

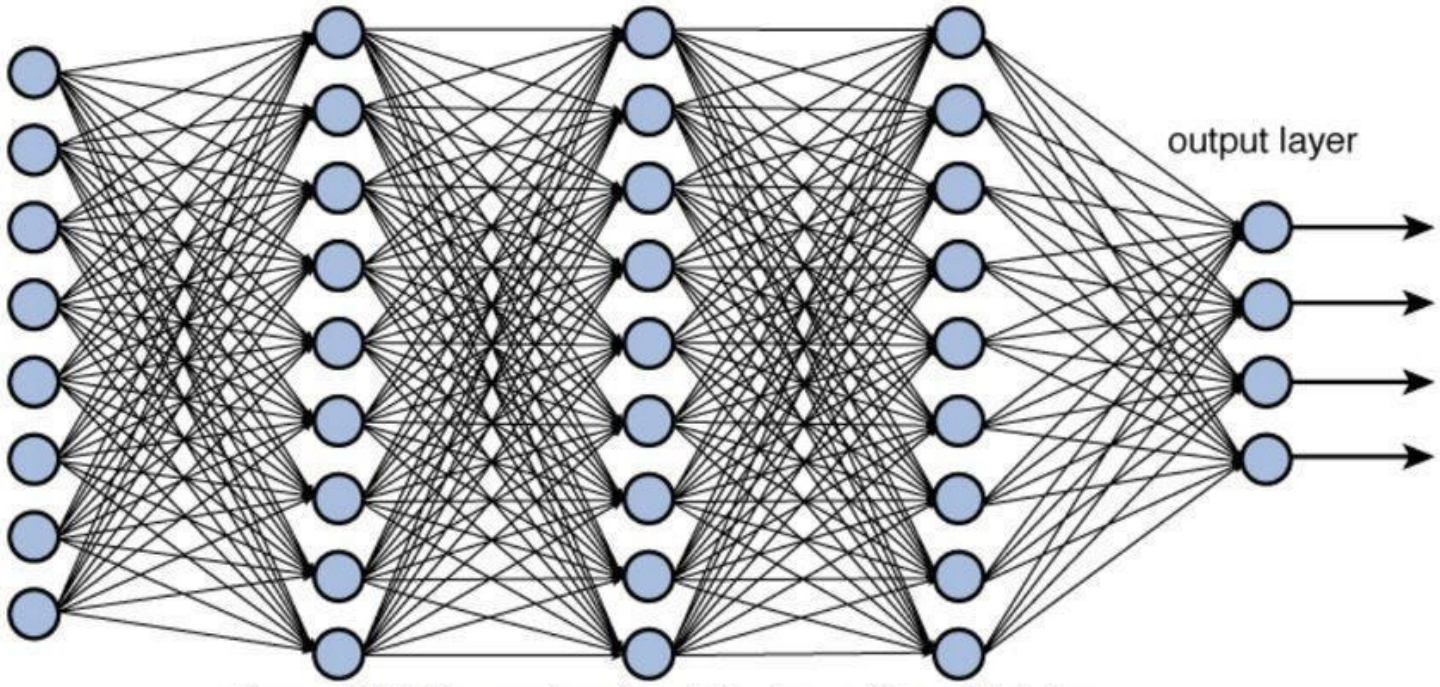
Hands-on Introduction to Deep Learning

Artificial Neural Networks

cnrs IDRIS

Goals of this section

- understanding the origin and development of neural networks
- mastering the fundamentals of neural networks



Algorithms

Artificial Intelligence

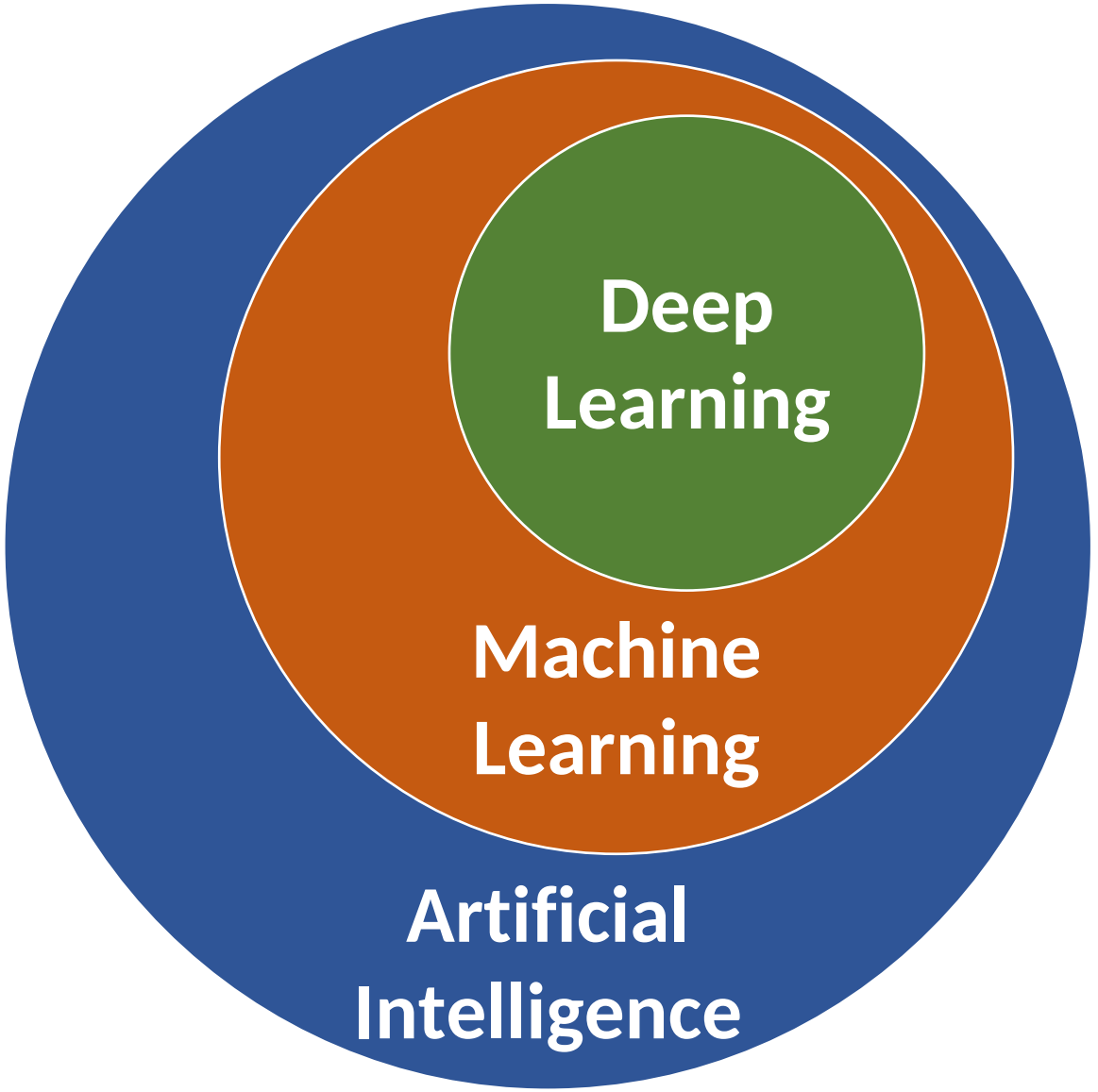
Systems capable of reproducing human actions or decisions

Machine Learning

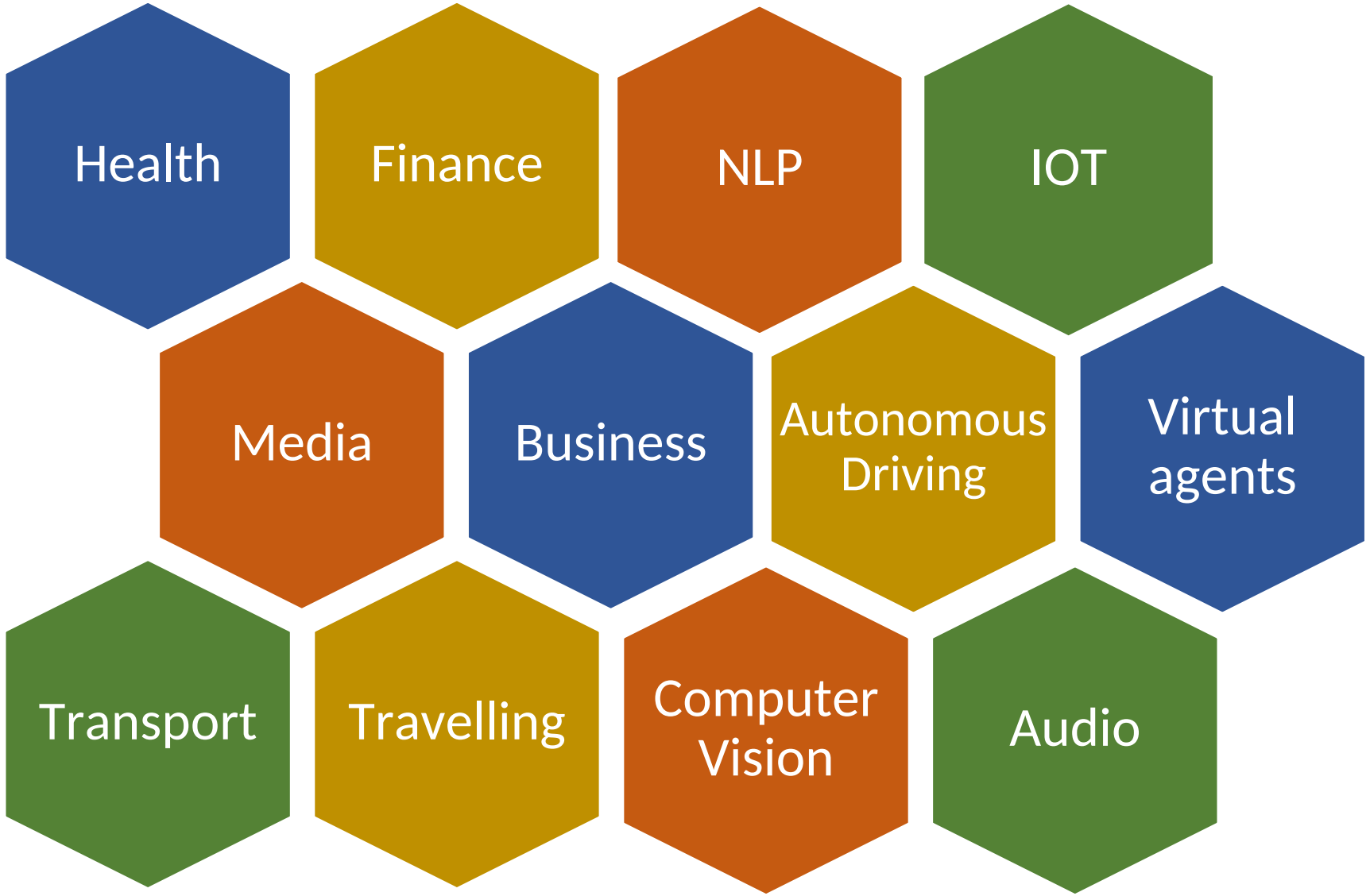
Algorithms mainly based on statistical methods, often with an iterative aspect from which comes the term « Learning »

Deep Learning

A group of models based on logical units called neurons and distributed in layers



Application fields



Application fields

IOT



- Personal assistant
- Smart connected object

Healthcare



- Medical Imaging
- Drug research

Finance



- Fraud detection
- Prediction of financial flows

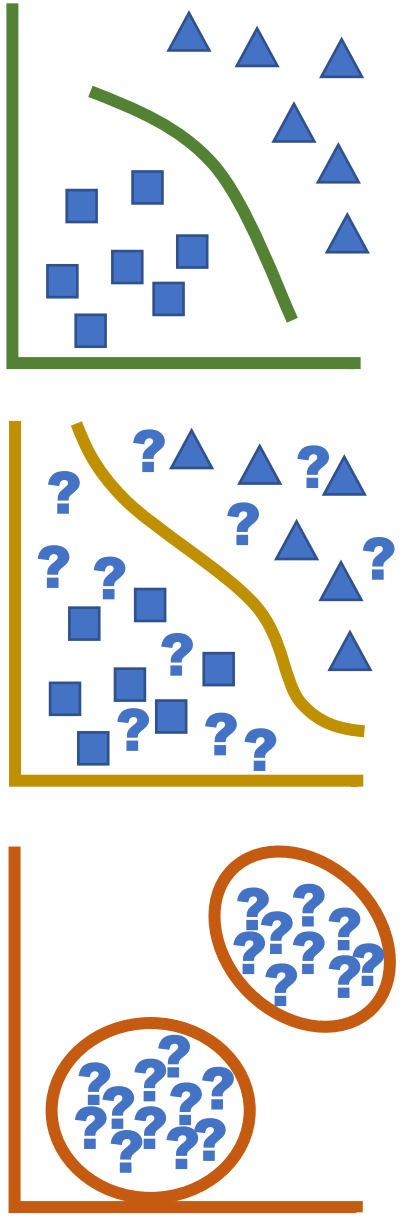
Autonomous driving



- Route optimization
- Autonomous driving

Type of problems

Data	Learning
Labelled	Supervised
Partially labelled	Semi-supervised
Unlabelled	Unsupervised



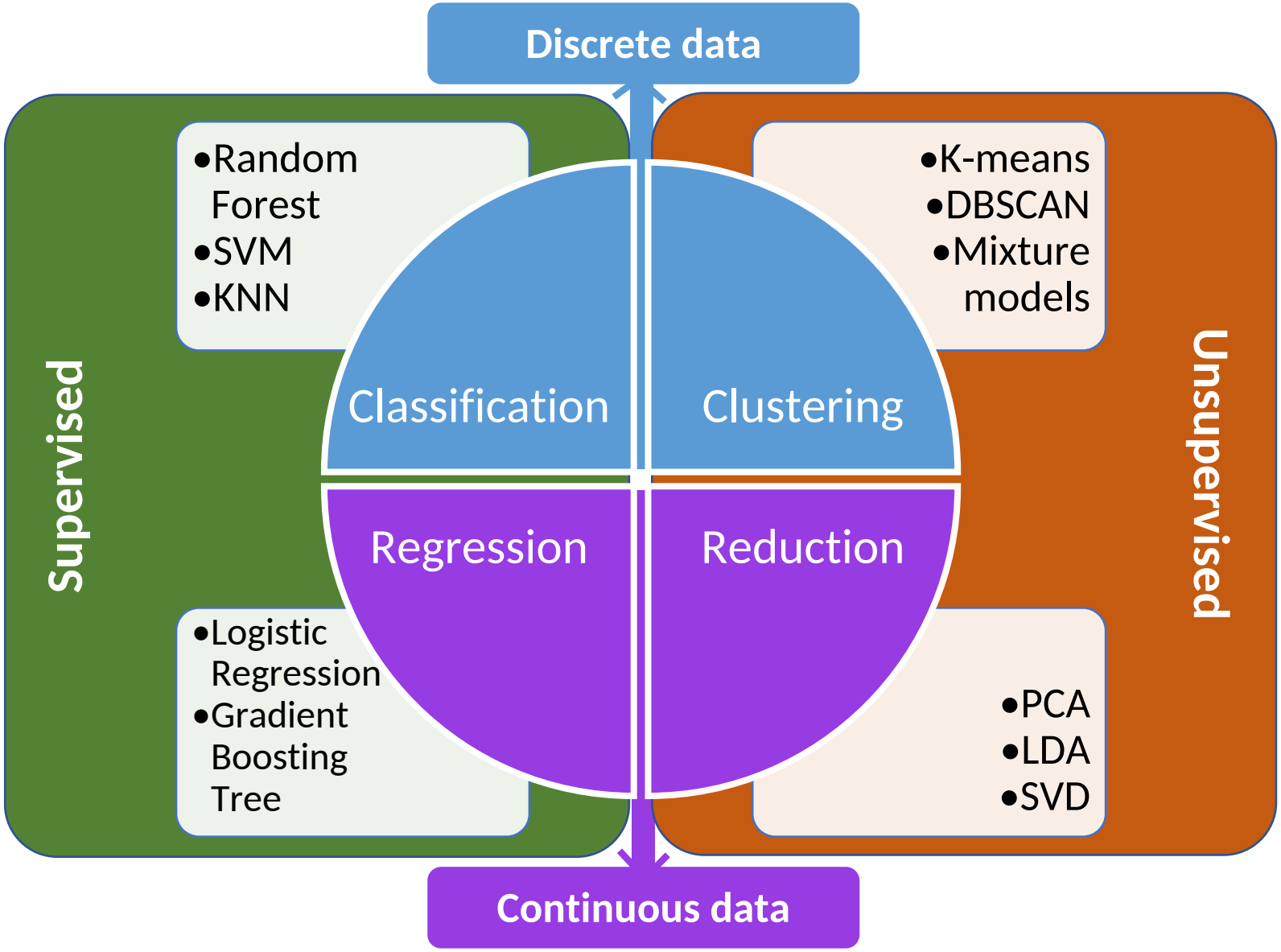
Algorithms

Discrete Data

data that can only take specific, separate values, usually whole numbers or categories you can count

Continuous Data

data that can take any value within a range, including fractions and decimals, usually something you measure



Bonus : to delve deeper

Random Forest: a collection of many decision trees, each trained slightly differently; the final prediction is decided by majority vote among all trees.

