CSE 493G1: Deep Learning

Friday Lecture 2: Backpropagation

Who Am I?

- Hi, I’m Shubhang Desai
  - BS + MS from Stanford CS
  - Moved to Seattle ~1.5 years ago

- Applied Scientist at Microsoft, on Ink AI Team
  - Spearheaded deep learning handwriting recognizer (HWR)
  - Working on HWR, ink analysis, etc.
  - Both image and sequence modelling tasks

- Passionate about teaching: CS 131 (Comp. Vision), CS 23 (Deep Learning), CS 21SI (AI + Social Good) @ Stanford

Plan for Today

- Background: the problem of gradient computation
- Intro to backpropagation algorithm
- Gradients of common computations
- Worked-thru examples
- What does it all mean?

Background

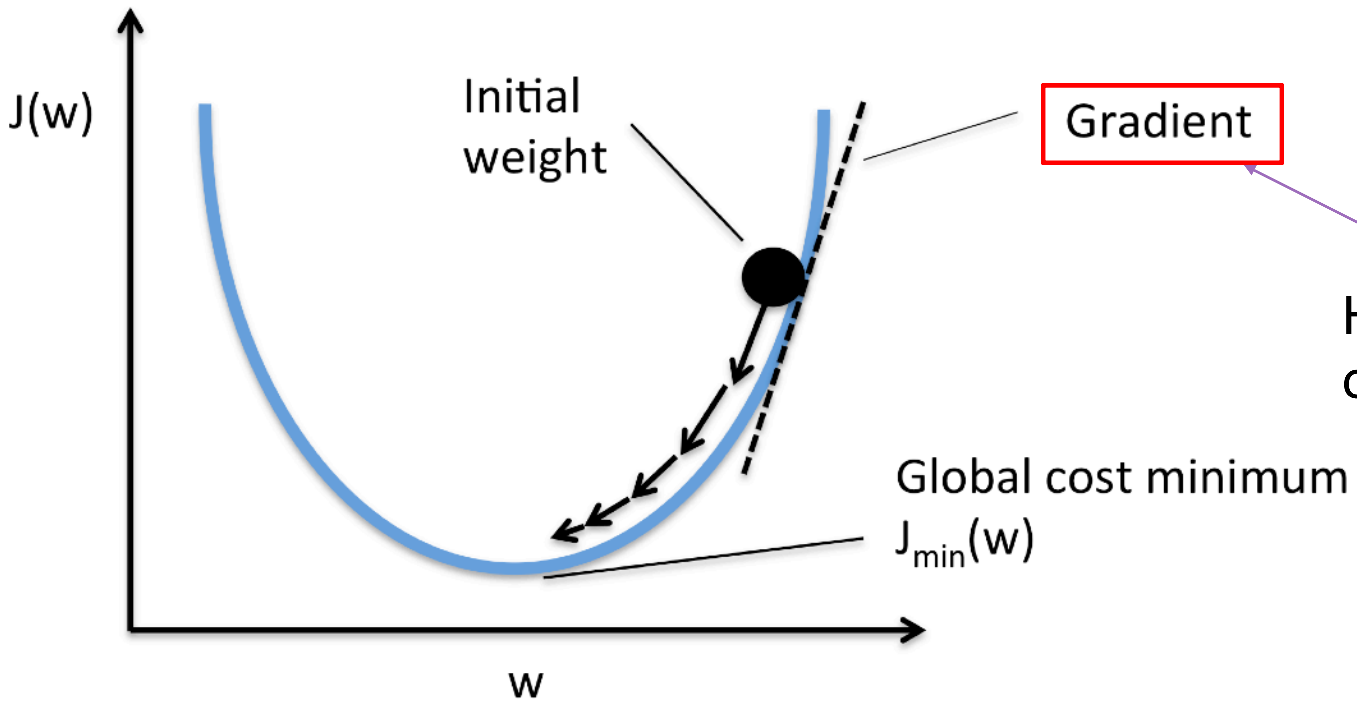
Refresher: Chain Rule

"Derivative of outside of inside equals derivative of outside times derivative of inside"

$$\frac{d}{dx} f\big(g(x)\big) = \frac{d}{dg(x)} f\big(g(x)\big) \times \frac{d}{dx} g(x)$$

