8. Neural Networks

Dongwoo Kim

dongwoo.kim@postech.ac.kr

CSED515 - 2023 Spring

Midterm Exam

- The midterm exam will be an offline exam.
- Time: 11:00am $\sim$ 12:20pm, 4/13 (Thursday)
- Location: TBA
- Scope: 1. Introduction $\sim$ 8. Neural Networks

Recall) Loss Functions for Binary Classification

$$\text{Linear regression:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \frac{1}{2}\left(1 - y^{(i)}\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)})\right)^2$$

$$\text{Logit regression:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \log\left(1 + \exp\left(-y^{(i)}\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)})\right)\right)$$

$$\text{Binary SVM:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \max\left\{0, 1 - y^{(i)}(\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)}))\right\}$$

$$\text{General binary cls:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \varepsilon \log\left(1 + \exp\left(\frac{L - y^{(i)}\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)})}{\varepsilon}\right)\right)$$

Using Parameterized Feature

Decision with sign $\left(\boldsymbol{w}^\top \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ where we learn $\boldsymbol{w}, \boldsymbol{\theta}$ from data

Linear regression: $\min_{\boldsymbol{w},b,\theta} \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \frac{1}{2}\left(1 - y^{(i)}\boldsymbol{w}^\top \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)^2$

Logit regression: $\min_{\boldsymbol{w},b,\theta} \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \log\left(1 + \exp\left(-y^{(i)}\boldsymbol{w}^\top \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\right)$

Binary SVM: $\min_{\boldsymbol{w},b,\theta} \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \max\left\{0, 1 - y^{(i)}(\boldsymbol{w}^\top \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\right\}$

General binary cls: $\min_{\boldsymbol{w},b,\theta} \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \varepsilon \log\left(1 + \exp\left(\frac{L - y^{(i)}\boldsymbol{w}^\top \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})}{\varepsilon}\right)\right)$

More General Framework

Decision with sign $\left(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)$ where we learn $\boldsymbol{w}$ from data

$$\text{Linear regression:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \frac{1}{2}\left(1 - y^{(i)} f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^2$$

$$\text{Logit regression:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \log\left(1 + \exp\left(-y^{(i)} f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)\right)$$

$$\text{Binary SVM:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \max\left\{0, 1 - y^{(i)} f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right\}$$

$$\text{General binary cls:} \quad \min_{\boldsymbol{w},b} \quad \frac{C'}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \varepsilon \log\left(1 + \exp\left(\frac{L - y^{(i)} f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})}{\varepsilon}\right)\right)$$

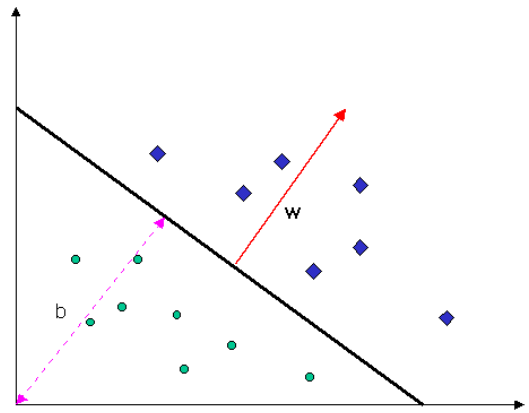
Neural Network for Function Approximation

Neural network: a powerful set of function models consisting of

- Perceptrons
- Multi-layer Perceptrons
- Convolutions
- Activation function
- Maximum/Average pooling
- Soft-max layer
- Dropout
- ...

Table of Contents

Recall) Linear Binary Classification



- A linear discriminant function which has the form

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$$

- Decision rule is given by $\text{sign}(f(\boldsymbol{x}))$

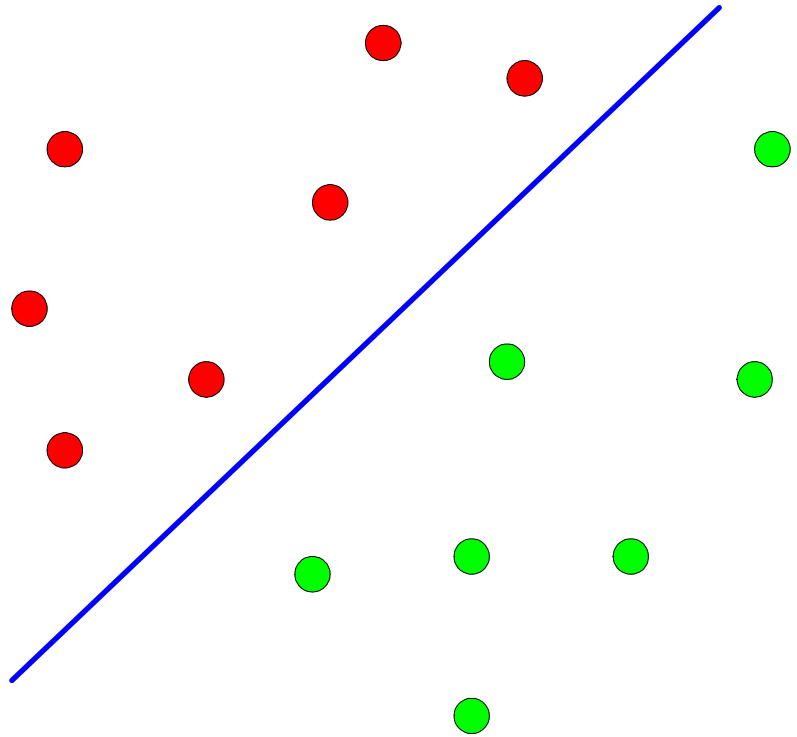
Perceptron

- Proposed by Rosenblatt in 1956
  - The New York Times reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."
- A single-layer neural network with thresholding activation function:

$$y = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

- Training: on-line and mistake-driven procedure
  - The weight vector is updated each time a training point is misclassified
- Convergence is guaranteed for linearly separable data (perceptron convergence theorem)

Linearly Separable



Perceptron Criterion

- Suppose that target values $\{y_n\}$ take either 1 or -1

$$y_n = \begin{cases} 1 & \text{if} \boldsymbol{x}_n \in \mathcal{C}_1 \\ -1 & \text{if} \boldsymbol{x}_n \in \mathcal{C}_2 \end{cases}$$

- We want to find[1] $\boldsymbol{w}$ s.t. $y_n = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_n) \quad \forall n$, or equivalently,

$$y_n \boldsymbol{w}^\top \boldsymbol{x}_n > 0 \quad \forall n$$

which is called perceptron criterion

[1] Remove bias term b for simplicity

Optimization for Perceptron Criterion

- The perceptron criterion leads to the following objective function to be minimized:

$$\mathcal{J}(\boldsymbol{w}) = -\sum_{n \in \mathcal{M}(\boldsymbol{w})} y_n \boldsymbol{w}^\top \boldsymbol{x}_n$$

where $\mathcal{M}(\boldsymbol{w}) = \{n : y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0\}$ is the set of misclassified samples under $\boldsymbol{w}$

- The gradient of $\mathcal{J}(\boldsymbol{w})$ is

$$\nabla_{\boldsymbol{w}} \mathcal{J}(\boldsymbol{w}) = -\sum_{n \in \mathcal{M}(\boldsymbol{w})} y_n \boldsymbol{x}_n$$

which leads the following GD update (based on only mistakes)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \sum_{n \in \mathcal{M}(\boldsymbol{w})} y_n \boldsymbol{x}_n$$

Perceptron Learning: Stochastic Gradient Descent

1. Select a training sample $(\boldsymbol{x}_m, y_m)$ randomly
2. Only if misclassified, $\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} + \alpha y_m \boldsymbol{x}_m$
3. Repeat step 1 and step 2 until convergence

Limitation of A Single Perceptron

- Convergence is guaranteed for linearly separable data (perceptron convergence theorem)
- What if not, e.g., XOR?!

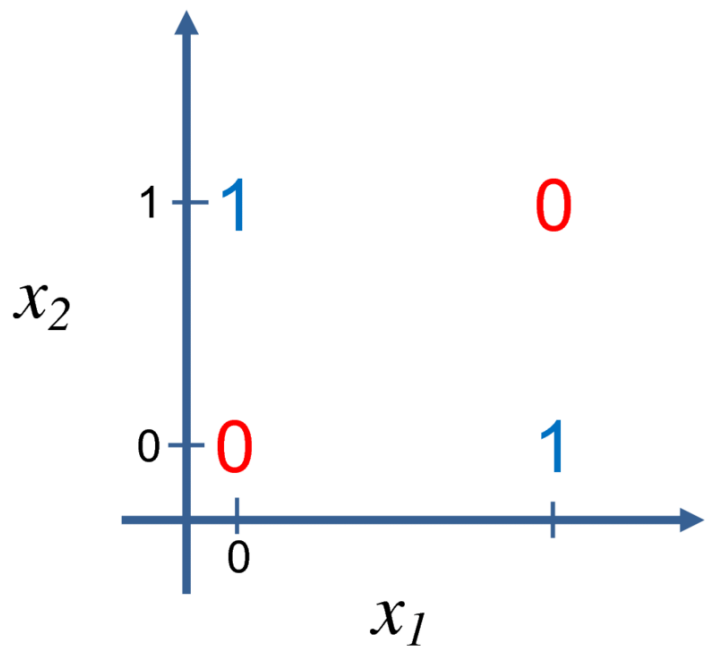
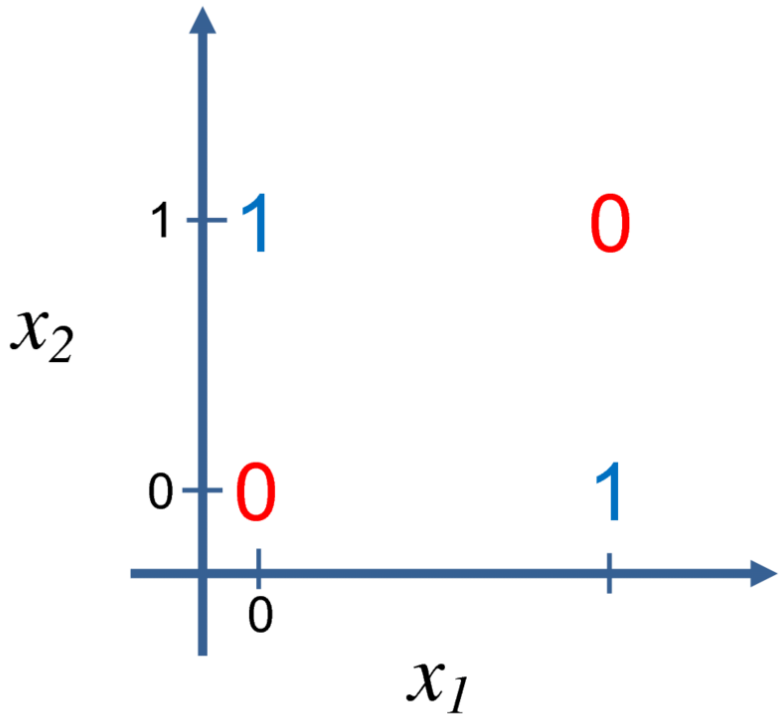
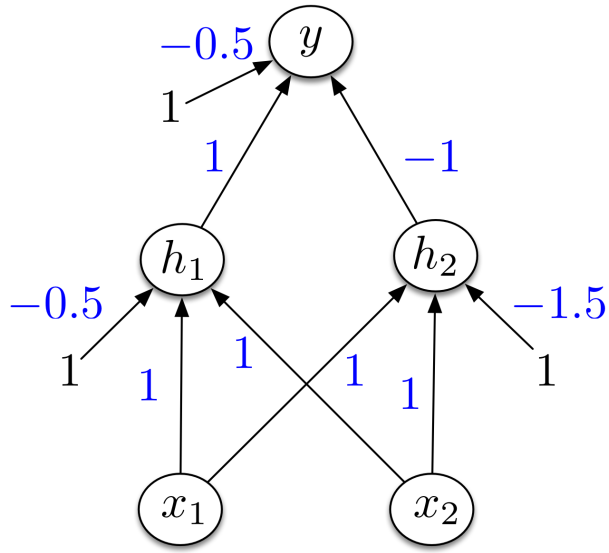


Table of Contents

XOR Problem

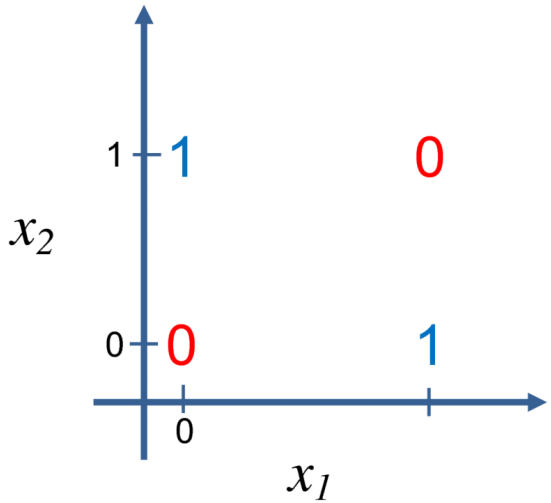


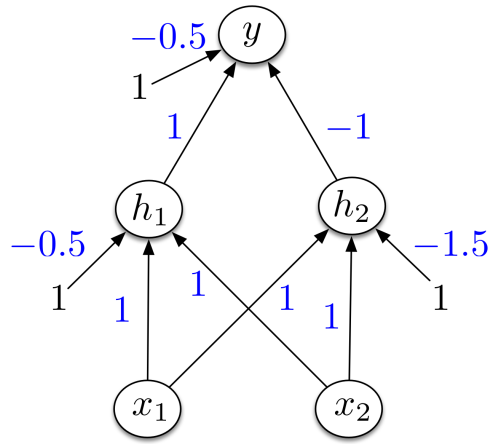


- Define two perceptrons h_1 and h_2 with identity function $\mathbb{1}$[2]:
  - $h_1(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 0.5 > 0]$
  - $h_2(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 1.5 > 0]$

[2] Returns 1 if true 0 otherwise

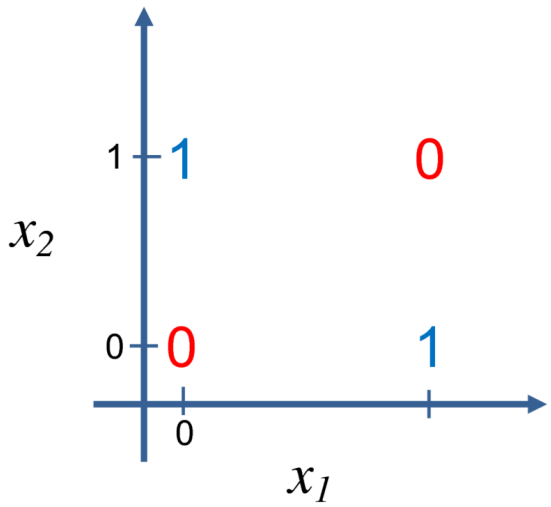
XOR Problem

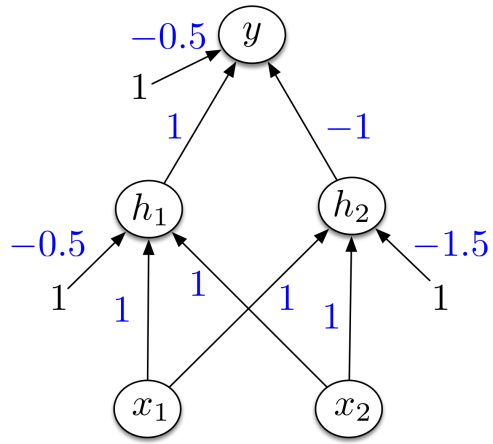




- $h_1(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 0.5 > 0] \Rightarrow x_1$ OR x_2
- $h_2(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 1.5 > 0] \Rightarrow x_1$ AND x_2

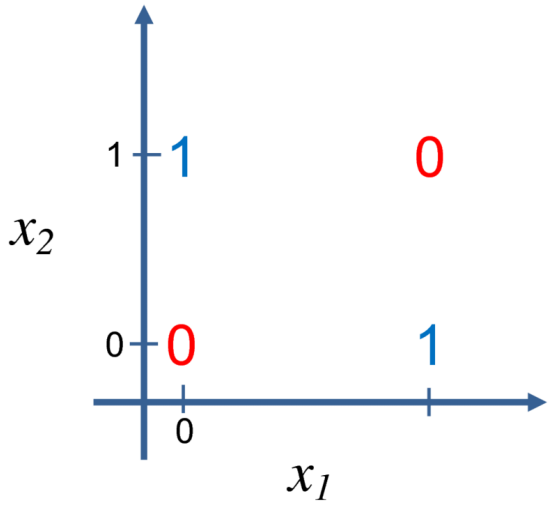
XOR Problem

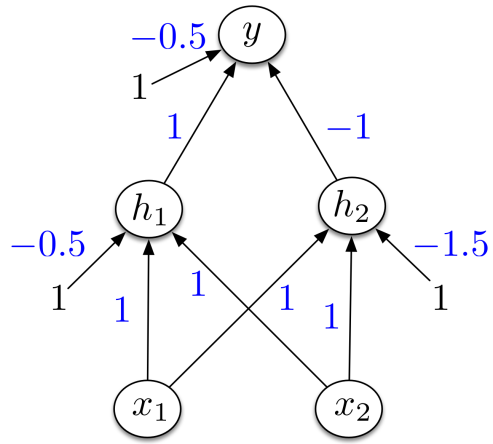




- $h_1(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 0.5 > 0] \Rightarrow x_1$ OR x_2
- $h_2(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 1.5 > 0] \Rightarrow x_1$ AND x_2
- $y(\mathbf{h}) = \mathbb{1}[h_1 - h_2 - 0.5 > 0] = \mathbb{1}[h_1 + (1 - h_2) - 1.5 > 0]$
  - i.e., h_1 AND (NOT h_2) = x_1 XOR x_2

XOR Problem





- $h_1(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 0.5 > 0] \Rightarrow x_1$ OR x_2
- $h_2(\mathbf{x}) = \mathbb{1}[x_1 + x_2 - 1.5 > 0] \Rightarrow x_1$ AND x_2
- $y(\mathbf{h}) = \mathbb{1}[h_1 - h_2 - 0.5 > 0] = \mathbb{1}[h_1 + (1 - h_2) - 1.5 > 0]$
  - i.e., h_1 AND (NOT h_2) = x_1 XOR x_2
- Draw the decision boundary of h_1 and h_2!

