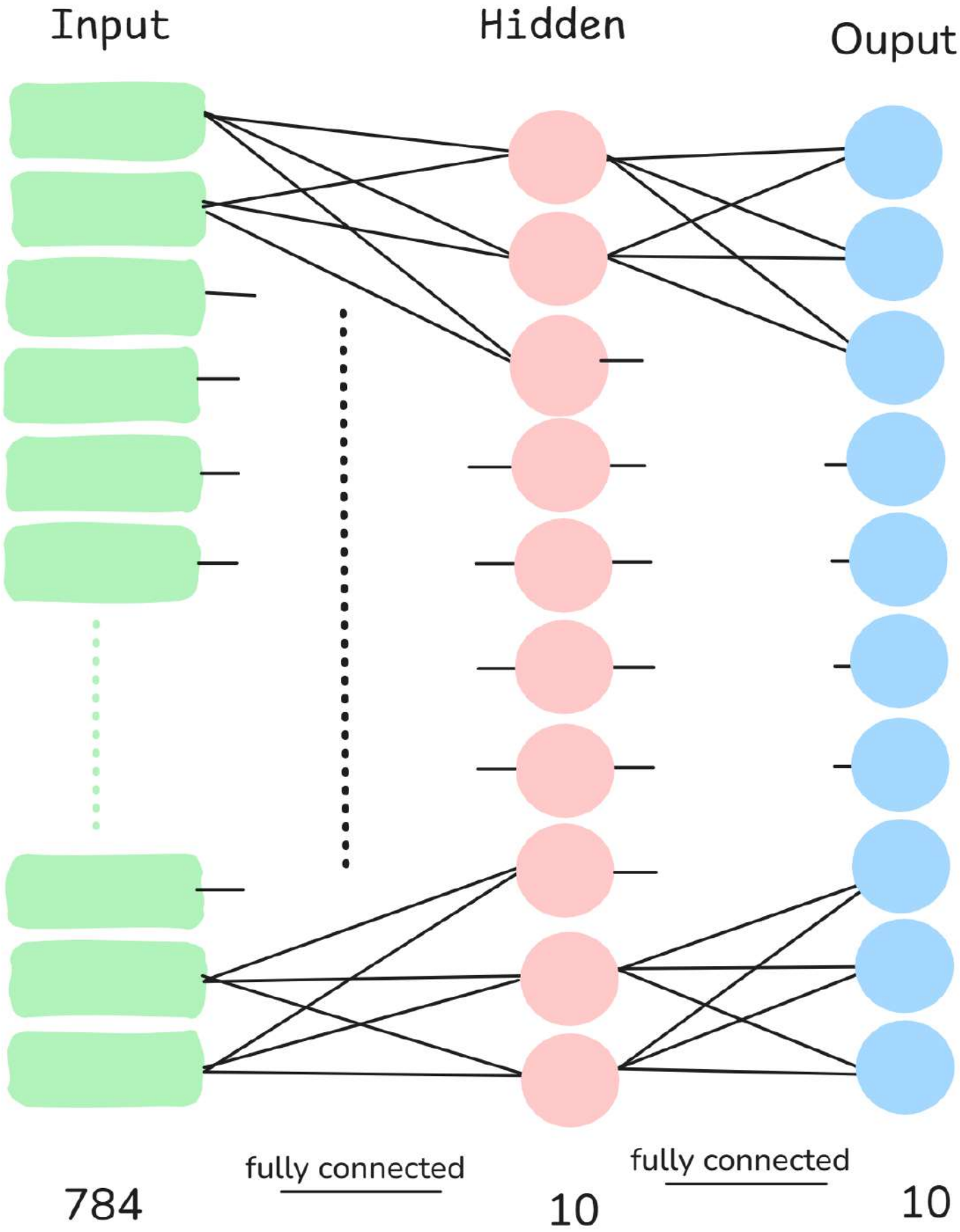
Neural Network #1



Etapes pour développer des réseaux de neurones artificiels

On répète en boucles les 4 étapes suivantes :

1- Forward Propagation :

On fait circuler les données de la première couche jusqu'à la dernière, afin de produire une sortie y.

2- Cost Function

Calculer l'erreur entre cette sortie et la sortie de référence y_{true} que l'on désire avoir.

