Comment l’IA « apprend » ? :

Quels déploiements dans nos métiers ?

Introduction simplifiée aux réseaux de neurones

EDF R&D TREE E3B 2026

Contexte

Généralement, quand on pense IA, on pense ChatGPT, Gemini, etc. : les LLM (Large Langage Models). Sur la base d'une requête **textuelle**, ils compilent des données **textuelles** et formulent une réponse **textuelle**. Bref, on discute avec un LLM.

Une piste, très différente des LLM, de l'utilisation de l'IA dans nos métiers portera sur **l'anticipation**.

A partir de datas historiques, les modèles – notamment les réseaux de neurones - apprendront à **prévoir** heure par heure l'évolution, par exemple, de la demande énergétique, d'une concentration de CO_2, d'une température, etc.

Une fois cette vision du futur établie, d'autres outils **simuleront différentes stratégies de pilotage** (ex : faut-il anticiper une montée en température, décaler des usages, etc.).

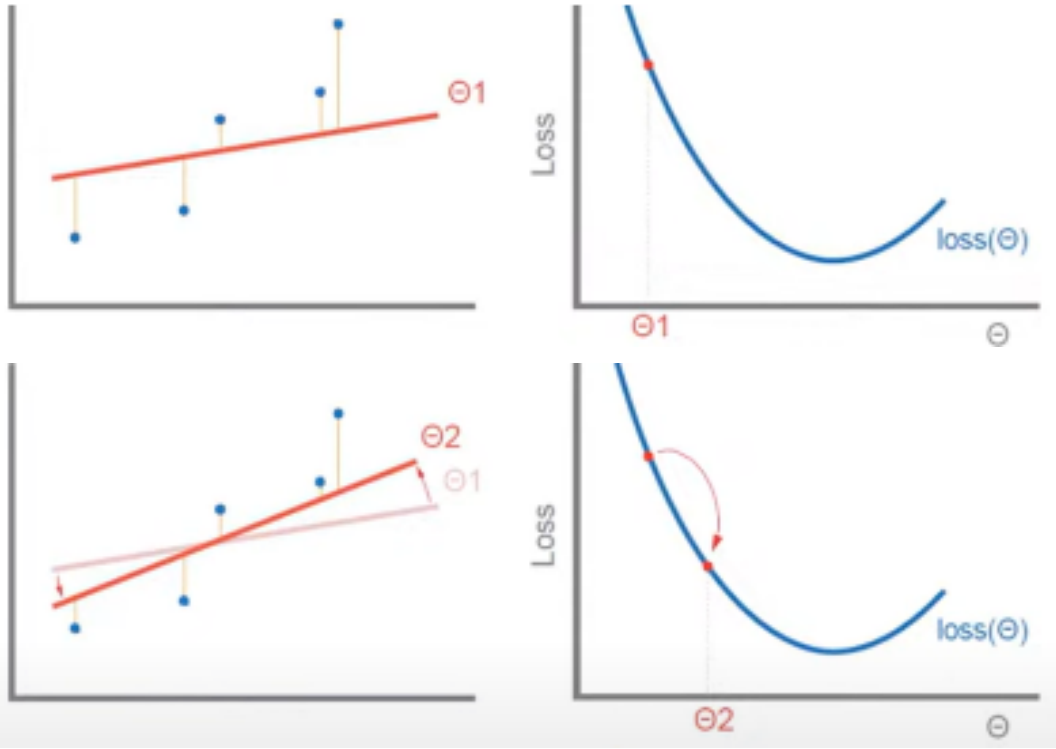
L'objectif sera alors de **choisir la stratégie optimale**, selon des critères énergétiques, économiques ou de confort.
Ce processus se répètera en continu pour réévaluer en permanence la meilleure stratégie, selon les nouvelles datas disponibles.

Cette boucle dynamique – **prévoir, simuler, optimiser, recommencer** – sera le cœur des futures IA dans nos métiers.

Observons justement comment un réseau de neurones apprend ?

