Formation Automatants

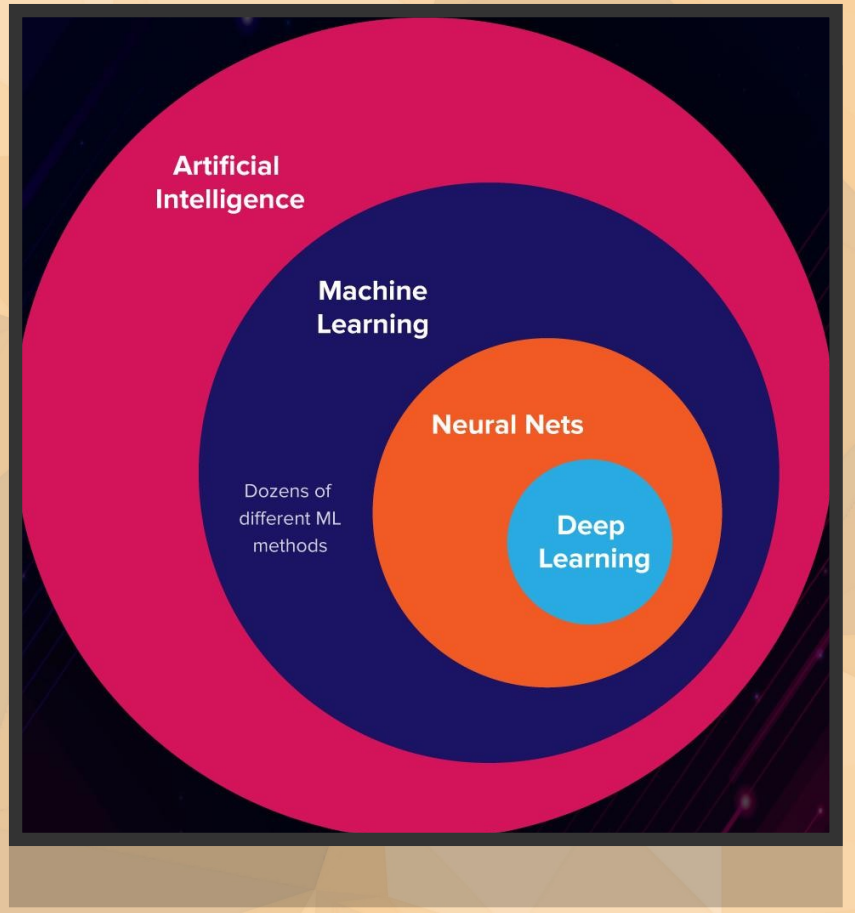
Introduction à l’IA

Formation d’introduction à l’IA

I – Machine Learning

II – Réseaux de neurones

III – Régression linéaire



Machine learning

I – Machine Learning

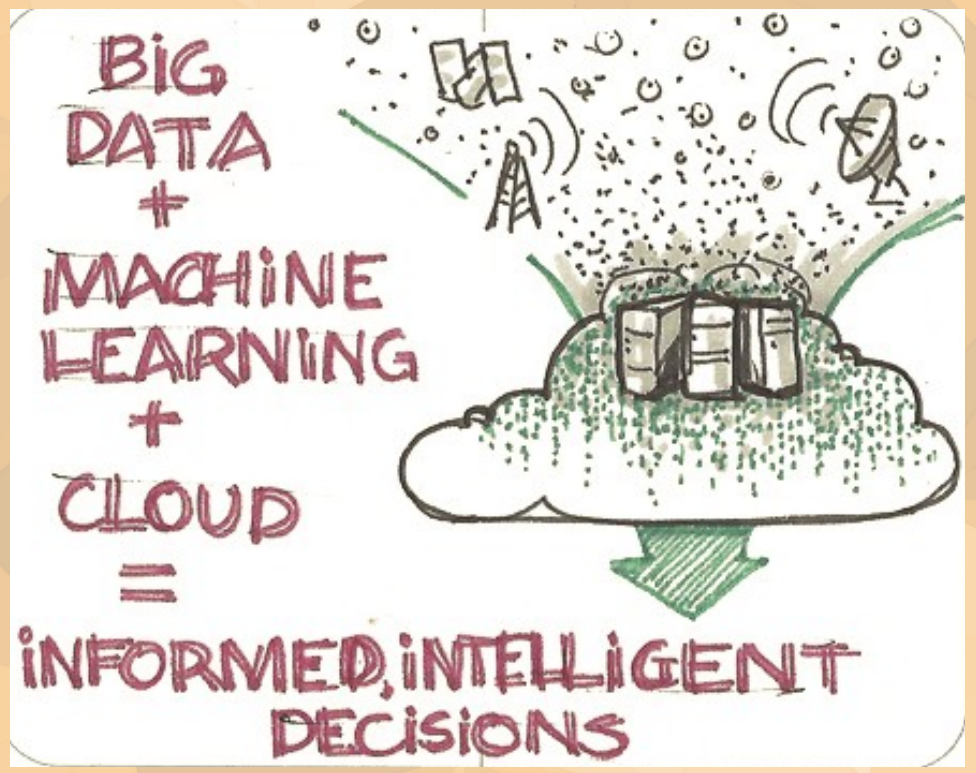
1) – Pourquoi faire apprendre aux machines ?
2)
3) – L’apprentissage supervisé
4)
5) – L’apprentissage non supervisé
6)
7) – L’apprentissage par renforcement
8)
9) – Slide de synthèse
10)