3- Backward Propagation

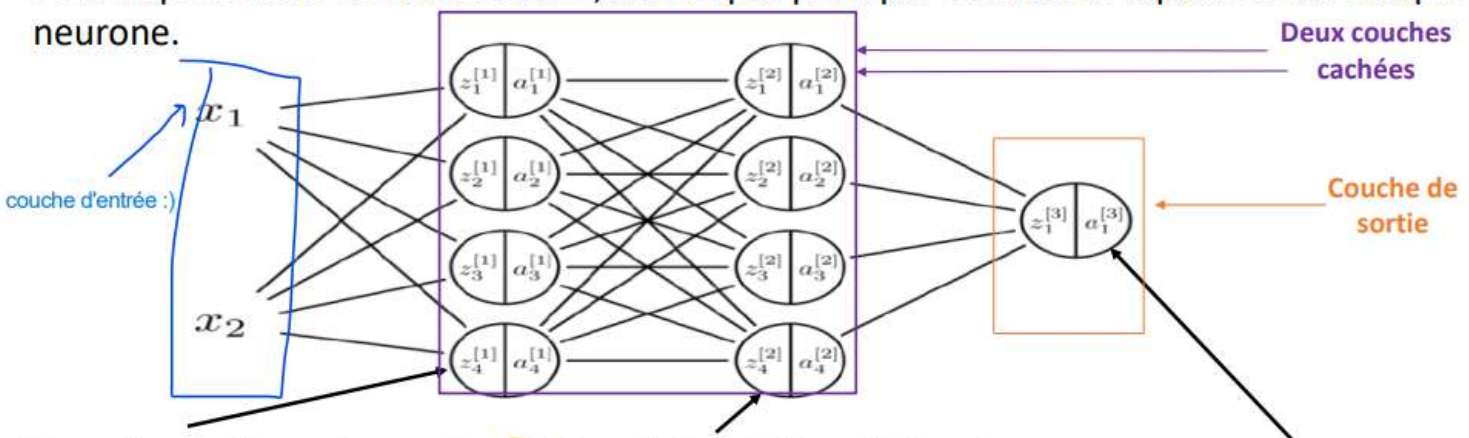
Mesurer comment cette fonction varie par rapport à chaque couche de notre modèle, en partant de la dernière et en remontant jusqu'à la toute première.

4- Gradient Descent

Corriger chaque paramètre du modèle grâce à l'algorithme de la descente de gradient, avant de reboucler vers la première étape, celle de la Forward Propagation, pour recommencer un cycle d'entraînement.

Comment construire un réseau de neurones

Pour implémenter de tels modèles, il n'est pas pratique d'écrire les équations de chaque neurone.



$$z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + b_1^{[1]} \qquad z_1^{[2]} = w_{11}^{[1]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + w_{13}^{[2]} a_3^{[1]} + w_{14}^{[2]} a_4^{[1]} + b_1^{[2]} \qquad z_1^{[3]} = w_{11}^{[3]} a_1^{[2]} + w_{12}^{[3]} a_2^{[2]} + w_{13}^{[3]} a_3^{[2]} + w_{14}^{[3]} a_4^{[2]} + b_1^{[3]}$$

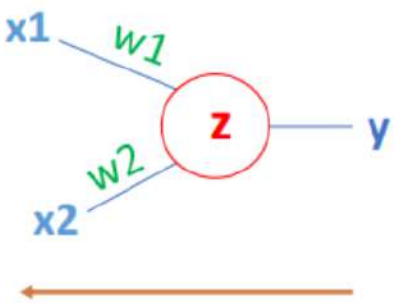
$$z_2^{[1]} = w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2 + b_2^{[1]} \qquad z_2^{[2]} = w_{21}^{[1]} a_1^{[1]} + w_{22}^{[2]} a_2^{[1]} + w_{23}^{[2]} a_3^{[1]} + w_{24}^{[2]} a_4^{[1]} + b_2^{[2]}$$

$$z_3^{[1]} = w_{31}^{[1]} x_1 + w_{32}^{[1]} x_2 + b_3^{[1]} \qquad z_3^{[2]} = w_{31}^{[1]} a_1^{[1]} + w_{32}^{[2]} a_2^{[1]} + w_{33}^{[2]} a_3^{[1]} + w_{34}^{[2]} a_4^{[1]} + b_3^{[2]}$$

$$z_4^{[1]} = w_{41}^{[1]} x_1 + w_{42}^{[1]} x_2 + b_4^{[1]} \qquad z_4^{[2]} = w_{41}^{[1]} a_1^{[1]} + w_{42}^{[2]} a_2^{[1]} + w_{43}^{[2]} a_3^{[1]} + w_{44}^{[2]} a_4^{[1]} + b_4^{[2]}$$

