Introduction to AI
Neural Nets basics

Machine Learning

- Automatically extract relevant generalities from a dataset
- Process: Training
- Relevant ? Depends upon the problem to solve

- Theoretical tools :
  - Mathematical analysis
  - Linear algebra
  - Probability / Statistics
  - ….

Types of applications: Supervised (classification)

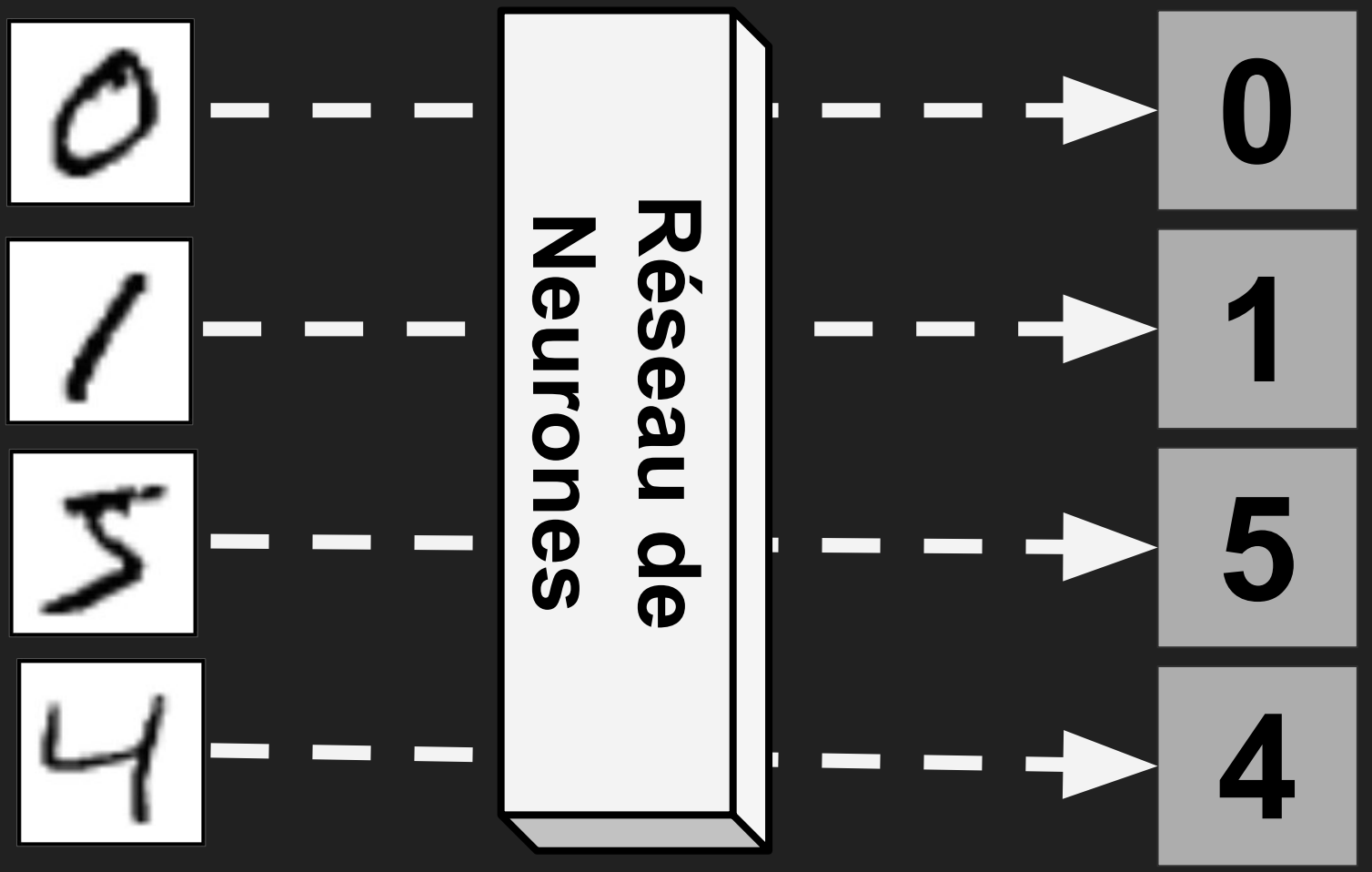
Data

Label / Target

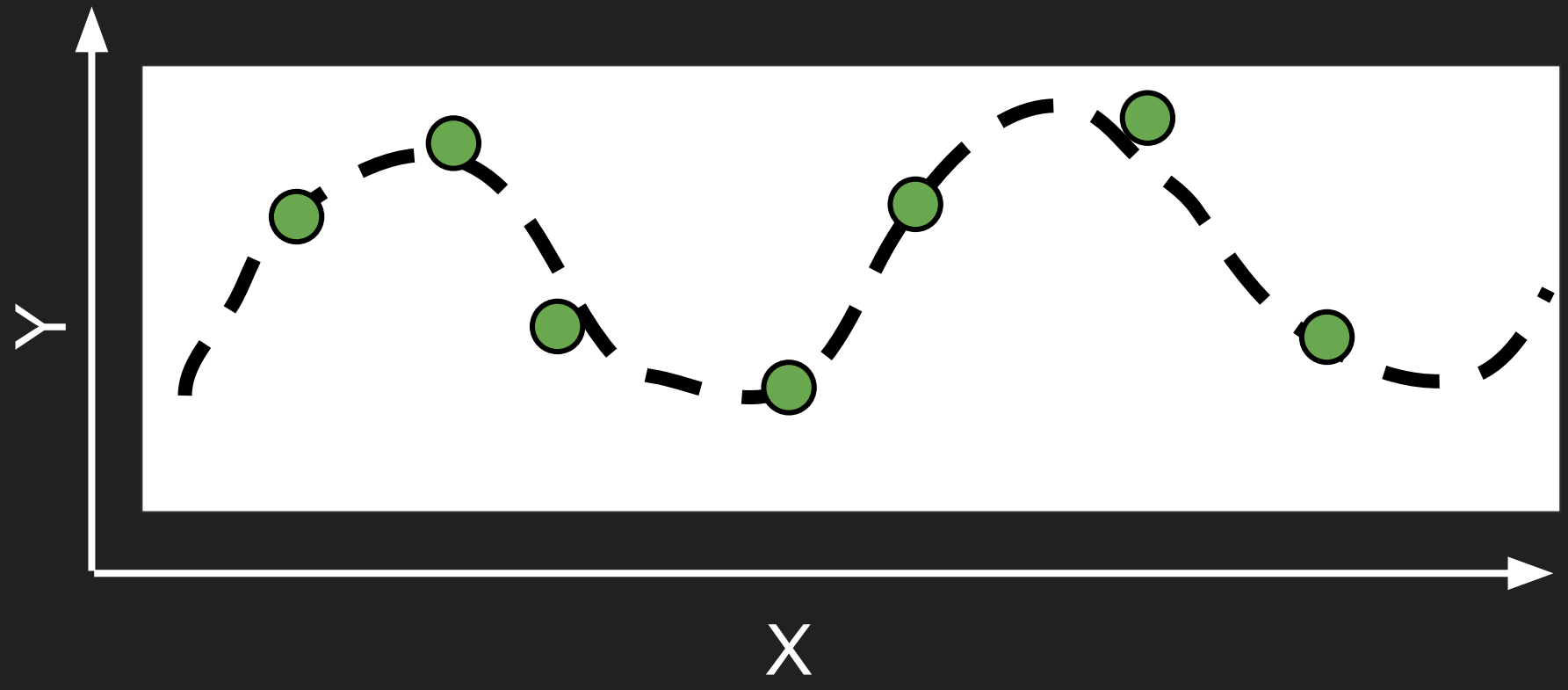
Dog

Cat

Types of applications: Supervised (classification)



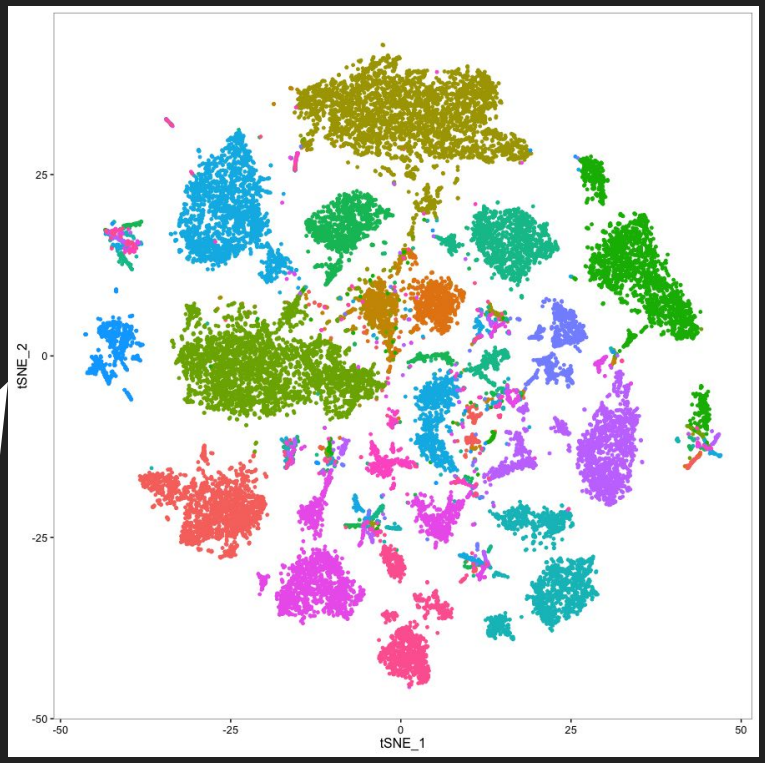
Types of applications: Supervised (regression)



Model a fonction from examples

Types of applications: Supervised (regression)

- No labels / Targets
- Discover structures in the dataset
  - Clustering
  - Visualization



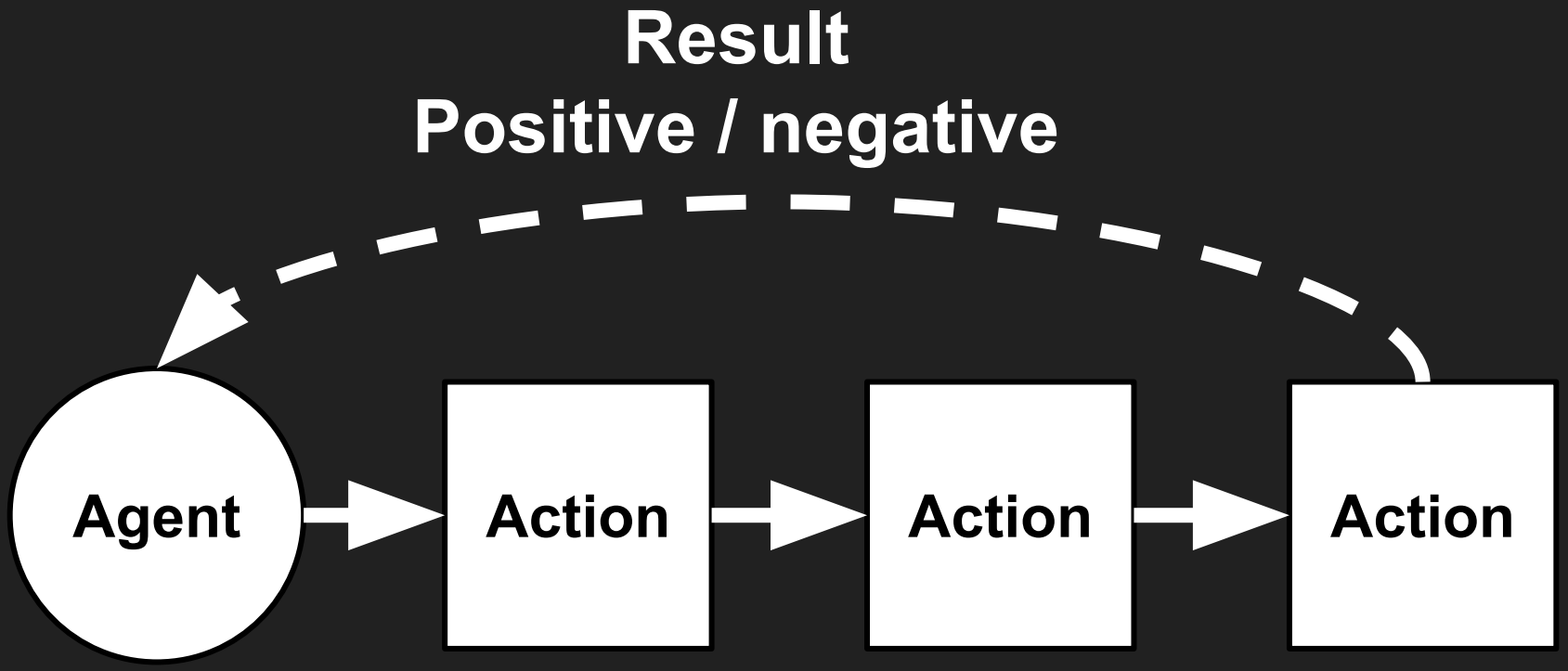
Satija lab

Gene expression

1 Gene: 20k values

1 Gene: 2D

Types applications: Reinforcement learning



Types applications: Generative models

- Learn to generate examples similar to training examples

Data handling

Encoding

Continuous

- **Images**
- **Pulse rates**
- **Temperature**

Discrete

- **Classes**
- **Categories**
- **...**

- Reduce range between values
  - log(values)
- Between [0, 1] (divide by the max)

1 Item => 1 unique id

0	Horse
1	Cow
2	Dog
...	...

Standard format: numpy arrays / pandas dataframes

Continues

- **Images**
- **Pulse rates**
- **Temperature**

float32

Discrete

- **Classes**
- **Categories**
- **...**

int16

```
data = numpy.asarray(data, dtype='int16')
```

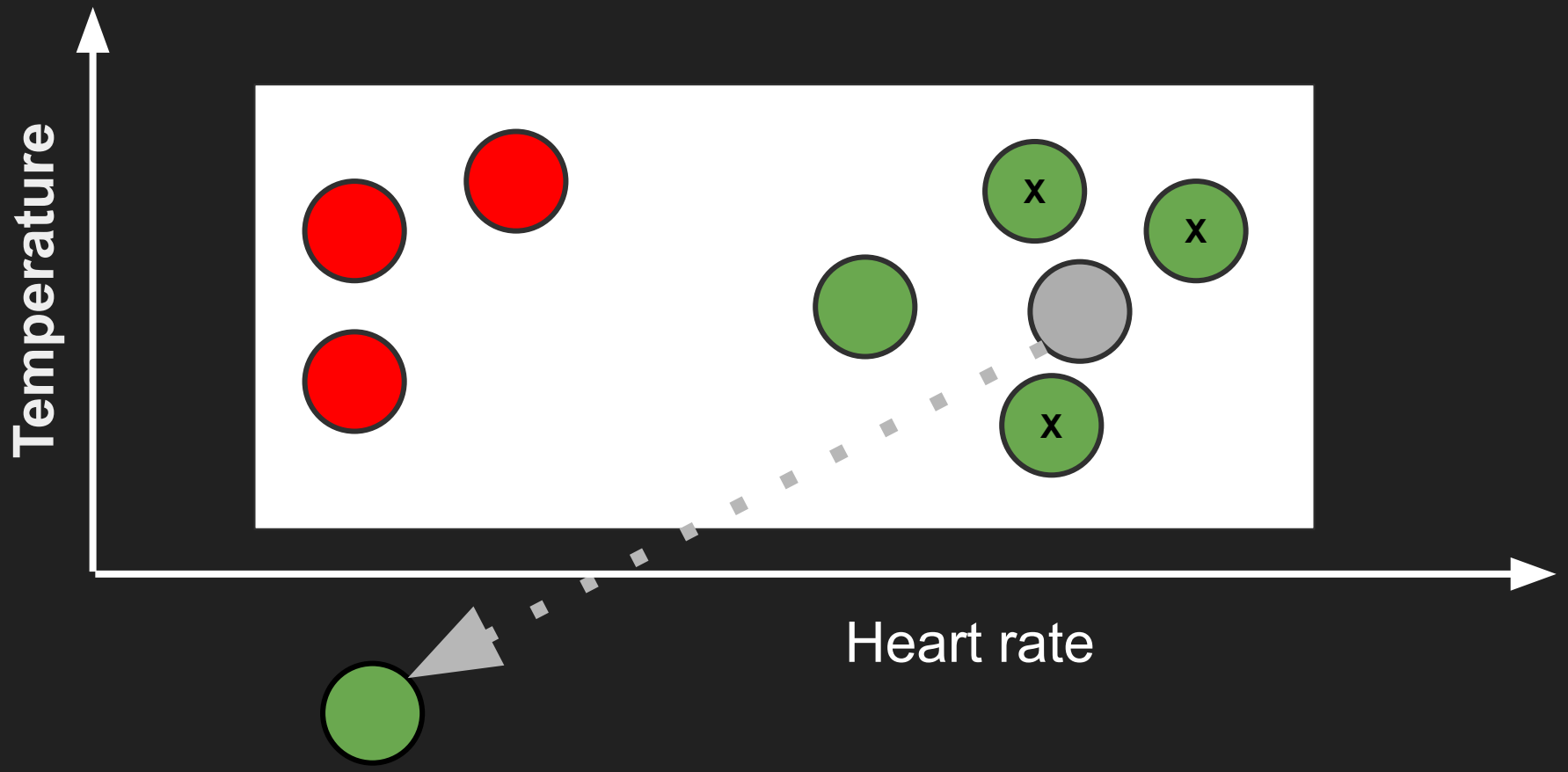
```
data = numpy.asarray(data, dtype='float32')
```

Apprentissage supervisé

Ensemble de données

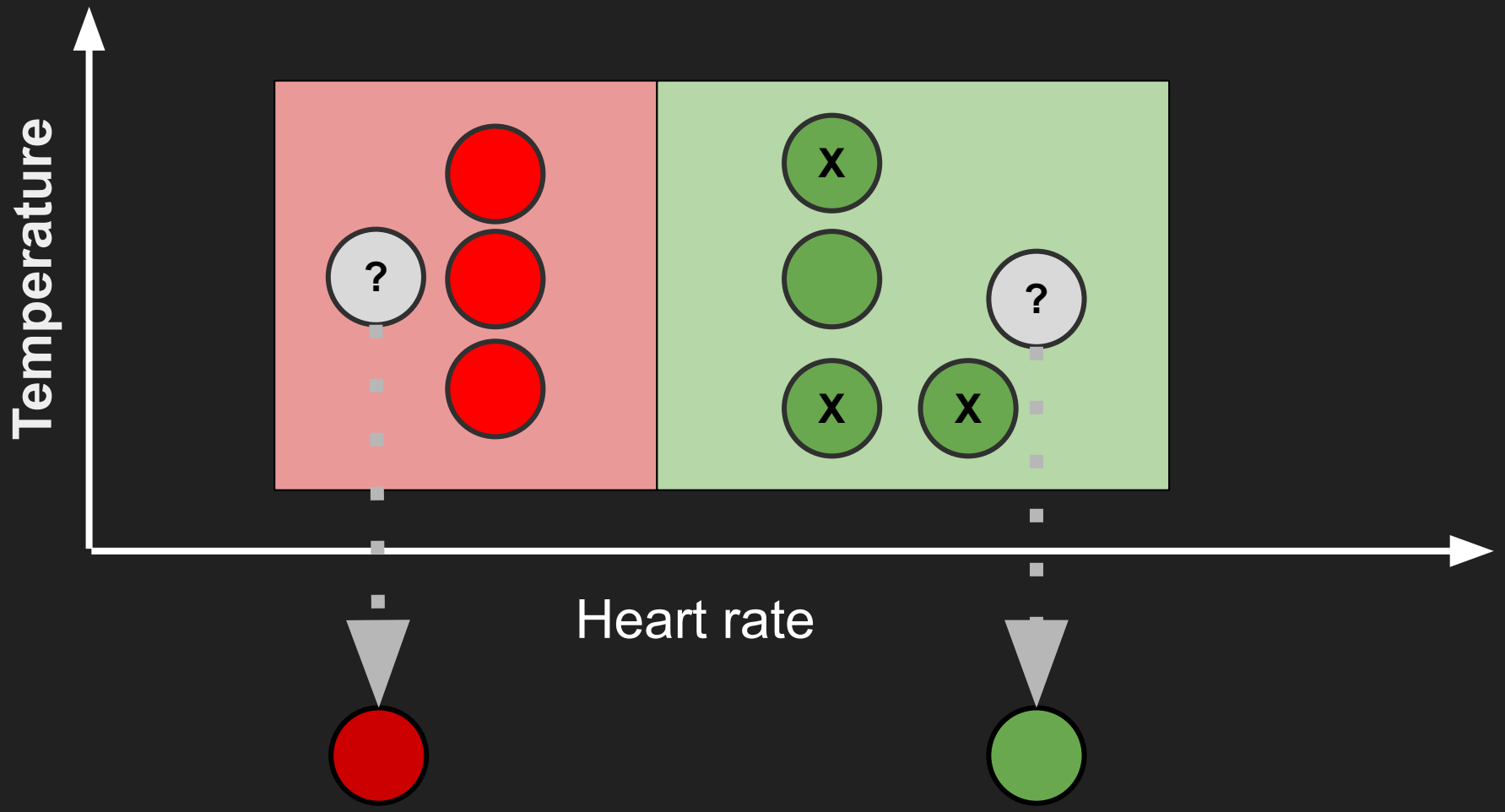
Temperature	Heart rate	Type
0.1	0.32	
0.9	0.5	
0.7	0.59	

Ex classification: K nearest neighbours (KNN)



Assign class of K (ex: 3)
nearest neighbours

Classification: As space partitioning



Artificial neural networks
I

Dataset: predict **Temperature** from **Heart rate**

Heart rate (x)	Temperature (z)
0.02	0.2
0.04	0.4
0.08	0.8

Model

$z = w * x$

Find the best value for w

$z = 10 * x$

Ensemble de données

Concentration (x)	Effet (z)
0.02	3.2
0.04	3.4
0.08	3.8

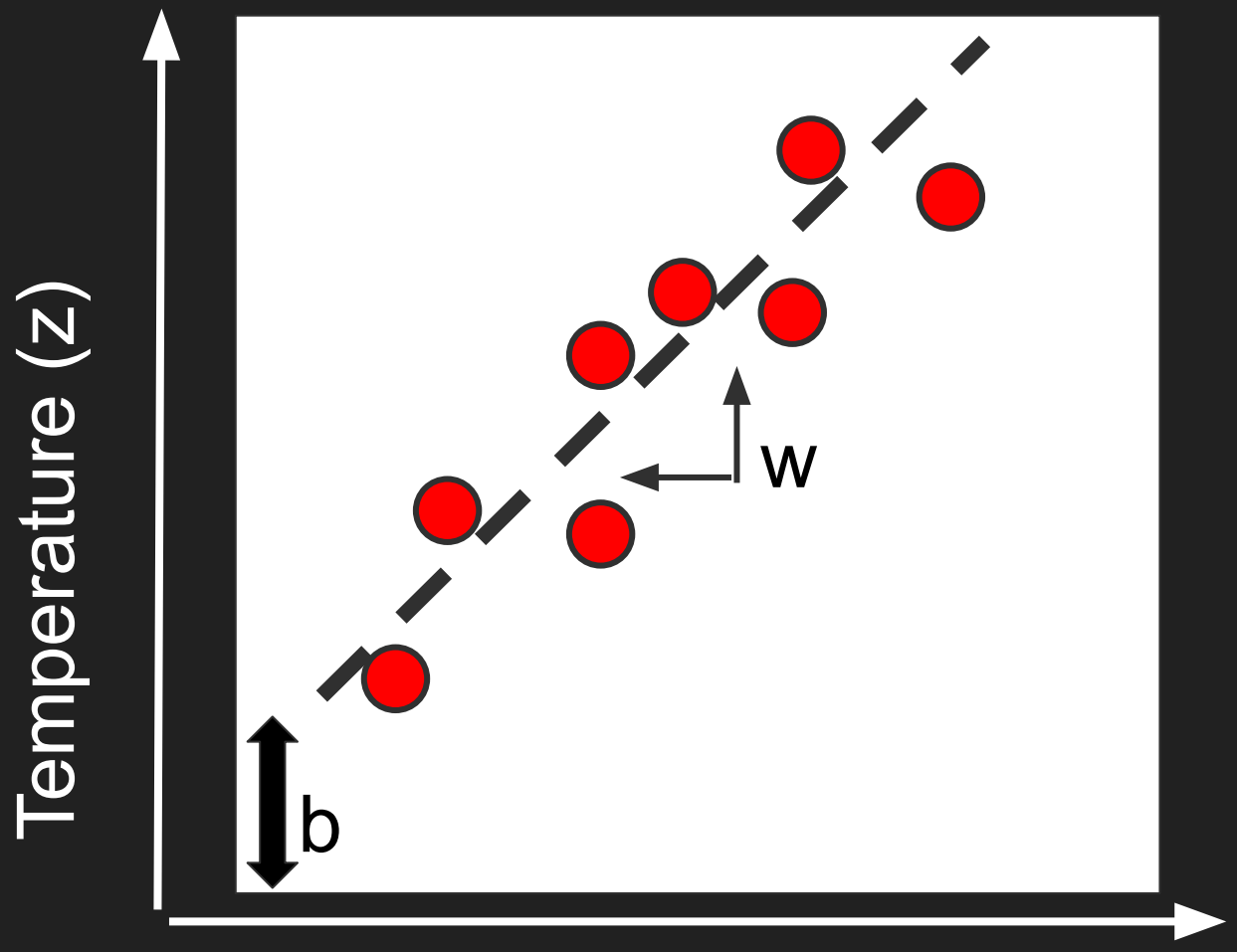
Modèle

$z = w * x + b$

$z = 10 * x + 3$

Biais (b)

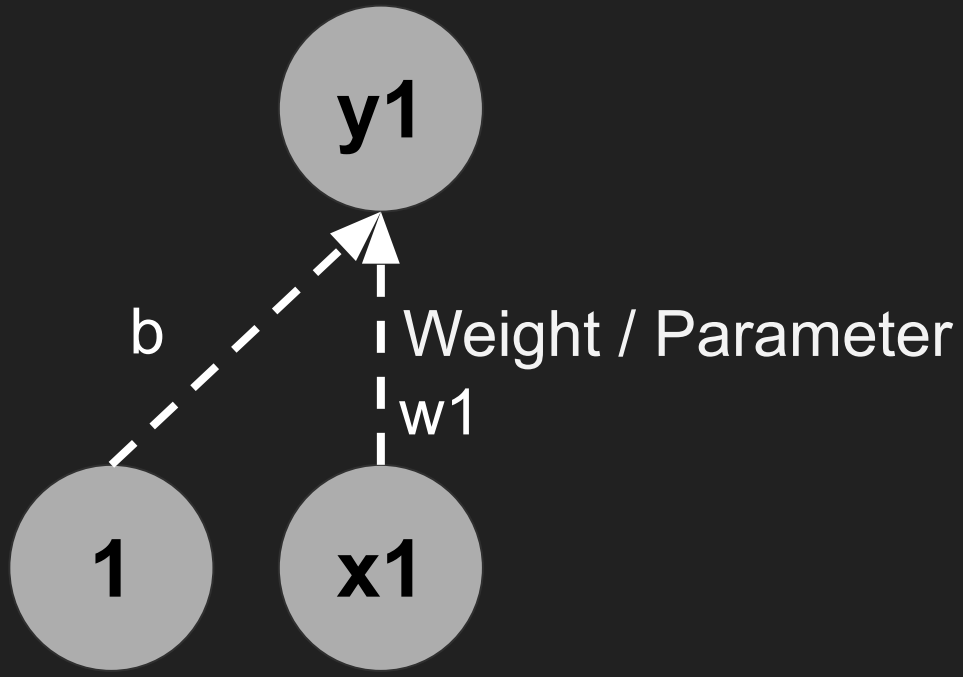
Régression linéaire



$z = w * x + b$

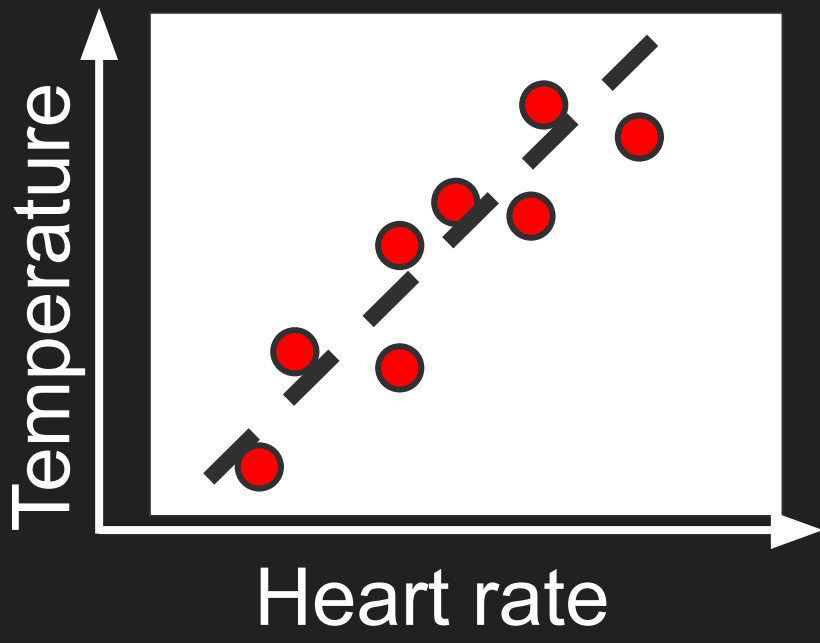
Baby neural network

Output layer (1 Neuron)