SVM (Support Vector Machine): finds the best boundary (line or curve) that separates classes by maximizing the margin, i.e. the distance between the boundary and the closest data points.

KNN (K-Nearest Neighbors): classifies a new point by looking at its K closest neighbors in the training data and assigning the most common class among them.

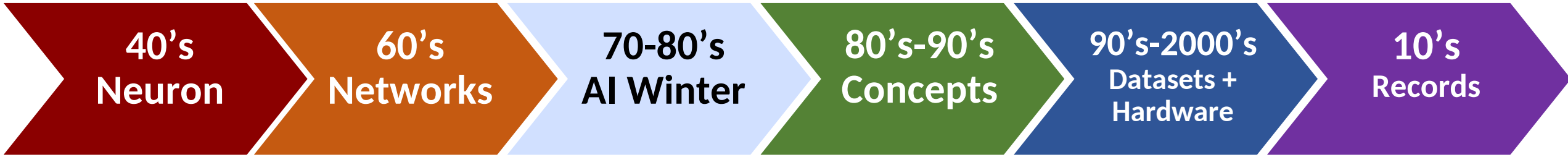
K-means: groups data into K clusters by repeatedly assigning points to the nearest cluster center, then updating each center to be the average of its assigned points.
DBSCAN: groups points that are densely packed together, and marks points in low-density regions as outliers/noise . Unlike K-means, it doesn't require choosing the number of clusters in advance.
Mixture models: assume the data comes from a mix of several underlying distributions (often Gaussian); the model estimates these distributions and assigns each point a probability of belonging to each one.

Logistic Regression: despite its name, mainly used for classification; it estimates the probability that an input belongs to a given class using a curve (the sigmoid function).

Gradient Boosting Tree: builds decision trees one after another, where each new tree focuses on correcting the errors made by the previous ones.

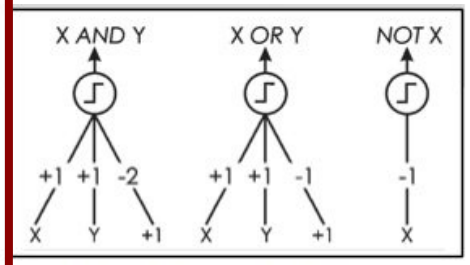
PCA (Principal Component Analysis): finds new axes (combinations of the original variables) that capture the most variation in the data, allowing it to be represented with fewer dimensions.
LDA (Linear Discriminant Analysis): similar to PCA, but finds axes that best separate known classes, rather than just capturing variation.
SVD (Singular Value Decomposition): a mathematical technique that breaks a data matrix into simpler components, often used as the underlying method behind PCA and other reduction techniques.

History of Deep Learning



History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**

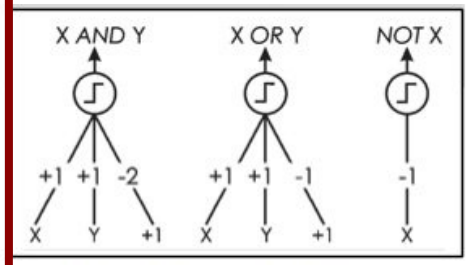


- Adjustable Weights
- Weights are not Learned

40's Neuron → **60's Networks** → **70-80's AI Winter** → **80's-90's Concepts** → **90's-2000's Datasets + Hardware** → **10's Records**

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**



- Adjustable Weights
- Weights are not Learned

40's Neuron

60's Networks

70-80's AI Winter

80's-90's Concepts

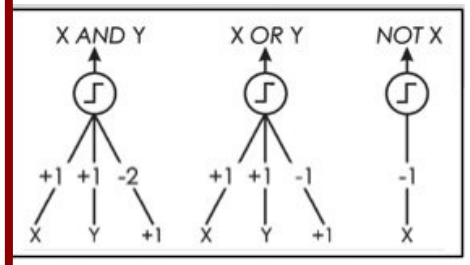
90's-2000's Datasets + Hardware

10's Records

1950 : **Turing's test**

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**

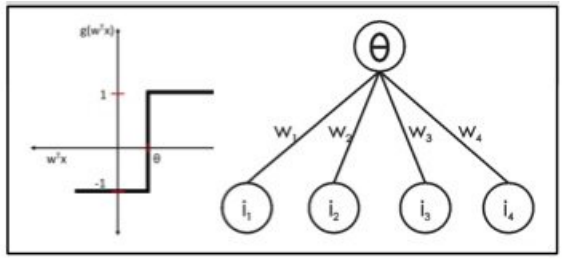


- Adjustable Weights
- Weights are not Learned

40's Neuron | **60's Networks** | **70-80's AI Winter** | **80's-90's Concepts** | **90's-2000's Datasets + Hardware** | **10's Records**

1950 : **Turing's test**

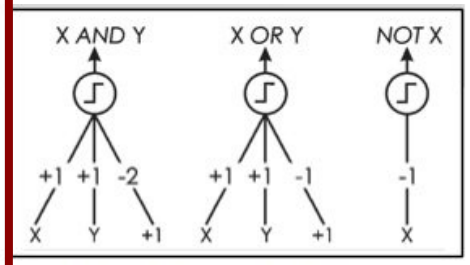
1958 : Frank Rosenblatt's **perceptron**



- Learnable Weights and Threshold

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**



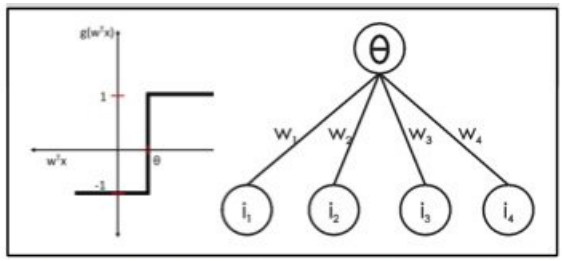
- Adjustable Weights
- Weights are not Learned

40's Neuron | **60's Networks** | **70-80's AI Winter** | **80's-90's Concepts** | **90's-2000's Datasets + Hardware** | **10's Records**

1950 : **Turing's test**

1960 : **ADALINE** Widrow & Hoff

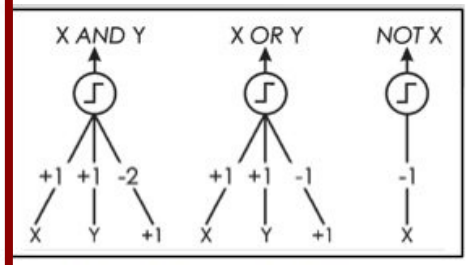
1958 : Frank Rosenblatt's **perceptron**



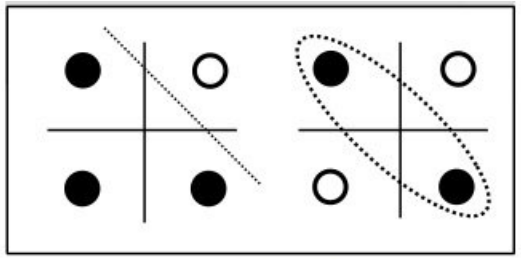
- Learnable Weights and Threshold

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**



- Adjustable Weights
- Weights are not Learned



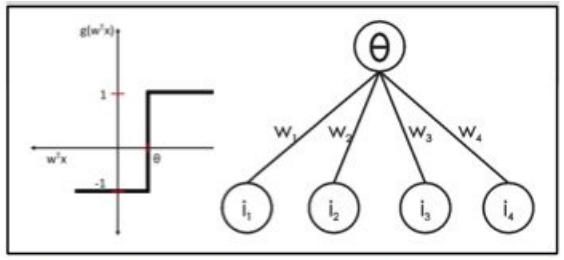
1969 : Minsky and Papert
XOR problem

40's Neuron | **60's Networks** | **70-80's AI Winter** | **80's-90's Concepts** | **90's-2000's Datasets + Hardware** | **10's Records**

1950 : **Turing's test**

1960 : **ADALINE**
Widrow & Hoff

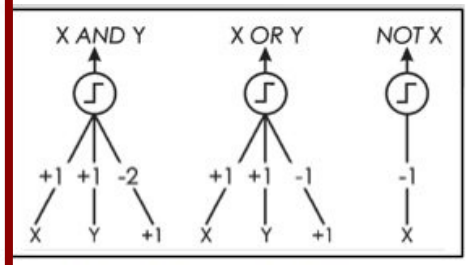
1958 : Frank Rosenblatt's **perceptron**



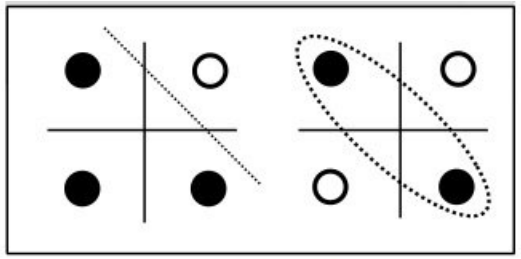
- Learnable Weights and Threshold

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**



- Adjustable Weights
- Weights are not Learned



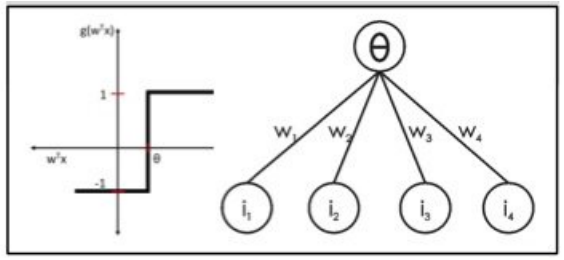
1969 : Minsky and Papert
XOR problem

40's Neuron | **60's Networks** | **70-80's AI Winter** | **80's-90's Concepts** | **90's-2000's Datasets + Hardware** | **10's Records**

1950 : **Turing's test**

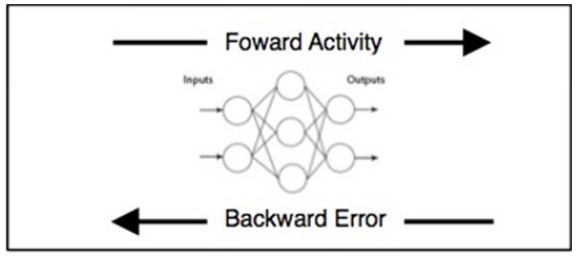
1960 : **ADALINE**
Widrow & Hoff

1958 : Frank Rosenblatt's **perceptron**



- Learnable Weights and Threshold

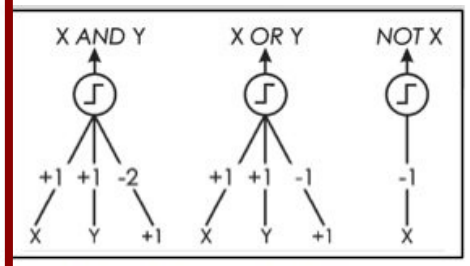
1986 : **Gradient retropropagation**
(Rumelhart, Hinton, Williams)



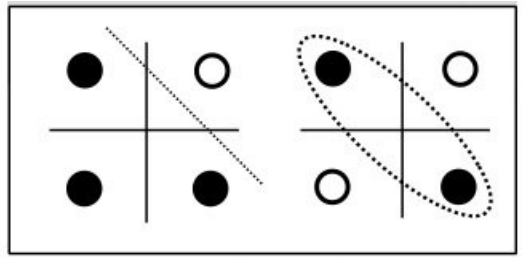
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**



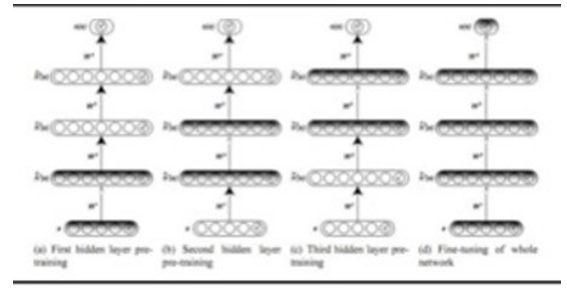
- Adjustable Weights
- Weights are not Learned



1969 : Minsky and Papert
XOR problem

1987-1993 : Second IA winter

hierarchical feature learning



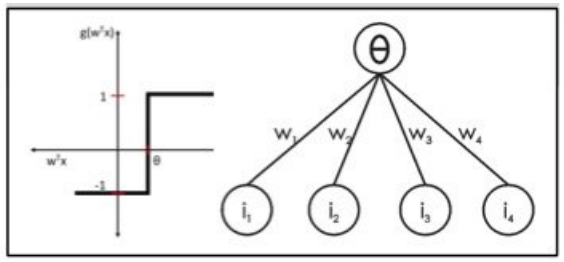
- Hierarchical feature Learning

40's Neuron | **60's Networks** | **70-80's AI Winter** | **80's-90's Concepts** | **90's-2000's Datasets + Hardware** | **10's Records**

1950 : Turing's test

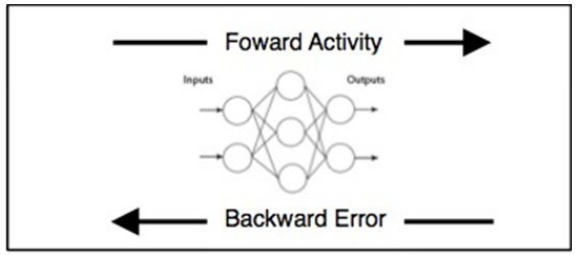
1960 : ADALINE Widrow & Hoff

1958 : Frank Rosenblatt's **perceptron**



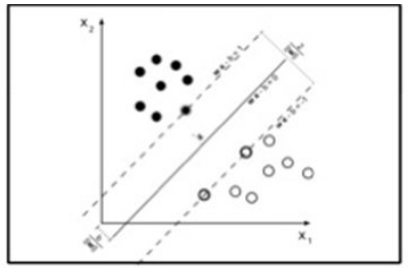
- Learnable Weights and Threshold

1986 : Gradient retropropagation (Rumelhart, Hinton, Williams)



- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

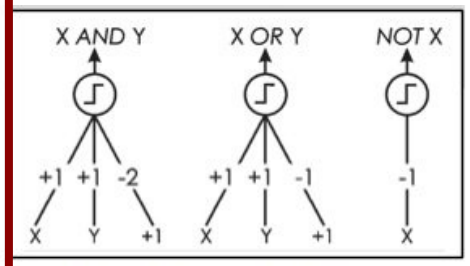
Support Vector Machine



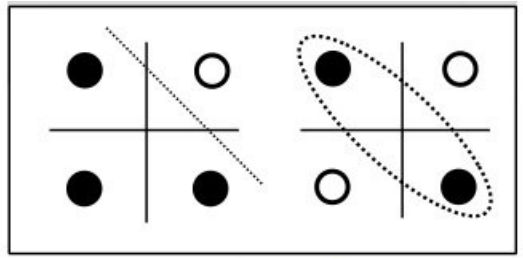
- Limitations of learning prior knowledge
- Kernel function: Human Intervention

History of Deep Learning

1943 : W.Pitts' and S. McCuloch's **artificial neuron**



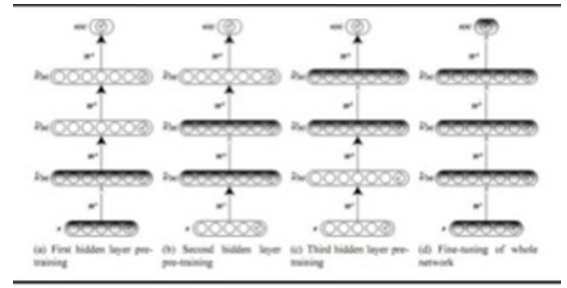
- Adjustable Weights
- Weights are not Learned



1969 : Minsky and Papert
XOR problem

1987-1993 : Second IA winter

hierarchical feature learning



- Hierarchical feature Learning

40's Neuron

60's Networks

70-80's AI Winter

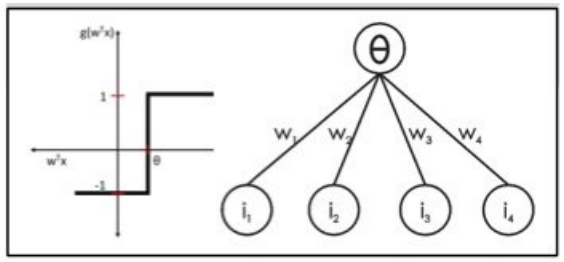
80's-90's Concepts

90's-2000's Datasets + Hardware

10's Records

1950 : **Turing's test**

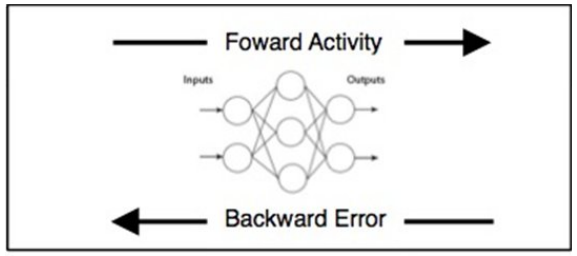
1958 : Frank Rosenblatt's **perceptron**



