



THE UNIVERSITY OF  
**MELBOURNE**

# Introduction to Deep Learning -- Recap of Neural network

**COMP90051 Statistical Machine Learning**

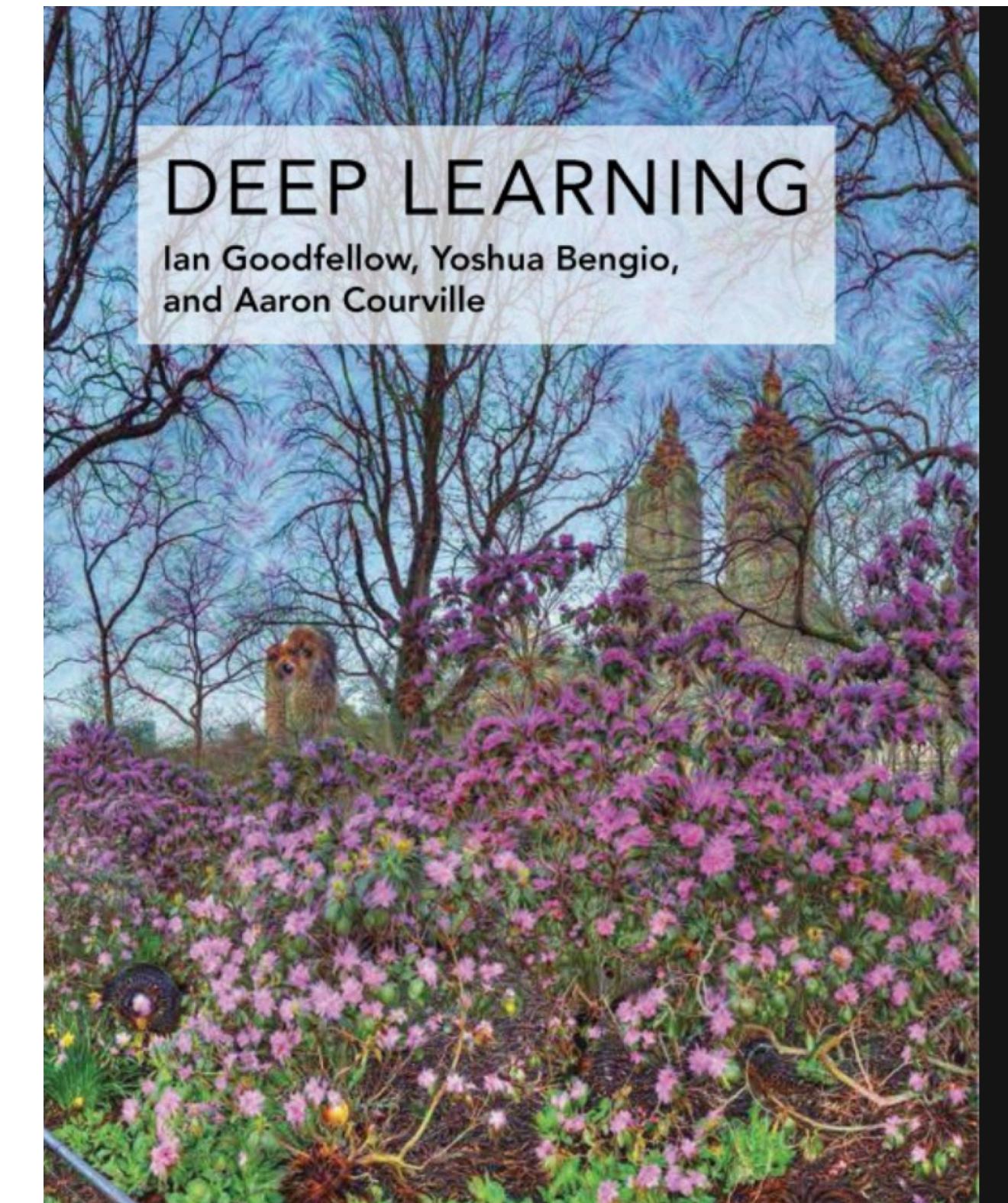
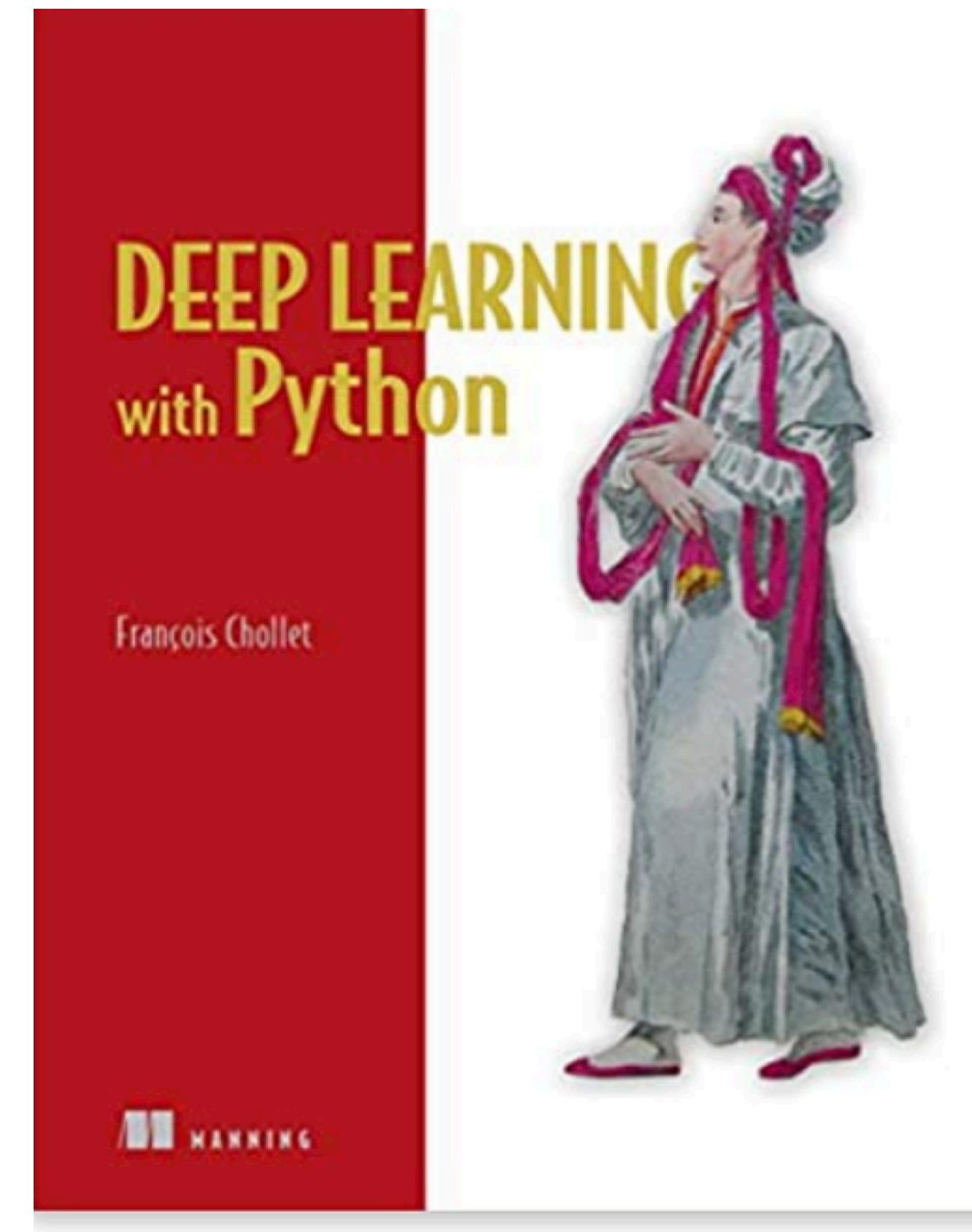
**QiuHong Ke**

Copyright: University of Melbourne

# Before we start

## Books & resources

- Deep Learning with Python, by Francois Chollet, available in unimelb library
- Deep Learning, by Ian Goodfellow and Yoshua Bengio and Aaron Courville, <https://www.deeplearningbook.org/>



# Angry birds

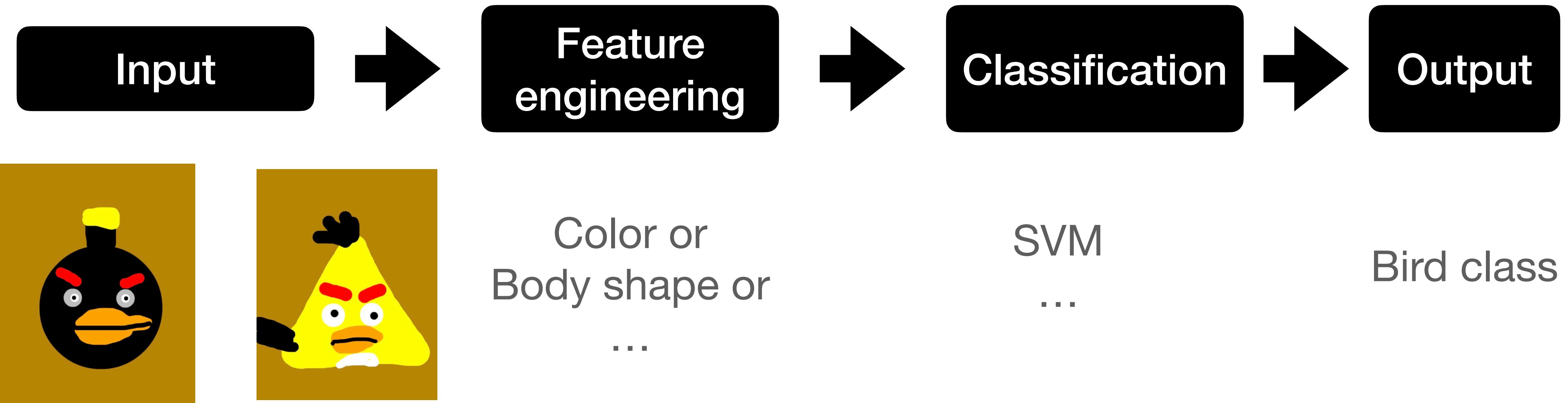


VS



Source: by Papachan (CC BY)  
<https://www.sketchport.com/drawing/4524248689278976/bomb-and-chuck>

# Bird classification



Source: by Papachan (CC BY)  
<https://www.sketchport.com/drawing/4524248689278976/bomb-and-chuck>

# In real life...

Boat tailed Grackle



Bobolink



Bohemian Waxwing



Brandt Cormorant



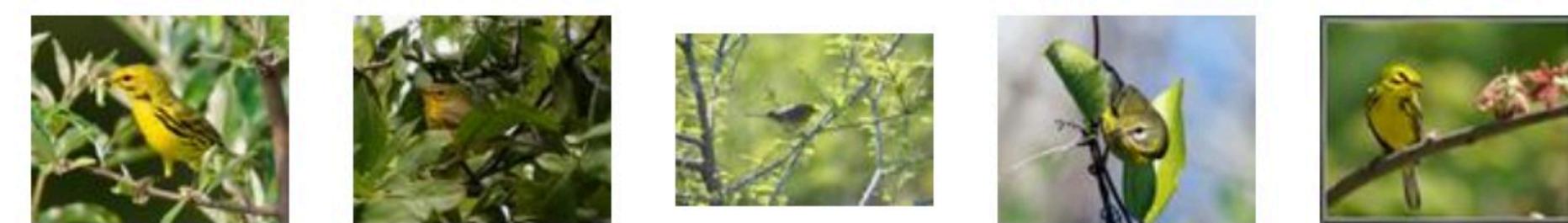
Brewer Blackbird



Pomarine Jaeger



Prairie Warbler



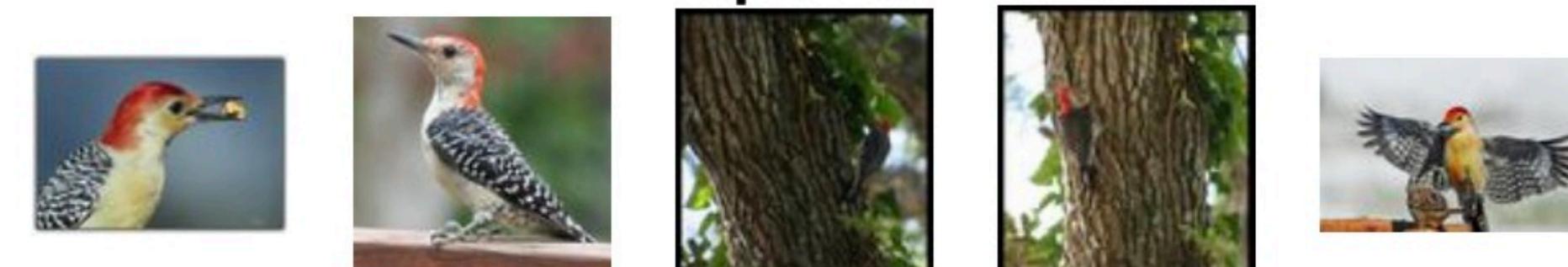
Prothonotary Warbler



Purple Finch



Red bellied Woodpecker



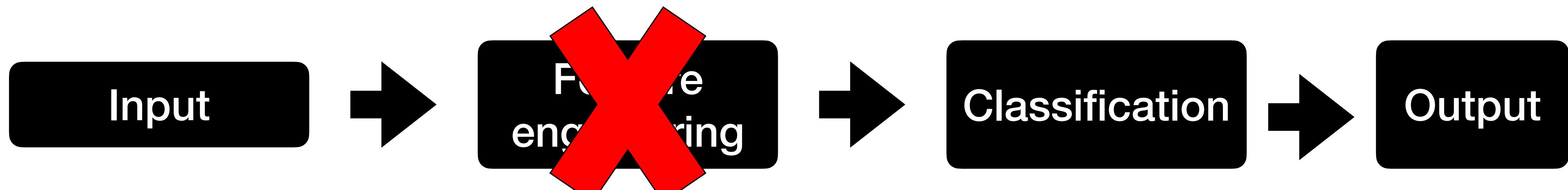
# More features for real birds...



forehead_color	<b>black</b>	<b>black</b>	<b>black</b>
breast_pattern	<b>solid</b>	<b>solid</b>	<b>solid</b>
breast_color	<b>white</b>	<b>white</b>	<b>white</b>
head_pattern	<b>plain</b>	<b>capped</b>	plain
back_color	<b>white</b>	white	<b>black</b>
wing_color	grey/white	<b>grey</b>	<b>white</b>
leg_color	<b>orange</b>	orange	<b>orange</b>
size	<b>medium</b>	large	<b>medium</b>
bill_shape	needle	dagger	<b>dagger</b>
wing_shape	<b>pointed</b>	<b>tapered</b>	<b>long</b>
...	...	...	...
primary_color	white	white	white

Source: Welinder, Peter, et al. "Caltech-UCSD birds 200." (2010).

# ‘REAL Bird’ classification



Use deep learning  
models!

Bird class

# Outline

- Introduction
- Neural network (Perceptron and Multi-layer Perceptron)
- Gradient descent algorithm

# Introduction

# Neural network

# Gradient descent

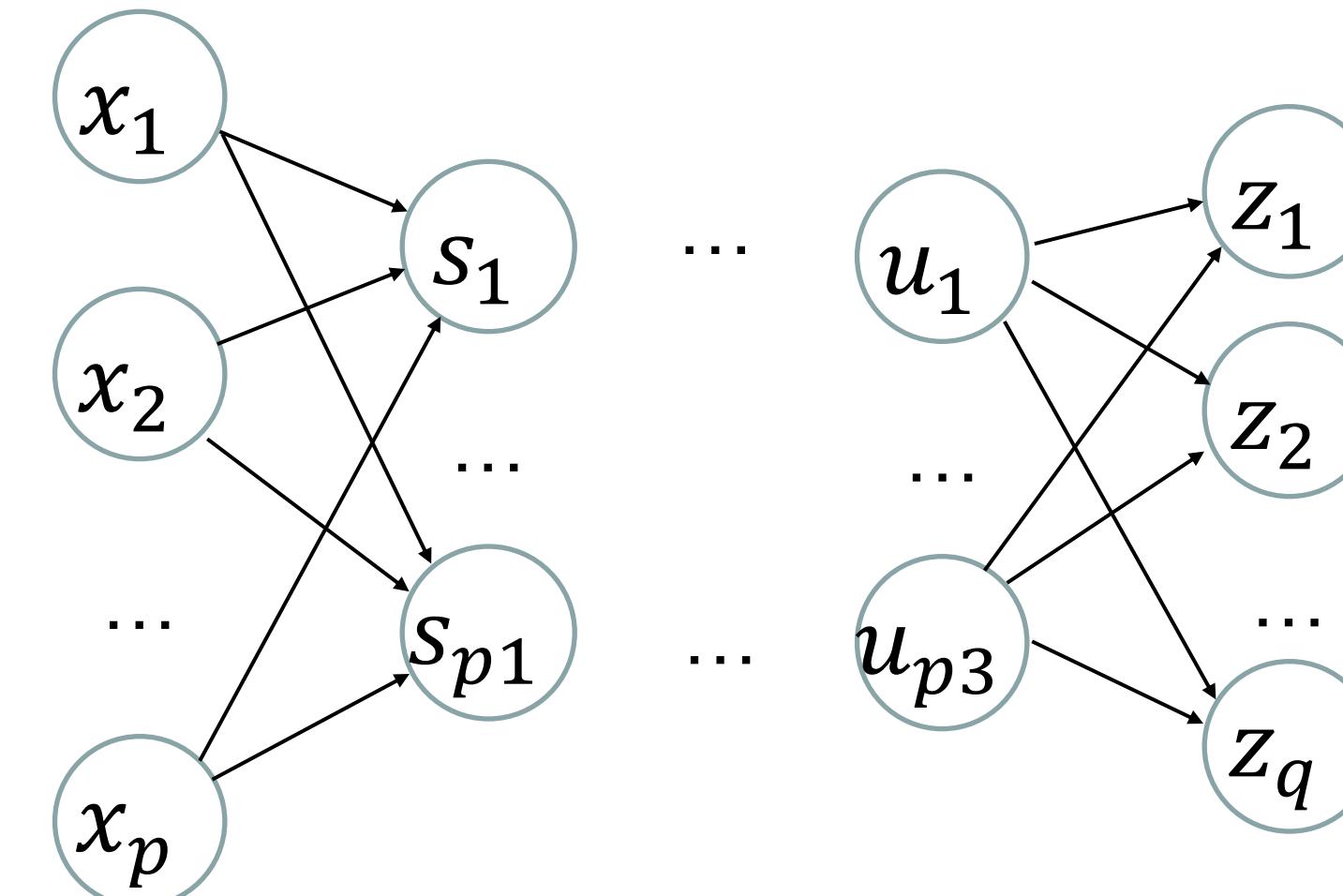
## Deep learning

Input →

Feature learning + classification

→ Output

## Deep neural network



many hidden layers  
9

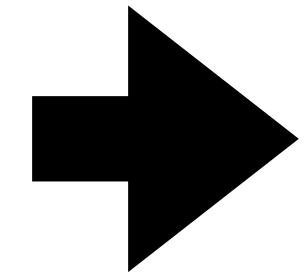
## Deep learning is everywhere

Near-human-level  
image classification  
speech recognition  
handwriting transcription  
autonomous driving

....