I – Machine Learning

1) – Pourquoi faire apprendre aux machines ?

Intelligence is the ability to avoid doing work, yet getting the work done.

Linus Torvalds (créateur de Linux)

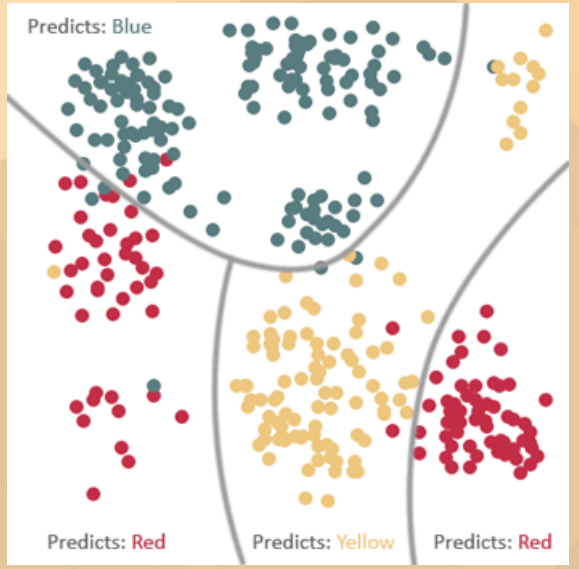


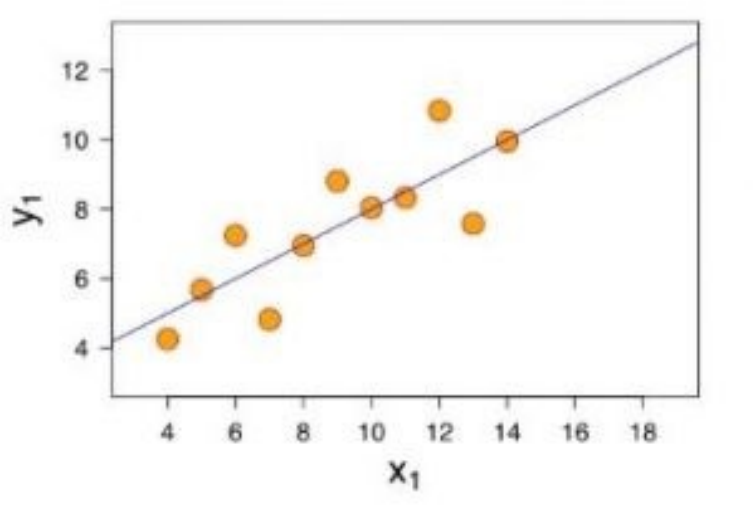
I – Machine Learning

2) – Apprentissage supervisé : l'algo s'entraîne sur des exemples

Algorithmes de classification : lorsque la variable d'arrivée est discrète, on est en fait en train de **classer les données** selon leur appartenance à une classe prédéfinie.

Dans l'exemple à droite, chaque donnée est un point du plan.
La variable d'arrivée qu'on associe à un couple prend une valeur parmi {Red, Blue, Yellow}.