Perceptrons as Logical Operators

- A perceptron can perform AND (boolean multiplication; $\times$) and OR (boolean addition; $+$) such as

$$ab, a\bar{b}, \bar{a}b, \bar{a}\bar{b}, a+b, a+\bar{b}, \bar{a}+b, \bar{a}+\bar{b}\ ,$$

  where $0 = \bar{1}$ and $1 = \bar{0}$

- Note that two AND's and one OR can perform XOR, i.e.,

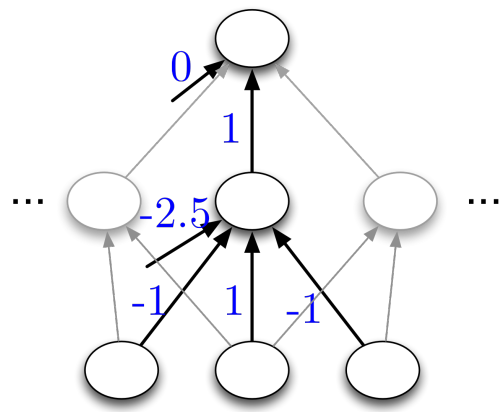
$$a \oplus b = a\bar{b} + \bar{a}b$$

Universal Approximation Theorem (Cybenko '89)[3]

Universality for *binary inputs and targets*:

- Hard threshold hidden units, linear output
- Strategy: 2^D hidden units, each of which responds to one particular input configuration

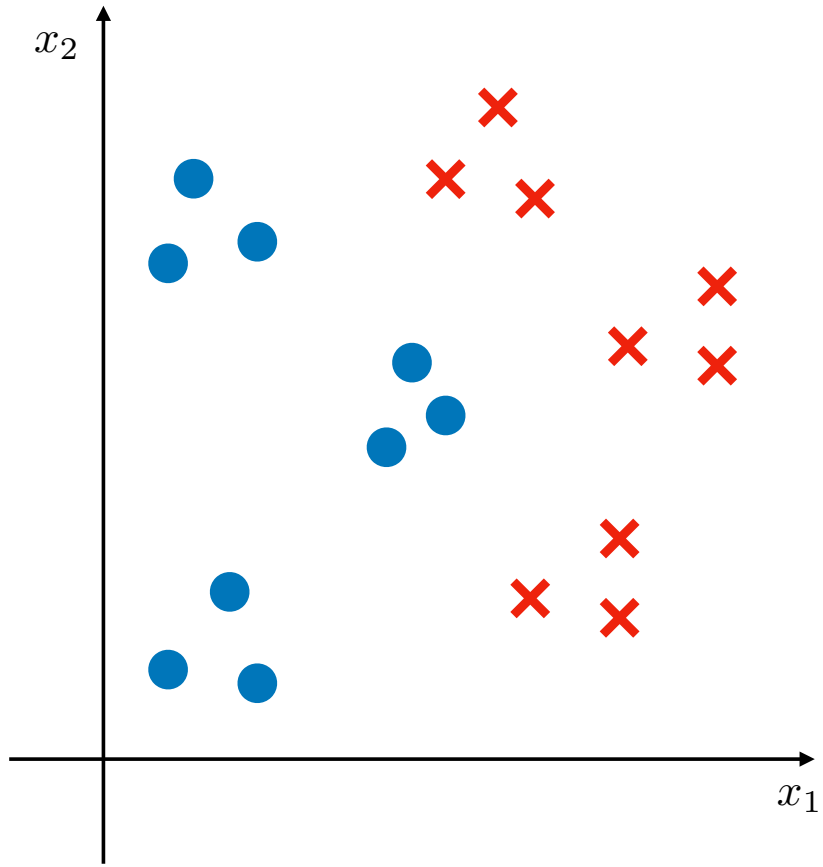
x_1	x_2	x_3	t
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$



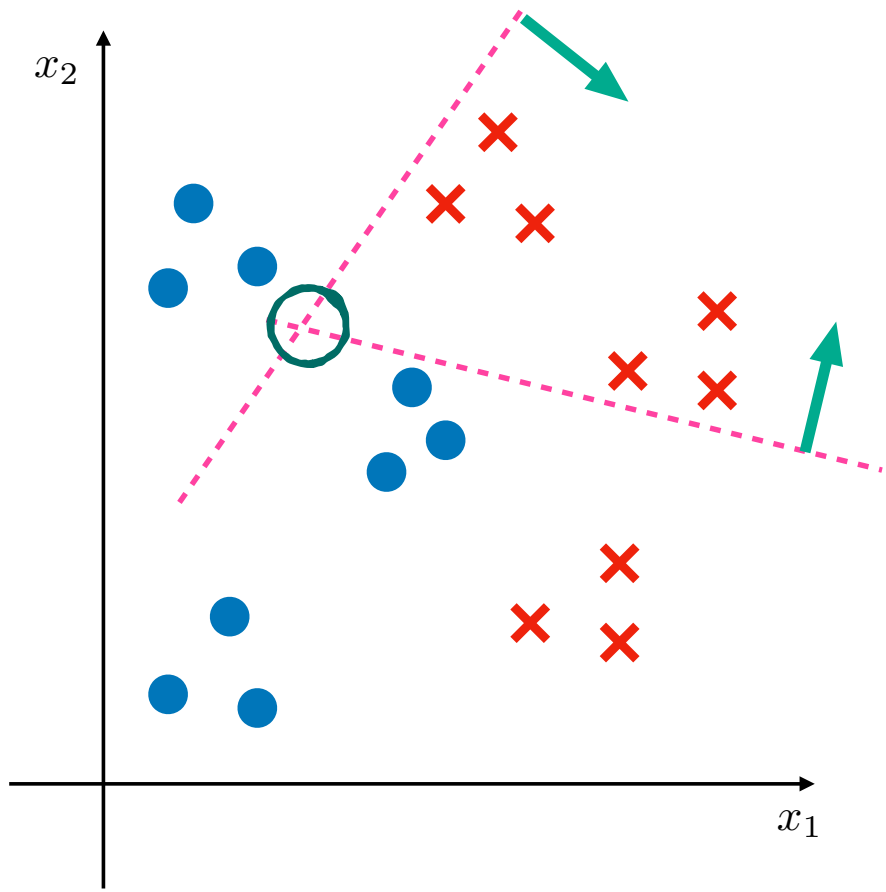
- Only requires one hidden layer, though it needs to be extremely wide.
- One wide (latent) layer is enough, but it is just a memorizer!

[3] The original theorem is slightly different from the one in this slide.

Other Example

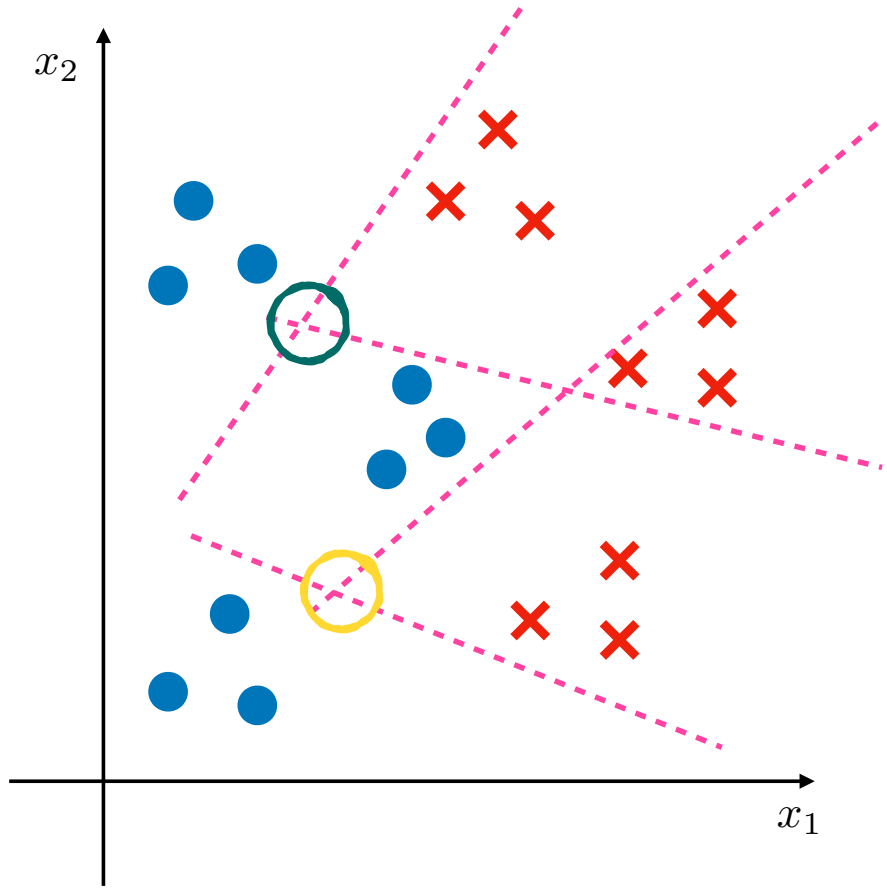


Other Example



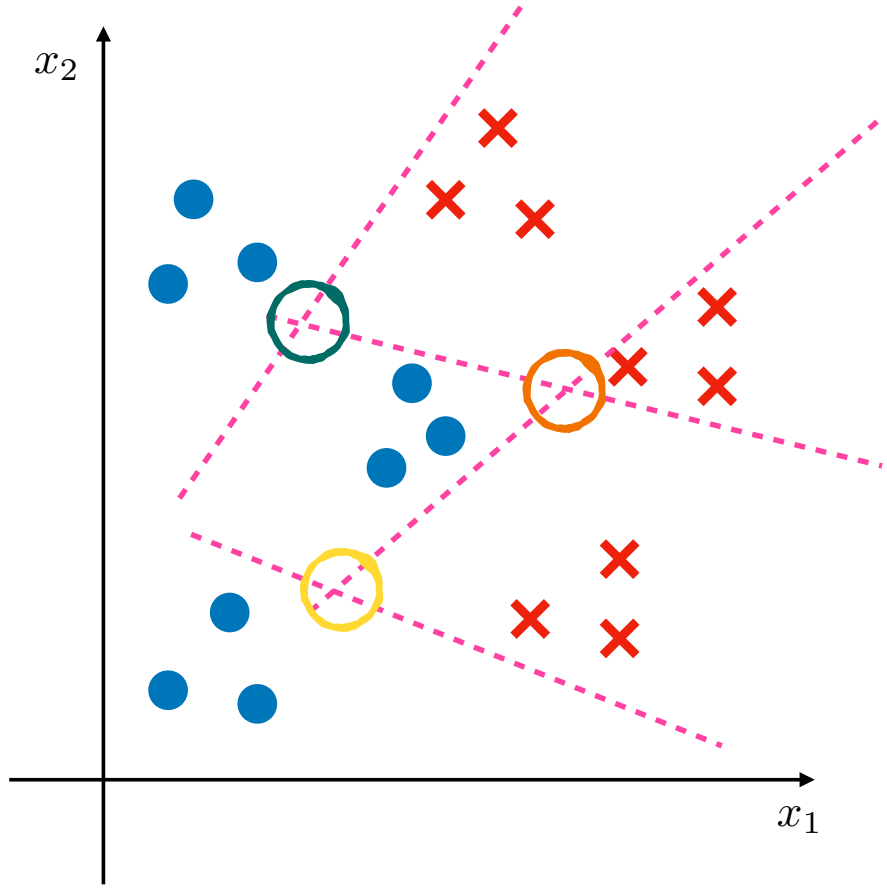
By taking AND of two decision boundaries, we can classify upper two cross($\times$) clusters correctly.

Four Perceptrons in Single Layer



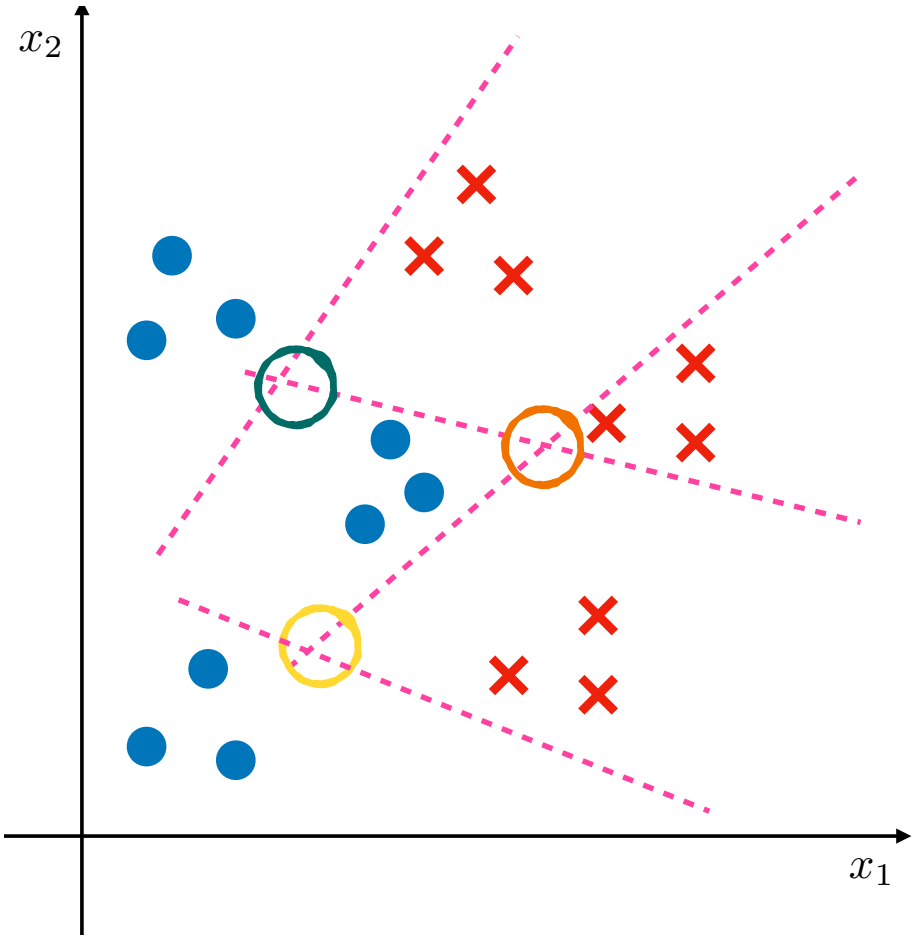
We can take the same approach for the lower right cross($\times$) cluster.

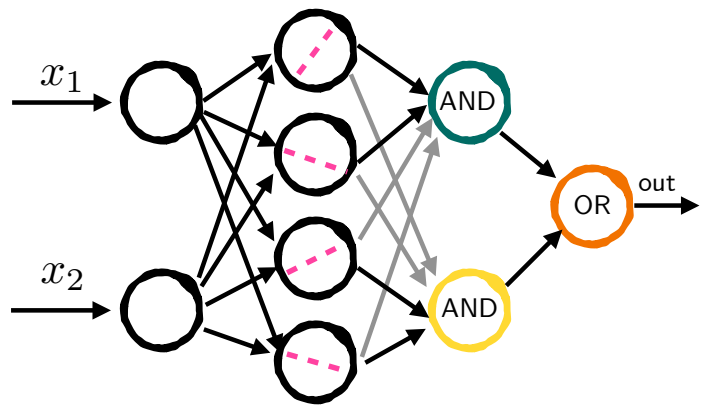
How to Combine?



How can we combine these four decision boundaries?

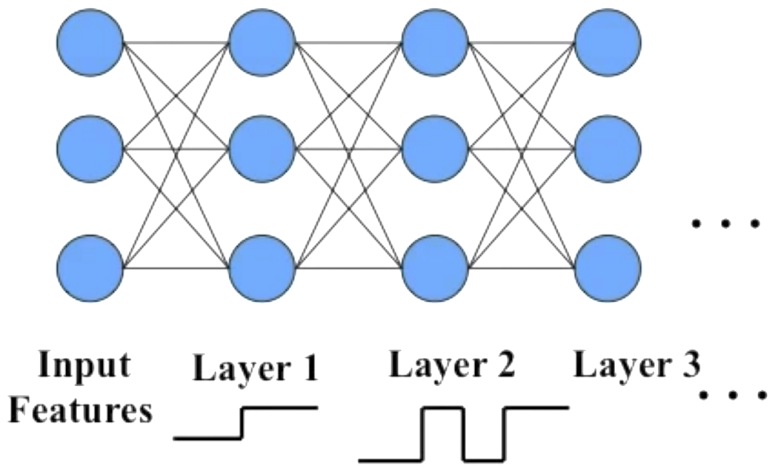
Three Layers! (Two Hidden Layers)





A Flexible Function Approximator

Perceptron is a simple building block, and a complex model can be expressed by building more perceptrons



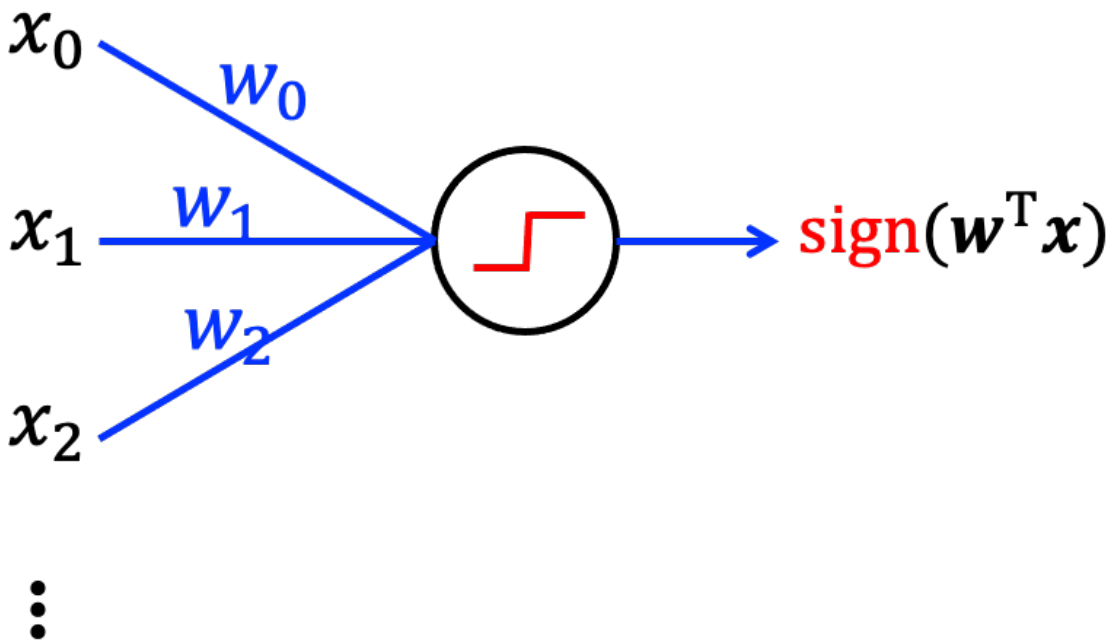
Deep Learning: A Number of Composition of Functions

