# Artificial Neural Networks and Backpropagation

E. Decencière

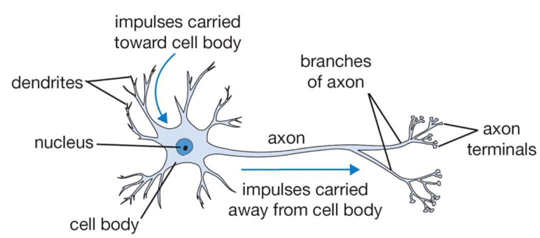
June 18, 2025

# Contents

# Chapter 1

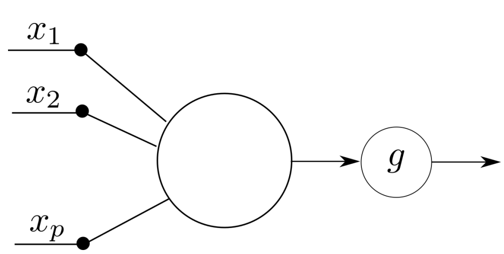
# Artificial Neurons

## 1.1 Biological Neuron




- A human neuron can have several thousand dendrites.
- The neuron sends a signal through its axon if, during a given interval of time, the net input signal (sum of excitatory and inhibitory signals received through its dendrites) is larger than a threshold.

## 1.2 Artificial Neuron

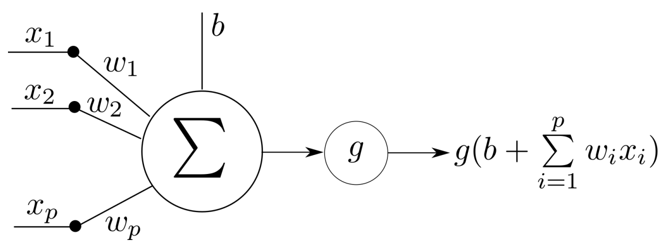


**General Principle**

An artificial neuron takes $p$ inputs $\{x_i\}_{1\leq i\leq p}$, combines them to obtain a single value, and applies an activation function $\mathtt{g}$ to the result.

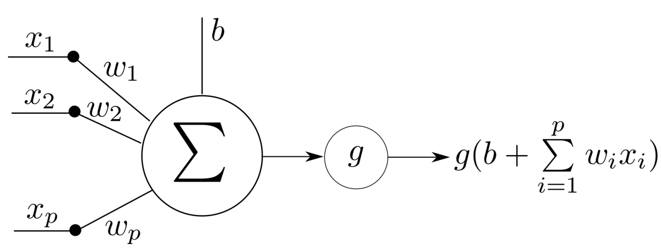
- The first artificial neuron model was proposed by [**?**].
- Input and output signals were binary.
- Input dendrites could be inhibitory or excitatory.

## 1.3 Modern Artificial Neuron




- The neuron computes a linear combination of the inputs $x_i$:
  - The weights $w_i$ are multiplied with the inputs.
  - The bias $b$ can be interpreted as a threshold on the sum.
- The activation function $\mathtt{g}$ decides, depending on its input, if a signal (the neuron's activation) is produced.

## 1.4 The Role of the Activation Function



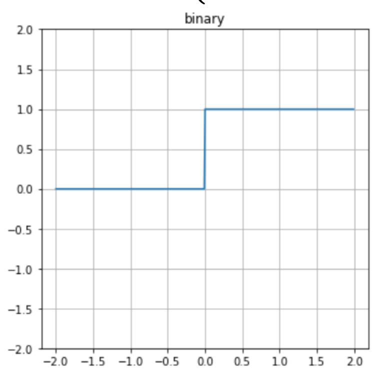

- The initial idea behind the activation function is that it works as a gate.
- If its input is "high enough", then the neuron is activated, i.e., a signal (other than zero) is produced.

- It can be interpreted as a source of abstraction: information considered as unimportant is ignored (or reduced).

## 1.5 Activation Functions

### 1.5.1 Binary Activation

$$\mathtt{g}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$




**Remarks**

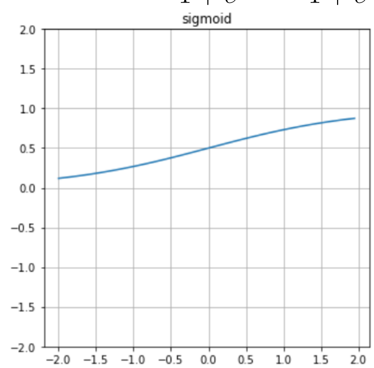
- Biologically inspired.
- \+ Simple to compute.
- \+ High abstraction.
- \- Gradient nil except on one point.
- In practice, almost never used.

### 1.5.2 Sigmoid Activation

**Remarks**

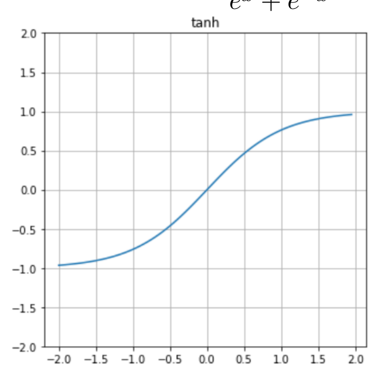
- \+ Similar to binary activation, but with usable gradient.
- Bijection between $\mathbb{R}$ and $]0, 1[$: no loss of information.
- \- Gradient tends to zero as we get away from zero.
- \- More computationally intensive.

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




### 1.5.3 Hyperbolic Tangent Activation

$$\mathrm{g}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$




**Remarks**

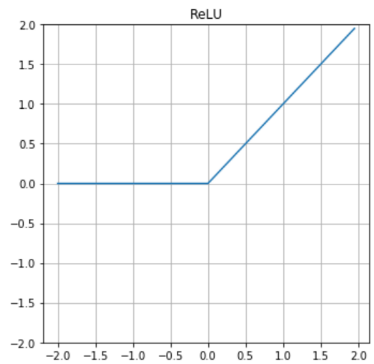
- Similar to sigmoid.

