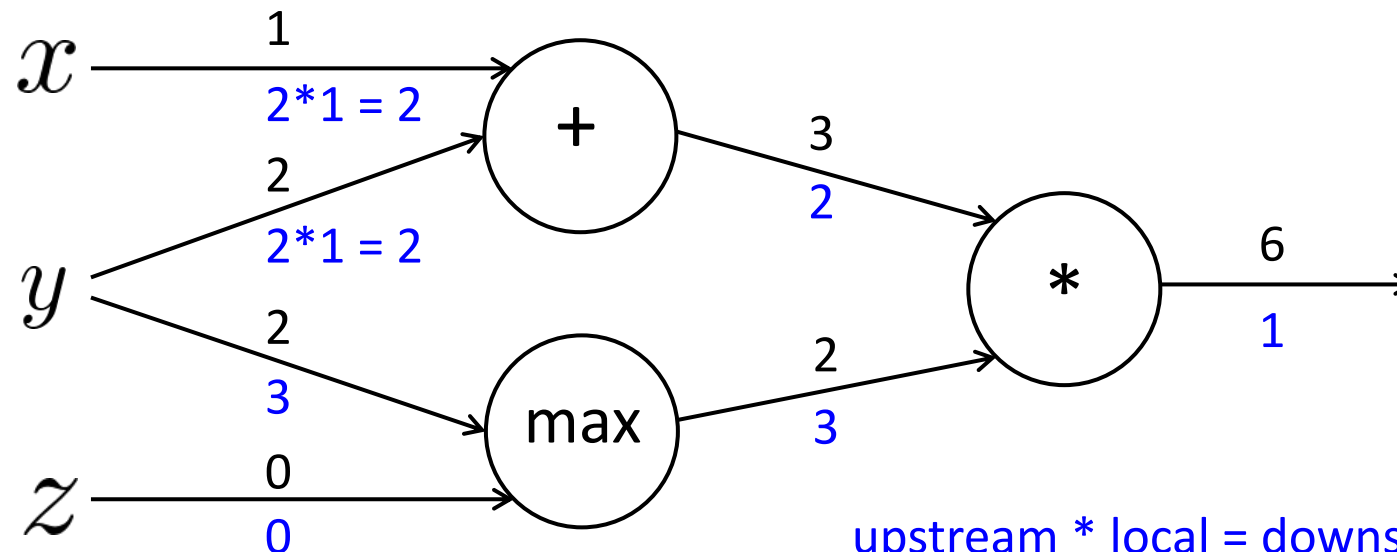
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$





# An Example

$$f(x, y, z) = (x + y) \max(y, z)$$

$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

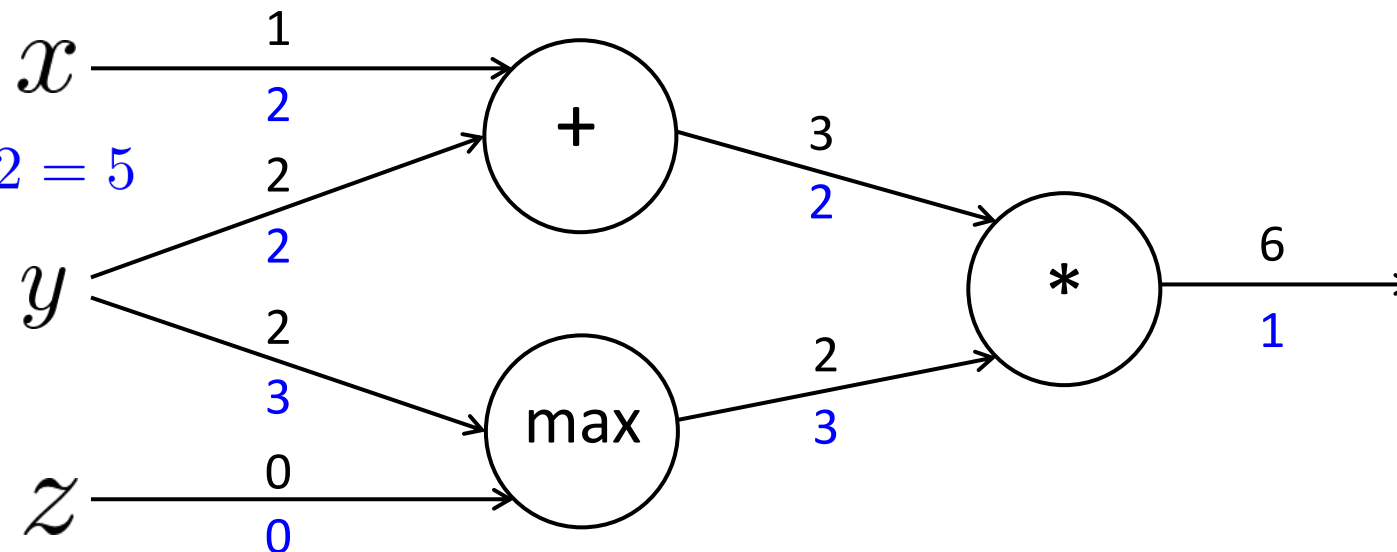
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$

