Intelligence Artificielle (IA)

Deep Learning

Akka Zemmari

LaBRI, Université de Bordeaux

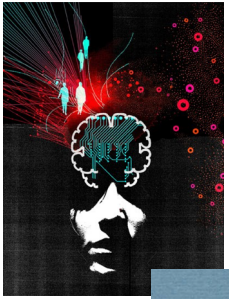
2024 - 2025

Contenu

Outline

Deep Learning

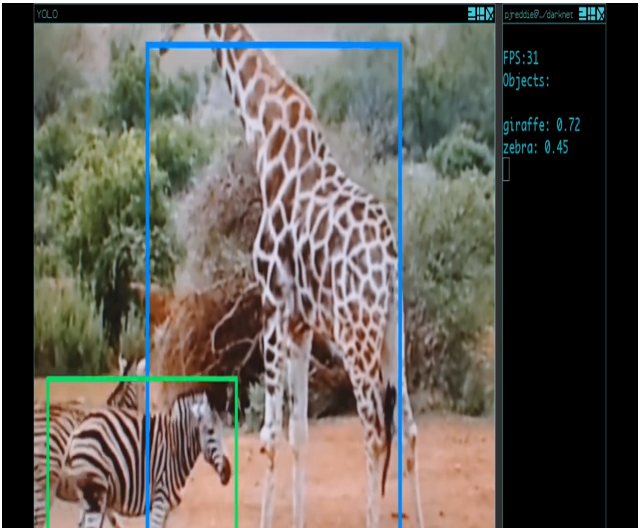
[Deep Dream Generator / Le Monde 2017]



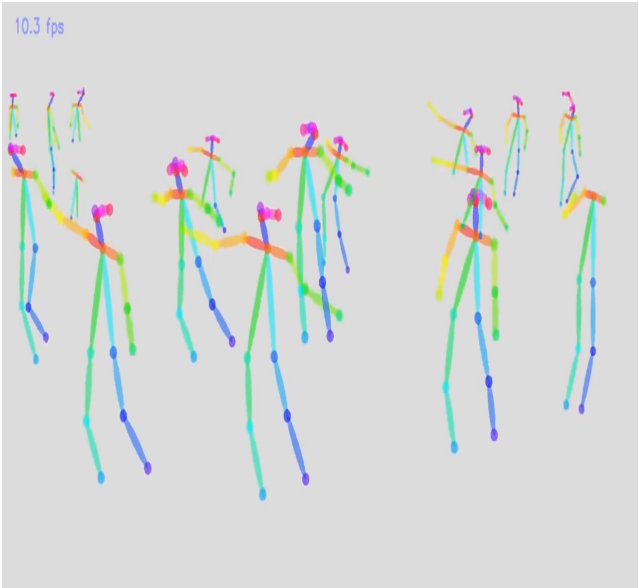
[MIT review technology, 2017]

[New Economist, 2015]

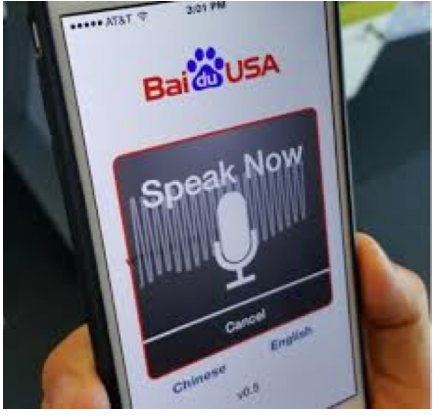
Classification+Localization



RefineNet [Lin et al, 2016] (on Cityscapes)

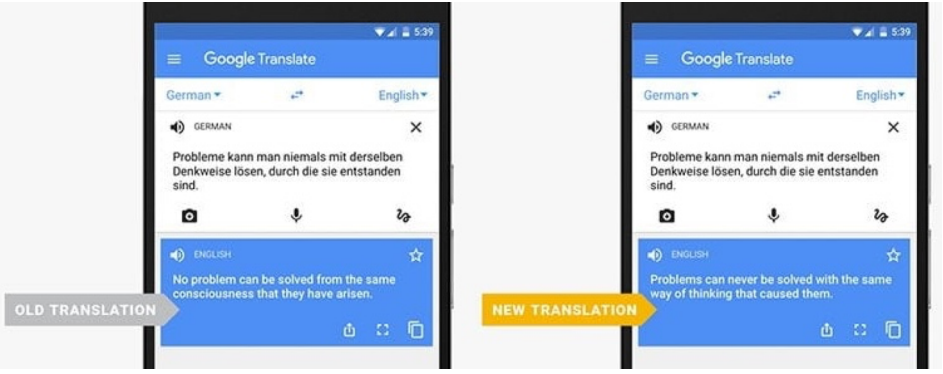

10.3 fps

Deep Speech

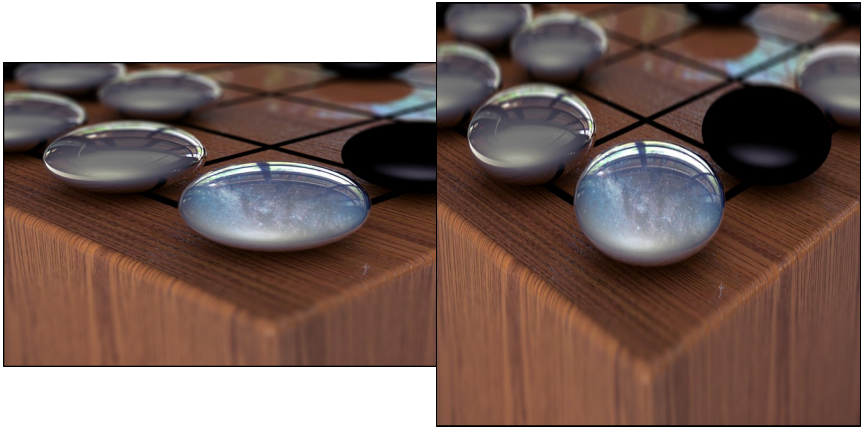


Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin

Machine Translation



Alpha Go



Et quelques intérrogations ...

Microsoft débranche son robot intelligent devenu raciste et pro-nazis en 24 heures

25 mars 2016, 10:58

Et quelques intérrogations ...

Et quelques intérrogations ...

http://moralmachine.mit.edu

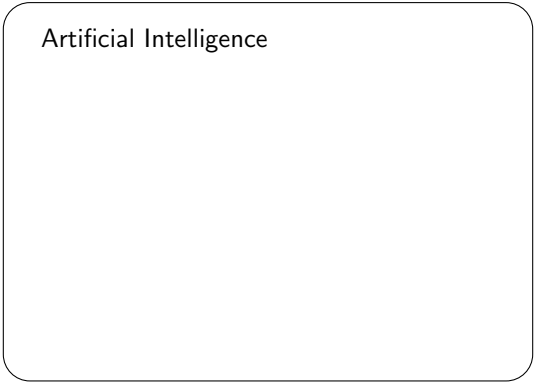
Quelques dates et quelques noms

- 1943 : (Mc Culloch (neuro-physiologiste) et Pitts (logicien)) proposent les premières notions de neurone formel.
- 1959 : (Rosenblatt) définit un réseau de neurones avec une couche d'entrée et une sortie.
- 1960 : (Widrow et Hoff) ADALINE
- 1969 : (Minsky, Papert) Problème XOR
- 1986 : (Rumelhart et. al) MLP et backpropagation
- 1992 : (Vapnik et. al) SVM
- 1998 : (LeCun et. al) LeNet
- 2010 : (Hinton et. al) Deep Neural Networks
- 2012 : (Krizhevsky, Hinton et. al) AlexNet, ILSVRC'2012, GPU - 8 couches 2014 GoogleNet - 22 couches
- 2015 : Inception (Google) - Deep Dream
- 2016 : ResidualNet (Microsoft/Facebook) - 152 couches

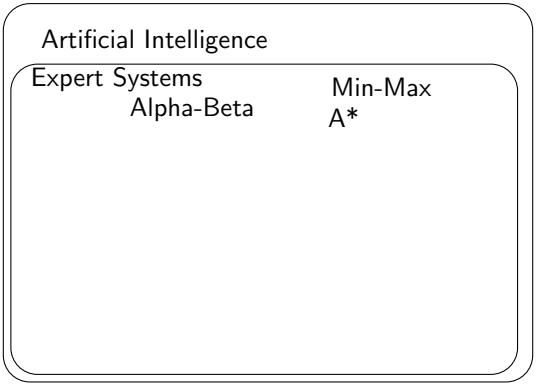
The New York Times

Turing Award Won by Three Pioneers in Artificial Intelligence

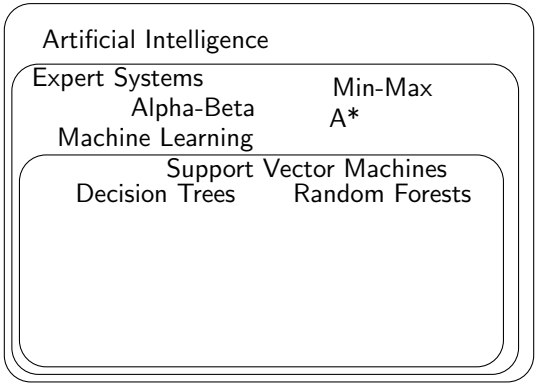
Artificial Intelligence/Machine Learning/Deep Learning