### 1.5.4 Rectified Linear Unit (ReLU) Activation

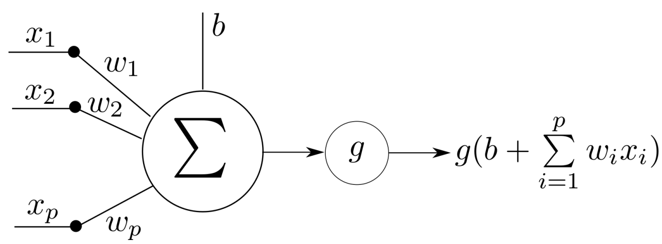
**Remarks**

+ Usable gradient when activated.
+ Fast to compute.
+ High abstraction.

$$\mathrm{g}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

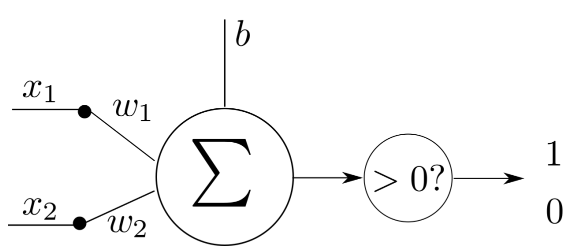



## 1.6 What Can an Artificial Neuron Compute?



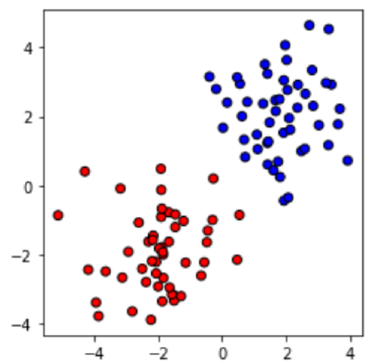

In $\mathbb{R}^p$, $b + \sum_{i=1}^{p} w_i x_i = 0$ corresponds to a hyperplane $H$. For a given point $\mathbf{x} = \{x_1, \ldots, x_p\}$, decisions are made according to the side of the hyperplane it belongs to.

## 1.7 Example in 2D

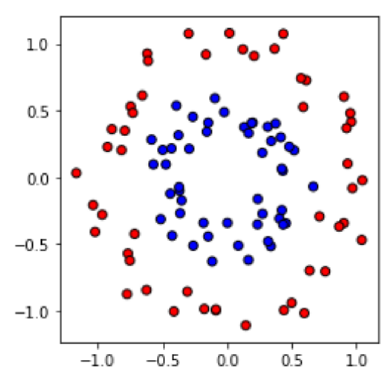


- $p = 2$: 2-dimensional inputs (can be represented on a screen!).
- Activation: binary.
- Classification problem.

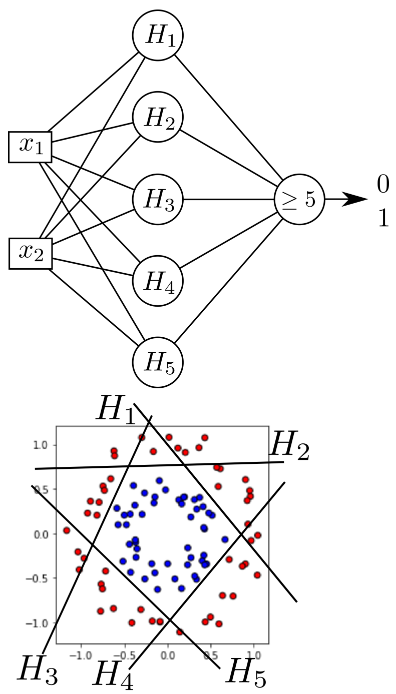
## 1.8 Gaussian Clouds




## 1.9 Circles



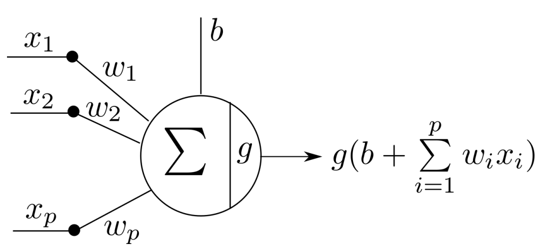
## 1.10 Solution with a Simple Neural Network

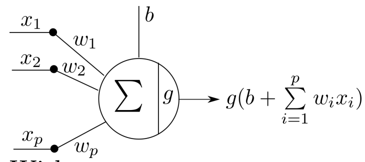



**Intuition**

Combining several neurons, one can build complex classifiers.

## 1.11 Compact Representation






With

$$\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix} = (w_1, \dots, w_p)^T$$

and

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} = (x_1, \dots, x_p)^T$$

## 1.12 Notations

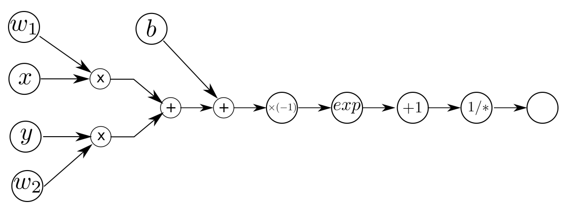
We can simply write:

$$\mathsf{g}(b + \sum_{i=1}^{p} w_i x_i) = \mathsf{g}(b + \mathbf{w}^T \mathbf{x})$$

# Chapter 2

# Artificial Neural Networks

## 2.1 Computational Graph

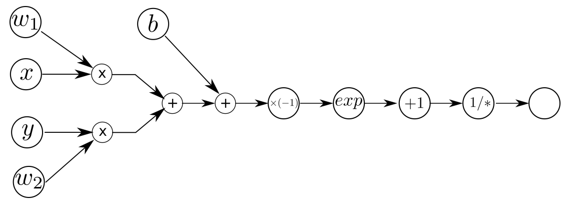



**Definition**

A computational graph is an acyclic directed graph such that:

- A node is a mathematical operator.
- To each edge is associated a value.
- Each node can compute the values of its output edges from the values of its input edges.
  - Nodes without input edges are *input nodes*. They represent the input values of the graph.
  - Similarly, output values can be held in the *output nodes*.

- In this course, we will only consider *acyclic* computational graphs.
- Computing a *forward pass* through the graph means choosing its input values, and then progressively computing the values of all edges.