- Remark1. the representation power of MLP is from composing functions, i.e., OR of two AND's
- Remark2. training is nothing but optimization, and the gradient method is heuristic but efficient
- Deep learning: a composition of differentiable functions, i.e., to approximate function F,

$$F(\boldsymbol{x}) \approx f_{\boldsymbol{w}_L}(f_{\boldsymbol{w}_{L-1}}(...f_{\boldsymbol{w}_1}(\boldsymbol{x})...))$$

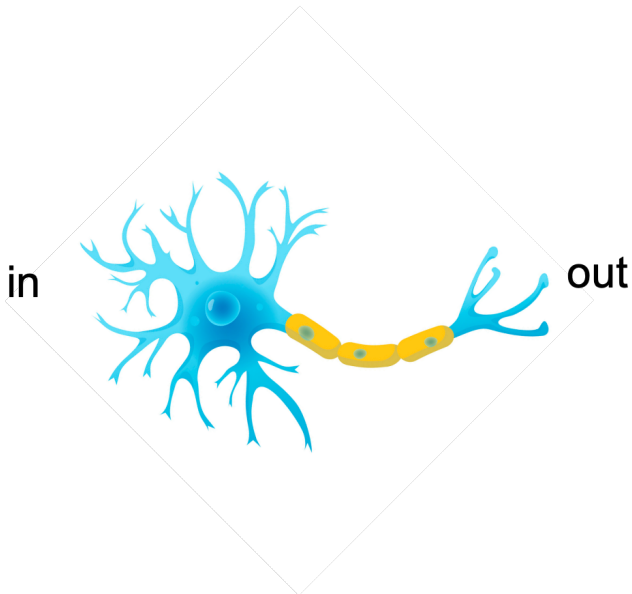
Perceptron

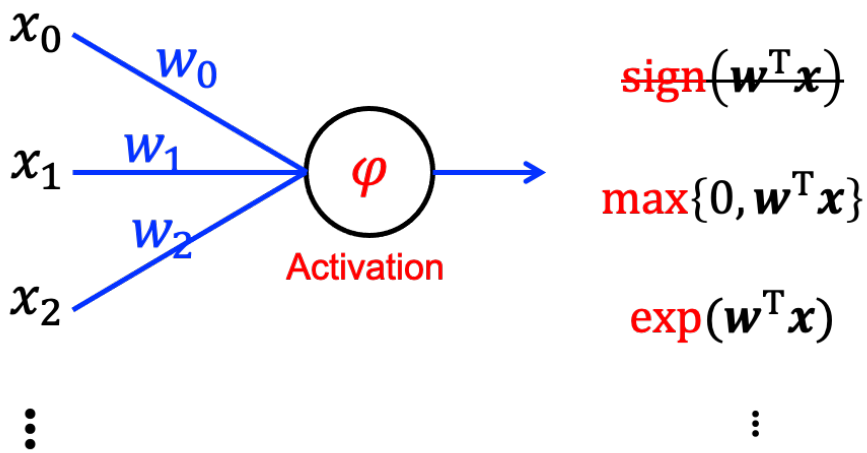
Although a single perceptron with sign is trainable, MLP with sign's is not since the derivative of sign is 0 almost everywhere



A Neuron: McCulloch-Pitts Model (1943)

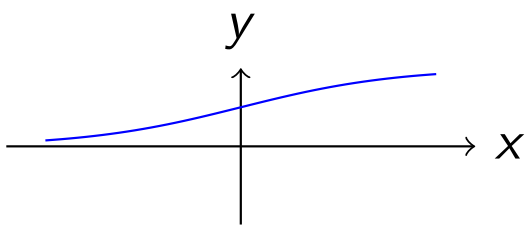
The sign function can be generalized and approximated by other activation functions



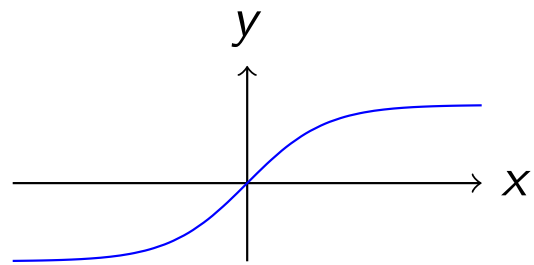


Activation Functions

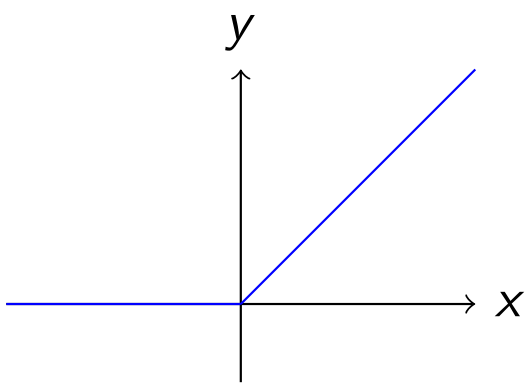
Sigmoid: $\varphi(x) = \frac{1}{1+\exp(-x)}$



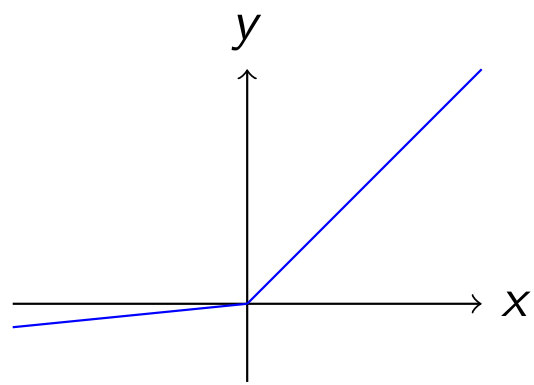
tanh: $\varphi(x) = \tanh(x)$



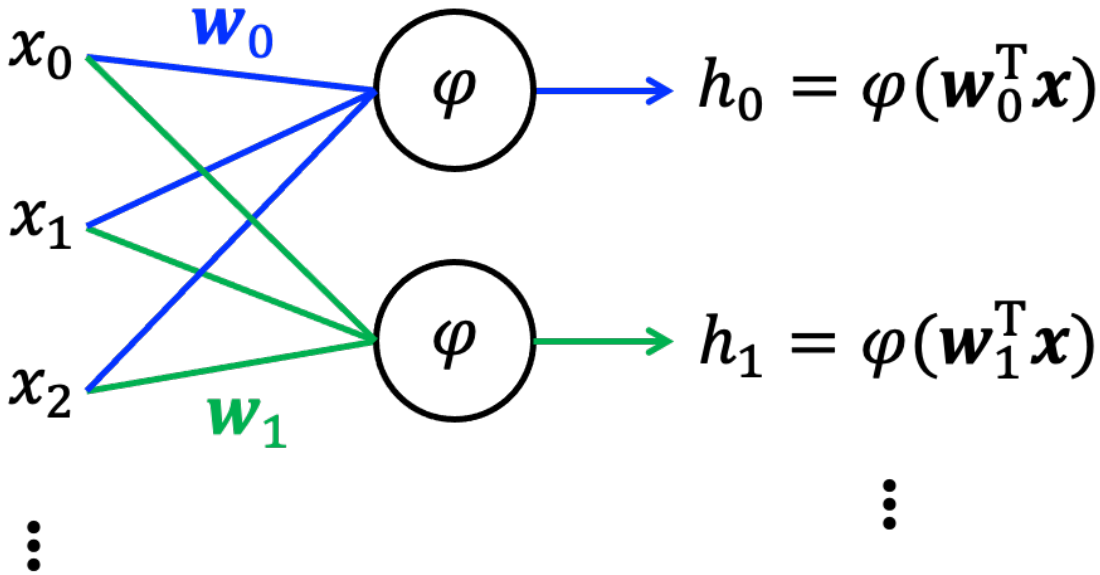
ReLU (*): $\varphi(x) = \max\{0, x\}$



Leaky ReLU: $\varphi(x) = \max\{0.1x, x\}$



Fully Connected Layer



A single layer with D input features and H hidden units can be written as

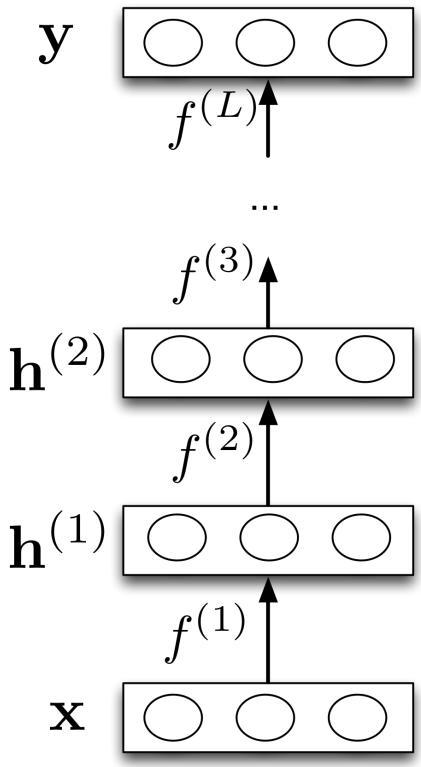
$$\mathbf{h} = \varphi(\boldsymbol{W}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{x} \in \mathbb{R}^D$, $\boldsymbol{W} \in \mathbb{R}^{H \times D}$, $\mathbf{b} \in \mathbb{R}^H$.

Multilayer Perceptron

- Stacks of fully-connected layers form a multilayer perceptron (MLP)

$$\boldsymbol{h}^{(1)} = f^{(1)}(\mathbf{x}) = \varphi(\boldsymbol{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\boldsymbol{h}^{(2)} = f^{(2)}(\boldsymbol{h}^{(1)}) = \varphi(\boldsymbol{W}_2\boldsymbol{h}^{(1)} + \mathbf{b}_2)$$
$$\vdots$$
$$\boldsymbol{y} = f^{(L)}(\boldsymbol{h}^{(L-1)})$$



- Or more simply:

$$\boldsymbol{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

- Neural nets provide modularity: we can implement each layer's computations as a black box.

Optimization

- Once we define a loss

$$\mathcal{L} = \sum_{n=1}^{N} \ell(\mathbf{y}_n, \hat{\mathbf{y}}_n)$$

- Let $\mathbf{h}_1$, $\mathbf{h}_2$,..., $\mathbf{h}_L$ be the outputs of hidden layers.

$$\begin{aligned}
\mathbf{h}_1 &= \sigma(\boldsymbol{W}_1 \mathbf{x} + \mathbf{b}_1) \\
\mathbf{h}_2 &= \sigma(\boldsymbol{W}_2 \mathbf{h}_1 + \mathbf{b}_2) \\
&\vdots \\
\hat{\mathbf{y}}_n = \mathbf{h}_L &= \boldsymbol{W}_L \mathbf{h}_{L-1} + \mathbf{b}_{L-1}
\end{aligned}$$

- We need to compute the partial derivative w.r.t. the model parameter $\boldsymbol{W}_\ell$ and $\mathbf{b}_\ell$ to train the model.

Question

Assume we model single-hidden layer neural network for regression with squared loss and sigmoid activation (ignore the bias term):

$$\mathcal{L} = \sum_{n=1}^{N} (y_n - w_2 \sigma(\boldsymbol{w}_1^\top \mathbf{x}_n))^2$$

$$w_2 \in \mathbb{R}, \boldsymbol{w}_1 \in \mathbb{R}^D, \mathbf{x}_n \in \mathbb{R}^D$$

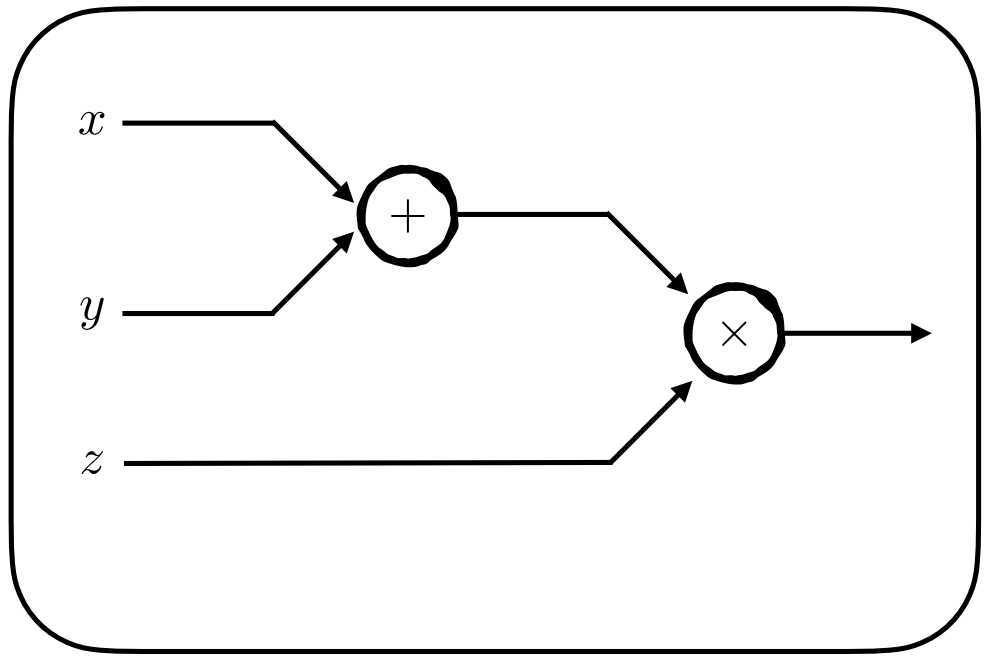
- To use gradient descent, we need to compute partial derivative w.r.t. parameters.
- What would be
- $\partial \mathcal{L} / \partial w_2 =$?
- $\partial \mathcal{L} / \partial \boldsymbol{w}_1 =$?

Compute Partial Derivatives

- All we need to train the model is the partial derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{b}_\ell}$ for all $\ell = 1, ..., N$.
- You may wish to compute the partial derivative analytically (by writing down the equations on a paper).
- This doesn't seem good idea since
  - You need lots of matrix calculus (and paper)
  - What if you want to change the loss? you need to start from beginning
  - Not feasible for complex models.
- *Back-propagation* is the solution.

Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

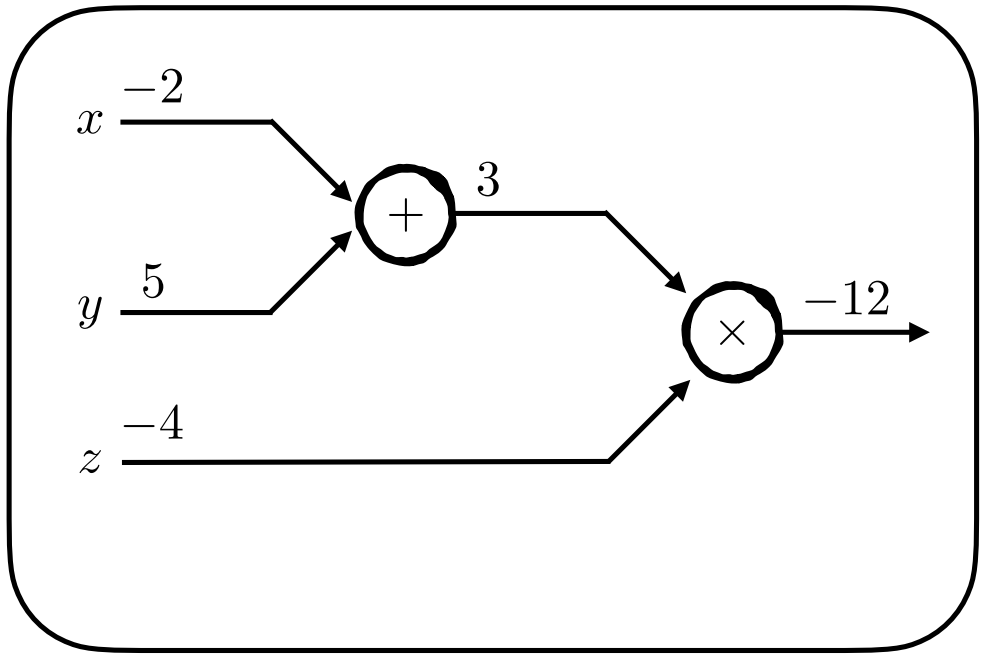


Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

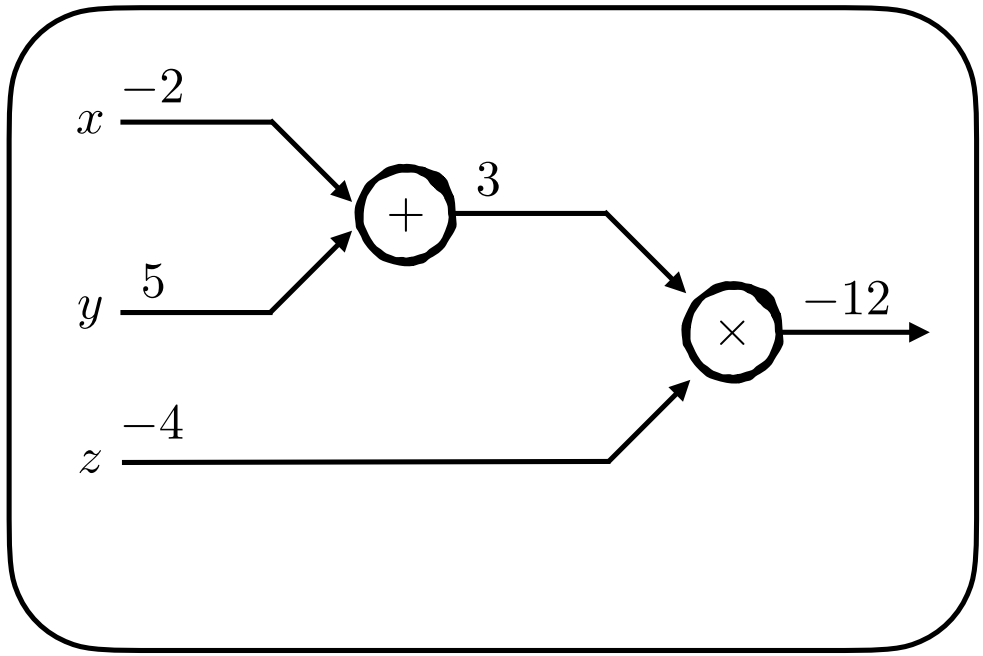
$$q = x + y$$

$$f = qz$$



Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$



$$q = x + y$$

$$f = qz$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$

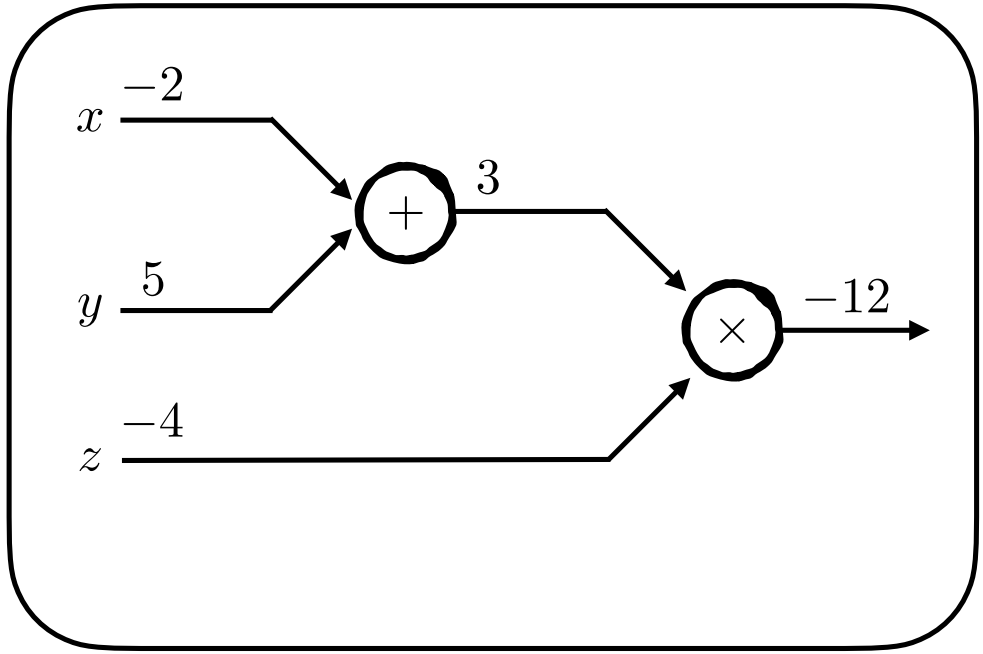
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