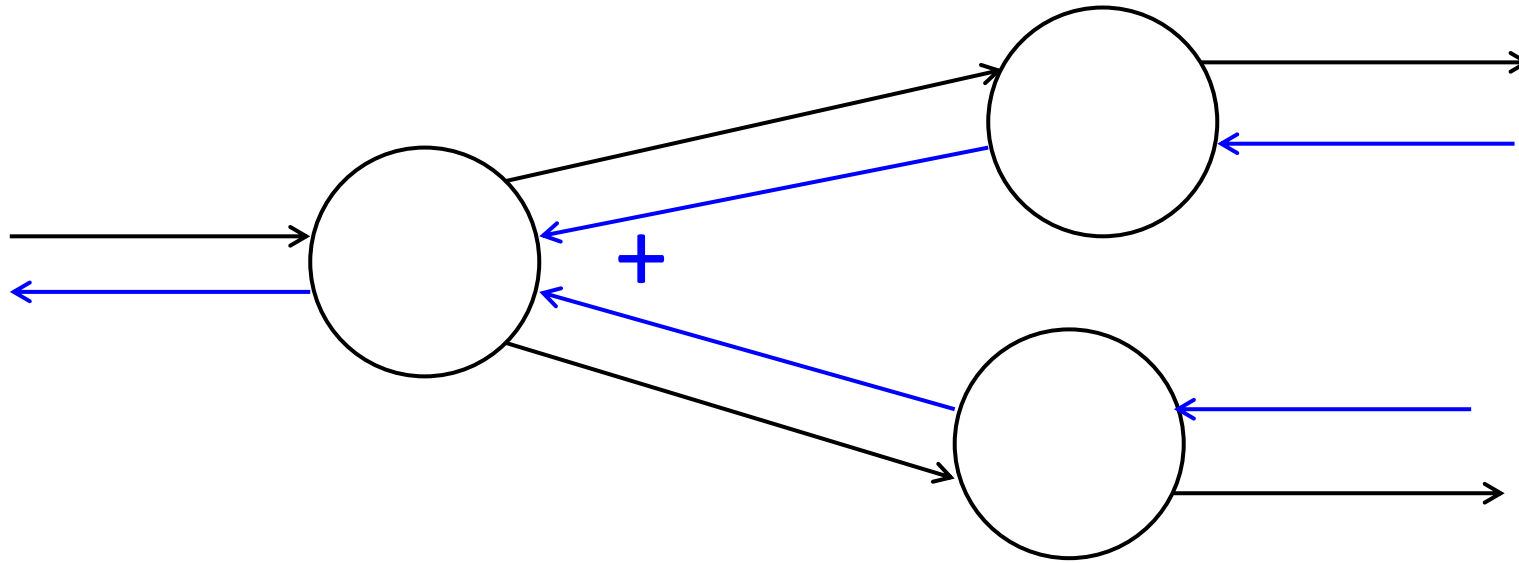
$$\frac{\partial f}{\partial x} = 2$$

$$\frac{\partial f}{\partial y} = 3 + 2 = 5$$

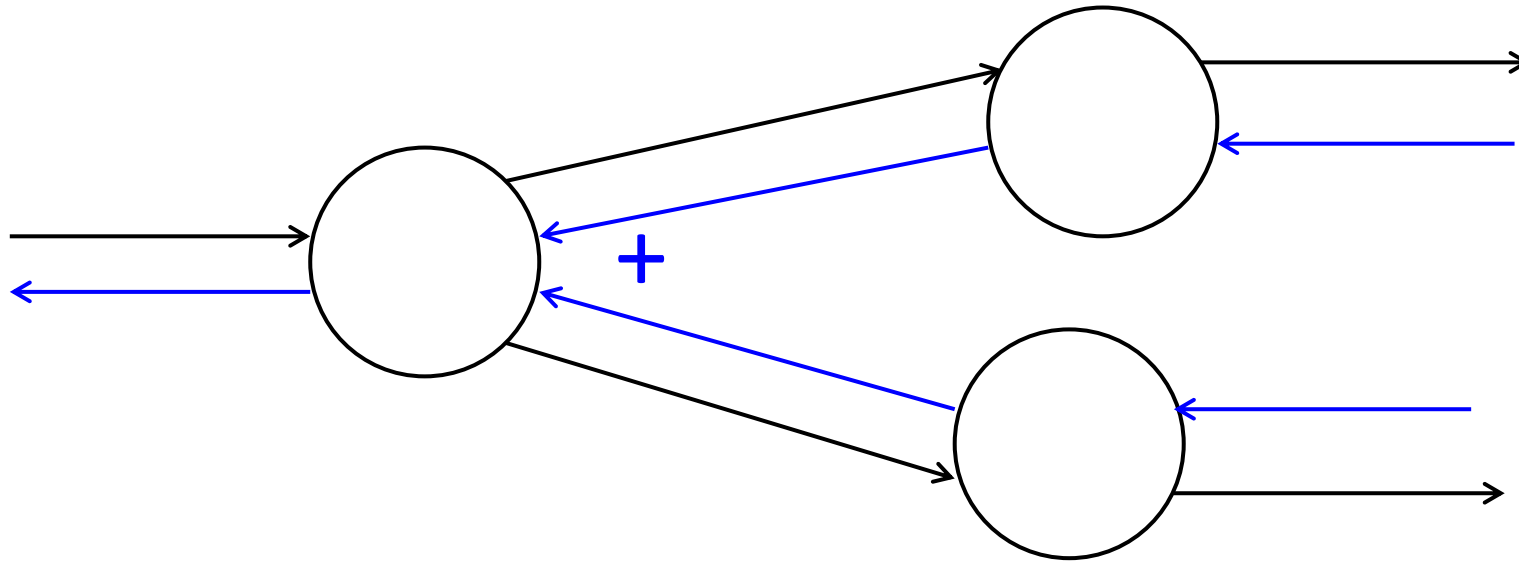
$$\frac{\partial f}{\partial z} = 0$$



# Gradients sum at outward branches