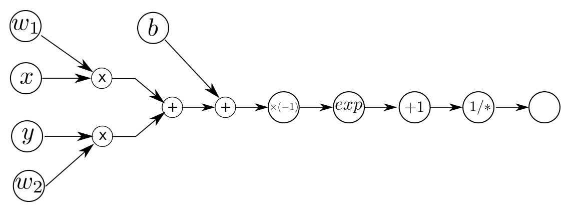



## 2.2 Computational Graph Example

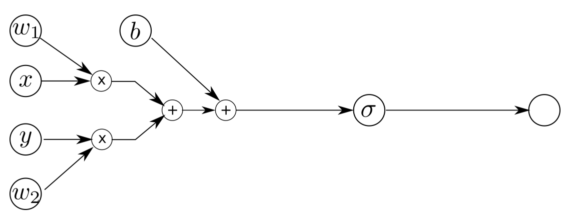
Computational graph of:

$$\sigma(w_1 x + w_2 y + b)$$

where $\sigma$ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$




The graph can be represented at different levels of detail:




## 2.3 First Architectures

### 2.3.1 Neural Network (NN)

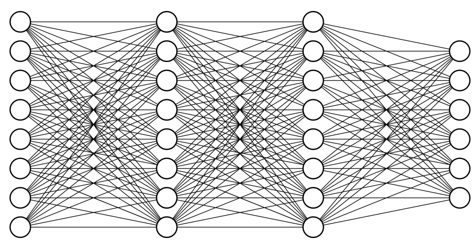
**Definitions**

- An artificial neural network is a computational graph, where the nodes are artificial neurons.
- The input layer is the set of neurons without incoming edges.
- The output layer is the set of neurons without outgoing edges.

### 2.3.2 Network Layers

**Definition**

- Neurons are usually organized in layers.
- Any layers other than input and output layers are called hidden layers.



### 2.3.3 Other Types of Neural Networks

In the following of this course, except when otherwise specified, all NNs will be feed-forward. Indeed, this is the preferred type of NN for image processing.

**What about other architectures?**

- Recurrent neural networks (RNN).
- Long short-term memory networks (LSTM).

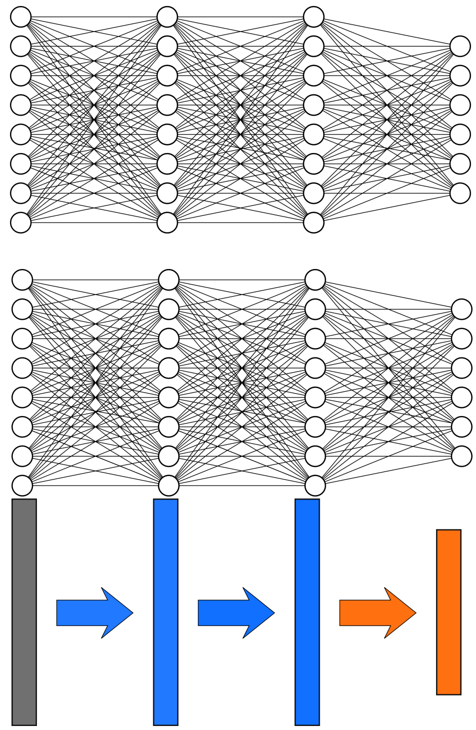
+ More powerful than feed-forward NNs.

- Complex dynamics; more difficult to train.

- Mainly used for processing temporal data.

### 2.3.4 Fully-Connected Layer

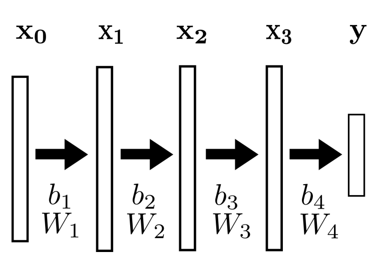
- A layer is said to be fully-connected (FC) if each of its neurons is connected to all the neurons of the previous layer.
- If a FC layer contains $r$ neurons, and the previous layer $q$, then its weights are a 2D dimensional array (a matrix) of size $q \times r$.



### 2.3.5 Graphical Representation of NNs

- Data is organized into arrays, linked with operators.
- A layer corresponds to an operator between arrays.

### 2.3.6 The Equations of a Fully Connected Neural Network



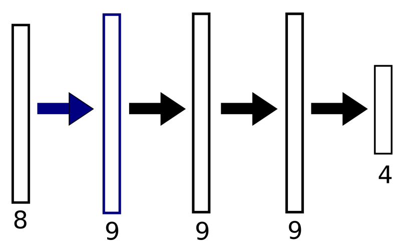

For $i \in \{1, 2, 3\}$:

$$\mathbf{x}_i = \mathbf{g}_i(\mathbf{x}_{i-1}^t \mathbf{W}_i + \mathbf{b}_i)$$

$$\mathbf{y} = \mathrm{g}_4(\mathbf{x}_4^t \mathbf{W}_4 + \mathbf{b}_4)$$

What would happen if all activation functions $\mathbf{g}_i$ were equal to the identity function?

### 2.3.7 Number of Parameters




- How many parameters does the above network contain?

A/ 270

B/ 274

C/ 301

D/ 39

### 2.3.8 Batch Processing

In a learning context, one may want to process $n$ vectors of length $p$ at the same time. They can be grouped into a matrix $\mathbf{X}$ of size $n \times p$. The $n$ corresponding outputs $\mathbf{y}_i$ can also be grouped into a matrix $\mathbf{Y}$. The resulting equations are:

For $i \in \{1, 2, 3\}$:
$$\mathbf{X}_i = \mathrm{g}_i(\mathbf{X}_{i-1} \mathbf{W}_i + \mathbf{b}_i)$$

$$\mathbf{Y} = \mathrm{g}_4(\mathbf{X}_4 \mathbf{W}_4 + \mathbf{b}_4)$$

This can accelerate processing thanks to hardware architectures such as Graphical Processing Units (GPUs) but can also play an important role in optimization.

### 2.3.9 From Neurons to Arrays

- Neurons are organized into arrays (0-D, 1-D, 2-D, 3-D ...).

- Artificial neural networks can be seen as computational graphs processing arrays.

## 2.4 Modelling Power

### 2.4.1 A Composition of Differentiable Functions

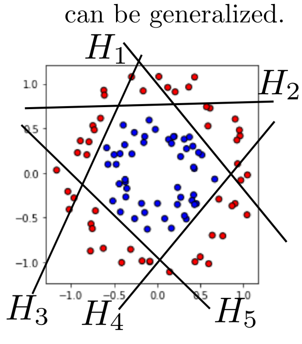
- The functions composing an artificial neural network are differentiable (almost everywhere), so that it can be optimized via gradient descent. Therefore, an ANN is differentiable (almost everywhere).

- In fact, any continuous function on a closed bounded domain can be approached within any error margin by an artificial neural network.

