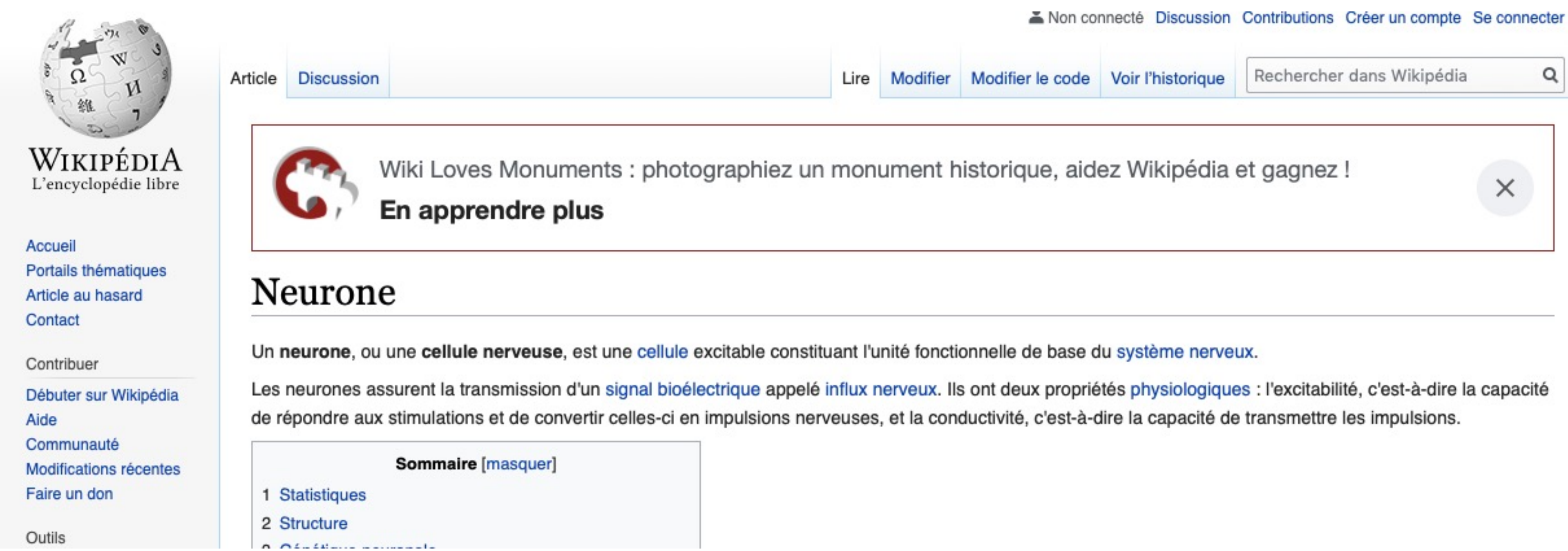
Les réseaux de neurones

Pascal Poncelet
LIRMM
Pascal.Poncelet@lirmm.fr
http://www.lirmm.fr/~poncelet

Qu’est ce que c’est qu’un neurone ?



Non connecté Discussion Contributions Créer un compte Se connecter

Article Discussion Lire Modifier Modifier le code Voir l’historique Rechercher dans Wikipédia

WIKIPÉDIA
L’encyclopédie libre

Accueil
Portails thématiques
Article au hasard
Contact

Contribuer

Débuter sur Wikipédia
Aide
Communauté
Modifications récentes
Faire un don

Outils

Wiki Loves Monuments : photographiez un monument historique, aidez Wikipédia et gagnez !
En apprendre plus

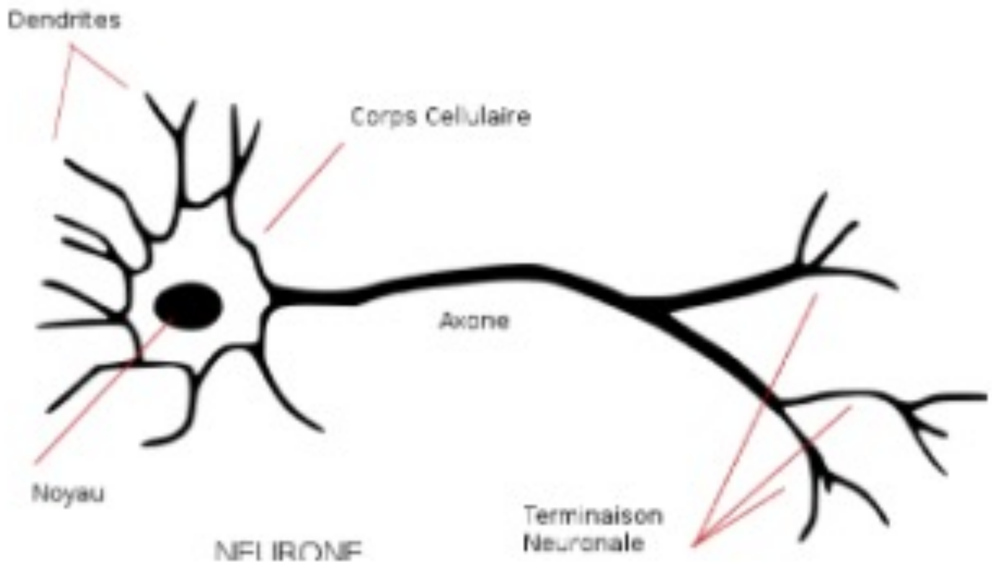
Neurone

Un **neurone**, ou une **cellule nerveuse**, est une cellule excitable constituant l'unité fonctionnelle de base du système nerveux.

Les neurones assurent la transmission d'un signal bioélectrique appelé influx nerveux. Ils ont deux propriétés physiologiques : l'excitabilité, c'est-à-dire la capacité de répondre aux stimulations et de convertir celles-ci en impulsions nerveuses, et la conductivité, c'est-à-dire la capacité de transmettre les impulsions.

Sommaire [masquer]

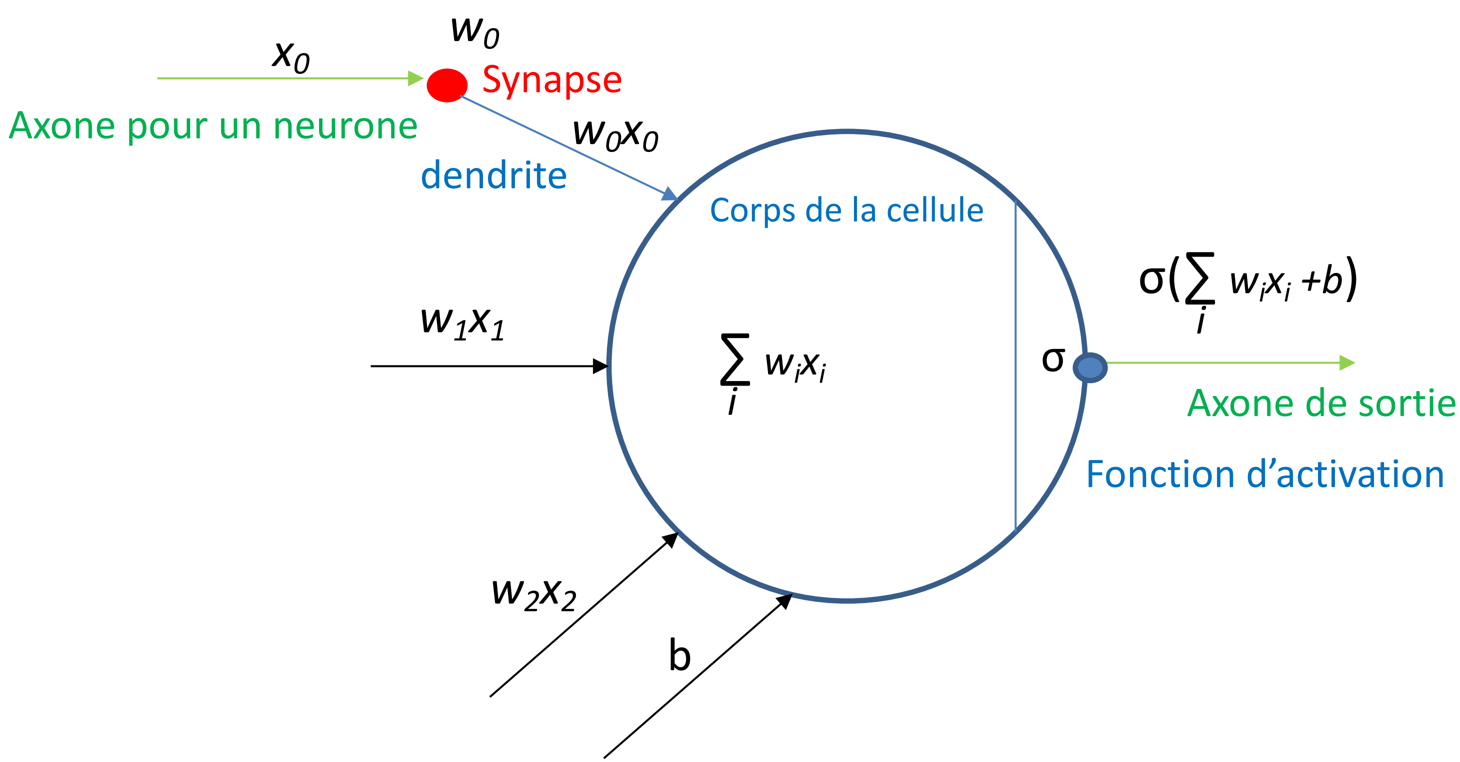
Un neurone



Les dendrites reçoivent l'influx nerveux d'autres neurones. Le neurone évalue alors l'ensemble de la stimulation reçue. Si celle-ci est suffisante, il est excité : il transmet un signal (0/1) le long de l 'axone et l'excitation est propagée jusqu'aux autres neurones qui y sont connectés via les synapses.