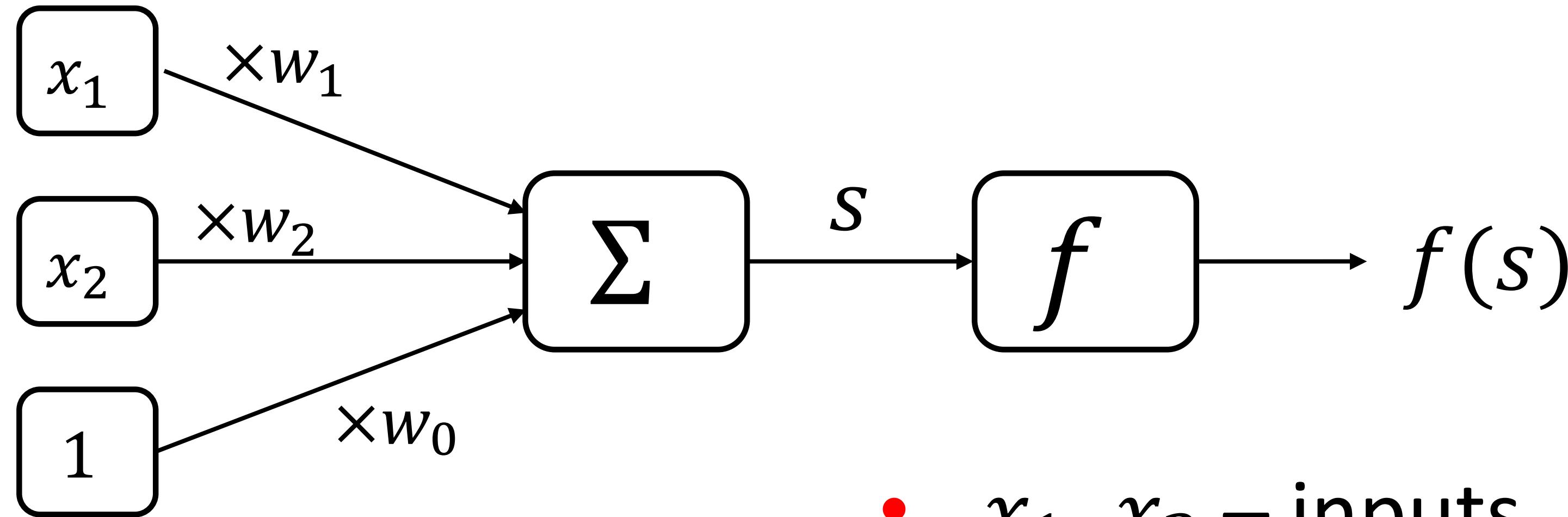
## Why now deep learning is popular?

- Large dataset
- Powerful computing resources
- Better algorithms:
  - Weight-initialization schemes
  - Optimisation schemes
  - Activation functions
  - ...



Network: deeper than deeper

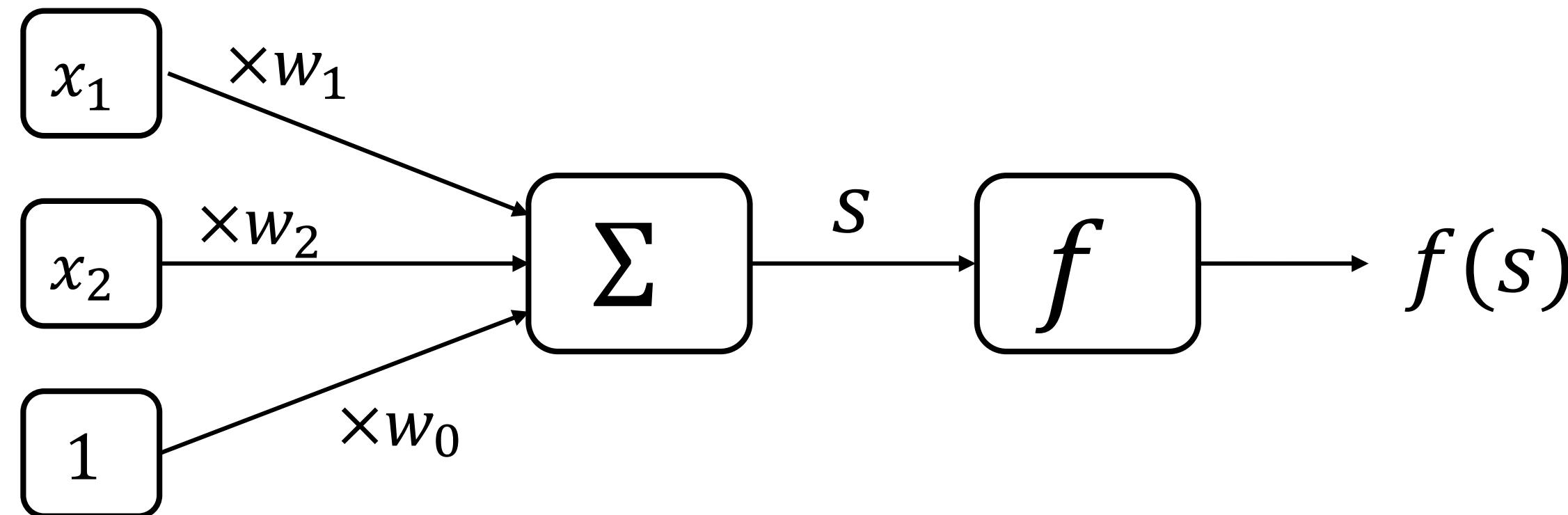
## Perceptron



$$s = \sum_{i=0}^m x_i w_i$$

- $x_1, x_2$  – inputs
- $w_1, w_2$  – synaptic weights
- $w_0$  – bias weight
- $f$  – activation function

## Perceptron is a linear binary classifier



*if  $s \geq 0$  :  $f(s)$  is positive class  
if  $s < 0$  :  $f(s)$  is negative class*

Step function

$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ 0, & \text{if } s < 0 \end{cases}$$

Sign function

$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ -1, & \text{if } s < 0 \end{cases}$$

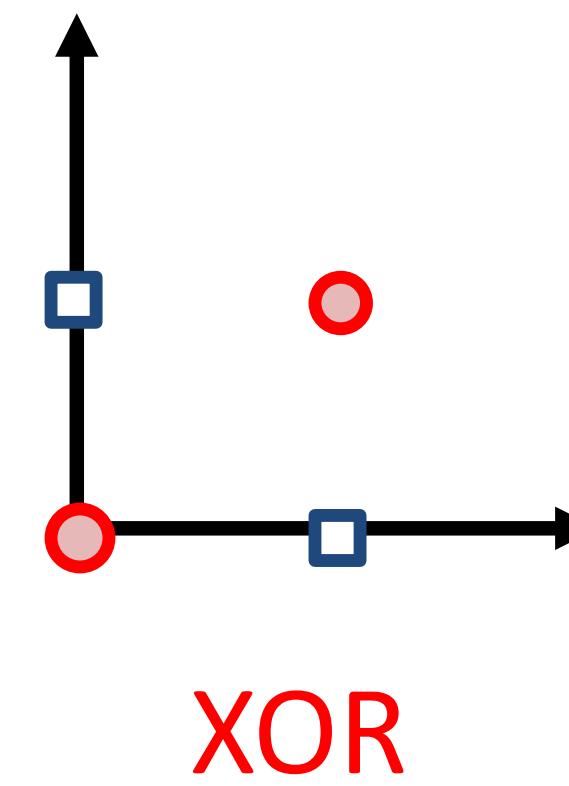
## Simple example

Exercise: find weights of a perceptron capable of perfect classification of the following dataset

$x_1$	$x_2$	$y$
0	0	Class B
0	1	Class B
1	0	Class B
1	1	Class A

## Limitations of perceptron learning

- If the data is linearly separable, the perceptron training algorithm will converge to a correct solution
  - \* It will converge to some solution (separating boundary), one of infinitely many possible ← bad!
- However, if the data is not linearly separable, the training will fail completely rather than give some approximate solution
  - \* Ugly 😞

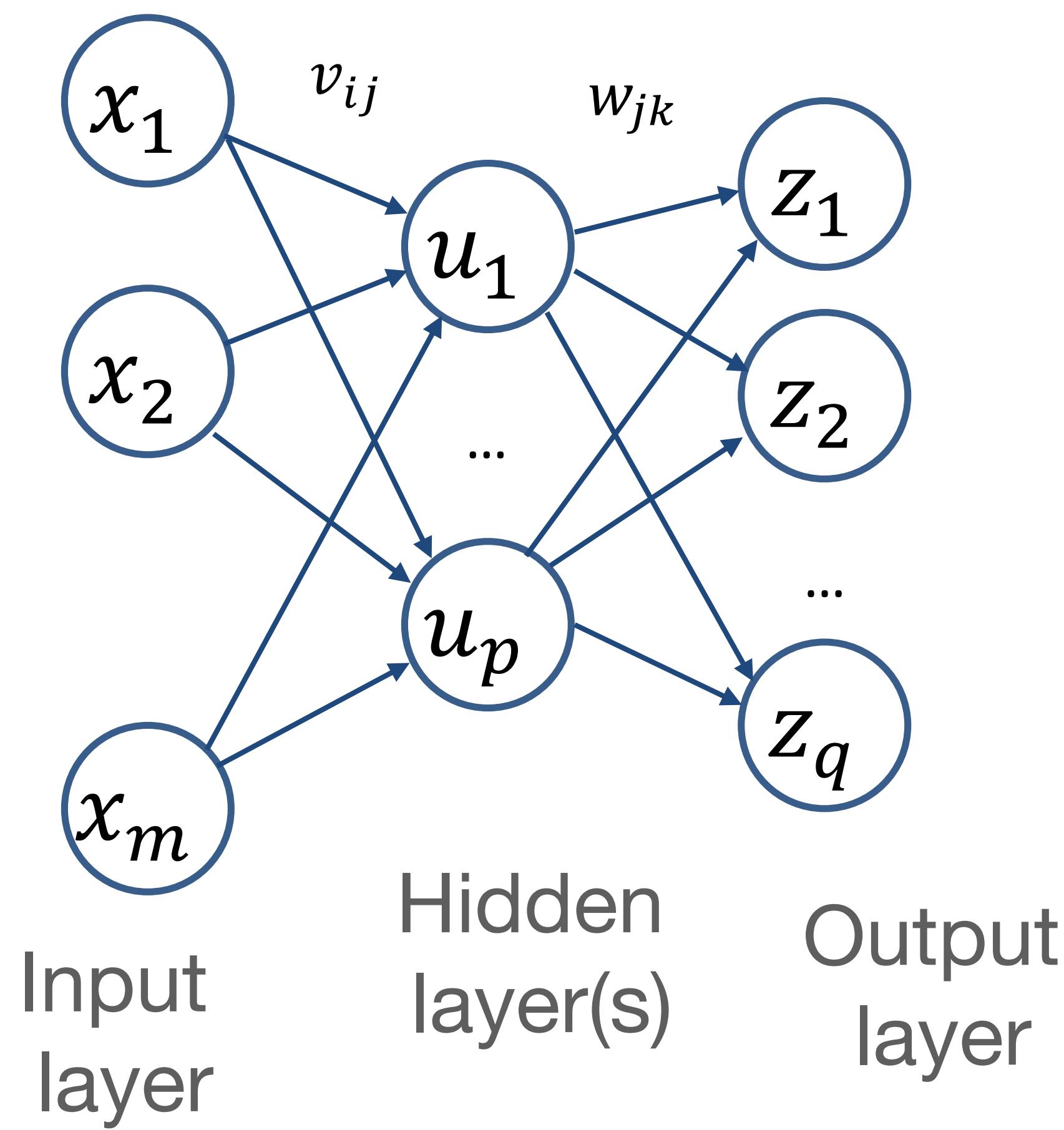


## Multi-layer perceptron: function composition

$$u_j = g(r_j)$$

$$r_j = \sum_{i=0}^m x_i v_{ij}$$

$g(x)$ : activation function



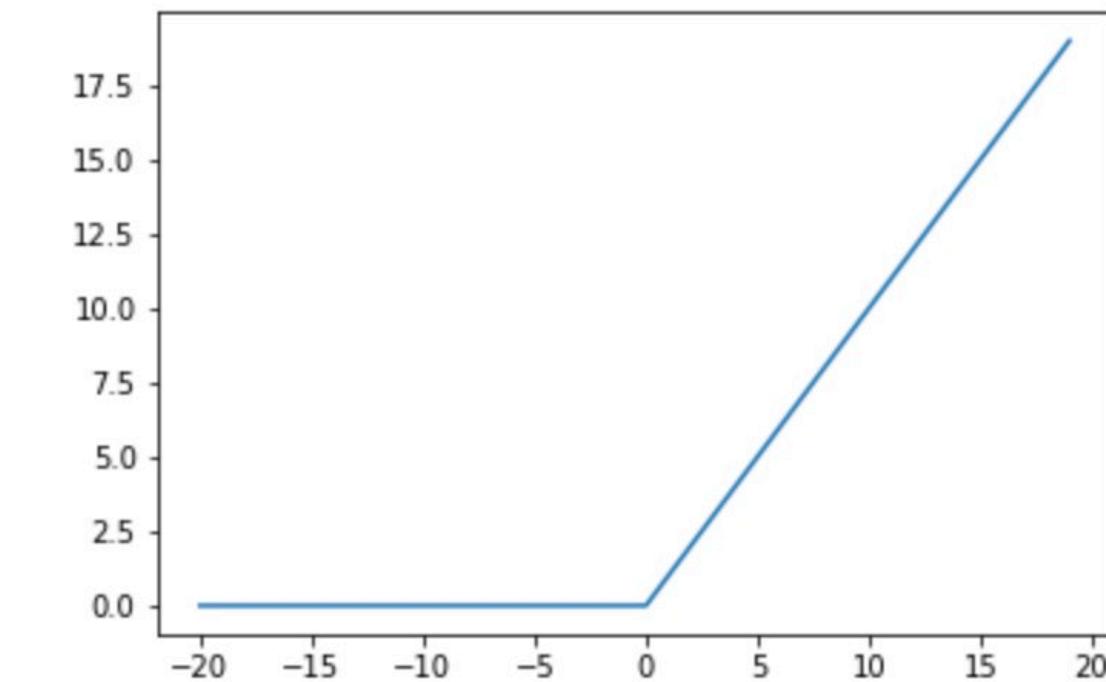
$$z_k = h(s_k)$$

$$s_k = \sum_{j=0}^p u_j w_{jk}$$

$h(x)$ : activation function

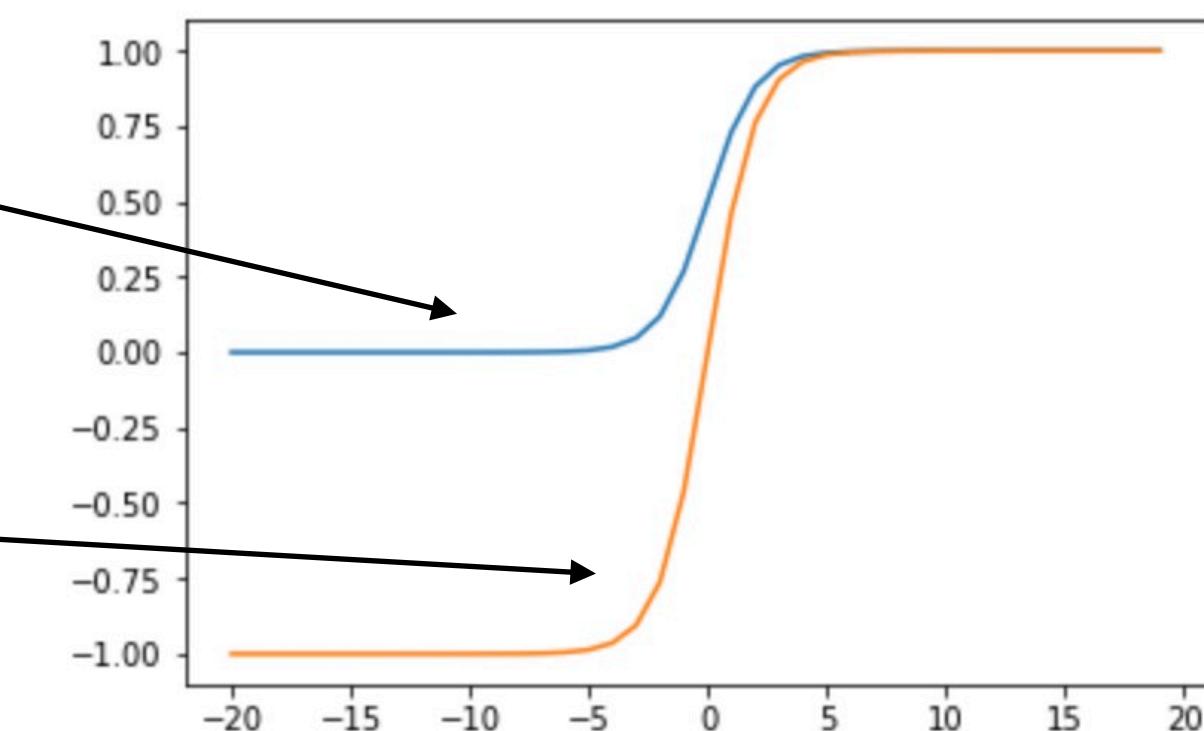
## Common non-linear activation functions

Relu:  $f(x) = \max(0, x)$



Sigmoid:  $f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$

TanH  $f(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{2e^{2x}}{e^{2x} + 1} - 1$



Softmax:  $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^C e^{x_j}} \quad i = 1, \dots, C$   
(Multi-class classification)

## How to train your ~~dragon~~ network?



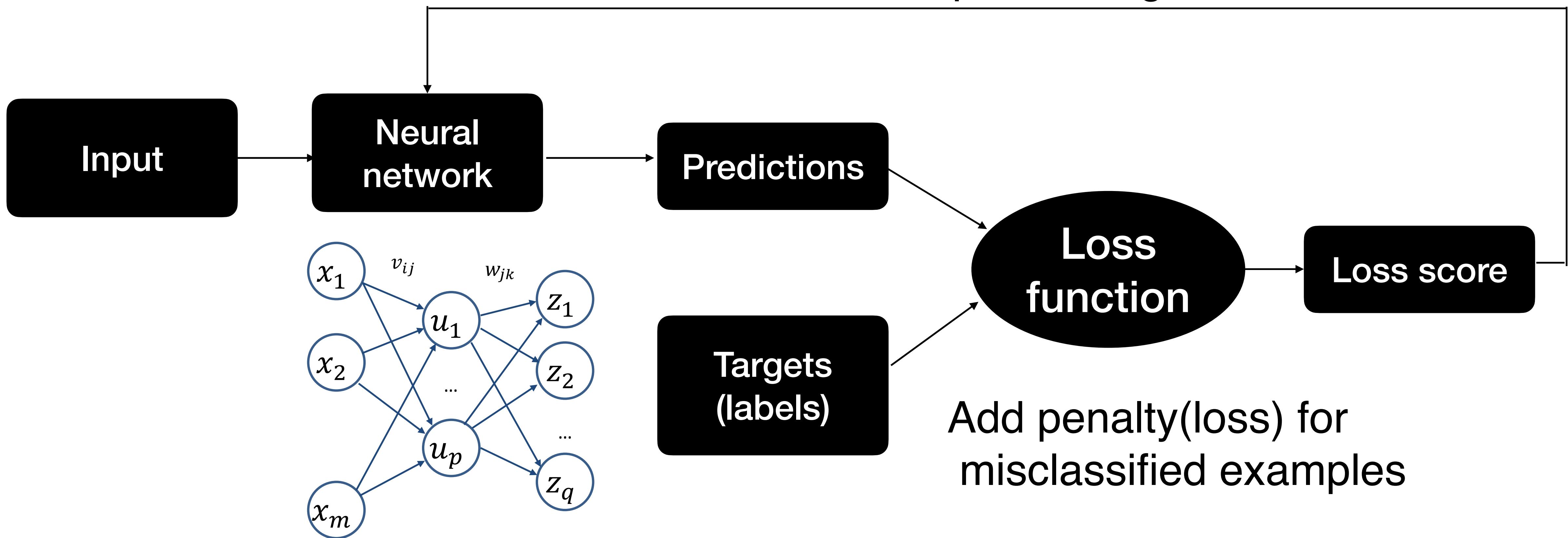
Adapted from Movie Poster from  
Flickr user jdxyw (CC BY-SA 2.0)<sup>18</sup>

“Training”: adjust weights to minimise loss.