# Gradients sum at outward branches



$$a = x + y$$

$$b = \max(y, z)$$

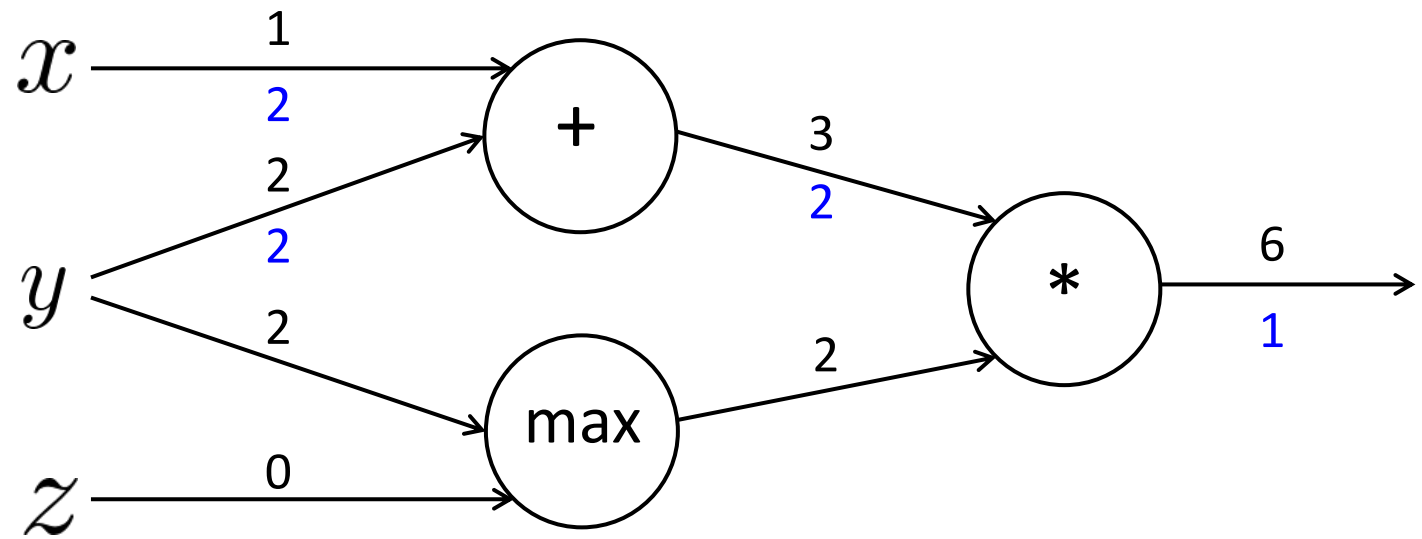
$$f = ab$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial y} + \frac{\partial f}{\partial b} \frac{\partial b}{\partial y}$$