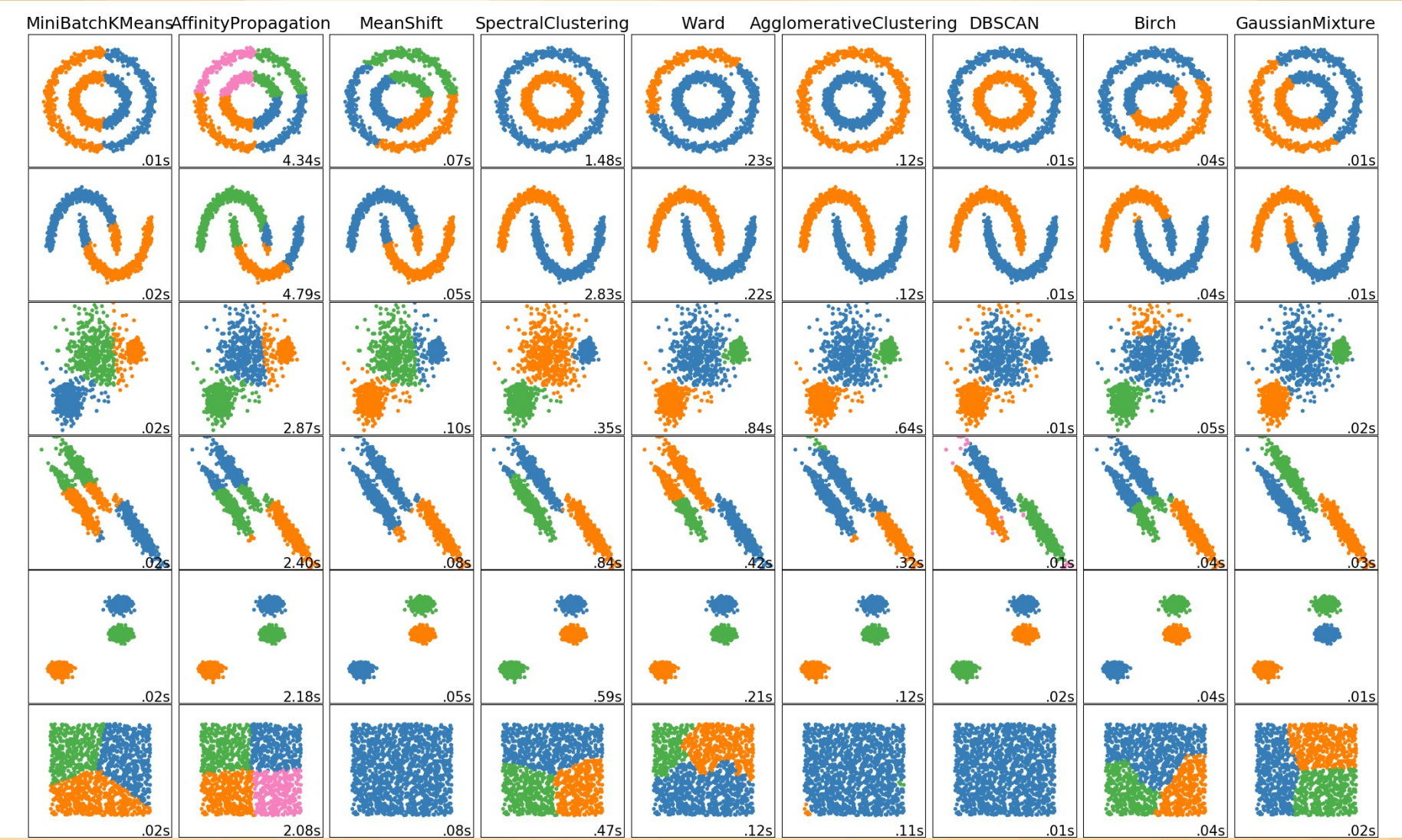


Algorithmes de régression : lorsque la variable d'arrivée est continue, on va plutôt *réduire* les observations pour **obtenir un modèle continu,** en fait une fonction.

Dans l'exemple de régression linéaire à gauche, le modèle obtenu est la fonction *affine dont le graphe est tracé en bleu.*