- Learnable Weights and Threshold

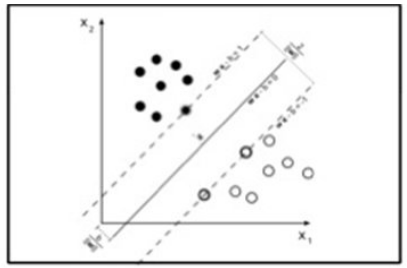
1960 : **ADALINE**
Widrow & Hoff

1986 : **Gradient retropropagation**
(Rumelhart, Hinton, Williams)



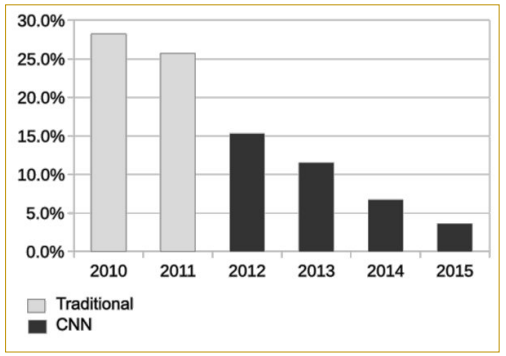
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

Support Vector Machine

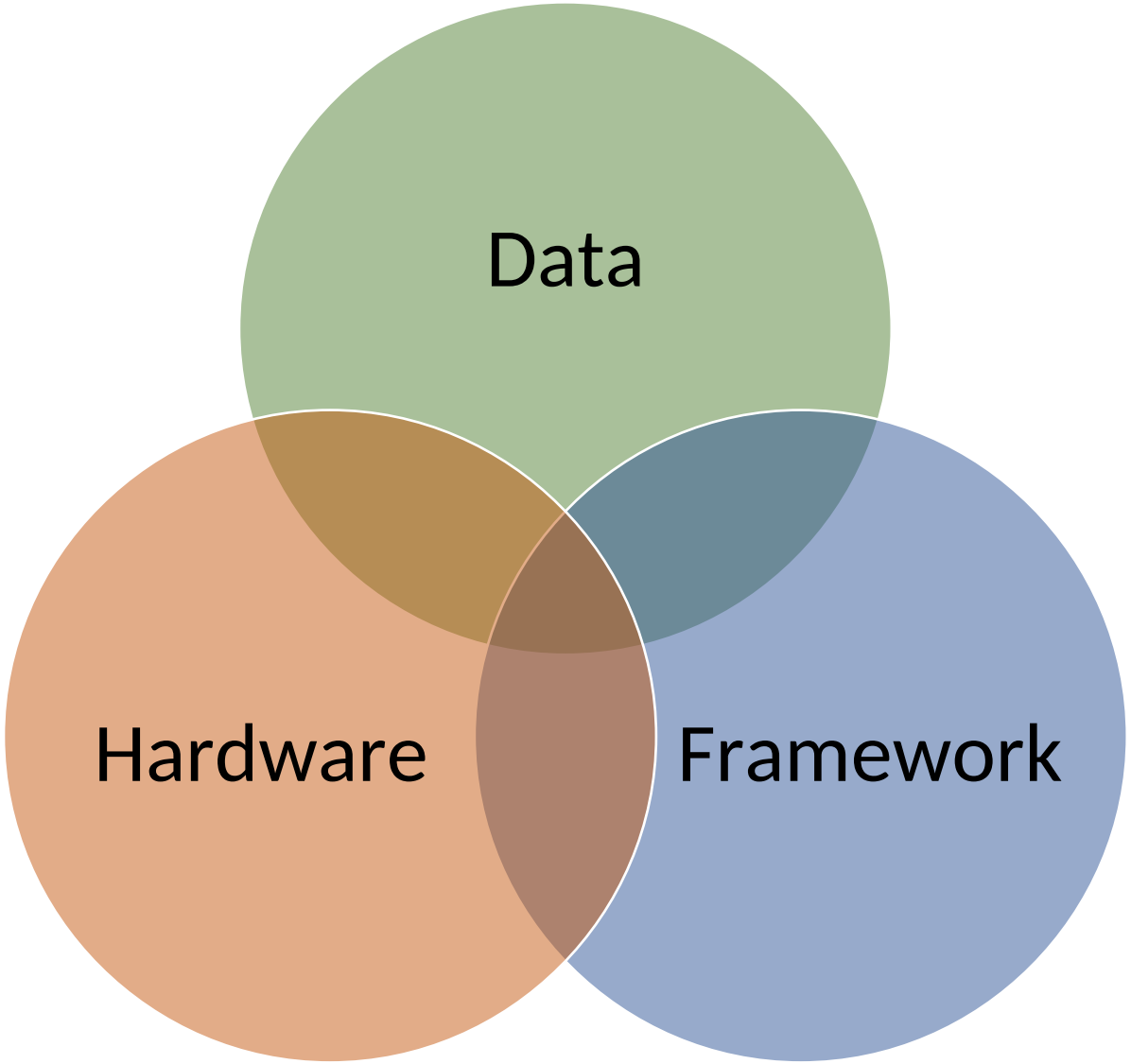


- Limitations of learning prior knowledge
- Kernel function: Human Intervention

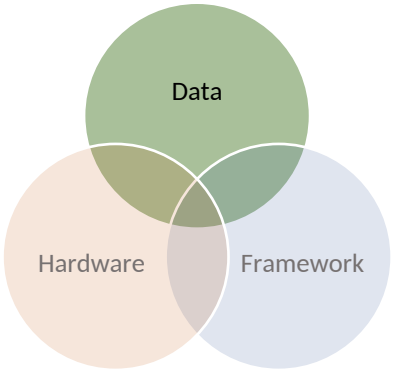
2012 : **CNN**

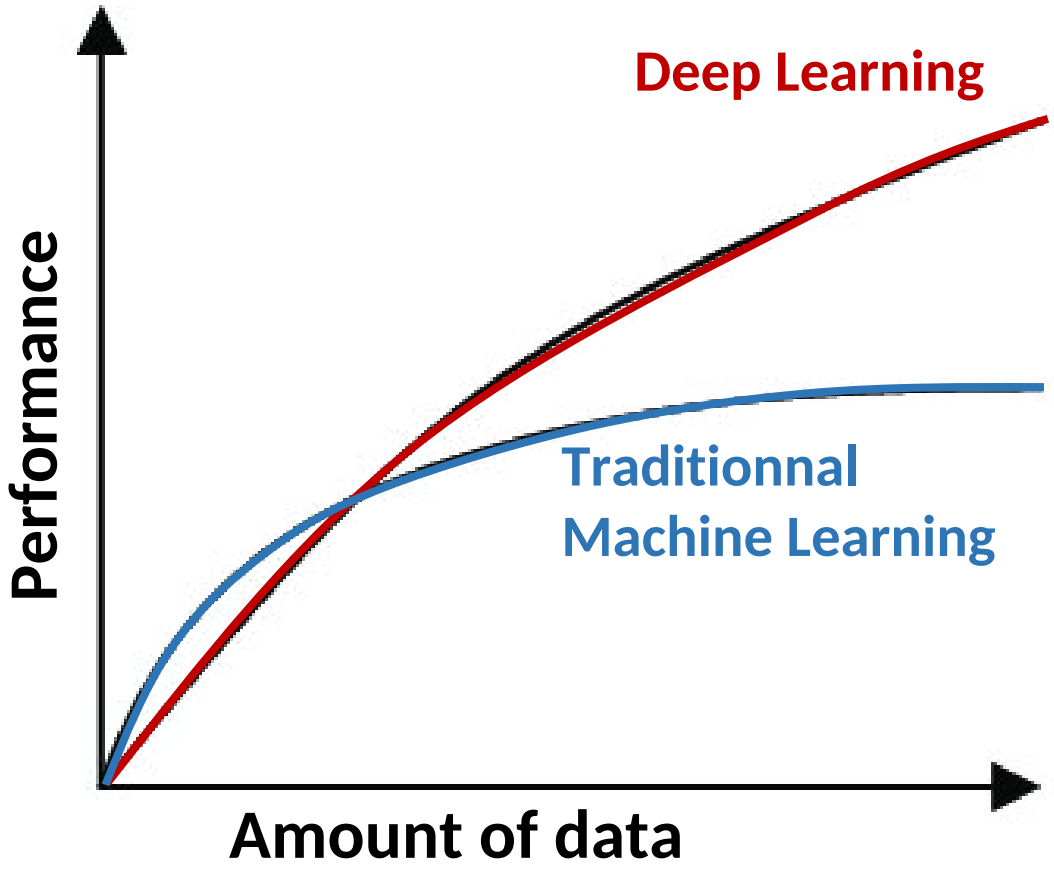


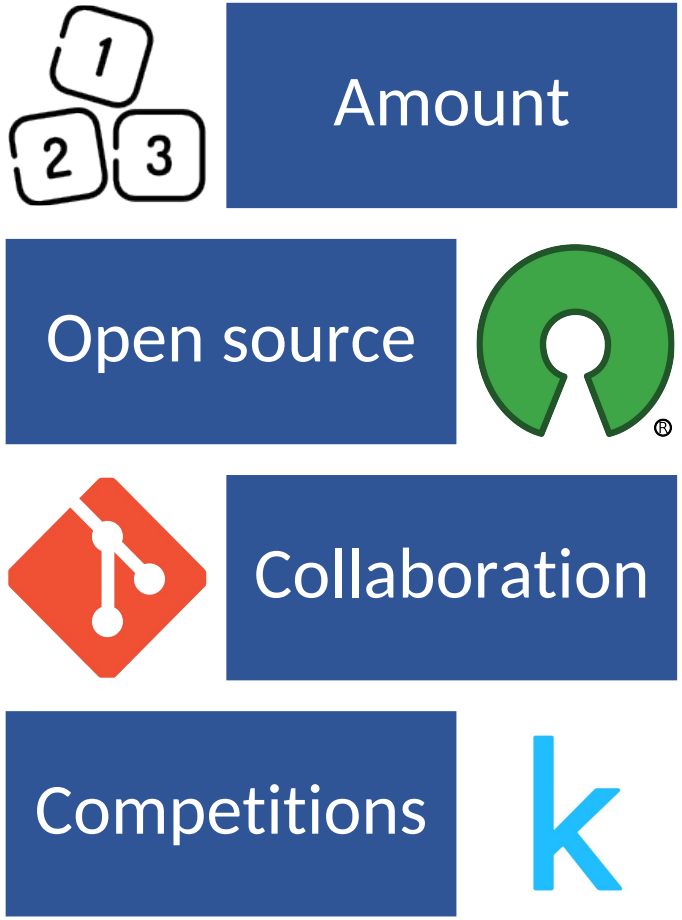
3 decisive factors



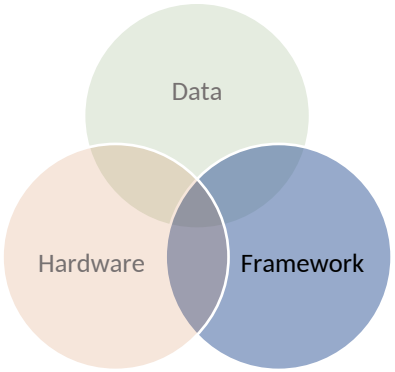
Data

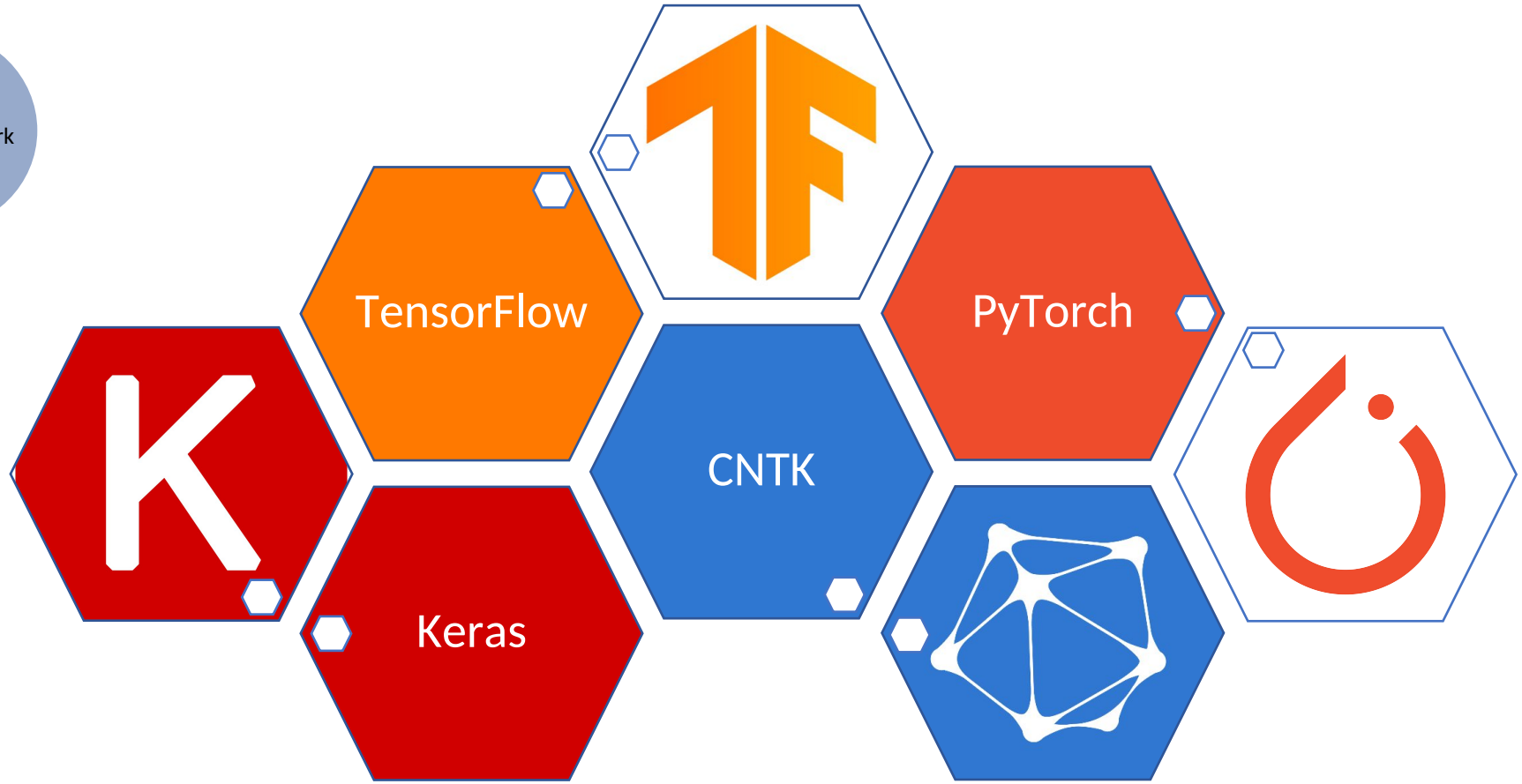

Data
Hardware
Framework


Deep Learning
Traditionnal
Machine Learning
Performance
Amount of data

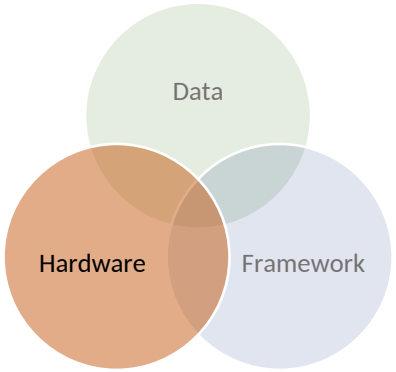

Amount
Open source
Collaboration
Competitions
k

Frameworks and libraries





Hardware



CPU

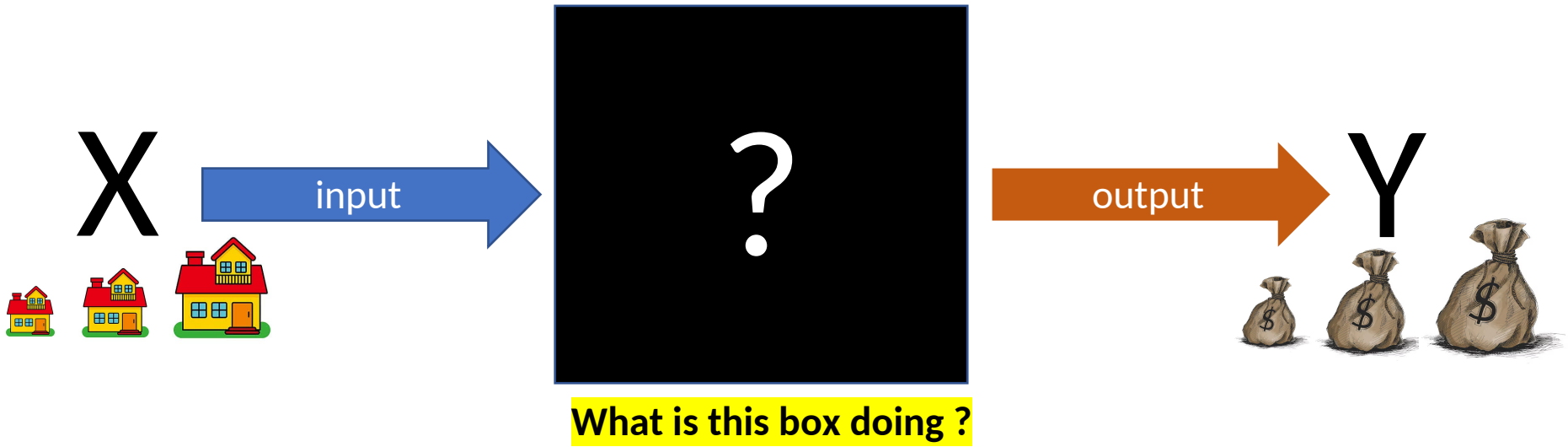
GPU

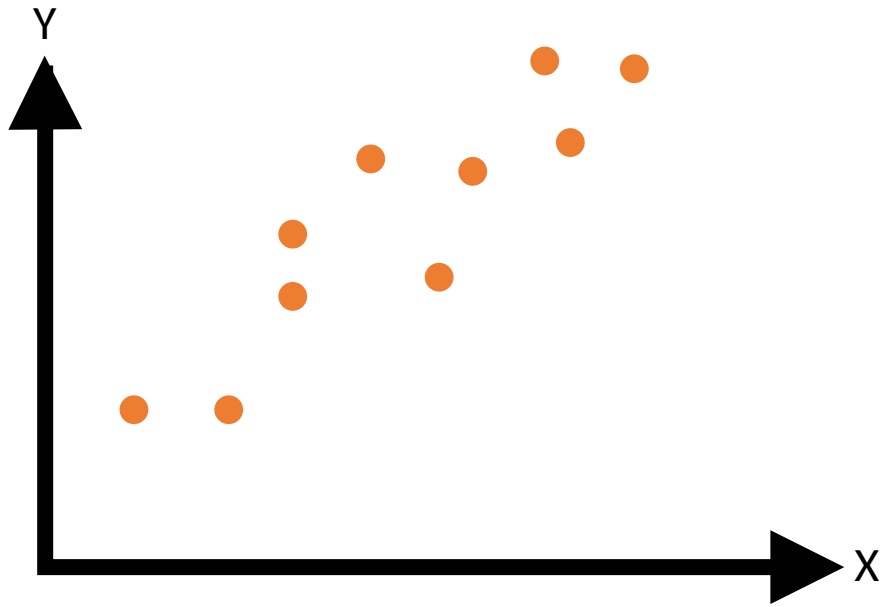
TPU

FPGA



Predictive models : linear regression





↳ We need to find a function that relates X to Y as much as possible. The simplest case is a linear relationship : a straight line

Linear regression

The « real » Y :

$$Y = X \cdot \Theta + N$$

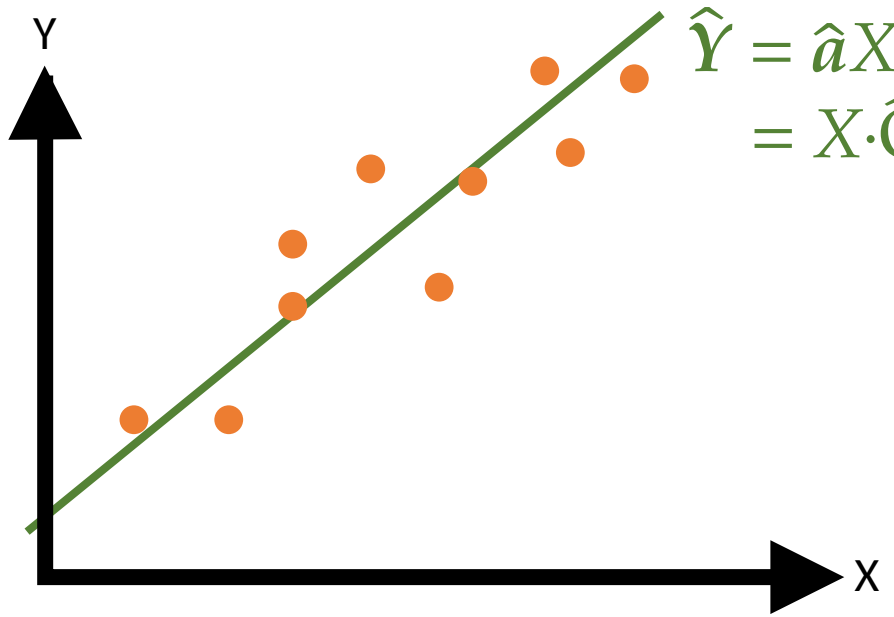
with :

$$\Theta = \begin{pmatrix} a \\ b \end{pmatrix}$$

N , noise

The estimated Y :

$$\hat{Y} = X \cdot \hat{\Theta}$$



We need to find a function that relates X to Y as much as possible

↳ It all comes down to finding $\hat{\Theta}$

