RÉSUMÉ THÉORIQUE – FILIÈRE INTELLIGENCE ARTIFICIELLE

M108 - Appréhender l'apprentissage profond

75 heures

MODALITÉS PÉDAGOGIQUES

1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques

2

LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life

3

DES CONTENUS TÉLÉCHARGEABLES

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life

4

DU CONTENU INTERACTIF

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life

5

DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage

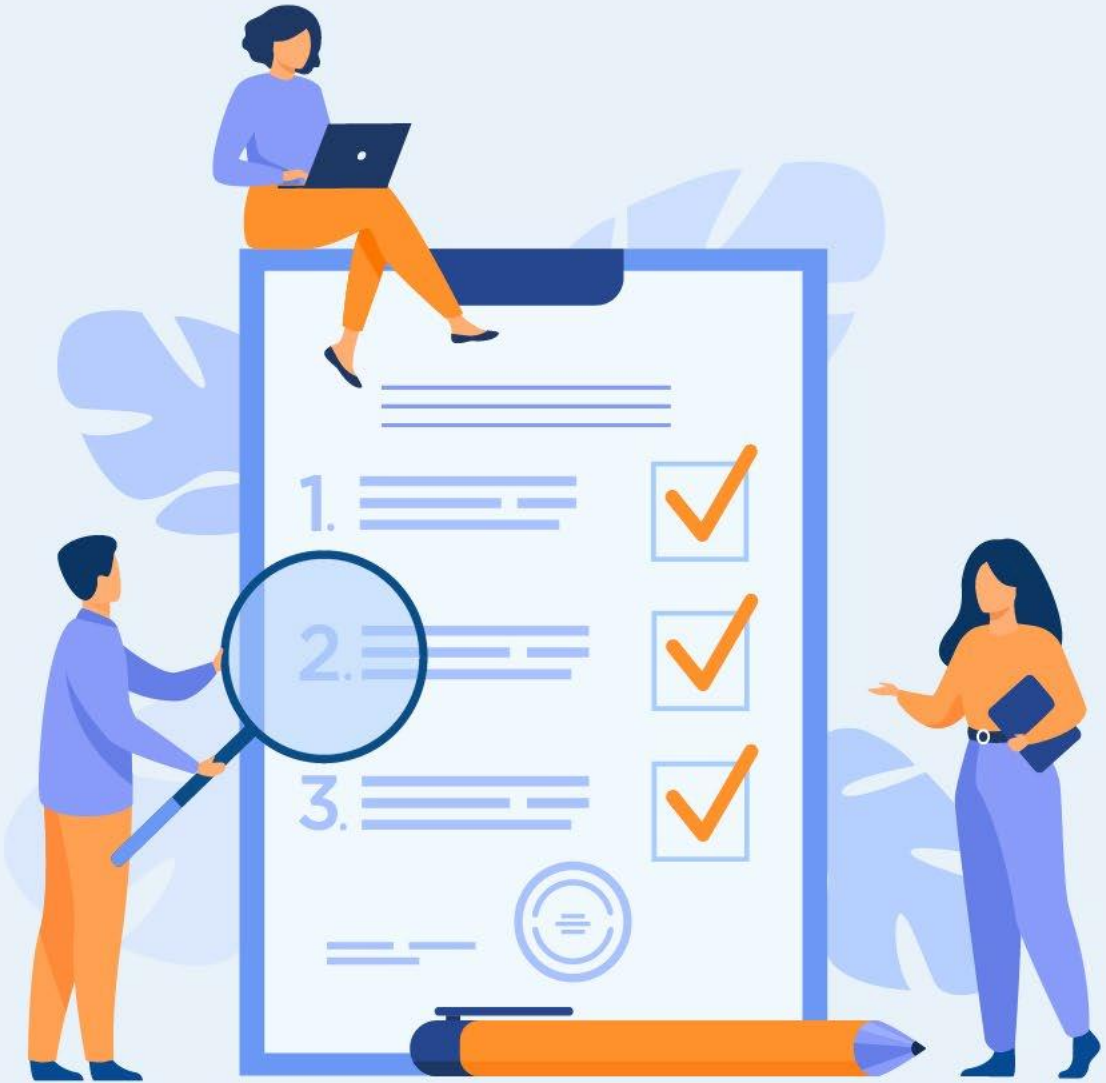
PARTIE 1

COMPRENDRE LES FONDAMENTAUX DU DEEP LEARNING

Dans ce module, vous allez :

- Introduire les réseaux de neurones
- Maîtriser les techniques de régularisation

18 heures



CHAPITRE 1

Les Bases de l'Apprentissage Profond

Ce que vous allez apprendre dans ce chapitre :

- Différencier entre l’apprentissage automatique et profond
- Faire l’analogie entre un réseau de neurone biologique et artificiel
- Connaître le principe de la descente de gradient
- Découvrir le perceptron et le perceptron multicouche

10 heures

1. Qu'est-ce que l'apprentissage profond ?

L'apprentissage profond (*deep learning*) est une méthode où les machines **apprennent automatiquement à partir d'exemples**, sans être explicitement programmées pour une tâche.

•**Exemple concret** :

- Si vous montrez des milliers de photos de chats et de chiens à un système d'apprentissage profond, il apprendra à les distinguer tout seul, sans que vous lui expliquiez les règles (ex. : "un chat a des moustaches").

2. Différences entre IA, Machine Learning (ML) et Deep Learning (DL)

L'Intelligence Artificielle (IA)

•**But** : Créer des machines capables de "penser" comme des humains.

•**Exemples larges** : Jeux d'échecs, robots, chatbots.

Le Machine Learning (Apprentissage Automatique)

•**Sous-domaine de l'IA** : Les machines apprennent à partir de données, sans règles prédéfinies.

•**Exemple** : Un modèle qui prédit le prix d'une maison en fonction de sa taille, du quartier, etc.

•**Limite** : Nécessite souvent des humains pour choisir les caractéristiques (*features*) importantes (ex. : "la superficie est plus importante que la couleur").

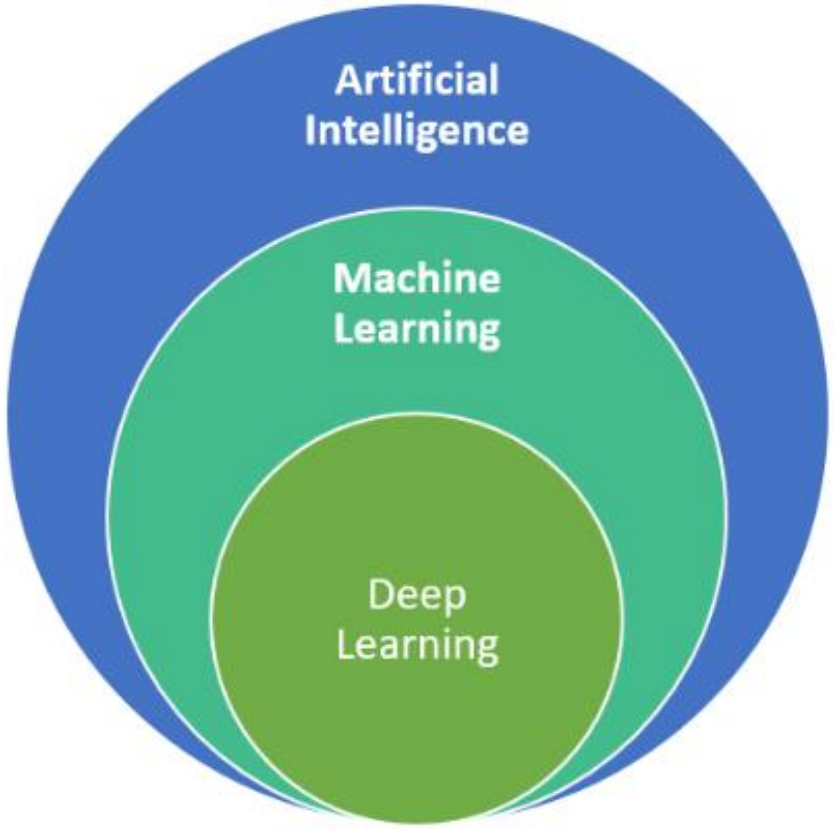
Le Deep Learning (Apprentissage Profond)

•**Sous-domaine du ML** : Utilise des **réseaux neuronaux artificiels** avec de nombreuses couches (*deep* = profond).

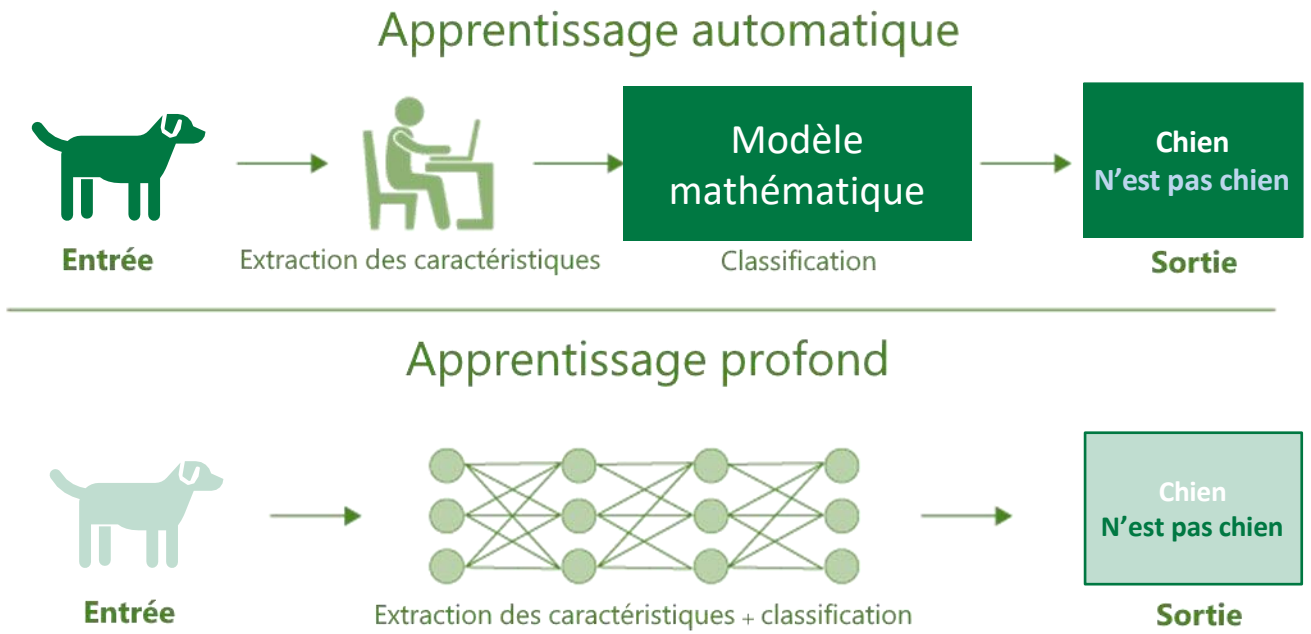
•**Avantage** : Découvre *automatiquement* les caractéristiques importantes dans les données.