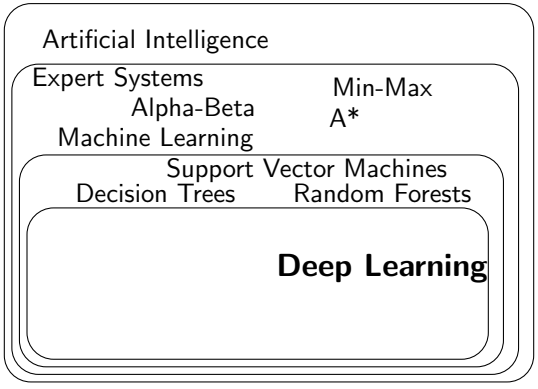
Artificial Intelligence/Machine Learning/Deep Learning



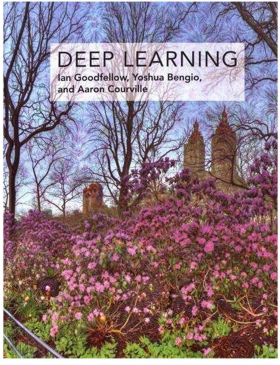
Artificial Intelligence/Machine Learning/Deep Learning



Artificial Intelligence/Machine Learning/Deep Learning



Références du cours - Livres



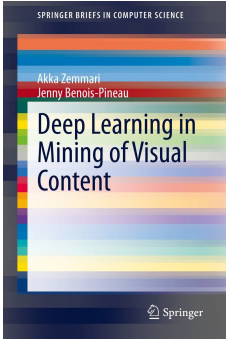
THE BOOK

■ **Deep Learning** ■,

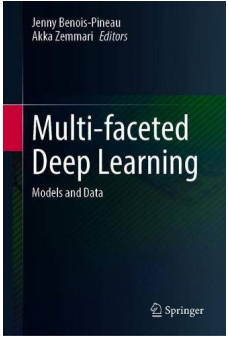
I. Goodfellow, Y. Bengio, A. Courville,

Références du cours - Livres

- **Deep Learning in Mining of Visual Content**.
  A. Zemmari & J. Benois-Pineau.
  Springer.

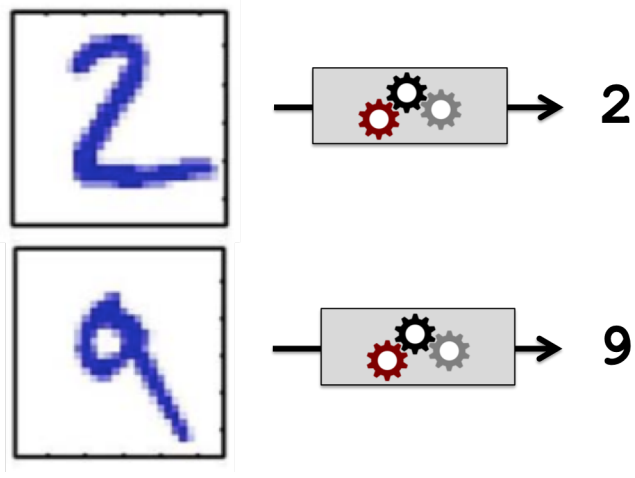


- **Multi-faceted Deep Learning : Models and Data**.
  J. Benois-Pineau & A. Zemmari (Editors). Springer.



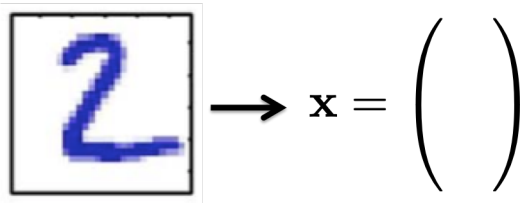
Outline

Hand-written Digit Recognition

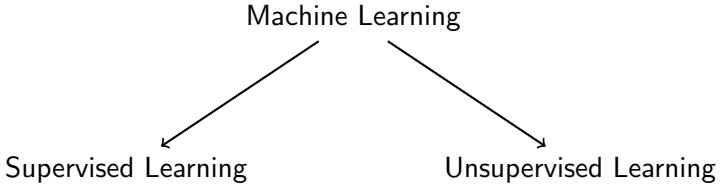


Hand-written Digit Recognition

Formalisation



- Image $= 28 \times 28$ pixels
- $f : x \in \mathbb{R}^{784} \rightarrow \{0, 1, 2, \cdots, 9\}$


Machine Learning
Supervised Learning
Unsupervised Learning

Machine Learning

Supervised Learning

Unsupervised Learning

{( , face), ( , face), ( , face), … ,

( , non-face), ( , non-face), ( , non-face),

... }

 → face or non-face?

Machine Learning

Supervised Learning

Unsupervised Learning

Machine Learning

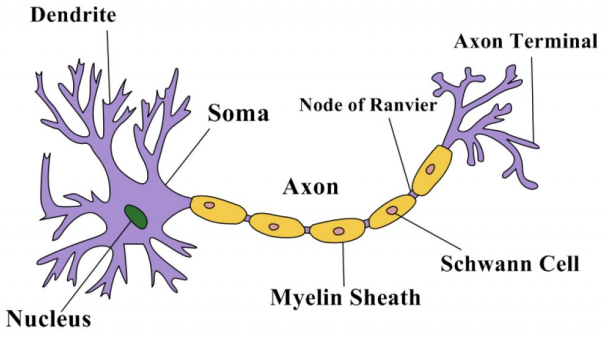
Supervised Learning

Unsupervised Learning

Outline

D'abords il y a le neurone

Neurone biologique



D'abords il y a le neurone

Neurone formel (Mc Culloch and Pitts(1943))

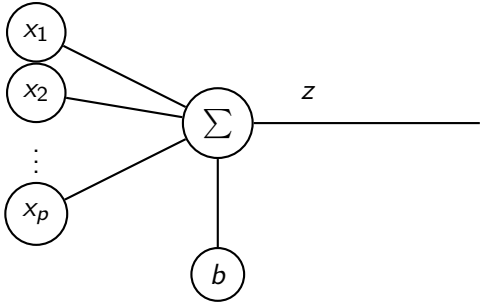


Figure – Un neurone formel.

D'abords il y a le neurone

Neurone formel (Mc Culloch and Pitts(1943)

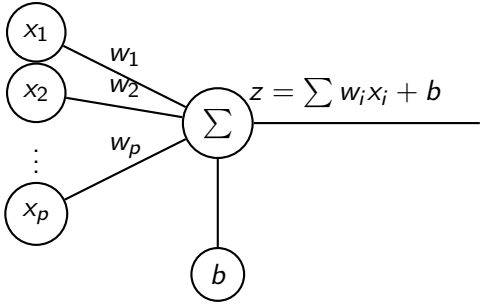


Figure – Un neurone formel.

DEMO

D'abords il y a le neurone

Neurone formel (Mc Culloch and Pitts(1943)

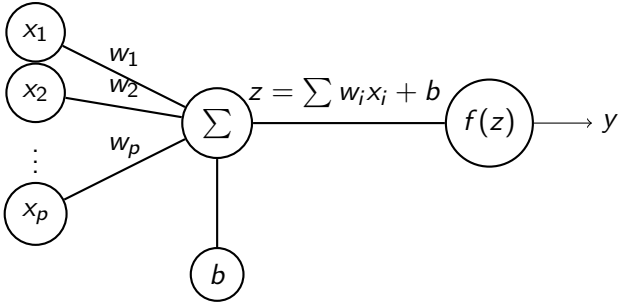


Figure – Un neurone formel.