Linear regression

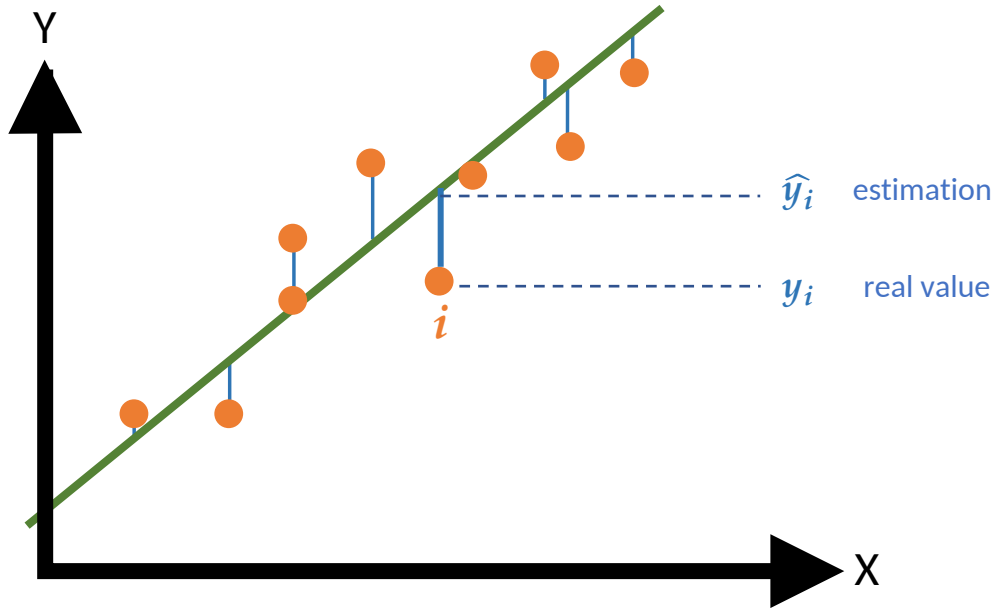
The « real » Y :

$$Y = X \cdot \Theta + N$$

with :

$$\Theta = \begin{pmatrix} a \\ b \end{pmatrix}$$

N , noise



The estimated Y :

$$\hat{Y} = X \cdot \hat{\Theta}$$

Cost function

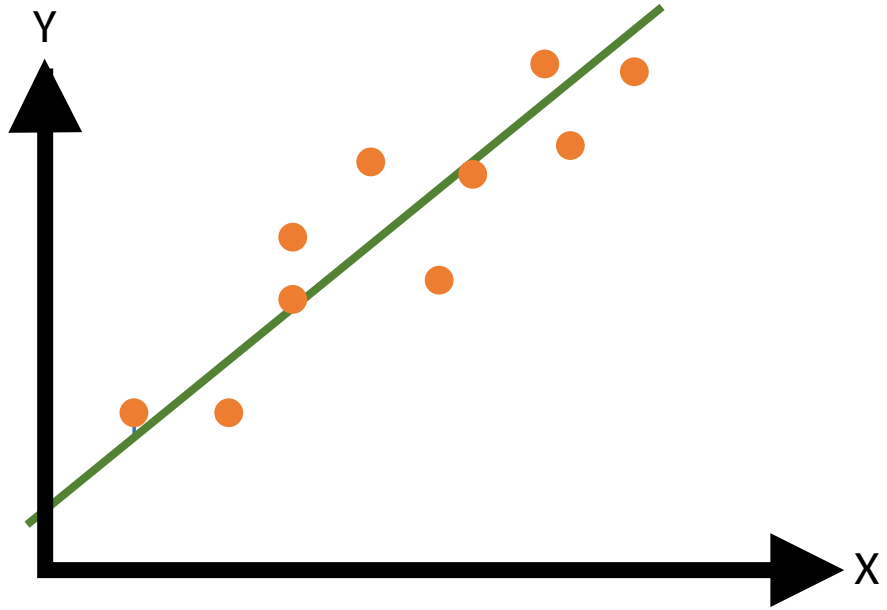
$$\min_{\theta} J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

- The smaller $J(\theta)$ is, the closer to the truth is the estimation
- We are looking for the parameters a and b such as the mean squared error is minimized

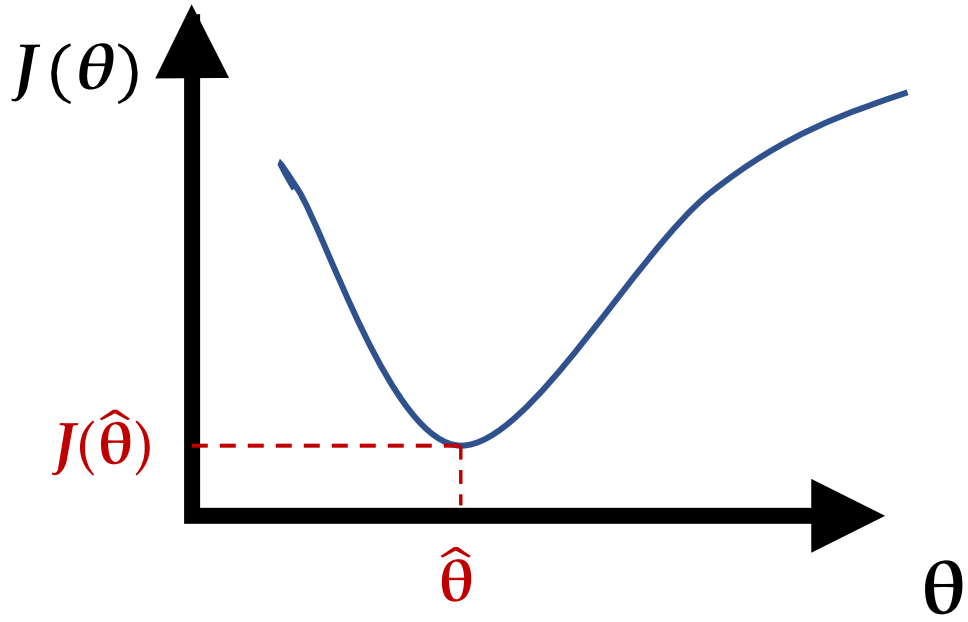
Linear regression

Cost function

$$\min_{\theta} J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



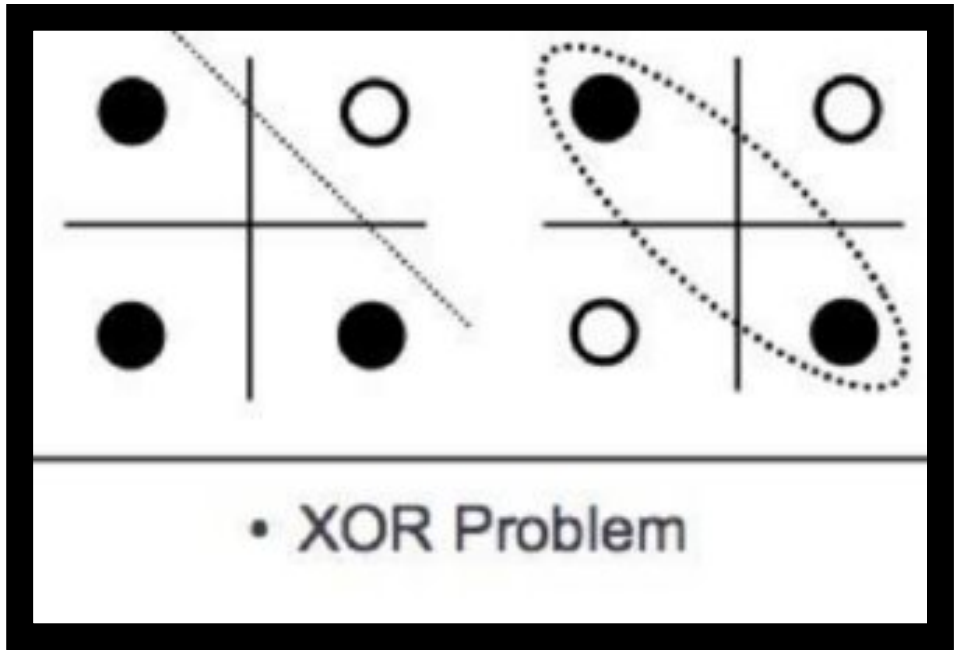
This is a convex problem



But for a non- linear situation, it is not that simple...

What about a non-linear problem?

What if we tried to combine two linear classifiers to resolve a non-linear problem ?



$$f = W_1 x$$

with W : weights matrix

$$\begin{aligned} \hat{f} &= W_2 f \\ &= W_2 (W_1 x) \\ &= \underbrace{(W_2 \cdot W_1)}_{\hat{W}} x \end{aligned}$$

$$\hat{f} = \hat{W} x$$

the combinaison of linear classifiers is a new linear classifier

This doesn't work . We need to break the linearity with an **activation function**

Logistic regression

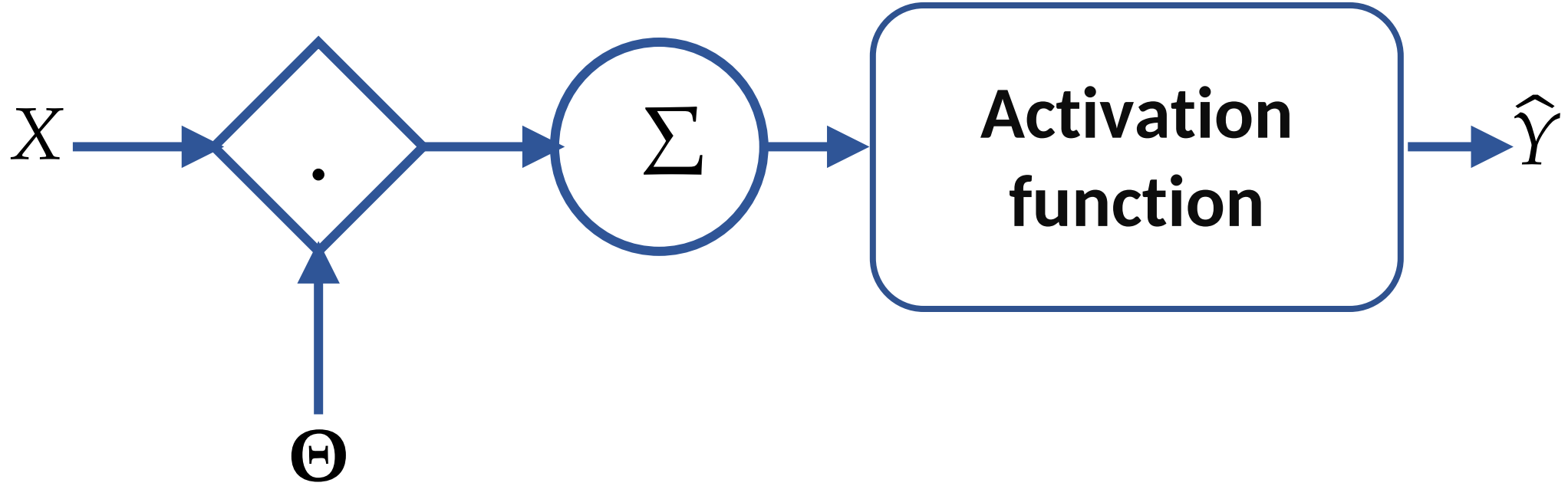
This is
a neuron

A neuron

A small computing unit that takes several inputs, combines them using weighted parameters, and passes the result through an activation function to produce an output.