Input layer (1 Neuron)

$$F(x) = w1 * x + b$$

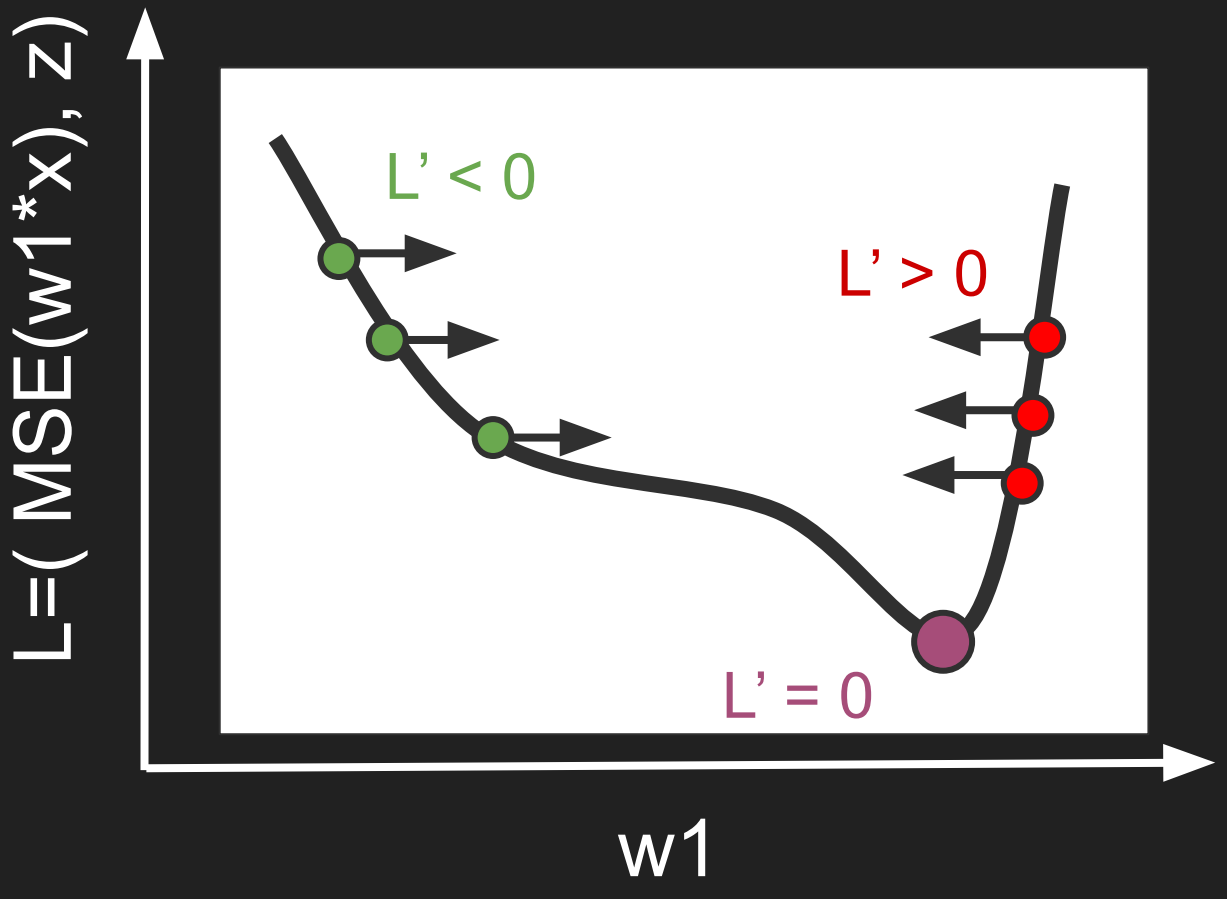


Finding, w1 automatically

1. Initialise w1 randomly
2. Select **n** random examples (input: x, target:z)
3. Compute y = w1*x
4. Compare y and expected value: z
    a. Loss, L = MeanSquaredError(z, y) = (z-y)^2
5. Find a value of w1 that reduces L
6. Start again from ‘2.’

ML = Find w1 automatically
How can we find w1 automatically?

Find a value of w1 that reduces L, how?



The variable is

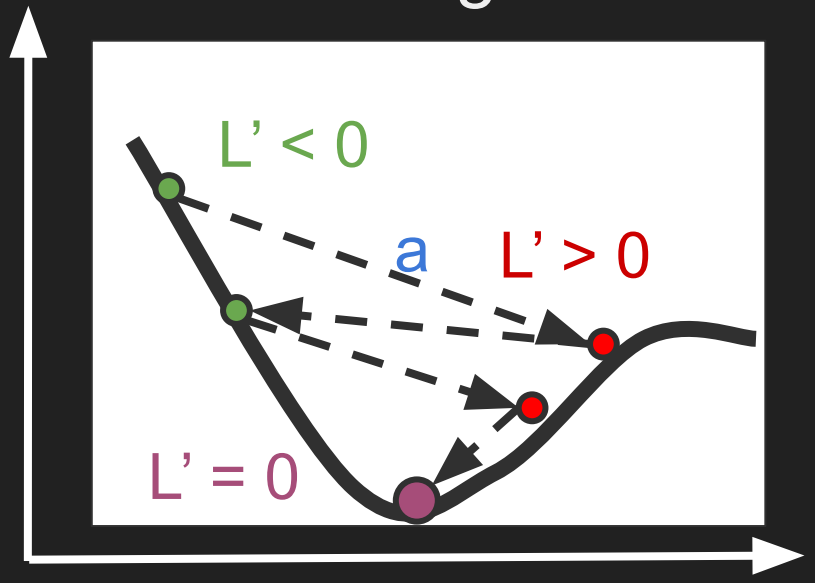
W1

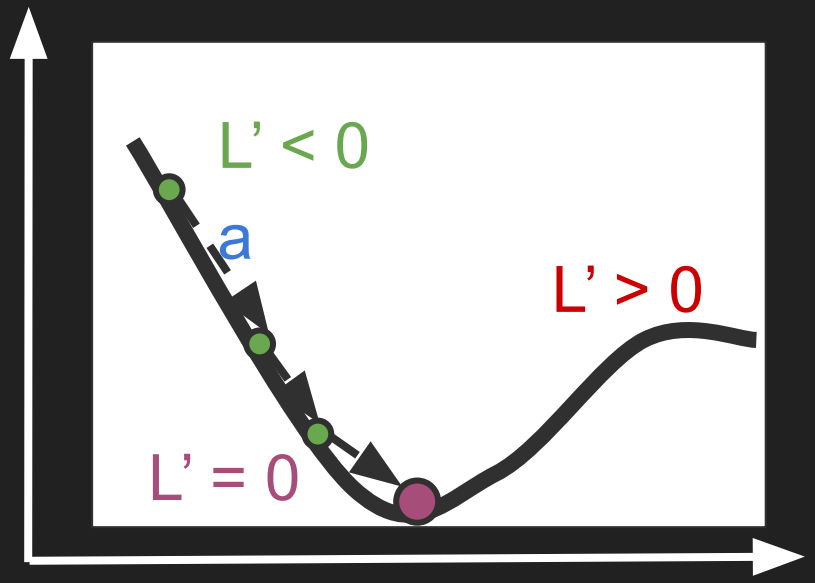
Not X

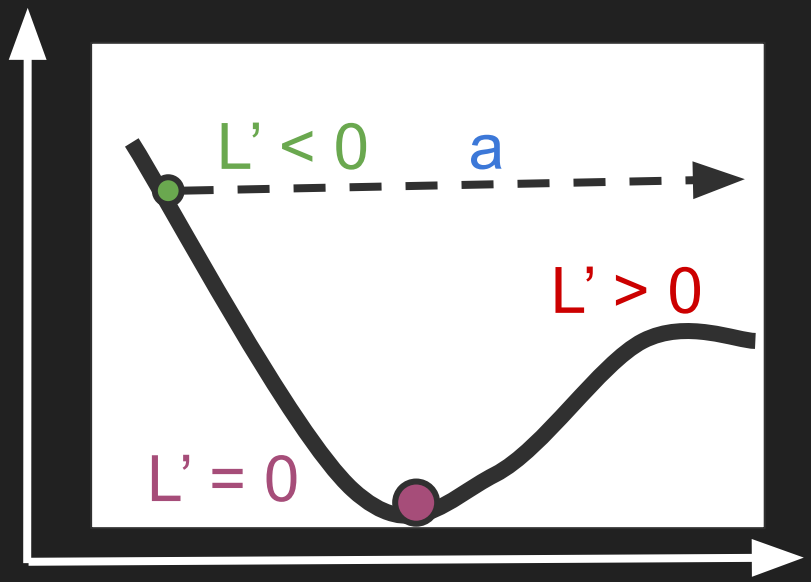
Gradient Descent

W1 = W1 - L'

Good convergence Vs Catastrophic bounces



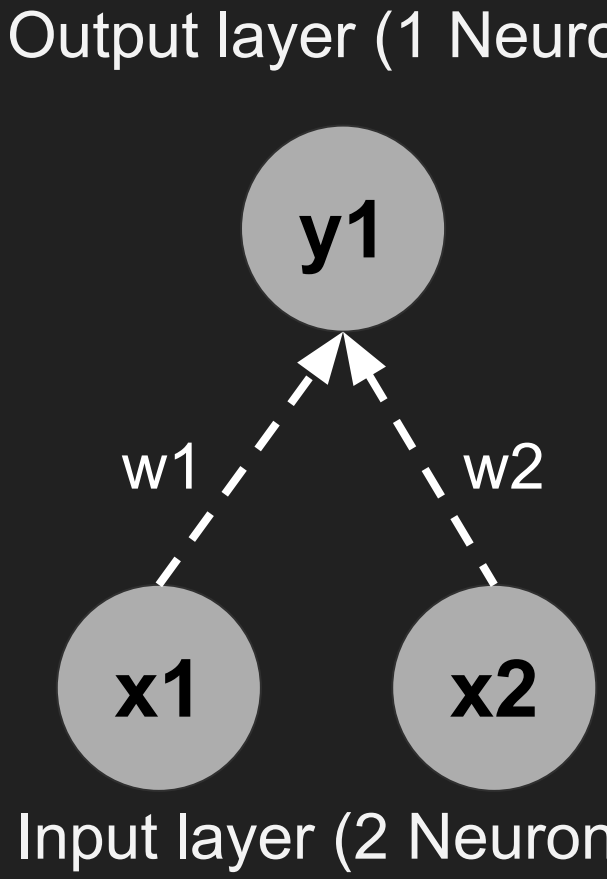




W1 = W1 - a*L’

