

Deep Learning & Neural Networks

Dr. Shahira Ahmed

DS710

Lecture 8





Faculty of Engineering
جامعة ميدوcean
Midocean University

Deep Learning & Neural Networks DS710

Lecture 8
**Deep Learning Optimization: Techniques
and Parameter Tuning**

Dr. Shahira Ahmed

www.midocean.ae



LEARNING OUTCOMES

- ◉ Understand the role of optimizers and loss functions in deep learning.
- ◉ Learn various optimization techniques used in deep learning.
- ◉ Gain knowledge of parameter tuning strategies for optimizers and loss functions.
- ◉ Be able to apply appropriate optimization techniques and tune parameters effectively in your deep learning projects



LECTURE OUTLINE

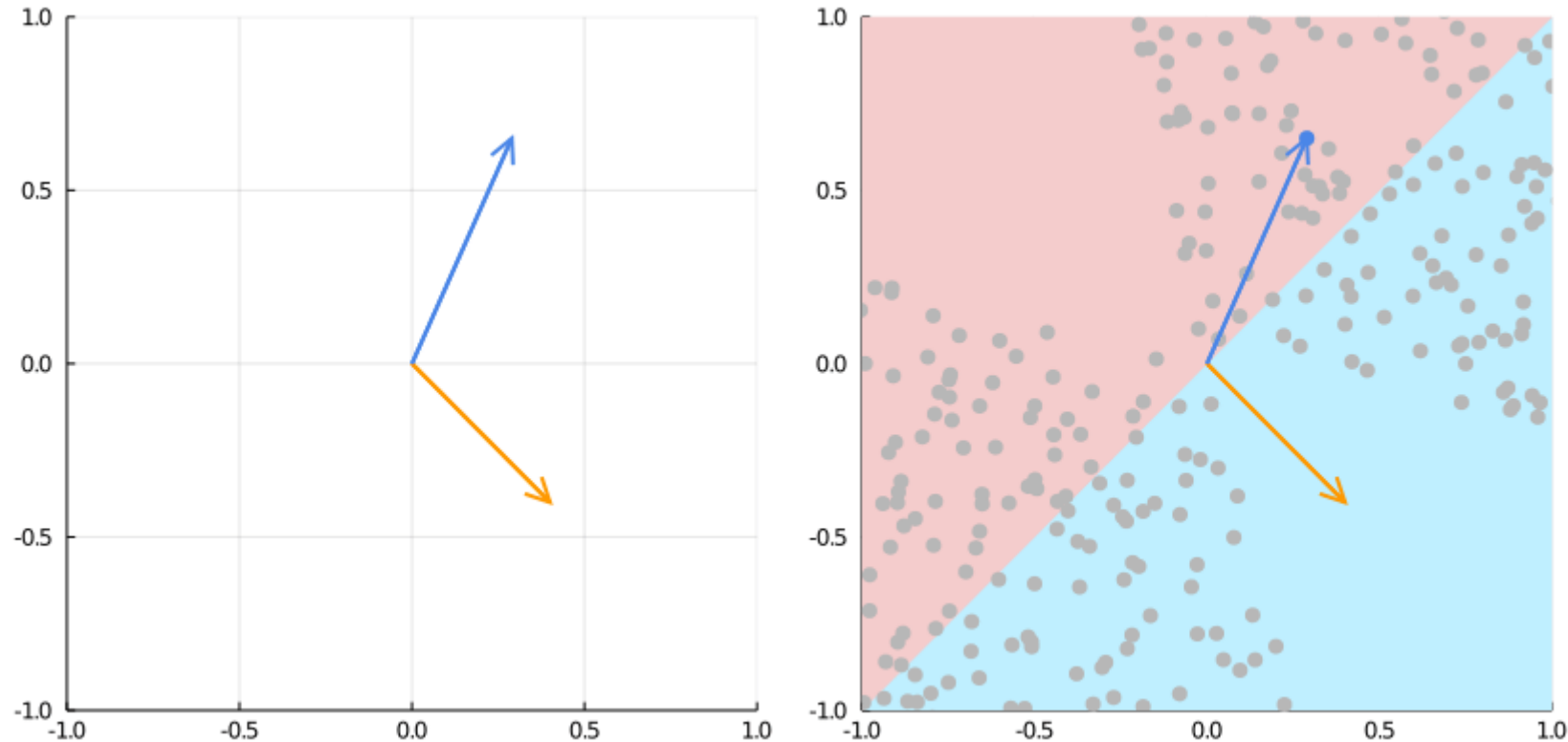
- ◉ Optimizers in Deep Learning
 - ◉ Gradient descent and its variants
 - ◉ Momentum-based optimizers
 - ◉ Adaptive optimizers
 - ◉ Pros and cons of each optimizer
- ◉ Hyperparameter tuning

The Perceptron Learning Algorithm

- In the Perceptron algorithm, we computed the **dot product** between a **single weight vector and the input vector** to make a prediction.
- If the prediction was wrong, we simply **added/subtracted** the input vector to/from the weight vector.
- The goal was to increase (by adding) or decrease (by subtracting) the dot product between the weight vector and the input vector.
- we wanted to improve our predictions more carefully, we scaled the input vector by a **small learning rate** before using it to update the weight vector.

<https://medium.com/the-feynman-journal/what-makes-backpropagation-so-elegant-657f3afb8d>

The Perceptron Learning Algorithm



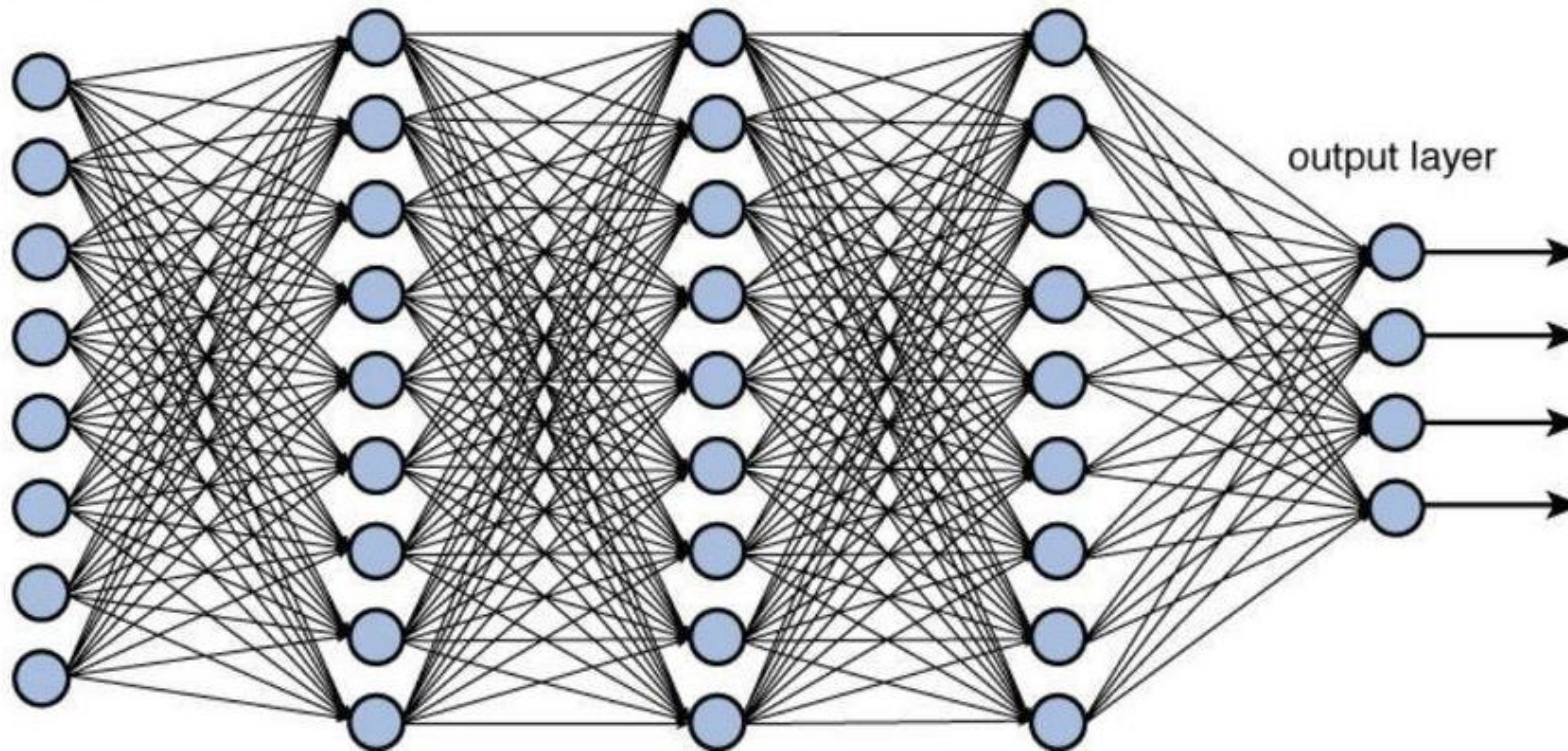
<https://medium.com/the-feynman-journal/what-makes-backpropagation-so-elegant-657f3afb8bd>

- MLP doesn't just have a single weight vector — it has **many weight matrices**, each one made up of many weight vectors.
- How we scale the input vector before using it to update each weight vector.
- Goal: update the weight vectors in a way such that the combination of all updates results in the MLP becoming more accurate in making predictions.

<https://medium.com/the-feynman-journal/what-makes-backpropagation-so-elegant-657f3afb8bd>

- In deep learning, optimizers are algorithms that **adjust the model's parameters** during training to **minimize a loss function**. They enable neural networks to learn from data by iteratively **updating weights and biases**.
- Optimizer algorithms are optimization method that helps improve a deep learning **model's performance**, accuracy and speed training of the deep learning model.

- Choosing the **right weights** for the model as a deep learning model generally consists of **millions of parameters**.



- **Backpropagation**, short for "backward propagation of errors"
- The goal of backpropagation is to compute gradient and then we can use any gradient based algorithm to **adjust the weights and biases** of a neural network in order to **minimize the difference between the predicted outputs and the actual outputs** for a given set of training data.

The backpropagation algorithm can be summarized in the following steps:

(1) **Forward** propagation: The input data is fed forward through the network, and the activations of each neuron are computed layer by layer until the output is generated. This step is also known as the forward pass.

(2) Error computation: The error or cost function is calculated by comparing the network's output with the desired output for the given input.

(3) Backward propagation: The gradients of the weights and biases are computed by traversing the network in reverse order, starting from the output layer and moving towards the input layer. The gradients are computed using the chain rule, which allows the error to be apportioned to each neuron in the network.

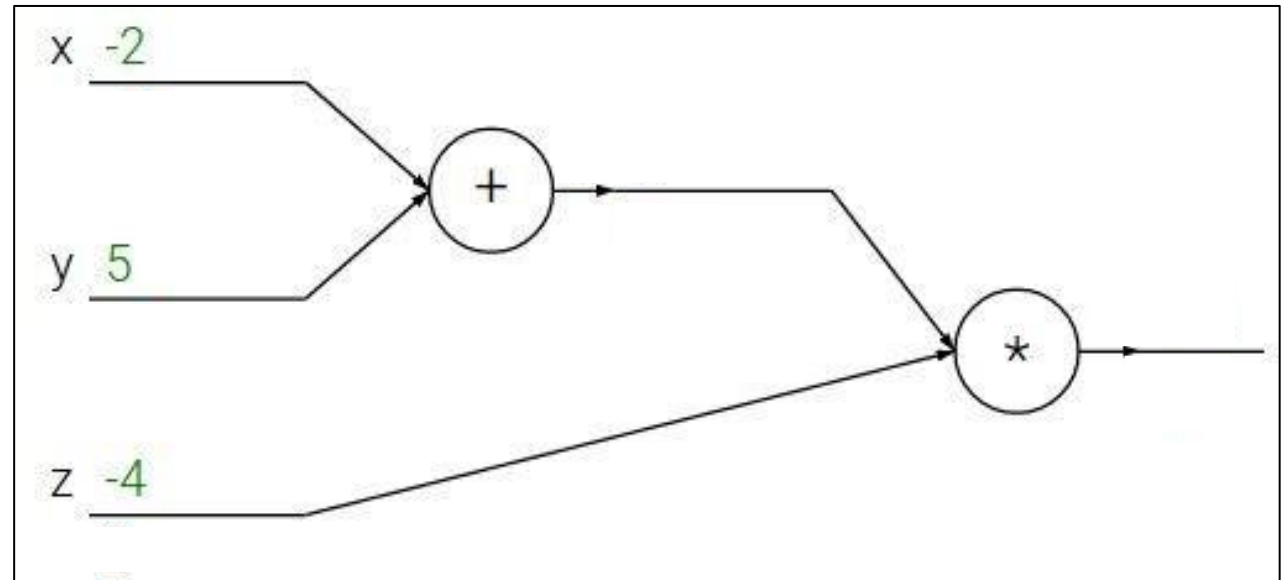
(4) **Weight and bias updates:** The weights and biases of the network are adjusted using an **optimization algorithm, such as gradient descent**, to minimize the error. The magnitude of the updates is determined by the gradients computed in the previous step.

Iteration: Steps 1 to 4 are repeated for multiple iterations or epochs until the network's performance reaches a satisfactory level or a stopping criterion is met.

Backpropagation: a simple example

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

e.g. $x = -2, y = 5, z = -4$

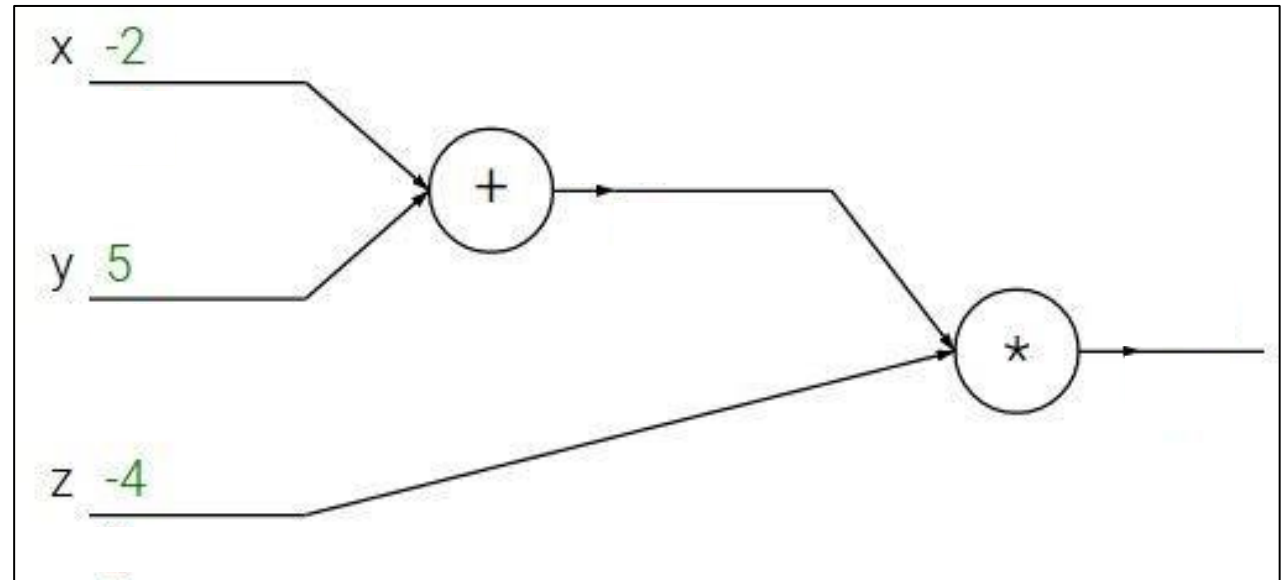


Backpropagation: a simple example

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

e.g. $x = -2$, $y = 5$, $z = -4$

Forward pass



Backpropagation: a simple example

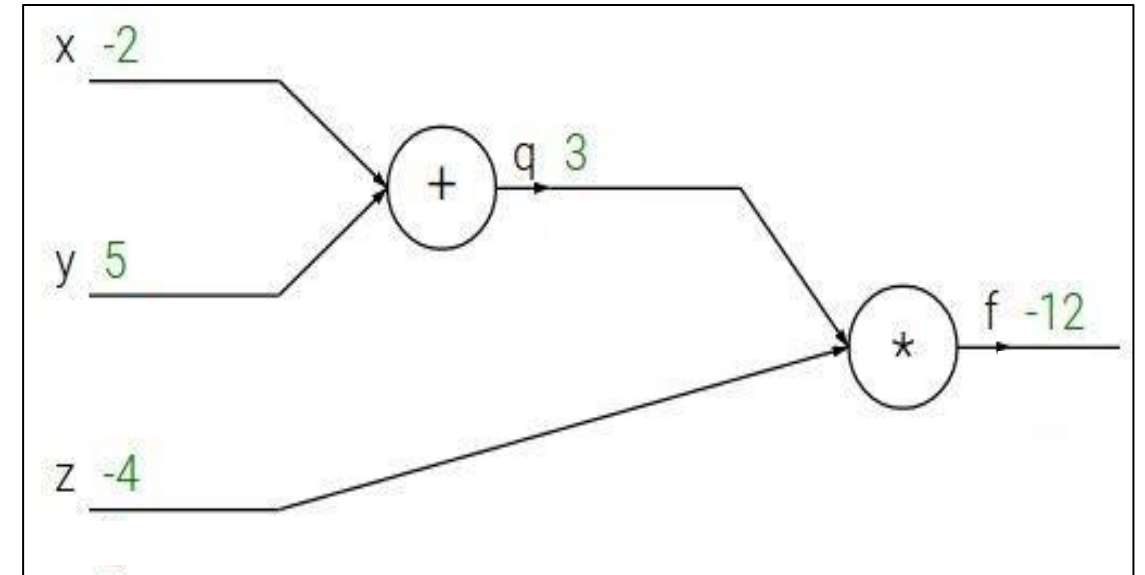
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

