

# Training Neural Networks

CMPUT 261: Introduction to Artificial Intelligence

P §7.1-7.4.1

# Lecture Outline

1. Recap & Logistics
2. Gradient Descent for Neural Networks
3. Automatic Differentiation
4. Back-Propagation

*After this lecture, you should be able to:*

- trace an execution of forward-mode automatic differentiation
- trace an execution of backward-mode automatic differentiation
- construct a finite numerical algorithm for a given computation
- explain why automatic differentiation is more efficient than the method of finite differences
- explain why automatic differentiation is more efficient than symbolic differentiation
- explain why backward mode automatic differentiation is more efficient for typical deep learning applications

# Logisitics

- **Assignment #3** will be released today
  - Due **Tuesday, November 21**
  - Submit via eClass
- **Midterm:** still being marked
  - Progress bar: [===== ] 75%

# Recap: Nonlinear Features

$$y = f(\mathbf{x}; \mathbf{w}) = g(\mathbf{w}^T \mathbf{x}) = g\left(\sum_{i=1}^n w_i x_i\right)$$

**Generalized linear model:** **Activation function**  $g$  of linear combination of inputs

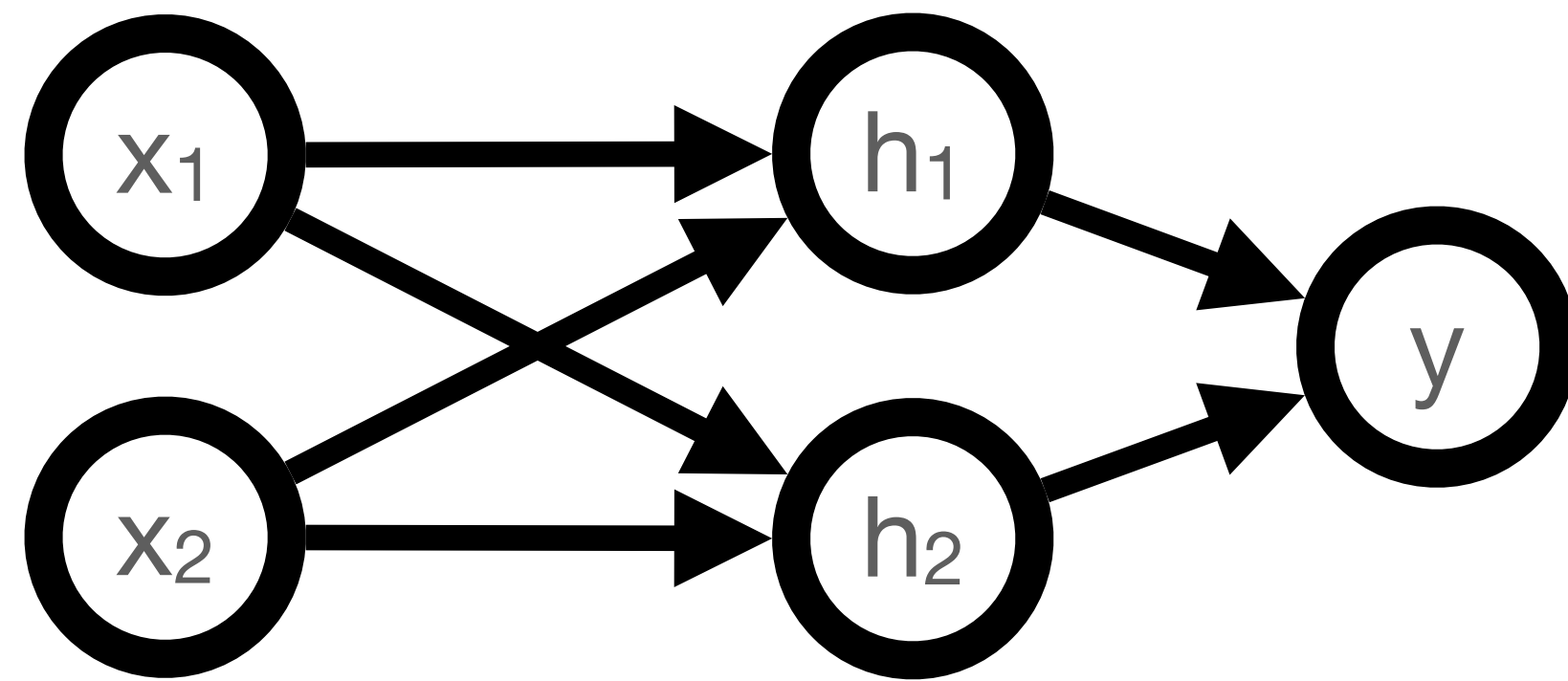
**Extension:** Learn a generalized linear model on **richer inputs**

1. Define a **feature mapping**  $\phi(\mathbf{x})$  that returns **functions** of the original inputs
2. Learn a linear model of the **features** instead of the **inputs**

$$y = f(\mathbf{x}; \mathbf{w}) = g(\mathbf{w}^T \phi(\mathbf{x})) = g\left(\sum_{i=1}^n w_i [\phi(\mathbf{x})]_i\right)$$