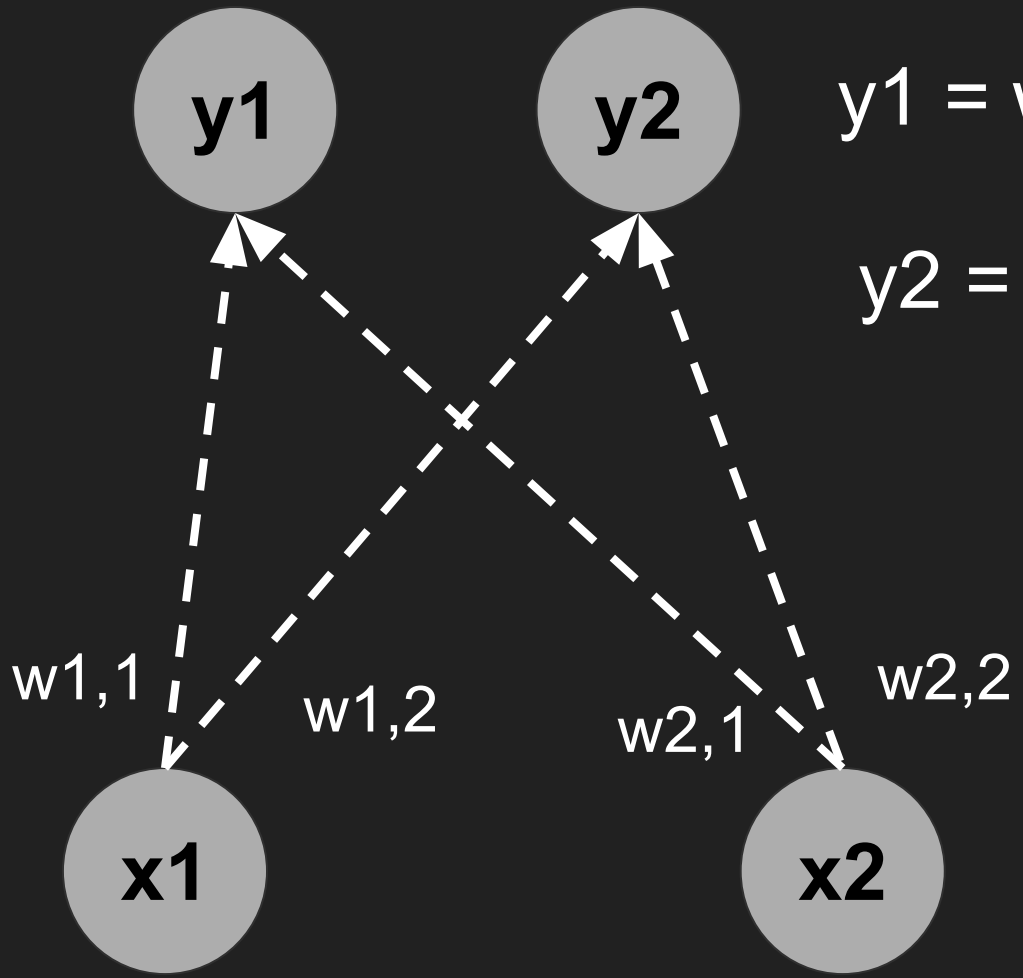
a: Learning rate

Baby neural network II



$$y1 = w1 * x1 + w2 * x2$$

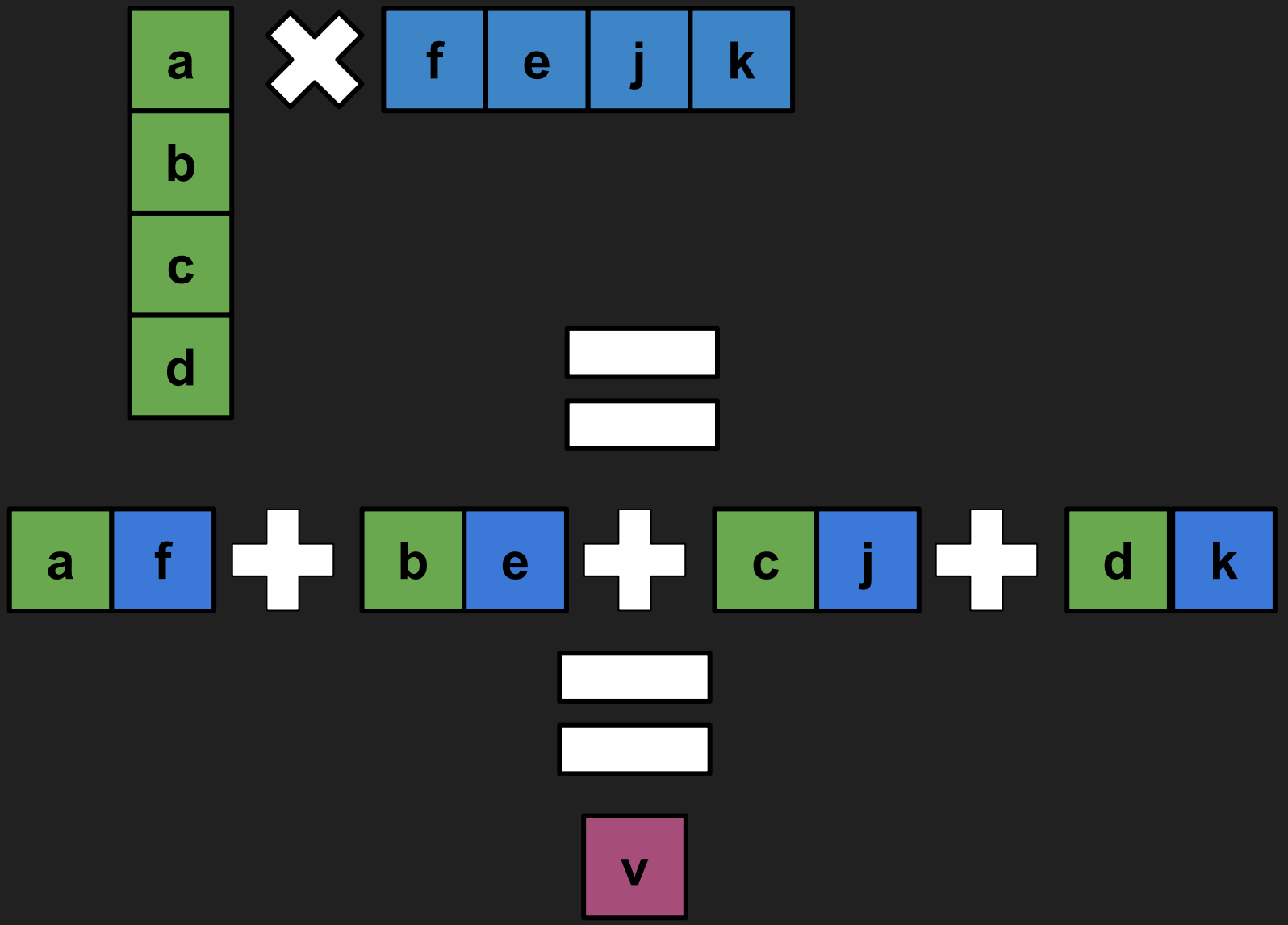
Baby neural network III



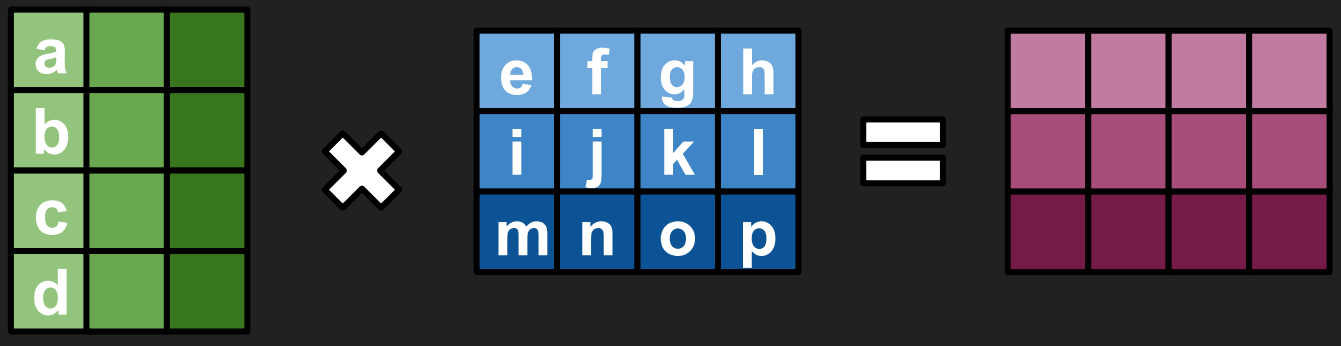
$$y1 = w1,1 * x1 + w2,1 * x2$$

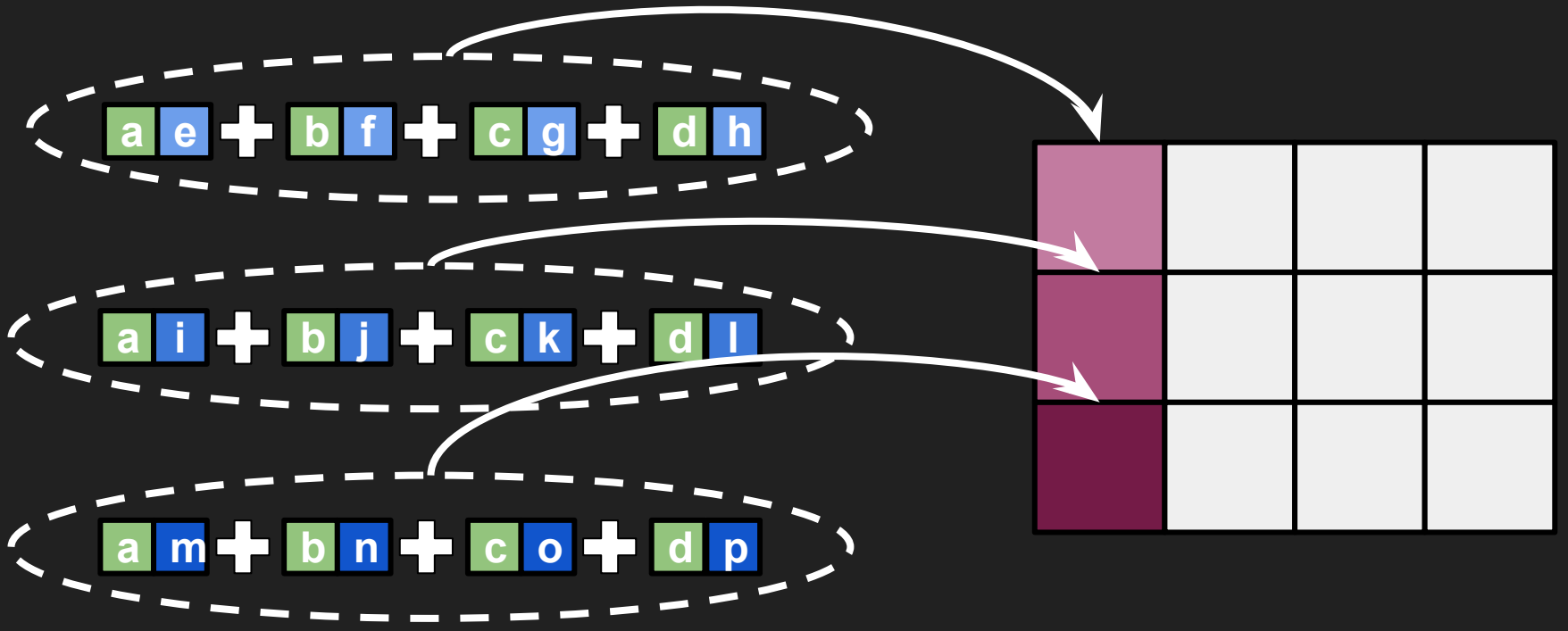
$$y2 = w2,1 * x1 + w2,2 * x2$$

Linear algebra - vectors

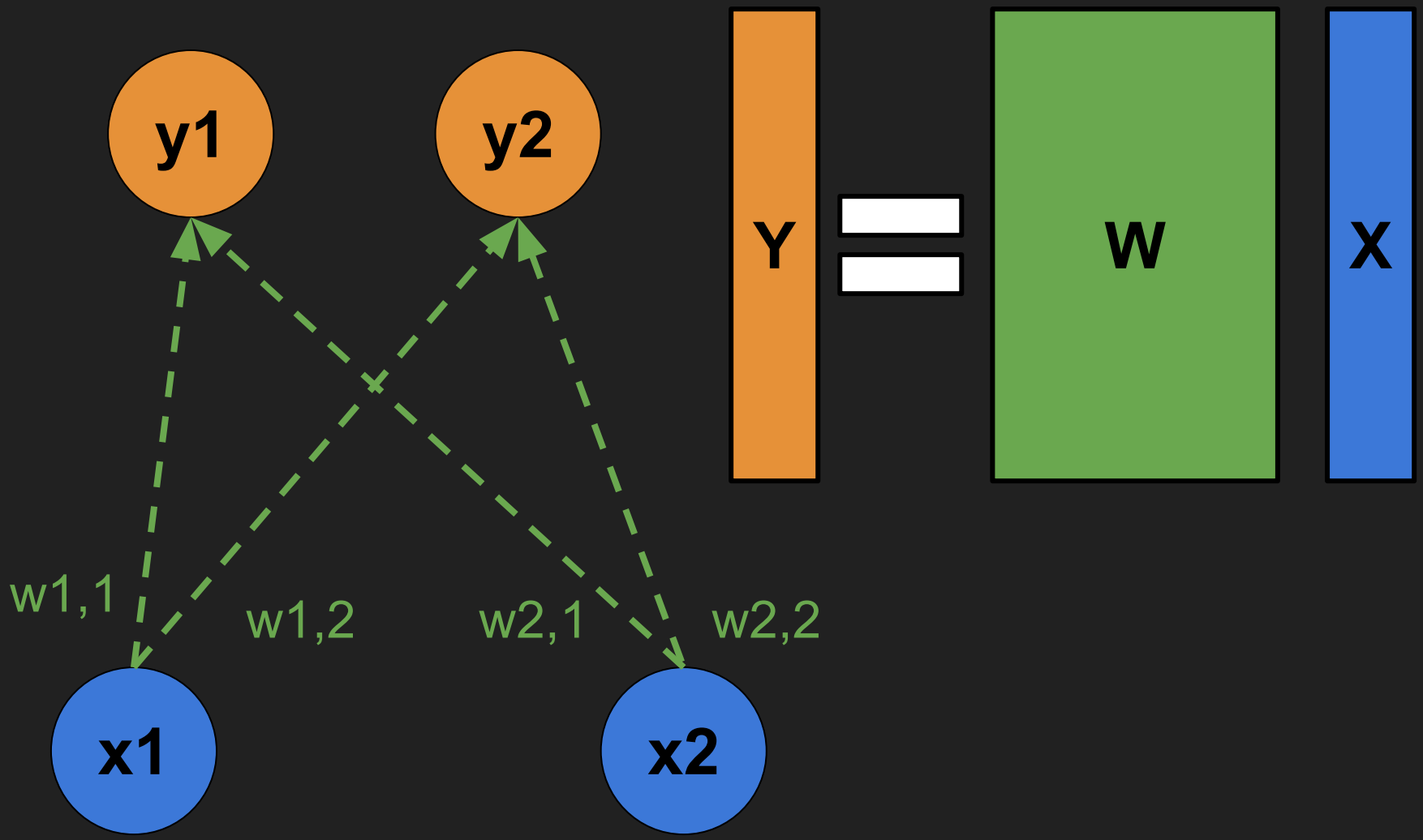


Linear algebra - Matrices

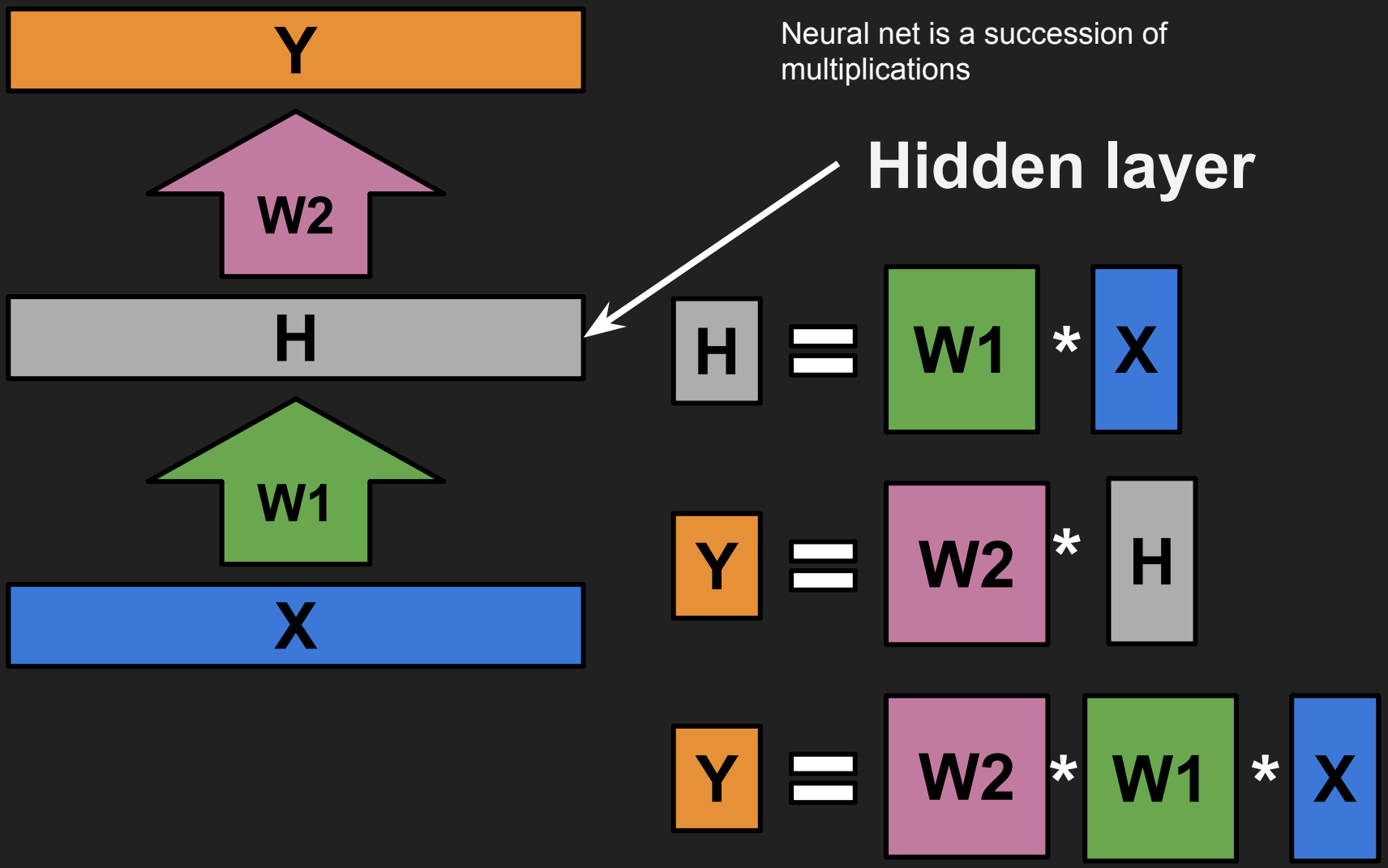




How networks are represented



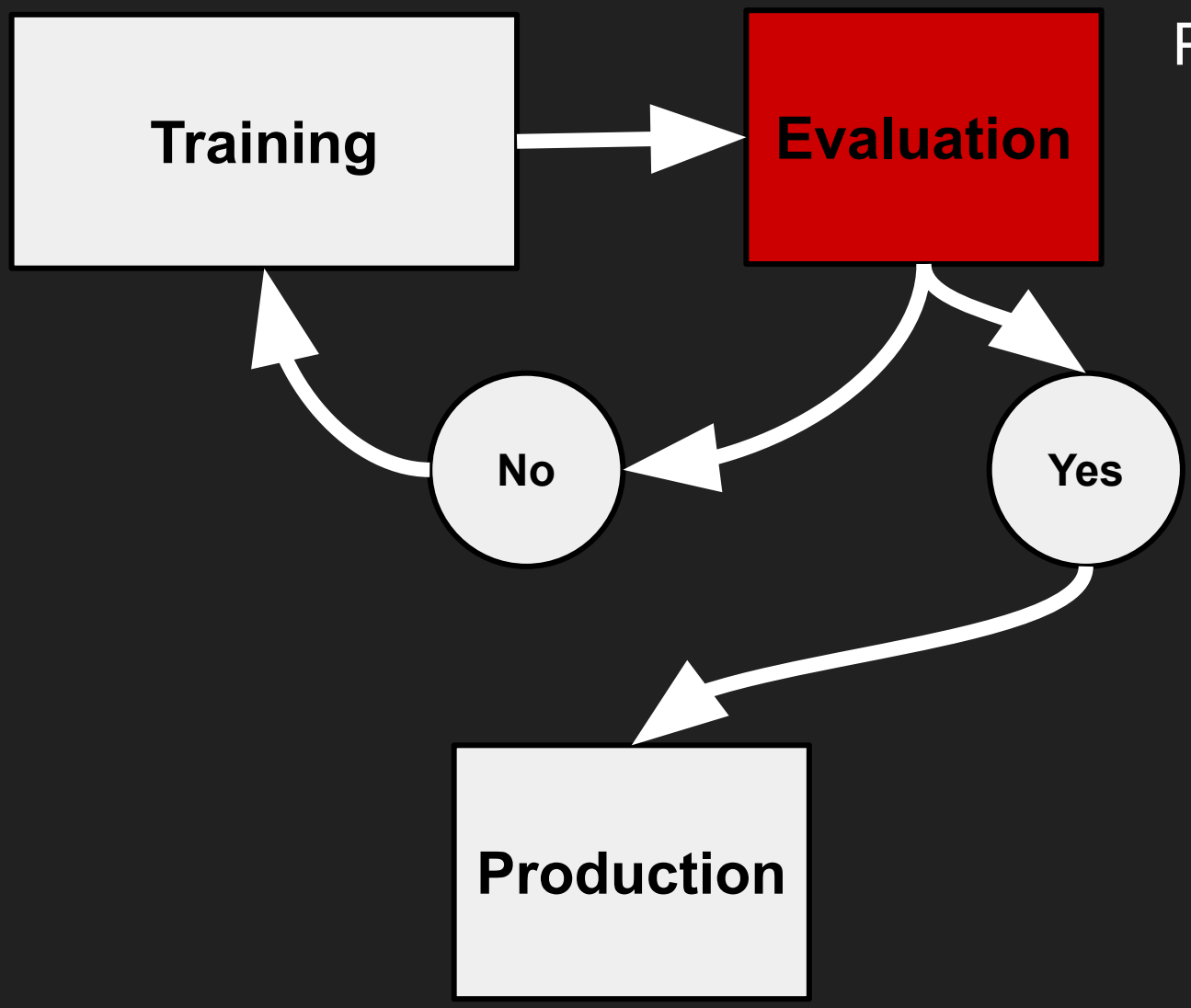
How networks are represented



Tensors, is just a name for (almost) everything

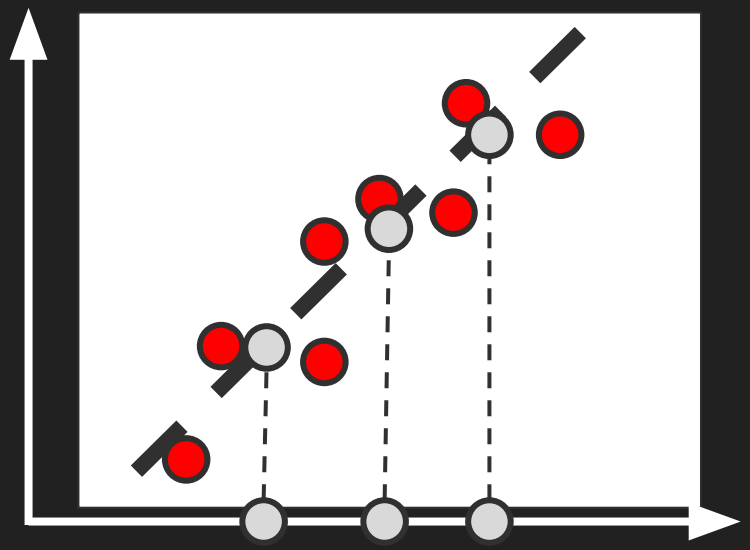
- Number: 0D tensor
- Vector: 1D tensor
- Matrix: 2D tensor
- ....

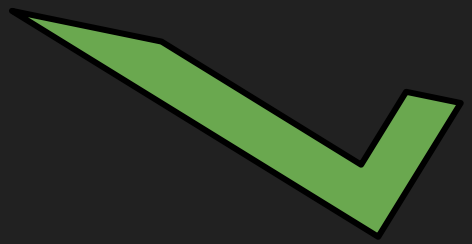
Workflow



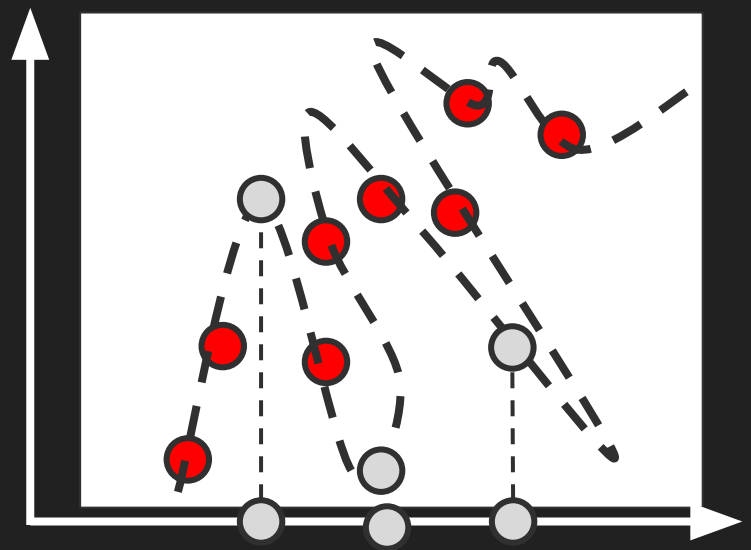
Performances on
“*real life*” data

Overdoing it (overfitting): Regression



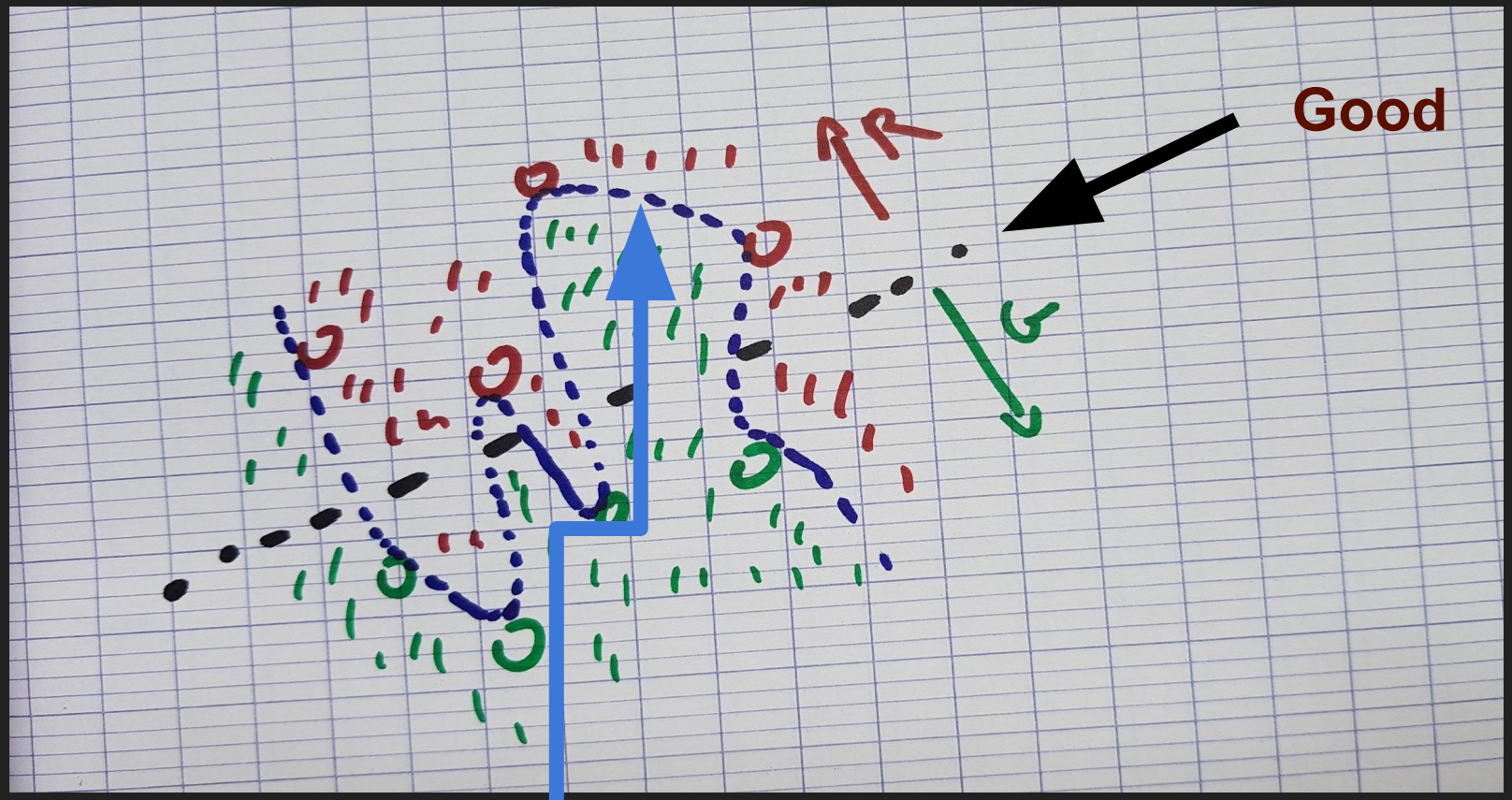


Good



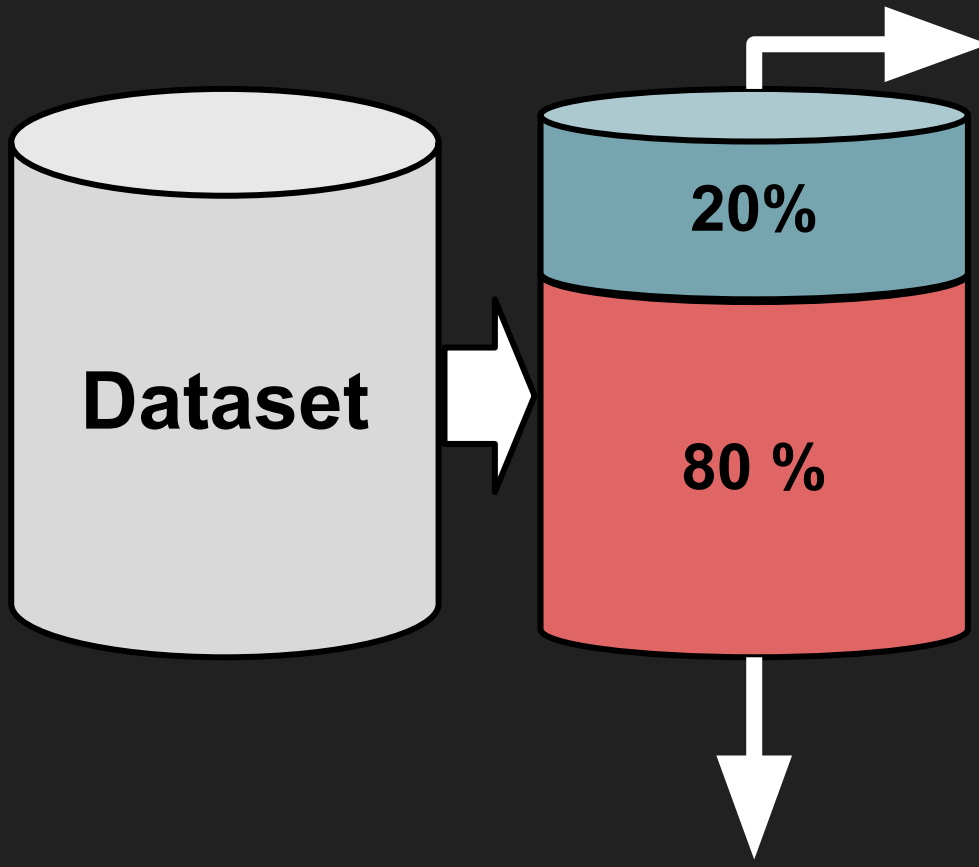
Accuracy on training set: 100%
Real life: Very bad