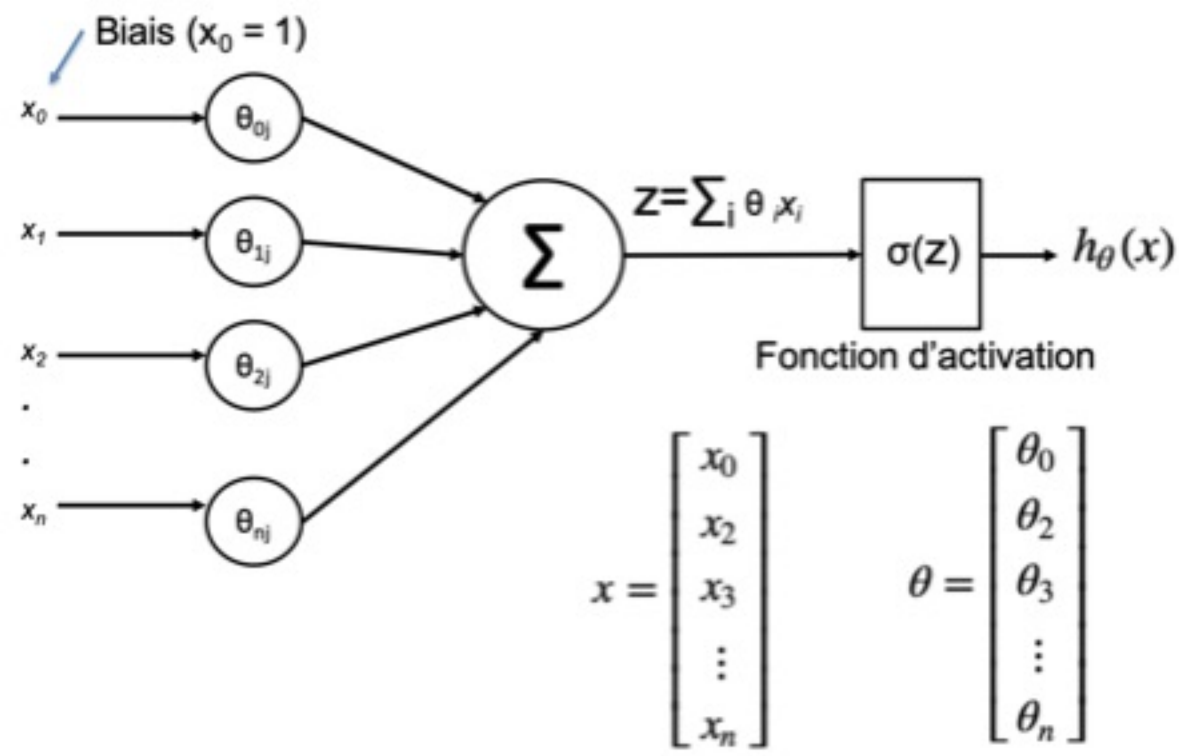
Un neurone artificiel

"Un réseau de neurones artificiels, ou réseau neuronal artificiel, est un système dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques" (Wikipedia)

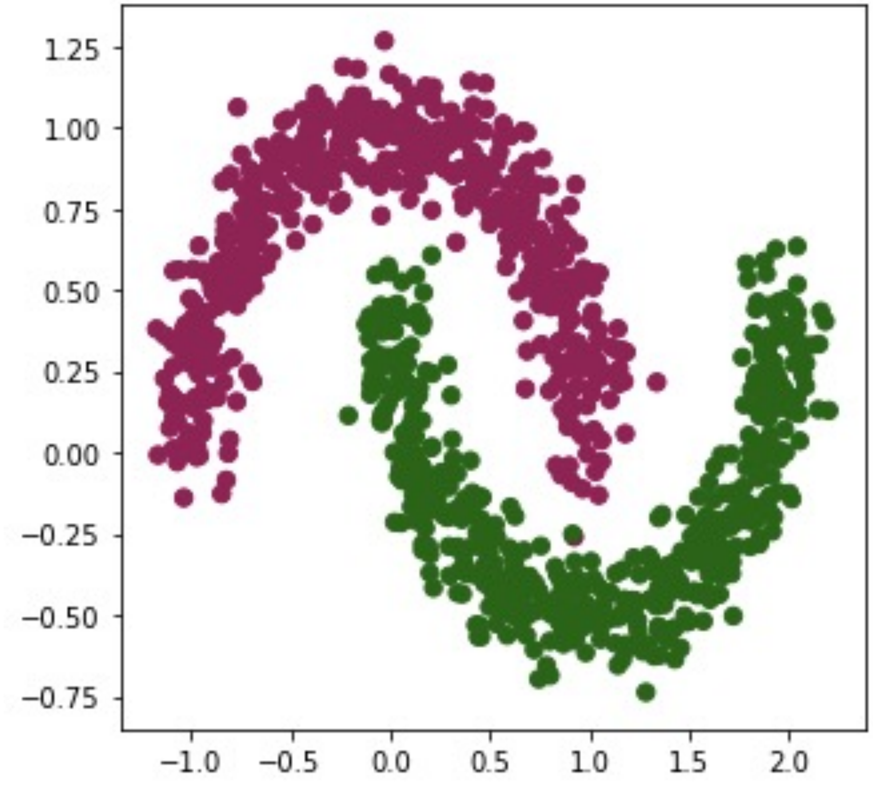


La régression logistique

- Rappel : utilisation d'une Sigmoid

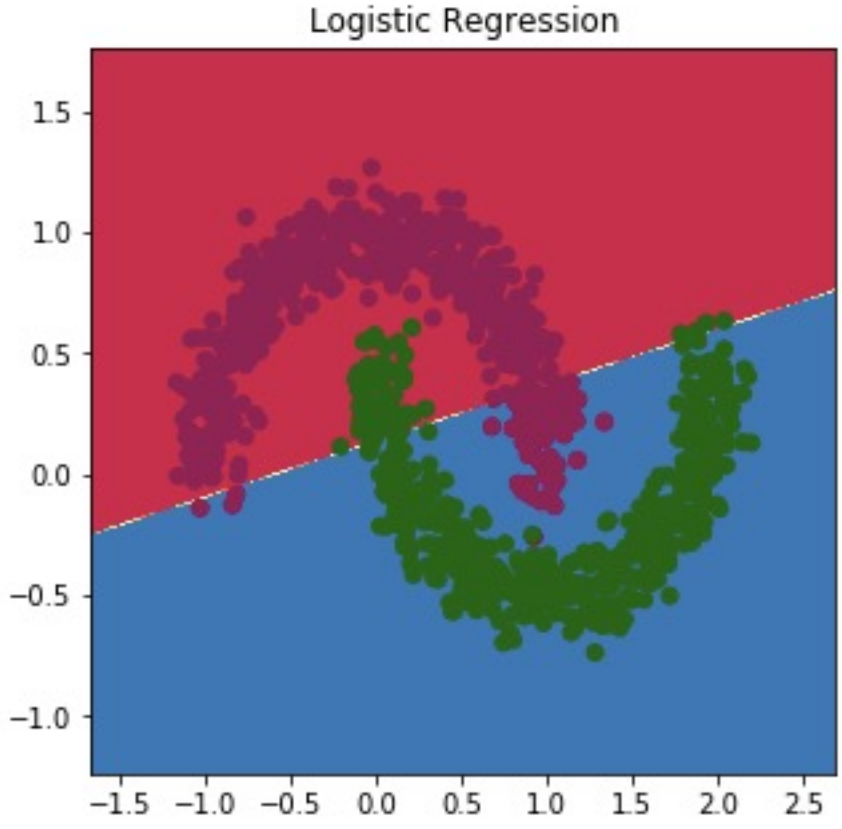


Appliquons la sur ce jeu de données



Quelle est la frontière de prédiction ?