Refresher: Gradient Descent

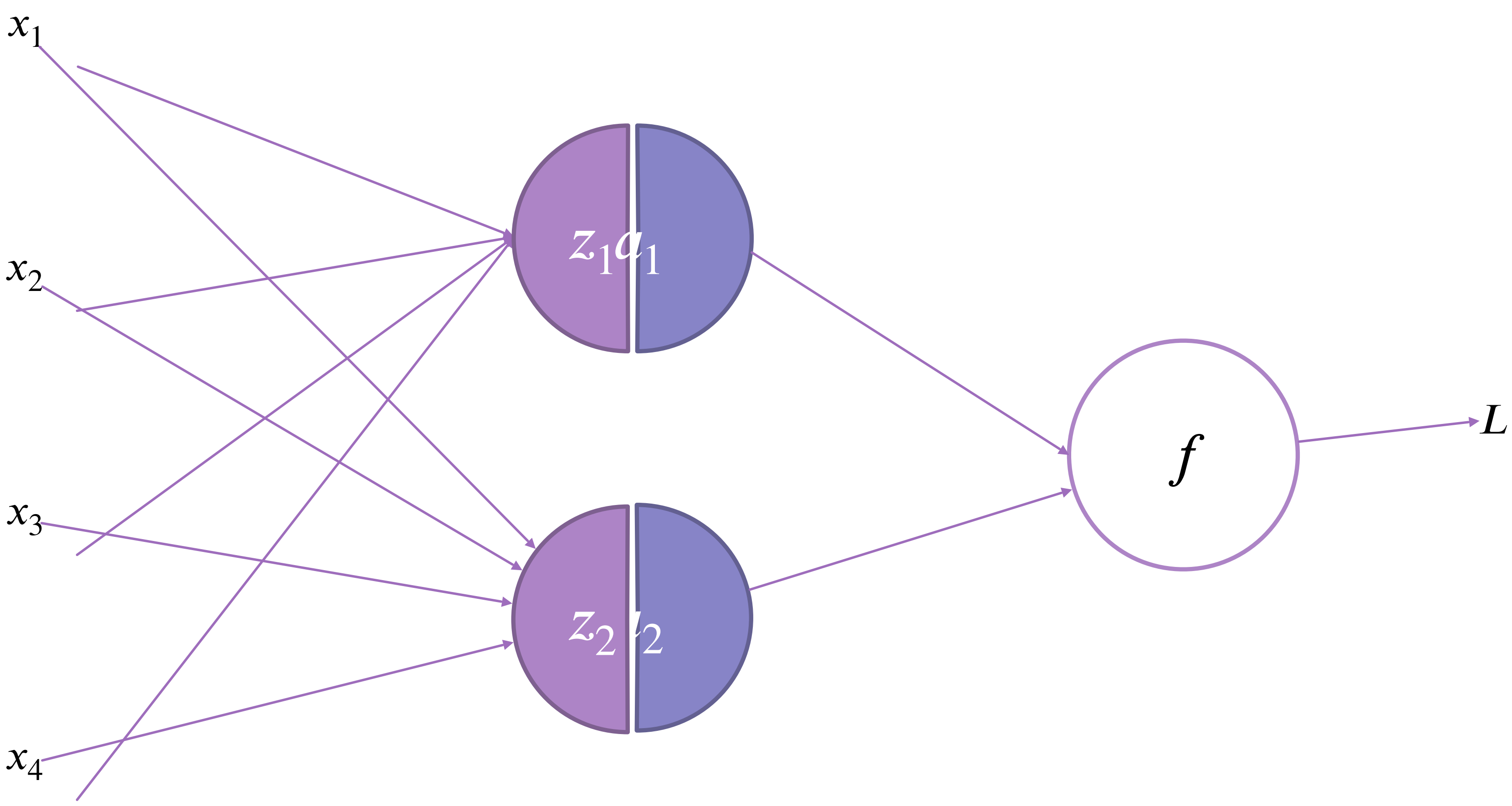
Iteratively moving neural network weights in the direction of the gradient to minimize loss:



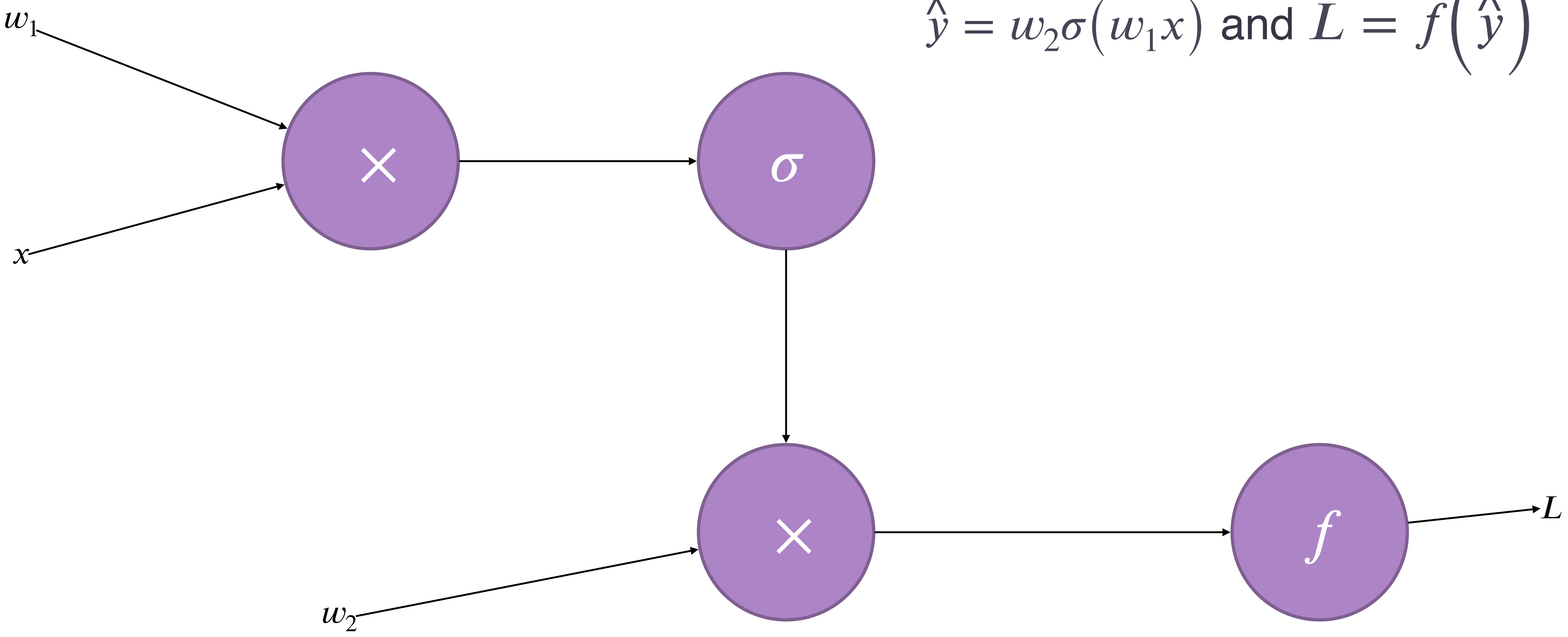
How do we actually compute this?