Overdoing it (overfitting): Classification



Accuracy on training set: 100%
Real life: **Very bad**

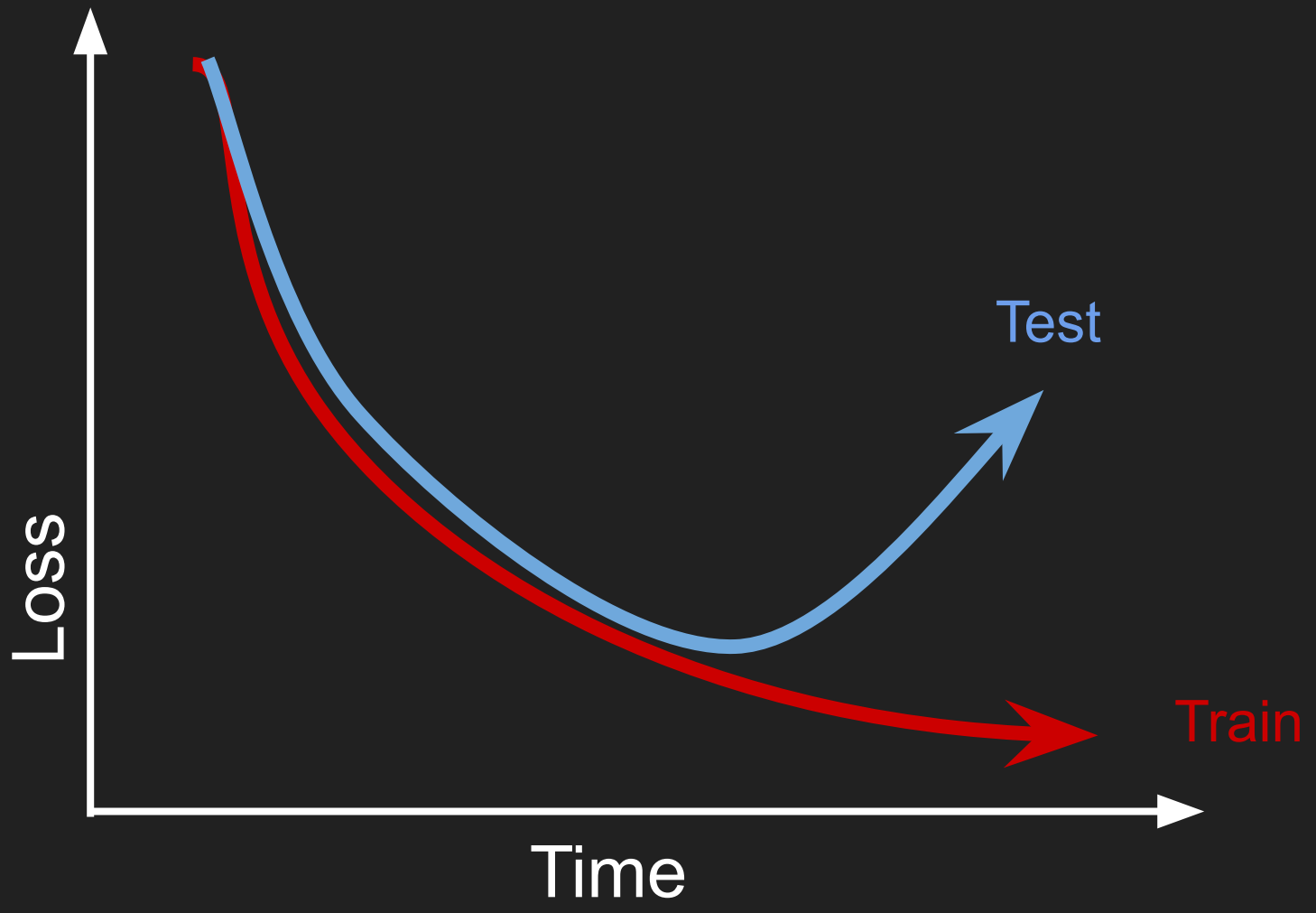
Evaluation: Train / Test split



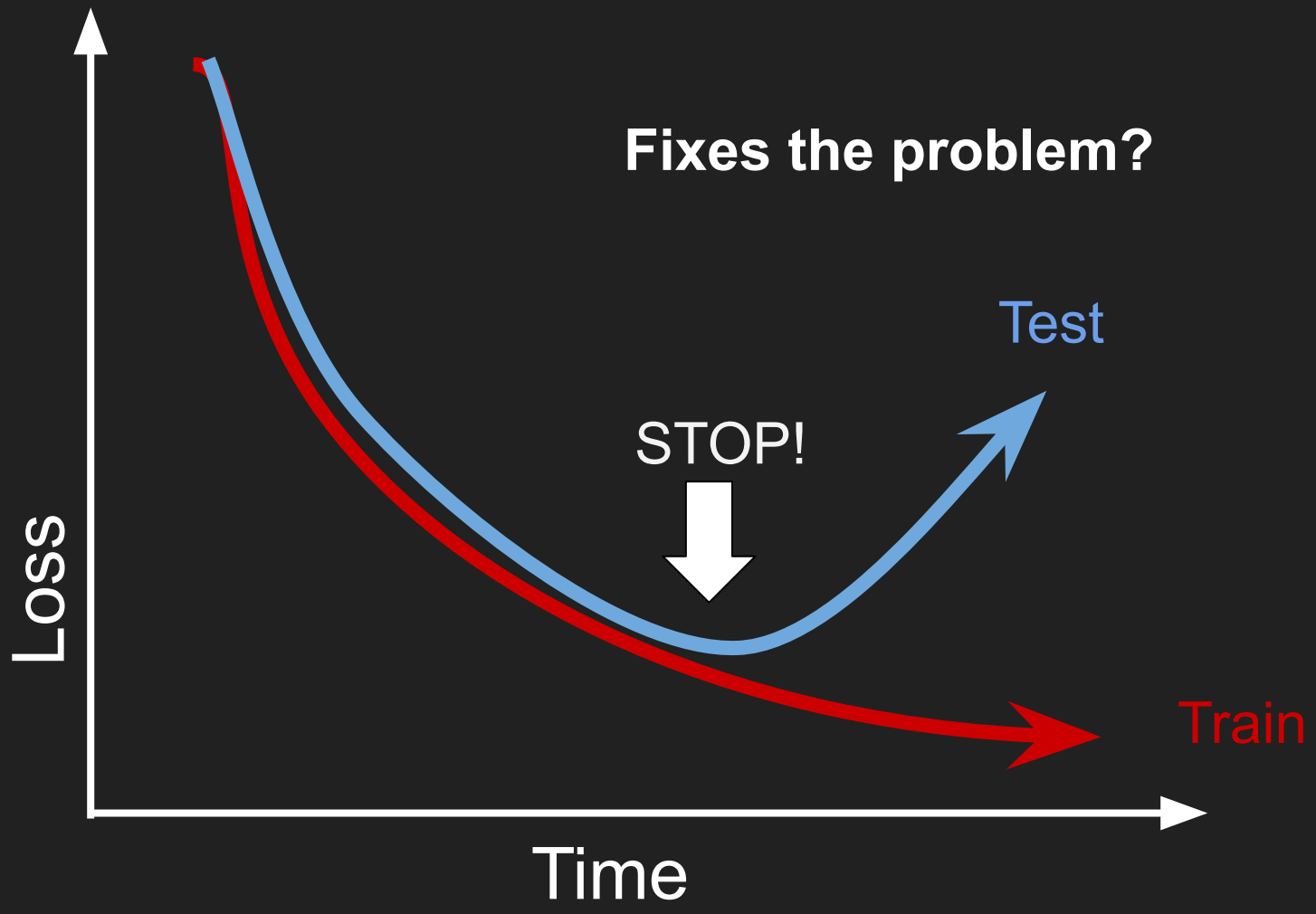
Test

- **Never** used for training**!**
- Mesure model performances on real life data

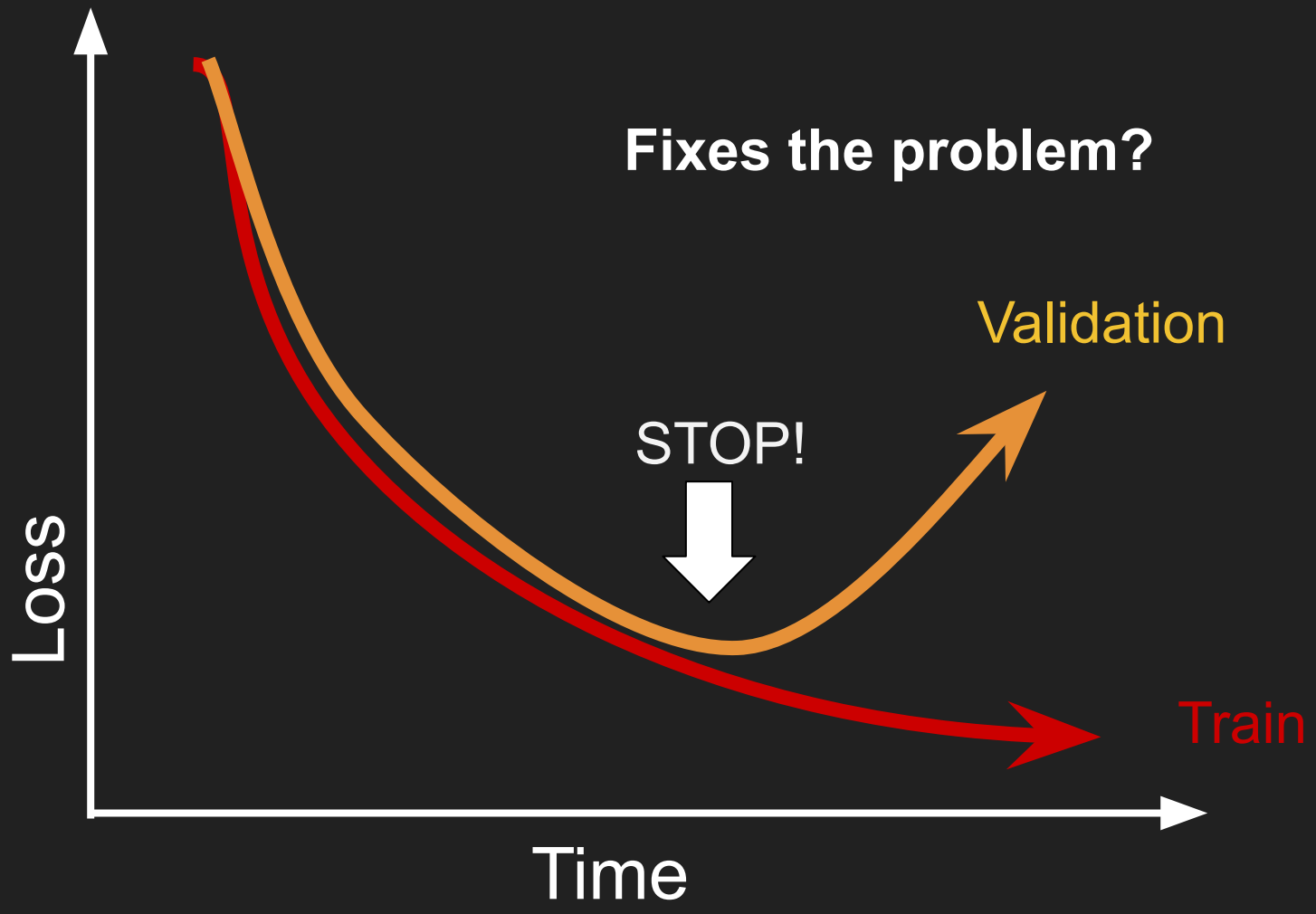
Spotting overfitting



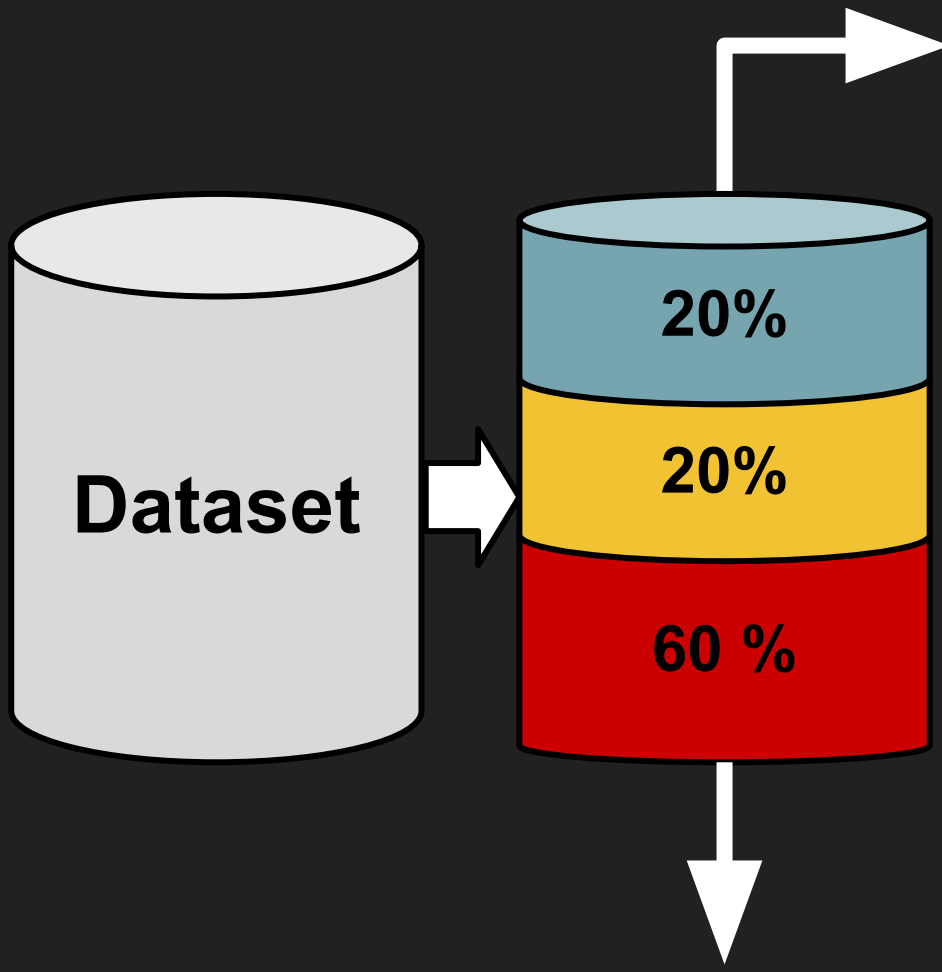
Regularisation: Early stopping



Regularisation: Early stopping



Train / Test / Validation



Test

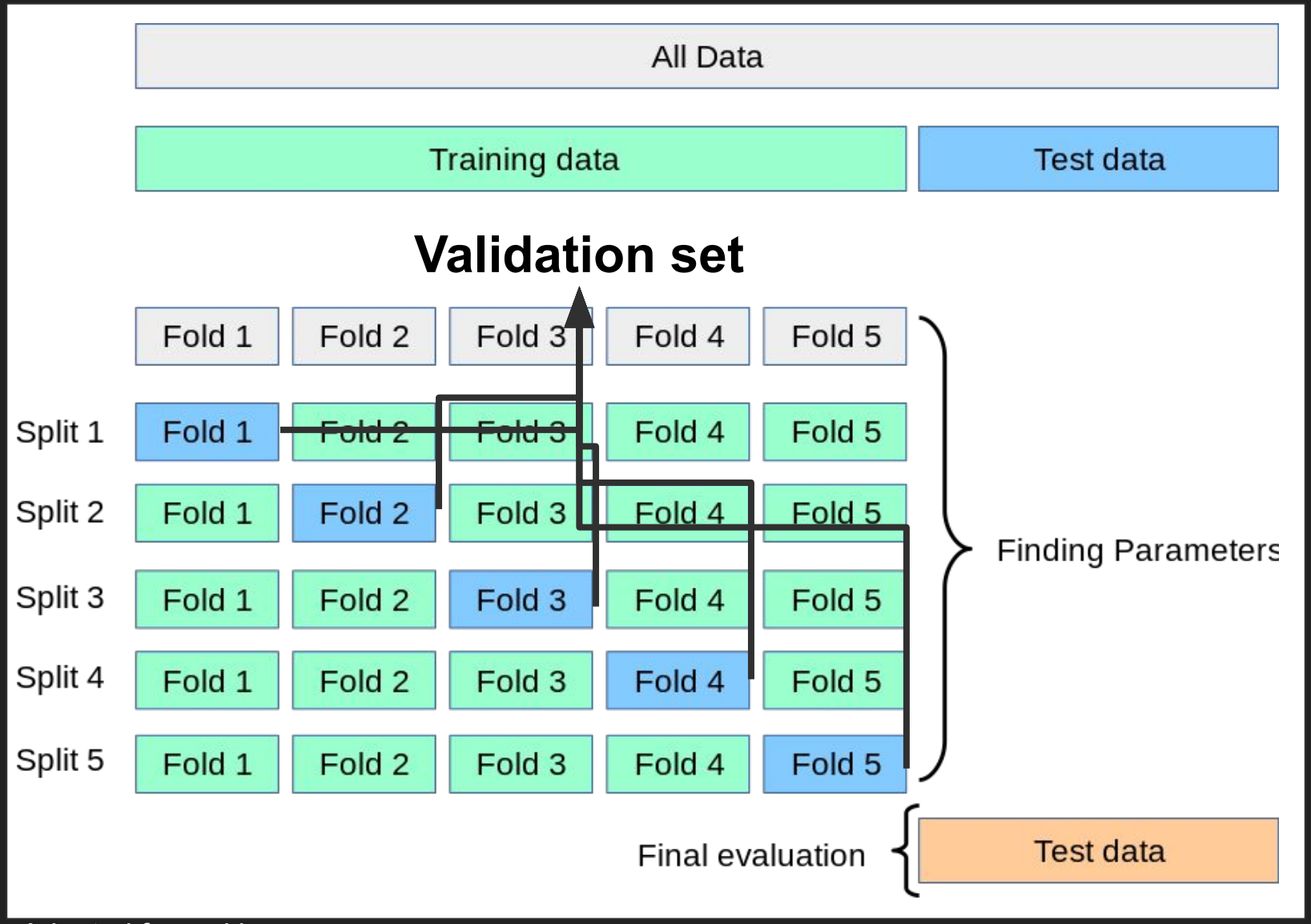
- Mesure model performances

Validation

- Choice of hyper-parameters (Ex: number of layers)

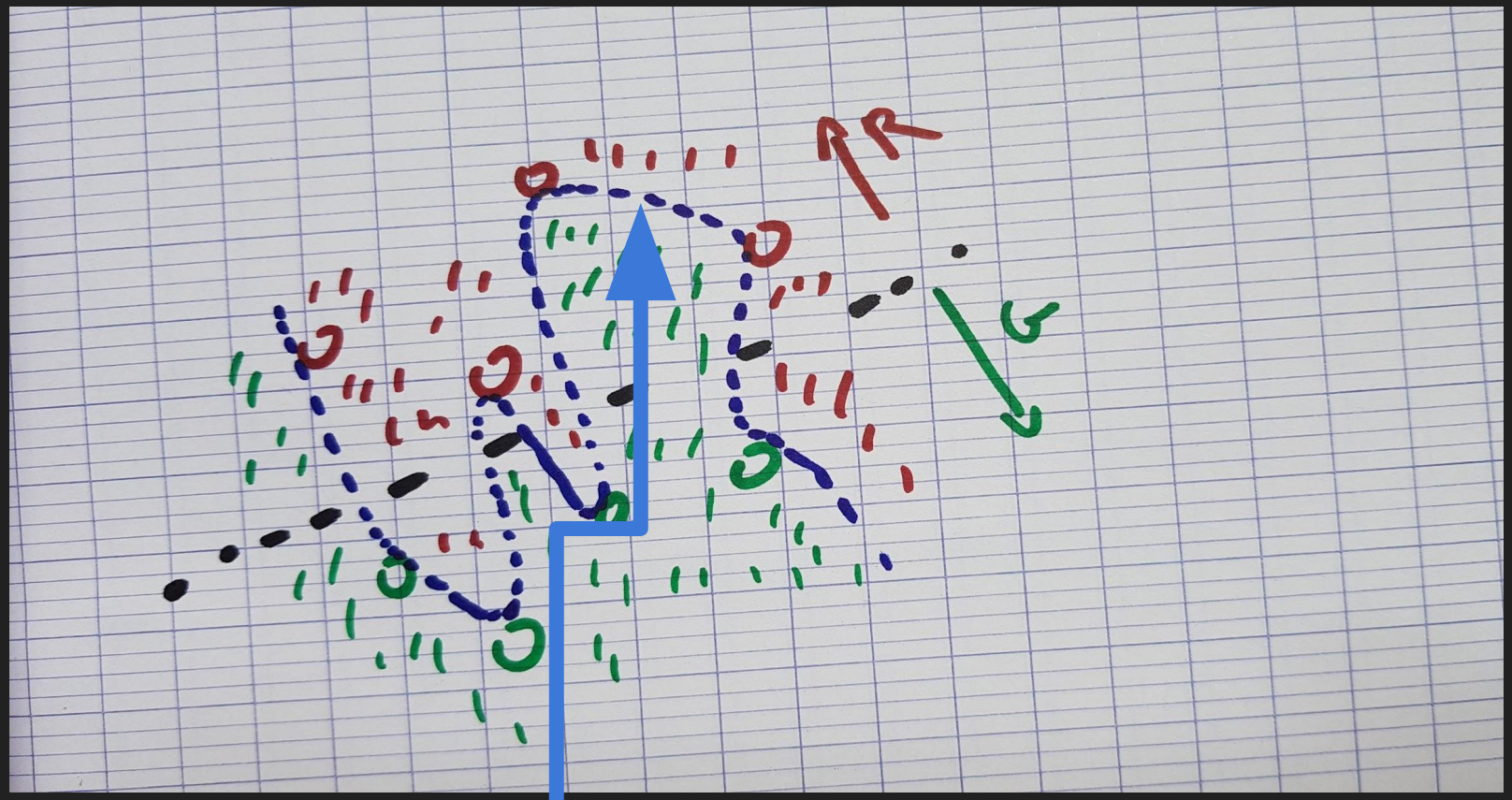
For training **only**!(train)

Cross validation: no separate ‘validation’ set



Adapted from sklearn

Regularisation: limit network capacity



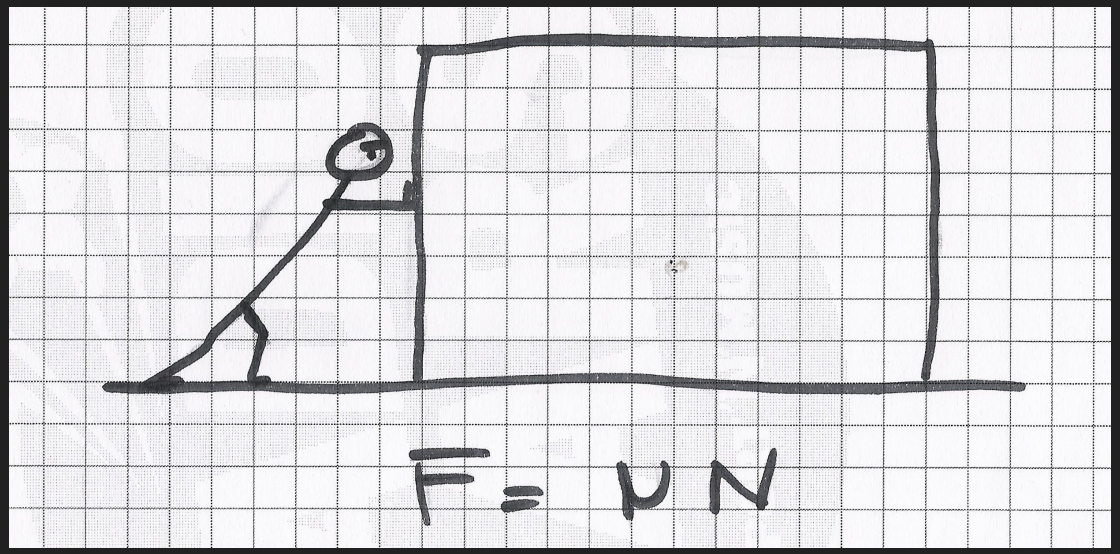
Too much “flex” in weights.
Weights are too high.

Preventing overfitting, Regularisation: L1

MeanSquaredError(Y, Z)

Loss_reg = Loss + **0.0001 * ||W||**

Coefficient



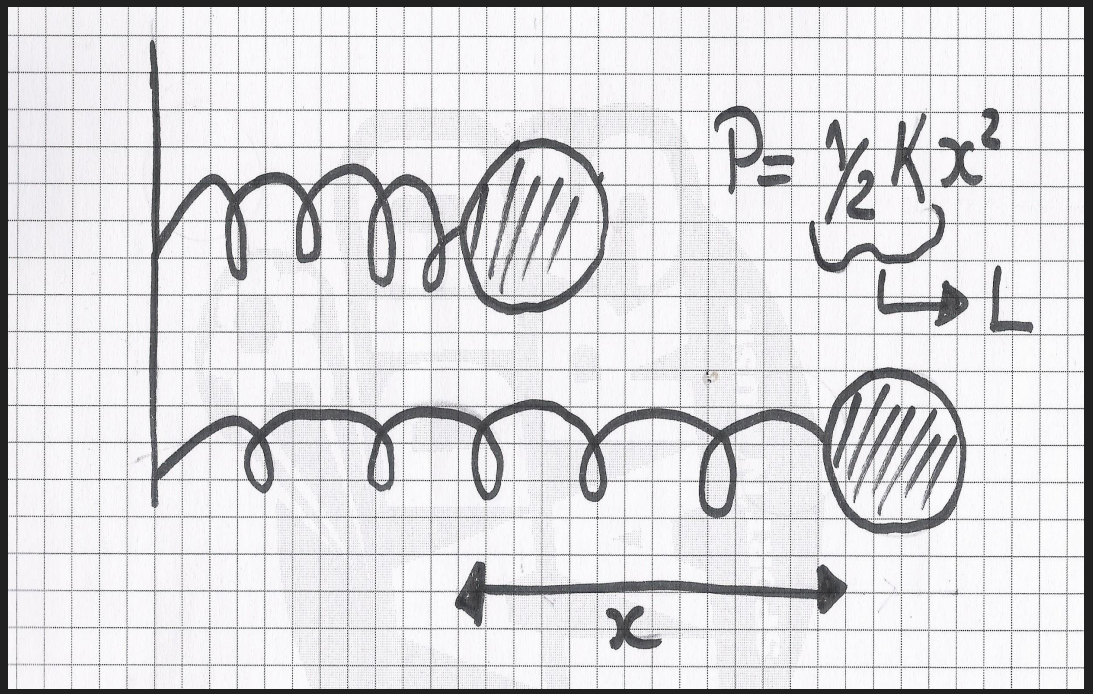
Weights -> 0

Preventing overfitting, Regularisation: L2

MeanSquaredError(Y, Z)

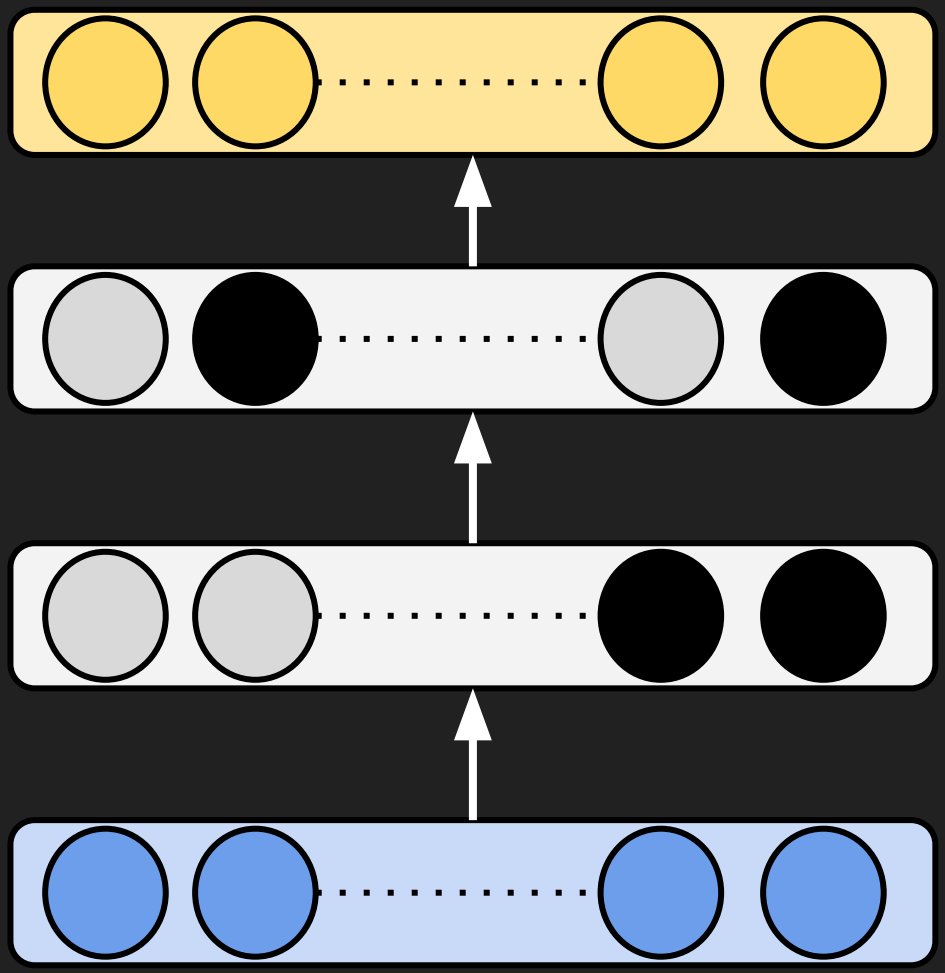
Loss_reg = Loss + **0.00018 * ||W||^2**

Coefficient



Weights -> small

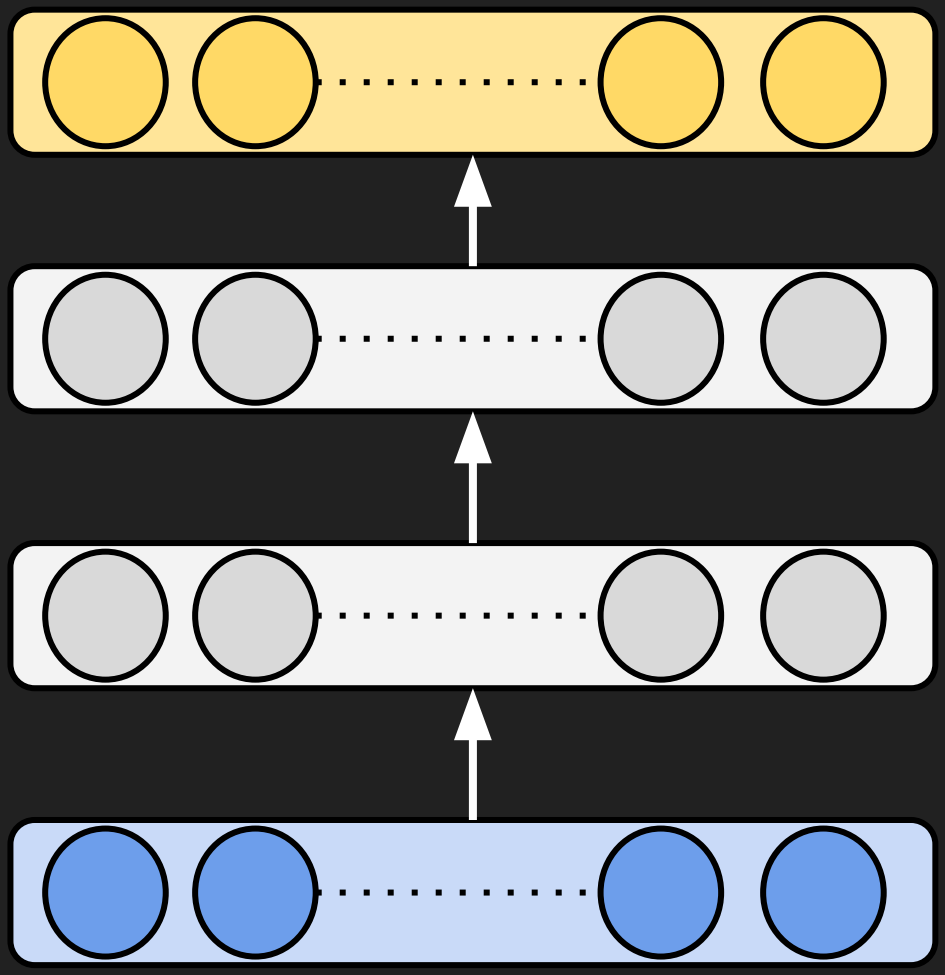
Regularisation: Dropout



- Big network -> A lot of small networks together
- Redundancy between neuron
  - Noise resistance