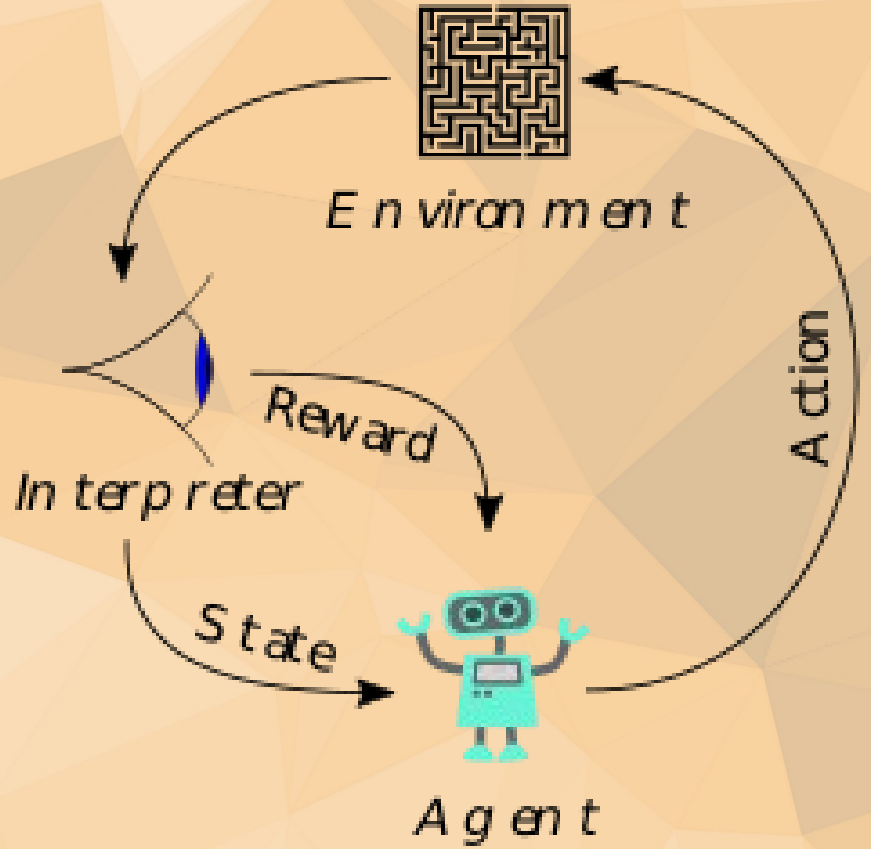
I – Machine Learning

3) – Apprentissage non supervisé : et si on laissait l'algo se débrouiller ?