Appliquons la sur ce jeu de données



Une droite

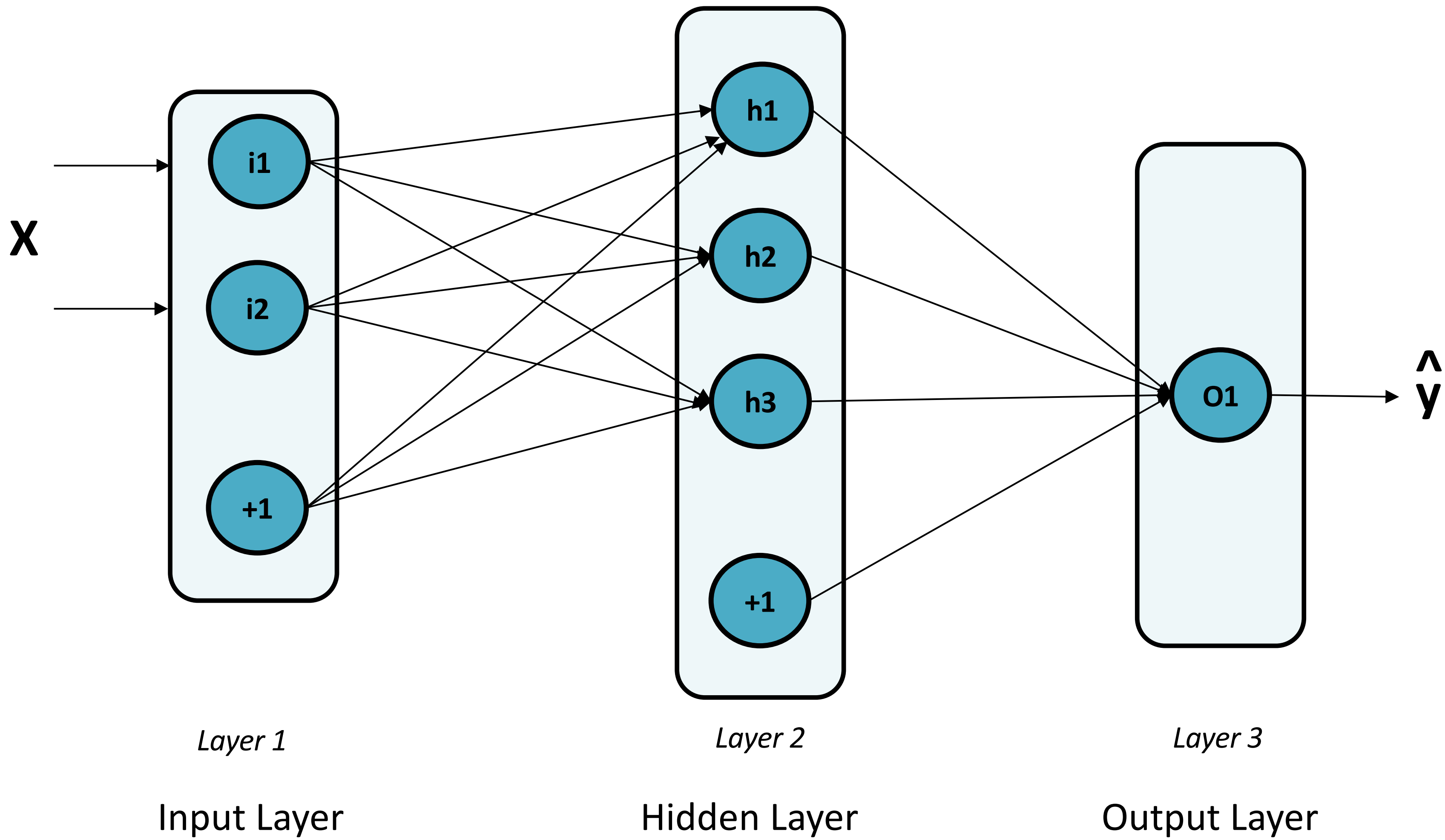
Les réseaux de neurones

- Les réseaux de neurones se composent des éléments suivants :
  - Une couche d'entrée qui reçoit l'ensemble des caractéristiques (features), i.e. les variables prédictives
  - Un nombre arbitraire de couches cachées
  - Une couche de sortie, $\hat{y}$, qui contient la variable à prédire
  - Un ensemble de poids W qui vont être ajoutés aux valeurs des features et de biais b entre chaque couche
  - Un choix de fonction d'activation pour chaque couche cachée, σ

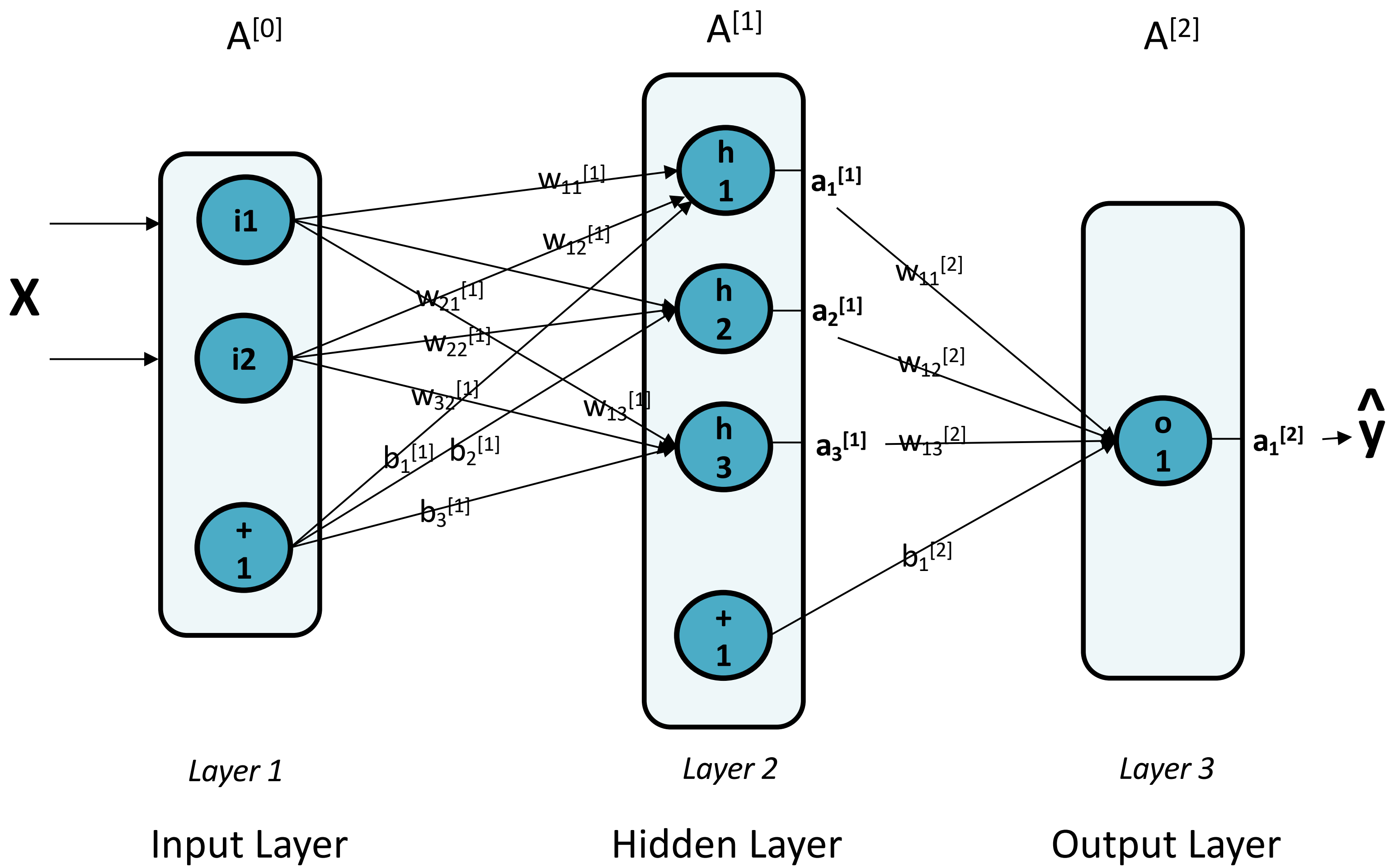
Couche de sortie

- Elle doit avoir autant de neurones qu'il y a de sorties au problème de classification :
- *régression* : 1 seul neurone (C.f. notebook descente de gradient)
- *classification binaire* : 1 seul neurone avec une fonction d'activation qui sépare les deux classes
- *classification multi-classe* : 1 neurone par classe et une fonction d'activation Softmax pour avoir la classe appropriée en fonction des probabilités de l'entrée appartenant à chaque classe

