Image processing for Earth Observation

4 – neural networks

Devis TUIA

EPFL, fall semester 2024

École polytechnique fédérale de Lausanne

Content (6 weeks)

- W1 General concepts of image classification / segmentation
  Traditional supervised classification methods (RF)
- W2 Traditional supervised classification methods (SVM)
  Best practices

- W3 Elements of neural networks
- W4 Convolutional neural networks
- W5 Convolutional neural networks for semantic segmentation
- W6 Sequence modeling, change detection

And now: deep learning!

- In the rest of the classes, we will focus on a ~~new~~ type of model: neural networks

And now: deep learning!

- In the rest of the classes, we will focus on a ~~new~~ type of model: neural networks
- Neural networks are the base of what we call today deep learning

AI can figure out a place's politics by analyzing cars on Google Street View

A neural network combed through 50 million images.

By Rob Verger *November 30, 2017*

And now: deep learning!

- In the rest of the classes, we will focus on a ~~new~~ type of model: neural networks
- Neural networks are the base of what we call today deep learning (DL)
- In the next 3 weeks we will learn about
  - neural networks in general
  - convolutional neural networks (CNNs)
  - how to use CNNs for semantic segmentation and sequence prediction

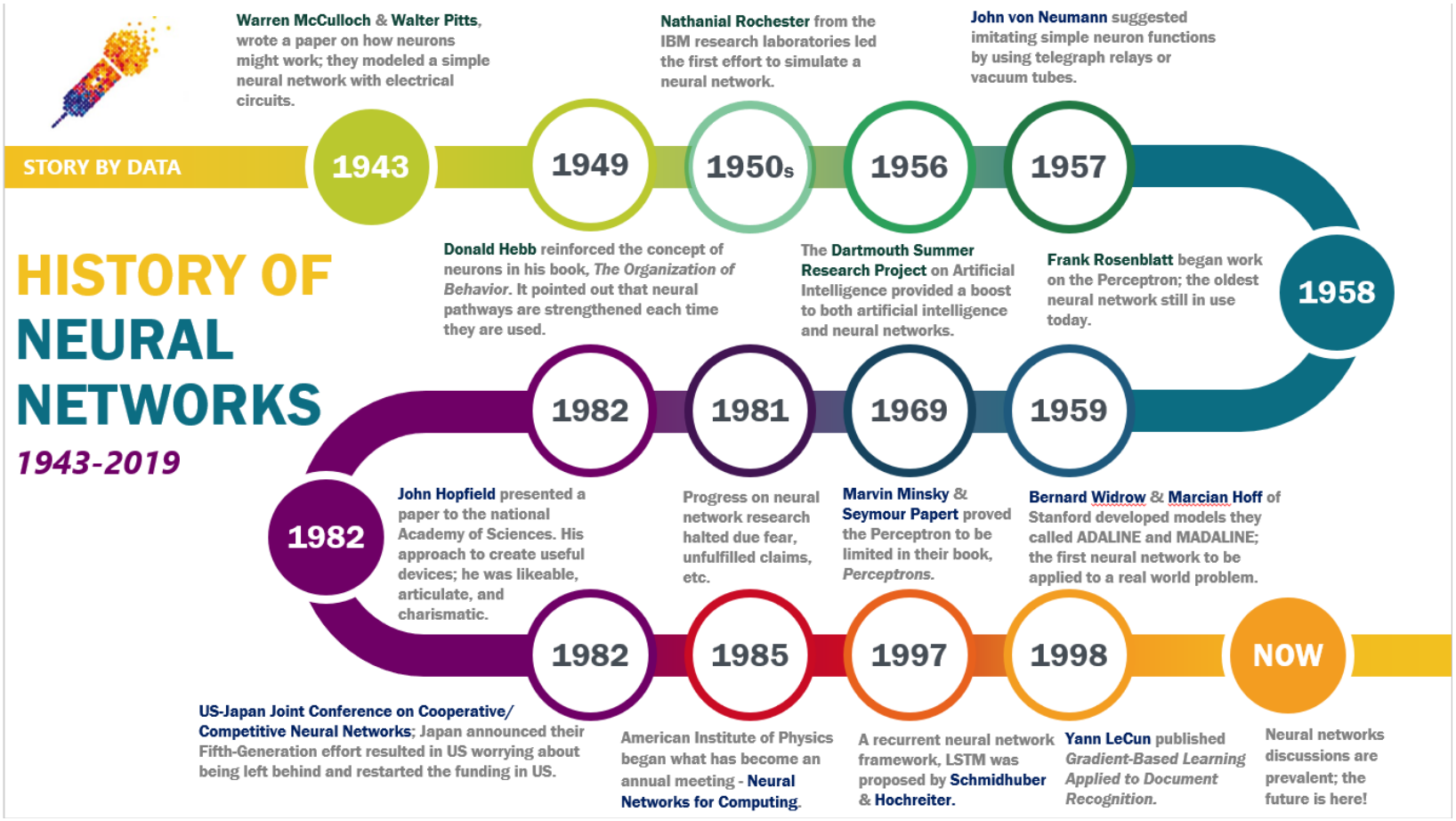
- **Disclaimer:** we will cover the basics only. For an in-depth study of DL, please consider more in depth courses in IC, STI.

Neural networks

History and myths

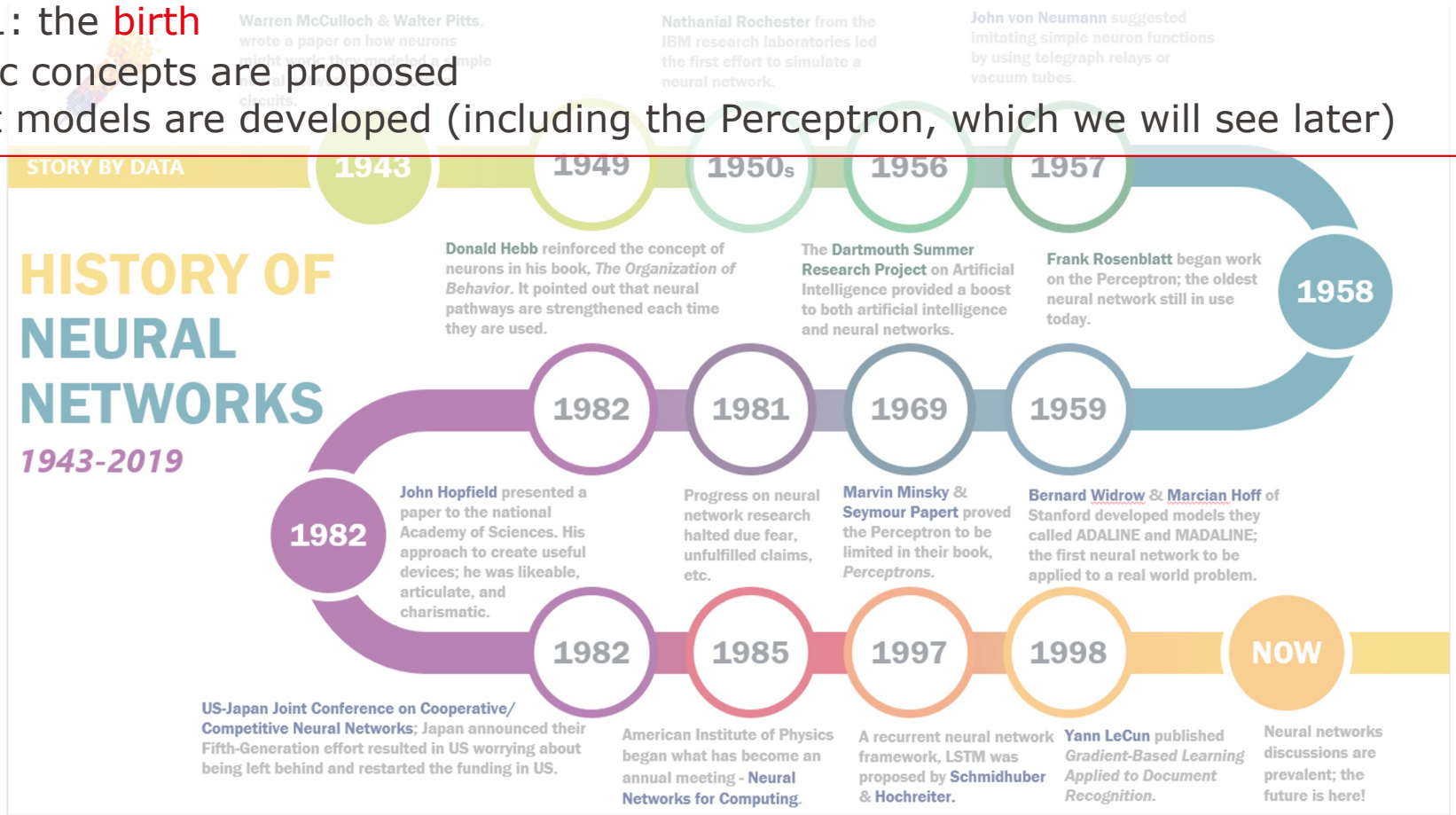
Basic principles of neural networks

A bit of history



A bit of history

Phase 1: the birth
- Basic concepts are proposed
- First models are developed (including the Perceptron, which we will see later)



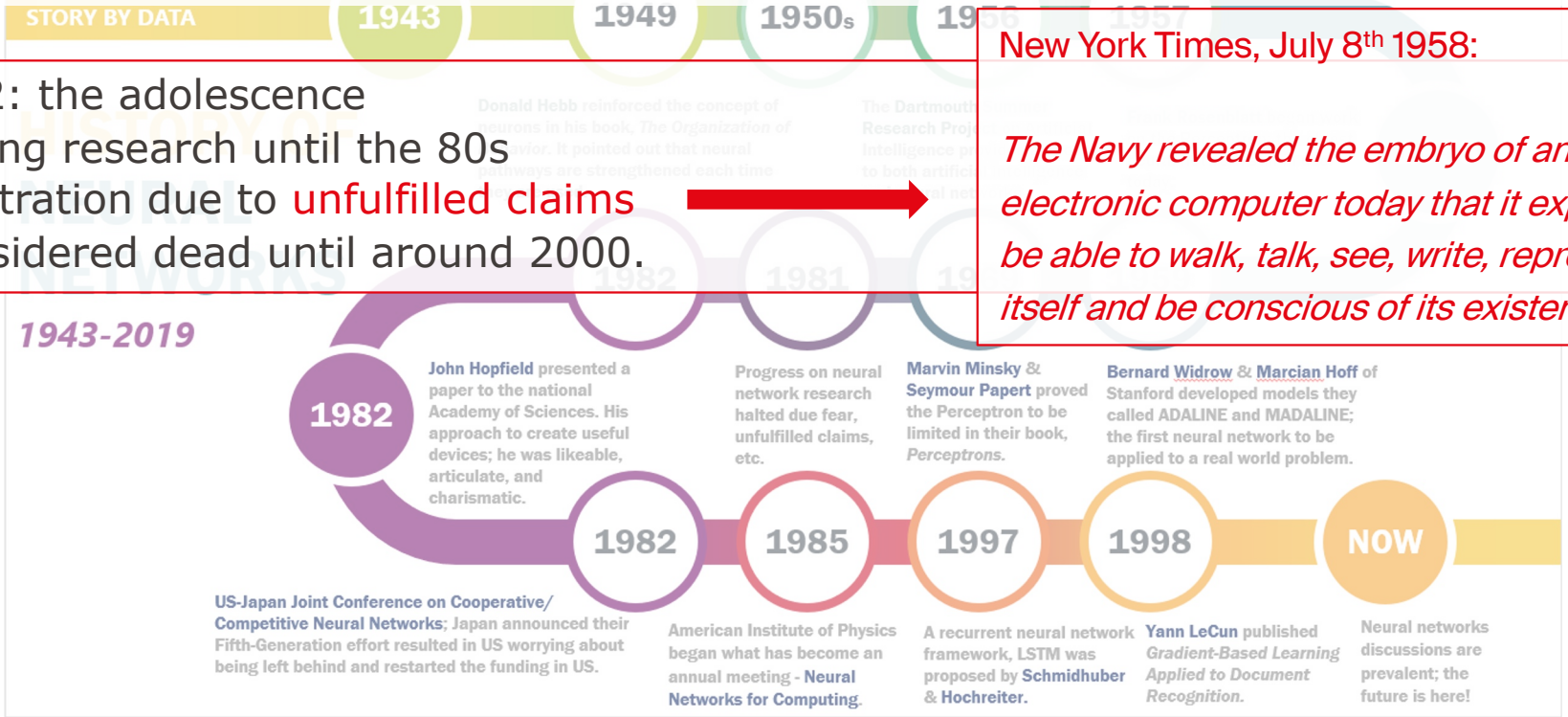
A bit of history

Phase 1: the birth
- Basic concepts are proposed
- First models are developed (including the Perceptron we will see later)

Phase 2: the adolescence
- Strong research until the 80s
- Frustration due to unfulfilled claims
- Considered dead until around 2000.