- Exemple : Pour reconnaître un chat, le réseau apprend d'abord les contours, puis les formes, puis les détails (yeux, moustaches).

•**Applications typiques** : Vision par ordinateur, reconnaissance vocale, traduction automatique.

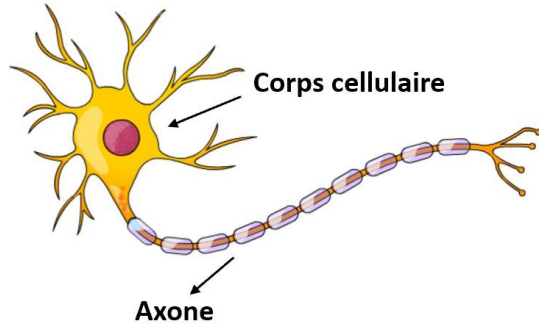


L'apprentissage automatique consiste en un ensemble d'algorithmes qui analysent les données, apprennent des modèles, puis appliquent ce qu'ils ont appris pour prendre des décisions intelligentes. La particularité de ces techniques traditionnelles est que la plupart des caractéristiques doivent être identifiées par un expert du domaine. Par contre, en apprentissage profond, c'est le réseau qui s'occupe de l'extraction des caractéristiques et l'apprentissage des modèles.



3. Comment fonctionne un réseau de neurones ?

Le neurone artificiel



•Inspiré du cerveau humain : Reçoit des informations, les traite, et transmet un signal.

•**Analogie** : Un groupe de personnes qui votent "oui" ou "non" pour prendre une décision.

Les 3 étapes clés d'un réseau neuronal :

1.Entrée (Input) : Données brutes (ex. : pixels d'une image).

2.Traitement par couches cachées : Chaque couche extrait des motifs de plus en plus complexes.

1. Exemple pour une image de chat :
    1. Couche 1 : Détecte les bords et les contours.
    2. Couche 2 : Reconnaît des formes (yeux, oreilles).
    3. Couche 3 : Combine les formes pour identifier un chat.

PARTIE 1

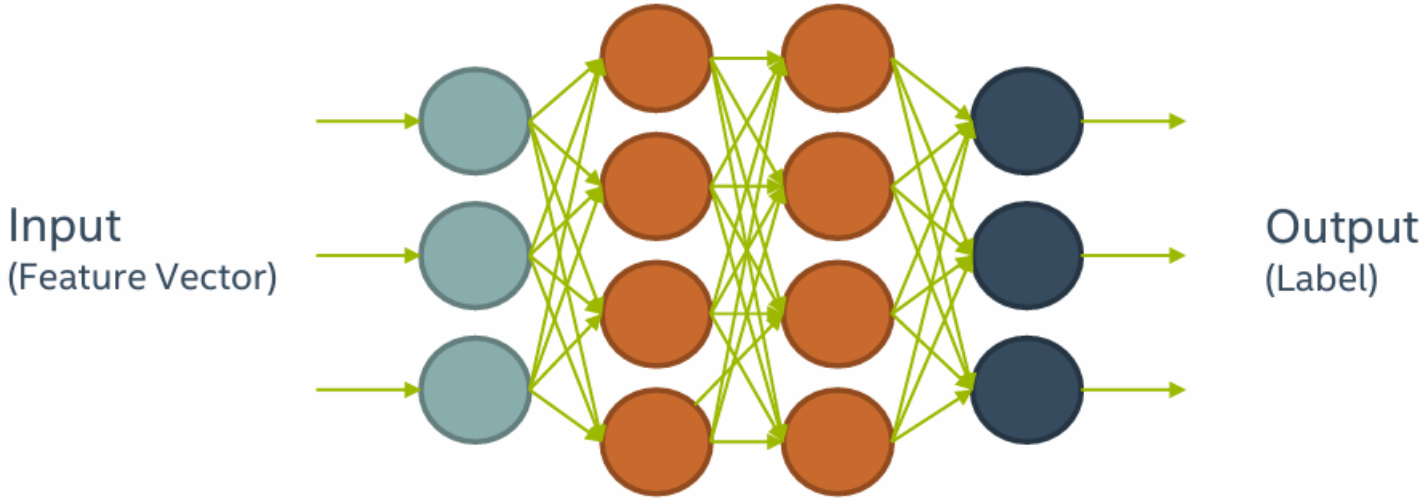
3. **Sortie (Output)** : Résultat final (ex. : "90% de chances que ce soit un chat").

Fonction d'activation :

•Détermine si un neurone doit "s'activer" et transmettre l'information.

•**Exemple** :

- Si un neurone détecte une "forme ronde", il active le neurone suivant pour chercher un "œil".



4. Pourquoi le deep learning est-il révolutionnaire ?

1.Automatisation des caractéristiques :

- Plus besoin de dire au modèle quoi regarder (ex. : "les chats ont des moustaches").
- Le réseau découvre tout seul les motifs utiles.

2.Performances sur des données complexes :

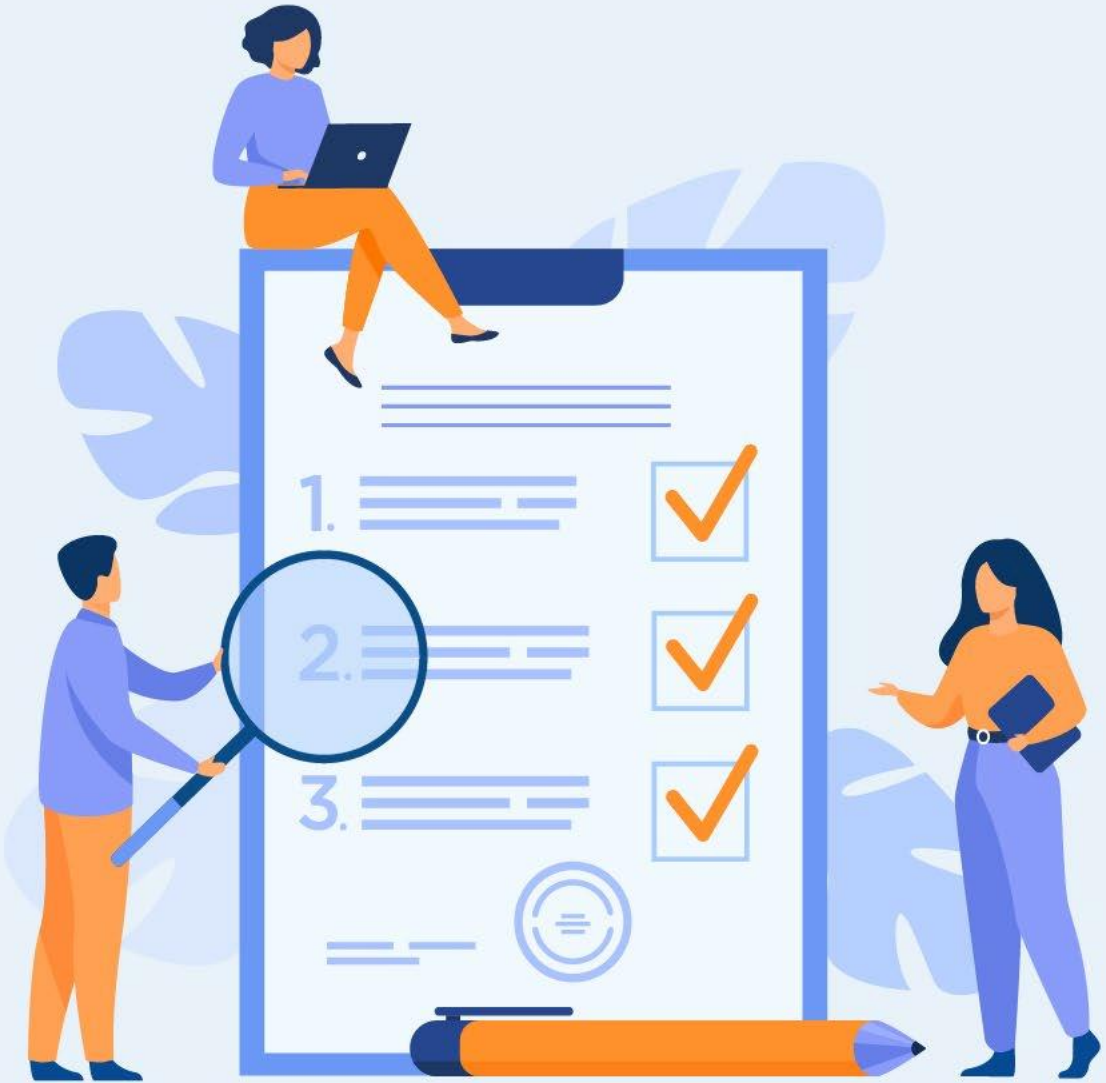
- Images, sons, vidéos, texte (données non structurées).
- Exemple : Avant le DL, reconnaître un chat dans une photo nécessitait des algorithmes manuels complexes.

3.Flexibilité :

- Le même principe s'applique à des domaines variés (médical, automobile, artistique).