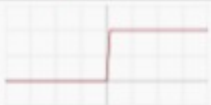
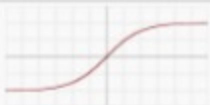
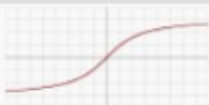
Un exemple de réseau



Un exemple de réseau

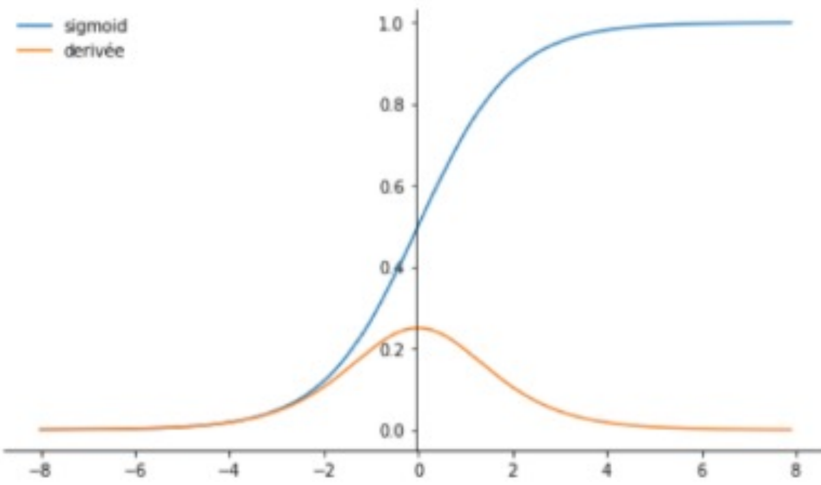


Choix de la fonction d'activation

Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Attention aux propriétés

- les réseaux de neurones utilisent la descente de gradient
  - le comportement de la dérivée des fonctions est important
- sigmoid transforme de grandes valeurs d'entrée dans des valeurs comprises entre 0 et 1
  - modification importante de l'entrée entraîne une modification mineure de la sortie
  - la dérivée est encore plus petite



Disparition de gradient

- *Vanishing gradient*
- Généralement des réseaux avec beaucoup de couches
- De trop petites petites valeurs de gradient (le gradient de la fonction de perte approche 0) indiquent que les poids des premiers layers ne seront pas mis à jour efficacement à chaque étape
- Imprécision globale du réseau
  - *Exemple : réseau composé de nombreuses couches avec une sigmoid*

Mort d'un neurone

- *Dead neuron*
- C'est un neurone qui, lors de l'apprentissage, ne s'active plus
- Lié au fait que les dérivées sont très petites ou nulles. Le neurone ne peut donc pas mettre à jour les poids
- Les erreurs ne se propageant plus, ce neurone peut affecter les autres neurones du réseau
  - *Exemple : ReLu qui renvoie 0 quand l'entrée est inférieure ou égale à 0. Si chaque exemple donne une valeur négative, le neurone ne s'active pas et après la descente de gradient le neurone devient 0 donc ne sera plus utilisé. Le Leaky Relu permet de résoudre ce problème.*

Explosion de gradient

- *Explosing gradient*
- le problème se pose lorsque des gradients d'erreur important s'accumulent et entraînent des mises à jour importantes des poids. Cela amène un réseau instable : les valeurs de mises à jour des poids peuvent être trop grandes et être remplacées par des NaN donc non utilisables
- Le problème est lié au type de descente de gradient utilisé (Batch vs mini-batch), au fait qu'il y a peut être trop de couches dans le réseau et bien sûr à certaines fonctions d'activation qui favorisent ce problème

Saturation de neurones

- *Saturated neurons*
- le problème est lié au fait que les grandes valeurs (resp. petites) atteignent un plafond et qu'elles ne changent pas lors de la propagation dans le réseau
- Principalement lié aux fonctions sigmoid et tanh. sigmoid, pour toutes les valeurs supérieures à 1 va arriver sur un plateau et retournera toujours 1. Pour cela, ces deux fonctions d'activations sont assez déconseillées en deep learning (préférer Relu ou Leaky Relu)

Connaître les propriétés

https://dashee87.github.io/deep%20learning/visualising-activation-functions-in-neural-networks/

Deux étapes

- Forward propagation
- Backward propagation

Forward propagation

Nous avons vu que : $\mathbf{z}_i^{[l]} = \mathbf{w}_i^T . \mathbf{a}^{[l-1]} + b_i \qquad \mathbf{a}_i^{[l]} = \sigma^{[l]}(\mathbf{z}_i^{[l]})$

En prenant la notation matricielle :

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} . \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]}$$

$$\mathbf{A}^{[l]} = \sigma^{[l]}(\mathbf{Z}^{[l]})$$

Nous savons que : $\mathbf{A}^{[0]} = X$

Avec Relu et sigmoid comme fonctions d'activation :

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]} . \mathbf{A}^{[0]} + \mathbf{b}^{[1]}$$
$$\mathbf{A}^{[1]} = ReLu^{[1]}(\mathbf{Z}^{[1]})$$

$$\mathbf{Z}^{[2]} = \mathbf{W}^{[2]} . \mathbf{A}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{A}^{[2]} = Sigmoid^{[2]}(\mathbf{Z}^{[2]})$$

Résultat $\hat{\mathbf{y}} = \mathbf{A}^{[2]}$

démonstration

- Voir notebook réseaux de neurones

Backward propagation

- L'objectif de la Backward propagation est tout d'abord d'évaluer la différence entre la valeur prédite et la valeur réelle : calcul du coût/perte
- Cross entropy

$$Cost(\hat{y},y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

- Propager l’erreur dans tous le réseau pour mettre à jour les différents poids : Backward propagation

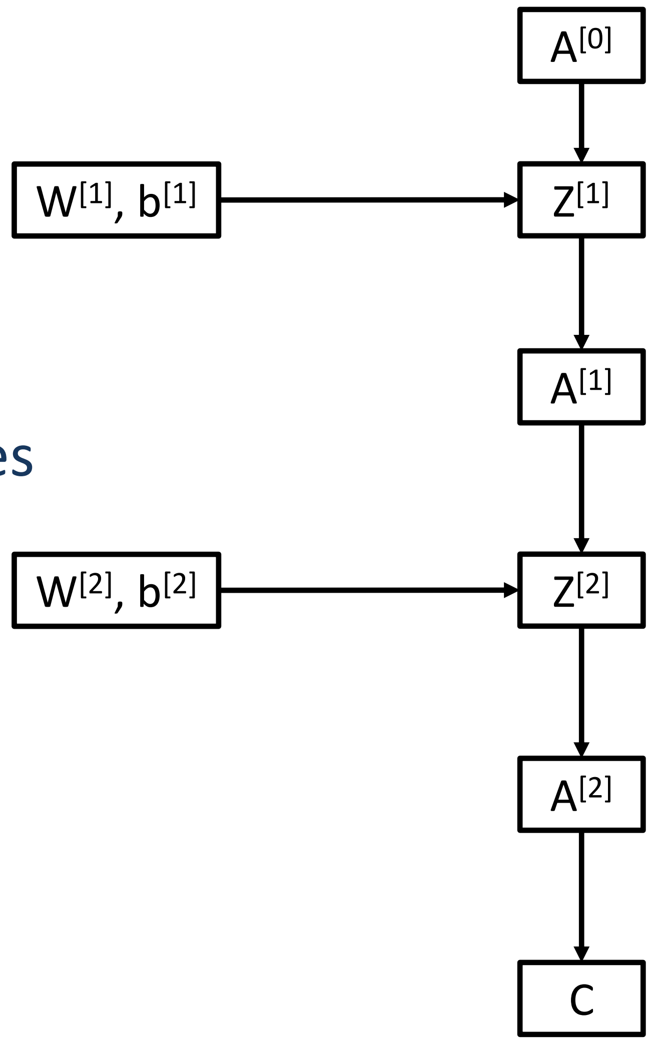
Backward propagation

Rappel Forward propagation

$$
\begin{aligned}
\mathbf{A}^{[0]} &= X \\
\mathbf{Z}^{[1]} &= \mathbf{W}^{[1]} \bullet \mathbf{A}^{[0]} + \mathbf{b}^{[1]} \\
\mathbf{A}^{[1]} &= \sigma^{[1]}(\mathbf{Z}^{[1]}) \\
\mathbf{Z}^{[2]} &= \mathbf{W}^{[2]} \bullet \mathbf{A}^{[1]} + \mathbf{b}^{[2]} \\
\mathbf{A}^{[2]} &= \sigma^{[2]}(\mathbf{Z}^{[2]}) \\
&\vdots \\
\mathbf{Z}^{[L]} &= \mathbf{W}^{[L]} \bullet \mathbf{A}^{[L-1]} + \mathbf{b}^{[L]} \\
\mathbf{A}^{[L]} &= \sigma^{[L]}(\mathbf{Z}^{[L]}) = \hat{\mathbf{y}}
\end{aligned}
$$

