COMPUTER VISION
MEI/1

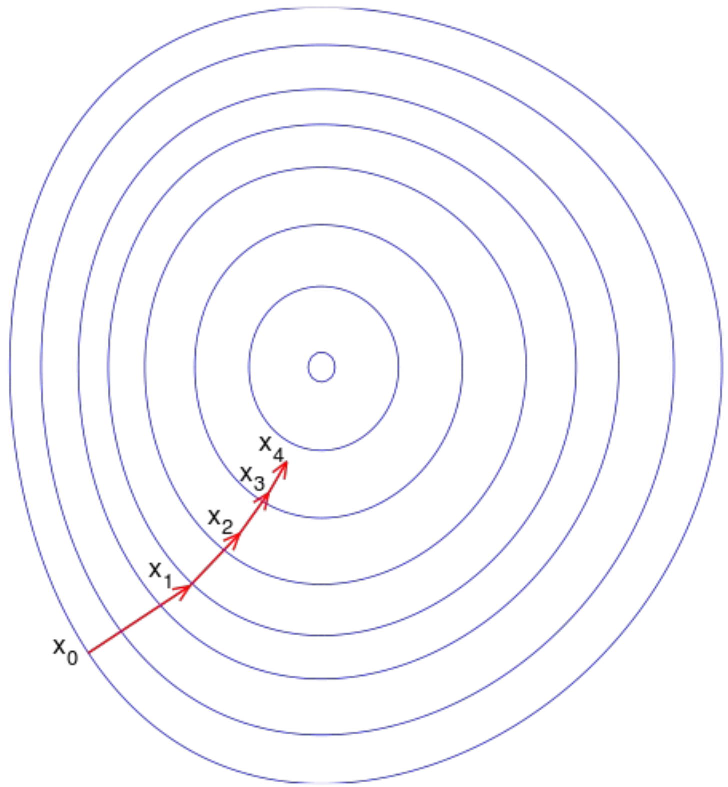
University of Beira Interior, Department of Informatics

Hugo Pedro Proença

hugomcp@di.ubi.pt, **2023/24**

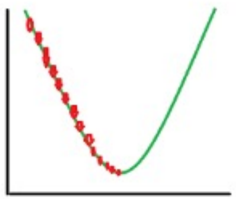
Neural Networks: Gradient Descent

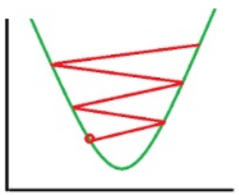
- Gradient descent is a **first-order iterative optimization** algorithm for **finding the minimum** of a function.
  - To find a (local?) minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient.
- It is based on the observation that if a multi-variable function F**(x)** is defined and **differentiable** in a neighborhood of a point $\mathbf{x}_{(t)}$ then F(**x**) decreases fastest if one goes from $\mathbf{x}_{(t)}$ in the direction of the negative gradient of F at $\mathbf{x}_{(t)}$: $-\nabla F(\mathbf{x}_{(t)})$



Neural Networks: Gradient Descent

- It follows that, if:
  - $\mathbf{x}_{(t+1)} = \mathbf{x}_{(t)} - \gamma \nabla F(\mathbf{x}_{(t)})$
- then, for γ small enough, then $F(\mathbf{x}_{(t)}) \geq F(\mathbf{x}_{(t+1)})$
- Based on this observation, in practice one starts with an initial guess $\mathbf{x}_{(0)}$ typically random and update iteratively $\mathbf{x}_{(t+1)}$ such that the sequence $\{\mathbf{x}_{(i)}\}$ converges to a minimum.
- The **learning rate** γ plays a major role in the final results of the optimization algorithm.
  - Too **small** values would take too long time to achieve a minimum;
  - Too **large** values might be even worse: might lead to **diverging** sequences.

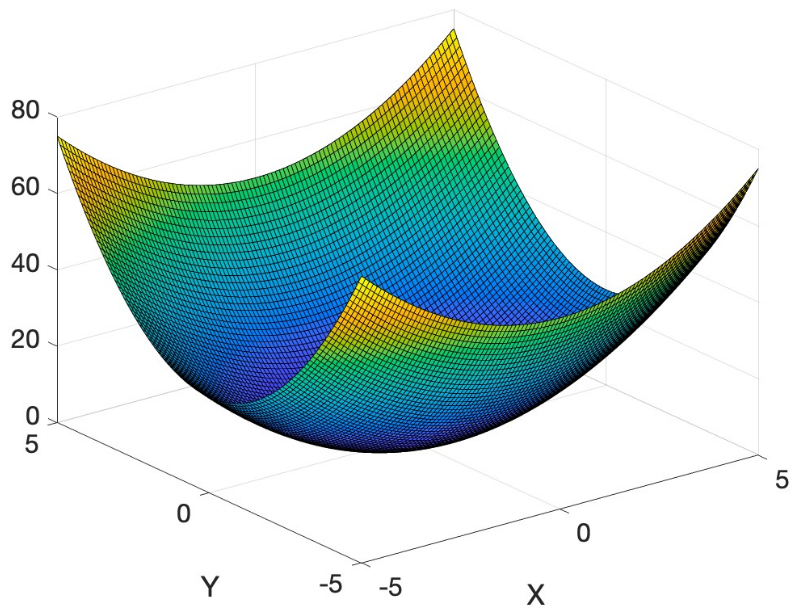




Gradient Descent: Unimodal Functions

- Consider the following 2D function:
  - $f(x, y) = x^2 + 2\,y^2$

1. Plot the function in the [-5.0, 5.0] x [-5.0, 5.0] interval



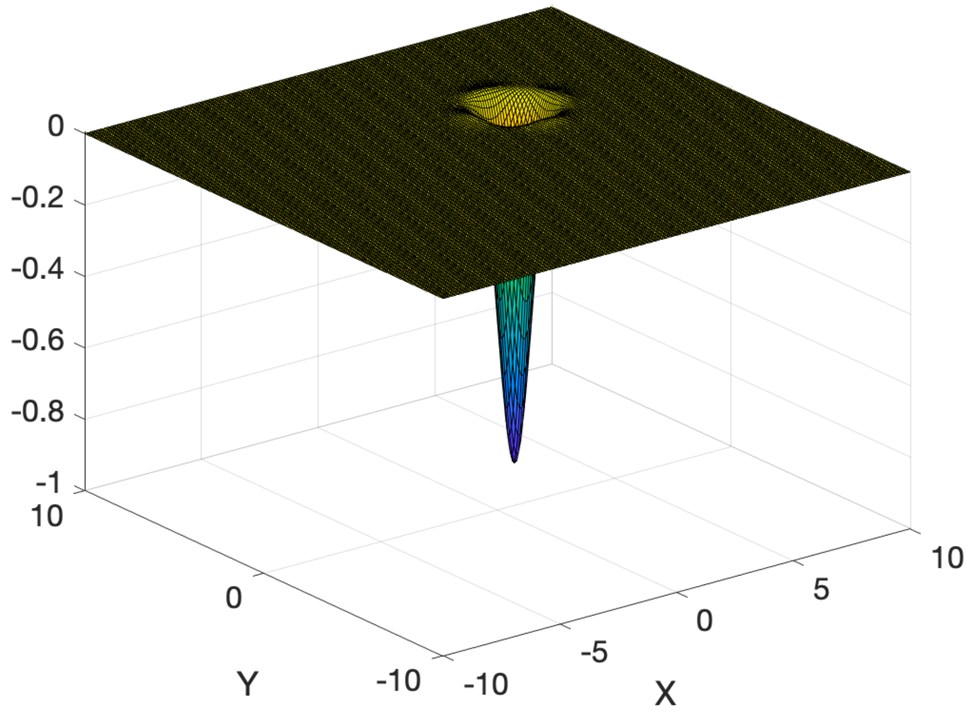
2. Create a Python script that, starting from a random point (x_0, y_0), use the gradient descent algorithm to find the function minimum.