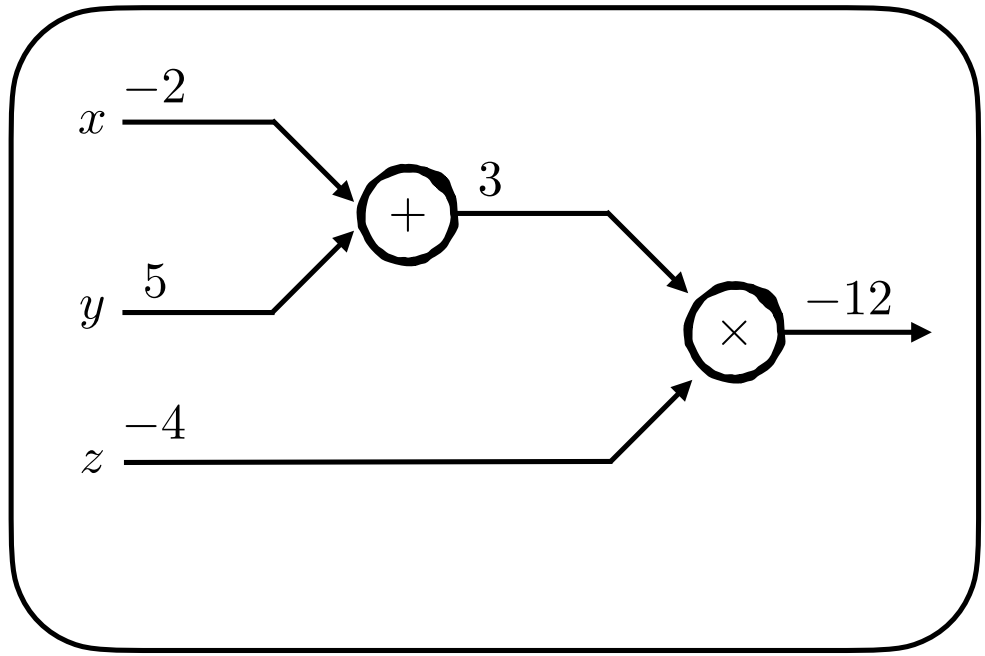
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



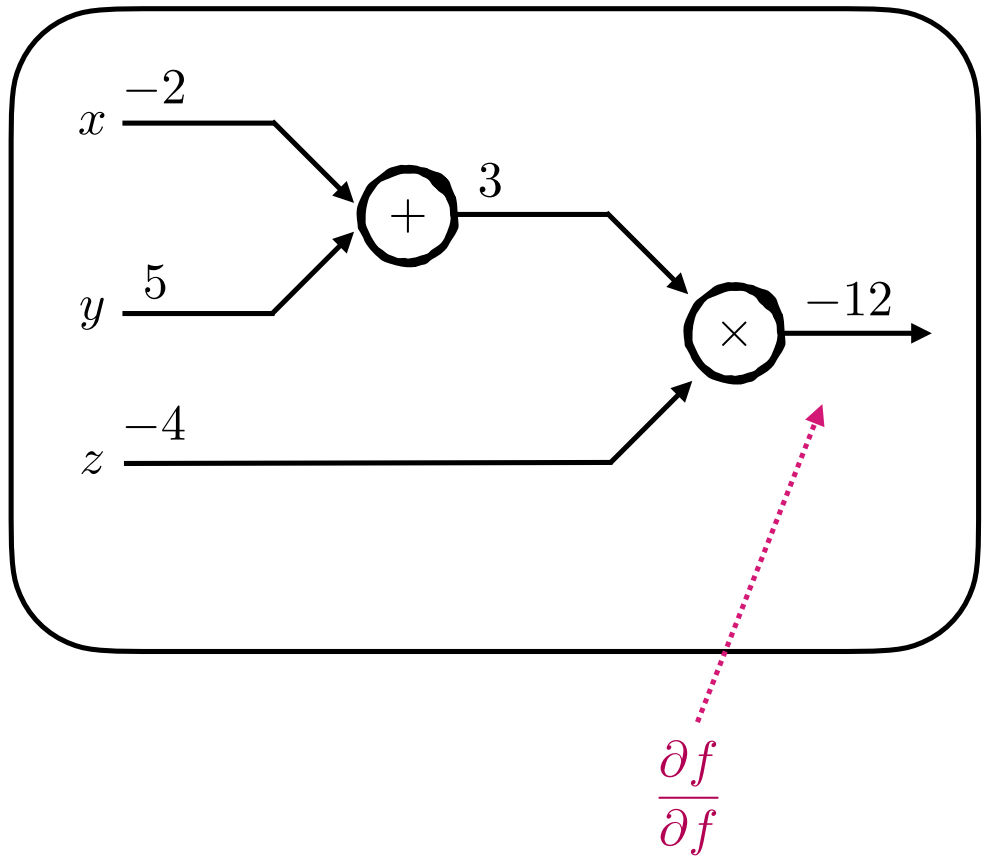
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



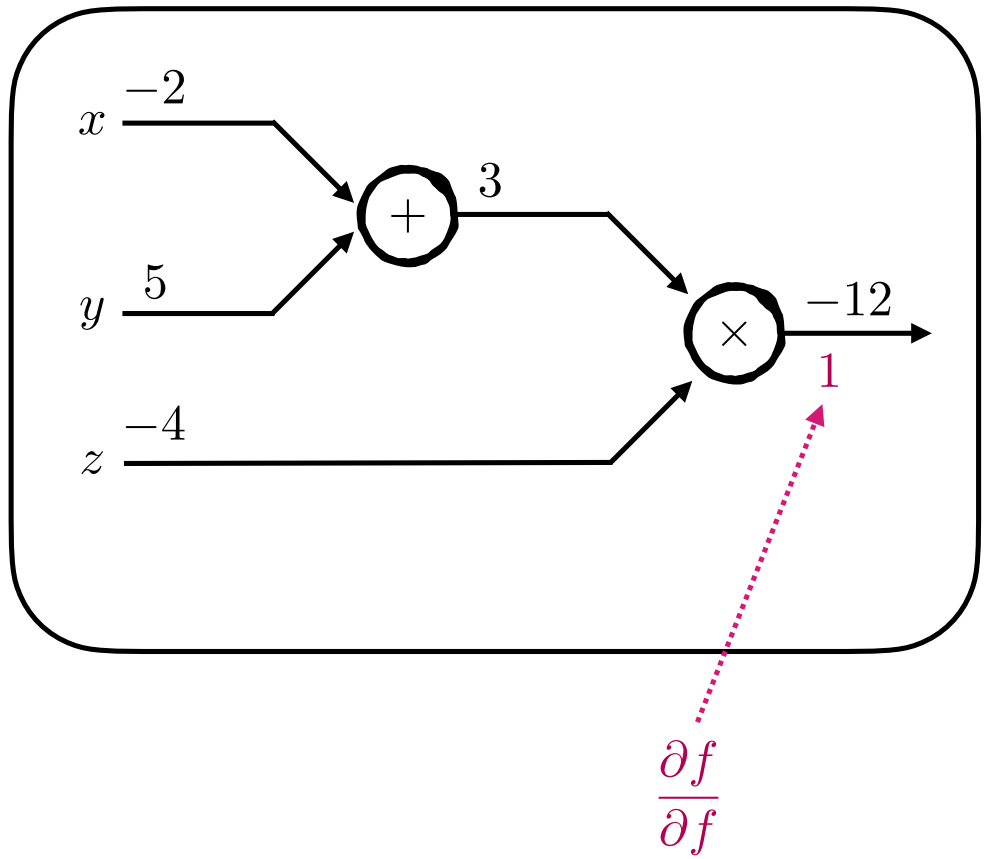
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



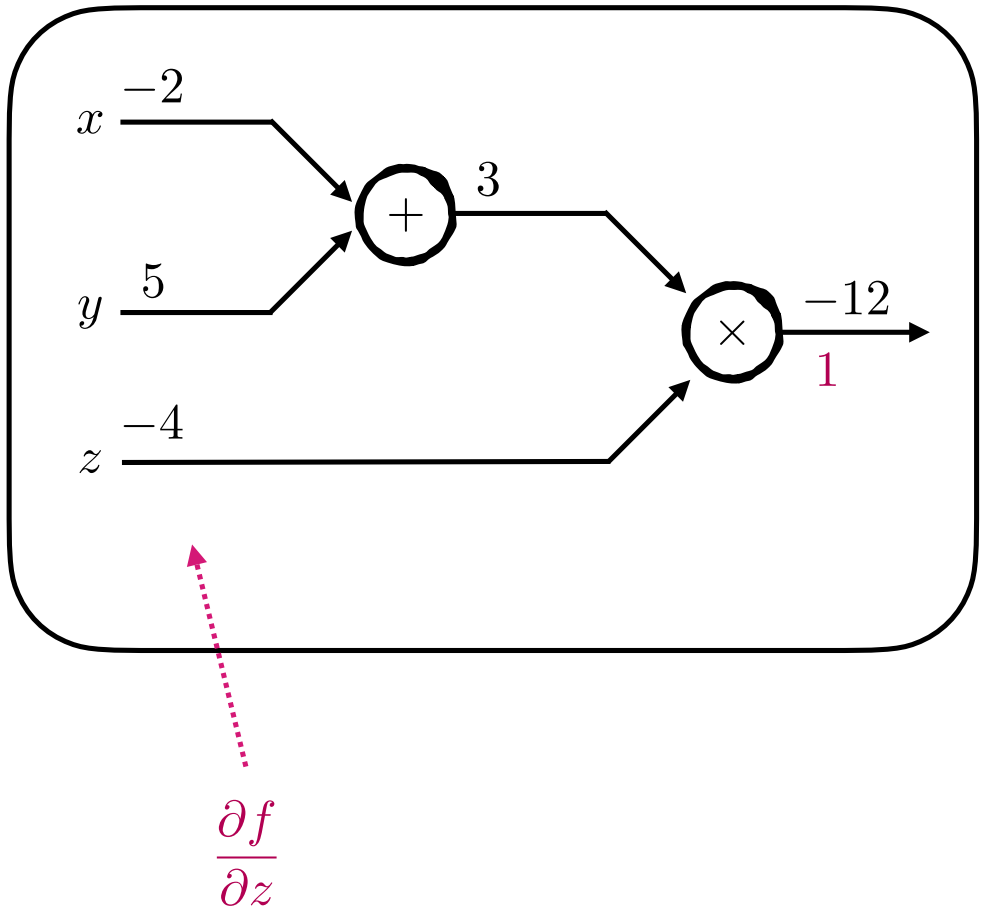
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \boxed{\frac{\partial f}{\partial z} = q}$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



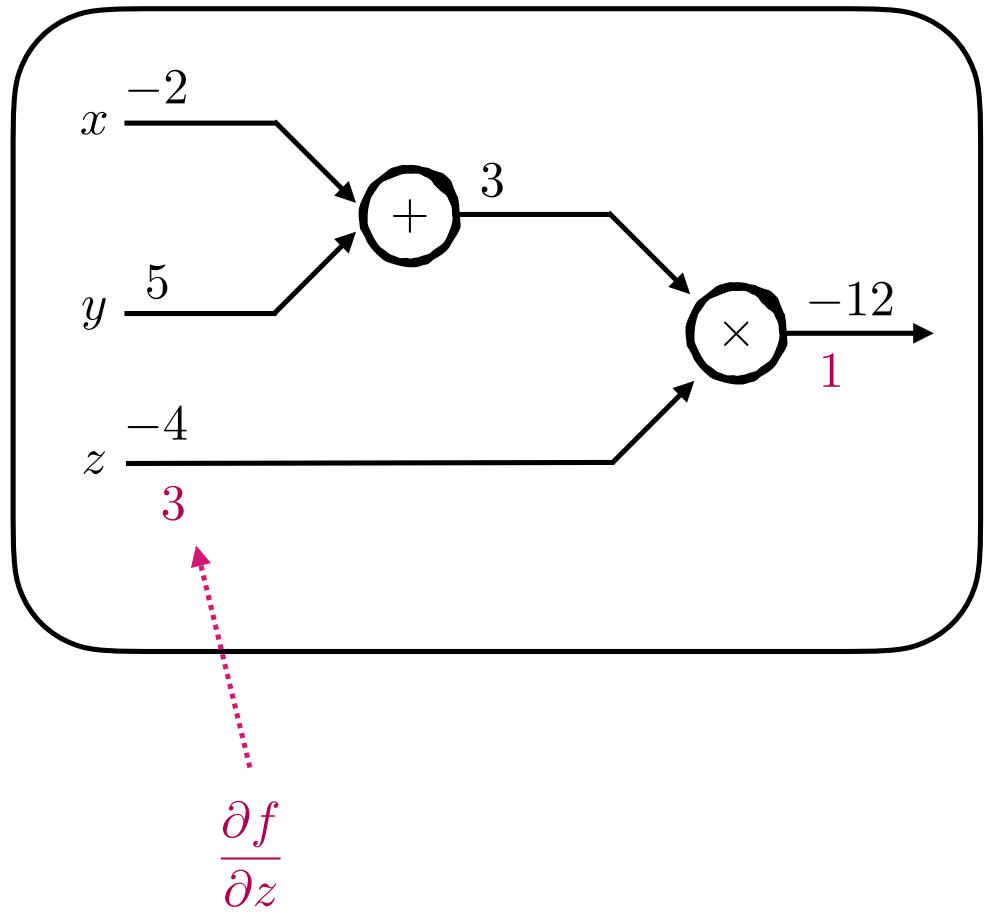
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \boxed{\frac{\partial f}{\partial z} = q}$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



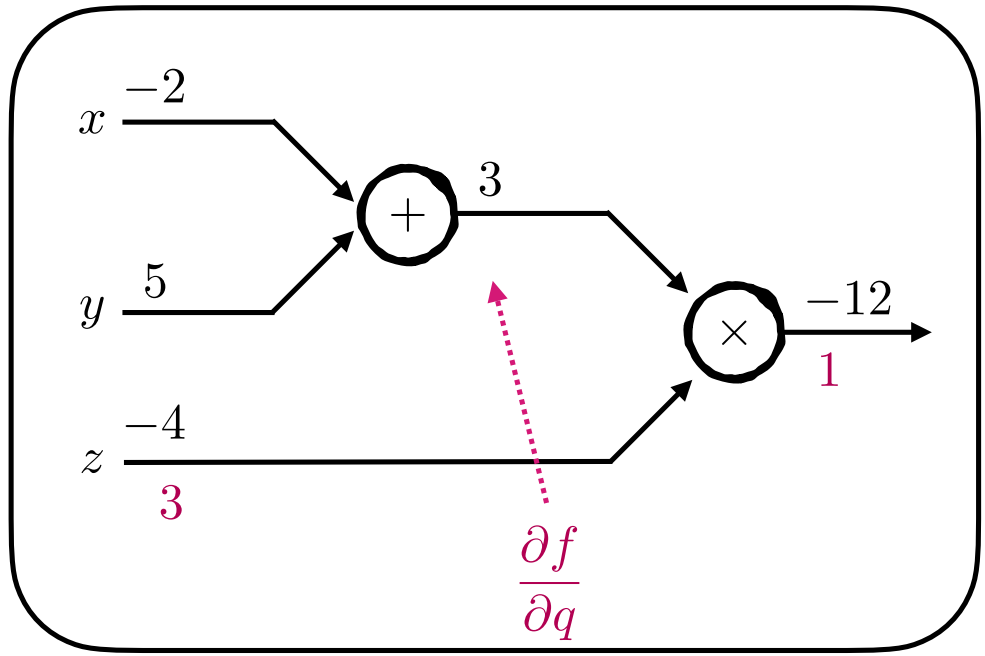
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \boxed{\frac{\partial f}{\partial q} = z} \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



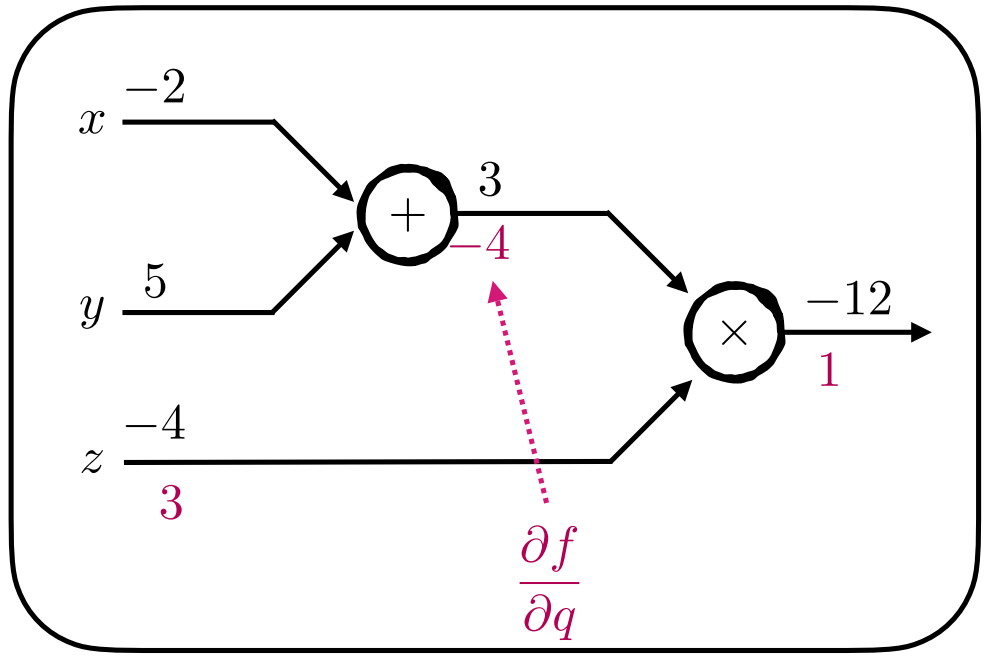
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \boxed{\frac{\partial f}{\partial q} = z} \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