# Recap:

## Feedforward Neural Network



$$h_1(\mathbf{x}; \mathbf{w}^{(1)}, b^{(1)}) = g \left( b^{(1)} + \sum_{i=1}^n w_i^{(1)} x_i \right)$$

$$y(\mathbf{x}; \mathbf{w}, \mathbf{b}) = g \left( b^{(y)} + \sum_{i=1}^n w_i^{(y)} h_i(\mathbf{x}_i; \mathbf{w}^{(i)}, b^{(i)}) \right)$$

- A **neural network** is many **units composed** together
- **Feedforward neural network:**  
Units arranged into **layers**
  - Each layer takes outputs of **previous layer** as its **inputs**

# Recap: Chain Rule of Calculus

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

i.e.,

if  $z = h(x) = f(g(x))$  and  $y = g(x)$

$$h(x) = f(g(x)) \implies h'(x) = f'(g(x))g'(x)$$

If we know formulas for the derivatives of **components** of a function, then we can build up the derivative of their composition mechanically

# Chain Rule of Calculus: Multiple Intermediate Arguments

What if  $h(x) = f(g_1(x), g_2(x))$ ?

$$\frac{dh}{dx} = \frac{\partial f}{\partial g_1} \frac{dg_1}{dx} + \frac{\partial f}{\partial g_2} \frac{dg_2}{dx}$$

$$\text{i.e., } h'(x) = g_1'(x) \frac{\partial f(t_1, t_2)}{\partial t_1} \bigg|_{\substack{t_1 = g_1(x) \\ t_2 = g_2(x)}} + g_2'(x) \frac{\partial f(t_1, t_2)}{\partial t_2} \bigg|_{\substack{t_1 = g_1(x) \\ t_2 = g_2(x)}}$$

# Recap: Training Neural Networks

- Specify a **loss**  $L$  and a set of **training examples**:

$$E = (\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})$$

- Training by **gradient descent**:

1. Compute **loss** on training data:  $L(\mathbf{W}, \mathbf{b}) = \sum_i \ell \left( \underbrace{f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b})}_{\text{Prediction}}, \underbrace{y^{(i)}}_{\text{Target}} \right)$

Loss function (e.g., squared error)

2. Compute **gradient** of loss:  $\nabla L(\mathbf{W}, \mathbf{b})$

3. **Update parameters** to make loss smaller:

$$\begin{bmatrix} \mathbf{W}^{new} \\ \mathbf{b}^{new} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^{old} \\ \mathbf{b}^{old} \end{bmatrix} - \eta \nabla L(\mathbf{W}^{old}, \mathbf{b}^{old})$$



# Three Representations

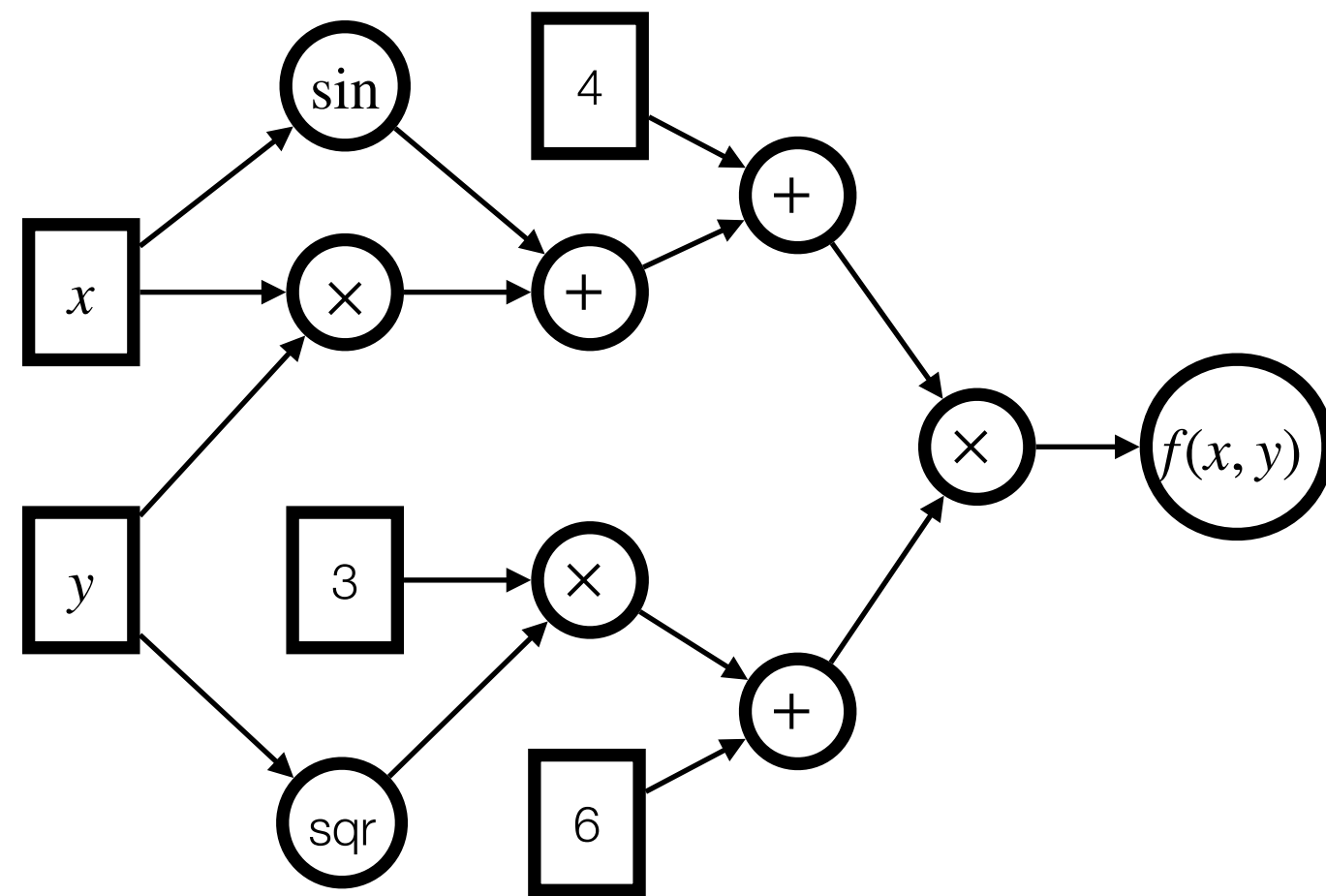
A function  $f(x, y)$  can be represented in multiple ways:

1. As a **formula**:

$$f(x, y) = (xy + \sin x + 4)(3y^2 + 6)$$

3. As a **finite numerical algorithm**

2. As a **computational graph**:



$$s_1 = x$$

$$s_2 = y$$

$$s_3 = s_1 \times s_2$$

$$s_4 = \sin(s_1)$$

$$s_5 = s_3 + s_4$$

$$s_6 = s_5 + 4$$

$$s_7 = \text{sqr}(s_2)$$

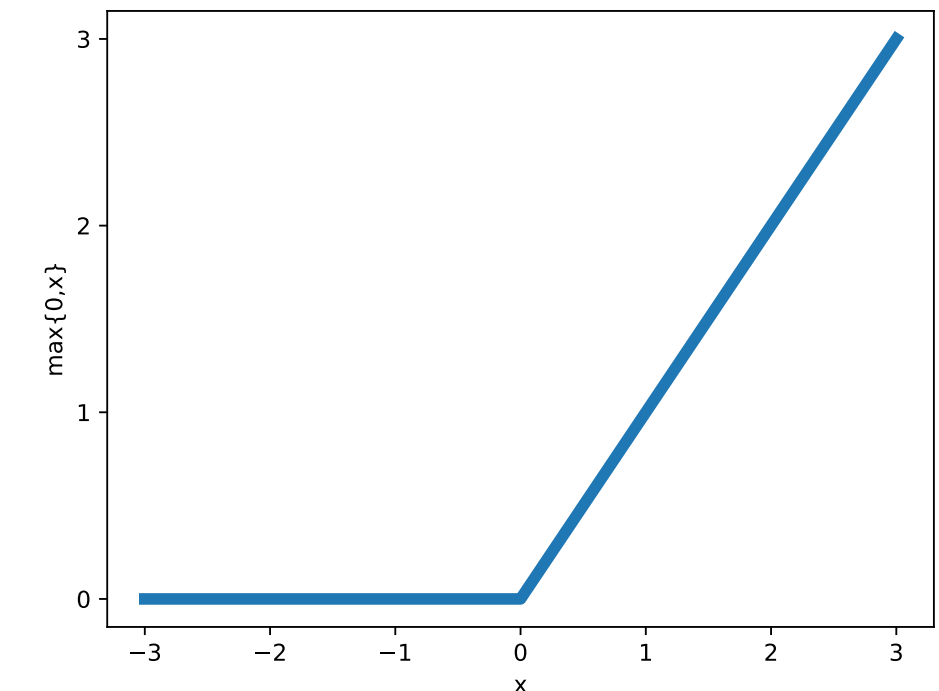
$$s_8 = 3 \times s_7$$

$$s_9 = s_8 + 6$$

$$s_{10} = s_6 \times s_9$$

# Symbolic Differentiation

$$\begin{array}{ll} z = f(y) & \\ y = f(x) & z = f(f(f(w))) \\ x = f(w) & \end{array} \quad \frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} = f'(f(f(w)))f'(f(w))f'(w)$$



$$f(w) = \begin{cases} w & \text{if } w > 0 \\ 0 & \text{otherwise.} \end{cases}$$

- We can differentiate a nested **formula** by recursively applying the **chain rule** to derive a **new formula** for the gradient
- **Problem:** This can result in a lot of repeated subexpressions
- **Question:** What happens if the nested function is defined **piecewise**?