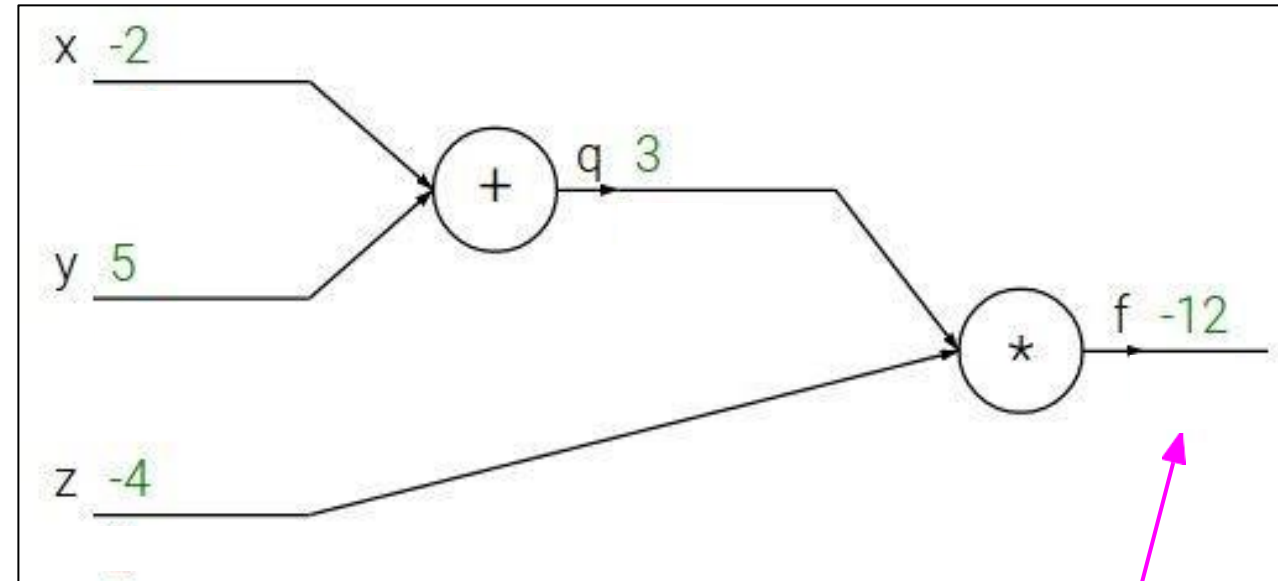
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backward pass

$$\frac{\partial f}{\partial f}$$

Backpropagation: a simple example

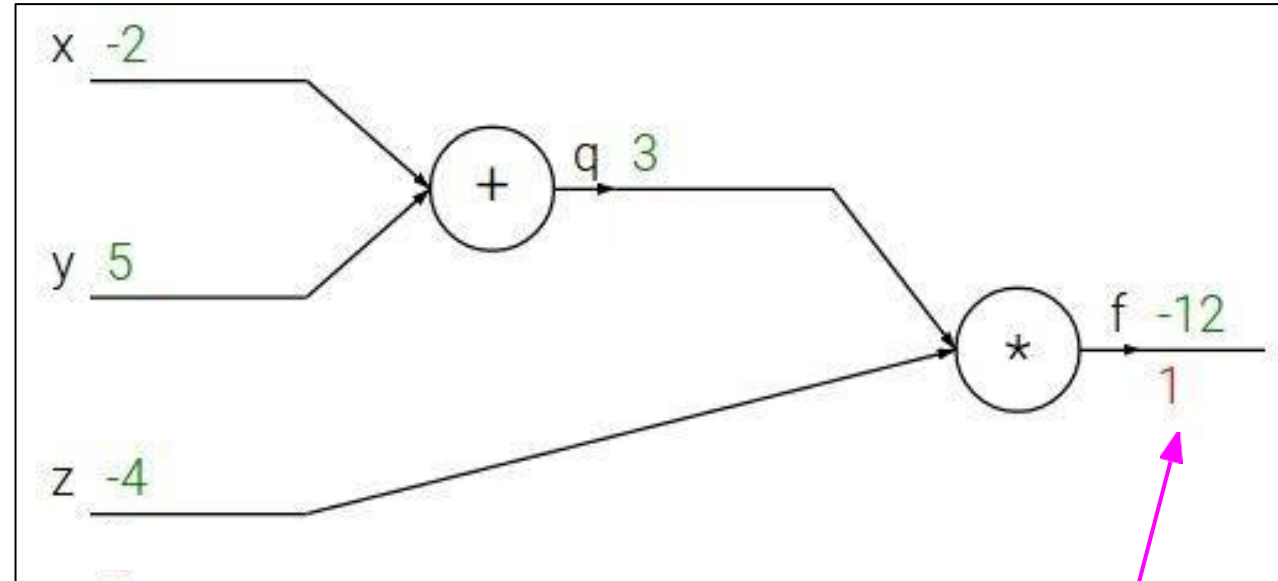
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial f}$$

Backpropagation: a simple example

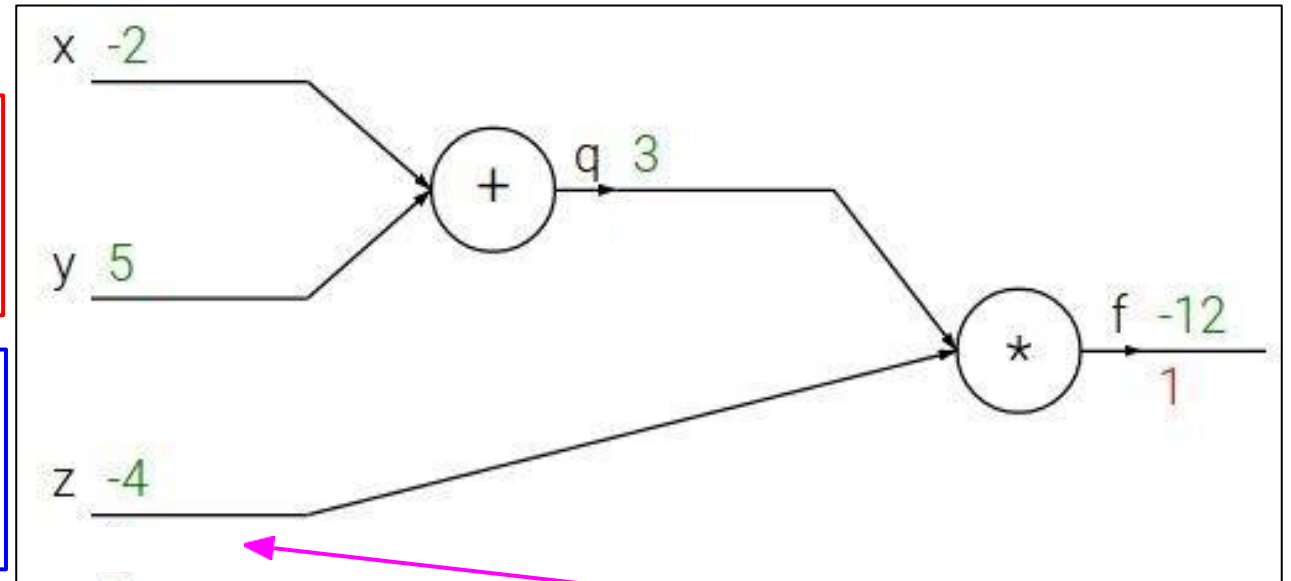
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



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

Backpropagation: a simple example

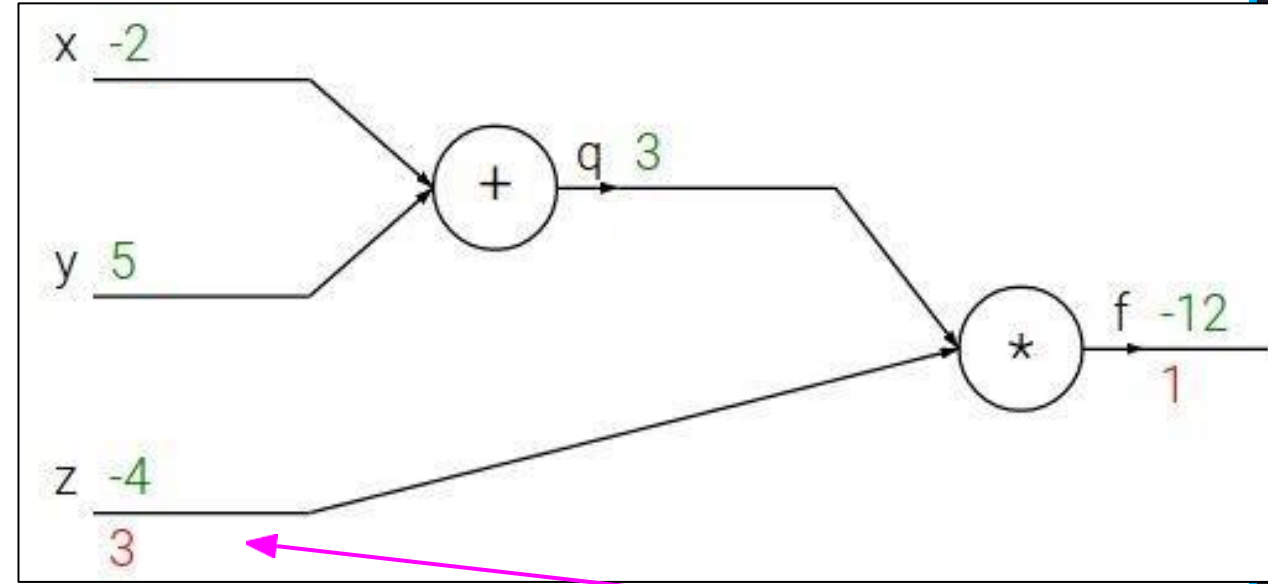
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



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

Backpropagation: a simple example

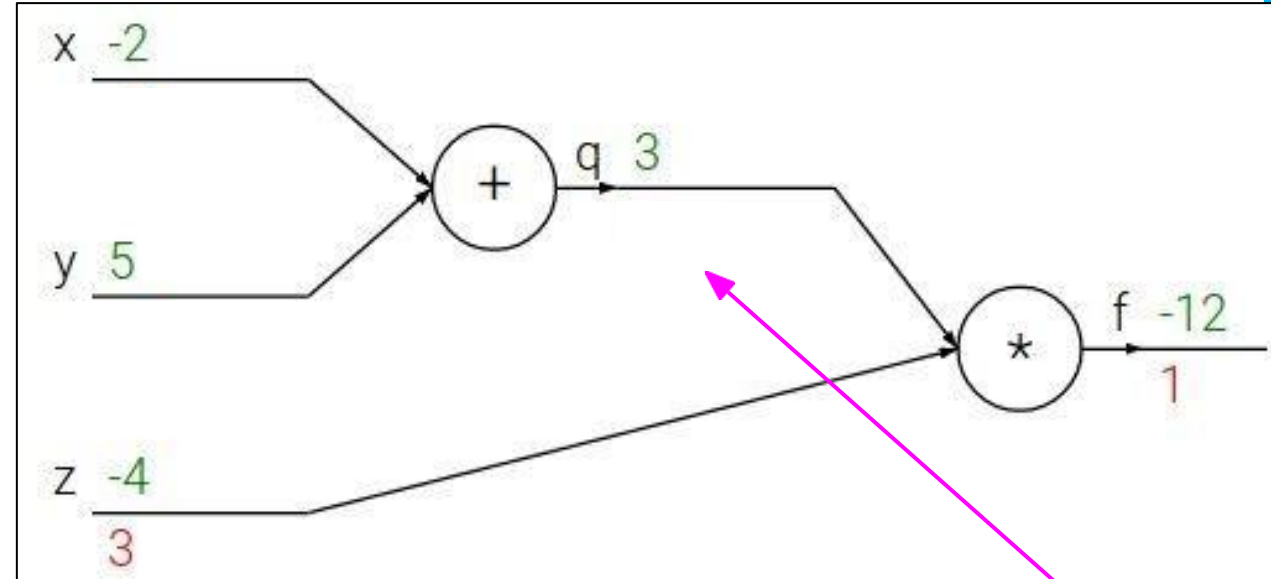
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: a simple example

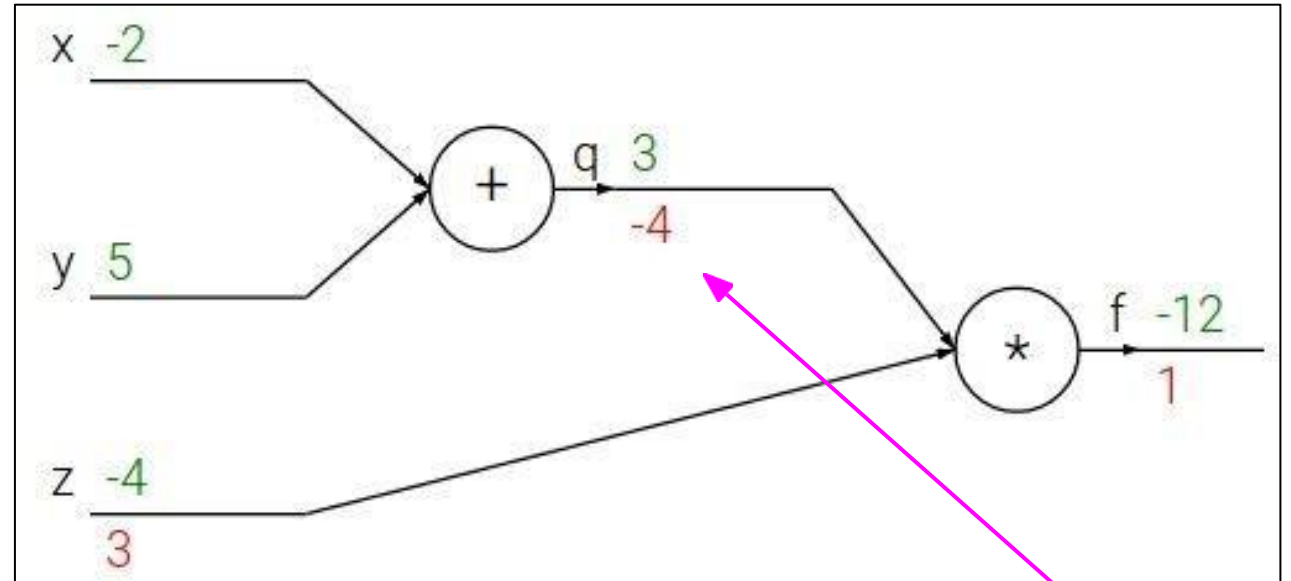
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: a simple example

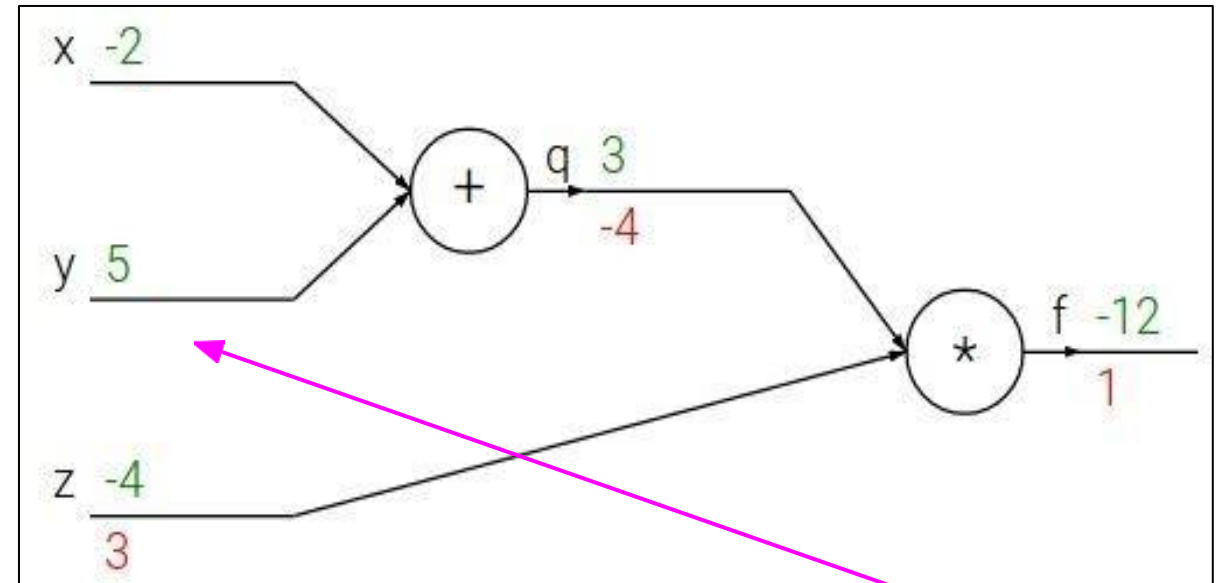
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Backpropagation: a simple example