D'abords il y a le neurone

Neurone formel (Mc Culloch and Pitts(1943)

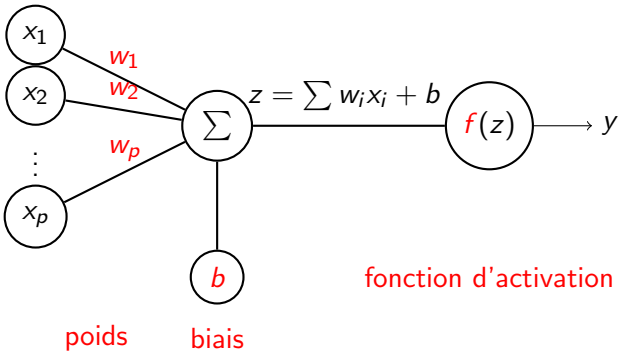
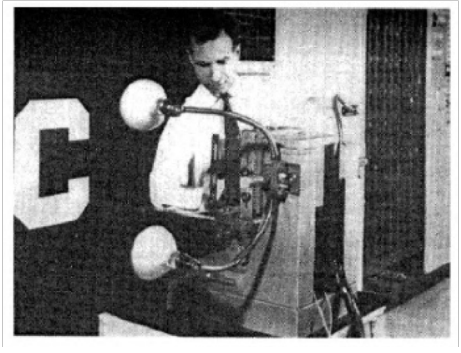
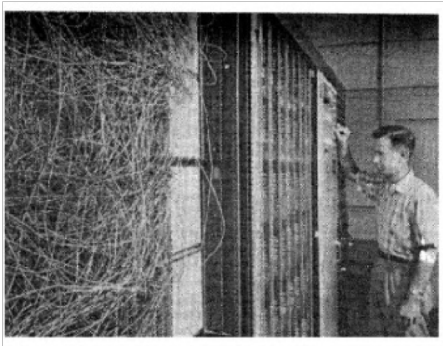


Figure – Un neurone formel.

Perceptron (1958, Franck Rosenblatt)





Perceptron

$$\text{Input} = \text{des vecteurs } x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}$$

But du Perceptron : classification binaire supervisée

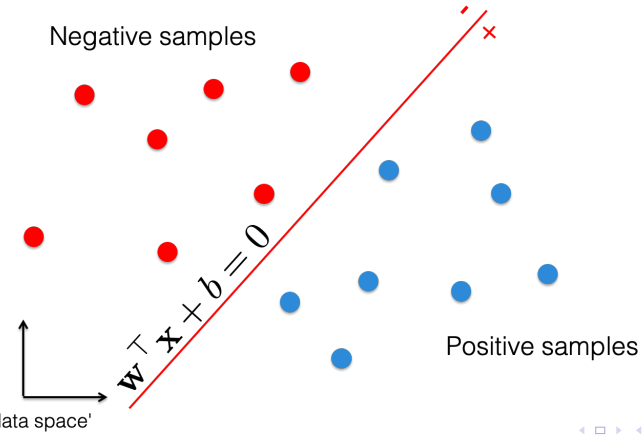
- "Classification" : prédire la classe (ou la catégorie) d'un vecteur x fourni en entrée
  Exemple, on veut savoir si x est une image de chat, de chien, de cheval, ...
- "Classification binaire" : on se limite à 2 classes.
  Exemple : chat (classe positive) et non chat (classe negative).

$\Rightarrow$ On cherche une fonction $f(x)$ de la forme :

$$f : \mathbb{R}^p \to \{+1, -1\}$$

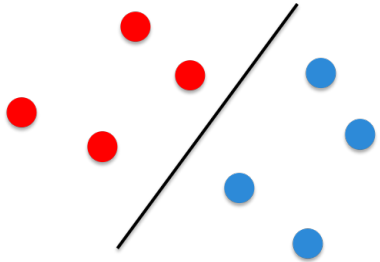
Perceptron

Interpretation géométrique

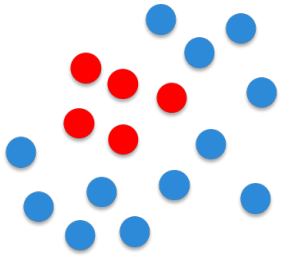


Perceptrons book (Minsky and Papert, 1969)

A perceptron can only correctly classify data points that are linearly separable :



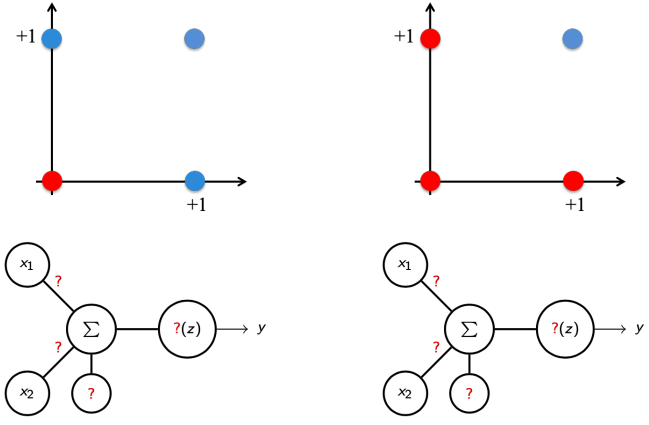
linearly separable



nonlinearly separable

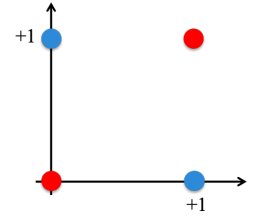
Perceptrons book (Minsky and Papert, 1969)

A perceptron can correctly classify data points that are linearly separable :



Perceptrons book (Minsky and Papert, 1969)

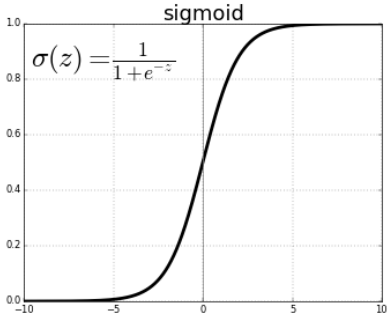
A perceptron can only correctly classify data points that are linearly separable :



D'abords il y a le neurone

Exemple de fonction d'activation : sigmoïd

$$f(z) = \frac{1}{1 + e^{-z}}.$$



DEMO

D'abords il y a le neurone

Autres fonctions d'activation :

- fonction linéaire : f est la fonction identité,
- seuil : $f(z) = \mathbb{1}_{[a,\infty[}(z)$,
- ReLU : $f(z) = \max(0, z)$ (rectified linear unit),
- radiale : $f(z) = \sqrt{1/2\pi}e^{(-z^2/2)}$,
- ...

D'abords il y a le neurone

De manière compacte :

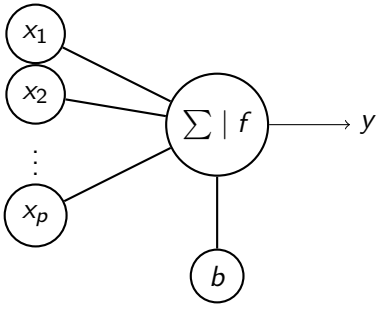


Figure – Un neurone formel.

Outline

Réseau de neurones

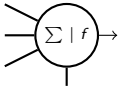


Figure – Un réseau de neurones.

Réseau de neurones

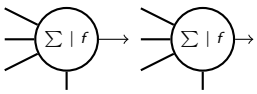


Figure – Un réseau de neurones.

Réseau de neurones

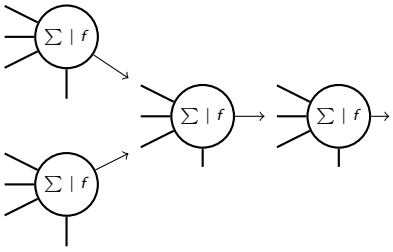


Figure – Un réseau de neurones.

Réseau de neurones

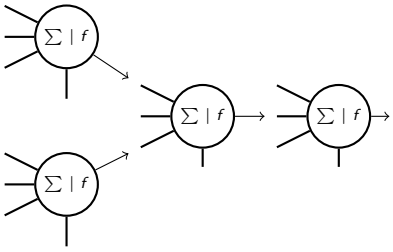


Figure – Un réseau de neurones.

- Les neurones ont chacun leurs propres poids et biais,

Réseau de neurones

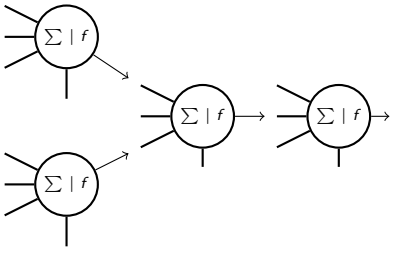
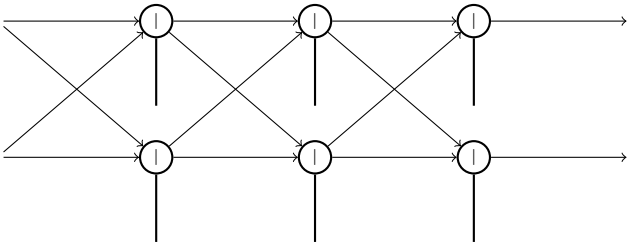


Figure – Un réseau de neurones.

