Apprentissage supervisé

Introduction au connexionnisme

Master en informatique
de l'Université Lyon 2 (M1)

Didier Puzenat

Planning

Objectif : apprentissage automatique supervisé,
plutôt côté connexionnisme que mathématiques.

Enseignement :

- 7 heures de CM : $3 \times 2\text{h} + 1\text{h}$;
- 14 heures de TD sur machine : $7 \times 2\text{h}$ ($\times$ 2 groupes).

Évaluation :

- CM : validé en TD ;
- TD : rendus des fiches de TD via Moodle
  plus éventuellement un QCM sur la dernière séance de TD.

Définition de l'intelligence selon Wikipédia

Intelligence (≪ naturelle ≫)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation. L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.

Et pour l'intelligence artificielle ?

Définition de l'intelligence selon Wikipédia

Intelligence (≪ naturelle ≫)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation. L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.

Et pour l'intelligence artificielle ? Pourquoi pas la même définition ? ?

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma

Asimov : cerveau positronique, les lois de la robotique (pour le meilleur et le pire), etc.

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma

Data : encore du positronique, guidé par la recherche d'une humanité (par exemple via les émotions)...

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma

L'agent Smith : une IA purement logicielle, a priori sans émotion et avec un complexe de supériorité (puis une IA dépassée qui fait de la résistance)

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma






TRON le simple programme, le MCP un programme qui a très ≪ mal tourné ≫ et Quorra une ≪ IA ≫ non programmée qui émerge de la complexité de son environnement

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma







Jane (l'IA amie de Ender) : émerge également de la complexité, reconnue par Ender comme étant une nouvelle forme de vie « consciente »

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma








Caprica (l'ado, pas celle avec la robe rouge ;-) : les premiers pas du transhumanisme

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma









Terminator : le fantasme de l'IA qui prend le pouvoir (skynet) avec une armée de robots plus ou moins évolués (dont un T2 doué d'une certaine capacité d'apprentissage)

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma










C-3PO (aka Z-6PO) : ne comprend rien aux humains mais efficace dans son domaine d'expertise (traducteur)

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma











Hall 9000 : pour moi. . .le danger de l'anthropomorphisme (des humains) dans la relation IA-humain

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma












Le WOPR : il apprend très bien, pour lui le nombre de morts n'est qu'une variable, mais heureusement il préfère les échecs à la guerre thermonucléaire globale

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la littérature et du cinéma

Marvin : l'IA fidèle mais dépressive car beaucoup trop intelligente pour être heureuse

L'IA dans la littérature et au cinéma (et dans les séries)

Réaliste pour les années 2020 (2020-2030) :

- Hall 9000 : prévu pour tenir compagnie, assistant numérique, joue aux échecs, identifie les personnes, lit sur les lèvres, prend des décisions en conséquence.
- C-3PO et D2R2 (voire le Terminator T2) : comprend son environnement physique (voir T2 (2025 ?) *versus* Yolo 2016) mais pas humain, capacités moteurs limitées.
- WOPR (et Skynet ?) : de l'aide à la décision à la... décision.

L'IA dans la littérature et au cinéma (et dans les séries)

Réaliste pour les années 2020 (2020-2030) :

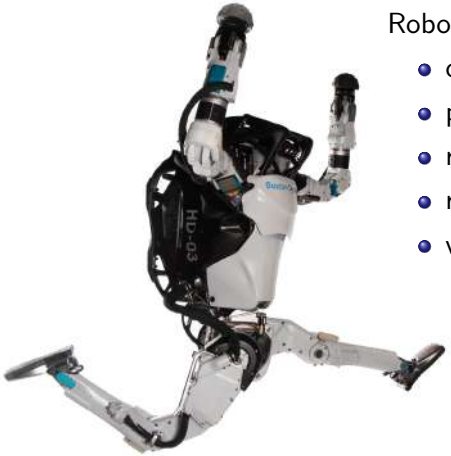
- Hall 9000 : prévu pour tenir compagnie, assistant numérique, joue aux échecs, identifie les personnes, lit sur les lèvres, prend des décisions en conséquence.
- C-3PO et D2R2 (voire le Terminator T2) : comprend son environnement physique (voir T2 (2025 ?) *versus* Yolo 2016) mais pas humain, capacités moteurs limitées.
- WOPR (et Skynet ?) : de l'aide à la décision à la... décision.

Les autres :

pour le moment encore du domaine de la SF

→ système à base de règles sans explosion combinatoire ?
cerveau positronique ? vie émergente ? singularité ?
scan du cerveau humain ? et circuit suffisant pour un tel scan ?

C-3PO et T2 plausibles en 2023 ? Exemple d'Altas de Boston Dynamics



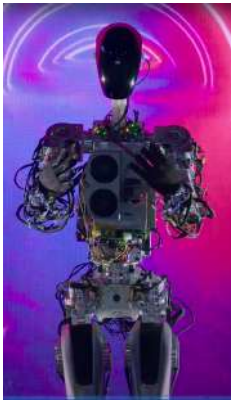
Robot Atlas :

- capable de maintenir son équilibre ;
- puissant (prévu pour la manutention de colis) ;
- relativement léger (moins de 90 kg pour 1,5 m) ;
- routines de mouvements combinables ;
- vision 2D (image) et 3D (nuage de points).

C-3PO et T2 plausibles en 2023 ? Exemple du Tesla Bot

Robot Tesla :

- low cost, 20 k$ versus 75 k$ pour Spot (le chien jaune) ;
- relativement low tech ;
- vision 2D (image) sur la base de ce qui est déjà développé pour la conduite autonome ;
- encore loin de la promesse...



L'IA dans le cinéma... coté Hollywood en septembre 2023

Le danger de l'IA vu par des scientifiques et entrepreneurs

2014 : Stephen Hawking :

- « sous-estimer l'IA serait la plus grave erreur de notre histoire »
- « on peut imaginer que cette technologie déjoue les marchés financiers, dépasse les chercheurs humains, manipule les dirigeants humains et développe des armes qu'on ne peut pas même comprendre »
- « la meilleure ou la pire chose qui puisse arriver à l'humanité »

2016 : Pétition contre l'IA dans les armes signée par Stephen Hawking, Elon Musk (Tesla, Space X, Starlink, Neuralink, etc.), Steve Wozniak (Apple), Noam Chomsky (linguiste), Demis Hassabis (fondateur de DeepMind), etc.

Et les politiques ?

Vladimir Poutine :

≪ le pays qui deviendra leader de ce secteur sera celui qui dominera le monde ≫

Et les politiques ?

Vladimir Poutine :

≪ le pays qui deviendra leader de ce secteur sera celui qui dominera le monde ≫

Et les politiques ?

Barack Obama :

≪ Historiquement, nous avons absorbé les nouvelles technologies, de nouveaux emplois ont été créés et notre niveau de vie s'est généralement amélioré. L'intelligence artificielle promet de créer une économie plus productive et efficace. Si elle est bien exploitée, cela peut générer énormément de prospérité et d'opportunités. ≫

Un film récent assez proche d'un (mauvais) futur proche possible

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Le cerveau humain : un moyen et un modèle !

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Alan Turing : un des pères de l'informatique, a contribué à l'IA par son « test de Turing »

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

McCulloch et Pitts : premier modèle informatique du neurone biologique (années 40)

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Frank Rosenblatt : perceptron (1957), modèle inspiré des théories cognitives

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Teuvo Kohonen : carte auto-adaptatives dites ≪ cartes de Kohonen ≫

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Paul John Werbos : perceptron multi-couches et rétro-propagation du gradient

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Vladimir Vapnik : théorie de l'apprentissage statistique, machines à support vectoriel

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Yoshua Bengio : le deep-learning ou ≪ apprentissage profond ≫

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la cybernétique.