Intro to Backprop: A 2-Layer MLP

2-Layer MLP



$$\hat{y} = w_2\sigma(w_1x) \text{ and } L = f(\hat{y})$$



2-Layer MLP Equations and Gradients

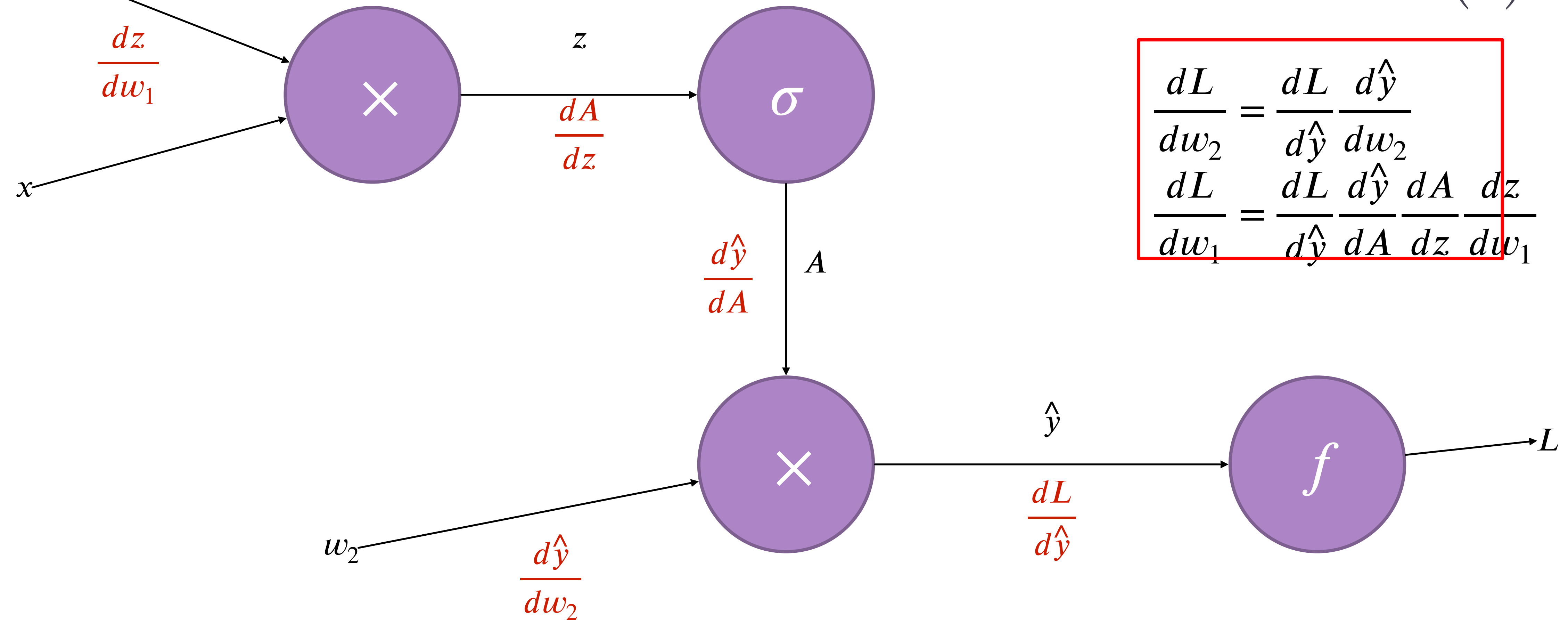
$L = f(\hat{y})$, where $\hat{y} = w_2 A$, $A = \sigma(z)$, and $z = w_1 x$

$$\frac{dL}{dw_1} = \boxed{\frac{dL}{d\hat{y}}}\frac{d\hat{y}}{dw_1} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dw_1} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dz}\frac{dz}{dw_1}$$

$$\frac{dL}{dw_2} = \boxed{\frac{dL}{d\hat{y}}}\frac{d\hat{y}}{dw_2}$$

Notice that this value is used in both gradient computations!

$$\hat{y} = w_2\sigma(w_1 x) \text{ and } L = f(\hat{y})$$



$$\boxed{\begin{aligned}\frac{dL}{dw_2} &= \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dw_2} \\ \frac{dL}{dw_1} &= \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dz}\frac{dz}{dw_1}\end{aligned}}$$

The Backpropagation Algorithm

- Treat entire network as a computational graph, each computation as a node
- We can independently compute local gradient at each node given node inputs
- Accumulate gradients from back (loss) to front (weights) using chain rule (simple multiplication!)

