

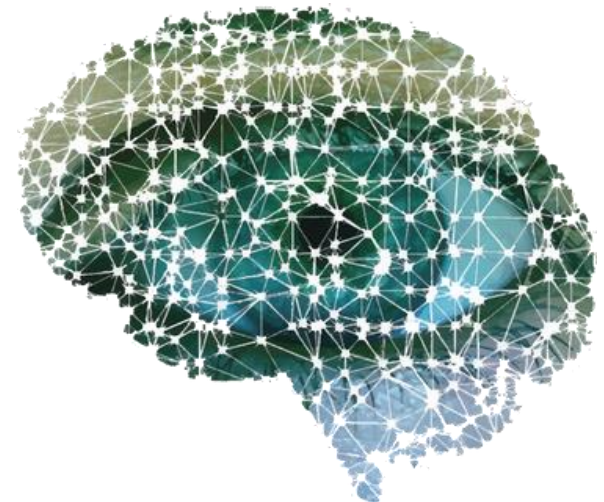
APPRENTISSAGE PROFOND ET IMAGES

LES BASES

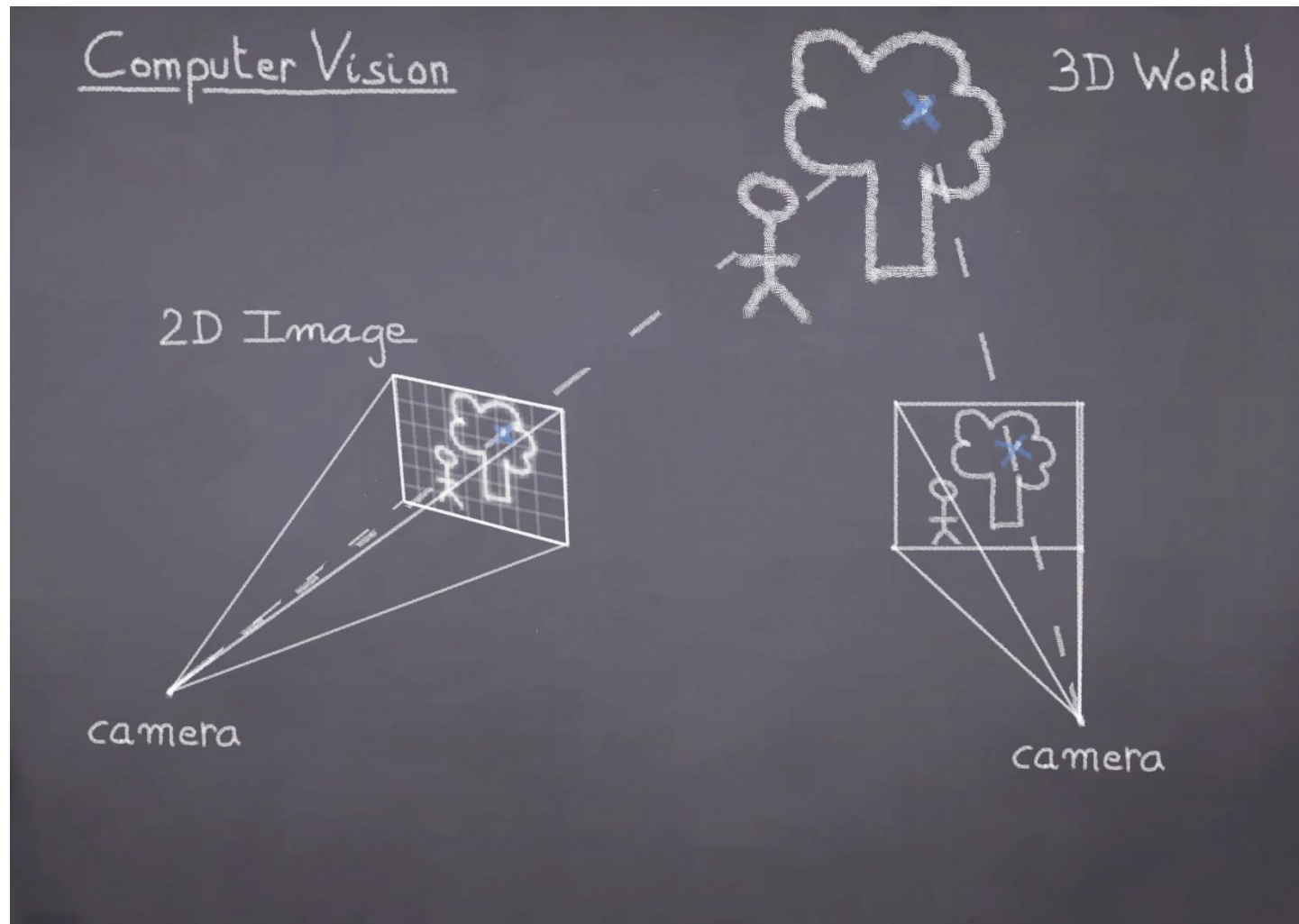
Alexandre Meyer¹

¹Equipe SAARA, laboratoire LIRIS

Master ID3D et IA



Vision par ordinateur



Vision par ordinateur et vision humaine



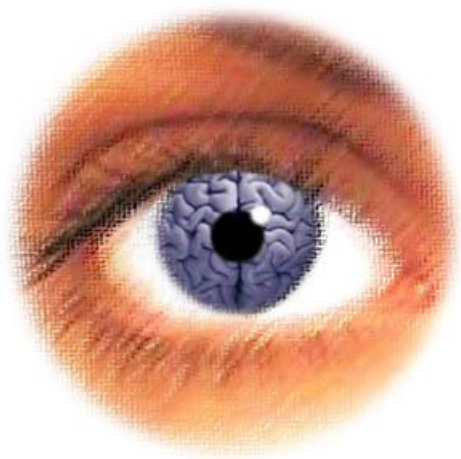
Nature

≠

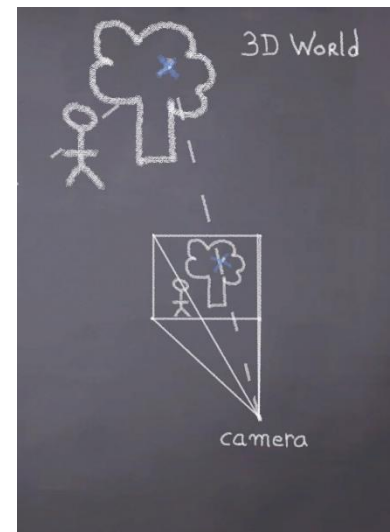
inspire l'



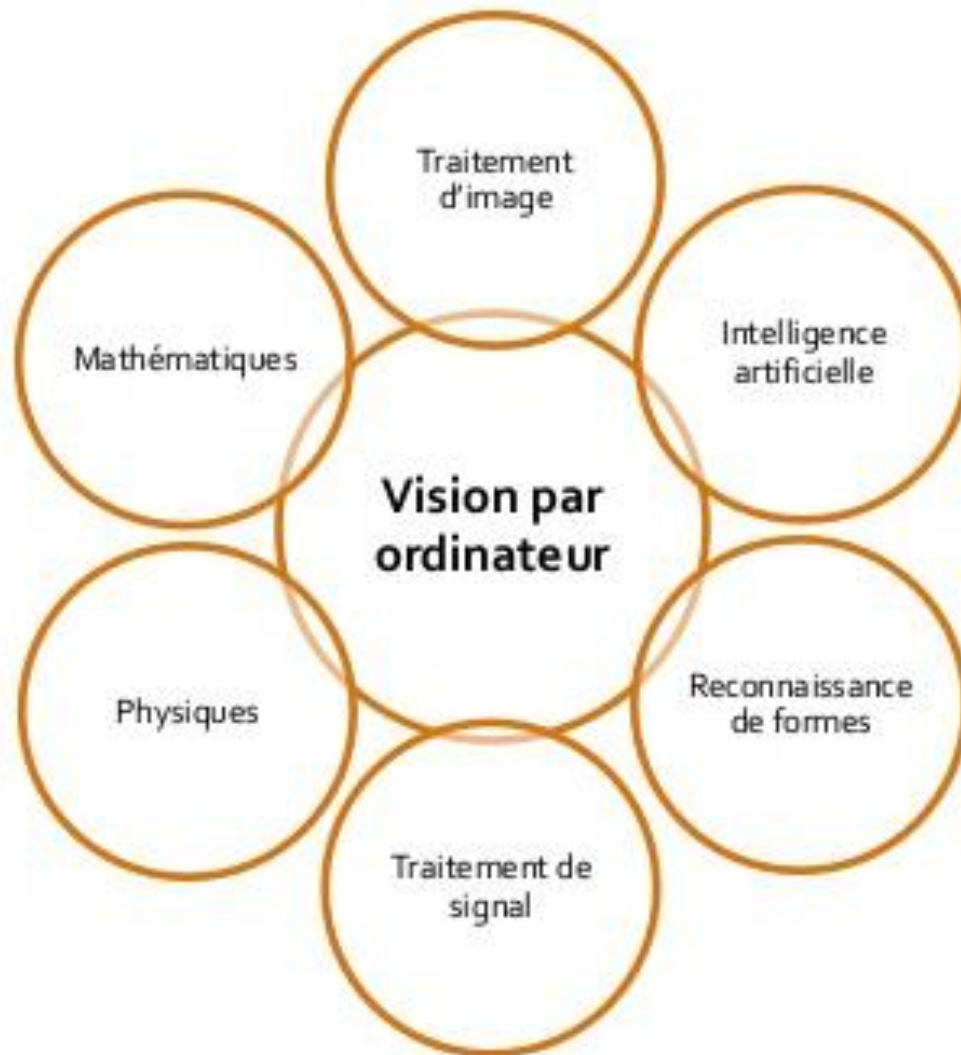
Artificiel



≠



Vision par ordinateur



Deep Learning et computer vision

Vision par ordinateur a toujours travaillé avec l'apprentissage machine

L'apprentissage profonde a permis de faire sortir la vision des labos

- Image classification
- Segmentation
- Object detection and tracking
- Image generation
- Laying reconstruction
- Action recognition
- ...

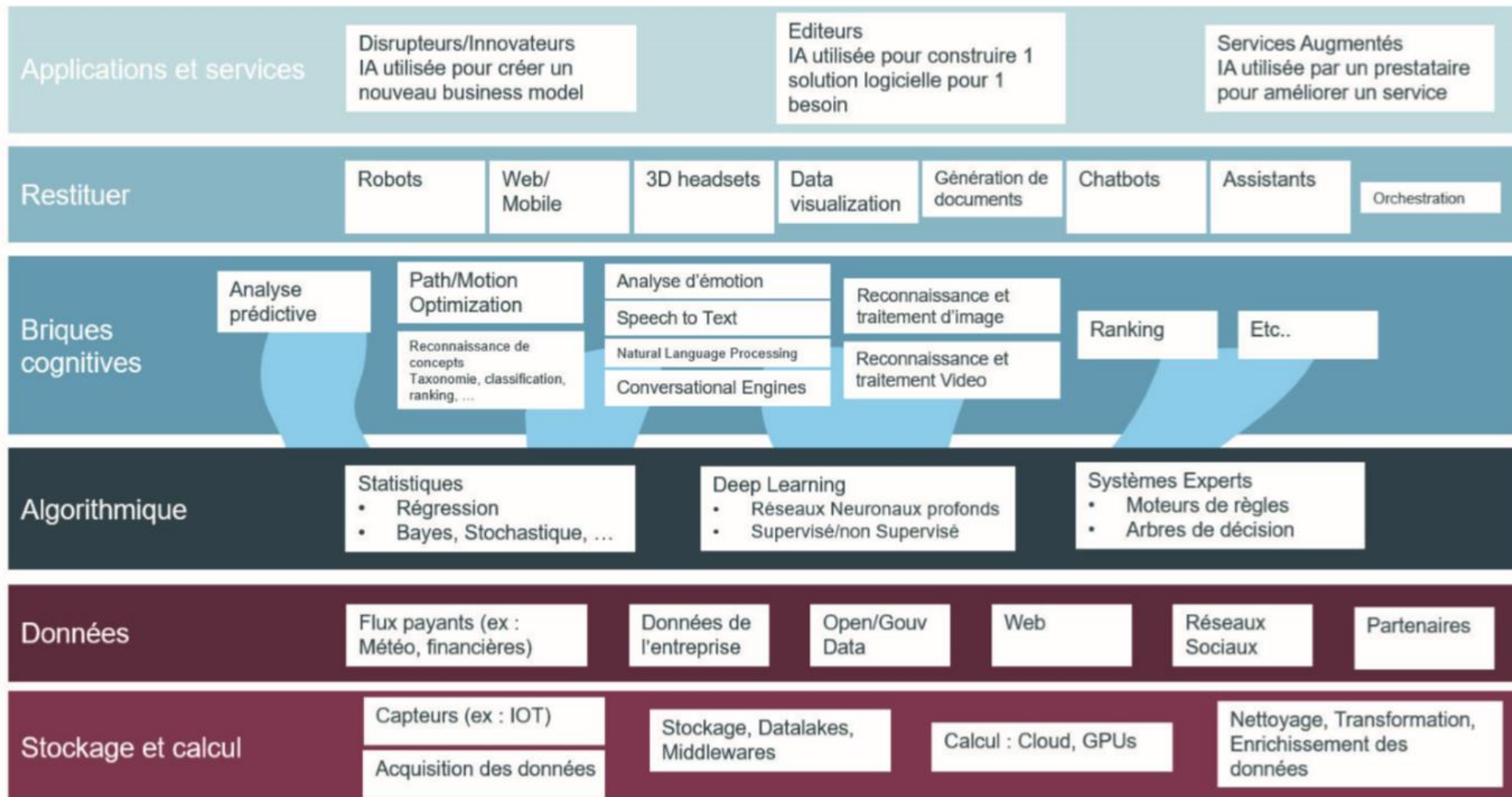


RESEAUX DE NEURONES (PROFONDS)

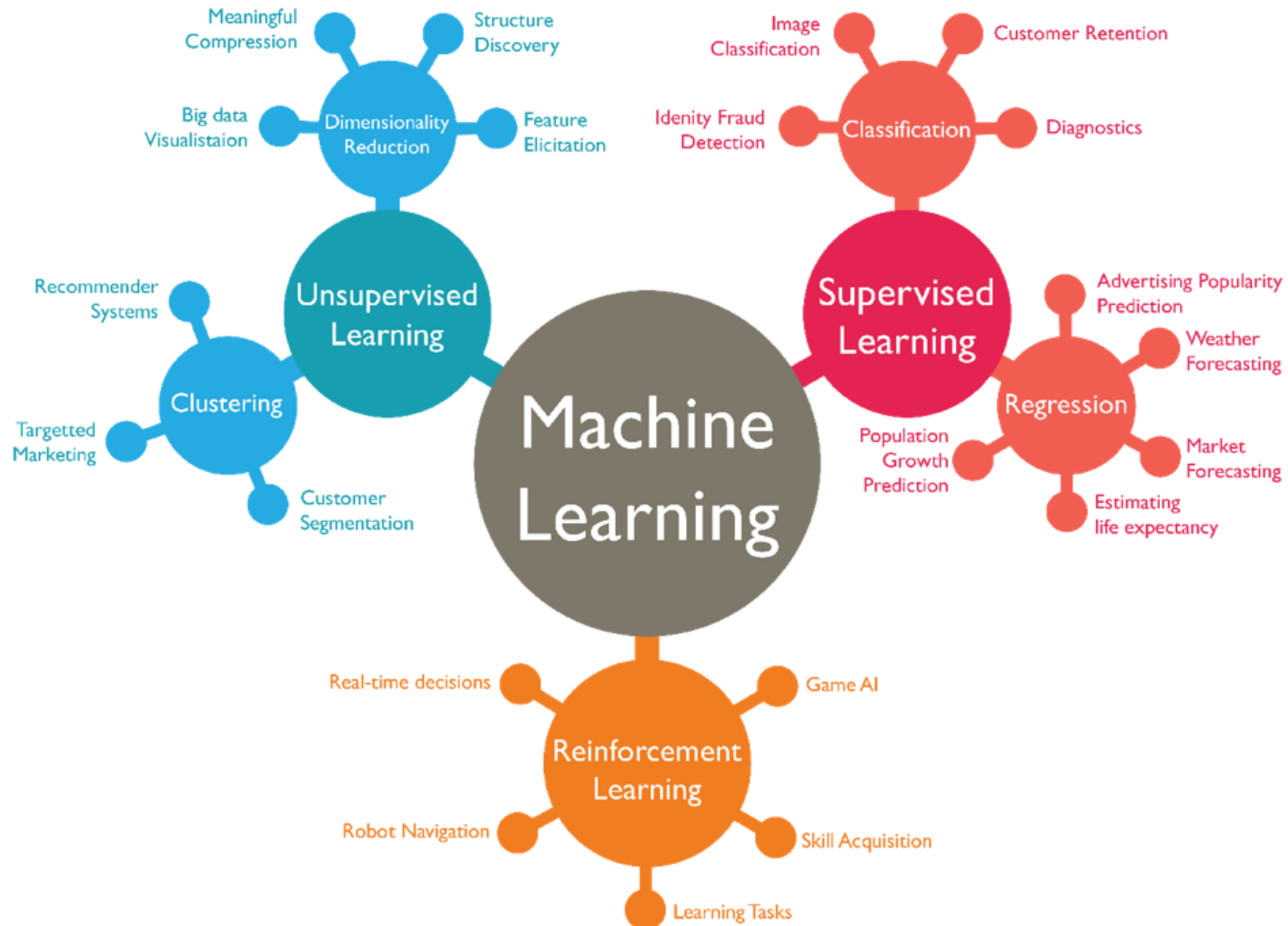
AI

- ...
- Machine Learning (apprentissage machine)
 - Random Forest
 - SVM
 - Bayésien
 - ...
 - Neural Network
 - Deep Learning (dans ce cours voir ça comme un « outils »)
 - Apprentissage par renforcement

IA / ML / DL



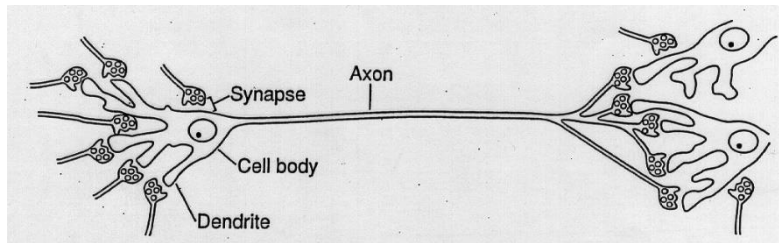
ML : Supervisé / Non supervisé



Réseau de neurones artificiel (RNA)

un modèle de calcul inspiré du cerveau humain

- Cerveau humain :
 - 10 milliards de neurones
 - 60 milliards de connexions (synapses)
 - Un synapse peut être inhibant ou excitant.
- RNA :
 - Un nombre fini de processeurs élémentaires (neurones).
 - Liens pondérés passant un signal d'un neurone vers d'autres.
 - Plusieurs signaux d'entrée par neurone

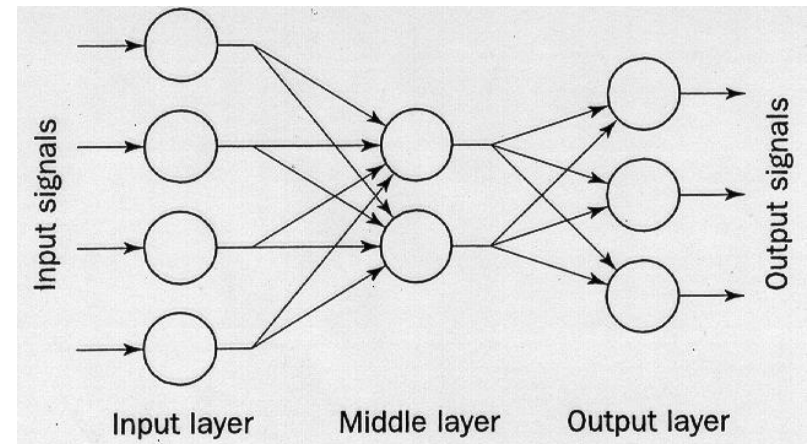


Cerveau

cellule (soma)
dendrites
synapses
axon

RNA

neurone
entrées
poids
sortie

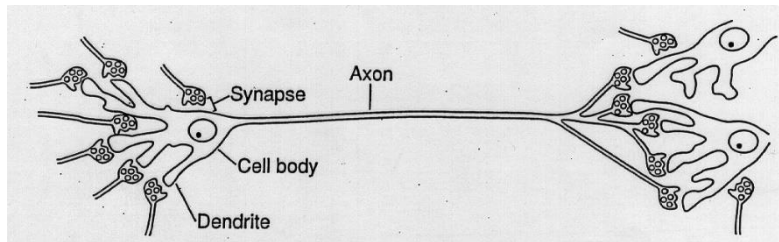


[McCulloch-Pitts, 1943]

Réseau de neurones artificiel (RNA)

un modèle de calcul inspiré du cerveau humain

- Cerveau humain :
 - 10 milliards de neurones
 - 60 milliards de connexions (synapses)
 - Un synapse peut être inhibant ou excitant.