Investissements massifs des géants du Net : Google, Apple, Facebook, Amazon, etc.

Le Connexionnisme est au centre de recherches pluridisciplinaires

Domaines s'intéressant aux réseaux de neurones artificiels :

- Biologie : neuropsychologie et fonctionnement du cerveau, réseaux de neurones...

Le Connexionnisme est au centre de recherches pluridisciplinaires

Domaines s'intéressant aux réseaux de neurones artificiels :

- Biologie : neuropsychologie et fonctionnement du cerveau, réseaux de neurones...
- Statistiques : alternative aux méthodes classiques

Le Connexionnisme est au centre de recherches pluridisciplinaires

Domaines s'intéressant aux réseaux de neurones artificiels :

- Biologie : neuropsychologie et fonctionnement du cerveau, réseaux de neurones...
- Statistiques : alternative aux méthodes classiques
- Informatique fondamentale : mémoire associative, théorie des graphes, réseau d'automates...

Le Connexionnisme est au centre de recherches pluridisciplinaires

Domaines s'intéressant aux réseaux de neurones artificiels :

- Biologie : neuropsychologie et fonctionnement du cerveau, réseaux de neurones...
- Statistiques : alternative aux méthodes classiques
- Informatique fondamentale : mémoire associative, théorie des graphes, réseau d'automates...
- Informatique appliquée : reconnaissance de formes, robotique, traitement de la parole... parallélisme massif, circuit VLSI, tolérance aux pannes...

Le Connexionnisme est au centre de recherches pluridisciplinaires

Domaines s'intéressant aux réseaux de neurones artificiels :

- Biologie : neuropsychologie et fonctionnement du cerveau, réseaux de neurones...
- Statistiques : alternative aux méthodes classiques
- Informatique fondamentale : mémoire associative, théorie des graphes, réseau d'automates...
- Informatique appliquée : reconnaissance de formes, robotique, traitement de la parole... parallélisme massif, circuit VLSI, tolérance aux pannes...
- Sciences Cognitives : modélisation (mémoire, émotions, raisonnement...), validation d'hypothèses...

Les réseaux de neurones

- Définition en biologie Cellules vivantes mettant en œuvre une très grande quantité de neurones connectés par des synapses.

  = siège de la mémoire, de l'intelligence, de la conscience...

Les réseaux de neurones

- Définition en biologie Cellules vivantes mettant en œuvre une très grande quantité de neurones connectés par des synapses.

  = siège de la mémoire, de l'intelligence, de la conscience...

- Définition en informatique Ensemble d'automates élémentaires interconnectés qui s'influencent par l'intermédiaire de ces connexions.

  $\Rightarrow$ peut être simulé sur un ordinateur !
  - application : e.g. science de l'ingénieur
  - recherche : e.g. test d'hypothèse biologiques

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
logique	analogique
séquentiel	parallèle
localisé	distribué
programmation	adaptation

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
données structurées ← logique	analogique → *données brutes*
séquentiel	parallèle
localisé	distribué
programmation	adaptation

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = « systèmes experts », « logique floue »

Intelligence Artificielle « IA symbolique »	Connexionnisme « IA numérique »
données structurées ← logique	analogique → *données brutes*
séquentiel	parallèle → *très rapide*
localisé	distribué
programmation	adaptation

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = « systèmes experts », « logique floue »

Intelligence Artificielle « IA symbolique »	Connexionnisme « IA numérique »
données structurées ← logique	analogique → *données brutes*
séquentiel	parallèle → *très rapide*
localisé	distribué → *tolérance aux pannes*
programmation	adaptation

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
données structurées ← logique	analogique → *données brutes*
séquentiel	parallèle → *très rapide*
localisé	distribué → *tolérance aux pannes*
programmation	adaptation → *pas de développement*

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
données structurées ← logique	analogique → *données brutes*
maîtrisé ! ← séquentiel	parallèle → *très rapide*
localisé	distribué → *tolérance aux pannes*
programmation	adaptation → *pas de développement*

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
données structurées ← logique	analogique → *données brutes*
maîtrisé ! ← séquentiel	parallèle → *très rapide*
rassurant ! ← localisé	distribué → *tolérance aux pannes*
programmation	adaptation → *pas de développement*

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
données structurées ← logique	analogique → *données brutes*
maîtrisé ! ← séquentiel	parallèle → *très rapide*
rassurant ! ← localisé	distribué → *tolérance aux pannes*
connu ! ← programmation	adaptation → *pas de développement*

Comparaison IA symbolique / numérique

L'Intelligence Artificiel comprend :

- l'IA numérique = les réseaux de neurones artificiels
- l'IA symbolique = ≪ systèmes experts ≫, ≪ logique floue ≫

Intelligence Artificielle ≪ IA symbolique ≫	Connexionnisme ≪ IA numérique ≫
données structurées ← logique	analogique → *données brutes*
maîtrisé ! ← séquentiel	parallèle → *très rapide*
rassurant ! ← localisé	distribué → *tolérance aux pannes*
connu ! ← programmation	adaptation → *pas de développement*

Conclusion : l'IA numérique a eu du mal à percer !

Choisir entre IA symbolique et numérique

- IA symbolique à utiliser pour des problèmes de « haut niveau », lorsque l'on possède déjà :
  - des variables clairement identifiées
  - une très bonne expertise sur le domaine (les règles)

- IA numérique à utiliser pour les données brutes, éventuellement avec du bruit, peu structurées...

Choisir entre IA symbolique et numérique

- IA symbolique à utiliser pour des problèmes de « haut niveau », lorsque l'on possède déjà :
  - des variables clairement identifiées
  - une très bonne expertise sur le domaine (les règles)

  Par exemple :
  - aide à la décision, diagnostique...
  - modélisation de processus cognitifs de haut niveau
- IA numérique à utiliser pour les données brutes, éventuellement avec du bruit, peu structurées...

  Par exemple :
  - classification, prévision...
  - modélisation de pré-traitements intelligents

Les fondements biologiques du connexionnisme

- XVII Cerveau = rôle central comme organe de commande
- XIX début de la neuropsychologie

  = rapport entre ≪ la matière ≫ et la fonction
  = les fonctions cérébrales sont localisées et cartographiées

Les fondements biologiques du connexionnisme

- XVII Cerveau = rôle central comme organe de commande
- XIX début de la neuropsychologie

  = rapport entre « la matière » et la fonction
  = les fonctions cérébrales sont localisées et cartographiées

Ordres de grandeur comparés des réseaux de neurones :

- système nerveux central : 10^{11} neurones
  un GPU récent (AMD RTX 3080) : 10^{10} transistors
  un CPU récent (AMD Ryzen 3990X) : 40 milliards $\rightarrow$ 10^{13} !
- mais 1 mm^3 de néo-cortex :
  - 60 000 neurones