Backward propagation

Opérations effectuées dans le réseau



X

$Z^{[1]}=W^{[1]}A^{[0]}+b^{[1]}$

$A^{[1]}=\sigma^{[1]}(Z^{[1]})$

$Z^{[2]}=W^{[2]}A^{[1]}+b^{[2]}$

$A^{[2]}=\sigma^{[2]}(Z^{[2]})$

$C=-(y\log(A^{[2]})+(1-y)\log(1-A^{[2]})$

Backward propagation

- Objectif : reporter sur le réseau l'ensemble des modifications à apporter à partir du coût obtenu

- Repartir en sens inverse en calculant à chaque fois les dérivées du coût par rapport aux fonctions associées jusqu'au dernier niveau ($A^{[1]}$)

Backward propagation

- Chaîne de dérivation (*chain rule*) $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$

- $\mathbf{C}$ dépend de $\mathbf{A}^{[2]}$, $\mathbf{A}^{[2]}$ dépend lui même de $\mathbf{Z}^{[2]}$, $\mathbf{Z}^{[2]}$ qui dépend lui même de $\mathbf{W}^{[2]}$ et de $\mathbf{b}^{[2]}$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[2]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[2]}} \cdot \frac{\partial \mathbf{A}^{[2]}}{\partial \mathbf{Z}^{[2]}} \cdot \frac{\partial \mathbf{Z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[2]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[2]}} \cdot \frac{\partial \mathbf{A}^{[2]}}{\partial \mathbf{Z}^{[2]}} \cdot \frac{\partial \mathbf{Z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

Backward propagation

- De la même manière
- Pour avoir la dérivée partielle de $\mathbf{C}$ par rapport à $\mathbf{W}^{[1]}$ et $\mathbf{b}^{[1]}$, $\mathbf{Z}^{[2]}$ dépend de $\mathbf{A}^{[1]}$, qui elle même dépend de $\mathbf{Z}^{[1]}$ et que finalement $\mathbf{Z}^{[1]}$ dépend de $\mathbf{W}^{[1]}$ et $\mathbf{b}^{[1]}$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[2]}} \cdot \frac{\partial \mathbf{A}^{[2]}}{\partial \mathbf{Z}^{[2]}} \cdot \frac{\partial \mathbf{Z}^{[2]}}{\partial \mathbf{A}^{[1]}} \cdot \frac{\partial \mathbf{A}^{[1]}}{\partial \mathbf{Z}^{[1]}} \cdot \frac{\partial \mathbf{Z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

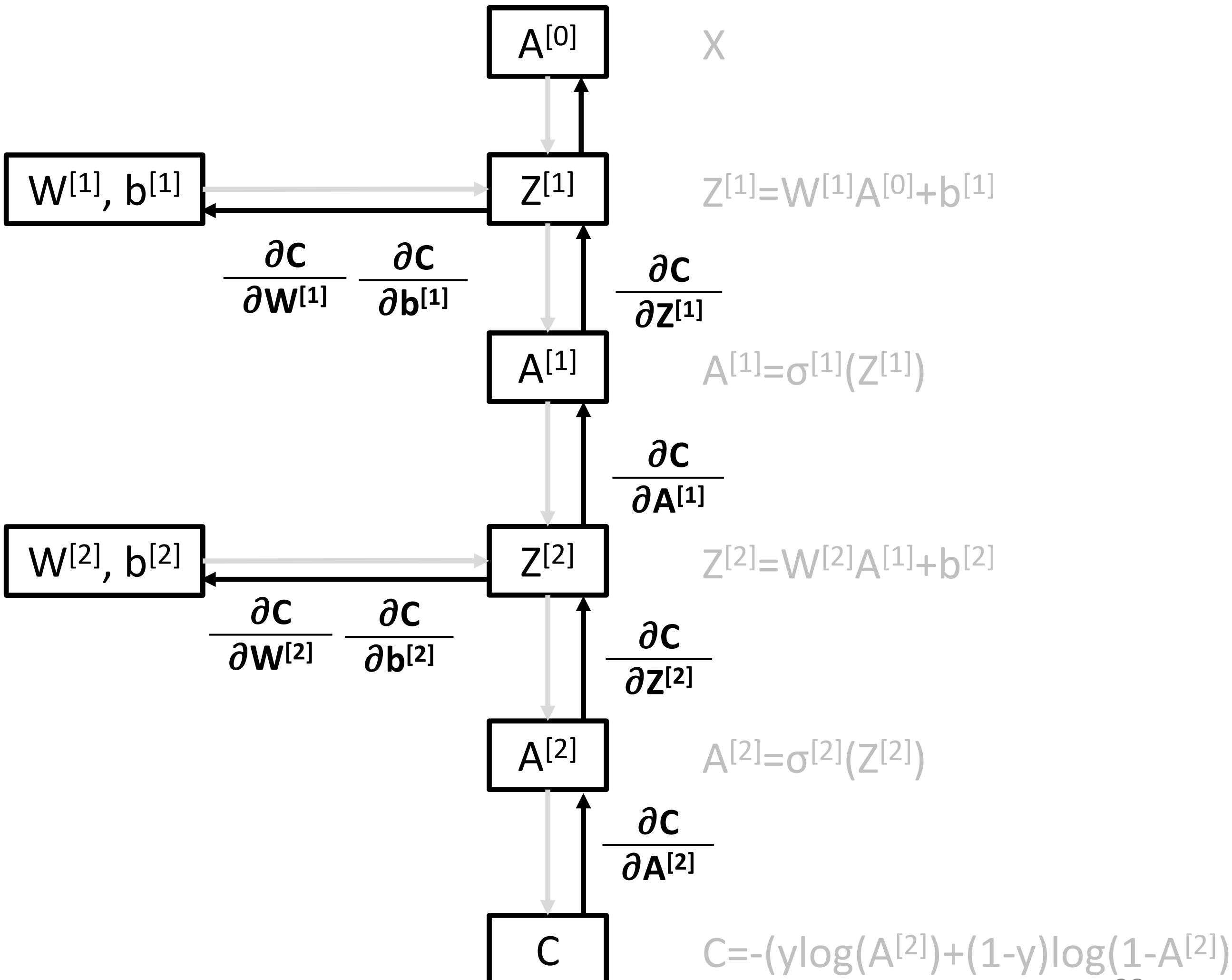
$$\frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[1]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[2]}} \cdot \frac{\partial \mathbf{A}^{[2]}}{\partial \mathbf{Z}^{[2]}} \cdot \frac{\partial \mathbf{Z}^{[2]}}{\partial \mathbf{A}^{[1]}} \cdot \frac{\partial \mathbf{A}^{[1]}}{\partial \mathbf{Z}^{[1]}} \cdot \frac{\partial \mathbf{Z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$

Backward propagation

dérivées partielles de la fonction de coût en fonction des poids et des biais d'une couche *l*

$$\frac{\partial C}{\partial W^{[l]}} = \frac{\partial C}{\partial Z^{[l]}} \cdot \frac{\partial Z^{[l]}}{\partial W^{[l]}}$$

$$\frac{\partial C}{\partial b^{[l]}} = \frac{\partial C}{\partial Z^{[l]}} \cdot \frac{\partial Z^{[l]}}{\partial b^{[l]}}$$



Backward propagation

- Pour obtenir les dérivées partielle de $\mathbf{C}$ par rapport à $\mathbf{W}^{[l]}$ et $\mathbf{b}^{[l]}$ il faut calculer les dérivées partielles :

$$\frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[L]}}, \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[L]}}, \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}} \frac{\partial \mathbf{Z}^{[l]}}{\partial \mathbf{W}^{[l]}}, \frac{\partial \mathbf{Z}^{[l]}}{\partial \mathbf{b}^{[l]}}$$

$$\frac{\partial C}{\partial A^{[L]}}$$

$$\frac{\partial C}{\partial A^{[L]}} = \frac{\partial(-y\log(A^{[L]}) - (1-y)\log(1-A^{[L]}))}{\partial A^{[L]}}$$