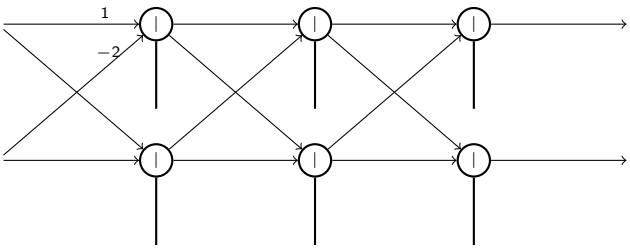
- Les neurones ont chacun leurs propres poids et biais,
- Les paramètres du réseau sont alors les différents poids et les différents biais. On note tous ces paramètres θ.

DEMO

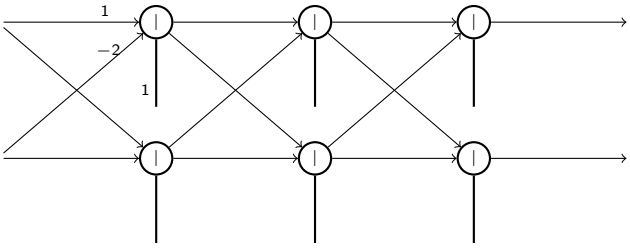
Réseaux de neurones complètement connectés



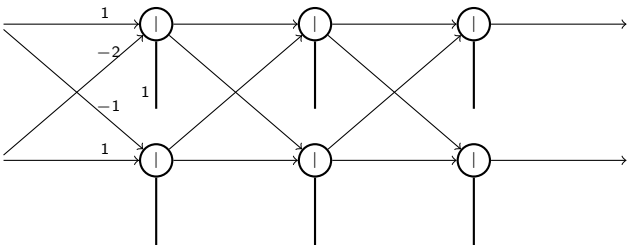
Réseaux de neurones complètement connectés



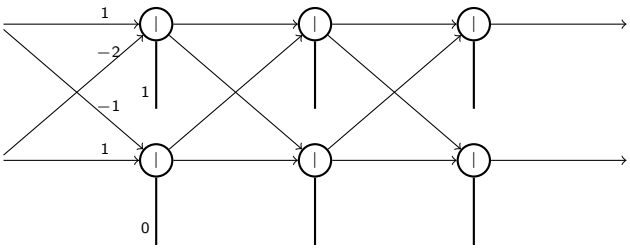
Réseaux de neurones complètement connectés



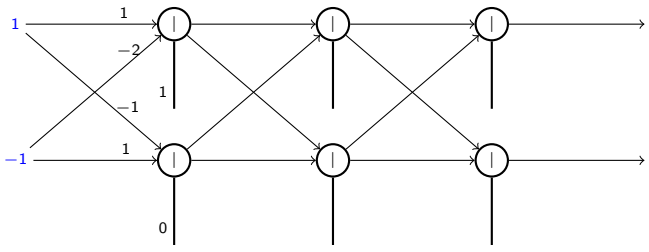
Réseaux de neurones complètement connectés



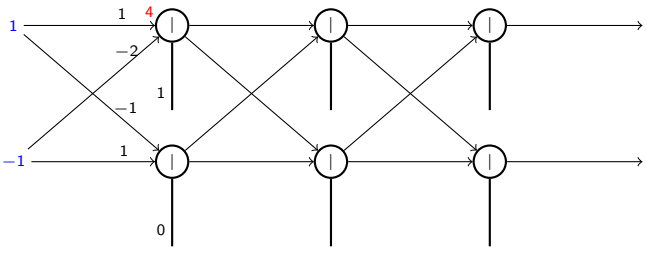
Réseaux de neurones complètement connectés



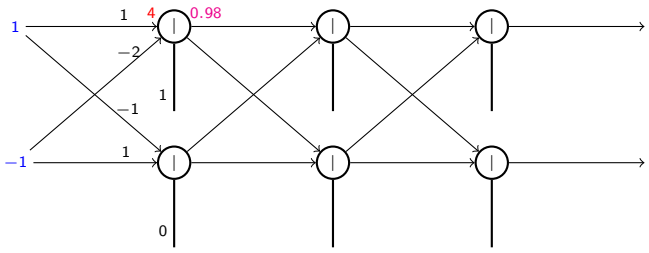
Réseaux de neurones complètement connectés



Réseaux de neurones complètement connectés

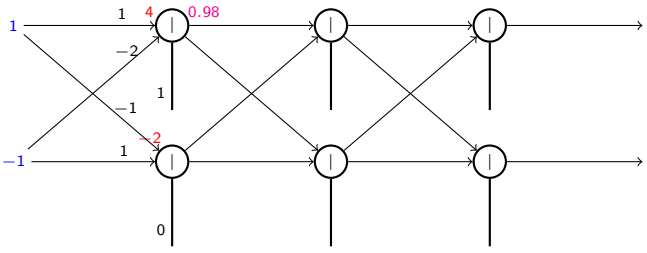


Réseaux de neurones complètement connectés

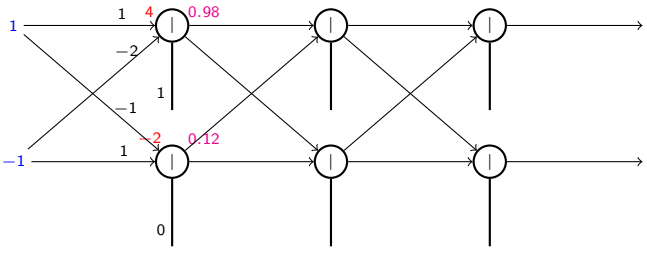


Si f est la fonction sigmoïd : $f(z) = \frac{1}{1+e^{-z}}$

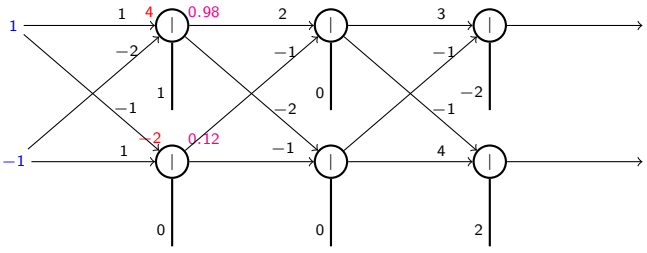
Réseaux de neurones complètement connectés



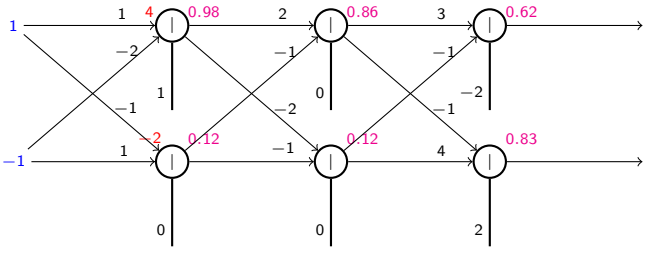
Réseaux de neurones complètement connectés



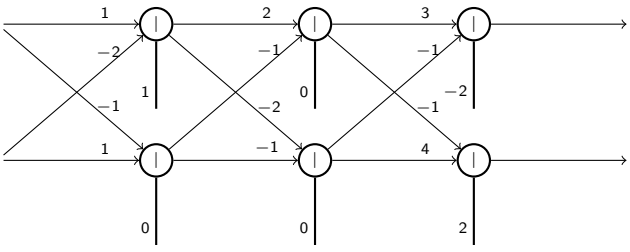
Réseaux de neurones complètement connectés



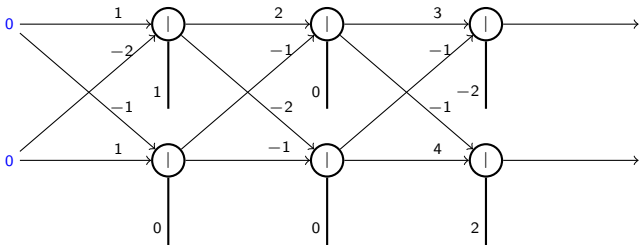
Réseaux de neurones complètement connectés



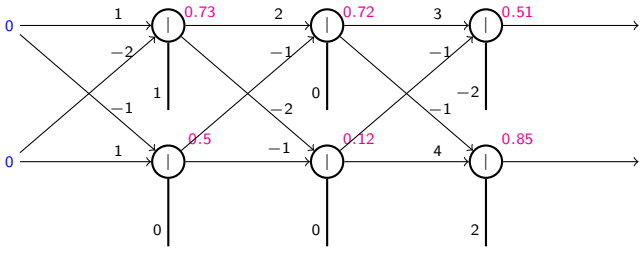
Réseaux de neurones complètement connectés