# Node Intuitions

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

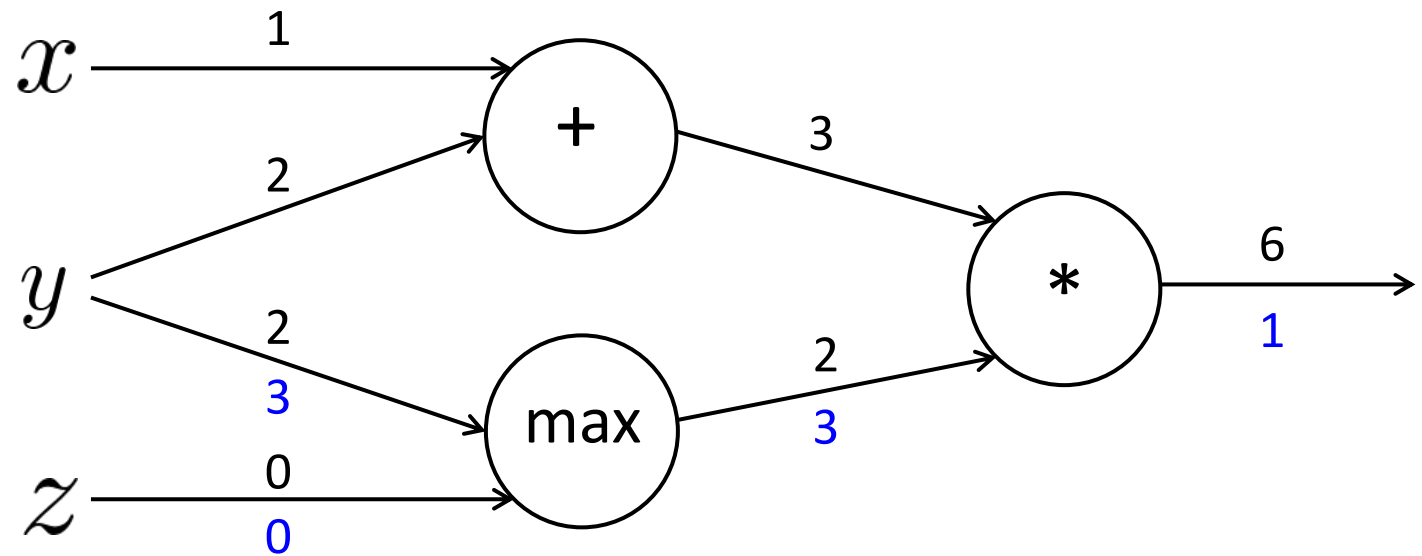
- + “distributes” the upstream gradient to each summand



# Node Intuitions

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

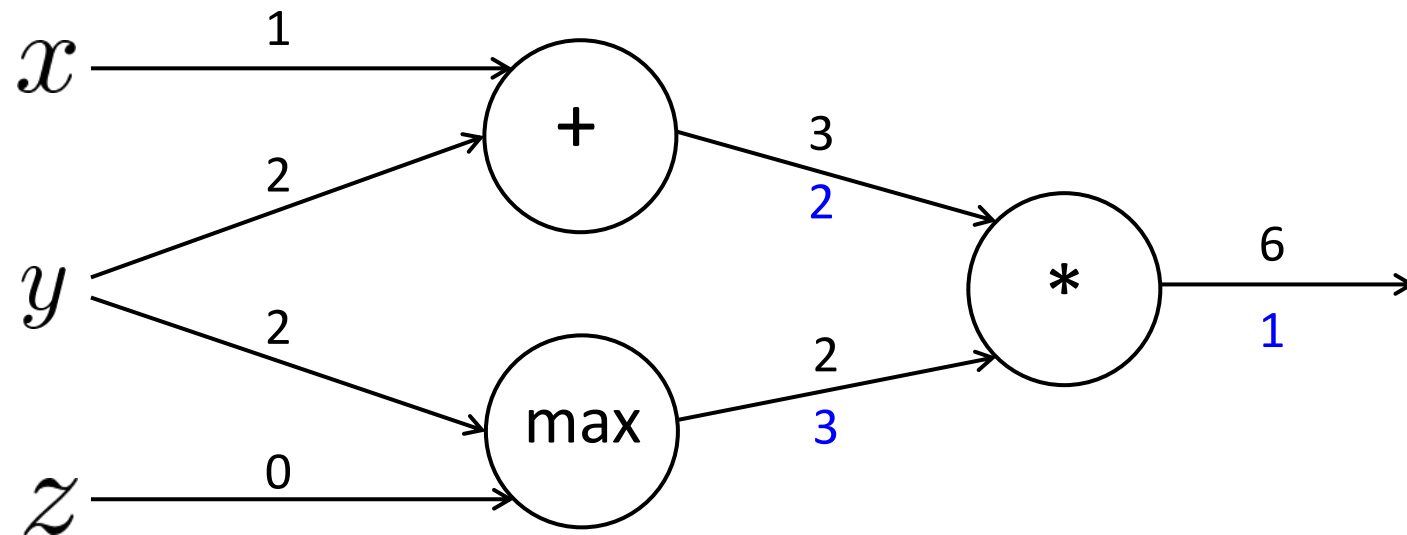
- + “distributes” the upstream gradient to each summand
- max “routes” the upstream gradient