How?

Training loop:

Update weights



## Derivative

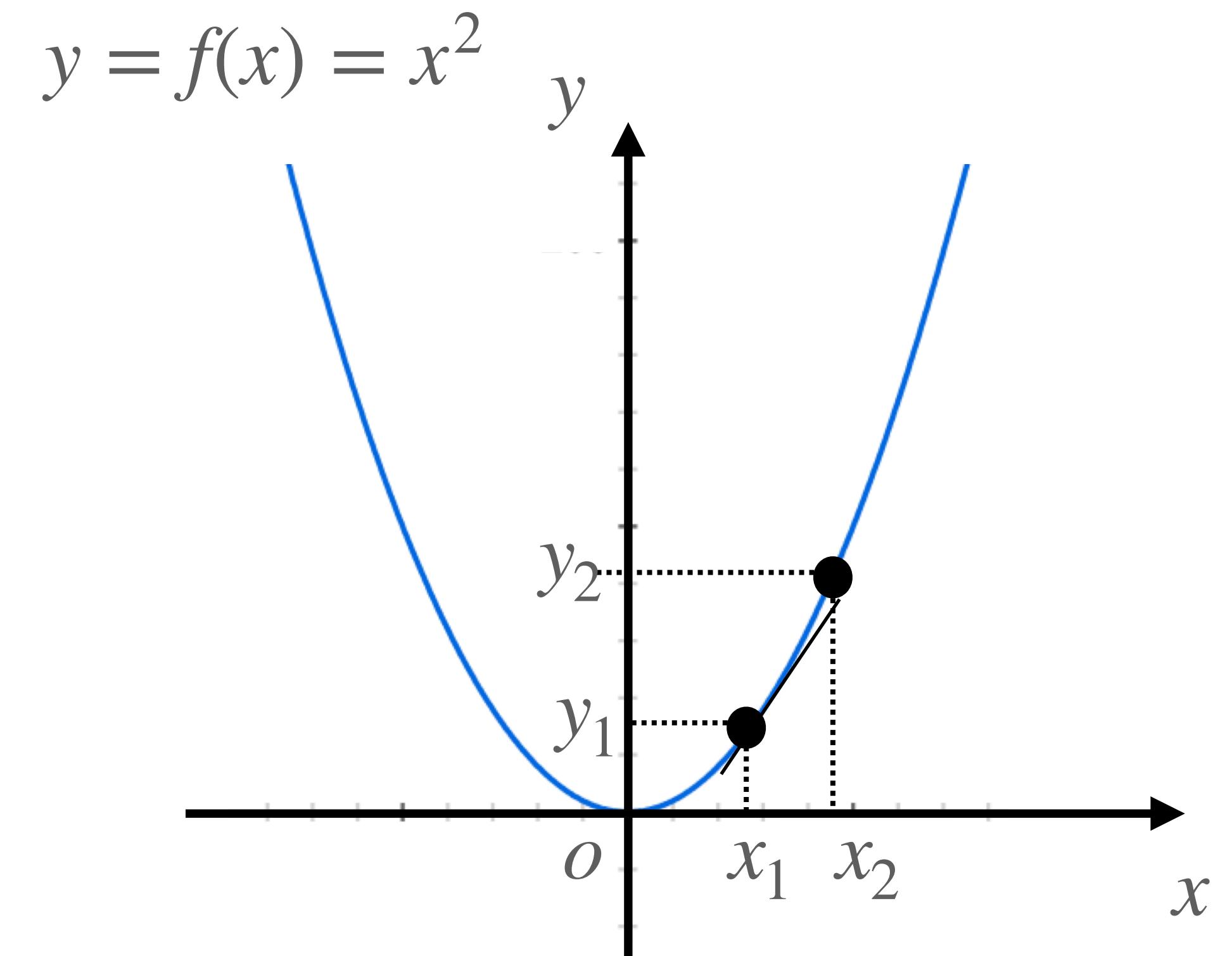
Input  $x$ , output  $y$ :  $f(x)$

$$\Delta x \rightarrow 0 : f(x_1 + \Delta x) - f(x_1) = a\Delta x$$

$a$  : *rate of change (derivative) at  $x_1$*

derivative: a vector from origin.

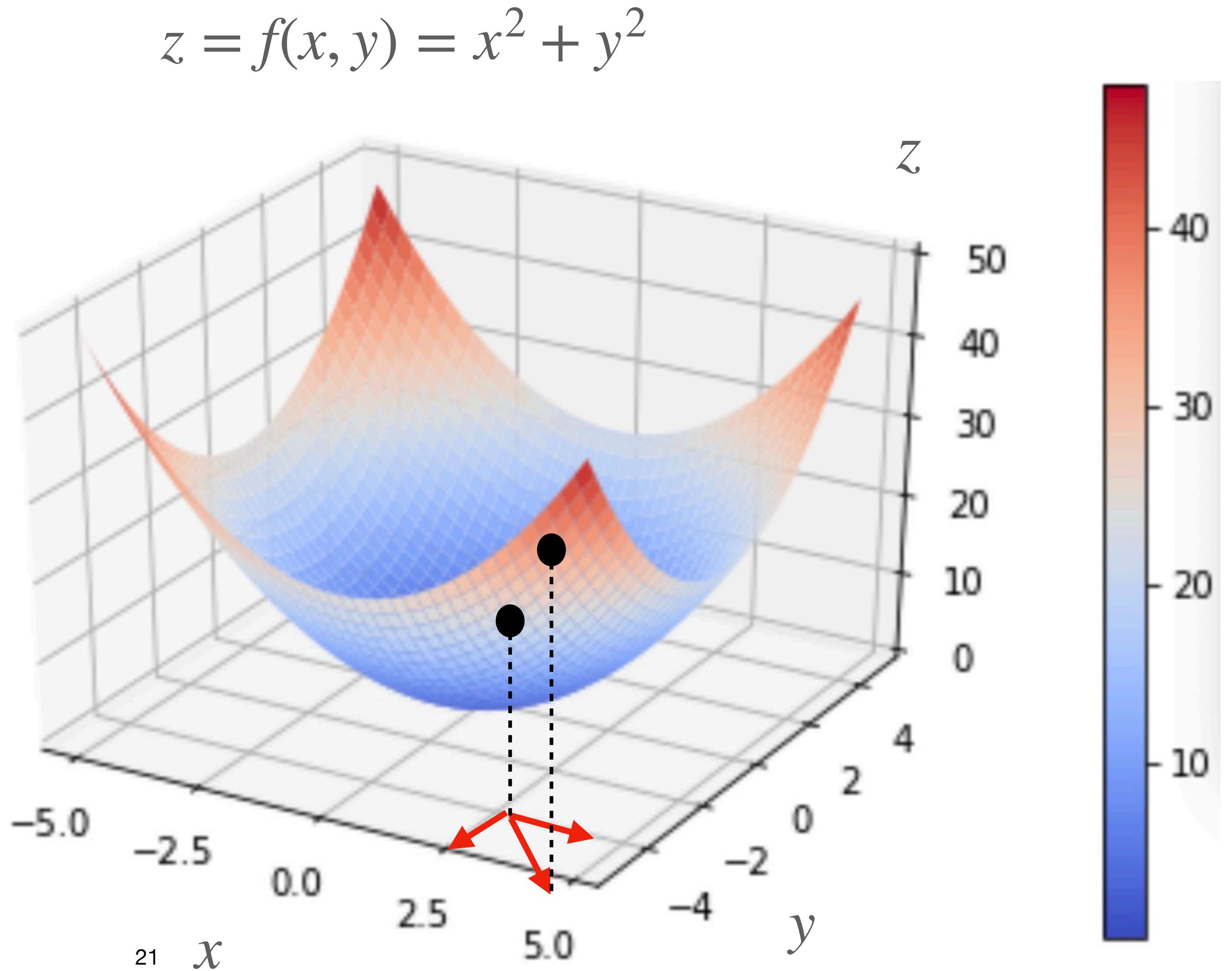
Move  $x$  along OPPOSITE direction of  
the vector: decrease  $f(x)$



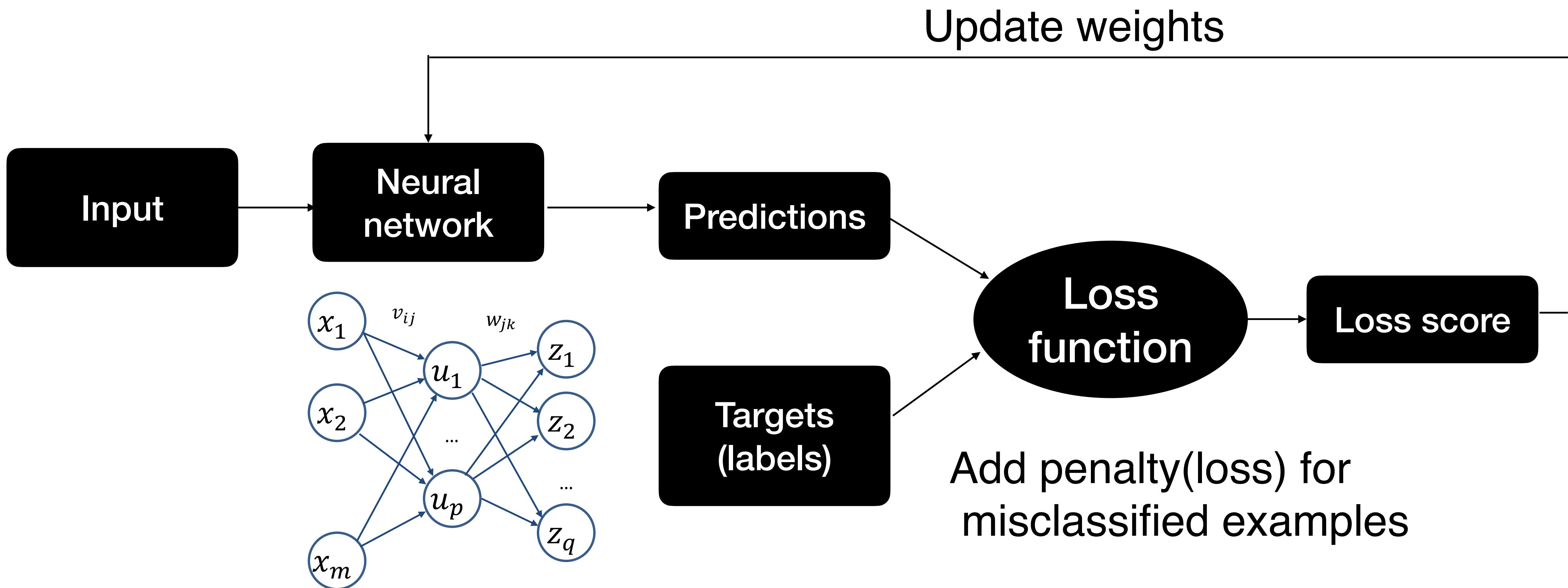
## Gradient

$$G = \left[ \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \right]$$

Move input  $(x, y)$  along  
OPPOSITE direction of  
gradient vector:  
decrease output  $f(x, y)$



## Loss = function (weights)



## Loss = function (weights)

To reduce loss, update weights:

$$w_i^{new} = w_i^{old} - \eta * \Delta L(w_i)$$

$$\Delta L(w_i) = \frac{\partial L}{\partial w_i}$$

$\eta$  : learning rate

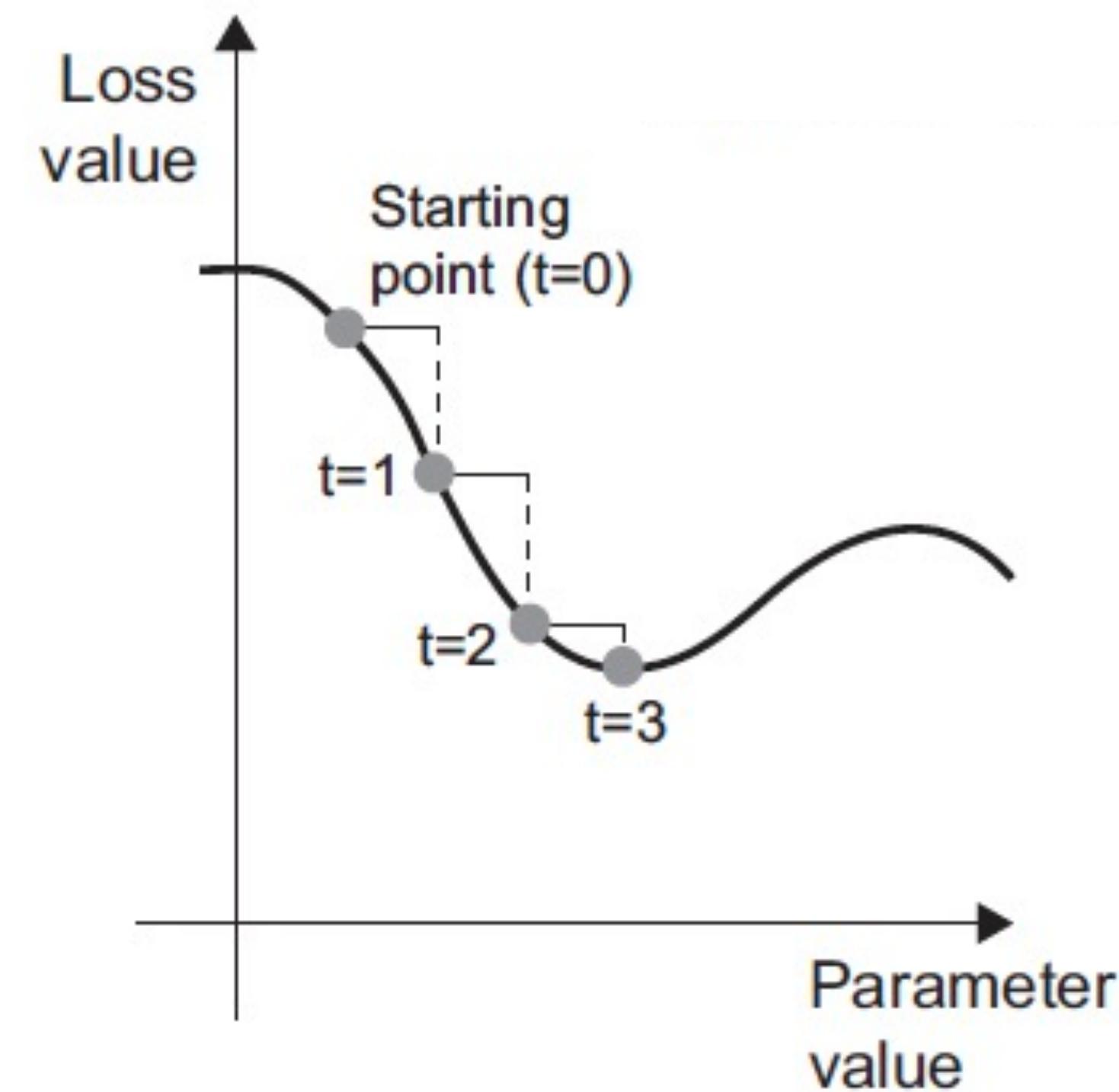


Figure 2.11 in Deep learning with python by Francois Chollet

## Loss = function (weights)

$\eta$  : learning rate

Small  $\eta$ : local optimal value

Large  $\eta$ : random location

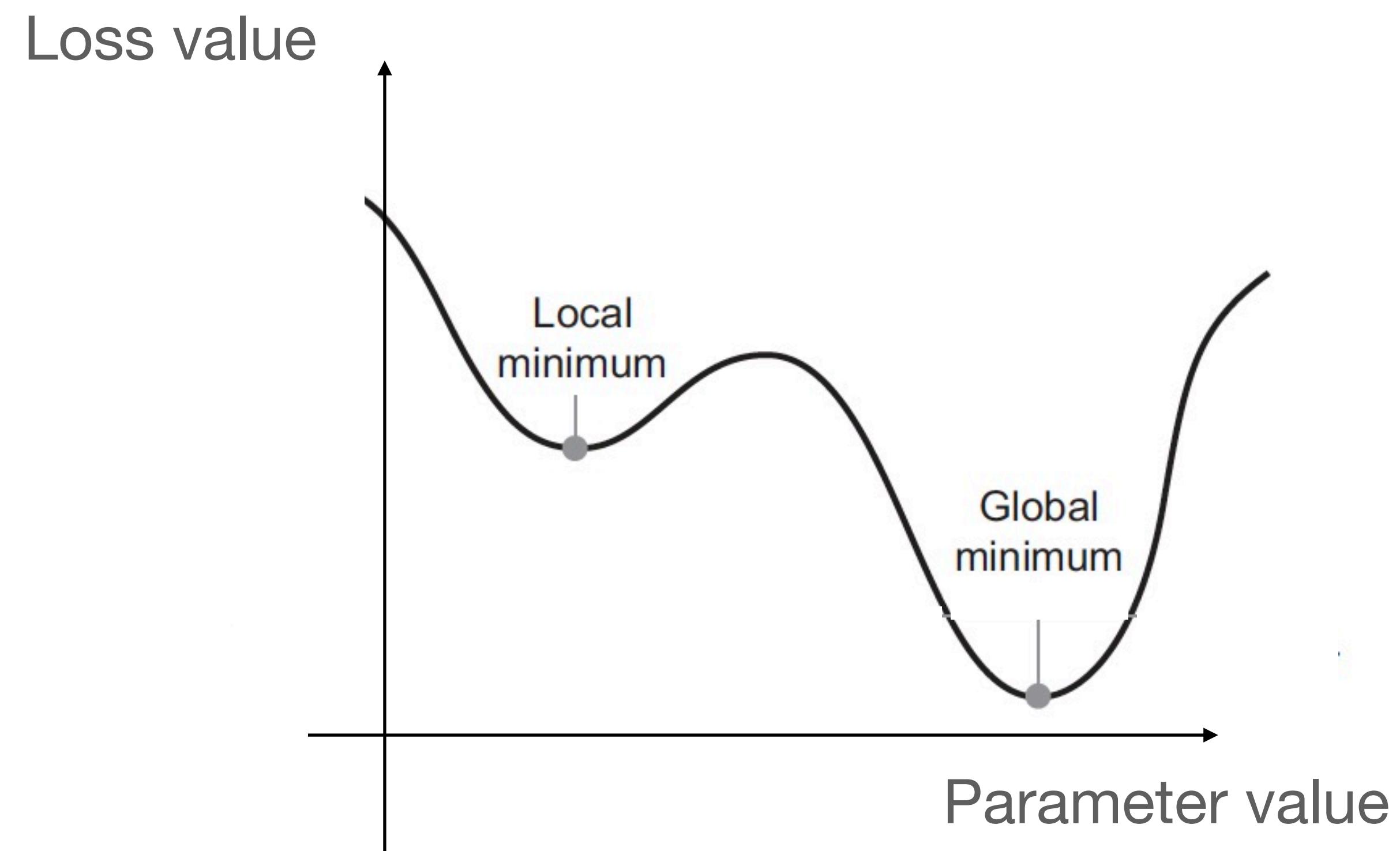


Figure 2.13 in Deep learning with python by Francois Chollet

## Gradient descent Algorithm

- Randomly shuffle/split all training examples in  $B$  batches
- Choose initial  $\theta^{(1)}$
- For  $i$  from 1 to  $T$
- For  $j$  from 1 to  $B$
- Do gradient descent update using data from batch  $j$
- Advantage of such an approach: computational feasibility for large datasets