I – Machine Learning

4) – Apprentissage par renforcement



Réseaux de neurones

II – Réseaux de neurones

1) – Neurone
2)
3) – Couche de neurones
4)
5) – Réseau de neurones

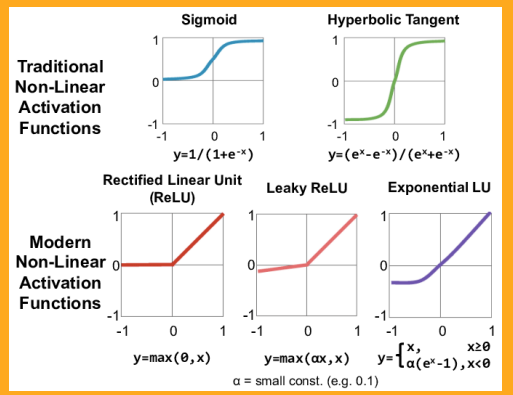
II – Réseaux de neurones

1) – Définition d'un neurone

$$X = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$$

Biais : b Poids : $W = \begin{pmatrix} w_1 \\ \dots \\ w_n \end{pmatrix}$

Fonction d'activation : f



$\rightarrow f(W.X + b)$

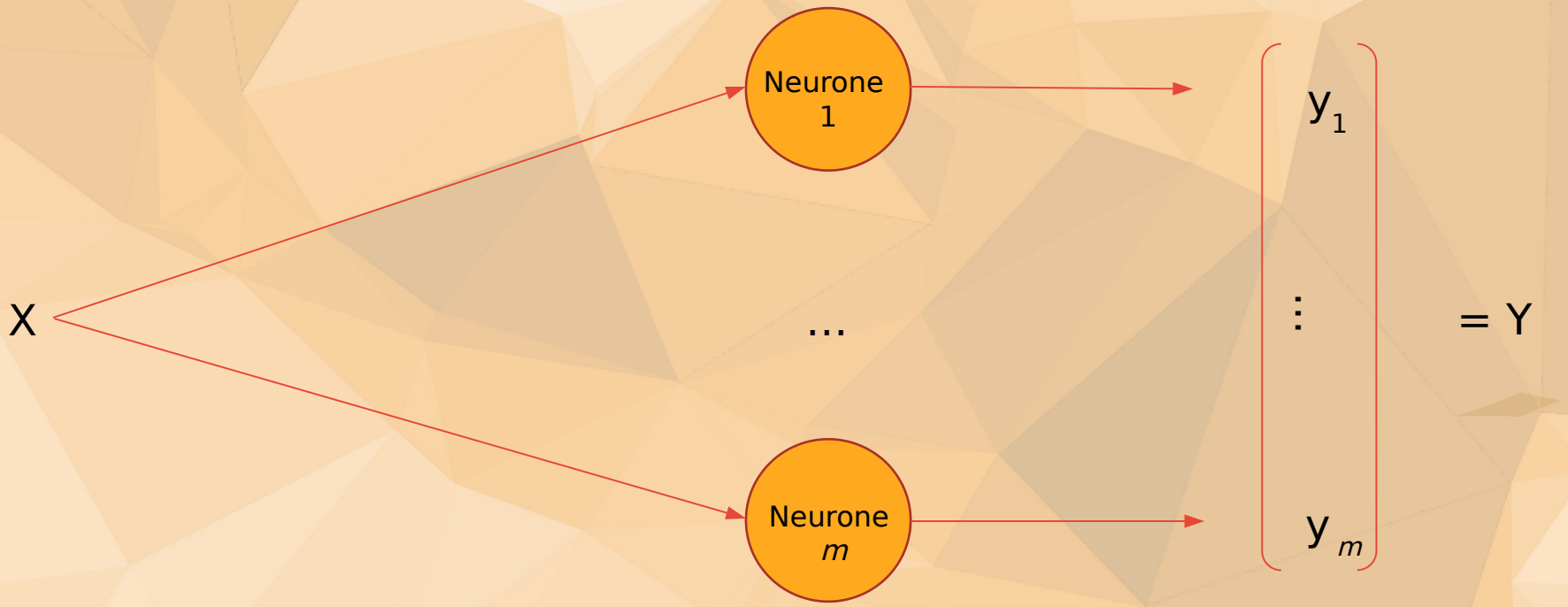
Donnée : un vecteur X

Traitement par un neurone

Sortie : un scalaire

II – Réseaux de neurones

2) – Architecture d'une couche de neurones



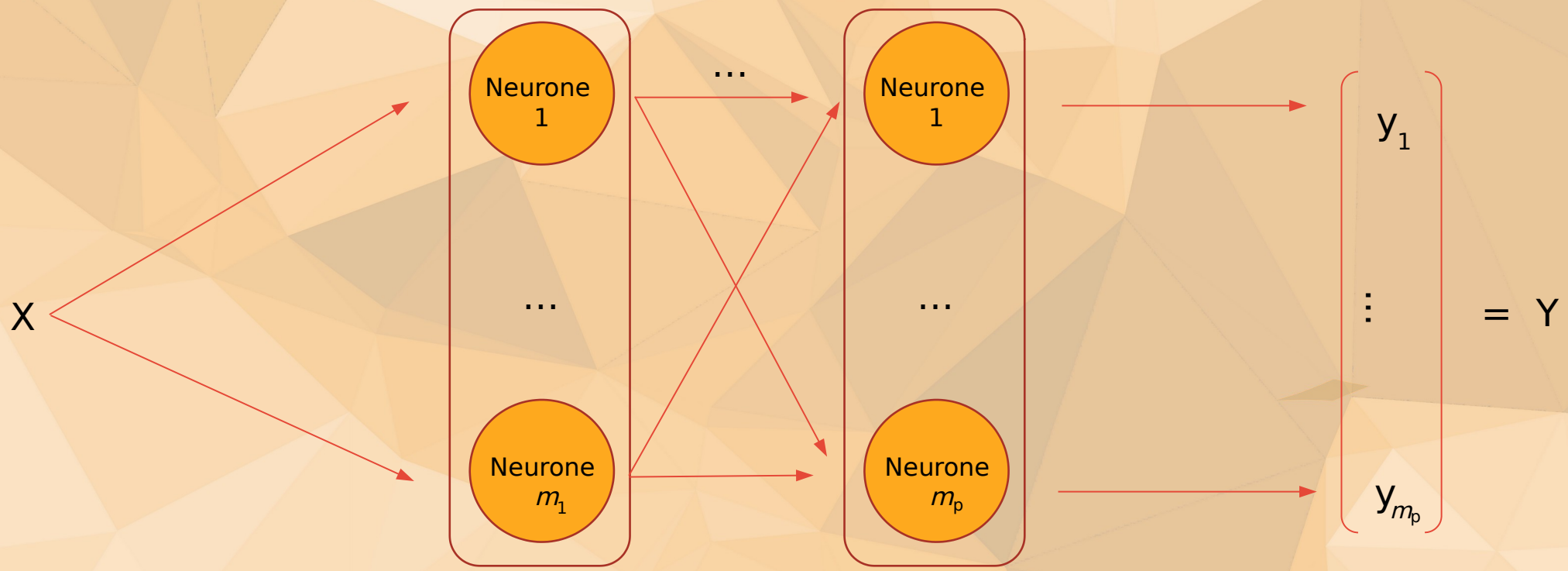
Donnée : un vecteur X

Traitement par *m* neurones