### 2.4.2 Universal Approximation Theorem

- We have previously seen that a neuron can be used as a linear classifier and that combining several of them one can build complex classifiers.

- We will see that this observation can be generalized.



Let $f$ be a continuous real-valued function of $[0,1]^p$ ($p \in \mathbb{N}^*$) and $\epsilon$ a strictly positive real. Let $\mathtt{g}$ be a non-constant, increasing, bounded real function (*the activation function*).
Then there exists an integer $q$, real vectors $\{\mathbf{w}_i\}_{1 \le i \le q}$ of $\mathbb{R}^p$, and reals $\{b_i\}_{1 \le i \le q}$ and $\{v_i\}_{1 \le i \le q}$ such that for all $\mathbf{x}$ in $[0,1]^p$:

$$\left| f(\mathbf{x}) - \sum_{i=1}^{q} v_i \mathtt{g}(\mathbf{w}_i \mathbf{x} + b_i) \right| < \epsilon$$

A first version of this theorem, using sigmoidal activation functions, was proposed by [?]. The version above was demonstrated by [?].

### 2.4.3 Universal Approximation Theorem: What Does It Mean?

$$\left| f(\mathbf{x}) - \sum_{i=1}^{q} v_i \mathtt{g}(\mathbf{w}_i \mathbf{x} + b_i) \right| < \epsilon$$

This means that function $f$ can be approximated with a neural network containing:

- An input layer of size $p$.
- A hidden layer containing $q$ neurons with activation function $\mathtt{g}$, weights $\mathbf{w}_i$ and biases $b_i$.
- An output layer containing a single neuron, with weights $v_i$ (and an identity activation function).

### 2.4.4 Universal Approximation Theorem in Practice

- The number of neurons increases very rapidly with the complexity of the function.
- Empirical evidence has shown that multi-layer architectures give better results.
- For learning tasks, the function to be modelled is only known on a finite number of points.

A NN can potentially have a lot of parameters. How can we set them?

# Chapter 3

# Training a Neural Network

## 3.1 Introduction

- We have seen that NNs have a lot of potential. However, how can the parameters $\boldsymbol{\theta} = (\mathbf{W}_i, \mathbf{b}_i)$ be set?
- What is our objective?

## 3.2 Supervised Learning Problem

We recall that our training set contains $n$ samples:

$$(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathcal{Y}$$

Where $\mathcal{Y} = \mathbb{R}$ in the regression case and $\mathcal{Y} = \{0, 1\}$ in the binary classification case.

We choose a loss function $l$ and we choose a family $f_{\boldsymbol{\theta}}$ of functions from $\mathbb{R}^p$ into $\mathbb{R}$, depending on a set of parameters $\boldsymbol{\theta}$, and find the value $\boldsymbol{\theta}^*$ of $\boldsymbol{\theta}$ that minimizes:

$$\frac{1}{n} \sum_{i=1}^{n} l(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)$$

For the sake of simplicity, we have dropped the regularization term.

## 3.3 Loss Functions

### 3.3.1 Choosing a Loss Function

- The choice of the loss function depends on the type of problem (regression or classification) and is tightly linked to the application.

### 3.3.2 The Standard Loss for Regression Problems: Squared Error Loss

In the regression case, we have $\mathcal{Y} = \mathbb{R}$.

**Squared Error Loss**

$$l(f_{\boldsymbol{\theta}}(x), y) = (f_{\boldsymbol{\theta}}(x) - y)^2$$

### 3.3.3 Binary Cross-Entropy

In the simplest classification case, we have $\mathcal{Y} = \{0, 1\}$.

**Binary Cross-Entropy Loss**

$$l(f_{\boldsymbol{\theta}}(x), y) = -y \log(f_{\boldsymbol{\theta}}(x)) - (1 - y) \log(1 - f_{\boldsymbol{\theta}}(x))$$

- For this expression to be mathematically sound, $f_{\boldsymbol{\theta}}(x)$ must belong to $]0, 1[$. In practice, in the case of NN, this can be achieved by using a sigmoid as last activation.
- Note that the expression above is equivalent to:

$$l(f_{\boldsymbol{\theta}}(x), y) = \begin{cases} -\log(1 - f_{\boldsymbol{\theta}}(x)) & \text{if } y = 0 \\ -\log(f_{\boldsymbol{\theta}}(x)) & \text{if } y = 1 \end{cases}$$

## 3.4 How to Minimize the Loss?

## 3.5 Gradient Descent

### 3.5.1 Gradient Descent in the Scalar Case

### 3.5.2 Gradient Descent

**Definition**

Gradient descent is an optimization algorithm. For a differentiable function $L$, a positive real $\eta$ (the learning rate) and a starting point $\boldsymbol{\theta}_0$, it computes a sequence of values:

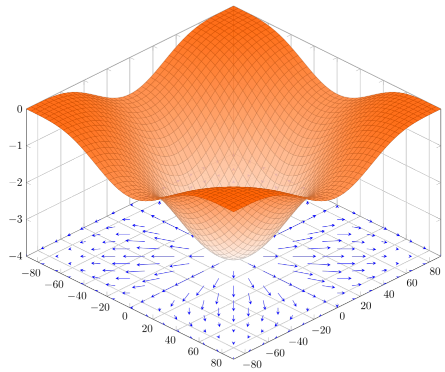
$$\forall t \in \mathbb{N} : \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t)$$

**Property**

For a given $t$, if $\eta$ is small enough, then:

$$L(\boldsymbol{\theta}_{t+1}) \leq L(\boldsymbol{\theta}_t)$$

Gradient descent is an essential tool in optimization.



**Definition: Gradient**

Let $L$ be a differentiable function from $\mathbb{R}^n$ into $\mathbb{R}$. Its gradient $\nabla L$ is:

$$\nabla L(x) = \begin{pmatrix} \frac{\partial L}{\partial \mathbf{x}_1}(x) \\ \vdots \\ \frac{\partial L}{\partial \mathbf{x}_n}(x) \end{pmatrix}$$

### 3.5.3 Gradient Descent: Stopping Criteria

In practice:

$$\forall t \in [0, \ldots, E-1] : \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t)$$

- Choose $E$ (the number of epochs) based on experience.
- Track the quality of the model using a validation dataset and stop when the validation loss does not improve.