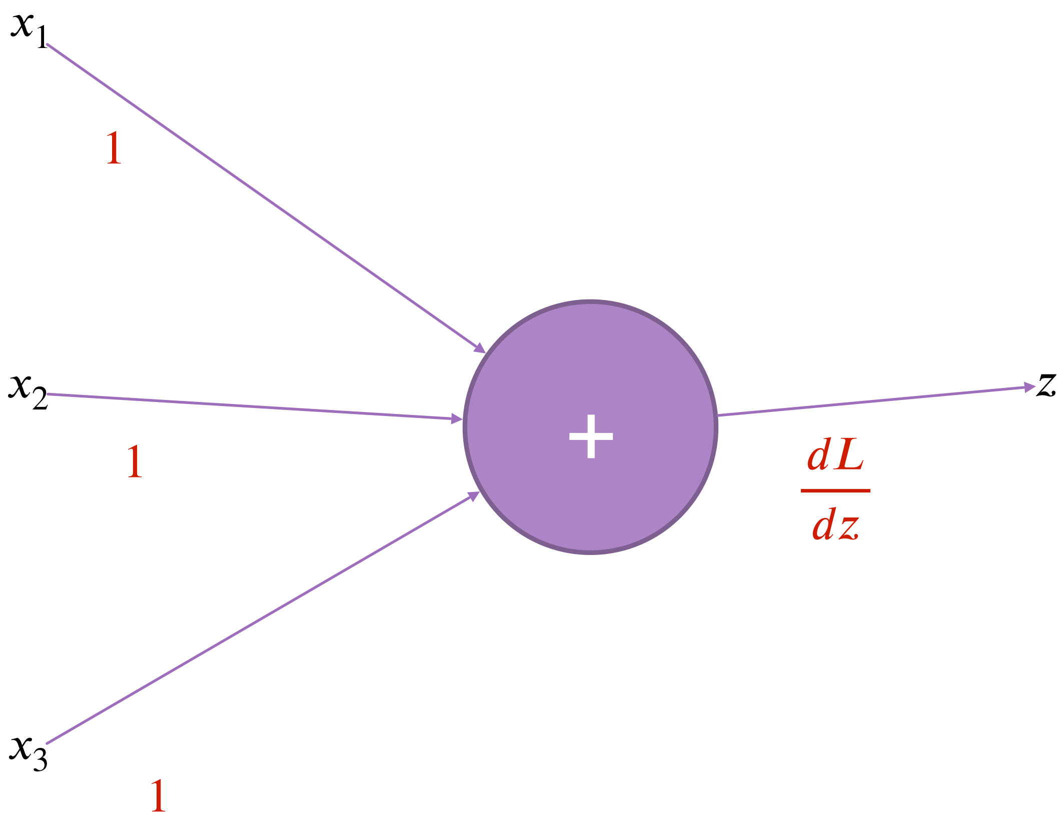
Computations and their Gradients

Summation

$$z = \sum_i x_i,\ L = f(Z)$$

$$\frac{\partial z}{\partial x_i} = 1$$

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

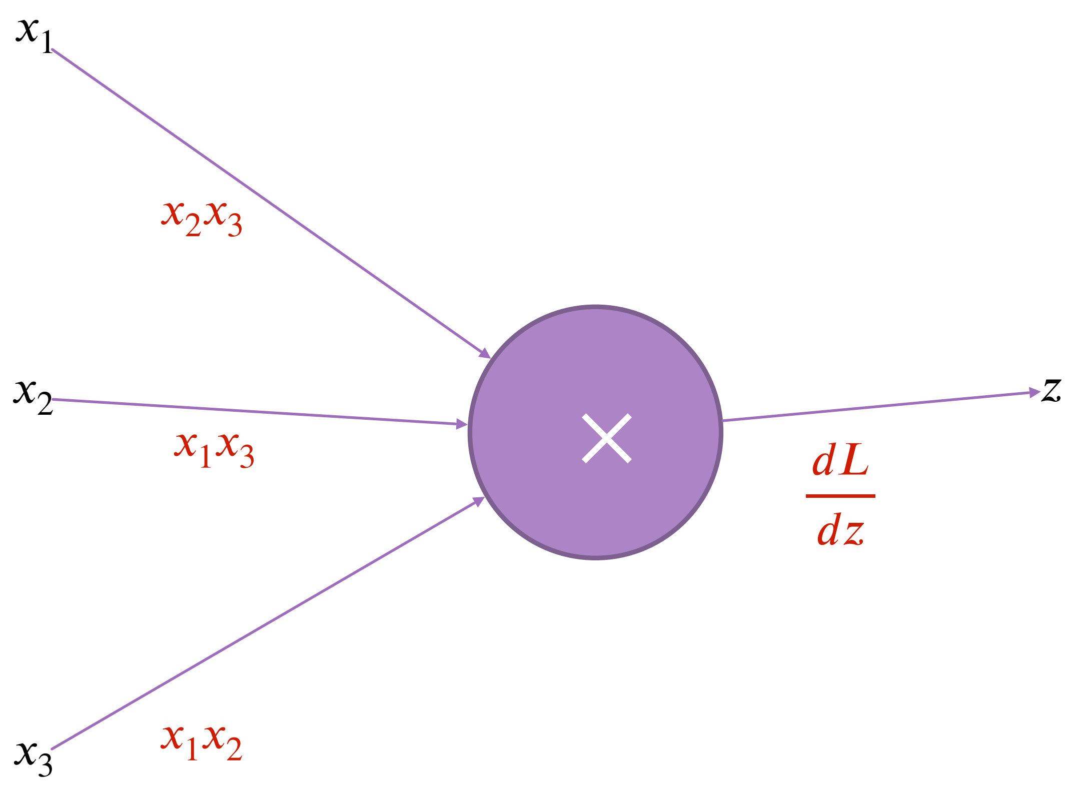


Multiplication

$$z = \prod_i x_i, \; L = f(Z)$$