Les fondements biologiques du connexionnisme

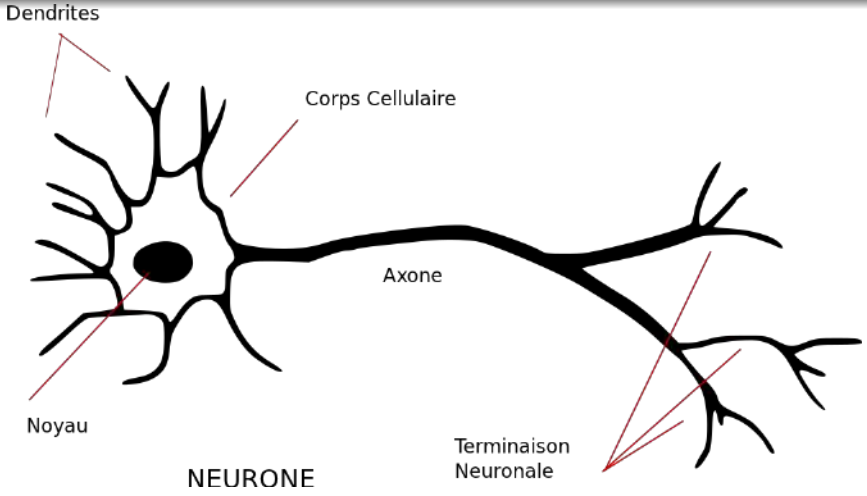
- XVII Cerveau = rôle central comme organe de commande
- XIX début de la neuropsychologie

  = rapport entre ≪ la matière ≫ et la fonction
  = les fonctions cérébrales sont localisées et cartographiées

Ordres de grandeur comparés des réseaux de neurones :

- système nerveux central : 10^{11} neurones
  un GPU récent (AMD RTX 3080) : 10^{10} transistors
  un CPU récent (AMD Ryzen 3990X) : 40 milliards $\rightarrow$ 10^{13} !
- mais 1 mm^3 de néo-cortex :
  - 60 000 neurones
  - 3 10^9 synapses !
  - 1,5 km de dendrites !
  - 3 km d'axone !

Les fondements biologiques du connexionnisme



Rappel du fonctionnement d'un neurone biologique

Le rôle du neurone biologique est de :

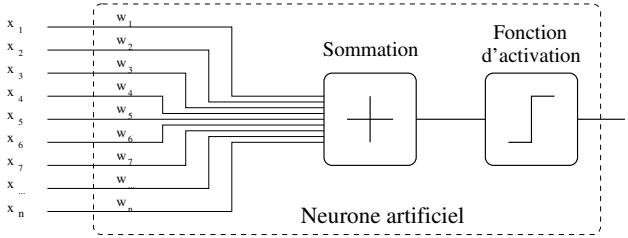
1. recevoir l'influx des autres neurones, via des dendrites
2. « intégrer » les influx reçus, dans le corps cellulaire
3. engendrer éventuellement un influx nerveux
4. transmettre l'influx à un autre neurone, via l'axone
   (l'axone fait de 1 mm à 1 m !)

Attention : l'influx, le « potentiel d'action », est codé en fréquence
$\Rightarrow$ tous les potentiels d'action on la même amplitude

Le neurone formel de Mc Culloch et Pitts (1943)

Principe :

- vecteur d'entrée
  $X = (x_1, x_2, \ldots, x_n)$
- vecteur poids
  $W = (w_1, w_2, \ldots, w_n)$
- fonction d'activation

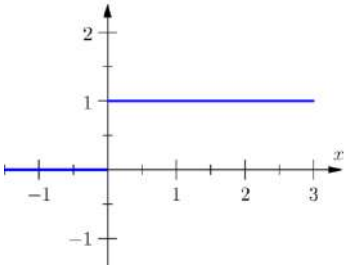


Remarque : on est passé d'un codage en fréquence à un codage en amplitude !

Exemple de « l'automate booléen à seuil »

Description de l'automate booléen à seuil :

- vecteur d'entrée booléen
- fonction d'activation de Heaviside (notée $\mathcal{H}$)
  $\Rightarrow$ sortie aussi booléenne
- poids et le seuil réels



Fonctionnement :

1. calcul de la somme pondérée des entrées : $\sum_{i=1}^{n} w_i x_i$
2. calcul de la sortie de la cellule :

$$s = \mathcal{H}(\sum_{i=1}^{n} w_i x_i - \theta) = \begin{cases} 1 & si \ \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0 & si \ \sum_{i=1}^{n} w_i x_i < \theta \end{cases}$$

Présentation du perceptron de Rosenblatt (1958)

Description du perceptron :

- entrées réelles
- fonction d'activation de Heaviside (notée $\mathcal{H}$) $\Rightarrow$ sortie encore booléenne
- poids et le seuil réels

Principe : un perceptron, avec 1 neurone, divise l'espace d'entrée $\mathbb{R}^n$ en deux (et seulement deux) sous-espaces séparés par un hyperplan d'équation :

$$w_1 x_1 + \ldots + x_i w_i + \ldots + x_n w_n - \theta = 0$$

$\Rightarrow$ on va pouvoir faire un $\ll$ choix $\gg$ $\Rightarrow$ de la classification

Interprétation géométrique du perceptron

Si on se limite à $\mathbb{R}^2$:

le perceptron divise l'espace d'entrée en deux demi-plans par une droite d'équation :

$$w_1 x_1 + x_2 w_2 - \theta = 0$$

$\Rightarrow$ selon l'entrée de la cellule, on considère un demi-plan ou l'autre

$\Rightarrow$ le perceptron est capable d'effectuer une discrimination de l'espace d'entrée en deux classes linéairement séparables.

Simulation des fonctions *ET*, *OU*, et *XOR*

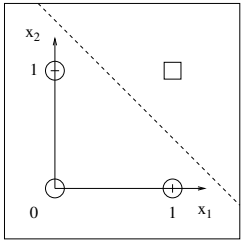
Principe : on veut simuler les fonctions logiques booléennes *ET*, *OU*, et le *OU exclusif*, avec un perceptron.

x_1	x_2	*ET*	*OU*	*OU exclusif*
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Simulation des fonctions *ET*, *OU*, et *XOR*

Principe : on veut simuler les fonctions logiques booléennes *ET*, *OU*, et le *OU exclusif*, avec un perceptron.

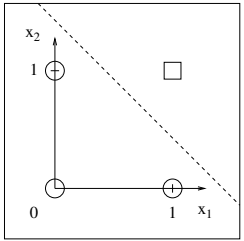
x_1	x_2	*ET*	*OU*	*OU exclusif*
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

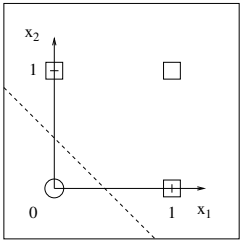


Simulation des fonctions *ET*, *OU*, et *XOR*

Principe : on veut simuler les fonctions logiques booléennes *ET*, *OU*, et le *OU exclusif*, avec un perceptron.

x_1	x_2	*ET*	*OU*	*OU exclusif*
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

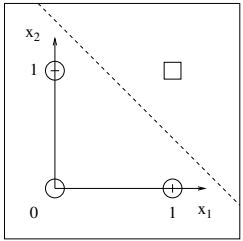


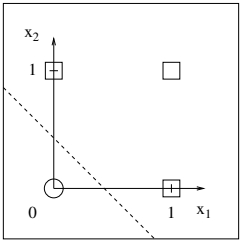


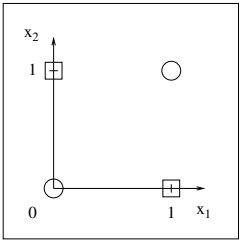
Simulation des fonctions *ET*, *OU*, et *XOR*

Principe : on veut simuler les fonctions logiques booléennes *ET*, *OU*, et le *OU exclusif*, avec un perceptron.

x_1	x_2	*ET*	*OU*	*OU exclusif*
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0







Le problème du « ou exclusif » (XOR)

$\Rightarrow$ Impossible de discriminer les valeurs de vérité du *OU exclusif* !

