cnrs formation entreprises

Introduction to Neural Networks

Direction de l'innovation et des relations avec les entreprises

June 28th, 2023

Aurélien Garivier, Rémi Gribonval, Mathurin Massias, Titouan Vayer

Lecture 5: introduction to Neural Networks

- **Context and Definitions**
  - The success of « deep learning »
  - A Neural Network in practice
- **Training a network**
  - Reminders on Gradient Descent
  - Auto-differentiation
  - A few special focuses
- **Framework :** PyTorch
  - Classification and regression in practice
  - Auto-encodeurs

Successes of « deep learning »



https://www.youtube.com/watch?v=MwtVkPKx3RA

https://www.youtube.com/watch?v=yZu24pddzwk

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

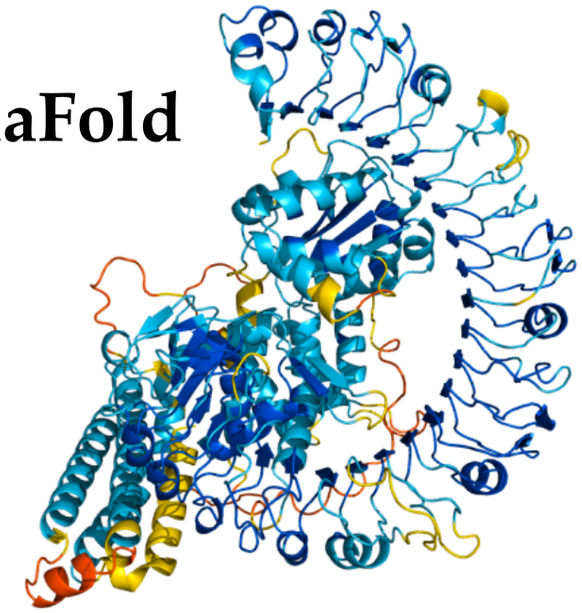
Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.



https://www.craiyon.com/

https://stablediffusionweb.com/

AlphaFold







A neural network in practice

- From a linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

Data

A neural network in practice

- From a linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

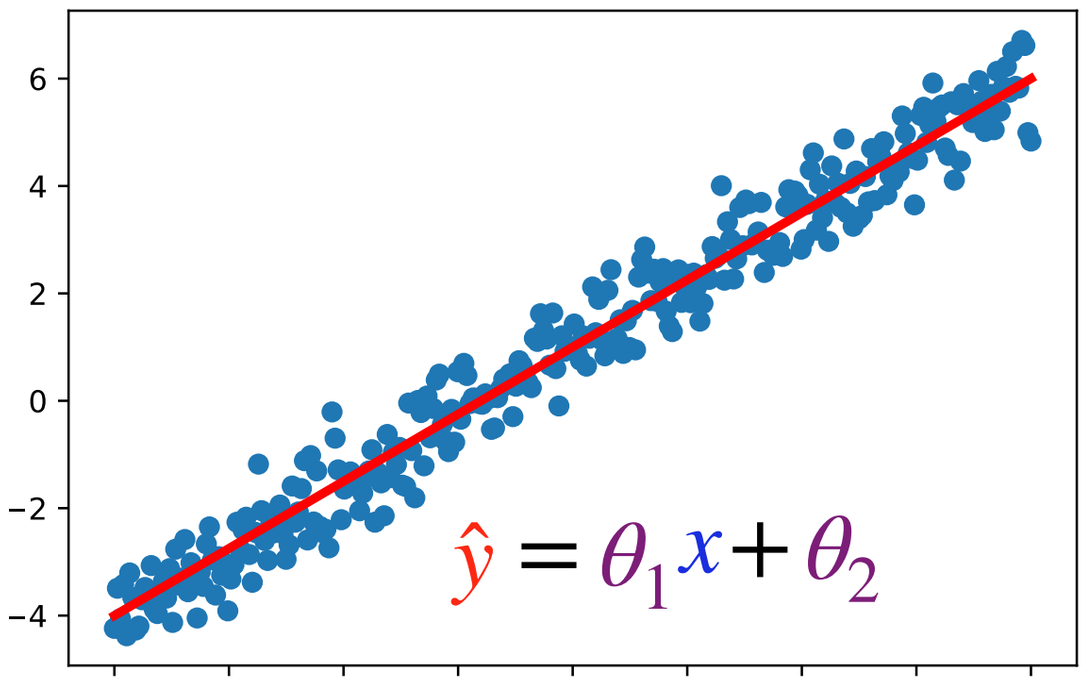
Parameters (to fit)

A neural network in practice

- Du Model linéaire:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

Output (to be predicted)

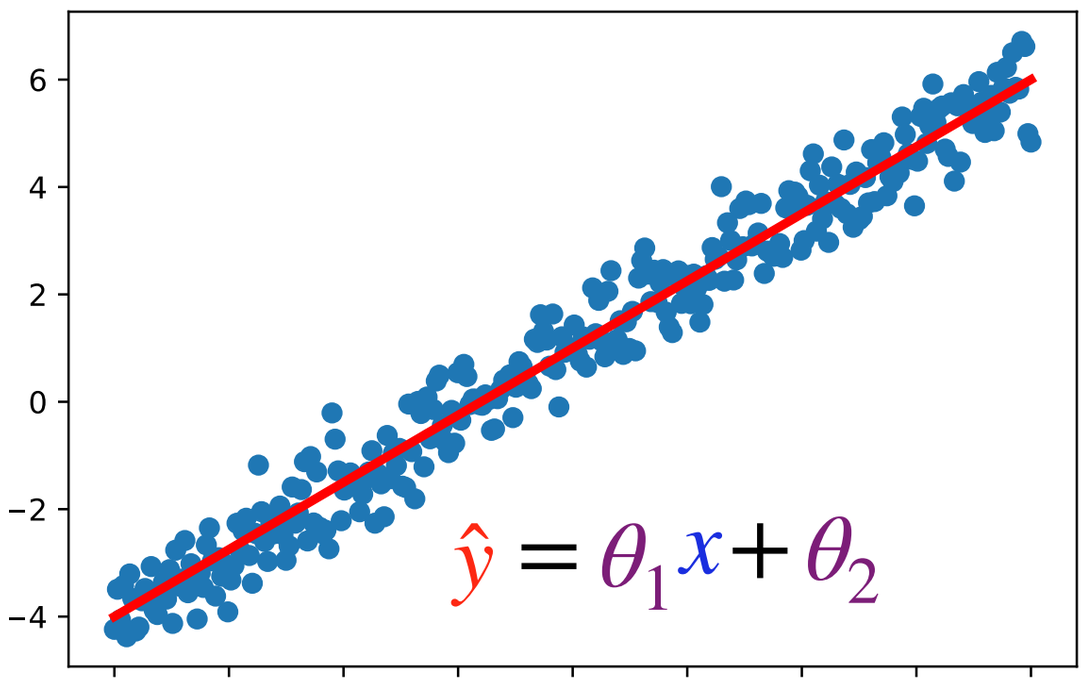


A neural network in practice

- From the linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

Output (to be predicted)



Quadratic loss:

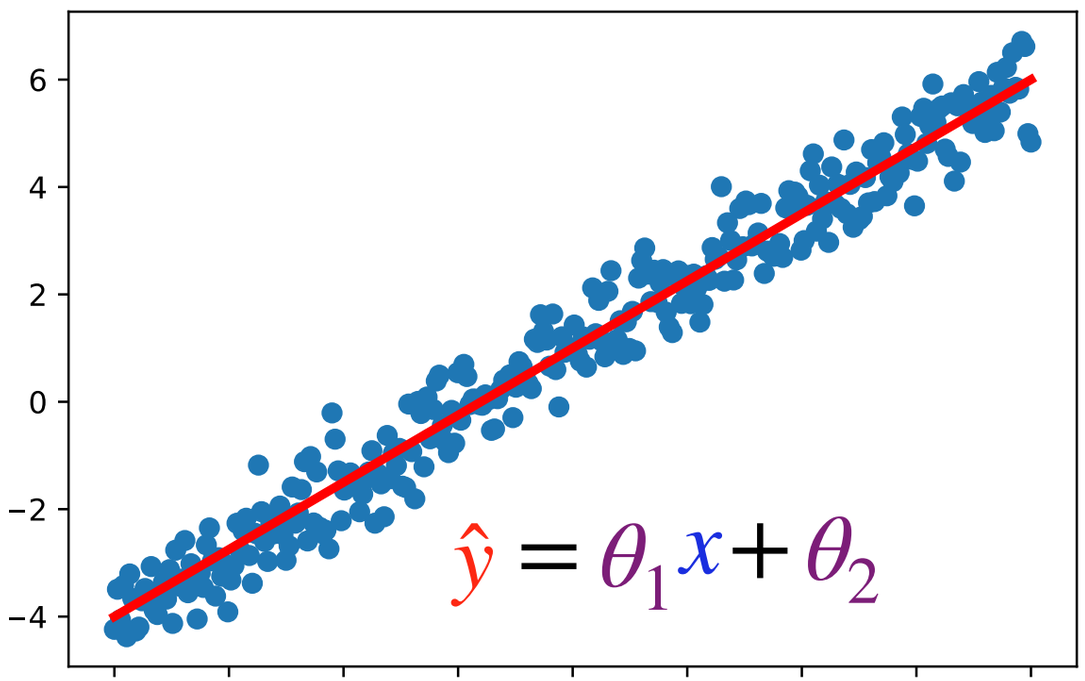
$$\min_{\boldsymbol{\theta}=(\theta_1,\cdots,\theta_n)} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\theta X}\|^2$$

A neural network in practice

- From the linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

Output (to be predicted)



Quadratic loss with penalization

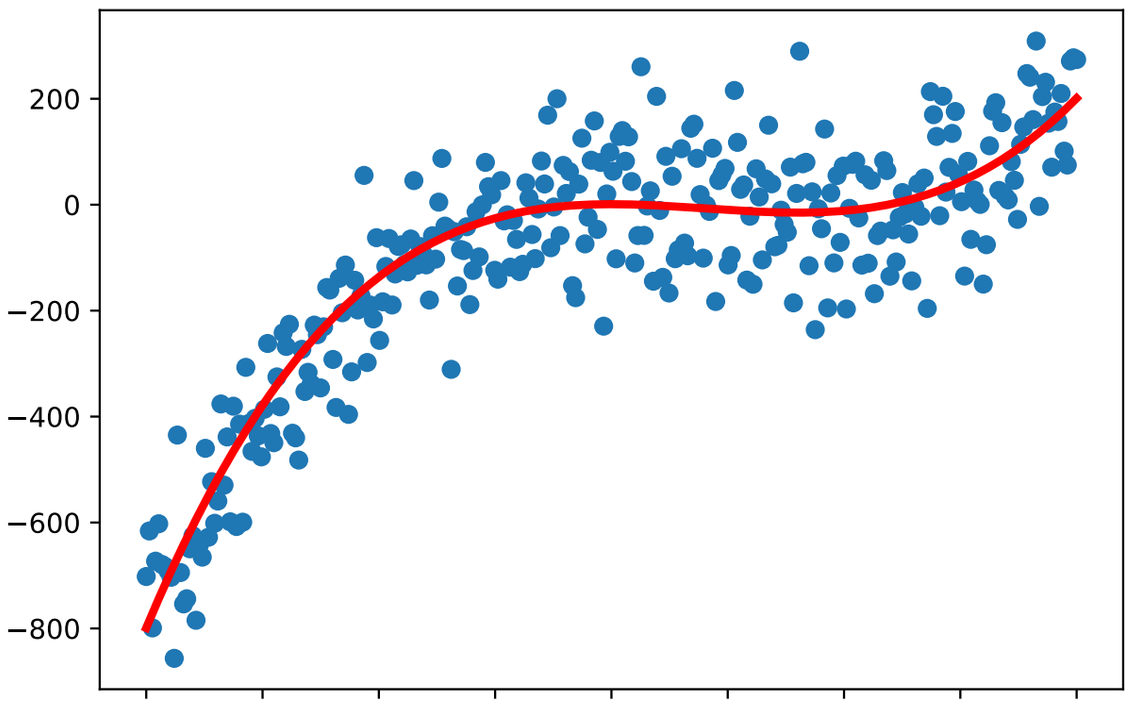
$$\min_{\boldsymbol{\theta}=(\theta_1,\cdots,\theta_n)} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\theta X}\|^2 + \lambda \|\boldsymbol{\theta}\|^2$$

A neural network in practice

- From the linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

- To non-linear models:



$$\mathbf{y} \approx f_{\theta}(\mathbf{x}_1, \cdots, \mathbf{x}_n)$$

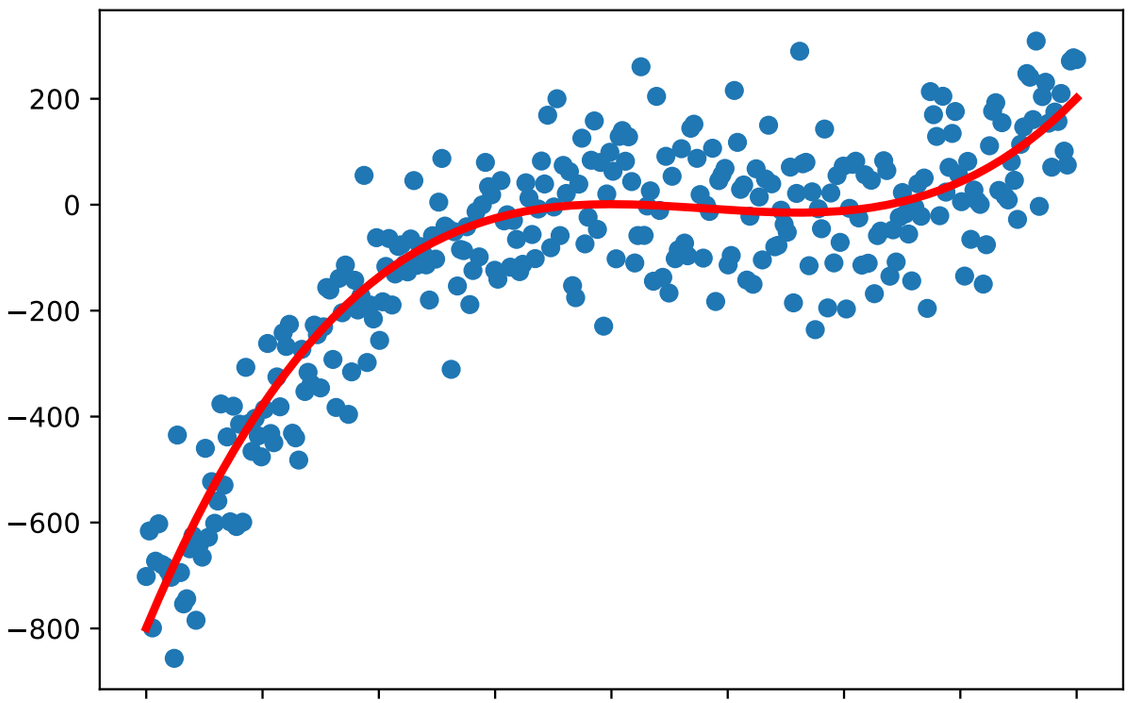
Non-linear function depending on some **parameters**

A neural network in practice

- From the linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

- To non-linear models:



$$\mathbf{y} \approx f_{\boldsymbol{\theta}}(\mathbf{x}_1, \cdots, \mathbf{x}_n)$$

Non-linear function depending on some parameters

Q: How to choose function $f_{\boldsymbol{\theta}}$?

A neural network in practice

- From the linear model:

$$\mathbf{y} \approx \theta_1 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \cdots + \theta_n \mathbf{x}_n$$

- To non-linear models:

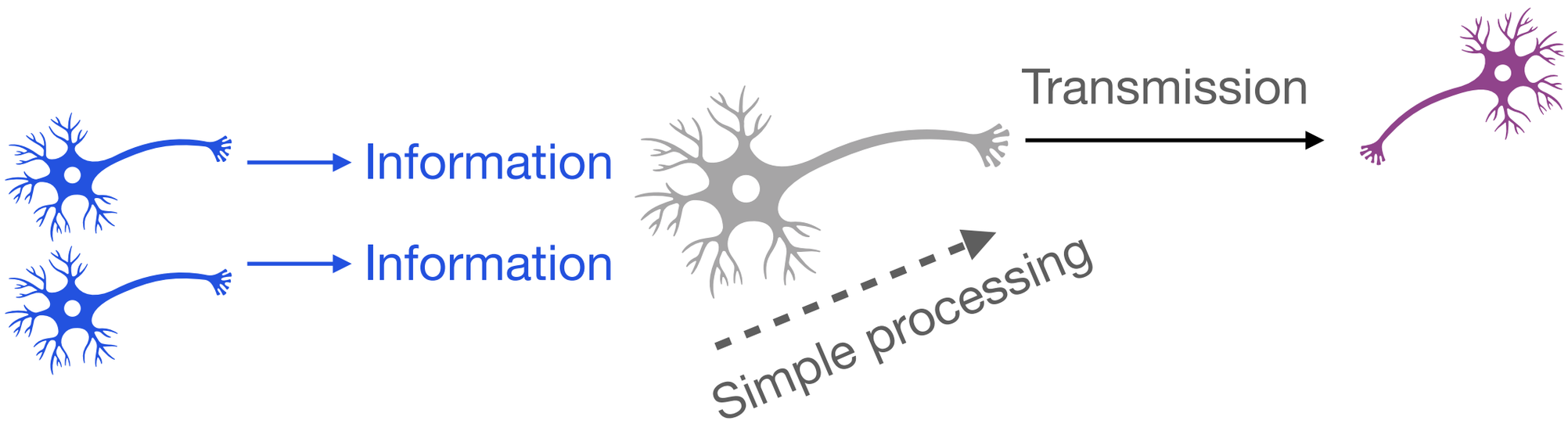
$$\mathbf{y} \approx f_{\boldsymbol{\theta}}(\mathbf{x}_1, \cdots, \mathbf{x}_n)$$

- To neural networks:

$f_{\boldsymbol{\theta}} =$ Composition of **multiple simple functions**

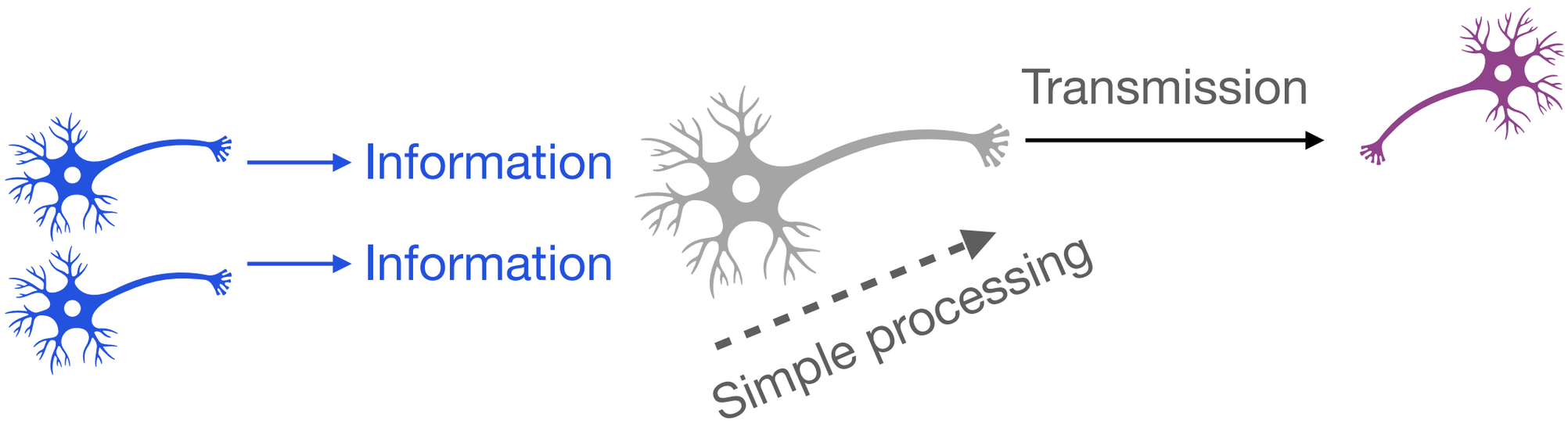
A neural network in practice

Origin: biological metaphor

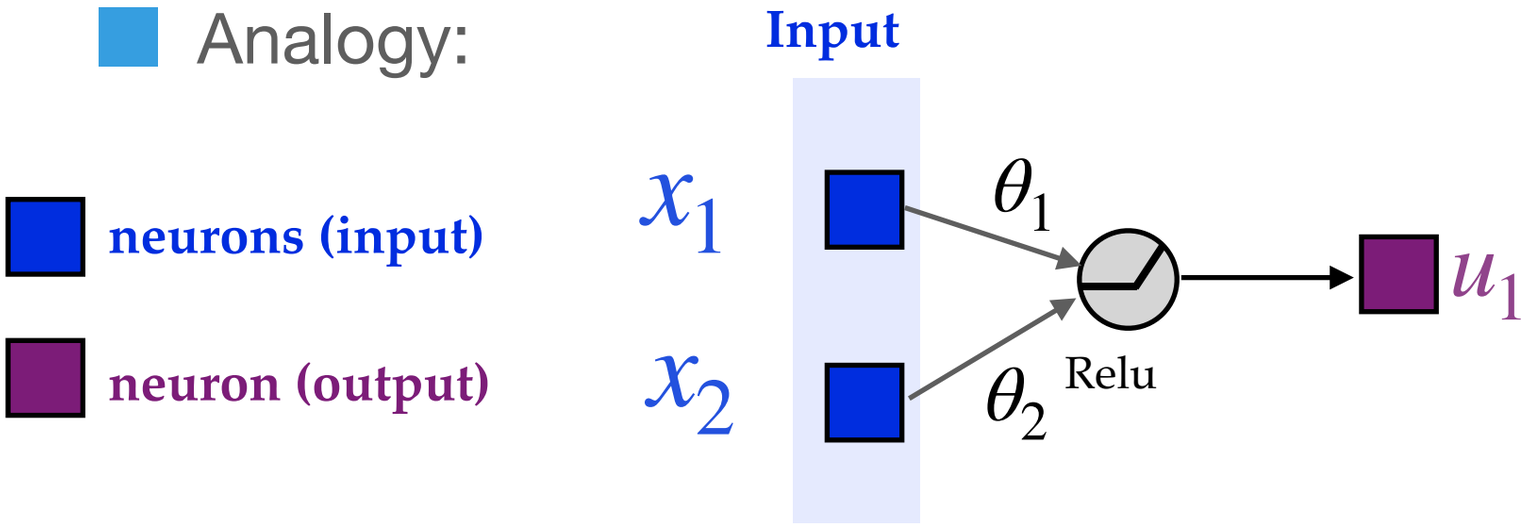


A neural network in practice

Origin: biological metaphor

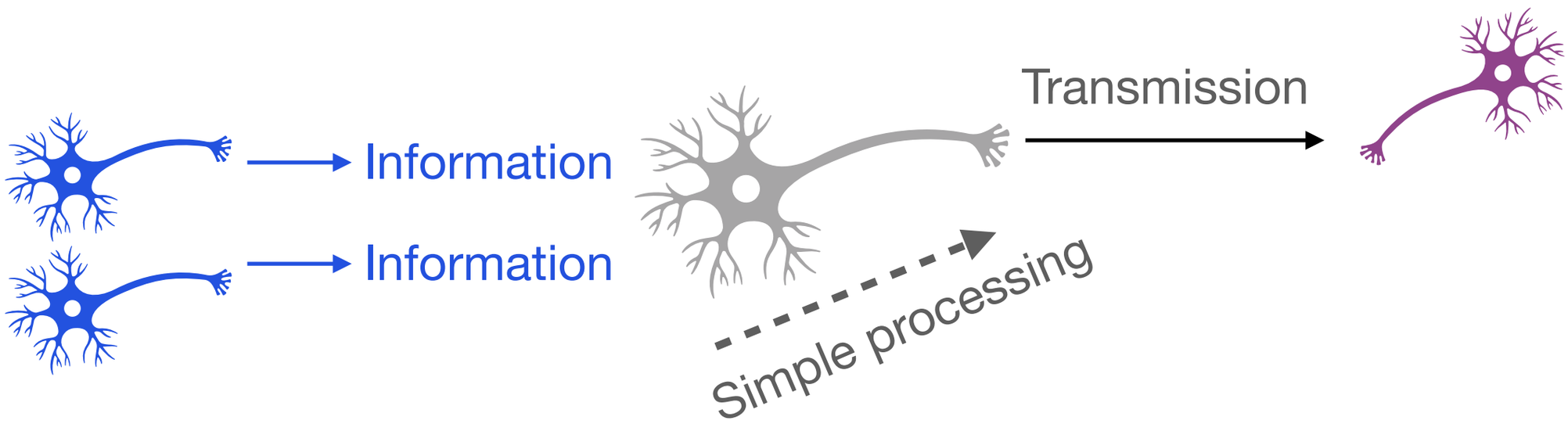


Analogy:



A neural network in practice

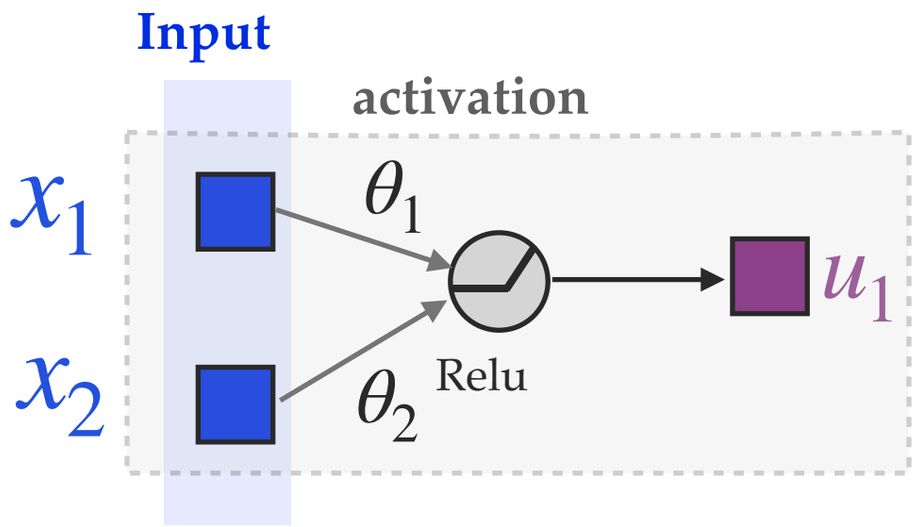
Origin: biological metaphor



Analogy:

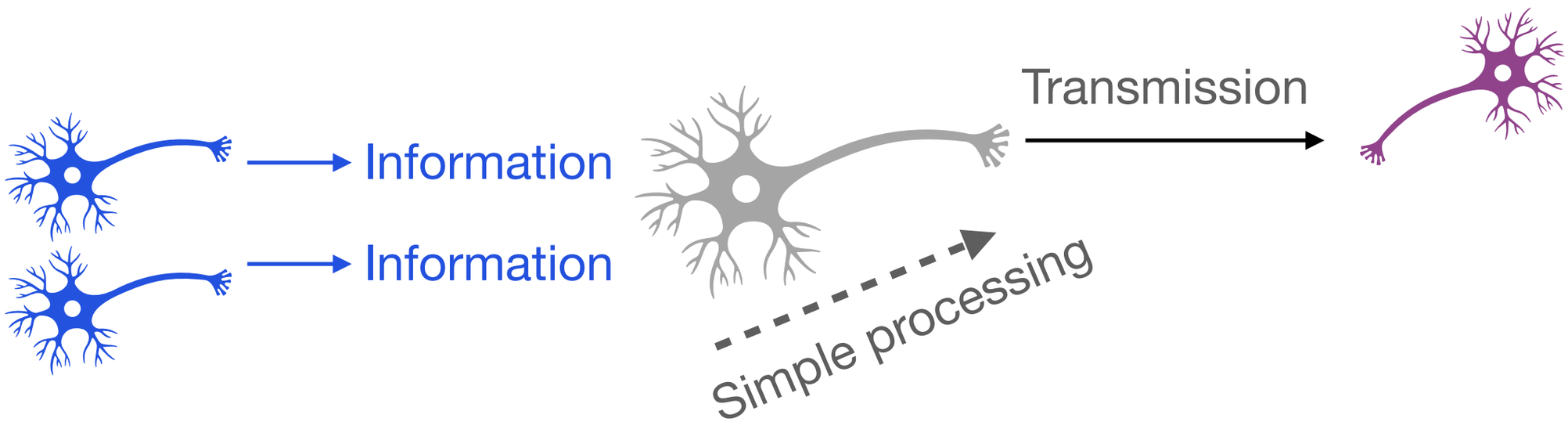
neurons (input)

neuron (output)



A neural network in practice

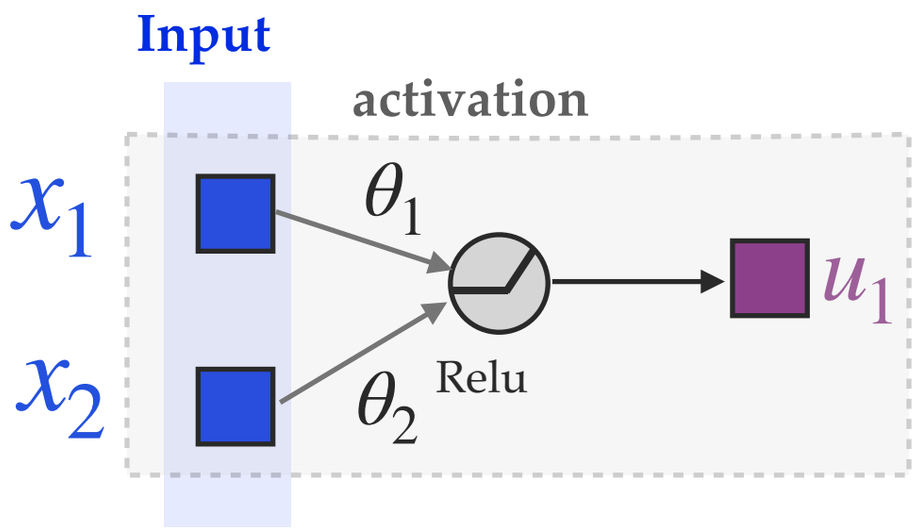
Origin: biological metaphor



Analogy:

neurons (input)

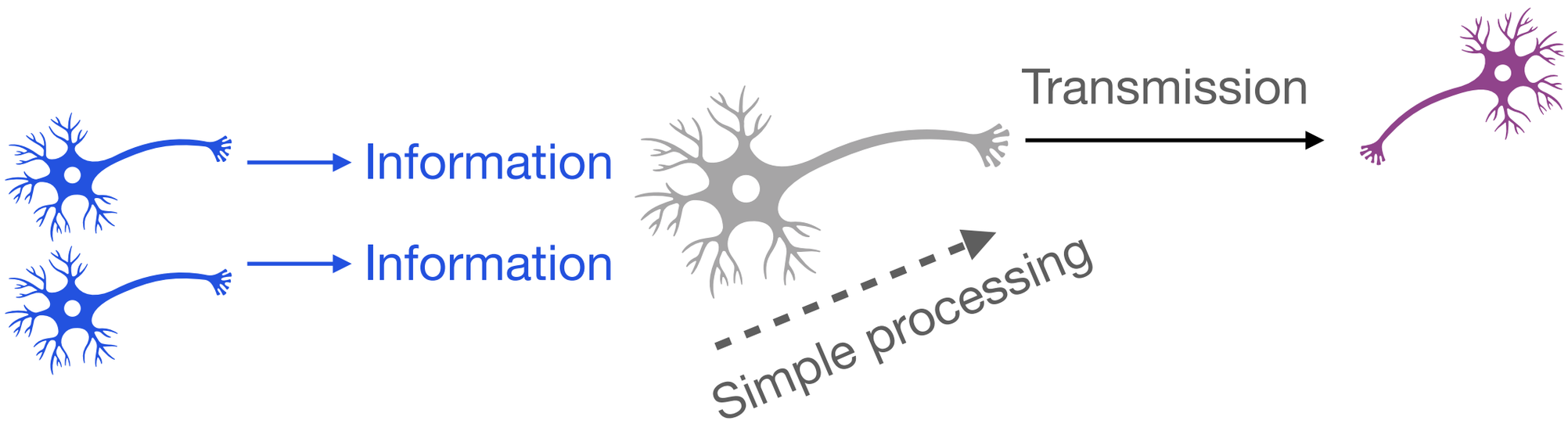
neuron (output)



$$u_1 = \max(\theta_1 x_1 + \theta_2 x_2 + \theta_3, 0)$$

A neural network in practice

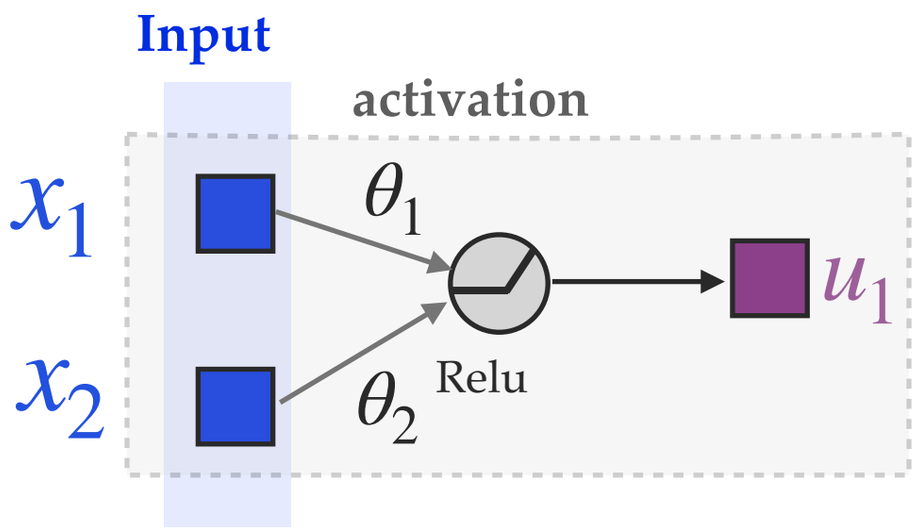
Origin: biological metaphor



Analogy:

neurons (input)

neuron (output)

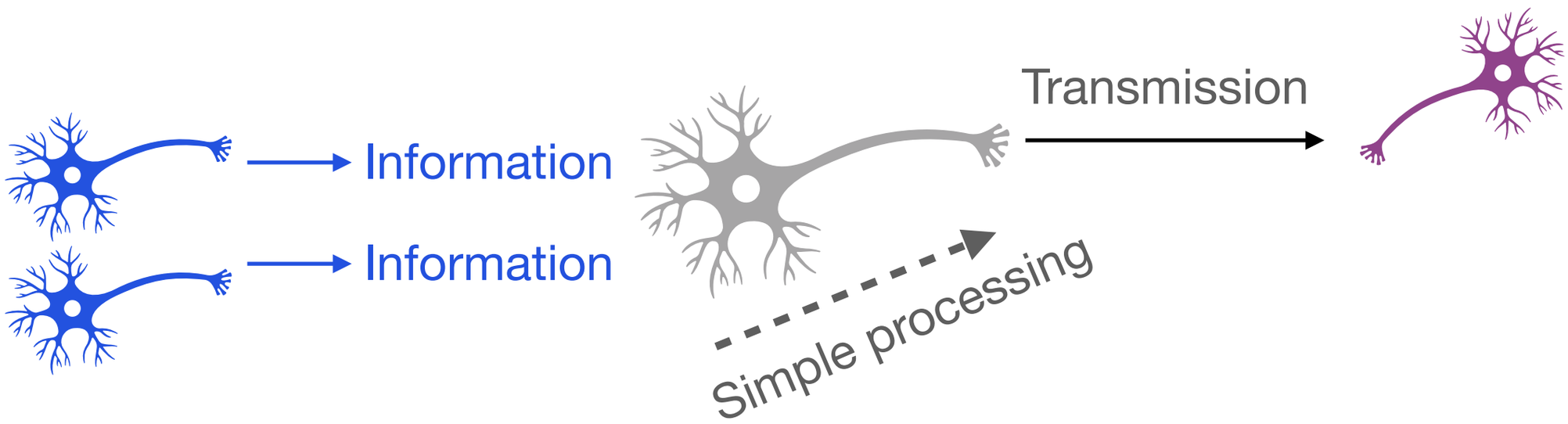


$$u_1 = \max(\theta_1 x_1 + \theta_2 x_2 + \theta_3, 0)$$

$\theta_1 x_1 + \theta_2 x_2 + \theta_3$ **linear combination** of information

A neural network in practice

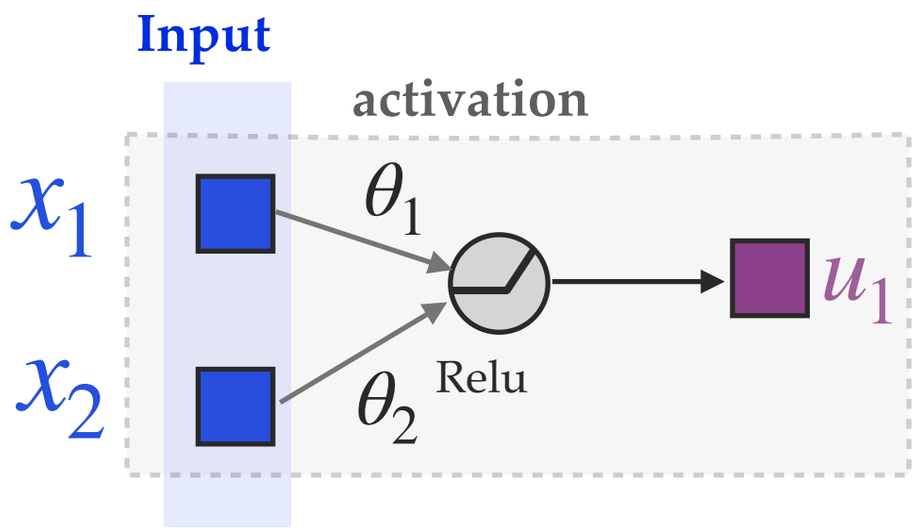
Origin: biological metaphor



Analogy:

neurons (input)

neuron (output)

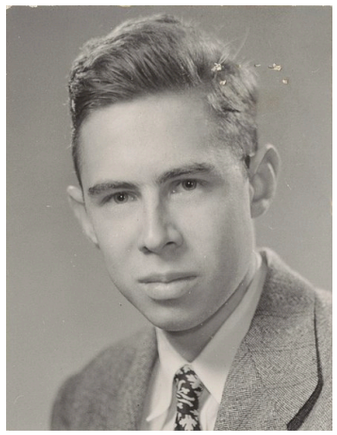


$$u_1 = \max(\theta_1 x_1 + \theta_2 x_2 + \theta_3, 0)$$

$\theta_1 x_1 + \theta_2 x_2 + \theta_3$ **linear combination** of information

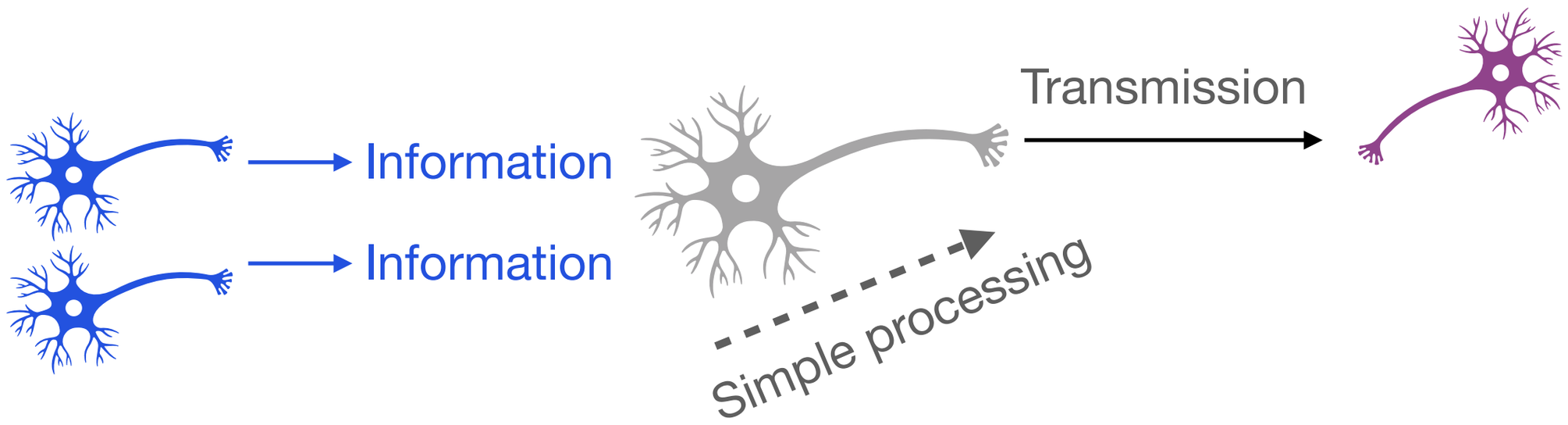
$\max(.,0)$ Thresholding

A neural network in practice



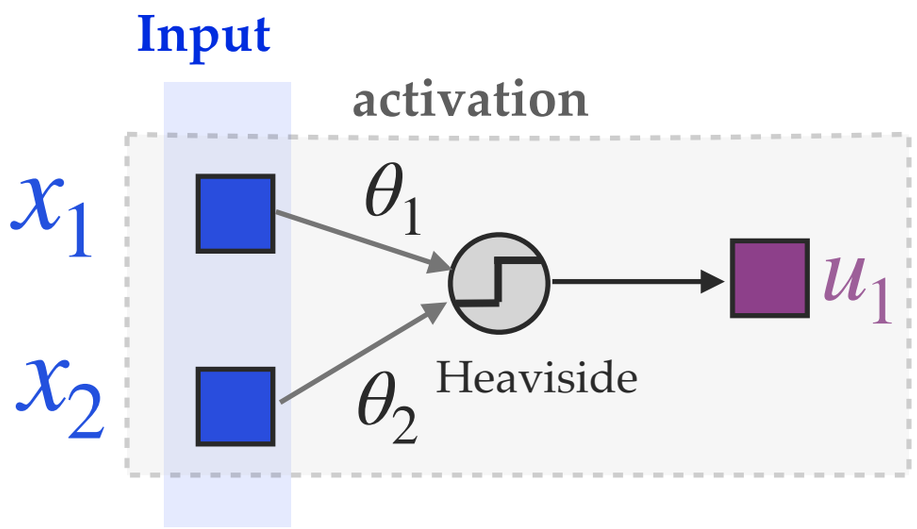
[Rosenblatt, 1957]

Origin: biological metaphor



Analogy:

- neurons (input)
- neuron



$$u_1 = \max(\theta_1 x_1 + \theta_2 x_2 + \theta_3, 0)$$

$\theta_1 x_1 + \theta_2 x_2 + \theta_3$ — **linear combination** of information

$0 \ if \ x < 0$
$1 \ if \ x > 0$ — Thresholding

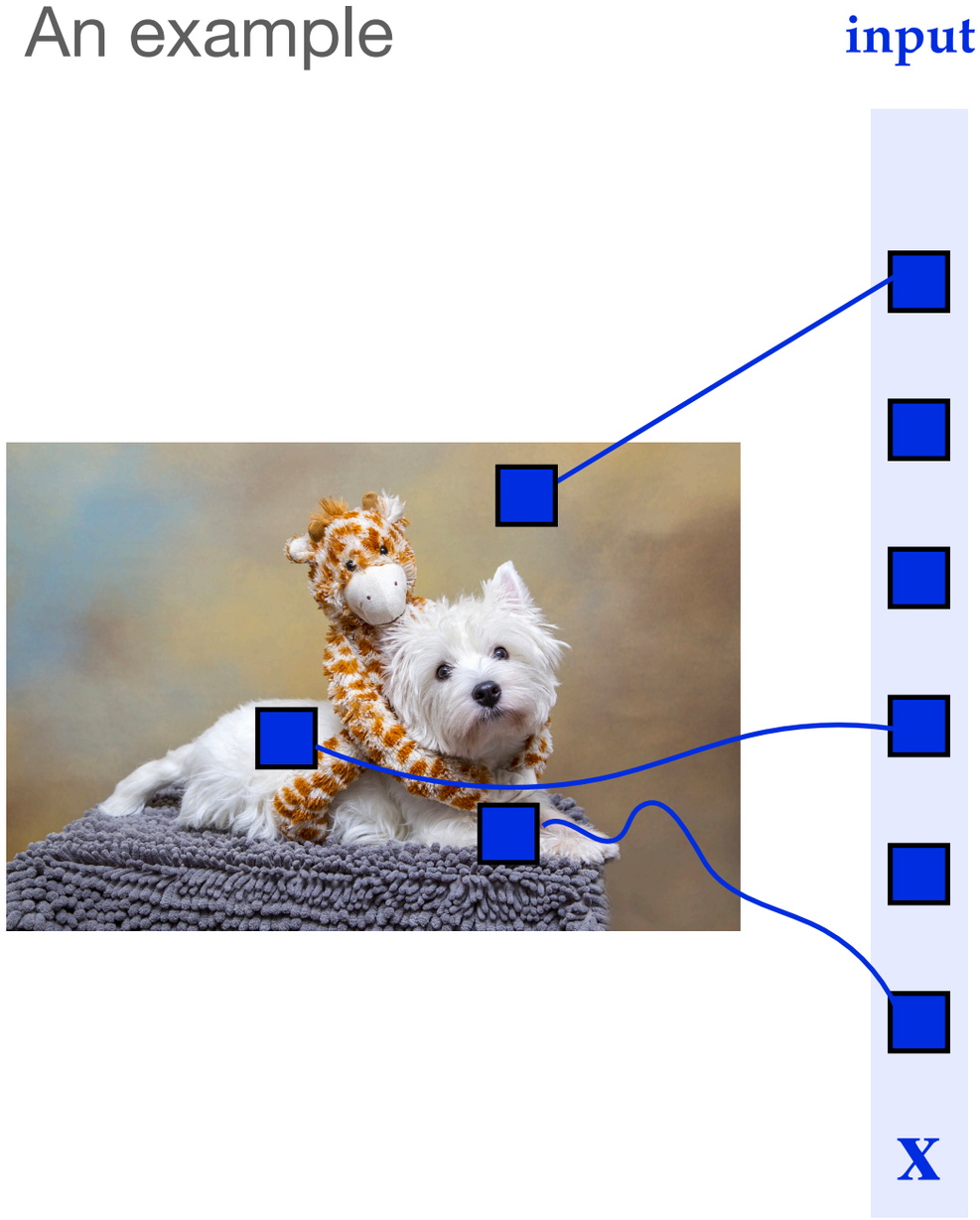
A neural network in practice

- An example



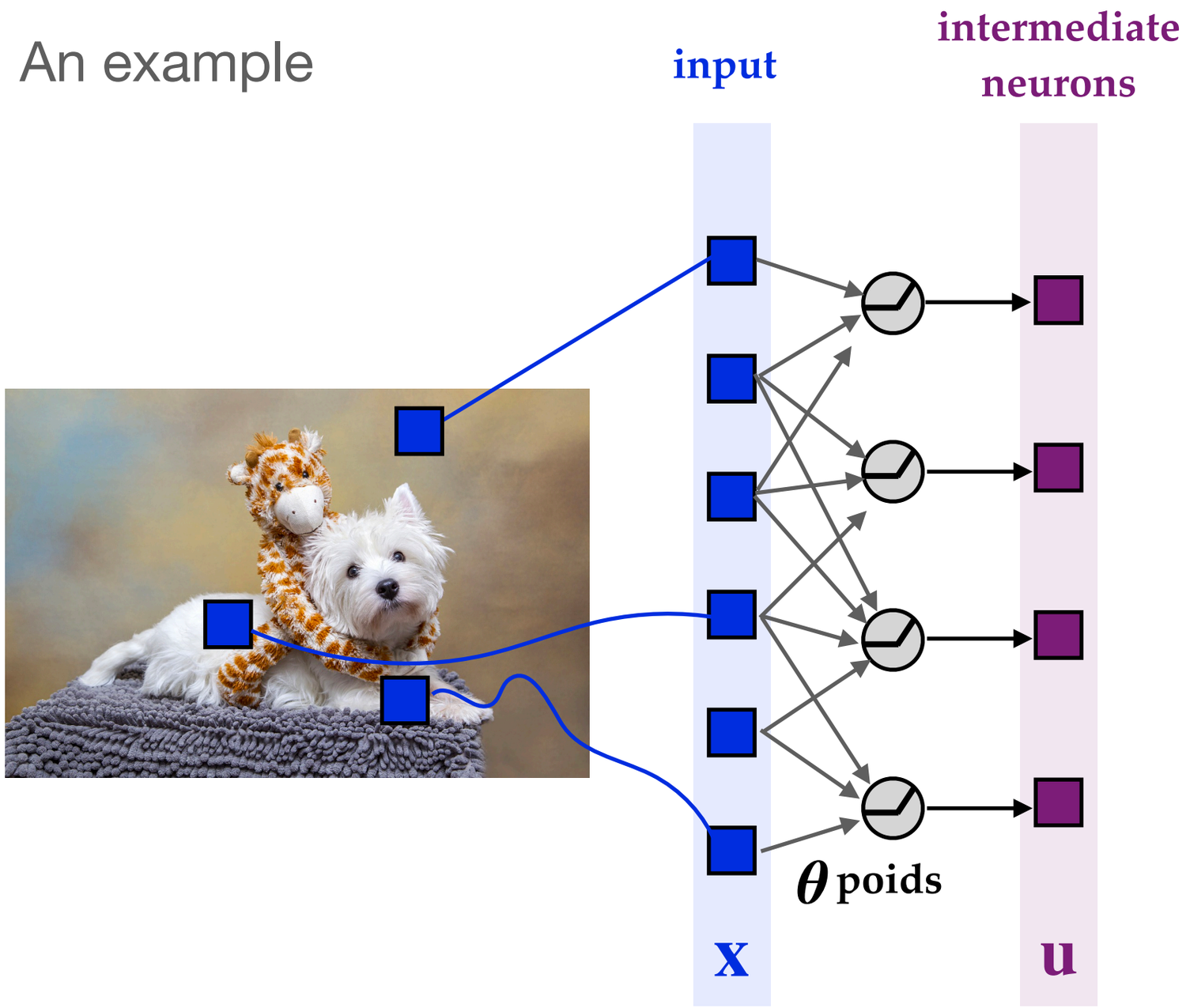
A neural network in practice

- An example



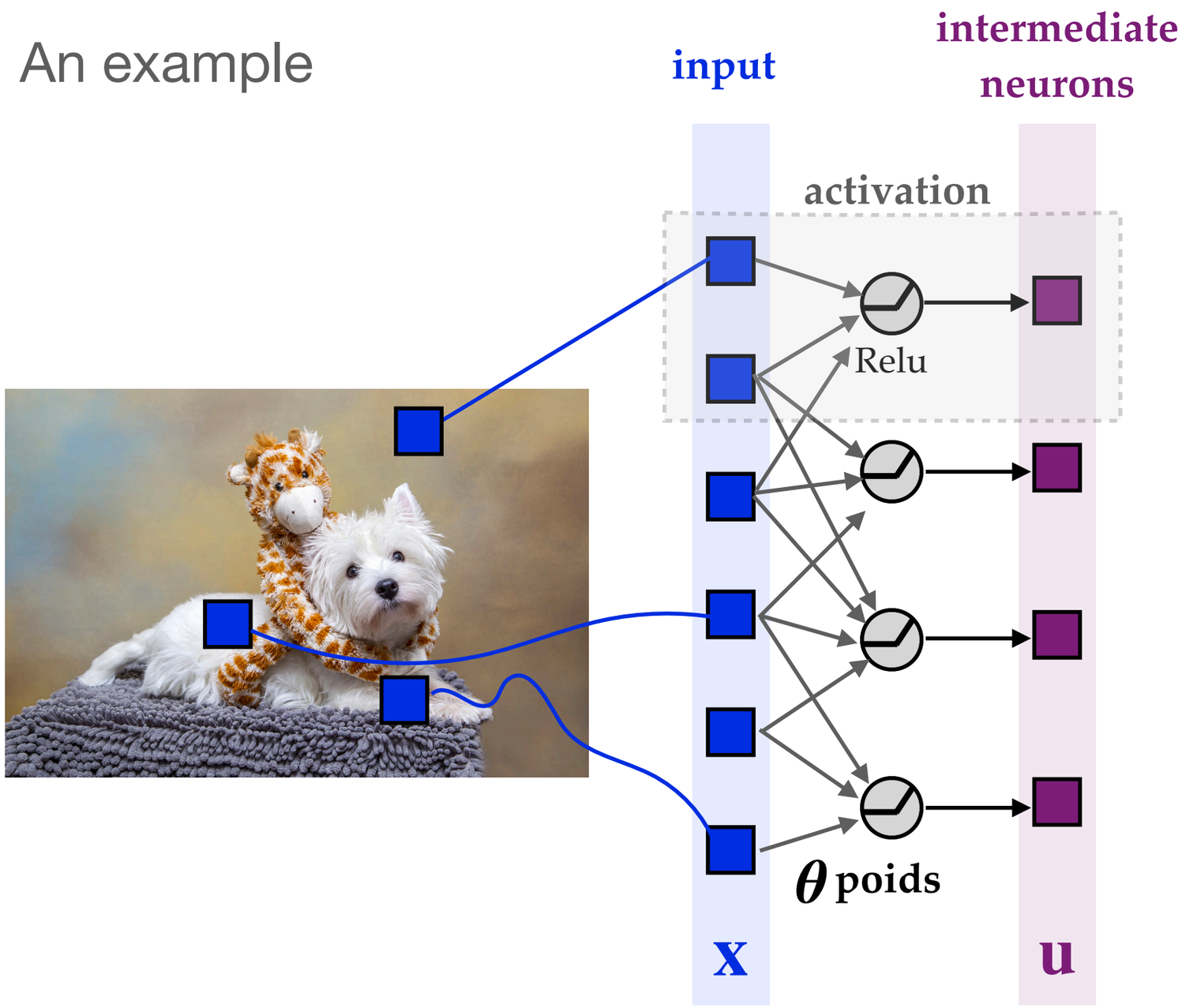
A neural network in practice

- An example



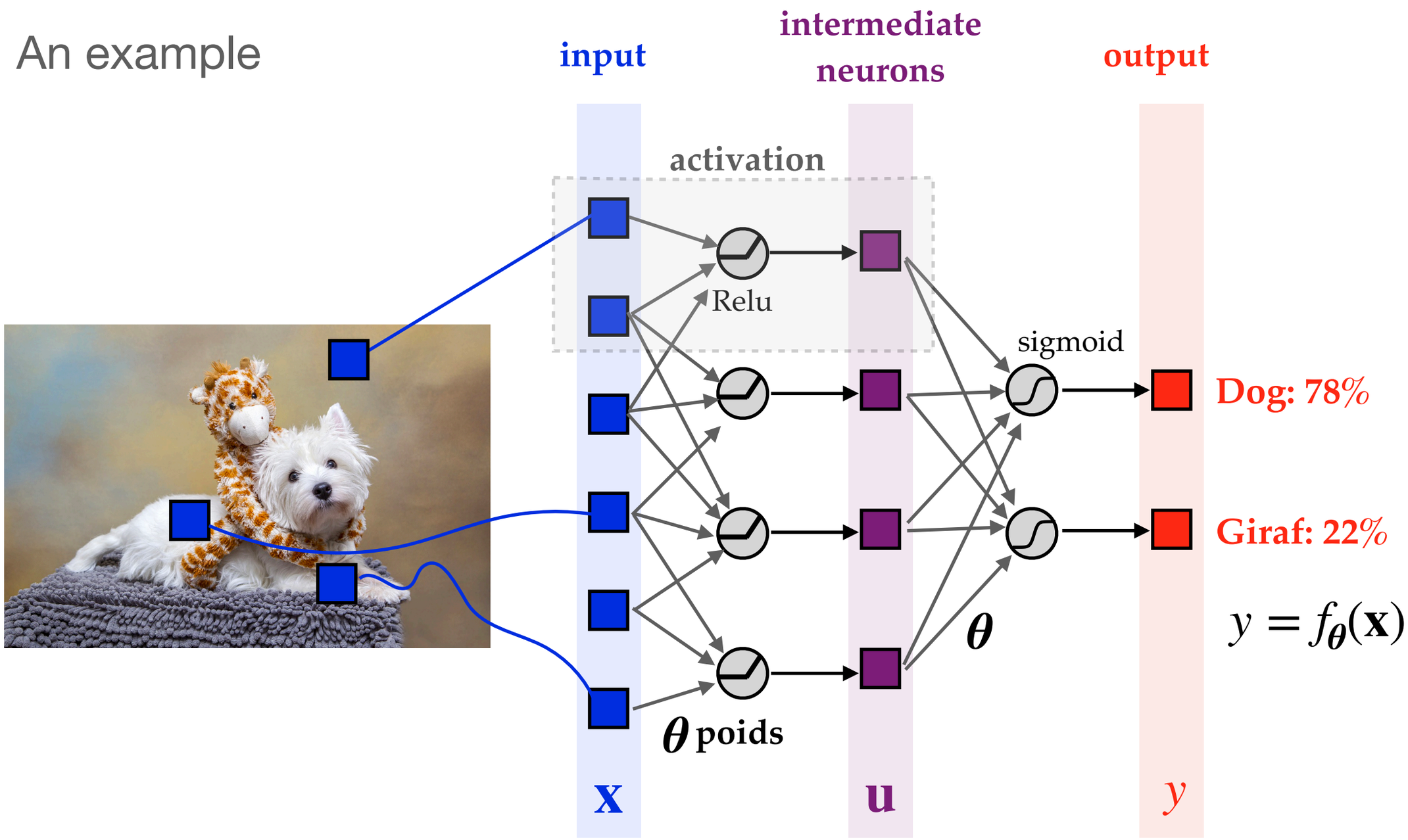
A neural network in practice

- An example



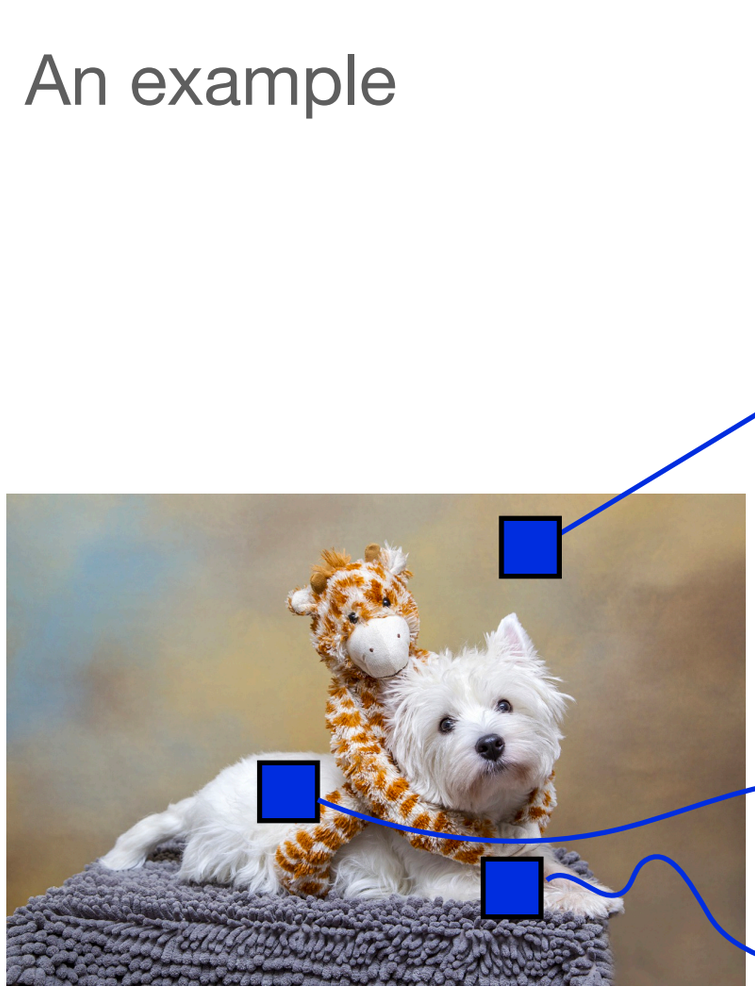
A neural network in practice

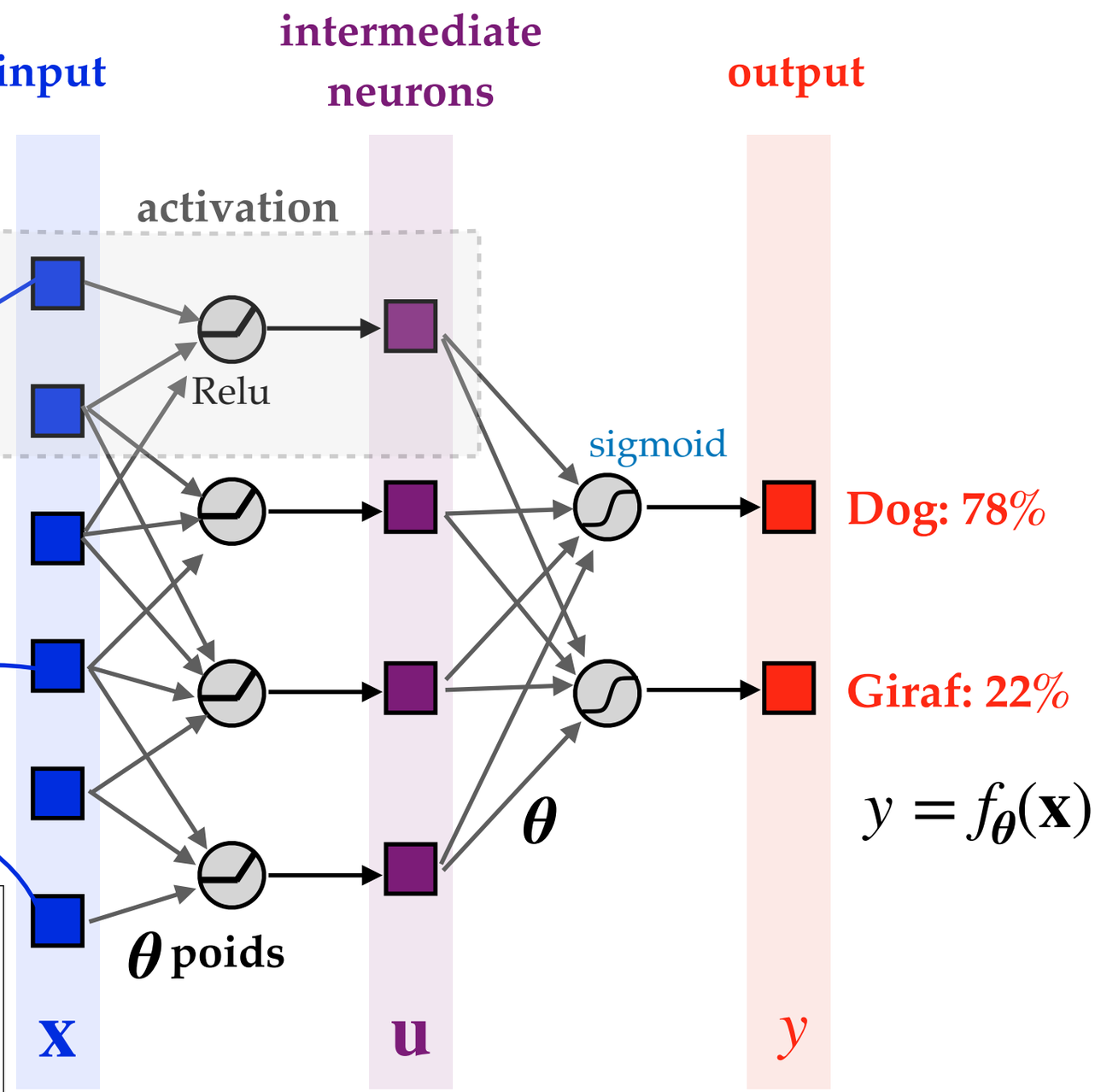
- An example



A neural network in practice

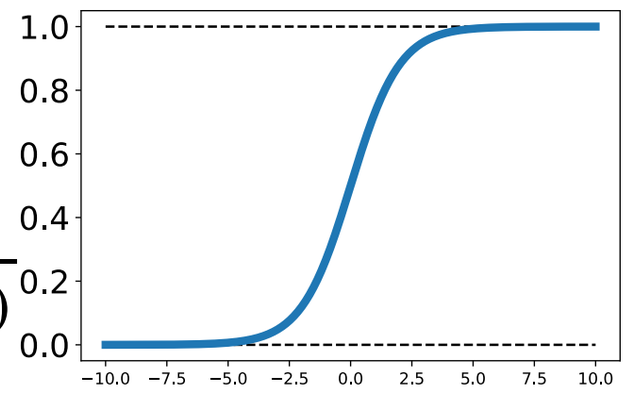
- An example





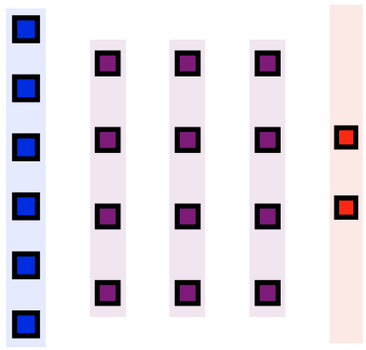
Sigmoid:

$$\sigma[u] = \frac{1}{1+\exp(-u)}$$



A neural network in practice

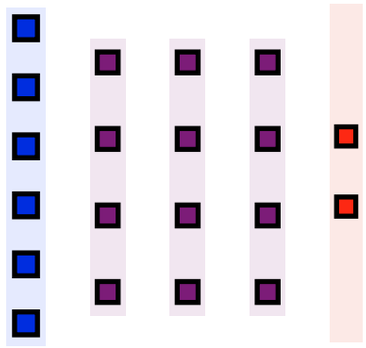
Architecture of a neural network



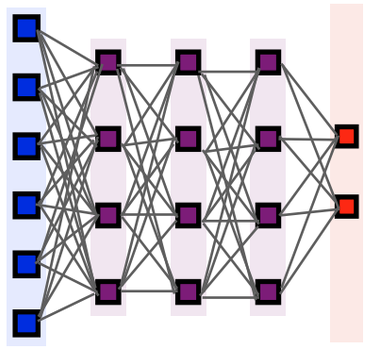
multi-couches

A neural network in practice

- Architecture of a neural network

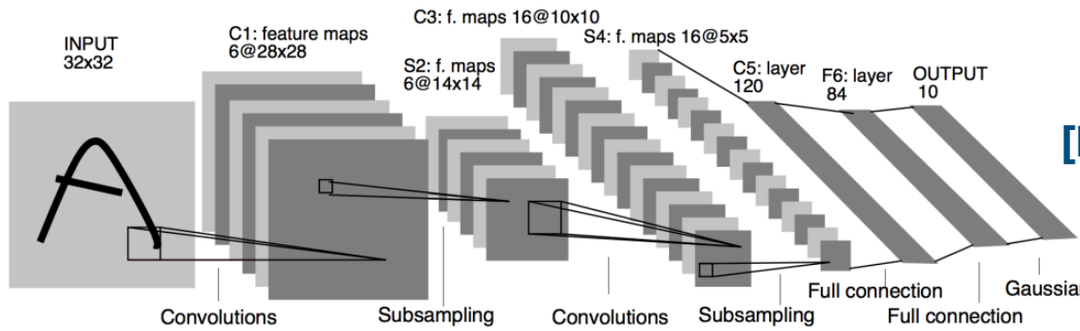


multi-couches



fully-connected

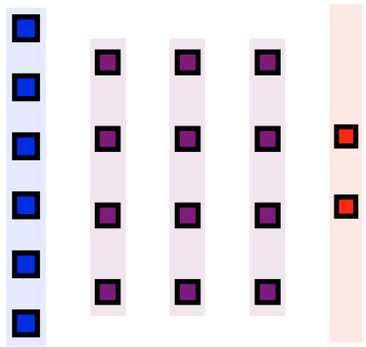
convolutional



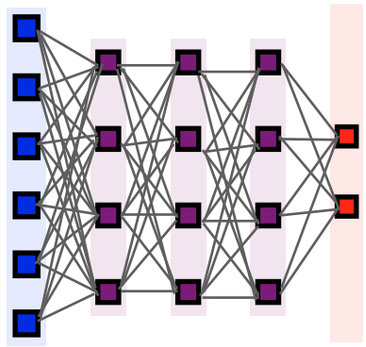
[LeCun, 1989]

A neural network in practice

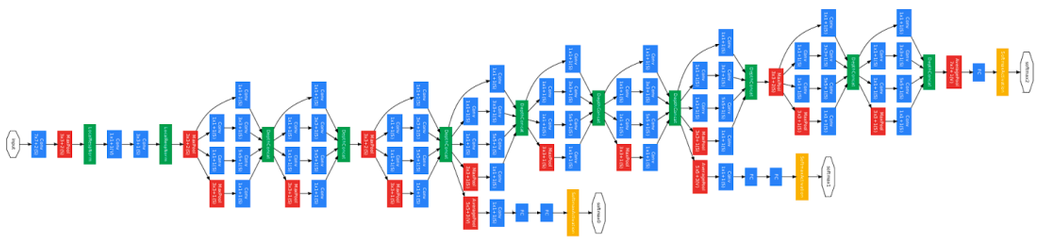
Architecture of a neural network



multi-couches

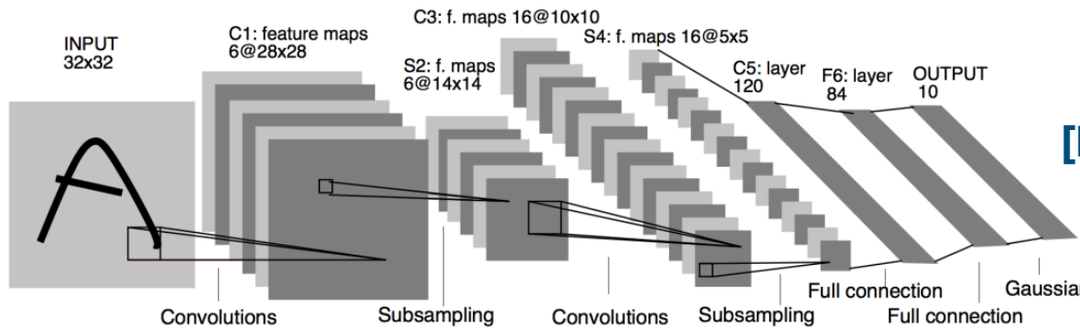


fully-connected



deep-learning

convolutional

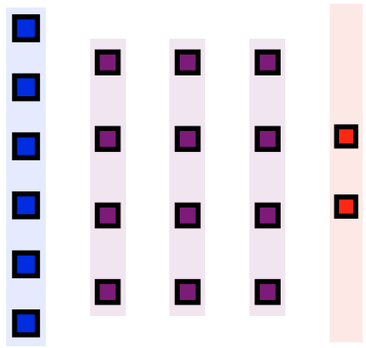


[LeCun, 1989]

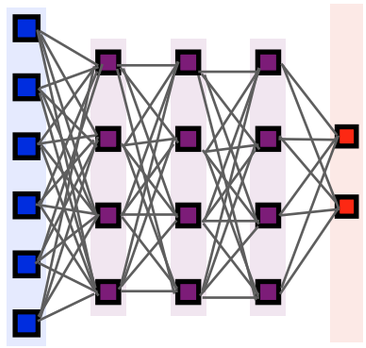
also: generative, recurrent, transformers, attention networks…

A neural network in practice

- Architecture of a neural network

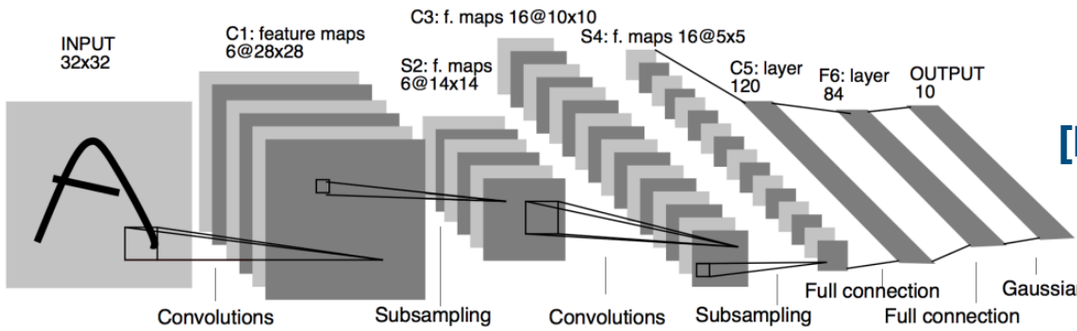


multi-couches

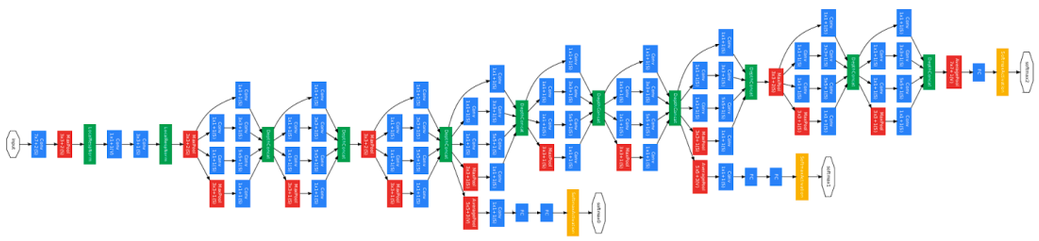


fully-connected

convolutional



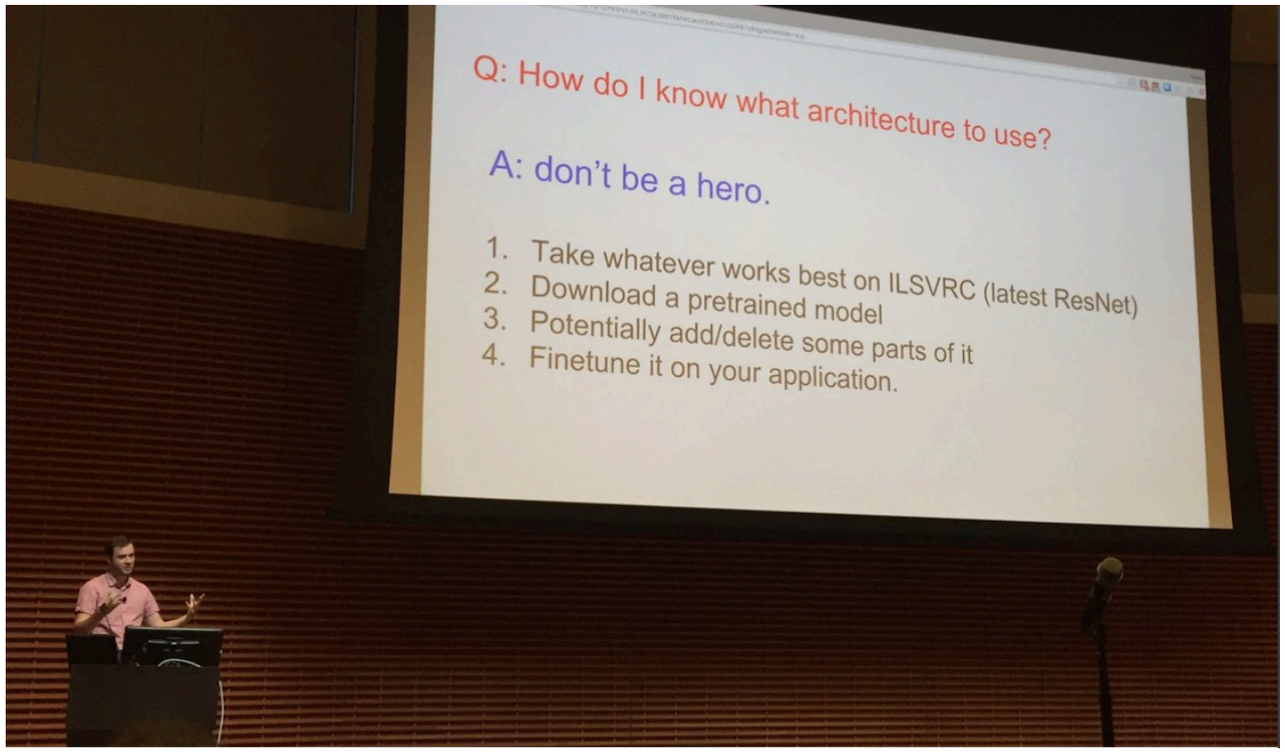
[LeCun, 1989]



deep-learning

also: generative, recurrent, transformers, attention networks…

- Don’t be a hero

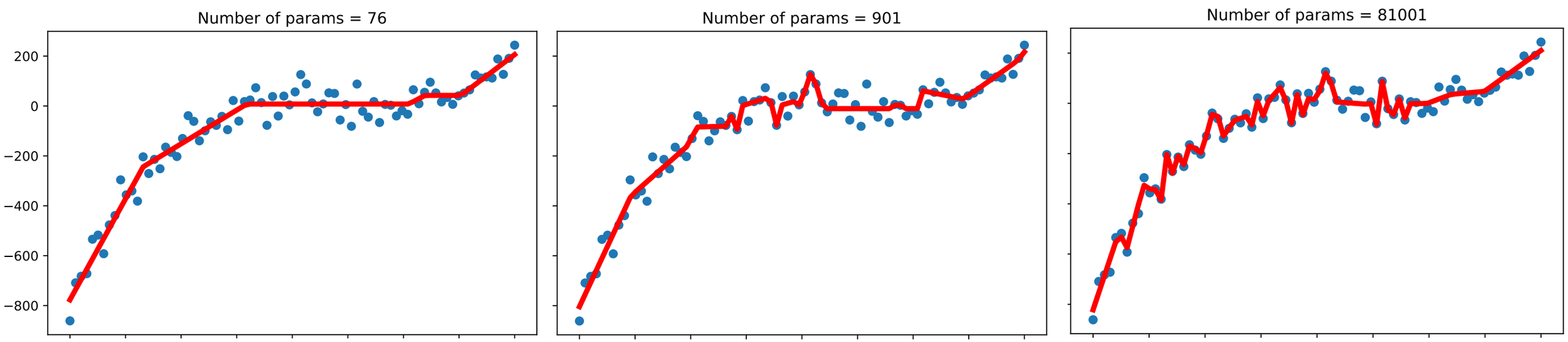


A neural network in practice

- **Why using a neural network?**
  - Very expressive model
  - Can reach outstanding performance

A neural network in practice

- **Why using a neural network?**
  - Very expressive model
  - Can reach outstanding performance
- **Why not using a neural network?**
  - Very expressive model (overfitting)



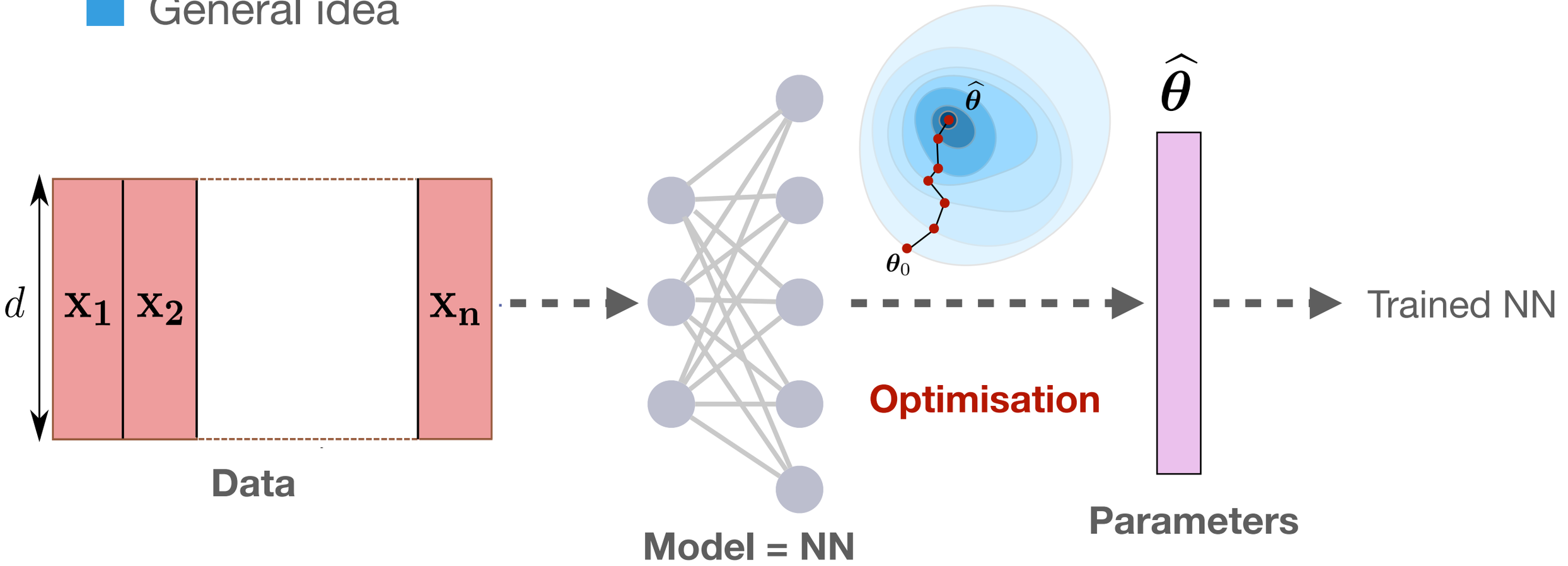
  - Choice of the architecture, requires a lot of data, often hard to train (non convex => local minima)

Introduction to Neural Networks

- **Context and definition**
  - The success of "deep learning"
  - A neural network in practice
- **Training a neural network**
  - Reminders on gradient descent
  - Automatic Differentiation
- **The framework** PyTorch
  - Classification and regression in practice
  - A quick demonstration on generative models

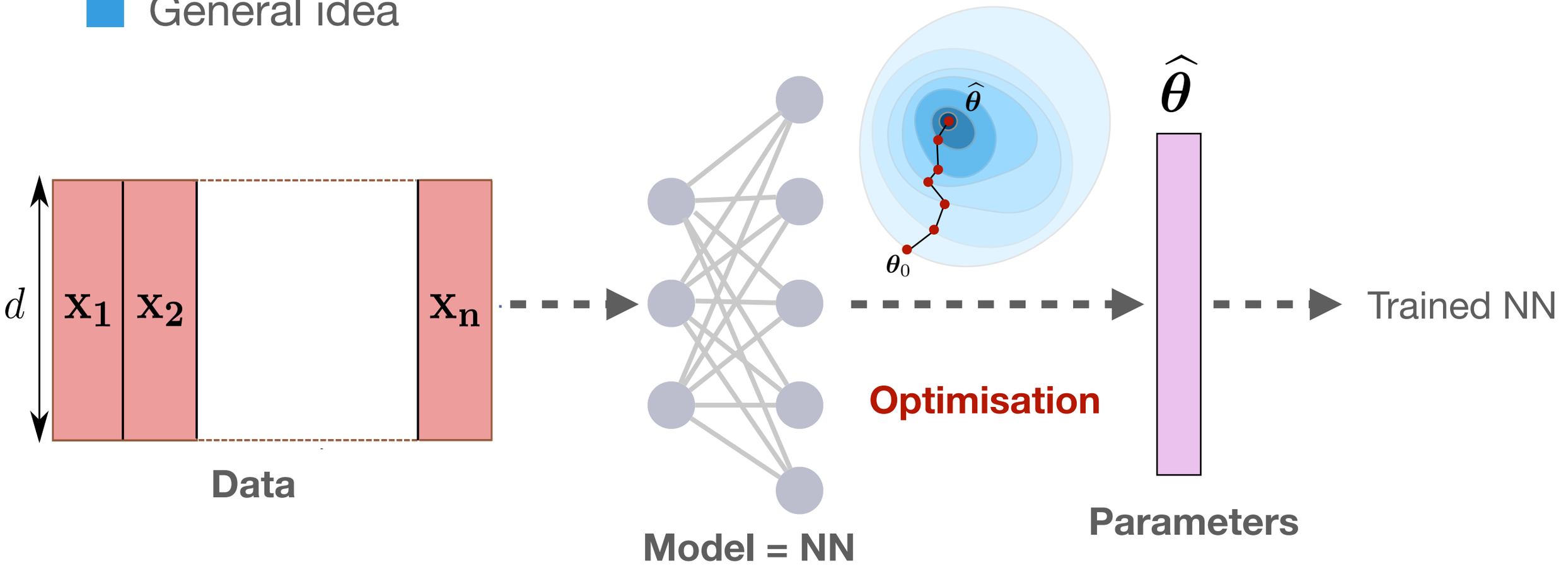
Training a neural network

- General idea



Training a neural network

- General idea



- **Optimisation**: gradient descent (and stochastic variants)

Training a neural network

- To train a neural network, we need:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- Performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The objective is to find the parameters minimizing

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

- = minimzing the loss on the training data

Training a neural network

- To train a neural network, we need:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- A performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The goal is to identify the parameters minimizing:

Regression : $\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i))^2$

$y_i \in \mathbb{R}$

$f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in \mathbb{R}$

Training a network

- Pour Training a network de neurone nous avons besoin:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- A performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The goal is to identify the parameters minimizing:

Binary classification:

$y_i \in \{-1,1\}$

$f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in \mathbb{R}$

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i f_{\boldsymbol{\theta}}(\mathbf{x}_i)\}$$

Hinge-loss

Training a neural network

- To train a neural network, we need:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- A performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The goal is to identify the parameters minimizing:

Binary classification:

$y_i \in \{0,1\}$

$f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in [0,1]$

e.g. with sigmoid/softmax

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

Logistic loss or **cross entropy**

Training a neural network

- To train a neural network, we need:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- A performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The goal is to identify the parameters minimizing:

! To minimize !