5. Vocabulaire de base à retenir

- **Réseau neuronal** : Structure inspirée du cerveau, composée de couches de neurones.
- **Entraînement** : Phase où le modèle apprend à partir d'exemples étiquetés.
- **Données d'entraînement** : Exemples utilisés pour apprendre (ex. : 10 000 images de chats/chiens).
- **Prédiction** : Résultat produit par le modèle après apprentissage.



CHAPITRE 2

Architecture des Réseaux Neuronaux

Ce que vous allez apprendre dans ce chapitre :

- Différencier entre l'apprentissage automatique et profond
- Faire l'analogie entre un réseau de neurone biologique et artificiel
- Connaître le principe de la descente de gradient
- Découvrir le perceptron et le perceptron multicouche

10 heures

1. Structure d'un Réseau Neuronal

Un réseau neuronal est organisé en **couches interconnectées**, comme une chaîne de montage où chaque étape affine la compréhension des données.

Les 3 types de couches :

1.Couche d'entrée (Input Layer) :

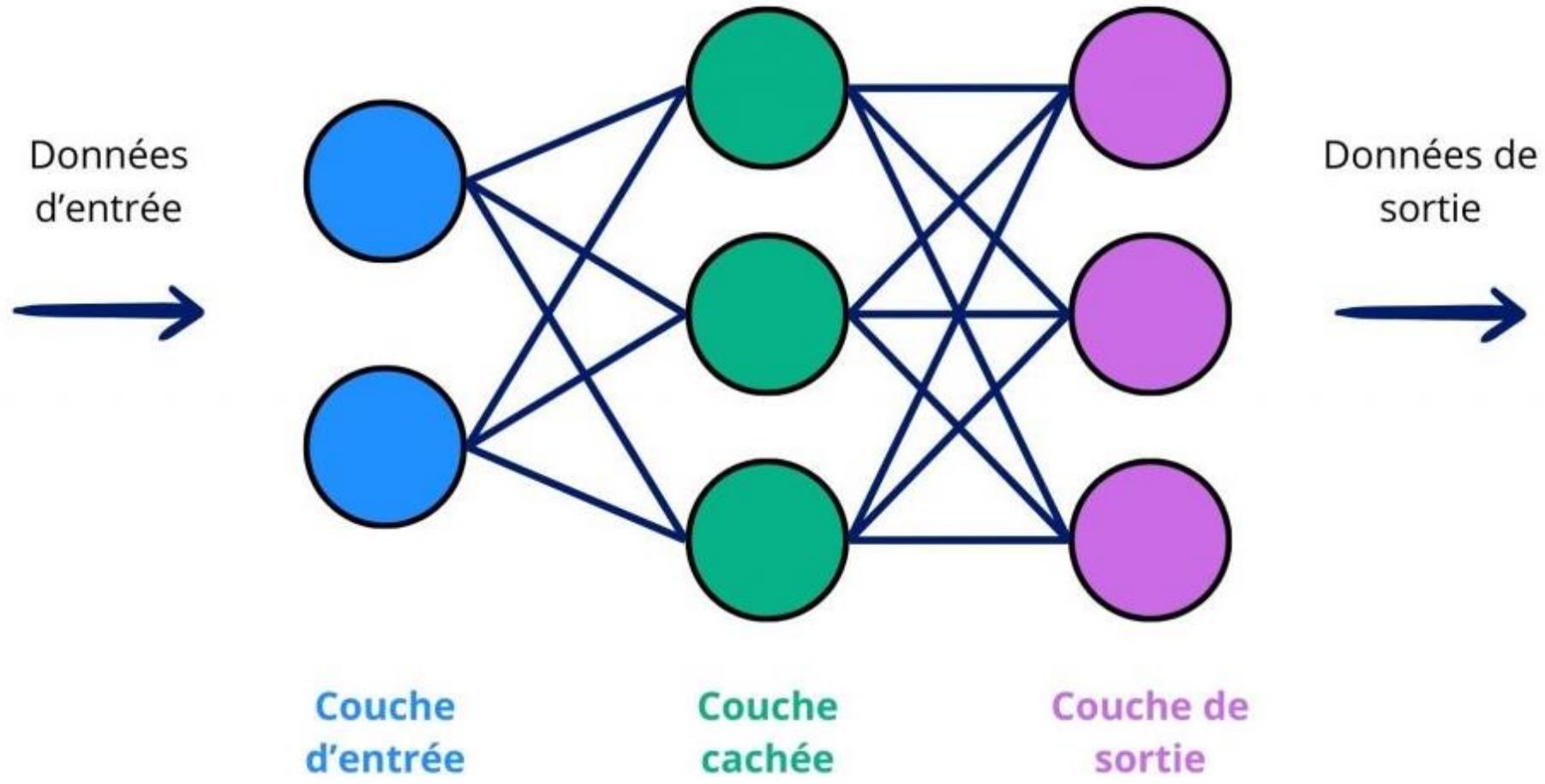
- **Rôle** : Reçoit les données brutes (ex. : pixels d'une image, mots d'un texte).
- **Analogie** : Comme vos yeux qui captent la lumière pour voir une image.
- **Exemple** : Pour une image de 28x28 pixels (MNIST), la couche d'entrée a 784 neurones (un par pixel).

2.Couches cachées (Hidden Layers) :

- **Rôle** : Extraire des motifs de plus en plus complexes.
- **Nombre de couches** : Définit la "profondeur" du réseau (d'où "deep learning").

- **Exemple de progression** :
  - Couche 1 : Détecte des bords ou des taches de couleur.
  - Couche 2 : Reconnaît des formes simples (yeux, nez).
  - Couche 3 : Combine les formes pour identifier un visage.

3. **Couche de sortie (Output Layer)** :
   - **Rôle** : Fournir le résultat final (prédiction, classification).
   - **Exemples** :
     - Classification binaire (chat/chien) → 1 neurone (probabilité).
     - Classification multi-classes (10 chiffres) → 10 neurones (probabilité par chiffre).



2. Les Neurones : Unités de Base

Fonctionnement simplifié d'un neurone :

1.Réception : Le neurone reçoit des signaux des neurones de la couche précédente.

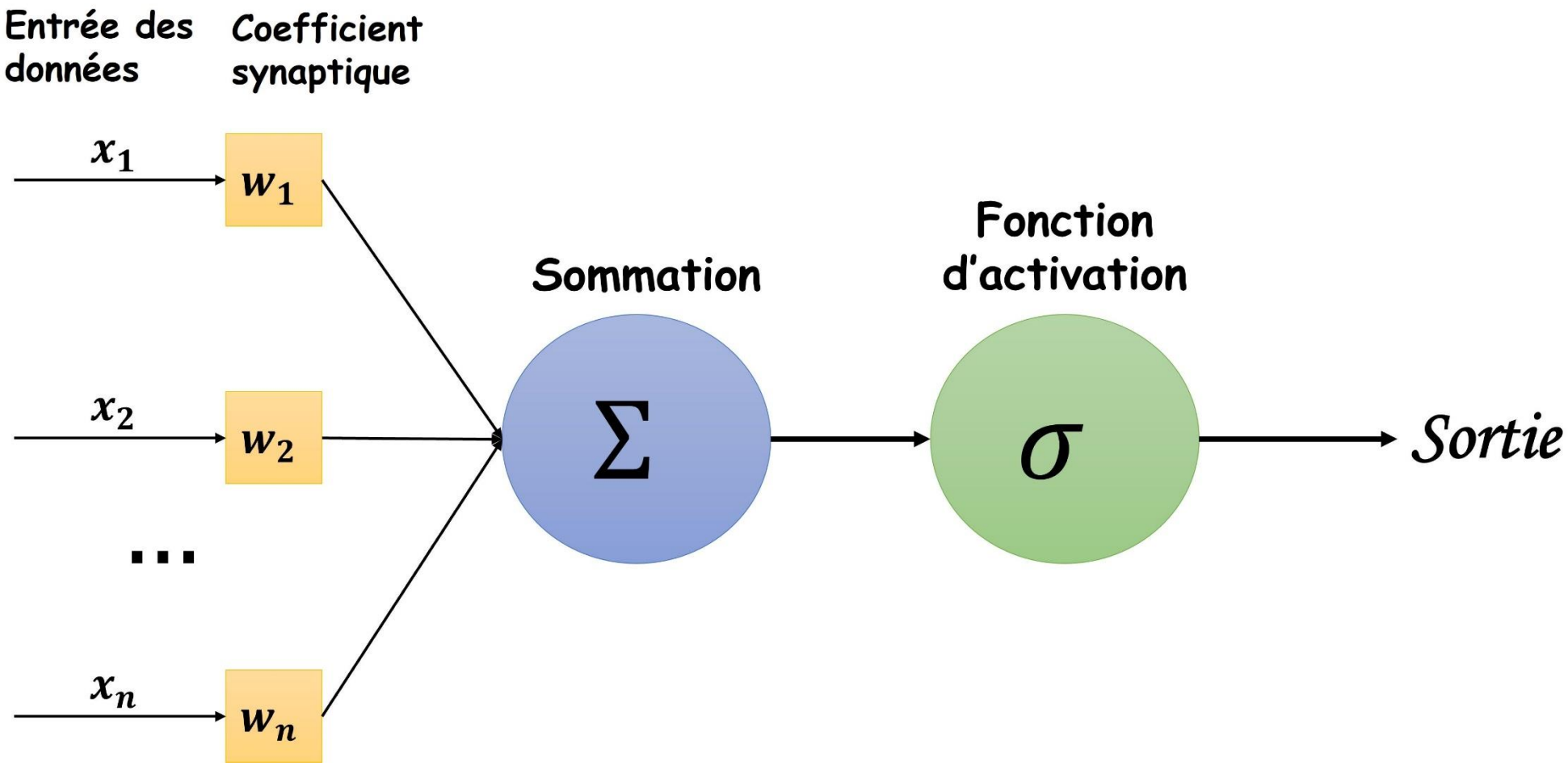
2.Pondération : Il attribue une "importance" à chaque signal (ex. : un contour horizontal est plus pertinent qu'un pixel noir isolé).

3.Activation : Détermine si le neurone transmet l'information à la couche suivante.

Analogie :

•Imaginez un comité qui vote pour prendre une décision :

- Chaque membre (neurone) donne son avis avec un certain poids.
- Si la somme des avis dépasse un seuil, la décision est transmise.