Artificial neural networks
II

Deep neural networks: A lot of layers

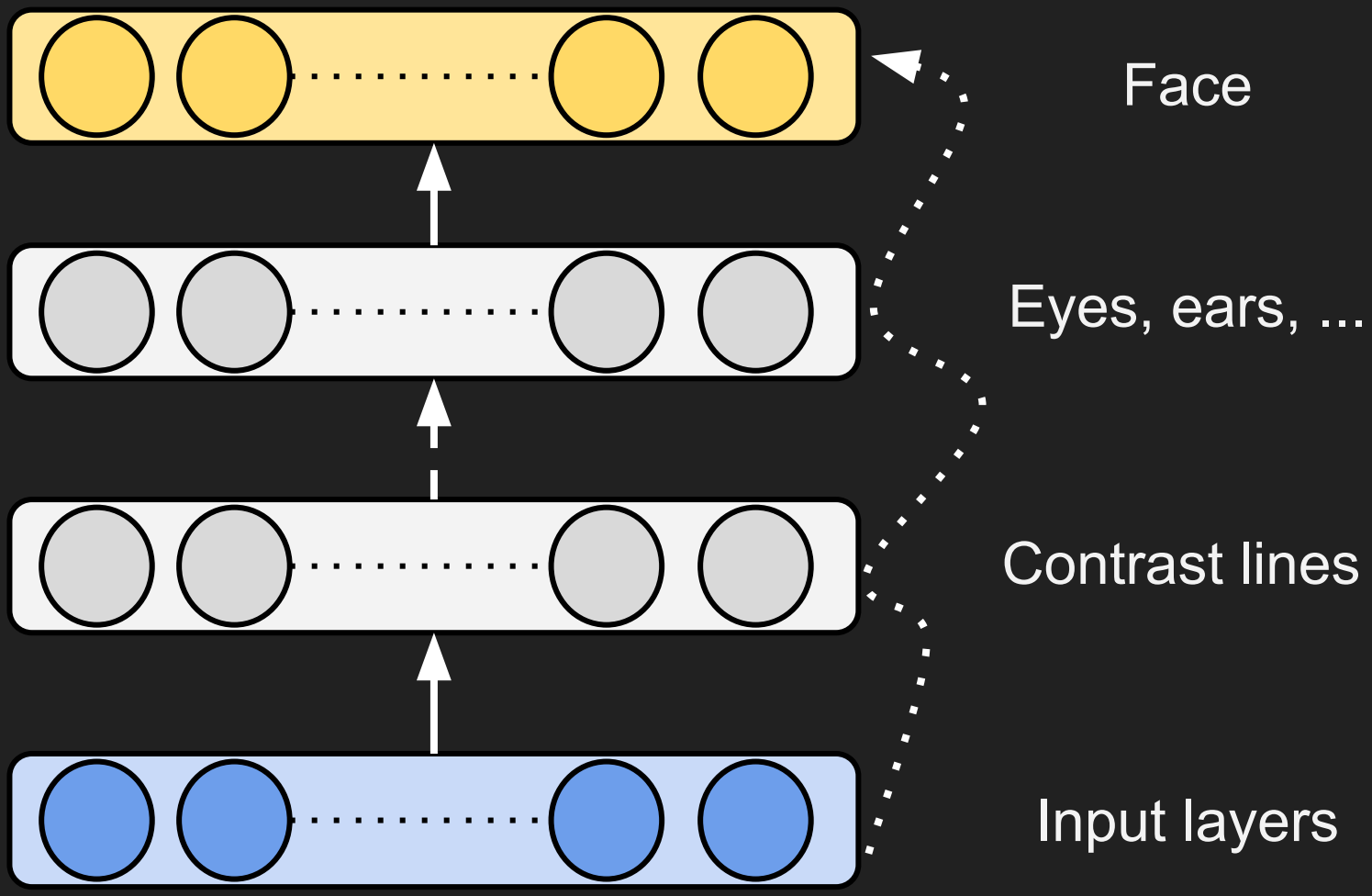


Output layer

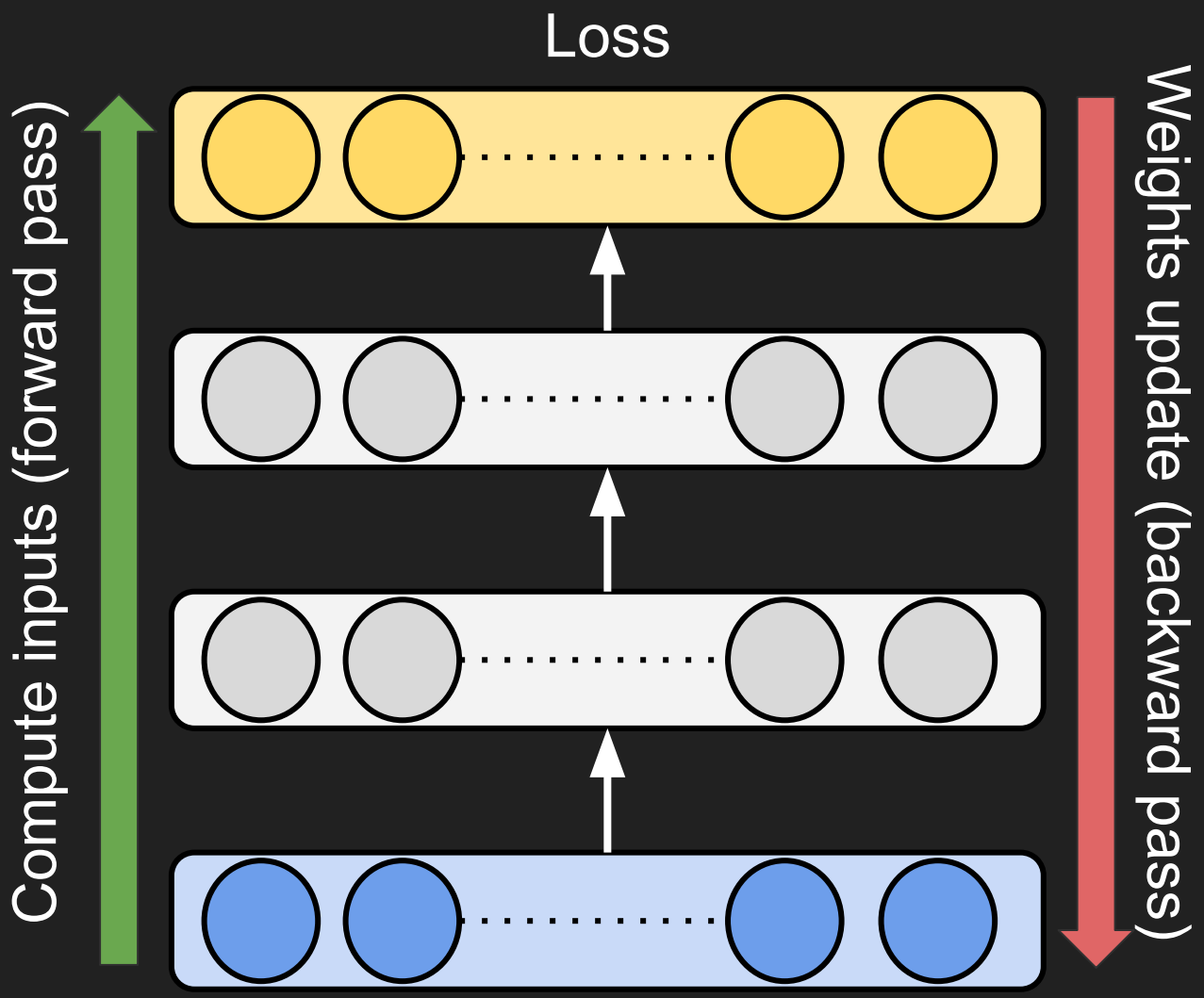
Hidden layers

Input layers

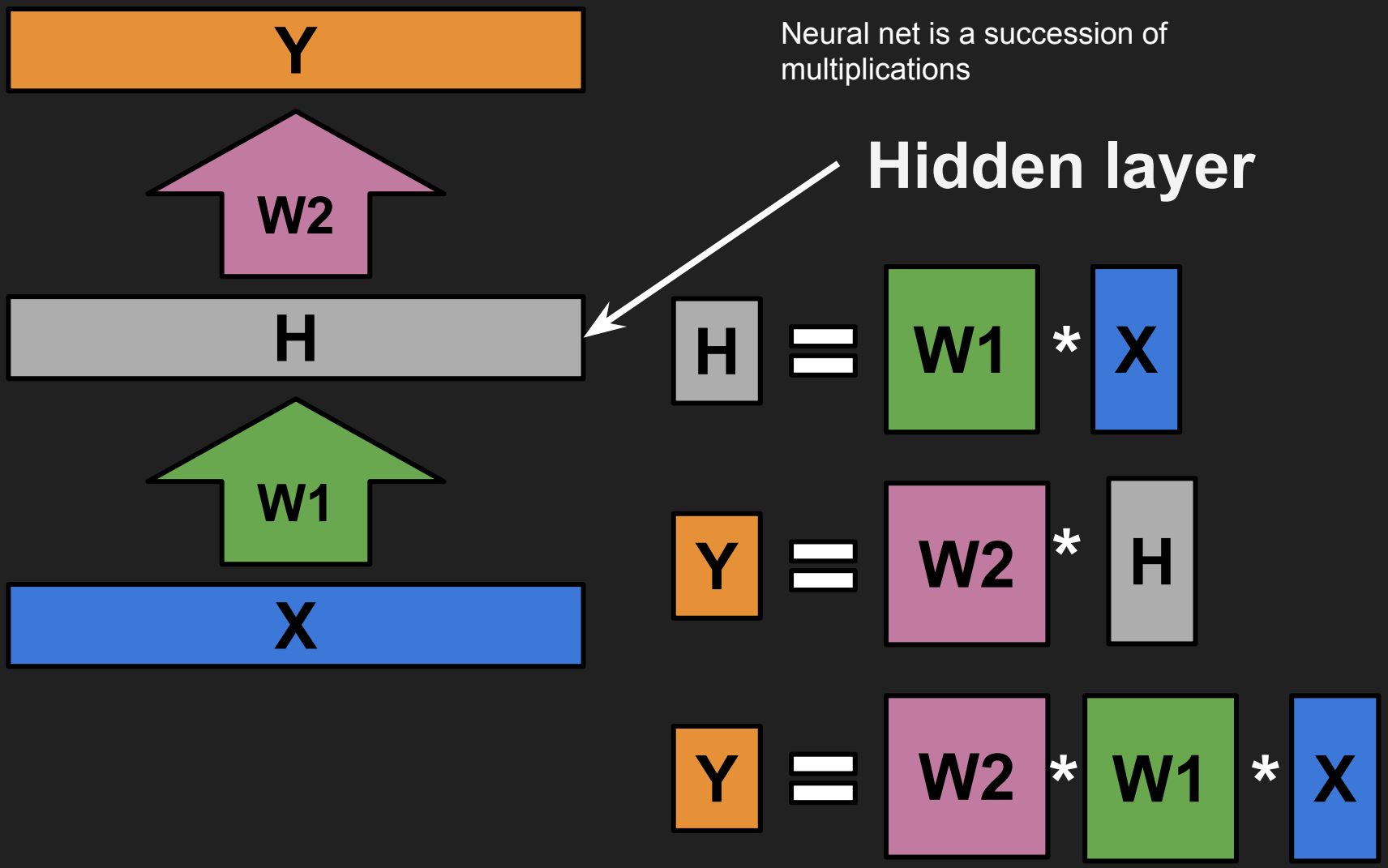
Deep neural networks: Higher levels of abstraction



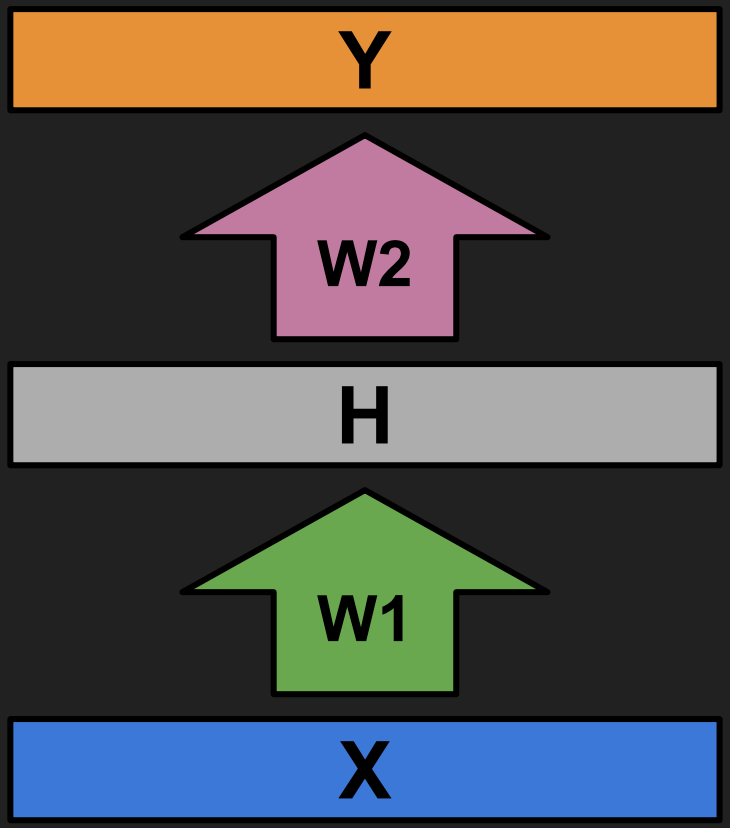
Gradient descent (chain rule)



How networks are represented



Networks as a succession of functions



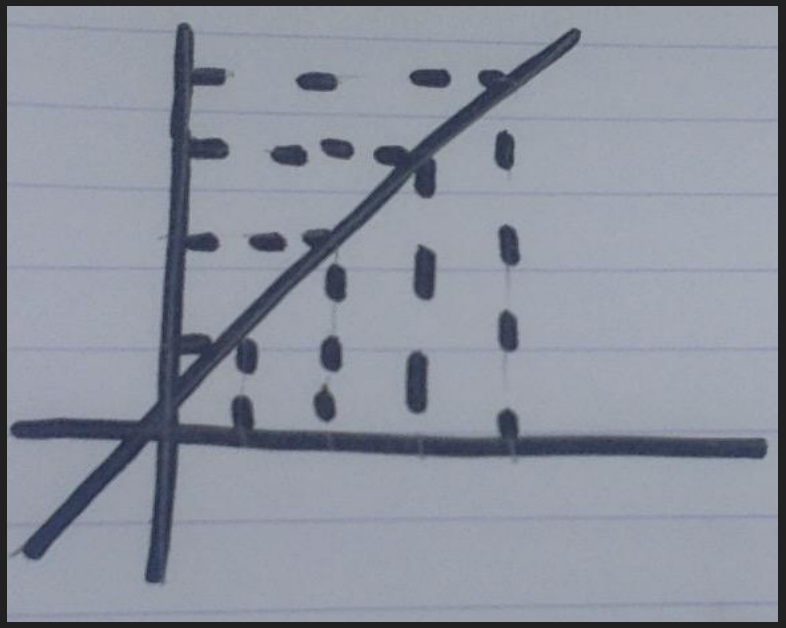
$H = g(x)$

$Y = f(g(x))$

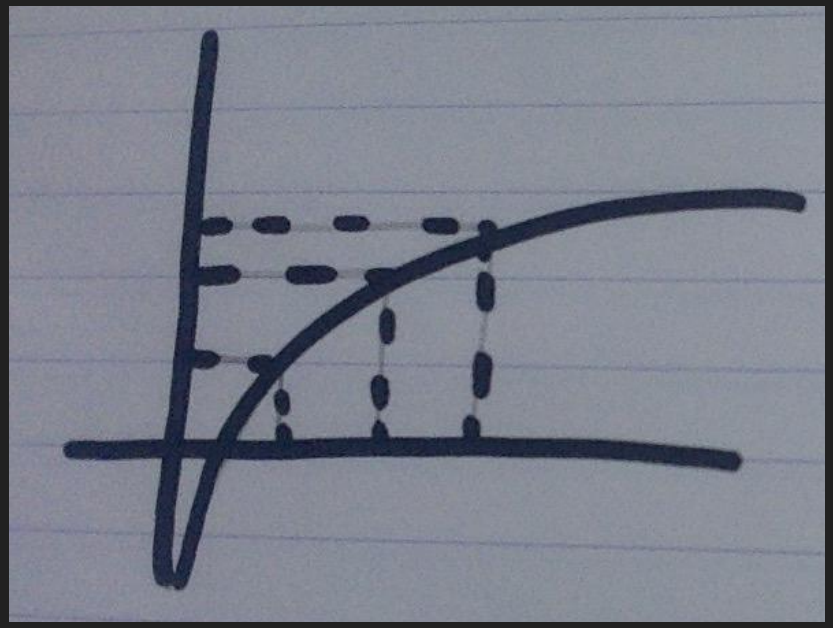
$$\frac{dy}{dw_1} = \frac{dy}{dh} * \frac{dh}{dw_1}$$

$$f'(g(w_1)) = f'(h) * g'(w_1) = f'(g(w_1)) * g'(w_1)$$

Non-linearities

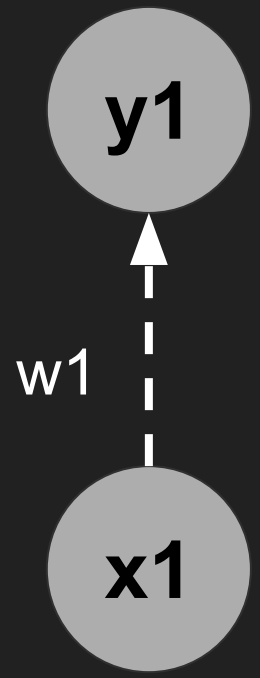


- Y = ax
- Scaling, rotations
- Conserve relationships between points



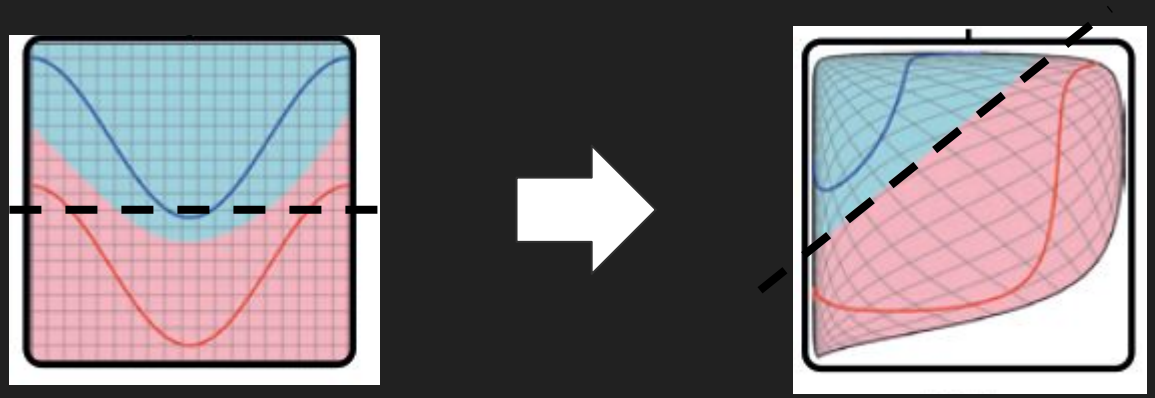
- Change relationships between points
- Space is “bended”
- More flexibility

Why use non-linearities, activation functions

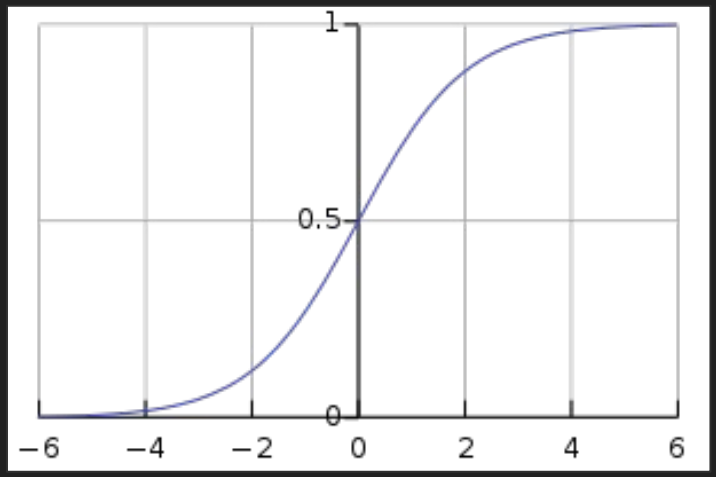


$$y1 = Act(w1 * x1)$$

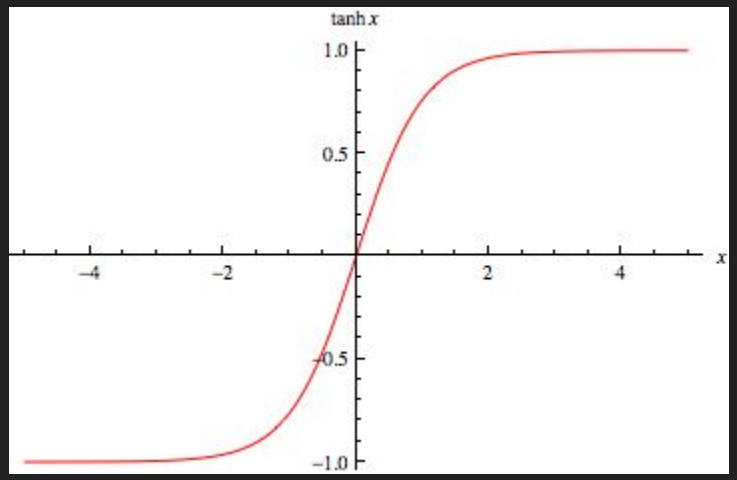
- Act: Activation function
- Act: non-linearity



Common, non-linearities



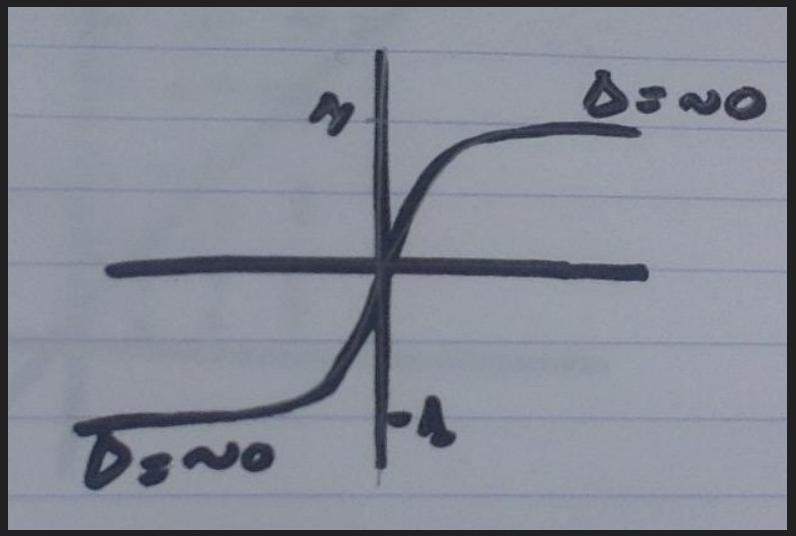
Sigmoid

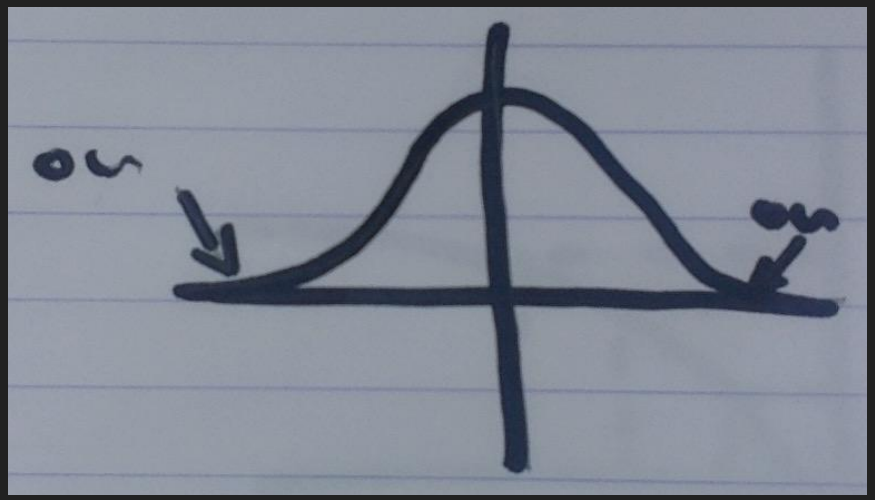


Tanh

y1 = tanh(w1 * x1)

Non-linearities: Saturation





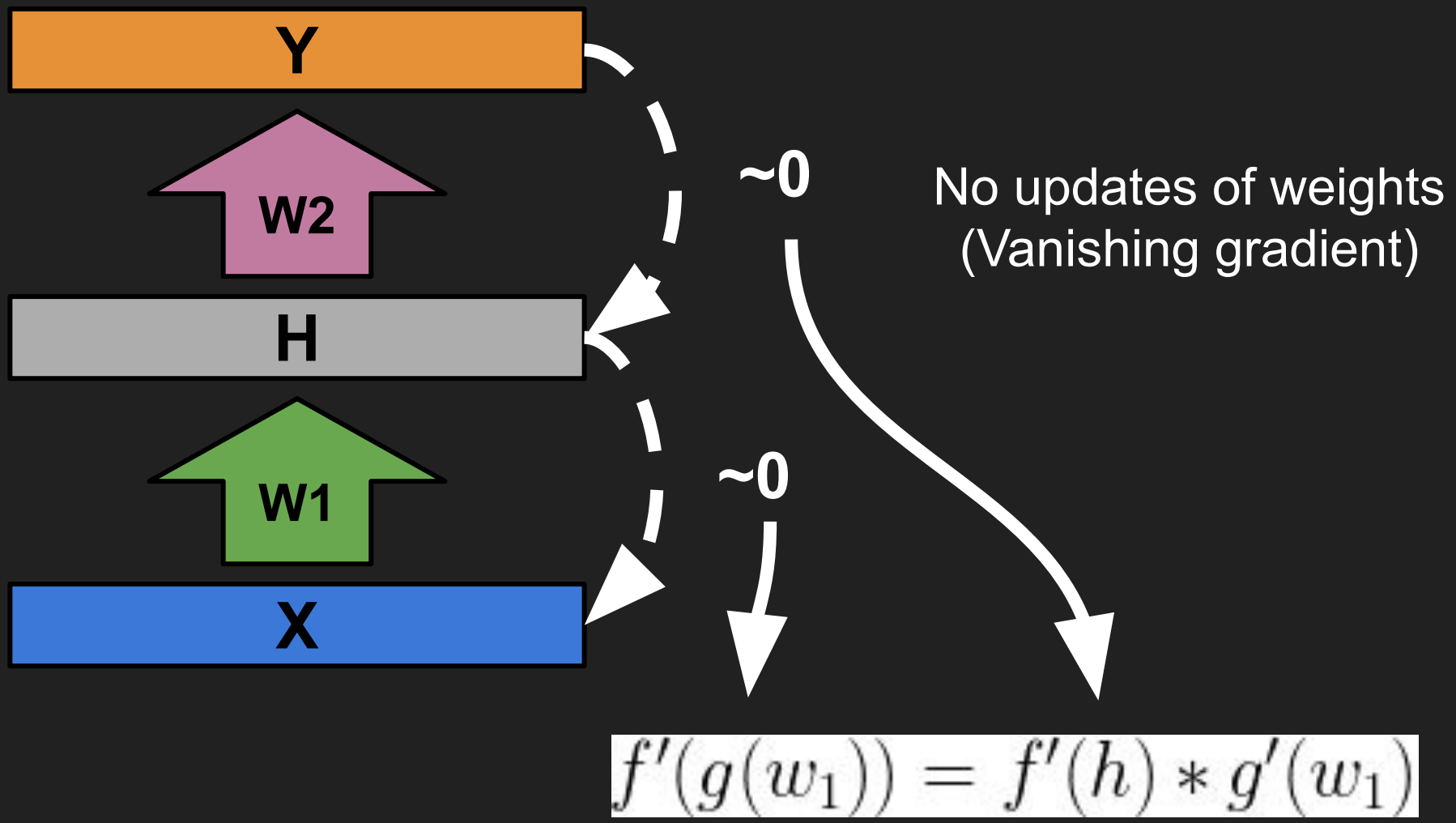
Y close to target

Neuron activation close to 1

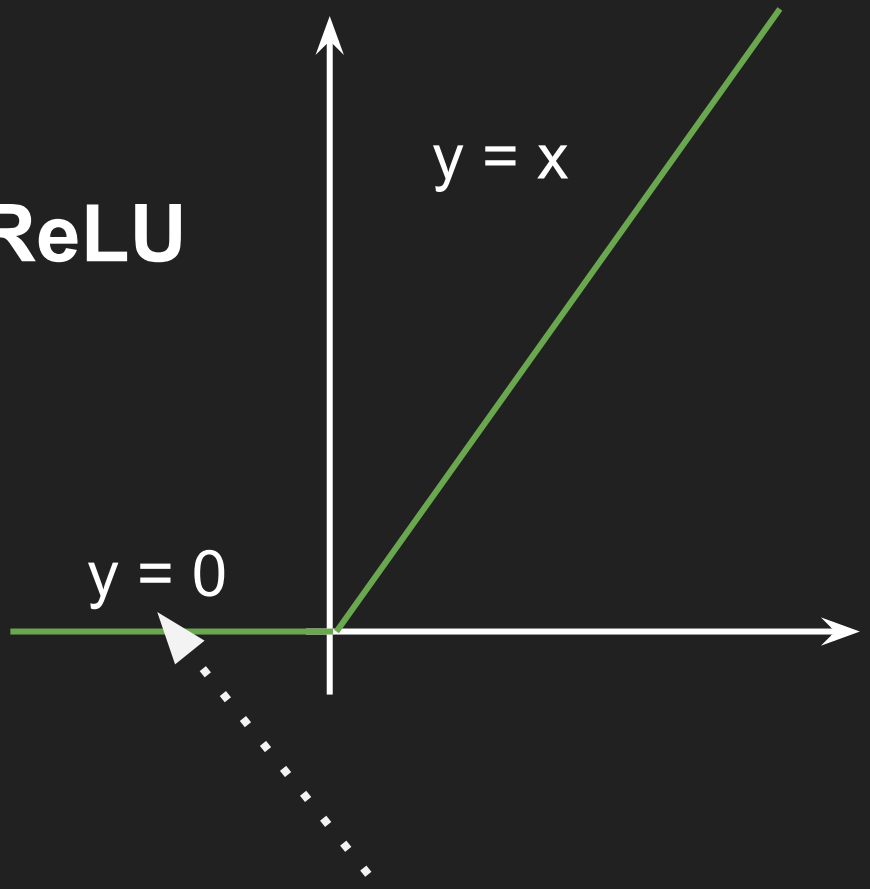
Derivative close to 0

Little update of parameter

Networks as a succession of functions

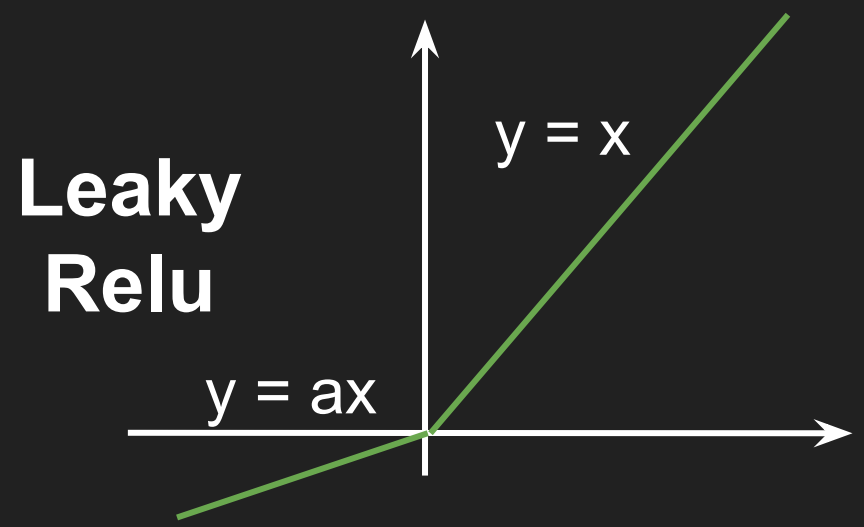


ReLU: Rectified Linear Unit



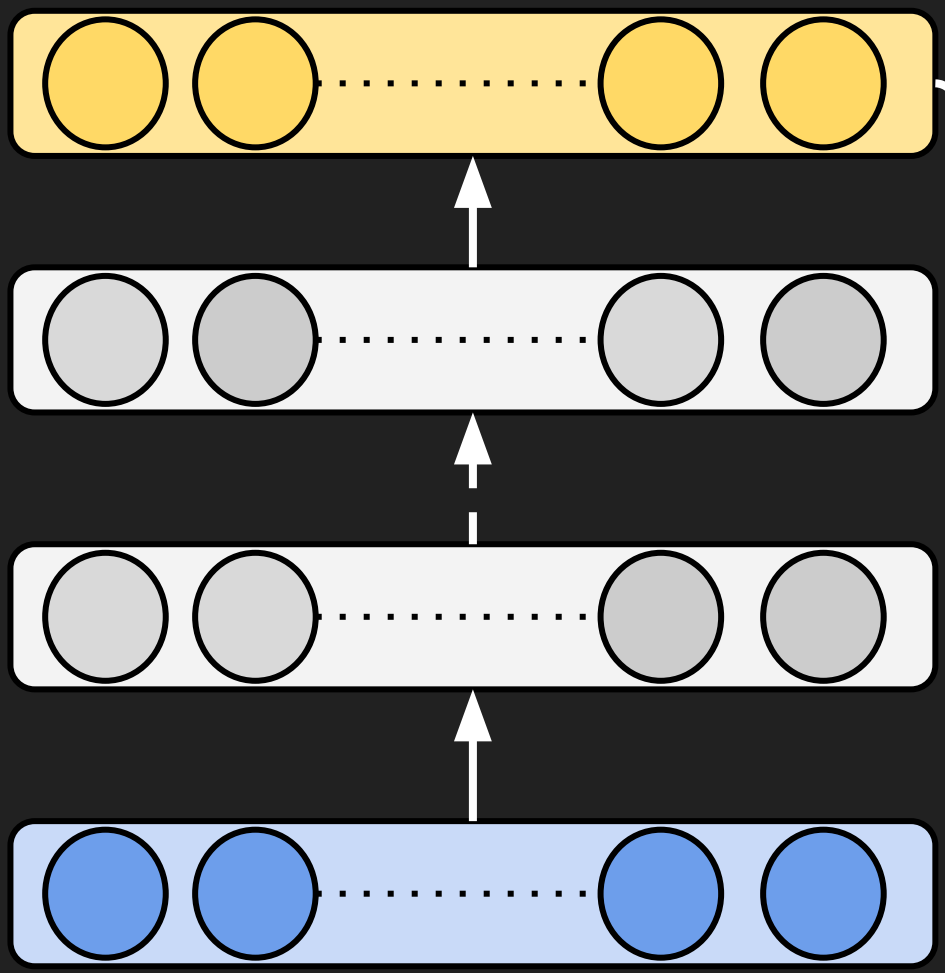
- Death of the neuron

- Fast to compute
- Non-saturating



Multi-class classification (species,…)