### 3.5.4 Towards Stochastic Gradient Descent

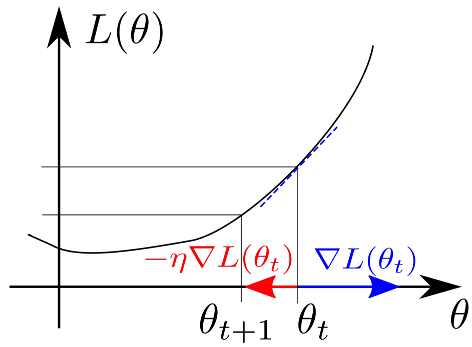
The loss function we initially defined depends on the whole training set:

$$L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$$

- If $n$ is very large, computing $L$ is impractical.
- A computation on the whole training set leads to a single update of the model parameters - convergence can therefore be slow.

### 3.5.5 Stochastic Gradient Descent

In stochastic gradient descent, the parameters are updated for each sample $i$.




$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t)$$

$L$ is called the learning rate.

- First, the loss is computed

$$L(\boldsymbol{\theta}_t) = l(y_i, f(\mathbf{x}_i, \boldsymbol{\theta}_t))$$

- The gradient $\nabla L(\boldsymbol{\theta}_t)$ is computed and
- Finally the parameters are updated:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t)$$

- Note that the learning rate $\eta$ can have a different value than in classic gradient descent.

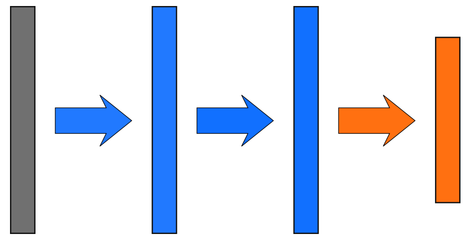
### 3.5.6 Mini-Batch Processing

- One can (and often does) choose an intermediate solution between the full gradient and the stochastic gradient: mini-batch gradient.
- The training database is then separated into subsets containing $m$ samples ($m < n$).
- This has a regularization effect on the optimization with respect to the stochastic gradient and speeds up computation thanks to the vectorization capacity of hardware architectures such as GPUs.

## 3.6 Backpropagation

### 3.6.1 Gradient Descent Applied to Neural Networks

- In the case of neural networks, the loss $L$ depends on each parameter $\boldsymbol{\theta}_i$ via the composition of several functions.
- Analytical derivation is possible, but complex - and has to be re-computed when the network architecture is modified.
- Using the chain rule theorem leads to an efficient solution: backpropagation.



### 3.6.2 Chain Rule Theorem

Let $f_1$ and $f_2$ be two differentiable real functions ($\mathbb{R} \rightarrow \mathbb{R}$). Then for all $x$ in $\mathbb{R}$:

$$(f_2 \circ f_1)'(x) = (f_2' \circ f_1)(x).f_1'(x)$$

**Leibniz Notation**

Let us introduce variables $x$, $y$ and $z$:

$$x \xrightarrow{f_1} y \xrightarrow{f_2} z$$

Then:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y} \cdot \frac{\mathrm{d}y}{\mathrm{d}x}$$

### 3.6.3 The Backpropagation Algorithm

- The backpropagation algorithm is used in a neural network to efficiently compute the partial derivatives of the loss with respect to each parameter of the network.
- One can trace the origins of the method to the sixties.
- It was first applied to NN in the eighties [**?**, **?**].

### 3.6.4 The Backpropagation Algorithm: Intuition

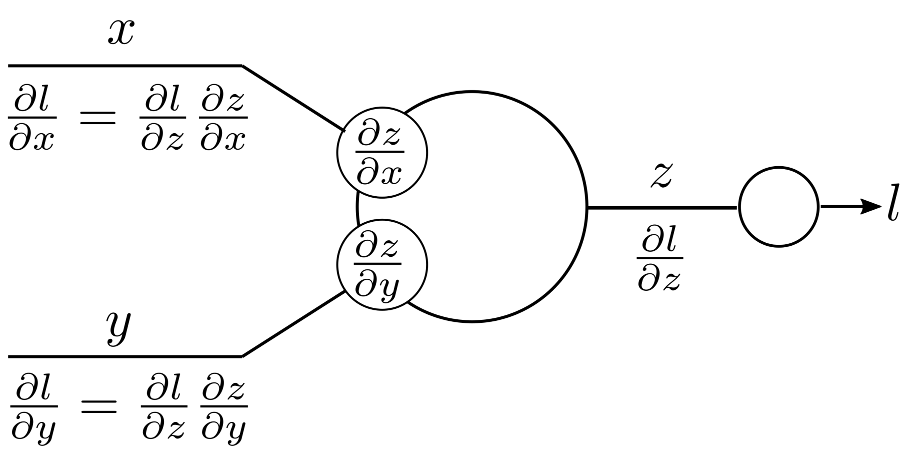
- Given a computational graph, the main idea is to compute the local derivatives during a forward pass.
- Then, during a backward pass, the partial derivatives of the loss with respect to each parameter are computed.

### 3.6.5 Simple Backpropagation Example

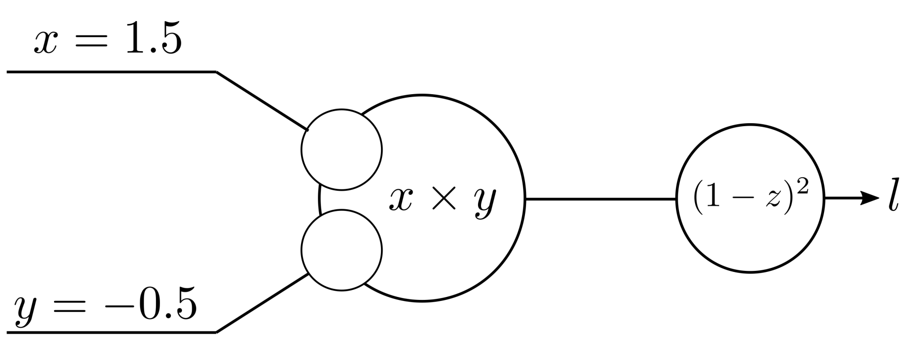
$$x \xrightarrow{\frac{\partial y}{\partial x}} y \xrightarrow{\frac{\partial z}{\partial y}} z \xrightarrow{\frac{\partial l}{\partial z}} l$$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial x} \qquad \frac{\partial l}{\partial y} = \frac{\partial l}{\partial z}\frac{\partial z}{\partial y} \qquad \frac{\partial l}{\partial z} = \frac{\partial l}{\partial l}\frac{\partial l}{\partial z} \qquad \frac{\partial l}{\partial l} = 1$$