Introduction simplifiée aux réseaux de neurones

Tout d'abord, partons d'un concept familier : la Régression Linéaire (RLi)



Objectif : tracer une droite qui s'ajuste au mieux à un nuage de points (données issues de capteurs)

On minimise l'écart entre les points et la droite, appelé **fonction de perte** (Loss)

L'**ajustement** se fait par itération : la **descente de gradient**. La droite retenue est celle qui minimise le Loss

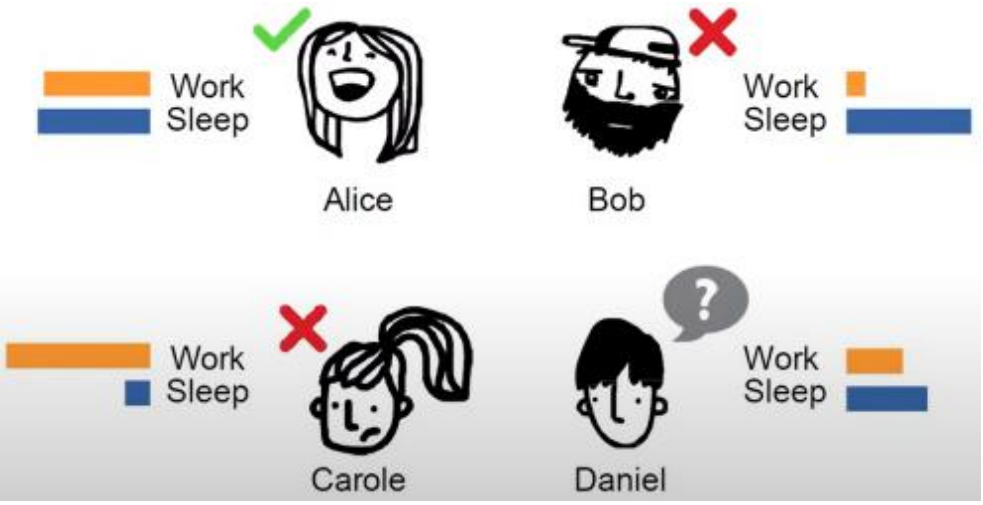
(ex. RLi avec Excel)

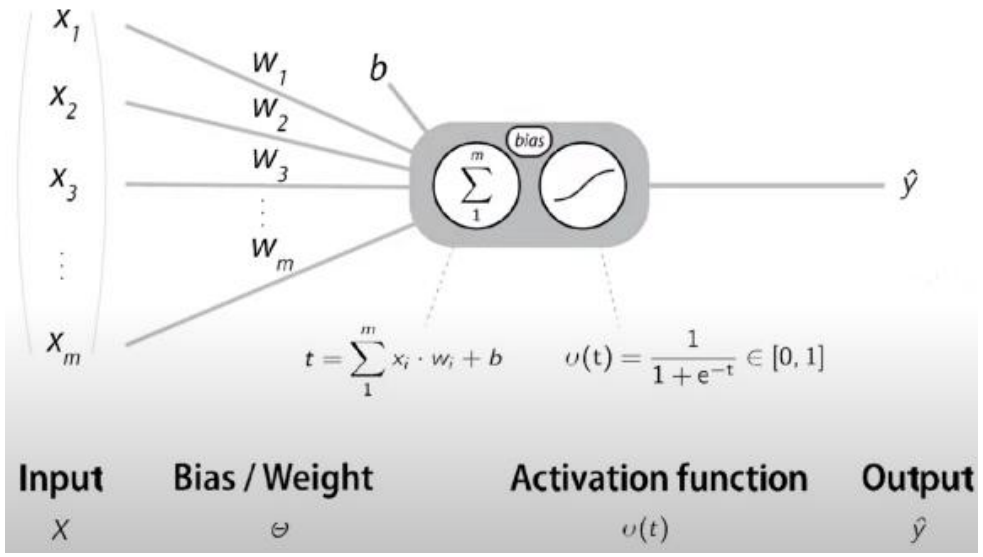
La RLi permet de décrire des relations assez simples (i.e y = ax + b), et qui ont effectivement une composante linéaire.

Mais comment faire si la sortie que l'on cherche à décrire n'est pas continue et prend des valeurs binaires (oui/non, on/off , fort/faible, petit/grand, etc.) ?

Introduction simplifiée aux réseaux de neurones

La Régression Logistique (RLo)





Daniel va-t-il réussir son examen selon ses temps de travail (x_1) et de sommeil (x_2)

$t = w_1.x_1 + w_2.x_2 + b$, où w_1 et w_2 sont des « poids » et b une constante appelée « biais ».