3. Plot the path from (x_0, y_0) -> (x_t, y_t), using different learning rates

Gradient Descent: Unimodal Functions

- Easom's Function. Repeat the previous exercise, for the function below
  - $f(x, y) = -\cos(x) * \cos(y) * \exp(-((x-\pi)^2 + (y-\pi)^2))$

1. Plot the function in the [-10.0, 10.0] x [-10.0, 10.0] interval



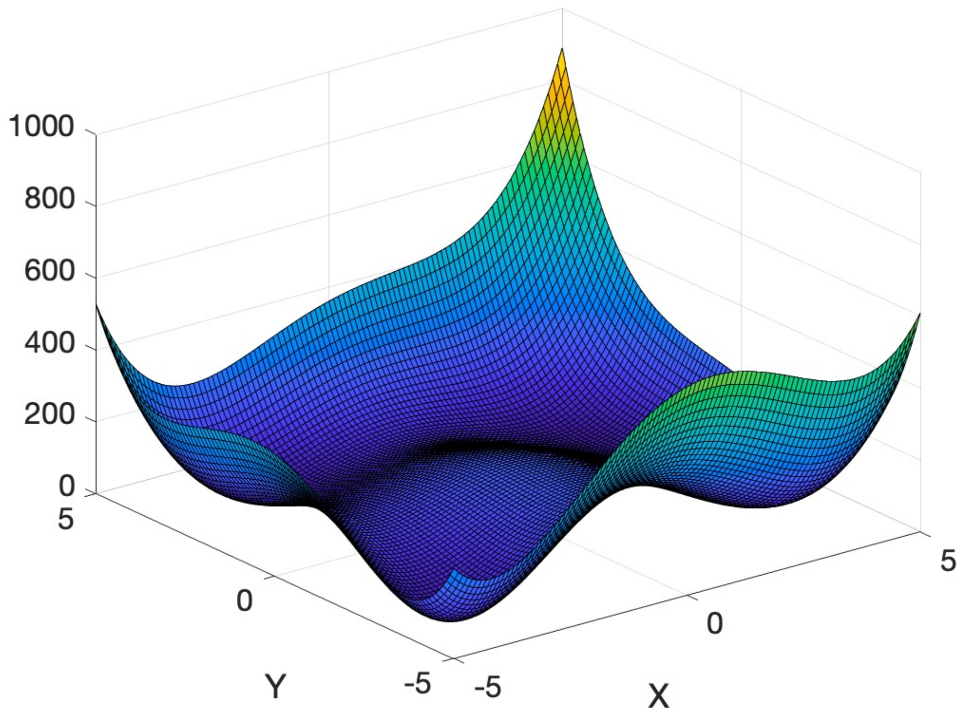
2. Create a Python script that, starting from a random point (x_0, y_0), use the gradient descent algorithm to find the function minimum.

3. Plot the path from (x_0, y_0) -> (x_t, y_t), using different learning rates

Gradient Descent: Multimodal Functions

- Himmelblau's Function. Repeat the previous exercise, for the function below
  - $f(x, y) = (x^2 + y \; - 11)2 + (y^2 + x \; - 7)^2$

1. Plot the function in the [-5.0, 5.0] x [-5.0, 5.0] interval



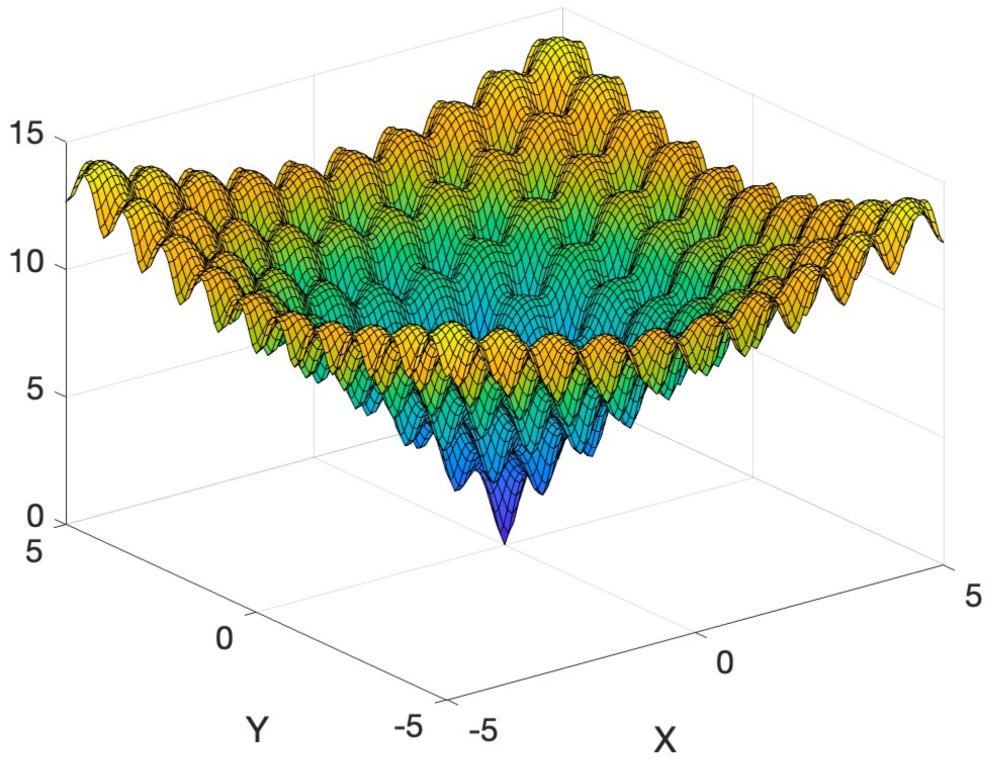
2. Create a Python script that, starting from a random point (x_0, y_0), use the gradient descent algorithm to find the function minimum.