# Automatic Differentiation: Forward Mode

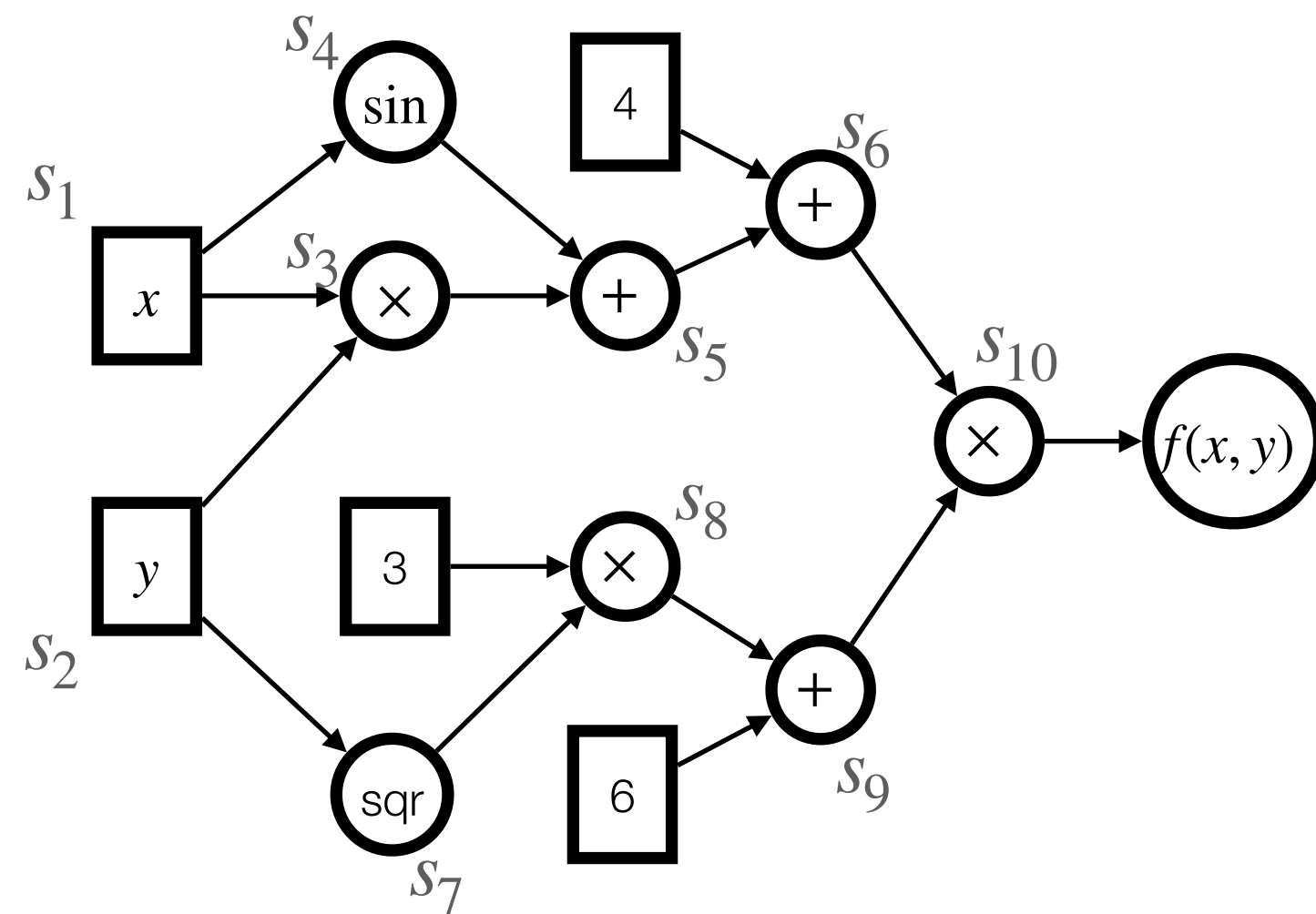
- The forward mode converts a **finite numerical algorithm** for computing a function into an **augmented** finite numerical algorithm for computing the **function's derivative**
- For each step, a new step is constructed representing the derivative of the corresponding step in the original program:

$$\begin{array}{l} s_1 = x \\ s_2 = y \\ s_3 = s_1 + s_2 \\ s_4 = s_1 \times s_2 \\ \vdots \end{array} \quad \Longrightarrow \quad \begin{array}{l} s'_1 = 1 \\ s'_2 = 0 \\ s'_3 = s'_1 + s'_2 \\ s'_4 = s_1 \times s'_2 + s'_1 \times s_2 \\ \vdots \end{array}$$

- To compute the partial derivative  $\frac{\partial s_n}{\partial s_1}$ , set  $s'_1 = 1$  and  $s'_2 = 0$  and run augmented algorithm
- This takes roughly twice as long to run as the original algorithm (**why?**)

# Forward Mode Example

Let's compute  $\left. \frac{\partial f}{\partial y} \right|_{x=2, y=8}$  using forward mode:



**Question:** What is the problem with this approach for **neural networks**?

$s_1 = x$	$= 2$	$s'_1 = 0$
$s_2 = y$	$= 8$	$s'_2 = 1$
$s_3 = s_1 \times s_2$	$= 16$	$s'_3 = s_1 \times s'_2 + s'_1 \times s_2 = 2$
$s_4 = \sin(s_1)$	$\approx 0.034$	$s'_4 = \cos(s_1) \times s'_1 = 0$
$s_5 = s_3 + s_4$	$= 16.034$	$s'_5 = s'_3 + s'_4 = 2$
$s_6 = s_5 + 4$	$= 20.034$	$s'_6 = s'_5 = 2$
$s_7 = \text{sqr}(s_2)$	$= 64$	$s'_7 = s'_2 \times 2 \times s_2 = 16$
$s_8 = 3 \times s_7$	$= 192$	$s'_8 = 3 \times s'_7 = 48$
$s_9 = s_8 + 6$	$= 198$	$s'_9 = s'_8 = 48$
$s_{10} = s_6 \times s_9$	$= 3966.732$	$s'_{10} = s_6 \times s'_9 + s'_6 \times s_9 = 1357.632$

# Forward Mode Performance

- To compute the full gradient of a function of  $m$  inputs requires computing  $m$  partial derivatives
- In forward mode, this requires  $m$  forward passes
- For our toy examples, that means running the forward pass twice
- Neural networks can easily have **thousands** of parameters
- We don't want to run the network *thousands of times* for each gradient update!