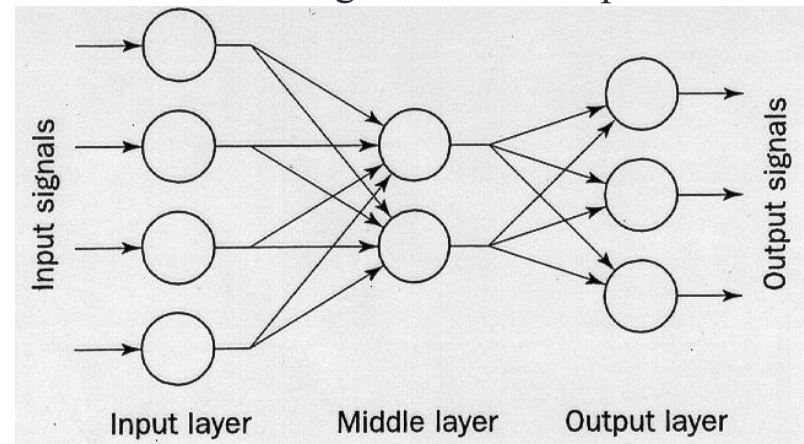
Cerveau

cellule (soma)
dendrites
synapses
axon

RNA

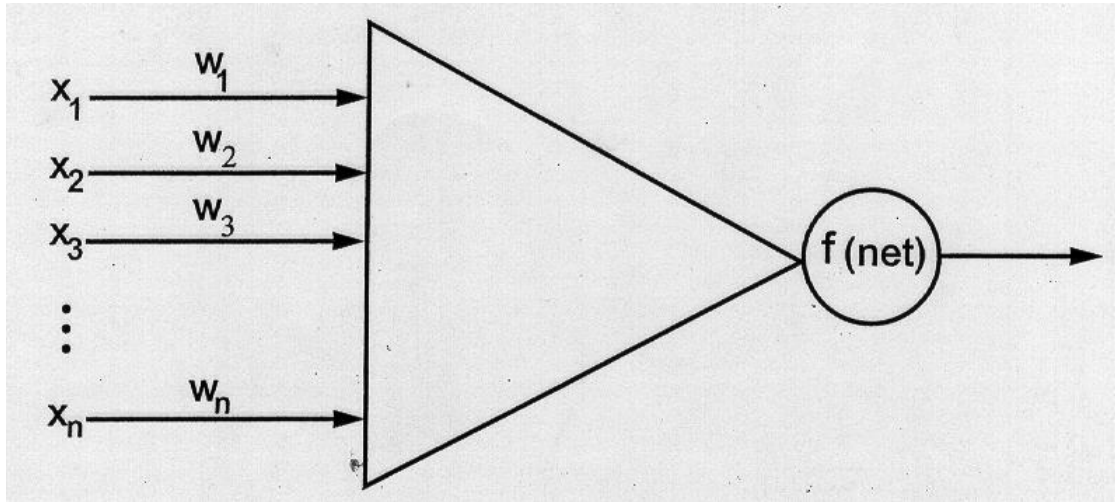
neurone
entrées
poids
sortie

- RNA :
 - Un nombre fini de processeurs élémentaires (neurones).
 - Liens pondérés passant un signal d'un neurone vers d'autres.
 - Plusieurs signaux d'entrée par neurone



Une boîte noire qui transforme des nombres en d'autres nombres en « regardant » une base d'apprentissage. Après apprentissage on veut $f(\text{input}) = \text{output}$

Introduction générale aux réseaux de neurones



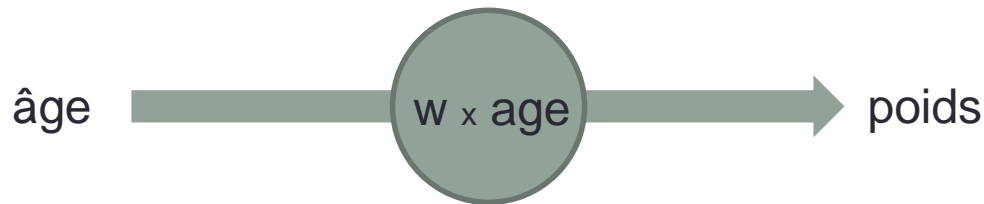
[McCulloch-Pitts, 1943]

Un type de neurone simple : $nD \rightarrow 1D$ avec $f=\text{sign}$

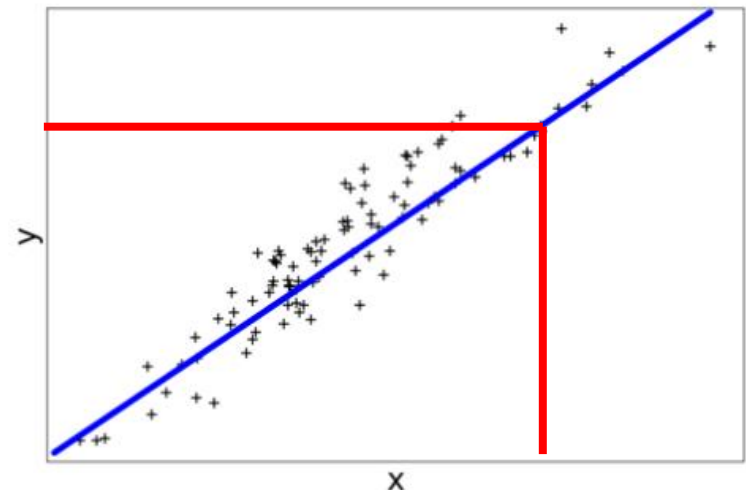
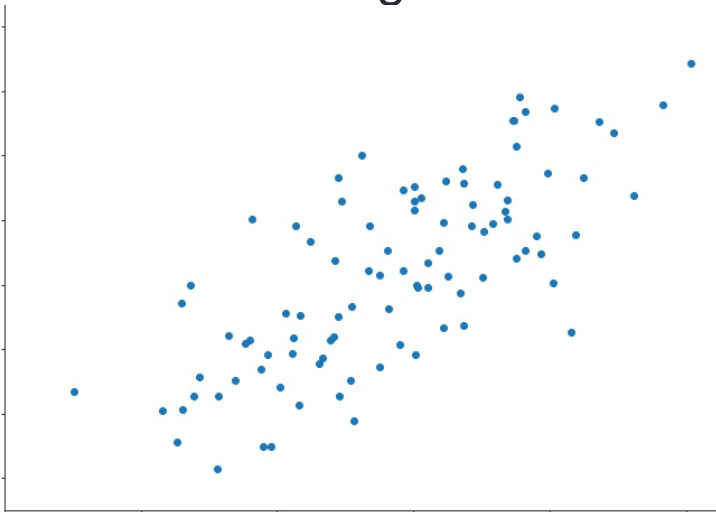
- $\text{net} = \sum w_i x_i$ x = données d'entrée, w =les poids
- $f(\text{net}) = +1$ si $\text{net} \geq 0$, -1 sinon ($\text{net} < 0$) f = Fonction d'activation du neurone
- C.à-d. ici : 1 neurone = $\text{sign}(\sum w_i x_i)$

Exemple de régression ~ réseau de neurones

- Soit un problème simple, par exemple prédire le poids d'un enfant en fonction de son âge



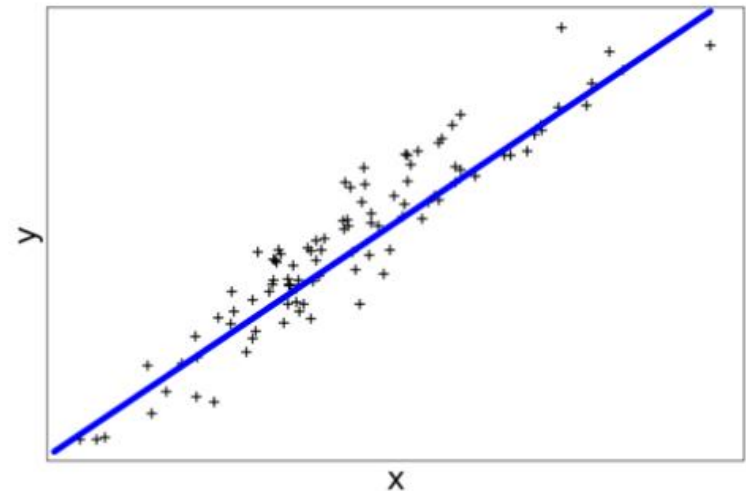
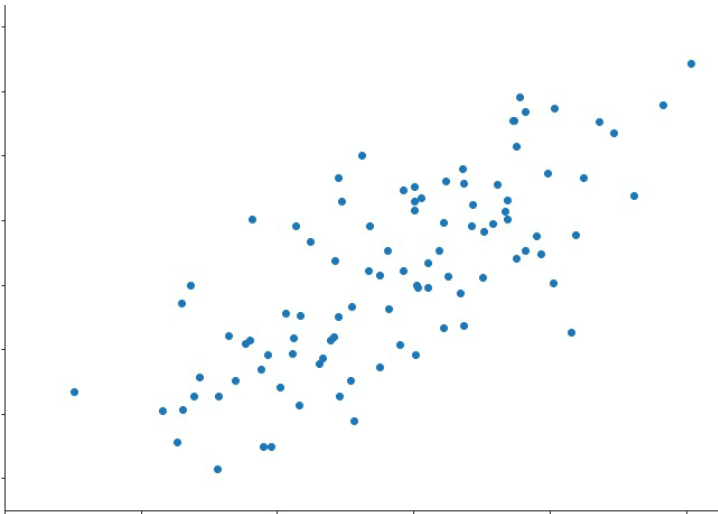
- Exploiter le neurone pour prédire
 - Pour un âge → estimation du poids



Exemple de régression ~ réseau de neurones

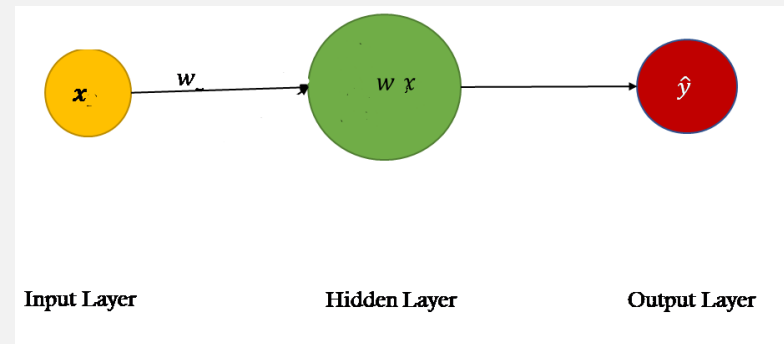
Apprentissage consiste à trouver les poids w_{ij} par optimisation

- Base de connaissance, un jeu d'apprentissage
- une série de couple (entrée, sortie) donc de
- **Intuition : initialiser w_{ij} au hasard, puis descente de gradient (~)**



Exemple de régression ~ réseau de neurones

Input	Desired output
0	0
1	2
2	4
3	6
4	8



- 1 neurone ultra simple
 - sans activation
 - ni bias
 - input/output à 1D

$$w \cdot \text{input} = \text{output}$$

- Optimisation = entraînement = trouver w à partir de la base d'apprentissage (le tableau en haut à gauche)

Exemple de régression ~ réseau de neurones

- Initialisation au hasard de w à 3

$w \cdot \text{input} = \text{output}$

Input	Actual output of model 1 ($y = 3 \cdot x$)
0	0
1	3
2	6
3	9
4	12

Exemple de régression ~ réseau de neurones

- Initialisation au hasard de w à 3
 $w \cdot \text{input} = \text{output}$
- Fonction d'erreur : loss

Input	Actual output	Desired output
0	0	0
1	3	2
2	6	4
3	9	6
4	12	8

Exemple de régression ~ réseau de neurones

- Initialisation au hasard de w à 3
 $w \cdot \text{input} = \text{output}$
- Fonction d'erreur : loss

Input	actual	Desired	Absolute Error	Square Error
0	0	0	0	0
1	3	2	1	1
2	6	4	2	4
3	9	6	3	9
4	12	8	4	16
Total:	-	-	10	30

Exemple de régression ~ réseau de neurones

- Initialisation au hasard de w à 3
 $w \cdot \text{input} = \text{output}$
- Différentiation : $d\text{loss}/dw = (\text{loss}(w) - \text{loss}(w+\text{delta}))/\text{delta}$
 - Faire varier un peu $w \Rightarrow w=3.0001$
 - Calcul du gradient (dérivée partielle en cas de $\text{dim} > 1$)

Input	Output	W=3	sq.e (3)	W=3.0001	sq.e
0	0	0	0	0	0
1	2	3	1	3.0001	1.0002
2	4	6	4	6.0002	4.0008
3	6	9	9	9.0003	9.0018
4	8	12	16	12.0004	16.0032
Total:	-	-	30	-	30.006

→ Descente de gradient

$d\text{loss}/dw = (30.006-30)/0.0001 > 0 \rightarrow$ Augmenter w augmente l'erreur

→ Avancer dans le sens opposé au gradient $w_{\text{new}} = w - \text{lambda.gradient}$

Exemple de régression ~ réseau de neurones

- Initialisation au hasard de w à 3
 $w \cdot \text{input} = \text{output}$
- Différentiation : $d\text{loss}/dw = (\text{loss}(w) - \text{loss}(w+\text{delta}))/\text{delta}$
 - Faire varier un peu $w \Rightarrow w=3.0001$
 - Calcul du gradient (dérivée partielle en cas de $\text{dim} > 1$)

On verra plus tard
Learning rate

Input	Output	W=3	sq.e (3)	W=3.0001	sq.e
0	0	0	0	0	0
1	2	3	1	3.0001	1.0002
2	4	6	4	6.0002	4.0008
3	6	9	9	9.0003	9.0018
4	8	12	16	12.0004	16.0032
Total:	-	-	30	-	30.006

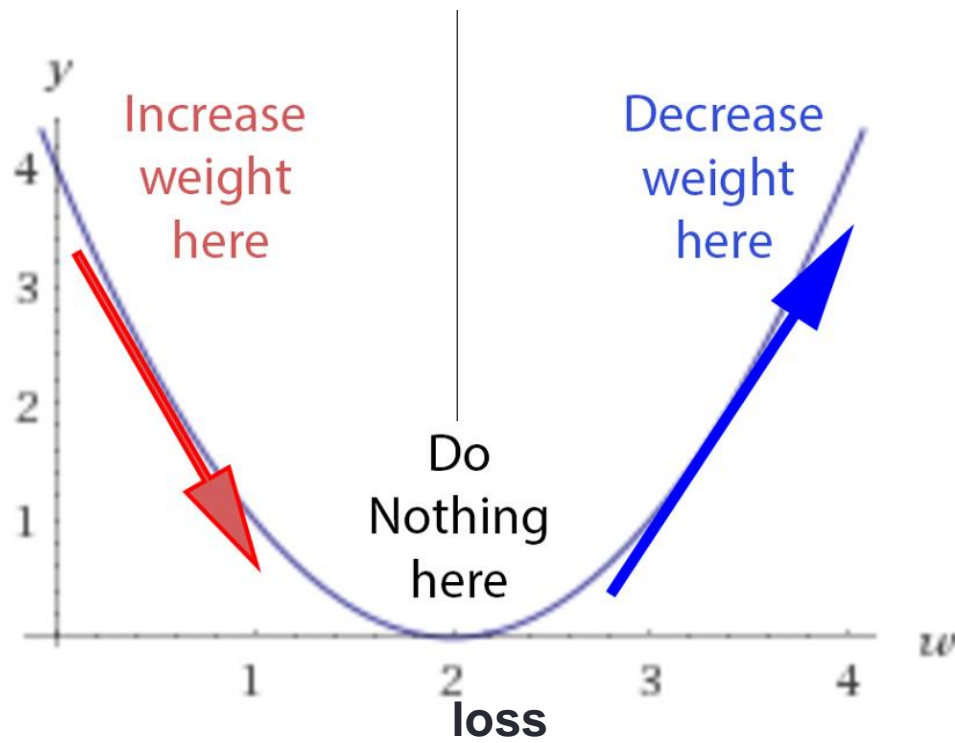
→ Descente de gradient

$d\text{loss}/dw = (30.006-30)/0.0001 > 0 \rightarrow$ Augmenter w augmente l'erreur