New York Times, July 8th 1958:

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

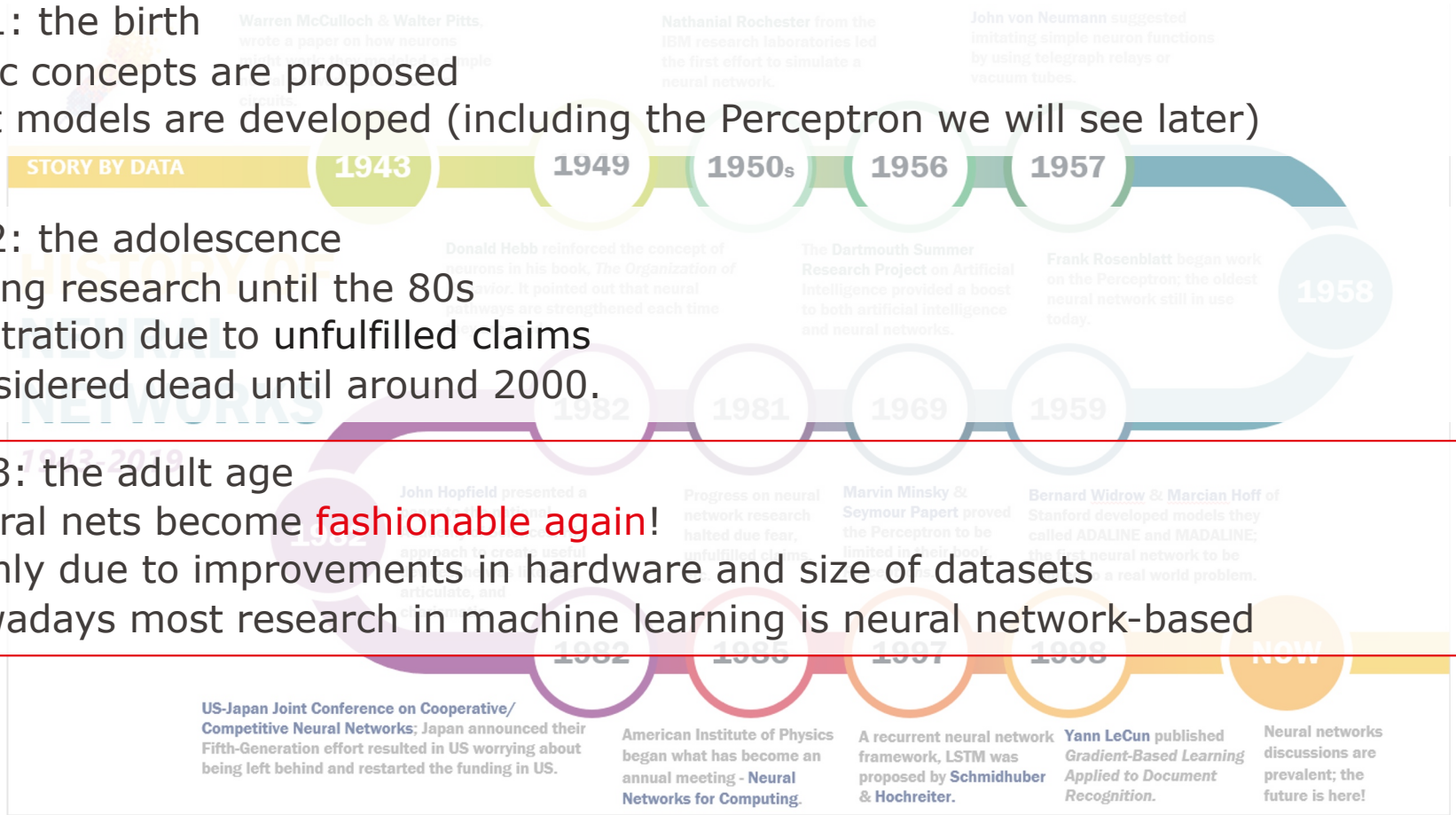


A bit of history

Phase 1: the birth
- Basic concepts are proposed
- First models are developed (including the Perceptron we will see later)

Phase 2: the adolescence
- Strong research until the 80s
- Frustration due to unfulfilled claims
- Considered dead until around 2000.

Phase 3: the adult age
- Neural nets become fashionable again!
- Mainly due to improvements in hardware and size of datasets
- Nowadays most research in machine learning is neural network-based

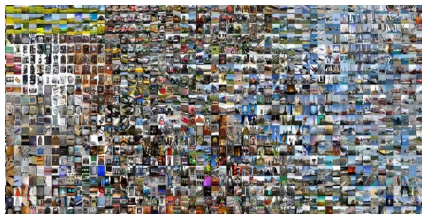


Key moments

- 1943 : McCulloch and Pitts: model of neuron
- 1958 : Rosenblatt: **perceptrons**
- 1968 : Minsky and Papert point out limitations: perceptrons are linear
- 1982 : Hopfield network (associative memory), → **Nobel Prize in Physics 2024!** Kohonen's self-organising map (clustering), Fukushima's Neocognitron (vision)
- 1986 : Rumelhart, Hinton and Williams: training of **multilayer perceptrons**
- 1989 : LeCun introduces convolutional networks
- 2012 : Hinton (re)introduces these: **deep learning**

So it's not that new?

- No, neural nets have been around for a while, and also DL
- What is new is that nowadays we can learn these models efficiently
- Since we have
  - Data



ImageNet

http://www.image-net.org
Natural images
1000 classes (person, car, cat, ...)
>14 Million images

  - Large computational resources

Microsoft Azure

Google Earth Engine

And this also in EO!

Ohh, but that's so far fetched!

- Well … I am sure you can name a few apps that you use that make use of neural nets.
  - Google translate
  - Alexa
  - Photo upload in Whatsapp/Facebook
  - Amazon web search
  - …

Okok, but that's super complicated!

D. Tuia. ECEO

- True, if you want to master all the complexity of it
- Research in fundamentals of DL is just starting
- For using these algorithms, there is A LOT of resources.
- In particular python libraries that allow building DL models "easily".
- The one you will use: Pytorch.
- There are many others (Keras, Tensorflow, ...)

- But before diving into Pytorch, let's learn how NN and DL work.

EPFL

Neural networks

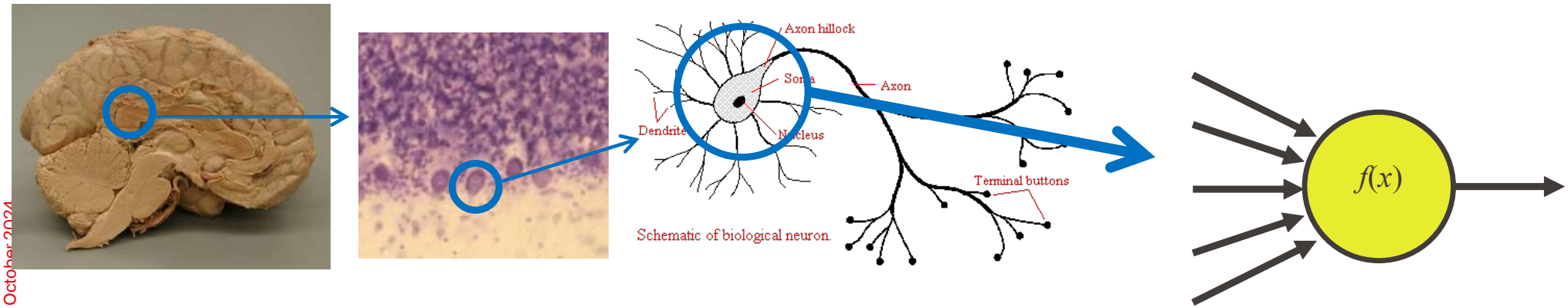
History and myths

Basic principles

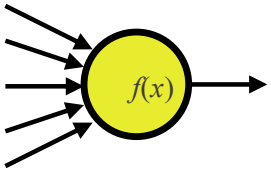
- Artificial neurons

The mathematical model of a neuron

- Some (not all!) networks are originally inspired by the brain
- Neural nets are large, densely interconnected networks of simple processing units, a.k.a neurons
- It remains just a parallel, the artificial neuron is just an approximation!



The neuron model



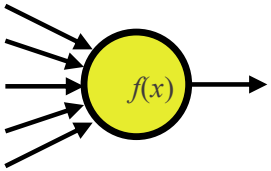
The neuron was first proposed by Warren McCulloch and Walter Pitts in 1943. It was modeled with electrical circuits.

The output y was modeled as a weighted linear combination of inputs x_j *(j are the variables describing each one of your samples)*

$$y = \sum_j \beta_j x_j + \beta_0$$

D. Tuia. ECEO

The neuron model



The neuron was first proposed by Warren McCulloch and Walter Pitts in 1943. It was modeled with electrical circuits.

The output y was modeled as a weighted linear combination of inputs x_j

Moreover, a transfer function, or activation function was used to make a decision :

$$y = f\left(\sum_j \beta_j x_j + \beta_0\right)$$

Activation functions

$$y = f\left(\sum_j \beta_j x_j + \beta_0\right)$$

- Let's call v the result of the parenthesis. Examples of functions:

$$f(v) = \begin{cases} 1 & v \geq 0 \\ 0 & v < 0 \end{cases}$$

Classifies into two groups

$$f(v) = \frac{1}{1 + \exp(-v)}$$

Outputs numbers between 0 and 1

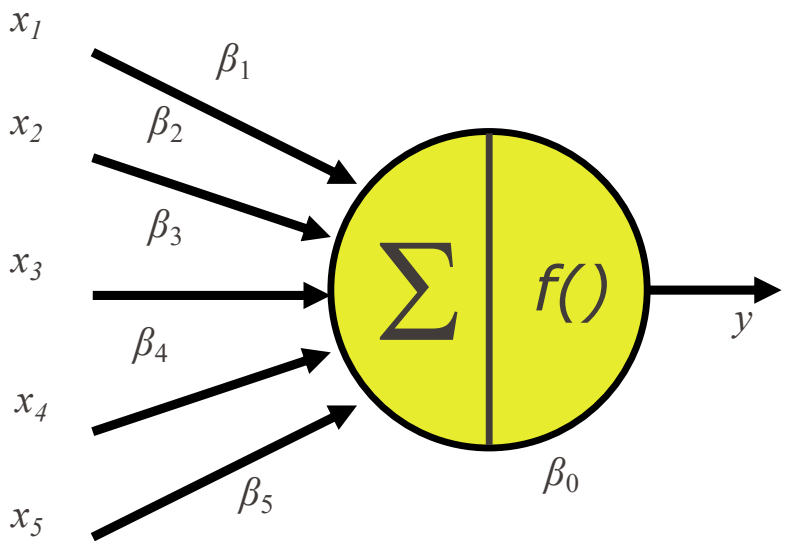
$$f(v) = v$$

Outputs the linear combination itself

The artificial neuron

$$y = f\left(\sum_j \beta_j x_j + \beta_0\right)$$

- A network made of many McCulloch-Pitts neurons can approximate any function, given the right weights β.
- But how to find the right weights?
- We will use **gradient descent**



Neural networks (NN)

History and myths

Basic principles of NN

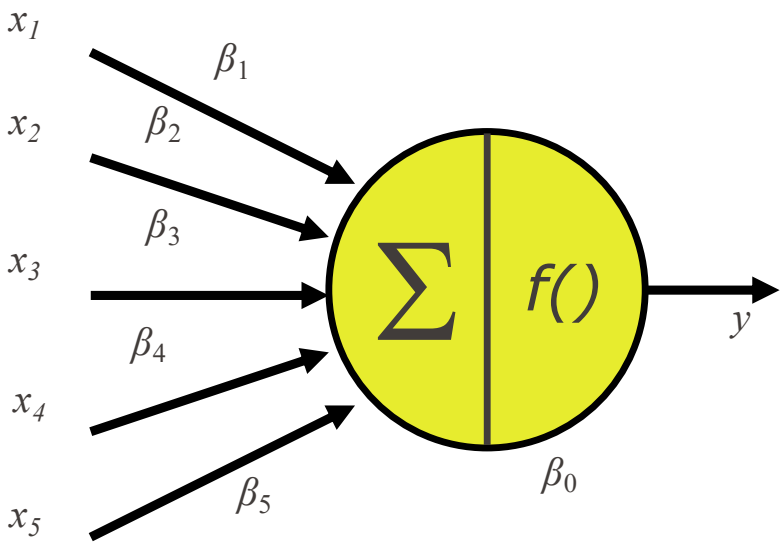
- Artificial neurons
- Gradient descent

Gradient descent in a nutshell

$$y = f\left(\sum_j \beta_j x_j + \beta_0\right)$$

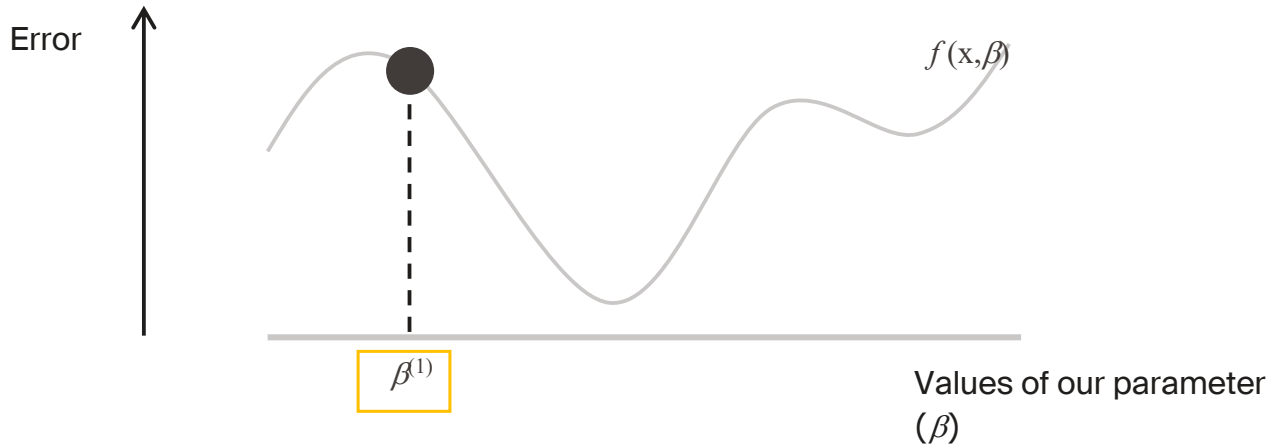
D. Tuia. ECEO

- We want to "find the β_s".
- Roughly
  - We initialise
  - We calculate error of the current set of weights
  - We modify the β_s to make the error decrease
  - We repeat until satisfaction



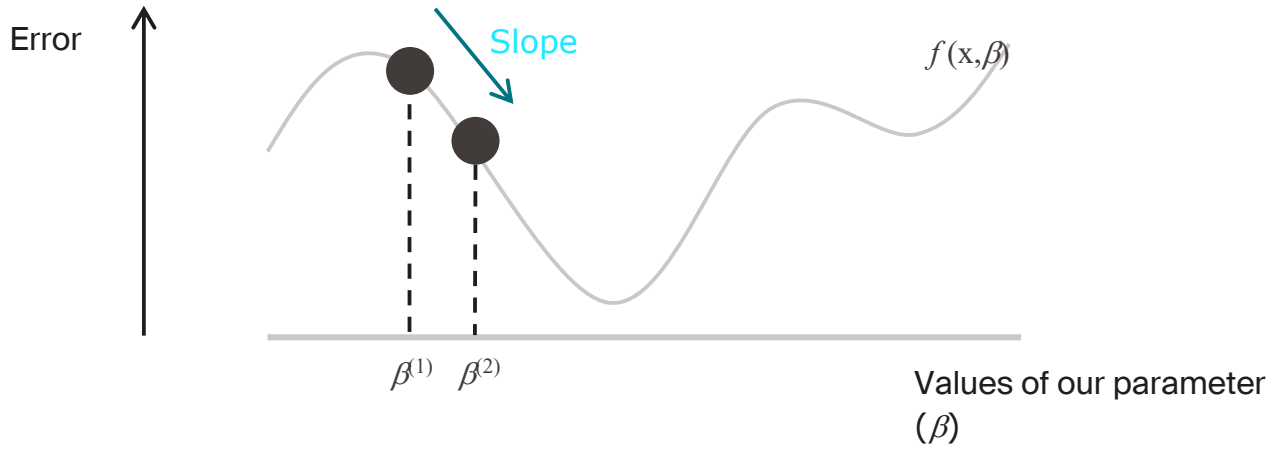
Gradient descent

- We want to find the minimum of an error function f(x,β), which you don't know
- We start with an initial guess of the parameter β