La limitation est significative :

le pourcentage de fonctions booléennes linéairement séparables diminue très rapidement si on augmente la dimension de l'espace d'entrée (n) :

n	séparables	total
2	14	16
3	104	256
4	1882	65536
5	94572	$4{,}3\ 10^9$
6	5028134	$1{,}8\ 10^{19}$

Théorème de Mc Culloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

Théorème de Mc Culloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

$\Rightarrow$ on a les 3 opérateurs de la logique booléenne

$\Rightarrow$ on peut synthétiser toutes les fonctions booléennes

Théorème de Mc Culloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

$\Rightarrow$ on a les 3 opérateurs de la logique booléenne
$\Rightarrow$ on peut synthétiser toutes les fonctions booléennes

Théorème : Toute fonction booléenne de n variables peut être synthétisée par un réseau d'automates à seuil possédant n entrées

Théorème de Mc Culloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

$\Rightarrow$ on a les 3 opérateurs de la logique booléenne
$\Rightarrow$ on peut synthétiser toutes les fonctions booléennes

Théorème : Toute fonction booléenne de n variables peut être synthétisée par un réseau d'automates à seuil possédant n entrées

. . .mais encore faut-il savoir le construire !

Apprentissage et Généralisation

Objectif : trouver les w_i et θ automatiquement
en présentant une base d'exemples au réseau

Apprentissage et Généralisation

Objectif : trouver les w_i et θ automatiquement
en présentant une base d'exemples au réseau

Il faut :

- un algorithme d'apprentissage
- une base d'exemples
  = couples (vecteur d'entrée X, sortie désirée pour X)

Exemple : algorithme d'apprentissage du perceptron

Algorithme :

1. On présente un exemple = on calcule la sortie du réseau
2. Si la sortie ne correspond pas à la sortie désirée
   = si on est pas du bon coté de l'hyperplan
   $\Rightarrow$ on modifie les poids et le seuil
3. On boucle...

Remarques : l'algorithme converge toujours
... si c'est linéairement séparable !

La généralisation

Définition : la généralisation est

la réponse du réseau sur des exemples qui n'étaient pas dans la base d'apprentissage

Objectifs :

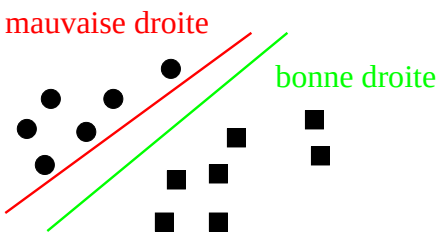
- connaître la réponse pour des exemples « inconnus »
  couples (vecteur d'entrée X, ?)
- évaluer la qualité de l'apprentissage
  couples (vecteur d'entrée X, sortie connue)

Performances en généralisation

Les performances en généralisation dépendent :

- de la qualité de l'algorithme d'apprentissage
- de l'adéquation du réseau (topologie, taille, dynamique...)
- de la qualité des exemples utilisés pour l'apprentissage, il faut notamment une bonne distribution des exemples

Exemple d'un mauvais apprentissage typique de l'algorithme classique du perceptron :



Le danger du sur-apprentissage

Dans une base d'apprentissage « réelle », on a souvent :

- un bruit important
- quelques exemples non représentatifs
- éventuellement des erreurs d'étiquetage...

⇒ pour certains problèmes,
il est inutile de chercher à apprendre **tous** les exemples

⇒ arrêt de l'apprentissage une fois un certain pourcentage
de la base d'exemples apprise (« taux d'apprentissage »)

⇒ évite un « sur-apprentissage »

= *en apprenant trop « par cœur » on généralise moins bien*

The credit assigment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant plusieurs couches de perceptrons

mais on n'a pas la « sortie désirée » des couches non terminales

$\Rightarrow$ comment apprendre ?

The credit assigment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant plusieurs couches de perceptrons

mais on n'a pas la ≪ sortie désirée ≫ des couches non terminales

$\Rightarrow$ comment apprendre ?

$=$ ≪ *credit assigment problem* ≫

Problème mis en évidence par Minsky et Papert en 1969 et resté ouvert 20 ans !

The credit assigment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant plusieurs couches de perceptrons

mais on n'a pas la ≪ sortie désirée ≫ des couches non terminales

$\Rightarrow$ comment apprendre ?

$=$ ≪ *credit assigment problem* ≫

Problème mis en évidence par Minsky et Papert en 1969 et resté ouvert 20 ans !

Solution : algorithme de la rétro-propagation du gradient

. . . on va voir plus tard

Résumé chronologique des débuts du connexionnisme

Années 40 : Van Neumann, Mc Culloch, Turing...
fondement d'une science des systèmes capables d'auto-organisation,
la cybernétique ;

Années 60 : Rosenblatt reprend les automates de Mc Culloch et Pitts
pour faire de la reconnaissance de formes ;

Années 70 : Minsky et Papert mettent en évidence les limites du Perceptron

$\Rightarrow$ années de l'IA symbolique (gros travaux) ;
on veut simuler directement le raisonnement,
grâce à de la logique et l'inférence (systèmes experts, logique floue)
mais problème : explosion combinatoire....

Années 90 : Algorithme de la rétro-propagation du gradient !

Exemple d'utilisations historiques du Perceptron

Exemple d'un Perceptron construit dans les années 60 pour (essayer de...) déterminer le genre sur la base de photos de visages :

Une anecdote et un commentaire de Marvin Minsky sur le Perceptron :

Les trois grandes familles

La capacité du réseau à résoudre le problème va dépendre de :

Les trois grandes familles

La capacité du réseau à résoudre le problème va dépendre de :

- l'algorithme d'apprentissage...
- des exemples à apprendre
- de l'architecture du réseau

Les trois grandes familles

La capacité du réseau à résoudre le problème va dépendre de :

- l'algorithme d'apprentissage...
- des exemples à apprendre
- de l'architecture du réseau :
  - topologie du réseau
  - dynamique du réseau (propagation de l'influx)
  - taille du réseau

Les trois grandes familles

La capacité du réseau à résoudre le problème va dépendre de :

- l'algorithme d'apprentissage...
- des exemples à apprendre
- de l'architecture du réseau :
  - topologie du réseau
  - dynamique du réseau (propagation de l'influx)
  - taille du réseau

Il existe 3 grandes familles de réseaux :

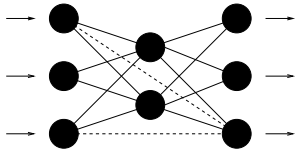
- les réseaux à couches
- les réseaux à connexions latérales
- les réseaux récurrents

Les réseaux à couches

Définition : réseau à couches = « graphe acyclique de cellules »

On peut définir une ou plusieurs :

- cellules d'entrée
- cellules de sortie



Remarques :

- les connexions sont orientées (entrée $\rightarrow$ sortie)
- le nombre de couches = nombre de cellules entre les cellules d'entrée et les cellules de sortie
- de loin les réseaux les plus utilisés !