→ Avancer dans le sens opposé au gradient $w_{\text{new}} = w - \text{lambda} \cdot \text{gradient}$

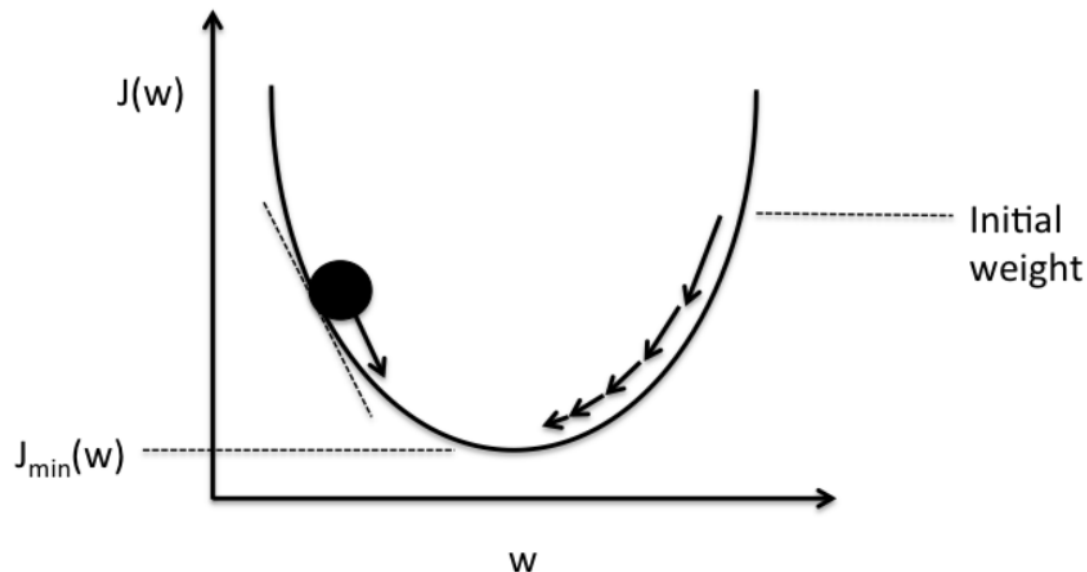
Exemple de régression ~ réseau de neurones

- Initialisation au hasard de w à 3
 w . input = output
- Différentiation : $df/dw = (f(w) - f(w+\text{delta}))/\text{delta}$
 - Faire varier un peu $w \Rightarrow w=3.0001$



Exemple de régression ~ réseau de neurones

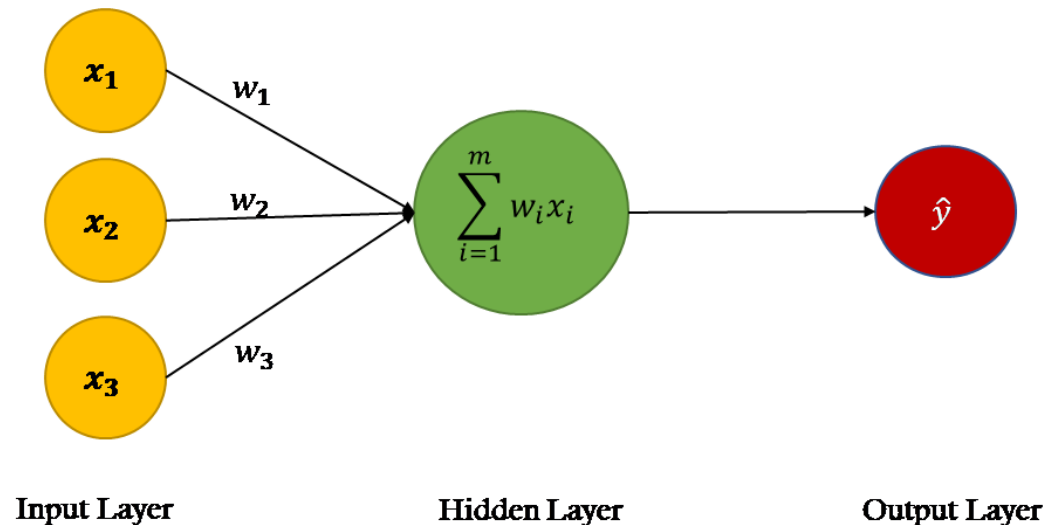
- Initialisation au hasard de w à 3
 $w \cdot \text{input} = \text{output}$
- Différentiation : $df/dw = (f(w) - f(w+\text{delta}))/\text{delta}$
 - Faire varier un peu $w \Rightarrow w=3.0001$



Schematic of gradient descent.

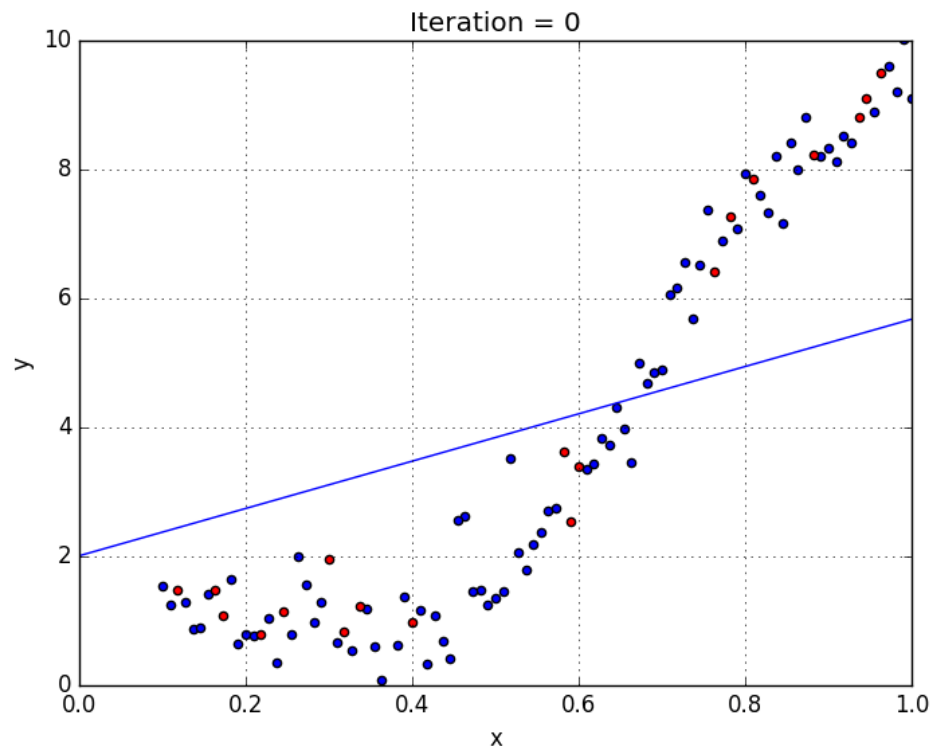
Vers des réseaux de neurones profonds

- Souvent un neurone à plusieurs entrées et plusieurs sorties
 - En générale problème non linéaire (mais dérivable)
 - ➔ Par exemple fonction d'activation non linéaire
 - Dimension élevée
 - Nombreux paramètres w

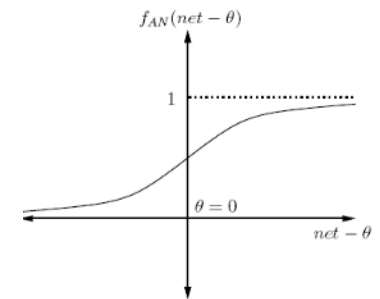
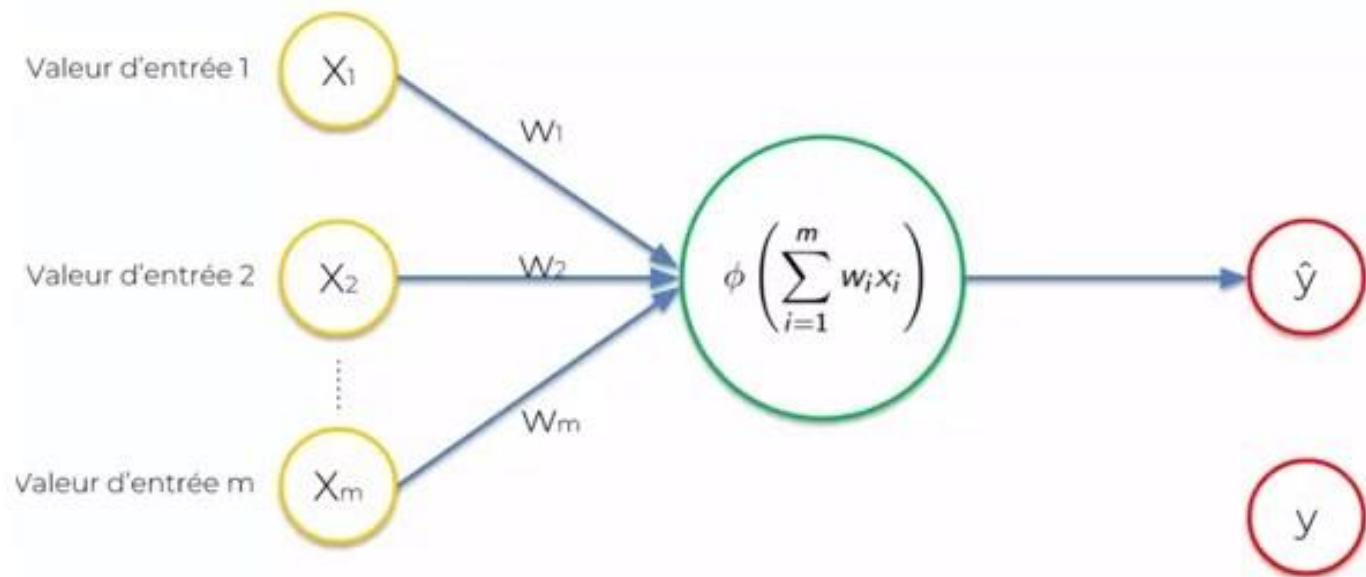


Vers des réseaux de neurones profonds

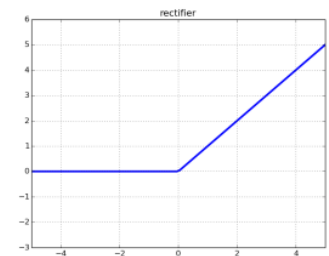
- Seulement souvent le problème est moins simple
 - **Moins linéaire, plus de dimensions**
 - ➔ **Ajout d'une fonction d'activation (non linéaire)**
 - ➔ **Et de plusieurs neurones**



Vers des réseaux de neurones profonds



(d) Sigmoid function

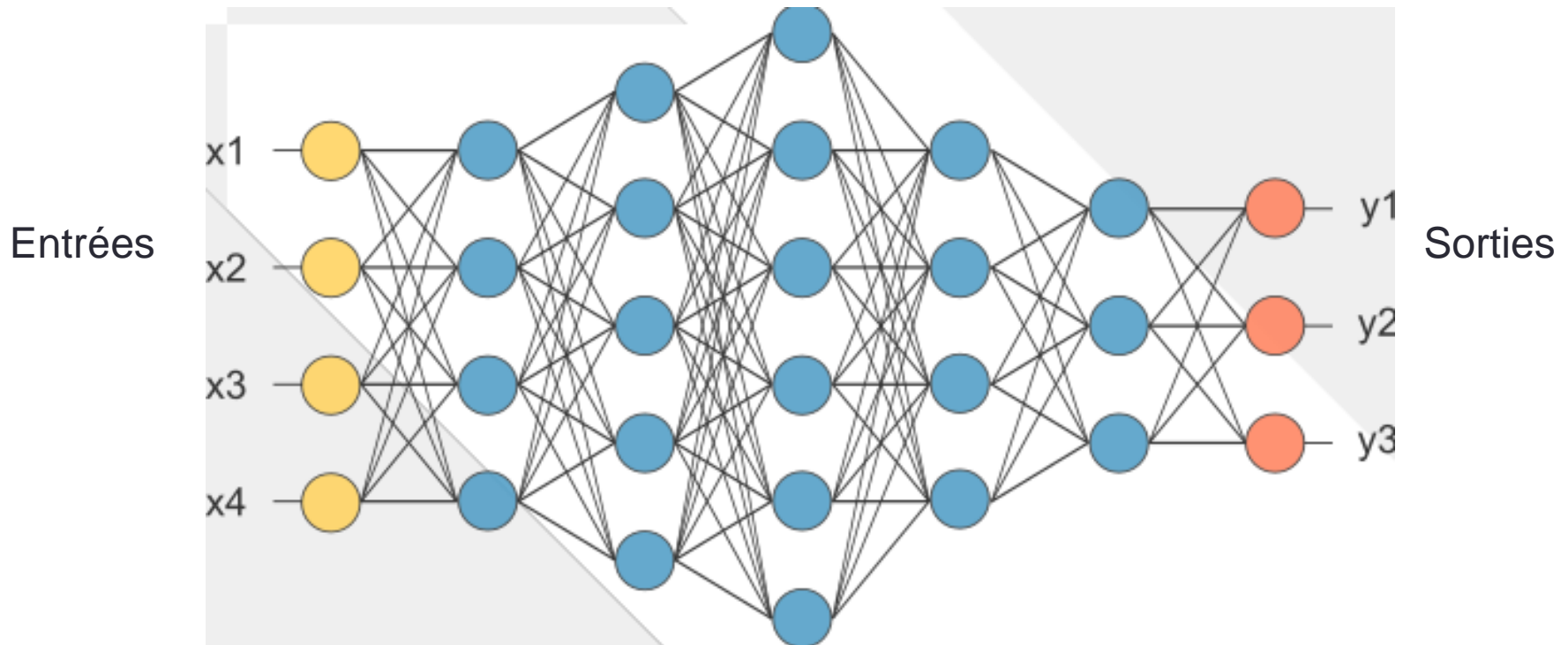


Apprentissage consiste à trouver les poids w_{ij} par optimisation

La fonction d'activation (non linéaire) permet d'approximer un problème/une fonction générale non linéaire.

Réseaux de neurones profonds

- Chaque cercle est un neurone $f(A.X+B)$
 - En bleu les couches cachées



Réseaux de neurones profonds

Pour chaque neurone

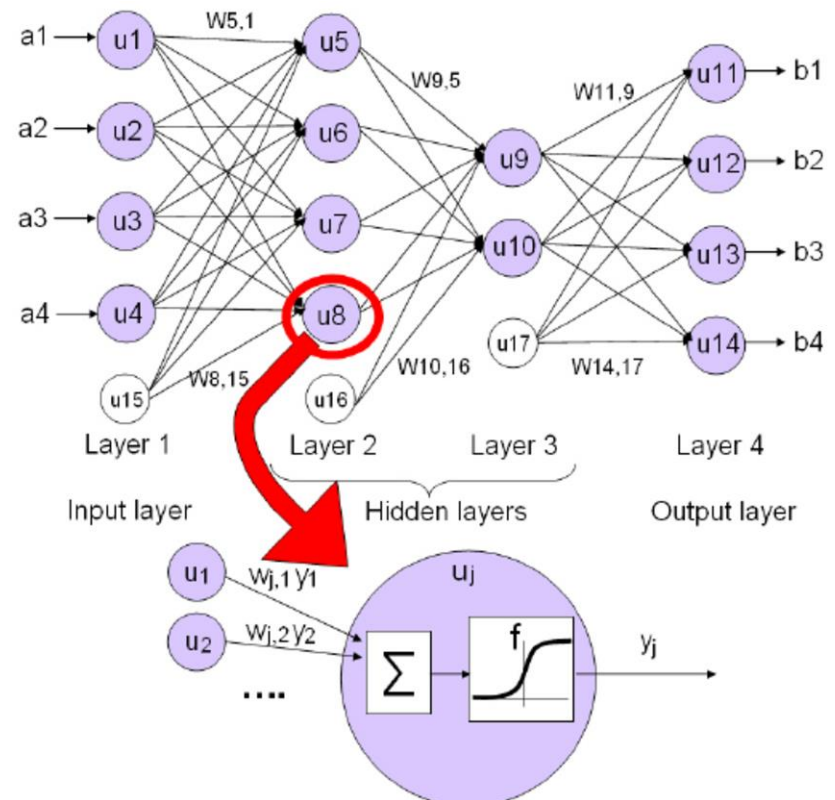
- n Dimensions en entrée → m Dimensions en sortie
- Les poids w_{ij} forment une matrice W

- sortie = $W \cdot \text{entrée} + B$

- $y_n = W \cdot x_m + B$

- La fonction d'erreur « loss » est calculée de manière globale (à la sortie output)