Iterations over the entire dataset are called epochs

## Stochastic gradient descent for perceptron

Choose initial guess  $\mathbf{w}^{(0)}, k = 0$

For  $i$  from 1 to  $T$  (epochs)

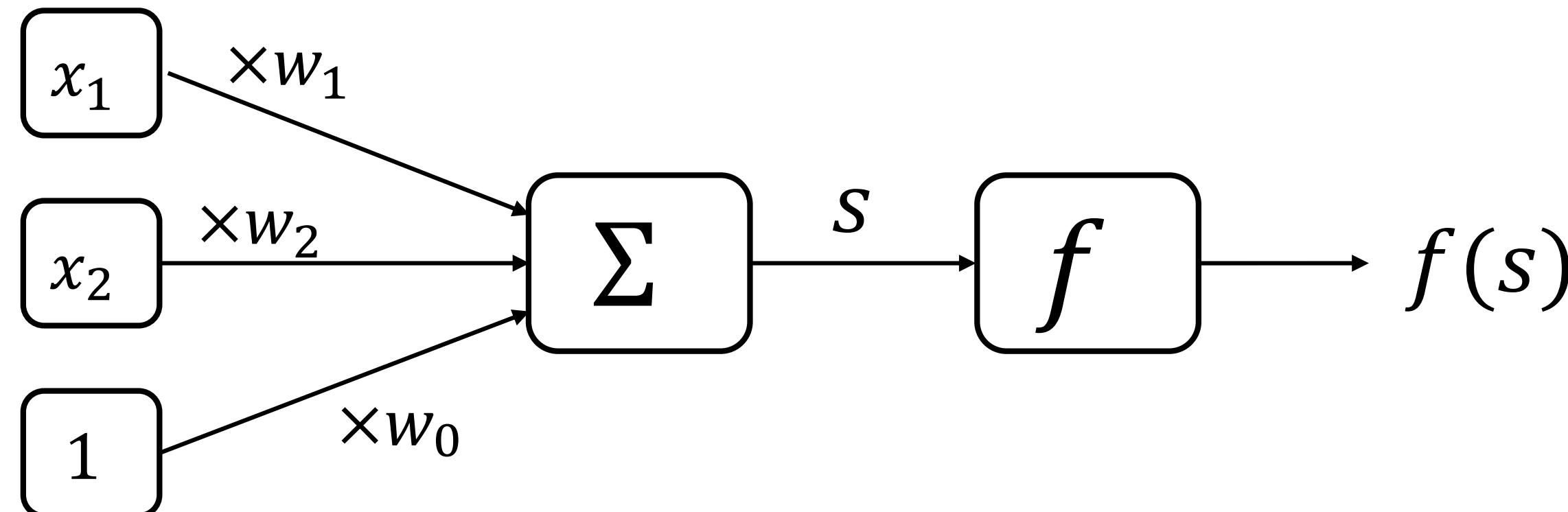
For  $j$  from 1 to  $N$  (training examples)

Consider example  $\{\mathbf{x}_j, y_j\}$

Update\*:  $\mathbf{w}^{(k++)} = \mathbf{w}^{(k)} - \eta \nabla L(\mathbf{w}^{(k)})$

## Simple example: Perceptron Model

Encode class: A: +1, B:-1

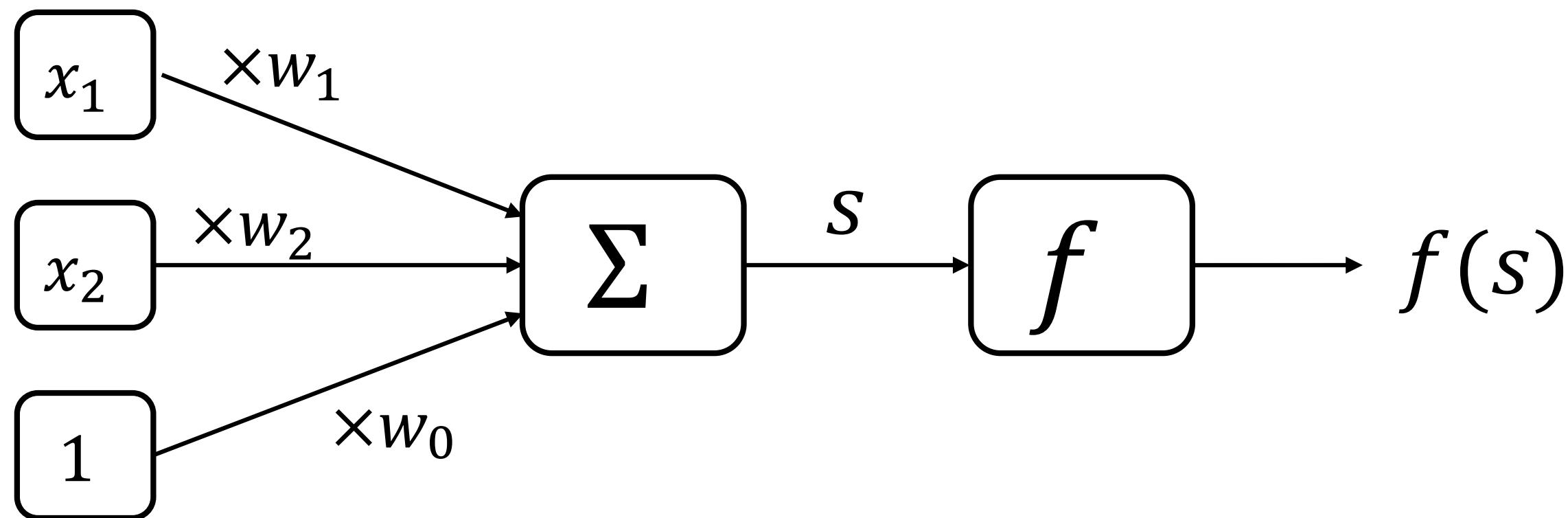


$\text{if } s \geq 0 : \text{ prediction } f(s) = 1$   
 $\text{if } s < 0 : \text{ prediction } f(s) = -1$

$$\text{where } s = \sum_{i=0}^m x_i w_i$$

$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1

## Simple example: Loss



$$L(s, y) = \max(0, -ys)$$

$$= \max(0, -y \cdot \sum_{i=0}^m x_i w_i)$$

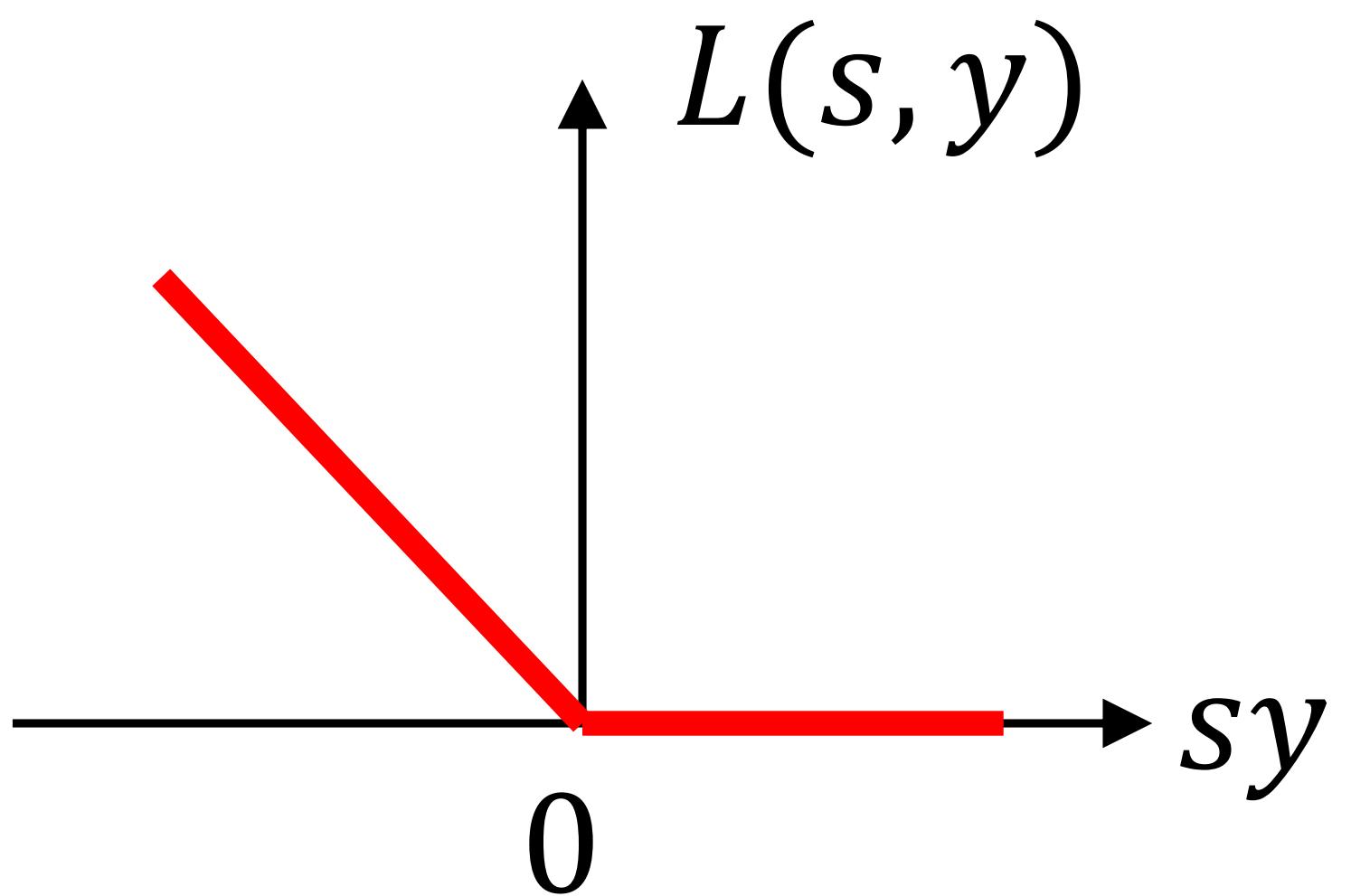
$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1

## Simple example: gradient

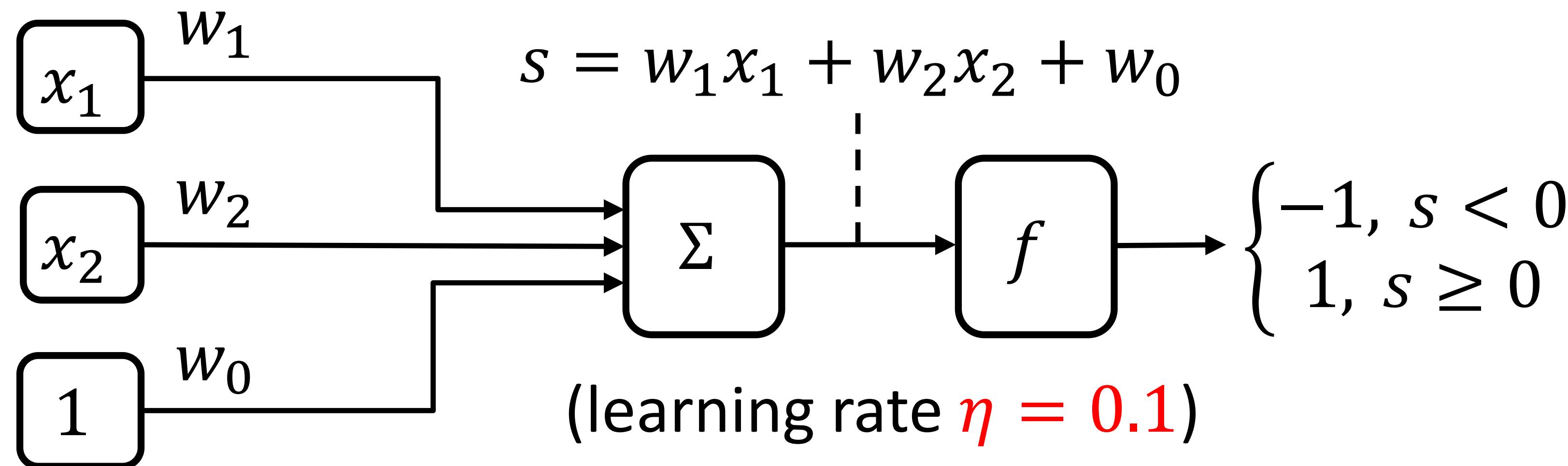
$$L(w) = \max(0, -ys) = \max(0, -y \sum_{i=0}^m x_i w_i)$$

$$\frac{\partial L}{\partial w_i} = -yx_i \text{ when } sy < 0$$

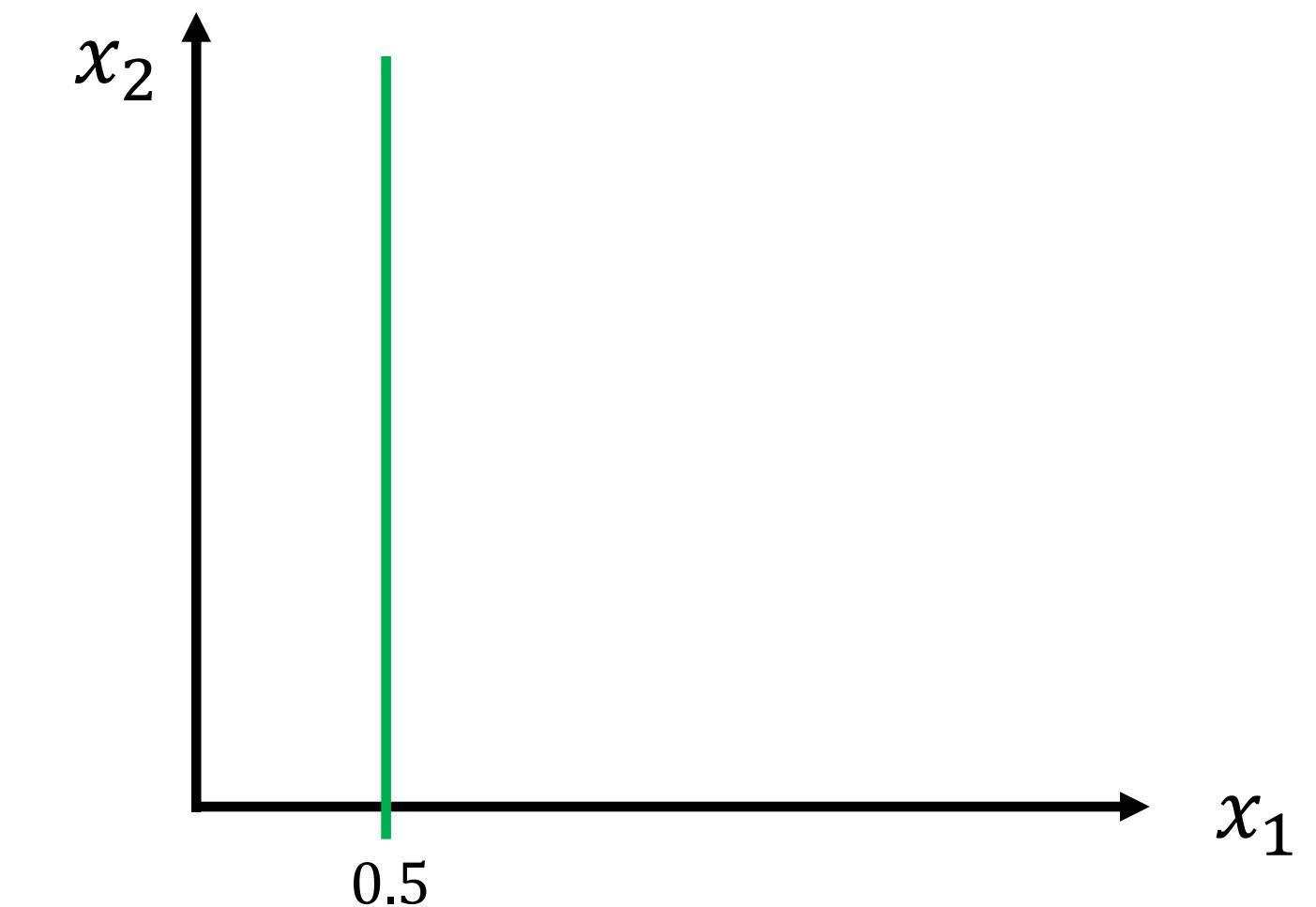
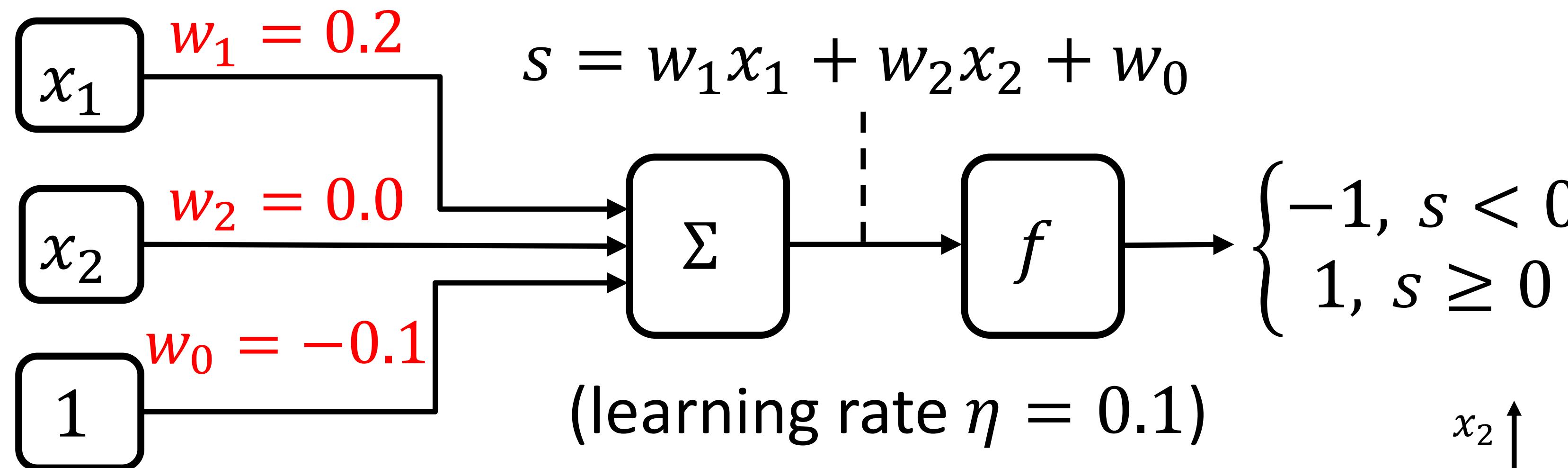
$$\frac{\partial L}{\partial w_i} = 0 \text{ when } sy > 0$$