3. Plot the path from (x_0, y_0) -> (x_t, y_t), using different learning rates

Gradient Descent: Multimodal Functions

- Ackley's Function. Repeat the previous exercise, for the function below
  - $f(x,y) = -20\ \exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right) + \exp(1) + 20$

1. Plot the function in the [-5.0, 5.0] x [-5.0, 5.0] interval



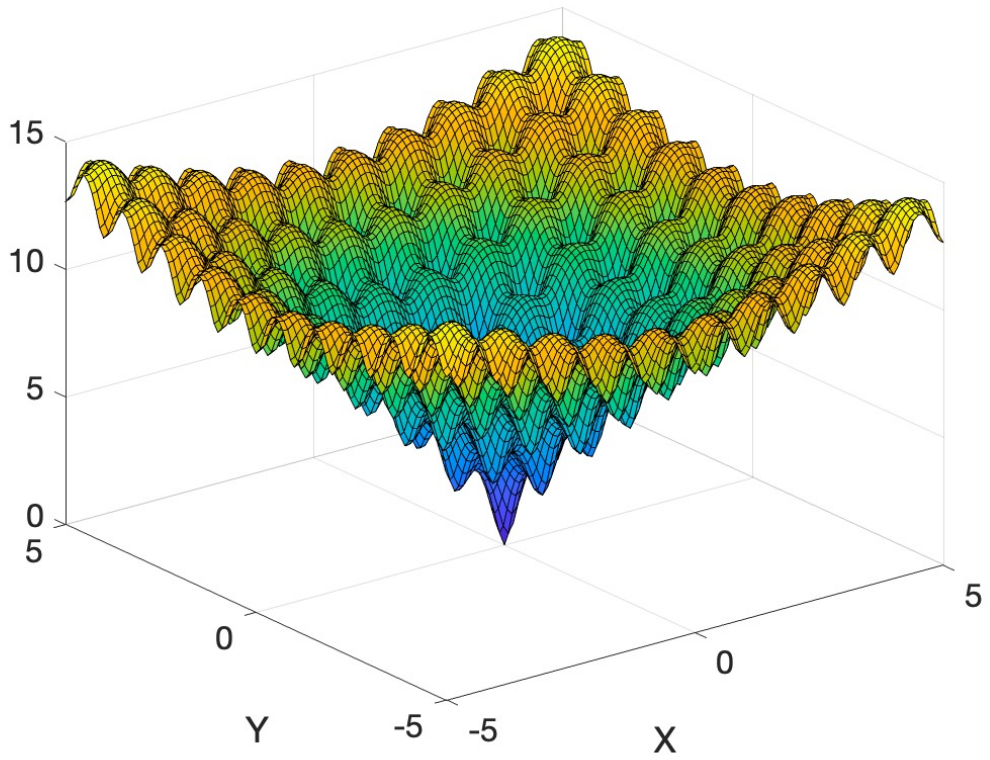
2. Create a Python script that, starting from a random point (x_0, y_0), use the gradient descent algorithm to find the function minimum.

3. Plot the path from $(x_0, y_0) \rightarrow (x_t, y_t)$, using different learning rates

Gradient Descent: Multimodal Functions

- Ackley's Function. Repeat the previous exercise, for the function below
  - $f(x, y) = -20\ \exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right) + \exp(1) + 20$

1. Plot the function in the [-5.0, 5.0] x [-5.0, 5.0] interval

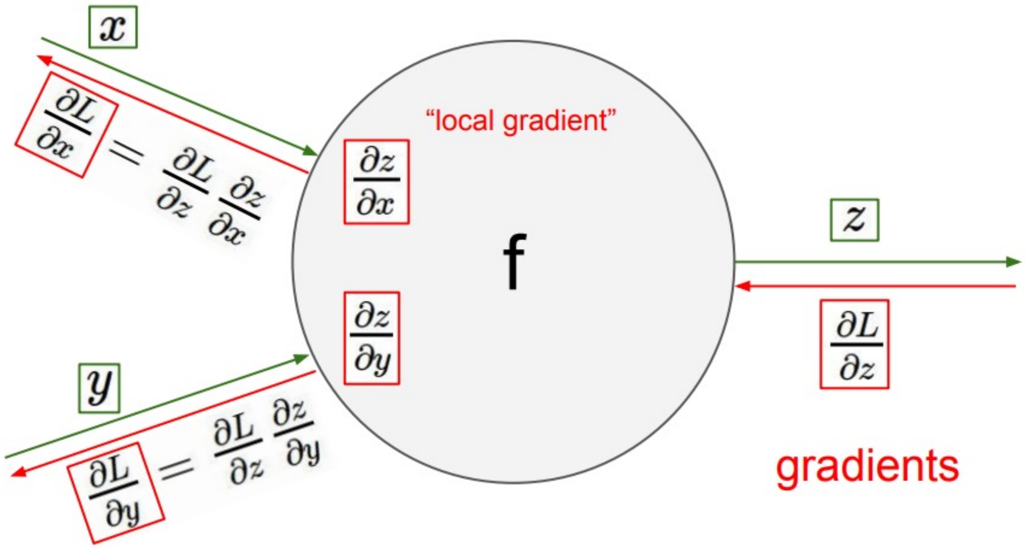


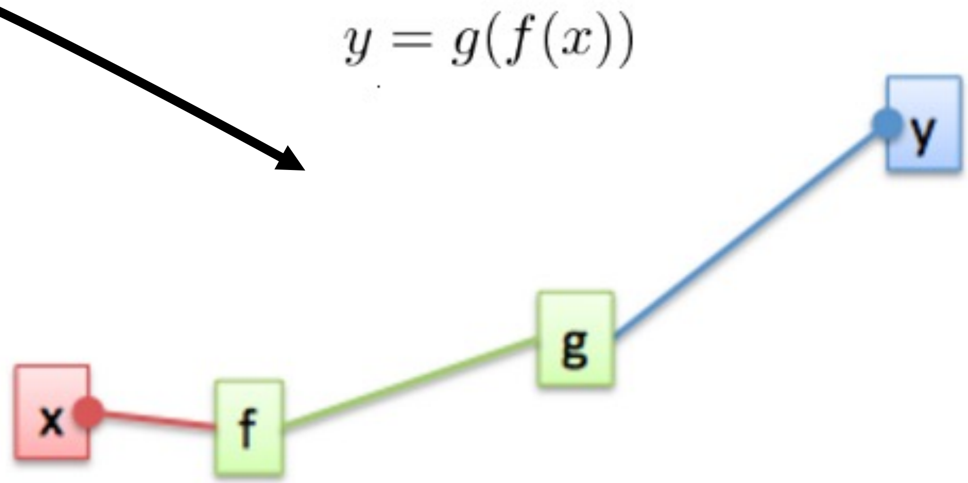
2. Create a Python script that, starting from a random point (x_0, y_0), use the gradient descent algorithm to find the function minimum.

3. Plot the path from (x_0, y_0) -> (x_t, y_t), using different learning rates

Backpropagation

- "**Backpropagation**" is the short name for "backward propagation of errors";
- Algorithm for supervised learning of multi-layer artificial neural networks based in gradient descent;
- The key concept is the **<u>chain rule</u>**:
  - **δg/δx = δg/δf . δf/δx**
- Calculates the gradient of the error function with respect to the neural network's weights;
- It is a **generalization** of the delta rule for perceptrons to multilayer feed-forward neural networks.