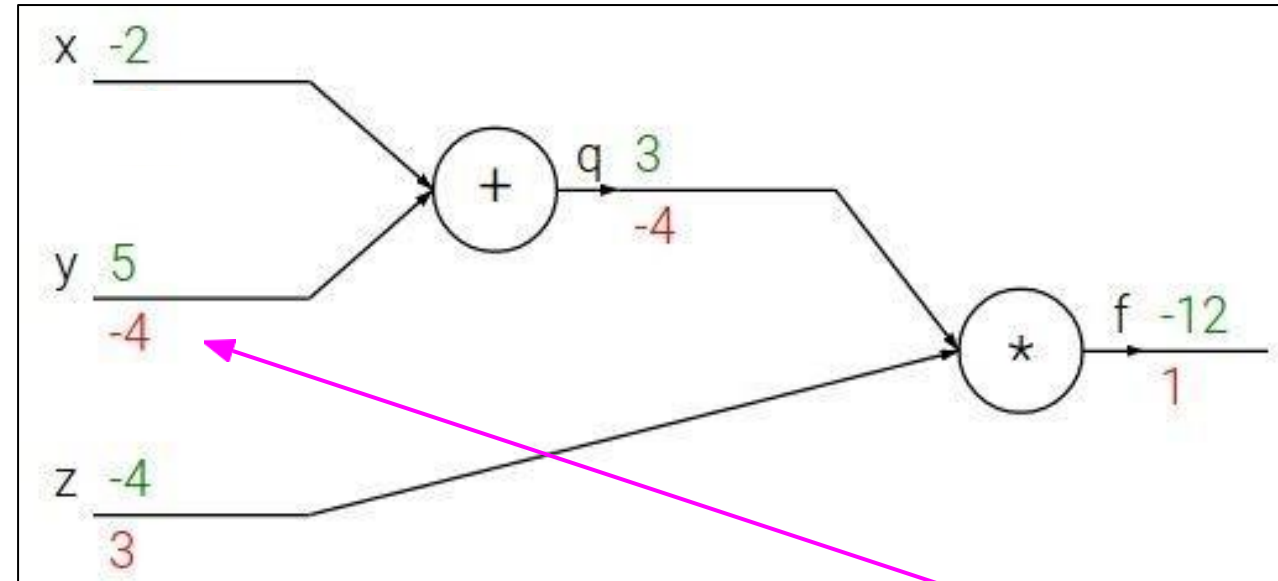
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

$$\frac{\partial f}{\partial y}$$

Backpropagation: a simple example

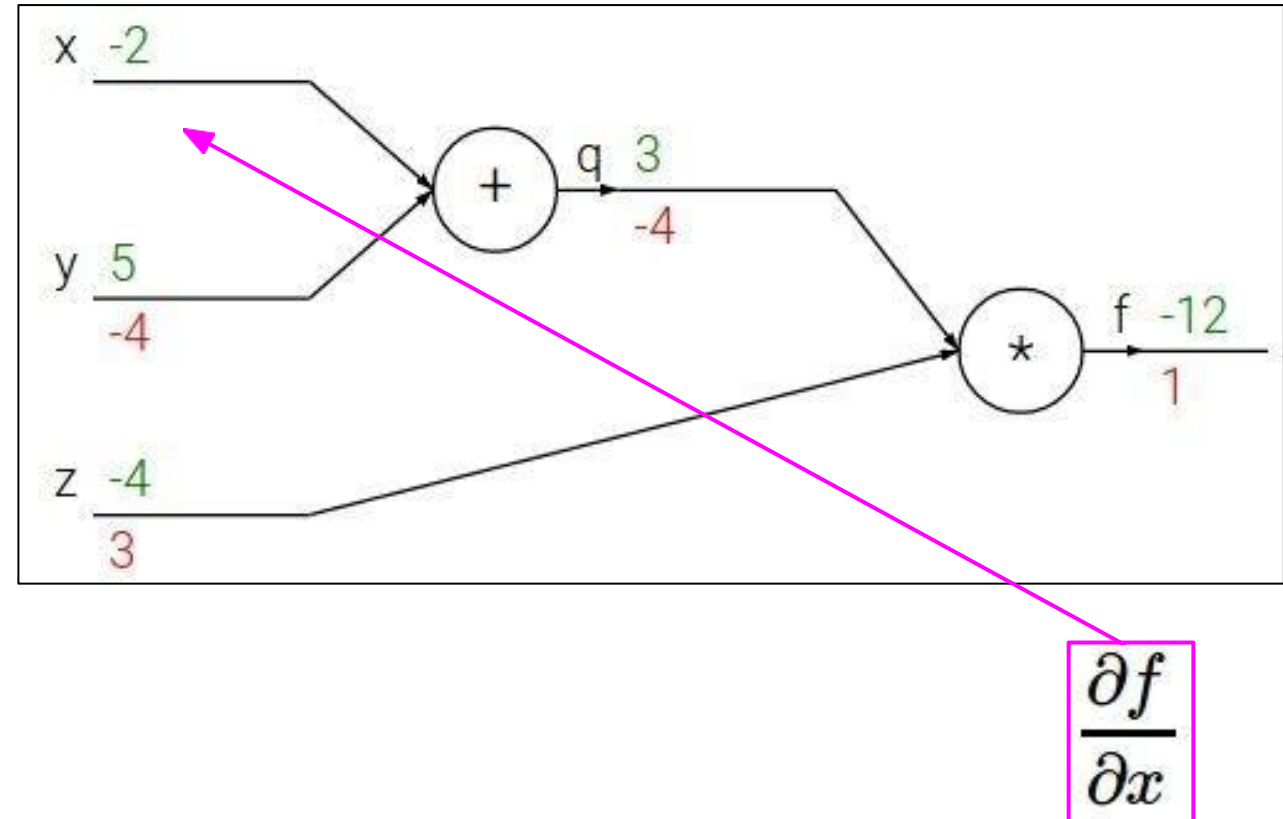
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

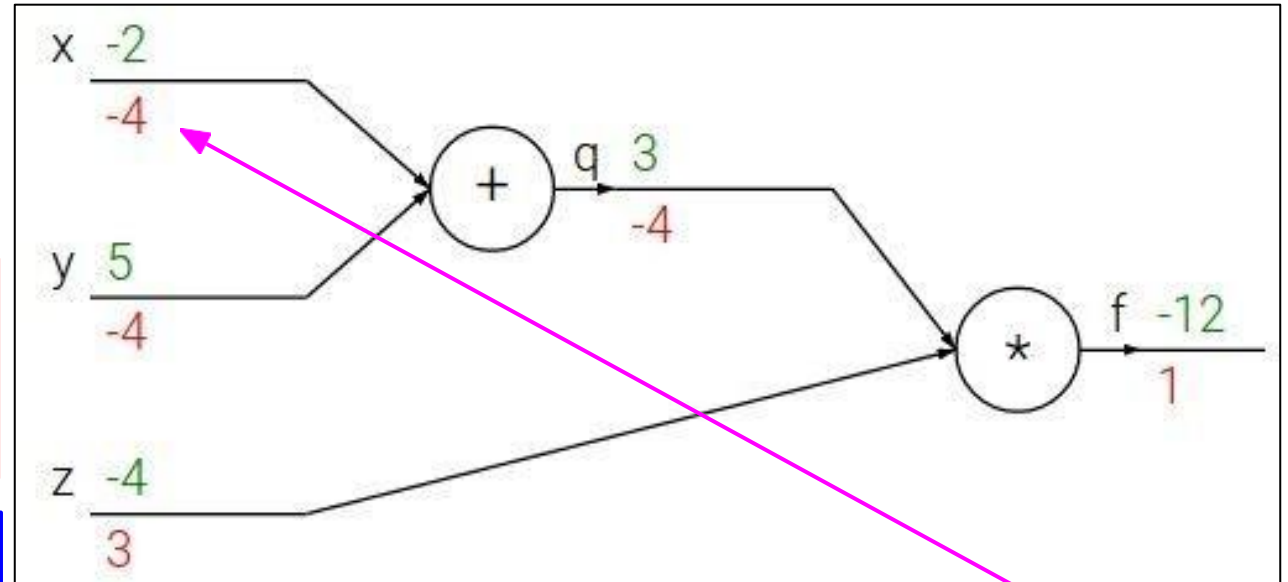
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

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

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

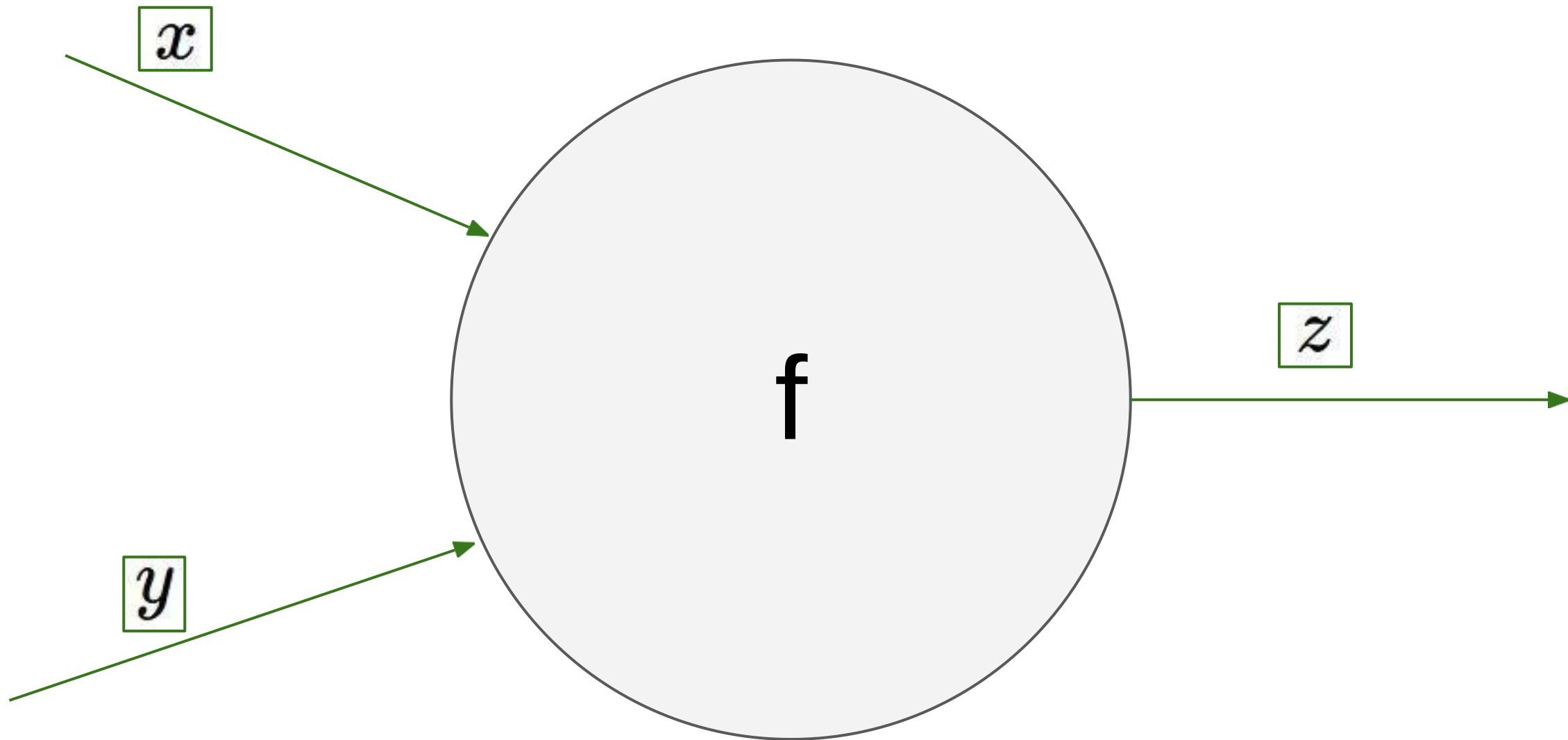
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

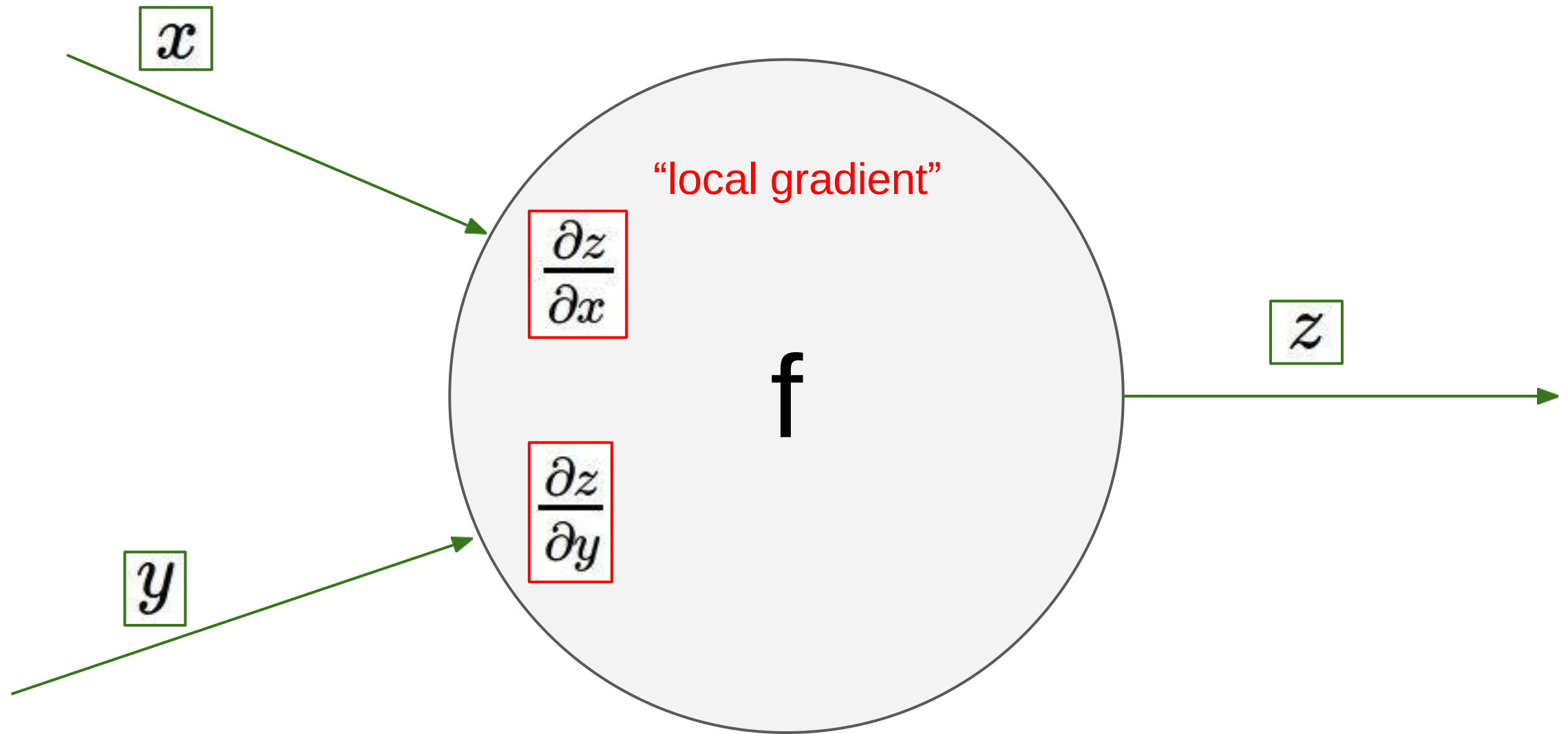


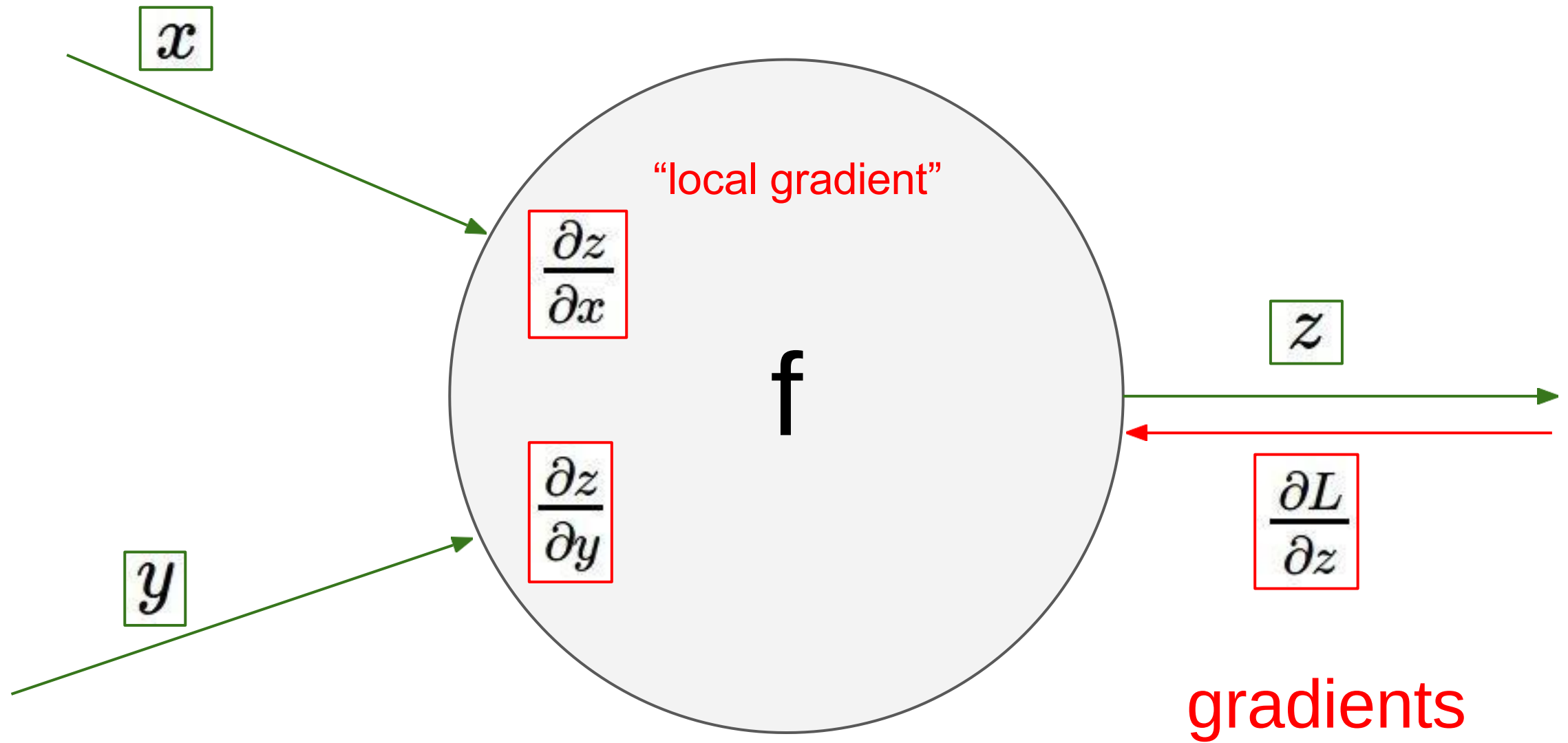
Chain rule:

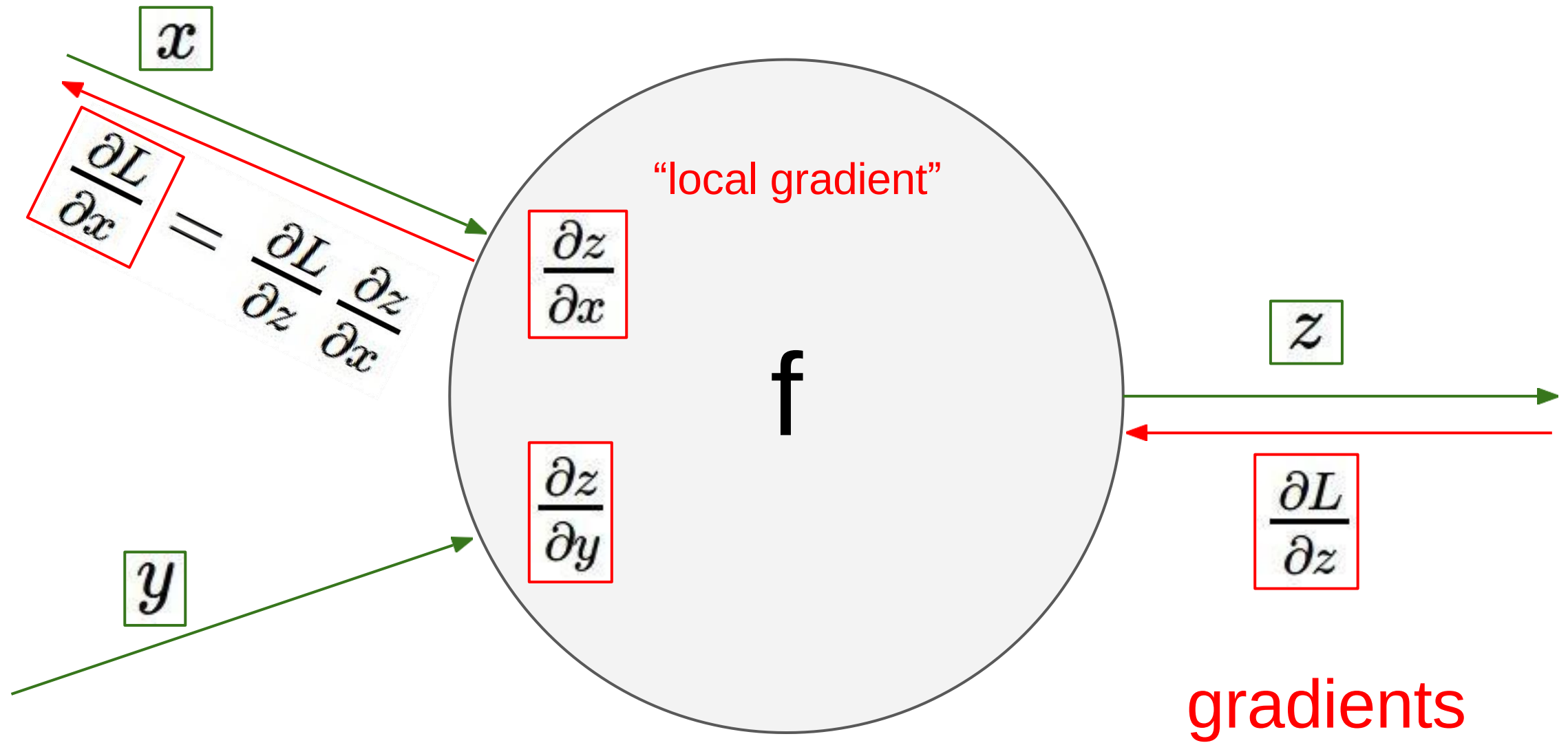
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

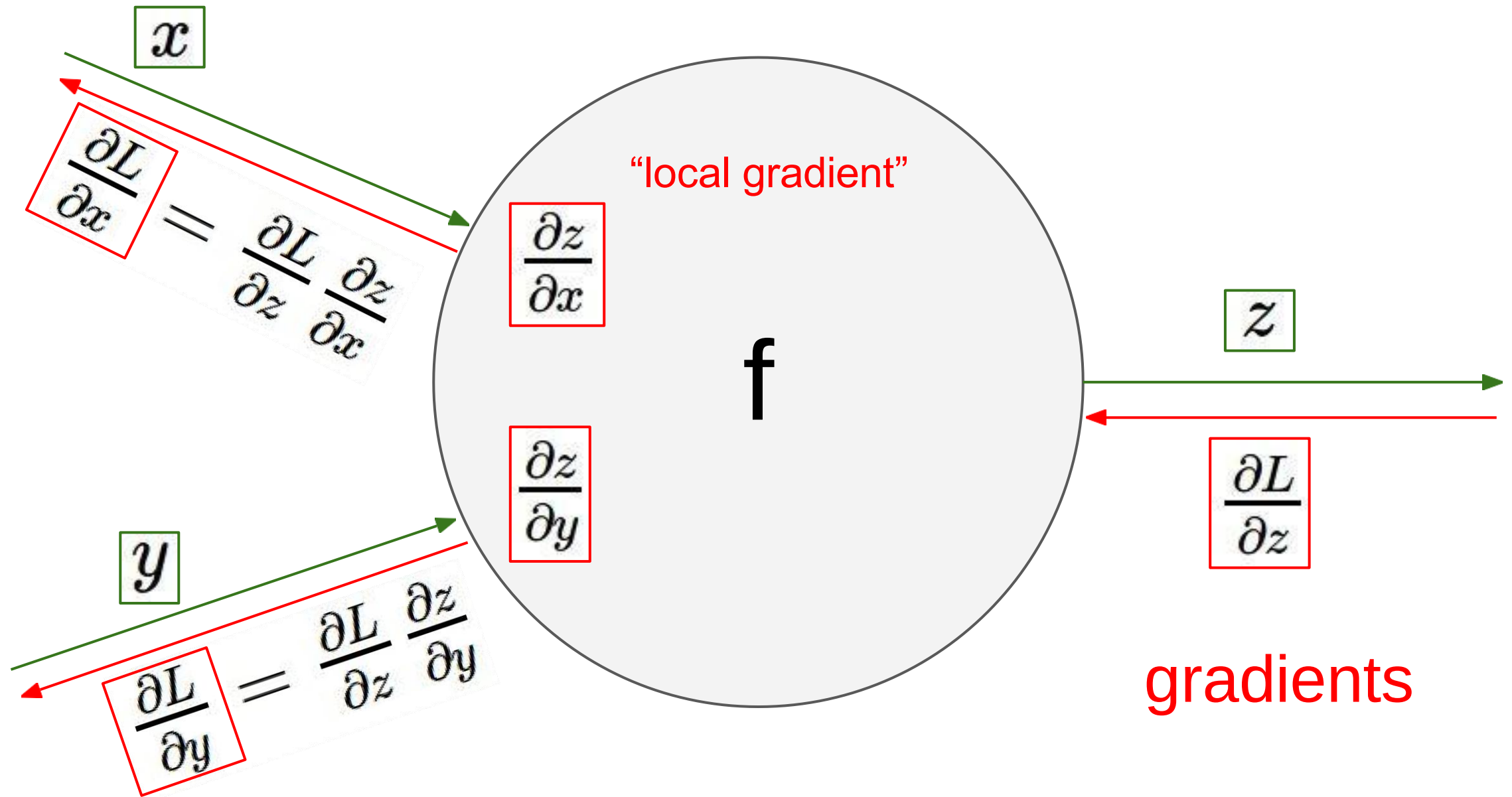
$$\frac{\partial f}{\partial x}$$





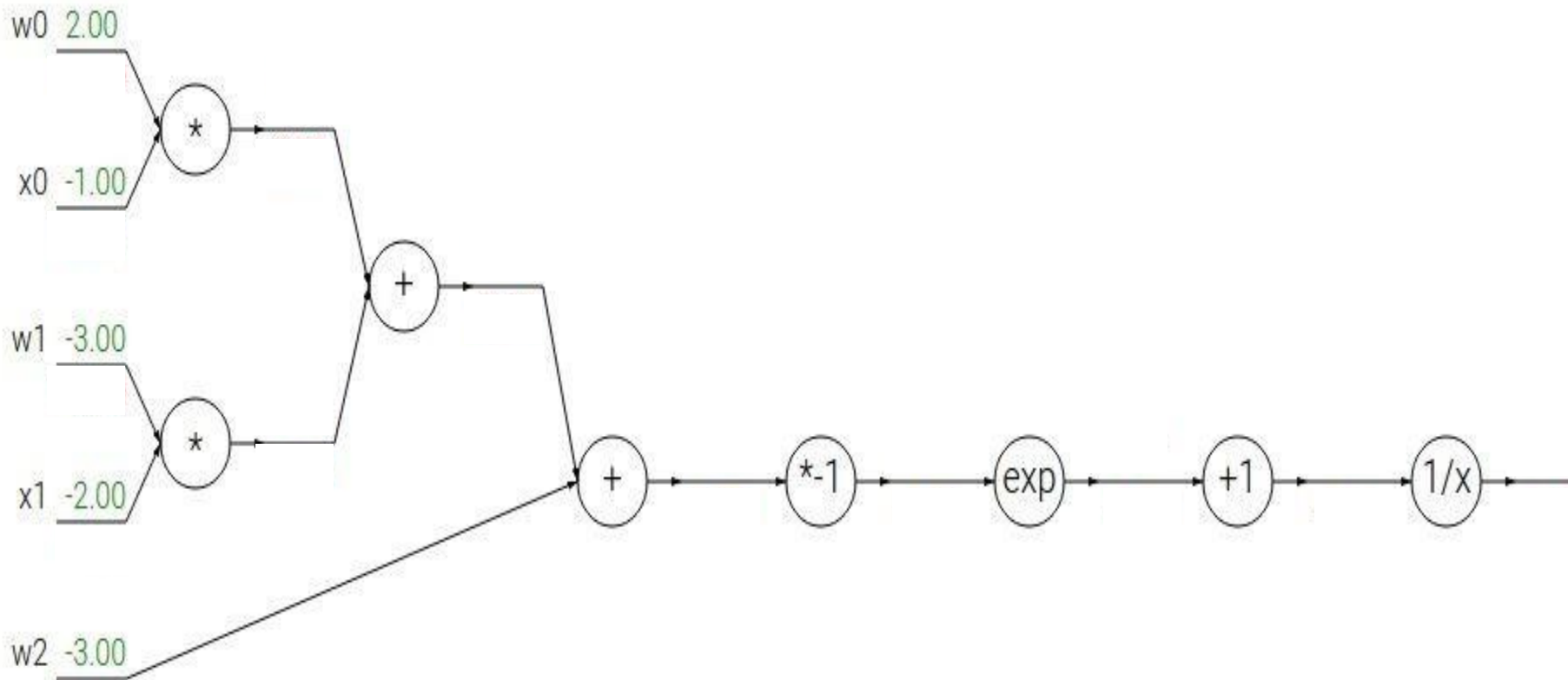






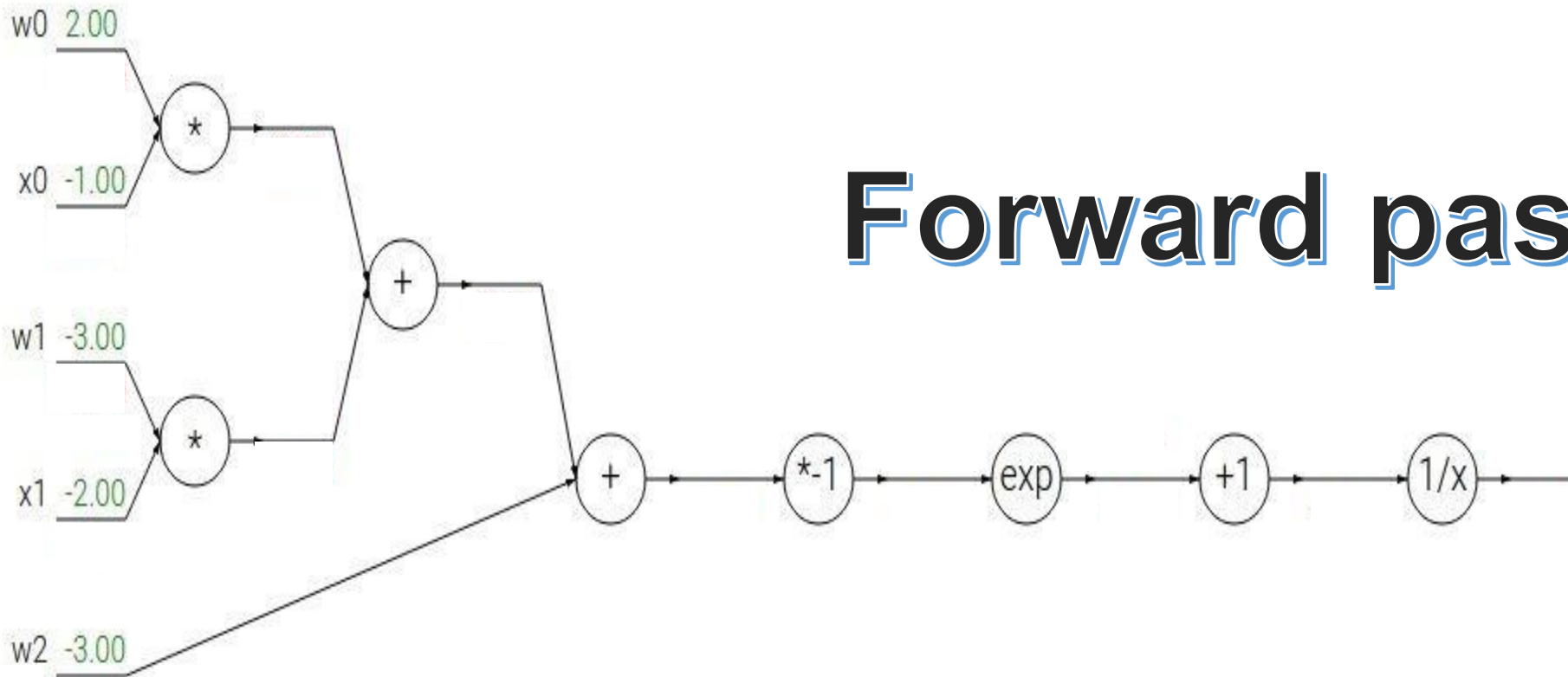
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