La dérivée de log(x) est : $\frac{\partial \log(x)}{\partial dx} = \frac{1}{x}$

Pour la partie gauche $-y\log(A^{[L]})$ nous avons : $\frac{-y}{A^{[L]}}$

Pour la partie droite $-(1-y)\log(1-A^{[L]})$ en appliquant la dérivée d'une fonction

$$\frac{\partial \log(g(x))}{\partial dx} = \frac{1}{g(x)} g'(x))$$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[\mathbf{L}]}}$$

comme la dérivée de $\mathbf{1 - A^{[L]}}$ est $\mathbf{-1}$ nous avons au final :

$$\frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[\mathbf{L}]}} = \frac{-\mathbf{y}}{\mathbf{A}^{[\mathbf{L}]}} - (-)\frac{(\mathbf{1-y})}{(\mathbf{1 - A^{[L]}})}$$

$$= \left(\frac{-\mathbf{y}}{\mathbf{A}^{[\mathbf{L}]}} + \frac{(\mathbf{1-y})}{(\mathbf{1 - A^{[L]}})}\right)$$

$$\boxed{\frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[\mathbf{L}]}} = \left(\frac{-\mathbf{y}}{\mathbf{A}^{[\mathbf{L}]}} + \frac{(\mathbf{1-y})}{(\mathbf{1-A^{[L]}})})\right)}$$

$$\frac{\partial C}{\partial Z^{[L]}}$$

$$\frac{\partial C}{\partial Z^{[L]}} = \frac{\partial C}{\partial A^{[L]}} \cdot \frac{\partial A^{[L]}}{\partial Z^{[L]}} = \frac{\partial C}{\partial A^{[L]}} * \sigma'^{[L]}(Z^{[L]})$$

$\sigma'^{[L]}(Z^{[L]})$: dérivée de la sigmoid (cf notebook descente de gradient)

$$\frac{\partial A^{[L]}}{\partial Z^{[L]}} = \mathrm{sigmoid}(Z^{[L]})(1 - \mathrm{sigmoid}(Z^{[L]})) = A^{[L]}(1 - A^{[L]})$$

$$\frac{\partial C}{\partial A^{[L]}} \cdot \frac{\partial A^{[L]}}{\partial Z^{[L]}} = \left(\frac{-y}{A^{[L]}} + \frac{(1-y)}{(1-A^{[L]})}\right) A^{[L]}(1 - A^{[L]})$$

$$\frac{\partial C}{\partial Z^{[L]}}$$

En multiplier par $(1 - A^{[L]})$ et $(A^{[L]})$ pour simplifier :

$$= \left(\frac{-y(1 - A^{[L]})}{A^{[L]}(1 - A^{[L]})} + \frac{A^{[L]}(1 - y)}{A^{[L]}(1 - A^{[L]})} \right) A^{[L]}(1 - A^{[L]})$$

$$= \left(\frac{-y(1 - A^{[L]}) + A^{[L]}(1 - y)}{A^{[L]}(1 - A^{[L]})} \right) A^{[L]}(1 - A^{[L]})$$

En supprimant $A^{[L]}(1 - A^{[L]})$

$$= \left(-y(1 - A^{[L]}) + A^{[L]}(1 - y)\right)$$

$$= -y + yA^{[L]} + A^{[L]} - A^{[L]}y$$

$$= -y + A^{[L]}$$

$$\boxed{\frac{\partial C}{\partial Z^{[L]}} = A^{[L]} - y}$$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}}$$

Pour un niveau $\mathbf{l}$ dérivée partielle de $\mathbf{C}$ par rapport à $\mathbf{Z}^{[l]}$

Si on connaît $\mathbf{Z}^{[l]}$ on peut calculer $\mathbf{Z}^{[L-1]}$, $\mathbf{Z}^{[L-2]}$, etc

$$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l+1]}} \bullet \frac{\partial \mathbf{Z}^{[l+1]}}{\partial \mathbf{A}^{[l]}} \bullet \frac{\partial \mathbf{A}^{[l]}}{\partial \mathbf{Z}^{[l]}}$$

$$\mathbf{Z}^{[l+1]} = \mathbf{W}^{[l+1]} \bullet \mathbf{A}[l] + \mathbf{b}[l+1]$$

$$\frac{\partial \mathbf{A}^{[l]}}{\partial \mathbf{Z}^{[l]}} = \sigma'^{[l]}(\mathbf{Z}^{[l]})$$

$$\frac{\partial \mathbf{Z}^{[l+1]}}{\partial \mathbf{A}^{[l]}} = \frac{\partial(\mathbf{W}^{[l+1]} \bullet \mathbf{A}[l] + \mathbf{b}[l+1])}{\partial \mathbf{A}^{[l]}}$$

$$= \mathbf{W}^{[l+1]}$$

$$\boxed{\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}} = ({\mathbf{W}^{[l+1]}}^{T} \bullet \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l+1]}}) * \sigma'^{[l]}(\mathbf{Z}^{[l]})}$$

$\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[l]}}$

Dérivée partielle de $\mathbf{Z}^{[l]}$ par rapport à $\mathbf{W}^{[l]}$

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \bullet \mathbf{A}[l-1] + \mathbf{b}[l]$$

$$\frac{\partial \mathbf{Z}^{[l]}}{\partial \mathbf{W}^{[l]}} = \frac{\partial(\mathbf{W}^{[l]} \bullet \mathbf{A}[l-1] + \mathbf{b}[l])}{\partial \mathbf{W}^{[l]}}$$

$$= \mathbf{A}^{[l-1]}$$

Dérivée partielle de $\mathbf{C}$ par rapport à $\mathbf{W}^{[l]}$

$$\boxed{\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[l]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}} \bullet \mathbf{A}^{[l-1]^{\mathrm{T}}}}$$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[l]}}$$

Dérivée partielle de $\mathbf{Z}^{[l]}$ par rapport à $\mathbf{b}^{[l]}$

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \bullet \mathbf{A[l-1] + b[l]}$$

$$\frac{\partial \mathbf{Z}^{[l]}}{\partial \mathbf{b}^{[l]}} = \frac{\partial(\mathbf{W}^{[l]} \bullet \mathbf{A[l-1] + b[l]})}{\partial \mathbf{b}^{[l]}}$$

$$= \mathbf{1}$$

Dérivée partielle de $\mathbf{C}$ par rapport à $\mathbf{b}^{[l]}$

$$\boxed{\frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[l]}} = \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}}}$$

Pour résumer

Pour le layer L
(dernier niveau)

$\frac{\partial \mathbf{C}}{\partial \mathbf{A}^{[L]}}$	$\left(\frac{-\mathbf{y}}{\mathbf{A}^{[L]}} + \frac{(1-\mathbf{y})}{(1-\mathbf{A}^{[L]})}\right)$
$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[L]}}$	$(\mathbf{A}^{[L]} - \mathbf{y})$
$\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[L]}}$	$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[L]}} \bullet (\mathbf{A}^{[L-1]^T})$
$\frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[L]}}$	$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[L]}}$

Pour un layer l

$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}}$	$(\mathbf{W}^{[l+1]^T} \bullet \frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l+1]}}) * \sigma'^{[l]}(\mathbf{Z}^{[l]})$
$\frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[l]}}$	$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}} \bullet \mathbf{A}^{[l-1]^T}$
$\frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[l]}}$	$\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[l]}}$

La descente de gradient

Il suffit d'utiliser les dérivées calculées précédemment et de reporter les modifications

Pour l du dernier layer au layer 1 {

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \eta \frac{\partial \mathbf{C}}{\partial \mathbf{W}^{[l]}}$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \eta \frac{\partial \mathbf{C}}{\partial \mathbf{b}^{[l]}}$$

}

Demo

- Notebook réseau de neurones (fonctions, classification binaire)

Classification multi-classes

- Jusqu’à présent : classification binaire
- Pour faire de la classification multi-classes, fonction d’activation : softmax

- Attribution des probabilités à chaque classe d'un problème à plusieurs classes et la somme de ces probabilités doit être égale à 1

Softmax

Formellement :

Entrée : vecteur **z** de C-dimensions (le nombre de classes possibles)

Sortie : vecteur **a** de C-dimensions de valeurs réelles comprises entre 0 et 1

$$\mathbf{a_i} = \frac{\mathbf{e^{z_i}}}{\sum_{\mathbf{k=1}}^{\mathbf{C}} \mathbf{e^{z_k}}}$$

Pour i = 1 … C

$$avec \sum_{i=1}^{C} = 1$$

où *C* est le nombre de classes

Softmax

Fonction instable

```
nums = np.array([4000, 5000, 6000])
print(softmax(nums))
```

```
[nan nan nan]
/Users/pascalponcelet/Desktop/Sicki-learn/Tools/tools/lib/python3.6/site-packages/ipykernel_launcher.py:2: RuntimeWarning: overflow encountered in exp
```

Multiplication par une constante au numérateur et au dénominateur

$$a_i = \frac{e^{z_i - \max(z)}}{\sum_{k=1}^{C} e^{z_k - \max(z)}}$$

```
def softmax(z):
    expz = np.exp(z - np.max(z))
    return expz / expz.sum(axis=0, keepdims=True)

nums = np.array([4, 5, 6])
print(softmax(nums))
print ("la somme des probabilités donne 1")
nums = np.array([4000, 5000, 6000])
print(softmax(nums))
```

```
[0.09003057 0.24472847 0.66524096]
la somme des probabilités donne 1
[0. 0. 1.]
```

Dérivée de softmax

Considérer que $\mathbf{g(x) = e^{z_i}}$ $\quad\quad$ $\mathbf{h(x) = \sum_{k=1}^{C} e^{z_k}}$