Back-Propagation

Consiste à retracer comment la Fonction Coût évolue de la dernière couche du réseau jusqu'à la toute première



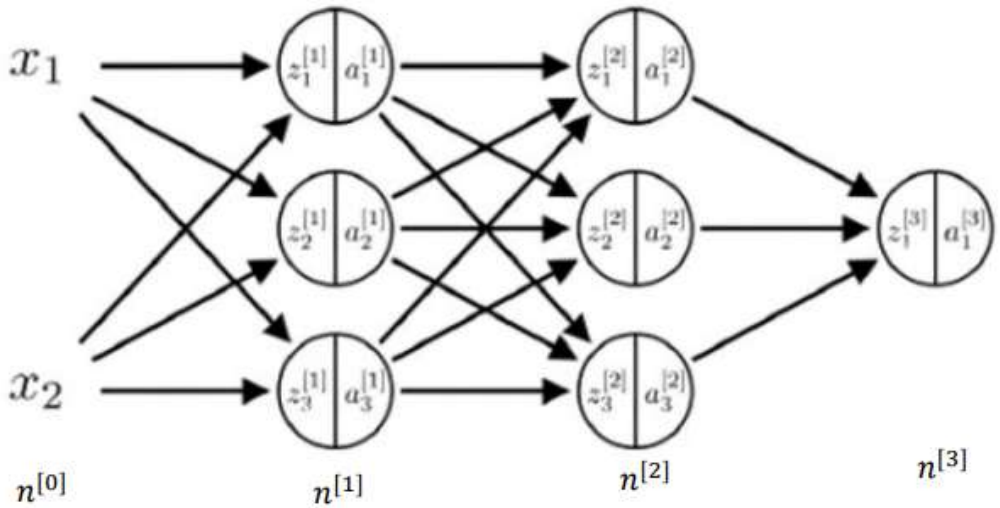
$$Z = W.X + b$$

A= max(0, x)

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial A} \times \frac{\partial A}{\partial Z} \times \frac{\partial Z}{\partial W}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial A} \times \frac{\partial A}{\partial Z} \times \frac{\partial Z}{\partial b}$$

Réseau de neurones profond



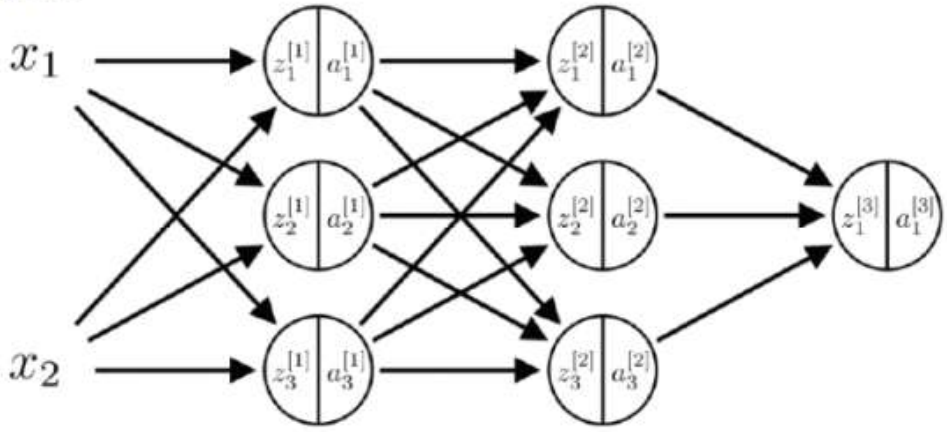
for c in (1,2,3)

$$W^{[c]} \in \mathbb{R}^{n^{[c]} \times n^{[c-1]}}$$

$$b^{[c]} \in \mathbb{R}^{n^{[c]} \times 1}$$

Réseau de neurones profond

Back-Propagation



for c in range(C_{final}, 1)