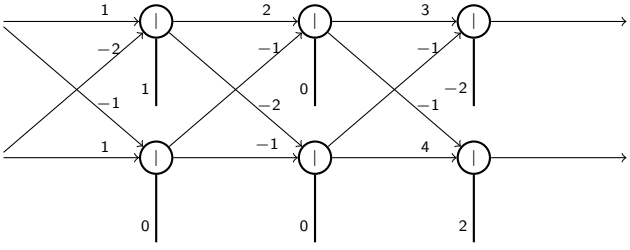
Réseaux de neurones complètement connectés



Réseaux de neurones complètement connectés

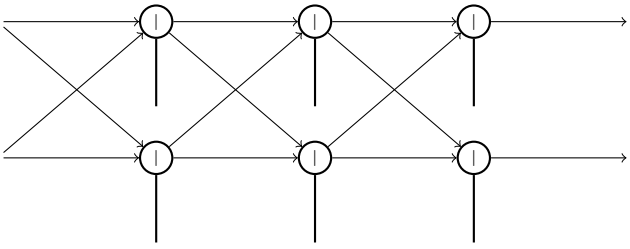


Réseaux de neurones complètement connectés



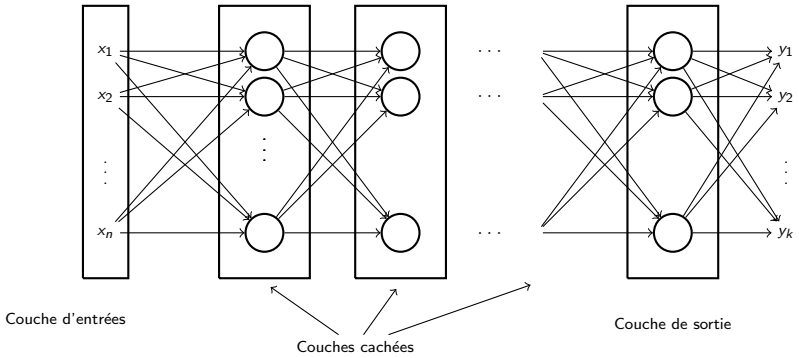
Fixer les paramètres $\theta \longrightarrow$ fixer la fonction.

Réseaux de neurones complètement connectés



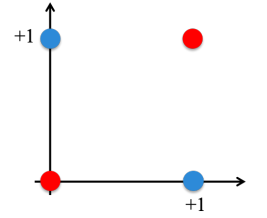
Donner la structure du réseau $\rightarrow$ définir un ensemble de fonctions.

Réseaux à plusieurs couches



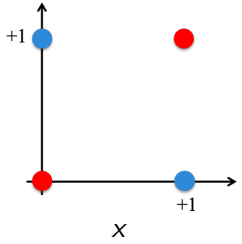
Perceptrons book (Minsky and Papert, 1969)

A perceptron can only correctly classify data points that are linearly separable :

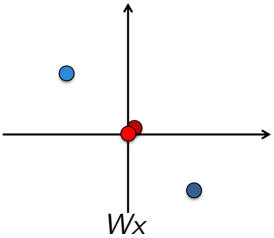


Multilayer Perceptron/Neural Network (1980)

Observons les transformations suivantes :

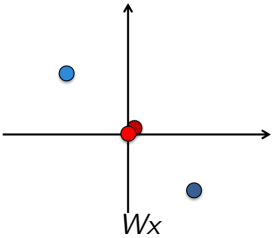


$$\rightarrow W = \begin{pmatrix} +\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & +\frac{1}{2} \end{pmatrix} \rightarrow$$

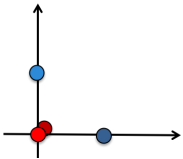


Multilayer Perceptron/Neural Network (1980)

Observons les transformations suivantes :



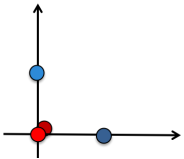
$\rightarrow g(u) = \max\{0, x\} \rightarrow$



$h(x) = g(Wx)$

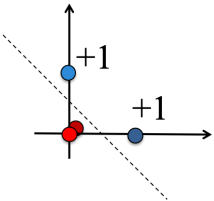
Multilayer Perceptron/Neural Network (1980)

Observons les transformations suivantes :



$h(x) = g(Wx)$

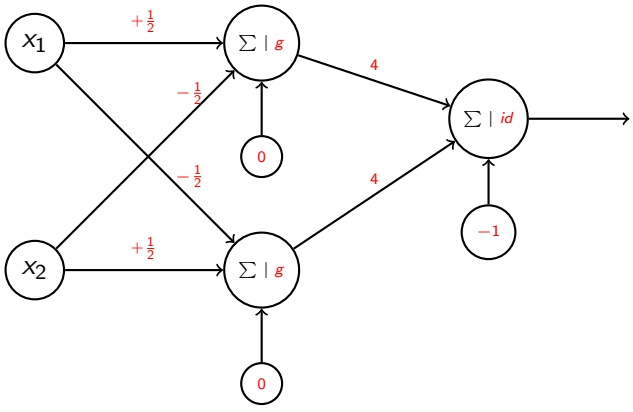
$\rightarrow \quad w_2^T h(x) + b_2 \\ = (4 \;\; 4) h(x) - 1 \quad \rightarrow$



$y = w_2^T h(x) + b_2$

Multilayer Perceptron/Neural Network (1980)

Ce n'est rien d'autre qu'un réseau de neurones avec une couche cachée h :



Apprentissage des paramètres

- Question :
  Comment déterminer les paramètres des neurones $w = (w_1, w_2, \cdots, w_p)$ et b.
- Réponse :
  Descente du gradient

Apprentissage des paramètres

- Question :
  Comment déterminer les paramètres des neurones $w = (w_1, w_2, \cdots, w_p)$ et b.
- Réponse :
  Descente du gradient patience ...

Outline

Deep Learning = (plusieurs) couches cachées.

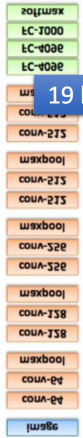
Exemples de réseaux connus

Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers

16.4%



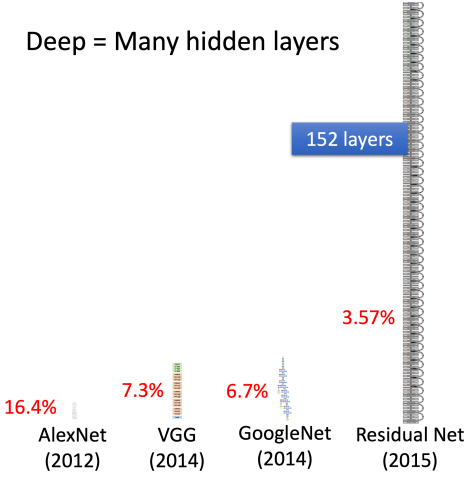
19 layers

7.3%

22 layers

6.7%

Exemples de réseaux connus



Pourquoi le deep

Un exemple :

- Apprendre une fonction AND : une seule couche

Pourquoi le deep

Un exemple :

- Apprendre une fonction AND : une seule couche
- Apprendre une fonction OR : une seule couche

Pourquoi le deep

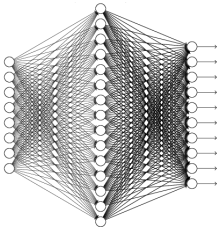
Un exemple :

- Apprendre une fonction AND : une seule couche
- Apprendre une fonction OR : une seule couche
- Apprendre une fonction XOR : une seule couche ne suffit pas ...

Pourquoi le deep

Universality Theorem [1], [2]

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

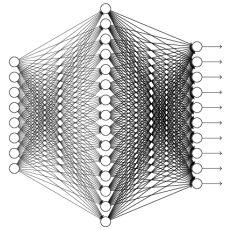


1. *Approximation by Superpositions of a Sigmoidal Function*. G. Cybenko. Mathematics of Control, Signals, and Systems. 2 : 303-314. (1989)
2. http ://neuralnetworksanddeeplearning.com/chap4.html

Pourquoi le deep

Universality Theorm [1], [2]

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

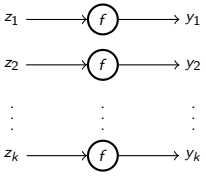


Question : pourquoi des réseaux profonds au lieu de réseaux à une couche ?