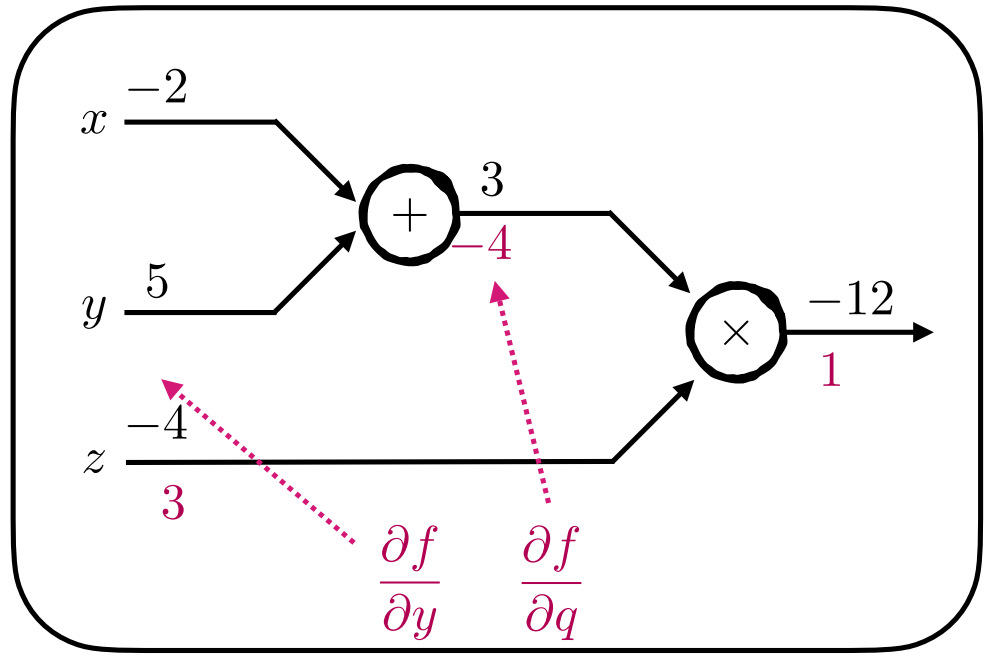
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



Chain rule: $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$

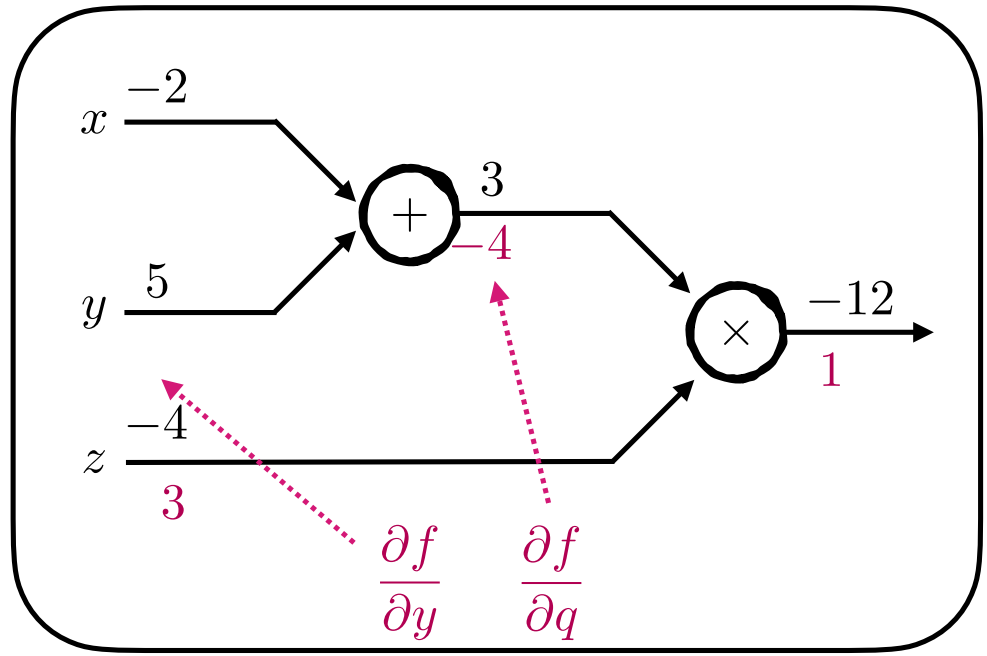
Backpropagation: Simple example

$f(x, y, z) = (x + y)z$ Let $x = -2$, $y = 5$, $z = -4$

$q = x + y$ $\quad \frac{\partial q}{\partial x} = 1$ $\quad \boxed{\frac{\partial q}{\partial y} = 1}$

$f = qz$ $\quad \boxed{\frac{\partial f}{\partial q} = z}$ $\quad \frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$



Chain rule: $\frac{\partial f}{\partial y} = \boxed{\frac{\partial f}{\partial q}}\boxed{\frac{\partial q}{\partial y}}$

Upstream Gradient | Downstream Gradient

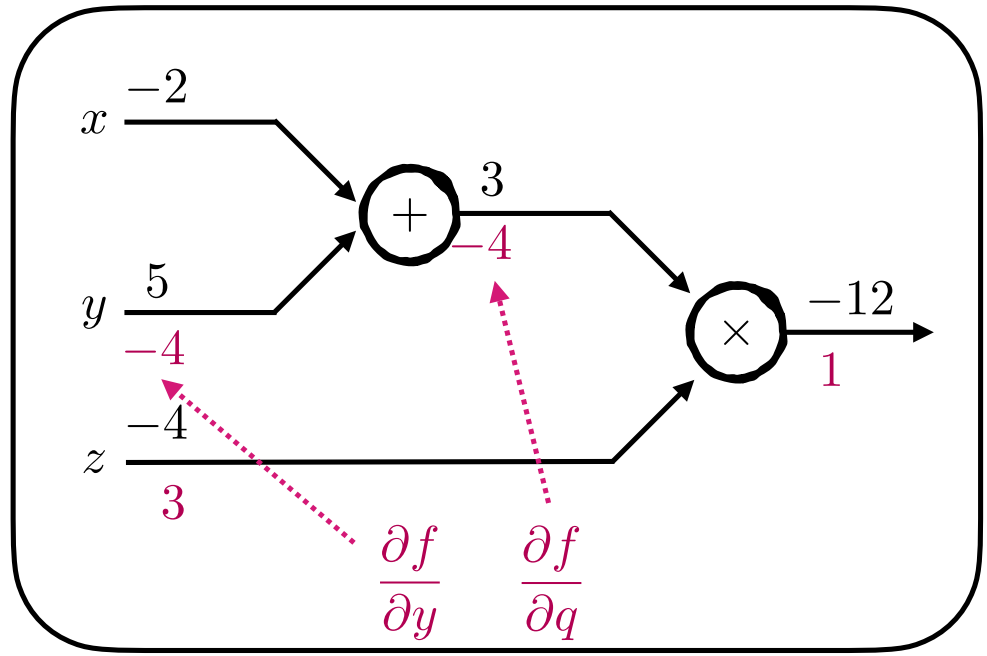
Backpropagation: Simple example

$f(x, y, z) = (x + y)z$ Let $x = -2$, $y = 5$, $z = -4$

$q = x + y$ $\quad \frac{\partial q}{\partial x} = 1$ $\quad \boxed{\frac{\partial q}{\partial y} = 1}$

$f = qz$ $\quad \boxed{\frac{\partial f}{\partial q} = z}$ $\quad \frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$



Chain rule: $\frac{\partial f}{\partial y} = \boxed{\frac{\partial f}{\partial q}}\boxed{\frac{\partial q}{\partial y}}$

Upstream Gradient | Downstream Gradient

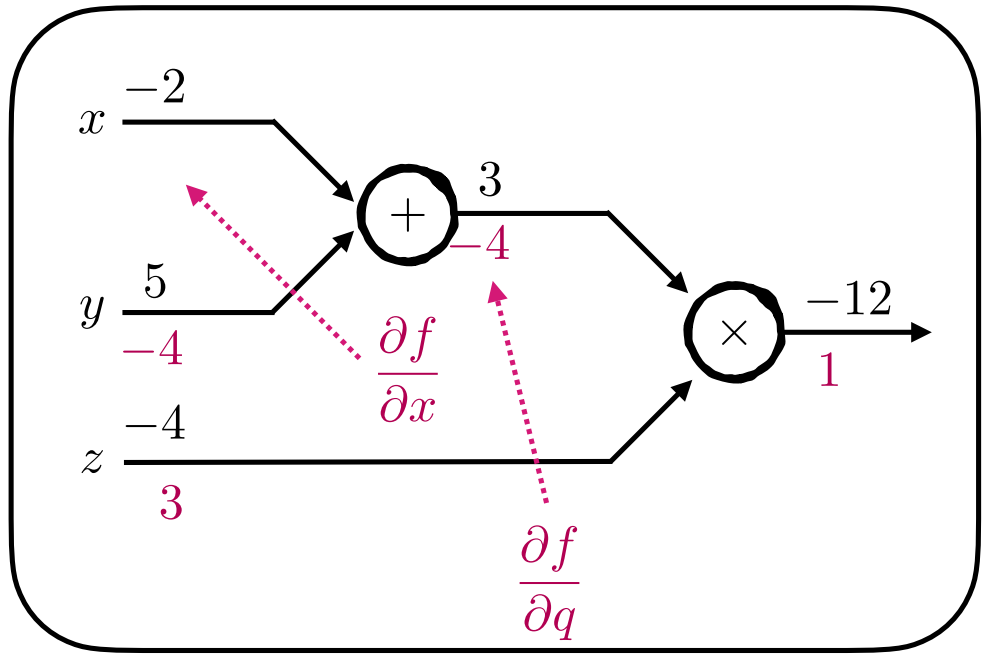
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



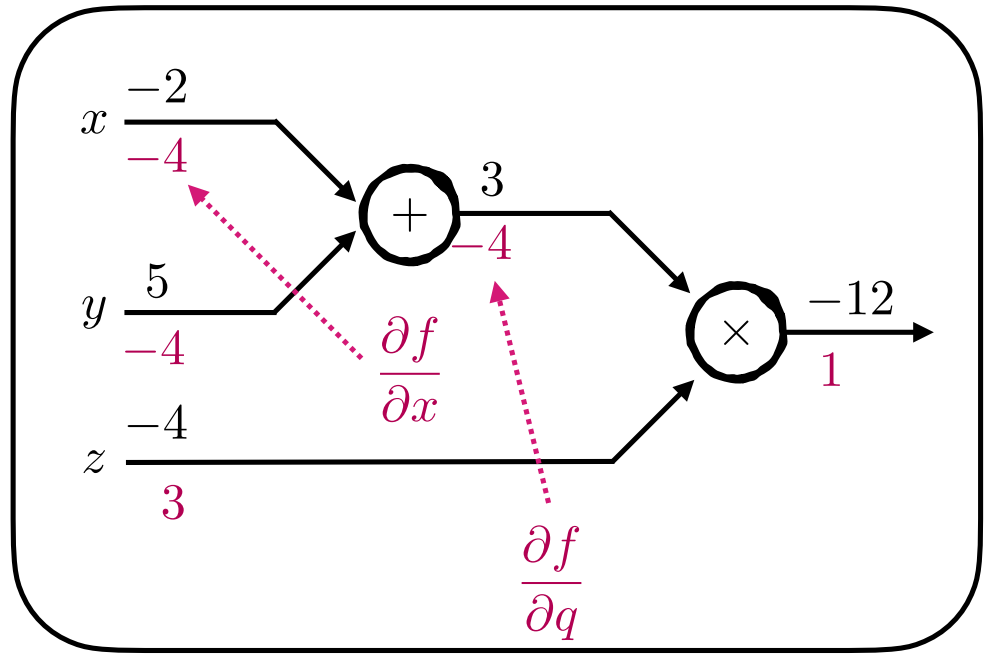
Backpropagation: Simple example

$$f(x, y, z) = (x + y)z \quad \text{Let} \quad \begin{aligned} x &= -2 \\ y &= 5 \\ z &= -4 \end{aligned}$$

$$q = x + y \qquad \boxed{\frac{\partial q}{\partial x} = 1} \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \boxed{\frac{\partial f}{\partial q} = z} \qquad \frac{\partial f}{\partial z} = q$$

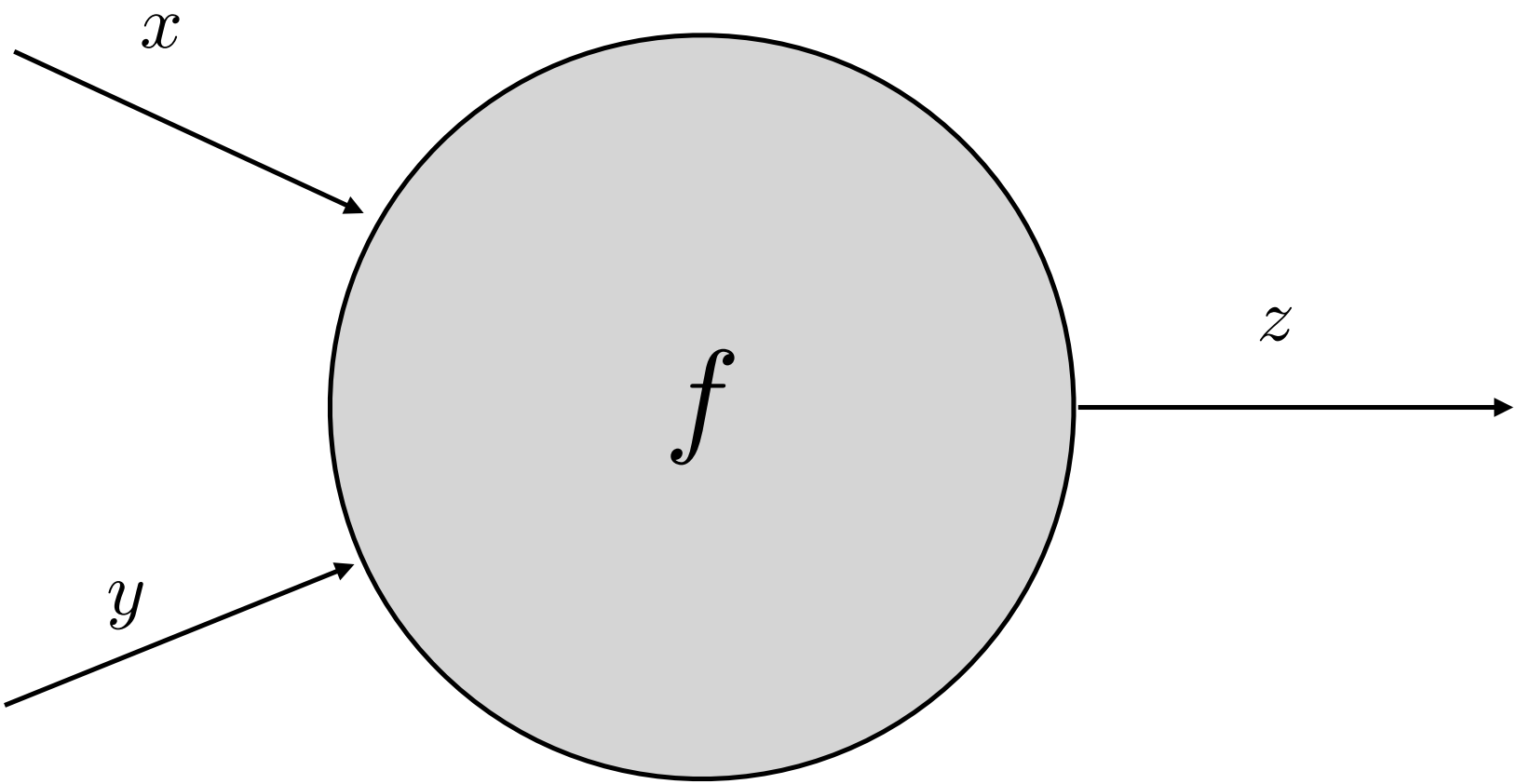
$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} = ?$$