### 3.6.6 Backpropagation Through a Neuron




### 3.6.7 Exercise 1



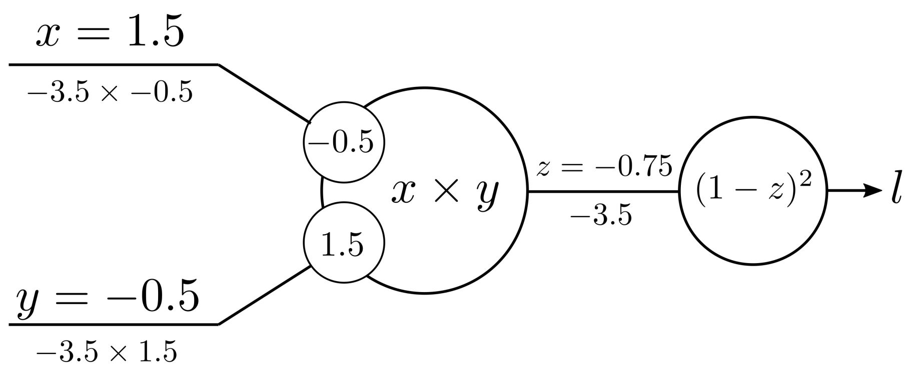

$\frac{\partial l}{\partial x}$ is equal to:

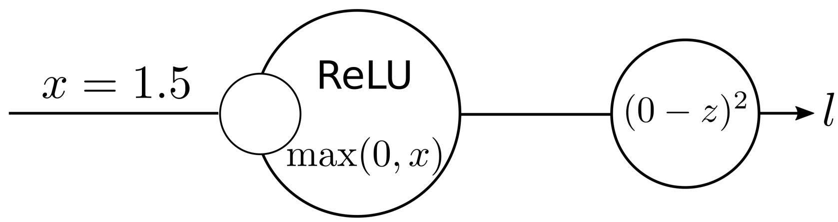
A/ 1.75

B/ $-2.25$

C/ $-1.5$

D/ 0.75




### 3.6.8 Exercise 1: Solution

### 3.6.9 Exercise 2

### 3.6.10 Exercise 2: Solution

### 3.6.11 Exercise 3

### 3.6.12 Exercise 3: Solution

### 3.6.13 Exercise 4

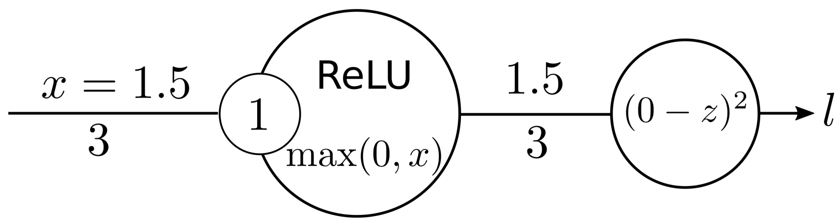
**Quiz**

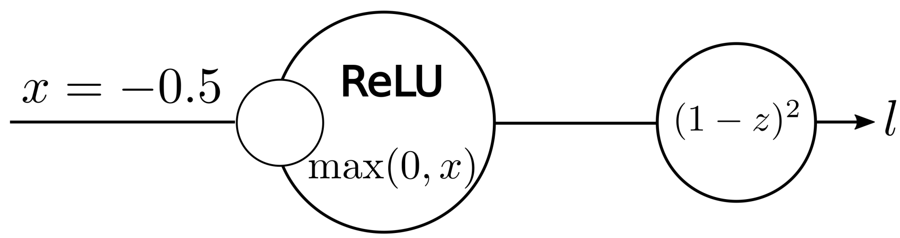
What's the value of $\frac{\partial l}{\partial x}$?

### 3.6.14 Exercise 4: Solution

### 3.6.15 Vector Calculus

- $L$ and $V$ are differentiable functions.




### 3.6.16 Matrix Calculus

- Function $\mathcal{M}$ is differentiable.

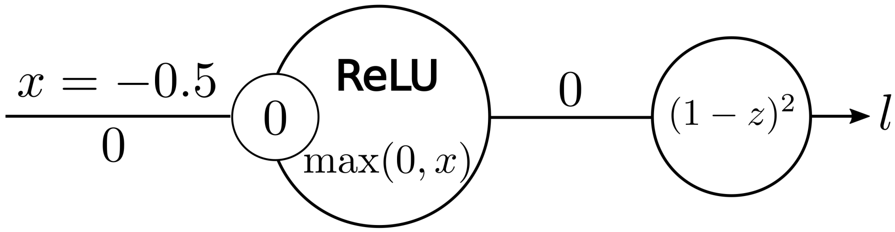
$$\begin{aligned} \mathcal{M} : \mathbb{R}^{(m,n)} &\longrightarrow \mathbb{R}^{(p,q)} \\ \mathbf{X} &\longmapsto \mathcal{M}(M) \end{aligned}$$

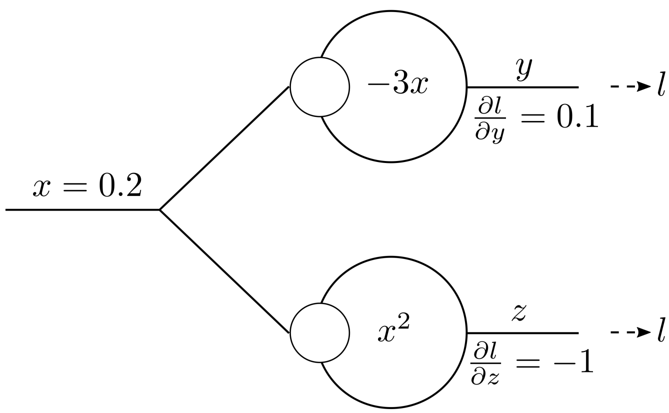
$$\boxed{\frac{\partial \mathcal{M}}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial \mathcal{M}_{1,1}}{\partial \mathbf{X}} & \dots & \frac{\partial \mathcal{M}_{1,q}}{\partial \mathbf{X}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{M}_{p,q}}{\partial \mathbf{X}} & \dots & \frac{\partial \mathcal{M}_{p,q}}{\partial \mathbf{X}} \end{pmatrix}}$$

This is an array of size $(m, n, p, q)$.