$$dW^{[c]} = \frac{1}{m} \times dZ^{[c]}.A^{[c-1]^T}$$

$$db^{[c]} = \frac{1}{m}\sum dZ^{[c]}$$

$$dZ^{[c-1]} = W^{[c]^T}.dZ^{[c]} \times A^{[c-1]}(1 - A^{[c-1]})$$

Réseau de neurones profond

Mise à jour des paramètres

for c in range(1, C_{final})

$$W^{[c]} = W^{[c]} - \alpha \times dW^{[c]}$$
$$b^{[c]} = b^{[c]} - \alpha \times db^{[c]}$$

Initialisation des Poids

Stratégies d'Initialisation		
Méthode	Formule	Quand l'utiliser ?
Aléatoire Simple	Petites valeurs aléatoires (souvent [-0.1,0.1])	Peu recommandé, peut causer des problèmes de convergence.
Xavier	$\sigma(w) = \sqrt{\frac{1}{n_{in} + n_{out}}}$	Pour les fonctions d'activation **tanh** ou **sigmoid**. Permet de maintenir une bonne variance des activations.
He Initialization	$\sigma(w) = \sqrt{\frac{2}{n_{in}}}$	Idéal pour les réseaux avec des activations **ReLU** ou ses variantes.

Où :

n_{in} = **nombre de neurones en entrée de la couche**
n_{out} = **nombre de neurones en sortie de la couche**

Learning Rate et Décroissance du Learning Rate

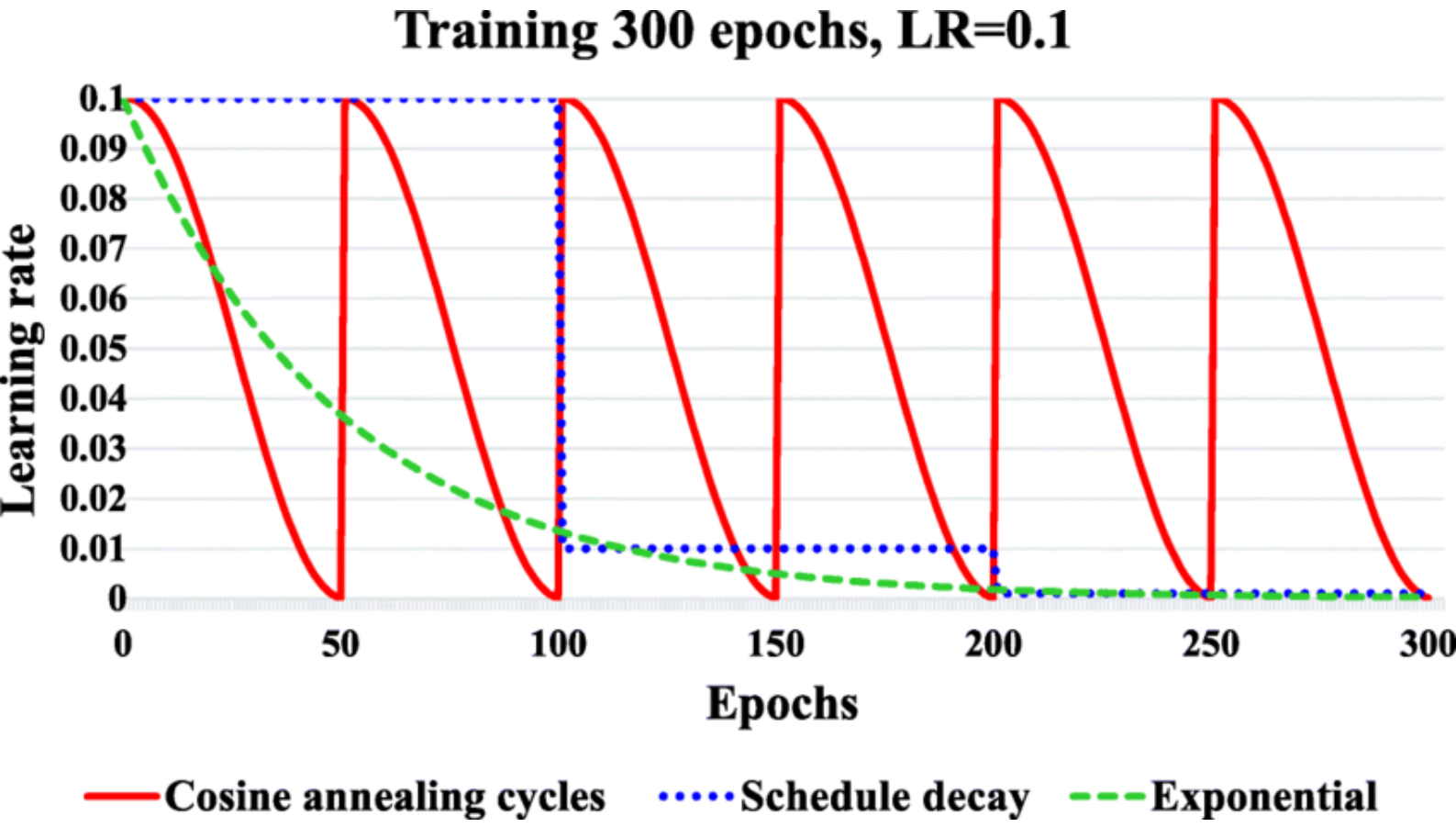
$$\alpha = \frac{1}{1 + decay_{rate} * epoch_{num}} \alpha_0$$