Activation function

A non-linear function applied to the weighted sum of a neuron's inputs, allowing the network to model non-linear relationships



Logistic regression

Loss function

Measures the error **for a single observation i** : the gap between the predicted value $\hat{y}_i$ and the true value y_i, for that specific data point.

Cost function

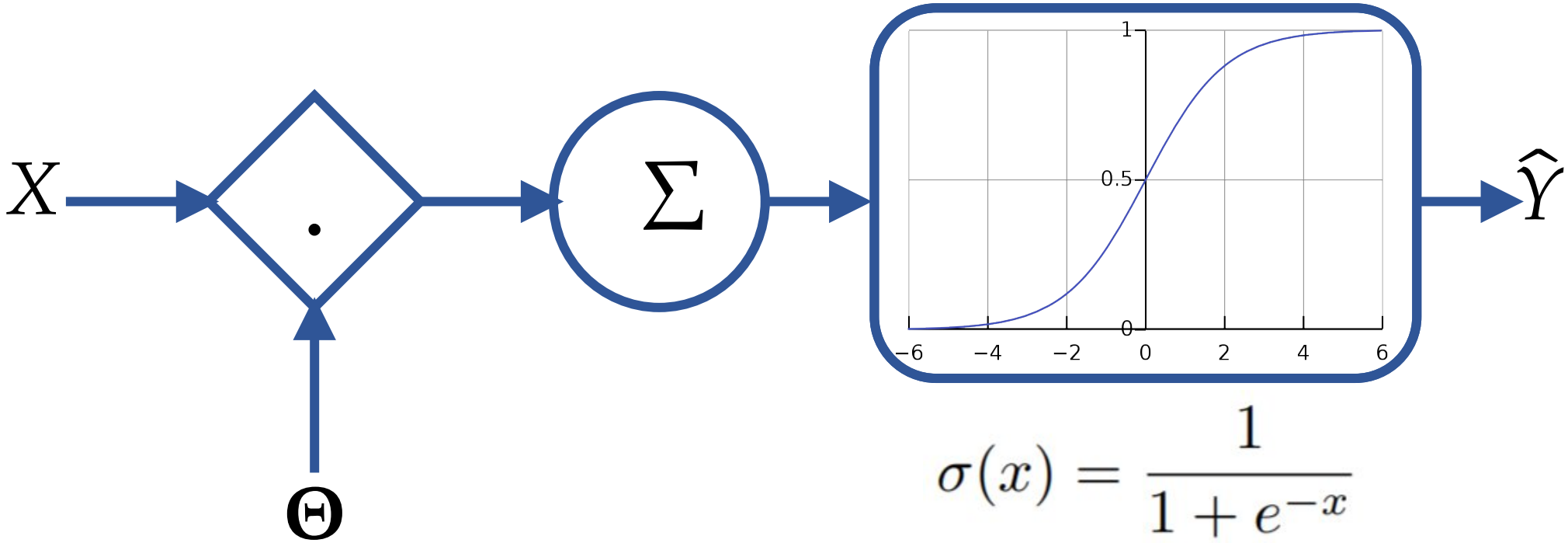
the average of the loss **over the entire dataset** (but computed from the loss rather than directly)

Binary Cross-entropy loss

$$L(\hat{y}_i, y_i) = -(y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i))$$

mostly used in classification

$y_i = 0$ *and* $\hat{y}_i$ *close to* 0 ➡ $L(\hat{y}_i, y_i)$ close to 0
$y_i = 1$ *and* $\hat{y}_i$ *close to* 1 ➡ $L(\hat{y}_i, y_i)$ close to 0
$y_i = 0$ *and* $\hat{y}_i$ *close to* 1 ➡ $L(\hat{y}_i, y_i)$ tends to ∞
$y_i = 1$ *and* $\hat{y}_i$ *close to* 0 ➡ $L(\hat{y}_i, y_i)$ tends to ∞



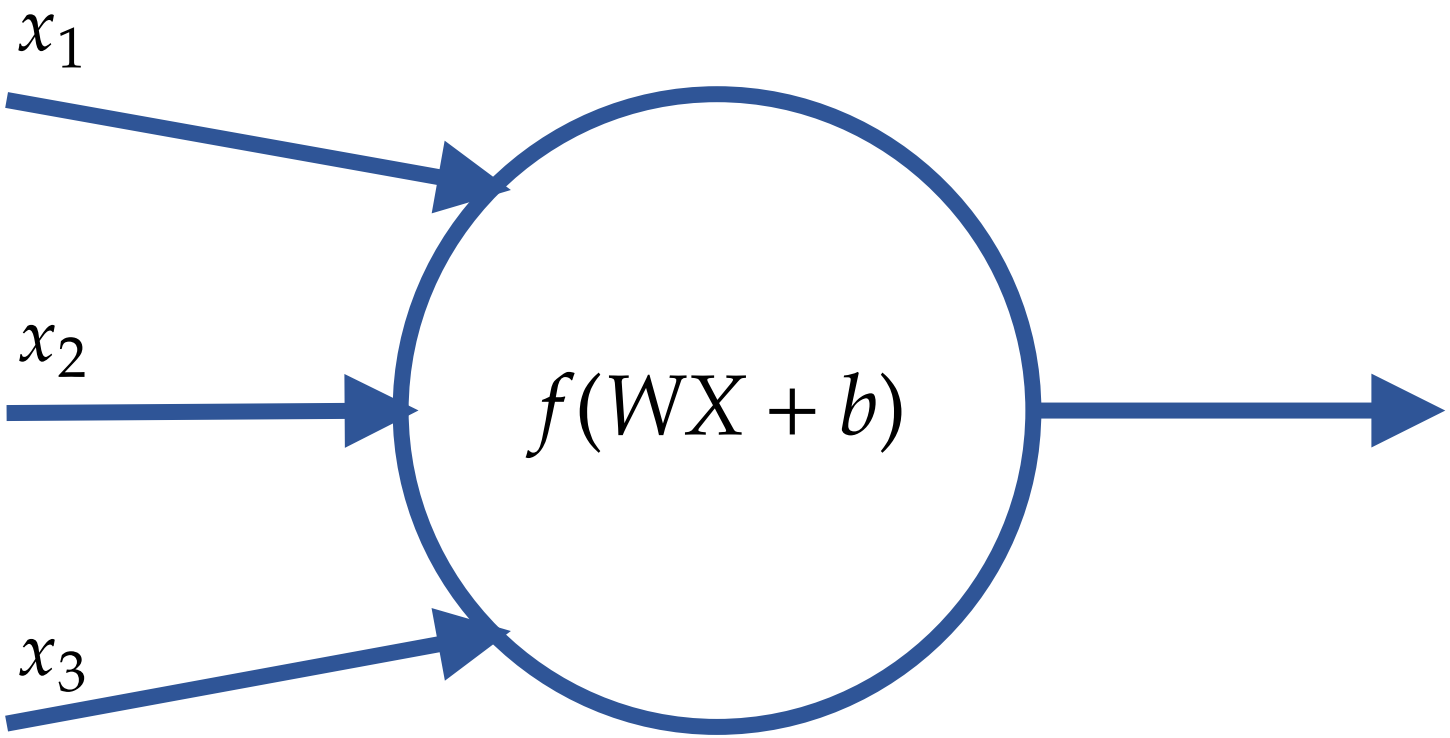
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Loss function

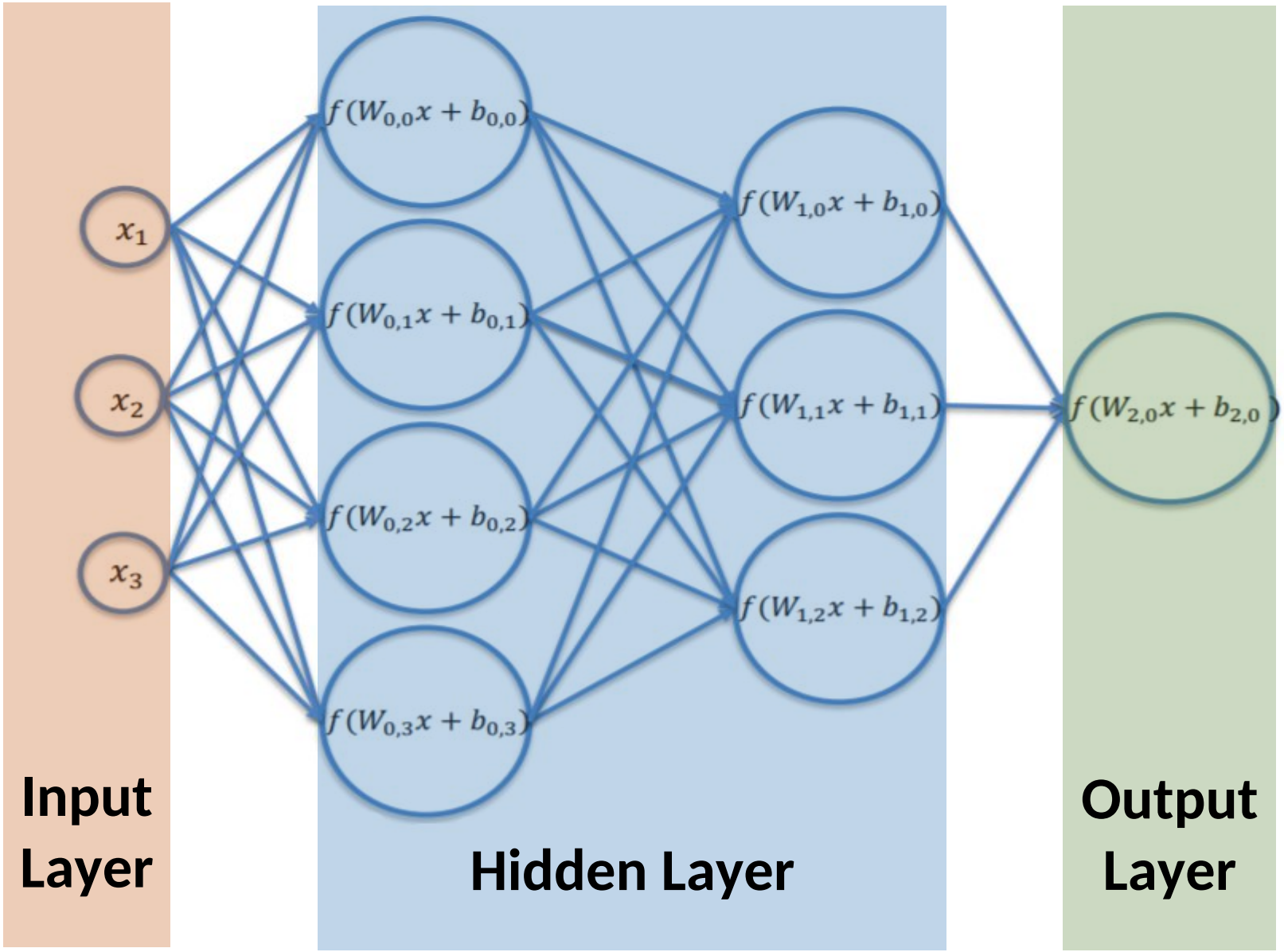
- **Regression loss**
  - Average absolute deviation : $L(y,\hat{y},\theta) = \frac{1}{n}\sum_i^n |y_i - \widehat{y_i}|$
  - Least squares method : $L(y,\hat{y},\theta) = \frac{1}{n}\sum_i^n (y_i - \widehat{y_i})^2$

- **Classification loss**
  - Cross-Entropy : $E(y,\hat{y},\theta) = -\frac{1}{n}\sum_i^n \sum_j^m y_{ij} \log \widehat{y_{ij}}$

1 neuron

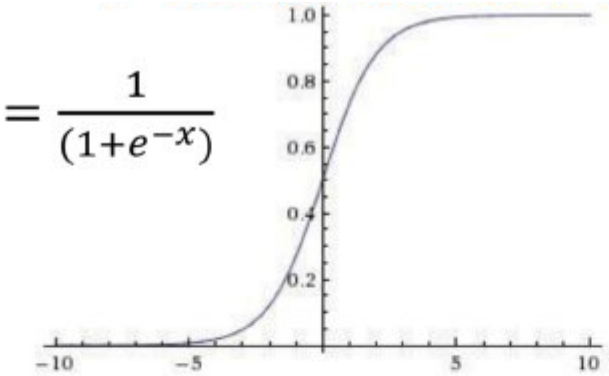


Neural networks

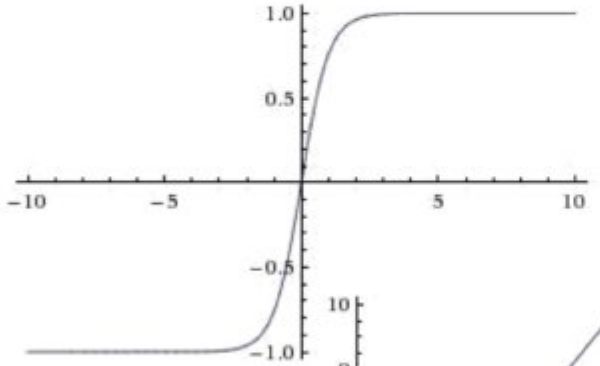


Activation functions

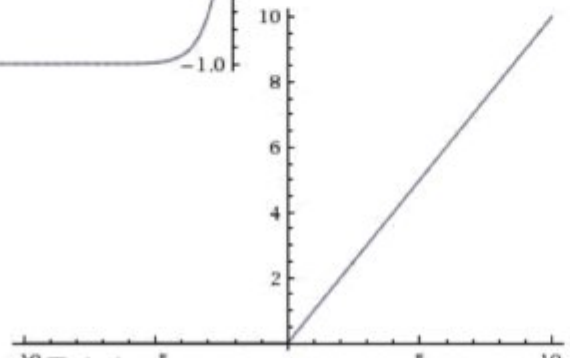
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



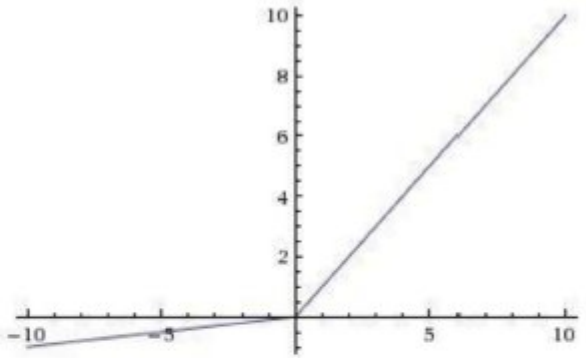
tanh: $\tanh(x)$



ReLU: $\max(0, x)$



Leaky ReLU: $\max(0.1x, x)$



Parametric ReLU: $\max(\alpha x, x)$

Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$

How to choose the right activation function ?



Here are the criteria to take in account :

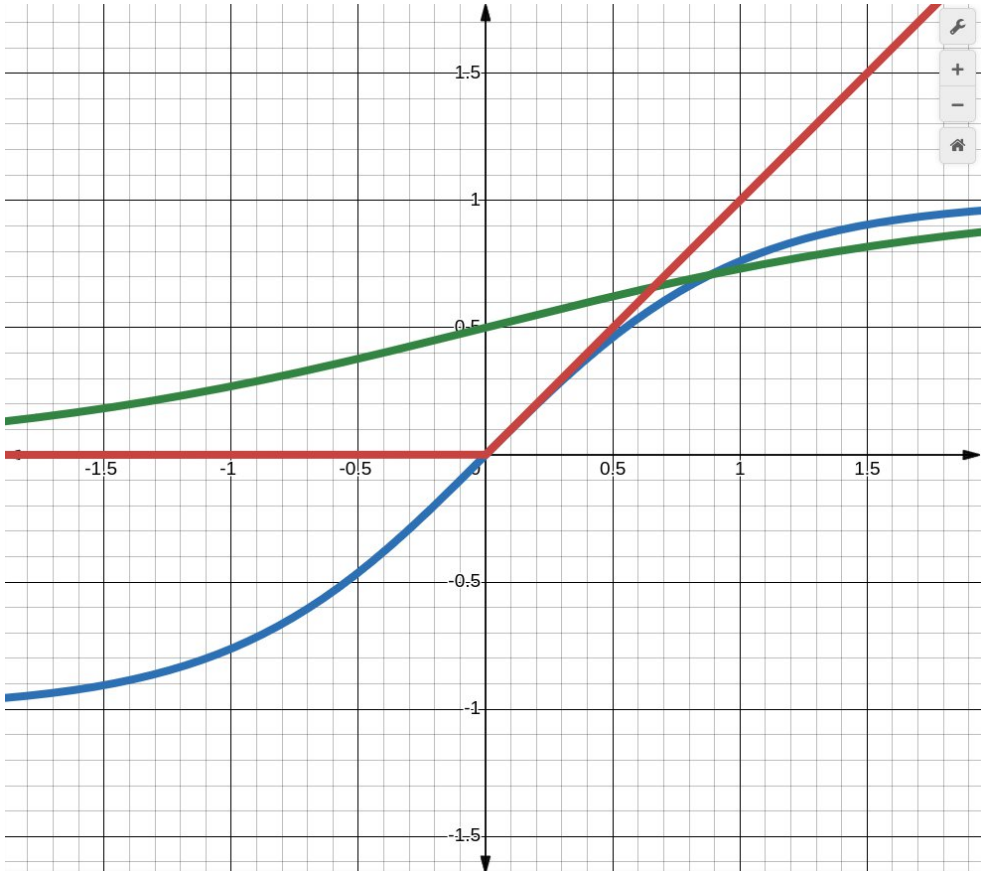
- Range
- Smoothness
- Monotonicity
- Zero-centered
- Monotonic derivative
- Computational cost

How to choose the right activation function ?