### 3.6.17 Backpropagation Through an Activation Function $g$

- Computing the full matrix $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ is impractical.

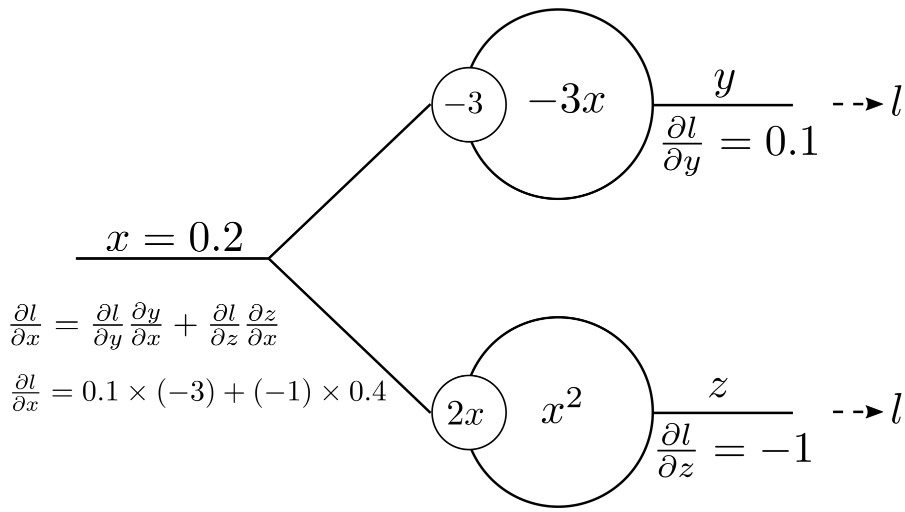



- But here $Y_{i,j}$ only depends on $X_{i,j}$: $Y_{i,j} = \mathsf{g}(X_{i,j})$.
- Therefore: $\frac{\partial \mathbf{Y}_{i,j}}{\partial \mathbf{X}_{i,j}} = \mathsf{g}'$.
- Finally:
$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \odot \mathsf{g}'(\mathbf{X}) ,$$
where $\odot$ is the term by term matrix multiplication or Hadamard matrix multiplication.

### 3.6.18 Backpropagation Through an Activation Function $g$

We will abusively write:
$$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} = \mathsf{g}'(\mathbf{X})$$




### 3.6.19 Backpropagation Through a Matrix Product

We will abusively write, only for matrix multiplication:

$$\frac{\partial \mathbf{T}}{\partial \mathbf{X}} = \mathbf{Y}^t$$
$$\frac{\partial \mathbf{T}}{\partial \mathbf{Y}} = \mathbf{X}^t$$

### 3.6.20 Backpropagation Through a Fully Connected Layer

.5 Setup: .5

$$\begin{aligned} p, q &\in \mathbb{N}^* \\ \mathbf{x} &\in \mathbb{R}^p \\ \mathbf{W} &\in \mathbb{R}^q \times \mathbb{R}^p \\ \mathbf{b}, \mathbf{t}, \mathbf{y} &\in \mathbb{R}^q \\ L &\in \mathbb{R} \end{aligned}$$

0.5 Forward pass:

$$\begin{aligned} \mathbf{t} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{y} &= \mathrm{g}(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ L &= L(\mathbf{y}) \end{aligned}$$

$$\begin{aligned} L : \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \mathbf{x} &\longmapsto L(\mathbf{x}) \end{aligned}$$

**Gradient**

$$\nabla_{\mathbf{x}} L = \frac{\partial L}{\partial \mathbf{x}} = \left(\frac{\partial L}{\partial x_1}, \ldots, \frac{\partial L}{\partial x_n}\right)$$

$$\begin{aligned} V : \mathbb{R}^p &\longrightarrow \mathbb{R}^q \\ \mathbf{y} &\longmapsto V(\mathbf{y}) \end{aligned}$$

**Jacobian**

$$J(V) = \frac{\partial V}{\partial \mathbf{y}} = \begin{pmatrix} \frac{\partial V_1}{\partial y_1} & \cdots & \frac{\partial V_1}{\partial y_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial V_q}{\partial y_1} & \cdots & \frac{\partial V_q}{\partial y_p} \end{pmatrix}$$

0.5 Local gradients:

$$\begin{aligned} \frac{\partial \mathbf{t}}{\partial \mathbf{W}} &= \mathbf{x}^t \\ \frac{\partial \mathbf{t}}{\partial \mathbf{b}} &= Id_{(q)} \\ \frac{\partial \mathbf{y}}{\partial \mathbf{t}} &= \mathbf{g}'(\mathbf{t}) \end{aligned}$$

Backpropagation:

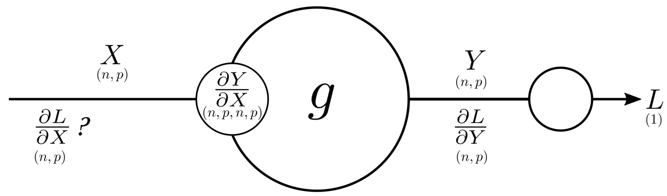
$$\begin{aligned} \frac{\partial L}{\partial \mathbf{t}} &= \frac{\partial L}{\partial \mathbf{y}} . \frac{\partial \mathbf{y}}{\partial \mathbf{t}} \\ &= \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{g}'(\mathbf{t}) \end{aligned}$$

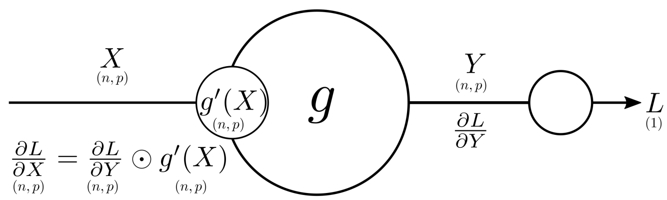
Backpropagation: .5

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{W}} &= \frac{\partial L}{\partial \mathbf{t}} . \frac{\partial \mathbf{t}}{\partial \mathbf{W}} \\ &= \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{g}'(\mathbf{t}) . \mathbf{x}^t \end{aligned}$$

.5

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{b}} &= Id^t . \frac{\partial L}{\partial \mathbf{t}} \\ &= \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{g}'(\mathbf{t}) \end{aligned}$$