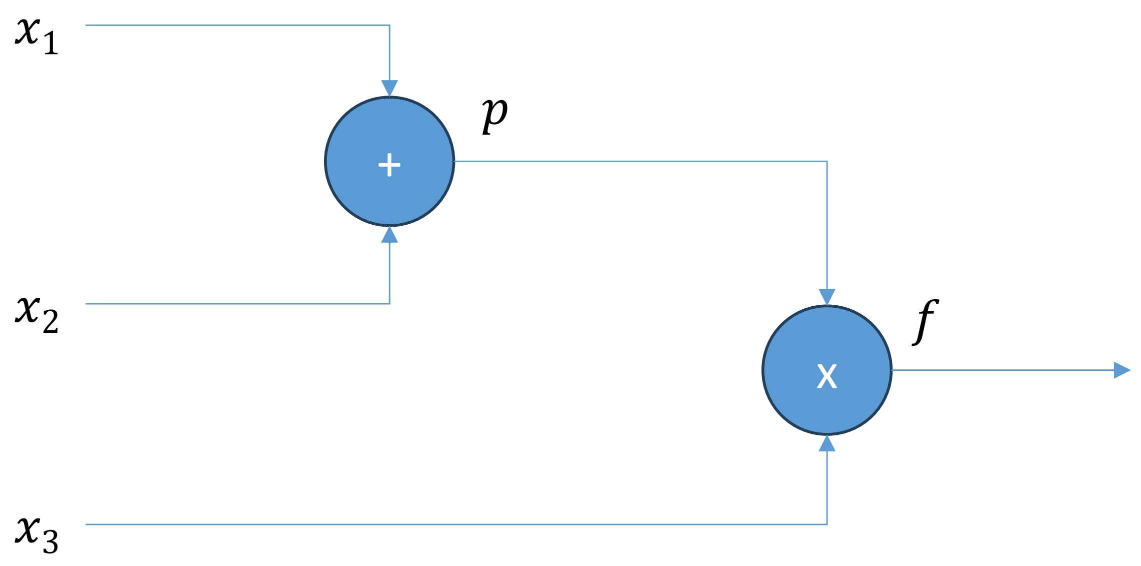


Backpropagation Example 1

- Consider the following simple function:

$$f(x_1, x_2, x_3) = (x_1 + x_2)x_3$$

- …and the corresponding network:

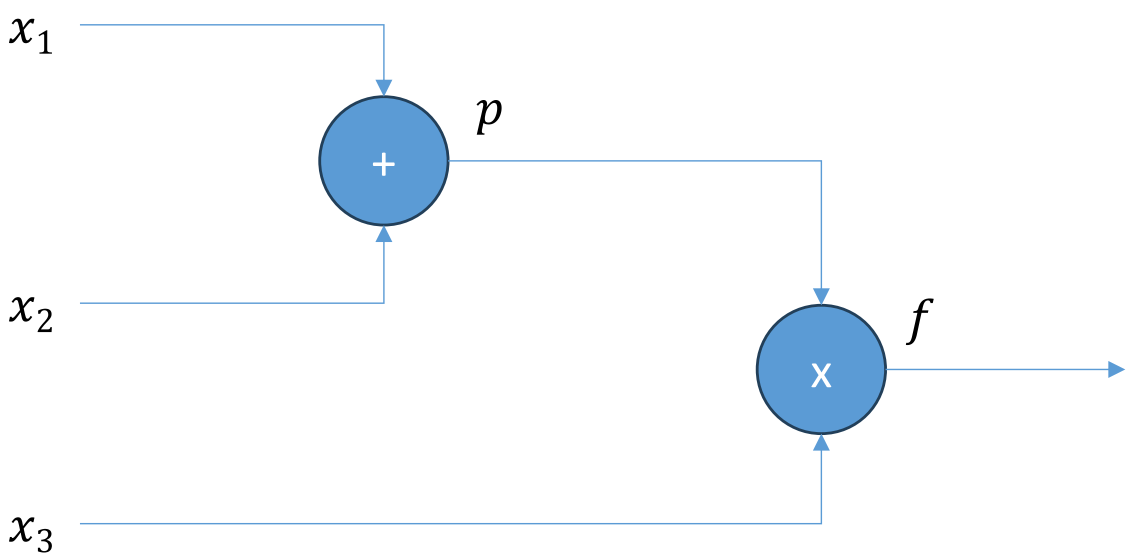


Backpropagation Example 1

- $p = x_1 + x_2; \quad \frac{\partial p}{\partial x_1} = 1; \quad \frac{\partial p}{\partial x_2} = 1$
- $f = px_3; \quad \frac{\partial f}{\partial p} = x_3; \quad \frac{\partial f}{\partial x_3} = p$

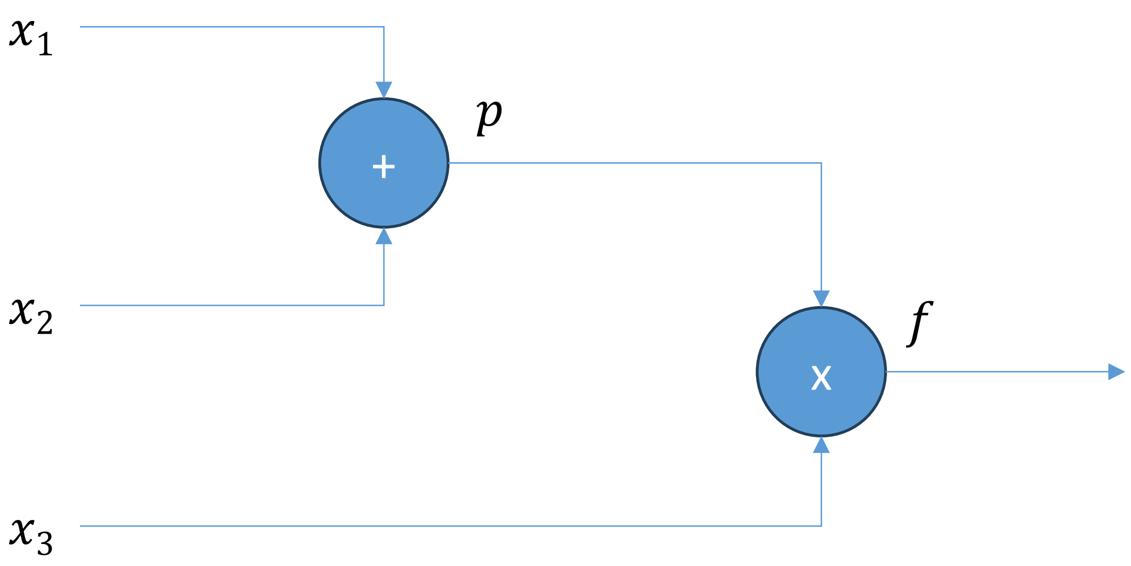
We need:

$\frac{\partial f}{\partial x_1}; \frac{\partial f}{\partial x_2}; \frac{\partial f}{\partial x_3}$



Backpropagation Example 1

- According to the chain rule, $\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial p}\frac{\partial p}{\partial x_1}$
- $\frac{\partial f}{\partial x_1} = x_3 1$ (*“if x_1 changes Δ, then f will change $x_3\Delta$”*)
- $\frac{\partial f}{\partial x_2} = x_3 1$