Last layer activation

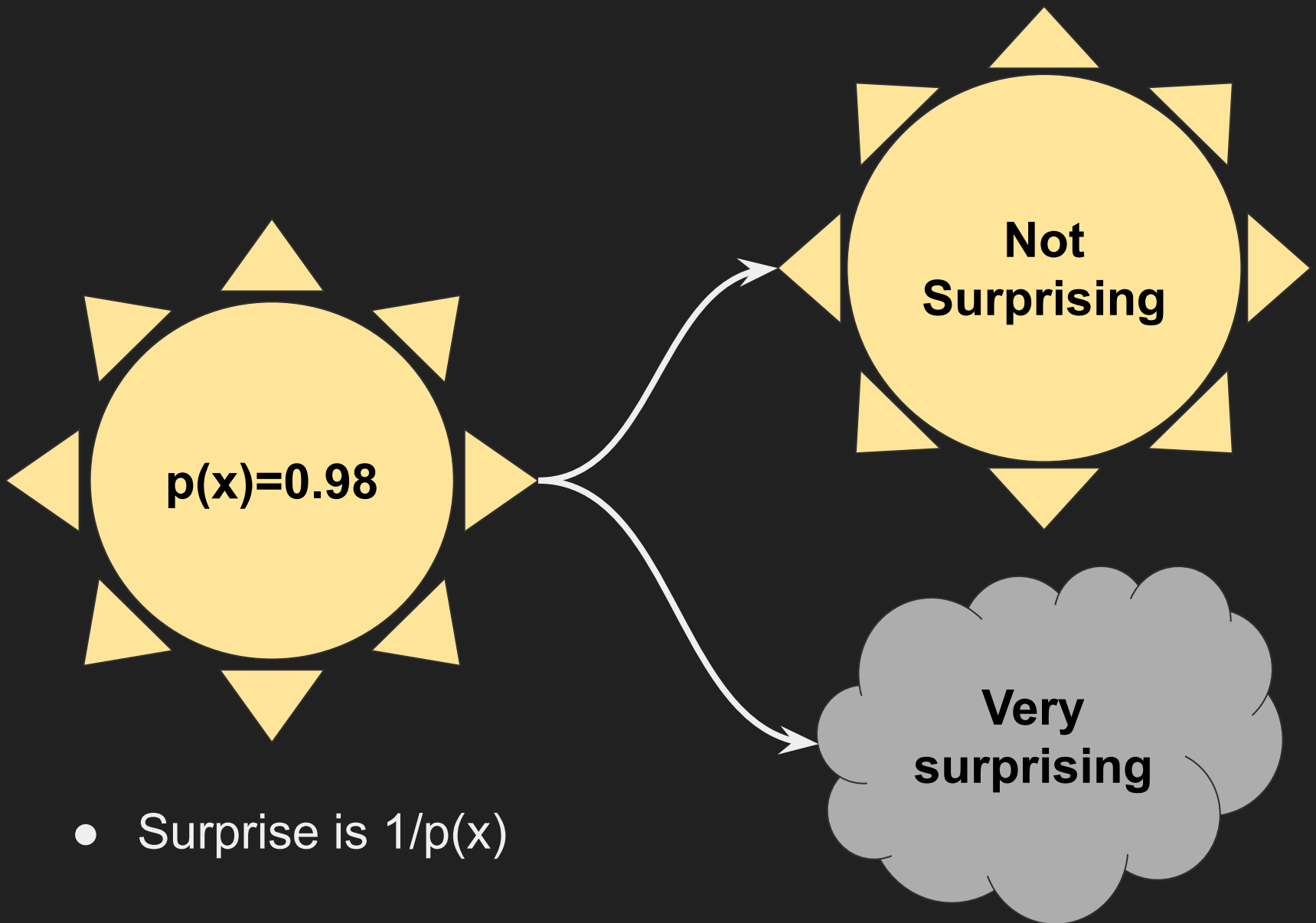


- 10 classes
  - 10 neurons
- Probability
  - Actication: [0, 1]
  - Sum acts = 1
- Softmax

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

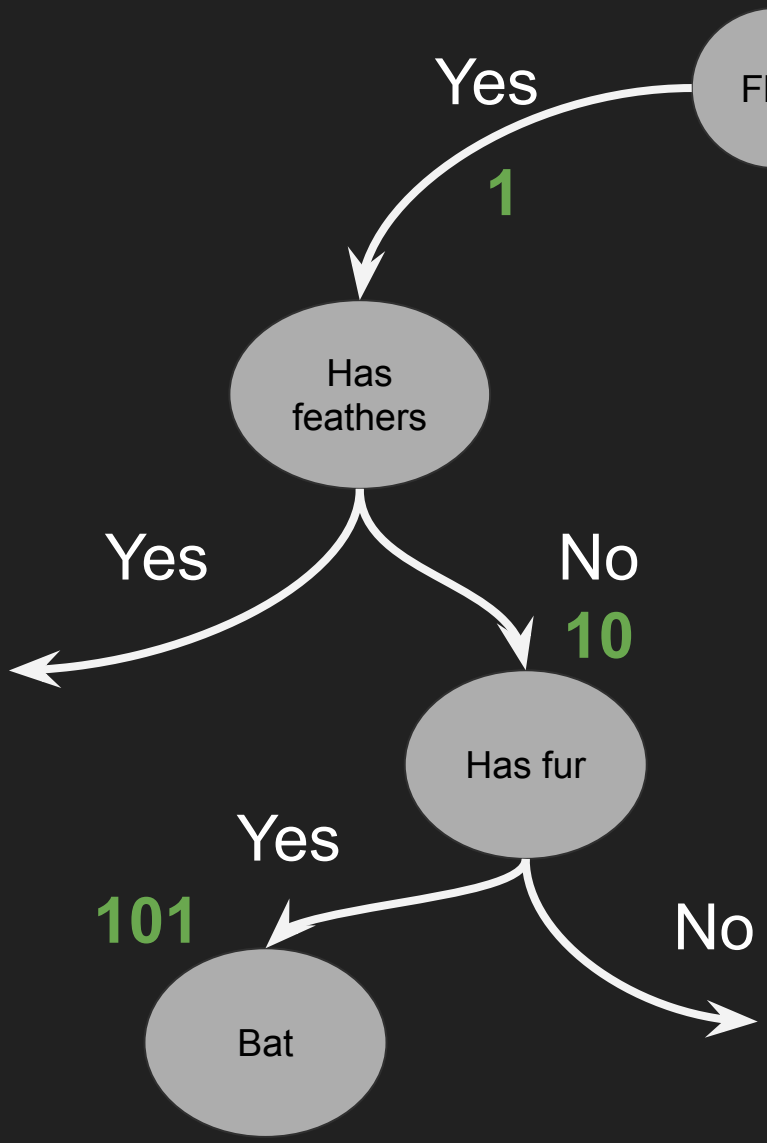
Multi-class classification cost functions

Information theory: Surprise



- Surprise is 1/p(x)

Information theory: Quantity of information



- Bat: 101 (5, decimal)
  - 3 questions
  - Log_2(bat)=3
- Qty of information
  - Nb of yes/no questions

Qty of information in surprise

- Log(1/p(x)) = -Log(p(x)) (> 0)

Average qty of surprise

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

Cost: Cross entropy loss function

True probability for class x

Network's output for class x

$$H(p,q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

The solution

q(x) = p(x)

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

Lowest average surprise when reference is P

A single True answer (class) for every input

True probability for class x

1 for x, 0 for all others

Network's output for class x

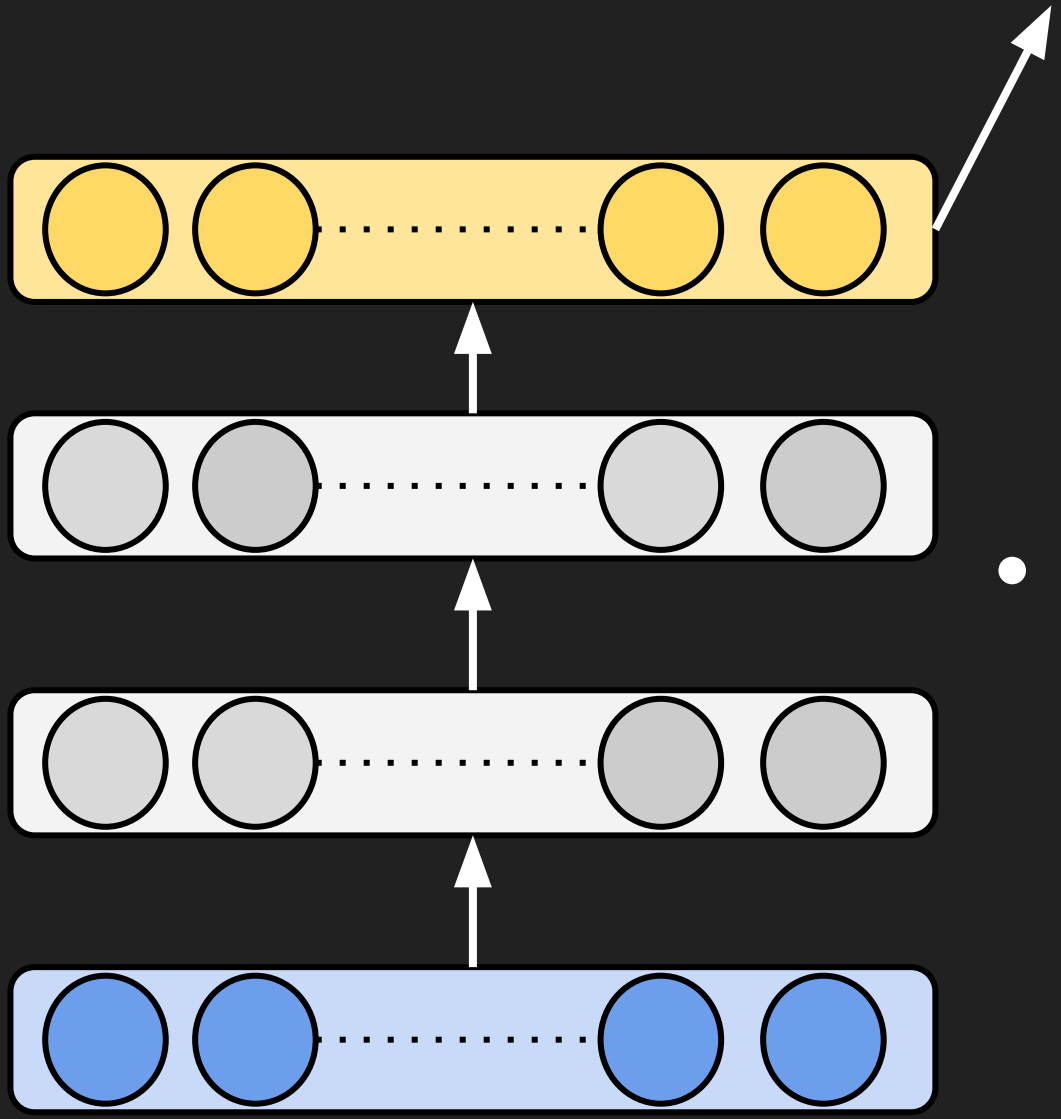
$$H(p,q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

Everything cancels out!

H(p, q) = -log(q(x))

Negative log-likelihood

Multi-class classification



- Activation: Softmax

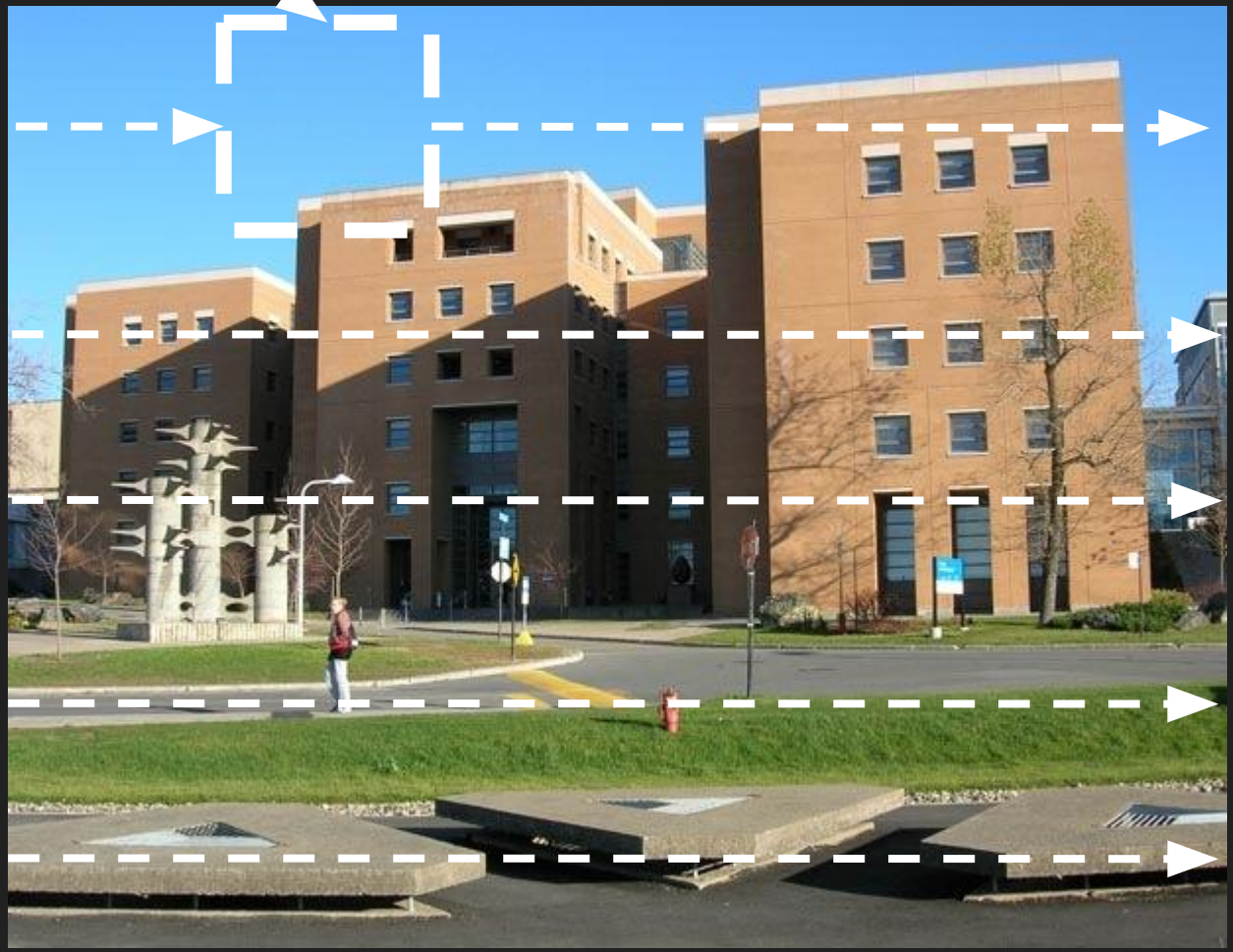
$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

- Loss
  - One right answer
    - Negative Log Likelihood
  - More than one
    - Cross entropy

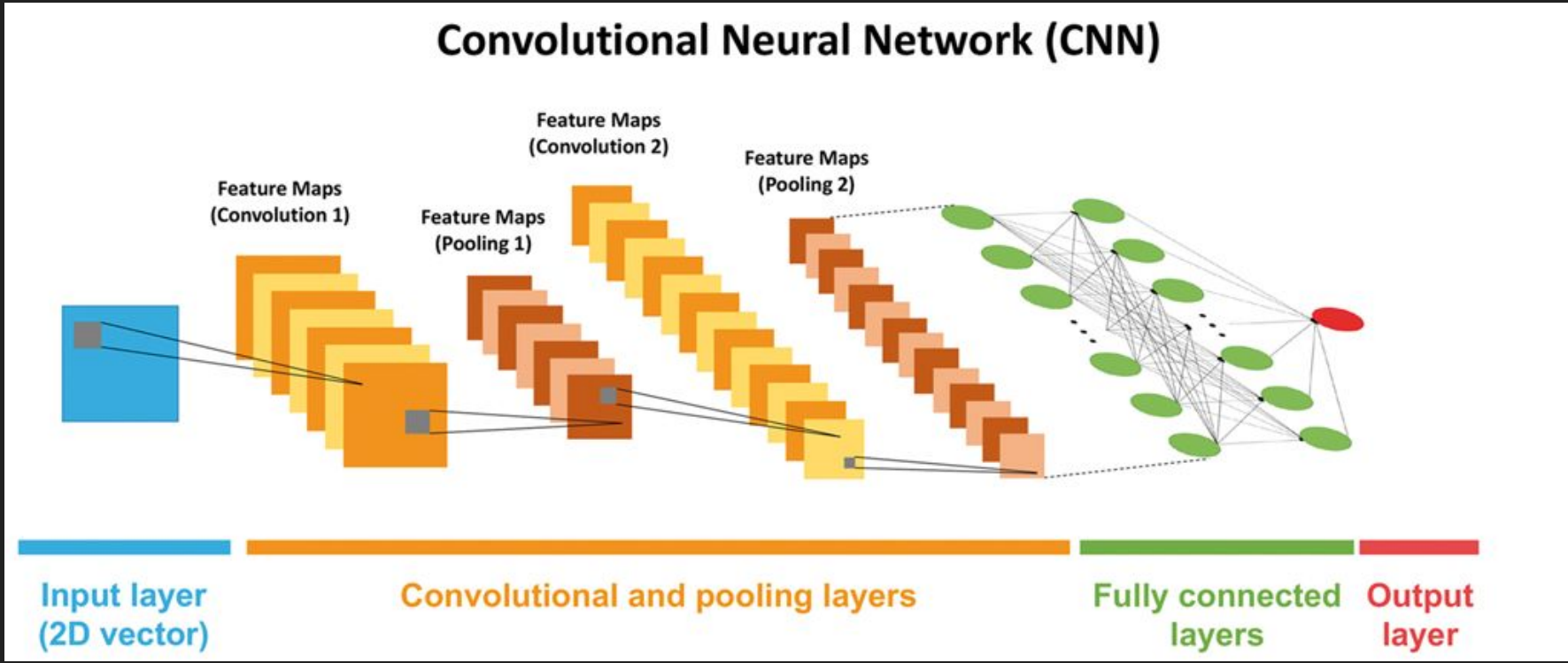
Artificial neural networks
III

Convolutional neural networks

- Small network
  - Filter
- Filters :
  - Small : 3x3
  - Numerous
- Learn patterns

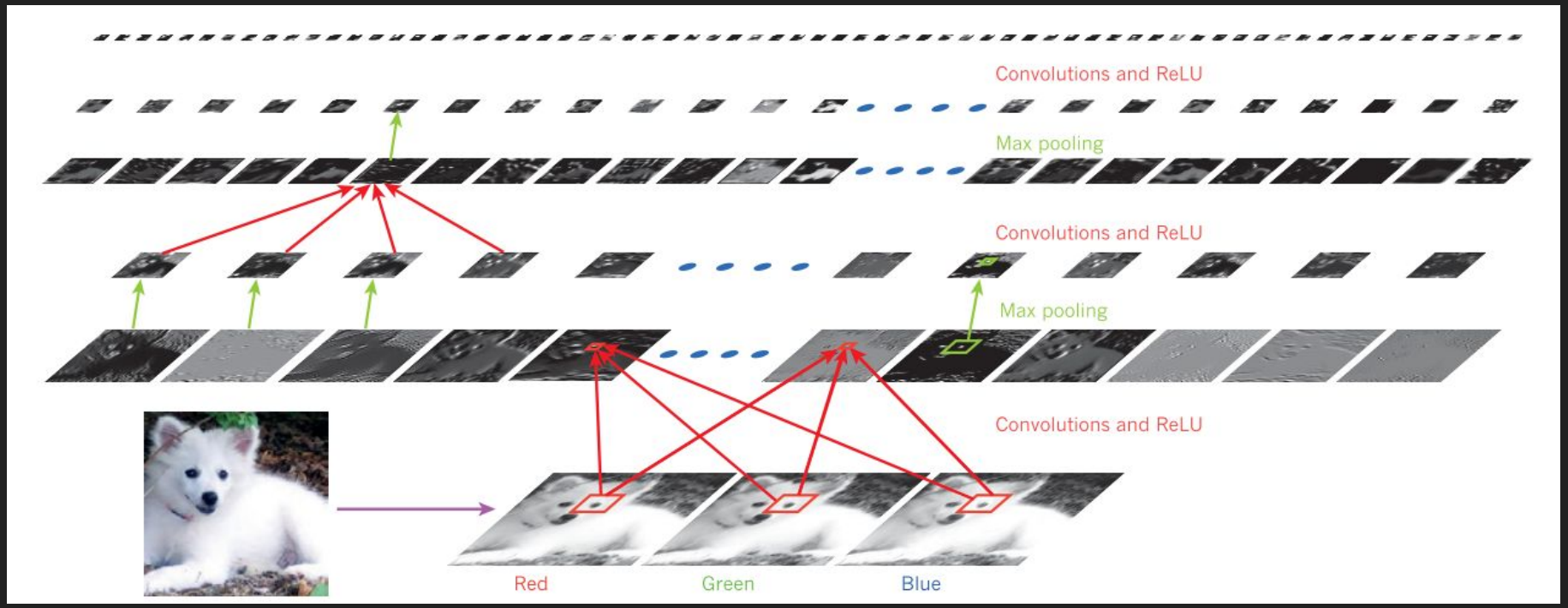


Réseaux de neurones à convolutions classiques



Sureyya Rifaioglu, et al, Brief in Bioinformatics, 2018 https://doi.org/10.1093/bib/bby061

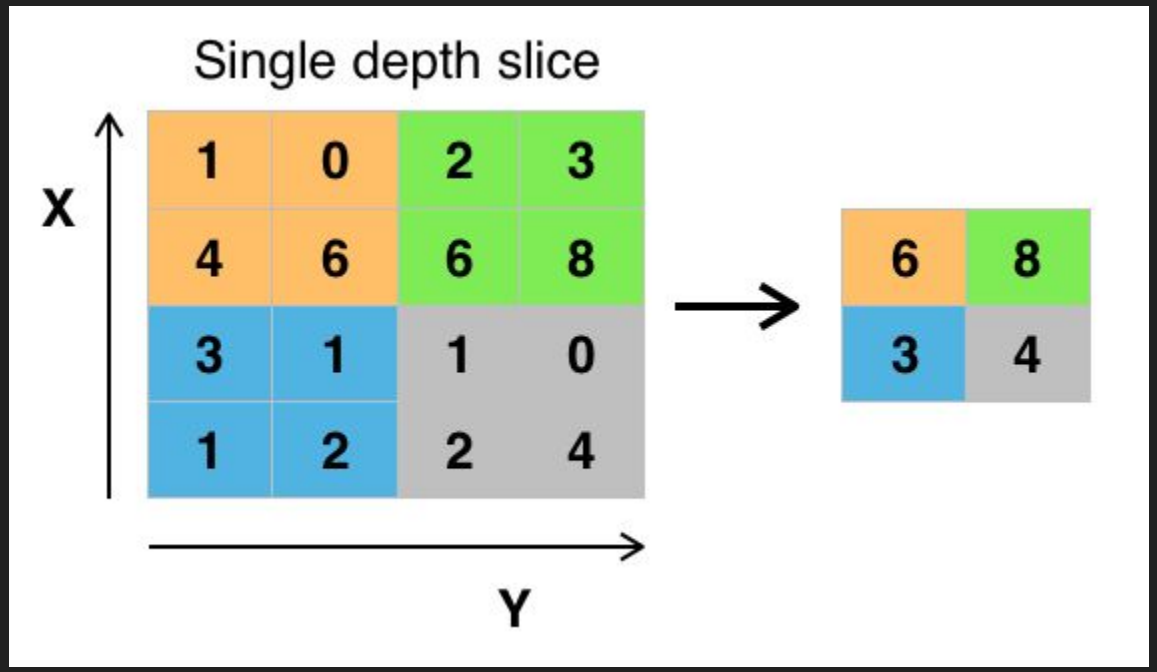
Réseaux de neurones à convolutions classiques



LeCun, Bengio, Hinton.
Nature 2015

Sub-sampling, ex: max-pooling

Aphex34, wikipedia



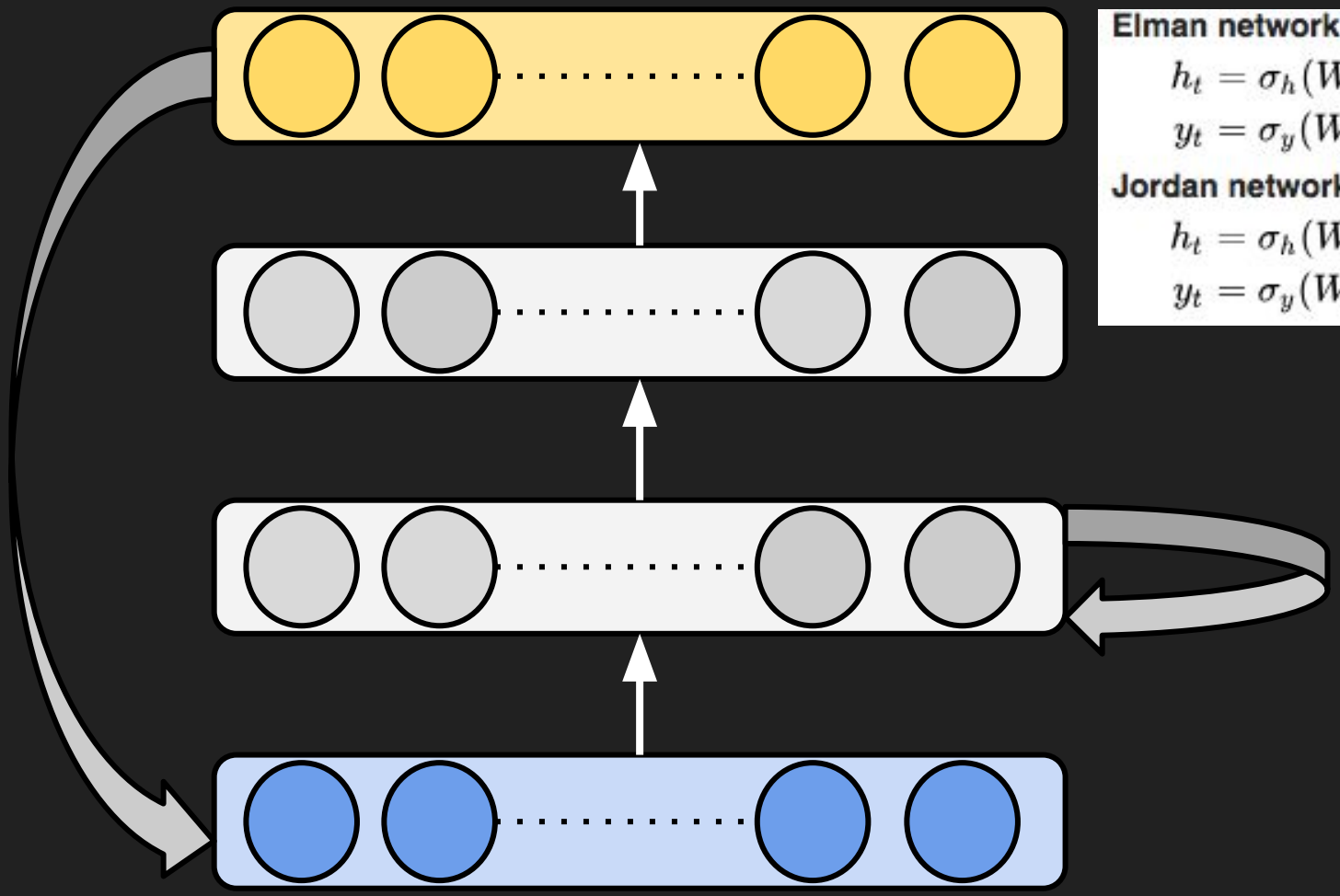
- Reduce number of parameters
- Resistant to small translations