La dérivée d'une fonction $\mathbf{f(x) = \frac{g(x)}{h(x)}}$ est

$$\mathbf{f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h(x)^2}}$$

Simplification de notation $\mathbf{\sum_C = \sum_{k=1}^{C} e^{z_k}}$

Pour i = 1..C nous avons $\mathbf{a_i = \frac{e^{z_i}}{\sum_C}}$

Dérivée de softmax

La dérivée $\frac{\partial \mathbf{a_i}}{\partial \mathbf{z_j}}$ de la sortie de softmax **a** par rapport à **z** :

Si i=j

$$\frac{\partial \mathbf{a_i}}{\partial \mathbf{z_i}} = \frac{\partial(\frac{\mathbf{e^{z_i}}}{\sum_C})}{\partial \mathbf{z_i}} = \frac{\mathbf{e^{z_i}} \sum_C - \mathbf{e^{z_i}} \mathbf{e^{z_i}}}{\sum_C^2} = \frac{\mathbf{e^{z_i}}}{\sum_C} \frac{\sum_C - \mathbf{e^{z_i}}}{\sum_C} = \frac{\mathbf{e^{z_i}}}{\sum_C}(\mathbf{1} - \frac{\mathbf{e^{z_i}}}{\sum_C}) = \mathbf{a_i}(\mathbf{1} - \mathbf{a_i})$$

Si i≠ j

$$\frac{\partial \mathbf{a_i}}{\partial \mathbf{z_j}} = \frac{\partial(\frac{\mathbf{e^{z_i}}}{\sum_C})}{\partial \mathbf{z_j}} = \frac{\mathbf{0} - \mathbf{e^{z_i}} \mathbf{e^{z_j}}}{\sum_C^2} = \frac{\mathbf{e^{z_i}}}{\sum_C} \frac{\mathbf{e^{z_j}}}{\sum_C} = -\mathbf{a_i} \mathbf{a_j}$$

Dérivée de softmax

Softmax avec la cross entropy (même principe que précédemment)

$$\boxed{\frac{\partial \mathbf{C}}{\partial \mathbf{Z}^{[\mathbf{L}]}} = \mathbf{A}^{[\mathbf{L}]} - \mathbf{y}}$$

Toutes les dérivées précédentes sont donc similaires

Demo

- Notebook réseau de neurones (classification multi-classes)

Une petite mise en œuvre concrète

- Importation de bibliothèque

```
# TensorFlow et keras
import tensorflow as tf
import keras
from keras import layers
from keras import models
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
```

Une petite mise en œuvre concrète

- Création du modèle

Première couche avec 25 neurones

2 variables prédictives

```
model = Sequential()
model.add(Dense(25, input_dim=2,activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

3 couches

Fonction d'activation : relu

La couche de sortie

C'est une classification binaire : 1 seul neurone et sigmoid

Une petite mise en œuvre concrète

- Un petit jeu de données

```
X,y = make_moons(n_samples=1000, noise=0.1)
validation_size=0.6 #40% du jeu de données pour le test
testsize= 1-validation_size
seed=30

# ATTENTION C'EST FAUX !!!! Pourquoi ?
X_train,X_test,y_train,y_test=train_test_split(X,y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)
```

Une petite mise en œuvre concrète

- Compilation du modèle

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

Une petite mise en œuvre concrète

- Entrainement du modèle

```
history = model.fit(X_train_init, y_train_init,
                    epochs=600, verbose=1)
```

Une petite mise en œuvre concrète

- Prédiction

```
y_test_hat=model.predict(X_test_init)
y_test_hat[y_test_hat <= 0.5] = 0.
y_test_hat[y_test_hat > 0.5] = 1.
accuracy_test = accuracy_score(y_test_init,
                               y_test_hat)
```

Un peu plus propre

```
# Là c'est juste !!! Pourquoi ?
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=validation_size,random_state=seed,test_size=testsize)

nb_folds=2
epochs=8
batch_size=64
```

Mini-batch gradient descent

```
 kfold = KFold(n_splits=nb_folds, shuffle=True)


# Cross-validation
fold_no = 1
for train, test in kfold.split(X_train, y_train):
	# Définition de l'architecture du modèle à chaque passage
.....
```

Un peu plus propre

```
# Cross-validation
fold_no = 1
for train, test in kfold.split(X_train, y_train):
        # Definition de l'architecture du modèle à chaque passage
        model = Sequential()
        model.add(Dense(25, input_dim=2,activation='relu'))
        model.add(Dense(25, activation='relu'))
        model.add(Dense(25, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