Here are the criteria to take in account :

Range
Smoothness
Monotonicity
Zero-centered
Monotonic derivative
Computational cost



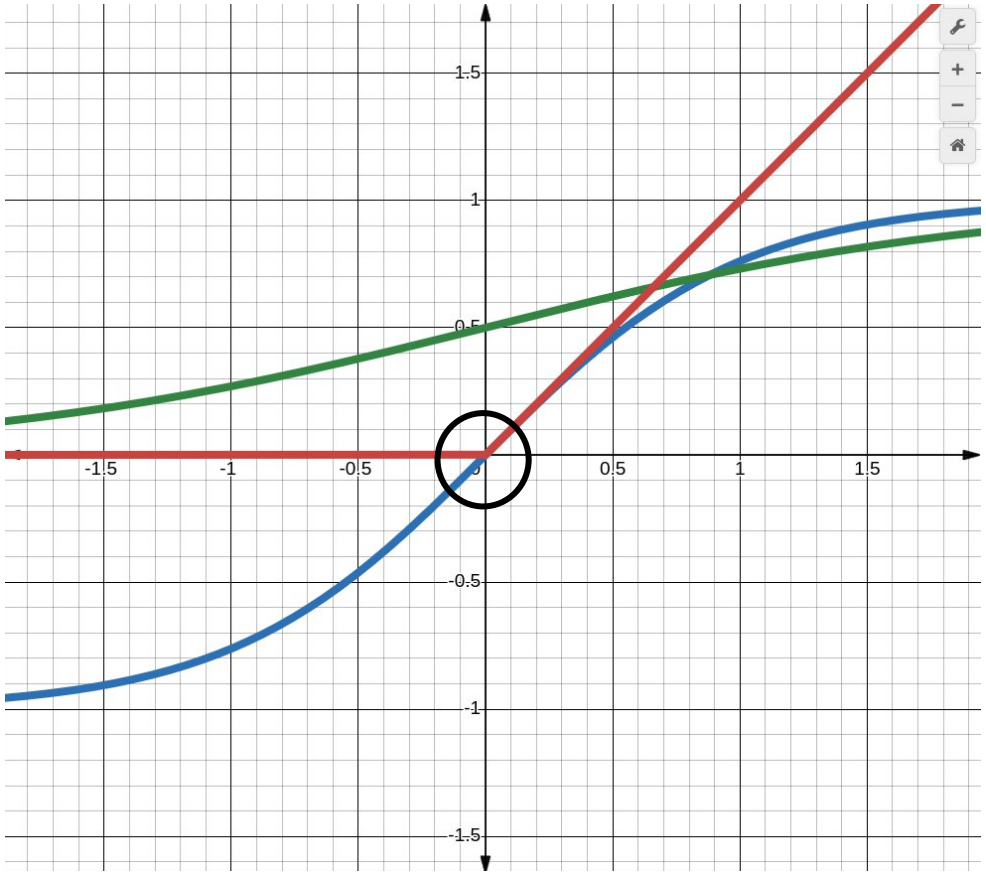
- ReLU : $[0,+\infty[$
- tanh(x) : $]-1,1[$
- sigmoïd : $]0,1[$

How to choose the right activation function ?



Here are the criteria to take in account :

- Range
- **Smoothness**
- Monotonicity
- Zero-centered
- Monotonic derivative
- Computational cost



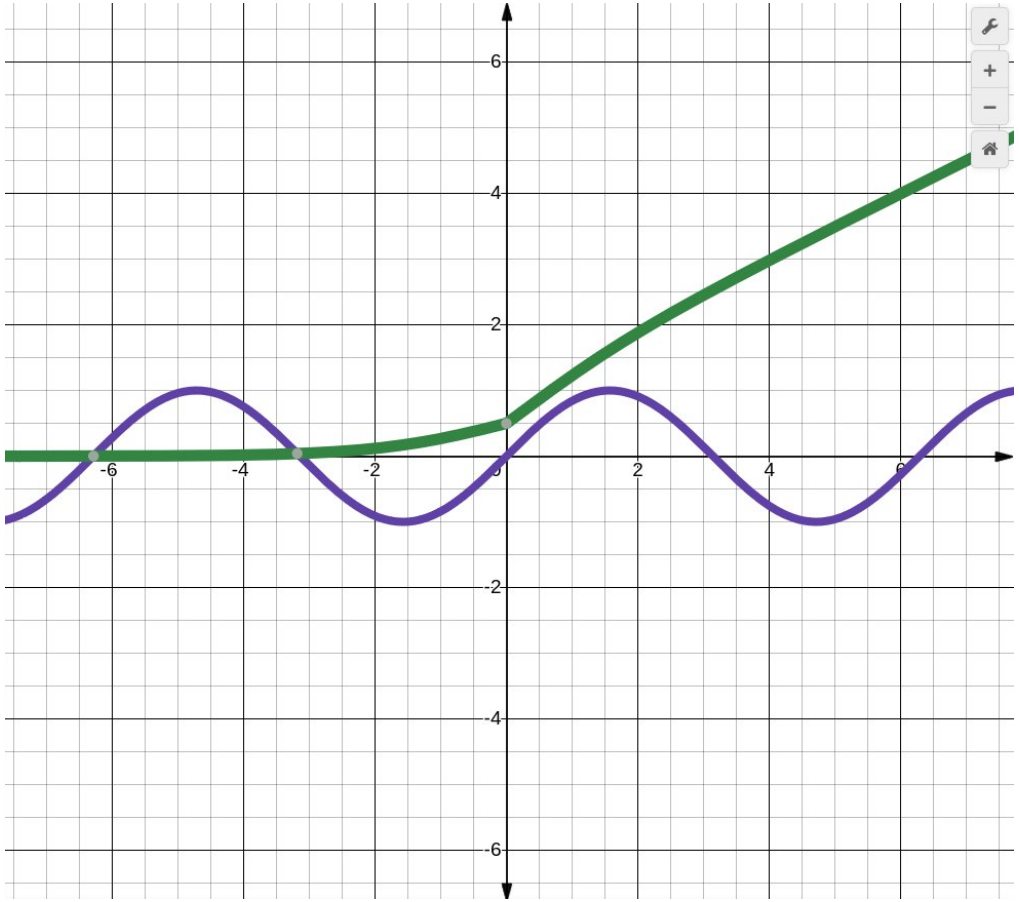
— **ReLU : angular**
— **tanh(x) : smooth**
— **sigmoïd : smooth**

How to choose the right activation function ?



Here are the criteria to take in account :

- Range
- Smoothness
- **Monotonicity**
- Zero-centered
- Monotonic derivative
- Computational cost



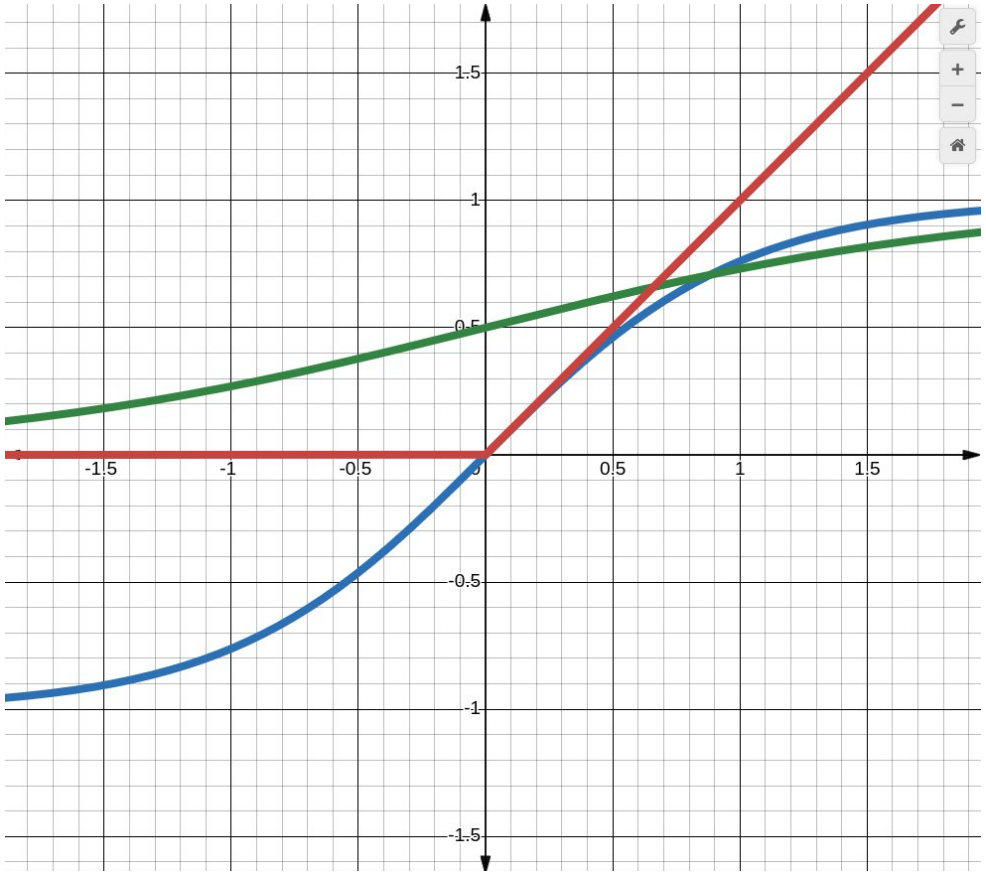
— Monotonous
— Not monotonous

How to choose the right activation function ?



Here are the criteria to take in account :

- Range
- Smoothness
- Monotonicity
- **Zero-centered**
- Monotonic derivative
- Computational cost



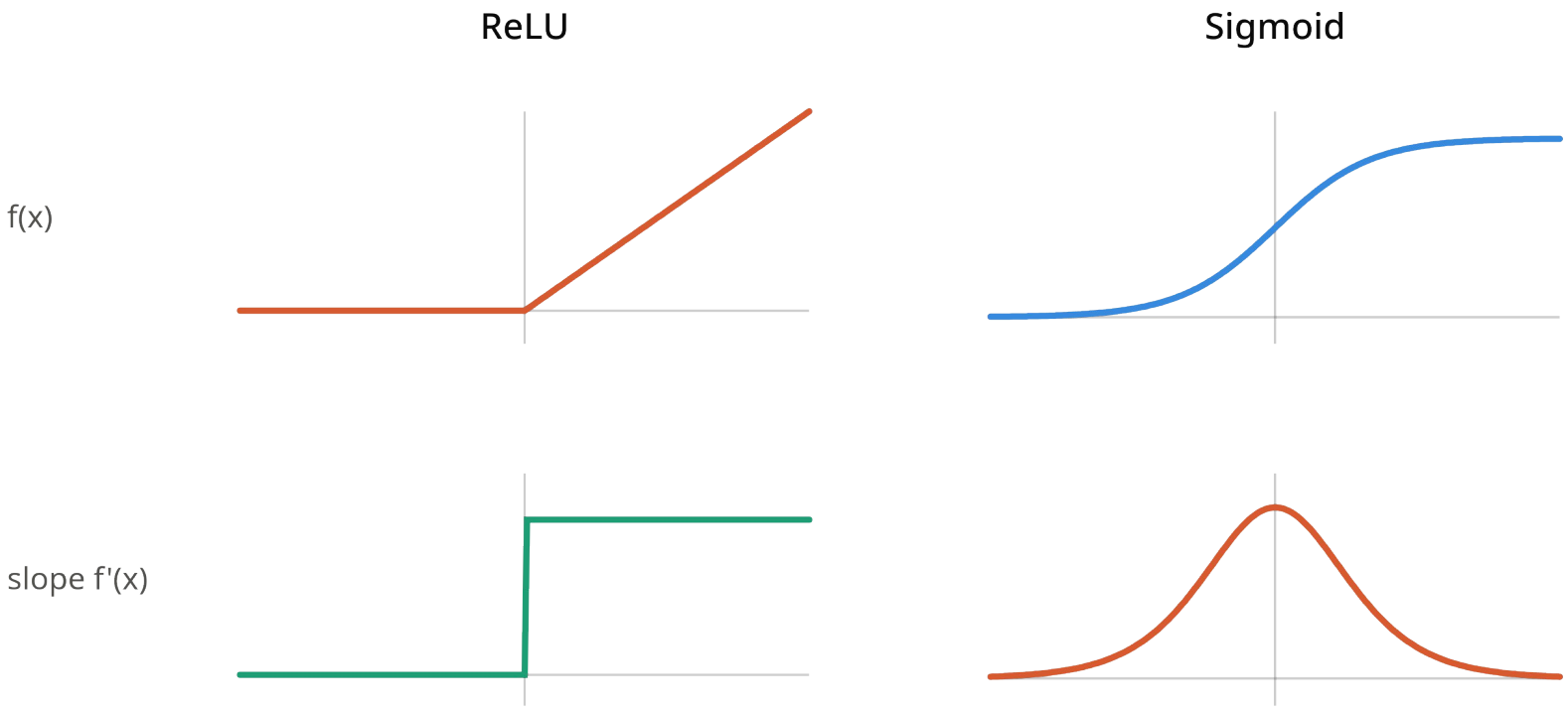
— tanh(x) : zero-centered
— ReLU : not zero-centered
— sigmoïd : not zero-centered

How to choose the right activation function ?



Here are the criteria to take in account :

- Range
- Smoothness
- Monotonicity
- Zero-centered
- **Monotonic derivative**
- Computational cost



How to choose the right activation function ?



Here are the criteria to take in account :

- Range
- Smoothness
- Monotonicity
- Zero-centered
- Monotonic derivative
- **Computational cost**

Sigmoid
1 / (1 + e^-x)

exponential | addition | division

ReLU
max(0, x)

comparison

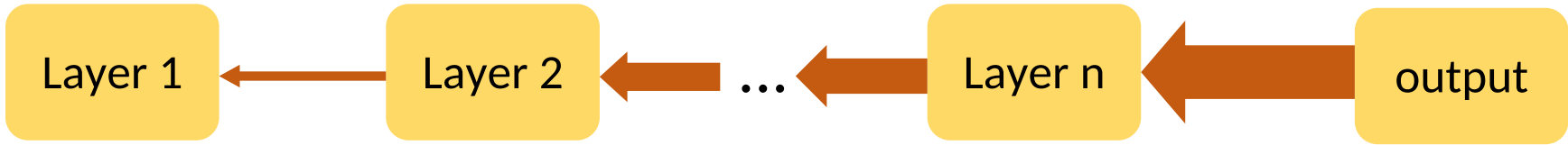
Applied to every neuron, at every step: a small saving per neuron becomes a big saving overall

Problems to avoid

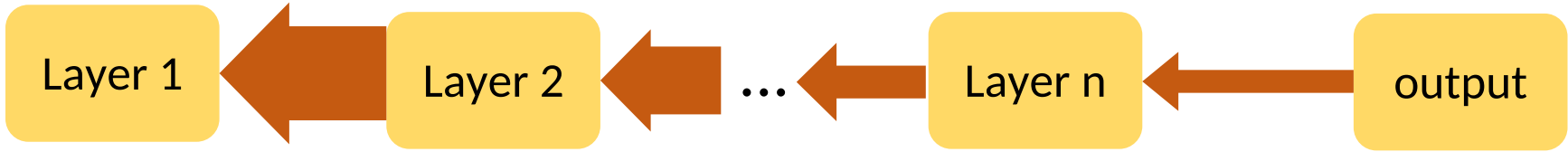
The Gradient

The slope of the cost function with respect to each weight; it tells the model in which direction, and by how much, to adjust its weights to reduce the error. During training, it is computed layer by layer, from the output back to the input (backpropagation).