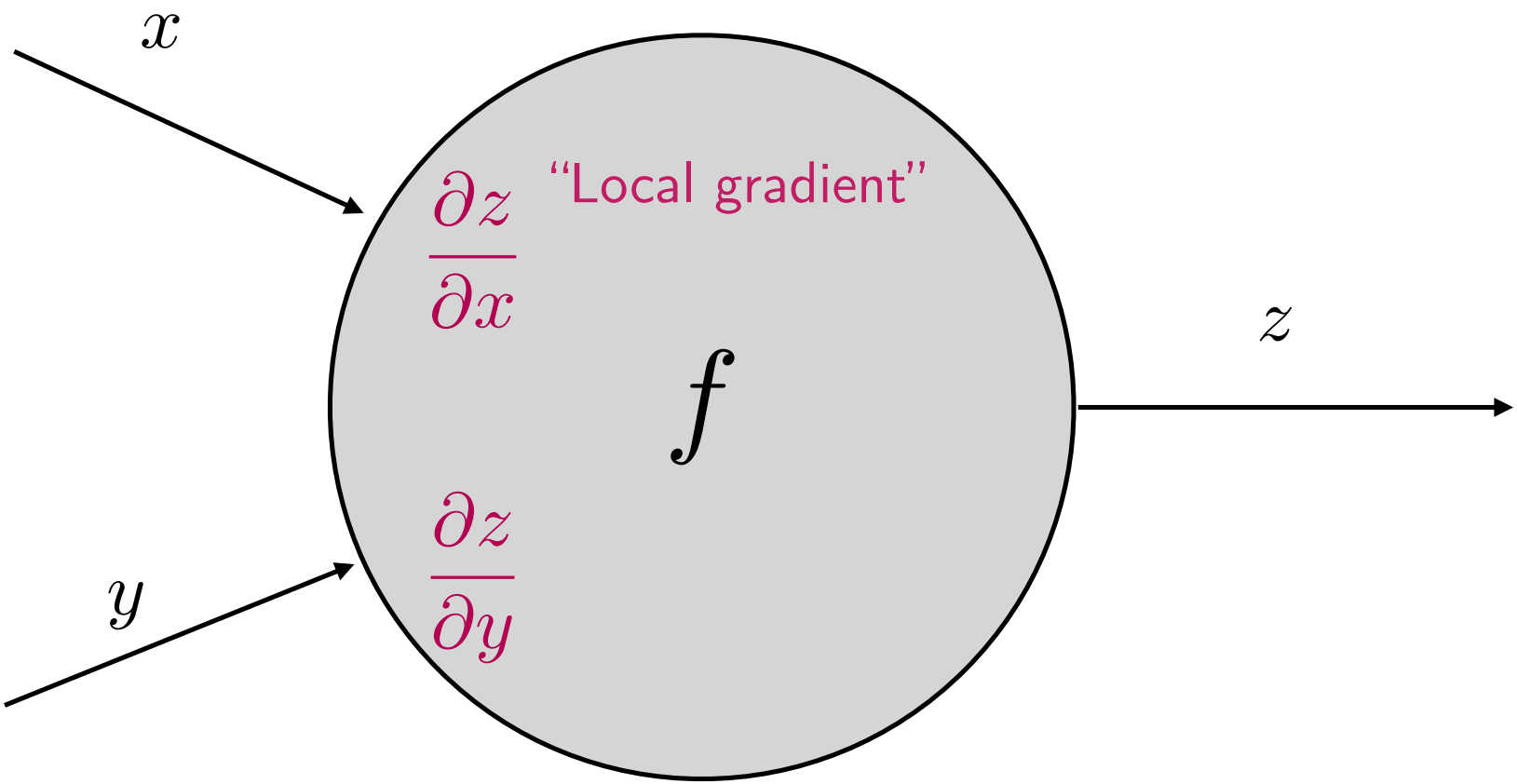
Chain rule: $\frac{\partial f}{\partial x} = \boxed{\frac{\partial f}{\partial q}} \boxed{\frac{\partial q}{\partial x}}$

Upstream Gradient | Downstream Gradient

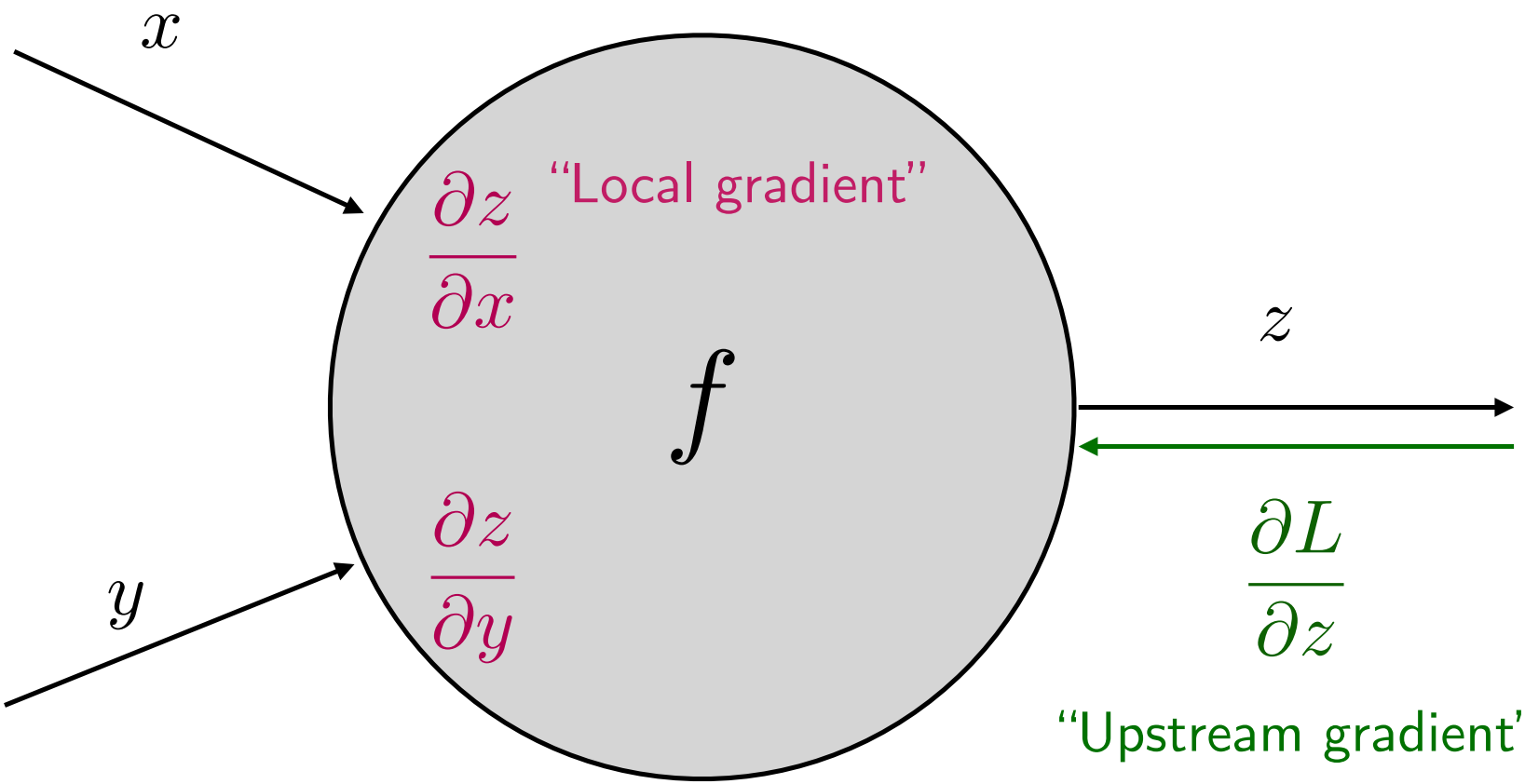
Backpropagation: A Closer Look



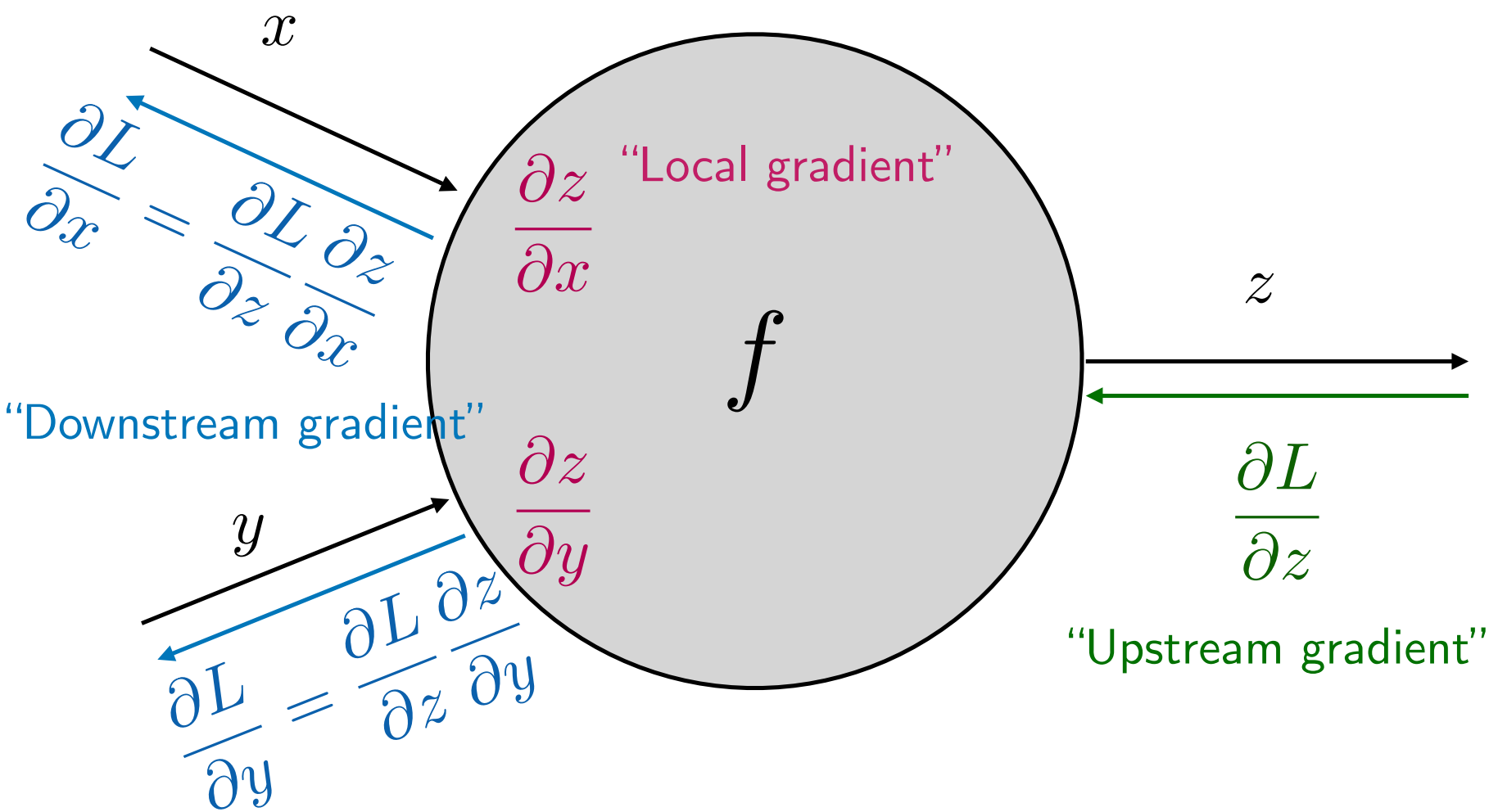
Backpropagation: A Closer Look



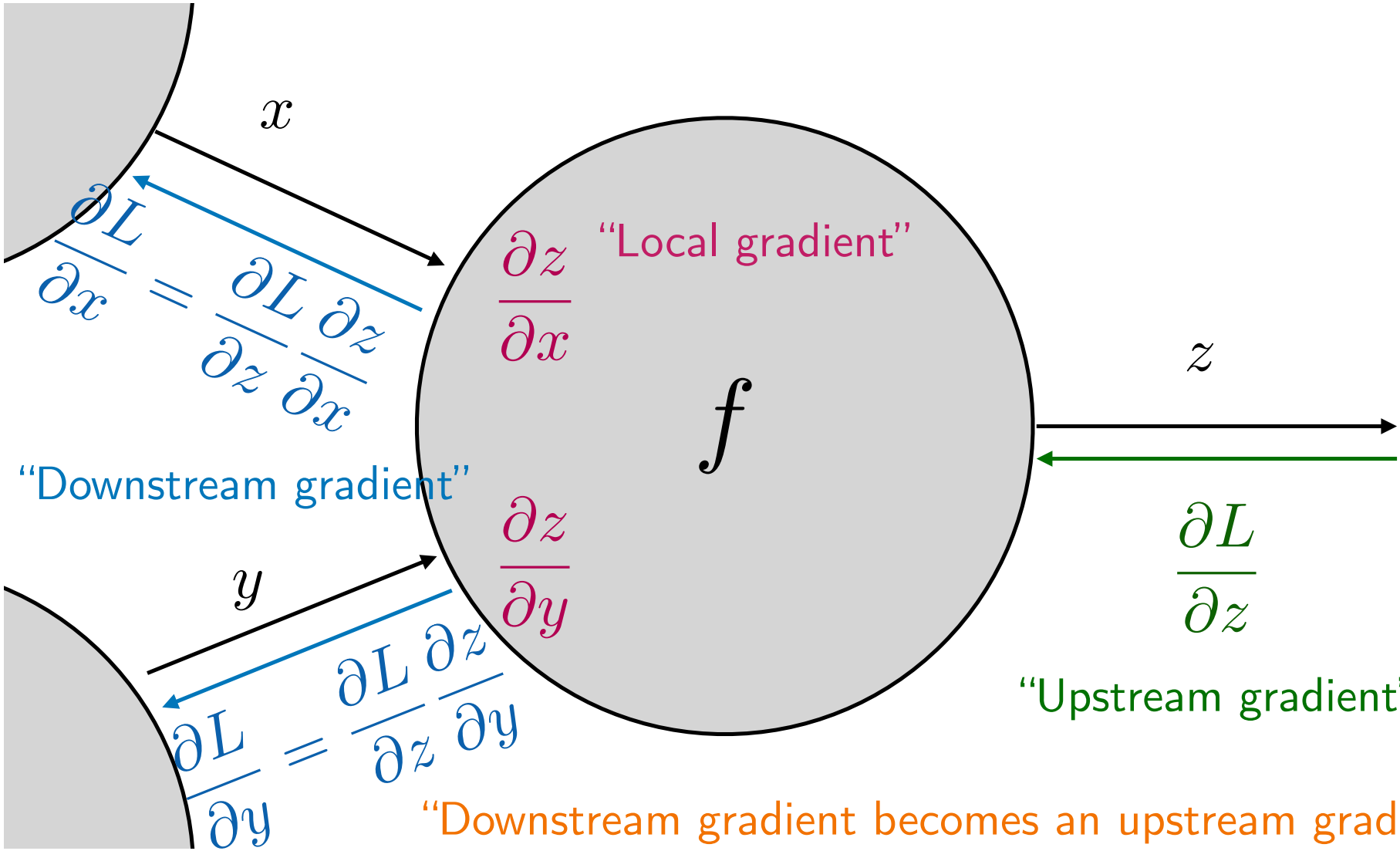
Backpropagation: A Closer Look



Backpropagation: A Closer Look

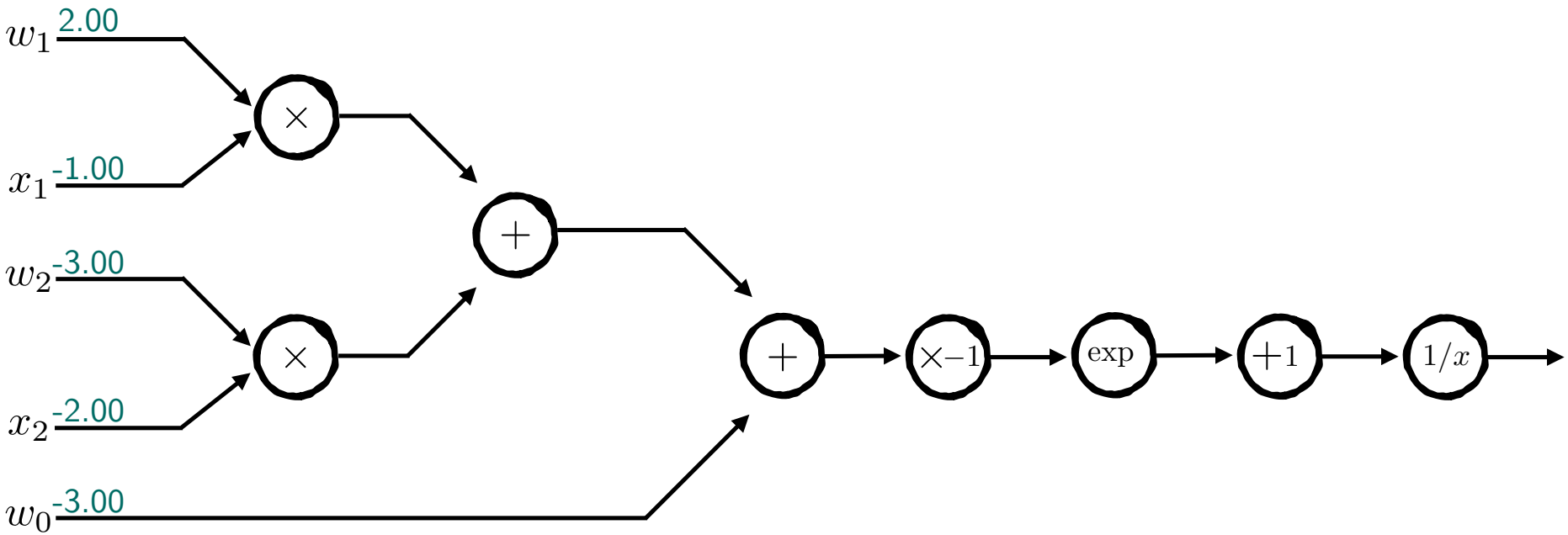


Backpropagation: A Closer Look



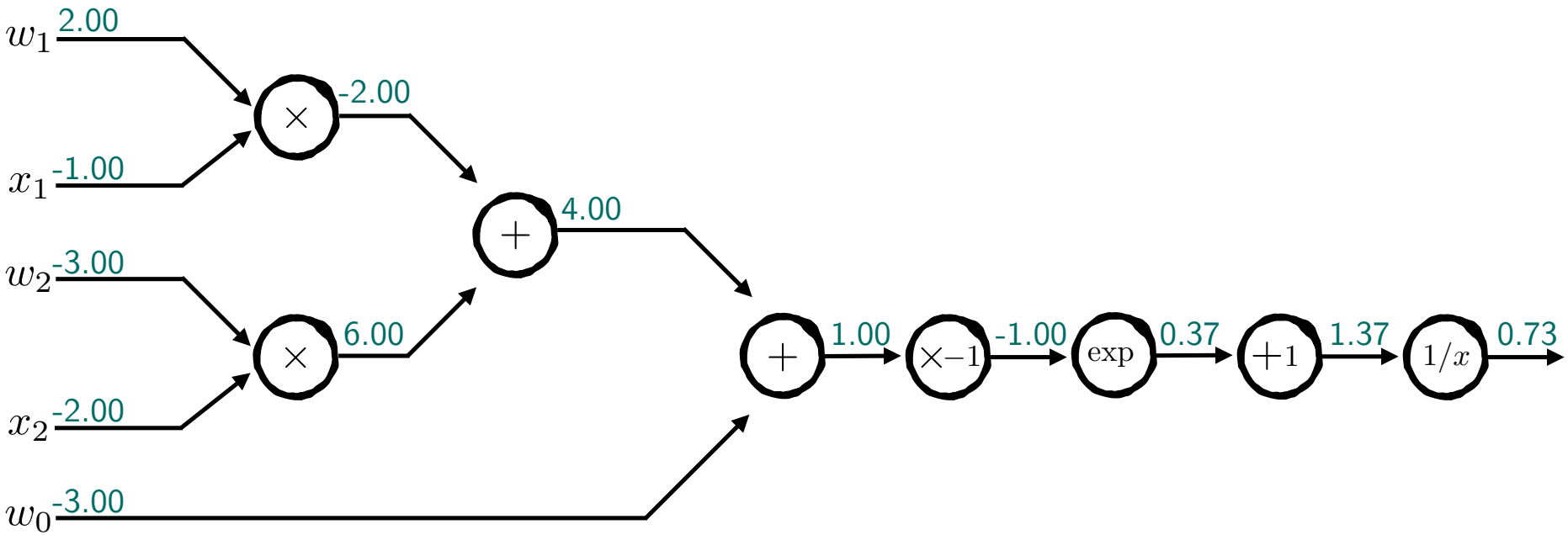
Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