3. Les Poids (Weights)

Rôle : Les poids déterminent **l'importance** de chaque entrée dans la décision du neurone.

Analogie : Imaginez un jury qui note un candidat :

•**Entrées** = Critères (ex: créativité, technique, présentation).

•**Poids** = Importance donnée à chaque critère (ex: créativité x2, technique x1, présentation x0.5).

→ Le total dépend des poids !

Exemple Concret :

•**Prédire le prix d'une maison** :

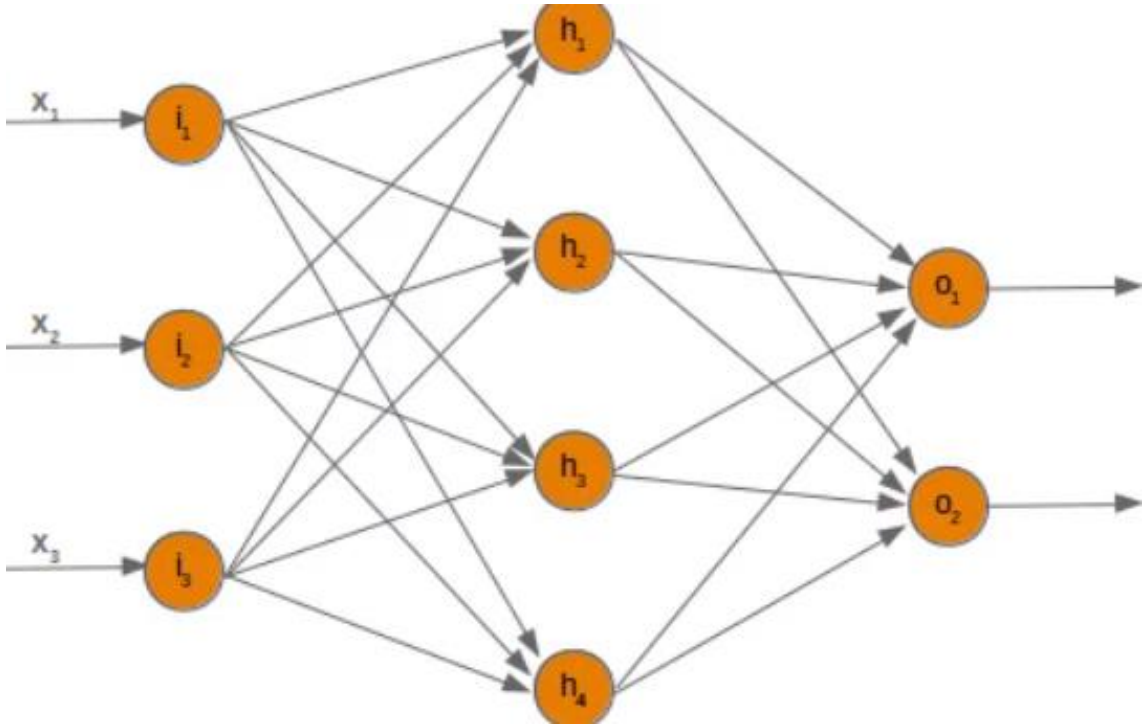
- Entrées : Surface (100m²), Nombre de chambres (3).
- Poids : Surface x 500€, Chambres x 20 000€.

  → Prix prédit = (100 x 500) + (3 x 20 000) = 65 000€.

Pourquoi c'est important ?

•Les poids **ajustent l'impact** de chaque information.

•**Apprentissage** : Le réseau modifie les poids pour minimiser l'erreur (ex: "Trop d'importance à la surface, réduisons ce poids !").



4. Le Biais (Bias)

Rôle : Le biais est une sorte de valeur supplémentaire qu'on ajoute dans un neurone pour aider le modèle à mieux apprendre.

Analogie :

Imagine un interrupteur de lumière qui ne s'allume que si tu appuies fort (par exemple, avec un poids d'au moins 10 kg).

Mais tu veux qu'il s'allume même avec un petit poids, disons 2 kg.

→ Tu ajoutes un ressort dans l'interrupteur pour le rendre plus sensible.

Ce ressort, c'est le biais : il aide à déclencher l'action, même quand l'entrée est faible.

Pourquoi c'est important ?

•Sans biais, le réseau ne peut pas apprendre des motifs **non centrés sur zéro**.

- Un motif centré sur zéro : c'est quand la sortie est 0 lorsque les entrées sont 0.
- Un motif non centré sur zéro : c'est quand la sortie n'est pas 0 même quand les entrées sont nulles ou faibles.

Même si toutes les entrées sont nulles, le biais (-1) permet une prédiction non nulle.

Cas des features (caractéristiques) où zéro a un sens

Imaginons que tu entraînes un réseau de neurones pour prédire si quelqu'un va acheter un produit.

Tu as une feature comme :

Nombre d'enfants = 0, 1, 2, ...

Ici, 0 n'est pas une absence de donnée, c'est une valeur réelle et significative : la personne n'a pas d'enfants.

Problème sans biais

Supposons que tu n'as pas de biais.

Si l'entrée est 0 (par exemple, 0 enfants), alors le neurone reçoit rien du tout de cette feature, comme si elle n'existait pas.

Le résultat de ce neurone pourrait être 0, donc pas d'activation, alors que le fait de ne pas avoir d'enfants est une information importante !

Solution : ajouter un biais

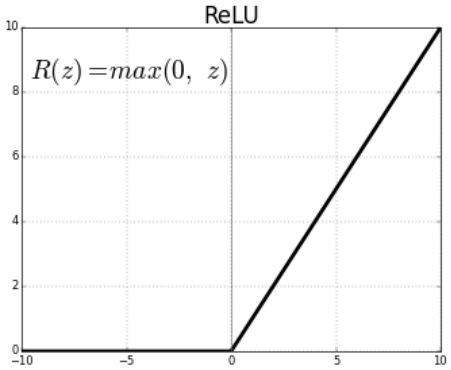
Le biais permet au neurone de s'activer même quand l'entrée est 0.

Il apporte une base d'activation minimale.

5. Les Fonctions d'Activation

Elles ajoutent de la **non-linéarité**, permettant au réseau d'apprendre des relations complexes.

Exemples de fonctions :

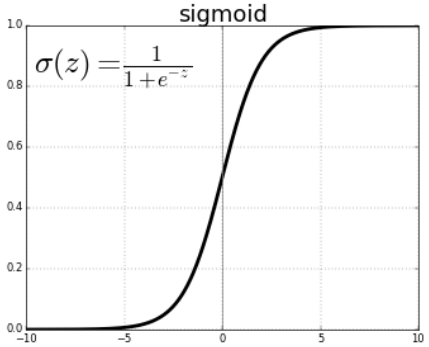


•**ReLU (Rectified Linear Unit)** :

- "Si le signal est positif, gardez-le ; sinon, ignorez-le."
- Utilisée dans la plupart des réseaux modernes.

•**Sigmoïde** :

- Transforme les valeurs en probabilités (entre 0 et 1).
- Exemple : "Quelle est la probabilité que cette image soit un chat ?"



Analogie :

•Comme un interrupteur qui s'allume uniquement si la lumière est suffisamment forte.

PARTIE 1

6. Comment ça Marche Ensemble ?

Formule Simplifiée :

Sortie du neurone = (Entrée$_1$ x Poids$_1$) + (Entrée$_2$ x Poids$_2$) + ... + Biais

Ensuite, cette valeur passe par une **fonction d'activation** (ex: ReLU, sigmoïde) pour donner la prédiction finale.

Exemple :

- **Problème** : Détecter si un animal est un chat (1) ou non (0).
- **Entrées** : Poils (1=oui), Taille (0.3=petit), Moustaches (1=oui).
- **Poids** : Poils x 2, Taille x (-1), Moustaches x 3.
- **Biais** = -1.

→ Calcul : (1x2) + (0.3x-1) + (1x3) + (-1) = **3.7**

→ Fonction sigmoïde : 3.7 → 0.975 → "Chat à 97,5%".

7. Pourquoi Plusieurs Neurones dans une Couche ?

Rôle d'un neurone : Un neurone est comme un **détective spécialisé** qui cherche un **type de motif précis** dans les données.

Exemple avec une Image de Chat :

- **1 neurone** : Peut détecter les **bords horizontaux** (ex: le sol).
- **10 neurones** :

→ 1 détecte les **bords horizontaux**,

→ 1 les **bords verticaux** (ex: pattes),

→ 1 les **couleurs sombres** (ex: yeux),

→ etc.

Conclusion :

- Plus il y a de neurones, plus le réseau peut **repérer de détails variés**.
- **Sans assez de neurones** : Le réseau est "myope" et ne voit pas tous les motifs.

Analogie :

Une équipe de médecins :

- **1 médecin** → Peut diagnostiquer quelques maladies.
- **10 médecins** spécialisés → Diagnostiquent plus de problèmes avec précision.

7. Pourquoi Plusieurs Couches ?

Rôle des couches : Les couches permettent de **combiner les motifs simples en motifs complexes**.

Exemple Hiérarchique :

- **Couche 1 (Entrée)** : Détecte des **lignes et courbes** (bords, points).
- **Couche 2** : Combine les lignes en **formes** (cercles, triangles → yeux, nez).
- **Couche 3** : Combine les formes en **objets** (visage, pattes).
- **Couche de sortie** : Reconnaît **"Chat"** ou **"Chien"**.

Pourquoi ça marche ?