Gradient descent

- We want to find the minimum of an error function f(x,β), which you don't know
- We start with an initial guess of the parameter β
- We change its value in the direction of maximal slope



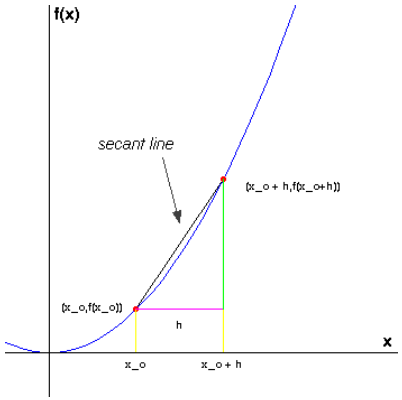
Gradient descent - how

- We want to find the minimum of an error function
- We start with an initial guess of the parameter β
- We change its value in the direction of maximal slope
-

The derivative measures the steepness of the graph of a function at some particular point on the graph.
Thus, the derivative is a slope.

$$\frac{\partial f(x)}{\partial x}|x_0 = f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

It is also a function, it varies according to x



Source:
https://www.ugrad.math.ubc.ca/coursedoc/math100/notes/derivs/deriv5.html

Gradient descent - how

To update a parameter β, we substract the value of its derivative from it

$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)}$$

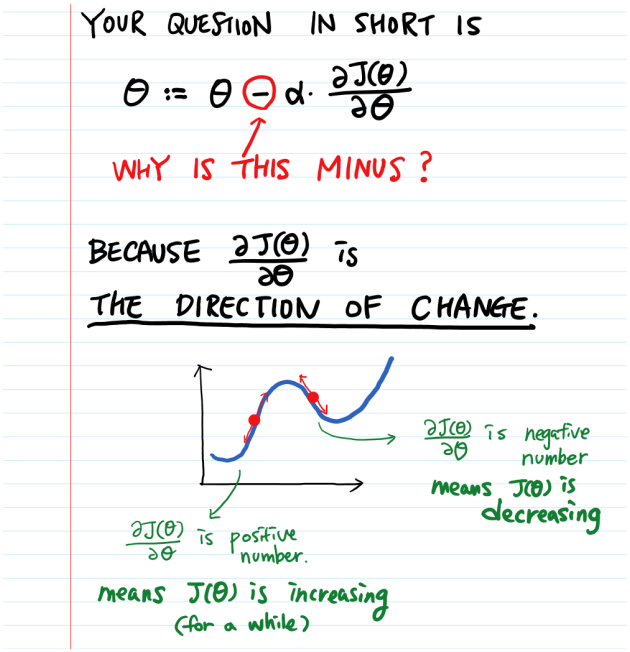
- *α is called a learning rate.*
- *r is the iteration*

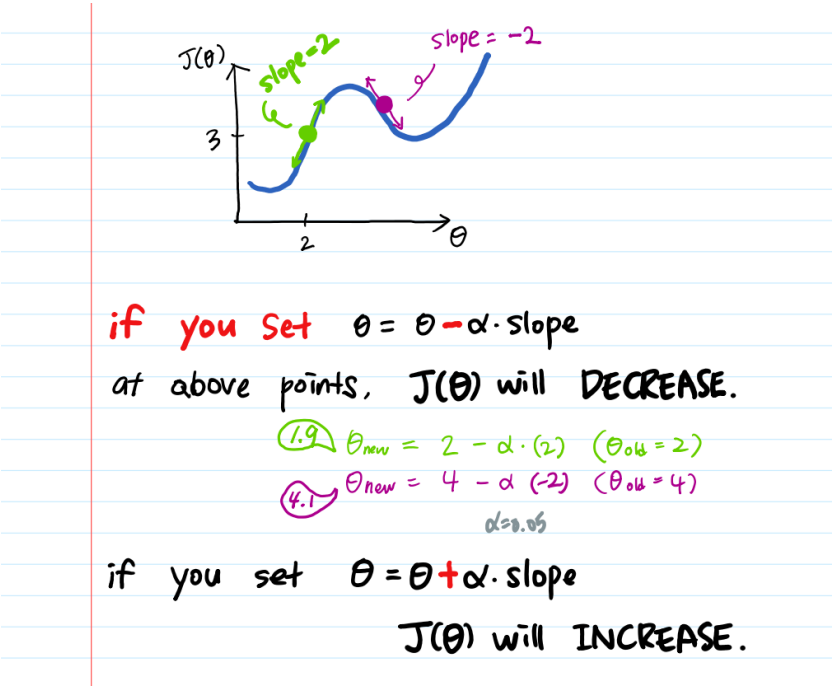
Remember, the derivative is a slope

Gradient descent - how

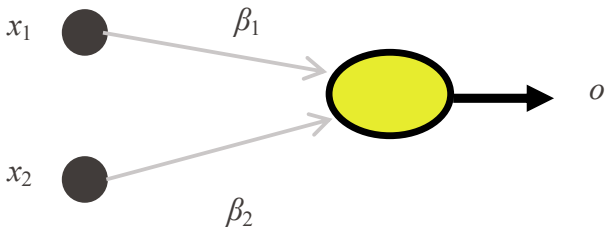
Source: https://medium.com/@aerinykim/why-do-we-subtract-the-slope-a-in-gradient-descent-73c7368644fa

- So why the negative sign?

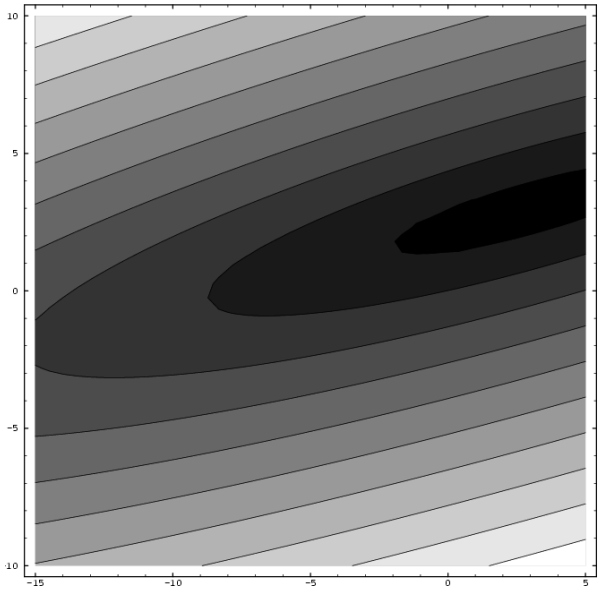




Some intuition



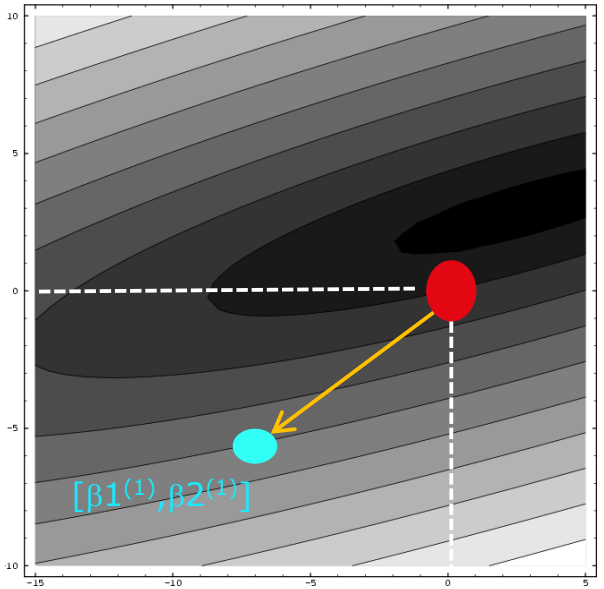
- In the 2D case this is simpler to visualize
- We consider a model with 2 parameters, β_1 and β_2
- (the grey tone of the plot is the error, the darker the smaller)



Some intuition

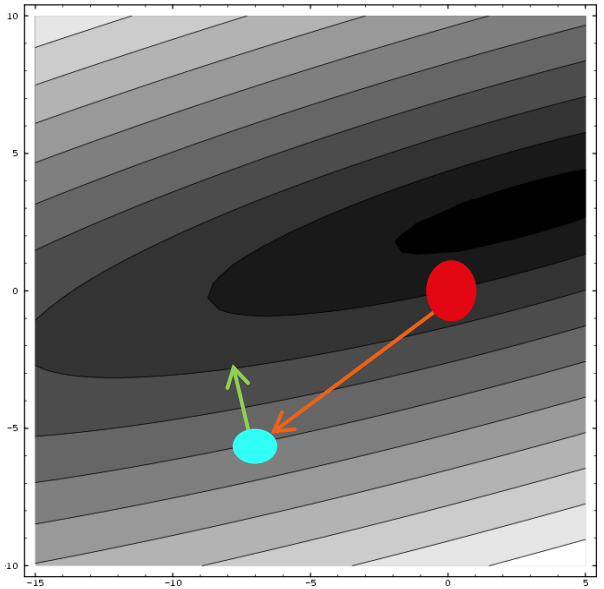
- In the 2D case this is simpler to visualize
- Your initial parameter $[\beta_1^{(1)},\beta_2^{(1)}]$ is a 2D vector from the origin [0,0]

- Where would the next gradient step go?



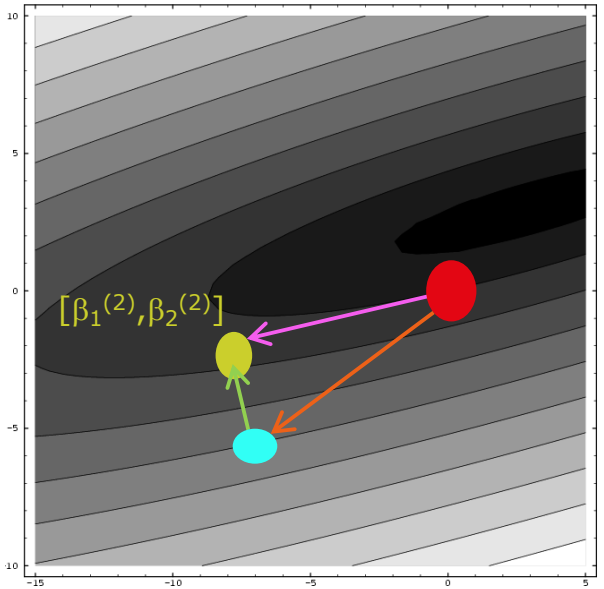
Some intuition

- In the 2D case this is simpler to visualize
- Your initial parameter $[\beta_1^{(1)},\beta_2^{(1)}]$ is a 2D vector from the origin [0,0]
- We compute the direction of maximal slope



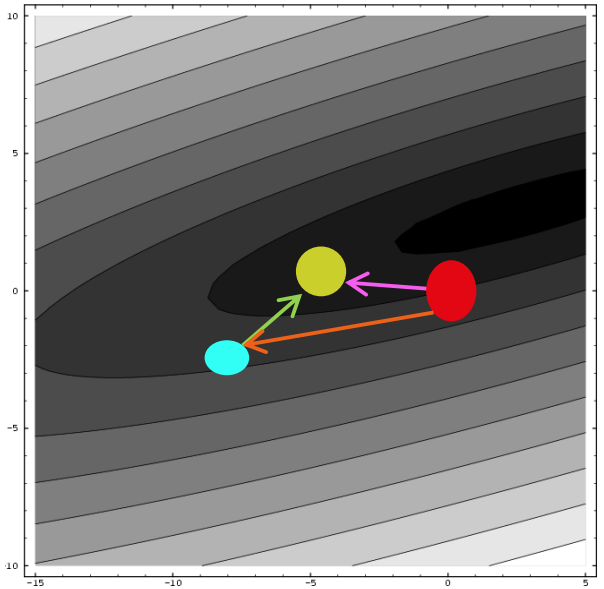
Some intuition

- In the 2D case this is simpler to visualize
- Your initial parameter $[\beta_1^{(1)},\beta_2^{(1)}]$ is a 2D vector from the origin [0,0]
- We compute the direction of maximal slope
- The new parameter $[\beta_1^{(2)},\beta_2^{(2)}]$ is the sum of the two vectors



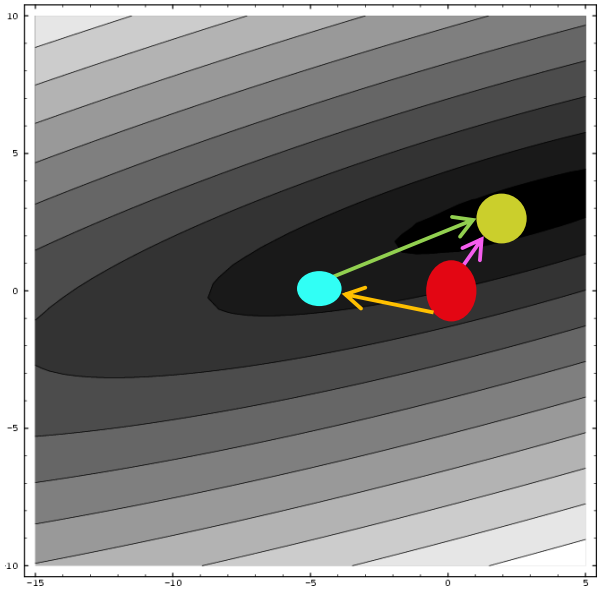
… and then again

- In the 2D case this is simpler to visualize
- Your initial parameter $[\beta_1^{(2)},\beta_2^{(2)}]$ is a 2D vector from the origin [0,0]
- We compute the direction of maximal slope
- The new parameter $[\beta_1^{(3)},\beta_2^{(3)}]$ is the sum of the two vectors