# Automatic Differentiation: Backward Mode

- **Forward mode** sweeps through the graph:

- For each  $s_i$ , computes  $s'_i = \frac{\partial s_i}{\partial s_1}$  for each  $s_i$
- The **numerator varies**, and the **denominator is fixed**

- **Backward mode** does the opposite:

- For each  $s_i$ , computes the **local gradient**  $\bar{s}_i = \frac{\partial s_n}{\partial s_i}$
- The **numerator is fixed**, and the **denominator varies**

- At the end, we have computed  $\bar{x}_i = \frac{\partial s_n}{\partial x_i}$  for each input  $x_i$

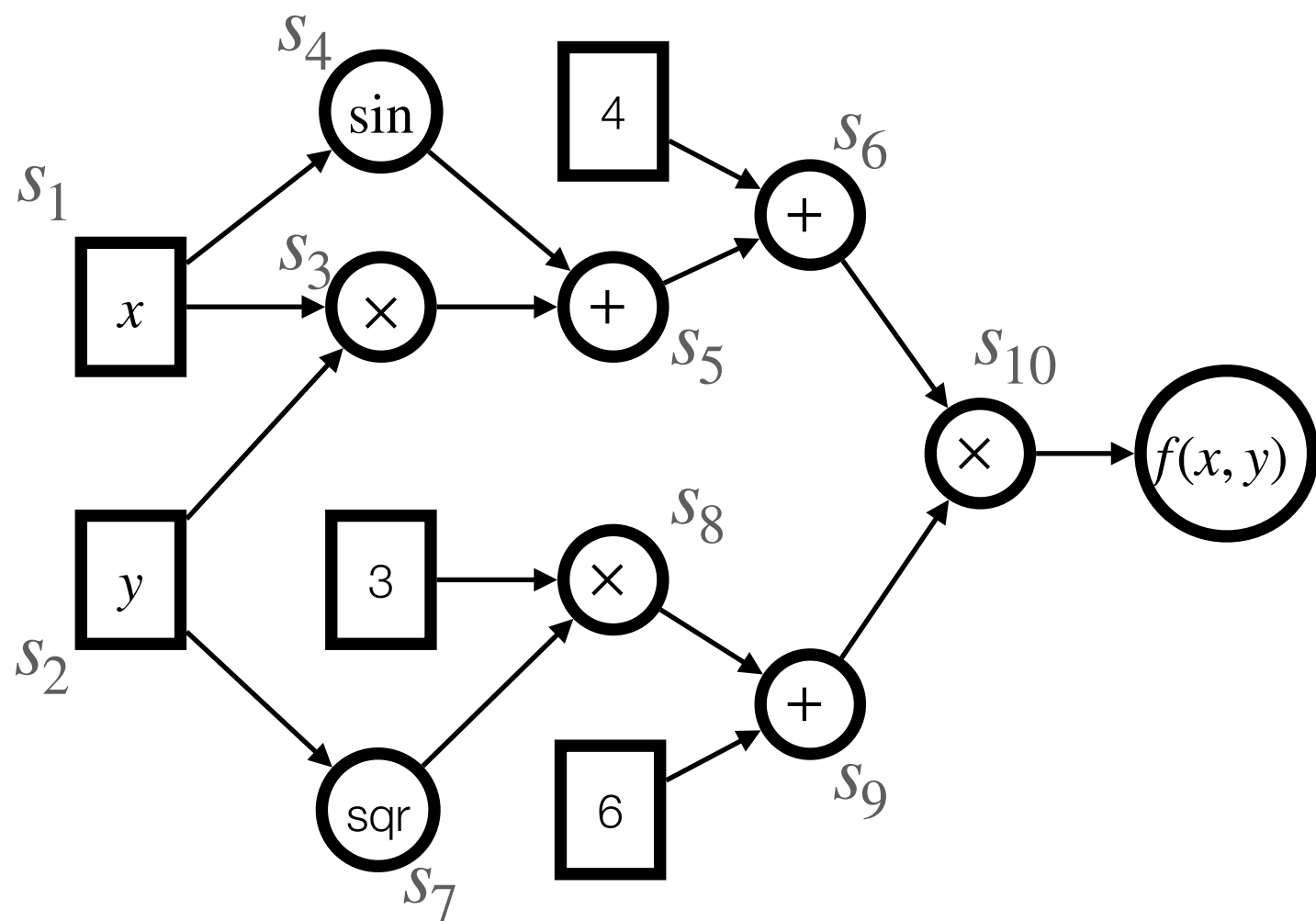
$$\begin{array}{l}
 s_1 = x \\
 s_2 = y \\
 s_3 = s_1 \times s_2 \\
 s_4 = \sin(s_1) \\
 s_5 = s_3 + s_4 \\
 s_6 = s_5 + 4 \\
 s_7 = \text{sqr}(s_2) \\
 s_8 = 3 \times s_7 \\
 s_9 = s_8 + 6 \\
 s_{10} = s_6 \times s_9
 \end{array}
 \left. \begin{array}{l}
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \end{array} \right\} \frac{\partial s_3}{\partial s_1} \left. \begin{array}{l}
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \end{array} \right\} \frac{\partial s_4}{\partial s_1}$$
  