# Node Intuitions

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

- + “distributes” the upstream gradient
- max “routes” the upstream gradient
- \* “switches” the upstream gradient



# Efficiency: compute all gradients at once

- Incorrect way of doing backprop:

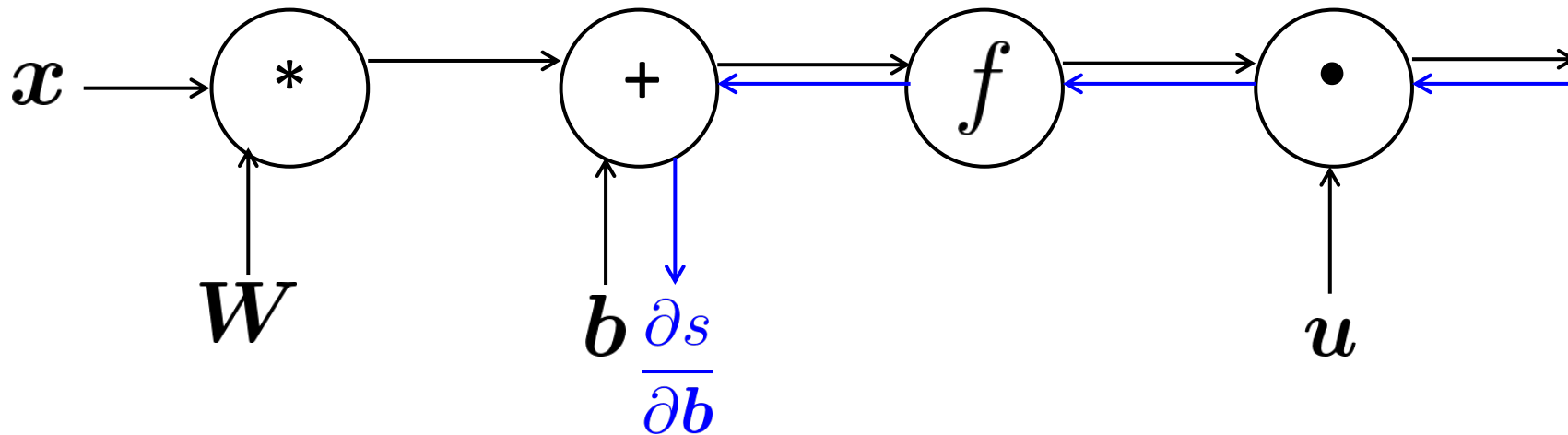
- First compute  $\frac{\partial s}{\partial b}$

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \text{ (input)}$$



# Efficiency: compute all gradients at once

- Incorrect way of doing backprop:

- First compute  $\frac{\partial s}{\partial b}$

- Then independently compute  $\frac{\partial s}{\partial W}$

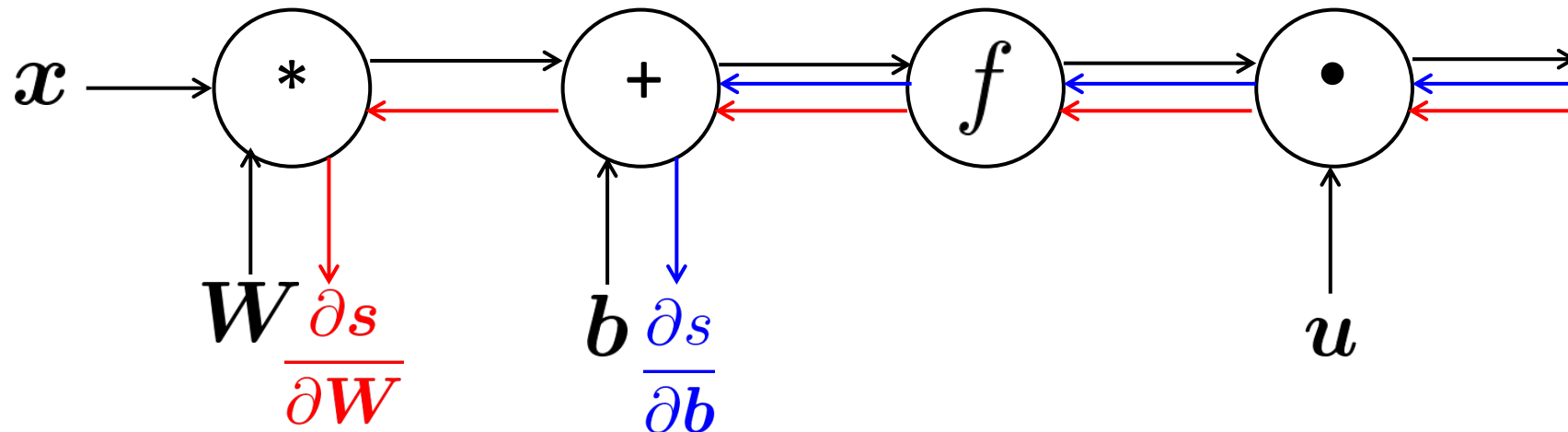
- Duplicated computation!

$$s = u^T h$$

$$h = f(z)$$

$$z = \mathbf{W}x + b$$

$$x \text{ (input)}$$





# Efficiency: compute all gradients at once

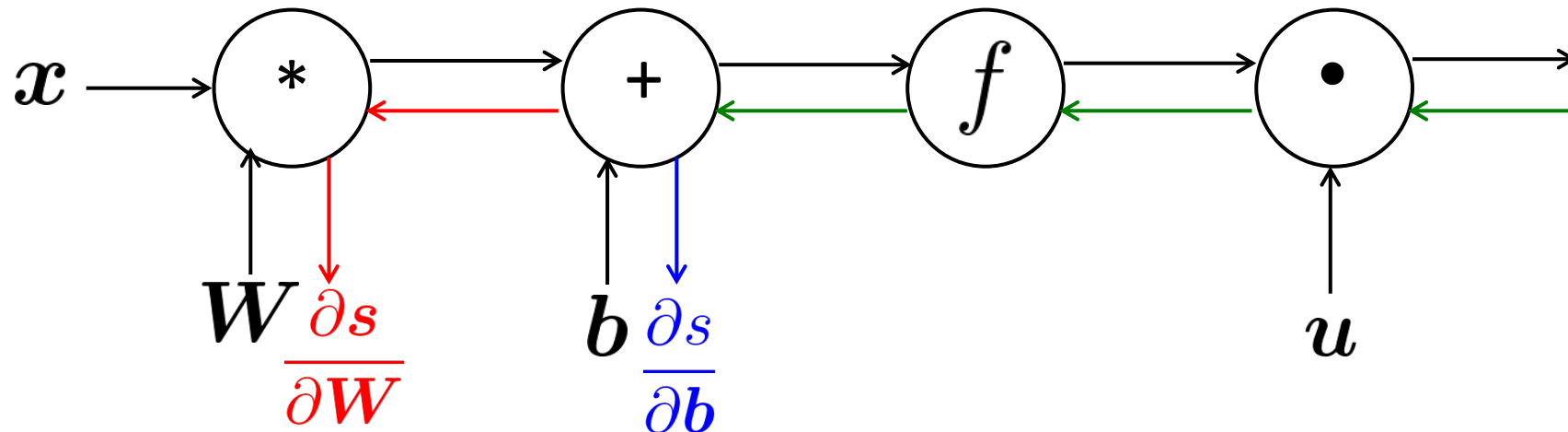
- Correct way:
  - Compute all the gradients at once
  - Analogous to using  $\delta$  when we computed gradients by hand