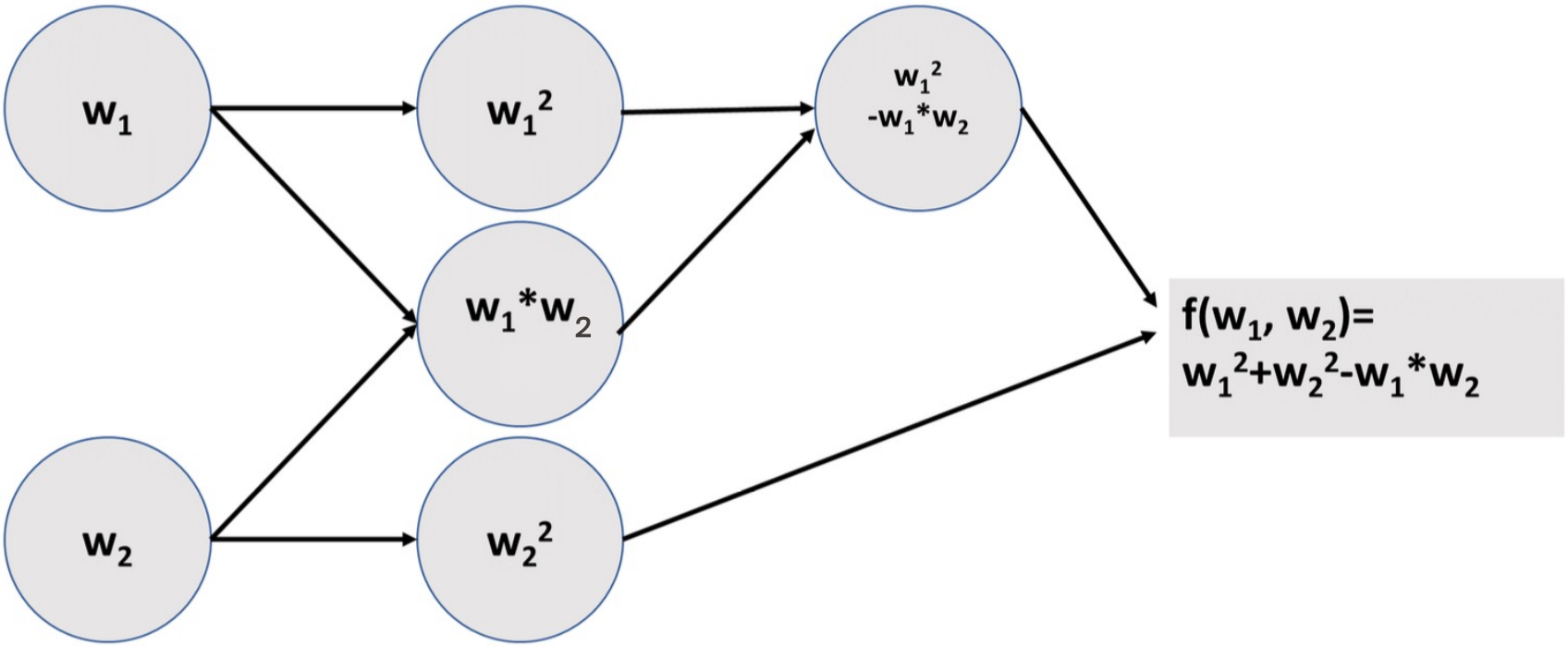
... and again

- In the 2D case this is simpler to visualize
- Your initial parameter $[\beta_1^{(3)},\beta_2^{(3)}]$ is a 2D vector from the origin [0,0]
- We compute the direction of maximal slope
- The new parameter $[\beta_1^{(4)},\beta_2^{(4)}]$ is the sum of the two vectors



A little exercice

- Let's take this neural network

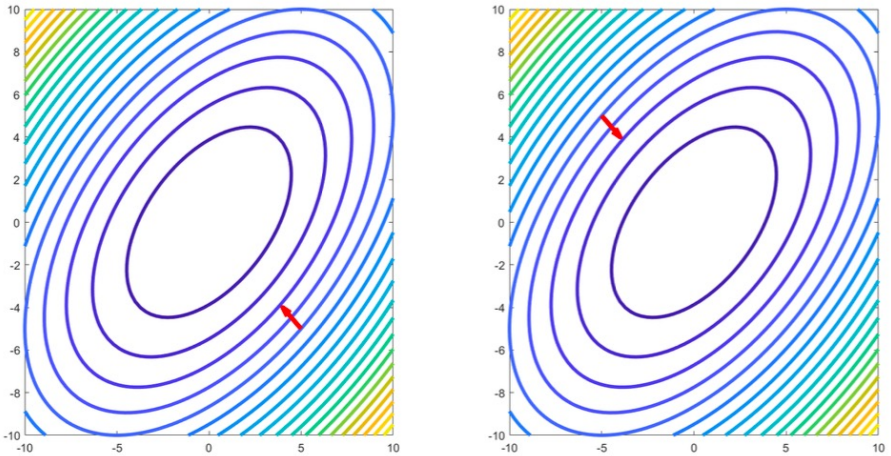


A little exercice

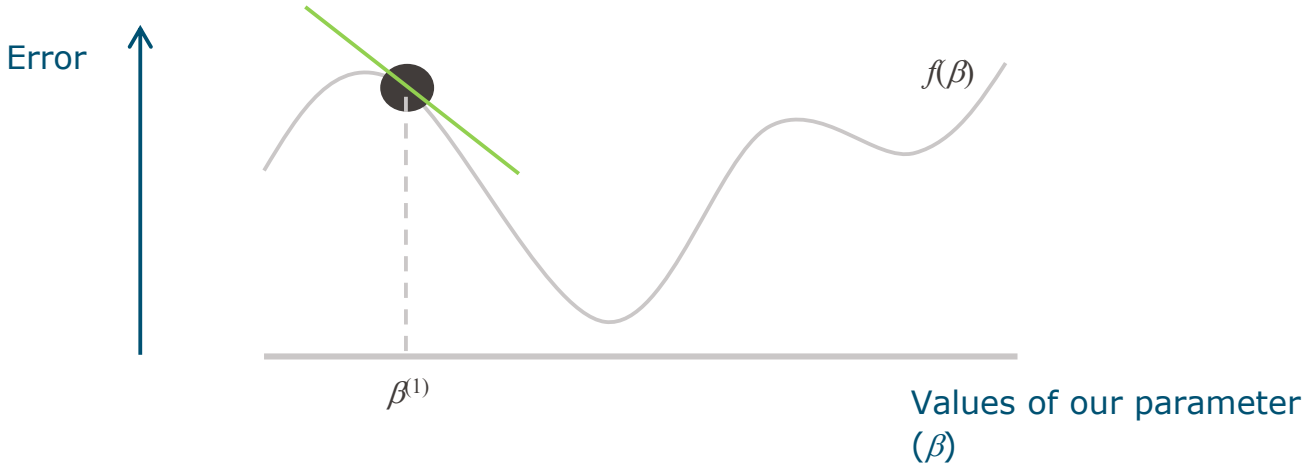
$$f(w_1, w_2) = w_1^2 + w_2^2 - w_1 * w_2$$

$$\nabla f(w_1, w_2) = \left[???, ???\right]^\top$$

Calculate the gradient at $(w_1, w_2) = (5, -5)$ and at $(w_1, w_2) = (-5, 5)$

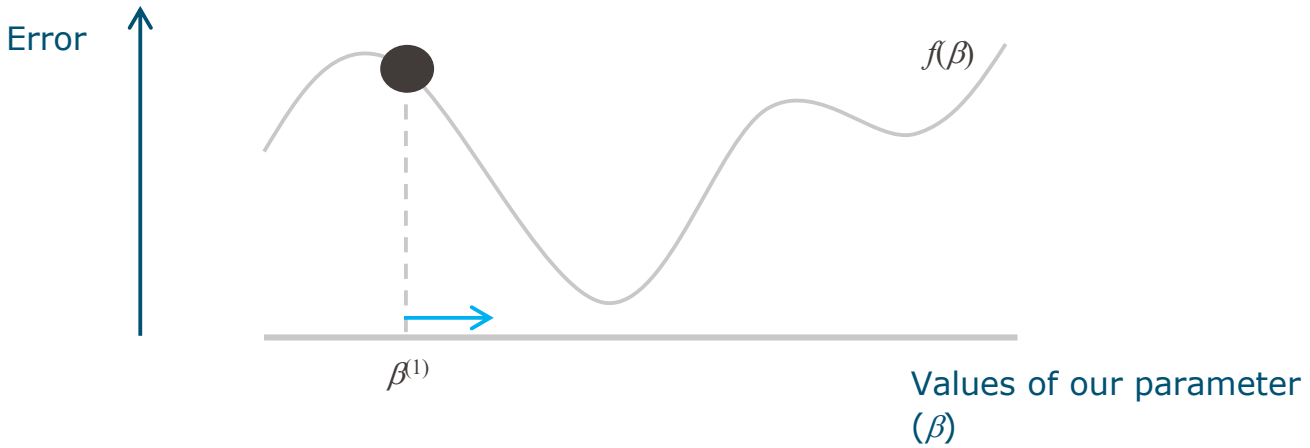


Remember
1: calculate the derivative



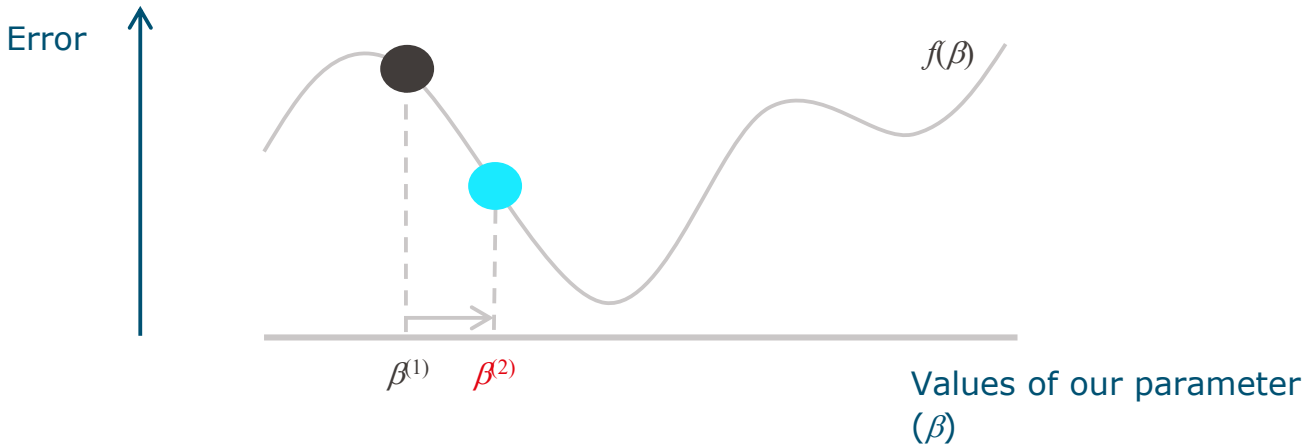
$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)}$$

Remember
2: α gives you the step vector (how far you go)



$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)}$$

Remember
3: this gives you the new value of β and the new error



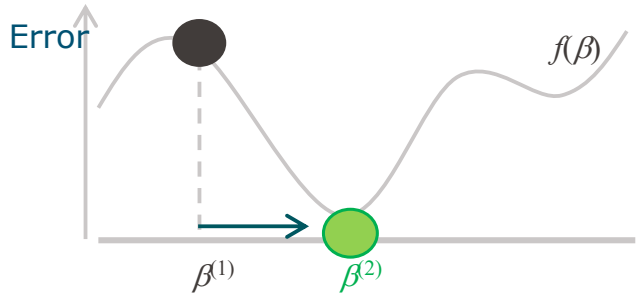
$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)}$$

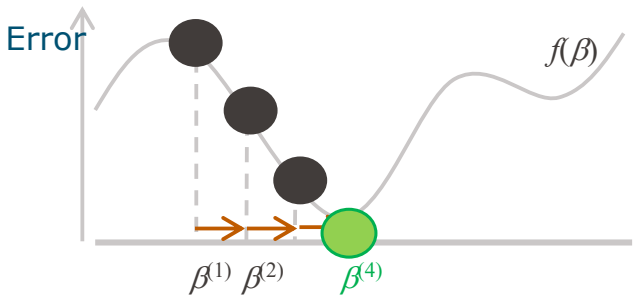
The step value (or learning rate)

$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)}$$

- It decides by how much you multiply the step vector

  Large α can lead to faster convergence than small



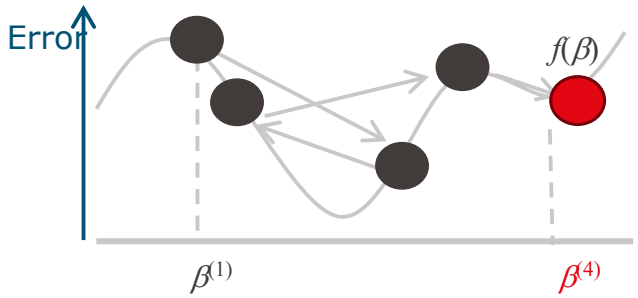


The learning rate

$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)}$$

- It decides by how much you multiply the step vector

  Too large α can lead to disaster



Momentum

- The last trick is called momentum
- Momentum discourages sharp changes of direction in the descent.
- It basically adds a term going in the direction of the previous step

$$\Delta\beta = \beta(r) - \beta(r-1)$$

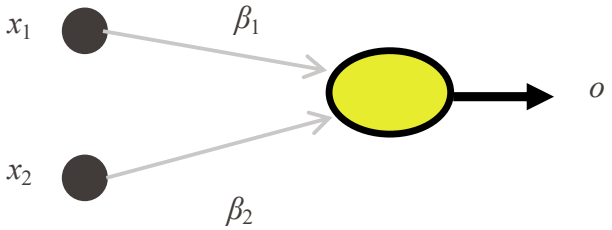
Gradient descent like before

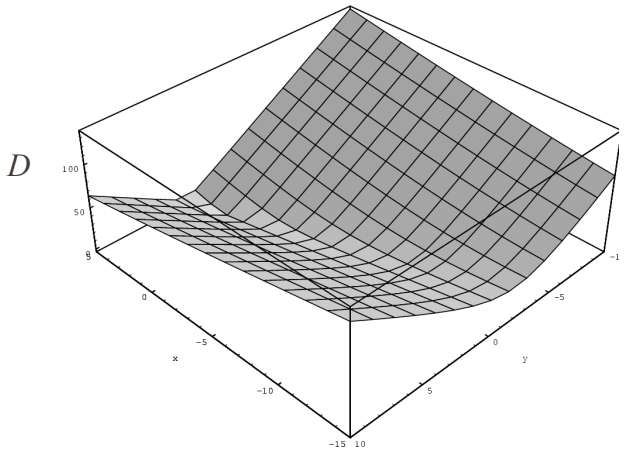
momentum

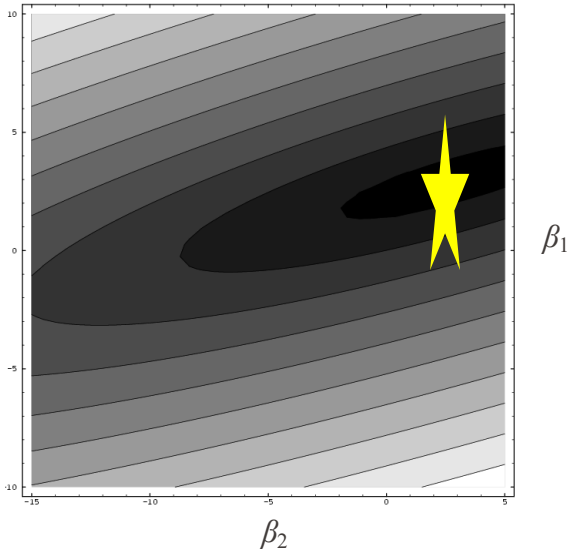
$$\beta(r+1) = \beta(r) - \alpha \frac{\partial f(x)}{\partial x}|_{\beta(r)} + \eta\Delta\beta$$