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



- = minimzing the loss on the training data

Gradient descent

Reminders on gradient descent

Goal:

$$\min_{\theta} f(\theta)$$

Reminders on gradient descent

Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0

2- Iterations: $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

Dérivée de f

Reminders on gradient descent

Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0 **2- Iterations:** $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

- Analogy : « it's all downhill »



θ_0

Reminders on gradient descent

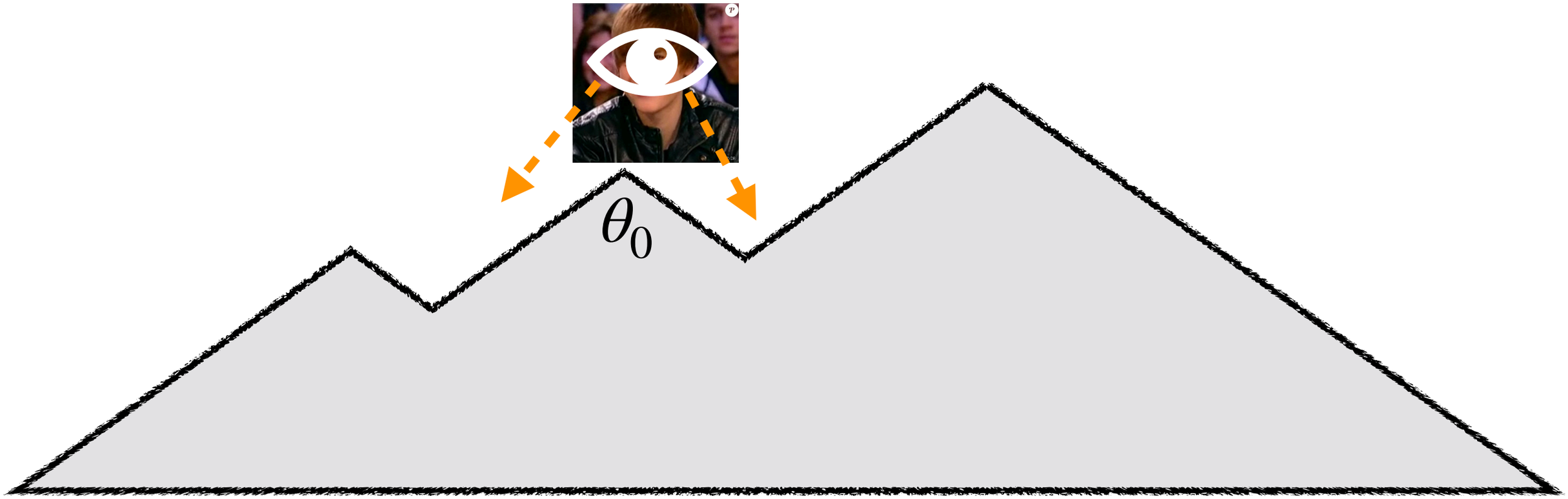
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0 **2- Iterations:** $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

- Analogy : « it's all downhill »



Reminders on gradient descent

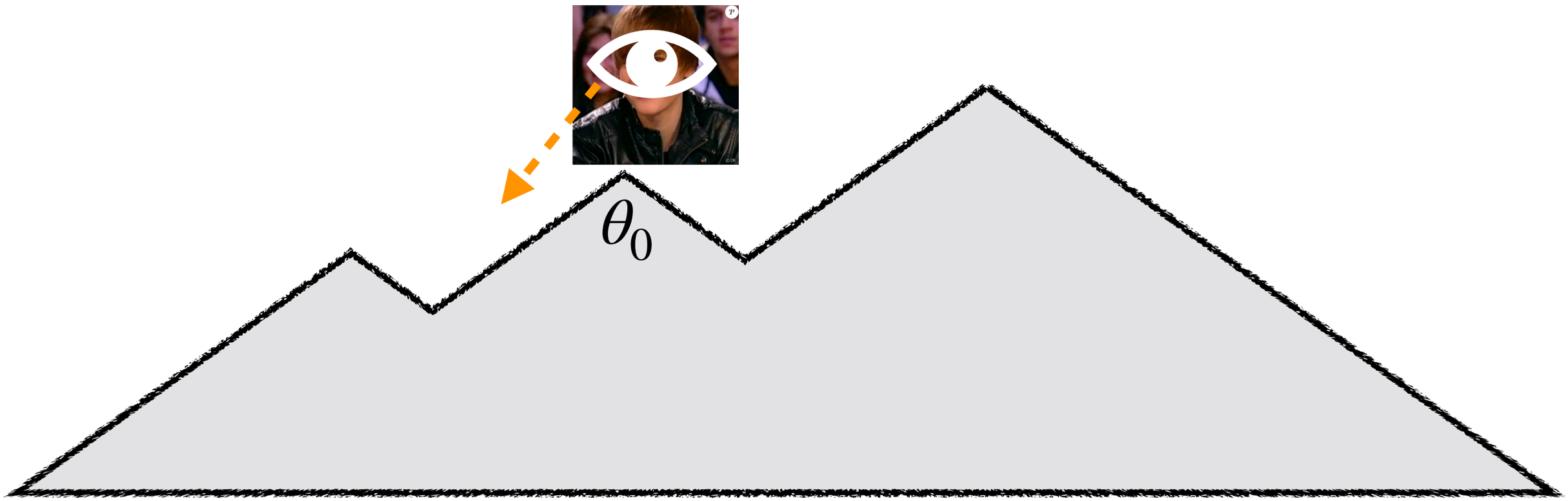
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0 **2- Iterations:** $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

- Analogy : « it's all downhill »



Reminders on gradient descent

Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0 **2- Iterations:** $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

- Analogy : « it's all downhill »



Reminders on gradient descent

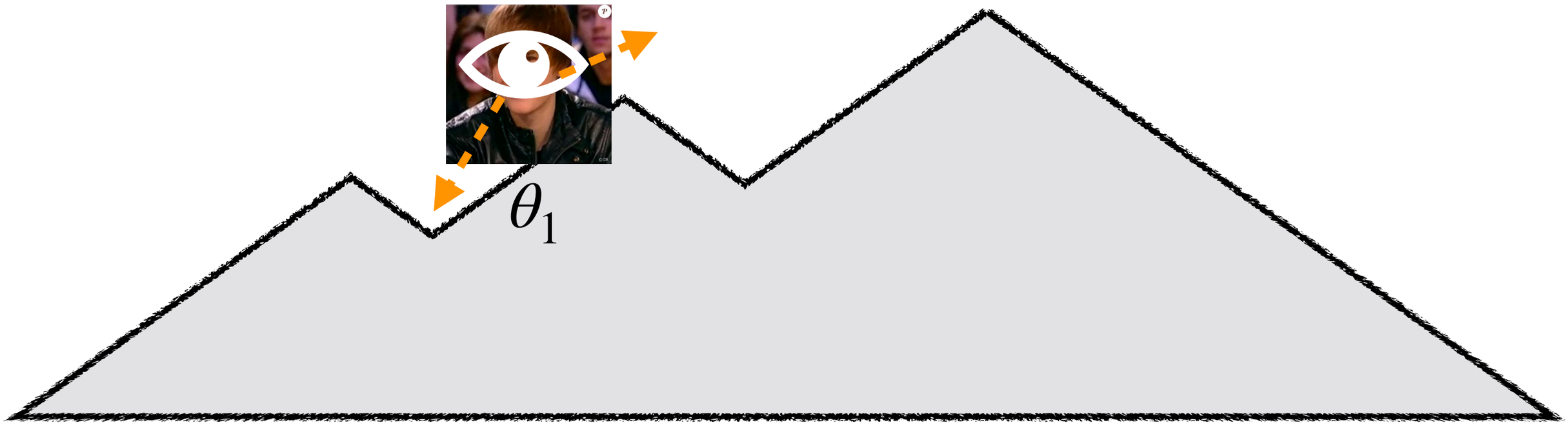
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0 **2- Iterations:** $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

- Analogy : « it's all downhill »



Reminders on gradient descent

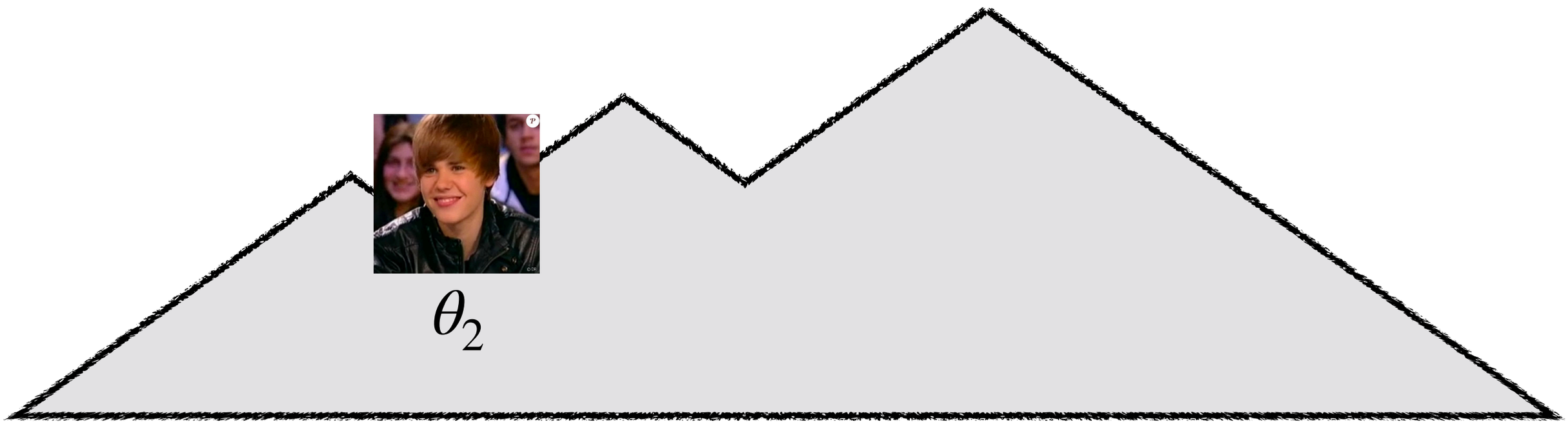
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0 **2- Iterations:** $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$

- Analogy : « it's all downhill »



Reminders on gradient descent

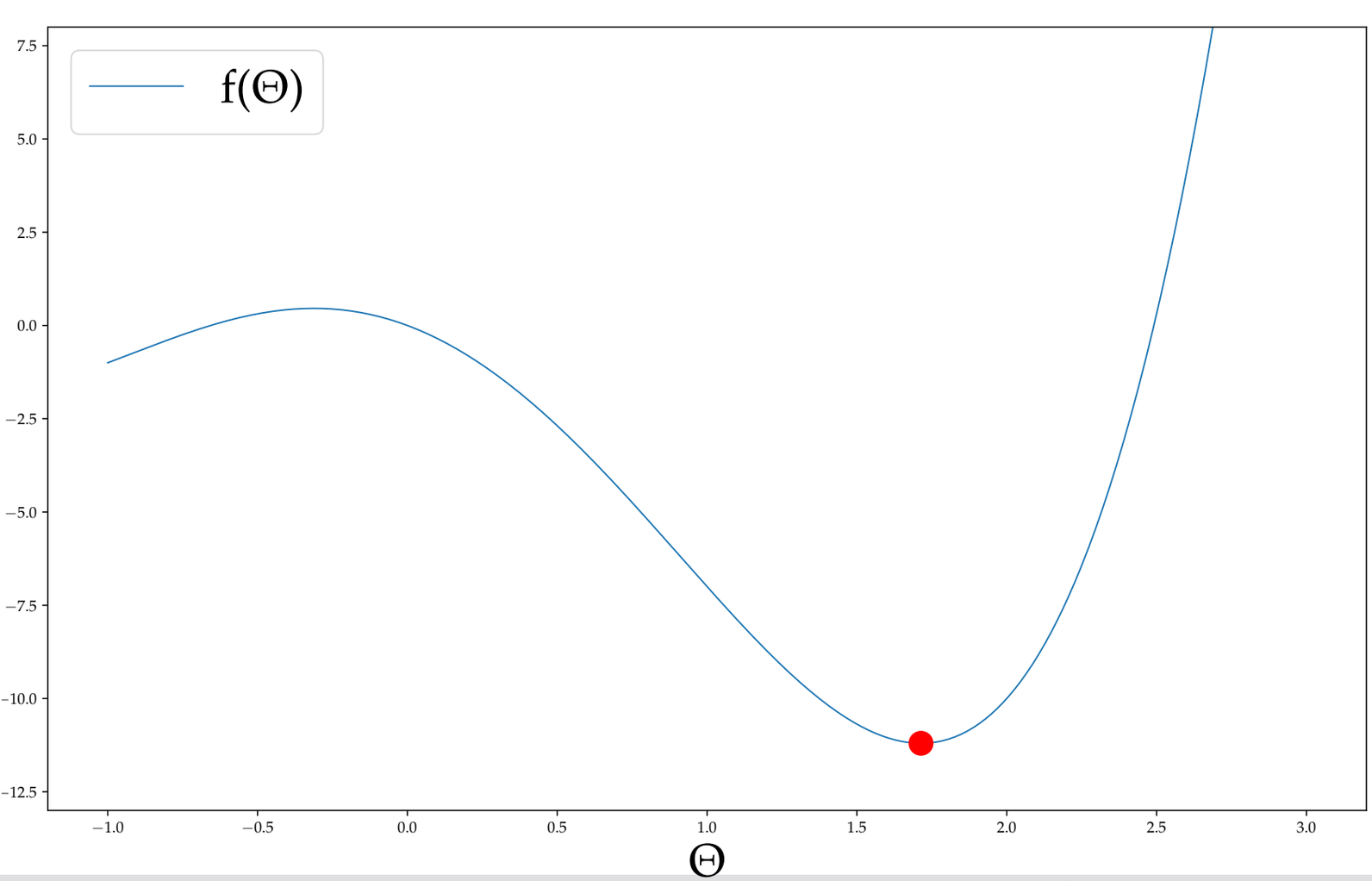
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process:

1- Starting point: θ_0

2- Iterations: $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$



- On a function:

η **descent « step » = learning rate**

$\frac{df}{d\theta}(\theta_{k-1})$ **« direction » of the descent**

Reminders on gradient descent

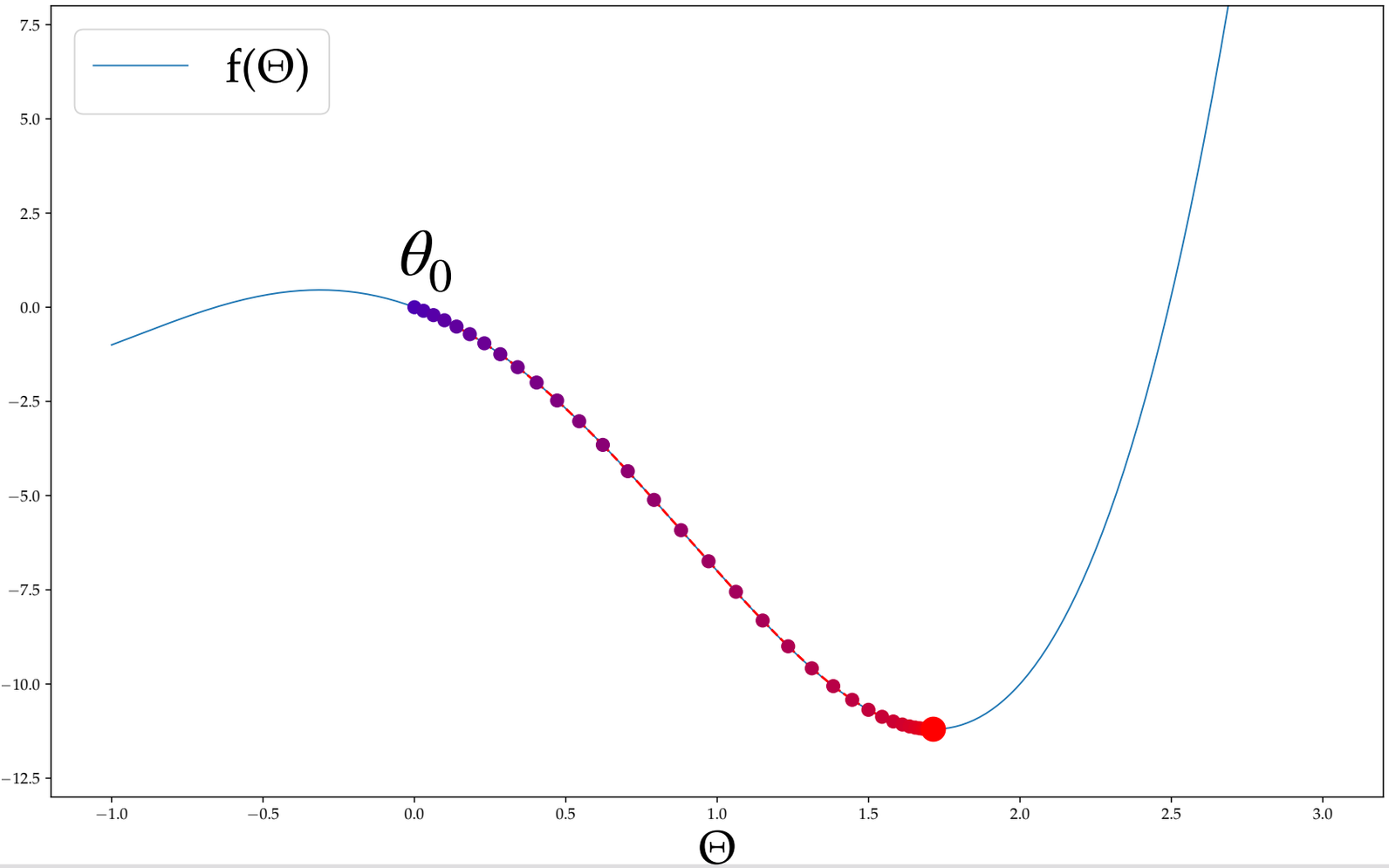
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0

2- Iterations: $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$



- On a function:

η **descent « step » = learning rate**

$\frac{df}{d\theta}(\theta_{k-1})$ **« direction » of the descent**

Reminders on gradient descent

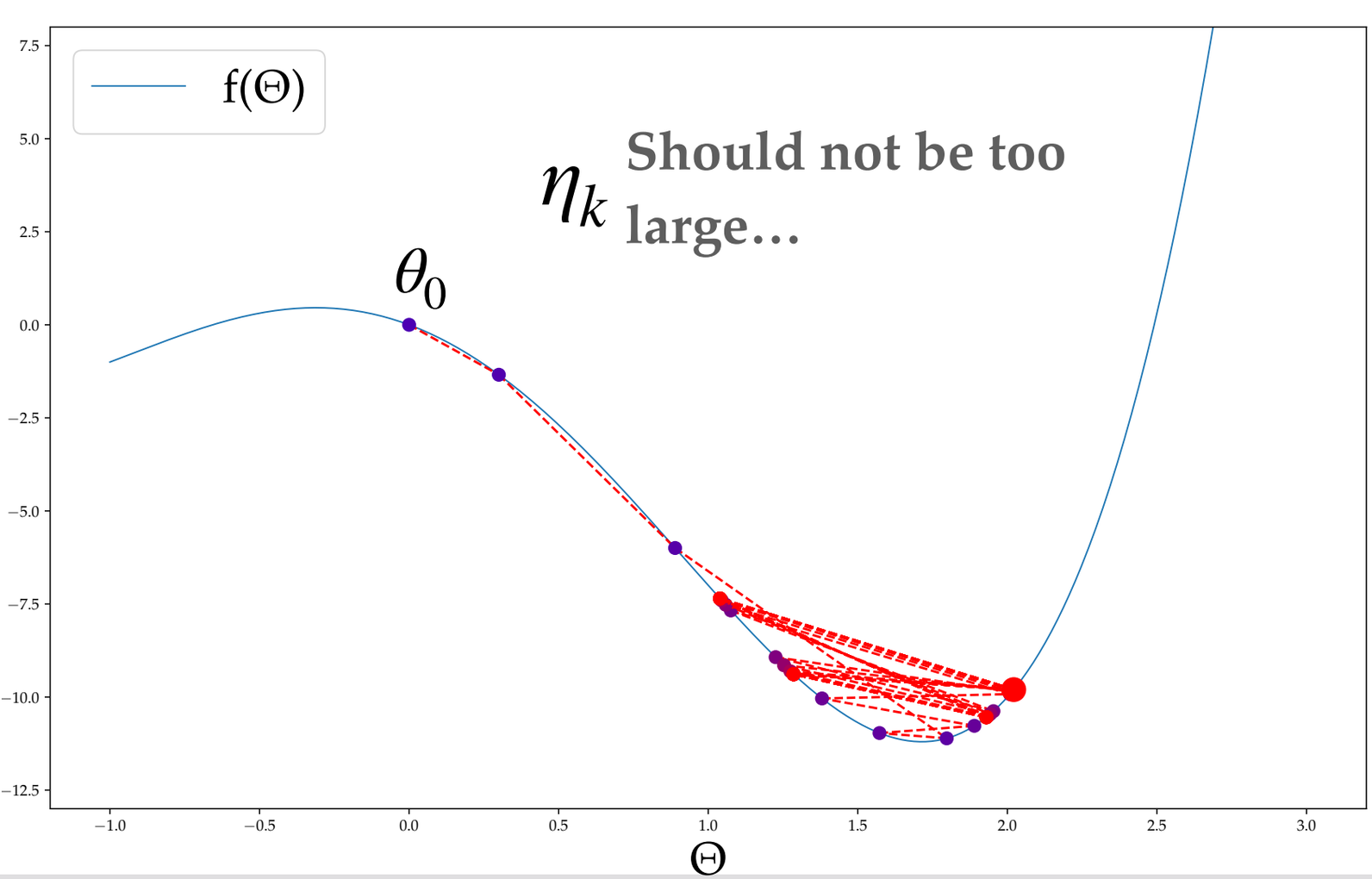
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0