Propagation de l'influx dans un réseau à couches

La propagation dans un réseau à couches est dite

« *feed forward* » = « alimenter et faire suivre »

Propagation de l'influx dans un réseau à couches

La propagation dans un réseau à couches est dite

« *feed forward* » = « alimenter et faire suivre »

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = les cellules de la première couche (entrée) sont « forcées »

Propagation de l'influx dans un réseau à couches

La propagation dans un réseau à couches est dite

« *feed forward* » = « alimenter et faire suivre »

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = les cellules de la première couche (entrée) sont « forcées »
2. l'influx se propage de couche en couche (entrée → sortie)
   = on active les couches les unes après les autres

Propagation de l'influx dans un réseau à couches

La propagation dans un réseau à couches est dite

« *feed forward* » = « alimenter et faire suivre »

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = les cellules de la première couche (entrée) sont « forcées »
2. l'influx se propage de couche en couche (entrée → sortie)
   = on active les couches les unes après les autres
3. la réponse du réseau peut être lue sur la dernière couche

Propagation de l'influx dans un réseau à couches

La propagation dans un réseau à couches est dite

« *feed forward* » = « alimenter et faire suivre »

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = les cellules de la première couche (entrée) sont « forcées »
2. l'influx se propage de couche en couche (entrée → sortie)
   = on active les couches les unes après les autres
3. la réponse du réseau peut être lue sur la dernière couche

Remarque : le temps de réponse est constant et connu ⇒ temps réel possible

Les réseaux à connexions latérales

Réseau à connexions latérales =
réseau à couches + connexions entre cellules d'une même couche

Les réseaux à connexions latérales

Réseau à connexions latérales =
réseau à couches + connexions entre cellules d'une même couche

Dynamique, pour chaque couche de l'entrée vers la sortie :

1. activation en fonction de la couche précédente,
   via les connexions inter-couches exclusivement

Les réseaux à connexions latérales

Réseau à connexions latérales =
réseau à couches + connexions entre cellules d'une même couche

Dynamique, pour chaque couche de l'entrée vers la sortie :

1. activation en fonction de la couche précédente,
   via les connexions inter-couches exclusivement
2. activation en fonction du nouvel état de la couche courante,
   via les connexions intra-couche exclusivement

Les réseaux à connexions latérales

Réseau à connexions latérales =
réseau à couches + connexions entre cellules d'une même couche

Dynamique, pour chaque couche de l'entrée vers la sortie :

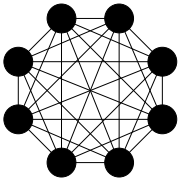
1. activation en fonction de la couche précédente,
   via les connexions inter-couches exclusivement
2. activation en fonction du nouvel état de la couche courante,
   via les connexions intra-couche exclusivement

Remarques :

- généralement peu de couches
- généralement pas de connexions latérales sur chaque couche

Exemple : les cartes de Kohonen (on verra plus tard...)

Les réseaux récurrents



Définition : réseau récurrents =

≪ graphe avec une ou plusieurs boucles ≫

Particularités :

- partiellement ou totalement interconnecté
- connexions orientées, éventuellement dans les 2 sens

Remarques :

- L'état d'activation de l'ensemble des cellules à un instant t forme une ≪ configuration d'états ≫
- Il existe des configurations d'états dites ≪ stables ≫

Propagation de l'influx dans un réseau récurrent

≪ *Feedback neural networks* ≫ = ≪ alimenté en retour ≫

Propagation de l'influx dans un réseau récurrent

≪ *Feedback neural networks* ≫ = ≪ alimenté en retour ≫

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = certaines (voire toutes) les cellules sont ≪ forcées ≫

Propagation de l'influx dans un réseau récurrent

≪ *Feedback neural networks* ≫ = ≪ alimenté en retour ≫

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = certaines (voire toutes) les cellules sont ≪ forcées ≫
2. pour chaque cellule :
   l'influx se propage d'une cellule à l'autre
   = on active toutes les cellules
   ⇒ forme une nouvelle ≪ configuration d'états ≫

Propagation de l'influx dans un réseau récurrent

≪ *Feedback neural networks* ≫ = ≪ alimenté en retour ≫

Dynamique du réseau :

1. l'exemple à apprendre (en apprentissage) ou la forme à reconnaître (en généralisation) est présentée au réseau
   = certaines (voire toutes) les cellules sont ≪ forcées ≫
2. pour chaque cellule :
   l'influx se propage d'une cellule à l'autre
   = on active toutes les cellules
   ⇒ forme une nouvelle ≪ configuration d'états ≫
3. si une configuration stable est atteinte, la réponse du réseau peut être lue sur certaines (voire toutes) les cellules

Ordre d'activation des cellules

En « *feedforwad* » l'ordre d'activation n'a pas d'importance :
il n'a pas d'influence sur le comportement / la sortie du réseau

Ordre d'activation des cellules

En « *feedforwad* » l'ordre d'activation n'a pas d'importance :
il n'a pas d'influence sur le comportement / la sortie du réseau
mais ce n'est pas le cas avec une dynamique « *feedback* » !

Important : dans un réseau récurrent l'ordre d'activation des cellules influe sur l'évolution des configurations d'états, et donc sur l'éventuelle réponse du réseau

Ordre d'activation des cellules

En « *feedforwad* » l'ordre d'activation n'a pas d'importance :
il n'a pas d'influence sur le comportement / la sortie du réseau
mais ce n'est pas le cas avec une dynamique « *feedback* » !

Important : dans un réseau récurrent l'ordre d'activation des cellules influe sur l'évolution des configurations d'états, et donc sur l'éventuelle réponse du réseau

L'activation peut être :

- « synchrone » : toutes les cellules s'activent « simultanément » sur la base de l'état d'activation précédent
- « asynchrone » : les cellules s'activent les unes après les autres, dans un ordre donné (fixé) ou aléatoire, sur la base de l'état d'activation courant

Lecture de la réponse (la sortie) d'un réseau récurrent

Dans un réseau à couches : la réponse est lue sur la dernière couche
après un temps constant et connu

Lecture de la réponse (la sortie) d'un réseau récurrent

Dans un réseau à couches : la réponse est lue sur la dernière couche après un temps constant et connu

Dans un réseau récurrent :

- le réseau peut converger vers une configuration d'états stable
  = propager de nouveau l'influx ne change pas la configuration
  $\Rightarrow$ la réponse du réseau est la configuration stable trouvée !
- le réseau peut converger vers un cycle, voire ne pas converger
  $\Rightarrow$ pas de réponse

Lecture de la réponse (la sortie) d'un réseau récurrent

Dans un réseau à couches : la réponse est lue sur la dernière couche
après un temps constant et connu

Dans un réseau récurrent :

- le réseau peut converger vers une configuration d'états stable
  = propager de nouveau l'influx ne change pas la configuration
  $\Rightarrow$ la réponse du réseau est la configuration stable trouvée !
- le réseau peut converger vers un cycle, voire ne pas converger
  $\Rightarrow$ pas de réponse

Remarque : le temps de réponse est variable (stabilisation)
$\Rightarrow$ temps réel délicat à mettre en œuvre

Principe de l'apprentissage d'un réseau récurrent

Principe : on utilise les états stables du réseau récurrent pour mémoriser les exemples

Principe de l'apprentissage d'un réseau récurrent

Principe : on utilise les états stables du réseau récurrent pour mémoriser les exemples

$\Rightarrow$ on modifie les poids / les seuils
afin de converger sur une forme apprise à partir d'une forme :

- incomplète
- complète mais bruitée
- incomplète et bruitée

Principe de l'apprentissage d'un réseau récurrent

Principe : on utilise les états stables du réseau récurrent pour mémoriser les exemples