$$\frac{\partial z}{\partial x_i} = \frac{z}{x_i}$$

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

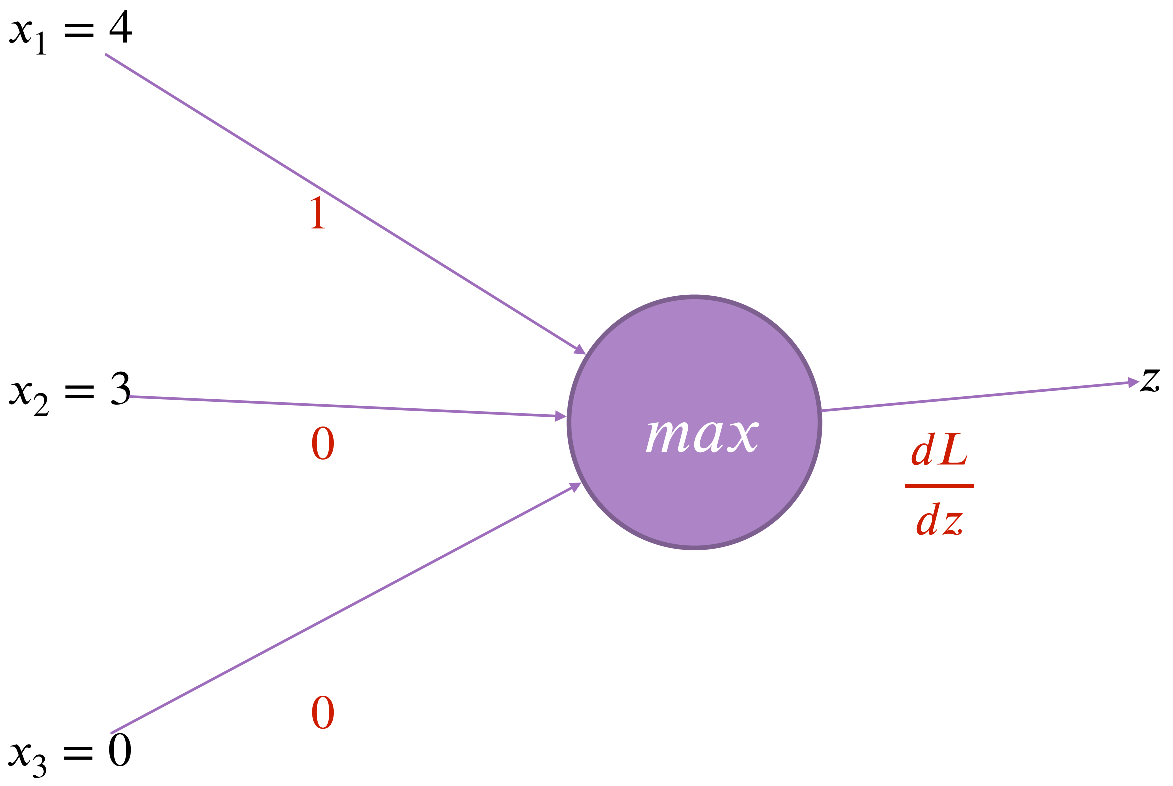


Min/Max

$$z = \max_i x_i,\ L = f(Z)$$

$$\frac{\partial z}{\partial x_i} = \mathbf{1}[x_i = z]$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x_i} = \frac{\partial L}{\partial z}\mathbf{1}[x_i = z]$$