$$s = u^T h$$

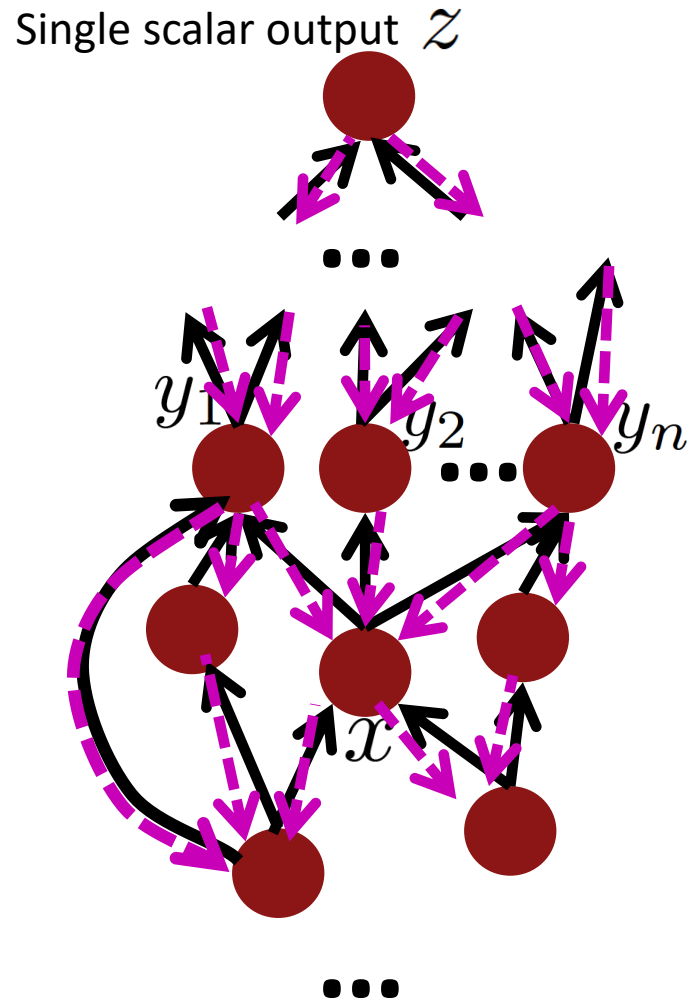
$$h = f(z)$$

$$z = \mathbf{W}x + b$$

$$x \text{ (input)}$$



# Back-Prop in General Computation Graph



1. Fprop: visit nodes in topological sort order
  - Compute value of node given predecessors
2. Bprop:

- initialize output gradient = 1

- visit nodes in reverse order:

Compute gradient wrt each node using gradient wrt successors

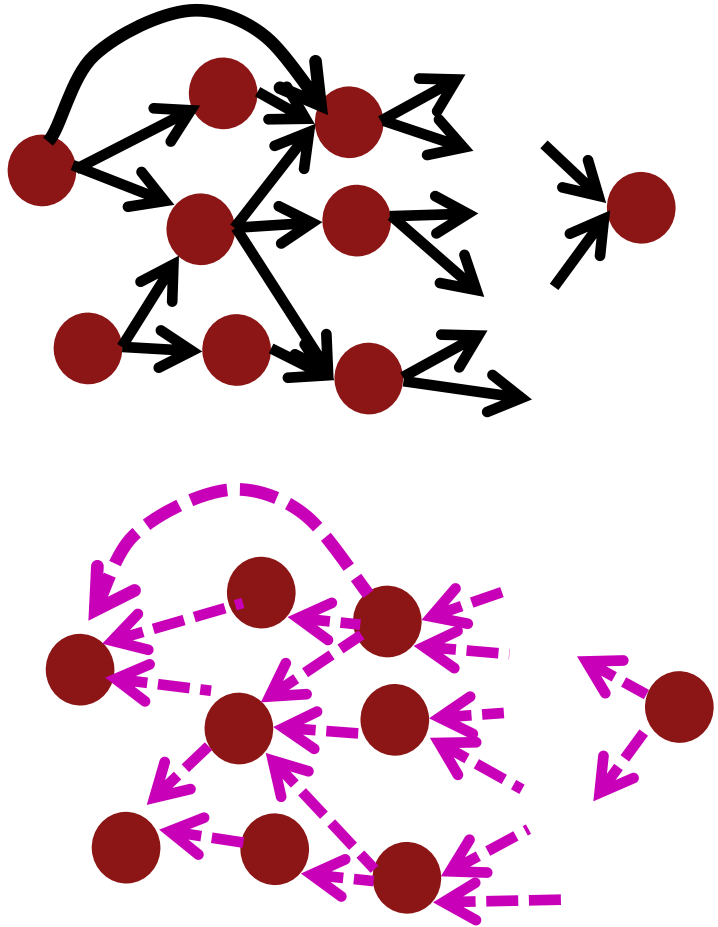
$\{y_1, y_2, \dots, y_n\}$  = successors of  $x$

$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Done correctly, big  $O()$  complexity of fprop and bprop is **the same**

In general, our nets have regular layer-structure and so we can use matrices and Jacobians...