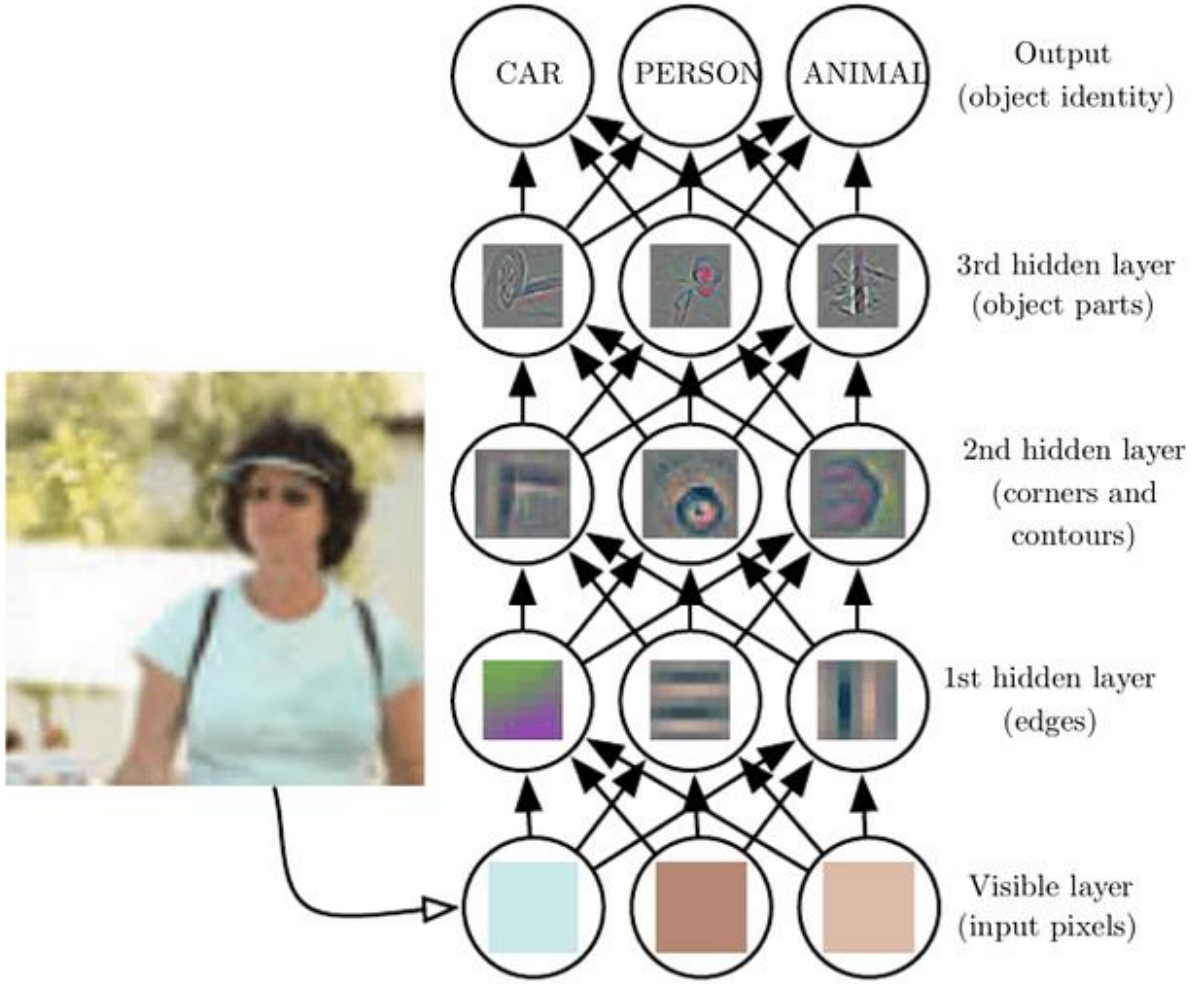
- **Premières couches** : Apprennent des motifs **simples**.
- **Couches profondes** : Apprennent des concepts **abstraits** (ex: "museau", "pelage").

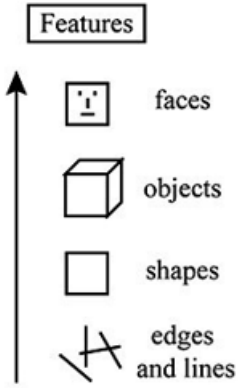
Analogie :

Une usine de montage :

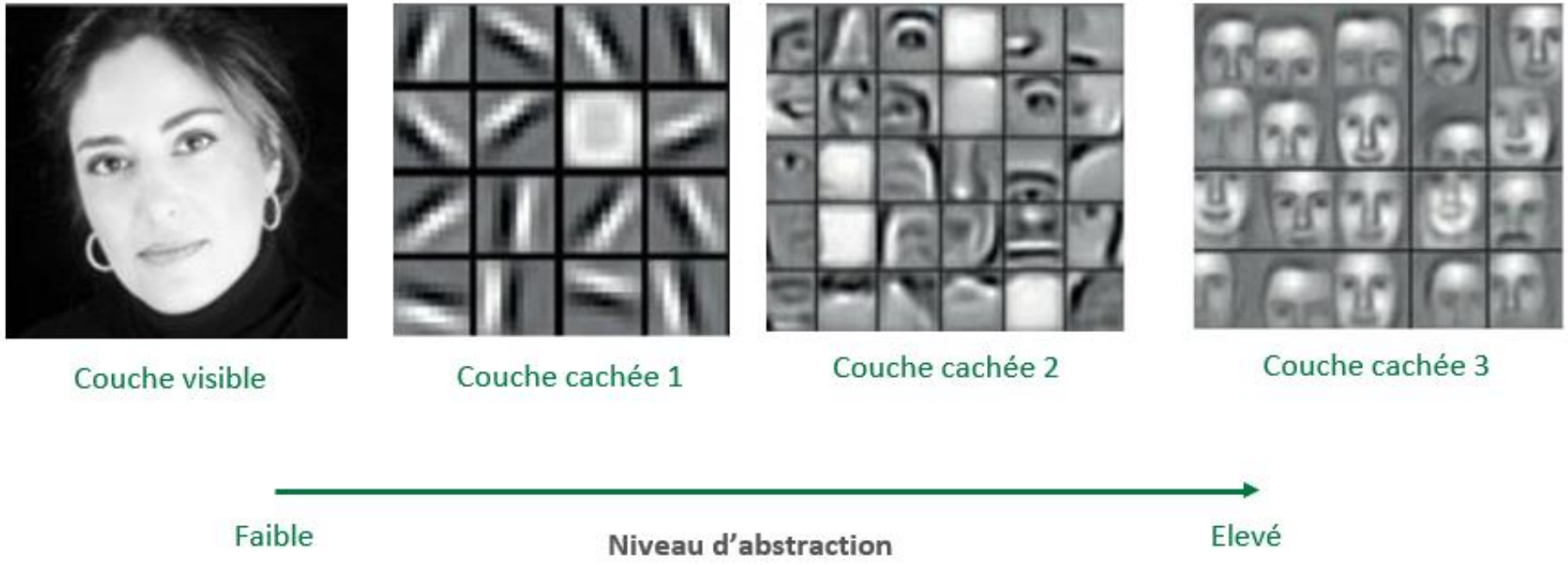
- **Étape 1** : Assemblage des pièces de base (roues, moteur).
- **Étape 2** : Montage du châssis.
- **Étape 3** : Peinture et finitions.

→ Résultat final : Une voiture complète.





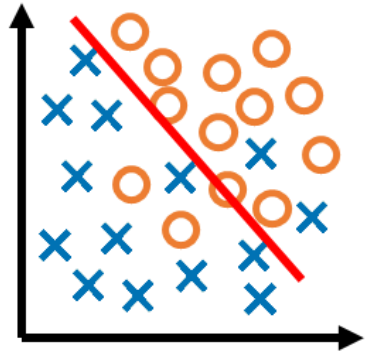
PARTIE 1



8. Que se Passe-t-il si on n'a Pas Assez de Neurones/Couches ?

•**Underfitting (Sous-apprentissage)** :

- Le réseau est trop "bête" pour comprendre les données.
- Exemple : Un réseau avec 2 neurones essayant de reconnaître des visages.
- **Résultat** : Erreurs élevées, même sur les données d'entraînement.

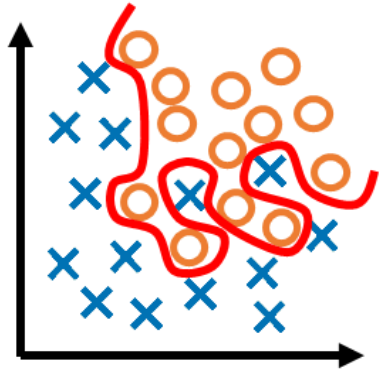


Sous apprentissage

9. Que se Passe-t-il si on a beaucoup de Neurones/Couches ?

•**Overfitting (Sur-apprentissage)** :

- Le réseau **mémorise** les données d'entraînement au lieu d'apprendre.
- Exemple : Un réseau avec 1000 couches pour prédire le prix d'une maison (trop complexe).
- **Résultat** : Performances nulles sur de nouvelles données.



Surapprentisage

PARTIE 1

10. Entraînement du Réseau : Comment ça marche ?

Processus en 3 étapes :

1.Forward Propagation :

- Les données traversent le réseau de l'entrée à la sortie → génère une prédiction.
- **Exemple** : Le réseau voit une image et dit "80% de chances que ce soit un chat".

2.Calcul de l'Erreur :

- Compare la prédiction à la vérité terrain (ex. : l'image était bien un chat).
- **Fonction de perte (Loss)** : Mesure combien la prédiction est "fausse".

3.Rétropropagation (Backpropagation) :

- Ajuste les poids des neurones pour réduire l'erreur.
- **Analogie** : Comme un chef cuisinier qui ajuste sa recette après avoir goûté le plat.

Apprentissage des Poids et Biais

Initialisation : Au départ, les poids et biais sont choisis **aléatoirement** (ex: entre -1 et 1).

•**Backward Propagation** : Le réseau utilise l'erreur (loss) pour ajuster les poids et biais :

- "Si j'augmente ce poids, l'erreur diminue ?" → Si oui, on l'augmente.
- **Learning Rate** : Contrôle la taille des ajustements.

Analogie :

•**Poids/Biais** = Réglages d'une radio.

•**Apprentissage** = Tourner les boutons pour avoir le meilleur son (moins de bruit = moins d'erreur).

11. Forward Propagation (Propagation Avant)

C’est quoi ?
C’est le processus où les données entrent dans le réseau de neurones, passent à travers les couches, et produisent une prédiction.

Analogie :
Imaginez une chaîne de montage :

- **Entrée** → Des pièces brutes (exemple : une image de chat).
- **Couches cachées** → Des ouvriers qui assemblent les pièces (exemple : détectent des contours, des formes).
- **Sortie** → Le produit final (exemple : "Chat" à 80% de confiance).