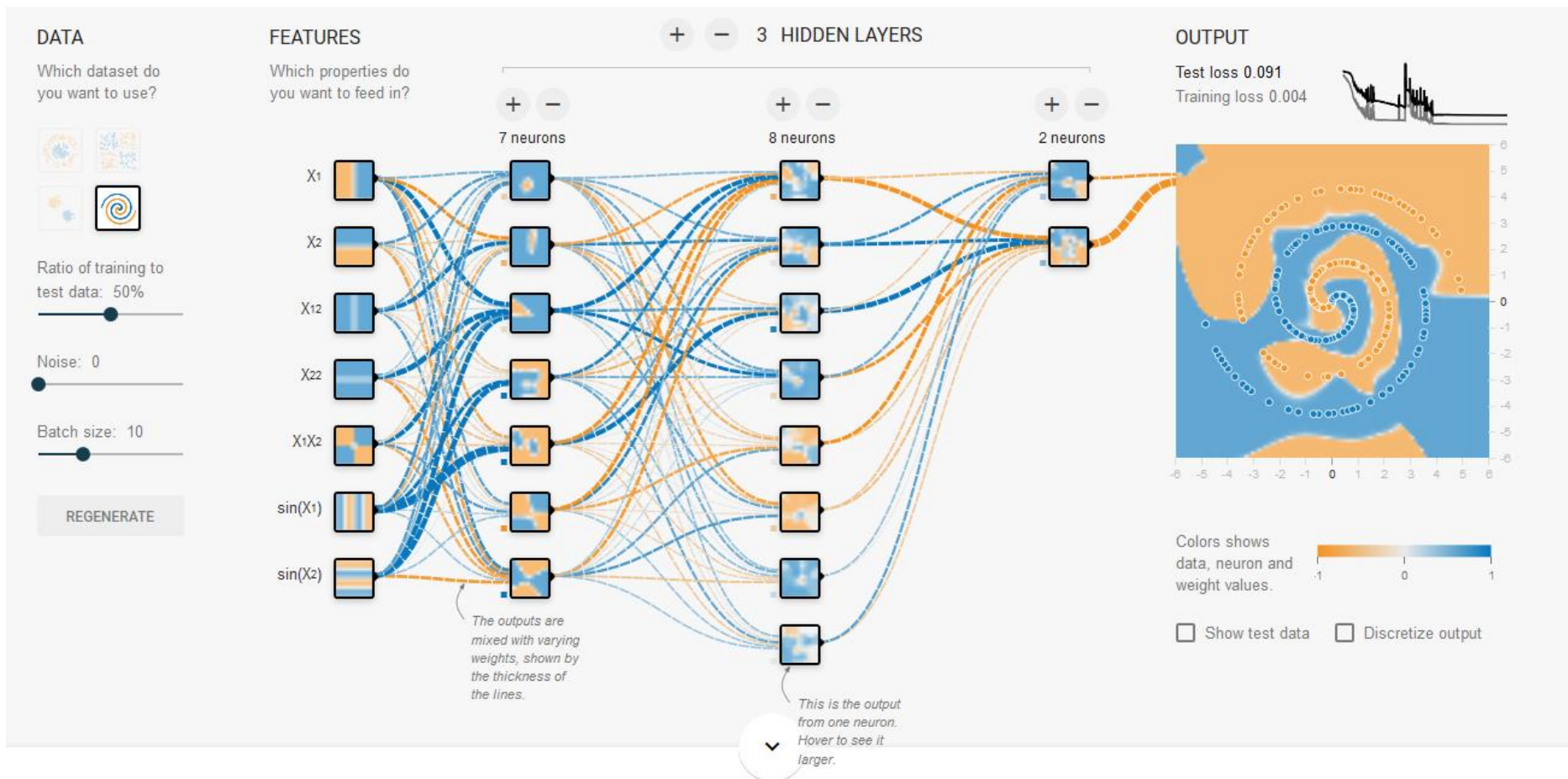
- Optimisation tous les neurones en même temps



Réseau de neurones

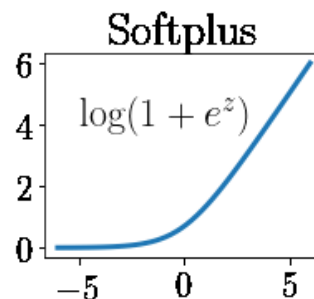
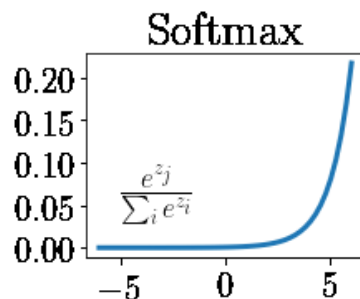
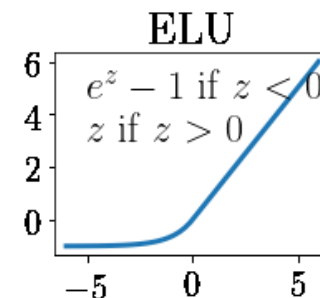
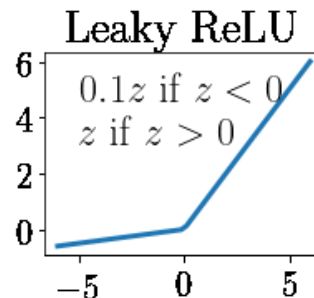
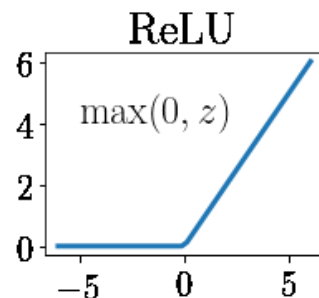
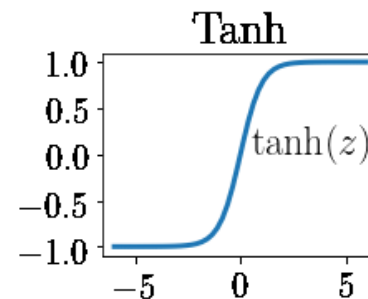
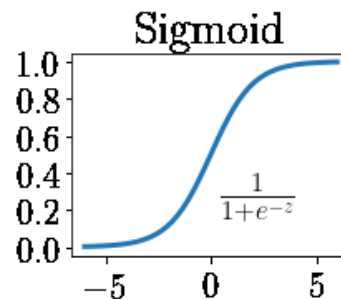
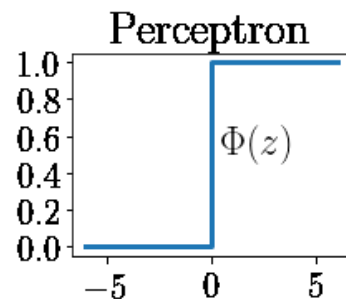
- Exemple de classification d'un nuage de points

<http://playground.tensorflow.org>



Problèmes non linéaires : fonctions d'activation

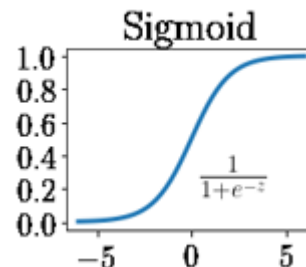
Les fonctions d'activation classique (possibilité d'avoir une fonction différente pour chaque neurone)



Fonctions d'activations

Sigmoïde (Logistique)

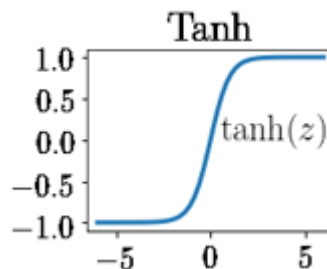
- **Formule** : $f(x) = \frac{1}{1+e^{-x}}$
- **Propriétés** :
 - Sortie comprise entre 0 et 1.
 - Idéale pour les modèles de classification binaire.
 - **Problèmes** :
 - Peut entraîner le problème du **gradient évanescent** dans les réseaux profonds, car les gradients deviennent très petits pour les entrées extrêmes.



Fonctions d'activations

Tanh (Tangente hyperbolique)

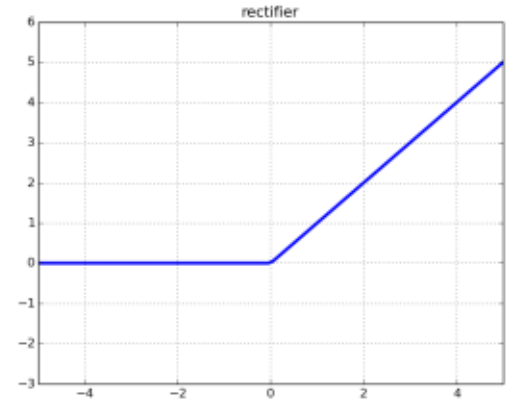
- **Formule :** $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Propriétés :**
 - Sortie comprise entre -1 et 1.
 - Généralement préférée à la fonction sigmoïde car centrée sur 0, facilitant la convergence pendant l'entraînement.
 - **Problèmes :** Peut également souffrir du gradient évanescent pour des valeurs extrêmes de xxx.



Fonctions d'activations

ReLU (Rectified Linear Unit)

- **Formule :** $f(x) = \max(0, x)$
- **Propriétés :**
 - Simple et efficace, largement utilisée dans les réseaux de neurones profonds.
 - Active uniquement les neurones ayant une sortie positive.
 - Aide à résoudre le problème du gradient évanescent dans les réseaux profonds.
 - **Problèmes :** Le phénomène de **neurones morts** peut apparaître lorsque trop de neurones produisent constamment des sorties nulles.
 - **Remarque :** non dérivable en 0 mais comme l'apprentissage se fait numériquement et que ReLU est dérivable partout ailleurs cela fonctionne



Fonctions d'activations

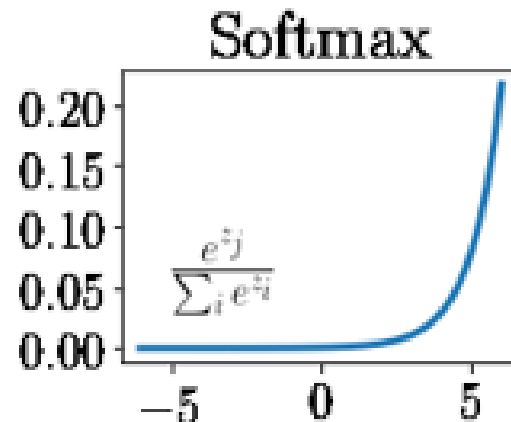
Leaky ReLU

- **Formule :** $f(x) = \max(0.01x, x)$
- **Propriétés :**
 - Une variante de ReLU qui permet aux valeurs négatives de ne pas être complètement nulles, en leur attribuant une petite pente (souvent 0,01).
 - Réduit le problème des neurones morts par rapport à ReLU standard.

Fonctions d'activations

Softmax

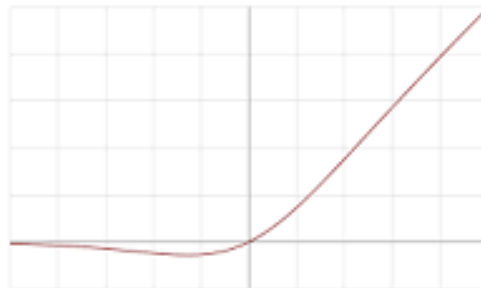
- **Formule :** $f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- **Propriétés :**
 - Convertit les valeurs de sortie en probabilités, où la somme des probabilités pour chaque classe est égale à 1.
 - Spécifique aux couches de sortie dans les tâches de **classification multiclasse**



Fonctions d'activations

Swish (proposée par Google)

- **Formule :** $f(x) = \frac{x}{1+e^{-x}}$
- **Propriétés :**
 - Une fonction d'activation plus récente, qui combine certains avantages de ReLU et de Sigmoid.
 - Permet une propagation plus fluide des gradients et fonctionne bien dans de nombreux scénarios.
 - Elle a montré des performances souvent meilleures que ReLU dans certains réseaux profonds, en offrant un compromis entre linéarité et non-linéarité



Fonctions d'activations

Choix de la fonction d'activation :

- **ReLU** et ses variantes (comme Leaky ReLU et ELU) sont les plus couramment utilisées dans les couches cachées des réseaux de neurones profonds
- **Sigmoïde** et **tanh** sont souvent utilisées dans les couches cachées des réseaux récurrents (RNN), bien que leur utilisation ait diminué au profit de ReLU et des fonctions comme Swish.
- **Softmax** est spécifiquement utilisée dans les couches de sortie pour la classification multiclasse.
- Le choix de la fonction dépend de la tâche, de l'architecture, etc.

La fonction d'erreur (LOSS)

- Entrainement ou apprentissage
 - = une optimisation pour trouver les bons poids W qui maximisent les résultats sur une base de connaissance (input X , output Y)
- La fonction d'erreur que l'on cherche à minimiser
 - Plusieurs types possibles suivant le problème

$$L1LossFunction = \sum_{i=1}^n |y_{true} - y_{predicted}|$$

$$L2LossFunction = \sum_{i=1}^n (y_{true} - y_{predicted})^2$$

- Généralement L2-Loss est préférée, mais si vous avez beaucoup d' *outliers* (valeurs aberrantes) dans vos données la L1 peut être meilleurs
- La fonction d'erreur peut comporter d'autres termes (somme)
- Même pour de la reconnaissance, de nombreuses autres fonctions d'erreur sont possibles : crossEntropy, etc. → voir plus loin

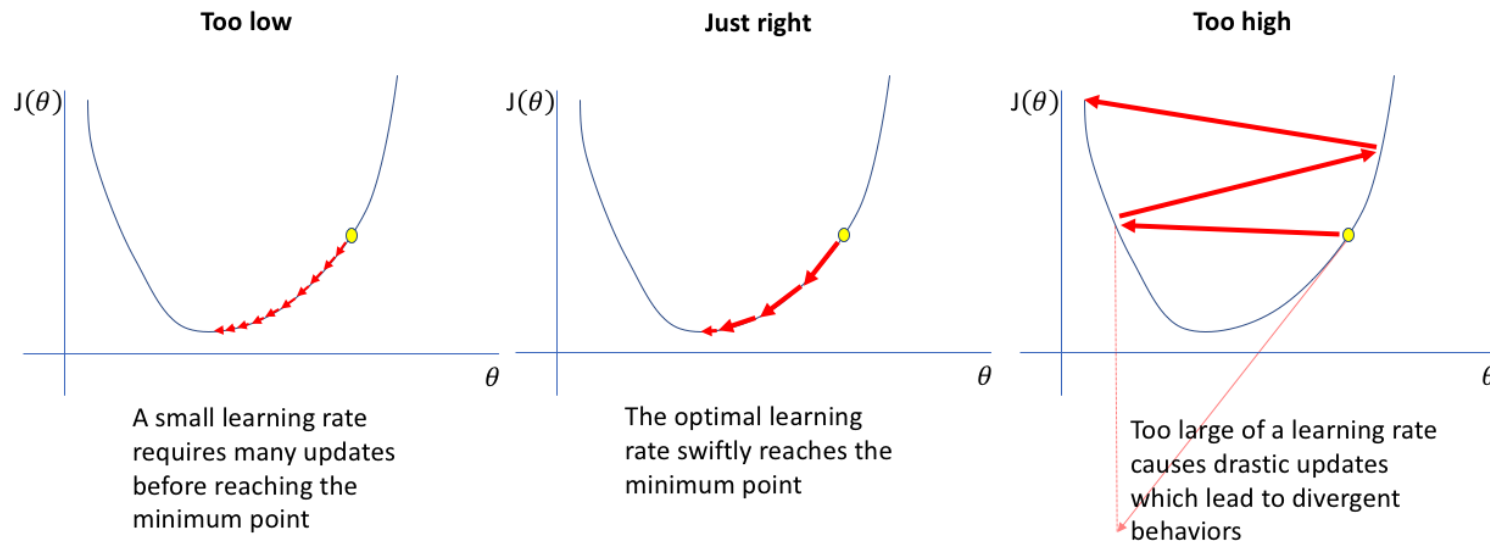
Entraînement : SGD + backpropagation

- Dans un réseau le calcul « naïf » du gradient peut être long, très long à calculer ! Avec N paramètres à optimiser (poids), le calcul du gradient demande N passes sur tout le réseau
 - Il faut calculer les N dérivées partielles
 - Bien sûr, on ne va pas faire ça
- Si la base de données d'apprentissage comporte M millions d'exemples, il faut passer M millions de fois dans le réseau (forward) pour avoir le résultat prédit
- Donc $N \times M$ passage dans le réseau pour une mise à jour des poids !

→ Des améliorations
 - SGD (Stochastic Gradient Descent)
 - Back propagation

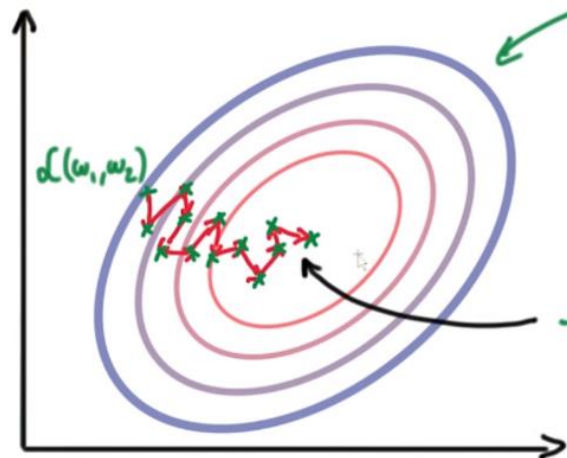
Réseau de neurones : entraînement

- Stochastic Gradient Descent (SGD)
 - **SGD est une Descente de gradient**
 - Learning rate = progression ou pas d'apprentissage
 - Il faut bien le choisir
 - Trop grand peut entrainer une dérive
 - Trop petit demandera plus de calcul et donc de temps



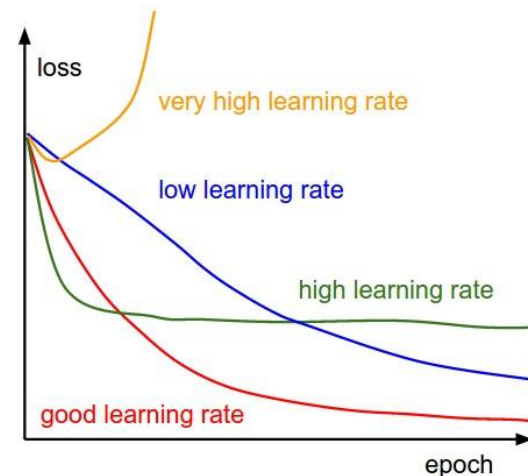
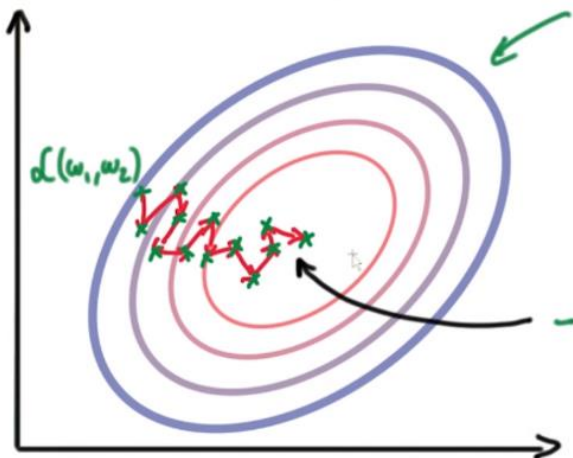
Réseau de neurones : entraînement

- Stochastic Gradient Descent (SGD)
 - Stochastic car la descente de gradient est calculée sur un sous-échantillonnage des données (epoch)
 - que M' exemples sont échantillonnés
 - Par exemple $M'=64$
 - ATTENTION : ces échantillons peuvent être très biaisés. La descente se fait alors en louvoyant.
 - Moment = comme stochastic le changement de direction se fait avec une inertie
 - Le gradient ne changera que lentement à chaque itération