Artificial neural networks
IV

Recurrent neural networks

- Sequences
- Temporal series
- Inputs of variable lengths
- Possess memory
- Can be hard to train (instability)
- Most used variant (arguably): Long Short-Term Memory (LSTM)

Recurrent neural networks



Elman network[19]

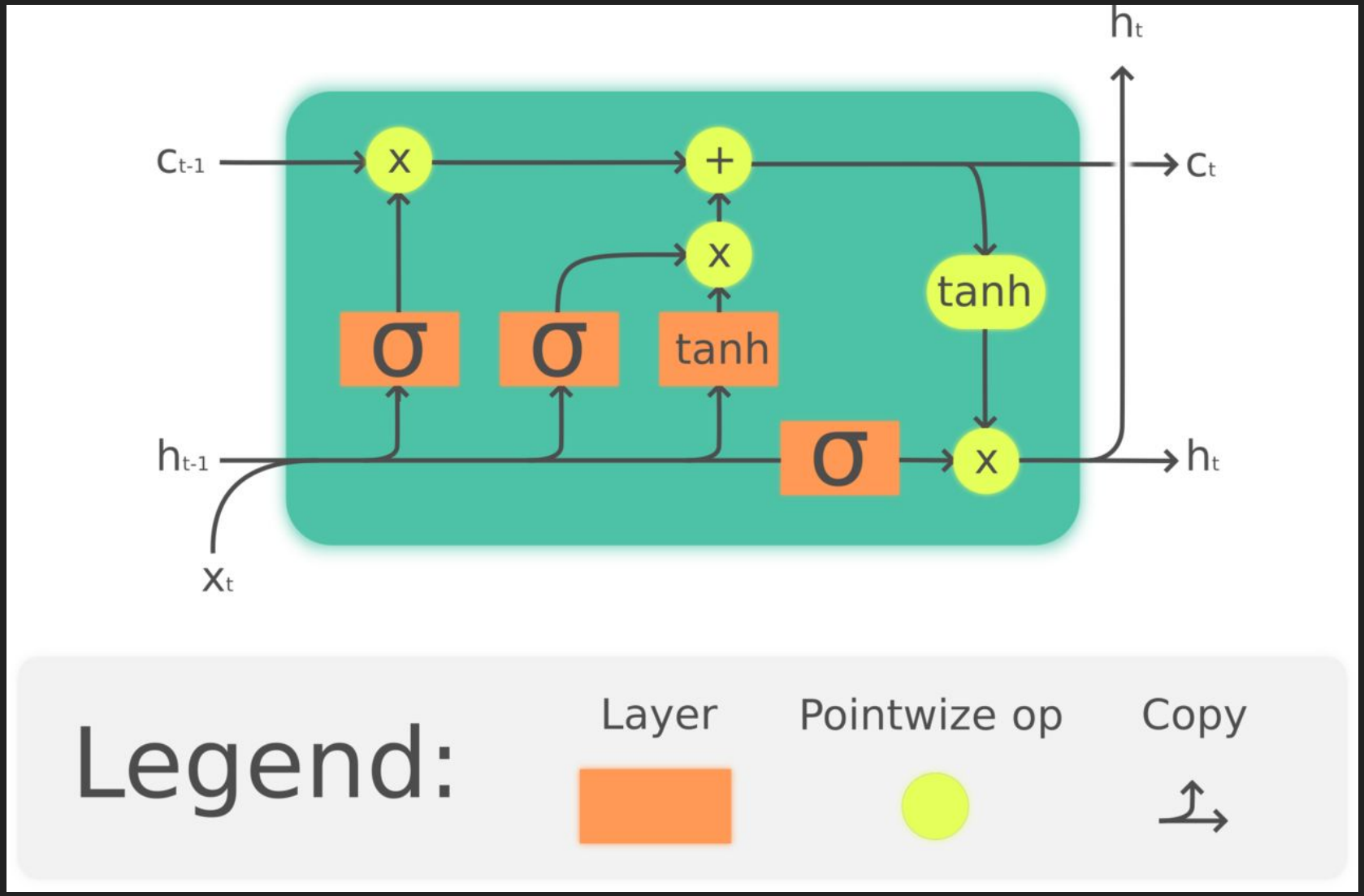
$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Jordan network[20]

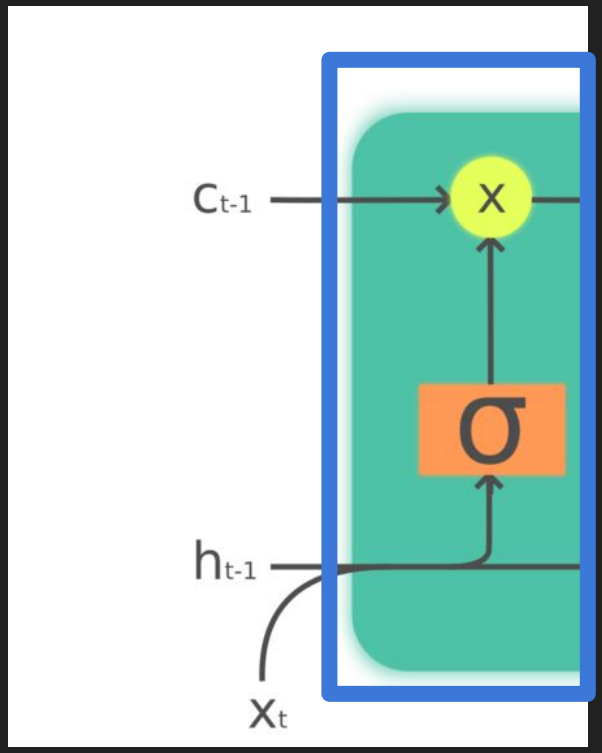
$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

wikipedia

One LSTM unit



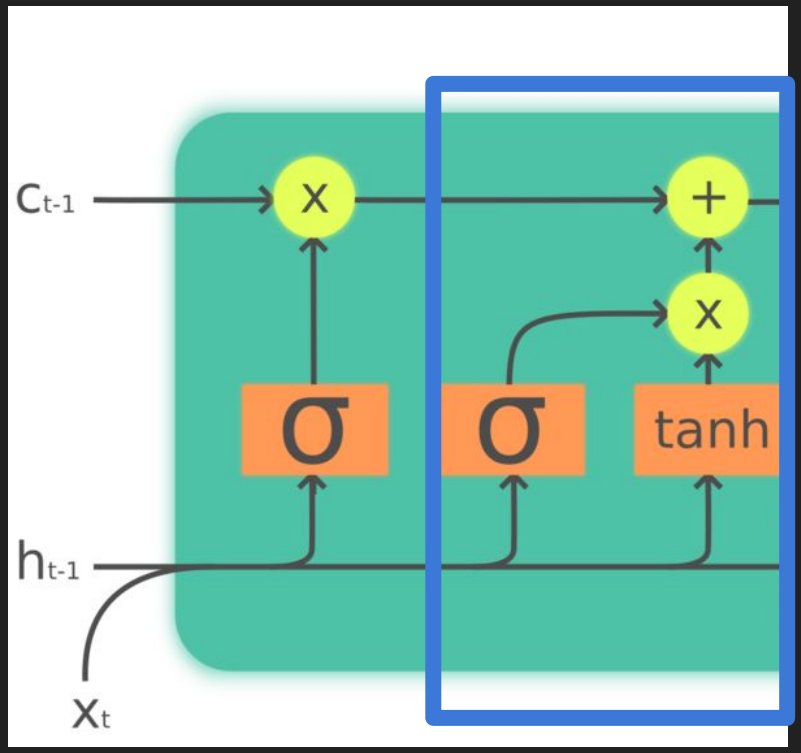
One LSTM unit, forget gate



How much of C_t-1 should be forgotten

- C: cell state
- H: previous output
- X: input

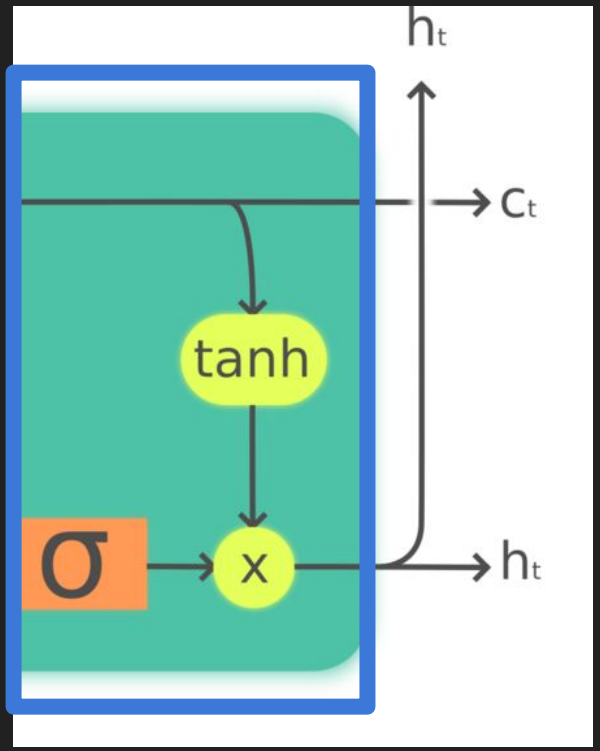
One LSTM unit, input gate



How much of X_t and H_t-1 should be added to state

- C: cell state
- H: previous output
- X: input

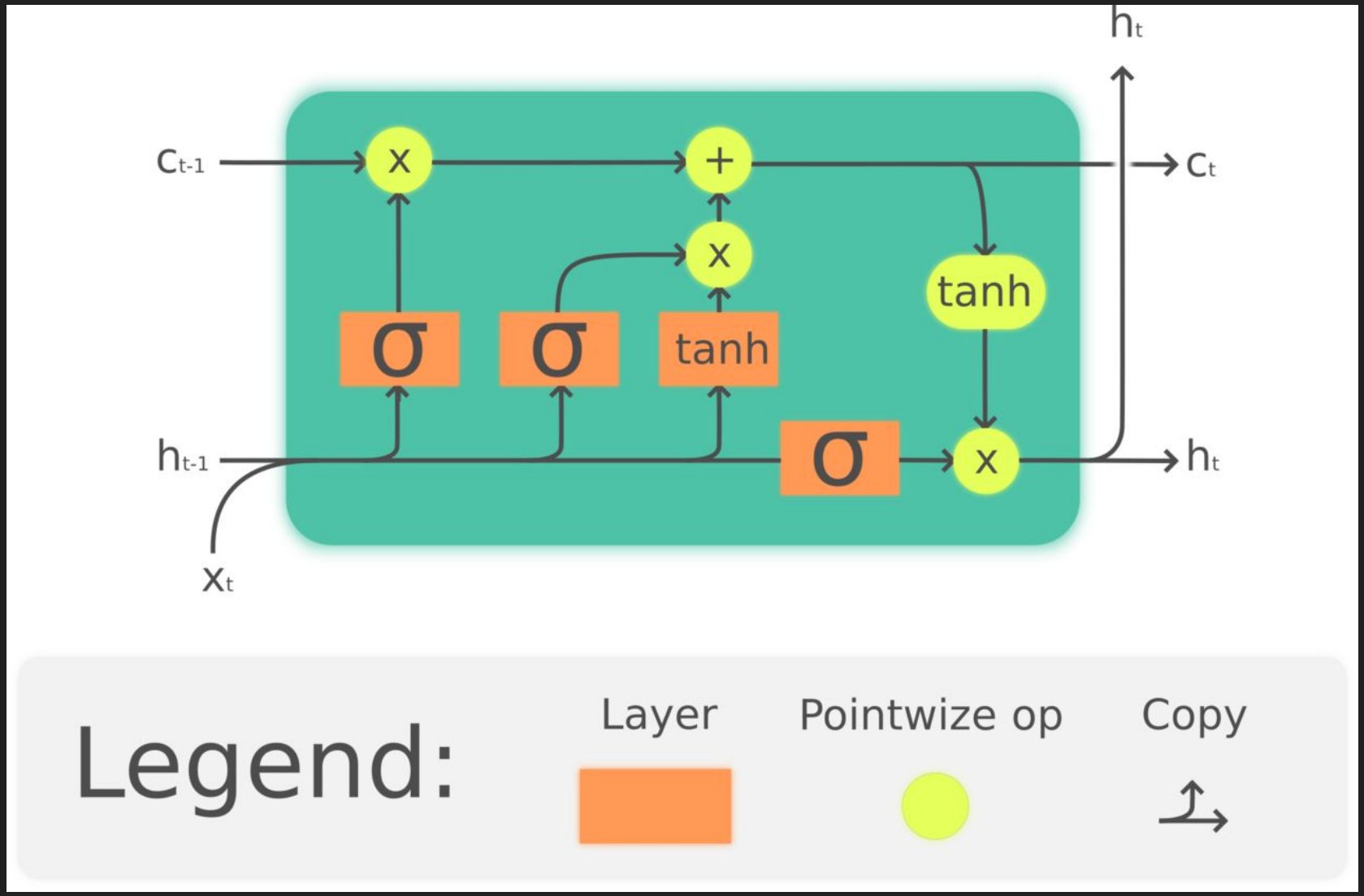
One LSTM unit, output gate



- C: cell state
- H: previous output
- X: input

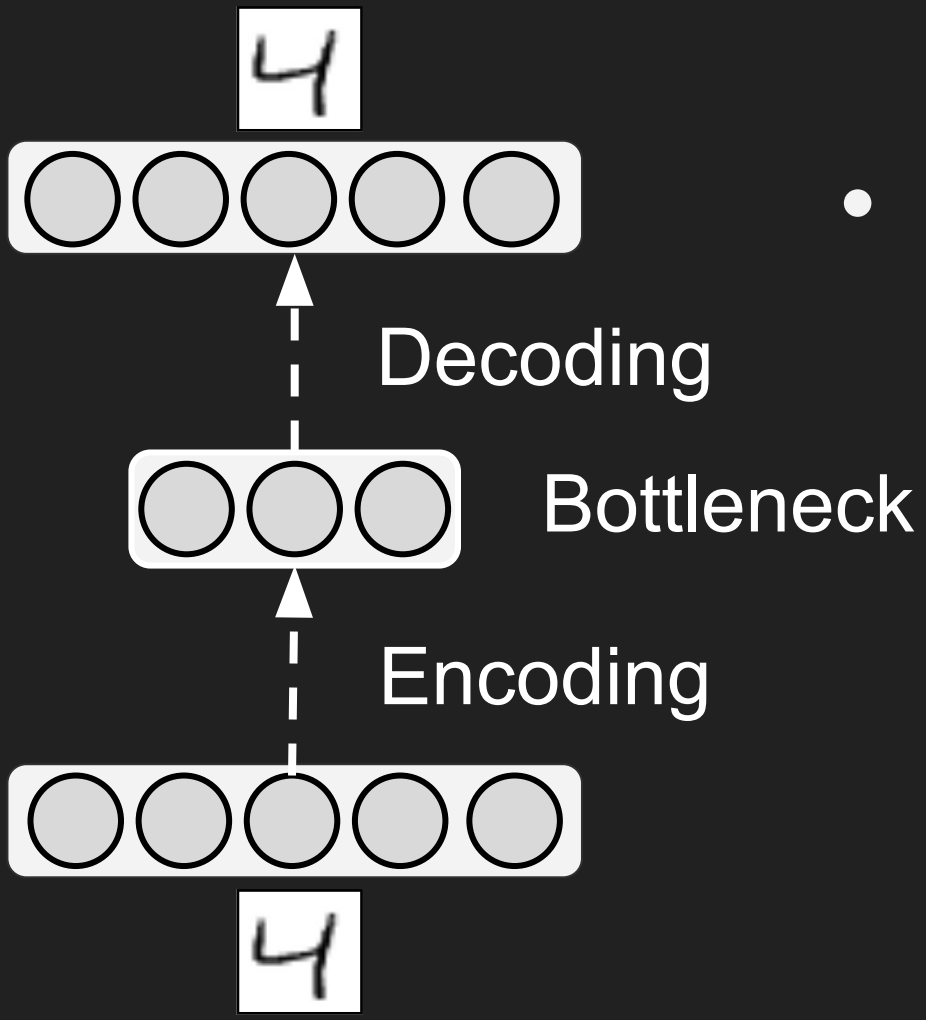
How much of C_t should outputted

One LSTM unit



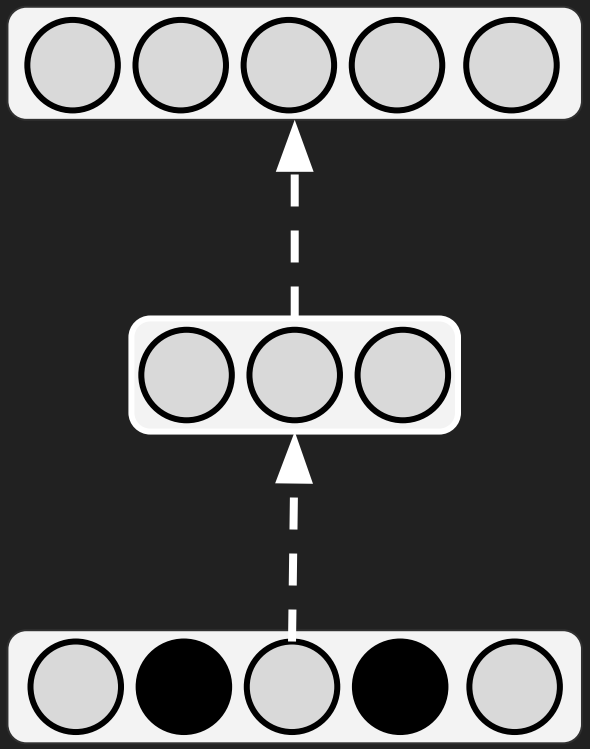
Artificial neural networks
V

Dimensionality reduction : Autoencoders



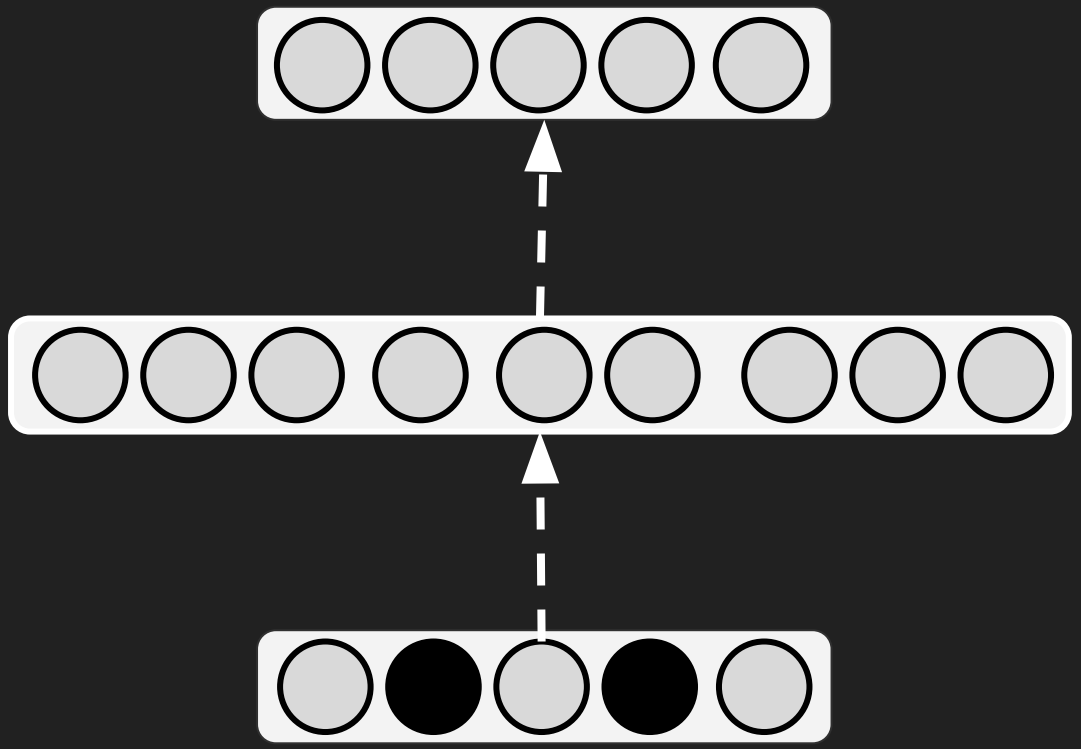
- Bottleneck neurons “summarise” inputs in lower dimensionality

Denoising autoencoders



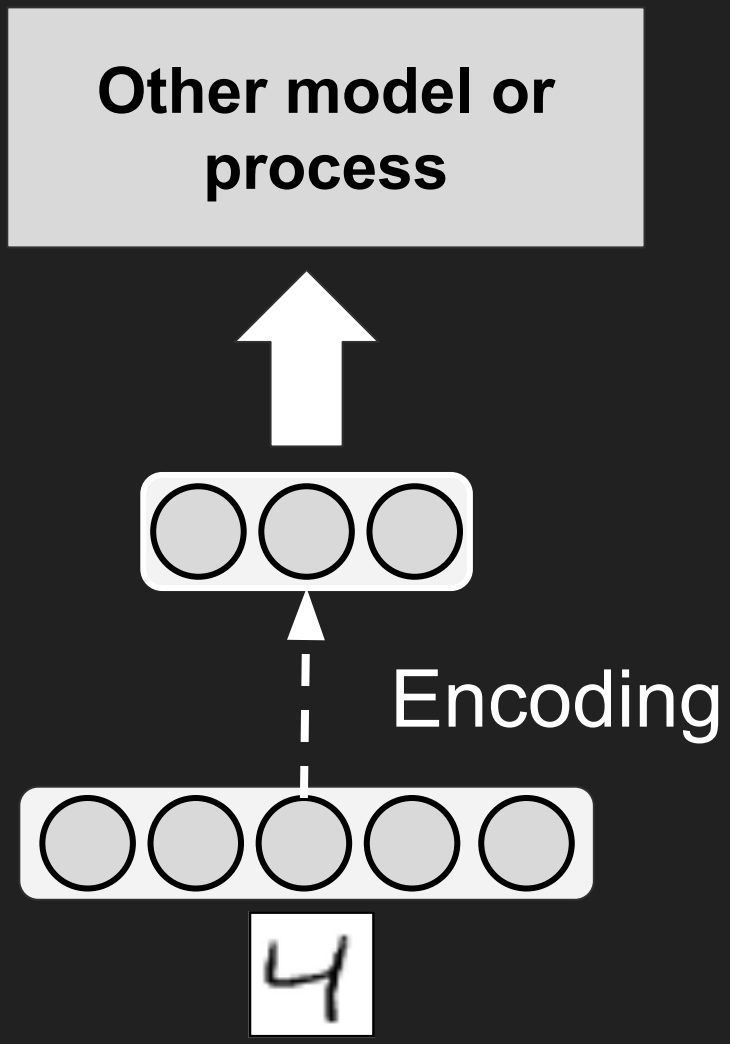
- Randomly mask part of the input
- Or add noise(ex: Normal)
- Improves generalisation

Denoising autoencoders (Over-complete)



- Learn to denoise inputs

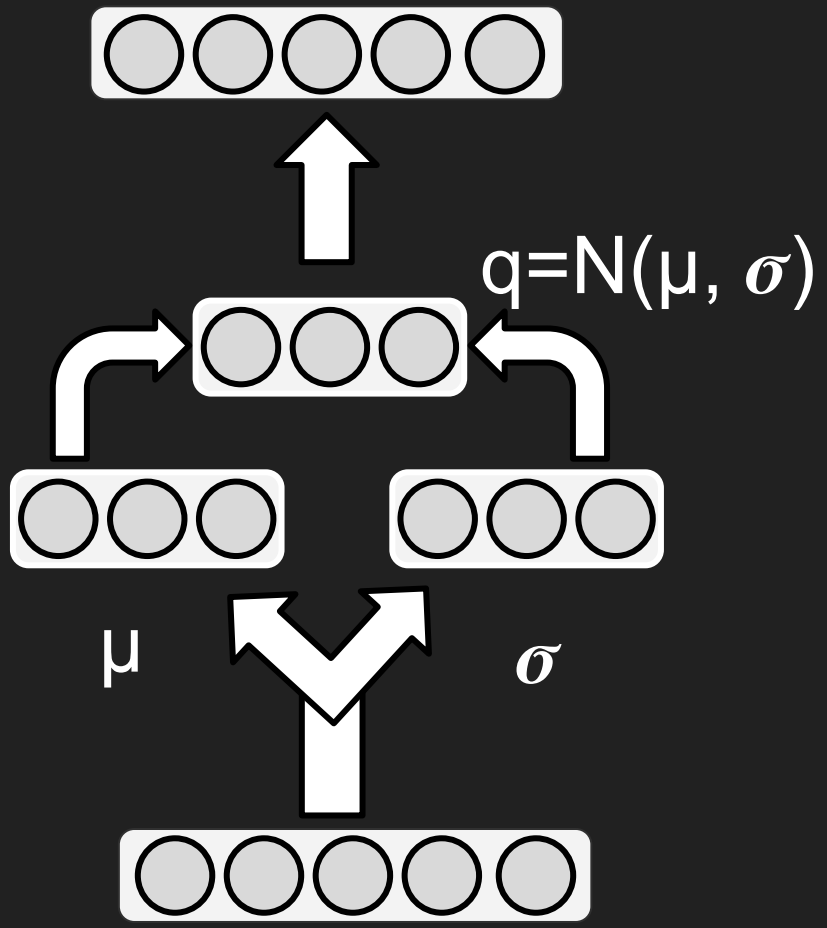
Autoencoders



- Reduce input size conserving information
- More “meaningful” input
- Improve generalisation
- Reduce overfitting