Pas de calculs ! On suit juste le flux des données vers l’avant.

12. Erreur et Loss (Perte)

- **Erreur** : La différence entre la prédiction du réseau et la vérité terrain.
  Exemple : Le réseau dit "Chat à 80%", mais c'est un chien (donc chat = 0%) → l'erreur sur "chat" est de 80%
  Erreur = |Prédiction - Valeur réelle|

- **Loss** : Une mesure globale de "à quel point le réseau s'est trompé".
  Exemple : Moyenne des erreurs sur 100 images → Loss = 30%.

Analogie :
La Loss est comme une "note" que le réseau reçoit. **Objectif :** Minimiser cette note.

13. Backward Propagation (Rétropropagation)

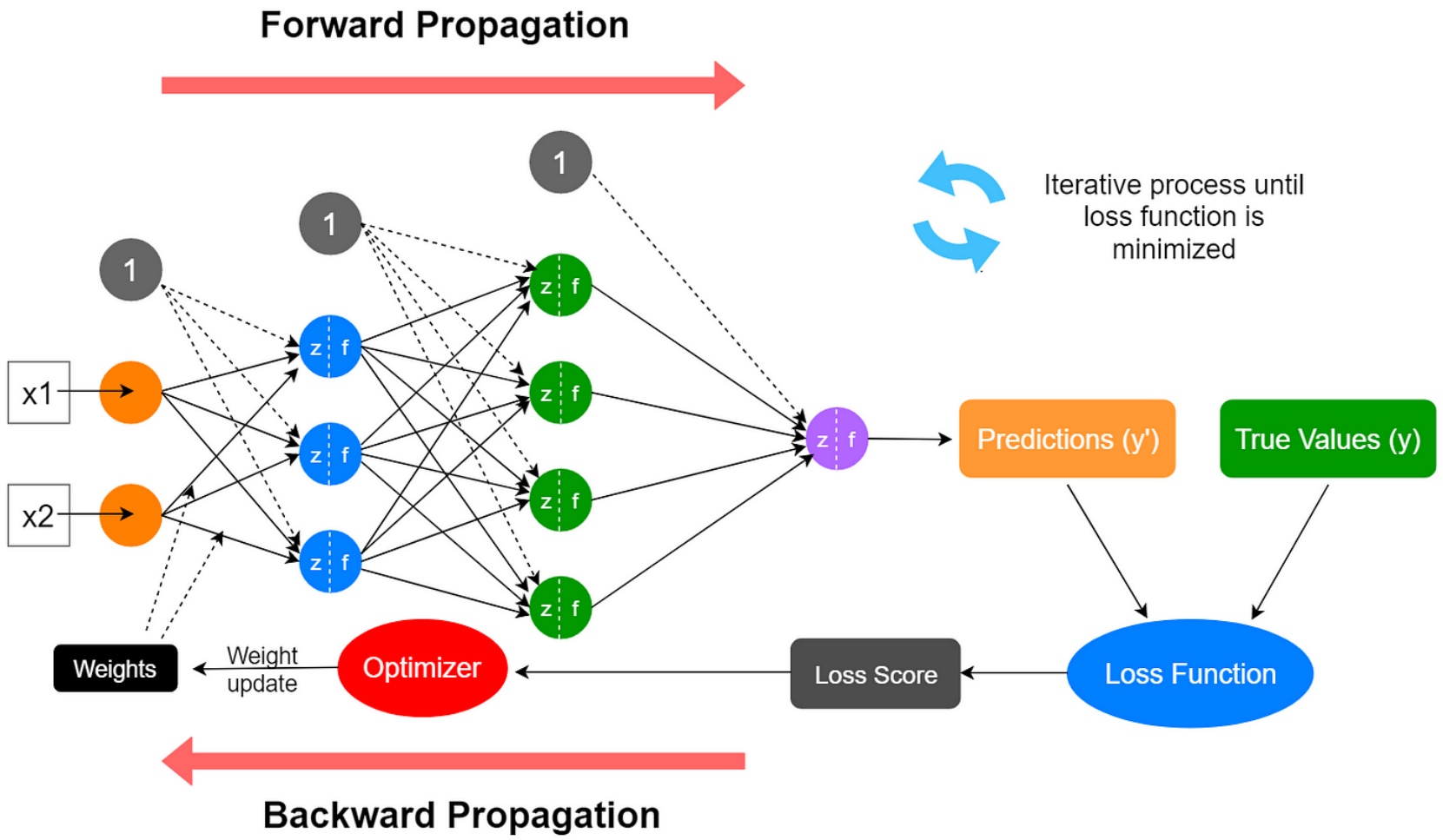
C’est quoi ?

Un mécanisme pour **corriger les erreurs** en ajustant les paramètres du réseau (les "poids").

Analogie :

•Après un examen raté, vous analysez **pourquoi** vous avez faux.

•Le réseau fait pareil : il remonte de la sortie vers l’entrée pour identifier **quels neurones ont causé l’erreur**.



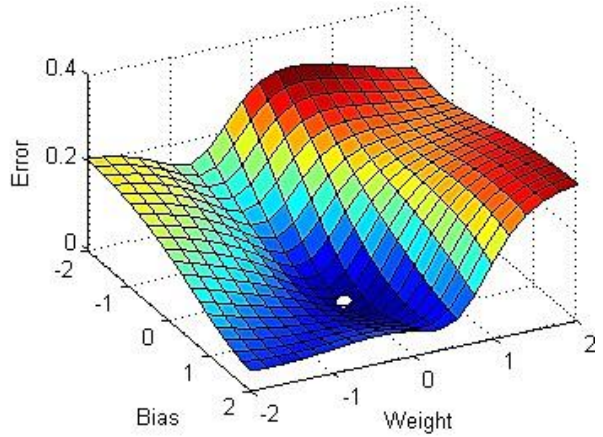
14. Descente de Gradient (Gradient Descent)

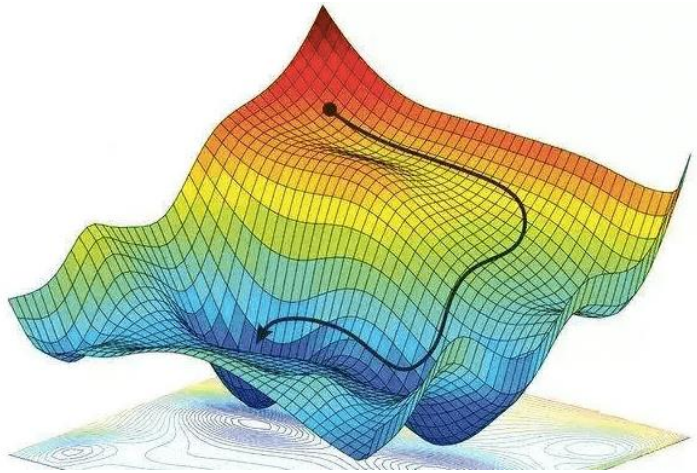
C'est quoi ?
Une méthode pour **minimiser la Loss** en ajustant les poids pas à pas.

Analogie :
Imaginez être en montagne dans le brouillard et vouloir descendre au plus bas :

- **Gradient** → La pente sous vos pieds (indique la direction à suivre).
- **Descente** → Vous faites un petit pas vers le bas.

Objectif : Trouver le point le plus bas (la Loss minimale) en suivant la pente.





PARTIE 1

15. Learning Rate (Taux d'Apprentissage)

C'est quoi ?

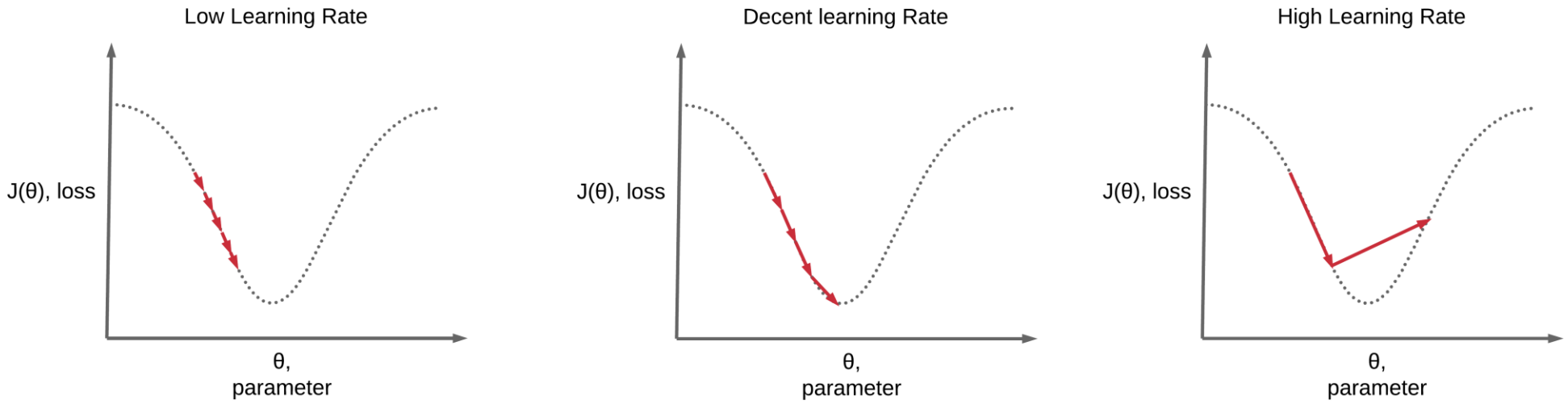
La taille des pas que vous faites pendant la descente de gradient.

Analogie :

- **Learning Rate trop grand** → Vous faites des pas géants et risquez de dépasser le point le plus bas.
- **Learning Rate trop petit** → Vous avancez très lentement (temps d'entraînement long).

Exemple :

- Learning Rate = 0.1 → Petits pas prudents.
- Learning Rate = 0.9 → Gros pas risqués.



16. Résumé

1.Forward : Données → Prédiction.

2.Loss : Calcule "à quel point c'est faux".

3.Backward : Identifie les neurones responsables.