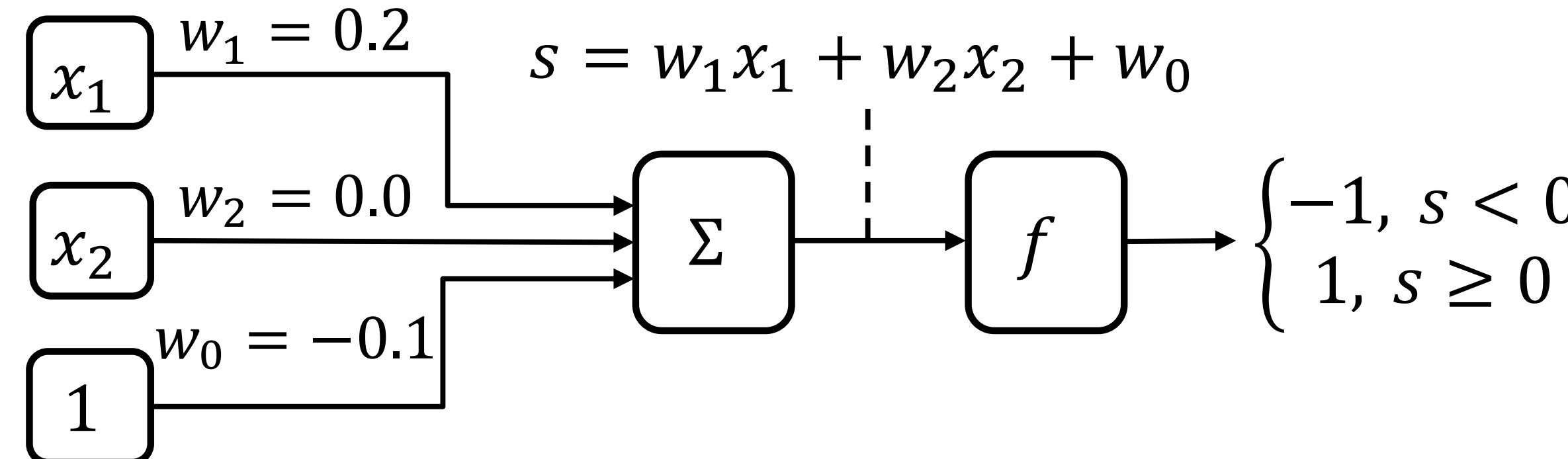
## Learn perceptron on the simple example: basic setup



## Learn perceptron on the simple example: Start with random weights



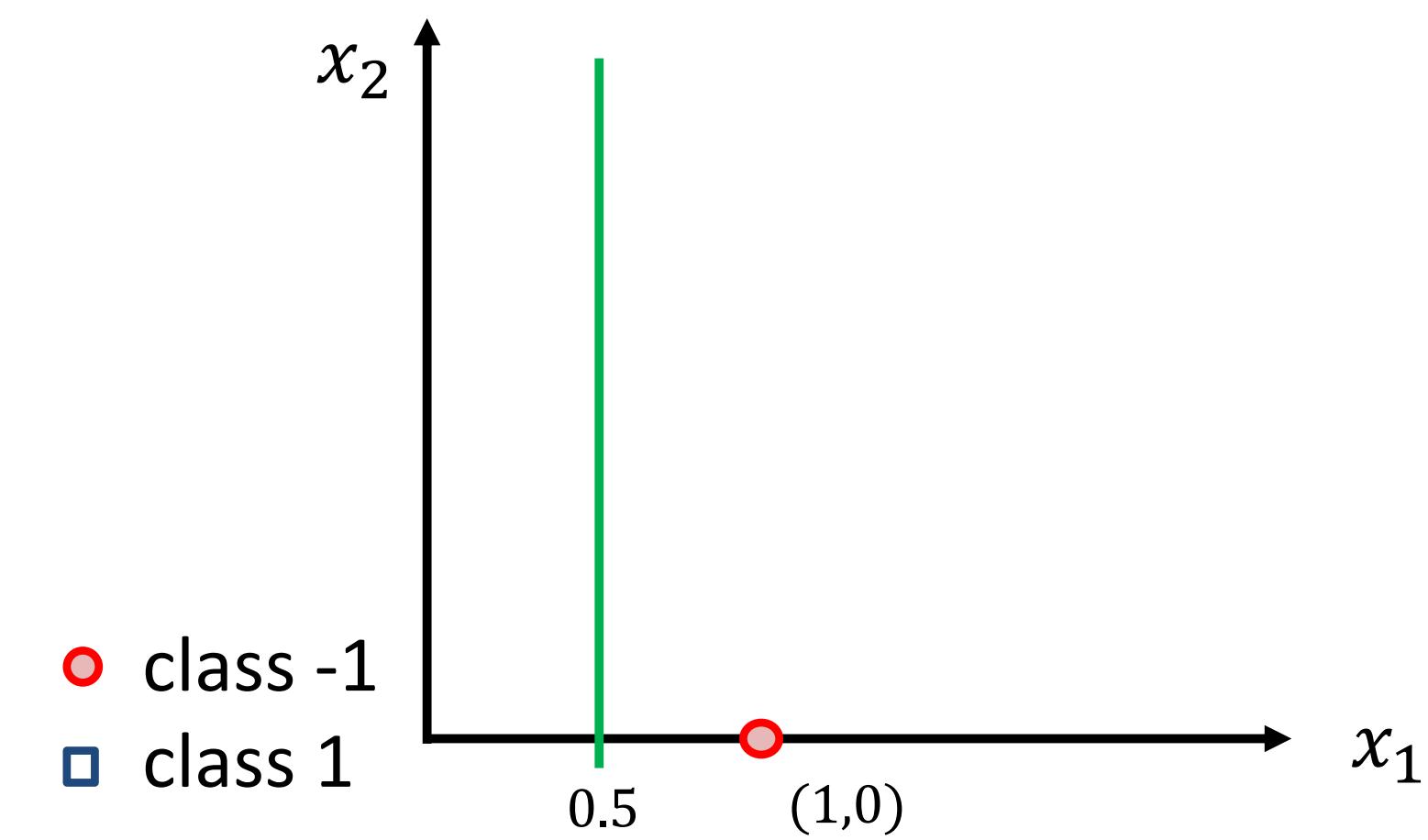
## Learn perceptron on the simple example: epoch 1, data point 1



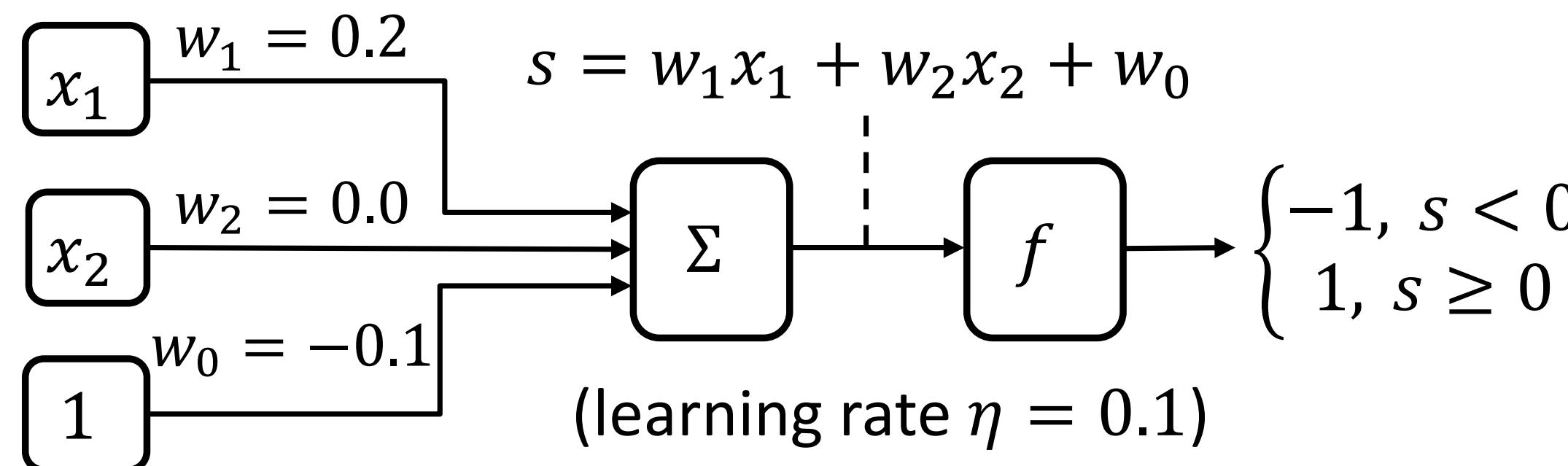
$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1

Prediction  $s$  on  $(1,0)$ :

$$w_1 \times 1 + w_2 \times 0 + w_0 \times 1 = 0.1 > 0$$



## Learn perceptron on the simple example: epoch 1, data point 1



$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1

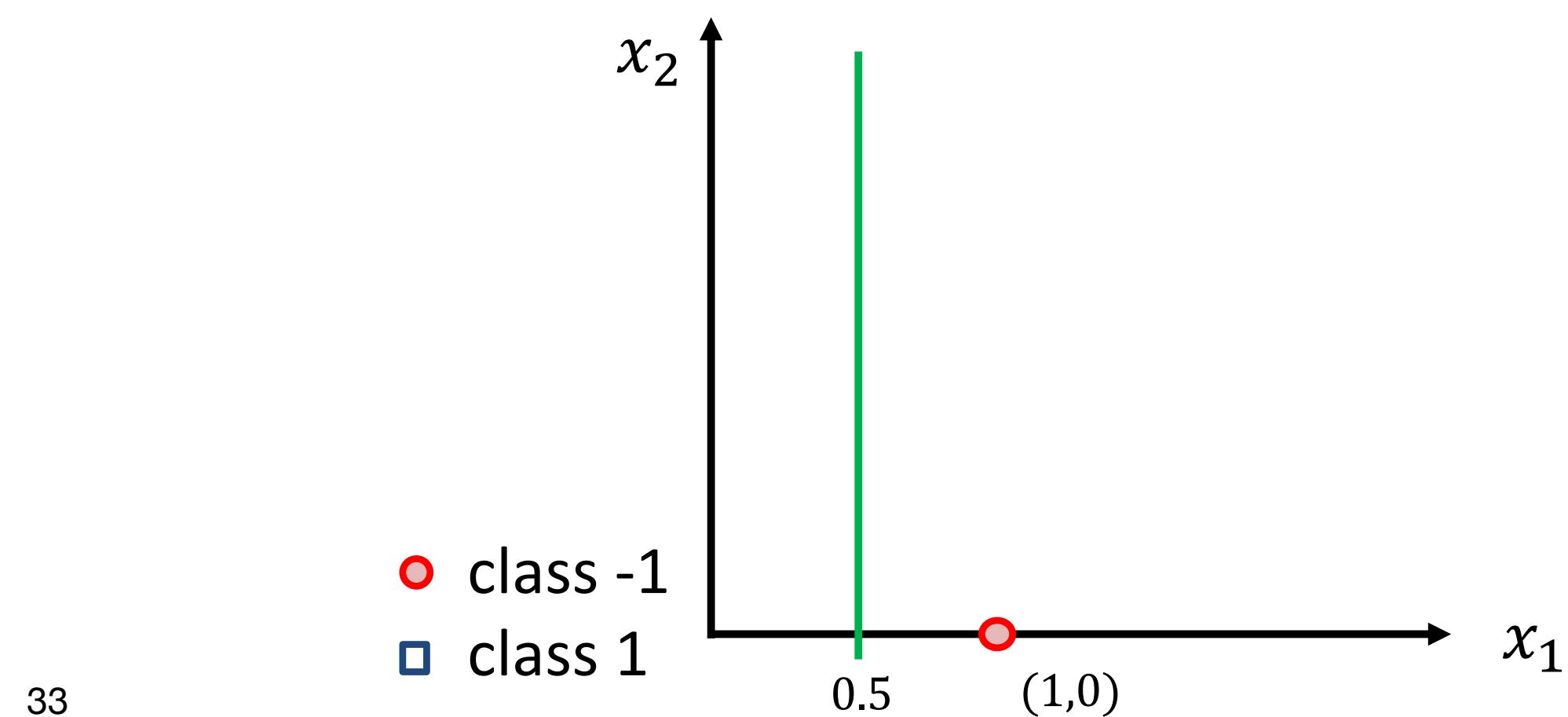
Gradient on (1,0):

$$s > 0, y = -1, sy < 0: \quad \frac{\partial L}{\partial w_i} = -yx_i$$

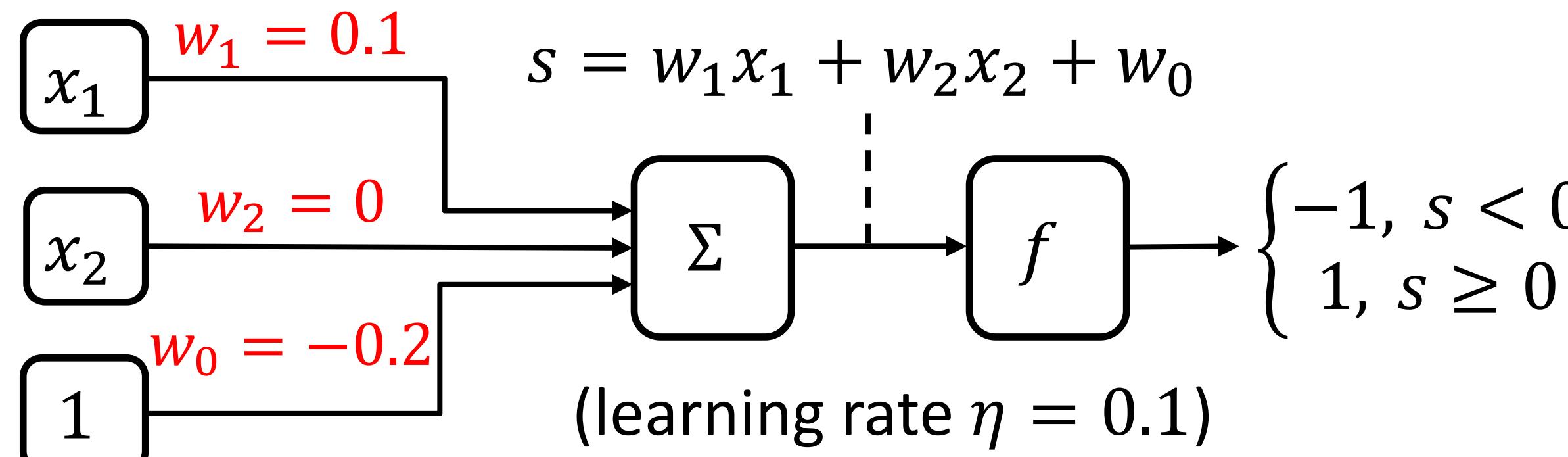
$$w_1 \leftarrow w_1 - \eta \cdot 1 = 0.1$$

$$w_2 \leftarrow w_2 - \eta \cdot 0 = 0$$

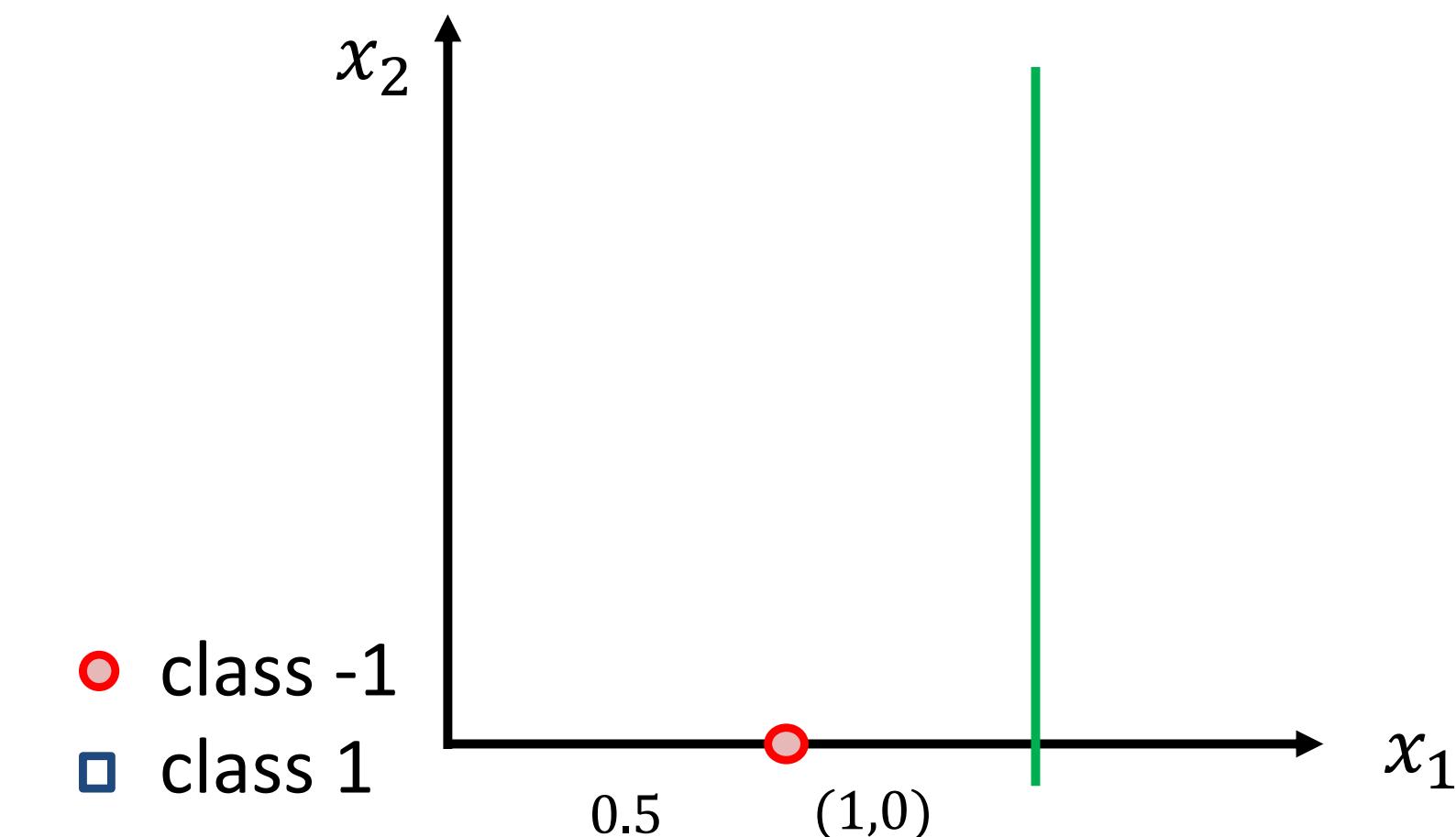
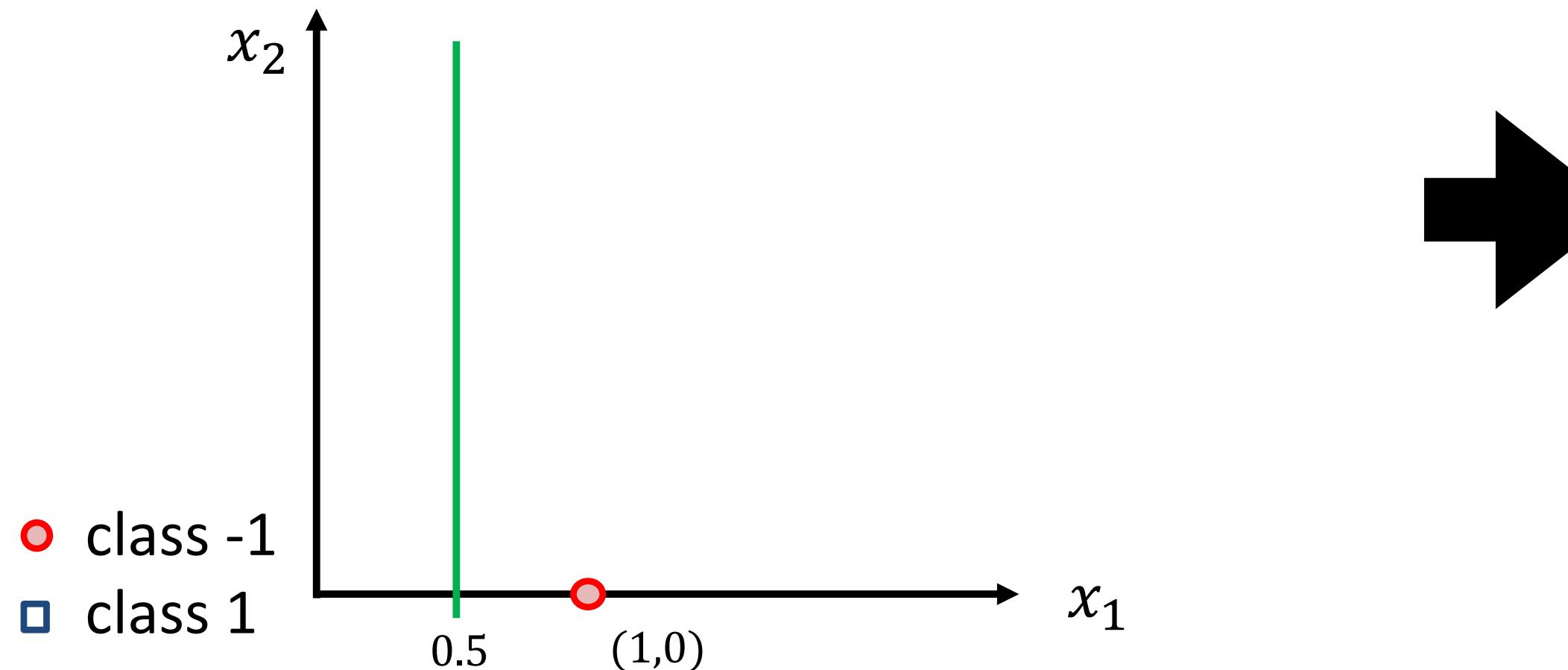
$$w_0 \leftarrow w_0 - \eta \cdot 1 = -0.2$$