        # Compilation du modèle
        model.compile(loss='binary_crossentropy', optimizer="adam", metrics= ['accuracy'])


        print('\nEntrainement pour le fold ', fold_no, ' ')
         # Fit data sur les données
         history = model.fit(X_train[train], y_train[train], validation_data=(X_train[test], y_train[test])
                    batch_size=batch_size,
                    epochs=epochs)
```

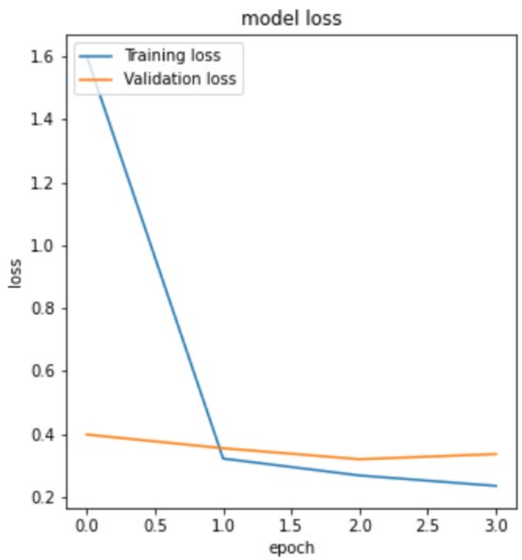
Remarque : ici on utilise bien X_train[test]

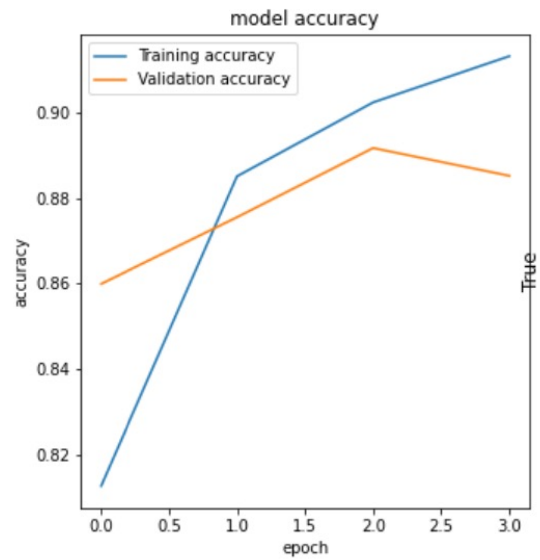
Les batchs

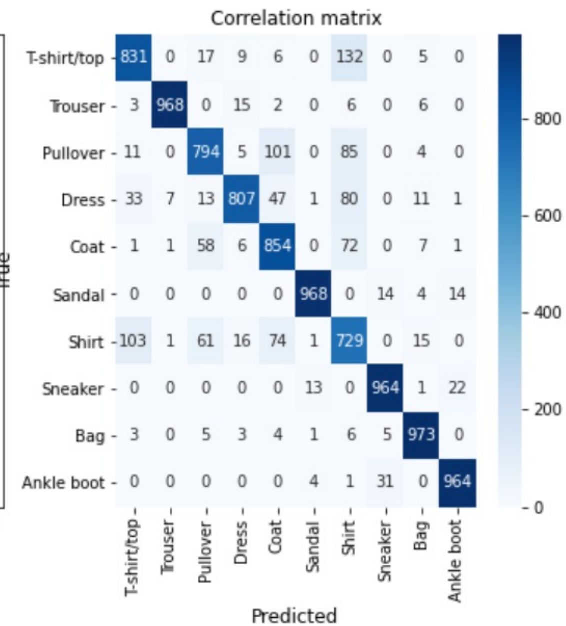
Un jeu de données de validation : à chaque epoch,
le modèle fait seulement un predict avec

Exemple d’utilisation de l’historique

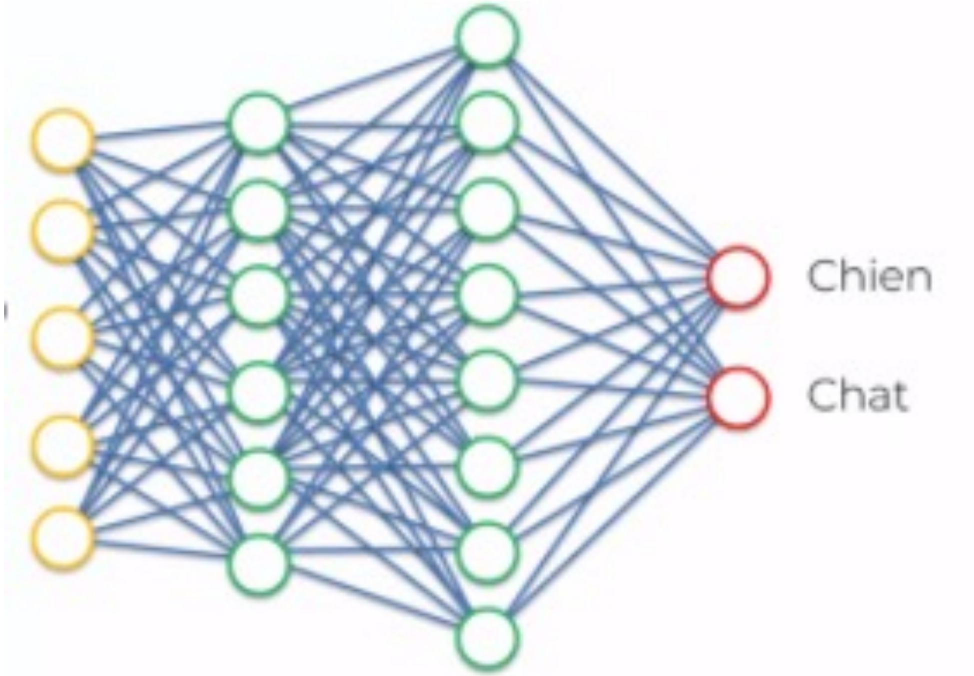
`Accuracy sur le jeu de test 0.8852`





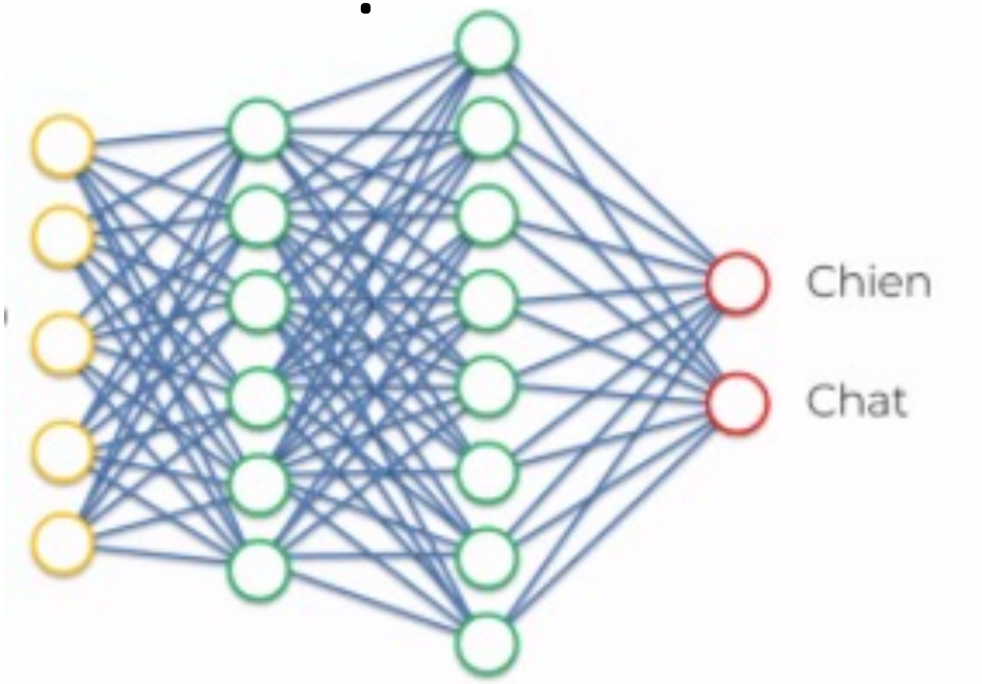


Que se passe-t'il dedans ?



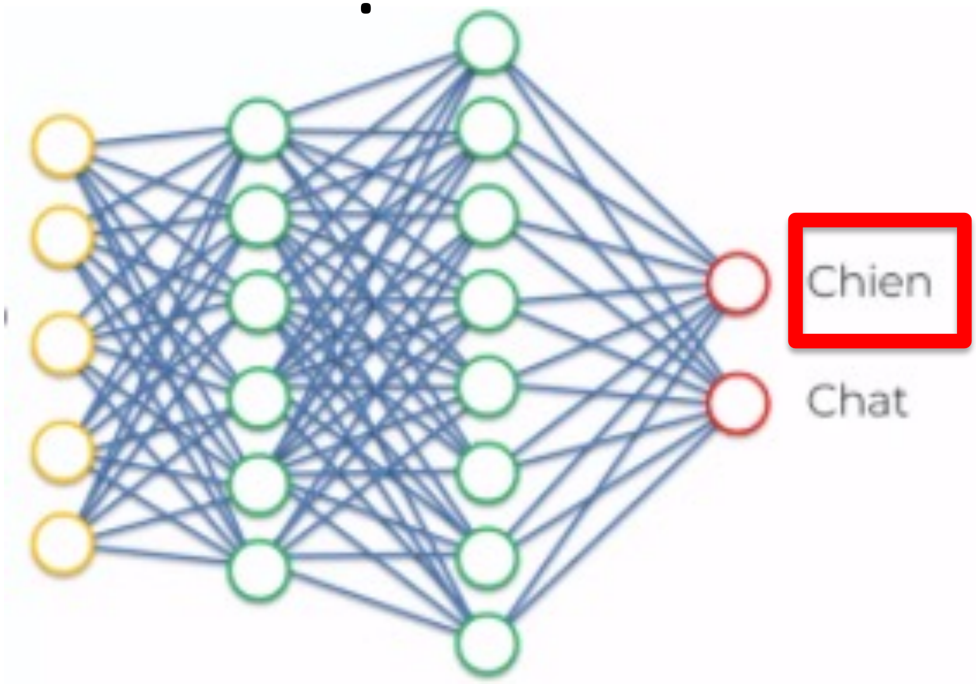
Que se passe-t'il dedans ?

?

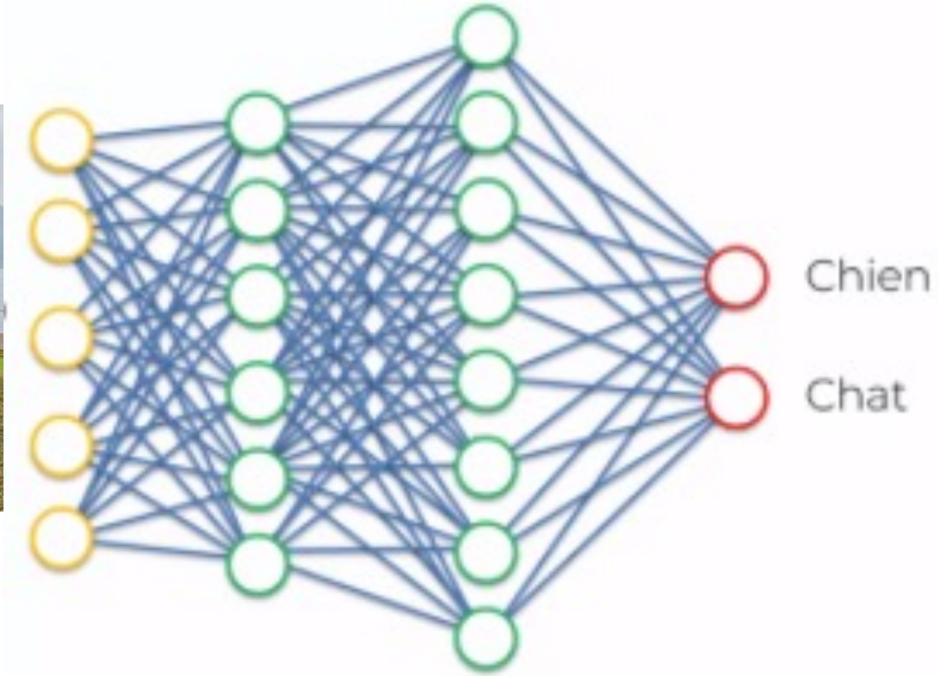


Que se passe-t'il dedans ?

?



Que se passe-t'il dedans ?



?

Rechercher des patterns

- Un pattern = une signature = chemin suivi par un objet dans le réseau

- Principe :
  – Apprendre le réseau
  – Récupérer pour chaque layer les fonctions d'activations
  – Regrouper, par layer, celles qui ont même valeur (clustering)
  – Afficher les résultats, appliquer un algorithme de recherche de patterns (séquences, trajectoires, etc.)

Demo

- http://www.lirmm.fr/~poncelet/RN

- Des questions ?