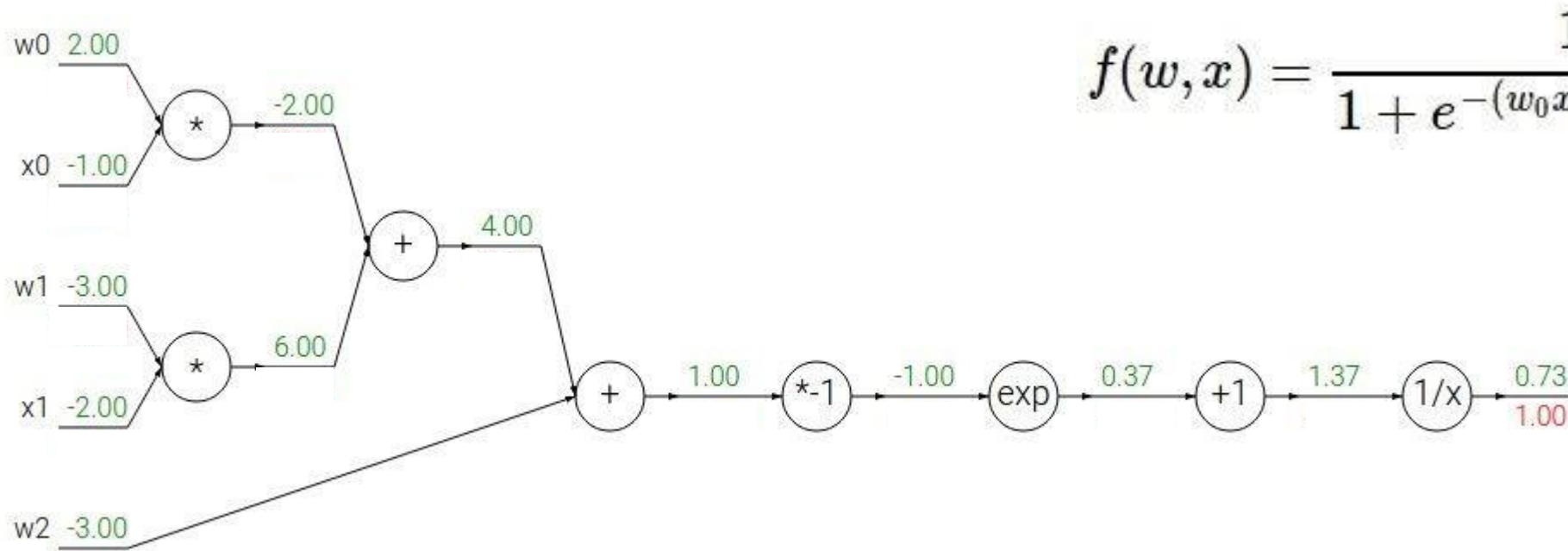
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



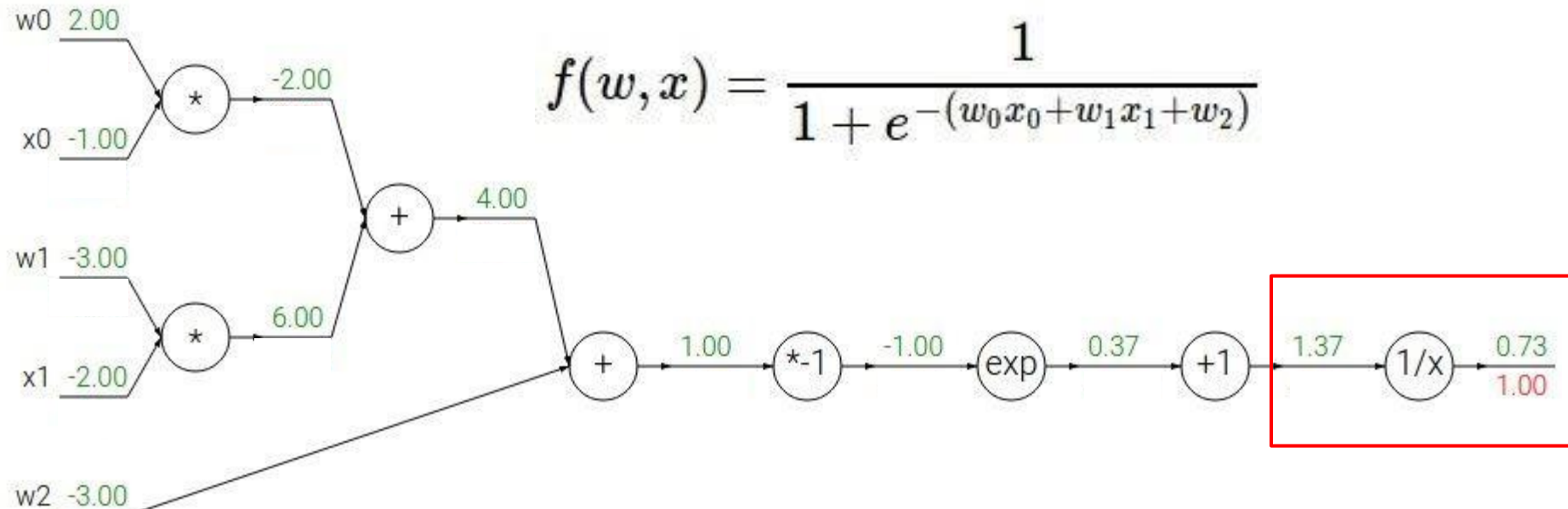
Forward pass

Another example:



$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$		$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

Another example:



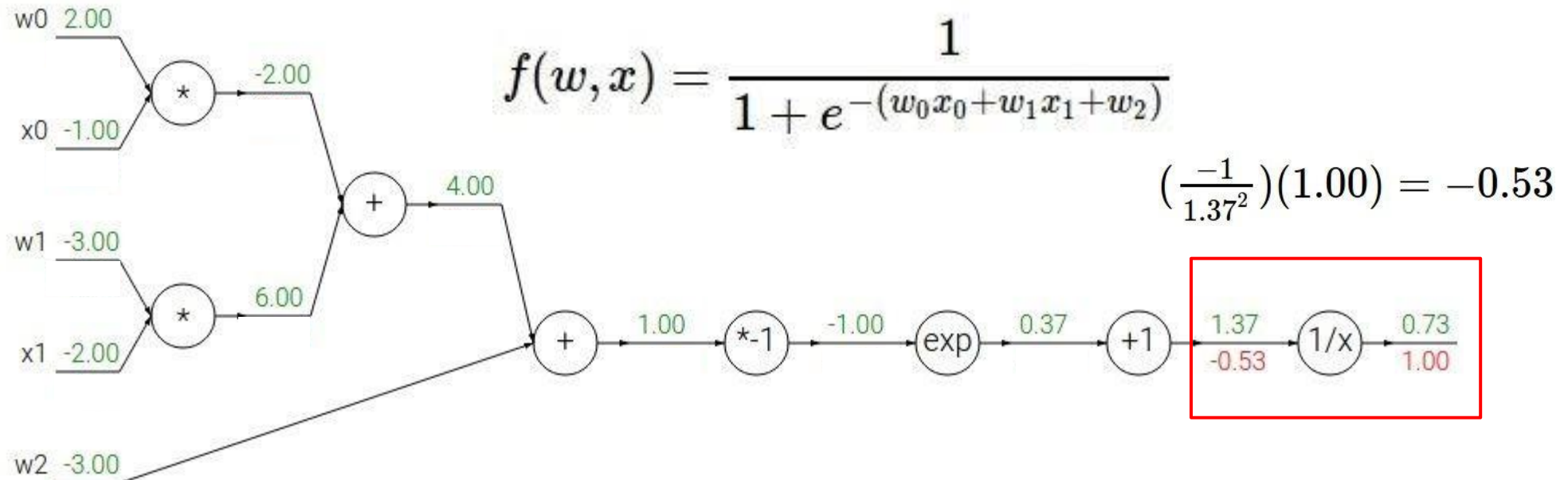
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example:



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

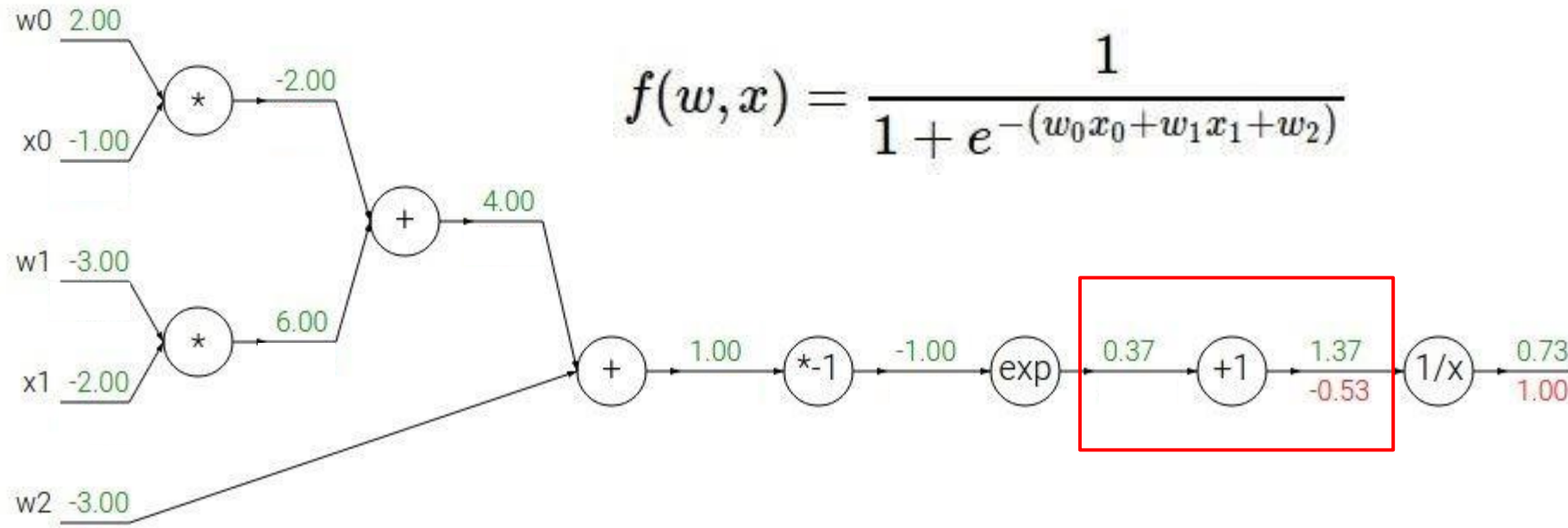
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

$\rightarrow$

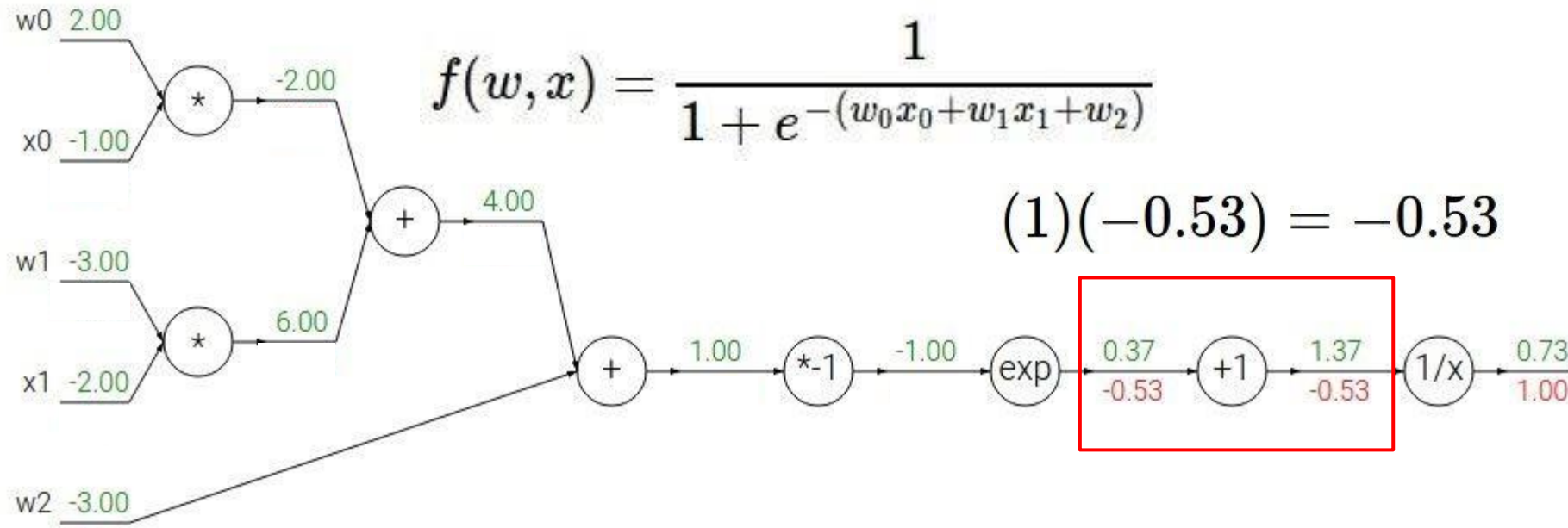
$$\frac{df}{dx} = 1$$

Another example:



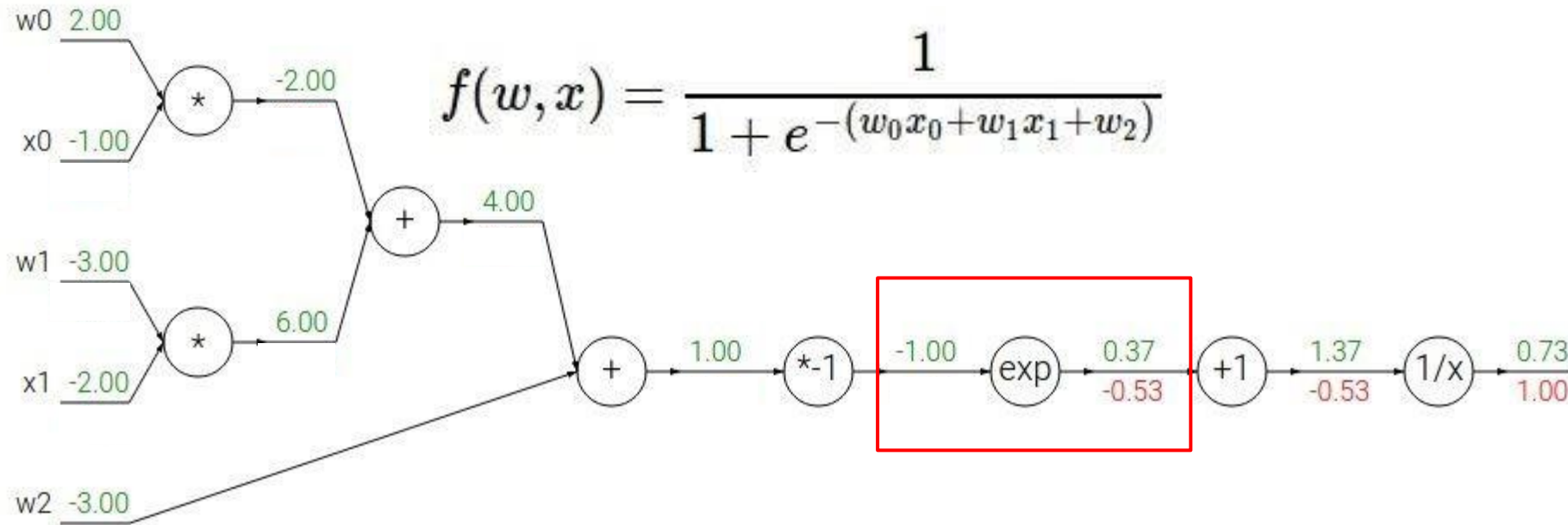
$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$		$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

Another example:



$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$		$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

Another example:



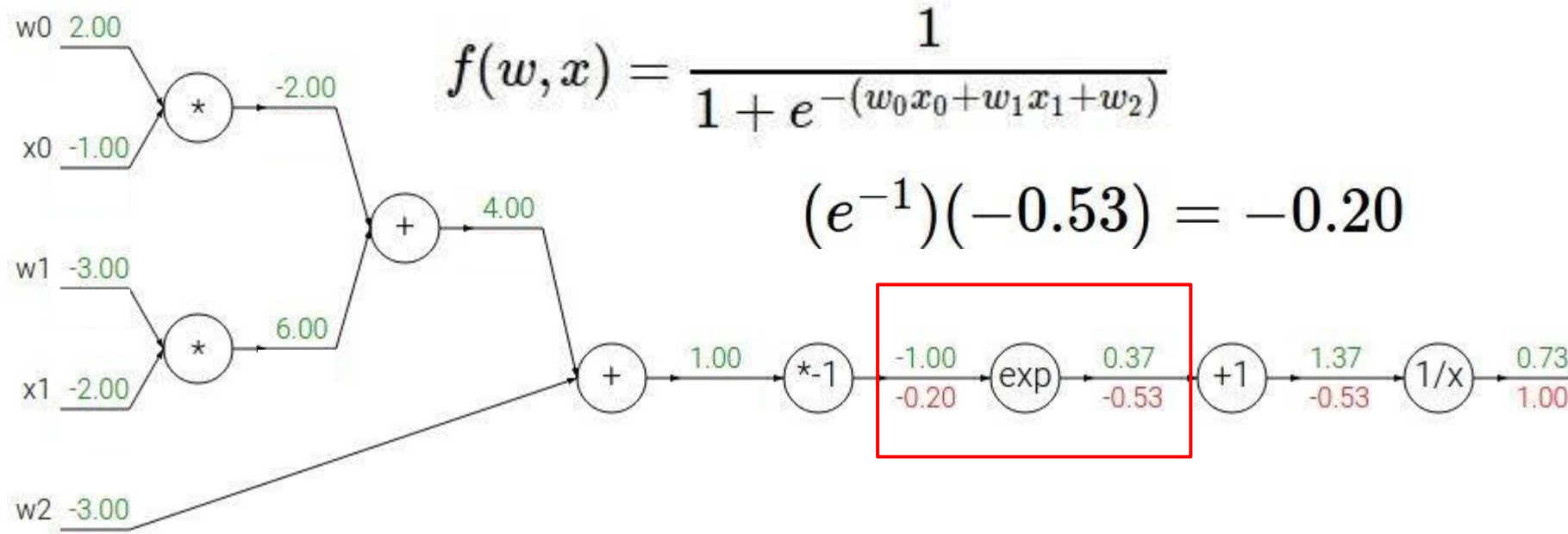
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example:



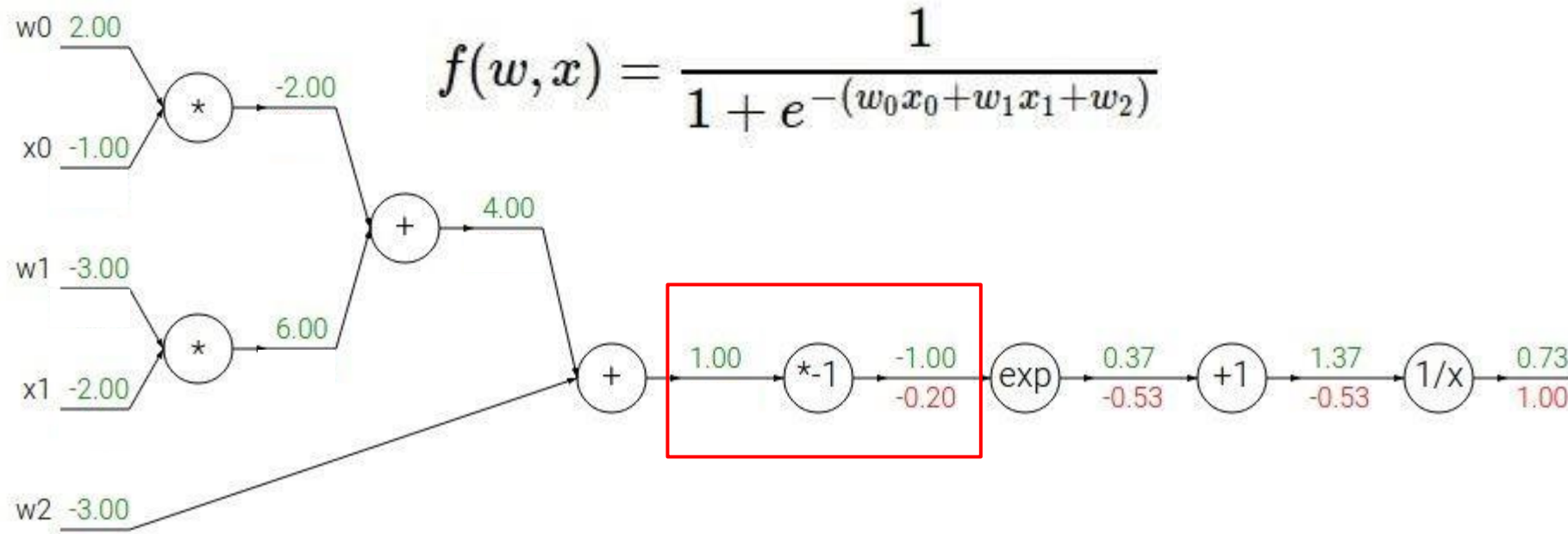
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example:



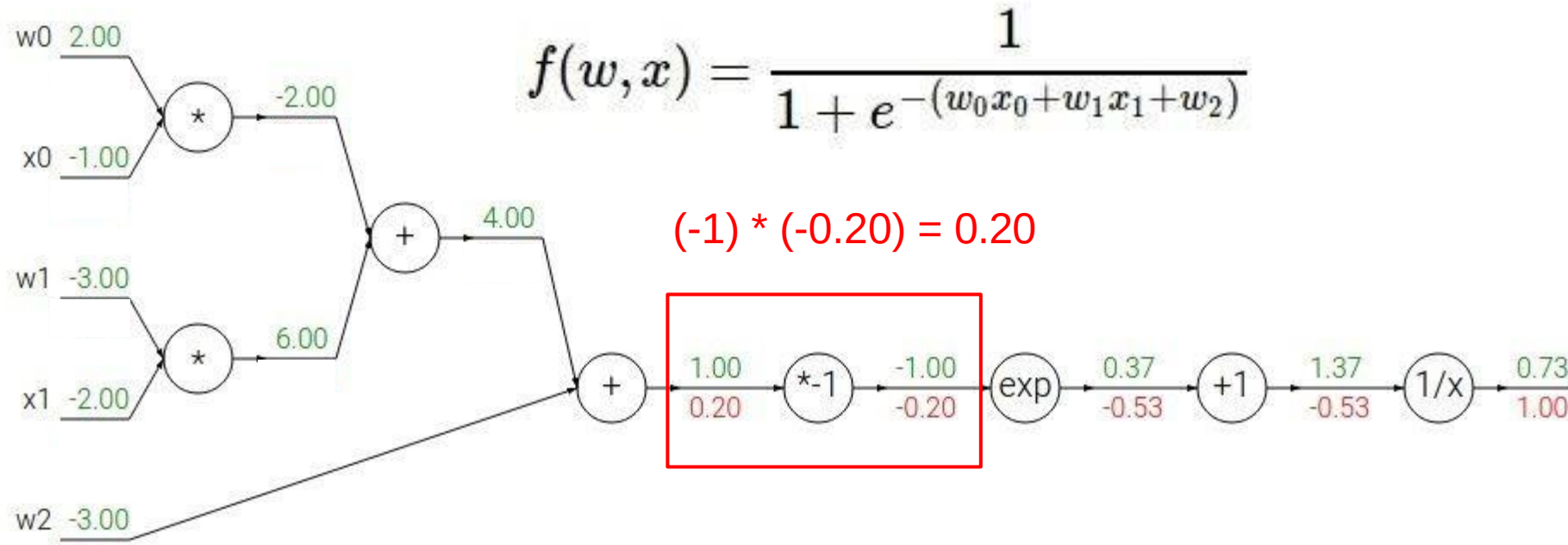
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example:



$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$(-1) * (-0.20) = 0.20$$

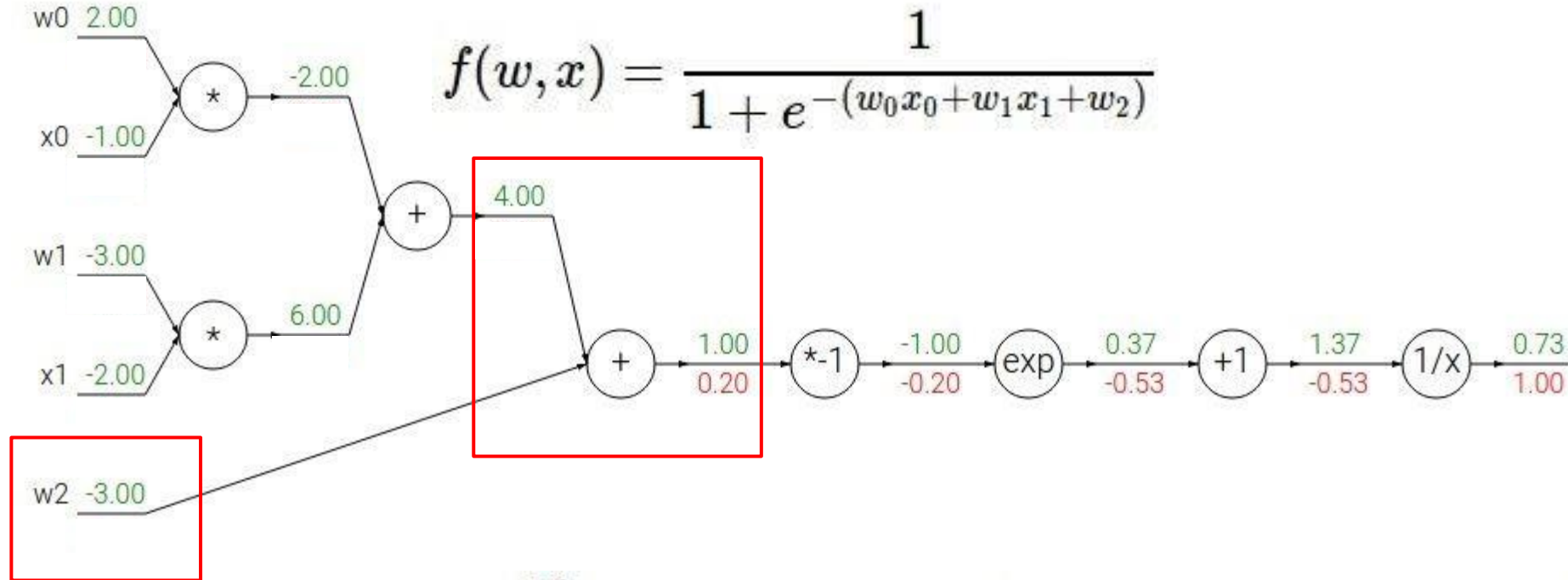
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example:



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

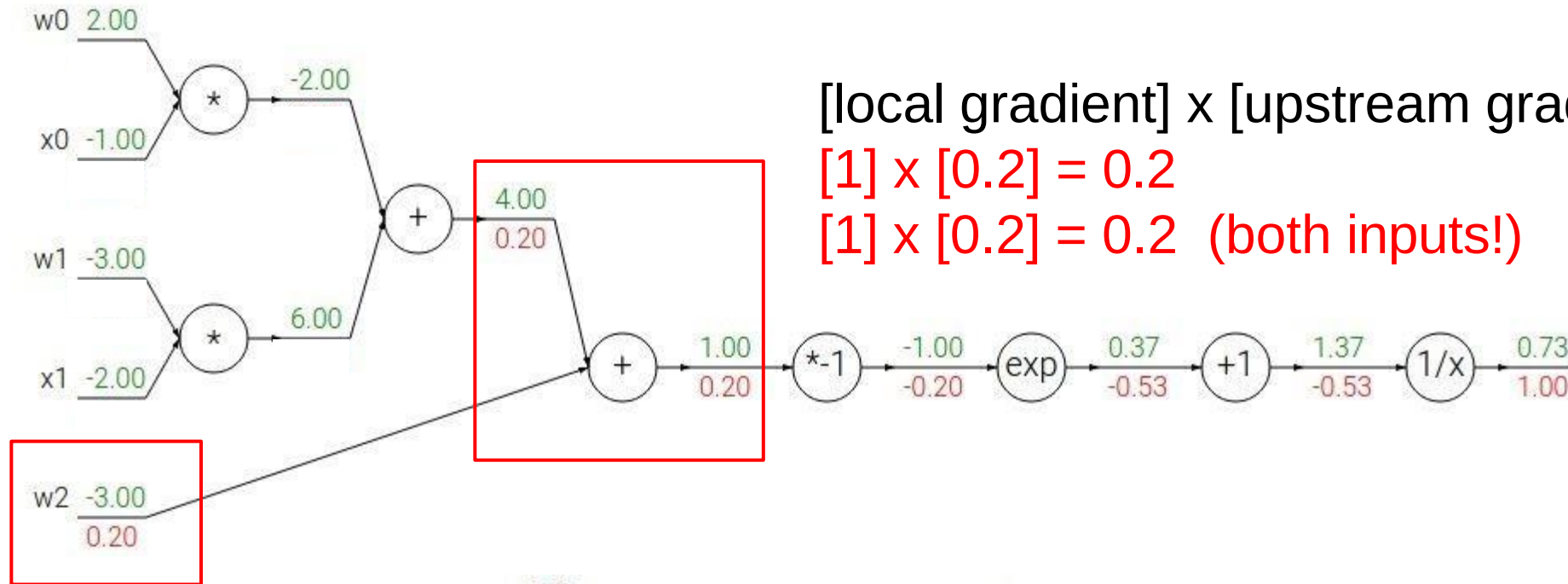
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

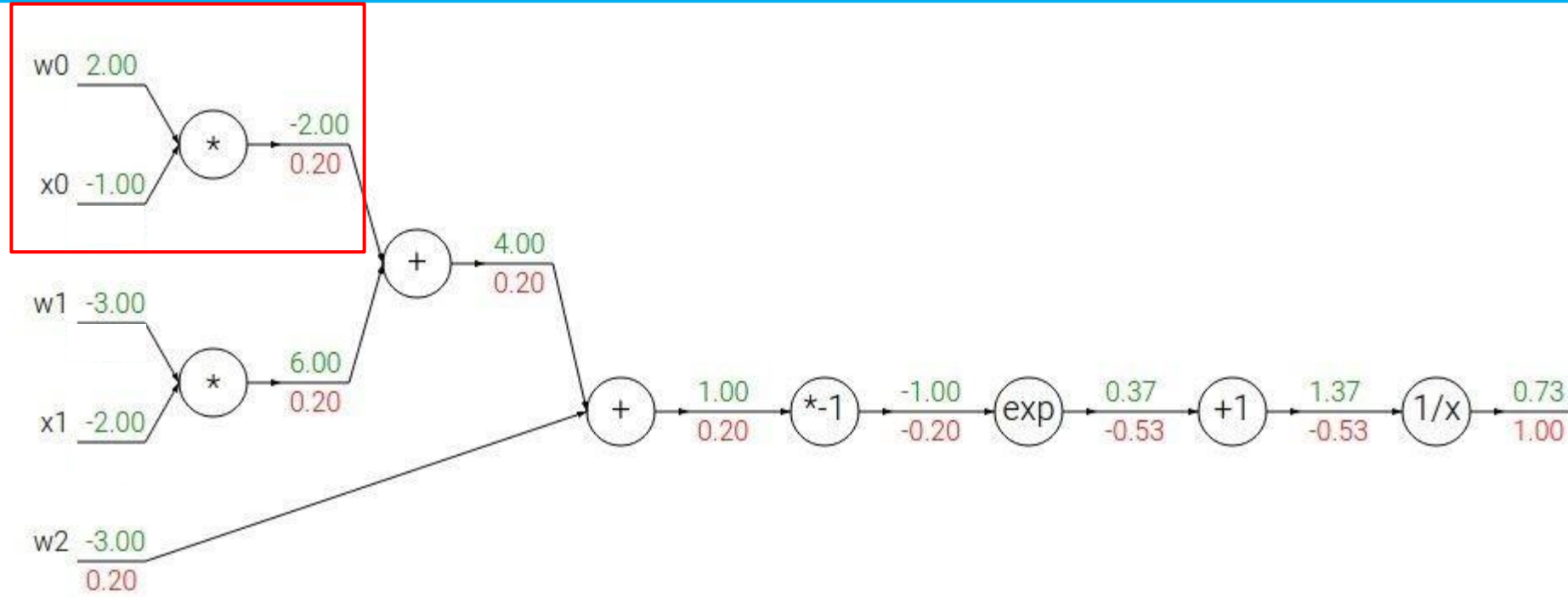
$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

→

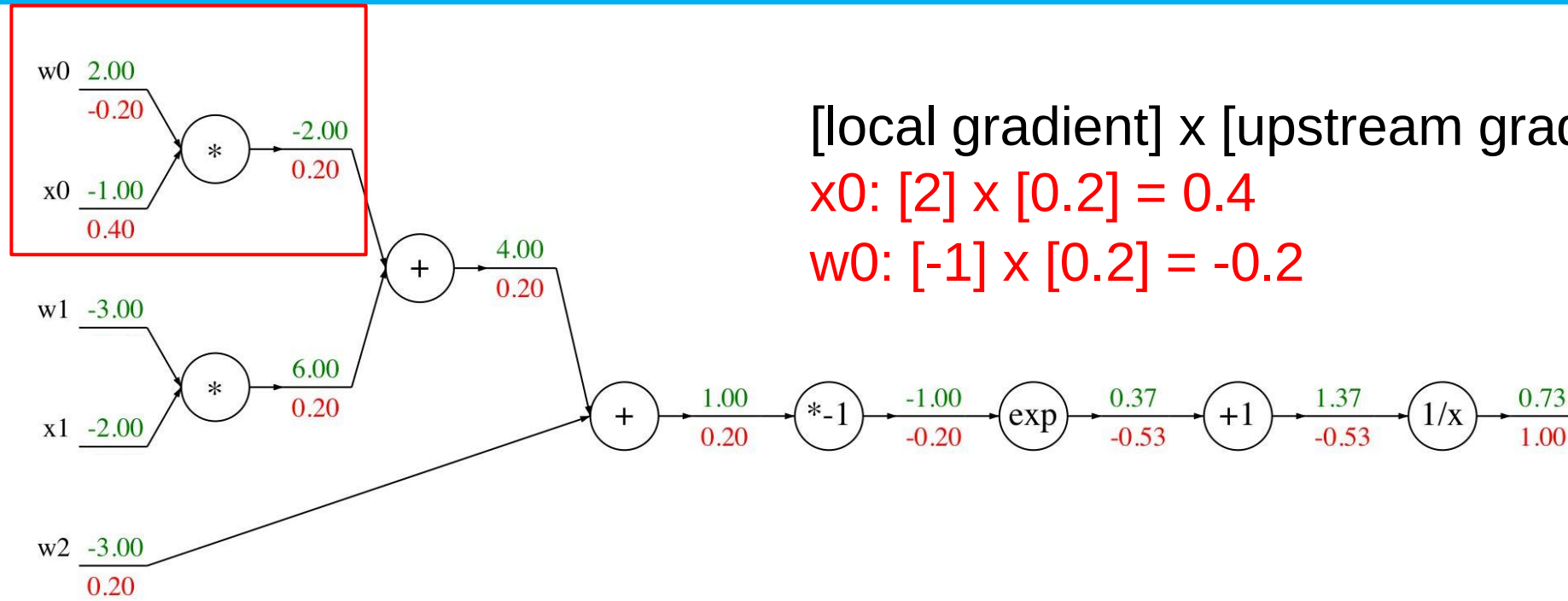
$$\frac{df}{dx} = 1$$

Another example:



$$\begin{array}{lcl}
 f(x) = e^x & \rightarrow & \frac{df}{dx} = e^x \\
 f_a(x) = ax & \rightarrow & \frac{df}{dx} = a
 \end{array}
 \quad \Bigg| \quad
 \begin{array}{lcl}
 f(x) = \frac{1}{x} & \rightarrow & \frac{df}{dx} = -1/x^2 \\
 f_c(x) = c + x & \rightarrow & \frac{df}{dx} = 1
 \end{array}$$