4.Descente de Gradient : Ajuste les poids avec le learning rate.

Input → [Forward] → Prediction → [Loss] → [Backward] → [Gradient Descent] → Répéter!

17. Exemple

Problème : Prédire si un animal est un chat ou un chien.

1.Forward : Le réseau voit une image → Prédit "Chat à 80%".

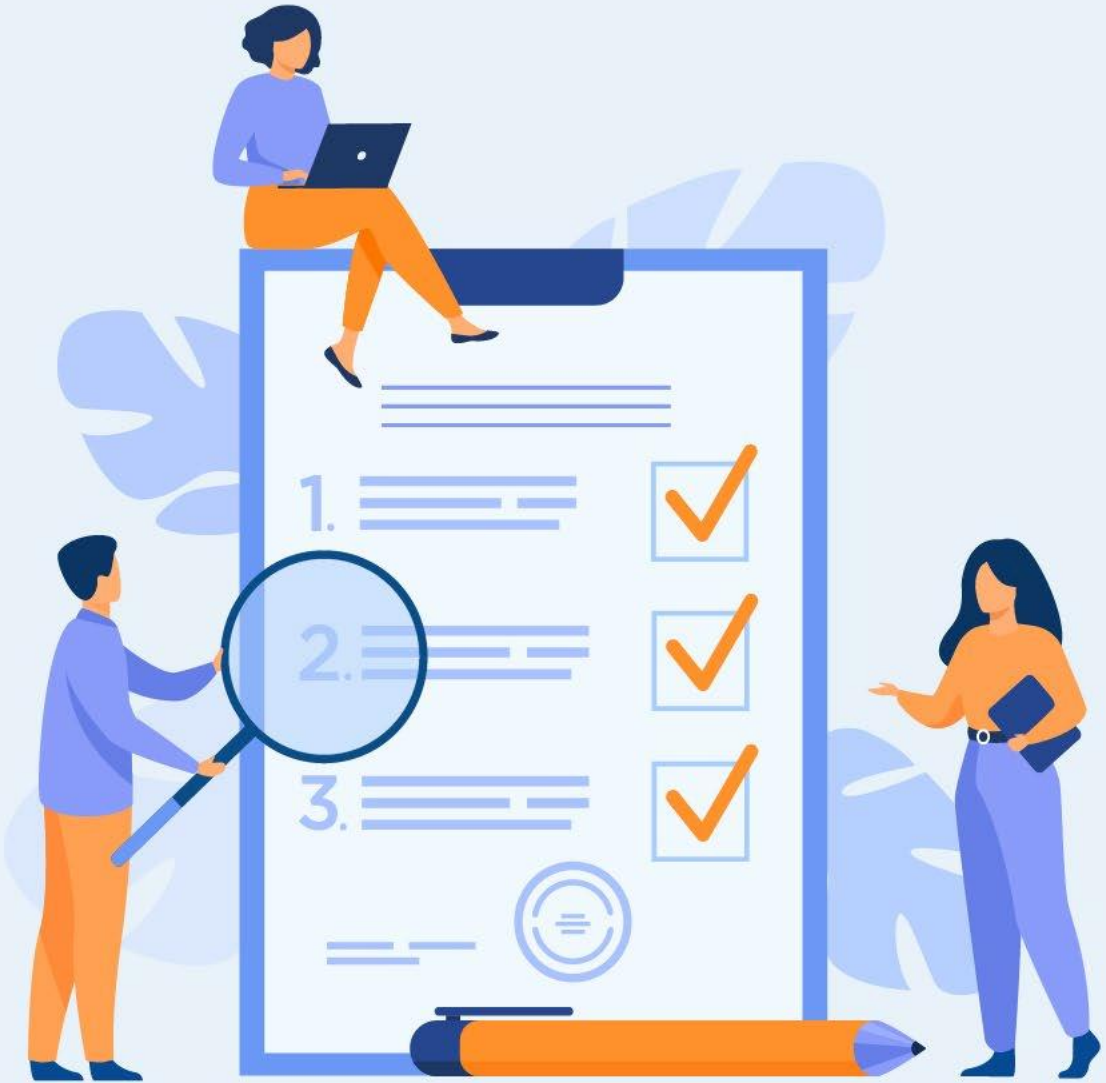
2.Loss : C'était un chien → Loss = 80%.

3.Backward : Le réseau se dit : "J'ai trop regardé les oreilles pointues, je dois réduire leur importance".

4.Descente de Gradient : Il ajuste les poids liés aux oreilles (learning rate détermine la force de l'ajustement).

Attention!! :

Le learning rate est hyper important ! Trop grand = instable, trop petit = lent. On le teste souvent par essai-erreur.



CHAPITRE 2

Keras et les Paramètres Clés

Ce que vous allez apprendre dans ce chapitre :

- Différencier entre l'apprentissage automatique et profond
- Faire l'analogie entre un réseau de neurone biologique et artificiel
- Connaître le principe de la descente de gradient
- Découvrir le perceptron et le perceptron multicouche

10 heures

1. Keras, c'est quoi ?

•**Keras** est une bibliothèque Python pour créer des réseaux de neurones **rapidement**, comme un Lego de l'IA.

•**Avantage** : Pas besoin de coder les maths derrière (forward/backward propagation, gradients…), Keras le fait automatiquement.

2. Les Paramètres à Comprendre

A. activation (Fonction d'Activation)

À quoi ça sert ?

Détermine si un neurone doit "s'allumer" (ex: 1) ou non (ex: 0) en fonction du signal reçu.

Choix courants :

- relu (*Rectified Linear Unit*) :
  - **Pourquoi ?** Simple et efficace pour la plupart des couches cachées.
  - **Effet** : Si le signal est négatif → 0. Positif → Garde la valeur.
- sigmoid ou tanh :
  - **Pourquoi ?** Pour les problèmes de classification **oui/non** (ex: "Chat ou chien ?").
  - **Effet** : Transforme les valeurs en probabilités entre 0 et 1 (sigmoid).
- softmax :
  - **Pourquoi ?** Pour la classification **multi-classes** (ex: "Chat, Chien, Oiseau").
  - **Effet** : Donne des probabilités qui totalisent 100% (ex: [0.7, 0.2, 0.1]).

```
model.add(Dense(32, activation='relu'))  # Couche cachée avec ReLU
model.add(Dense(1, activation='sigmoid'))  # Sortie binaire (oui/non)
```

PARTIE 1

B. optimizer (Optimiseur)

À quoi ça sert ?

Choisit **comment** le réseau ajuste ses poids pendant l'entraînement (descente de gradient améliorée).

Choix courants :

- adam (*Adaptive Moment Estimation*) :
  - **Pourquoi ?** Le plus populaire ! Adapte automatiquement le learning rate.
  - **Utilisation** : Par défaut pour la plupart des cas.
- sgd (*Stochastic Gradient Descent*) :
  - **Pourquoi ?** Simple, mais nécessite de régler manuellement le learning rate.
  - **Utilisation** : Si vous voulez contrôler précisément l'apprentissage.

```
model.compile(optimizer='adam', loss='binary_crossentropy')
```

C. loss (Fonction de Perte)

À quoi ça sert ?

Mesure **à quel point le réseau se trompe** (comme la "note" du réseau).

Choix courants :

- binary_crossentropy :
  - **Pour** la classification **binaire** (ex: "Malade" vs "En bonne santé").
- categorical_crossentropy :
  - **Pour** la classification **multi-classes** (ex: "Chat, Chien, Oiseau").
- mse (*Mean Squared Error*) :
  - **Pour** la **régression** (ex: Prédire un prix, une température).

```
model.compile(optimizer='adam', loss='binary_crossentropy')
```

D. epochs (Nombre d'Époques)

À quoi ça sert ?
Définit **combien de fois** le réseau voit **l'ensemble du jeu d'entraînement**.

Analogie :

•Si vous étudiez un livre pour un examen :

- **1 epoch** = Lire le livre **1 fois**.
- **10 epochs** = Le relire **10 fois**.

Attention !

•Trop d'epochs → Le réseau **mémorise** les données (*overfitting*).

•Pas assez → Le réseau ne comprend pas (*underfitting*).

Astuce : Commencez avec 10-50 epochs, ajustez selon les résultats.

E. batch_size (Taille de Lot)

À quoi ça sert ?

Définit **combien d'exemples** le réseau voit **avant de mettre à jour ses poids**.

Analogie :

•Si vous révisez 100 exercices de maths :

- **batch_size=10** → Vous lisez 10 exercices, puis corrigez vos erreurs.
- **batch_size=100** → Vous lisez tous les 100, puis corrigez.

Choix courants :

•**batch_size=32** : Une valeur standard (bon compromis vitesse/précision).

•**batch_size=64, 128** : Pour aller plus vite si vous avez beaucoup de données.

3. Comment Choisir ces Paramètres ?

Règles de Base :

1.activation :

- Couches cachées → relu.
- Sortie :
  - Classification binaire → sigmoid.
  - Multi-classes → softmax.
  - Régression → **Pas d'activation** (ou linear).

2.optimizer : Commencez avec adam.

3.loss :

- Régression → mse.
- Classification binaire → binary_crossentropy.
- Multi-classes → categorical_crossentropy.

4.epochs :

- Démarrez avec **20 epochs**, observez si la Loss baisse.
- Si la Loss stagne → Arrêtez plus tôt (*Early Stopping*).

5.batch_size :

- Utilisez **32, 64, ou 128** selon la mémoire de votre ordinateur.

4. Choisir le Nombre de Couches d'Entrée et de Sortie

A. Couche d'Entrée (Input Layer)

Rôle : C'est la porte d'entrée des données dans le réseau.

Comment choisir sa taille ?

•**La règle d'or** : **1 neurone = 1 caractéristique (feature)** de vos données.

Exemple :

- Si vos données ont **3 features** (ex: âge, salaire, taille) → **3 neurones en entrée**.
- Si vous travaillez avec des images de **28x28 pixels** → **784 neurones** (car 28×28 = 784).

Cas Particuliers :

•**Images** : On "déplie" l'image en une liste de pixels (ex: 784 pour du 28x28).