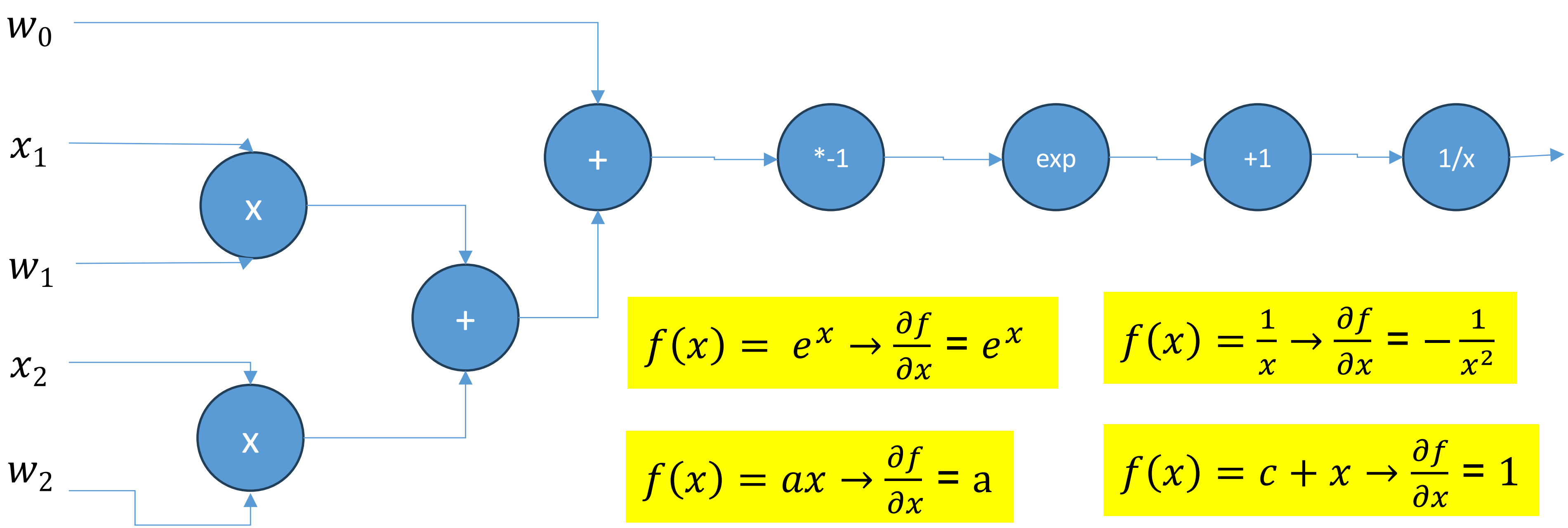


Backpropagation Example 2

- A slight more complex example regarding the well known “logistic regression” classifier:

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

vectors



$$f(x) = e^x \rightarrow \frac{\partial f}{\partial x} = e^x$$

$$f(x) = \frac{1}{x} \rightarrow \frac{\partial f}{\partial x} = -\frac{1}{x^2}$$

$$f(x) = ax \rightarrow \frac{\partial f}{\partial x} = a$$

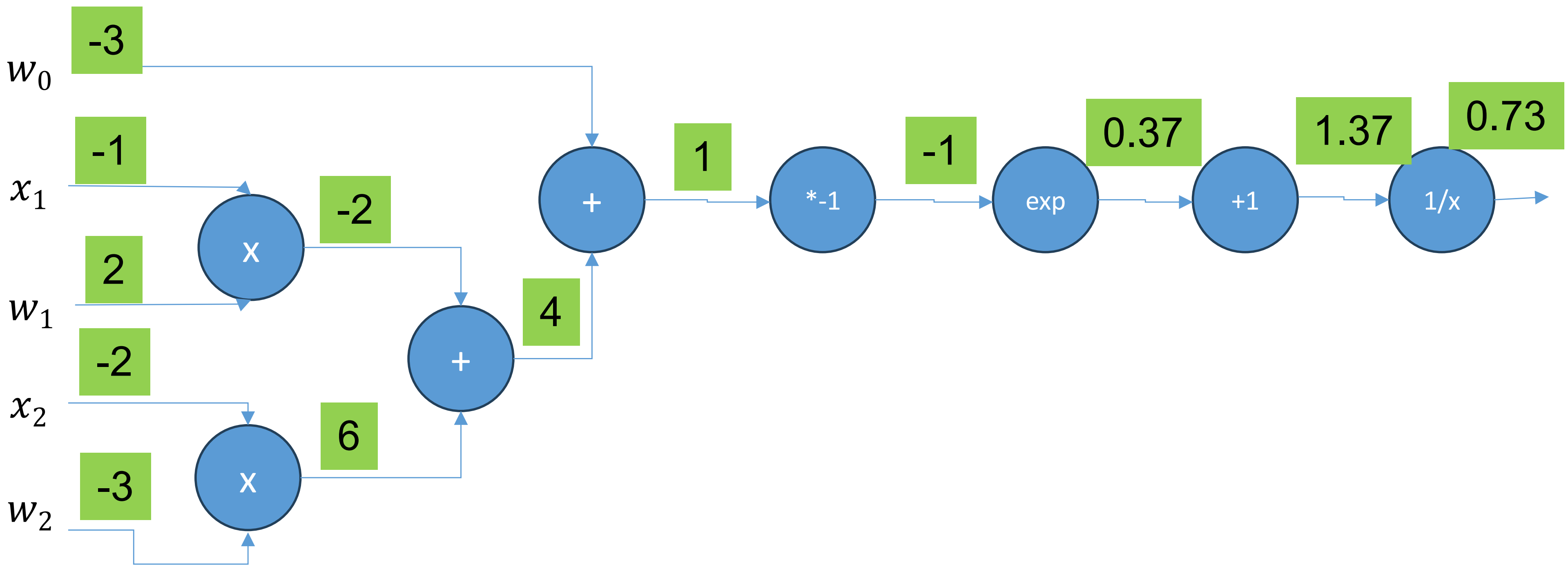
$$f(x) = c + x \rightarrow \frac{\partial f}{\partial x} = 1$$

Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]$; $\boldsymbol{w}$= $[-3,\ \ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + \ e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Forward Pass:**