Another example:



[local gradient] x [upstream gradient]

$$x_0: [2] \times [0.2] = 0.4$$

$$w_0: [-1] \times [0.2] = -0.2$$

$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x$$

→

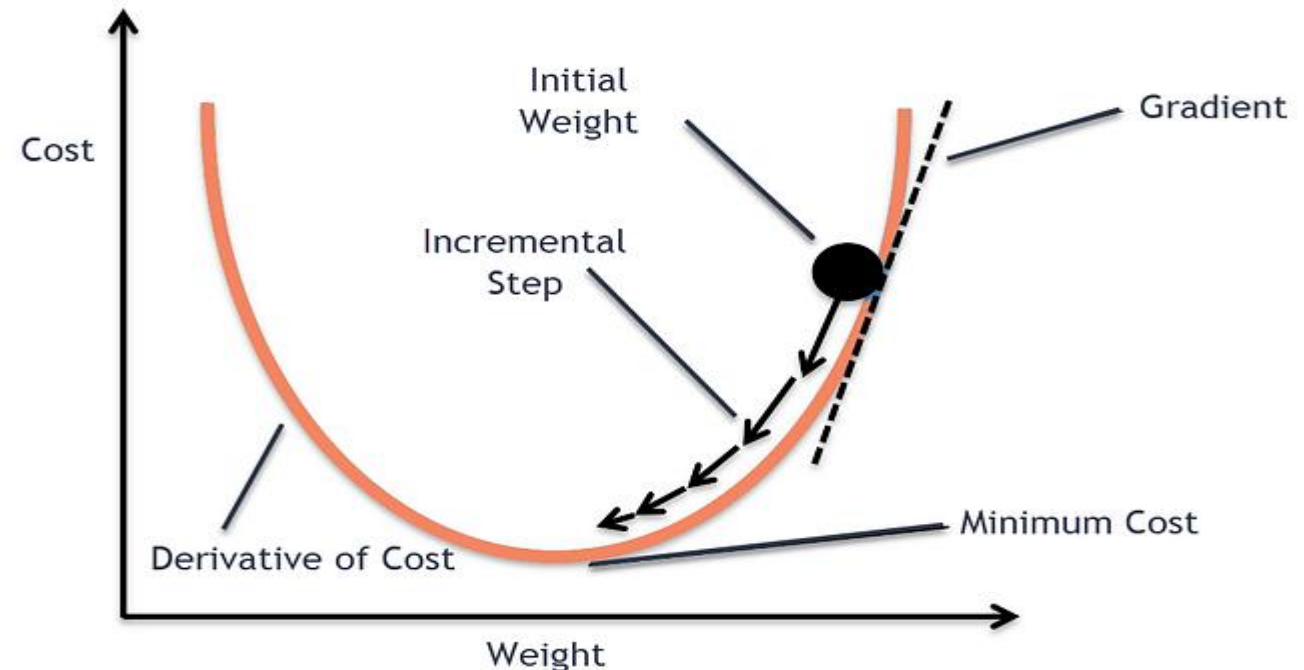
$$\frac{df}{dx} = 1$$

- Our aim is to get to the bottom of our graph(Cost vs weights)
- **Backpropagation** in neural networks also uses a gradient descent algorithm.

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}



- **Advantages:**

1. Easy computation.
2. Easy to implement.
3. Easy to understand.

- **Disadvantages:**

1. May **trap** at **local minima**.
2. Weights are changed after calculating gradient on the whole dataset. Requires **High Computational Cost**.
3. **Gradient Vanishing/Exploding**
4. Sensitive to **Initialization, Learning rate**

- It's a variant of Gradient Descent. It tries to update the model's parameters **more frequently**. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Batch Gradient Descent.

- **Advantages:**

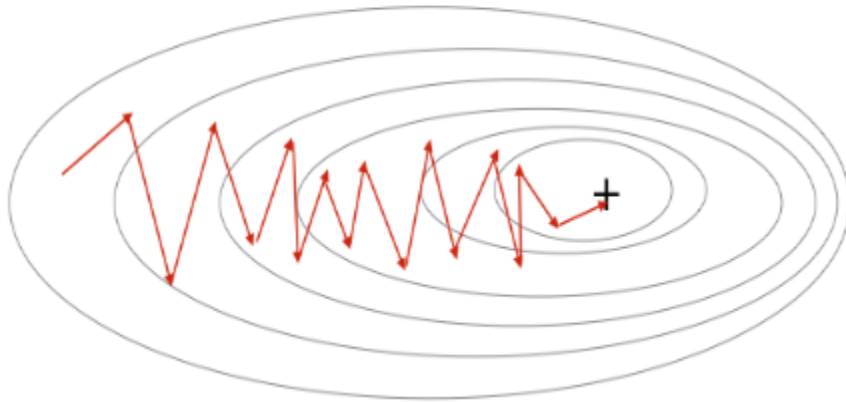
1. Frequent updates of model parameters hence, **converges in less time**.
2. Requires **less memory** as no need to store values of loss functions.
3. May get new **minima's**.

- **Disadvantages:**

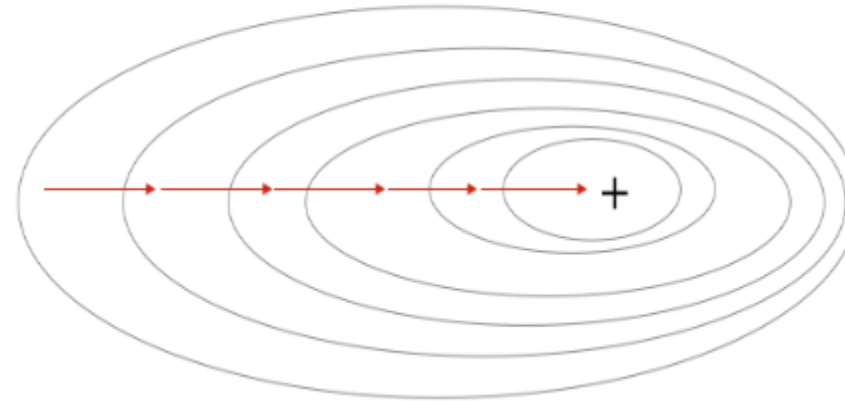
1. **High variance** in model parameters.
2. May shoot even after achieving global minima.
3. **Difficulty Converging**: Due to the noisy gradient estimates, SGD might have difficulties converging to an optimal solution

<https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6#:~:text=Optimizers%20are%20algorithms%20or%20methods,help%20to%20get%20results%20faster>

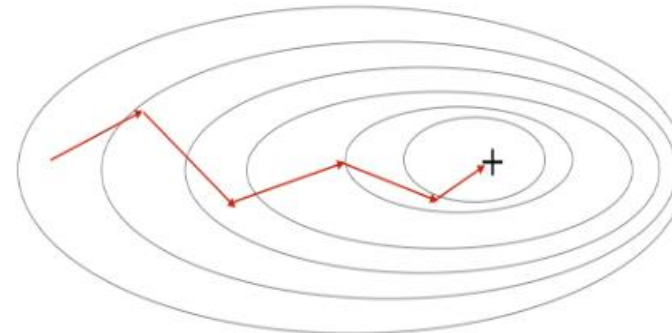
Stochastic Gradient Descent



Gradient Descent



Mini-Batch Gradient Descent



- It updates the model parameters after every batch. So, the dataset is divided into various **batches** and after **every batch**, the parameters are updated.

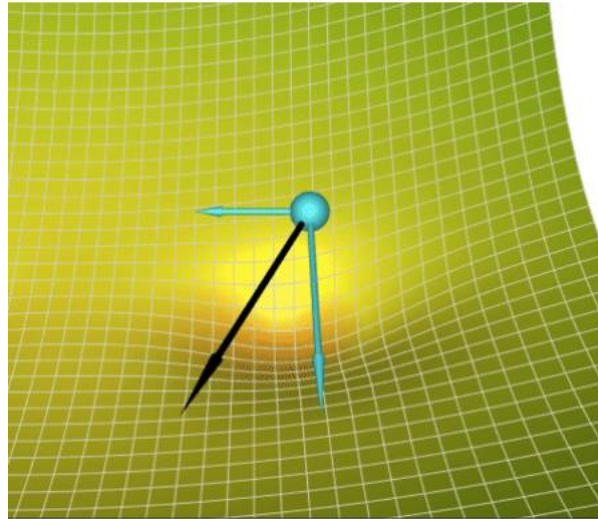
- **Advantages:**

1. Frequently updates the model parameters and also has less variance.
2. Requires medium amount of memory.

- **Disadvantages:**

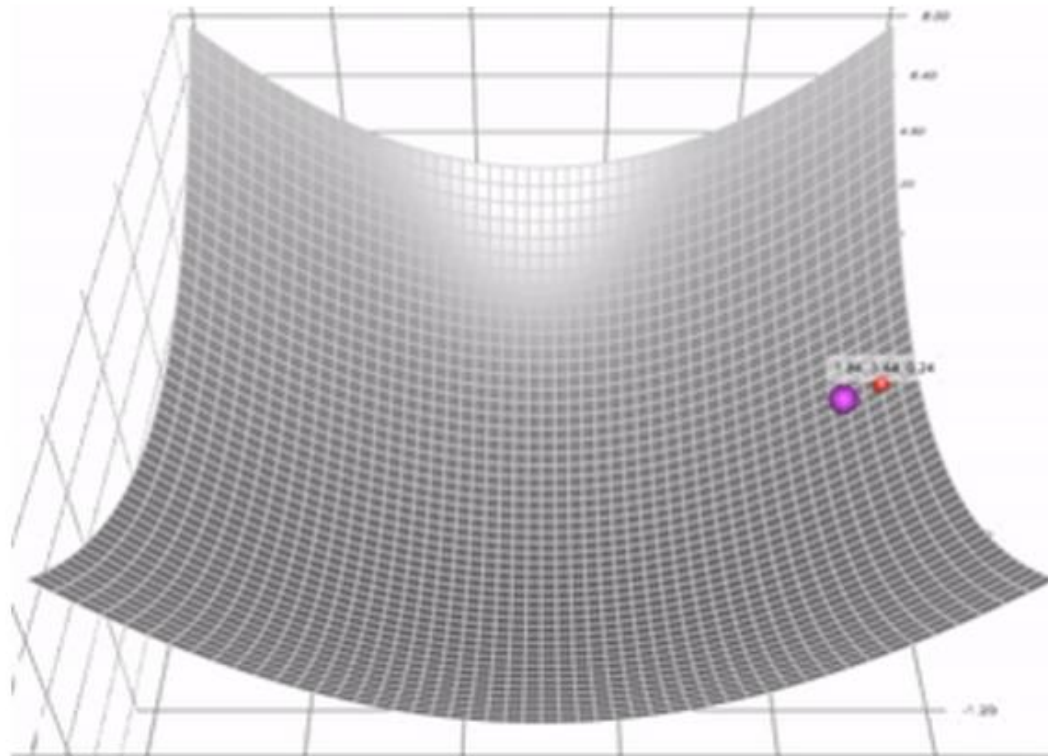
1. The update of MB-SGD is much noisy compared to the update of the GD algorithm.
2. Take a longer time to converge than the GD algorithm.
3. May get stuck at local minima.

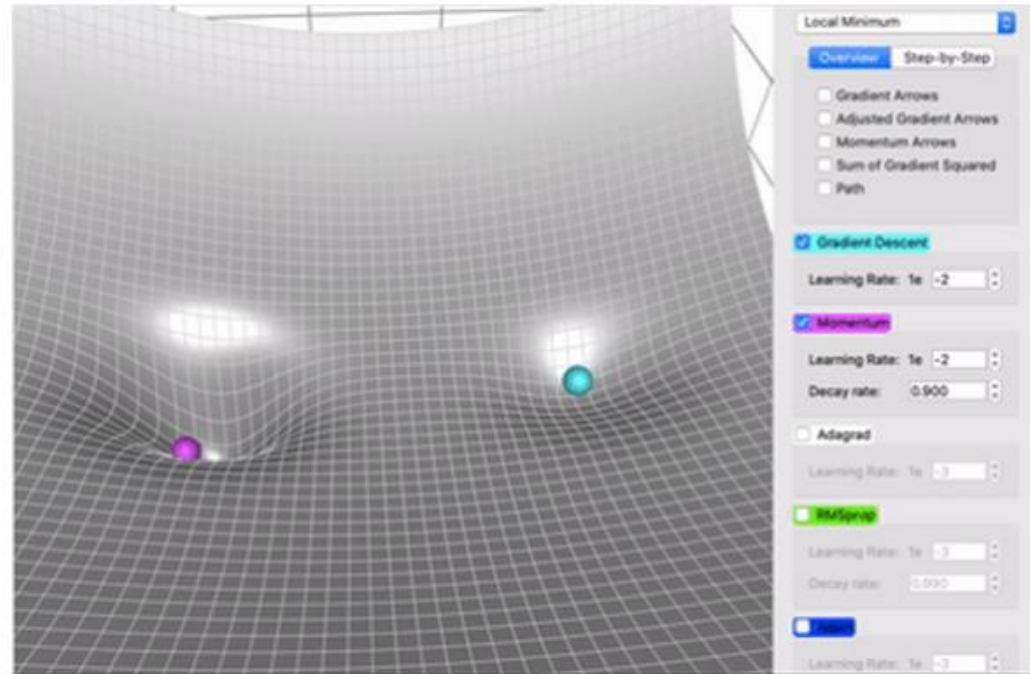
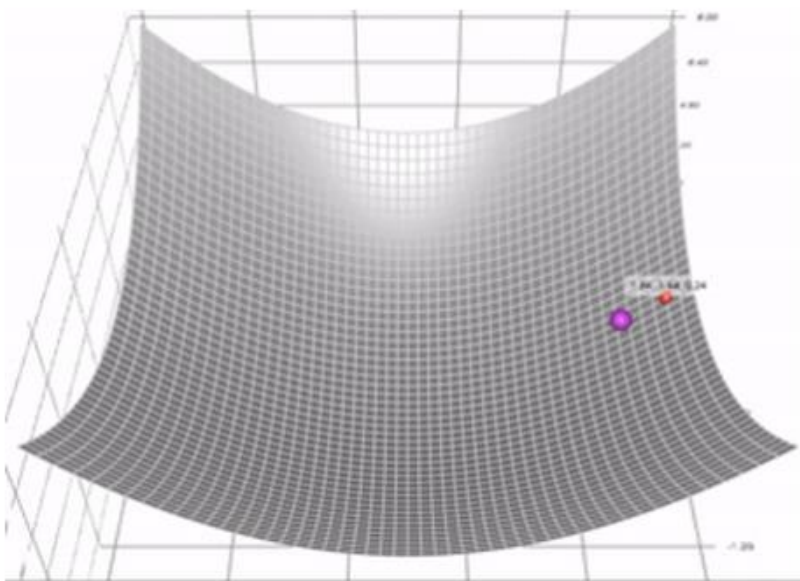
Vanilla Gradient Descent



<https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

- **Momentum** do some additional processing of the gradients to be faster and better.