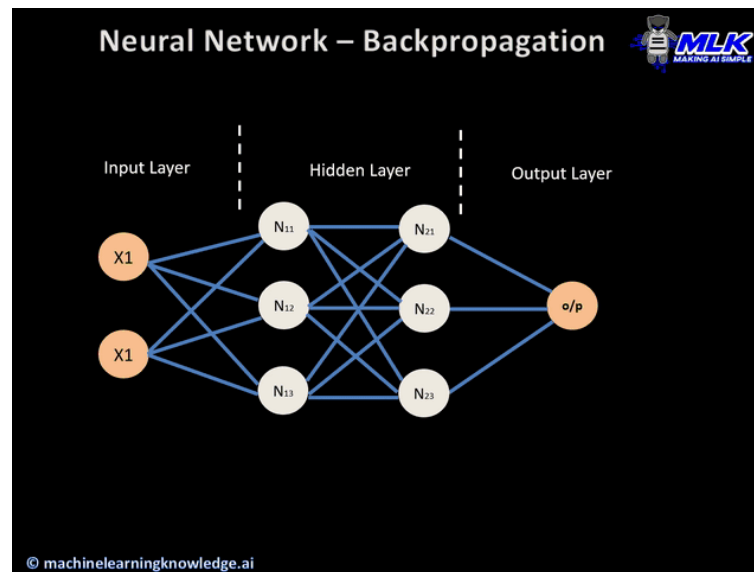
Réseau de neurones : entraînement

- Stochastic Gradient Descent (SGD)
 - Stochastic car la descente de gradient est calculée sur un sous-échantillonnage des données (epoch)
 - Learning rate = progression ou pas d'apprentissage
 - Moment = comme stochastic le changement de direction se fait avec une inertie
- Besoin des GPU pour la puissance de calcul
 - Par exemple, ImageNET 15 millions d'images avec 20 000 labels



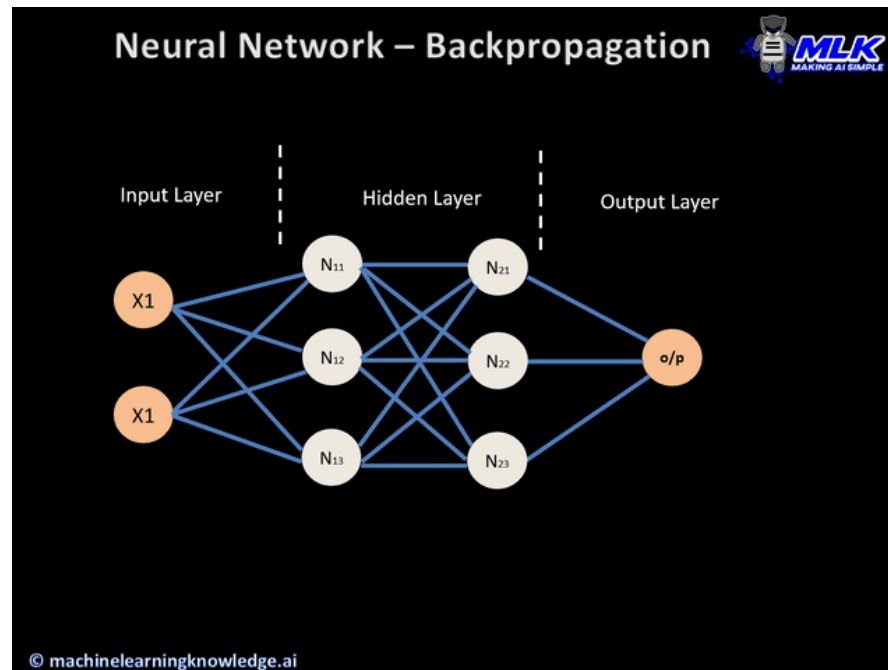
Entraînement : backpropagation

- Le calcul naïf (par différence finie) du gradient
 - long et convergence aléatoire
- L'algorithme de Back propagation permet
 - Calcul analytique dans chaque nœud : fonctions différentiables
 - Plus « juste » et plus efficace en termes de convergence
 - Calcul en remontant (plus efficace)



Entrainement : backpropagation

- L'algo de Back propagation permet de faire le calcul en remontant : en partant de l'erreur (la droite) et en remontant
 - A chaque étape, calcule de la valeur numérique du gradient dans chaque nœud
 - Rappel
ce gradient indique de combien varie l'erreur final si on fait varier le poids W
 - La descente de gradient utilise le gradient pour mettre à jour les poids

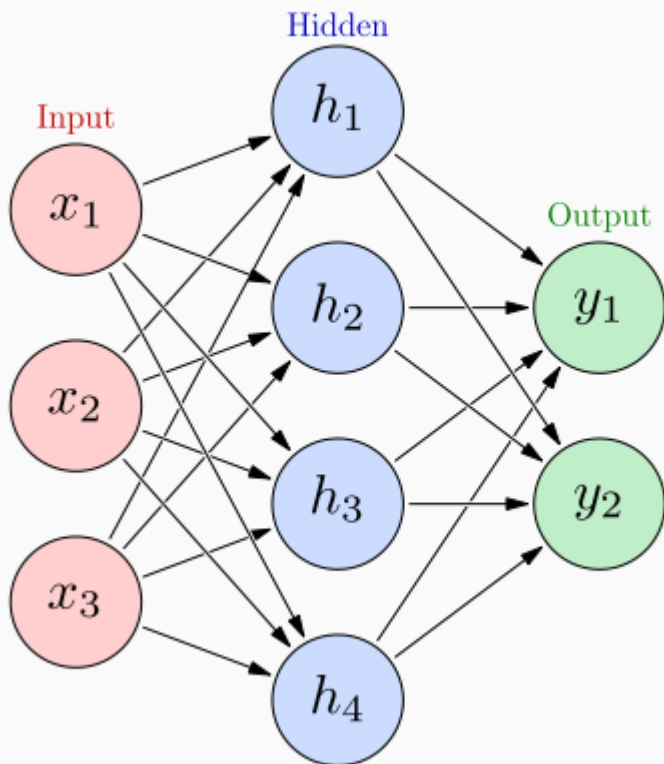


Un réseau from scratch

- En TP, code d'un réseau from scratch (python)
- Doc pour aller plus loin (voir page du TP)
 - https://www.idpoisson.fr/galerie/m2_reseaux_neurones/2_introduction_to_neural_networks.pdf
 - <https://www.miximum.fr/blog/introduction-au-deep-learning-1/>

Un réseau from scratch

w_{ij}^k est le poids entre le nœud précédent i et le suivant j de la couche cachée k
 g_k est la fonction d'activation de la couche k



$$h_1 = g_1 (w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1)$$

$$h_2 = g_1 (w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1)$$

$$h_3 = g_1 (w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1)$$

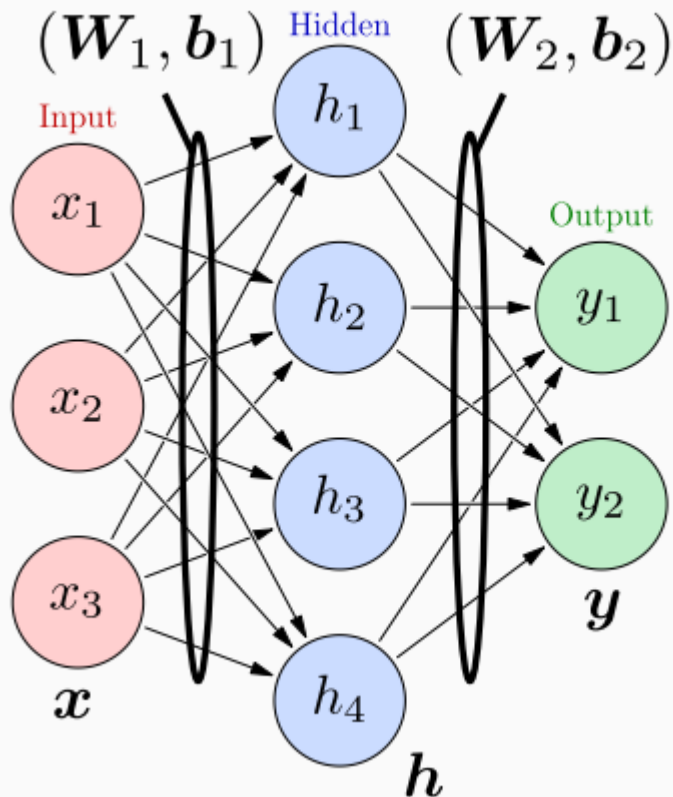
$$h_4 = g_1 (w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1)$$

$$y_1 = g_2 (w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2)$$

$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$

Entraînement : backpropagation

- Représentation matricielle
 - Optimisation des W et b (les poids)



$$h_1 = g_1 (w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1)$$

$$h_2 = g_1 (w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1)$$

$$h_3 = g_1 (w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1)$$

$$h_4 = g_1 (w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1)$$

$$h = g_1 (W_1 x + b_1)$$

$$y_1 = g_2 (w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2)$$

$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$

$$y = g_2 (W_2 h + b_2)$$

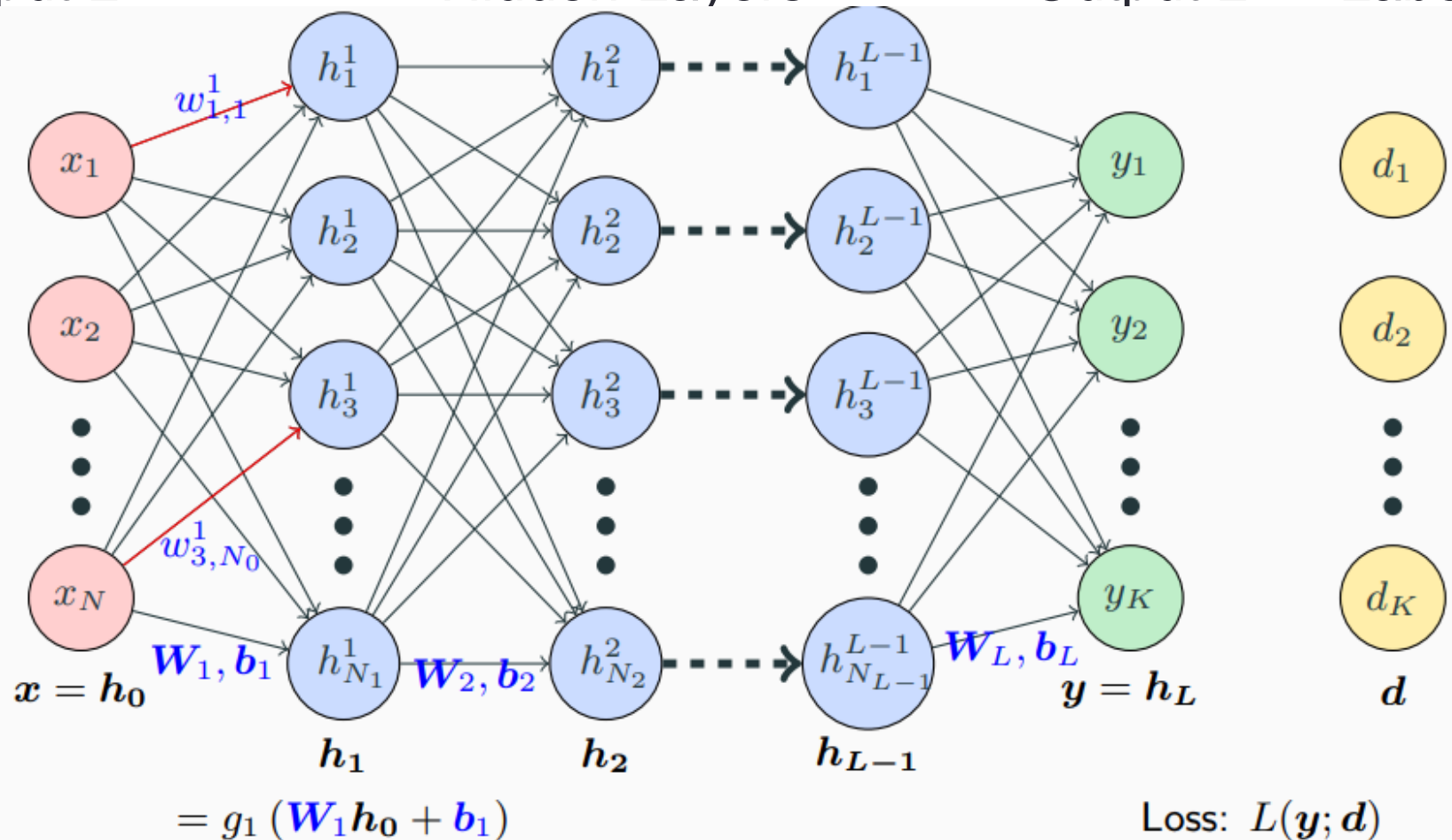
Feedforward

Input L

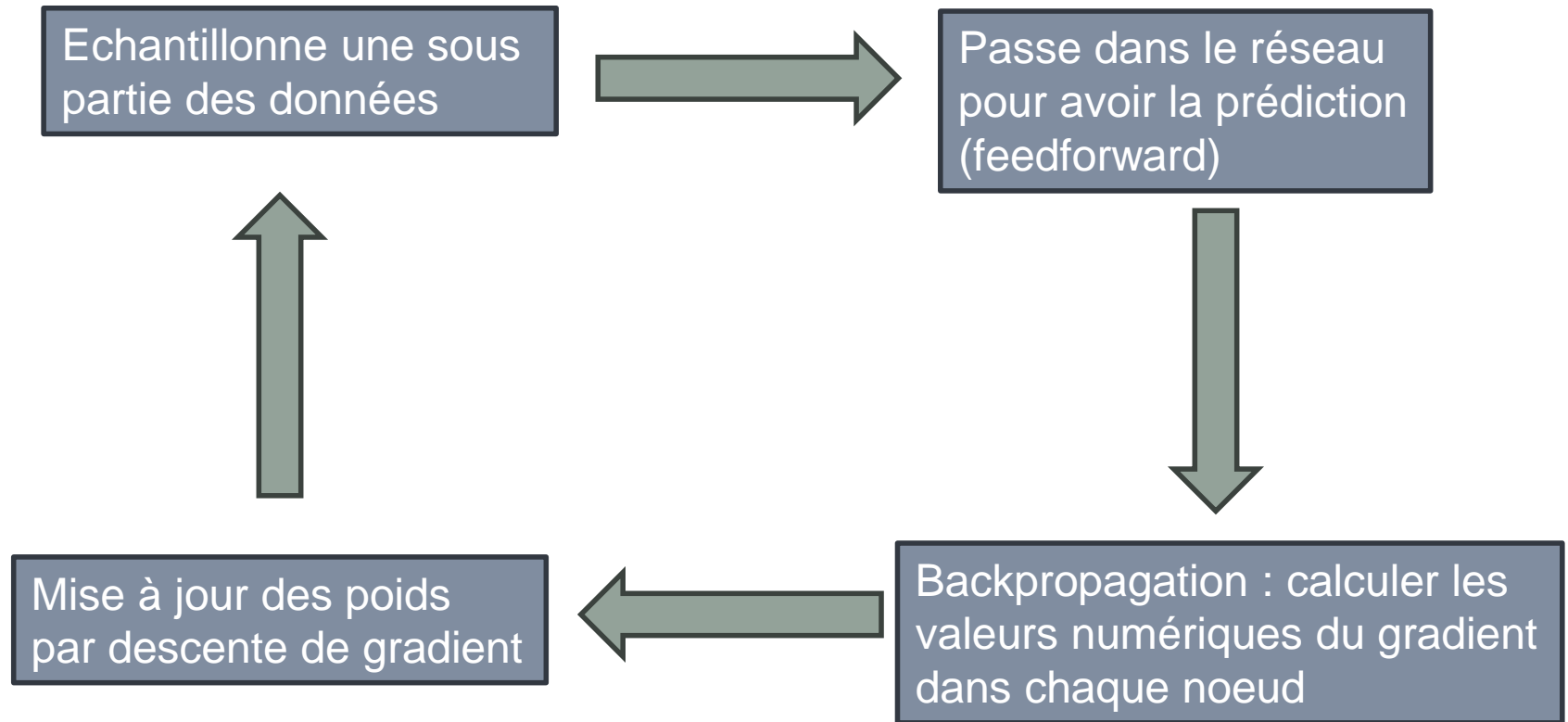
Hidden Layers

Output L

Label



Entraînement : apprentissage



- Les paramètres

$$\mathbf{W} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \dots, \mathbf{W}_L, \mathbf{b}_L)$$

Entrainement : apprentissage

- Les paramètres

$$\mathbf{W} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \dots, \mathbf{W}_L, \mathbf{b}_L)$$

- Fonction d'erreur E à minimiser (utilise la fonction /oss L)

Objective: $\min_{\mathbf{W}} E(\mathbf{W})$ where $E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i) \in \mathcal{T}} L(\mathbf{y}^i; \mathbf{d}^i)$

$$\Rightarrow \nabla E(\mathbf{W}) = \left(\frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_1} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_1} \quad \dots \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_L} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_L} \right)^T = 0$$

- Un minimum peut se trouver par descente de gradient
 - Répétition avec tous les exemples du batch l'un après l'autre

$$\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t - \gamma \nabla E(\mathbf{W}^t), \quad \gamma > 0$$

$$w_{i,j}^{k,t+1} \leftarrow w_{i,j}^{k,t} - \gamma \frac{\partial E(\mathbf{W}^t)}{\partial w_{i,j}^k}$$

Backpropagation
calcule le gradient

Apprentissage : LOSS

Objective: $\min_{\mathbf{W}} E(\mathbf{W})$ where $E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i) \in \mathcal{T}} L(\mathbf{y}^i; \mathbf{d}^i)$

$$\Rightarrow \nabla E(\mathbf{W}) = \left(\frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_1} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_1} \quad \dots \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_L} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_L} \right)^T = 0$$

- Les fonctions de loss classiques (fonction de coût)
 - Régression : erreur moyenne au carré (square loss) → MSE
 - Somme sur toutes les sorties

$$E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i) \in \mathcal{T}} \frac{1}{2} \|\mathbf{y}^i - \mathbf{d}^i\|_2^2 = \sum_{(\mathbf{x}^i, \mathbf{d}^i) \in \mathcal{T}} \frac{1}{2} \sum_k (y_k^i - d_k^i)^2$$

Apprentissage : LOSS

$$\text{Objective: } \min_{\mathbf{W}} E(\mathbf{W}) \quad \text{where} \quad E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i) \in \mathcal{T}} L(\mathbf{y}^i; \mathbf{d}^i)$$
$$\Rightarrow \nabla E(\mathbf{W}) = \left(\frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_1} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_1} \quad \dots \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_L} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_L} \right)^T = 0$$