Exemple :

Epoch	α
1	0.1
2	0.06
3	0.05
4	0.04
5	0.03

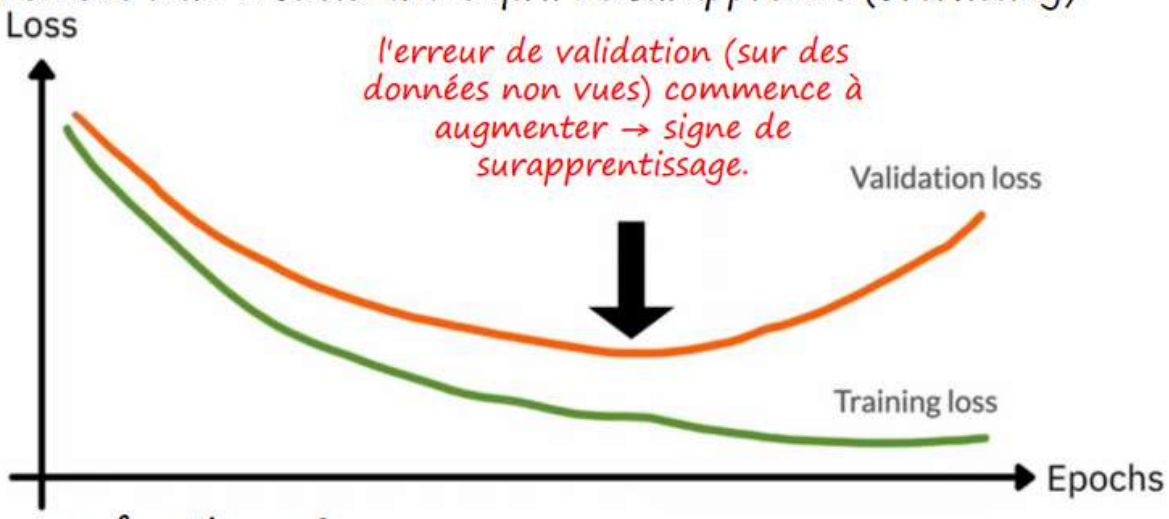
α_0=0.2
$decay_{rate} = 1$

Astuce : Une bonne pratique est de démarrer avec un learning rate plus grand et de le réduire progressivement au cours de l'entraînement.


Training 300 epochs, LR=0.1
Learning rate
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
50
100
150
200
250
300
Epochs
Cosine annealing cycles
Schedule decay
Exponential

Early Stopping

Early stopping est une technique de régularisation qui consiste à arrêter l'entraînement d'un modèle avant qu'il ne surapprenne (overfitting).



- Comment ça fonctionne ?

1) Suivre l'erreur sur le jeu de validation à chaque époque (epoch).

2) Si cette erreur cesse de s'améliorer après un certain nombre d'époques (appelé patience), on stoppe l'entraînement.