Sortie : *m* scalaires qui constituent un vecteur Y

II – Réseaux de neurones

3) – Architecture d'un réseau de neurones



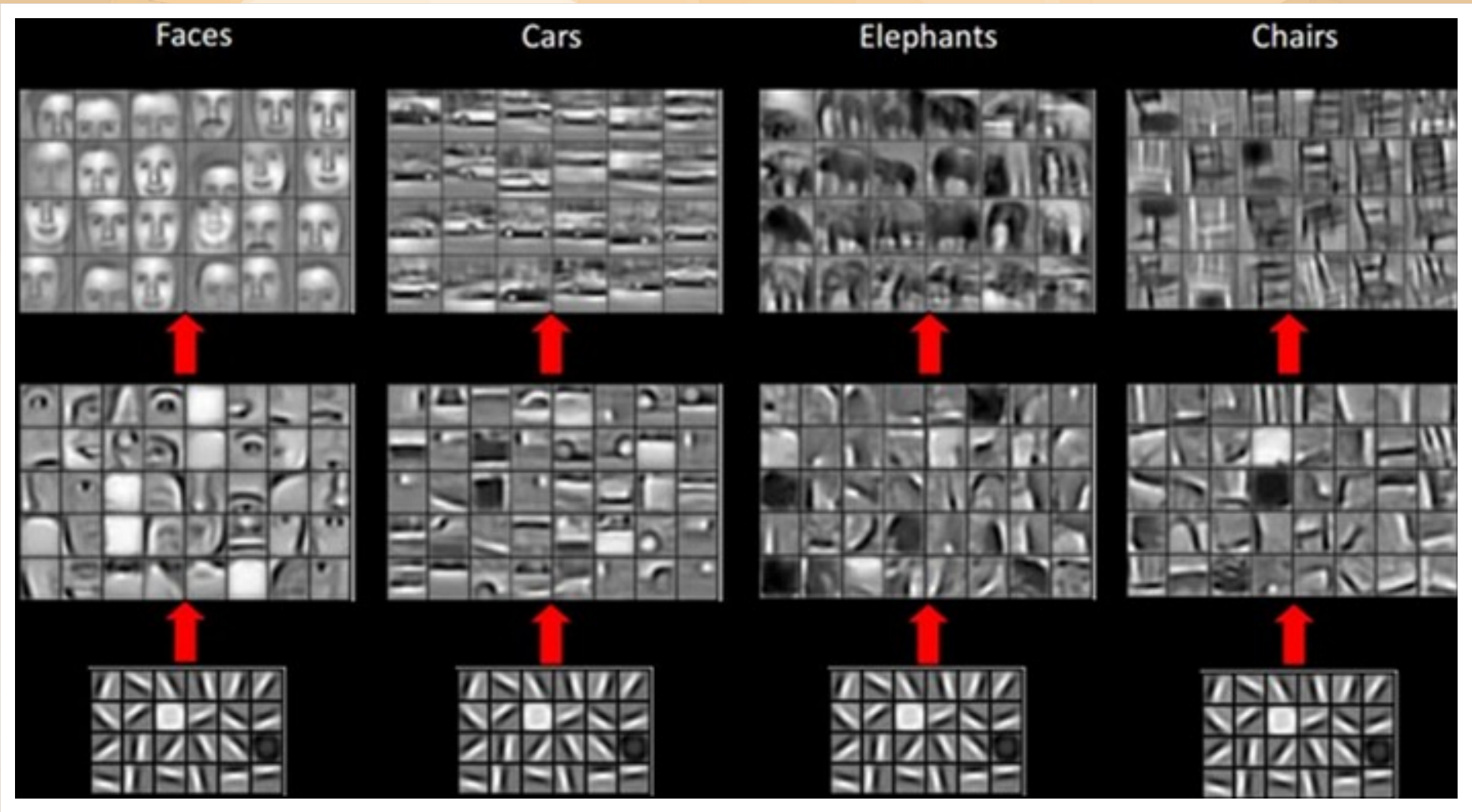
Donnée : un vecteur X

Traitement par *p* couches de neurones

Sortie : un vecteur Y

II – Réseaux de neurones

3) – Architecture d’un réseau de neurones : vers le Deep Learning



Régression linéaire & descente du gradient

Une méthode fondamentale abordée à travers un exemple détaillé

Régression linéaire et descente du gradient

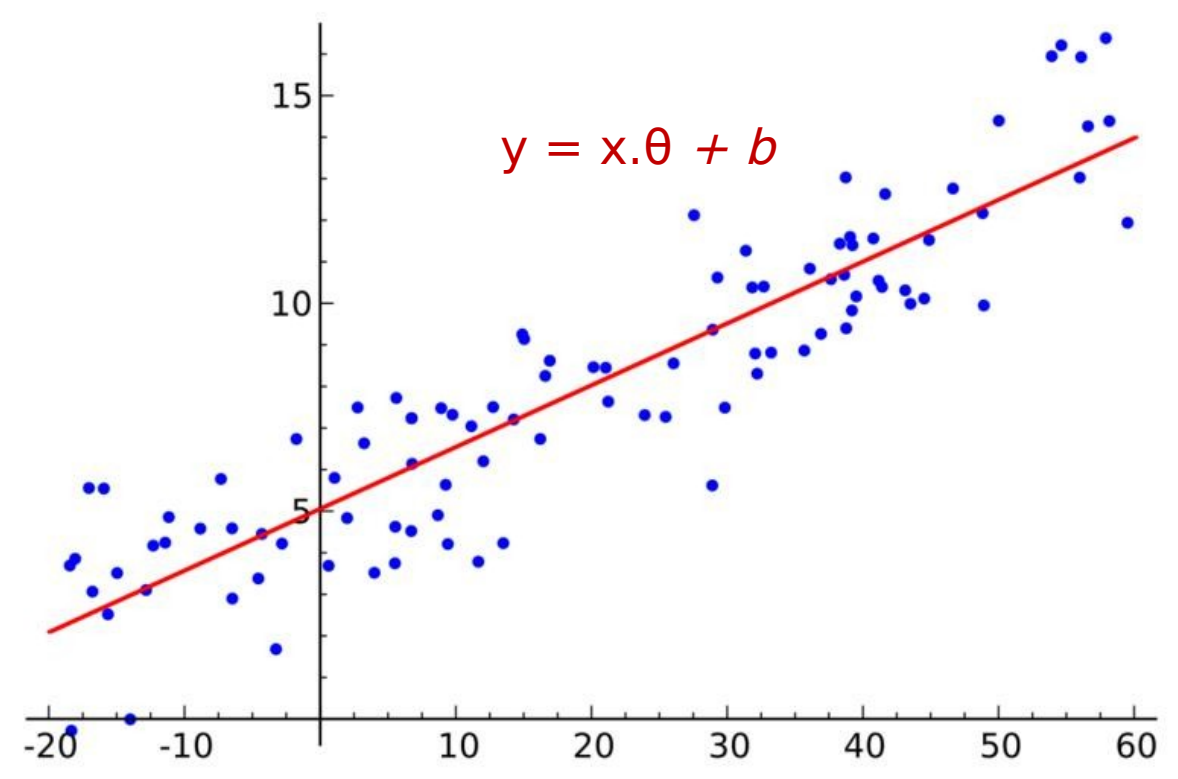
1) – Cadre du problème

2) – Résolution analytique

3) – Méthode du gradient

III – Régression linéaire et descente du gradient

1) – Cadre du problème



Données : un ensemble de points (x_i , y_i) du plan

Traitement : on choisit une fonction de coût J(θ, *b)* qui indique à quel point la fonction affine de paramètres (θ, *b)* modélise bien nos données.

Sortie : la pente θ et l'ordonnée à l'origine *b* d'une fonction affine qui minimise J.

III – Régression linéaire et descente du gradient

2) – Résolution analytique de la régression linéaire multivariée

Données : l'échantillon X est constitué de m coordonnées x_i, chacune correspondant à un point de l'ensemble de données :

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}.$$

On dispose aussi du vecteur $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$ qui contient les m valeurs associées aux x_i.

Objectif : trouver un vecteur θ tel que la forme linéaire $x \longmapsto x.\theta$ soit un bon estimateur de $x_i \longmapsto y_i$.