- Les fonctions de loss classiques
 - ...
 - Classification multi classes : CROSS ENTROPY
 - Classification binaire (y classe réelle, \hat{y} probabilité prédite)

$$\text{Loss}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

- pour la classification multiclasse

$$\text{Loss}(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i)$$

Apprentissage : LOSS

- La fonction d'erreur L

$$E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i)} L(\mathbf{y}^i; \mathbf{d}^i)$$

- Son gradient (dérivée)

$$\nabla E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i)} \nabla L(\mathbf{y}^i; \mathbf{d}^i)$$

- Avec les deux loss classiques

- Dérivée de L par rapport à y (sortie, dernière couche)

- Regression/Square error:

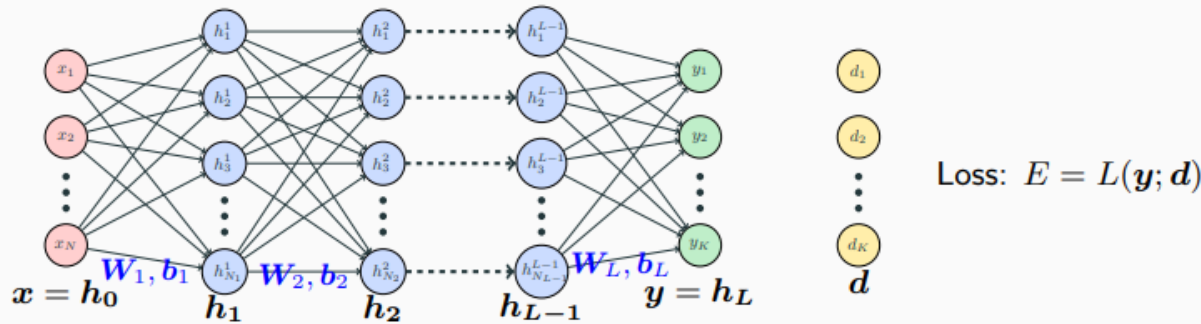
$$L(\mathbf{y}; \mathbf{d}) = \frac{1}{2} \|\mathbf{y} - \mathbf{d}\|_2^2 \Rightarrow \nabla_{\mathbf{y}} L(\mathbf{y}; \mathbf{d}) = \mathbf{y} - \mathbf{d}$$

- Multi-class classification/cross-entropy:

$$L(\mathbf{y}; \mathbf{d}) = -y_d + \log \left(\sum_{k=1}^K \exp(y_k) \right) \Rightarrow \nabla_{\mathbf{y}} L(\mathbf{y}; \mathbf{d}) = \text{softmax}(\mathbf{y}) - \mathbf{d}$$

- Maintenant il faut calculer la dérivée dans chaque couche
 - de la fonction d'erreur E par rapport à chaque poids

Apprentissage : rappel forward



Forward pass

Initialization:

$$h_0 = x$$

for layer $k = 1$ **to** L **do**

Linear unit:

$$a_k = W_k h_{k-1} + b_k$$

Componentwise non-linear activation:

$$h_k = g_k(a_k)$$

end

Output layer:

$$y = h_L$$

Compute loss:

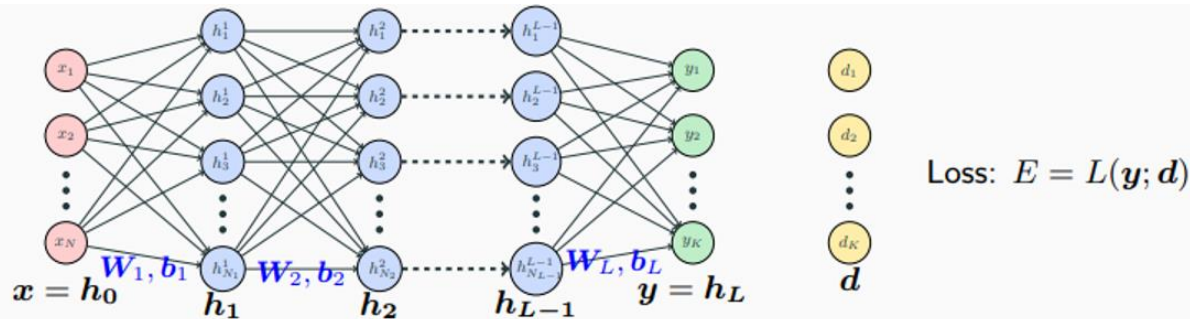
$$E = L(y; d)$$

$$a_L = W_L h_{L-1} + b_L$$

$$a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$$

$$h_k = g_k(a_k)$$

Apprentissage : backpropagation



Forward pass

Initialization:

$$\mathbf{h}_0 = \mathbf{x}$$

for layer $k = 1$ **to** L **do**

Linear unit:

$$\mathbf{a}_k = \mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k$$

Componentwise non-linear activation:

$$\mathbf{h}_k = g_k(\mathbf{a}_k)$$

end

Output layer:

$$\mathbf{y} = \mathbf{h}_L$$

Compute loss:

$$E = L(\mathbf{y}; \mathbf{d})$$

Backward pass

Goal: Compute the gradient with respect to all parameters

$$\frac{\partial E}{\partial w_{i,j}^k} = ? \quad \frac{\partial E}{\partial b_i^k} = ?$$

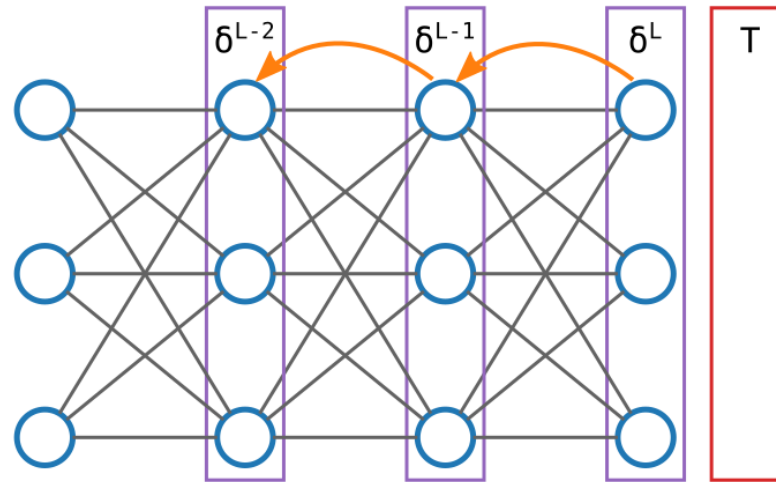
for all

$$k \in \{1, \dots, L\},$$

$$i \in \{1, \dots, N_k\},$$

$$j \in \{1, \dots, N_{k-1}\}.$$

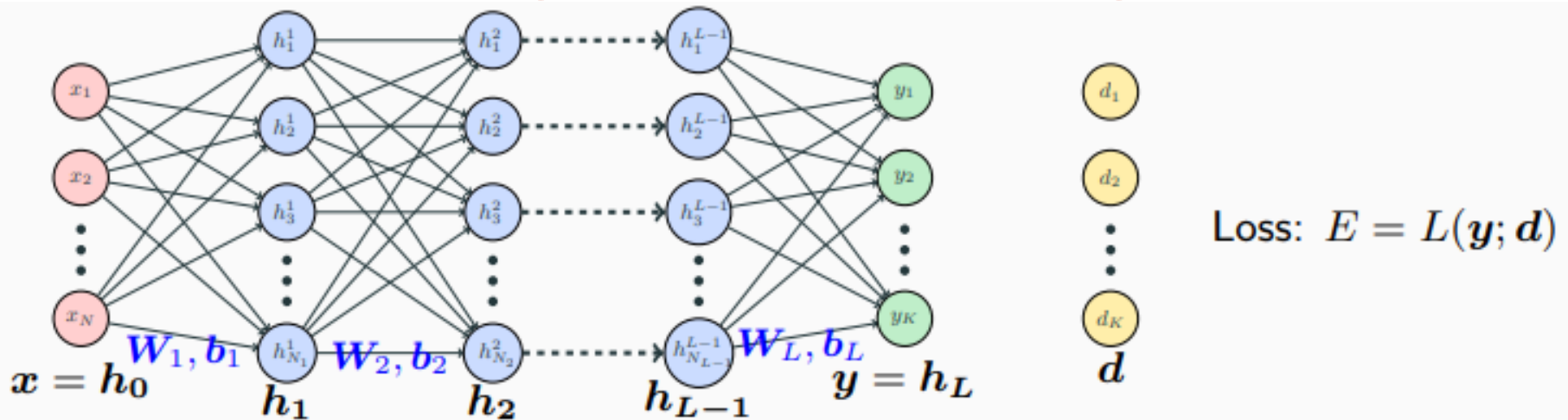
Apprentissage : backpropagation



- Remonte en arrière
 - Avec dérivation de la composée de deux fonctions dérivables

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \quad [f(g(x))]' = g'(f(x)) \cdot f'(x)$$

Apprentissage : backpropagation



- Pour la dernière couche L

$$\frac{dE}{dw_L^i} = \frac{dE}{dh_L^i} \frac{dh_L^i}{da_L^i} \frac{da_L^i}{dw_L^i}$$

Dérivée de
la loss

$$\nabla_{h_L} E$$

dérivée de la
fonction d'activation $g'_L(\mathbf{a}_L)$

h_i^{L-1} car

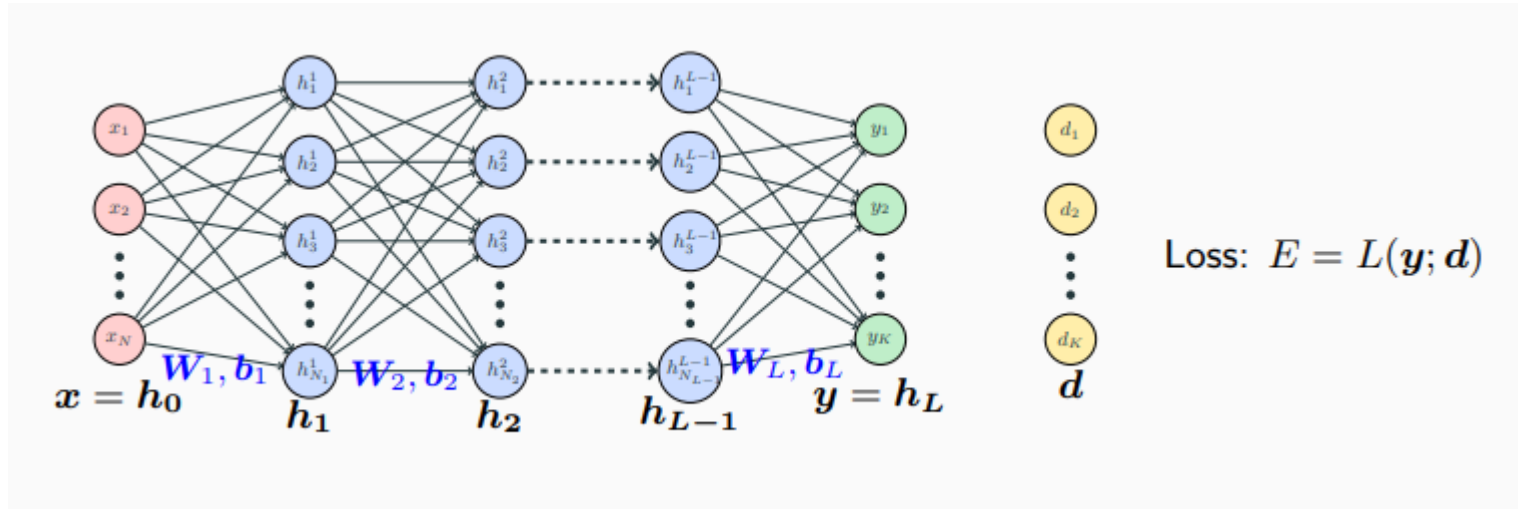
$$\nabla_{\mathbf{a}_L} E = \nabla_{h_L} E \odot g'_L(\mathbf{a}_L)$$

$$\nabla_{\mathbf{b}_L} E = \nabla_{\mathbf{a}_L} E$$

$$\nabla_{\mathbf{W}_L} E = \nabla_{\mathbf{a}_L} E \mathbf{h}_{L-1}^T$$

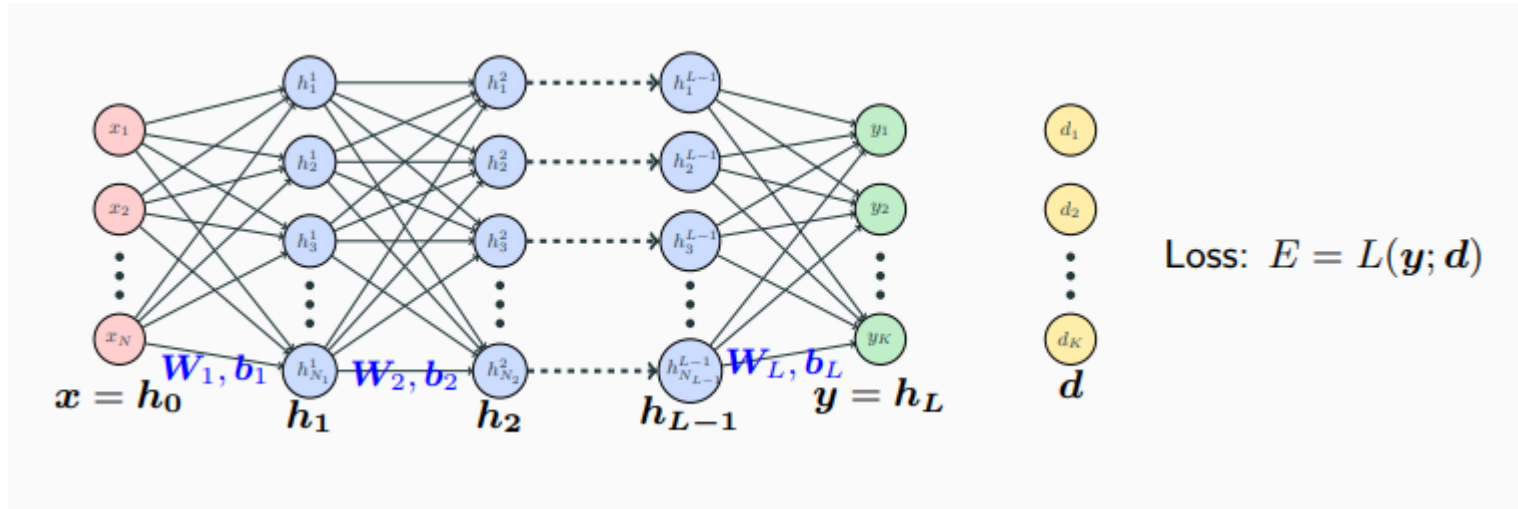
$$\mathbf{a}_L = \mathbf{W}_L \mathbf{h}_{L-1} + \mathbf{b}_L$$

Apprentissage : backpropagation



Comment calculer le gradient pour la couche L-1 ?

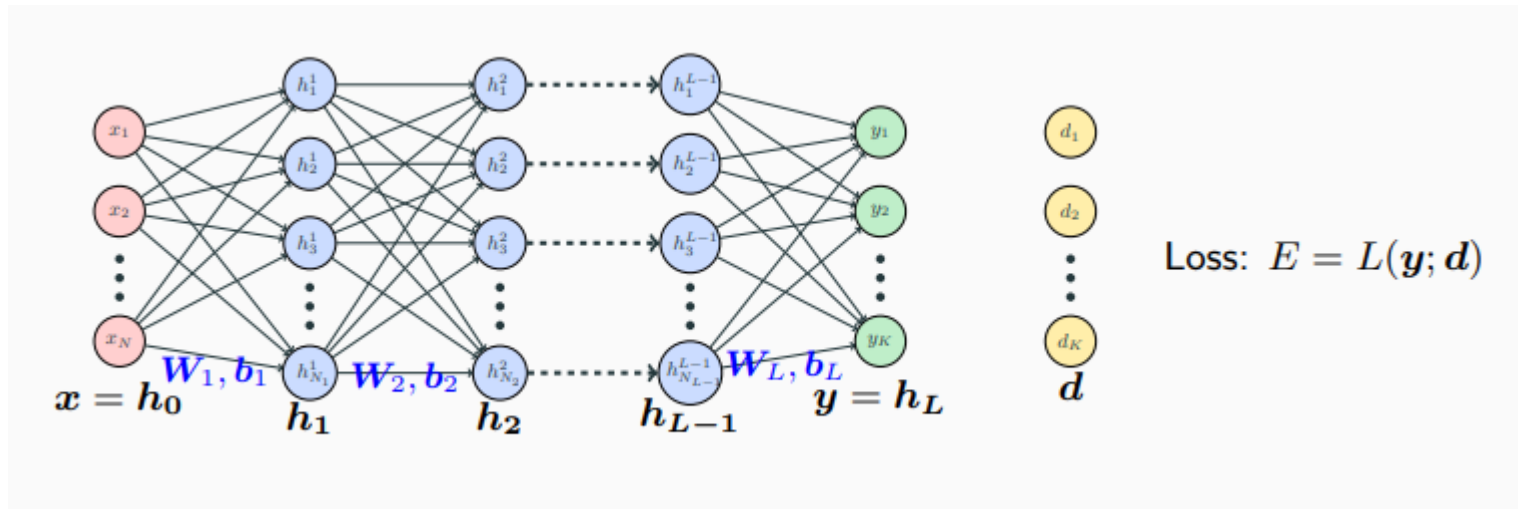
Apprentissage : backpropagation



Comment calculer le gradient pour la couche L-1 ?

- Un peu différent parce que h_j^{L-1} apparait dans tous les neurones de la couche L

Apprentissage : backpropagation



Comment calculer le gradient pour la couche L-1 ?