2- Iterations: $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$



- On a function:

η **descent « step » = learning rate**

$\frac{df}{d\theta}(\theta_{k-1})$ **« direction » of the descent**

Reminders on gradient descent

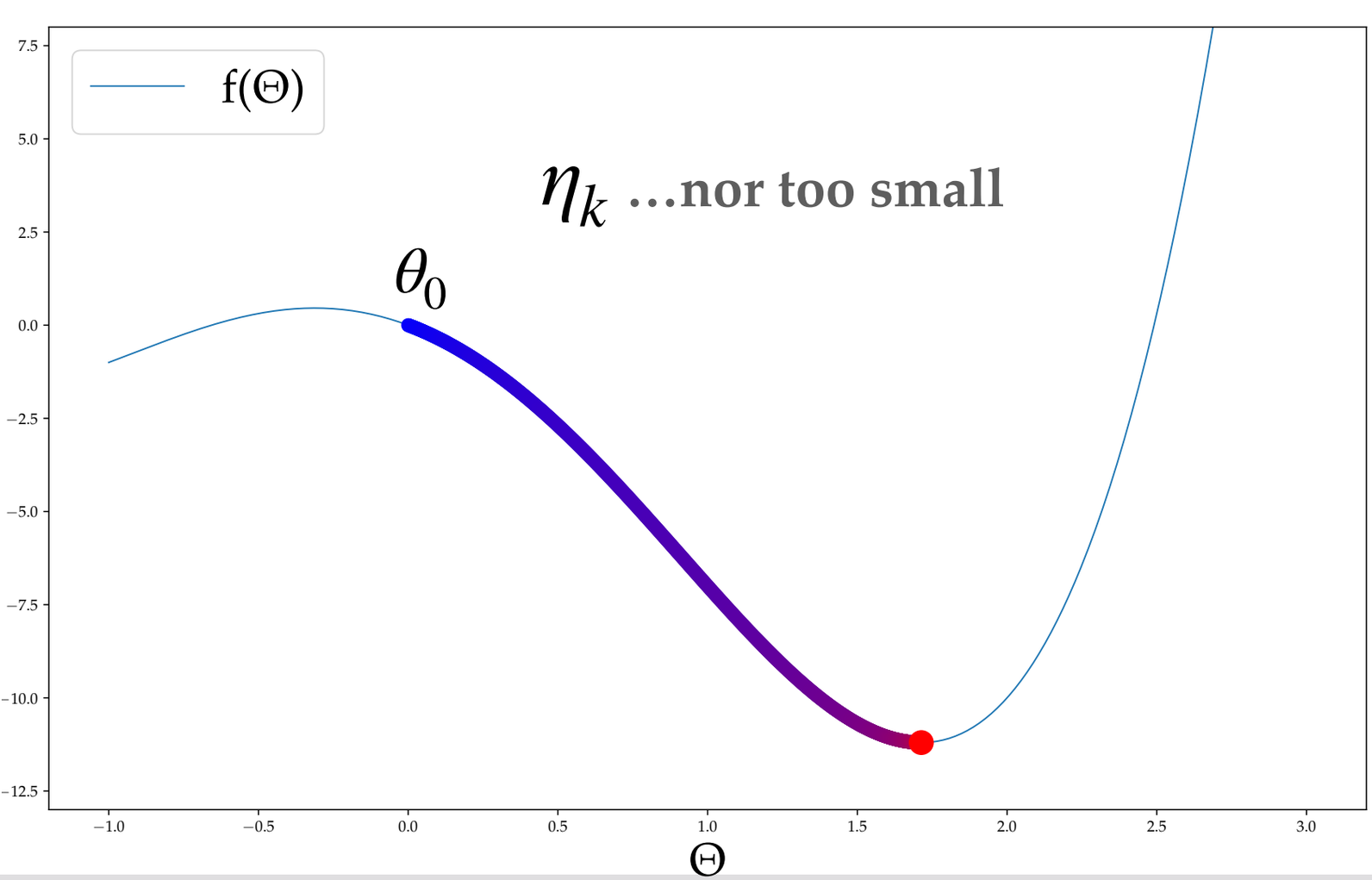
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0

2- Iterations: $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$



- On a function:

η **descent « step » = learning rate**

$\frac{df}{d\theta}(\theta_{k-1})$ **« direction » of the descent**

Reminders on gradient descent

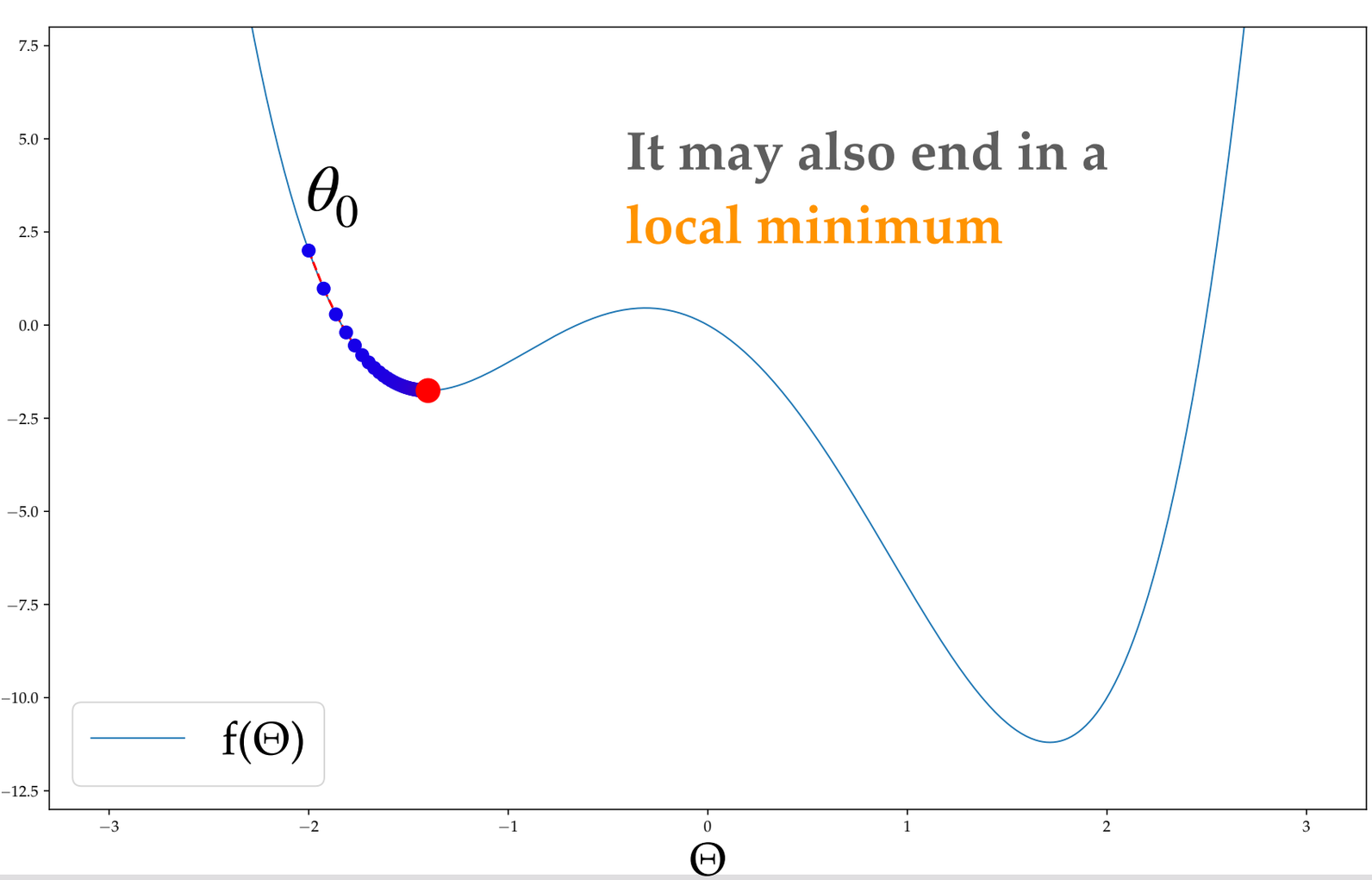
Objectif:

$$\min_{\theta} f(\theta)$$

- Gradient descent is an iterative process

1- Starting point: θ_0

2- Iterations: $\theta_k \leftarrow \theta_{k-1} - \eta \cdot \frac{df}{d\theta}(\theta_{k-1})$



- On a function:

η **descent « step » = learning rate**

$\frac{df}{d\theta}(\theta_{k-1})$ **« direction » of the descent**

Reminders on gradient descent

Objectif:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$$

- When we have several parameters

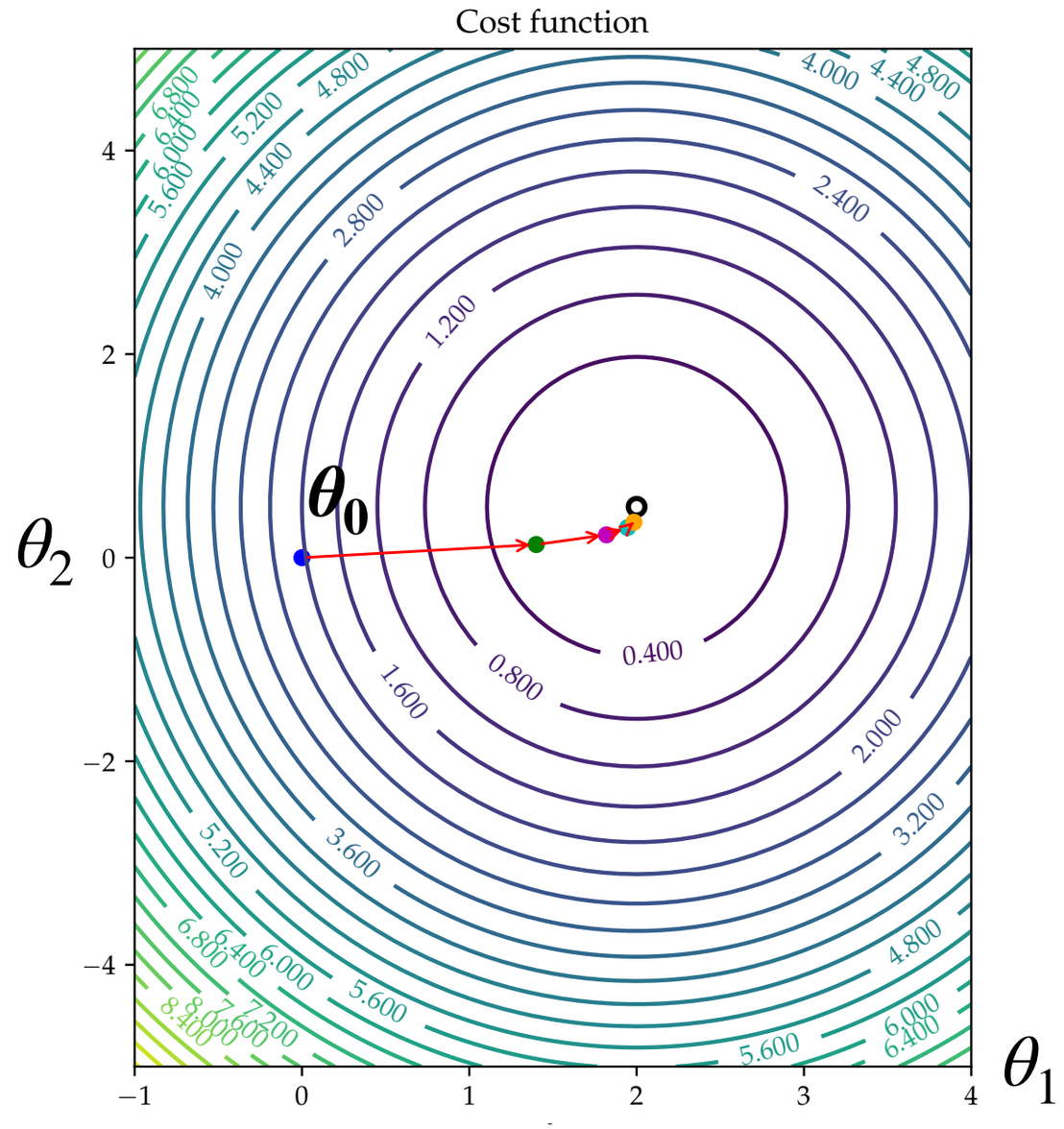
$$\boldsymbol{\theta} = (\theta_1, \cdots, \theta_d)$$

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{k-1})$$

Gradient of f

- Generalizes the derivative in higher dimension

- Same idea as before



Training a neural network

- To train a neural network, we need:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- A performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The goal is to identify the parameters minimizing:

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

- Gradient descent:

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \nabla_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i))$$

Training a neural network

- To train a neural network, we need:

1- A network architecture: $f_{\boldsymbol{\theta}}$

2- Sufficient training data: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$

3- A performance measure = loss function $L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$

- The goal is to identify the parameters minimizing:

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

- Gradient descent:

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i))$$

Training a neural network

- In practice, Gradient descent is often very slow

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i))$$

- « batch » gradient descent is to be preferred:



Etape 1- GD sur le batch 1:

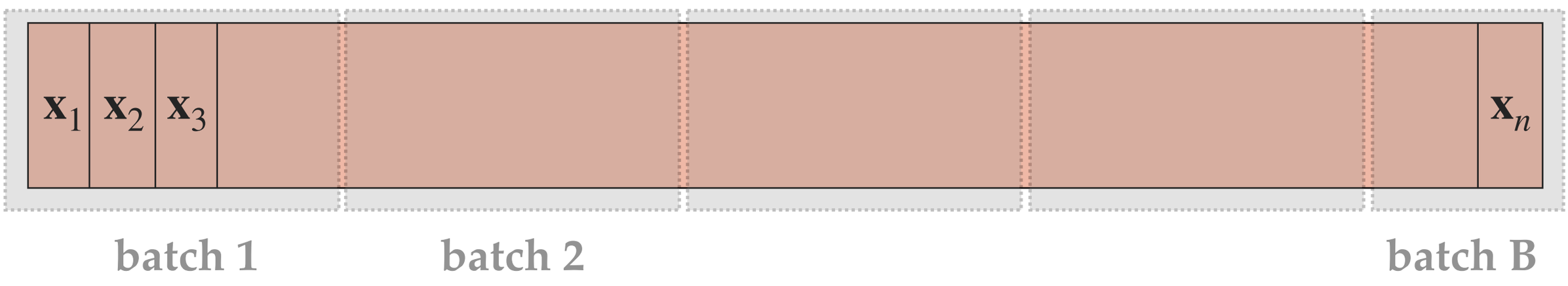
$$\boldsymbol{\theta}_1 \leftarrow \boldsymbol{\theta}_0 - \eta \cdot \frac{1}{|b_1|} \sum_{i \in b_1} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_0}(\mathbf{x}_i))$$

Training a neural network

- In practice, Gradient descent is often very slow

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i))$$

- « batch » gradient descent is to be preferred:



Step 2: GD on batch 2:

$$\boldsymbol{\theta}_2 \leftarrow \boldsymbol{\theta}_1 - \eta \cdot \frac{1}{|b_2|} \sum_{i \in b_2} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_1}(\mathbf{x}_i))$$

Training a neural network

- In practice, Gradient descent is often very slow

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i))$$

- « batch » gradient descent is to be preferred:

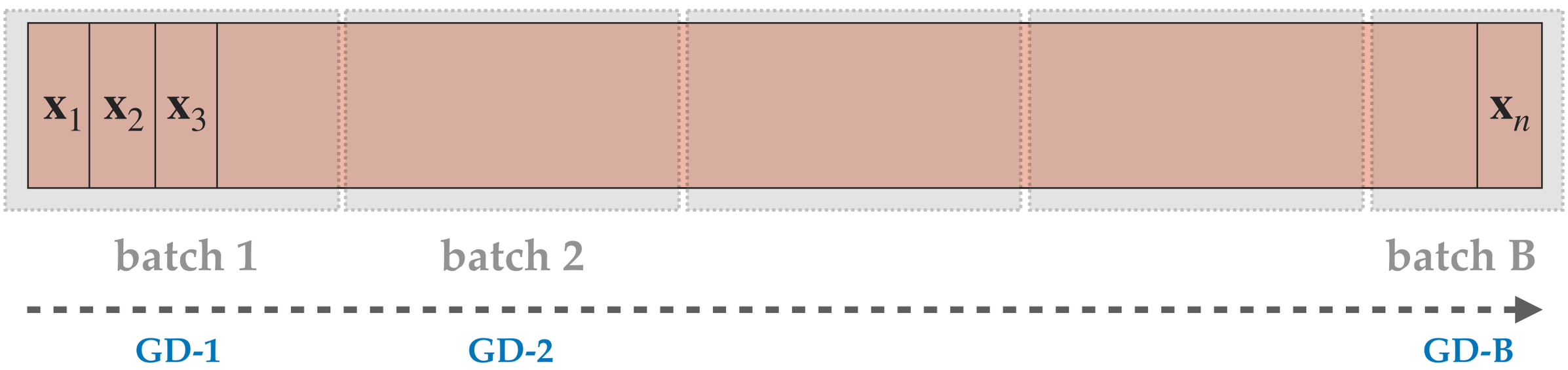


Training a neural network

- In practice, Gradient descent is often very slow

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}_{k-1}}(\mathbf{x}_i))$$

- « batch » gradient descent is to be preferred:



- A pass on the entire dataset is called an « epoch »
- Often, several epochs are required before convergence
- Many variants exist (Adam, momentum, accelerated GD …)

« Automatic differentiation »

- In all cases, training the neural network requires:

$$\textbf{The gradient} \quad \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$$

- **Impossible** to compute it by hand or to have an explicit expression!

« Automatic differentiation »

- In all cases, training the neural network requires:

$$\textbf{The gradient} \quad \nabla_{\boldsymbol{\theta}} L(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i))$$

- **Impossible** de le calculer à la main, de trouver son expression !

- **Auto-diff:** automatic way to compute this gradient



« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

« Automatic differentiation »

General idea:

A "complex" function: $$f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

By hand:

$$\frac{d}{dx} f(x) = \exp(\exp(x) + \exp(x)^2)(\exp(x) + 2\exp(x)^2) + \cos(\exp(x) + \exp(x)^2)(\exp(x) + 2\exp(x)^2)$$



Imagine a neural network with several layers...

« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:

x

$a = \exp(x)$

$b = a^2$

$c = a + b$

$d = \exp(c)$

$e = \sin(c)$

$f = d + e$

$f(x)$

« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:

« Forward »

x

$a = \exp(x)$

$b = a^2$

$c = a + b$

$d = \exp(c)$

$e = \sin(c)$

$f = d + e$

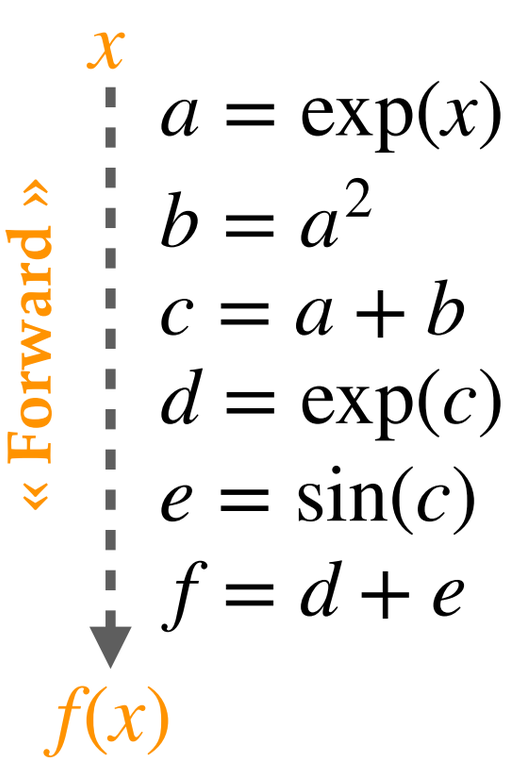
$f(x)$

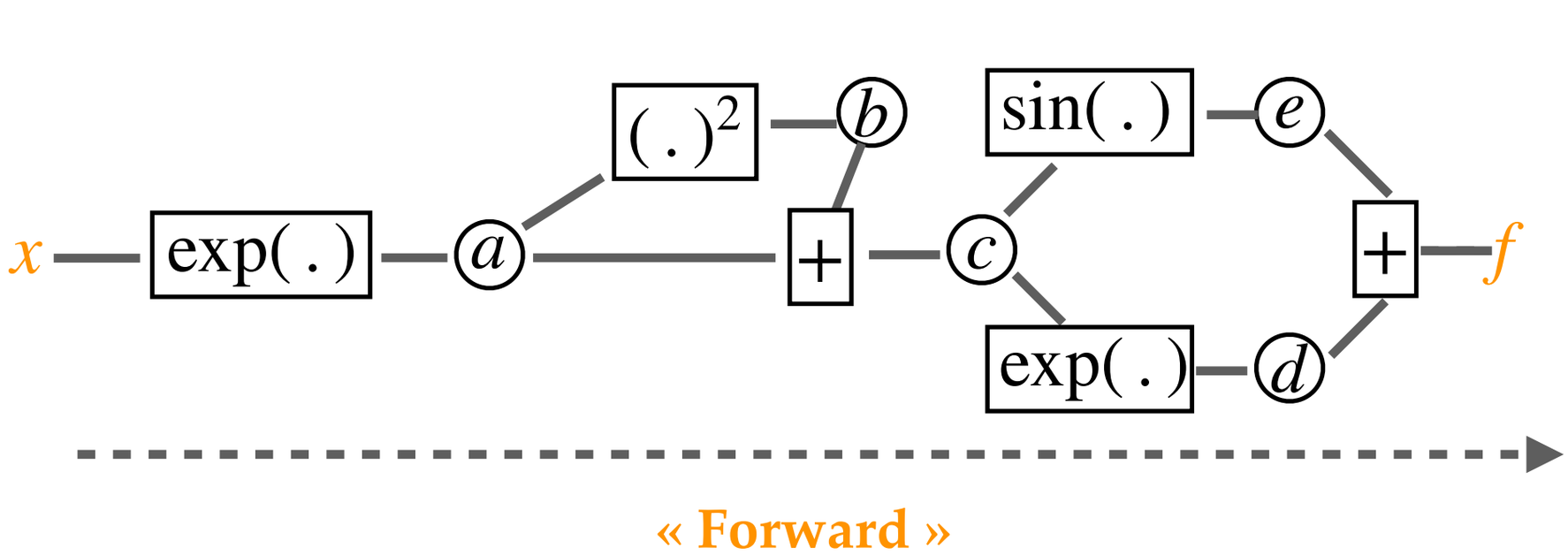
« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