$\Rightarrow$ on modifie les poids / les seuils
afin de converger sur une forme apprise à partir d'une forme :

- incomplète
- complète mais bruitée
- incomplète et bruitée

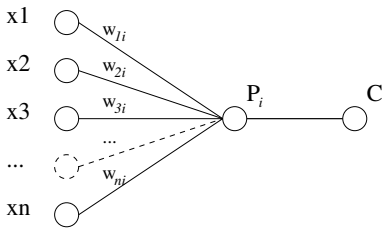
Exemple de réseau récurrent : les réseaux de Hopfield
(on verra plus tard, très bientôt ;-)

Les modèles à base de prototypes

Prototype = cellule dont les connexions entrantes, pondérées par des réels, stockent un exemple représentatif d'une classe

La fonction d'activation évalue la ressemblance entre :

- un exemple en entrée $X = (x_1 x_2 \dots x_n)$
- le prototype $P_i = (w_{1i} w_{2i} \dots w_{ni})$



Le choix de la mesure est lié à la nature de l'espace d'entrée

Hypersphères contre hyperplans

Un prototype s'active si l'entrée lui ressemble
= si l'exemple est dans la zone d'influence définie par ses poids
$\Rightarrow$ le prototype sépare l'espace en deux sous espaces :

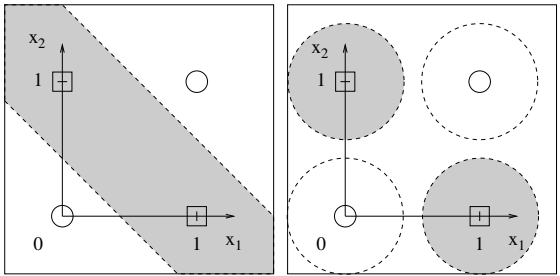
- à l'intérieur de l'hypersphère
- à l'extérieur de l'hypersphère

Hypersphères contre hyperplans

Un prototype s'active si l'entrée lui ressemble
= si l'exemple est dans la zone d'influence définie par ses poids
⇒ le prototype sépare l'espace en deux sous espaces :

- à l'intérieur de l'hypersphère
- à l'extérieur de l'hypersphère

Exemple sur le problème du XOR :



Prototypes et localité

Dans un réseau à base de prototypes, la ≪ connaissance ≫ stockée est :

- localisée
- cohérente avec l'entrée

Prototypes et localité

Dans un réseau à base de prototypes, la « connaissance » stockée est :

- localisée → facile à retrouver
- cohérente avec l'entrée → facile à « visualiser »

Prototypes et localité

Dans un réseau à base de prototypes, la « connaissance » stockée est :

- localisée → facile à retrouver
- cohérente avec l'entrée → facile à « visualiser »

⇒ on peut analyser la connaissance du réseau
= pouvoir explicatif (limité)

Apprentissage pour un réseau à base de prototypes

Pour apprendre, on peut :

- changer les poids
- changer les seuils
- ajouter des prototypes, on parle de « réseau incrémental »
- supprimer des prototypes, on parle « d'élagage »

Apprentissage pour un réseau à base de prototypes

Pour apprendre, on peut :

- changer les poids = déplacer le centre de l'hypersphere
- changer les seuils = changer le rayon de l'hypersphere
- ajouter des prototypes, on parle de « réseau incrémental »
- supprimer des prototypes, on parle « d'élagage »

Apprentissage pour un réseau à base de prototypes

Pour apprendre, on peut :

- changer les poids = déplacer le centre de l'hypersphere
- changer les seuils = changer le rayon de l'hypersphere
- ajouter des prototypes, on parle de « réseau incrémental »
- supprimer des prototypes, on parle « d'élagage »

Les principaux modèles à base de prototypes :

- les cartes de Kohonen (1982-84)
- les algorithmes LVQ de Kohonen (1988-90)
- le néocognitron de Fukushima (1980)
- le modèle ART de Carpenter et Grossberg (1987-88)

Le modèle de réseau récurrent de Hopfield (1982 / 1984)

Le modèle de Hopfield :

- réseau récurrent
- totalement interconnecté
- connexions symétriques : $w_{ij} = w_{ji}$
- pas « d'auto-connexions » : $w_{ii} = 0$
- activation asynchrone
  $\rightarrow$ on ne change qu'une cellule à la fois
  $\rightarrow$ a permis à Hopfield de prouver la convergence
- les points peuvent être appris ou calculés

Inspiration : Hebb (biologiste) $\rightarrow$ « il faut renforcer les poids des cellules activées simultanément »

Calcul des poids d'un réseau Hopfield

Apprentissage selon la règle de Hebb $\Rightarrow w_{ij} = w_{ij} + \mu y_i y_j$ avec :

- μ : paramètre déterminant l'intensité de l'apprentissage
- y_i : sortie de la cellule $i = \ll$ état $\gg$ de la cellule i

Calcul des poids :

- pour P exemples binaires (-1, 1) $X^1, X^2, \ldots, X^P$:

$$\begin{cases} w_{ij} = \sum_{p=1}^{P} x_i^p x_j^p \ pour \ i \neq j \\ w_{ij} = 0 \ pour \ i = j \end{cases}$$

- pour P exemples binaires plus classique, dans (0, 1) :

$$\begin{cases} w_{ij} = \sum_{p=1}^{P} [2x_i^p - 1][2x_j^p - 1] \ pour \ i \neq j \\ w_{ij} = 0 \ pour \ i = j \end{cases}$$

Apprentissage des poids d'un réseau Hopfield

On obtient les mêmes poids que par le calcul, par apprentissage :

1. démarrer avec des connexions nulles
2. présenter un exemple X^p : $y_i = x_i^p$ *pour* $i = 1 \rightarrow n$
3. modifier les connexions :

$$\begin{cases} +1 & \textit{si les cellules sont actives ou inactives simultanement} \\ -1 & \textit{sinon} \end{cases}$$

4. boucler en 2,
   jusqu'à ce que les P exemples aient été présentés

Généralisation dans un réseau Hopfield

Algorithme de généralisation :

1. on présente l'exemple X : $y_i = x_i$ *pour* $i = 1 \rightarrow n$
2. tant que le réseau n'a pas convergé, pour chaque cellule (dans un ordre aléatoire) :

   on active la cellule : $y'_i = \sum_{j=1}^{n} y_j w_{ji}$

   on détermine le nouvel état de la cellule : $\begin{cases} 1 & si\ y' > \theta_i \\ y_i & si\ y' = \theta_i \\ 0 & si\ y' < \theta_i \end{cases}$

Remarques :

- on a convergé si les états (y_i) ne changent plus
- on prend généralement tous les $\theta_i = 0$

Borne sur la capacité de stockage d'un réseau Hopfield

Quel est le nombre de formes que l'on peut stocker et reconnaître avec une « fréquence raisonnable » avec un réseau de Hopfield de n cellules, avec l'algorithme d'apprentissage vu ?