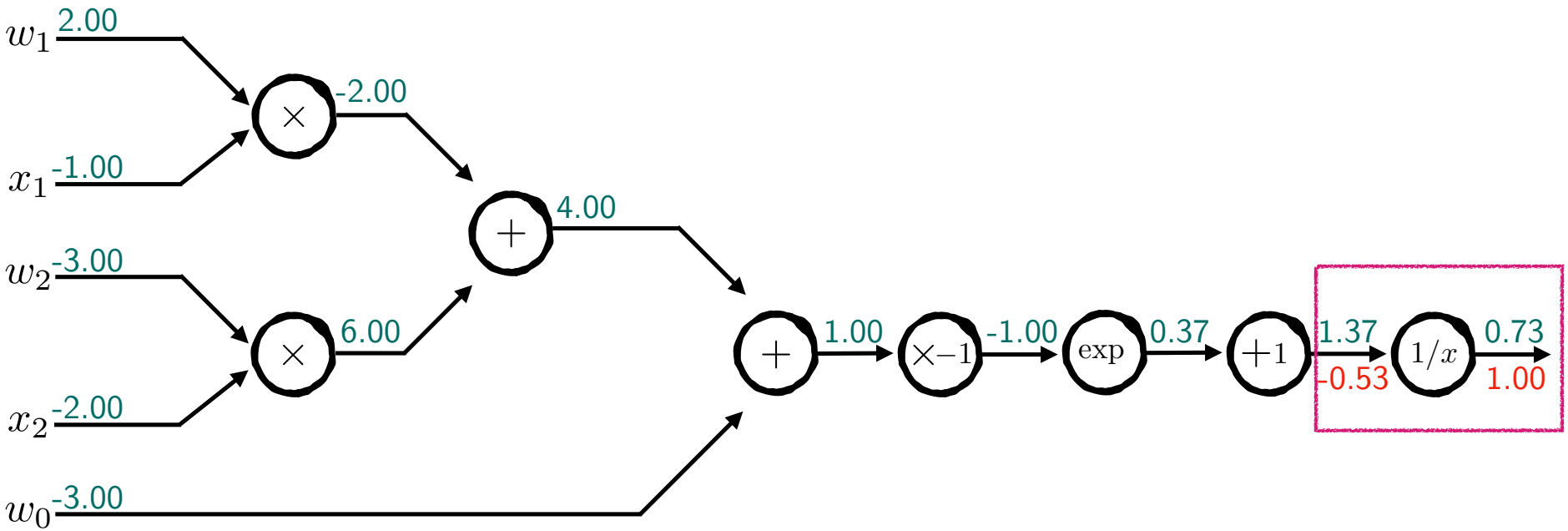
Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



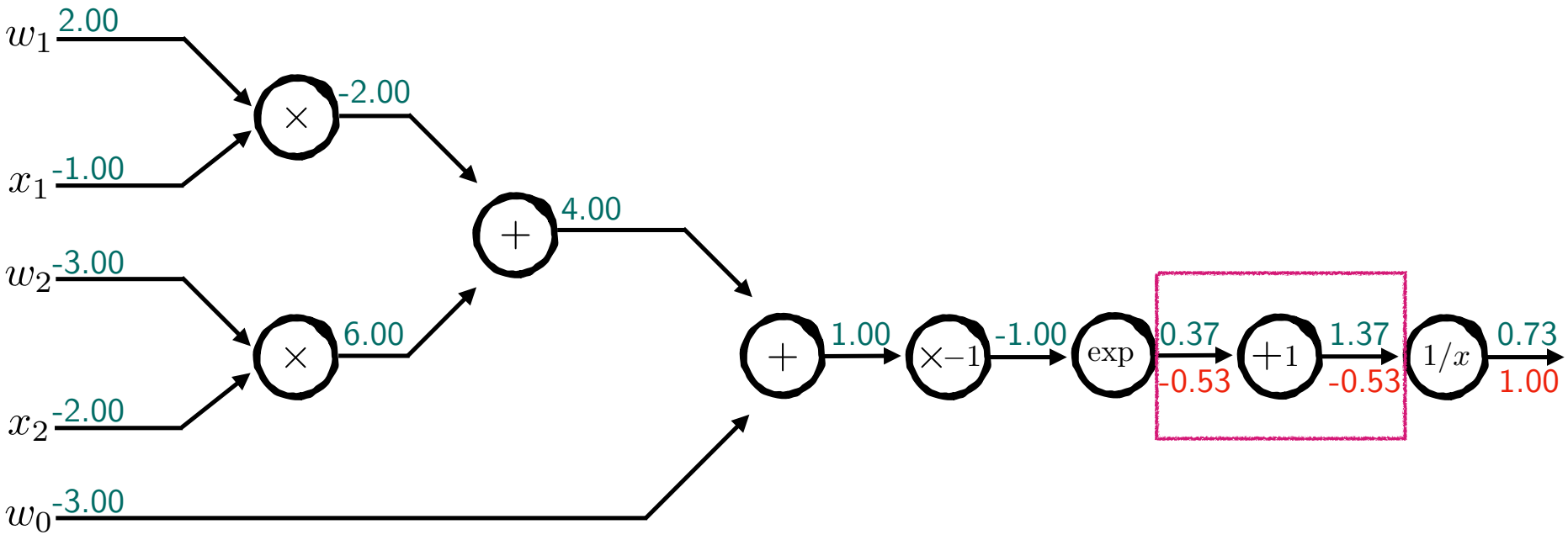
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

Upstream Gradient — Local Gradient

Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



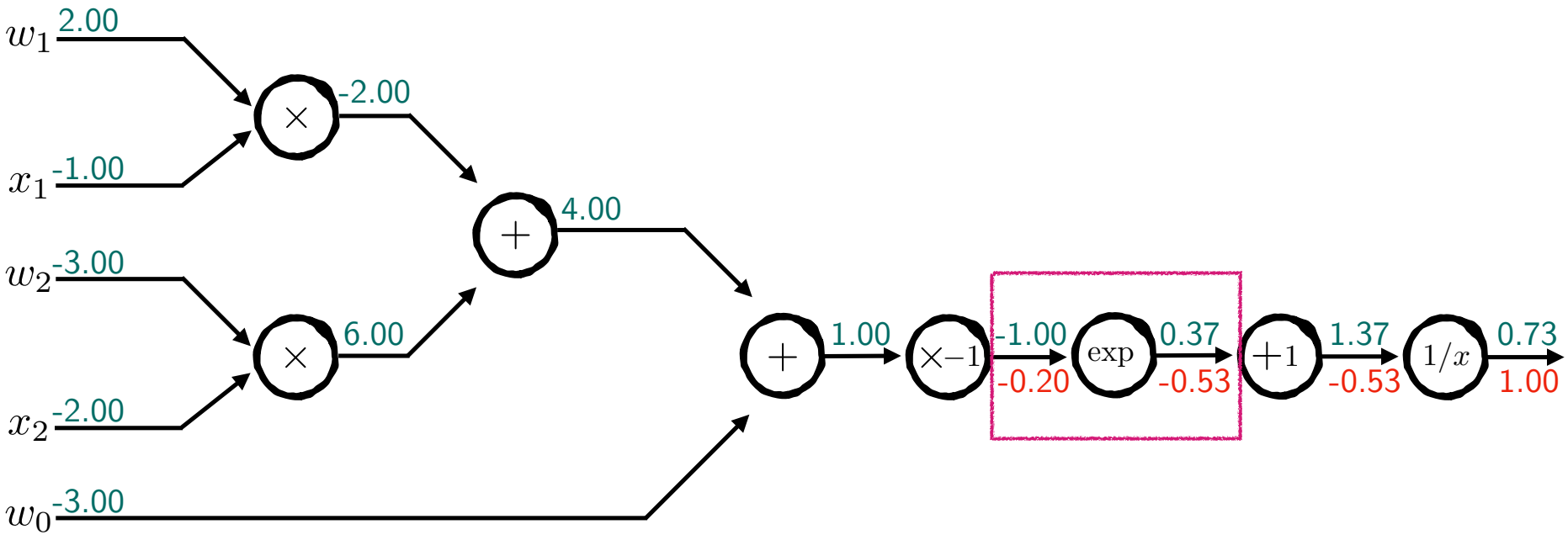
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$(-0.53)(1) = -0.53$$

Upstream Gradient | Local Gradient

Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$

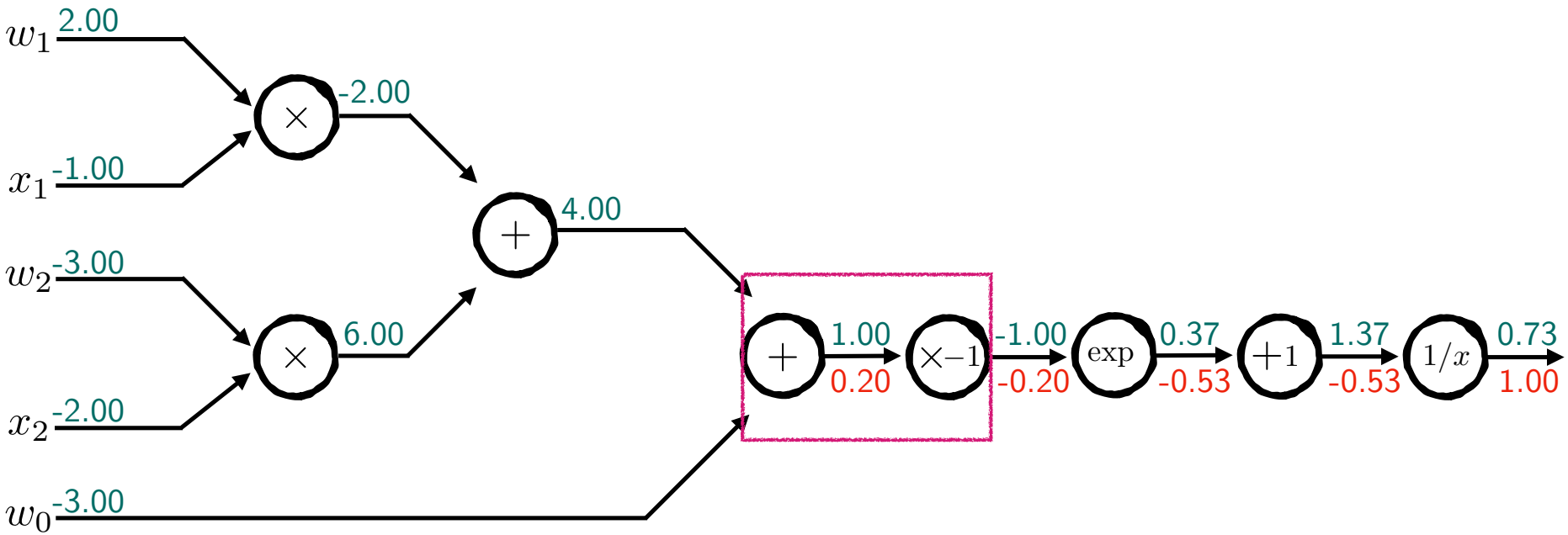


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$\underbrace{(-0.53)}_{\text{Upstream Gradient}}\underbrace{(e^{-1})}_{\text{Local Gradient}} = -0.20$$

Backpropagation: Softmax Layer

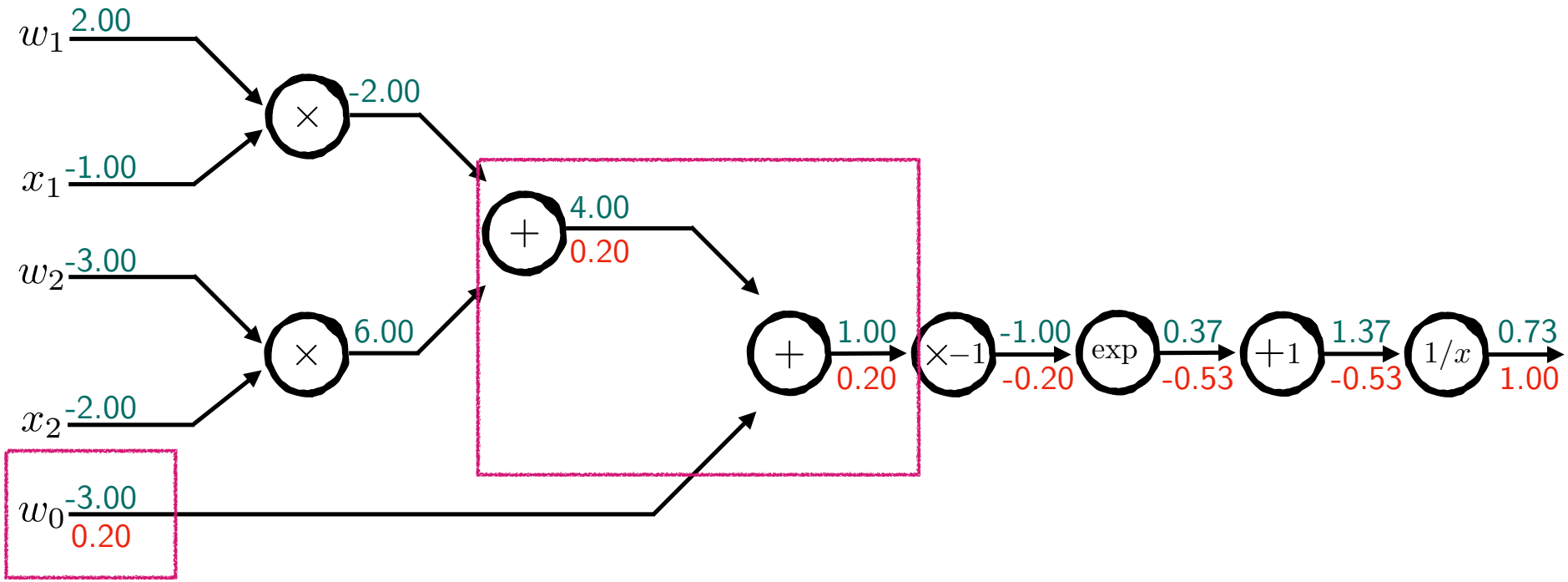
$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

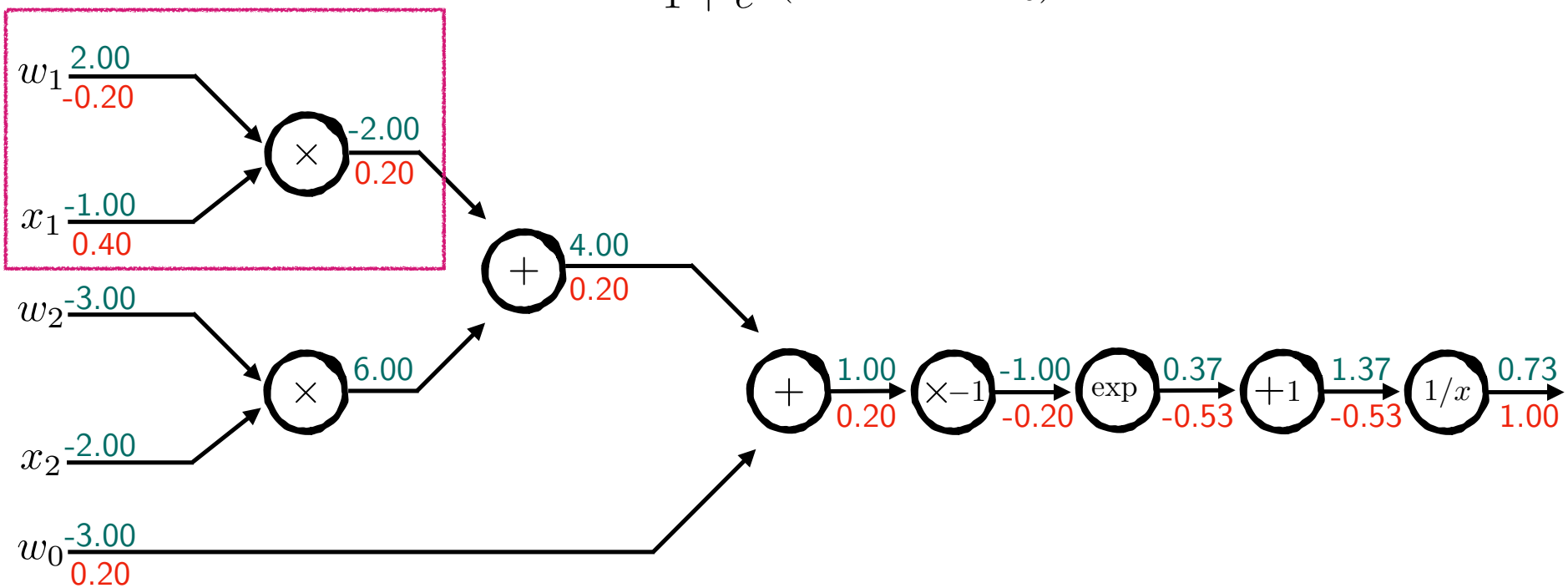
Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