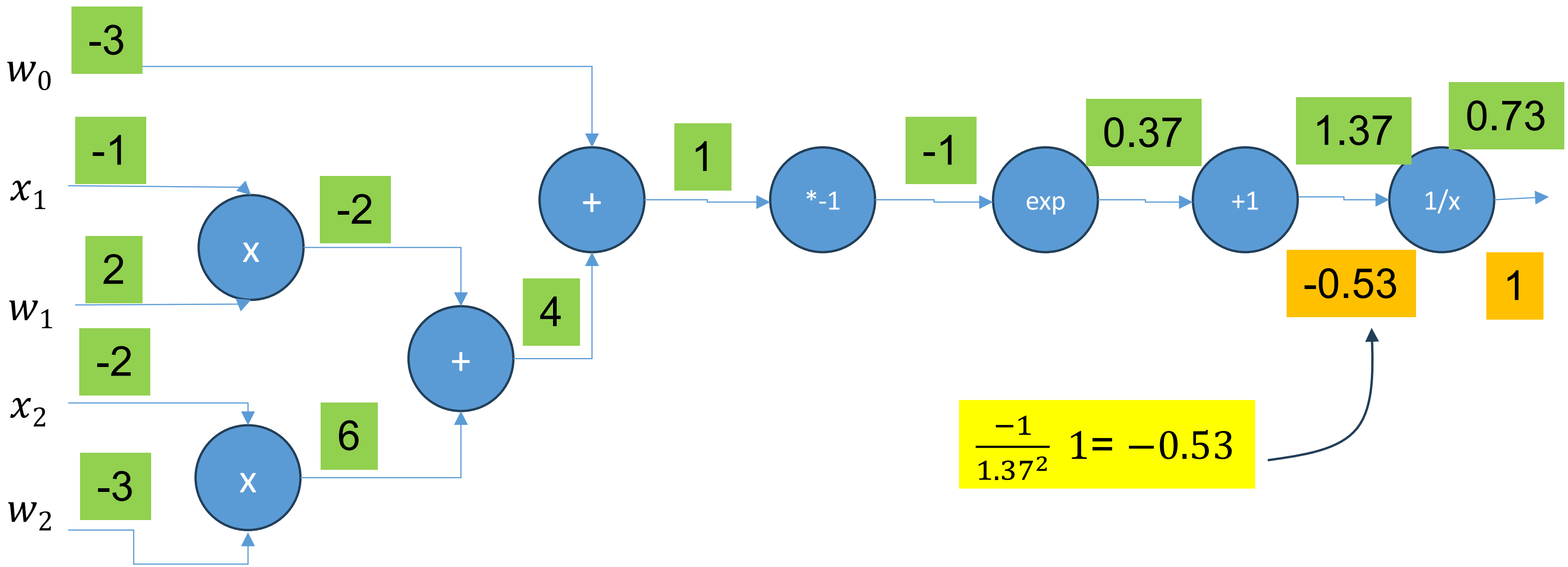


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]$; $\boldsymbol{w} = [-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 +\ e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**

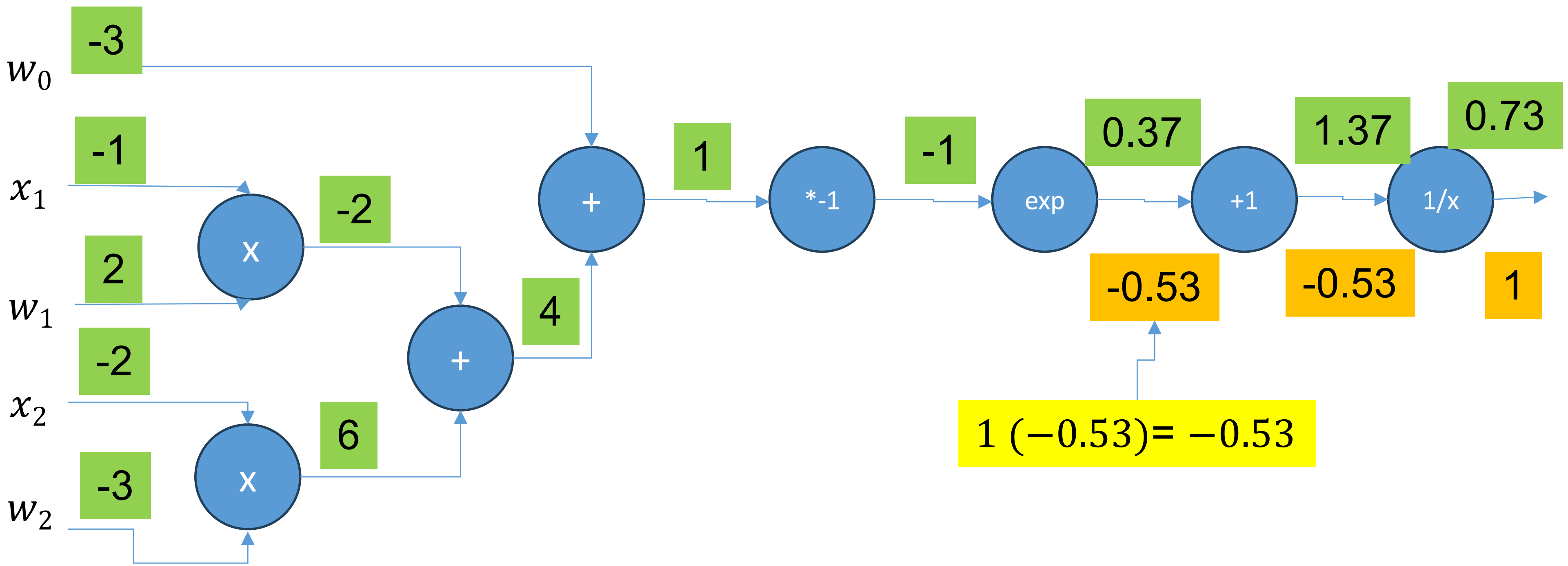


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]; \boldsymbol{w} = [-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 +\ e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**

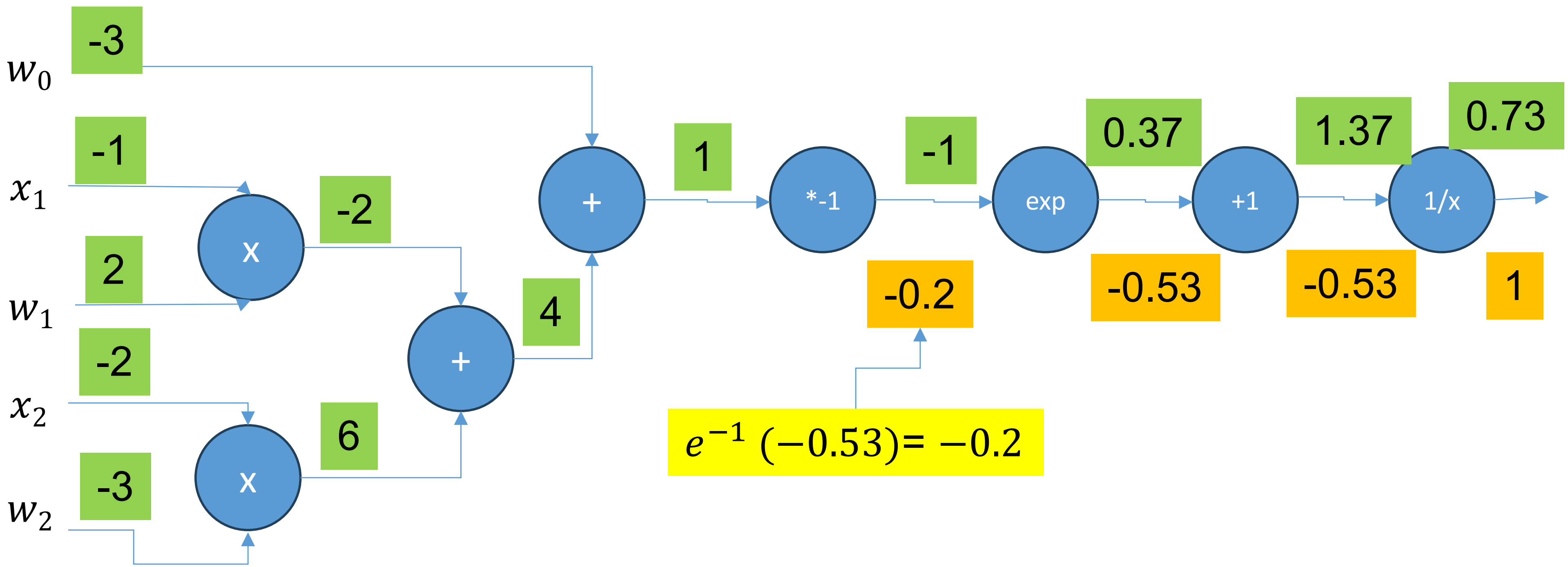


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]$; $\boldsymbol{w}$= $[-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + \ e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**