Vanishing gradients



Exploding gradients



Dying neurons

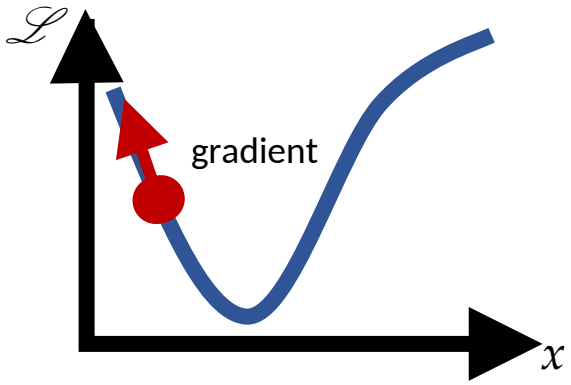


the weights are never updated

Optimization

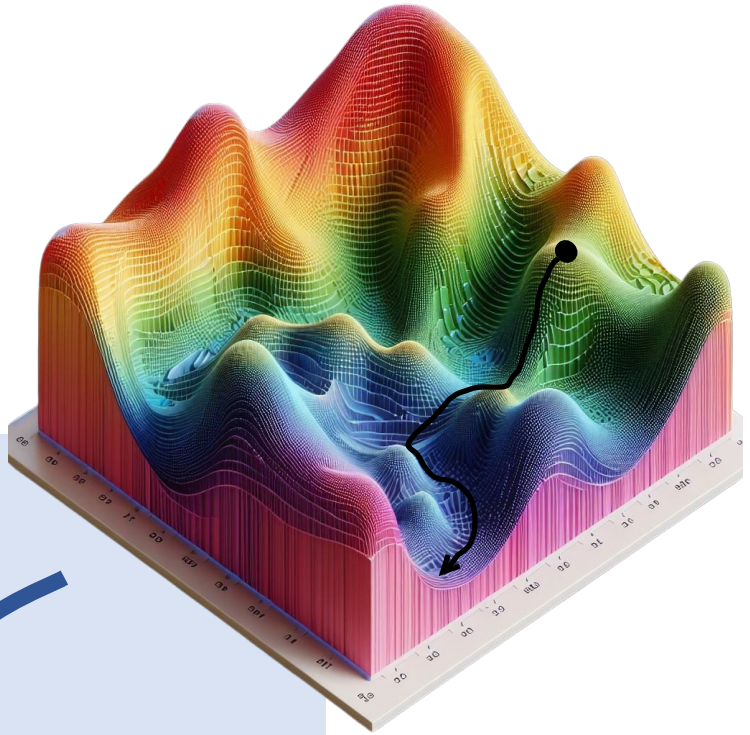
- **Linear systems**
  - LU, QR, Cholesky, Jacobi, Gauss-Seidel, CG, PCG, …
- **Non-linear systems**
  - First order : Gradient Descent, SGD
  - Second order : Newton, Gauss-Newton, LM, (L)BFGS
- **Others**
  - Genetic algorithms, Metropolis-Hastings, …
  - Complex and constrained solver : ADMM, Primal-Dual, …

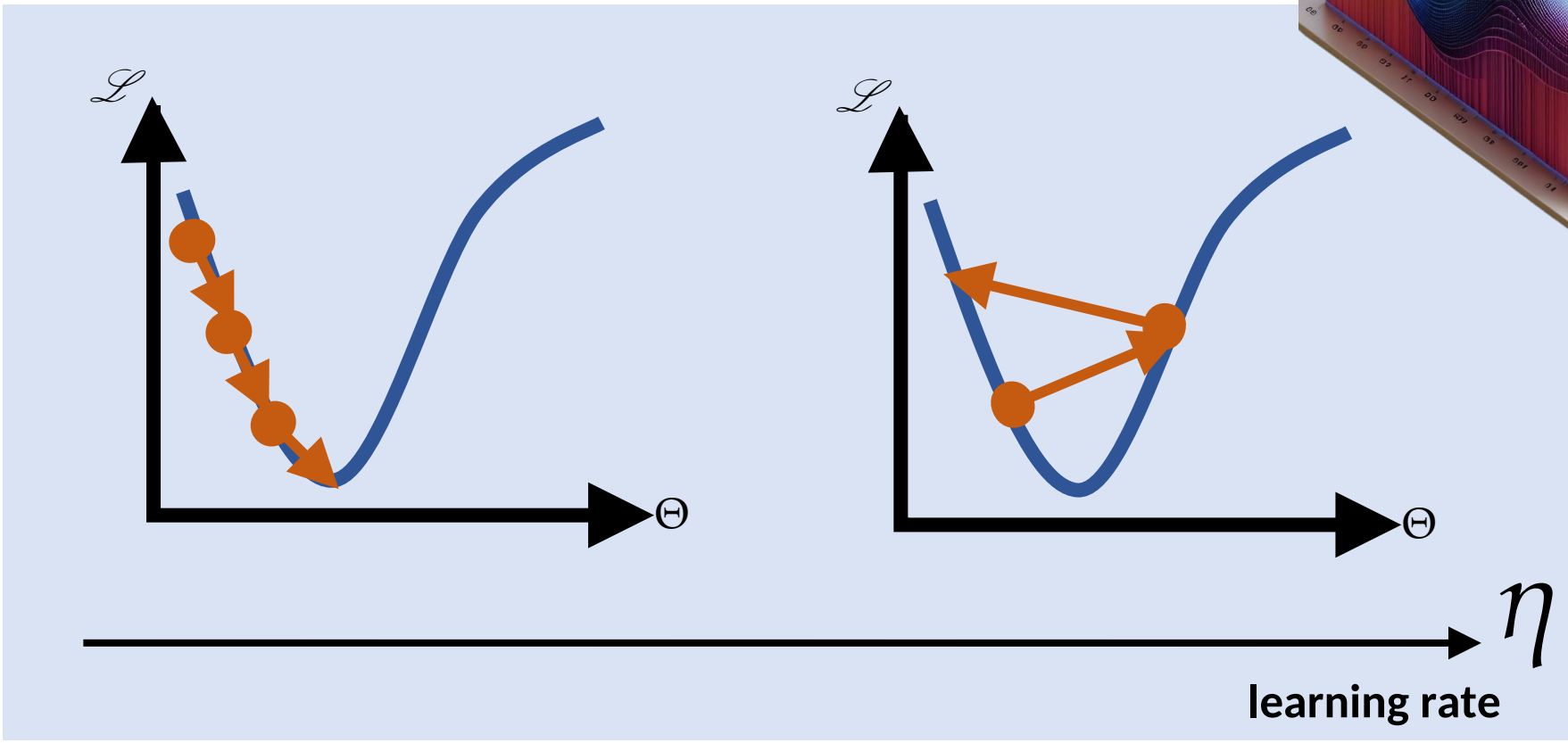
Gradient descent



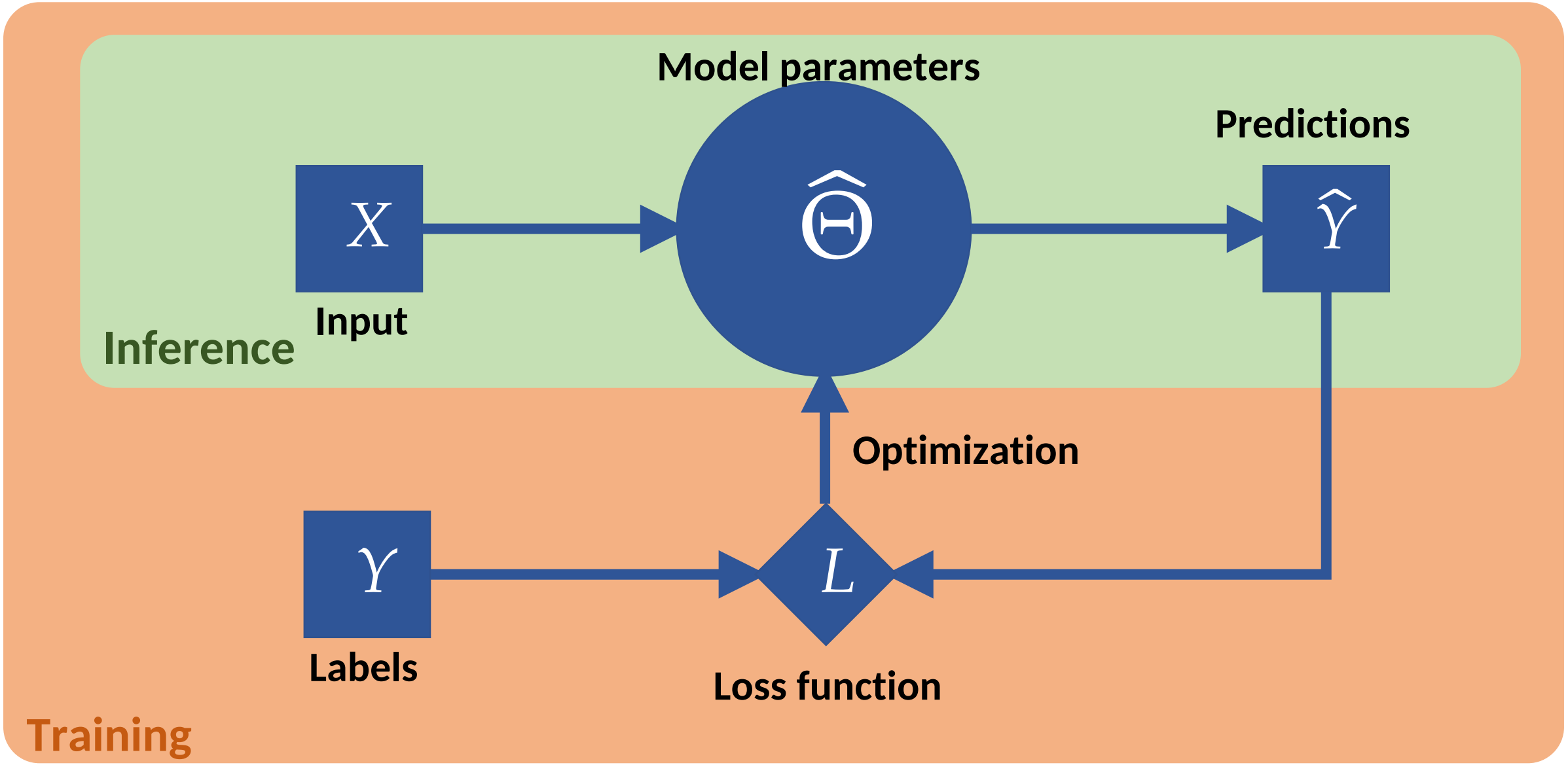
Learning rate η

A tuning hyperparameter in machine learning that controls the step size taken during each iteration of an optimization algorithm (such as gradient descent) as it moves toward the minimum of a loss function.

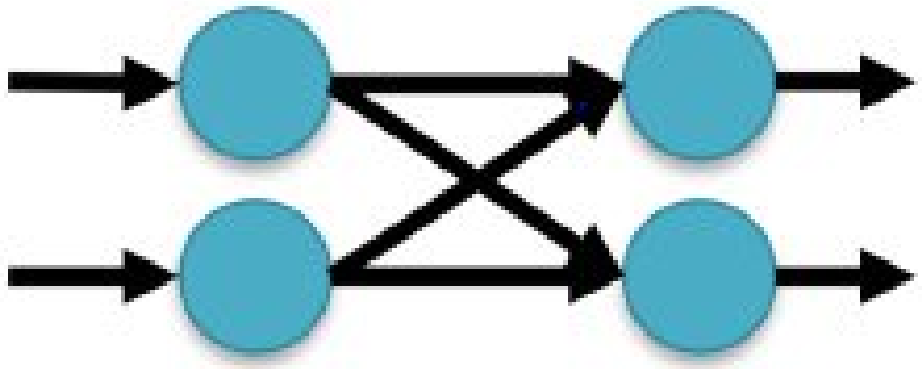




Training a model



Calculation graph and chain rule



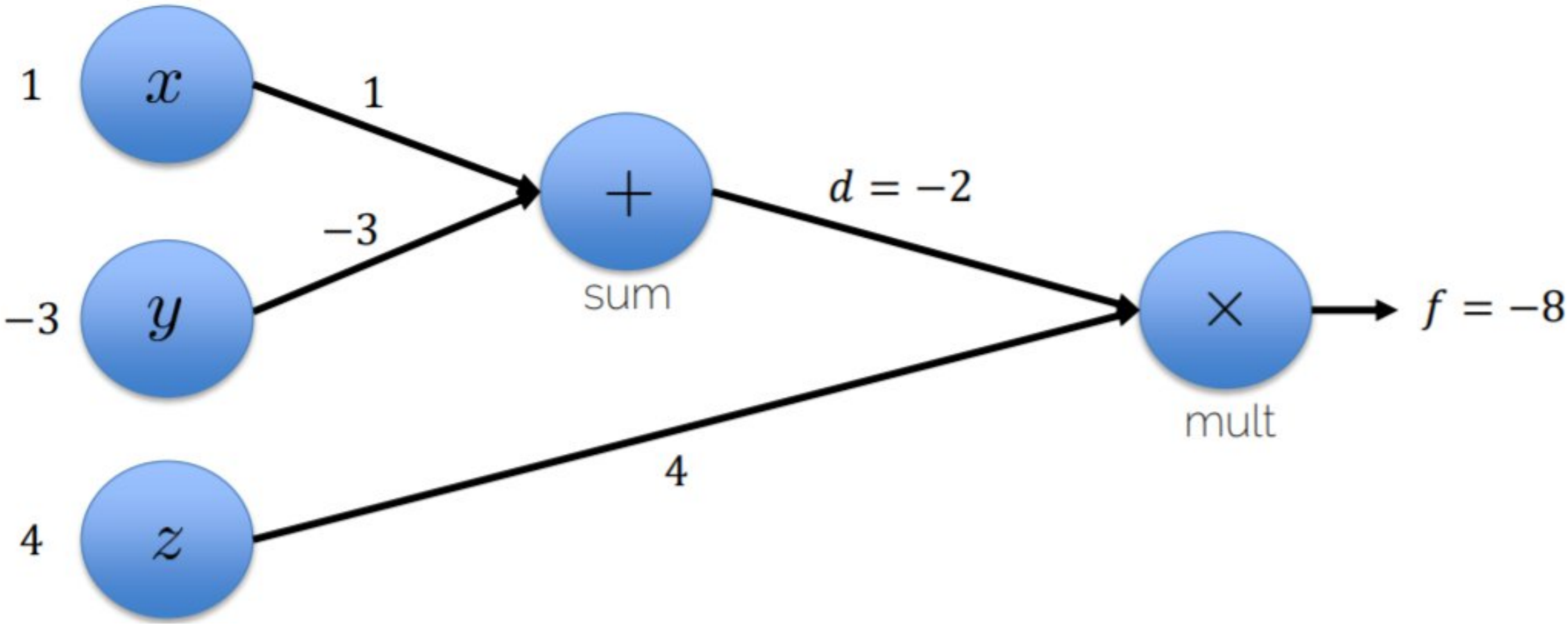
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial d} \cdot \frac{\partial d}{\partial x}$$

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{i,k}}$$

Forward pass

- $f(x, y, z) = (x + y) \cdot z$

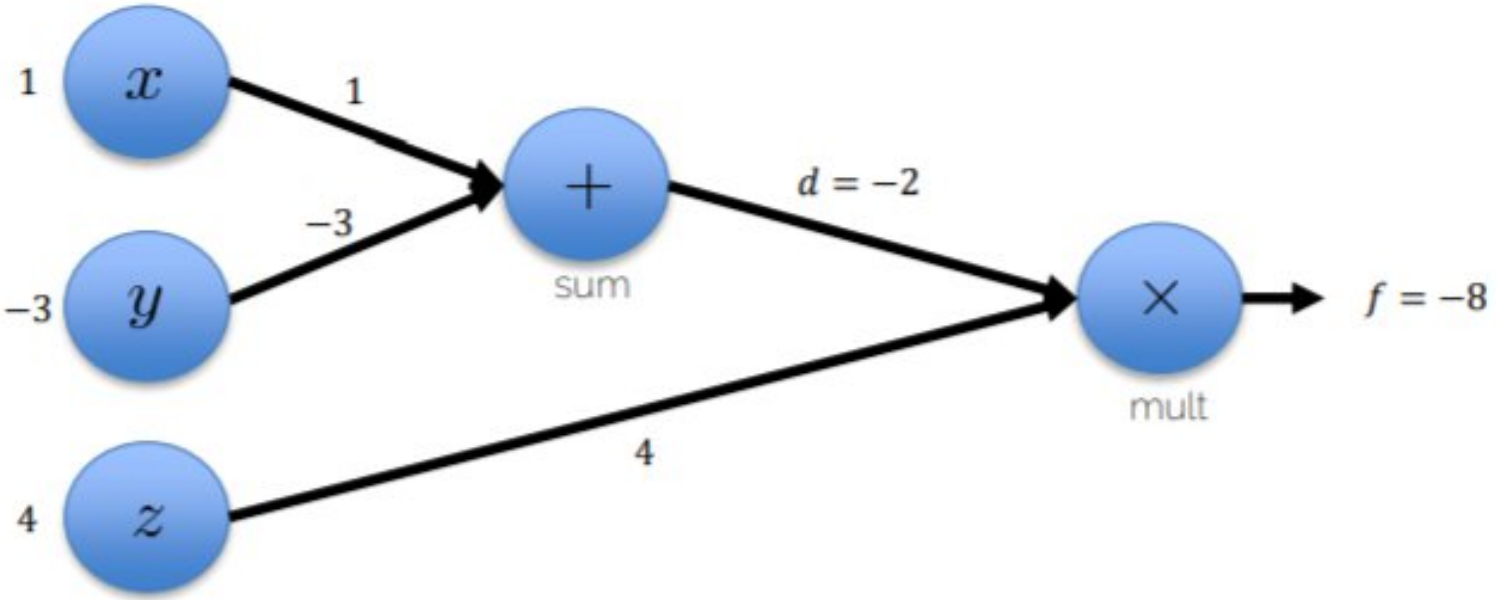
Initialization $x = 1, y = -3, z = 4$



Backward pass

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

with $x = 1, y = -3, z = 4$



$$d = x + y \qquad \frac{\partial d}{\partial x} = 1, \frac{\partial d}{\partial y} = 1$$