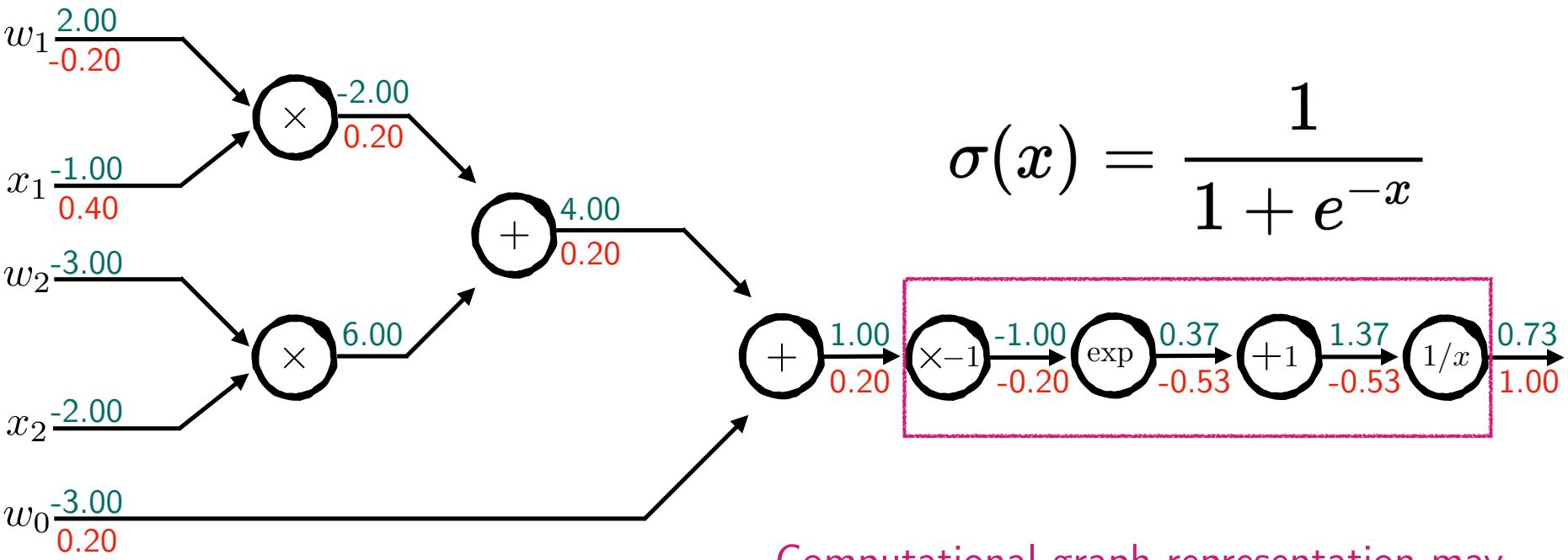
Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$



Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$

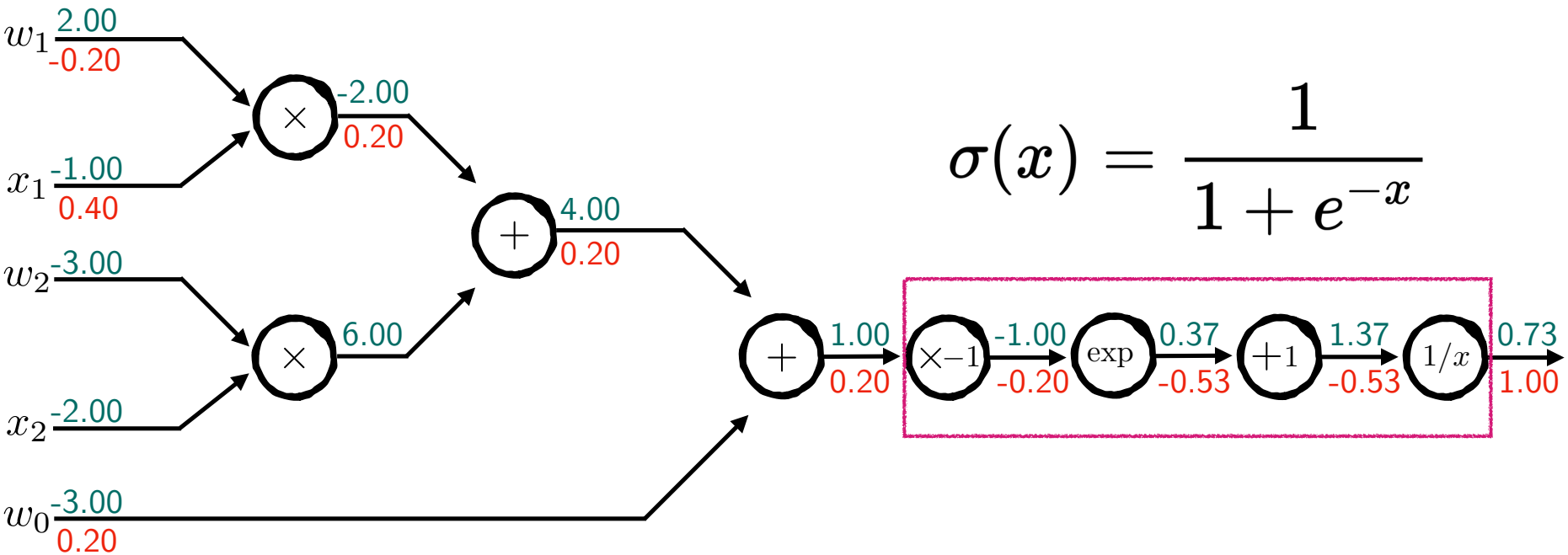


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$

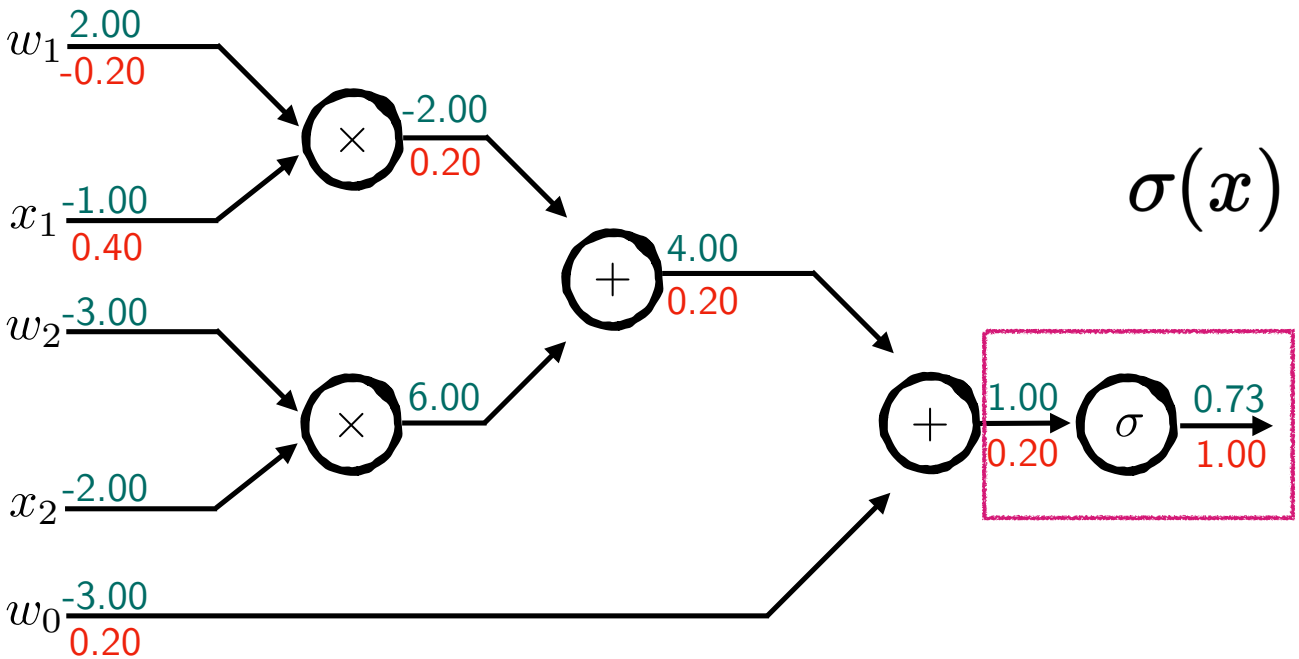


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$

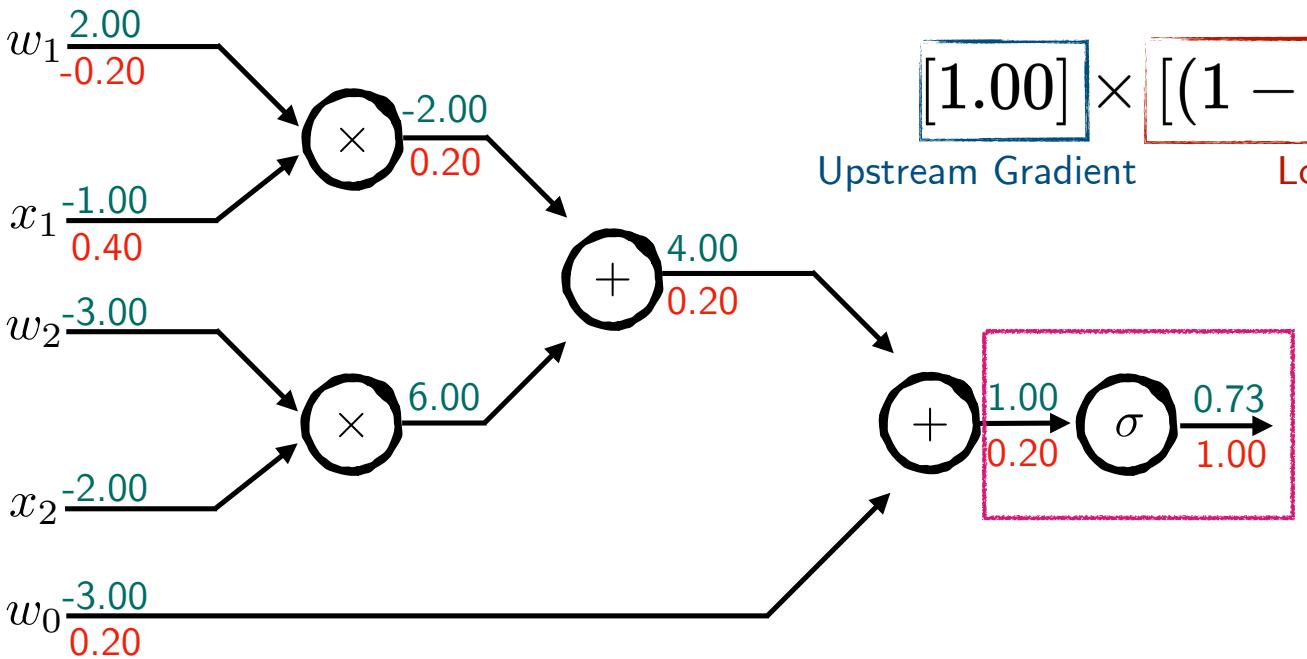


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

Backpropagation: Softmax Layer

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_0)}}$$

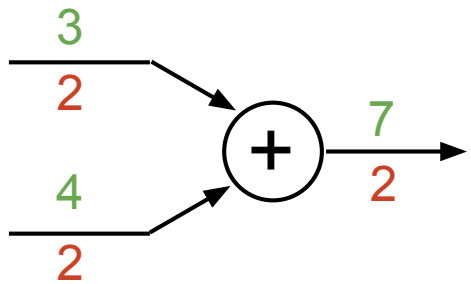


$$\underbrace{[1.00]}_{\text{Upstream Gradient}} \times \underbrace{[(1-0.73)(0.73)]}_{\text{Local Gradient}} = 0.2$$

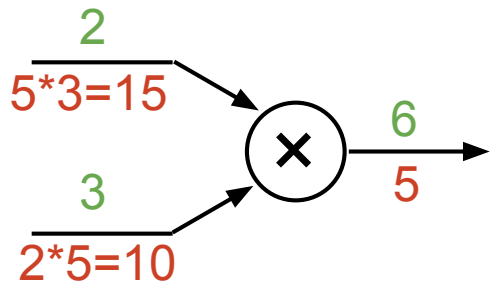
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

Patterns in Gradient Flow

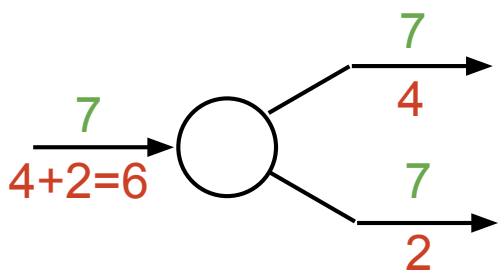
add gate: gradient distributor



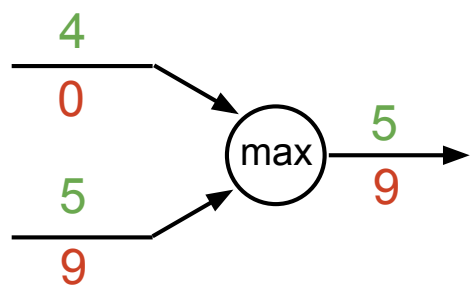
mul gate: “swap multiplier”



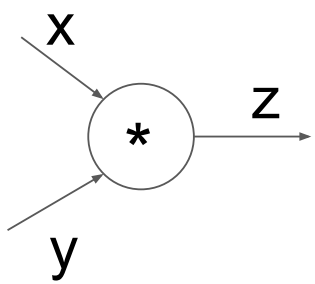
copy gate: gradient adder



max gate: gradient router



Implementation of Multiply Operation



(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Coming Back to NN

- All we need to train the model is the partial derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{b}_l}$ for all $\ell = 1, ..., N$.
- Let's compute $\partial \mathcal{L} / \partial \boldsymbol{W}_L$ first via chain-rule:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_L} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_L} \frac{\partial \mathbf{h}_L}{\partial \boldsymbol{W}_L}$$

- We can do the (stochastic) gradient descent with the value obtained.

Coming Back to NN

- All we need to train the model is the partial derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_\ell}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{b}_l}$ for all $\ell = 1, ..., N$.
- Let's compute $\partial \mathcal{L} / \partial \boldsymbol{W}_L$ first via chain-rule:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_L} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_L} \frac{\partial \mathbf{h}_L}{\partial \boldsymbol{W}_L}$$

- We can do the (stochastic) gradient descent with the value obtained.
- For $\partial \mathcal{L} / \partial \boldsymbol{W}_{L-1}$, we need to compute:

We can reuse this computed above.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{L-1}} = \overbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{h}_L}} \frac{\partial \mathbf{h}_L}{\partial \boldsymbol{W}_{L-1}}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_L} \underbrace{\frac{\partial \mathbf{h}_L}{\partial \mathbf{h}_{L-1}}} \frac{\partial \mathbf{h}_{L-1}}{\partial \boldsymbol{W}_{L-1}}$$

We need to compute partial derivative w.r.t input of layer L.

Table of Contents

Error Functions

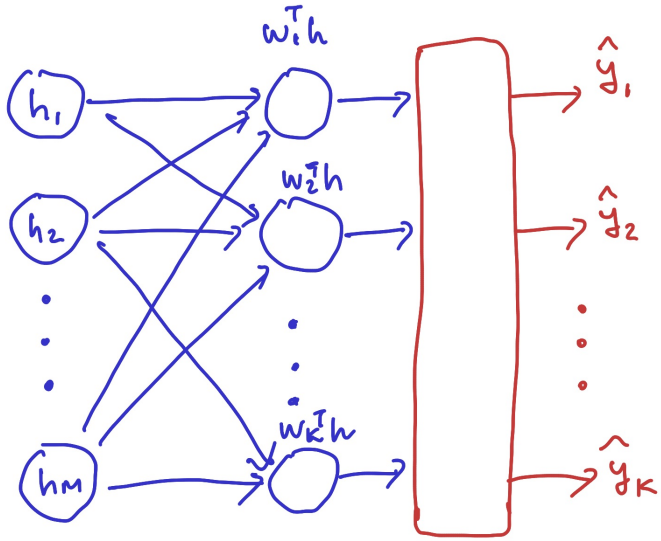
- Denote by $f(\mathbf{x}; \boldsymbol{\theta})$ a neural network parameterized by $\boldsymbol{\theta}$, which takes $\mathbf{x}$ as input.
- Denote by $\widehat{\mathbf{y}}_n$ the output of the neural network when input $\mathbf{x}_n$ is given. The corresponding target is $\mathbf{y}_n$.
- Squared error for regression:

$$\frac{1}{2N} \sum_{n=1}^{N} \|\mathbf{y}_n - \widehat{\mathbf{y}}_n\|^2$$

- Cross entropy error for classification: See next slides

Softmax Layer for Classification

For multi-class classification



$$\widehat{y}_k = \frac{\exp\left(\mathbf{w}_k^\top \mathbf{h}\right)}{\sum_{j=1}^{K} \exp\left(\mathbf{w}_j^\top \mathbf{h}\right)}$$

KL Divergence

Kullback-Leibler divergence is a metric to compare two distributions (when p generating samples):

$$\mathrm{KL}(p, q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Cross Entropy

Cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution q, rather than the "true" distribution p.

$$H(p; q) = \mathbb{E}_p[-\log q] = -\sum_x p(x) \log q(x)$$

Cross Entropy Error (for binary classes)

In the case of logistic regression, we have

$$p \in \{y, 1-y\}, \quad q \in \{\widehat{y}, 1-\widehat{y}\}$$

Then, the cross entropy error is calculated as

$$\mathcal{L} = \sum_{n=1}^{N} [-y_n \log \widehat{y}_n - (1-y_n) \log (1-\widehat{y}_n)]$$

Cross Entropy Error (for K classes)

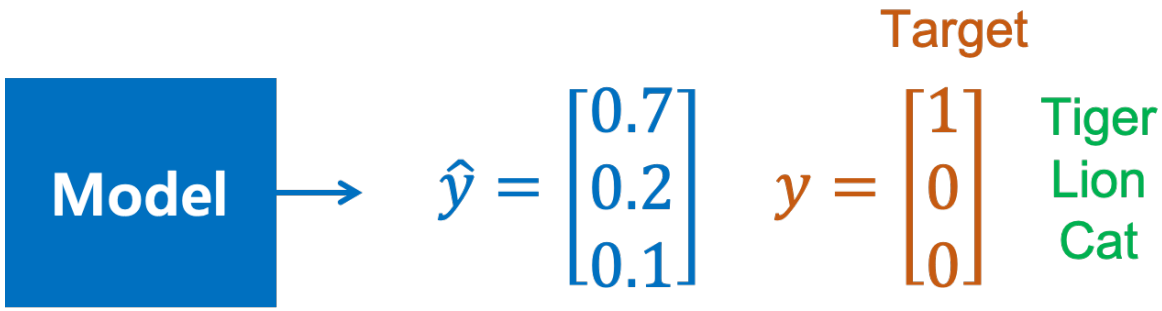
In the case of softmax regression, we have

$$p \in \{y_1, y_2, \ldots, y_K\}, \quad q \in \{\widehat{y}_1, \widehat{y}_2, \ldots, \widehat{y}_K\}.$$

Then, the cross entropy error is calculated as

$$\mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{K} [-y_{k,n} \log \widehat{y}_{k,n}]$$

Example of Cross Entropy Error (CE)



$$\begin{aligned} \text{CE} &= -1 \log 0.7 - 0 \log 0.2 - 0 \log 0.1 \\ &= -\log 0.7 \end{aligned}$$

More Loss Functions

Check the following link for more building blocks and loss functions ...

https://pytorch.org/docs/stable/nn.html

In `pytorch`

- L1Loss, MSELoss, CrossEntropyLoss, NLLLoss, KLDivLoss, HingeEmbeddingLoss, ...
- In addition, you can create your own loss function with the feature of `AutoDifferentiation`