- Rappel sur la dérivée d'une fonction affine

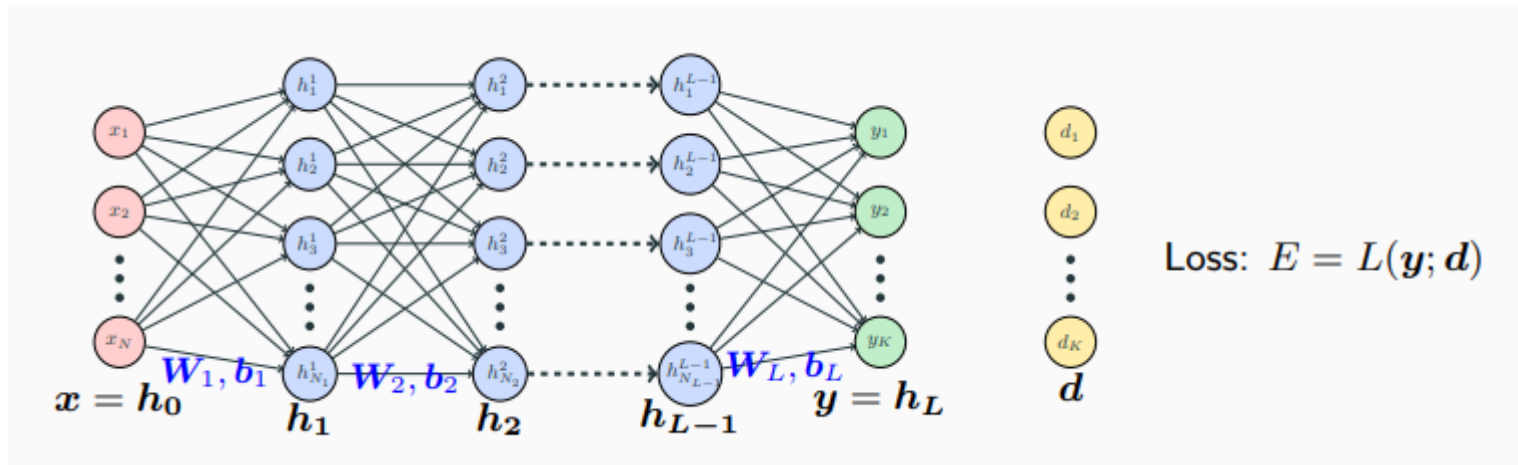
- Soit

$$\varphi(x) = f(Ax + b)$$

- sa dérivée

$$\nabla \varphi(x) = A^T \nabla f(Ax + b).$$

Apprentissage : backpropagation



Comment calculer le gradient pour la couche L-1 ?

$$\varphi(x) = f(Ax + b)$$

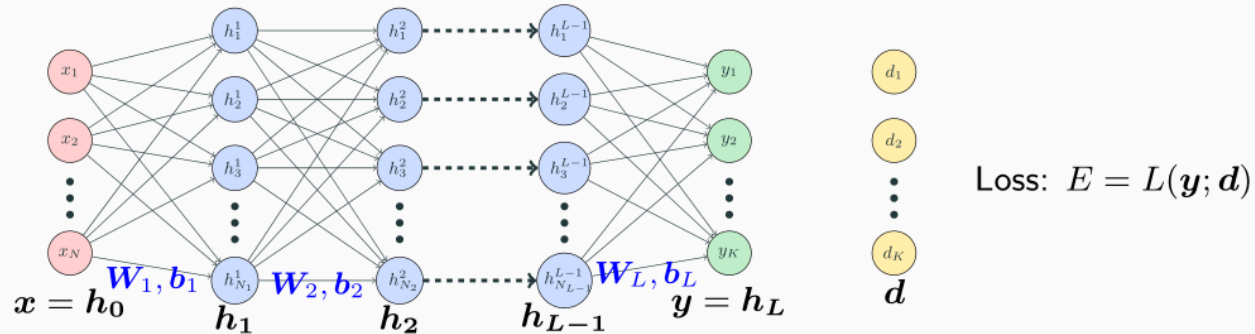
$$\nabla \varphi(x) = A^T \nabla f(Ax + b).$$

- f = fonction d'activation, noté g_L pour nous
- Donc avec $a_L = W_L h_{L-1} + b_L$ et $h_L = g_L(a_L)$, on obtient

$$\nabla_{h_{L-1}} E = W_L^T \nabla_{a_L} E$$

Couche L-1 dépend de L (à droite)
 ➔ remonte de droite à gauche

Apprentissage : backpropagation



Forward pass

Initialization:

$$\mathbf{h}_0 = \mathbf{x}$$

for layer $k = 1$ **to** L **do**

Linear unit:

$$\mathbf{a}_k = \mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k$$

Componentwise non-linear activation:

$$\mathbf{h}_k = g_k(\mathbf{a}_k)$$

end

Output layer:

$$\mathbf{y} = \mathbf{h}_L$$

Compute loss:

$$E = L(\mathbf{y}; \mathbf{d})$$

Backward pass

Initialization: Gradient of output layer:

$$\nabla_{\mathbf{h}_L} E = \nabla L(\mathbf{y}; \mathbf{d})$$

for layer $k = L$ **to** 1 **do**

Componentwise gain of error:

$$\delta_k = \nabla_{\mathbf{a}_k} E = \nabla_{\mathbf{h}_k} E \odot g'_k(\mathbf{a}_k)$$

Gradient of layer bias:

$$\nabla_{\mathbf{b}_k} E = \delta_k$$

Gradient of weights:

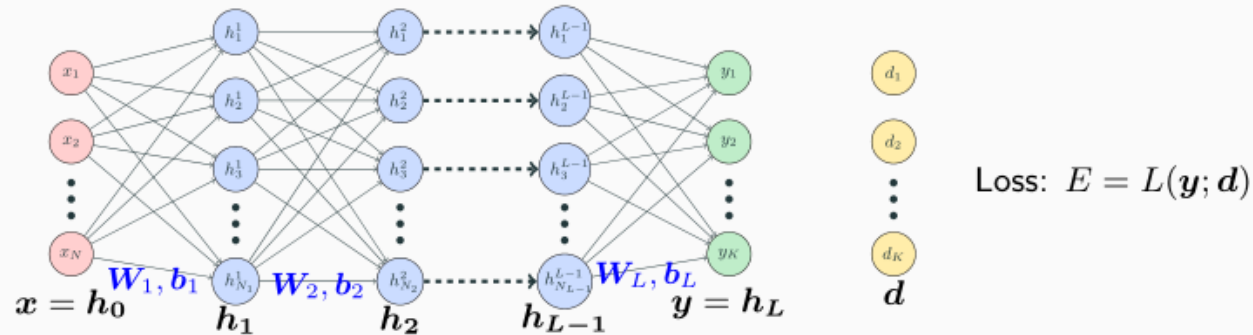
$$\nabla_{\mathbf{W}_k} E = \delta_k \mathbf{h}_{k-1}^T$$

Gradient of previous hidden layer:

$$\nabla_{\mathbf{h}_{k-1}} E = \mathbf{W}_k^T \delta_k$$

end

Apprentissage : backpropagation



Forward pass

Initialization:

$$\mathbf{h}_0 = \mathbf{x}$$

for layer $k = 1$ **to** L **do**

Linear unit:

$$\mathbf{a}_k = \mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k \text{ (stored)}$$

Componentwise non-linear activation:

$$\mathbf{h}_k = g_k(\mathbf{a}_k) \text{ (stored)}$$

end

Output layer:

$$\mathbf{y} = \mathbf{h}_L$$

Compute loss:

$$E = L(\mathbf{y}; \mathbf{d})$$

Backward pass

Initialization: Gradient of output layer:

$$\nabla_{\mathbf{h}_L} E = \nabla L(\mathbf{y}; \mathbf{d})$$

for layer $k = L$ **to** 1 **do**

Componentwise gain of error:

$$\delta_k = \nabla_{\mathbf{a}_k} E = \nabla_{\mathbf{h}_k} E \odot g'_k(\mathbf{a}_k)$$

Gradient of layer bias:

$$\nabla_{\mathbf{b}_k} E = \delta_k$$

Gradient of weights:

$$\nabla_{\mathbf{W}_k} E = \delta_k \mathbf{h}_{k-1}^T$$

Gradient of previous hidden layer:

$$\nabla_{\mathbf{h}_{k-1}} E = \mathbf{W}_k^T \delta_k$$

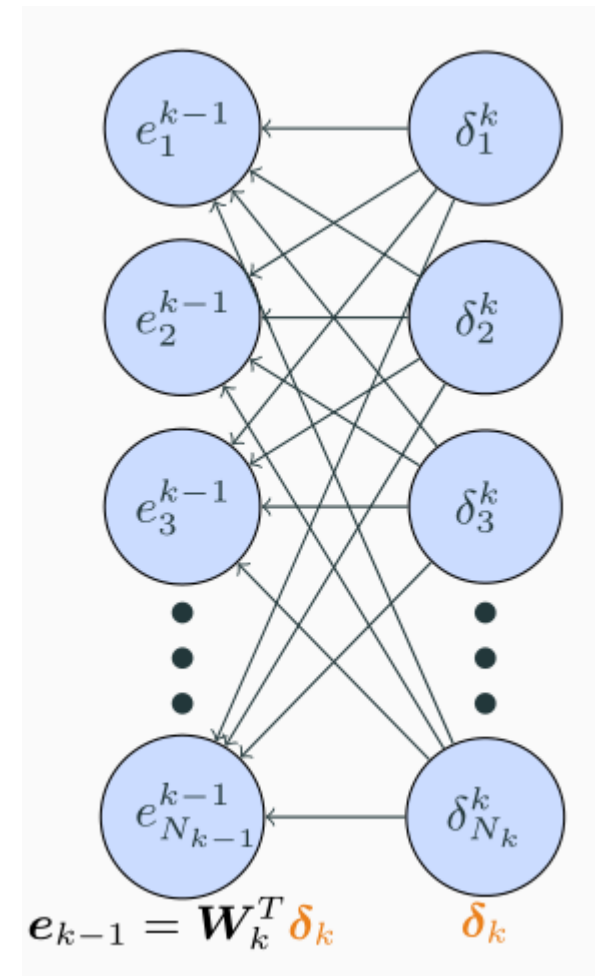
end

Apprentissage : backpropagation

- Gradient de la couche précédente (à gauche)

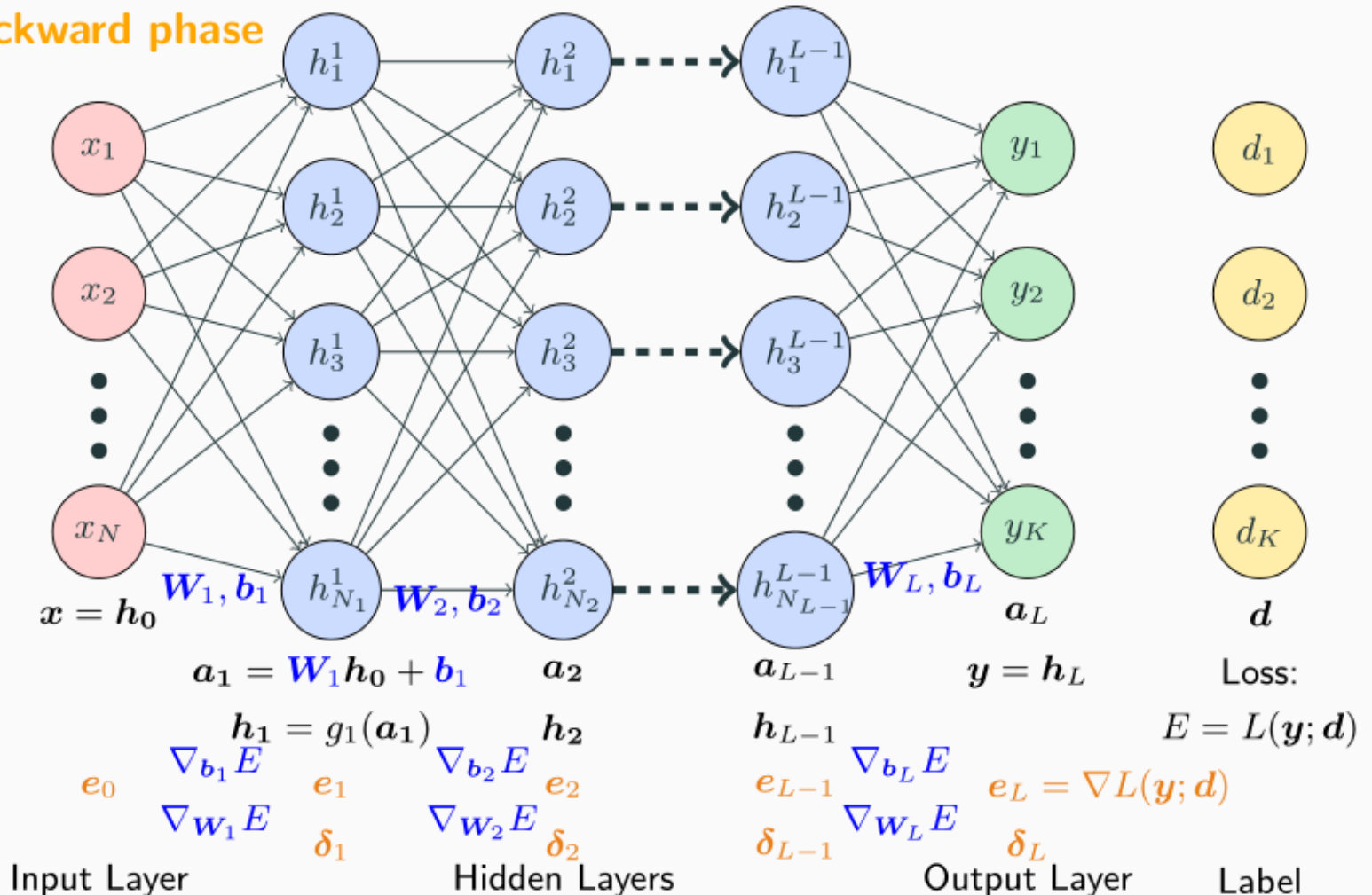
$$e_{k-1} = \nabla_{h_{k-1}} E = \mathbf{W}_k^T \boldsymbol{\delta}_k$$

- Multiplier par \mathbf{W}^T revient à passer dans le réseau dans le sens inverse (transformation linéaire)
- L'erreur est rétro propagée de la fin vers le début pour calculer le gradient



Apprentissage : backpropagation

Backward phase



Apprentissage : backpropagation

Remarques

- Le gradient est calculé pour chaque donnée du batch
- Il est sommé
 - pas besoin de garder les valeurs intermédiaires
 - et moyenné à la fin du batch
- Il faut remettre le gradient à 0 la fin d'un batch

➔ VOIR LE TP

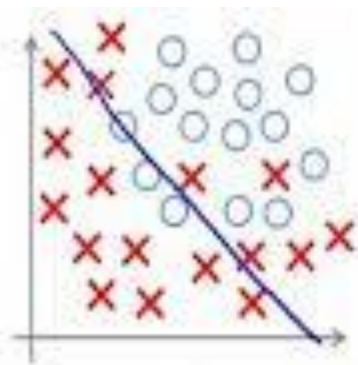
Réseau de neurones : du code

- Plateformes en python
 - TensorFlow (Google)
 - PyTorch (Yann Lecun, maintenant Facebook)
 - ...
- Accélération GPU
- Des exemples de code
 - Pytorch tutorial
 - Keras
 - une sur-couche de TensorFlow facilitant le codage des réseaux
 - <https://keras.io/examples/>
 - Kaggle

➔ Une communauté immense, des exemples de code (github, etc.) par millions, reproductibilité du code, etc.

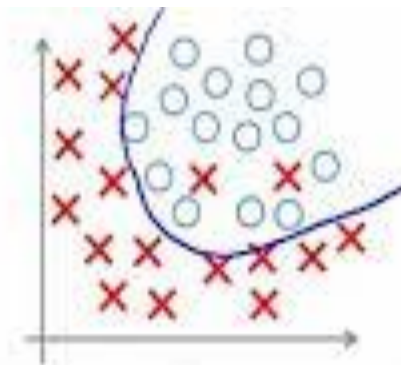
Surapprentissage / Overfitting

- Surapprentissage / Overfitting : c'est quoi ?
 - Quand un modèle apprend trop de détail/bruit sur les données avec pour conséquence d'être mauvais sur la généralisation (ie. Quand il verra des nouvelles données jamais vu avant)
 - Comme un étudiant qui « apprend par cœur » sans « comprendre »

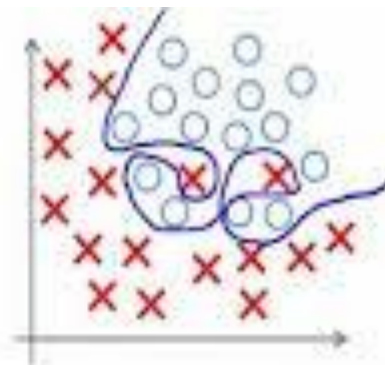


Under-fitting

(too simple to
explain the
variance)



Appropriate-fitting



Over-fitting

(overfitting – too
good to be true)

Régularisation

Les techniques de **Regularisation** sont les techniques utilisées pour résoudre le surapprentissage (overfitting) en apprentissage machine

Par exemple

- L1 Regularization or Lasso Regularization

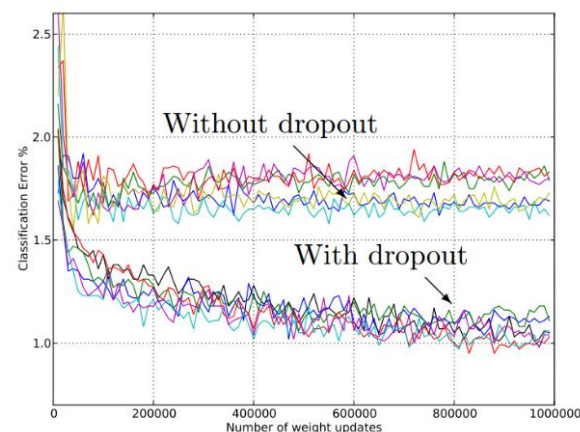
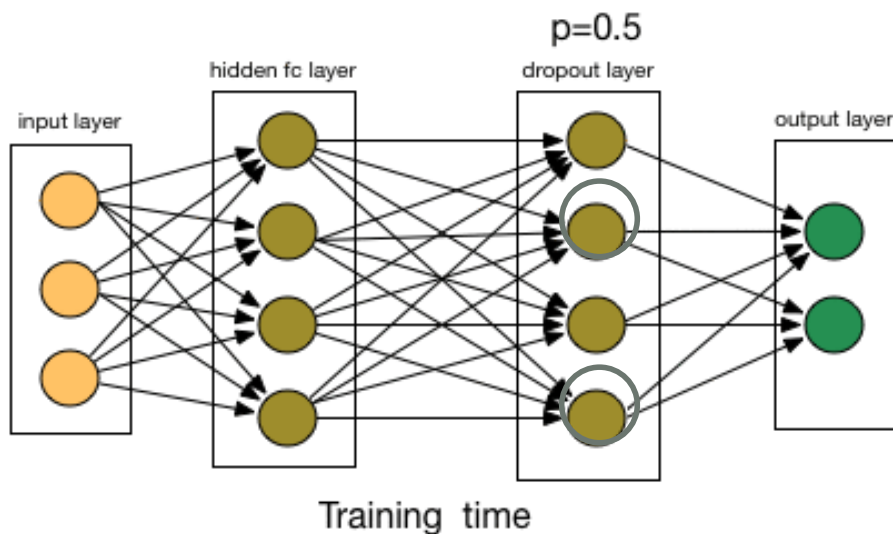
$$\text{Min}(\sum_{i=1}^n (y_i - w_i x_i)^2 + p \sum_{i=1}^n |w_i|)$$