$$\frac{\partial s_{10}}{\partial s_7} \left\{ \begin{array}{l}
 \frac{\partial s_{10}}{\partial s_8} \left\{ \begin{array}{l}
 s_8 = 3 \times s_7 \\
 s_9 = s_8 + 6 \\
 s_{10} = s_6 \times s_9
 \end{array} \right.
 \end{array} \right.$$



# Automatic Differentiation: Local Derivatives

The augmented algorithm computes local derivatives in **reverse** order:

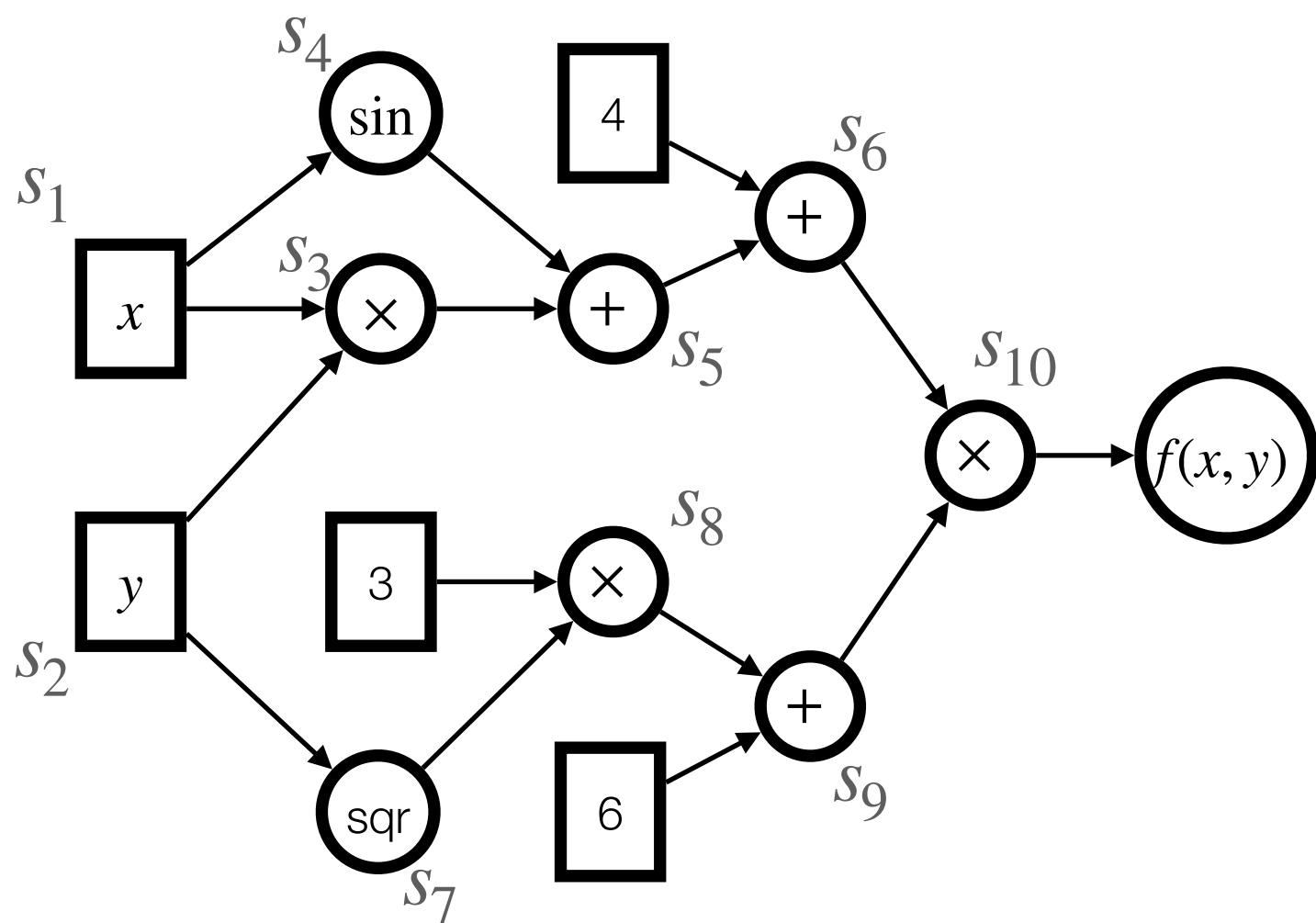


$$\begin{aligned}
 s_1 &= x \\
 s_2 &= y \\
 s_3 &= s_1 \times s_2 \\
 s_4 &= \sin(s_1) \\
 s_5 &= s_3 + s_4 \\
 s_6 &= s_5 + 4 \\
 s_7 &= \text{sqr}(s_2) \\
 s_8 &= 3 \times s_7 \\
 s_9 &= s_8 + 6 \\
 s_{10} &= s_6 \times s_9
 \end{aligned}$$