- Visualization

Variational autoencoders

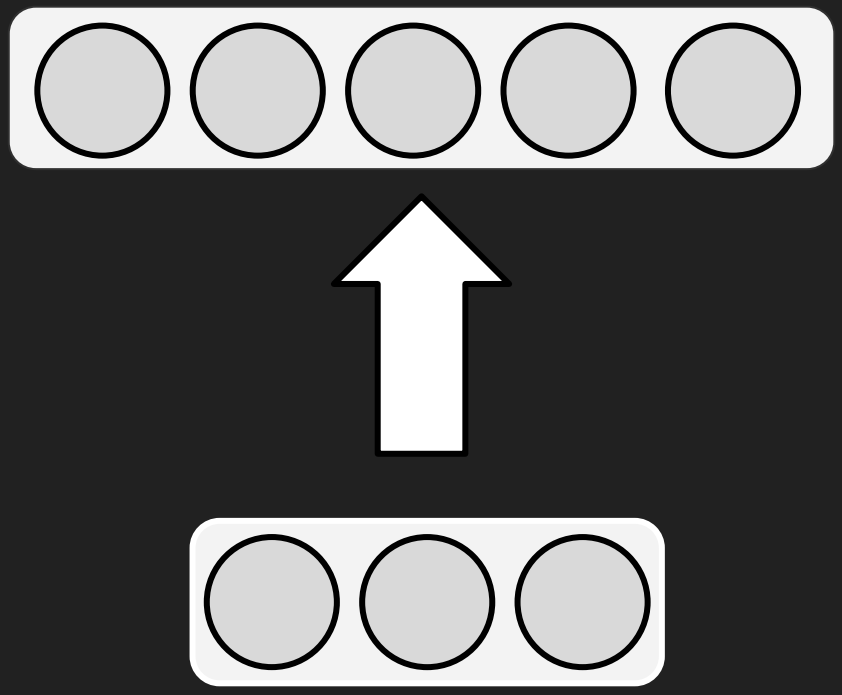


- L = C + KL(q || N(0, 1))
- Better results
- Forces latent space to be dense
- Generative model

$$\mathcal{L}(x, \hat{x}) + \sum_j KL\left(q_j(z|x) \,||p(z)\right)$$

Plus : Youtube: Ali Ghodsi, Lec : Deep Learning, Variational Autoencoder, Oct 12 2017 [Lect 6.2]

Variational autoencoders

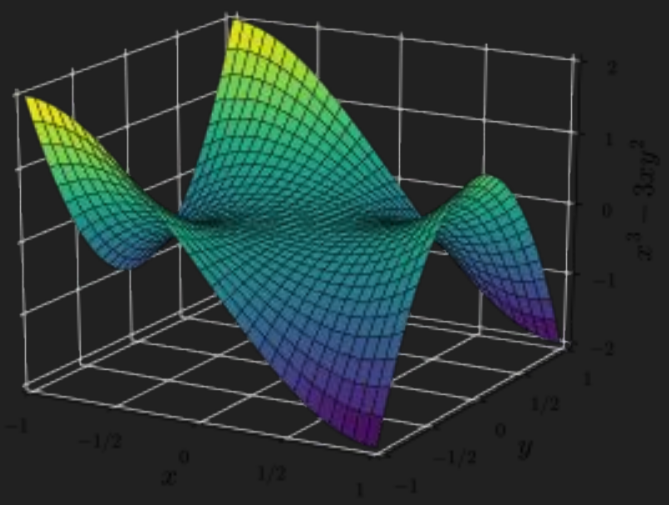


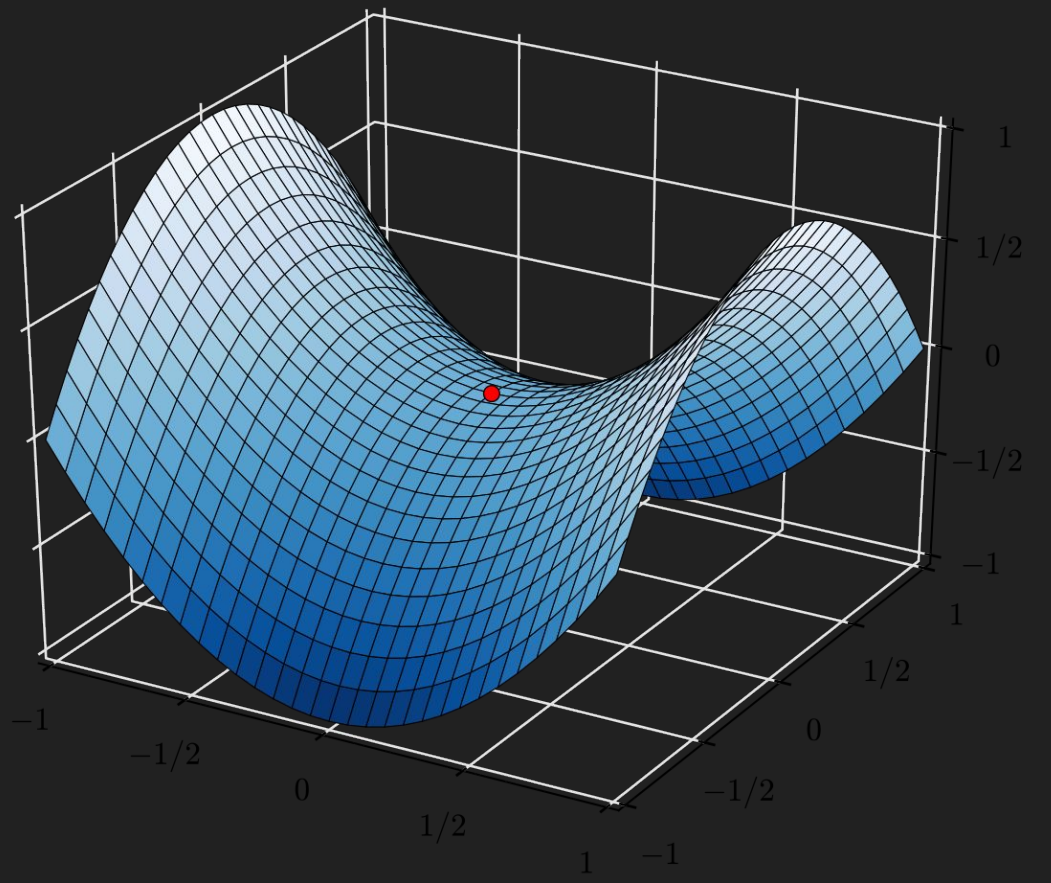
q=N(μ, σ)

- Sample from N(μ, σ)
- Give the result to the decoder

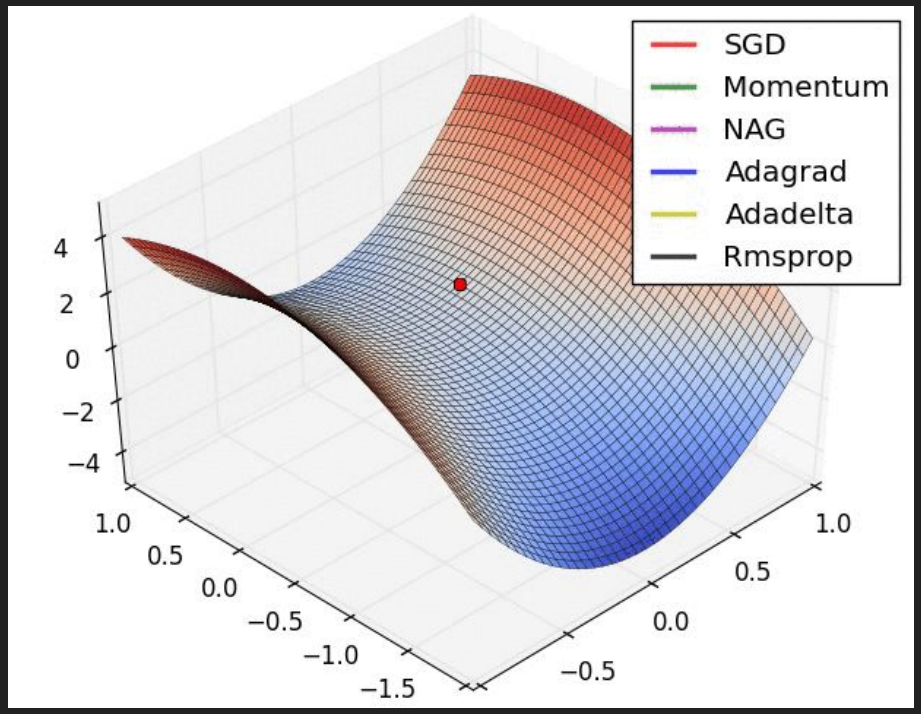
Réseaux de neurones artificiels
VI

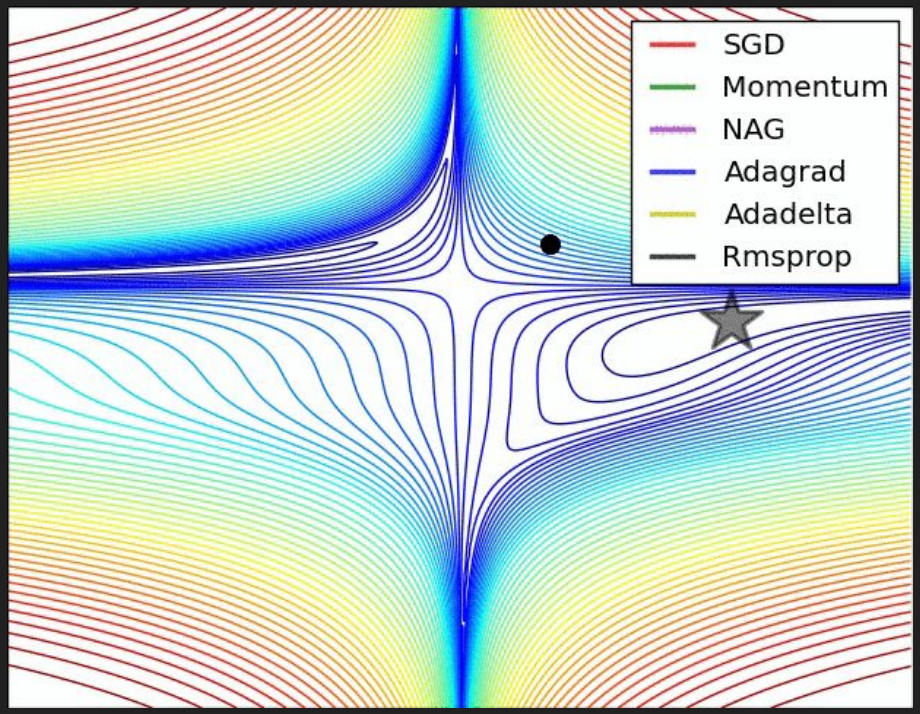
Loss surface





Stochastic gradient descent variants





Gradient descent with Momentum

- ‘Rolling ball’
- Add part of the previous gradient to the current (ex : 0.9)

Momentum rate

Learning rate

$$u_t = \gamma u_{t-1} + \alpha \frac{dC}{dw_1}$$

$$w_1 = w_1 - u_t$$

Adagrad

- Integrate history of updates
- Automatically scale learning rate
- Focus on less updated parameters

Learning rate,
Ex: 0.01

$$w_1 = w_1 - \sqrt{\frac{\alpha}{G + \epsilon}} w_1$$

Sum of previous gradients

Prevent div. by 0

Adam

- Popular
- Momentum
  - On average gradients
  - Uncentered variance
- Scale according to both

0.9

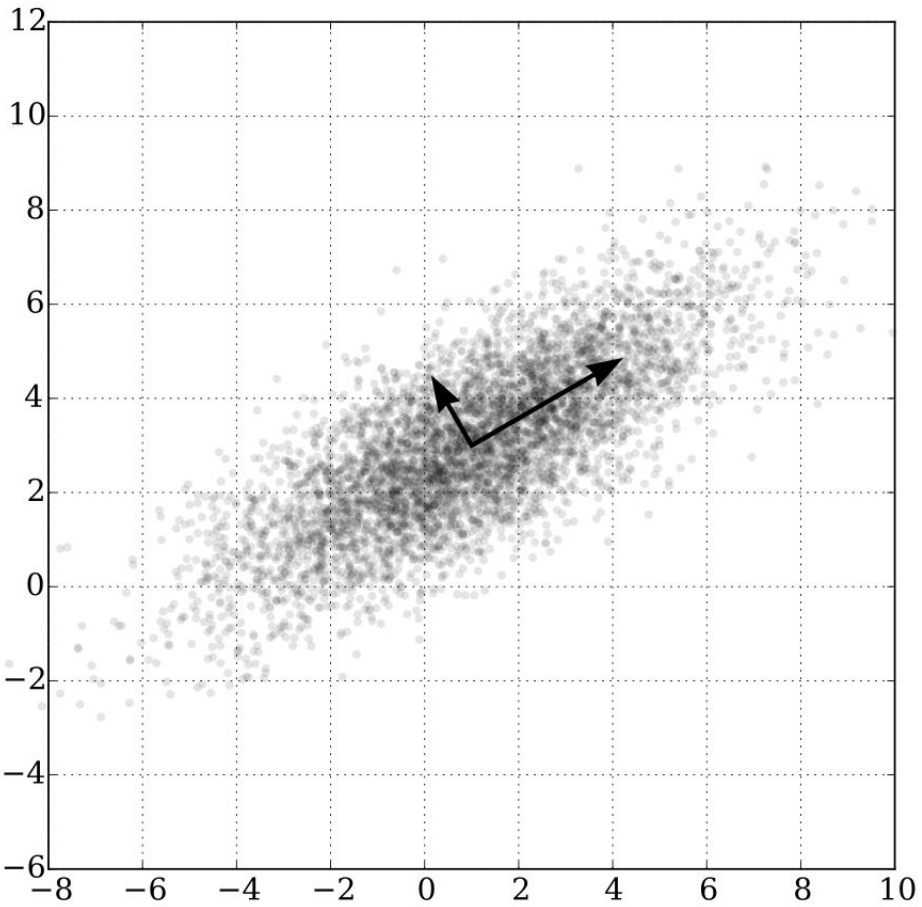
0.999

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) g_t^2$$

$$w_1 = w_1 - \frac{\alpha}{\sqrt{u_t} + \epsilon} m_t$$

Dimensionality reduction

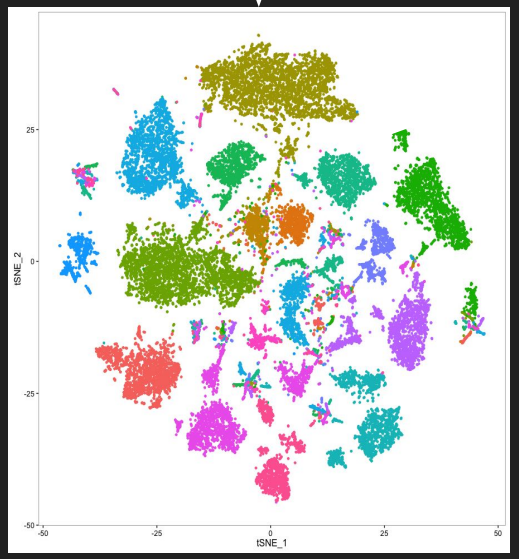
Principal Component Analysis



- Find the max directions of variance
- Project onto them

tSNE

Expressions de gènes (20K)

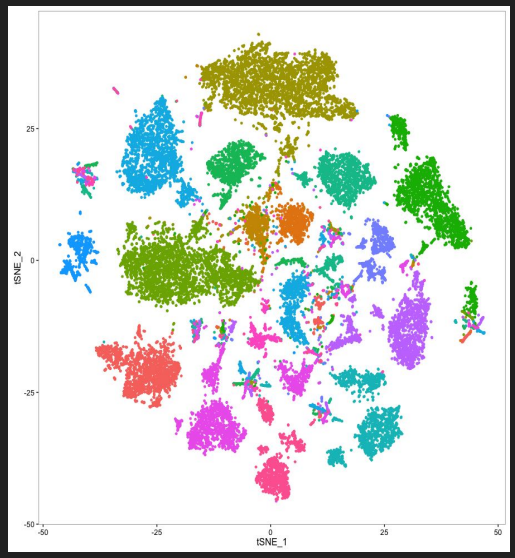


Représentation : 2D

- Entraîné par descente de gradient
- Stochastique
- Ne donne pas toujours exactement les mêmes résultats
- Les points “prochent” dans l’espace d’origine ont une forte probabilité d'être proches dans l’espace final

tSNE

20K dimensions → tSNE →



Points that are likely to be close in original space, are likely to be close in the 2D space

tSNE

0- Assign random coordinates to every point (y_1 and y_2)

1- Compute distance between points in the final 2D space, make a probability distribution out of it (q)

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

2- Compute distance between points in original space, make another probability distribution (p)

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)} \qquad p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

3- Move y_1 y_2 so that q is close to p using gradient descent with loss function:

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Tools

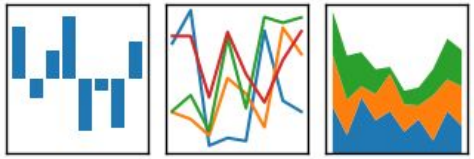
Entraînement

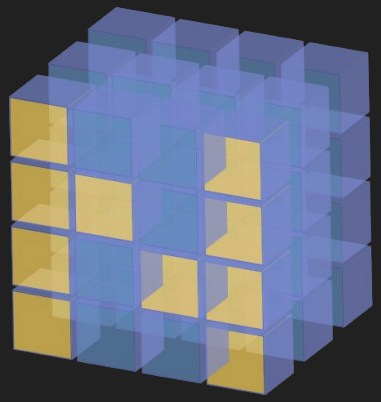
Outils : Calculs numériques

SciPy

pandas

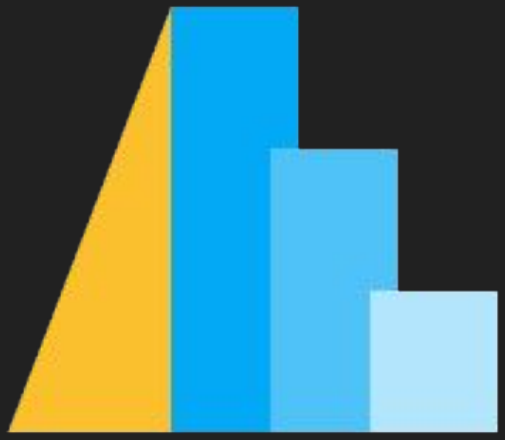
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$





NumPy

Outils : Visualisation



Altair

Visdom

Seaborn

Outils : bases de données

Outils : Apprentissage

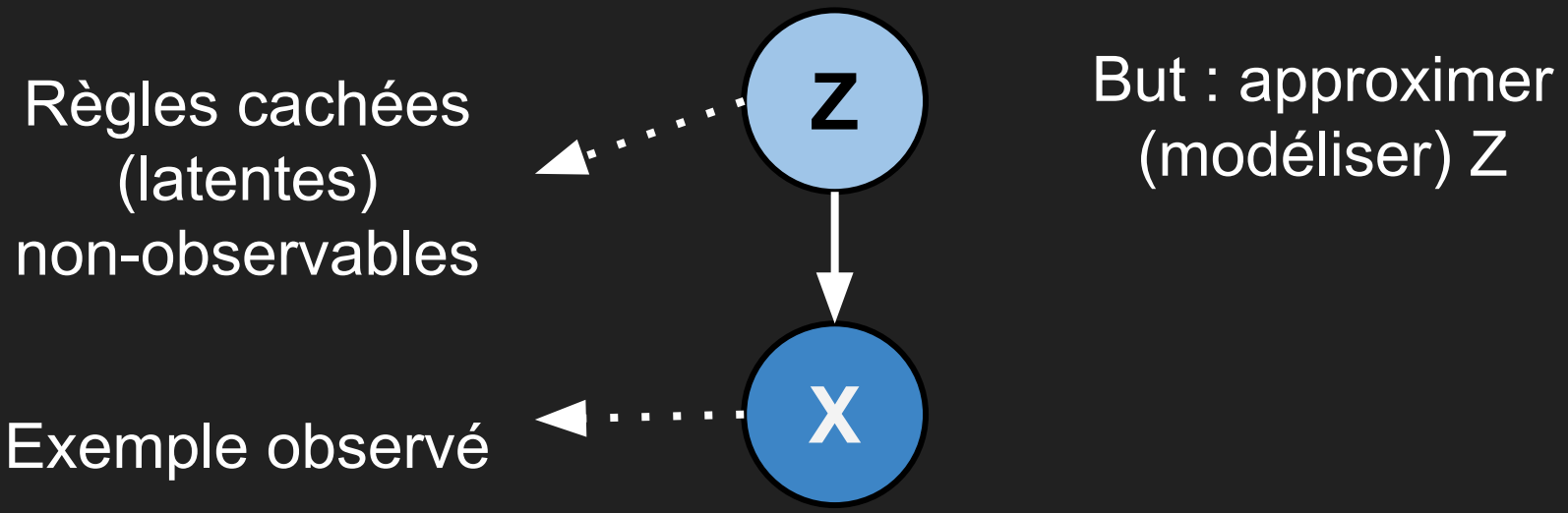
Keras

.FIN.

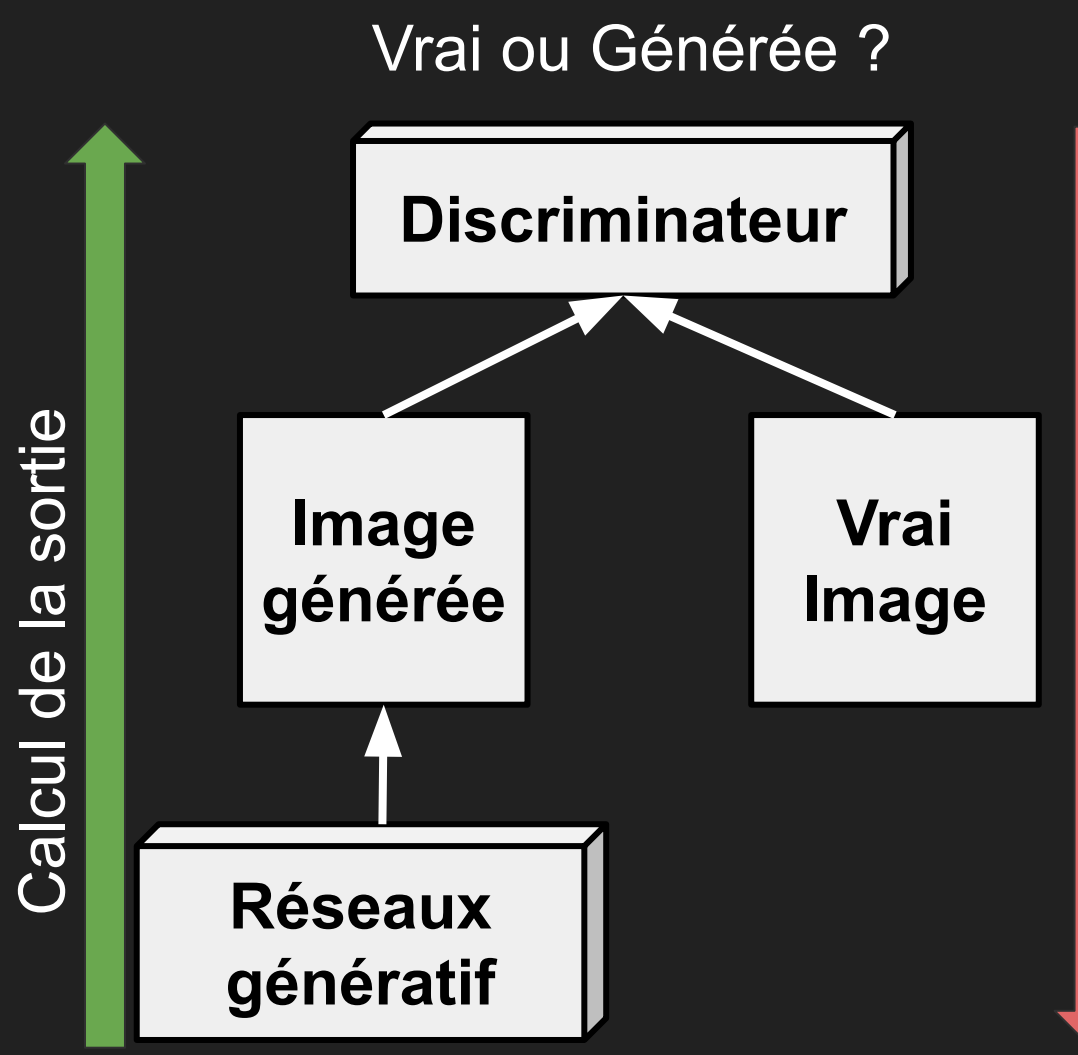
Réseaux de neurones artificiels
VII

Réseaux génératifs

- Apprendre les règles statistiques qui permettent de générer des sorties similaires aux exemples d'entraînement
- Application la plus connue : Images
- Mais aussi : générer des molécules aux effets similaires à celles de l'ensemble d'entraînement

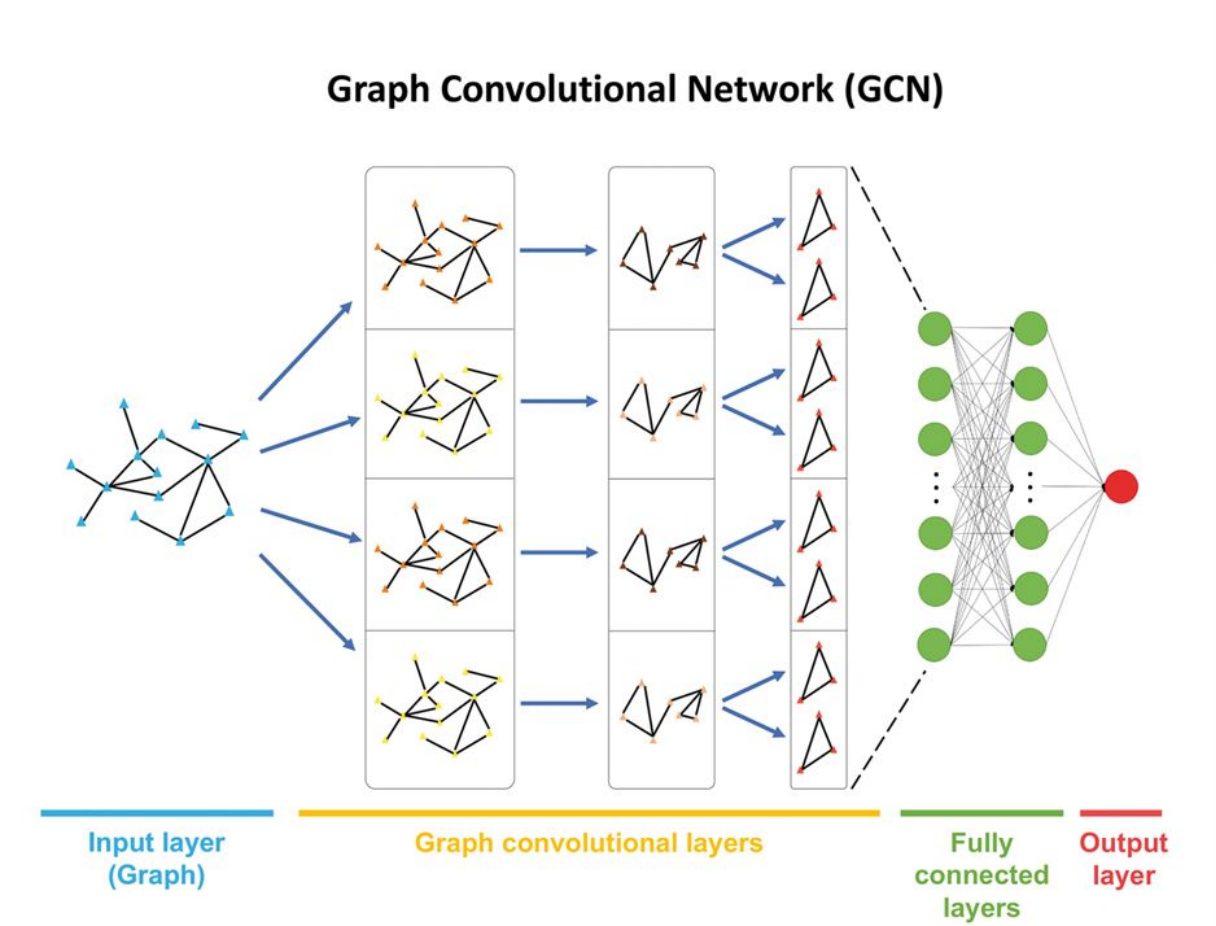


Réseaux Adversariels Génératifs (GANs)



- Excellents résultats
- Plus difficile à entraîner (dynamique adversariale)

Convolutions de graphes



- Utile pour les molécules

Sureyya Rifaioglu, et al, Brief in Bioinformatics, 2018 https://doi.org/10.1093/bib/bby061