Selon des études expérimentales de Hopfield :

$$P_{max} = 0,15n$$

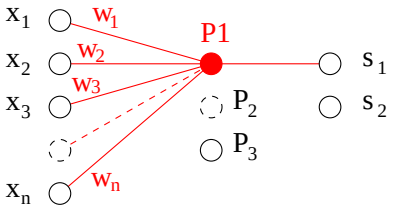
Borne théorique en 1987 :

$$P_{max} \equiv \frac{n}{2 \log_2 n}$$

Topologie de réseaux Learning Vector Quantization (LVQ)

Réseaux à 3 couches :

- l'entrée $X = (x_1, x_2, \ldots, x_n)$
- les prototypes $P_i = (w_1, w_2, \ldots, w_n)$
- les sorties



Algorithme de généralisation :

1. on présente un exemple (une forme) à reconnaître
2. tous les prototypes s'activent... et le plus fort gagne !
3. la sortie du prototype gagnant s'active → correspond en général à une classe

Algorithme d'apprentissage typique de LVQ

Pour chaque exemple X à apprendre :

pour i tel que $P_i = P_{meilleur}$:

si $C(P_i) = E$ alors $\forall j \quad w_{ji} \leftarrow w_{ji} + \alpha(x_j - w_{ji})$

sinon $\forall j \quad w_{ji} \leftarrow w_{ji} - \alpha(x_j - w_{ji})$

avec :

- $E =$ étiquette de la forme $X = (x_1 x_2 \dots x_n)$ en entrée
- $C(P_i) =$ classe du prototype P_i
- α ($\alpha < 1$) = paramètre de gain

Précision : α pondère les modifications et diminue lentement au cours de l'apprentissage

Variantes de l'apprentissage de LVQ

Il existe de nombreuses variantes, par exemple :

- repousser tous les perdants
- augmenter le θ du gagnant
  éventuellement diminuer le θ du perdant
- ...

Remarques sur le prototype gagnant :

- il devient le représentant d'une classe
- il est affiné d'exemple en exemple appris

Les cartes de Kohonen

Cartes de Kohonen = cartes « auto-organisatrices » = *Self Organizing Maps (SOM)*

Réseau à couches avec :

- 2 couches en non supervisé, 3 couches en supervisé
- la 2^e couche forme une carte, généralement 2D ou 3D
- connexions latérales sur la 2^e couche, selon le voisinage des cellules sur la carte

Nombre de cellules :

- couche 1 : selon la dimension de l'espace d'entrée
- couche 2 : selon la taille de la carte

Algorithmique des cartes de Kohonen

Algorithme de généralisation : idem LVQ

Algorithme d'apprentissage :

1. on présente un exemple
2. on cherche le prototype le plus activé
3. on approche le prototype gagnant et ses voisins sur la carte

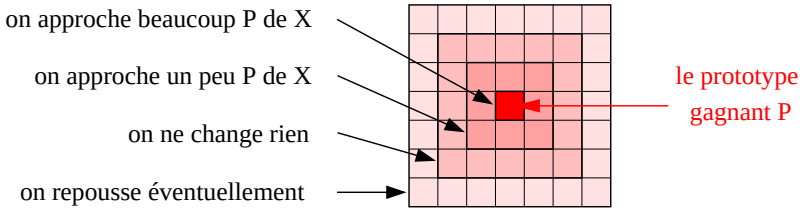
Remarques :

- en général on laisse les prototypes perdants inchangés
- c'est de l'apprentissage non supervisé mais...
  on peut ajouter une 3^e couche pour l'étiquetage $\rightarrow$ supervisé

Influence de la distance des voisins

Modification des poids :

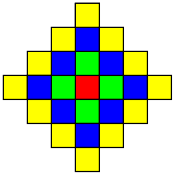
- la modification des poids dépend de la distance des voisins
- la modification peut être négative en limite de voisinage
  par exemple en utilisant une fonction type ≪ chapeau mexicain ≫



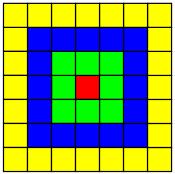
Forme et taille du voisinage

On peut définir plusieurs types de voisinages :

- les 4 voisins (et leurs voisins...)
- les 8 voisins (et leurs voisins...)



les 4 voisins



les 8 voisins

Remarques : il y a d'autres voisinages possibles, et on peut faire des cartes en 3D...

Auto-organisation de la carte !

On peut faire varier la taille du voisinage au cours de l'apprentissage, par exemple :

- au début, toute la carte
- à chaque ≪ passe ≫ de la base d'exemples on diminue le voisinage d'une cellule
- à la fin, on ne modifie que le prototype gagnant

On constate une auto-organisation de la carte :

- les prototypes représentants d'une même classe sont adjacents
  $\rightarrow$ facilite l'étiquetage en non supervisé
- la disposition des classes sur la carte rend compte de propriétés communes

Machine à vecteurs de support

En anglais *Support Vector Machine* ou *SVM*.

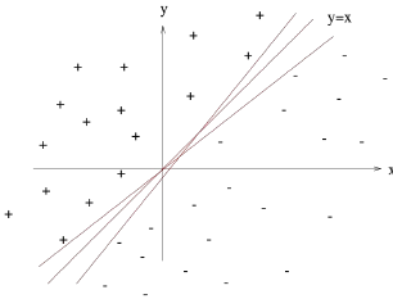
Intérêt :

- efficace avec des données de grandes dimensions
- faible nombre d'hyper paramètres
- garanties théoriques
- bons résultats en pratique

Origine : développé dans les années 1990
à partir des travaux de Vladimir Vapnik.

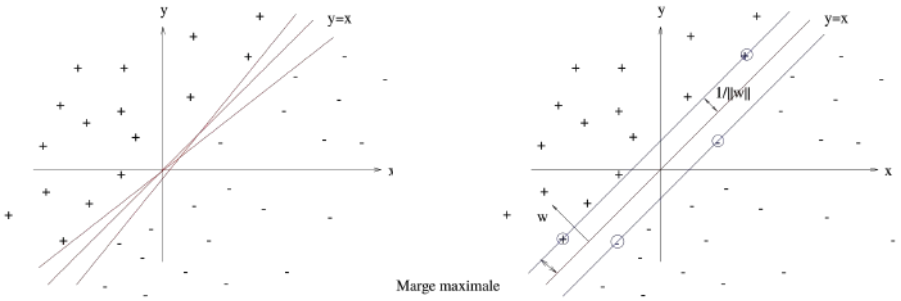
Machine à vecteurs de support – principe 1/2

On part d'un neurone type perceptron



Machine à vecteurs de support – principe 1/2

On part d'un neurone type perceptron
mais on recherche l'hyperplan séparateur avec « la marge maximale »



Marge maximale

Exemples les plus proches de l'hyperplan = « vecteurs supports »

Machine à vecteurs de support – principe 2/2

Problème : limité aux problèmes linéairement séparables...

Solution : transformer l'espace de représentation des données d'entrées en un espace de plus grande dimension dans lequel existera une séparatrice linéaire

Astuce : on va utiliser des fonctions noyaux particulières

Machine à vecteurs de support – principe 2/2

Problème : limité aux problèmes linéairement séparables...

Solution : transformer l'espace de représentation des données d'entrées en un espace de plus grande dimension dans lequel existera une séparatrice linéaire

Astuce : on va utiliser des fonctions noyaux particulières

Les fonctions noyau utilisées ne nécessitent pas la connaissance explicite de la transformation à appliquer pour le changement d'espace $\Rightarrow$ peu coûteux en calculs !