- L2 Regularization or Ridge Regularization

$$\text{Min}(\sum_{i=1}^n (y_i - w_i x_i)^2 + p \sum_{i=1}^n (w_i)^2)$$

Drop Out

- Dropout aide pour éviter le surapprentissage (overfitting)
 - force le réseau à apprendre de manière plus robuste
 - technique de [régularisation](#)
- Dropout annule temporairement aléatoirement certains neurone avec une proba p



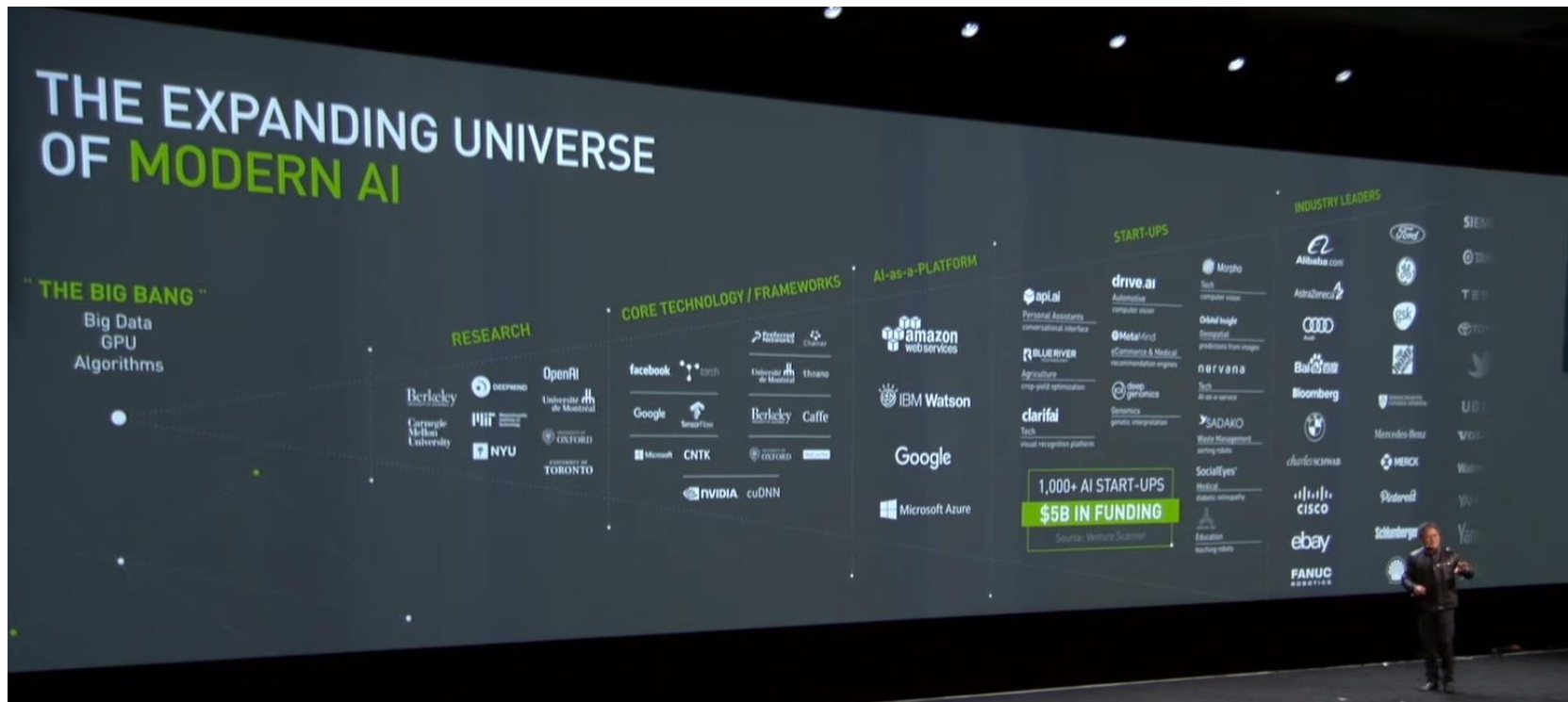
Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov

JMLR 2014

Explosion du deep learning

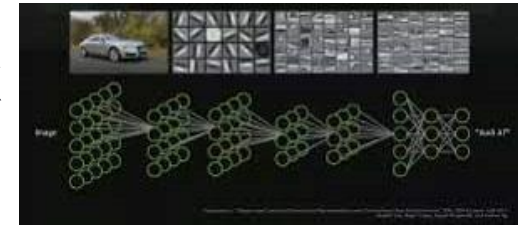
- Des données
- Du GPU
- Un algo de back-propagation rapide et facilement codable sur GPU
- Des algo, des algo, des algo ...



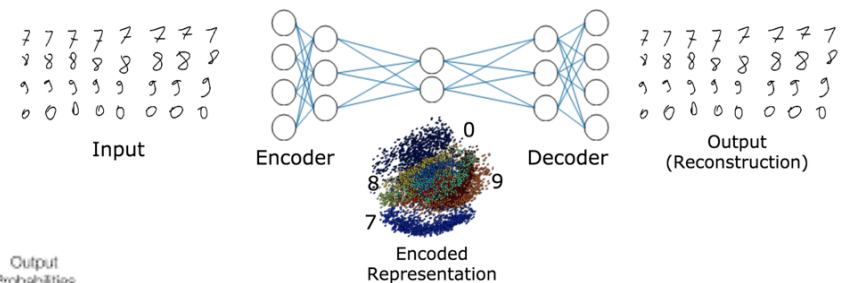
Explosion du deep learning

- Invention de différents « types » de réseaux

- ConvNN



- Auto-encoder



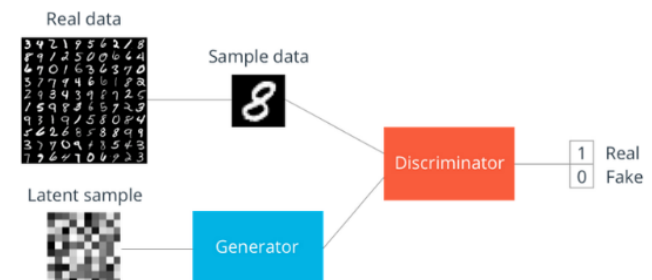
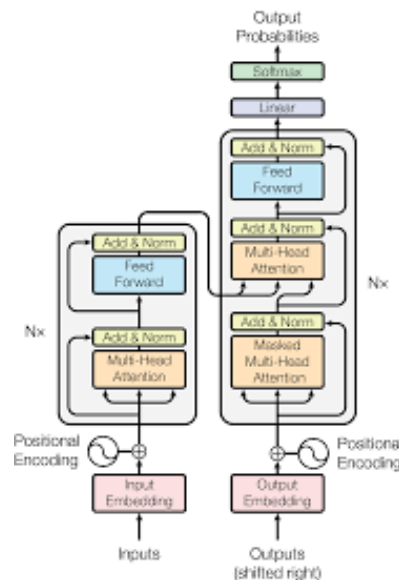
- GAN

- ...

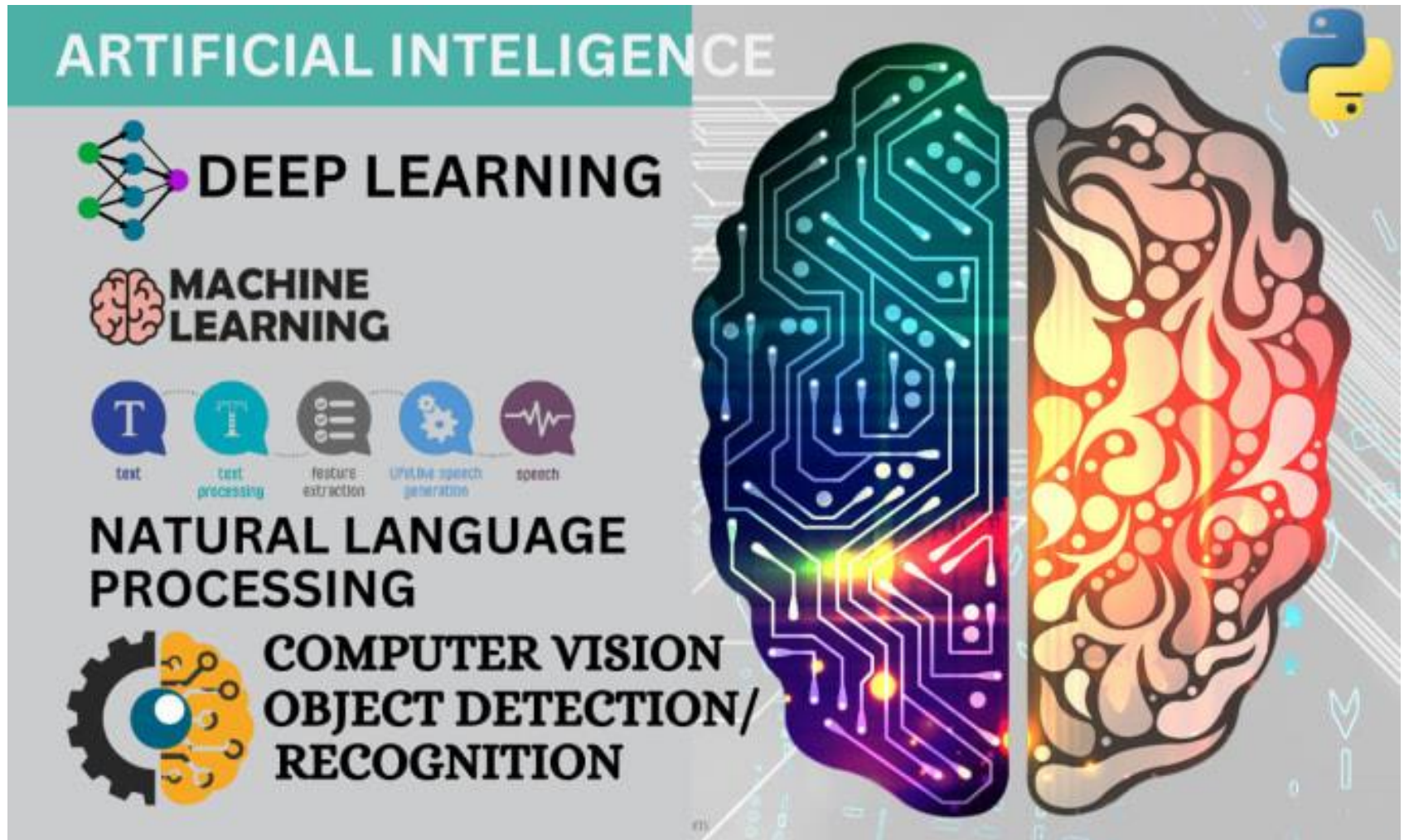
- Diffusion

- Transformer / attention

- **les stars du moment**



Conclusion 1



Conclusion 2

- Tous les domaines de la science ~~vont~~ sont touchés
- Les réseaux sont le moteur
- Les données le carburant



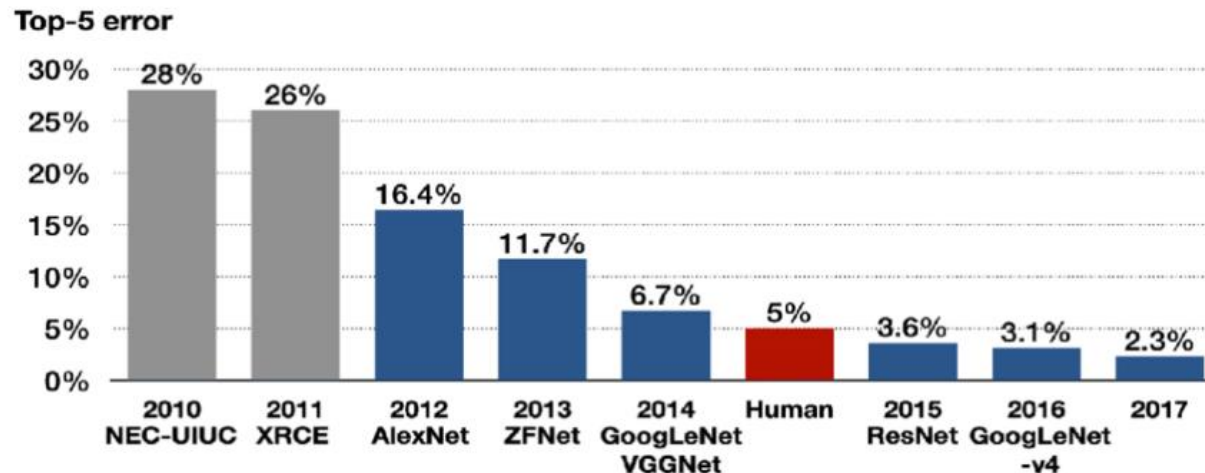
- Mais
 - Maintenant, les inventions se font en trouvant de nouvelles architectures : LSTM, AE, GAN, Transformer, Diffusion, etc.
 - Il reste beaucoup à faire ...
 - Explicabilité
 - Moins de données (frugal), renforcement, non supervisé, etc.
 - Transfert
 - Contrôle/ édition / interaction : apprentissage en continu, etc.
 - Je pense que les réseaux vont rester la brique de base, cela ne veut pas dire qu'il suffit des données et tout est réglé !

VRAC

Corpus d'images



- Classification
 - Vision humaine : repérer un prédateur ou un membre de sa famille
 - Concours IMAGENET → mettre un label sur une image
 - **10 millions d'images avec 11000 classes, 1.4To**



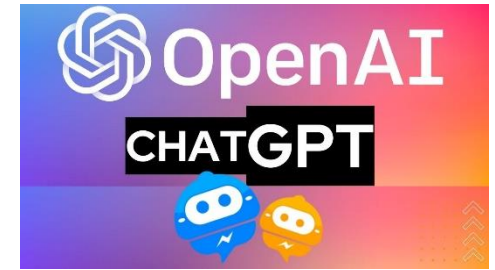
En bleu, méthodes à base de réseaux

ChatGPT Generative Pre-trained Transformer



- Dans son moteur, il y a deux parties
 - modèle géant de prédiction de texte combiné avec un modèle encyclopédique
 - modèle conversationnel
 - Apprentissage par renforcement à partir de retour humains
 - Seulement pour la conversation
- **Entrainé avec 500 milliards de mots**
- **175 Milliards de paramètres**
 - 100 milliards de neurone dans un cerveau humain mais ATTENTION à la comparaison

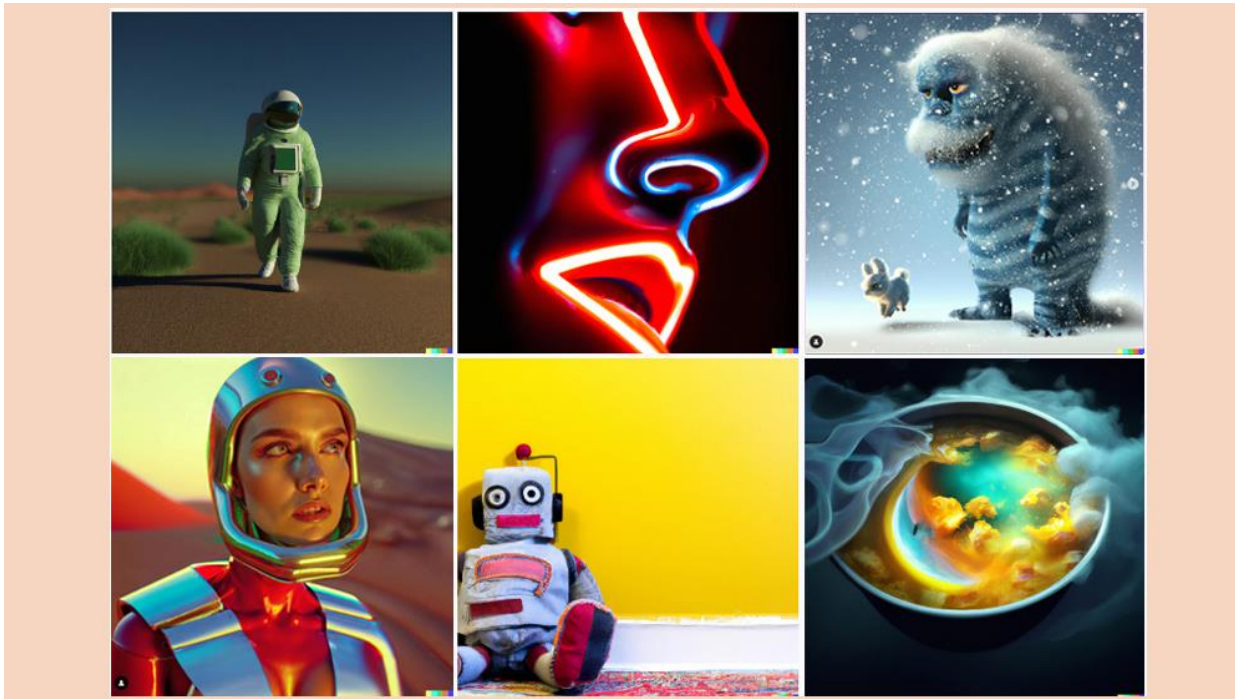
ChatGPT



- Existe dans les labos depuis 2018 mais
 - Grand public demandent de la puissance
 - Risque de mauvais buzz s'il se trompe !
 - Le modèle linguistique n'est pas actualisé (stop en 2021)
 - Le modèle conversationnel
 - Mémoire de 3000 mots pour la conversation
 - continue de s'affiner en fonction du retour des utilisateurstoutes les 3-4 semaines en moyenne
- ➔ pas de nouvelles connaissances mais générations phrases de meilleures qualités et, inversement, pénalise davantage les générations malvenues

Générateur d'images

- DALL-E : générateur d'images à partir de texte
 - Réseau de diffusion
 - auto-encoder basée sur un bruit inversible
 - Couplé à un modèle de texte



A suivre ...