Résolution : on peut montrer que la pente θ optimale est définie par l'*équation normale* :

$$X^T X\theta = X^T Y$$

III – Régression linéaire et descente du gradient

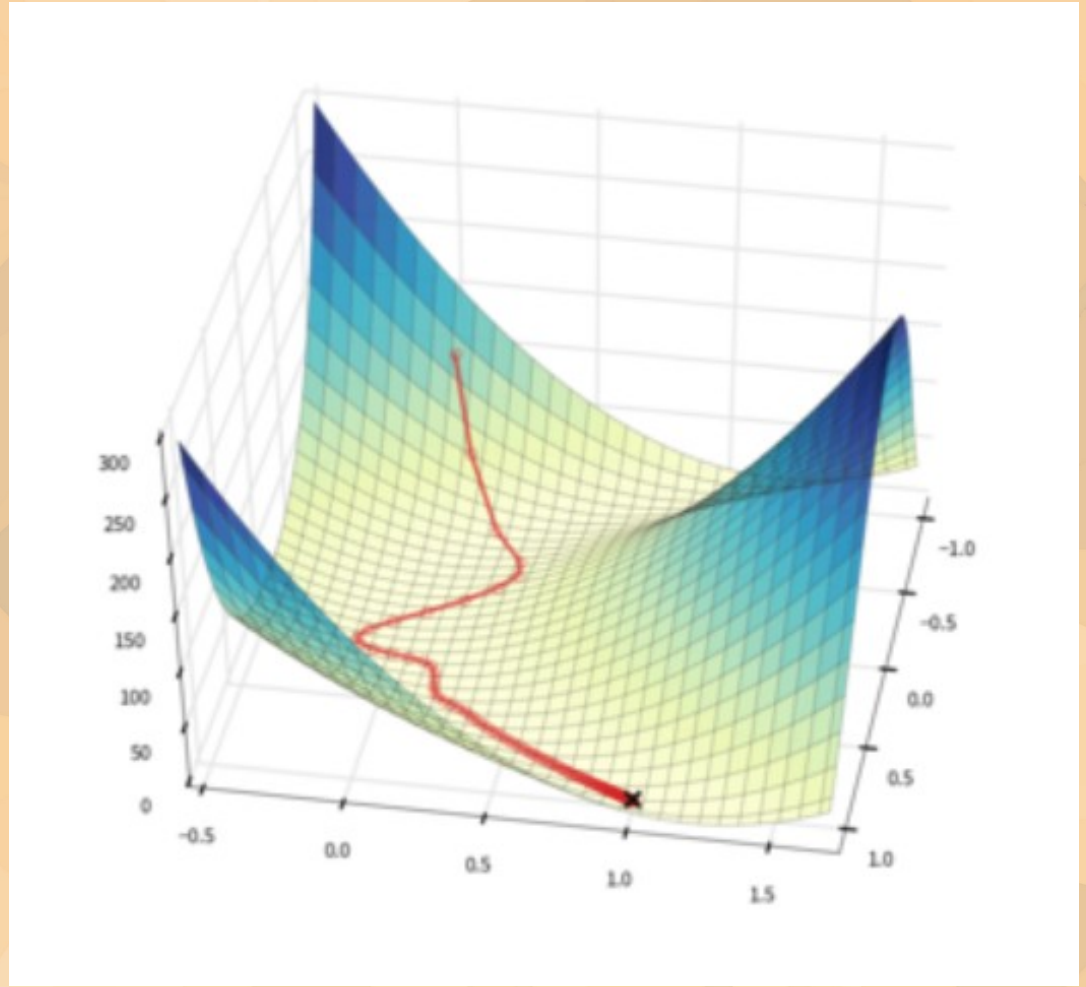
3) – Méthode du gradient appliquée à la régression linéaire

Objectif : trouver des valeurs de θ_0 et θ_1 successives
qui diminuent à chaque fois le coup.

Traitement : On utilise la dérivée de J en θ_j pour savoir s'il faut augmenter ou diminuer θ et avec quelle intensité.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Remarque : dans cette formule, le paramètre α est le taux d'apprentissage (*learning rate*), qui doit être choisi judicieusement pour moduler la vitesse de convergence de l'algorithme.



Formation d’introduction à l’IA

- Face à de nombreux problèmes de la vie réelle, on préfère faire apprendre à un algorithme une méthode de résolution, plutôt que de coder la solution. C’est le **machine learning**.

- En général, on recherche une fonction vectorielle comme solution. Une telle fonction peut s’approcher par un **réseau de neurones**, qui est la composée de nombreuses fonctions élémentaires (affines et d’activation) caractérisées par des poids réels.

- Pour trouver les poids qui résoudront notre problème, on va utiliser la **descente du gradient**. Cet algorithme consiste à modifier les poids « dans le bon sens », c’est-à-dire de telle manière que la modification diminue l’erreur commise par le réseau de neurone.

- Pour voir comment faire, rendez-vous à la formation sur la **backpropagation** !

Merci pour votre attention