$$\begin{aligned}
 \overline{s}_{10} &= \frac{\partial s_{10}}{\partial s_{10}} = 1 \\
 \overline{s}_9 &= \frac{\partial s_{10}}{\partial s_9} = s_6 \\
 \overline{s}_8 &= \frac{\partial s_{10}}{\partial s_8} = \frac{\partial s_{10}}{\partial s_9} \frac{\partial s_9}{\partial s_8} = \overline{s}_9 1 \\
 \overline{s}_7 &= \frac{\partial s_{10}}{\partial s_7} = \frac{\partial s_{10}}{\partial s_8} \frac{\partial s_8}{\partial s_7} = \overline{s}_8 3 \\
 \overline{s}_6 &= \frac{\partial s_{10}}{\partial s_6} = s_9 \\
 &\vdots
 \end{aligned}$$

$\frac{\partial \text{final output}}{\partial \text{immediate output}}$      $\frac{\partial \text{immediate output}}{\partial \text{self}}$

# Automatic Differentiation: Local Derivatives (2)



$$\begin{aligned}
 s_1 &= x \\
 s_2 &= y \\
 s_3 &= s_1 \times s_2 \\
 s_4 &= \sin(s_1) \\
 s_5 &= s_3 + s_4 \\
 s_6 &= s_5 + 4 \\
 s_7 &= \text{sqr}(s_2) \\
 s_8 &= 3 \times s_7 \\
 s_9 &= s_8 + 6 \\
 s_{10} &= s_6 \times s_9
 \end{aligned}$$

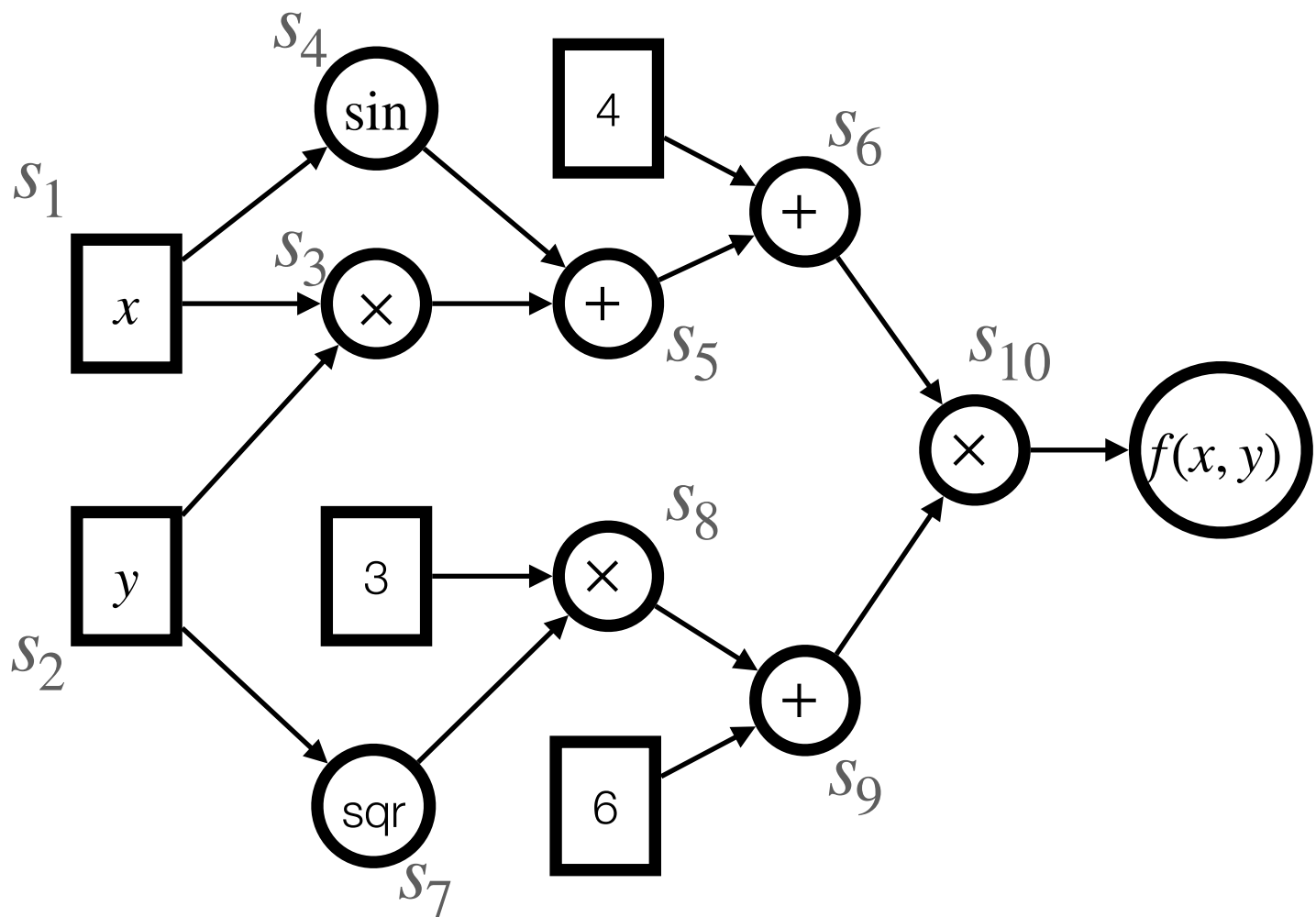
$$\begin{aligned}
 \overline{s}_6 &= \frac{\partial s_{10}}{\partial s_6} = s_9 \\
 \overline{s}_5 &= \frac{\partial s_{10}}{\partial s_5} = \frac{\partial s_{10}}{\partial s_6} \frac{\partial s_6}{\partial s_5} = \overline{s}_6 1 \\
 \overline{s}_4 &= \frac{\partial s_{10}}{\partial s_4} = \frac{\partial s_{10}}{\partial s_5} \frac{\partial s_5}{\partial s_4} = \overline{s}_5 1 \\
 \overline{s}_3 &= \frac{\partial s_{10}}{\partial s_3} = \frac{\partial s_{10}}{\partial s_5} \frac{\partial s_5}{\partial s_3} = \overline{s}_5 1 \\
 \overline{s}_2 &= \frac{\partial s_{10}}{\partial s_2} = \frac{\partial s_{10}}{\partial s_3} \frac{\partial s_3}{\partial s_2} + \frac{\partial s_{10}}{\partial s_7} \frac{\partial s_7}{\partial s_2} = \overline{s}_3 s_1 + \overline{s}_7 2s_2 \\
 \overline{s}_1 &= \frac{\partial s_{10}}{\partial s_1} = \frac{\partial s_{10}}{\partial s_3} \frac{\partial s_3}{\partial s_1} + \frac{\partial s_{10}}{\partial s_4} \frac{\partial s_4}{\partial s_1} = \overline{s}_3 s_2 + \overline{s}_4 \cos s_1
 \end{aligned}$$