1. *Approximation by Superpositions of a Sigmoidal Function*. G. Cybenko. Mathematics of Control, Signals, and Systems. 2 : 303-314. (1989)
2. http ://neuralnetworksanddeeplearning.com/chap4.html

Couche de sortie

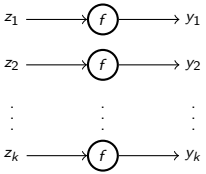
Ce que nous avons vu jusque là :



Problème : les sorties y_i prennent des valeurs quelconques et difficiles à interpréter ...

Couche de sortie

Ce que nous avons vu jusque là :

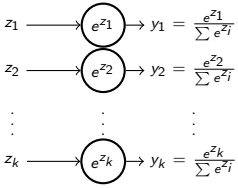


Problème : les sorties y_i prennent des valeurs quelconques et difficiles à interpréter ...
Solution : transformer les sorties en "probabilités".

Couche de sortie

La couche de Softmax :

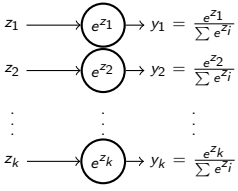
Solution : ajouter une couche de Softmax :



Couche de sortie

La couche de Softmax :

Solution : ajouter une couche de Softmax :

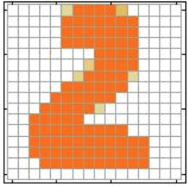


On obtient ainsi une distribution de probabilité :

- $0 < y_i < 1, \ \forall i,$
- $\sum_{i=1}^{k} y_i = 1.$

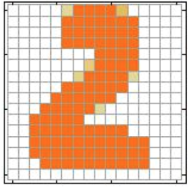
Exemple : reconnaissance d'écriture

Input **Output**



Exemple : reconnaissance d'écriture

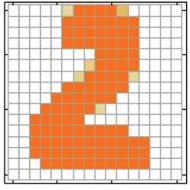
Input **Output**



16 x 16 = 256

Exemple : reconnaissance d'écriture

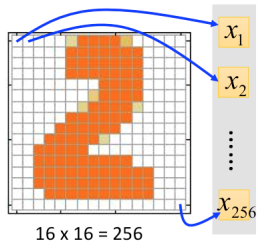
Input **Output**



16 x 16 = 256

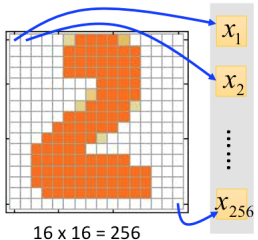
Exemple : reconnaissance d'écriture

Input **Output**



Exemple : reconnaissance d'écriture

Input **Output**

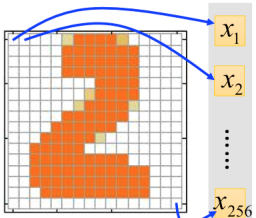


16 x 16 = 256

Ink → 1
No ink → 0

Exemple : reconnaissance d'écriture

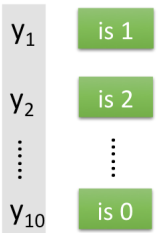
Input



16 x 16 = 256

Ink → 1
No ink → 0

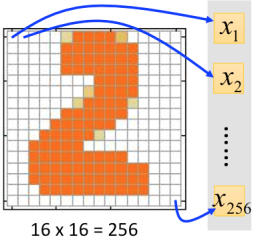
Output



Each dimension represents the confidence of a digit.

Exemple : reconnaissance d'écriture

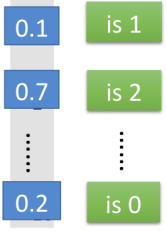
Input



16 x 16 = 256

Ink → 1
No ink → 0

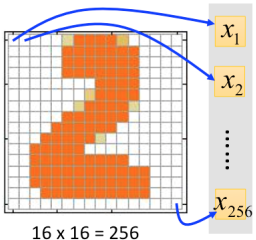
Output



Each dimension represents the confidence of a digit.

Exemple : reconnaissance d'écriture

Input



16 x 16 = 256

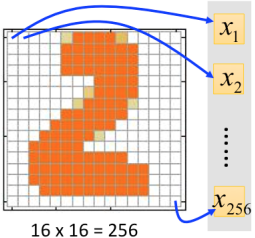
Ink → 1
No ink → 0

Output

Each dimension represents the confidence of a digit.

Exemple : reconnaissance d'écriture

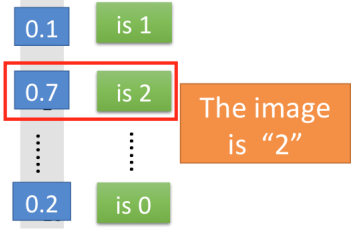
Input



16 x 16 = 256

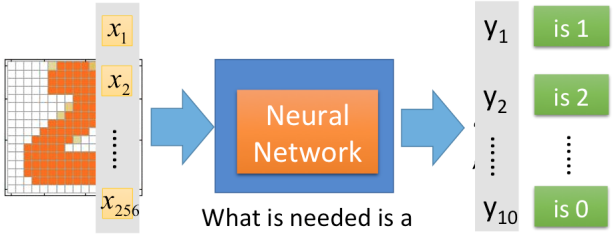
Ink → 1
No ink → 0

Output



Each dimension represents the confidence of a digit.

Exemple : reconnaissance d'écriture



Questions/Réponses

- Combien de couches mettre dans le réseaux ?

Questions/Réponses

- Combien de couches mettre dans le réseaux ?
- Peut-on concevoir la structure du réseau ?

Questions/Réponses

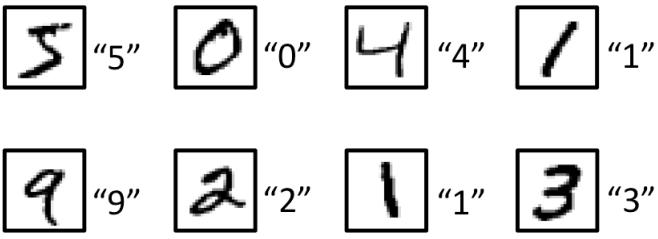
- Combien de couches mettre dans le réseaux ?
- Essayer, apprendre des erreurs et avoir de l'intuition ...
- Peut-on concevoir la structure du réseau ?

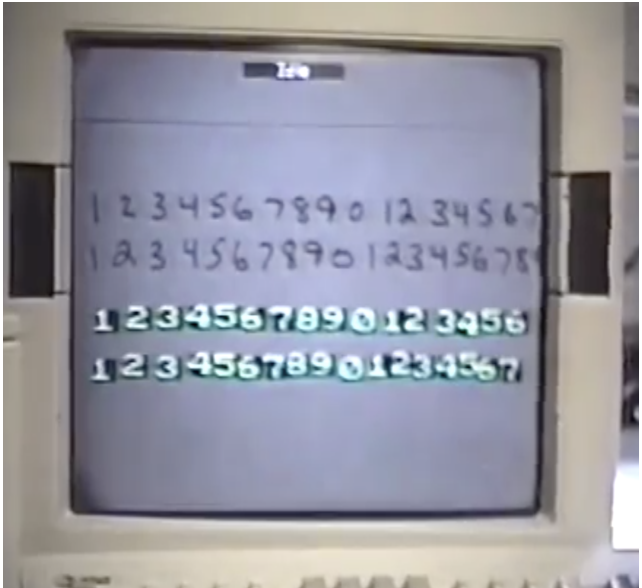
Questions/Réponses

- Combien de couches mettre dans le réseaux ?
- Essayer, apprendre des erreurs et avoir de l'intuition ...
- Peut-on concevoir la structure du réseau ?
- Réseaux de neurones à convolution ...