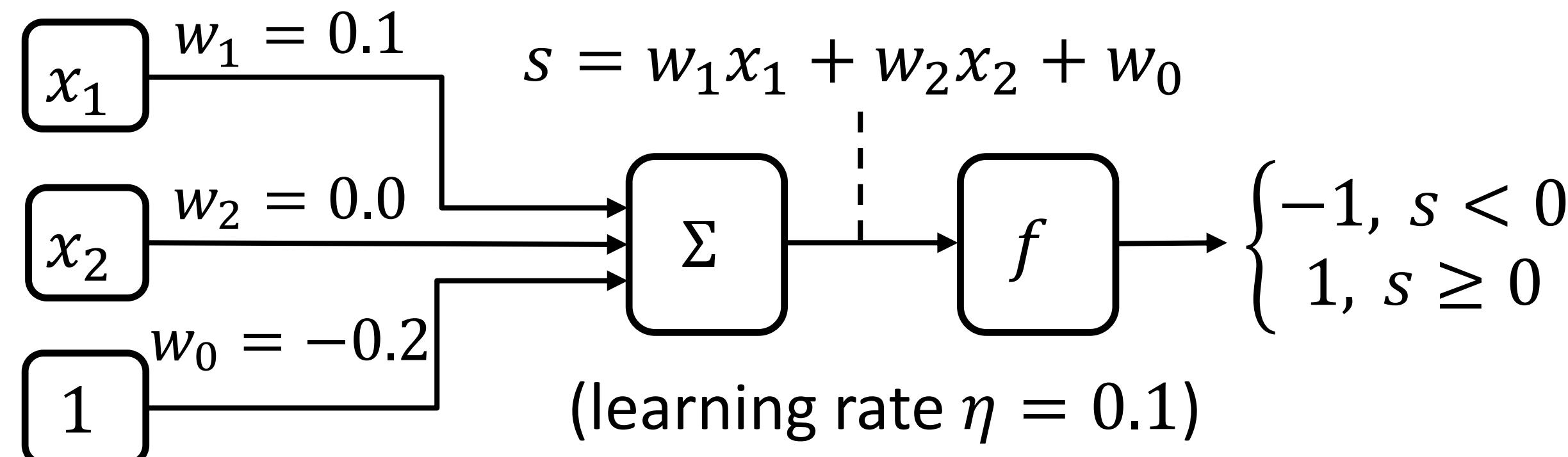
## Learn perceptron on the simple example: update weights



$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1



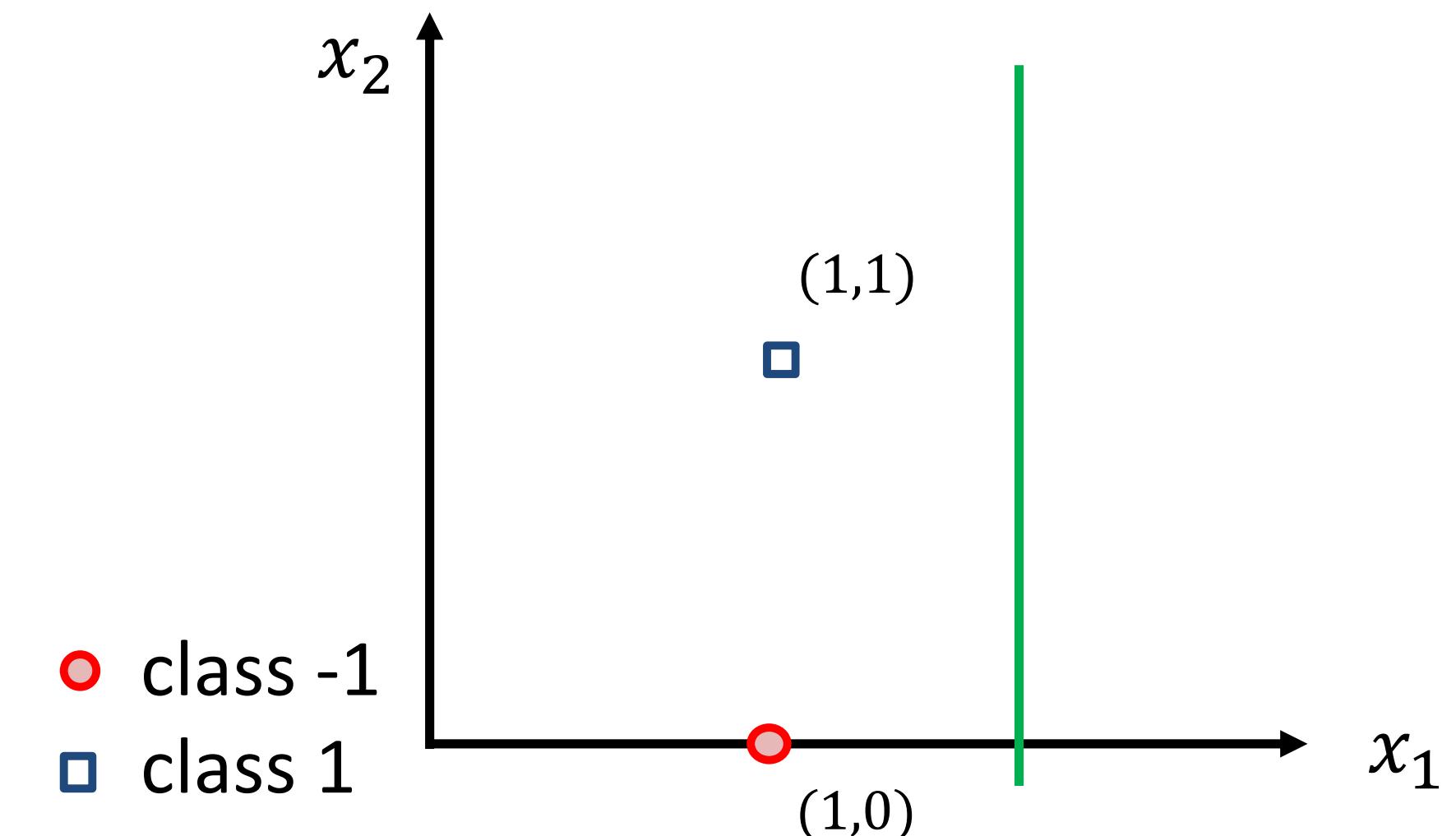
## Learn perceptron on the simple example: epoch 1, data point 2



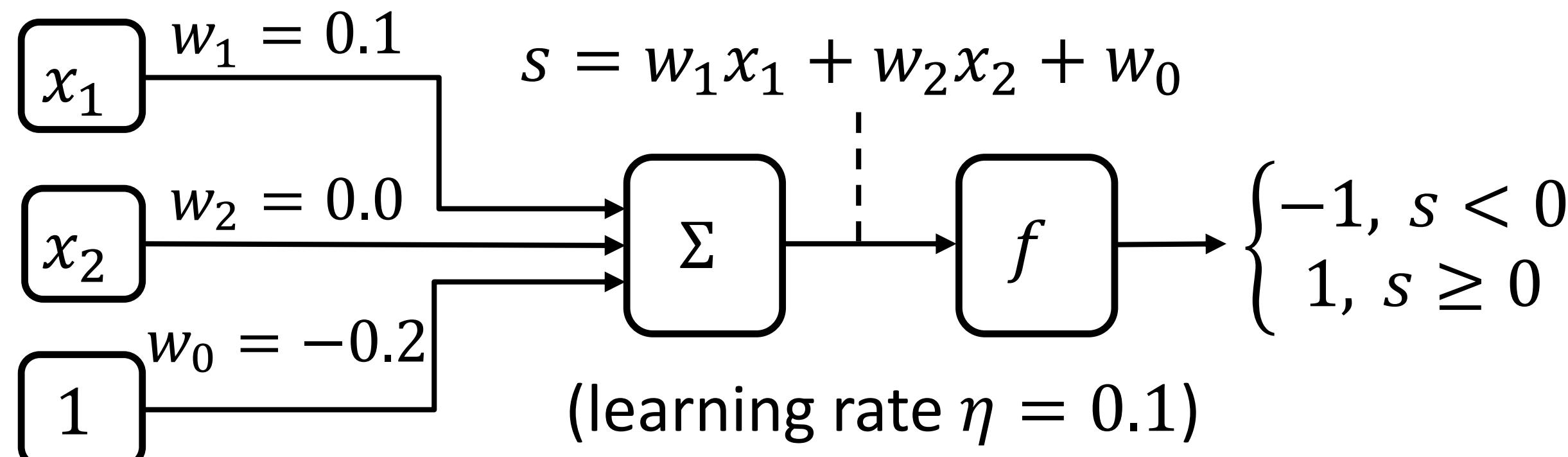
$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1

Prediction  $s$  on  $(1,1)$ :

$$w_1 \times 1 + w_2 \times 1 + w_0 \times 1 = -0.1 < 0$$



## Learn perceptron on the simple example: epoch 1, data point 2



$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1

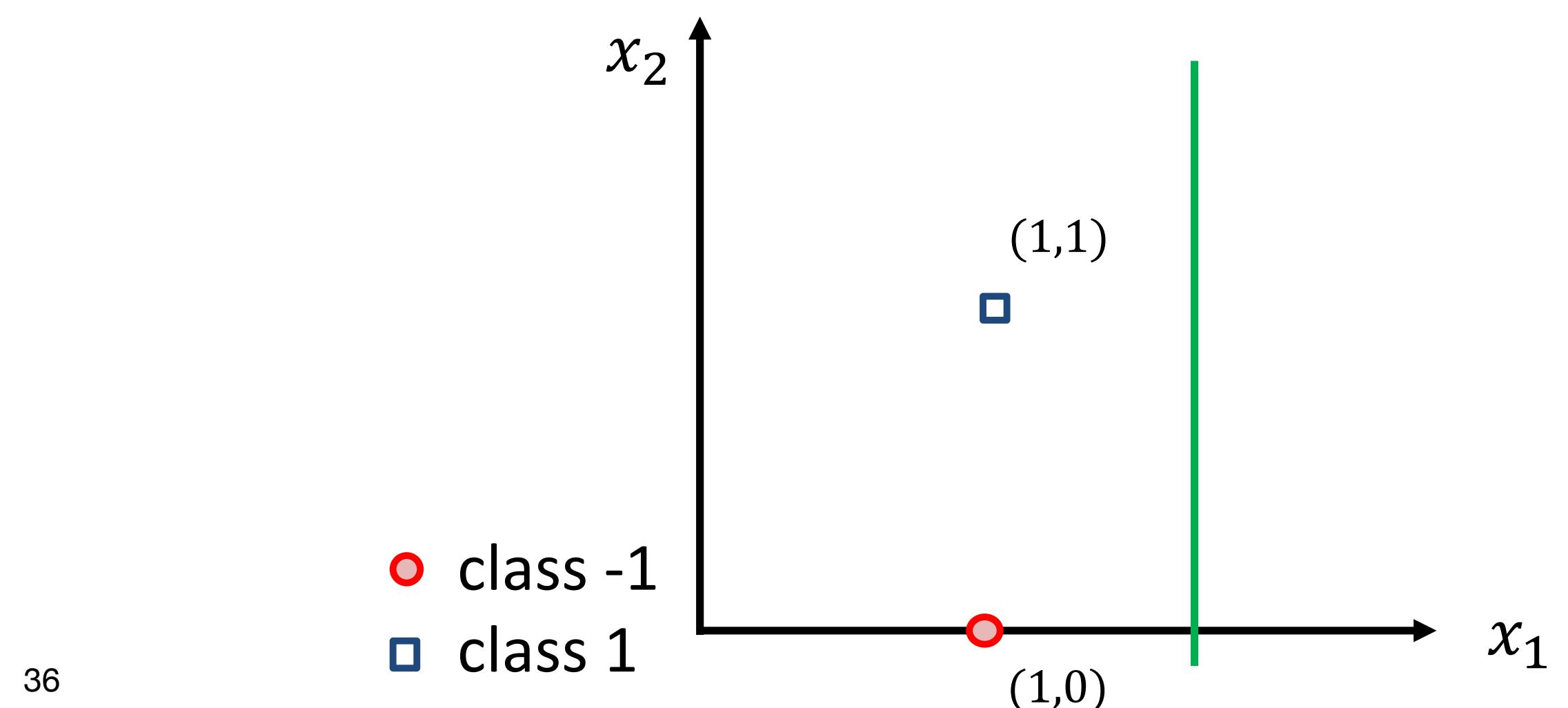
Gradient on (1,1):

$$s < 0, y=1, sy < 0: \quad \frac{\partial L}{\partial w_i} = -yx_i$$

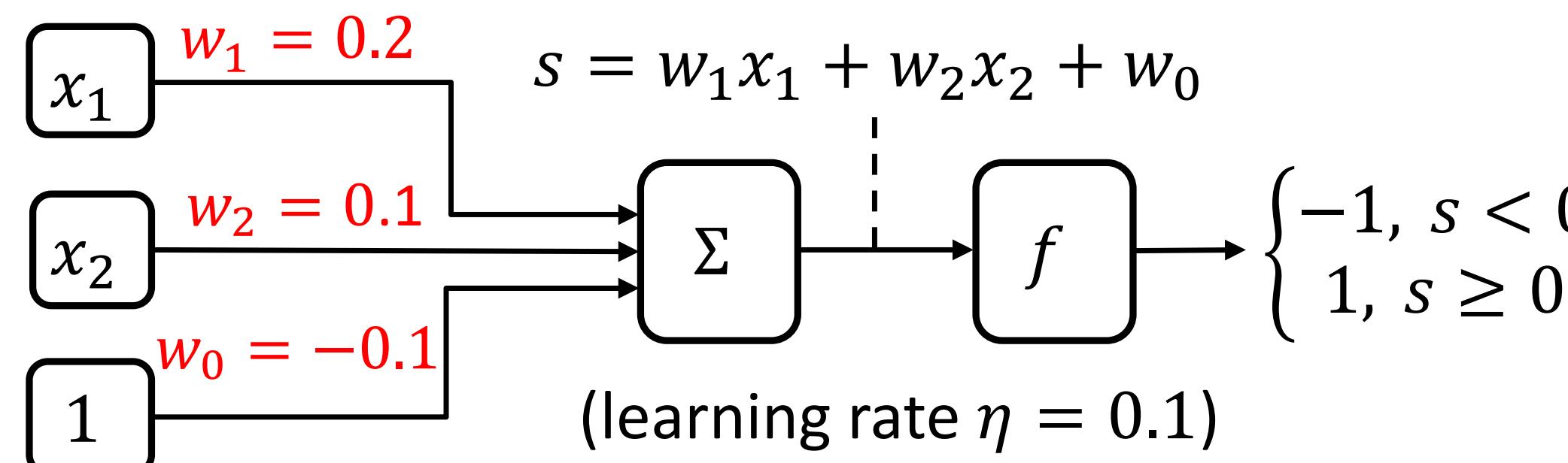
$$w_1 \leftarrow w_1 - \eta \cdot (-1) = 0.2$$