Numerical example

Perceptron with two weights $[\beta_1, \beta_2]$

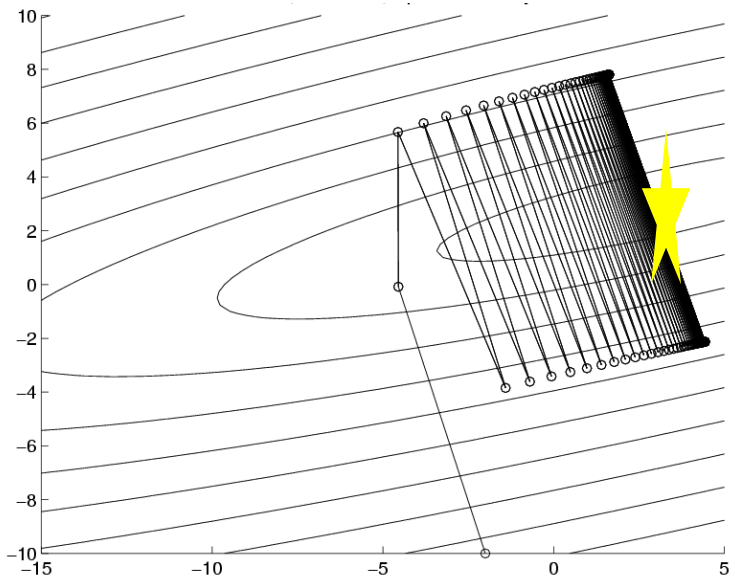




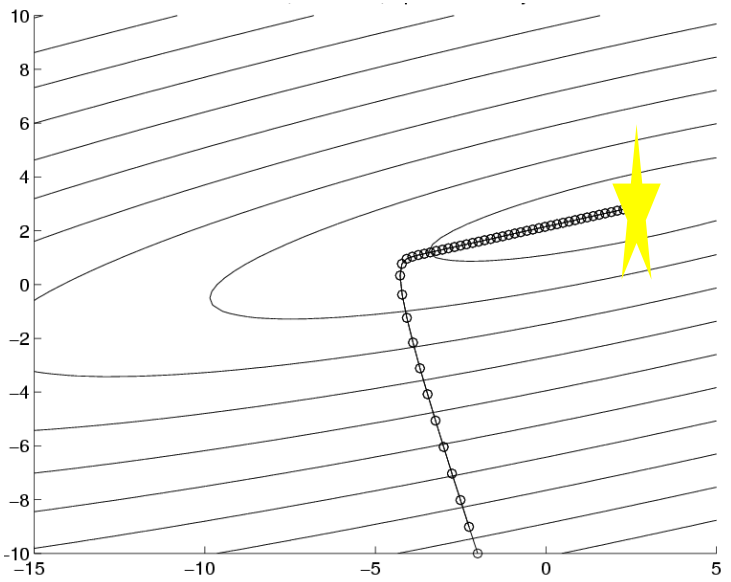


Numerical example

- Learning rate controls oscillation and speed



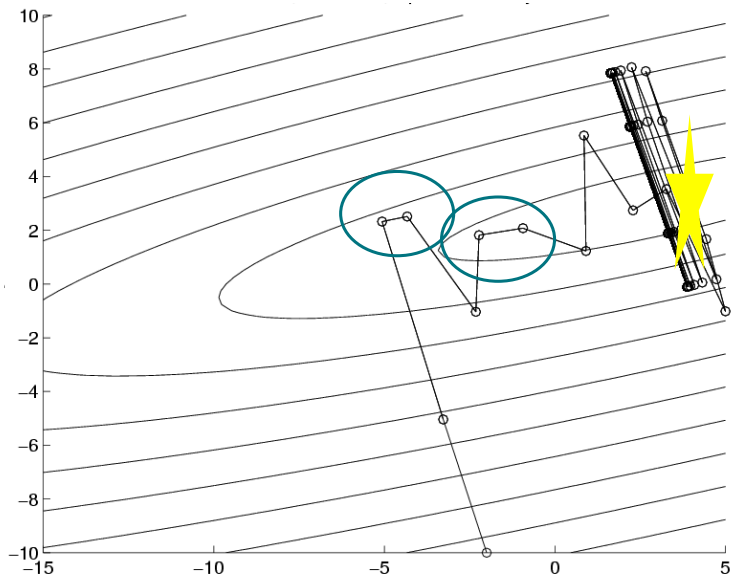
$\alpha = 1$: >100 iterations



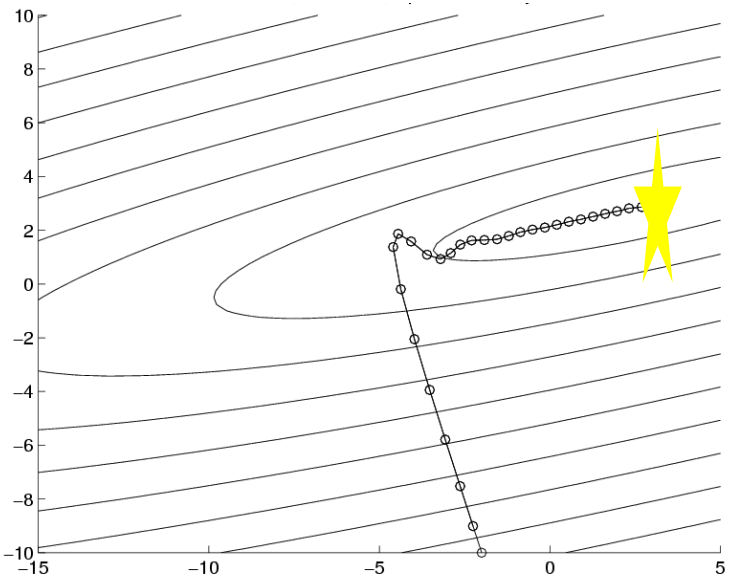
$\alpha = 0.1$: 52 iterations

Numerical example

- Momentum uses a bit of the previous step



$\alpha = 1, \eta = 0.5$: >100 iterations



$\alpha = 0.1, \eta = 0.5$: 29 iterations

D. Tuia. ECEO

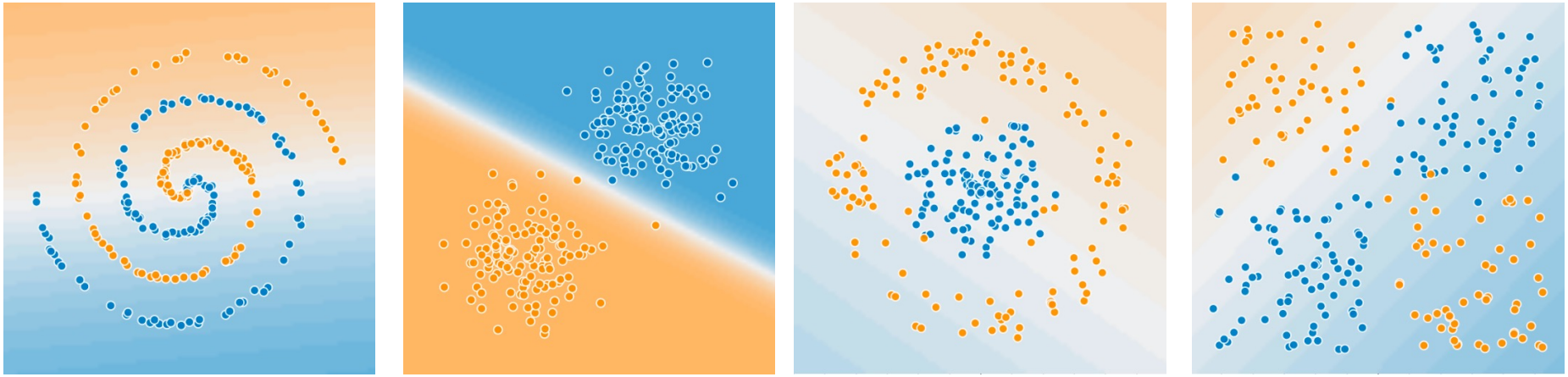
Neural networks (NN)

History and myths

Basic principles of NN

- Artificial neurons
- Gradient descent
- Multi-layer perceptron

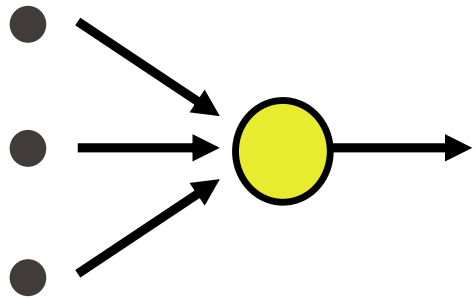
Which data sets can the perceptron handle?



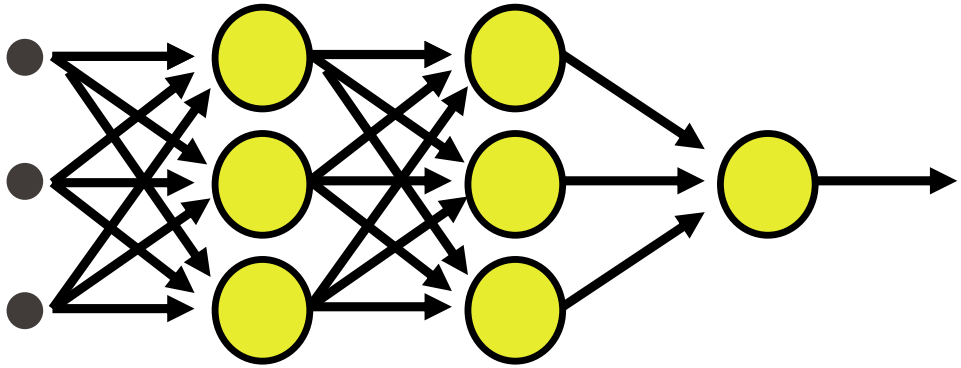
Back to the 80's

- The perceptron remained a linear classifier.
- It needed a nonlinear version.
- The Multilayer perceptron (MLP) was formalized in 1986 by Rumelhart and colleagues (including Geoffrey Hinton, considered to be one of the "fathers" of deep learning and Nobel prize for physics in 2024).

FROM the perceptron

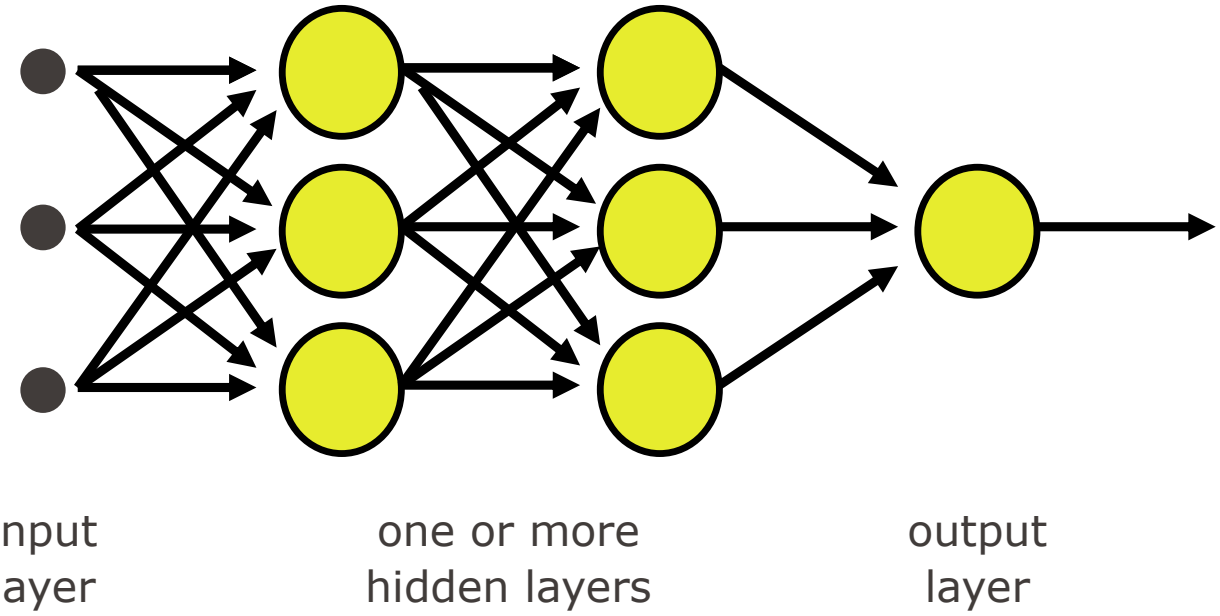


TO the MLP

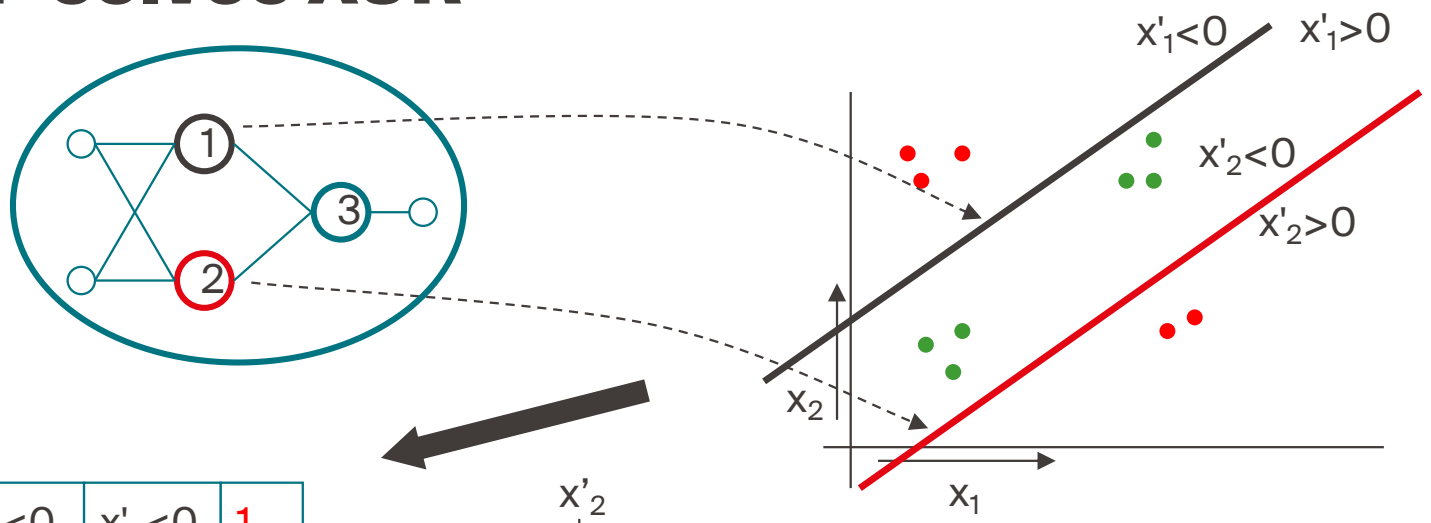


The multilayer perceptron (MLP)