# Automatic Differentiation

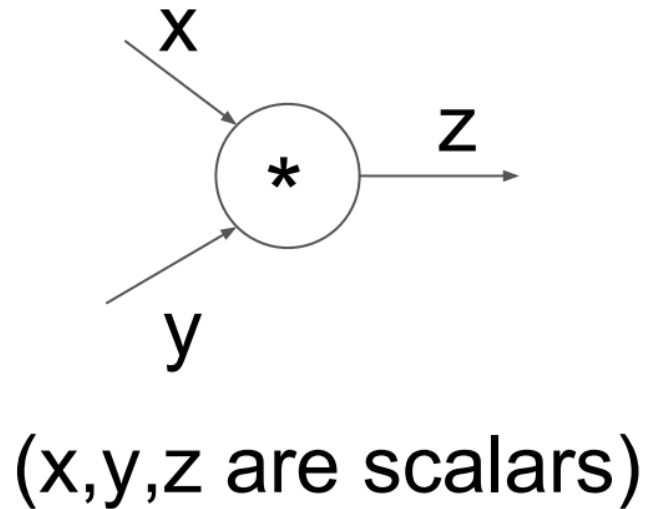


- The gradient computation can be automatically inferred from the symbolic expression of the fprop
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you but mainly leave layer/node writer to hand-calculate the local derivative

# Backprop Implementations

```
class ComputationalGraph(object):  
    #...  
    def forward(inputs):  
        # 1. [pass inputs to input gates...]  
        # 2. forward the computational graph:  
        for gate in self.graph.nodes_topologically_sorted():  
            gate.forward()  
        return loss # the final gate in the graph outputs the loss  
    def backward():  
        for gate in reversed(self.graph.nodes_topologically_sorted()):  
            gate.backward() # little piece of backprop (chain rule applied)  
        return inputs_gradients
```

# Implementation: forward/backward API



```
class MultiplyGate(object):
```

```
    def forward(x,y):
```

```
        z = x*y
```

```
        return z
```

```
    def backward(dz):
```

```
        # dx = ... #todo
```

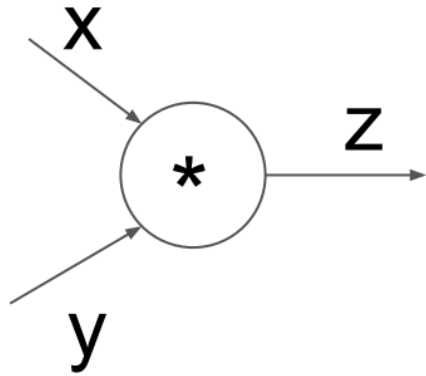
```
        # dy = ... #todo
```

```
        return [dx, dy]
```

$$\frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial x}$$

# Implementation: forward/backward API



(x,y,z are scalars)

```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        self.x = x # must keep these around!  
        self.y = y  
        return z  
    def backward(dz):  
        dx = self.y * dz # [dz/dx * dL/dz]  
        dy = self.x * dz # [dz/dy * dL/dz]  
        return [dx, dy]
```

# Manual Gradient checking: Numeric Gradient

- For small  $h$  ( $\approx 1e-4$ ),
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$
- Easy to implement correctly
- But approximate and **very** slow:
  - You have to recompute  $f$  for **every parameter** of our model
- Useful for checking your implementation
  - In the old days, we hand-wrote everything, doing this everywhere was the key test
  - Now much less needed; you can use it to check layers are correctly implemented

# Summary

**We've mastered the core technology of neural nets!** 🎉

- Backpropagation: recursively (and hence efficiently) apply the chain rule along computation graph
  - $[\text{downstream gradient}] = [\text{upstream gradient}] \times [\text{local gradient}]$
- Forward pass: compute results of operations and save intermediate values
- Backward pass: apply chain rule to compute gradients



# Why learn all these details about gradients?

- Modern deep learning frameworks compute gradients for you!
  - Come to the PyTorch introduction this Friday!
- But why take a class on compilers or systems when they are implemented for you?
  - Understanding what is going on under the hood is useful!
- Backpropagation doesn't always work perfectly
  - Understanding why is crucial for debugging and improving models
  - See Karpathy article (in syllabus):
    - <https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b>
  - Example in future lecture: exploding and vanishing gradients