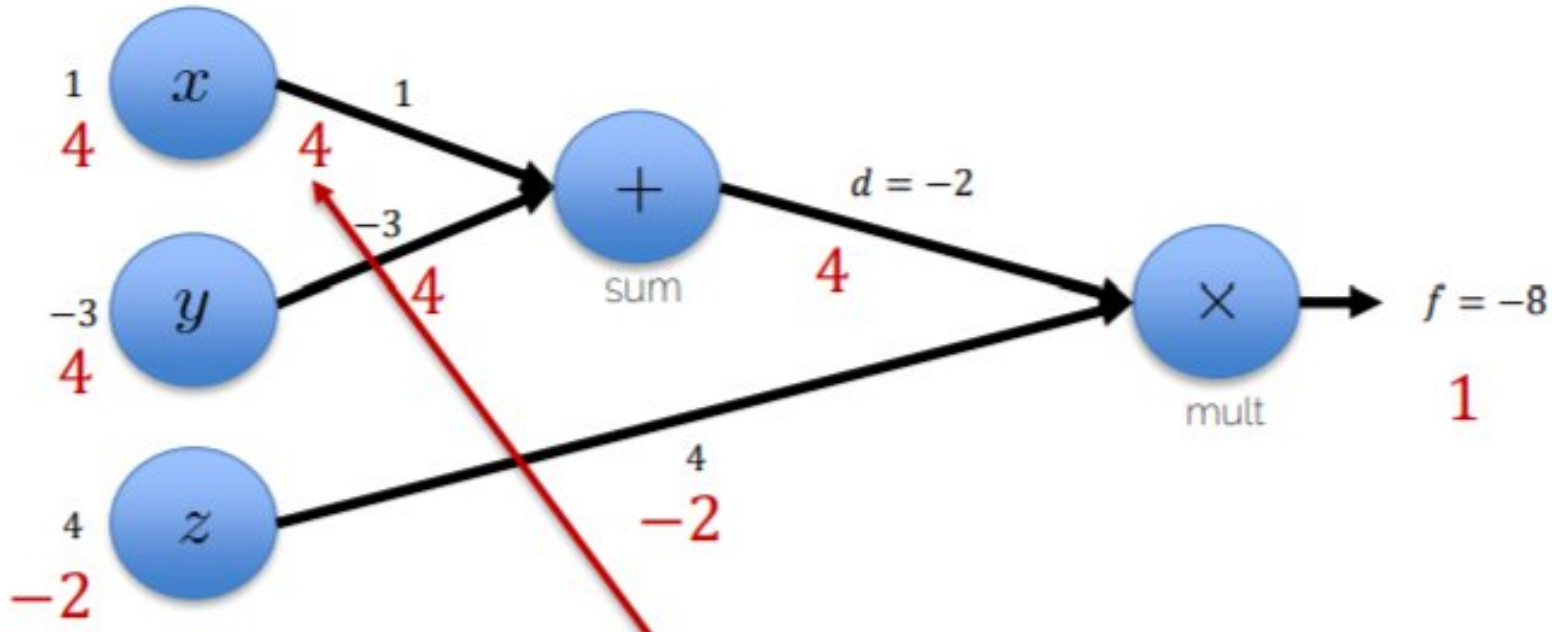
$$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

Backward pass

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

with $x = 1, y = -3, z = 4$



$$d = x + y \qquad \frac{\partial d}{\partial x} = 1 \quad \frac{\partial d}{\partial y} = 1$$

$$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d$$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

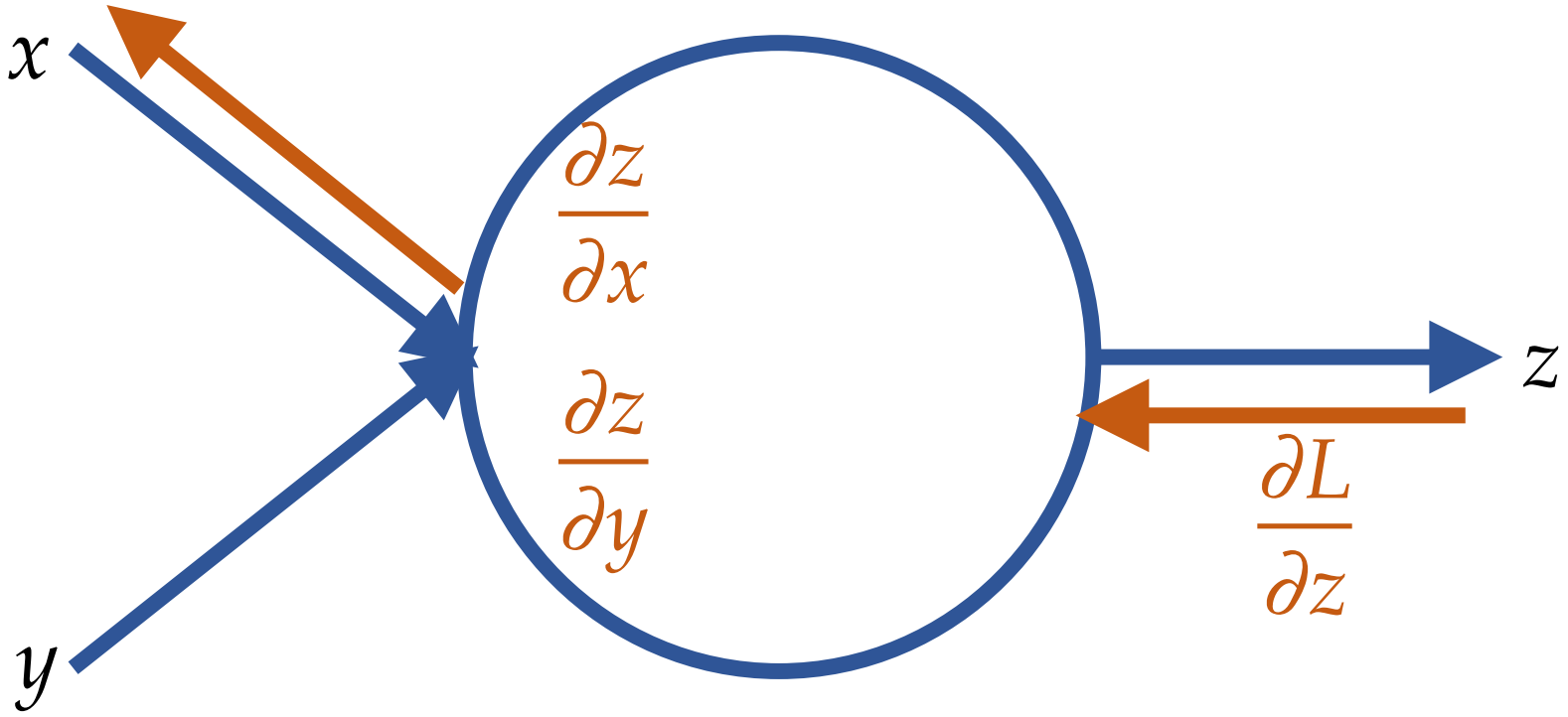
Chain Rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial d} \cdot \frac{\partial d}{\partial x}$$

$$\rightarrow \frac{\partial f}{\partial x} = 4 \cdot 1 = 4$$

Gradient Flow

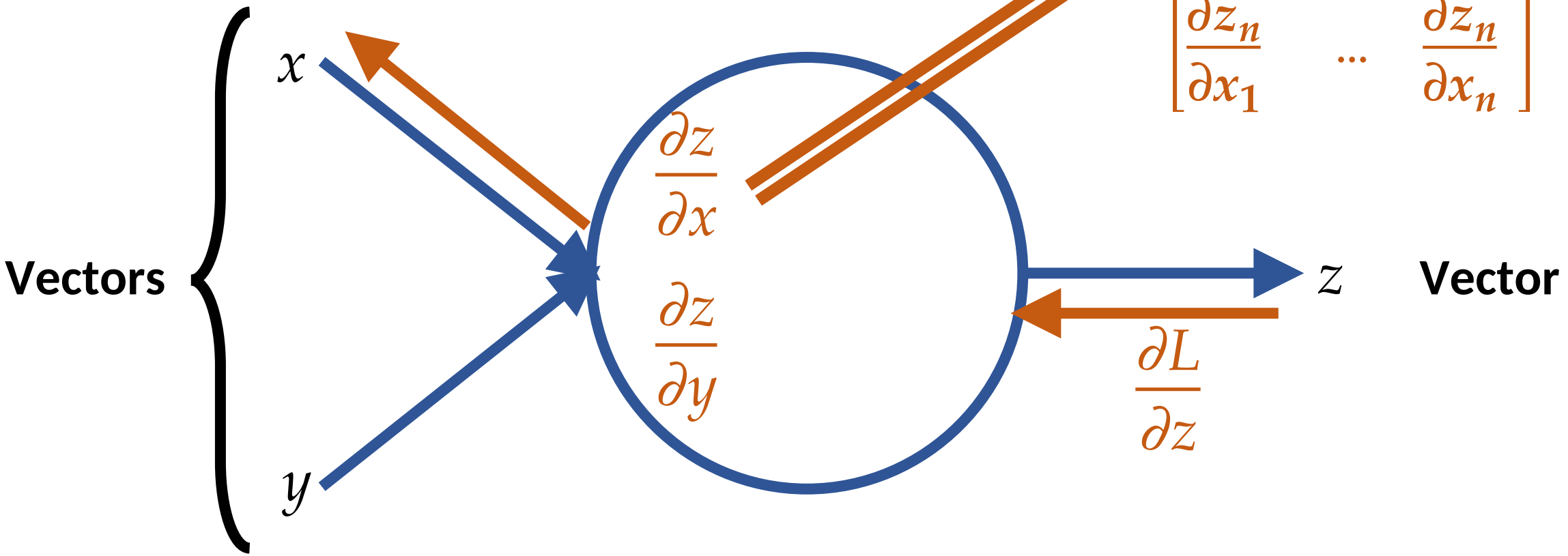
$$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$$



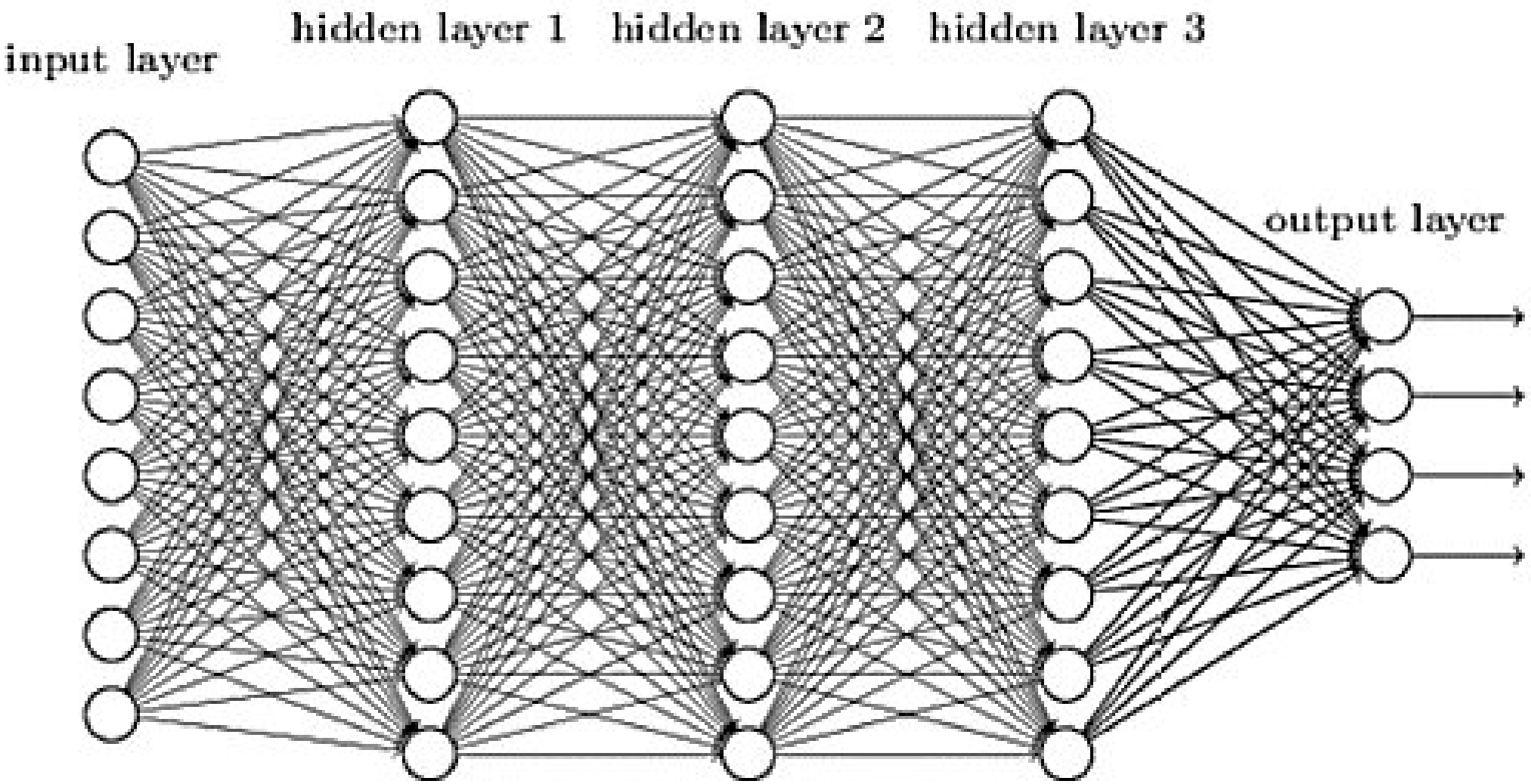
Gradient Flow

$$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$$

$$\begin{bmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_n}{\partial x_1} & \cdots & \frac{\partial z_n}{\partial x_n} \end{bmatrix}$$



Network size



References

- Chollet, Francois. *Deep learning with Python*. Simon and Schuster, 2021.
- *CS230 Deep Learning*. cs230.stanford.edu. Accessed 14 Mar. 2022.
- *I2DL*. niessner.github.io/I2DL. Accessed 14 Mar. 2022.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Definitions

Artificial Intelligence

Systems capable of reproducing human actions or decisions

Machine Learning

Algorithms mainly based on statistical methods, often with an iterative aspect from which comes the term «Learning »

Deep Learning

A group of models based on logical units called neurons and distributed in layers

Discrete Data

data that can only take specific, separate values, usually whole numbers or categories you can count

Continuous Data

data that can take any value within a range, including fractions and decimals, usually something you measure

A neuron

A small computing unit that takes several inputs, combines them using weighted parameters, and passes the result through an activation function to produce an output.

Activation function

A non-linear function applied to the weighted sum of a neuron's inputs, allowing the network to model non-linear relationships

Loss function

Measures the error **for a single observation i** : the gap between the predicted value $\hat{y}_i$ and the true value y_i, for that specific data point.

Cost function

the average of the loss **over the entire dataset** (but computed from the loss rather than directly)

The Gradient

The slope of the cost function with respect to each weight; it tells the model in which direction, and by how much, to adjust its weights to reduce the error. During training, it is computed layer by layer, from the output back to the input (backpropagation).

Learning rate η

A tuning hyperparameter in machine learning that controls the step size taken during each iteration of an optimization algorithm (such as gradient descent) as it moves toward the minimum of a loss function.