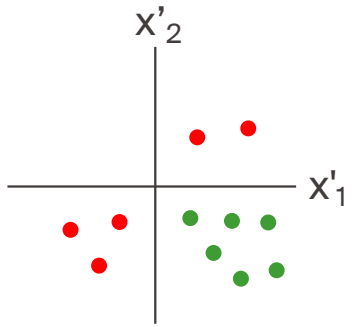
- It's a feed forward network: it goes from inputs to outputs, without loops
- Every neuron includes an activation function
  (e.g. sigmoid, see earlier slides 18-19)



Conceptual idea how MLP solves XOR

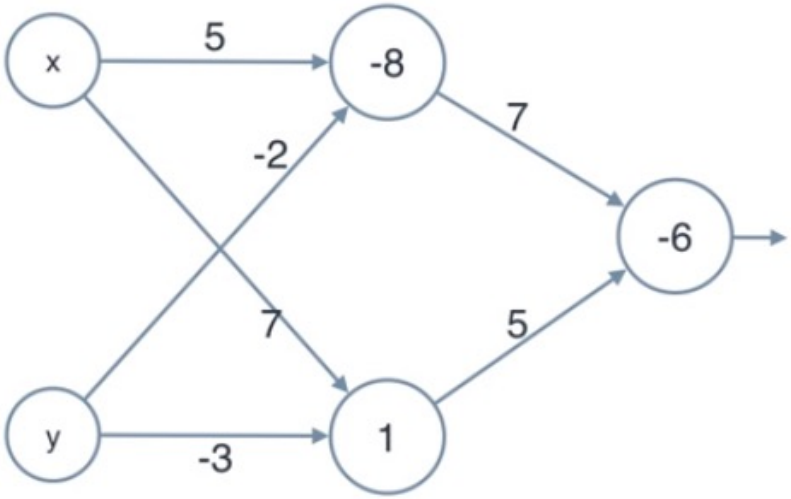


$x'_1<0$	$x'_2<0$	1
$x'_1<0$	$x'_2>0$	-
$x'_1>0$	$x'_2<0$	0
$x'_1>0$	$x'_2>0$	1

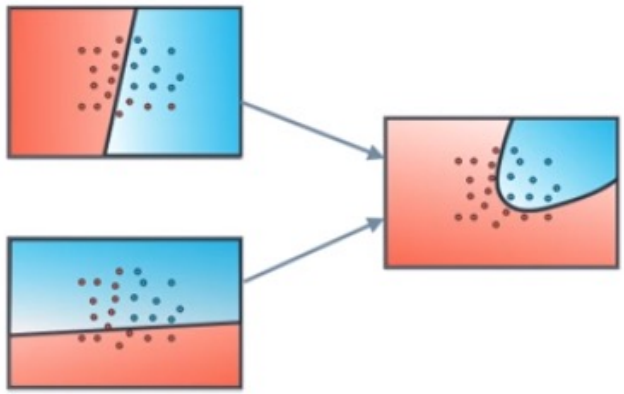


Can be solved with perceptron 3

Multi-Layer Perceptron



Combining regions



Courtesy Luis Serrano

Neural networks (NN)

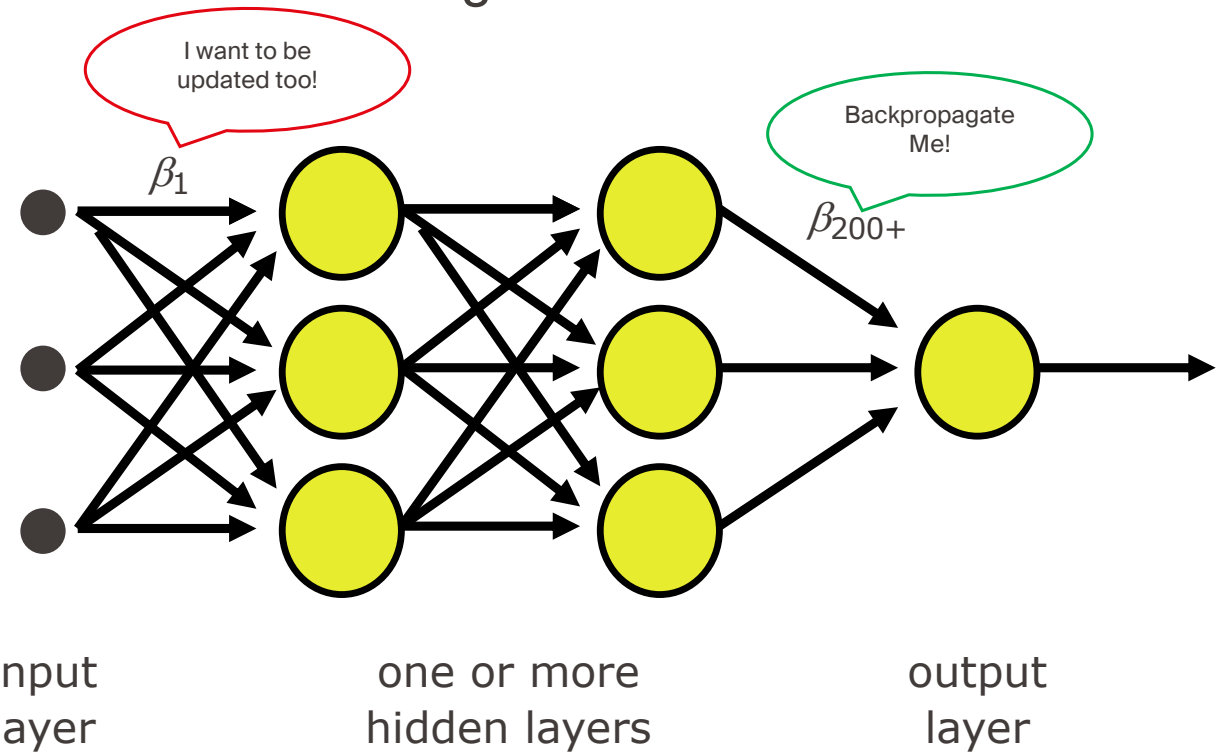
History and myths

Basic principles of NN

- Artificial neurons
- Gradient descent
- Multi-layer perceptron
- Training the MLP

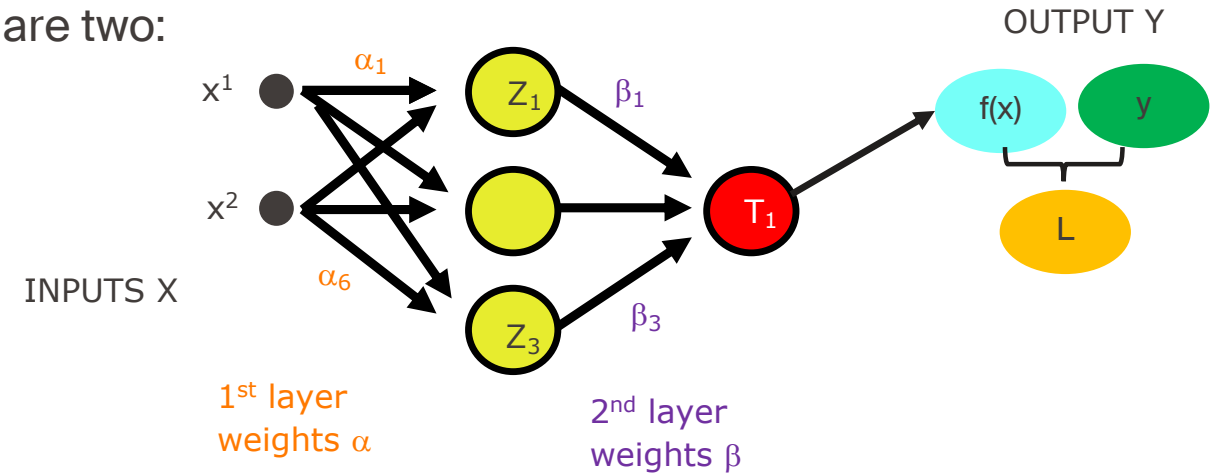
The multilayer perceptron (MLP)

- How to compute gradient descent on a cascade of operations?
- How to reach all trainable weights?



The phases of MLP training

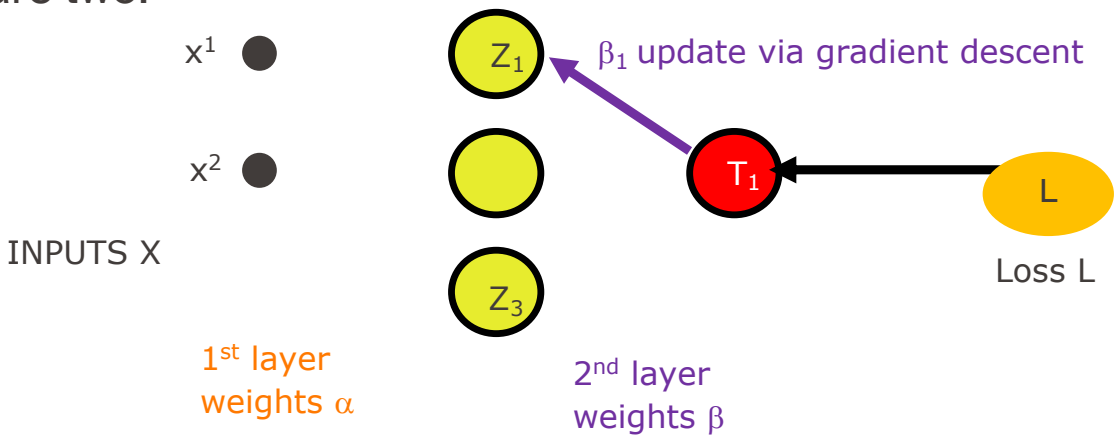
- There are two:



1. The **forward** pass: you put a data point in and obtain the prediction f(x).
   They you comparing with the truth y and compute the error (or loss, L)

The phases of MLP training

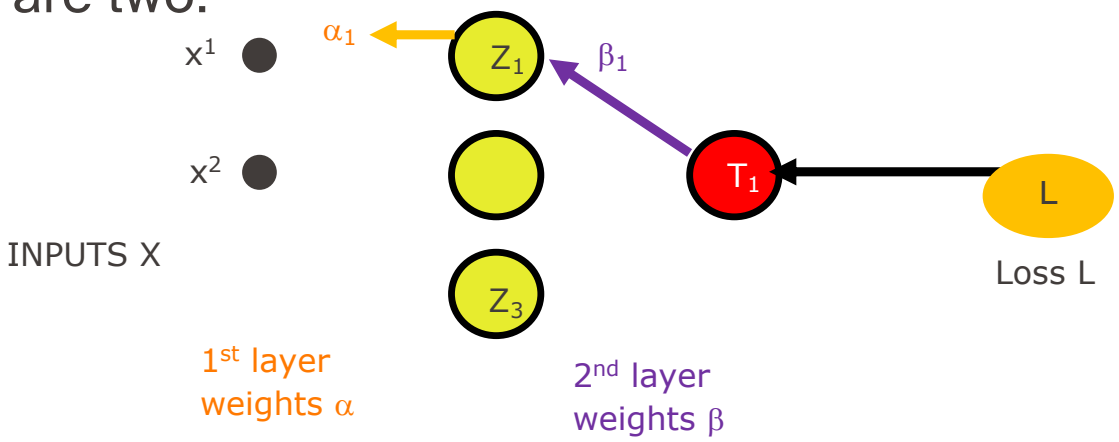
- There are two:



1. The forward pass: you put a data point in and obtain the prediction
2. The **Backward** pass: you update parameters by back-propagating the loss

The phases of MLP training

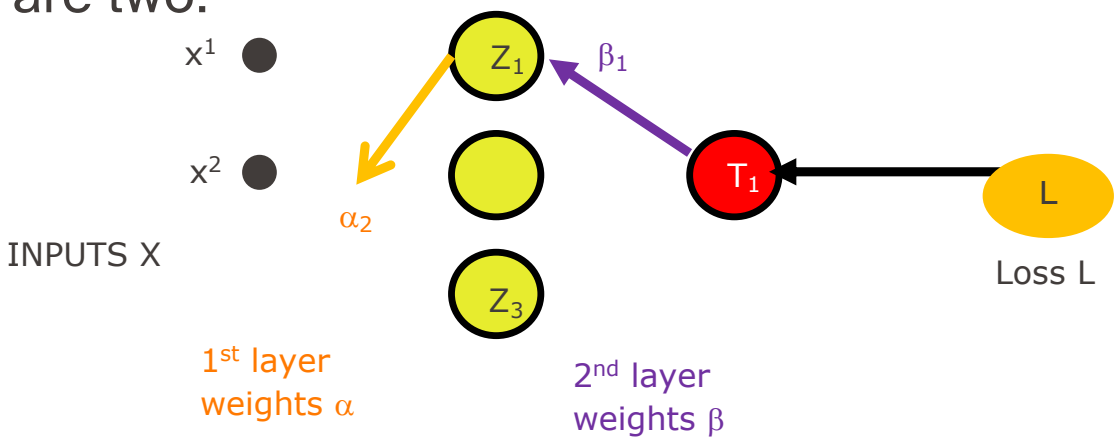
- There are two:



1. The forward pass: you put a data point in and obtain the prediction
2. The **Backward** pass: you update parameters by back-propagating the loss

The phases of MLP training

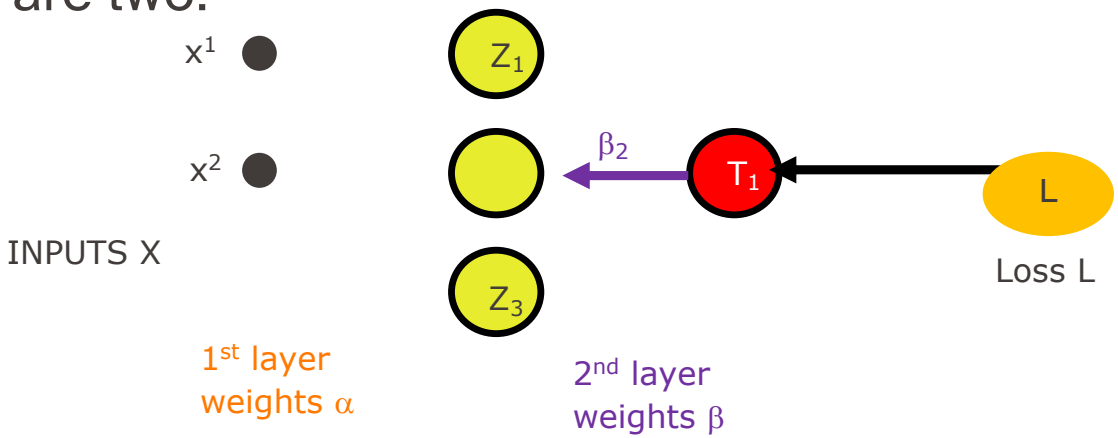
- There are two:



1. The forward pass: you put a data point in and obtain the prediction
2. The **Backward** pass: you update parameters by back-propagating the loss

The phases of MLP training

- There are two:



1. The forward pass: you put a data point in and obtain the prediction
2. The **Backward** pass: you update parameters by back-propagating the loss

And one to rule them all: the chain rule

- But how to reach weights that are earlier in the network?
- We use the chain rule
- It's a classical rule of derivatives:

"the derivative of a function applied to another function is the product of their derivatives"

$$\frac{\partial f(g(x))}{\partial x} = f'(g(x)) \cdot g'(x)$$

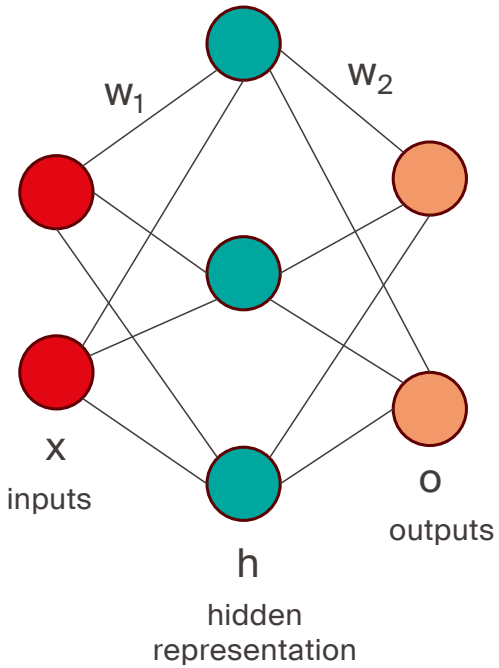
And one to rule them all: the chain rule

- But how to reach weights that are earlier in the network?
- We use the chain rule
- It's a classical rule of derivatives:

" the derivative of a function applied to another function is the product of their derivatives"

$$\frac{\partial f(g(h(x)))}{\partial x} = f'(g(h(x))) \cdot g'(h(x)) \cdot h'(x)$$

How to apply the chain rule: the network



Forward propagation:

$$z = w_1 x$$
$$h = \phi(z)$$
$$o = w_2 h$$

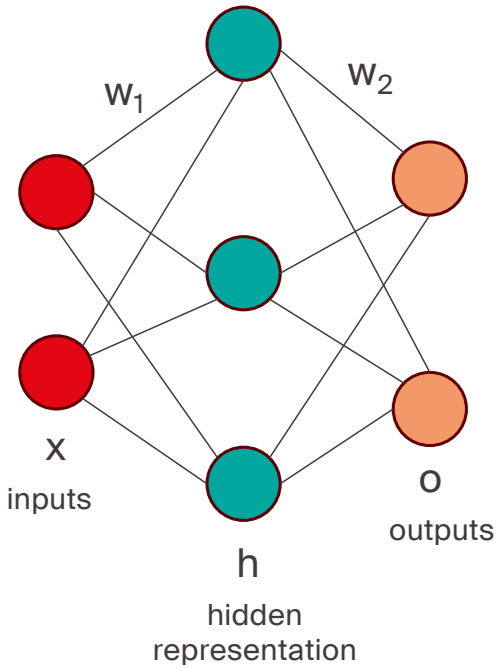
ϕ is an activation function (e.g. sigmoid)

Loss:

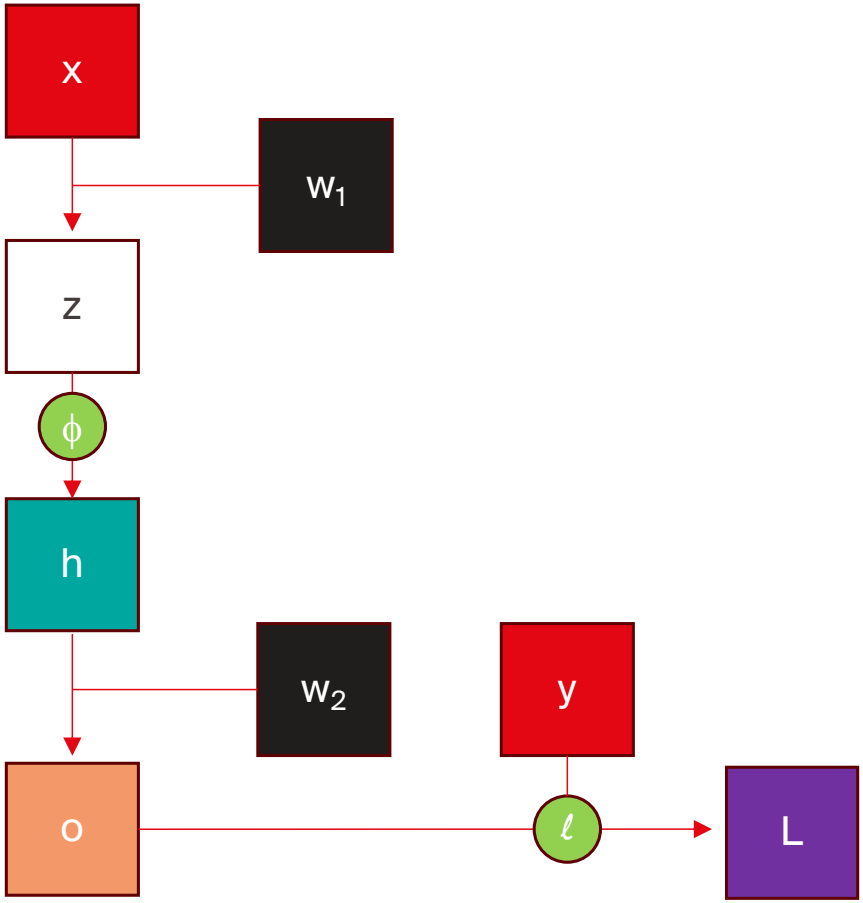
$$L = \ell(o, y)$$

y is the ground truth

How to apply the chain rule: the network is a computational graph



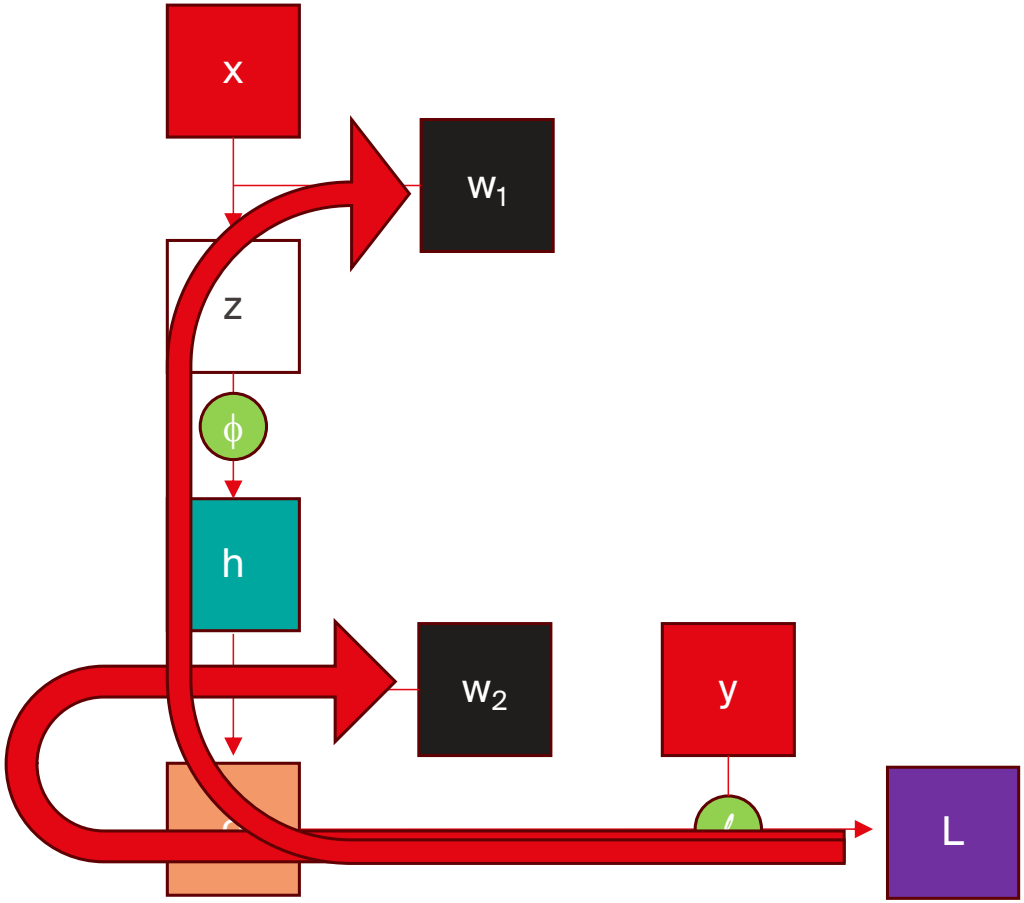
=



How to apply the chain rule: goal

- Find the best weights w_1 and w_2 to minimise Loss L

$$\frac{\partial L}{\partial w_1} \qquad \frac{\partial L}{\partial w_2}$$



How to apply the chain rule: w_2

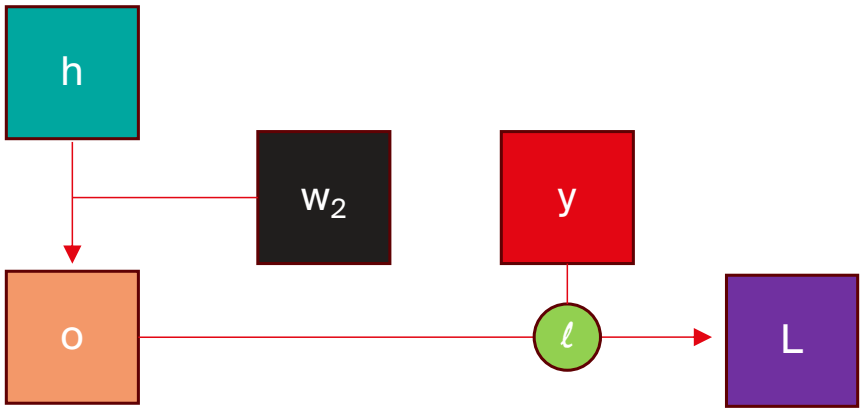
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial w_2}$$

$$= \frac{\partial L}{\partial o} \cdot h^\top$$

since

$$o = w_2 \cdot h$$

$$\frac{\partial o}{\partial w_2} = h^\top$$



How to calculate $\frac{\partial L}{\partial o}$

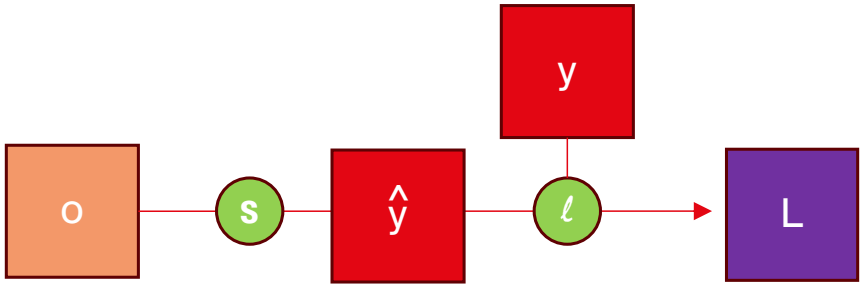
s = softmax = scales output between 0 and 1 (slide 19)

$$\hat{y}_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$$

L = classification loss = 0 if correct answer, > 0 otherwise

we use the cross entropy

$$\begin{aligned} L = \ell(\hat{y}, y) &= -\log p(y|x) \\ &= -\sum_i y_i \log \hat{y}_i \\ &= -\sum_i y_i \log [\frac{e^{o_i}}{\sum_j e^{o_j}}] \\ &= -\sum_i y_i [o_i - \log \sum_j e^{o_j}] \\ &= \sum_i y_i \log \sum_j e^{o_j} - \sum_i y_i o_i \\ &= \log \sum_j e^{o_j} - \sum_i y_i o_i \end{aligned}$$



How to calculate $\frac{\partial L}{\partial o}$

L = classification loss = 0 if correct answer,
> 0 otherwise

we use the cross entropy $$L = \ell(\hat{y}, y) = \log \sum_j e^{o_j} - \sum_i y_i o_i$$