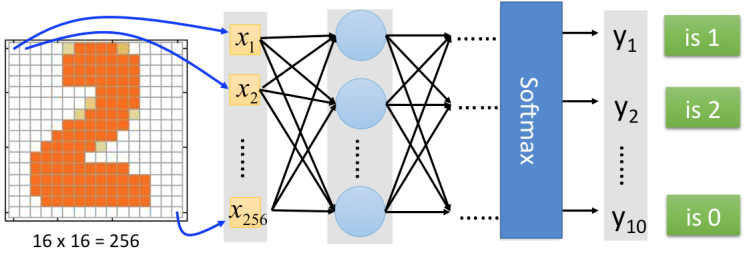
Données d'entrainement

Les Images et leurs étiquettes :



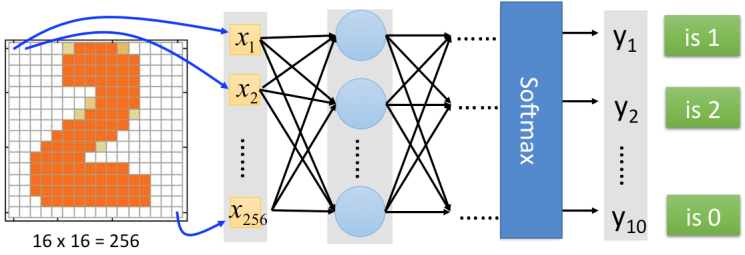


Objectif de l'entrainement



Ink → 1
No ink → 0

Objectif de l'entrainement



Ink → 1
No ink → 0

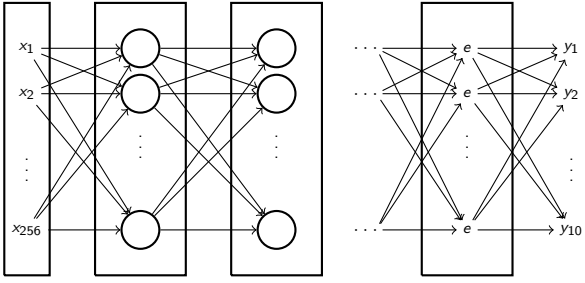
Input: 

$\Rightarrow y_1$ a la plus grande valeur,

Input: 

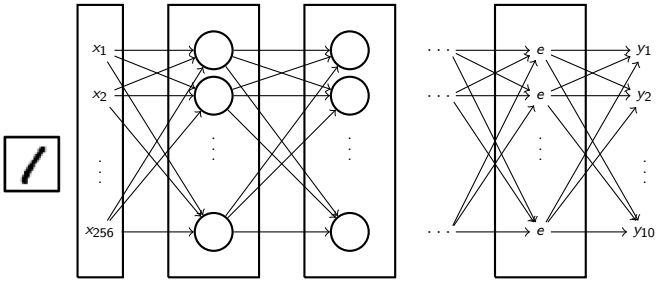
$\Rightarrow y_2$ a la plus grande valeur,

Fonction de perte (Loss Function)



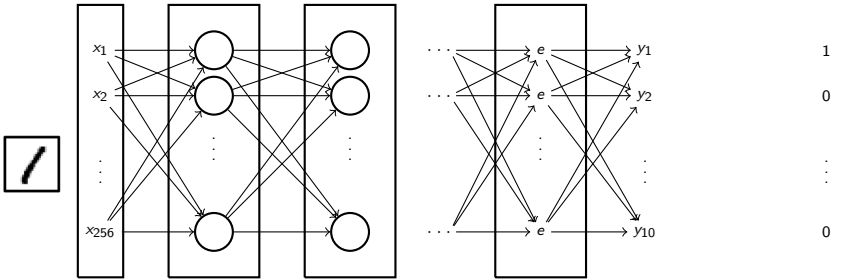
Couche de Softmax

Fonction de perte (Loss Function)

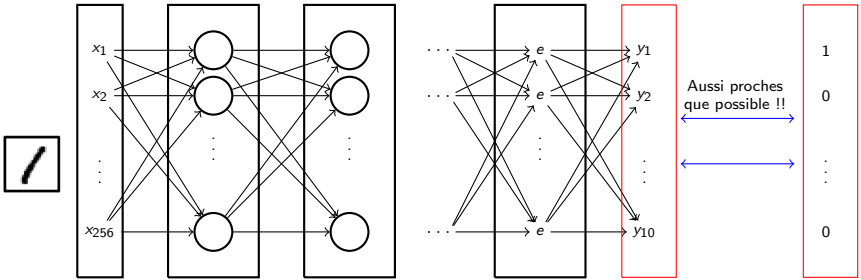


Couche de Softmax

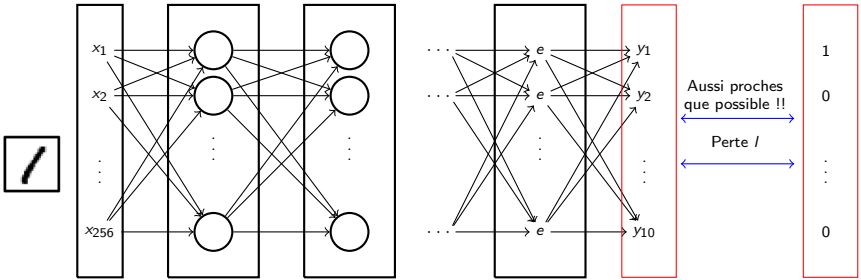
Fonction de perte (Loss Function)



Fonction de perte (Loss Function)



Fonction de perte (Loss Function)



Fonction de perte (Loss Function)

- La fonction de perte peut ê tre :
  - La somme des carrés des écarts :

$$I = \sum_{i=1}^{10} (y_i - \hat{y}_i)^2 .$$

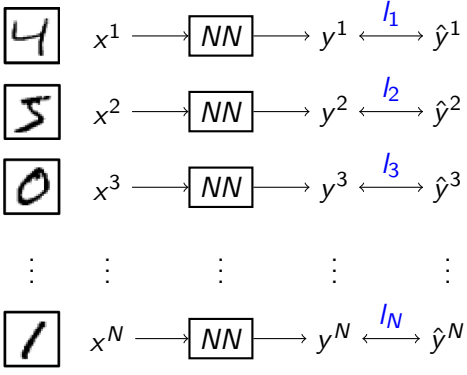
  - La *cross entropy* :

$$I = -\sum_{i=1}^{10} y_i \ln(\hat{y}_i).$$

Une bonne fonction (définie par les paramètres que l'on met sur le réseau) doit minimiser la *valeur totale* de la fonction de perte ...
Question : quelle est l'"intuition" derrière la définition de la cross-entropy ?

Perte totale

Pour tout l'ensemble de traitement :



Perte totale :

$$L = \sum_{i=1}^{N} l_i.$$

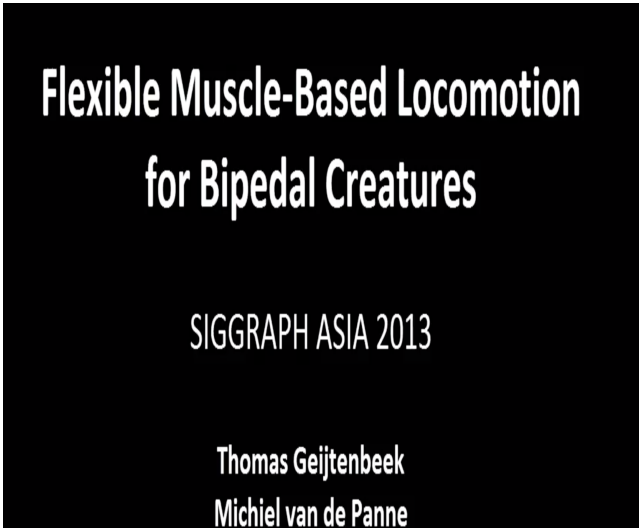
Objectif :

Minimiser cette fonction.

Outline

Apprentissage ?

Apprentissage ?



Apprentissage ?

Comment choisir la meilleure fonction de classification ?
Trouver les meilleurs paramètres θ qui minimisent la perte totale L.

- Solution 1. : Enumérer toutes les possibilités $\rightarrow$ infaisable ! !

Apprentissage ?

Comment choisir la meilleure fonction de classification ?
Trouver les meilleurs paramètres θ qui minimisent la perte totale L.

- Solution 1. : Enumérer toutes les possibilités $\rightarrow$ infaisable ! !
- Solution 2. : Descente du Gradient.

Descente du gradiant

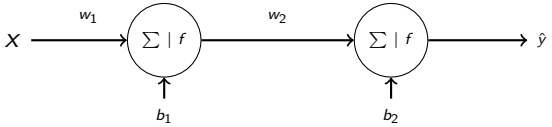
Algorithme de descente du gradient (voir cours annexe)

1. Initialiser avec θ_0 (au hasard) ;
2. Répéter
$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(\theta_t) ;$$
3. Jusqu'à convergence

- au hasard car quasiment impossible de trouver une valeur optimale
- $\nabla L(\theta_t)$ il s'agit du gradient
- η est un paramètre (pas d'apprentissage) qui permet de moduler la correction (η trop faible, lenteur de convergence ; η trop élevé, oscillation)
- convergence : Nombre d'itérations fixé, ou différence entre valeurs successives de θ_t ou $\nabla L(\theta_t)$ très petit

Algorithme de Back-propagartion

Considérons un réseau très simple :



Ecrivons les fonctions mathématiques de chaque couche :

$$z_1 = w_1 x + b_1 \quad (1)$$
$$y_1 = f(z_1) \quad (2)$$
$$z_2 = w_2 y_1 + b_2 \quad (3)$$
$$\hat{y} = f(z_2). \quad (4)$$

Algorithme de Back-propagartion

Pour simplifier, on considère que la fonction de perte (Loss function) est la somme des carrés des écarts :