$$w_2 \leftarrow w_2 - \eta \cdot (-1) = 0.1$$

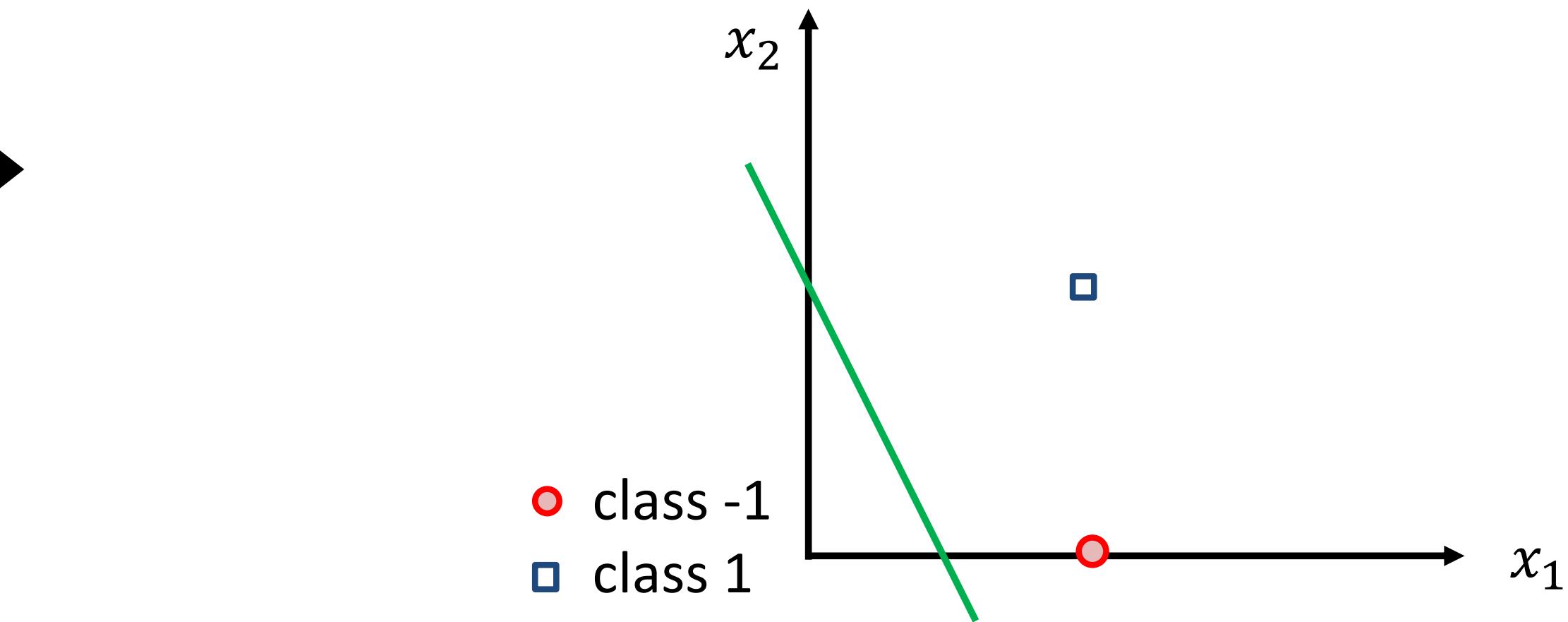
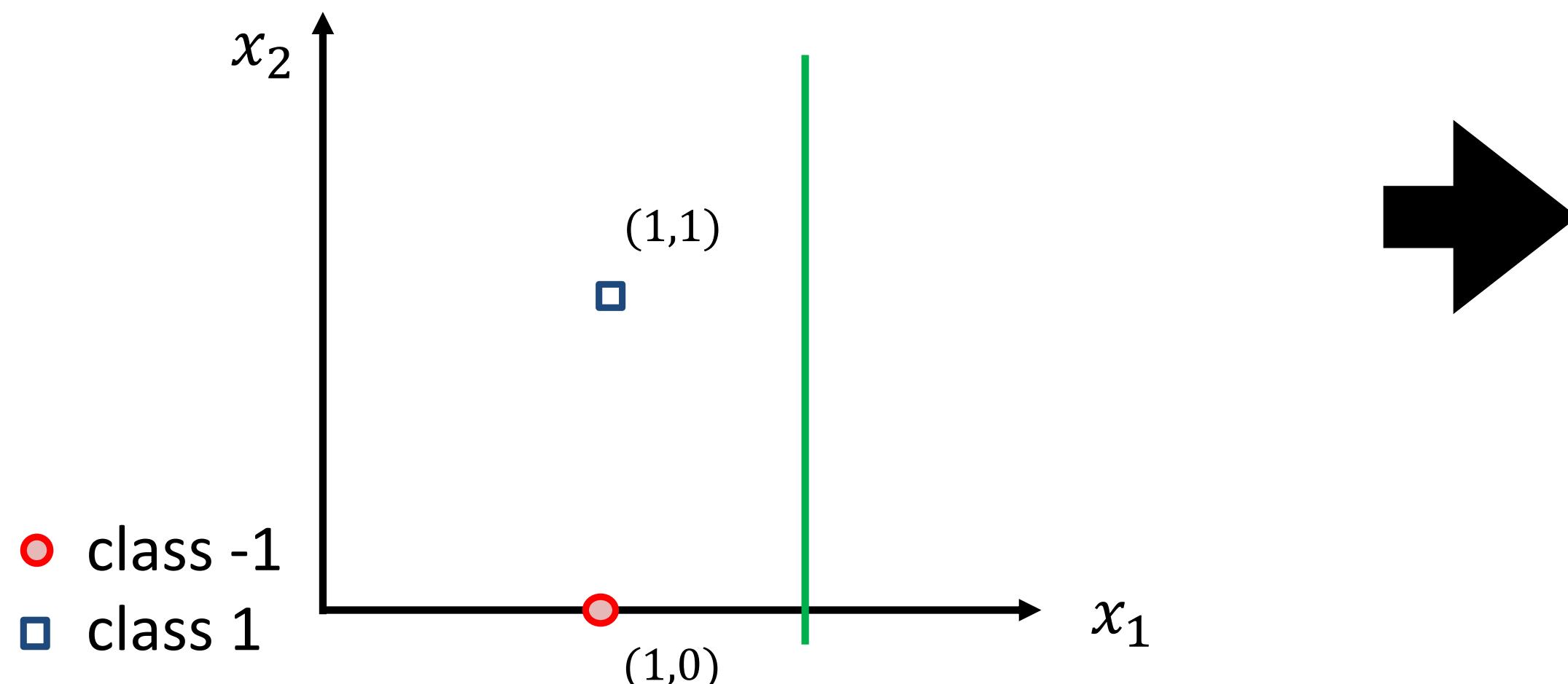
$$w_0 \leftarrow w_0 - \eta \cdot (-1) = -0.1$$



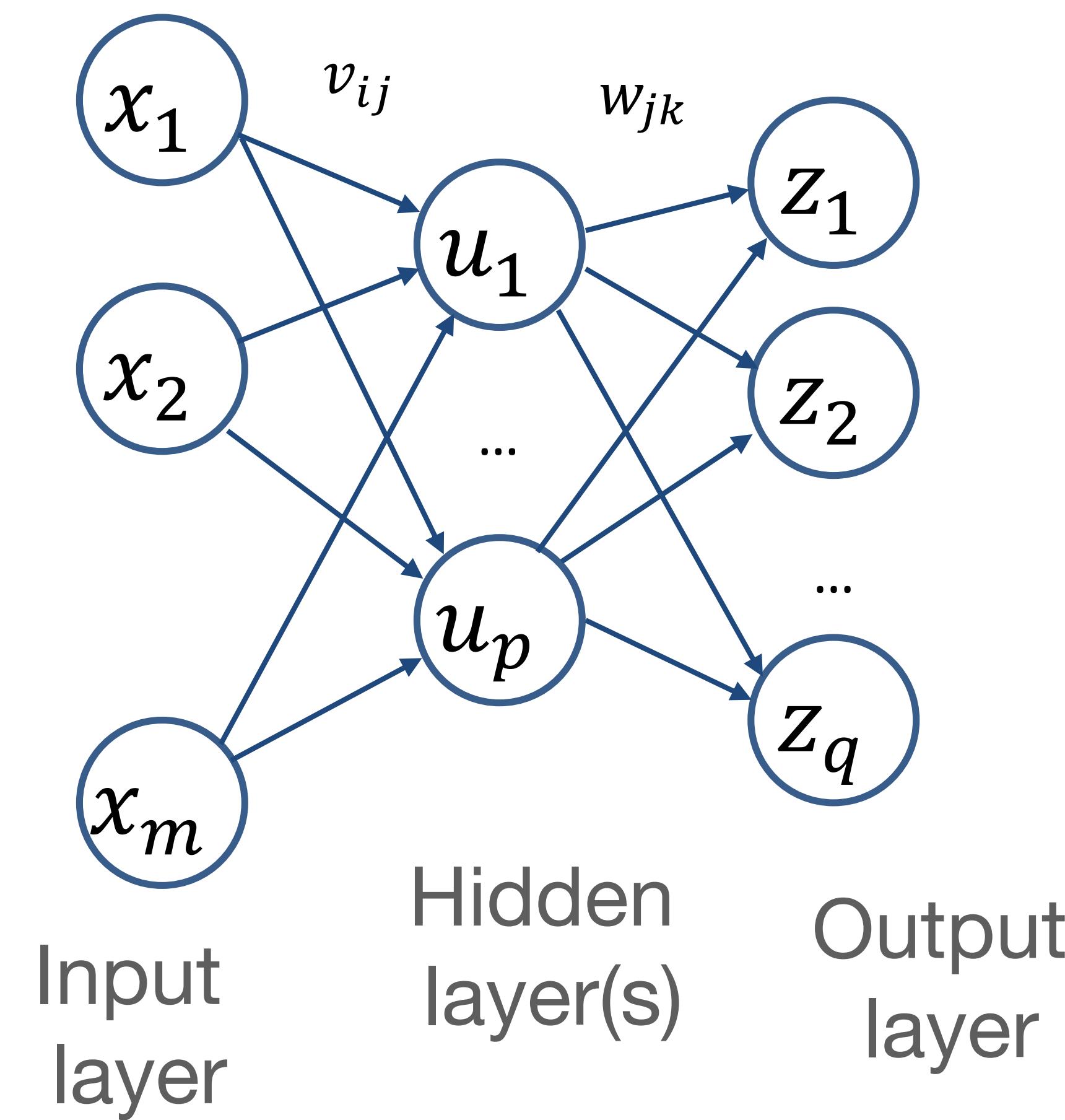
## Learn perceptron on the simple example: Update weights



$x_1$	$x_2$	$y$
0	0	-1
0	1	-1
1	0	-1
1	1	1



## Stochastic gradient descent for multi-layer perceptron



## Stochastic gradient descent for multiple-layer perceptron

Choose initial guess  $\theta^{(0)}, k = 0$

Here  $\theta$  is a set of all weights form all layers

For  $i$  from 1 to  $T$  (epochs)

For  $j$  from 1 to  $N$  (training examples)

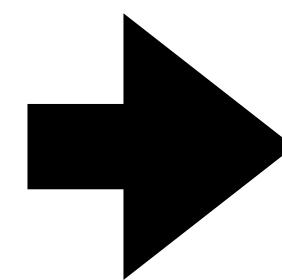
Consider example  $\{x_j, y_j\}$

Update:  $\theta^{(k+1)} = \theta^{(k)} - \eta \nabla L(\theta^{(k)})$ ;  $k \leftarrow k+1$

Need to compute partial derivatives  $\frac{\partial L}{\partial v_{ij}}$  and  $\frac{\partial L}{\partial w_j}$

## Chain rule

Given  $z = g(u)$     $u = f(x)$



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

Example :  $z = \sin(x^2)$

$$\begin{array}{c} z = \sin(u) \\ \uparrow \\ u = x^2 \end{array}$$

$$\begin{aligned} \frac{dz}{du} &= \cos(u) \\ \downarrow \\ \frac{dz}{dx} &= \frac{dz}{du} \frac{du}{dx} = 2x \cos(u) \end{aligned}$$

## Multi-layer perceptron: Function composition

Forward prediction

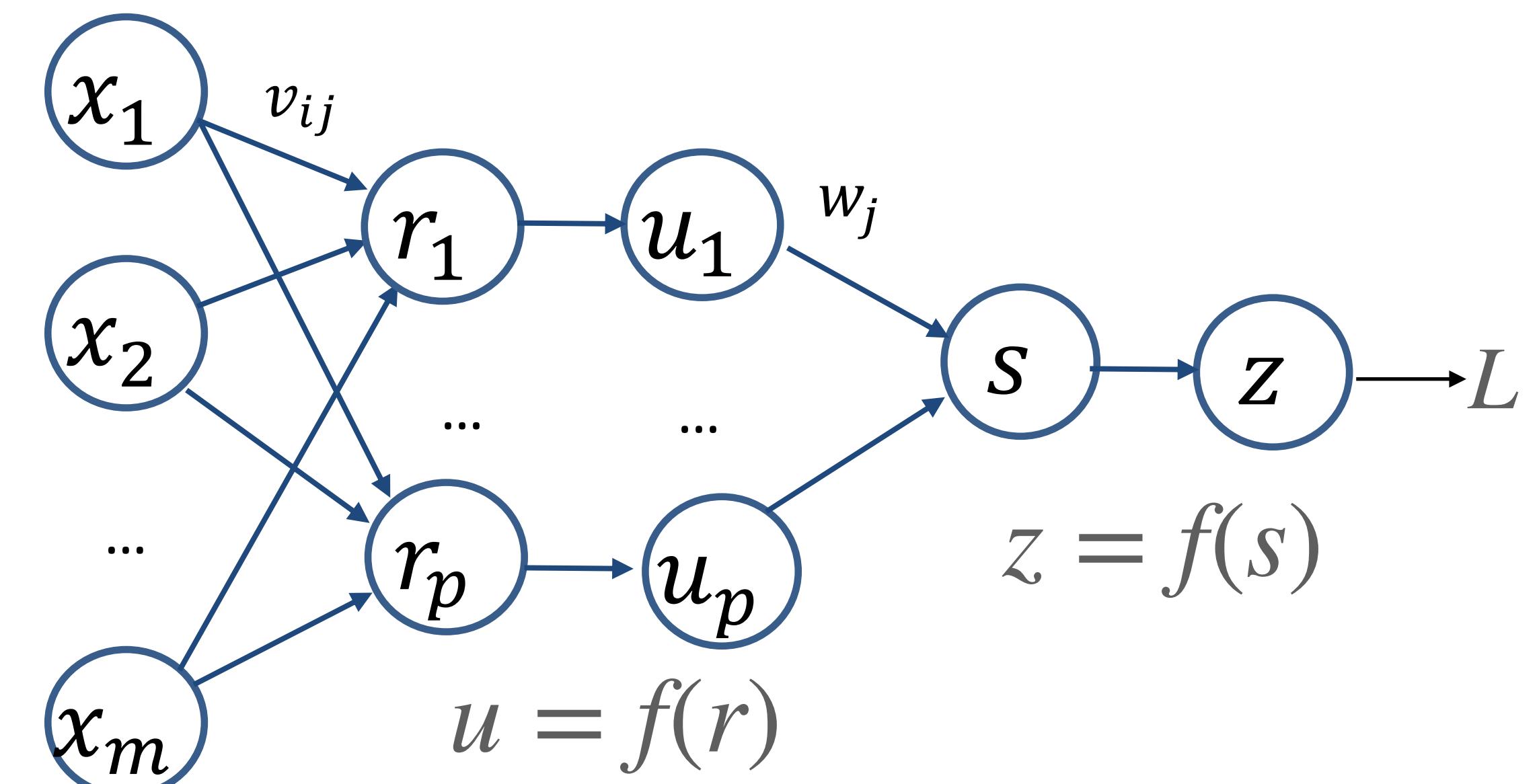
$$z = f(s) = \text{sigmoid}(s) = \frac{1}{1 + e^{-s}}$$

$$s = \sum_{j=0}^p w_j u_j$$

$$u_j = f(r_j) = \text{sigmoid}(r_j) = \frac{1}{1 + e^{-r_j}}$$

$$r_j = \sum_{i=0}^m x_i v_{ij}$$

$$x \rightarrow r \rightarrow u \rightarrow s \rightarrow z$$



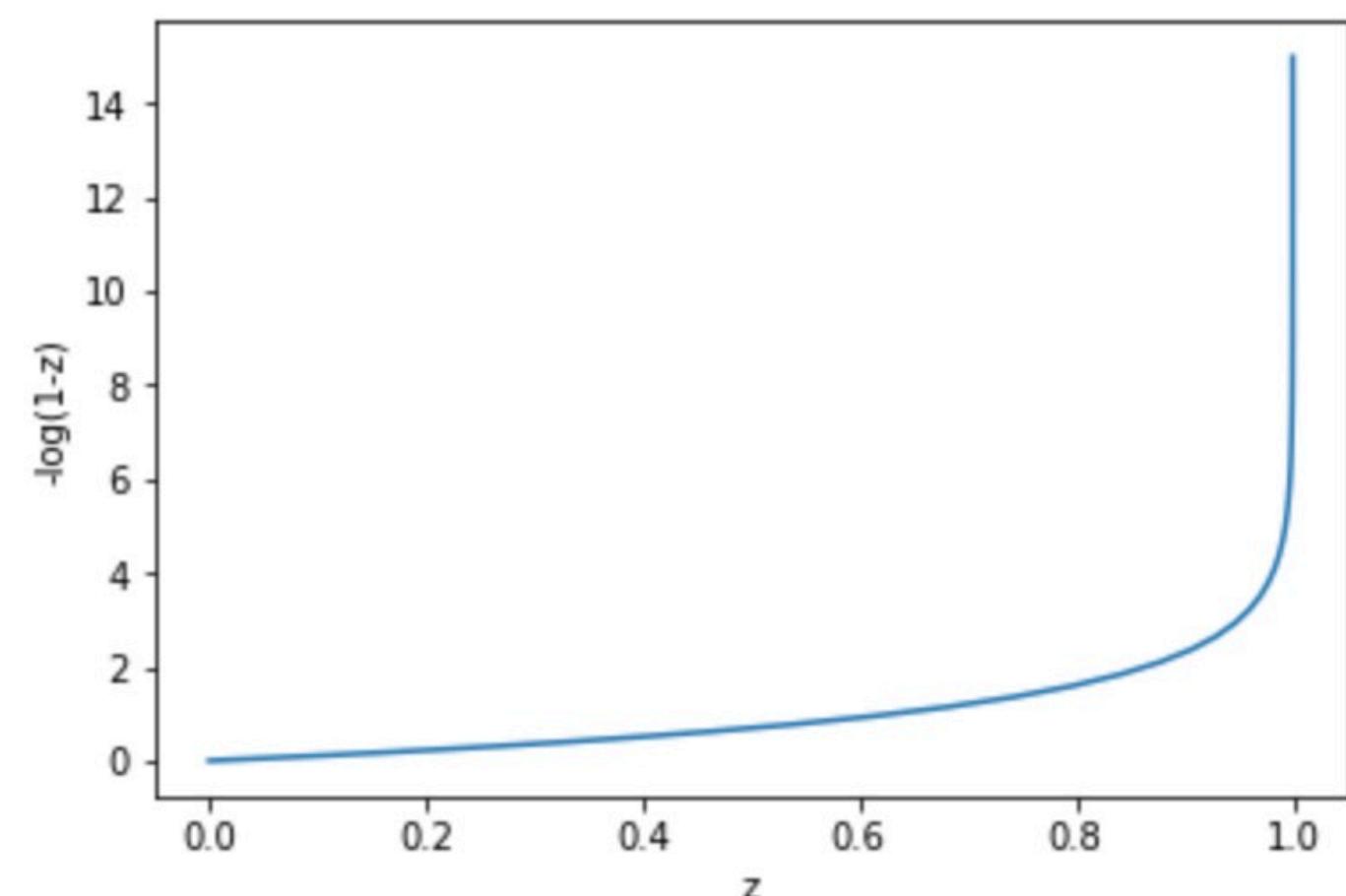
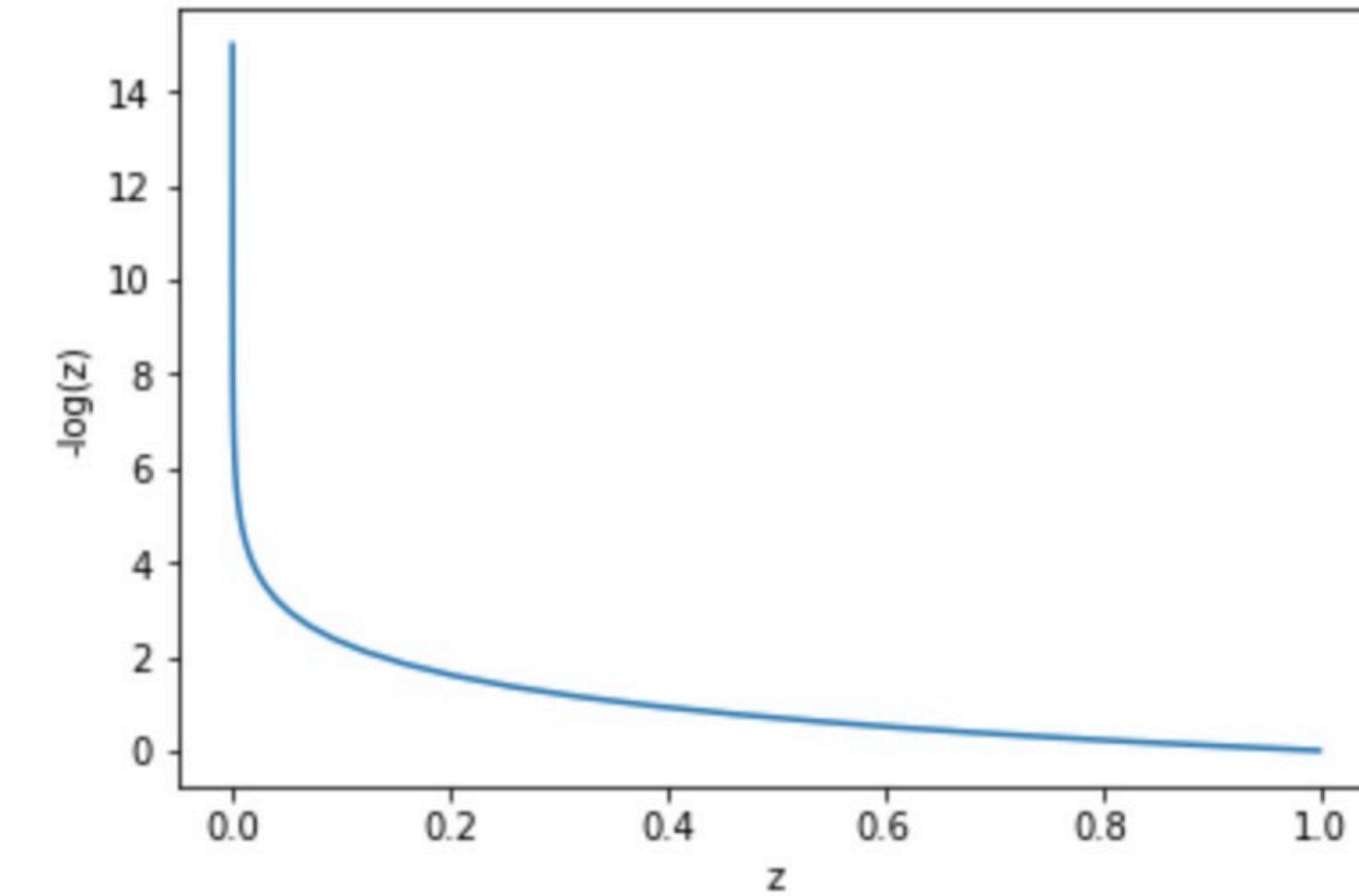
$f$ : activation functions