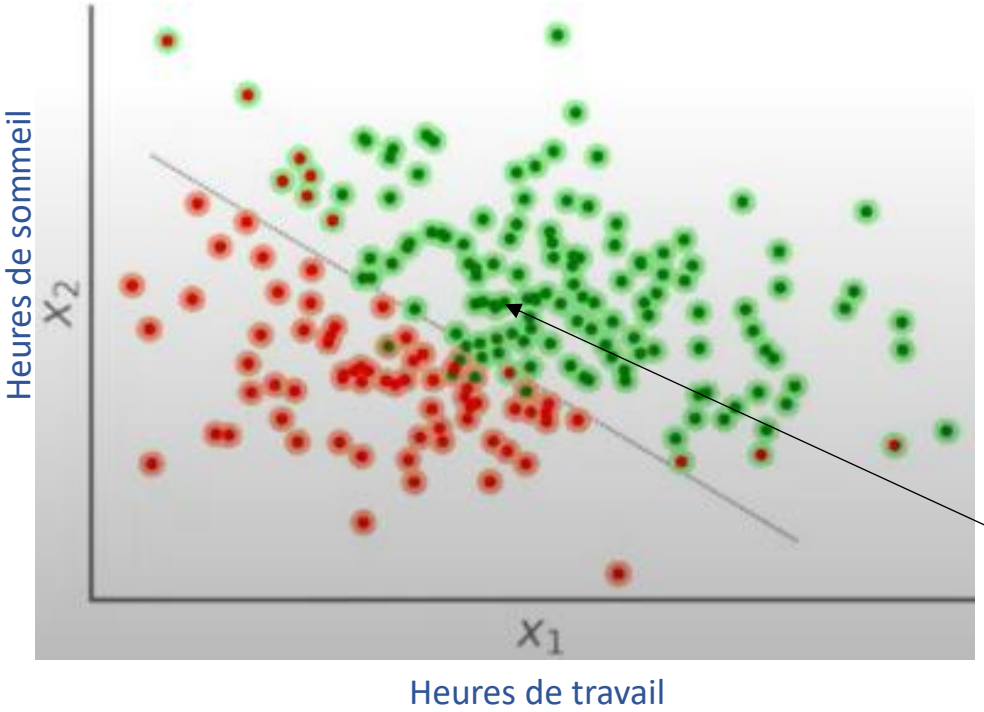
t est envoyée vers une fonction d'activation v(t) qui renvoie une valeur comprise entre 0 et 1.

Un étudiant sera classé « réussite » si v(t) >=0.5;
et « échec » dans le cas contraire.

Comme précédemment, la descente de gradient recherche les valeurs de poids et de biais qui minimisent l'erreur de prédiction sur la population d'étudiants : c'est la **phase dite d'entrainement** du modèle.

Introduction simplifiée aux réseaux de neurones

La Régression Logistique (RLo)



Ici, 92% des points sont correctement prédits.
Présence de « faux-réussite» (étudiants qui ont échoué) dans la zone verte
Présence de « faux-échecs» (étudiants qui ont réussi) dans la zone rouge

En positionnant les temps de travail et de sommeil de Daniel, nous connaissons alors sa probabilité de réussite. C'est la phase d'utilisation du modèle, dite d'inférence.