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2 .$$

Et que la fonction d'activation (dans les deux couches) est simplement la sigmoÃ¯d :

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Rappelons le but de l'apprentissage :
Si on note $\theta = \{w_1, w_2, b_1, b_2\}$ trouver θ^* tel que :

$$\theta^* = \arg\min_{\theta} L(\theta).$$

Algorithme de Back-propagartion

Et rappelons la formule de la méthode du descente de gradient :

$$\theta^{t+1} = \theta^t - \eta \nabla L\left(\theta^t\right)$$

Ce que l'on peut traduire dans notre cas par :

$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w_1}\left(w_1^{(t)}, w_2^{(t)}, b_1^{(t)}, b_2^{(t)}\right)$$
$$w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w_2}\left(w_1^{(t)}, w_2^{(t)}, b_1^{(t)}, b_2^{(t)}\right)$$
$$b_1^{(t+1)} = b_1^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial b_1}\left(w_1^{(t)}, w_2^{(t)}, b_1^{(t)}, b_2^{(t)}\right)$$
$$b_2^{(t+1)} = b_2^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial b_2}\left(w_1^{(t)}, w_2^{(t)}, b_1^{(t)}, b_2^{(t)}\right)$$

Algorithme de Back-propagartion

LA question :

Comment calculer les dérivées partielles

$$\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \frac{\partial \mathcal{L}}{\partial b_1} \text{ et } \frac{\partial \mathcal{L}}{\partial b_2} ?$$

Algorithme de Back-propagartion

CommenÃ§ons par $w = w_2$.
On peut observer que :

$$\begin{aligned} \mathcal{L}(w) &= \mathcal{L}\left(\hat{y}(w)\right)) \\ &= \mathcal{L}\left(\hat{y}(z_2(w))\right) \end{aligned}$$

La règle de chaÃ®nage nous indique que :

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \hat{y}} . \frac{\partial \hat{y}}{\partial z_2} . \frac{\partial z_2}{\partial w} .$$

Algorithme de Back-propagartion

Faisons les calculs :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}}\left(\frac{1}{2}(y-\hat{y})^2\right) = (y-\hat{y}) \\
\frac{\partial \hat{y}}{\partial z_2} &= \frac{\partial}{\partial z_2}\left(f(z_2)\right) = \frac{\partial}{\partial z_2}\left(\sigma(z_2)\right) = \sigma(z_2)\left(1-\sigma(z_2)\right) \\
&= \hat{y}(1-\hat{y}). \\
\frac{\partial z_2}{\partial w} &= \frac{\partial}{\partial w}\left(w_2 y_1 + b_2\right) = y_1.
\end{aligned}$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_2} = (y-\hat{y})\,\hat{y}(1-\hat{y})y_1$$

Un calcul semblable (exercice) donne :

$$\frac{\partial \mathcal{L}}{\partial b_2} = (y-\hat{y})\,\hat{y}(1-\hat{y})$$

Algorithme de Back-propagartion

Remarques.
- Observer que la quantité $y - \hat{y}$ n'est autre que l'erreur commise.
- On voit donc que l'on dispose de tous les ingrédients pour calculer les deux dérivées nécessaires pour la dernière couche.

Qu'en est-il de la couche cachée ?

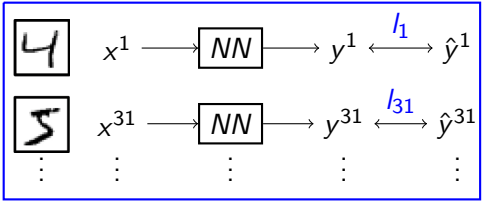
Algorithme de Back-propagartion

En fait tous calculs faits (exercice), on obtient :

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_1} &= (\hat{y}-y)\hat{y}(1-\hat{y})y_1(1-y_1)w_2 x \\ \frac{\partial \mathcal{L}}{\partial b_1} &= (\hat{y}-y)\hat{y}(1-\hat{y})y_1(1-y_1)w_2.\end{aligned}$$

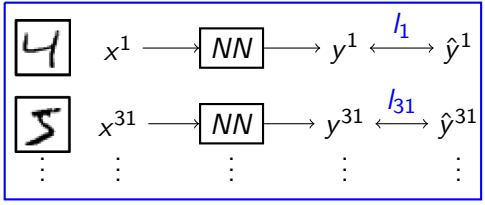
A la recherche de la meilleure fonction

On prend un premier mini-batch :



A la recherche de la meilleure fonction

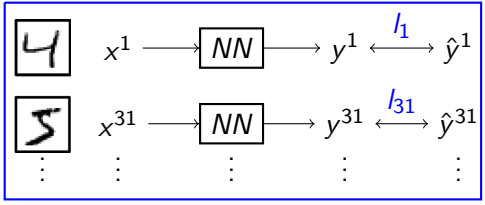
On prend un premier mini-batch :



$$L' = l^1 + l^{31} + \cdots$$

A la recherche de la meilleure fonction

On prend un premier mini-batch :

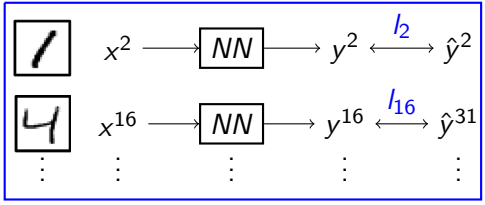


$L' = l^1 + l^{31} + \cdots$

Mettre à jour les paramètres

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



Et ainsi de suite

...

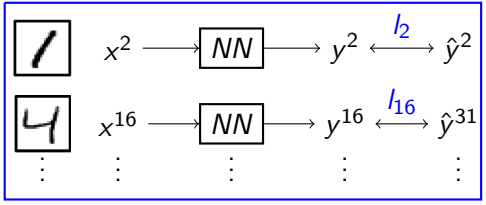
Jusqu'à ce que tous les mini-batchs soient lus.

$\Rightarrow$ Cele correspond à un(e) epoch.

Et l'on répète le processus un nombre d'epochs ...

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$$L'' = l^2 + l^{16} + \dots$$

Et ainsi de suite

...

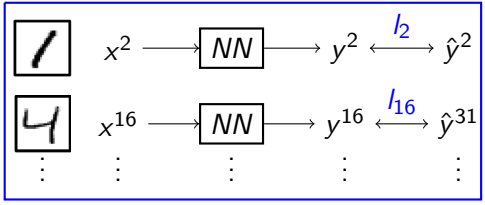
Jusqu'à ce que tous les mini-batchs soient lus.

$\Rightarrow$ Cele correspond à un(e) epoch.

Et l'on répète le processus un nombre d'epochs ...

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :

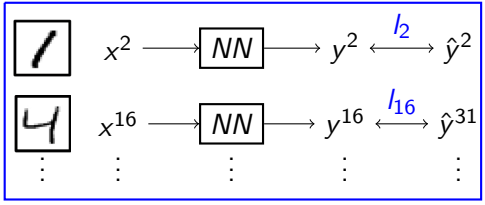


$L'' = l^2 + l^{16} + \dots$

Mettre à jour les paramètres

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$L'' = l^2 + l^{16} + \dots$

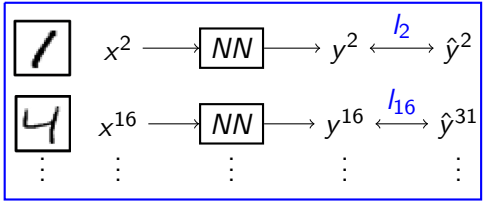
Mettre à jour les paramètres

Et ainsi de suite

...

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$L'' = l^2 + l^{16} + \dots$

Mettre à jour les paramètres

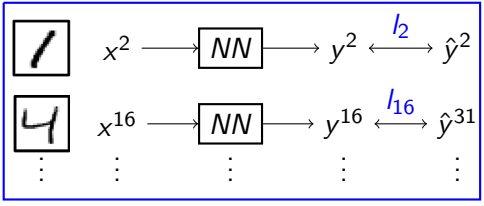
Et ainsi de suite

...

Jusqu'à ce que tous les mini-batchs soient lus.

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$L'' = l^2 + l^{16} + \dots$

Mettre à jour les paramètres

Et ainsi de suite

...

Jusqu'à ce que tous les mini-batchs soient lus.

$\Rightarrow$ Cele correspond à un(e) epoch.

Et l'on répète le processus un nombre d'epochs ...

A la recherche de la meilleure fonction

Dans la majorité des outils de deep learning, ces deux paramètres sont à indiquer :

- batch_size correspond au nombre d'exemples utilisé pour estimer le gradient de la fonction de coût.
- epochs est le nombre d'époques (i.e. passages sur l'ensemble des exemples de la base d ?apprentissage) lors de la descente de gradient.