•**Texte** : Chaque mot ou lettre est encodé numériquement (ex: 1 neurone par dimension d'encodage).

Piège : Ne pas confondre **features** et **exemples** !

•**Exemple** : 1000 images de 28x28 → **Input layer = 784 neurones** (pas 1000 !).

B. Couche de Sortie (Output Layer)

Rôle : Donne la réponse finale du réseau (prédiction).

Comment choisir sa taille ?

Tout dépend du **type de problème** :

Type de Problème	Nombre de Neurones de Sortie	Activation
Régression (Prédire un nombre)	**1 neurone** (ex: prix d'une maison)	Aucune (linear)
Classification Binaire (Oui/Non)	**1 neurone** (ex: "chien" à 70%)	sigmoid
Classification Multiclasse (Chat/Chien/Oiseau)	**1 neurone par classe** (ex: 3 neurones)	softmax

C. Exemples Concrets :

1. **Prédire le prix d'une maison** (régression) → 1 neurone de sortie.
2. **Détecter un spam** (binaire) → 1 neurone avec sigmoid.
3. **Reconnaître un chiffre de 0 à 9** (multiclasse) → 10 neurones avec softmax.

Pièges :

- Utiliser softmax pour un problème binaire → **Erreur** ! On utilise sigmoid.
- Mettre 2 neurones pour un problème binaire → Possible, mais inutilement compliqué.

D. Comment Vérifier ?

Entrée : Regardez la forme (shape) de vos données (X_train.shape).

- Si X_train a 100 exemples avec 10 features → input_shape=(10,).

Sortie : Regardez la forme de vos étiquettes (y_train.shape).

- Si y_train contient des classes [0, 1, 2] pour 3 classes → 3 neurones en sortie.

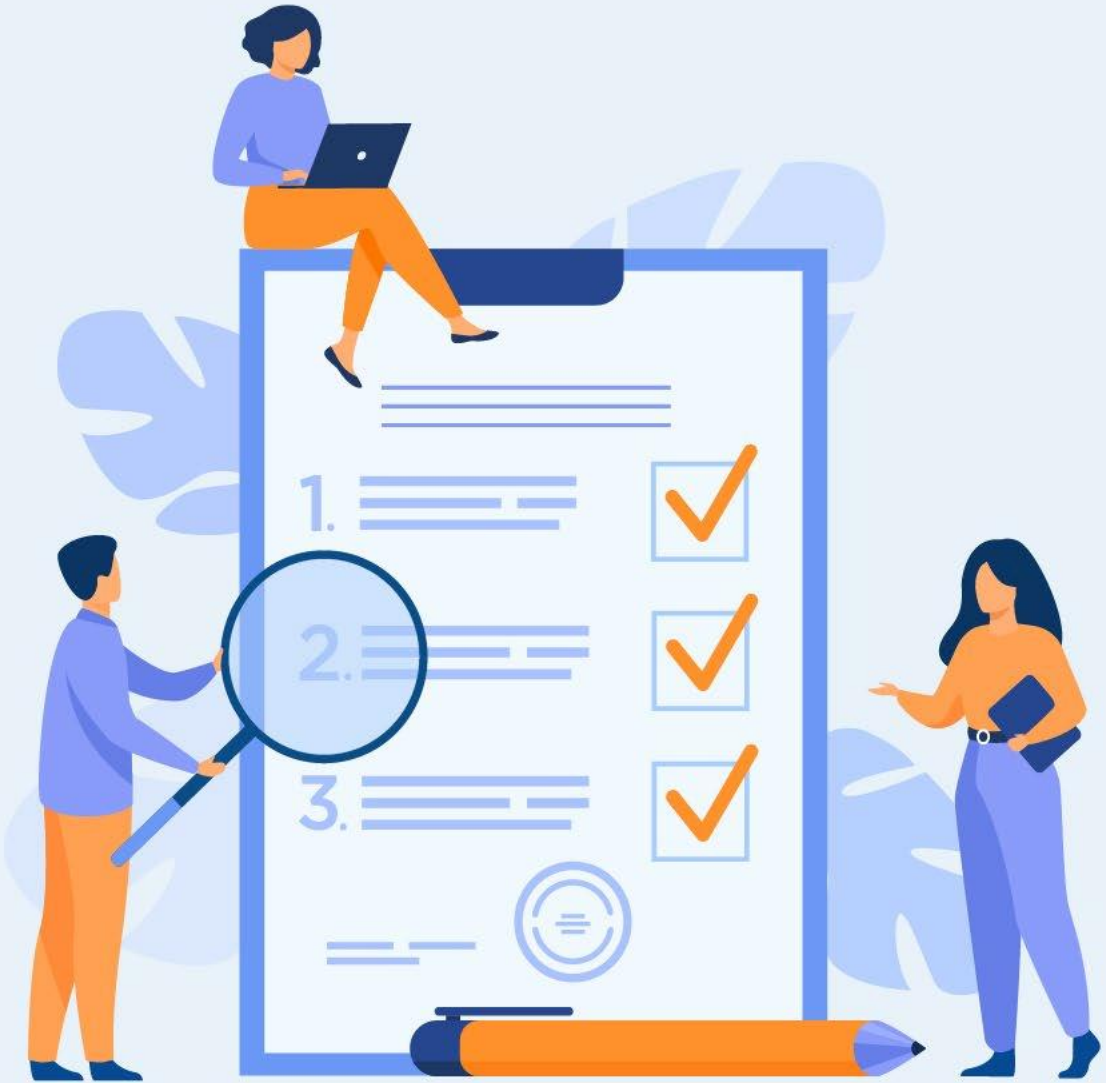
5. Exemple Complet

Problème : Classification Chat vs Chien (binaire).

```python
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(784,)))  # Couche cachée
model.add(Dense(1, activation='sigmoid'))  # Sortie binaire

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Entraînement
history = model.fit(X_train, y_train,
                    epochs=20,
                    batch_size=32,
                    validation_data=(X_test, y_test))
```

Explications :

- **activation**='relu' pour les neurones cachés.
- **activation**='sigmoid' pour la sortie (probabilité chat/chien).
- **loss**='binary_crossentropy' car classification binaire.
- **epochs**=20 : On entraîne 20 fois sur toutes les données.
- **batch**_size=32 : Met à jour les poids après chaque groupe de 32 images.



CHAPITRE 2

L’importance de la Normalisation et la Standardisation pour les Réseaux de Neurones

10 heures

1. Pourquoi c'est crucial ?

Les réseaux de neurones sont comme des chefs cuisiniers :

•Si les **ingrédients** (données) ne sont pas préparés de manière cohérente (ex: carottes en dés vs carottes entières), le plat (modèle) sera mal cuit.

•**Problème** : Les données brutes ont souvent des **échelles très différentes** (ex: âge = 0-100, salaire = 0-10 000€).

Conséquences sans Normalisation/Standardisation :

•Le réseau passe plus de temps à "comprendre" les échelles qu'à apprendre les motifs.

•L'entraînement est **lent** ou **instable** (ex: le réseau donne trop d'importance aux grandes valeurs).

2. Normalisation (Min-Max Scaling)

C'est quoi ? Transformer les données pour qu'elles soient **entre 0 et 1**.
Exemple :

- Avant : Âge = [20, 40, 60]
- Après : [0.0, 0.5, 1.0]

Quand l'utiliser ?

- Si les données ont des **limites naturelles** (ex: pixels d'une image = 0 à 255).
- Si la distribution des données **n'est pas en forme de cloche** (non gaussienne).

3. Standardisation (Z-Score Scaling)

C'est quoi ? Centrer les données autour de **0** avec un écart-type de **1**.
Exemple :

- Avant : Salaire = [30 000€, 50 000€, 70 000€]
- Après : [-1.0, 0.0, 1.0]

Quand l'utiliser ?

- Si les données suivent une **distribution en cloche** (gaussienne).
- Si les données ont des **valeurs extrêmes** (ex: revenus très élevés).

4. Analogies

Normalisation :

Imaginez que vous redimensionnez toutes vos photos pour qu'elles rentrent dans un cadre (ex: 0-1).

•**Avantage** : Toutes les photos ont la même taille.

•**Inconvénient** : Si une photo a un élément très brillant (valeur extrême), elle sera "écrasée".

Standardisation :

Imaginez que vous ajustez toutes les températures d'une pièce autour de 20°C.

•**Avantage** : Les variations (ex: +5°C ou -3°C) sont comparables.

•**Inconvénient** : Ne garantit pas de bornes fixes (ex: peut aller de -10 à +30 après transformation).

5. Quand Choisir l'une ou l'autre ?

Choisir la Normalisation si :

- Vos données ne sont pas gaussiennes.
- Vous travaillez avec des **images** (pixels entre 0 et 255 → normalisez à 0-1).
- Vos données ont des **limites claires** (ex: notes scolaires entre 0 et 20).
- Vous utilisez des algorithmes sensibles aux **intervalles fixes** (ex: réseaux de neurones avec sigmoïde en sortie).

Choisir la Standardisation si :

- Vos données suivent approximativement une distribution normale (gaussienne).
- Vos données ont des **valeurs extrêmes** (ex: revenus, tailles de villes).
- Vous utilisez des méthodes basées sur les **distances** (ex: SVM, k-NN) ou des réseaux de neurones avec **ReLU**.

6. Impact sur les Réseaux de Neurones

Avantages Communs :

- **Convergence plus rapide** : Le réseau apprend plus vite car les données sont "sur la même échelle".
- **Évite les saturations** : Les fonctions d'activation (ex: sigmoïde, ReLU) fonctionnent mieux avec des données centrées/réduites.

Exemple Concret

- **Problème** : Prédire le prix d'une maison (surface = 50-200m², nombre de chambres = 1-5).
- **Sans prétraitement** : La surface (valeurs 50-200) domine le nombre de chambres (1-5).
- **Avec Standardisation** : Les deux caractéristiques contribuent équitablement.

7. Pièges à Éviter

- **Normaliser des données avec des outliers** → Risque de tout écraser entre 0 et 1.

  Exemple : Si 99% des données sont entre 0-100, mais une valeur à 10 000.

- **Standardiser des données non gaussiennes** → Pas grave, mais parfois moins efficace.