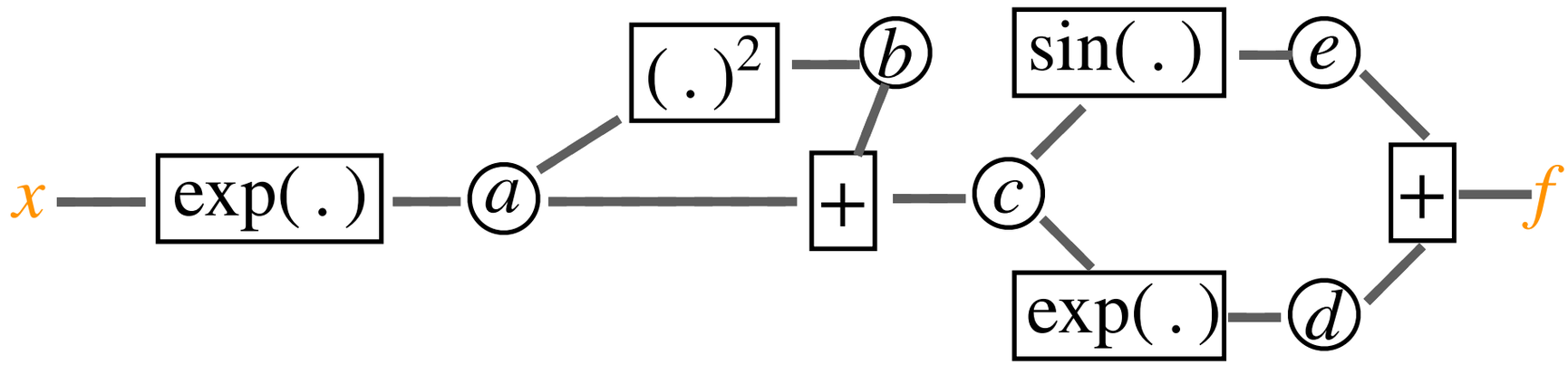


« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



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

« Backward »

« Automatic differentiation »

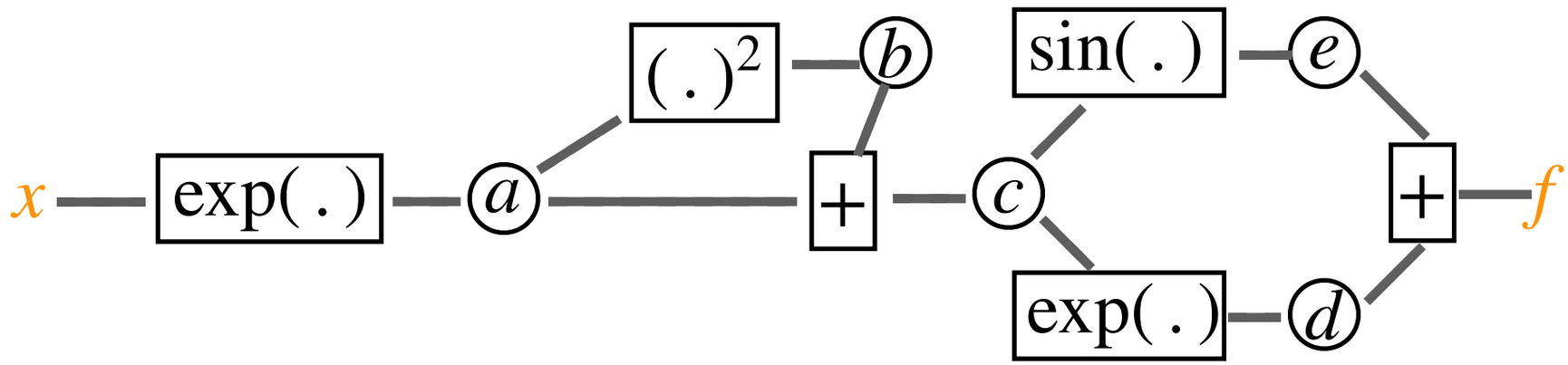
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



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

« Backward »

« Automatic differentiation »

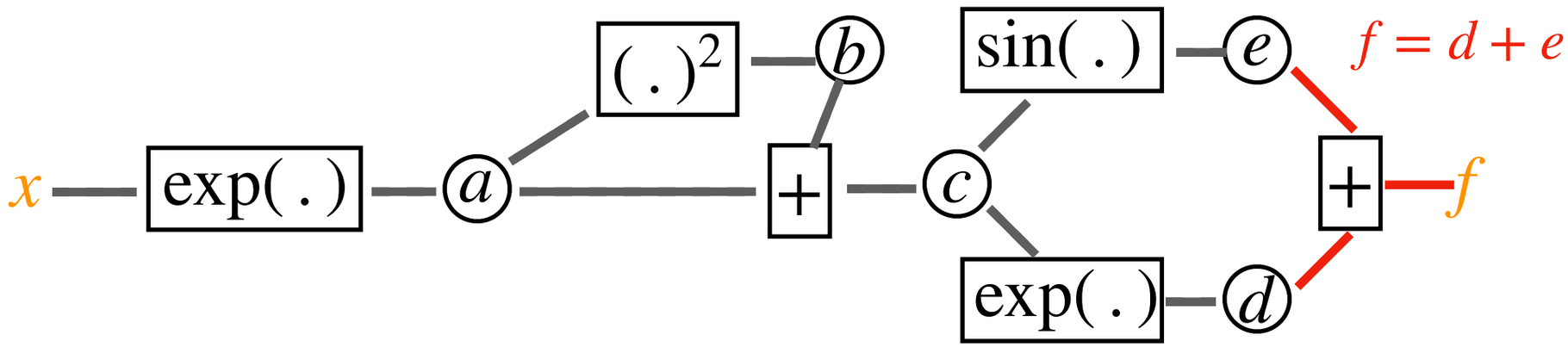
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



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

« Backward »

« Automatic differentiation »

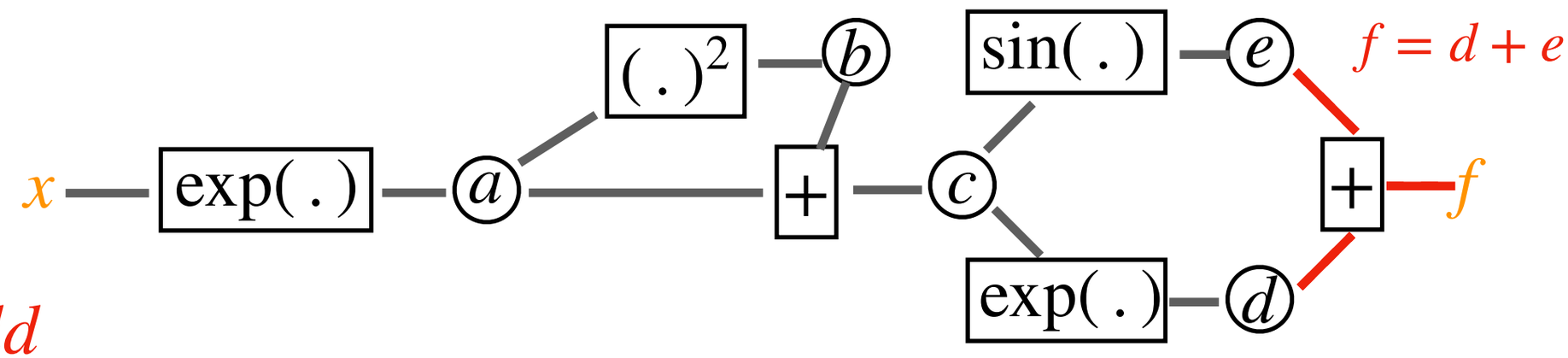
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{df}{dx} = \frac{df}{de}\frac{de}{dx} + \frac{df}{dd}\frac{dd}{dx}$$

« Backward »

« Automatic differentiation »

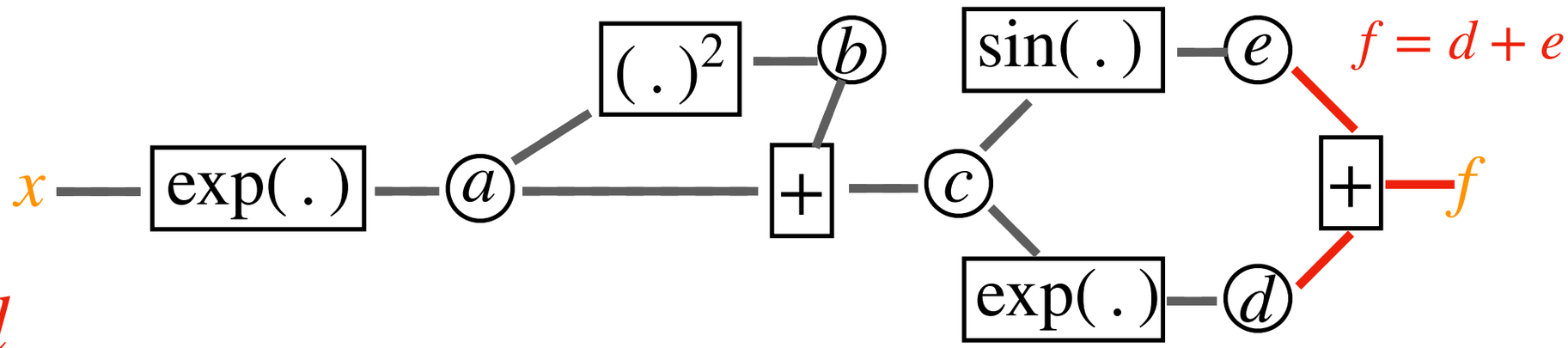
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{df}{dx} = 1\frac{de}{dx} + 1\frac{dd}{dx}$$

« Backward »

« Automatic differentiation »

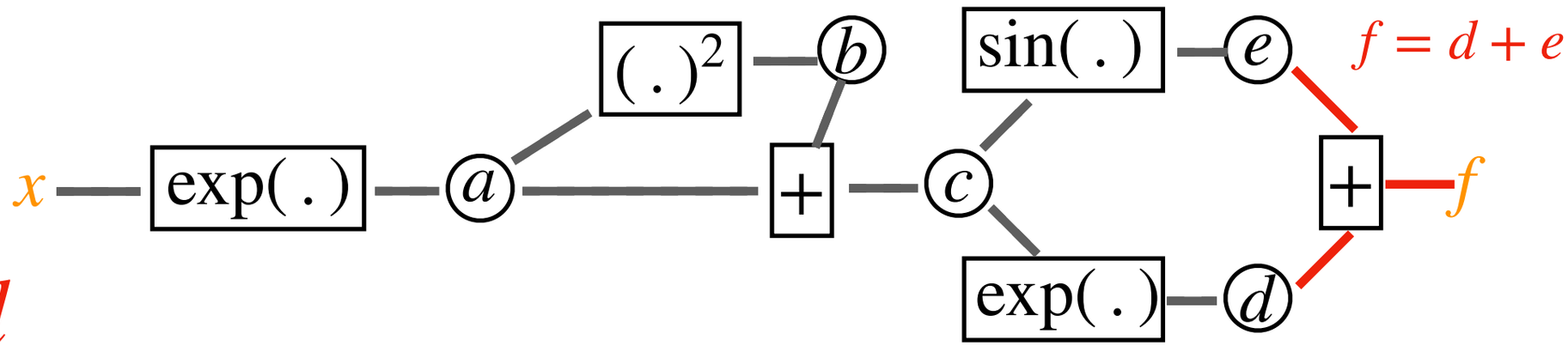
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{df}{dx} = \frac{de}{dx} + \frac{dd}{dx}$$

« Backward »

« Automatic differentiation »

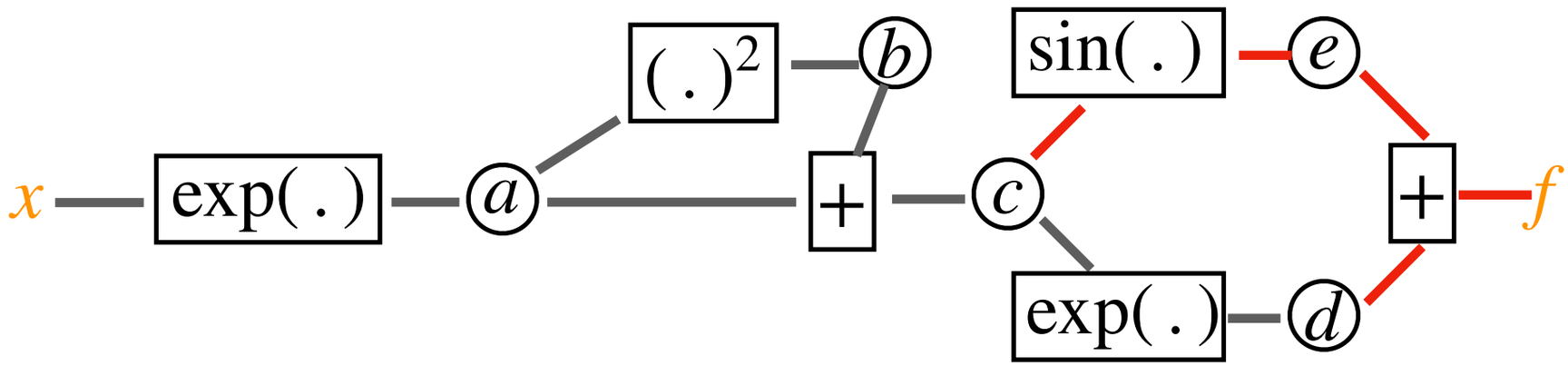
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{de}{dx} = \frac{de}{dc}\frac{dc}{dx}$$

« Backward »

« Automatic differentiation »

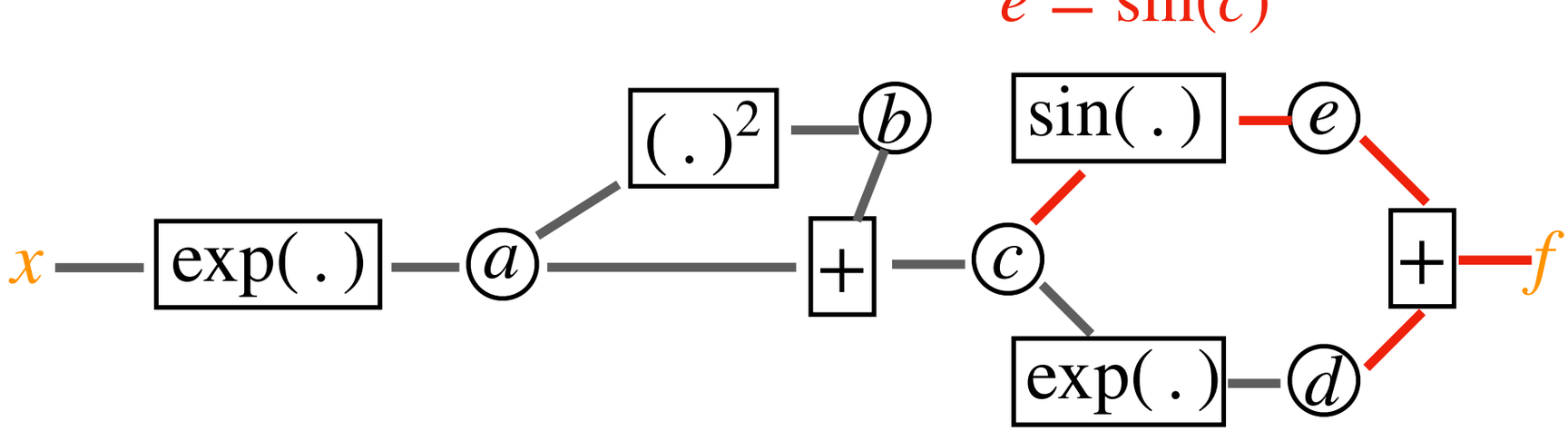
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{de}{dx} = \cos(c)\frac{dc}{dx}$$

« Backward »

« Automatic differentiation »

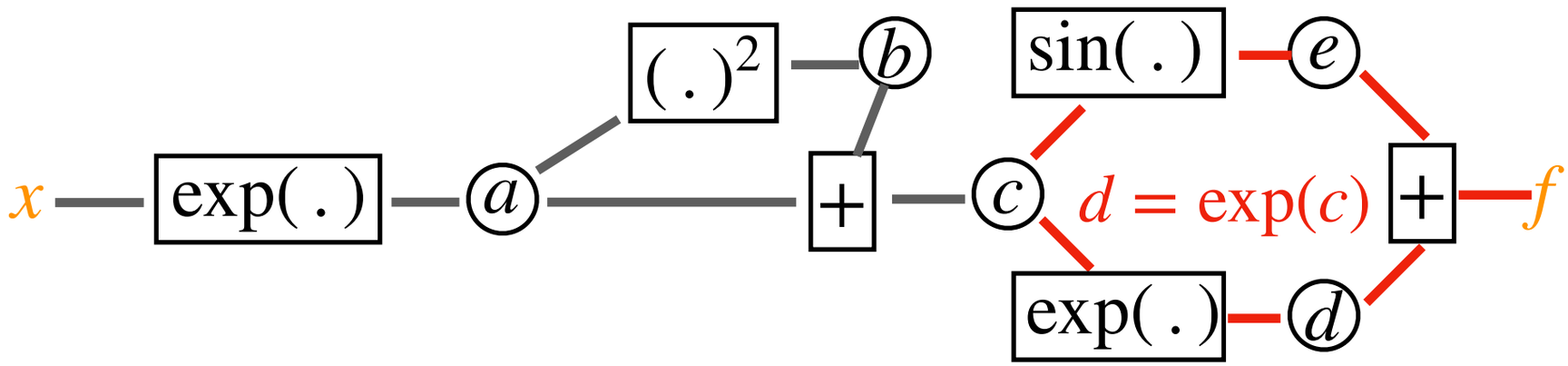
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{dd}{dx} = \exp(c)\frac{dc}{dx}$$

« Backward »

« Automatic differentiation »

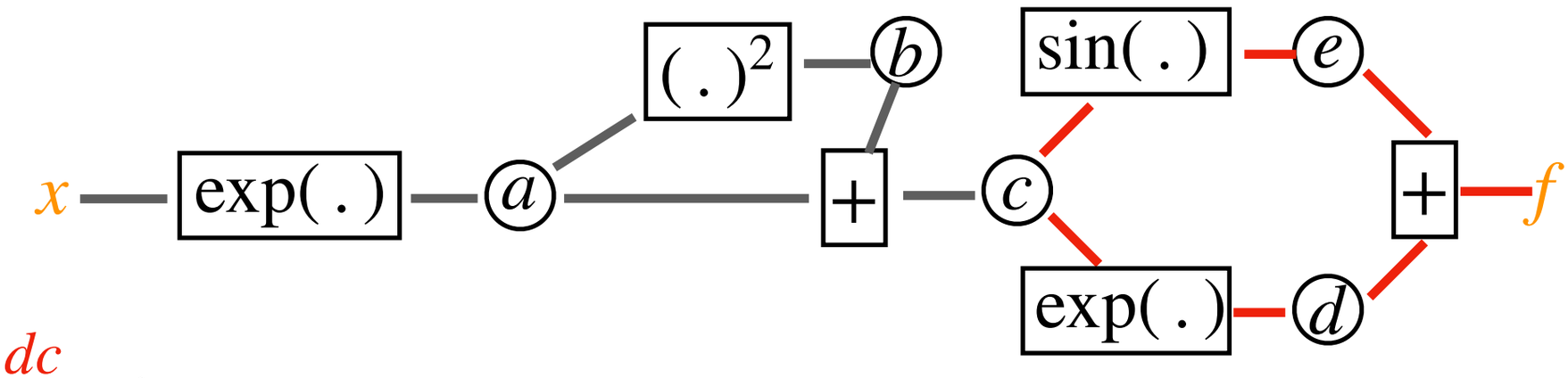
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{df}{dx} = (\exp(c) + \cos(c))\frac{dc}{dx}$$

« Backward »

« Automatic differentiation »