One term for **each** immediate output



# Backward Mode Example

Let's compute  $\left. \frac{\partial f}{\partial x} \right|_{x=2, y=8}$  and  $\left. \frac{\partial f}{\partial y} \right|_{x=2, y=8}$  using backward mode:



$$\begin{aligned} s_1 &= x &&= 2 \\ s_2 &= y &&= 8 \\ s_3 &= s_1 \times s_2 &&= 16 \\ s_4 &= \sin(s_1) &&\approx 0.034 \\ s_5 &= s_3 + s_4 &&= 16.034 \\ s_6 &= s_5 + 4 &&= 20.034 \\ s_7 &= \text{sqr}(s_2) &&= 64 \\ s_8 &= 3 \times s_7 &&= 192 \\ s_9 &= s_8 + 6 &&= 198 \\ s_{10} &= s_6 \times s_9 &&= 3966.732 \end{aligned}$$

$$\begin{aligned} \overline{s_{10}} &= 1 \\ \overline{s_9} &= \overline{s_{10}} s_6 = 20.034 \\ \overline{s_8} &= \overline{s_9} 1 = 20.034 \\ \overline{s_7} &= \overline{s_8} 3 = 60.102 \\ \overline{s_6} &= s_9 = 198 \\ \overline{s_5} &= \overline{s_6} 1 = 198 \\ \overline{s_4} &= \frac{\partial s_{10}}{\partial s_4} = \frac{\partial s_{10}}{\partial s_5} \frac{\partial s_5}{\partial s_4} = \overline{s_5} 1 = 198 \\ \overline{s_3} &= \frac{\partial s_{10}}{\partial s_3} = \frac{\partial s_{10}}{\partial s_5} \frac{\partial s_5}{\partial s_3} = \overline{s_5} 1 = 198 \\ \overline{s_2} &= \overline{s_3} s_1 + \overline{s_7} 2 s_2 \simeq 1357.632 \\ \overline{s_1} &= \overline{s_3} s_2 + \overline{s_4} \cos s_1 \simeq 1781.9 \end{aligned}$$

# Back-Propagation

$$L(\mathbf{W}, \mathbf{b}) = \sum_i \ell \left( f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b}), y^{(i)} \right)$$

**Back-propagation** is simply automatic differentiation in **backward mode**, used to compute the gradient  $\nabla_{\mathbf{W}, \mathbf{b}} L$  of the **loss function** with respect to its **parameters**  $\mathbf{W}, \mathbf{b}$ :

1. At each layer, compute the **local gradients** of the layer's computations
2. These local gradients will be used as inputs to the **next layer's** local gradient computations
3. At the end, we have a partial derivative for each of the parameters, which we can use to take a **gradient step**

# Summary

- The loss function of a **deep feedforward networks** is simply a very nested function of the **parameters** of the model
- **Automatic differentiation** can compute these gradients more efficiently than symbolic differentiation or finite-differences numeric computations
  - Symbolic differentiation is **interleaved** with numeric computation
  - In **forward mode**,  $m$  sweeps are required for a function of  $m$  parameters
  - In **backward mode**, only a single sweep is required
- **Back-propagation** is simply automatic differentiation **applied to neural networks** in backward mode