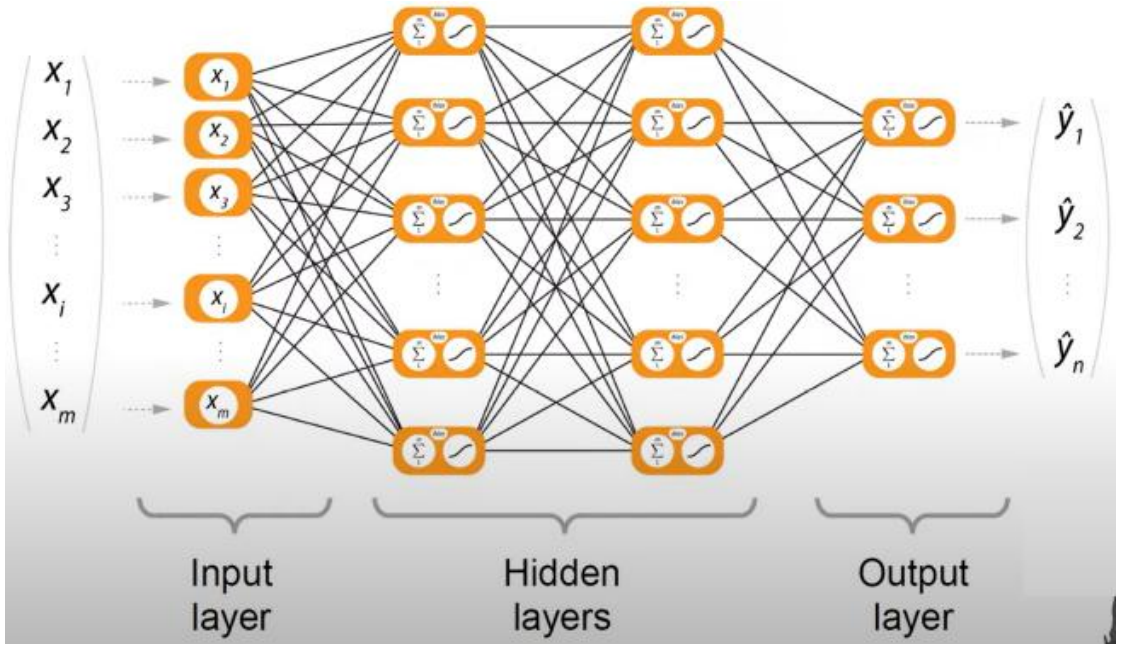
Daniel devrait probablement réussir

Subtilité importante, ici : les deux groupes sont séparés par…une droite. Ce type de modèle reste donc implicitement linéaire.

Comment faire pour rendre compte de structures de données non linéaires ?

Introduction simplifiée aux réseaux de neurones

Les réseaux de neurones profonds (DNN)



Chaque neurone effectue une opération similaire à celle du neurone unique : chaque connexion entre neurones a un poids et chaque neurone a un biais.

L'apprentissage se fait par itération : le réseau effectue un calcul avec le résultat attendu, puis ajuste tous les paramètres (en partant de la sortie vers l'entrée : rétropropagation du gradient)

Introduction simplifiée aux réseaux de neurones

Un débutant « débrouillard », avec un minimum de bagages, peut rapidement monter en compétence sur ces sujets.
Exemple d'itération avec un IA pour construire un modèle de prévision de charge de chauffage.

Je suis ingénieur en efficacité énergétique dans les bâtiments. Je débute sur les réseaux de neurones. Je cherche à créer un modèle qui estime, pour les 24 prochaines heures, la courbe de charge de chauffage d'un bâtiment (en kW) au pas horaire. Je dispose d'un historique de données sur une année où pour chaque puissance, je connais :

- La température extérieure ;
- La température de consigne ;
- Le jour de la semaine (1=lundi, 7 = dimanche)
- Le mois (janvier =1, décembre =12)
- L'heure (de 1 à 24).

Je veux créer un réseau de neurones qui me permette sur la base d'une prévision horaire de météo France sur les 24 prochaines heures, d'estimer la courbe de charge à venir.
Utilise Python. Mes données sont contenues dans le fichier « demo.xls ». Propose une méthode pour estimer la pertinence de la prédiction. Conseille moi sur la structure du réseau, notamment sur le nombre de couches.

3. Première approche recommandée : MLP

Entrées

Pour chaque heure :

- température extérieure
- consigne
- heure
- jour semaine
- mois

ET je vous conseille fortement d'ajouter :

- puissance à l'heure précédente
- moyenne glissante des 3 dernières heures
- éventuellement puissance J-1 même heure

Ces variables améliorent énormément les performances.

```
# ============================================================
# 5. CREATION DE VARIABLES D'INERTIE THERMIQUE
# ============================================================

# Puissance heure précédente
df["P_h-1"] = df["Puissance"].shift(1)

# Moyenne glissante sur 3h
df["P_roll3"] = df["Puissance"].rolling(window=3).mean()

# Puissance même heure jour précédent
df["P_j-1"] = df["Puissance"].shift(24)
```

4. Encodage des variables temporelles

Évitez :

- heure = 1…24
- mois = 1…12

Car le réseau croit que :

- 23 h est "loin" de 0 h
- décembre est "loin" de janvier

Utilisez plutôt un encodage cyclique :

Pour l'heure :

$$heure_{sin} = \sin\left(2\pi \frac{heure}{24}\right)$$

et :

$$heure_{cos} = \cos\left(2\pi \frac{heure}{24}\right)$$

Même chose pour :

- jour semaine
- mois.

C'est extrêmement important en modélisation énergétique.

```
# -------- HEURE --------
df["heure_sin"] = np.sin(2 * np.pi * df["Heure"] / 24)
df["heure_cos"] = np.cos(2 * np.pi * df["Heure"] / 24)

# -------- JOUR --------
df["jour_sin"] = np.sin(2 * np.pi * df["Jour"] / 7)
df["jour_cos"] = np.cos(2 * np.pi * df["Jour"] / 7)

# -------- MOIS --------
df["mois_sin"] = np.sin(2 * np.pi * df["Mois"] / 12)
df["mois_cos"] = np.cos(2 * np.pi * df["Mois"] / 12)
```