## Binary cross-entropy

$$L = - [y \log(z) + (1 - y)\log(1 - z)]$$

y: Labels of the data

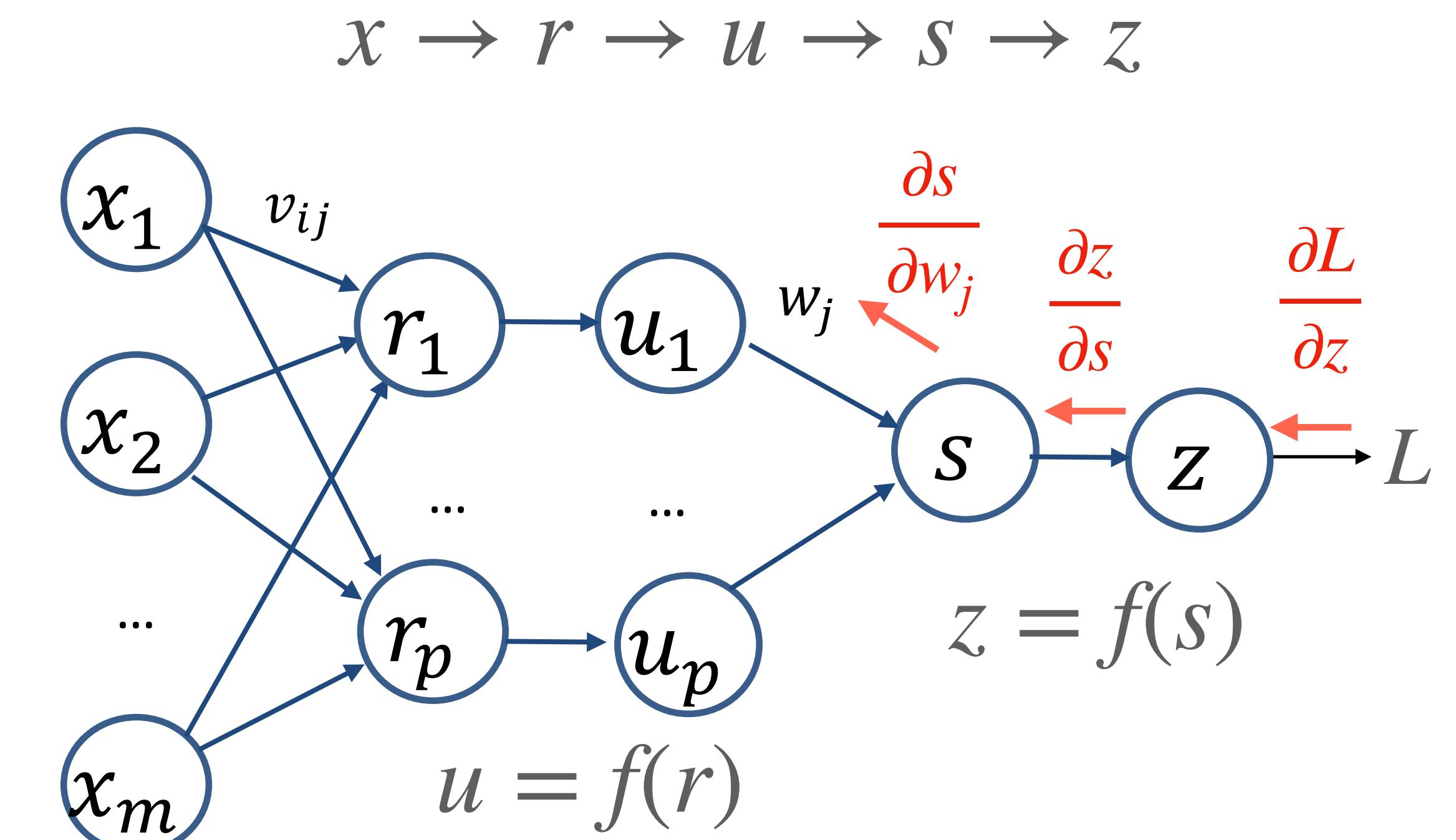
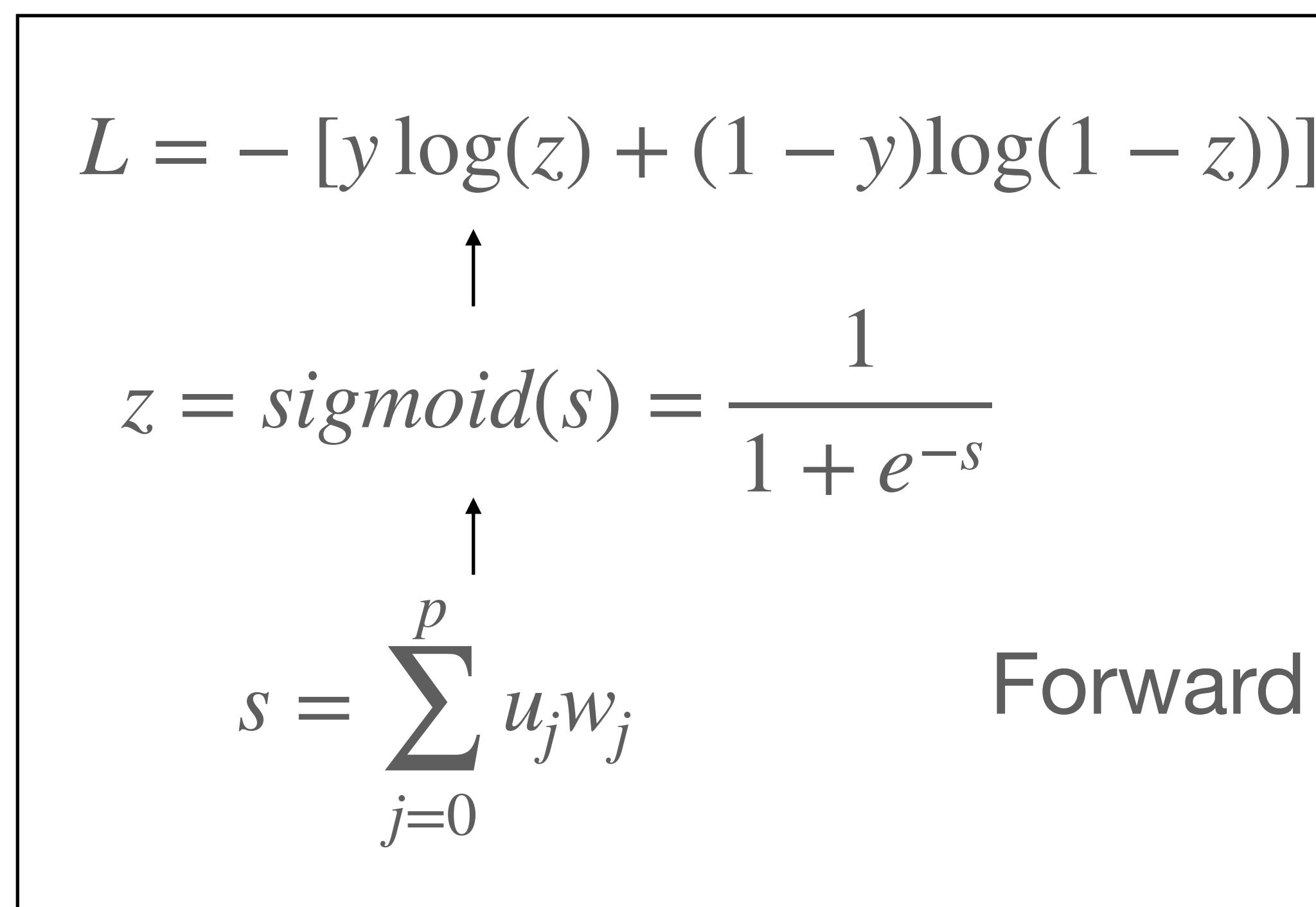
y=1 for positive class or 0 for negative class



## Multi-layer perceptron: Chain rule

Backward propagation:

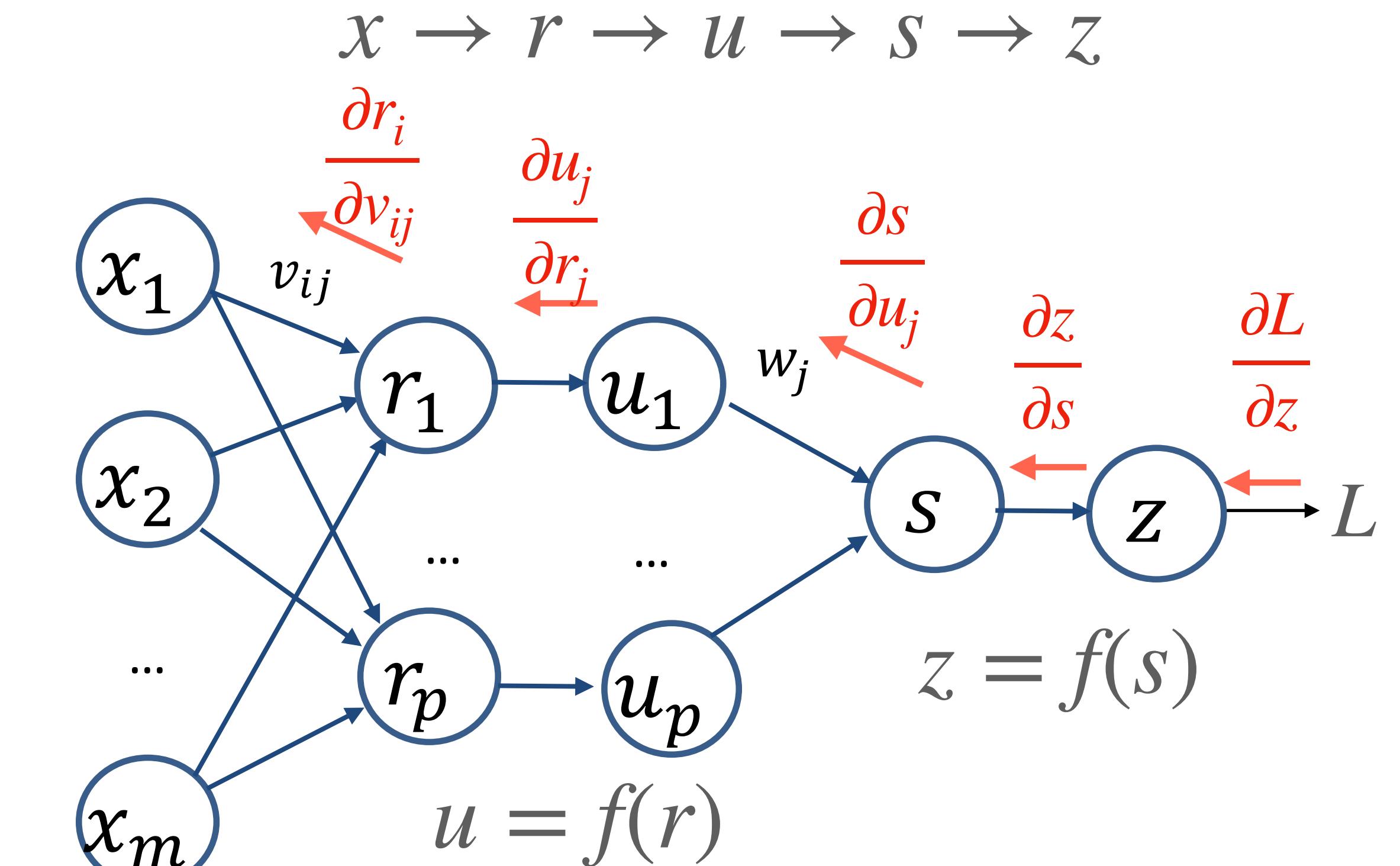
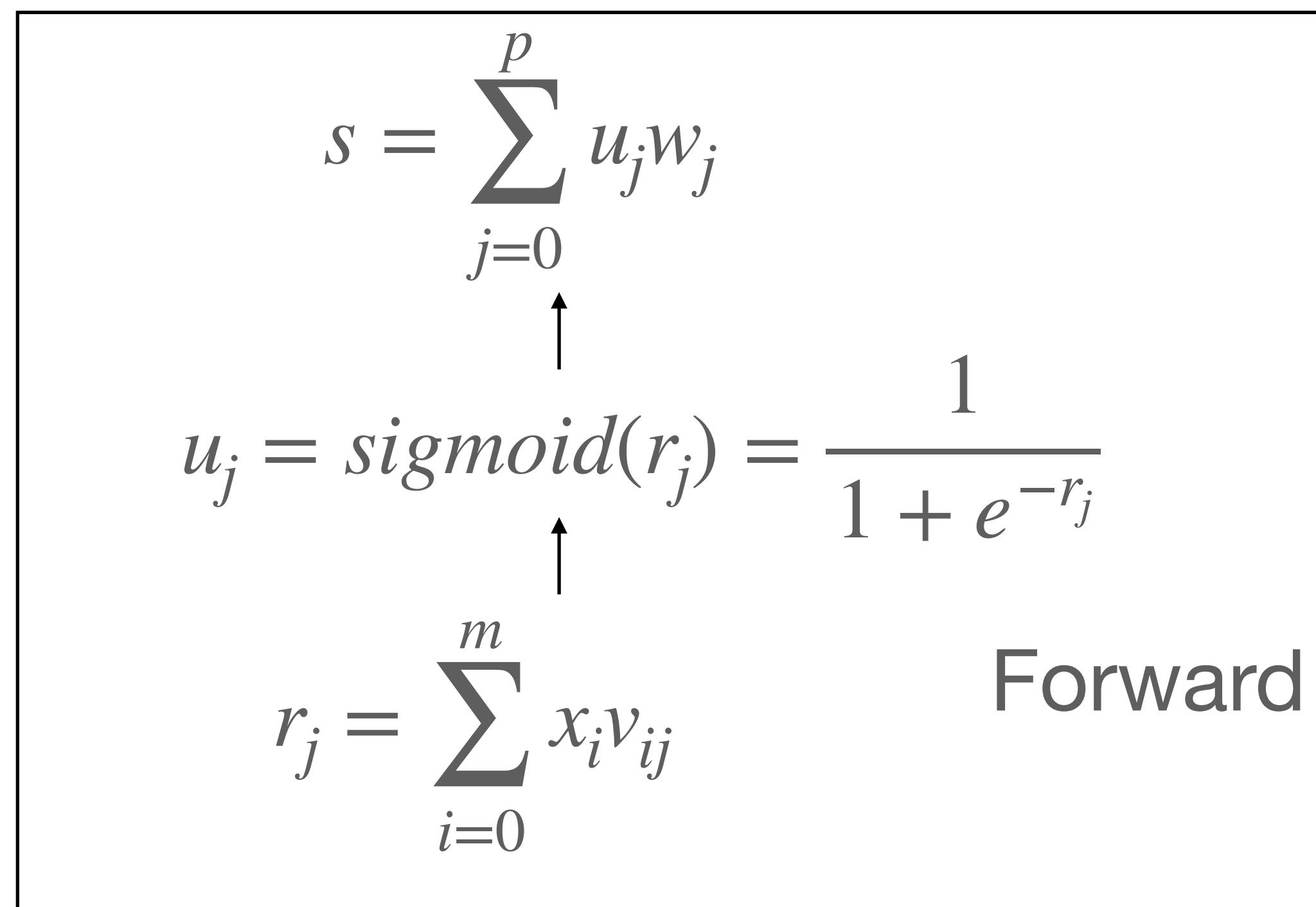
$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial s} \frac{\partial s}{\partial w_j}$$



## Multi-layer perceptron: Chain rule

Backward propagation:

$$\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial s} \frac{\partial s}{\partial u_j} \frac{\partial u_j}{\partial r_j} \frac{\partial r_j}{\partial v_{ij}}$$



## Forward prediction

$$L = -[y \log(z) + (1 - y)\log(1 - z)]$$



$$z = \text{sigmoid}(s) = \frac{1}{1 + e^{-s}}$$



$$s = \sum_{j=0}^p u_j w_j$$



$$u_j = \text{sigmoid}(r_j) = \frac{1}{1 + e^{-r_j}}$$



$$r_j = \sum_{i=0}^m x_i v_{ij}$$

## Backward propagation

$$\frac{\partial L}{\partial z} = -\frac{y}{z} + \frac{1-y}{1-z}$$



$$\frac{\partial L}{\partial s} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial s} = z - y, \quad \frac{\partial z}{\partial s} = z(1 - z)$$



$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial s} \frac{\partial s}{\partial w_j} = \frac{\partial L}{\partial s} u_j$$

$$\frac{\partial L}{\partial u_j} = \frac{\partial L}{\partial s} \frac{\partial s}{\partial u_j} = \frac{\partial L}{\partial s} w_j$$



$$\frac{\partial L}{\partial r_j} = \frac{\partial L}{\partial u_j} \frac{\partial u_j}{\partial r_j} = \frac{\partial L}{\partial u_j} (u_j)(1 - u_j)$$



$$\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial r_j} \frac{\partial r_j}{\partial v_{ij}} = \frac{\partial L}{\partial r_j} x_i$$

# Summary

- What is deep learning& difference with traditional machine learning?
- How to train neural network using gradient descent algorithm?
- How to train a perceptron using stochastic gradient descent?
- What is chain rule?
- How to perform forward prediction and back propagation in multilayer perceptron?

Next: Convolutional neural network

# References

- Deep learning with python by Francois Chollet, chapter 1 & 2.4
- Pattern Recognition and Machine Learning by Chris Bishop, chapter 4.1.7, 5.1-5.3.