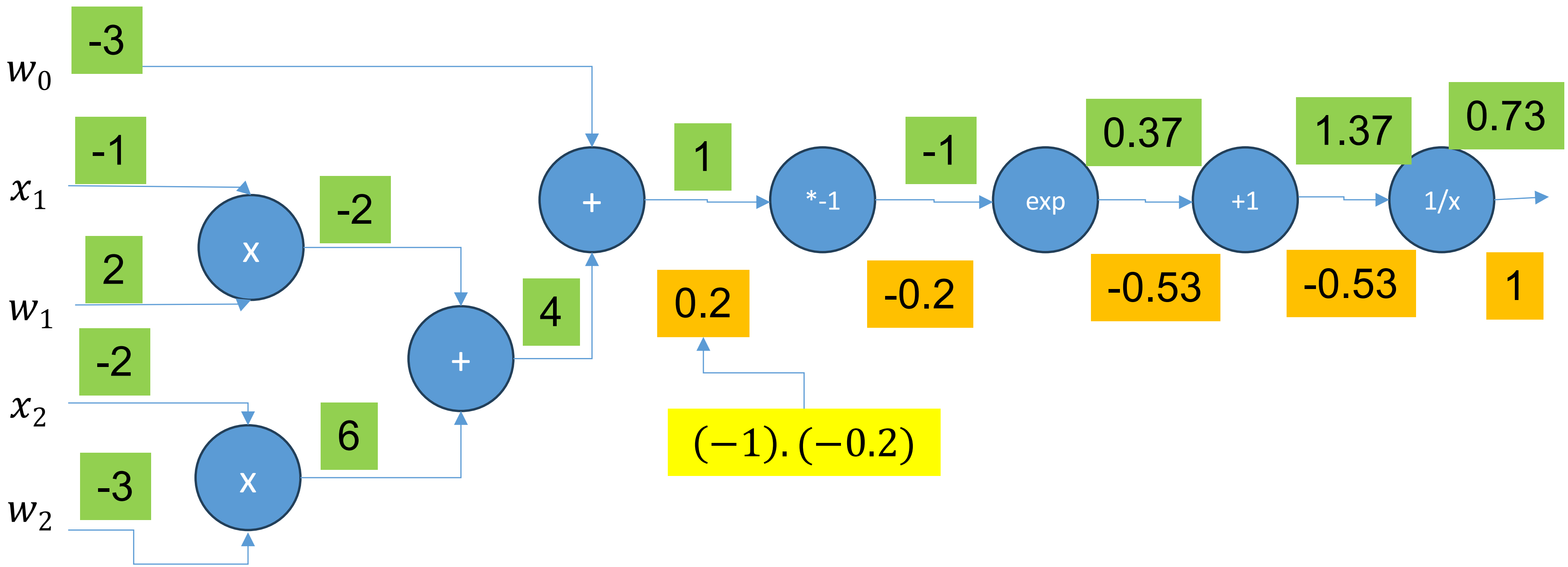


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]; \boldsymbol{w} = [-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 +\ e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**

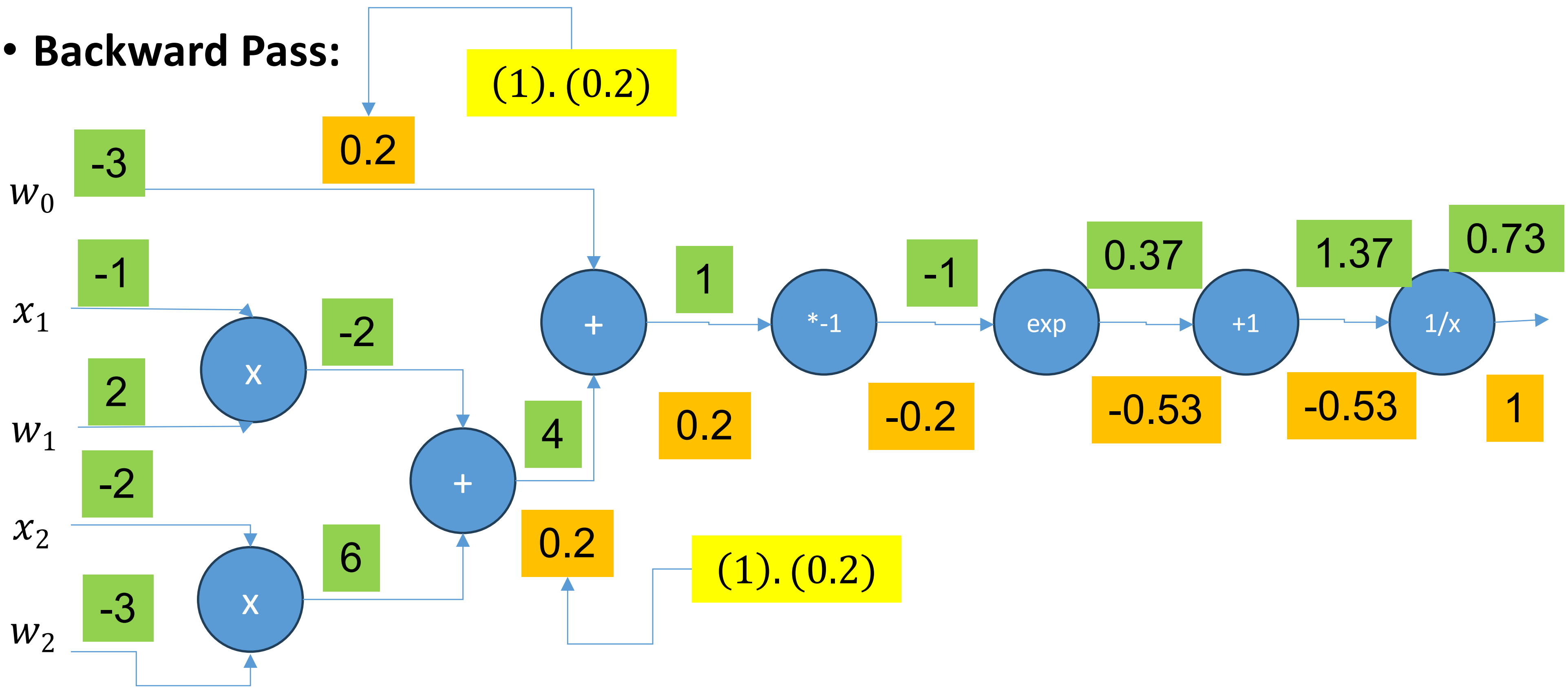


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]$; $\boldsymbol{w} = [-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**