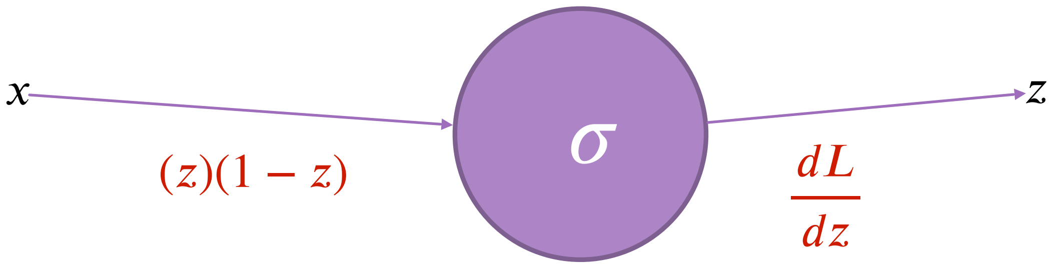


Sigmoid

$$z = \sigma(x),\ \ L = f(Z)$$

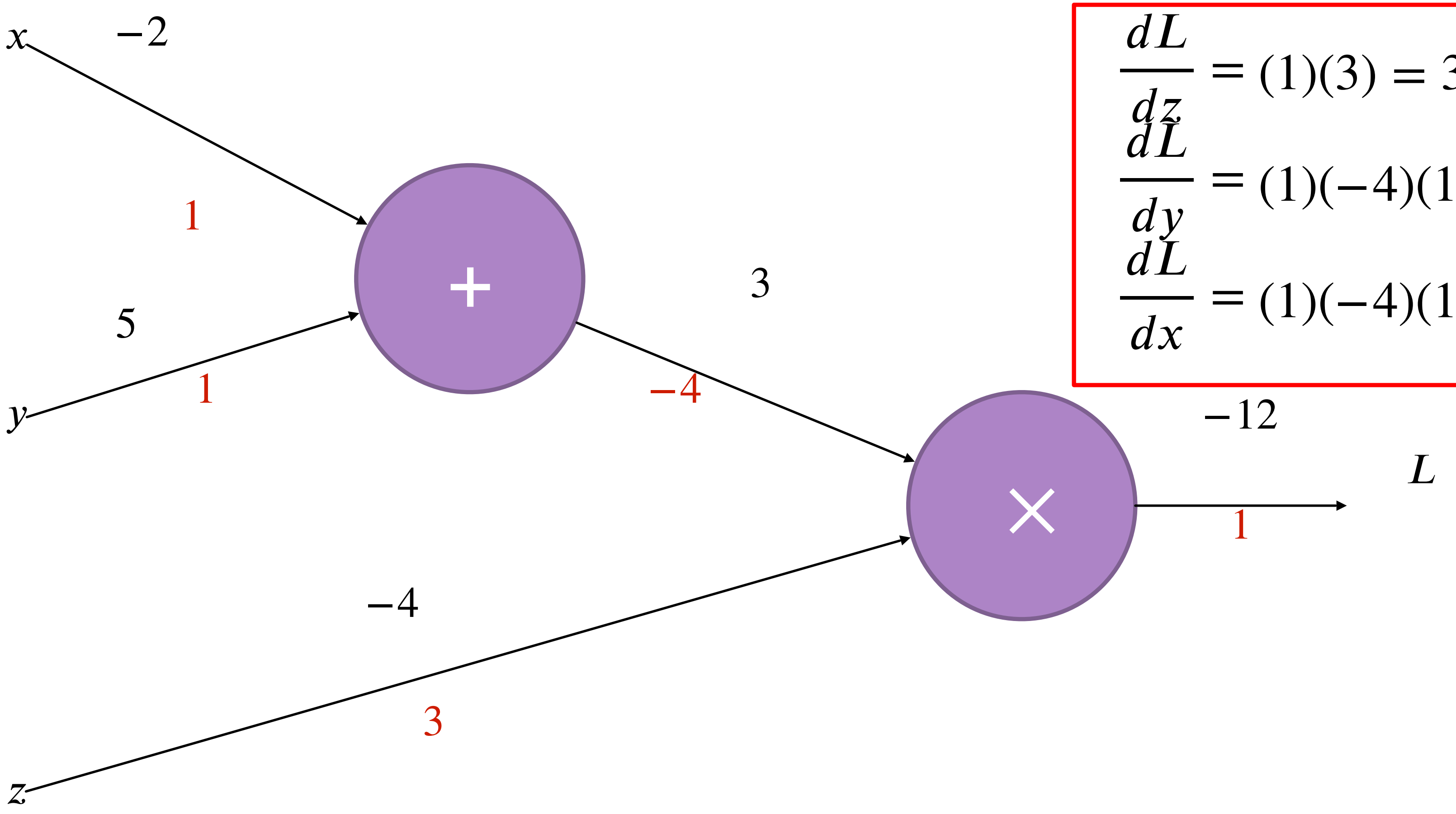
$$\frac{\partial z}{\partial x_i} = z(1 - z)$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x_i} = \frac{\partial L}{\partial z}z(1 - z)$$