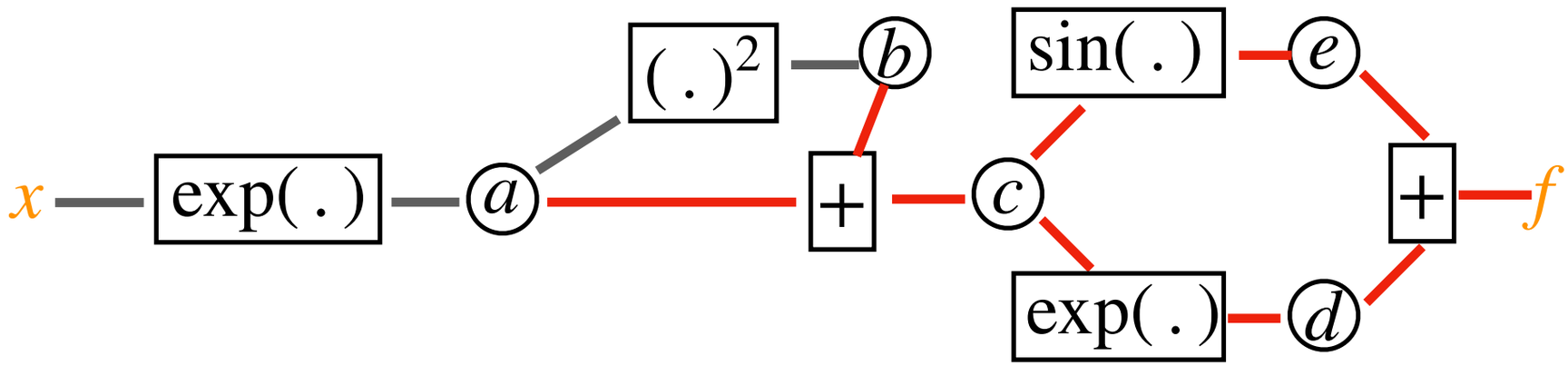
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



$$\frac{dc}{dx} = \ldots$$

« Backward »

« Automatic differentiation »

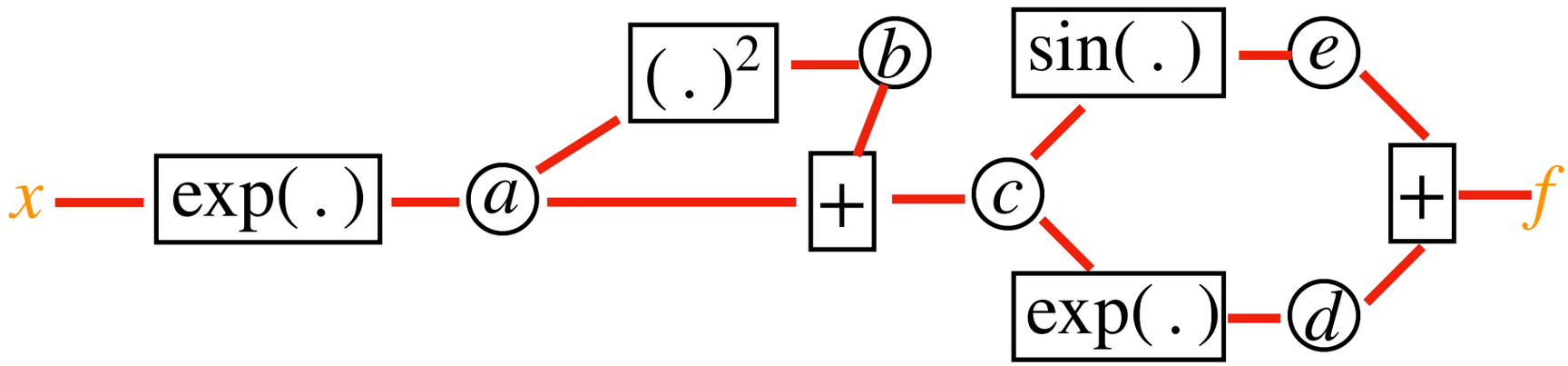
Chain rule:

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz}$$

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



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

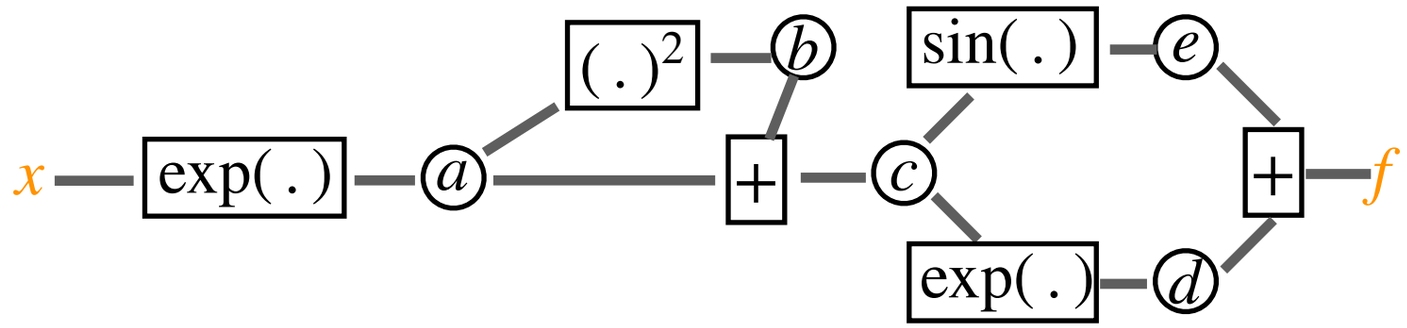
« Backward »

« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:

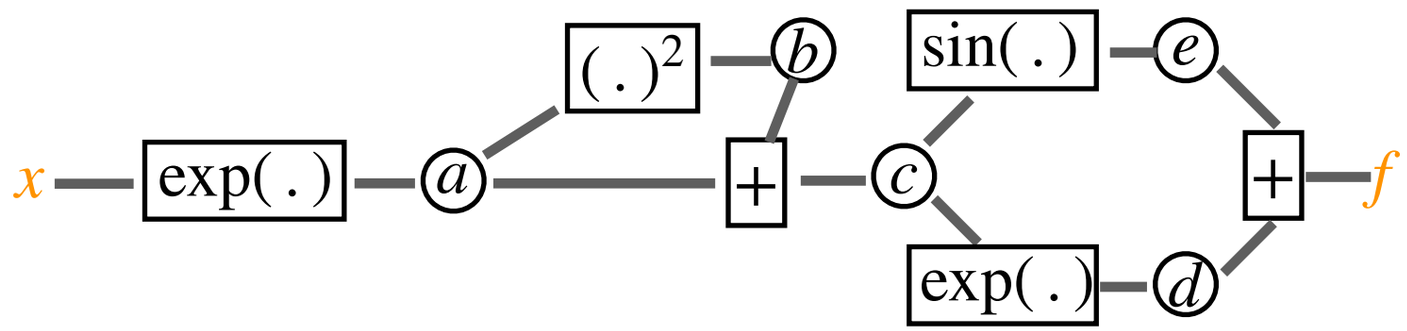


« Automatic differentiation »

General idea:

A "complex" function: $f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$

Automatic differentiation:



« Forward »

Input: $x = x_0$

For i in 1, …, N:

$$x_i \leftarrow g_i(z_{Pa(i)})$$

« Backward »

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

For i in N, N-1, …, 0:

$$x_i \leftarrow \sum_{j:i \in Pa(j)} \frac{df}{dx_j} \frac{dg_j}{dx_i}$$

« Automatic differentiation »

- **General idea:**
  - Permits to differentiate a function automatically
  - One only needs to know the graph defining the function
  - Some efficient implementations (GPU) exist
  - Computing the gradient is just as costly as computing the function
- **Limits:**
  - All layers must be differentiable
  - « gradient vanishing » problem for large networks
  - Very energy-intensive process for large networks!

Neural Networks: summary

- **What they are:**
  - A potentially very expressive and powerful class of functions
  - … made by composition of simple, elementary operations
- **How to train NN:**
  - Most often by (stochastic) gradient descent, and their variants
  - Uses automatic differentiation
  - **Warning:** non-convex optimization problem ! (Local minima)

Playground

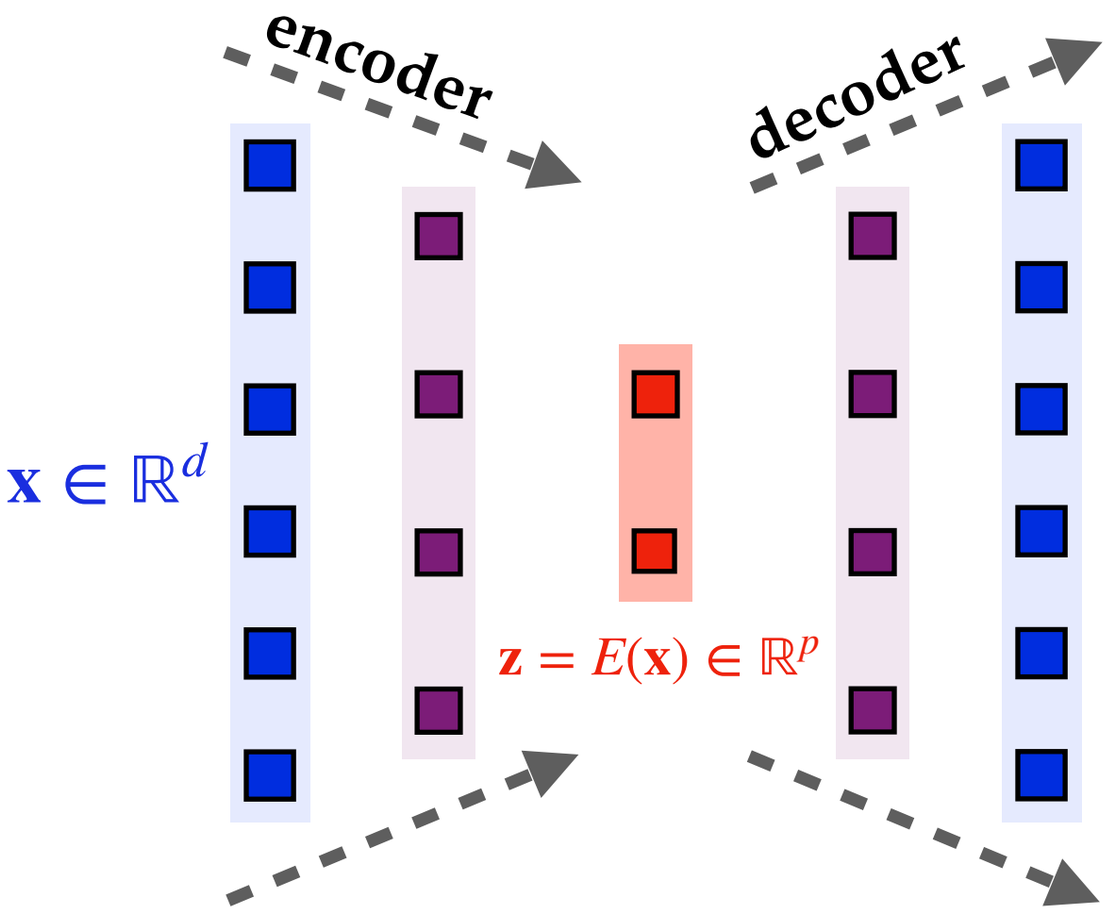
https://playground.tensorflow.org/

Focus on auto-encoders

- **Goal:** find a low-dimension representation of the data…
- … which should permit a good reconstruction

Focus on auto-encoders

- **Goal:** find a low-dimension representation of the data…
- … which should permit a good reconstruction

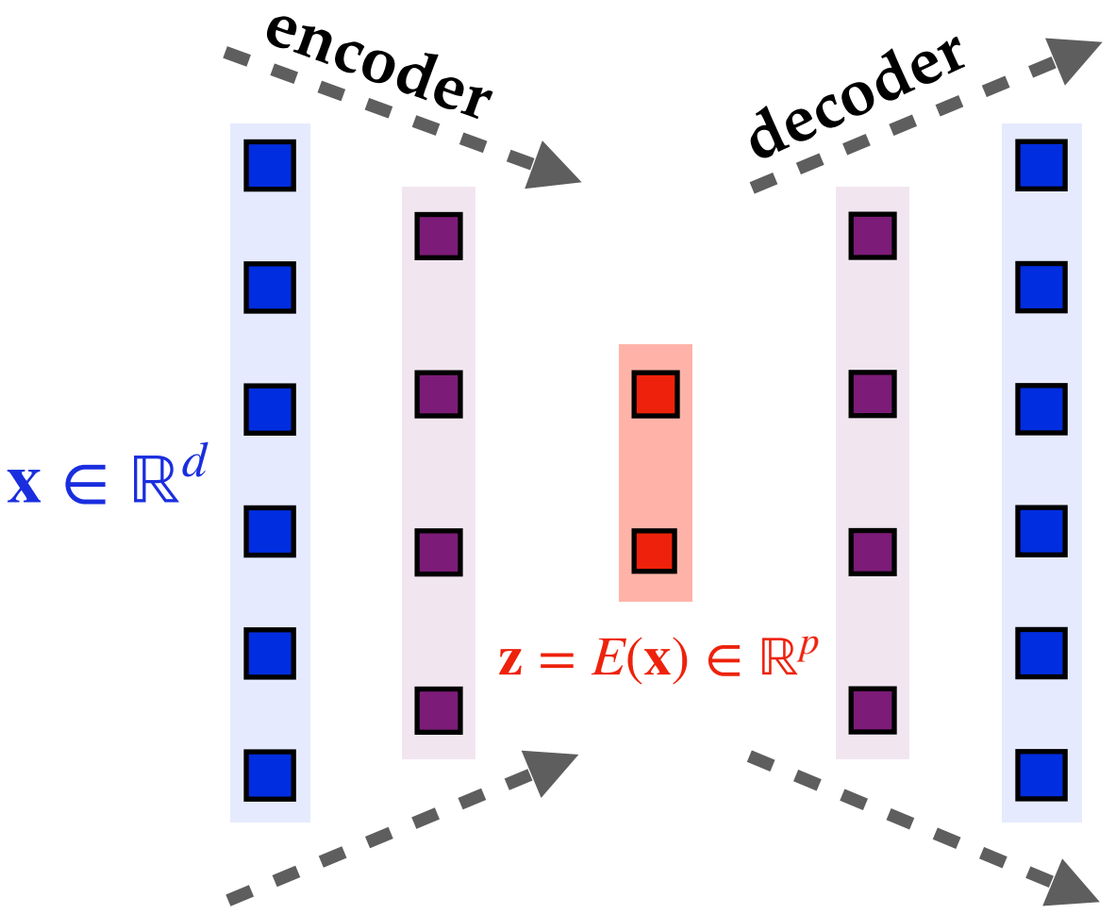


Minimize the reconstruction error

$$\min_{D,E} \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - D[E(\mathbf{x}_i)]\|^2$$

Focus on auto-encoders

- **Goal:** find a low-dimension representation of the data…
- … which should permit a good reconstruction



Advantages:

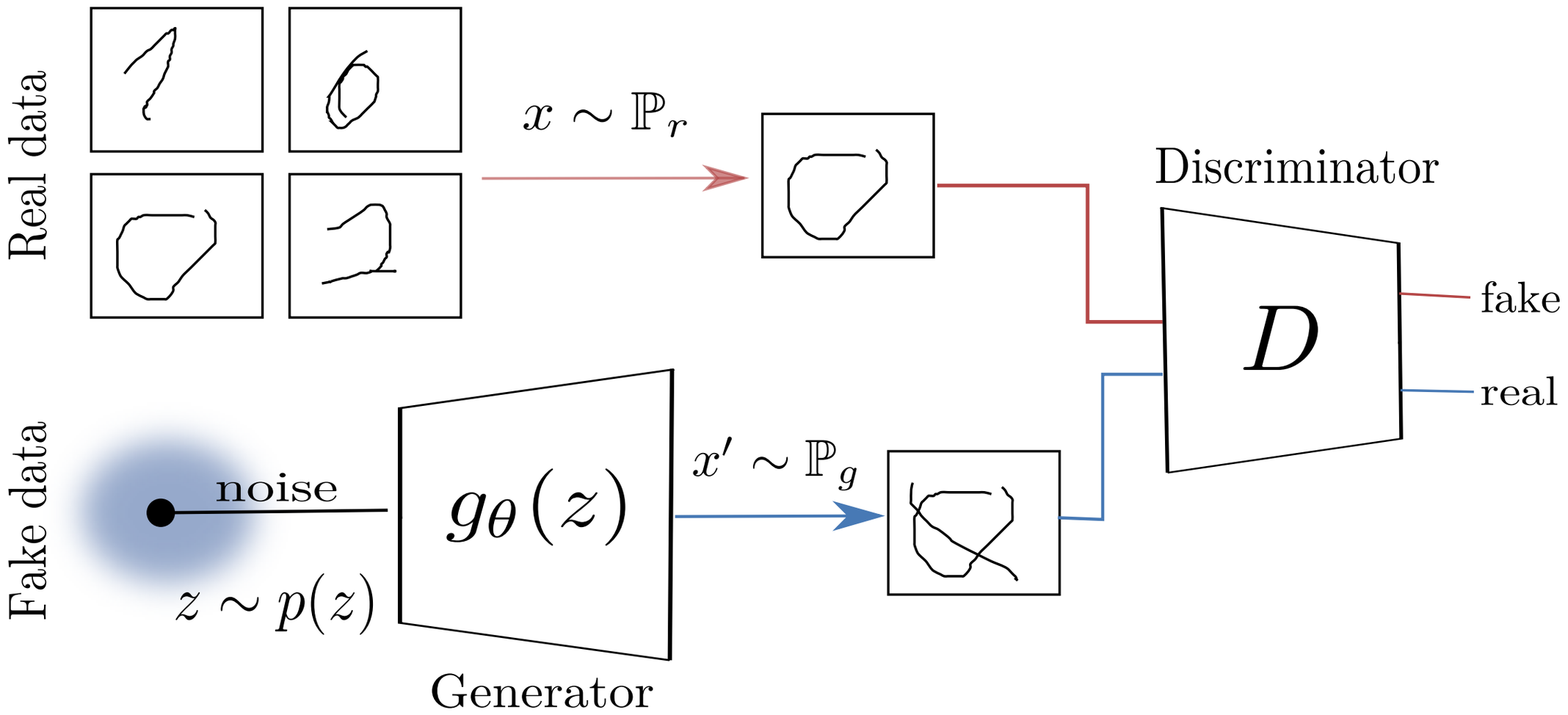
- Visualization/interpretation
- non linear dimensionality reduction

Minimize the reconstruction error

$$\min_{D,E} \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - D[E(\mathbf{x}_i)]\|^2$$

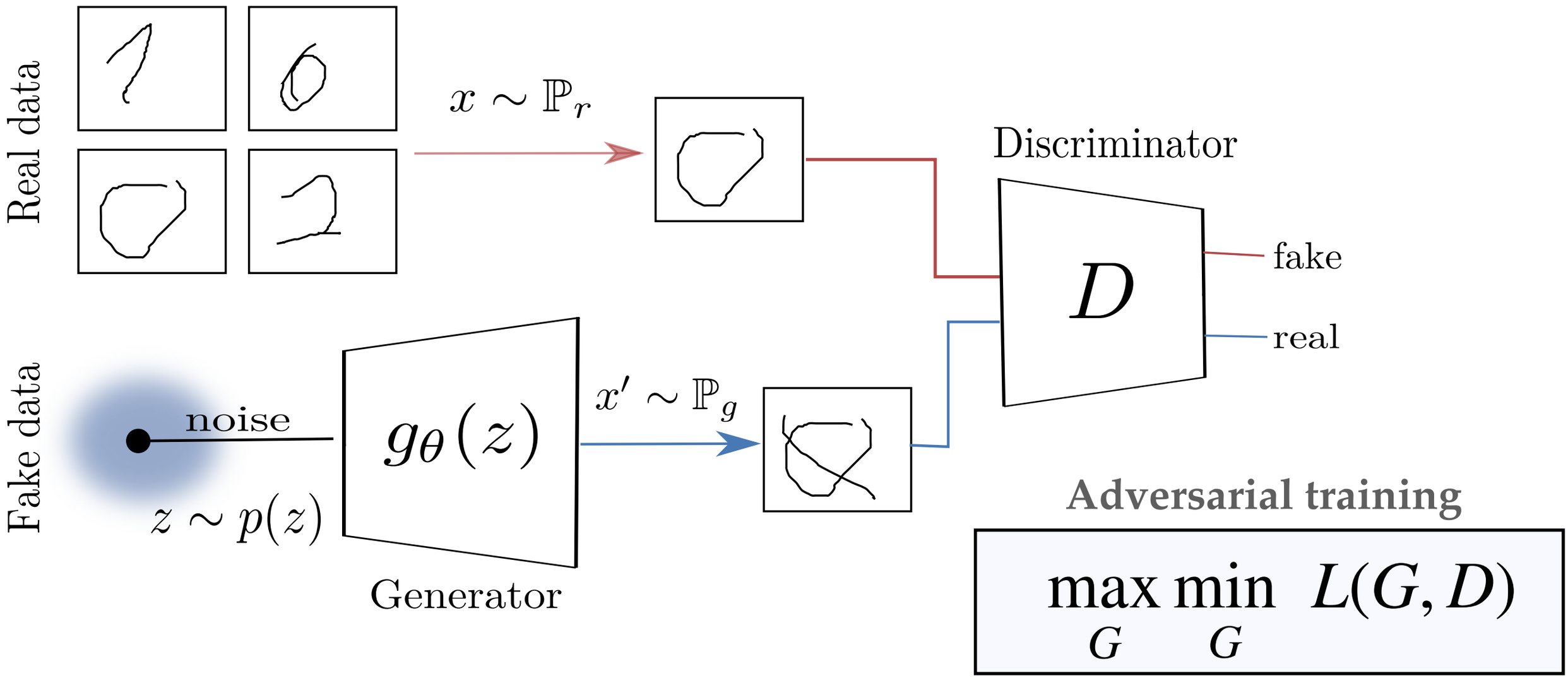
(Very short) focus on GANs

- **Goal:** generate new data that does not exist…
- … and that « looks like » true data



(Very short) focus on GANs

- **Goal:** generate new data that does not exist…
- … and that « looks like » true data



Adversarial training

$$\max_G \min_G L(G, D)$$