<https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

- Momentum for short borrows the idea from physics. Imagine rolling down a ball inside of a frictionless bowl. Instead of stopping at the bottom, the momentum it has accumulated pushes it forward, and **the ball keeps rolling back and forth.**

<https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

Gradient Descent:

- update step θ :
 - $\theta = \theta - \alpha * \nabla J(\theta)$
- Where:
 - α (alpha) is the learning rate.
 - $\nabla J(\theta)$ is the gradient of the loss function J with respect to the parameter θ

Momentum:

$$\theta = \theta - \alpha * \nabla J(\theta)$$

- update step parameter θ :

- $v(t) = \mu * v(t-1) - \alpha * \nabla J(\theta)$

$$\theta = \theta + v(t)$$

- Where:

- μ (mu) is the momentum coefficient, typically a value between 0 and 1.
 - $v(t)$ represents the velocity of the parameter at time step t

Advantages:

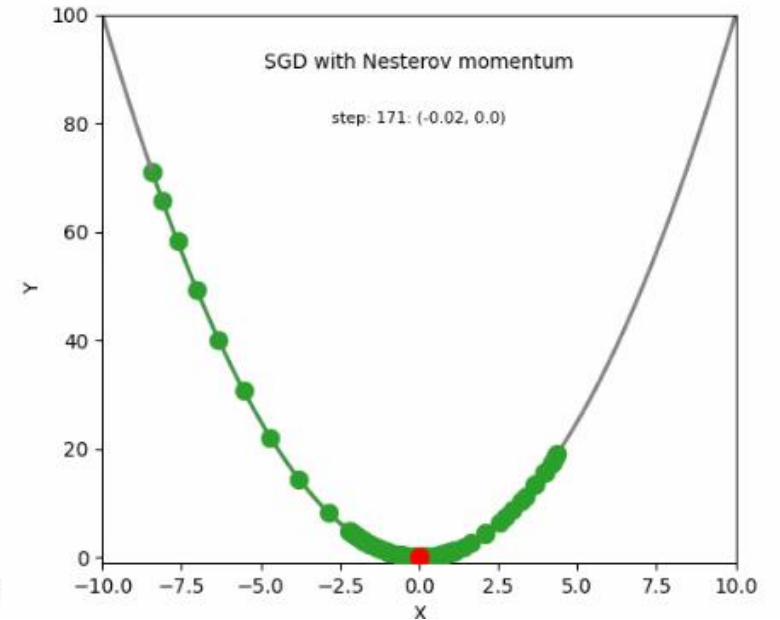
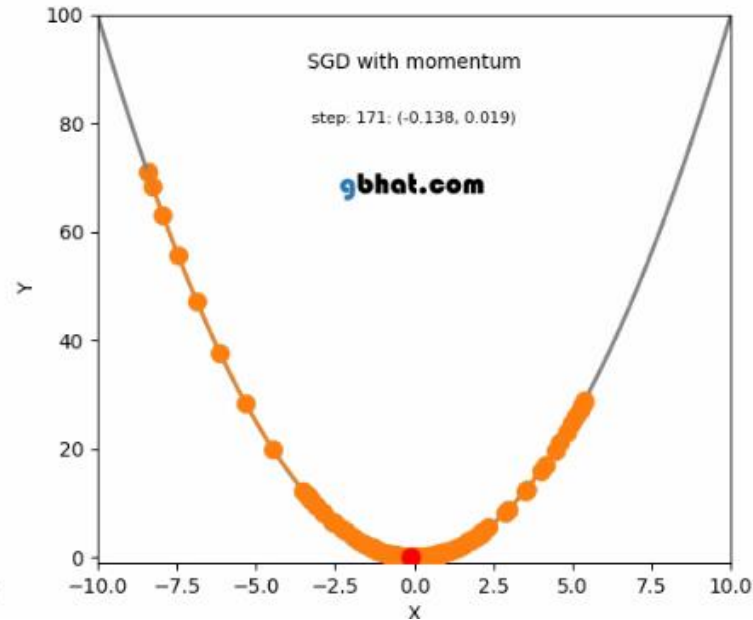
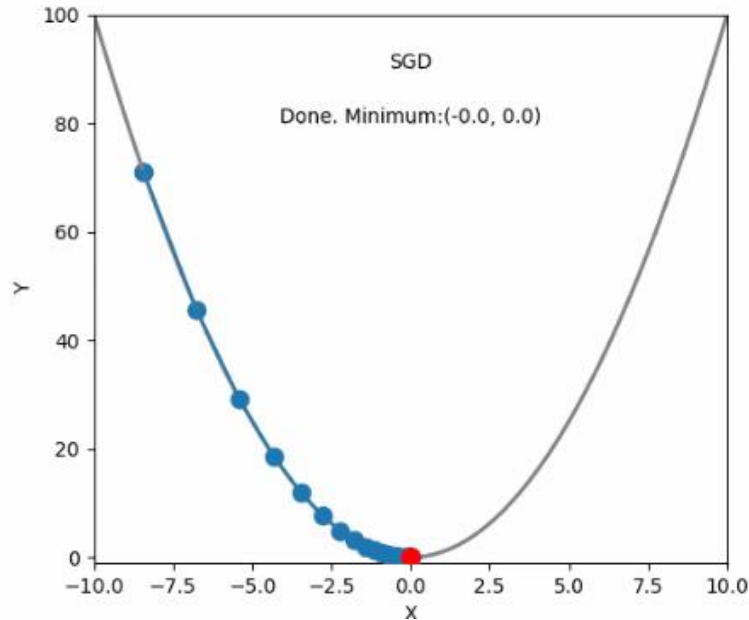
- Converges **faster** than gradient descent. Momentum simply moves faster (because of all the momentum it accumulates)
- Momentum has a shot at escaping local minima (because the momentum may propel it out of a local minimum).

Disadvantages:

- One more hyper-parameter is added which needs to be selected manually and accurately.
- The momentum term takes values **between 0 and 1** is usually set to **0.9** or a similar value

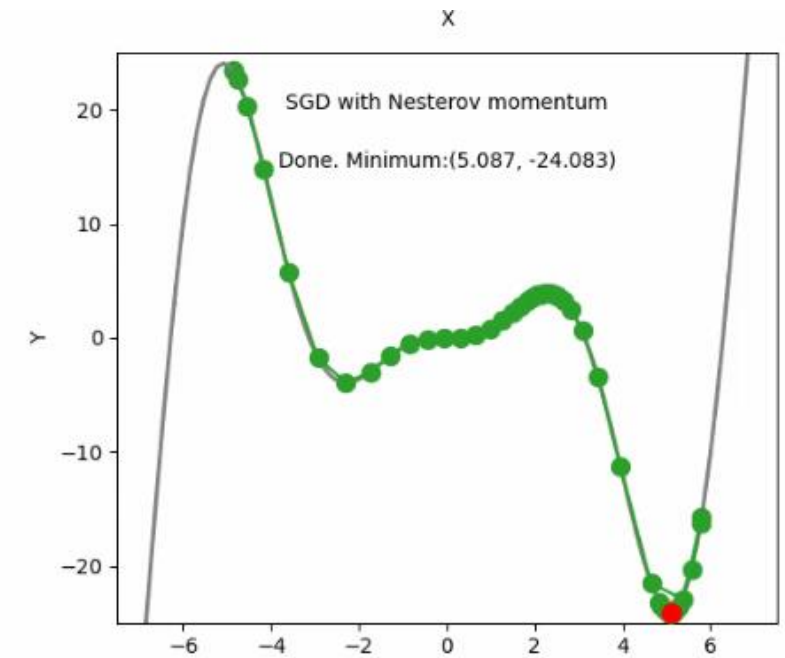
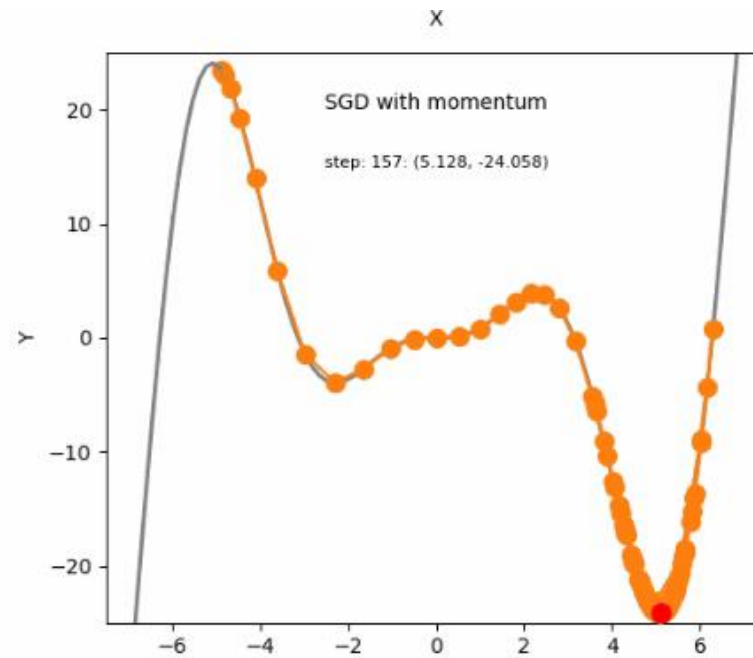
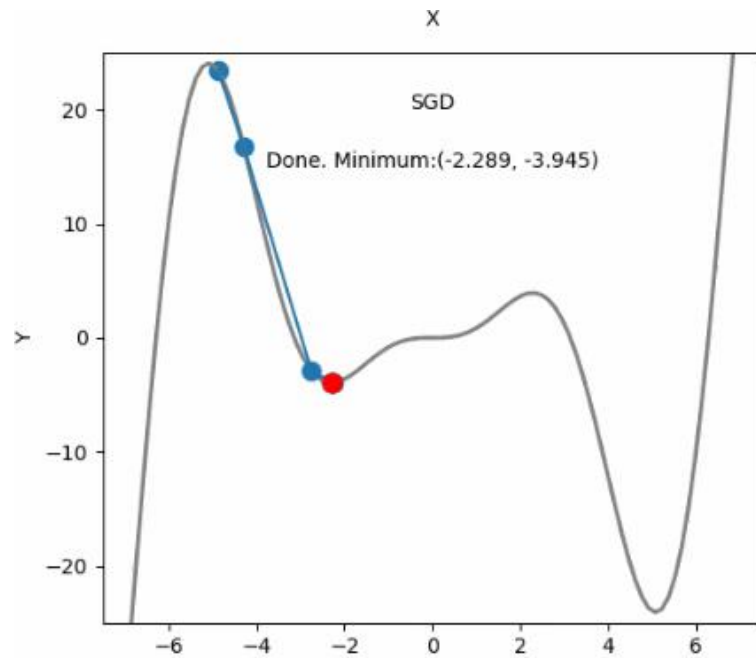
- In traditional momentum-based optimization, you would calculate the gradient of the hill's slope at your current position and use that information to update the ball's position. However, this approach can sometimes lead to overshooting the bottom of the hill or getting stuck in a local minimum.
- in NAG, Instead of just calculating the gradient at your current position, you take a small step forward in the direction of the momentum before estimating the gradient. You peek into the future to see where the ball might end up if it keeps rolling with the current momentum.

SGD vs Momentum vs NAG



https://gbhat.com/machine_learning/gradient_descent_nesterov.html

SGD vs Momentum vs NAG



https://gbhat.com/machine_learning/gradient_descent_nesterov.html

NAG: It updates the parameters based on the estimated gradient at the previous position

- $$v(t) = \mu * v(t-1) - \alpha * \nabla J(\theta - \mu * v(t-1))$$
$$\theta = \theta + v(t)$$

$$\theta = \theta - \alpha * \nabla J(\theta)$$

- Where:

$$v(t) = \mu * v(t-1) - \alpha * \nabla J(\theta)$$

- α (alpha) is the learning rate.
- $\nabla J(\theta - \mu * v(t-1))$ is the gradient of the loss function J with respect to the shifted parameter $\theta - \mu * v(t-1)$.
- μ (mu) is the momentum coefficient, typically a value between 0 and 1. (μ) controls the influence of the past gradient
- $v(t)$ represents the velocity of the parameter at time step t .

- **Advantages:**
 1. Does not miss the local minima.
 2. Slows if minima's are occurring.
- **Disadvantages:**
 1. the hyperparameter needs to be selected manually.

Adagrad (Adaptive Gradient Descent)

- The adaptive gradient descent algorithm uses different learning rates for each iteration. adapts the learning rate for each parameter based on the historical sum of squared gradients?

$$G(t) = G(t-1) + (\nabla J(\theta))^2$$

$$\theta = \theta - (\alpha / (\sqrt{G(t) + \epsilon})) * \nabla J(\theta)$$

Where:

- - $G(t)$ represents the accumulated squared gradients for each parameter at time step t .
- - α (alpha) is the learning rate.
- - $\nabla J(\theta)$ is the gradient of the loss function J with respect to the parameter θ .
- - ϵ (epsilon) is a small constant for numerical stability.

- The problem of AdaGrad, however, is that it is incredibly slow. This is because the *sum of gradient squared* only grows and never shrinks. RMSProp (for **Root Mean Square Propagation**) fixes this issue by adding a decay factor.

$$G(t) = \beta * G(t-1) + (1 - \beta) * (\nabla J(\theta))^2$$

$$\theta = \theta - (\alpha / (\sqrt{G(t) + \epsilon})) * \nabla J(\theta)$$

Where:

- $G(t)$ represents the exponentially decaying average of squared gradients for each parameter at time step t .
- α (alpha) is the learning rate.
- $\nabla J(\theta)$ is the gradient of the loss function J with respect to the parameter θ .
- β (beta) is the decay rate.
- ϵ (epsilon) is a small constant for numerical stability.

Adam(Adaptive Moment Estimation)

- Adam combines the concepts of **momentum** and **RMSprop** to provide **adaptive learning rates** as well as **momentum-based updates**.



الجامعة
البحرية
ميدوcean

LAB 1: Optimizers in Keras

Share

Share notebook

+ Code + Text Copy to Drive

```
1 model.compile(optimizer="",
2               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
3               metrics=['accuracy'])
4 model.fit(train_images, train_labels, epochs=5)
5 test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
6
7 print('\nTest accuracy:', test_acc)
```

```
Epoch 1/5
1875/1875 [=====] - 7s 3ms/step - loss: 0.2096 - accuracy: 0.9265
Epoch 2/5
1875/1875 [=====] - 11s 6ms/step - loss: 0.1940 - accuracy: 0.9304
Epoch 3/5
1875/1875 [=====] - 6s 3ms/step - loss: 0.1905 - accuracy: 0.9315
Epoch 4/5
```



00:03.07





الجامعة
البحرية
ميدوcean

LAB 2: Hyperparameters tuning in Keras

Filter

TensorFlow tutorials

Quickstart for beginners

Quickstart for experts

BEGINNER

ML basics with Keras

- Basic image classification
- Basic text classification
- Text classification with TF Hub
- Regression
- Overfit and underfit
- Save and load
- Tune hyperparameters with the Keras Tuner**
- More examples on keras.io

Learn the latest in machine learning, generative AI, and more at WiML Symposium 2023

Register

TensorFlow > Learn > TensorFlow Core > Tutorials

Was this helpful?

Introduction to the Keras Tuner

Run in Google Colab

View source on GitHub

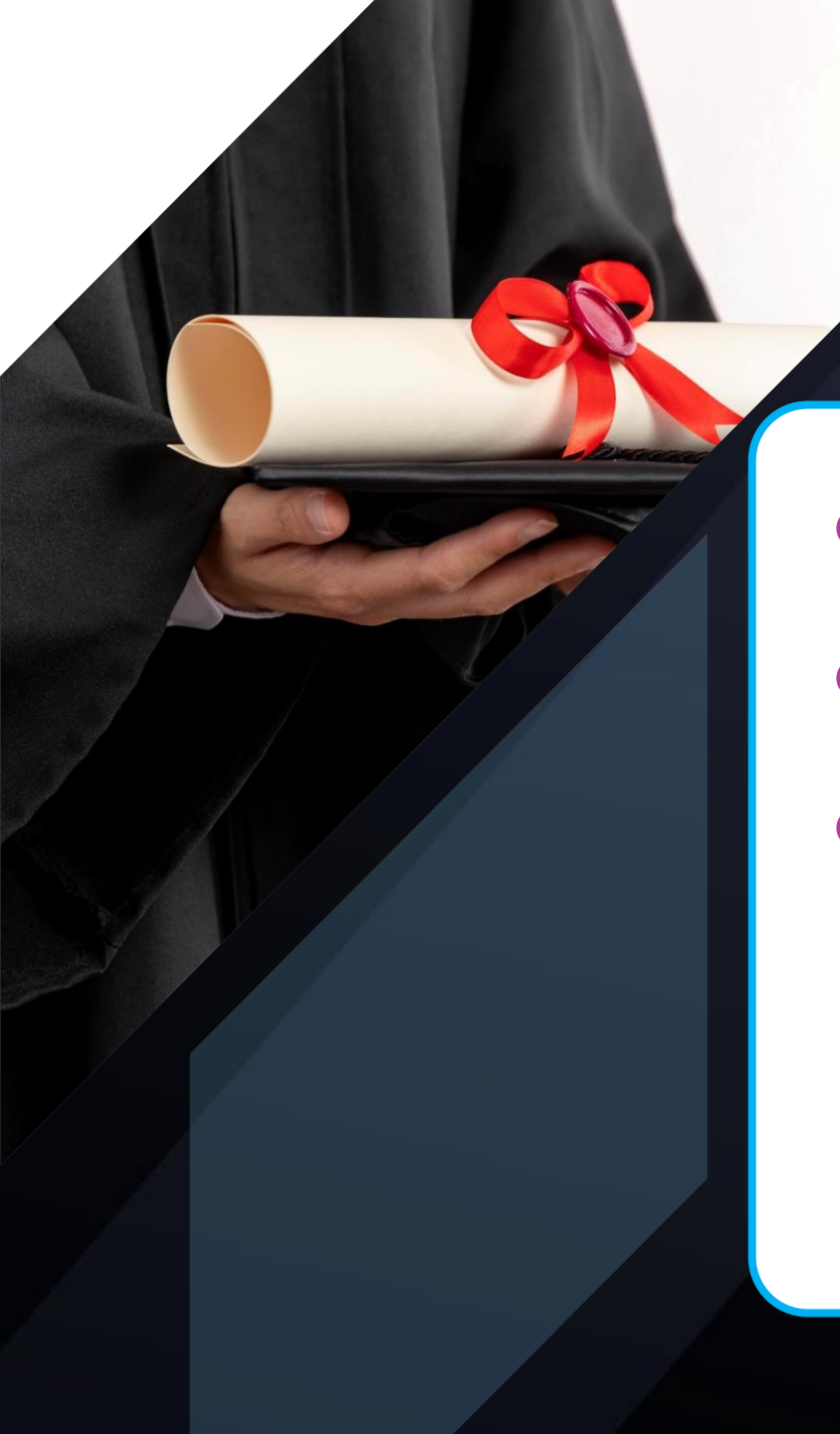
Download notebook

Overview

The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning application is called *hyperparameter tuning* or *hypertuning*.

On this page

- Overview**
- Setup
- Download and prepare the dataset
- Define the model
- Instantiate the tuner and perform hypertuning
- Train the model
- Summary



LECTURE SUMMARY

- ◉ Backpropagation
- ◉ Most common optimizers
- ◉ Hyperparameter tuning



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY

*Thank
you*



www.midocean.ae