## 3.7 Weights Initialization

### 3.7.1 Network Parameters Initialization

**General Idea**

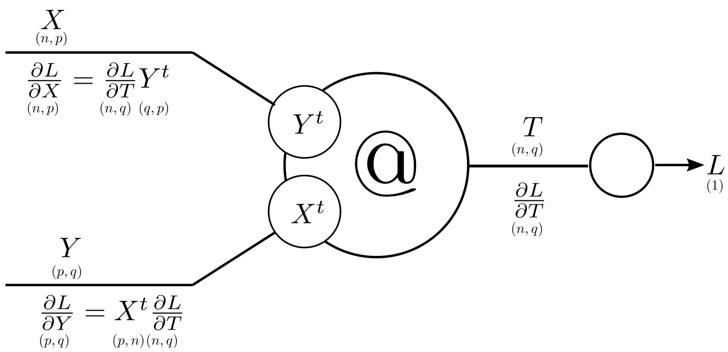
Inputs of activation functions should be in a range such that gradients are high.

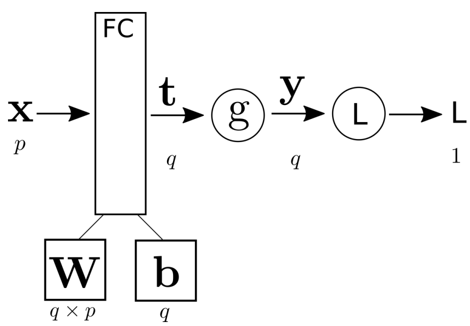
.3 .7

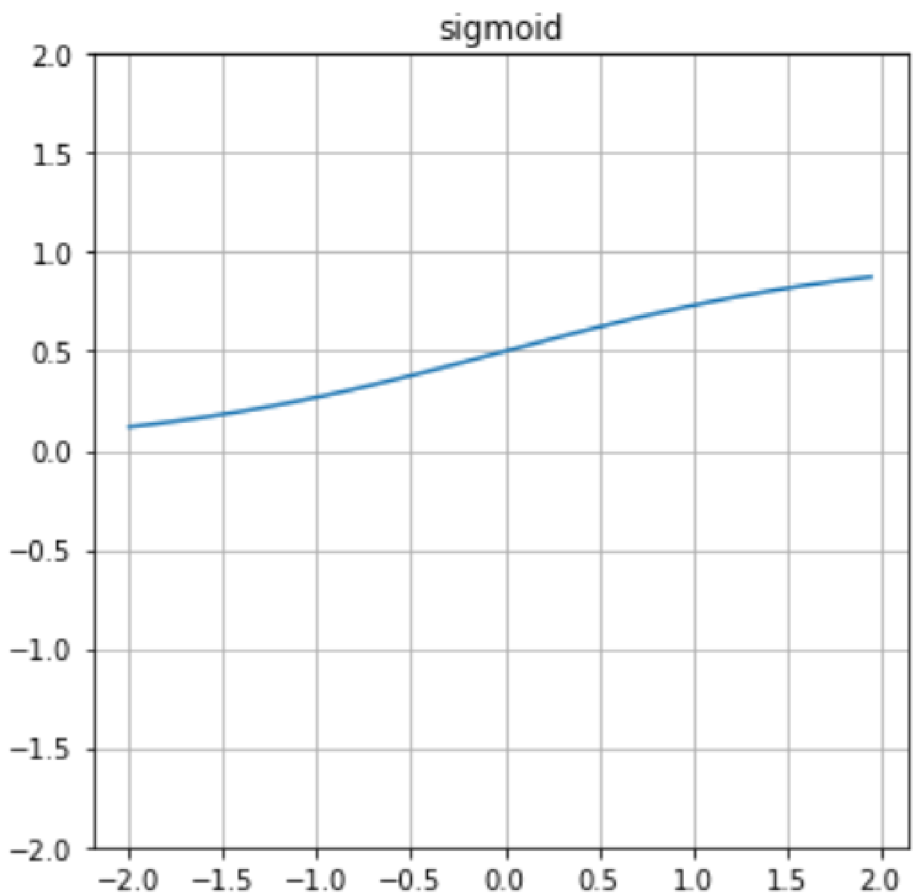
- Bias are set to zero.
- If weights are also initialized to zero, then in each layer the activations will remain equal – symmetry will never be broken.
- Empirical solutions are based on a Gaussian distribution of the weights, with *small* standard deviation.

### 3.7.2 Network Parameters Initialization: Current Practice

- [**?**]: they empirically show that a standard deviation of $1/\sqrt{n}$ gives good results (where $n$ is the number of inputs of a neuron).
- [**?**]: in the case of ReLU activations, they recommend a $2/\sqrt{n}$ standard deviation.


$\underset{(n,p)}{X}$
$\underset{(n,p)}{\frac{\partial L}{\partial X}} = \underset{(n,q)}{\frac{\partial L}{\partial T}} \underset{(q,p)}{Y^t}$
$Y^t$
@
$X^t$
$\underset{(n,q)}{T}$
$\underset{(n,q)}{\frac{\partial L}{\partial T}}$
$\underset{(1)}{L}$
$\underset{(p,q)}{Y}$
$\underset{(p,q)}{\frac{\partial L}{\partial Y}} = \underset{(p,n)}{X^t} \underset{(n,q)}{\frac{\partial L}{\partial T}}$


$\mathbf{x}$
$p$
FC
$\mathbf{t}$
$q$
g
$\mathbf{y}$
$q$
L
L
1
$\mathbf{W}$
$q \times p$
$\mathbf{b}$
$q$


sigmoid
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0

# Chapter 4

# Conclusion

We have seen:

- What is an artificial neuron and an artificial neural network (NN).
- The (potential) power of a NN.
- The backpropagation algorithm.
- NN learning basics.

Next step:

- Application to images.

# Bibliography


[Cybenko, 1989] Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:183–192.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*. arXiv: 1502.01852.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

[LeCun, 1985] LeCun, Y. (1985). Une procedure d'apprentissage pour reseau a seuil asymmetrique (A learning scheme for asymmetric threshold networks). In *proceedings of Cognitiva 85*.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Werbos, 1982] Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In Drenick, R. F. and Kozin, F., editors, *System Modeling and Optimization*, Lecture Notes in Control and Information Sciences, pages 762–770, Berlin, Heidelberg. Springer.