Computational Graph Example 1

$$L = (x + y)z$$



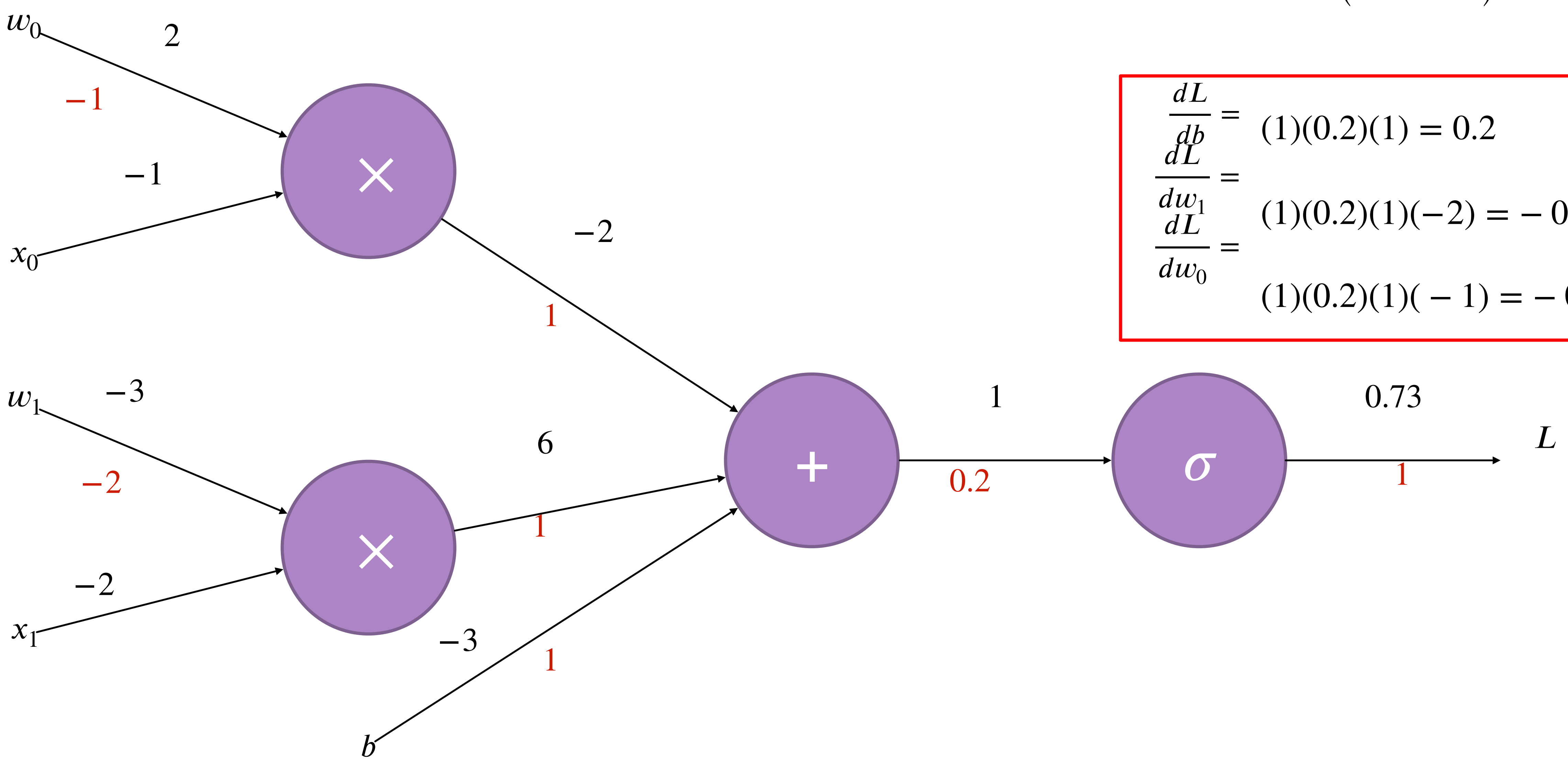
$$\frac{dL}{dz} = (1)(3) = 3$$

$$\frac{dL}{dy} = (1)(-4)(1) = -4$$

$$\frac{dL}{dx} = (1)(-4)(1) = -4$$

Computational Graph Example 2

$$L = \sigma\left(w^T x + b\right)$$



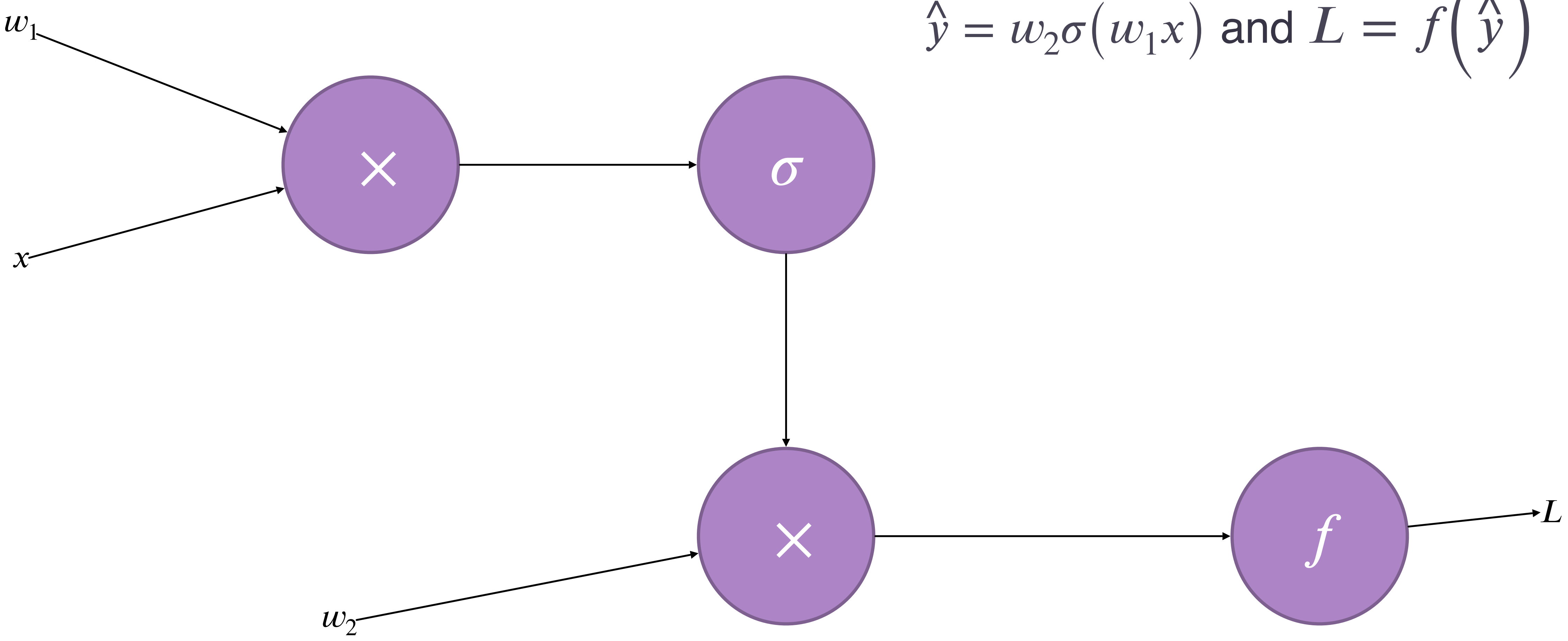
$$\frac{dL}{db} = (1)(0.2)(1) = 0.2$$

$$\frac{dL}{dw_1} = (1)(0.2)(1)(-2) = -0.4$$

$$\frac{dL}{dw_0} = (1)(0.2)(1)(-1) = -0.2$$

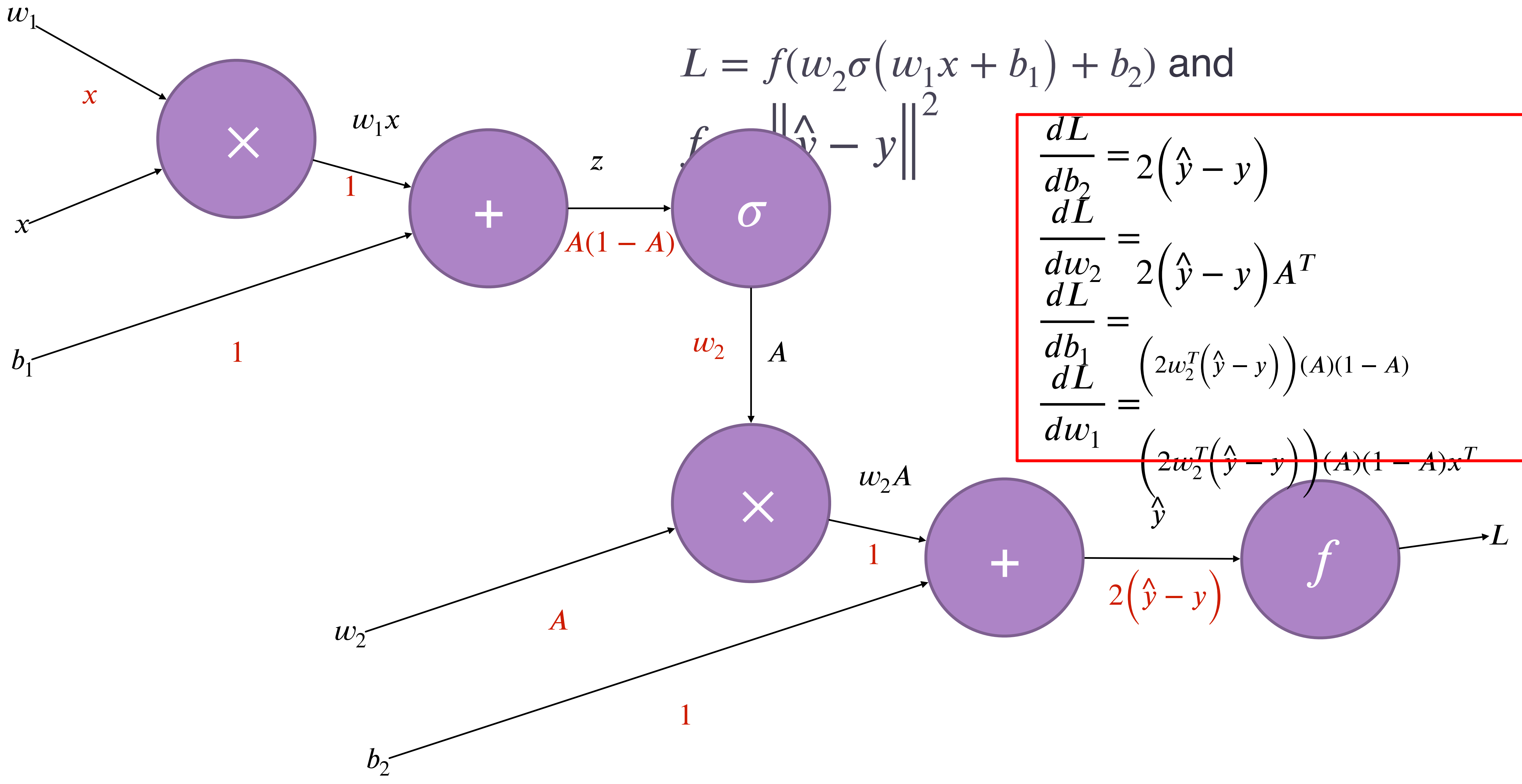
2-Layer MLP Example, Completed

$\hat{y} = w_2\sigma(w_1 x)$ and $L = f(\hat{y})$



$L = f(w_2\sigma(w_1 x + b_1) + b_2)$ and $f = \left\|\hat{y} - y\right\|^2$

$$\frac{dL}{db_2} = 2\left(\hat{y} - y\right)$$
$$\frac{dL}{dw_2} = 2\left(\hat{y} - y\right)A^T$$
$$\frac{dL}{db_1} = \left(2w_2^T\left(\hat{y} - y\right)\right)(A)(1 - A)$$
$$\frac{dL}{dw_1} = \left(2w_2^T\left(\hat{y} - y\right)\right)(A)(1 - A)x^T$$



More Mathematically…

Forward Prop	Backprop

What Does This All Mean For Us?

Not Much!

- Most NN ops can be broken down into simple computations with closed-form derivatives
- Frameworks like PyTorch and TensorFlow build the computational graph during the forward pass, and each node in the graph has its backward pass already written for us!
- As such, the framework can compute gradients automatically–a.k.a. auto-differentiation
  - Auto-diff allows us to focus on forward pass: if it's all differentiable, gradient comp. is taken care of
  - We may want to write the backward pass ourselves e.g. in C++ to speed it up, but this is rare
- It's still good to understand backprop at a theoretical level
  - Helps you debug when your model isn't converging!

Pseudocode

```
for X, y in train_dataset:
    y_hat = compute_prediction(model, X)
    l = compute_loss(y, y_hat)
    grads = compute_gradients(model, l)
    for name, grad in grads:
        model.take_update_step(name, grad)
```

Auto-diff enables us to compute gradients without writing any backwards pass code!

Real PyTorch Code

```
for X, y in train_dataset:
    y_hat = model(X)
    l = loss_fn(y, y_hat)
    l.backward()
    optim.step()
```

These APIs do gradient computation and weight update for you!

Summary

- Gradient descent requires gradient computation
- The popular method to do so in deep learning is backpropagation algorithm: greedily use chain rule to accumulate local gradients from “back to front”
- Frameworks such as PyTorch and TensorFlow do backprop for you (auto-differentiation)