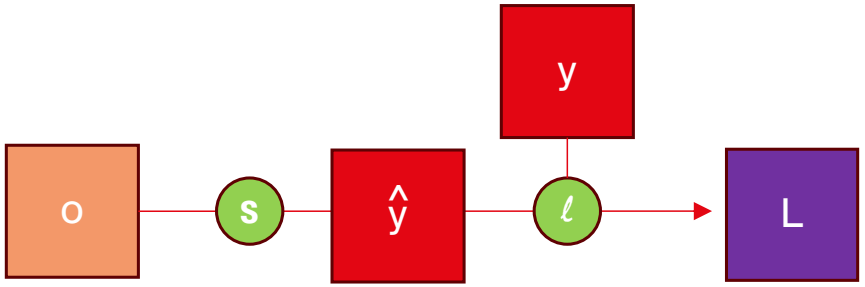
since $$\frac{\partial f}{\partial x} \log f(x) = \frac{1}{f(x)} f'(x)$$

$$\frac{\partial L}{\partial o_i} = \frac{e^{o_i}}{\sum_j e^{o_j}} - y_i$$

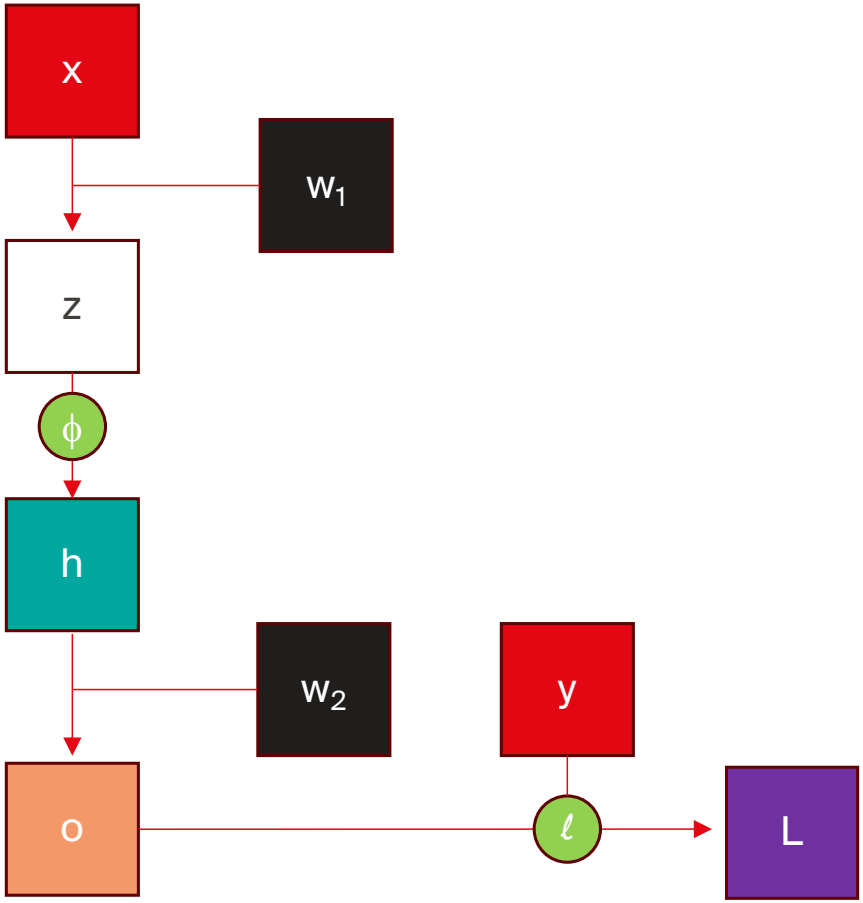
$$= \hat{y}_i - y_i$$

Prediction = between 0 and 1

True class = 1 or 0



How to apply the chain rule: w_1



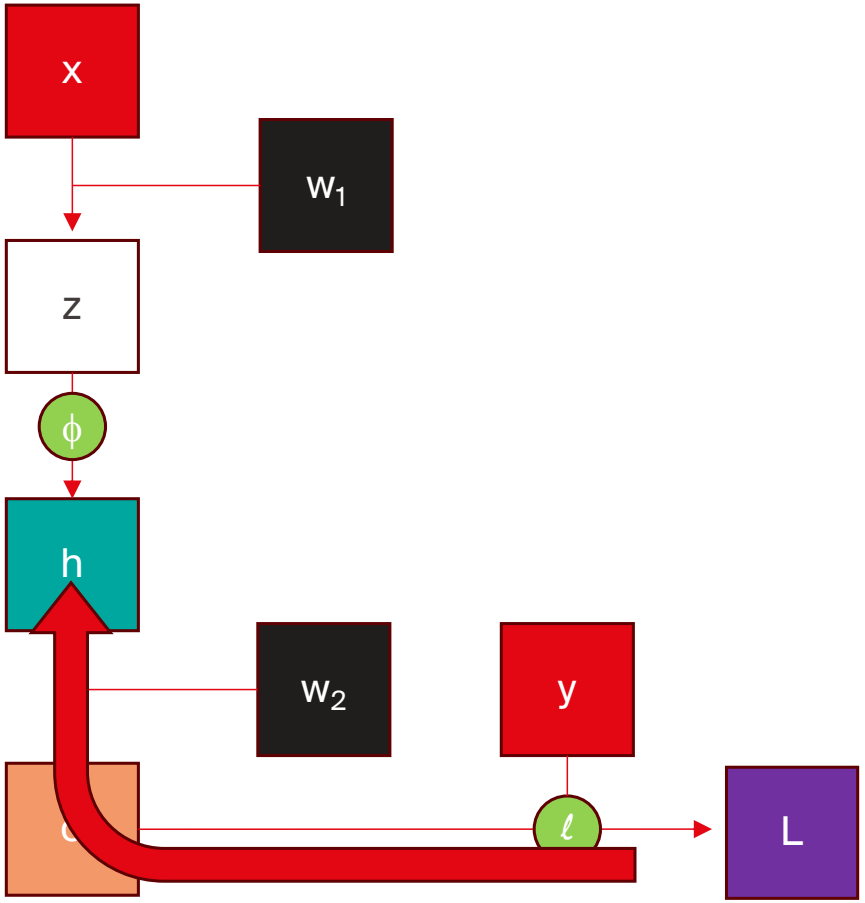
How to apply the chain rule: w_1

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial h}$$

$$= w_2^\top \frac{\partial L}{\partial o}$$

See previous slide 67

since $o = w_2 \cdot h$



How to apply the chain rule: w_1

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial h}$$

$$= w_2^\top \frac{\partial L}{\partial o}$$

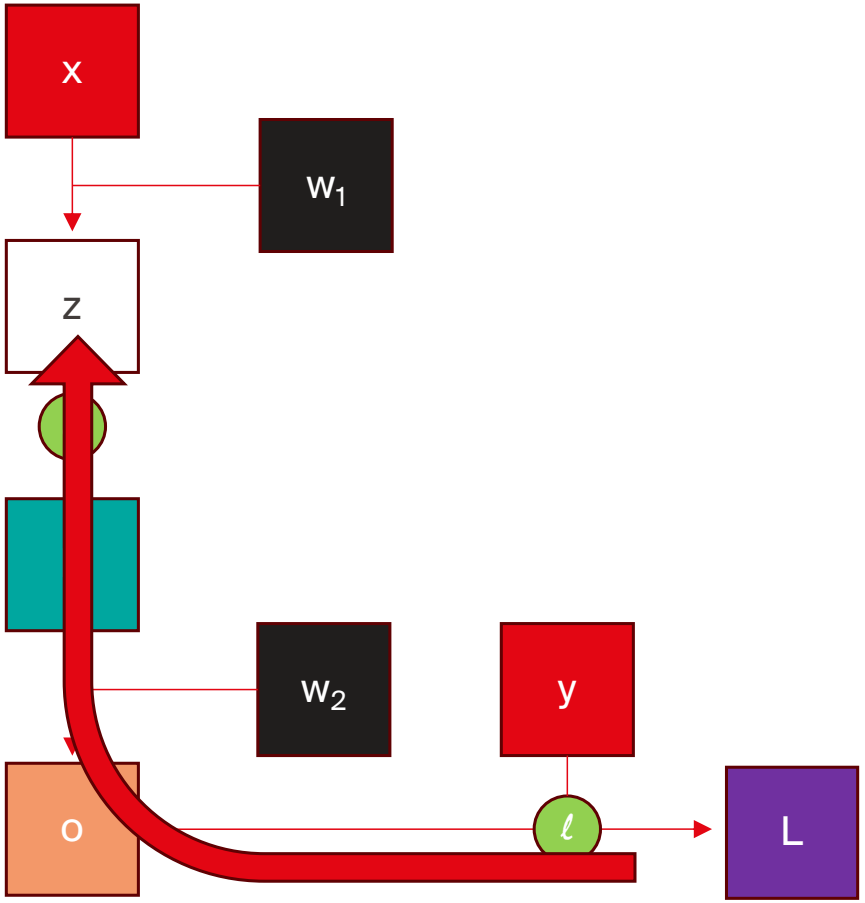
$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial z}$$

$$= \frac{\partial L}{\partial h} \odot \phi'(z)$$

????

since $h = \phi(z)$

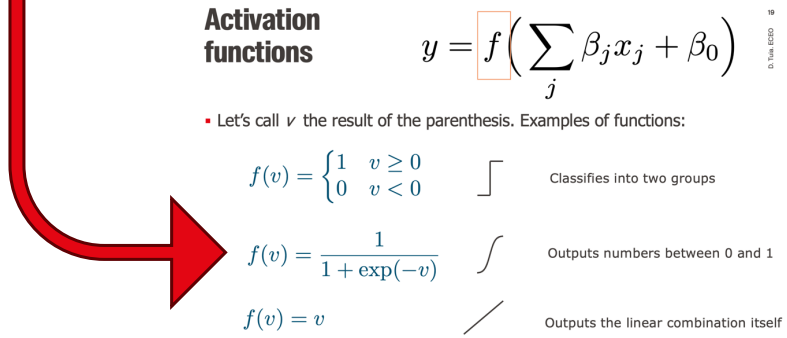
$$\frac{\partial h}{\partial z} = \phi'(z)$$

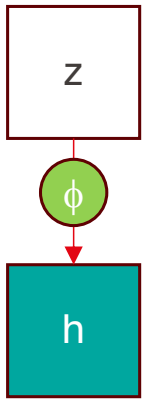


How to calculate $\phi'(z)$

How to calculate $\phi'(z)$

- Activation functions add nonlinearity to the network
- Otherwise it's a chain of linear operations
- We saw several functions at slide 19, e.g. the softmax a few slides ago (s)

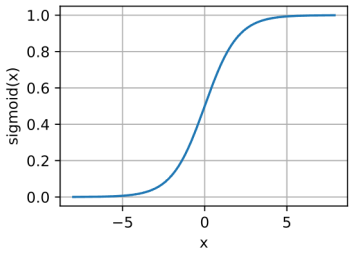




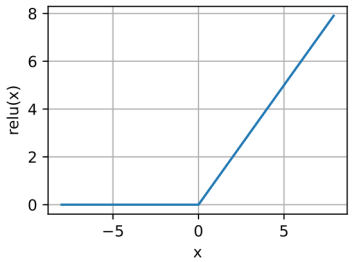
How to calculate $\phi'(z)$

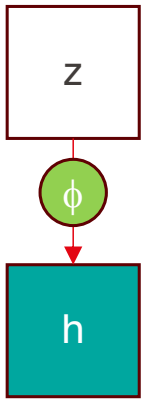
Sigmoid (or softmax)

- Output in [0,1]
- Close to a threshold function, with a smooth transition from 0 to 1
- In contrast to real threshold function, sigmoid is differentiable
- Often used at output neurons to get the probability for binary classification
- Usually not used in hidden layers: slow updating, vanishing gradient



vs





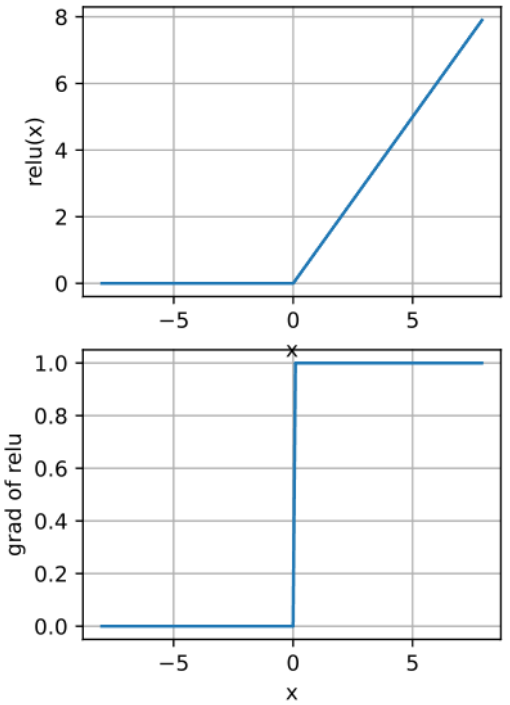
ReLU (Rectified Linear Unit)

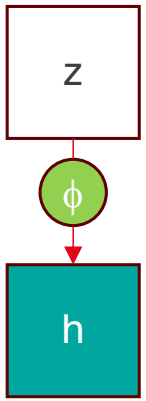
- $\mathrm{ReLU}(x) = \max(x, 0)$
- Its derivatives are simple: either they vanish (0) or they just let the argument through without modifications (1).
- This makes optimization better behaved and it mitigated well-documented problem of *vanishing gradients*
- So it is often used within intermediate neurons

How to calculate $\phi'(z)$ when using ReLU

- $\text{ReLU}(x) = \max(x, 0)$

- $\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$





How to apply the chain rule: w_1

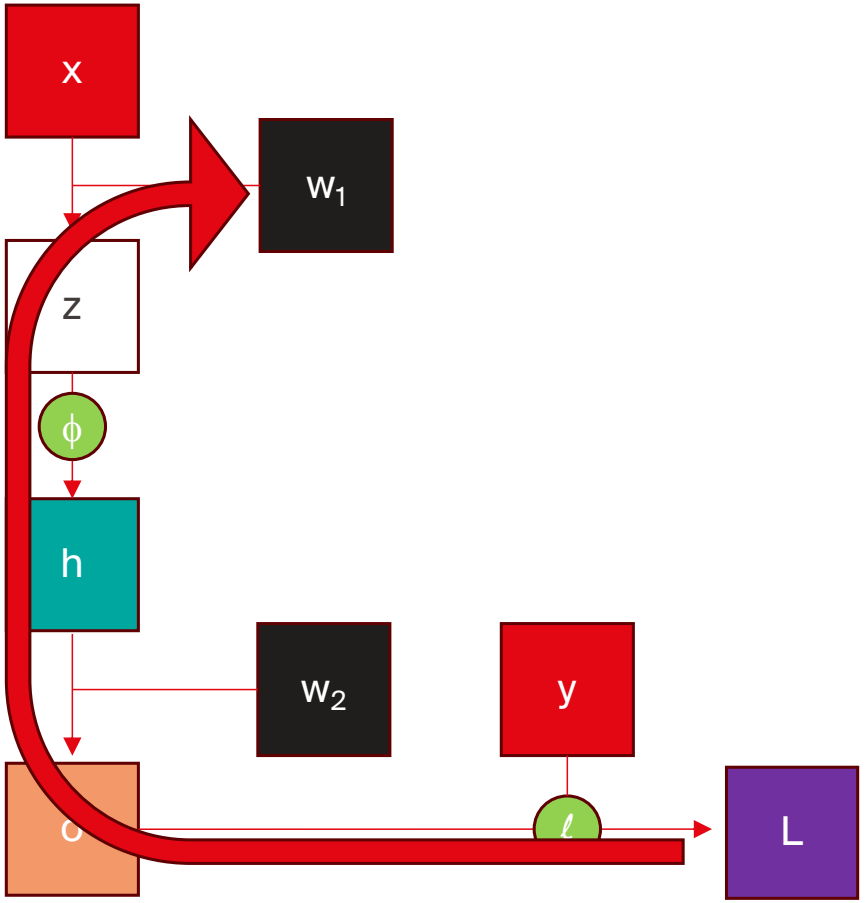
$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial h}$$
$$= w_2^\top \frac{\partial L}{\partial o}$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial z}$$
$$= \frac{\partial L}{\partial h} \odot \phi'(z)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$
$$= \frac{\partial L}{\partial z} x^\top$$

Since

$$z = w_1 \cdot x$$
$$\frac{\partial z}{\partial w_1} = x^\top$$



In summary

- NN update their parameters via gradient descent
- They have many parameters, recombined in a feed forward way
- They add nonlinearities at each step

- They can approximate very complex functions (so they overfit easily!)
- Remember to cross-validate!
  - Number of neurons
  - Number of layers
  - Learning rate
  - Momentum
  - ...

- In the next course we will see how to use this to build convolutional neural networks