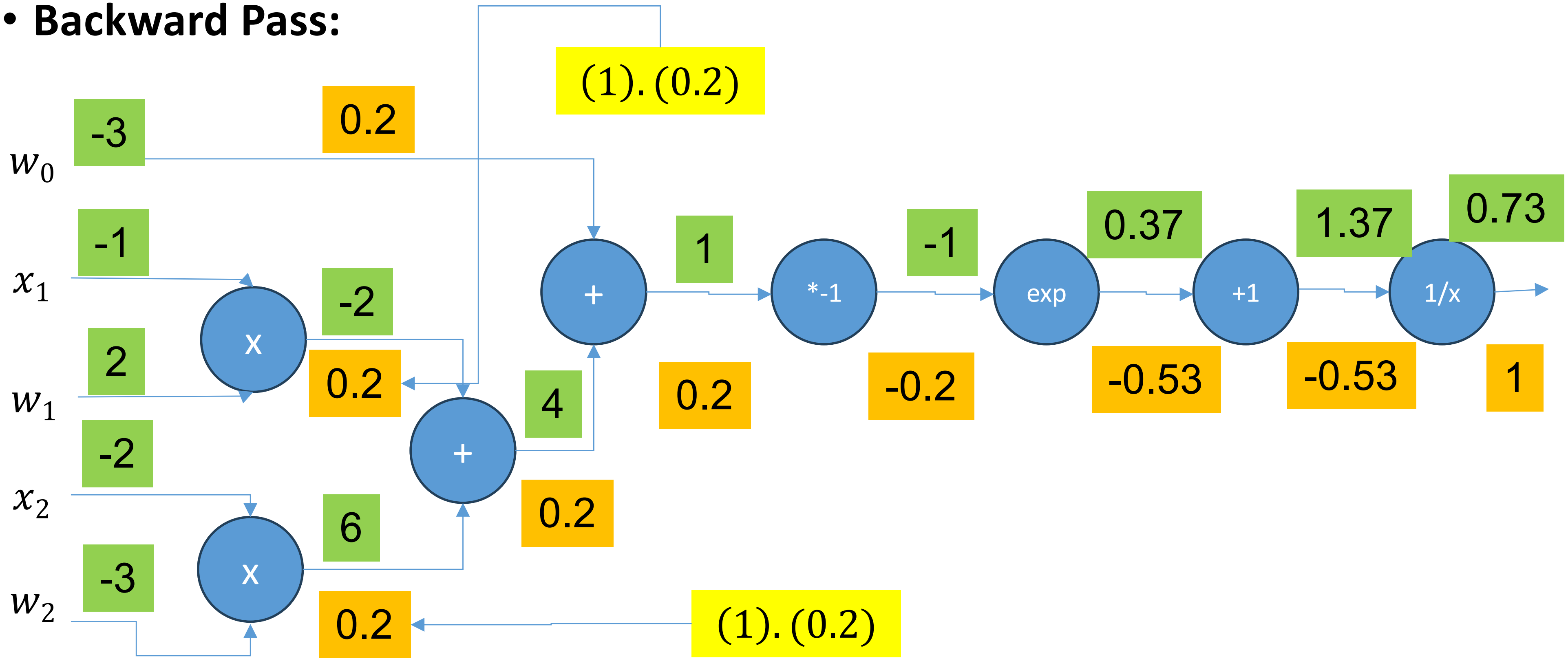


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]$; $\boldsymbol{w} = [-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 +\ e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**

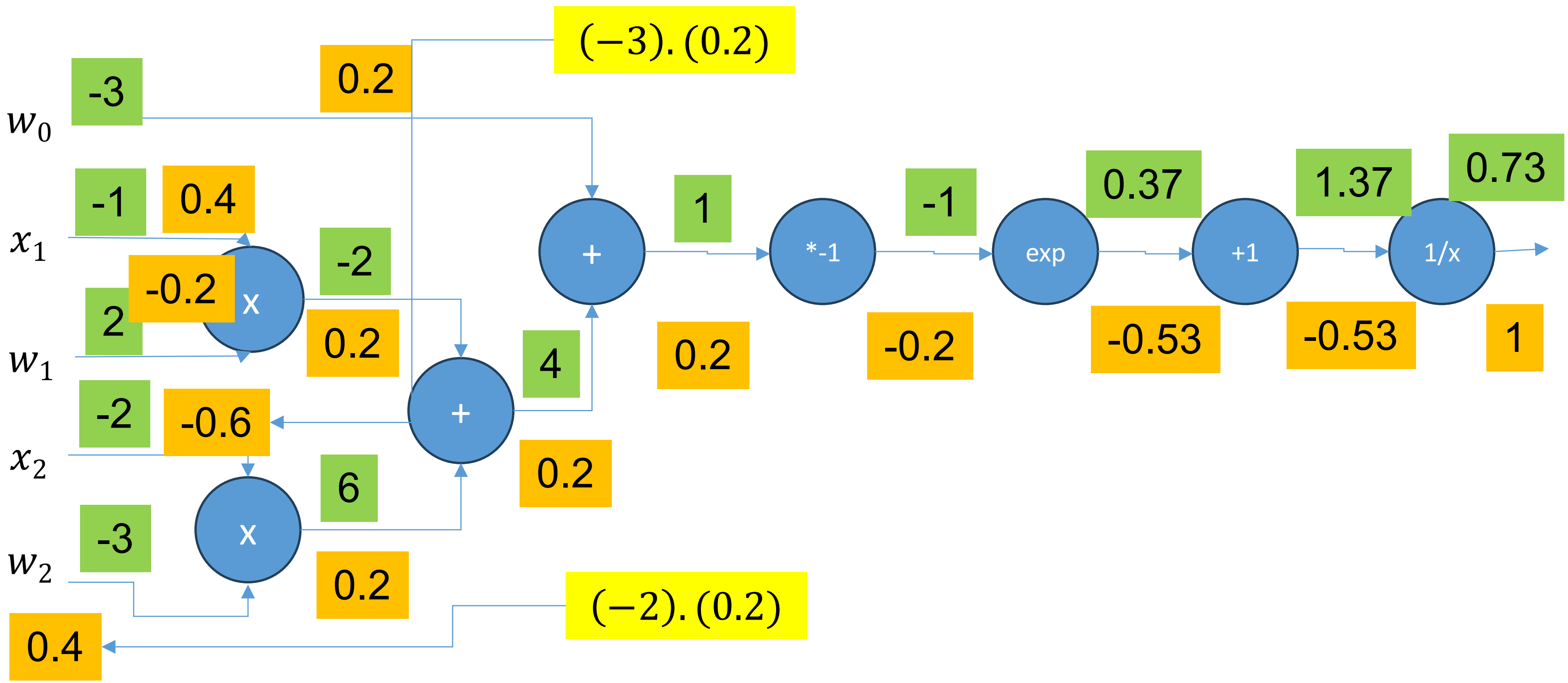


Backpropagation Example 2

- Consider the following input $\boldsymbol{x} = [-1, -2]; \boldsymbol{w}= [-3,\ 2, -3]$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

- **Backward Pass:**



Backpropagation

- In case of any branches, gradientes should be summed up.