Machine à vecteurs de support – extensions du modèle

On peut faire du multi-classes avec plusieurs classifieurs SVM :
méthodes *one-versus-all* ($\equiv$ *one-versus-the-rest*) et *one-versus-one*

Machine à vecteurs de support – extensions du modèle

On peut faire du multi-classes avec plusieurs classifieurs SVM :
méthodes *one-versus-all* ($\equiv$ *one-versus-the-rest*) et *one-versus-one*

1995 : introduction des « marges souples »
$\Rightarrow$ permettent de relâcher les contraintes sur les exemples
$\Rightarrow$ évite le sur-apprentissage
et permet de gérer des exemples mal étiquetés
mais introduit un hyper-paramètre « C »
(C = compromis nombre d'erreurs / largeur de la marge)

Machine à vecteurs de support – extensions du modèle

On peut faire du multi-classes avec plusieurs classifieurs SVM :
méthodes *one-versus-all* ($\equiv$ *one-versus-the-rest*) et *one-versus-one*

1995 : introduction des « marges souples »
$\Rightarrow$ permettent de relâcher les contraintes sur les exemples
$\Rightarrow$ évite le sur-apprentissage
et permet de gérer des exemples mal étiquetés
mais introduit un hyper-paramètre « C »
(C = compromis nombre d'erreurs / largeur de la marge)

1996 : méthode pour utiliser des SVM en régression (sortie réelle)
$\Rightarrow$ on peut tout faire $\Rightarrow$ devient très populaire

Comment apprendre avec un perceptron multi-couche ?

Rappel :

- une couche = on sait faire (perceptron)
- plusieurs couches = *credit assigment problem*

Comment apprendre avec un perceptron multi-couche ?

Rappel :

- une couche = on sait faire (perceptron)
- plusieurs couches = *credit assigment problem*

Idée : rétro-propager l'erreur de la sortie vers l'entrée,
en modifiant les poids des connexions en fonction de leur contribution à l'erreur

Comment apprendre avec un perceptron multi-couche ?

Rappel :

- une couche = on sait faire (perceptron)
- plusieurs couches = *credit assigment problem*

Idée : rétro-propager l'erreur de la sortie vers l'entrée,
en modifiant les poids des connexions en fonction de leur contribution à l'erreur

Rappel : apprentissage = modifier les poids

$\Rightarrow$ il faut expliciter l'erreur globale en fonction de tous les poids
puis dériver cette fonction par rapport aux poids dont on veut
connaître la contribution à l'erreur globale

$\Rightarrow$ il faut utiliser des fonctions d'activation dérivables !

La fonction sigmoïde

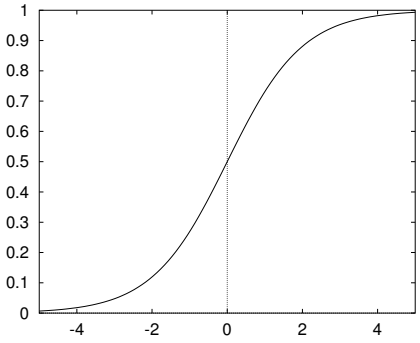
On veut toujours un « effet seuil » mais avec une fonction dérivable

La fonction sigmoïde

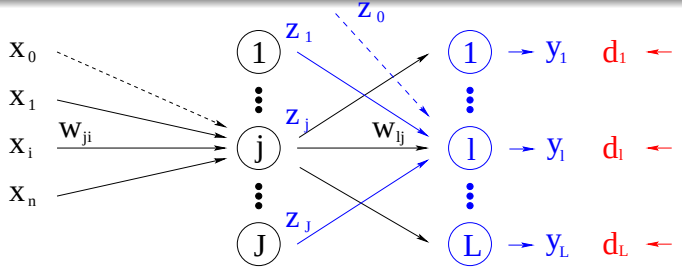
On veut toujours un « effet seuil » mais avec une fonction dérivable

On peut par exemple utiliser la fonction sigmoïde :

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



Topologie détaillée d'un perceptron multi-couche



Couche d'entrée	:	$X = (x_0, x_1, x_2, \ldots, x_n)$	$\in \mathbb{R}^{n+1}$ $(x_0 = 1)$
Couche cachée	:	$Z = (z_0, z_1, x_2, \ldots, z_J)$	$\in \mathbb{R}^{J+1}$ $(z_0 = 1)$
Couche de sortie	:	$Y = (y_1, y_2, \ldots, y_L)$	$\in \mathbb{R}^{L}$
Sortie désiré	:	$D = (d_1, d_2, \ldots, d_L)$	$\in \mathbb{R}^{L}$

Sortie désirée connue = l'apprentissage est dit « supervisé »

Apprentissage d'un perceptron multi-couche

Tant qu'on n'a pas atteint le taux d'apprentissage désiré :

1. on présente un exemple X
2. on active la couche cachée $\equiv$ on calcul Z
3. on active la couche de sortie $\equiv$ on calcul Y
4. on détermine l'erreur, entre Y et D
5. on modifie les poids entre la couche cachée et la couche de sortie, en fonction de leur contribution à l'erreur
6. on modifie les poids entre la couche d'entrée et la couche cachée, en fonction de leur contribution à l'erreur

Choix de la fonction d'activation des couches cachées

Pour la couche cachée, par exemple :

- sigmoïde :

$$f_h(net) = \frac{1}{(1 + e^{-\lambda net})}$$

- tangente hyperbolique :

$$f_h(net) = \tanh(\beta net)$$

avec :

- net = somme pondérée des entrées de la cellules, en l'occurrence : $net = \sum_{i=0}^{n} x_i w_{ji}$
- λ et β = paramètres de l'apprentissage (proches de 1)

Choix de la fonction d'activation de la couche de sortie

Pour la couche de sortie $\rightarrow$ dépend de l'application :

- approximation, prévision...

  $\Rightarrow$ sortie réelle et proportionnelle, par exemple : $f_o(net) = \lambda net$

- classification, décision...

  $\Rightarrow$ sortie binaire, par exemple : $f_o(net) = \frac{1}{(1+e^{-\lambda net})}$

Attention : Il faut que la sortie désirée (D) soit dans l'intervalle des valeurs que peut prendre f_o $\Rightarrow$ le cas échéant il faut normaliser

Fonction d'erreur

On veut apprendre m paires d'exemples : $\{X^k, D^k\}\ k = 1, \ldots, m$
$\equiv$ on veut adapter les $J(n+1) + L(J+1)$ poids du réseau
afin de minimiser l'erreur sur l'ensemble des exemples

$\Rightarrow$ on a besoin d'une fonction d'erreur !
$\equiv$ une mesure de la distance entre D et Y
$\Rightarrow$ très dépendant de l'application

Exemple : erreur quadratique : $d(W) = \frac{1}{2} \sum_{l=1}^{L} (d_l - y_l)^2$

avec W représentant l'ensemble des poids du réseau

Modification des poids

- couche de sortie, on connaît la sortie désirée :

$$\Delta w_{lj} = w_{lj}^{new} - w_{lj}^{old} = -\rho_o \frac{\partial E}{\partial w_{lj}} = -\rho_o (d_l - y_l)\ f_o'(net_l)\ z_j$$

- couche(s) cachés(s) :

$$\Delta w_{ji} = -\rho_h \frac{\partial E}{\partial w_{ji}} = \rho_h \left[\sum_{l=1}^{L} (d_l - y_l)\ f_o'(net_l)\ w_{lj} \right] f_h'(net_j)\ x_i$$

Utilisation de la bibliothèque FANN

FANN is FUN !

Fast Artificial Neural Network (FANN) library is a free open source neural network library, which implements multilayer artificial neural networks in C with support for both fully connected and sparsely connected networks.

FANN includes a framework for easy handling of training data sets.

FANN is easy to use, versatile, well documented, and fast. Bindings to more than 20 programming languages are available.

Quelques caractéristiques de FANN

- Multilayer Artificial Neural Network Library in C
- Backpropagation training
- Easy to use (create, train and run with just three functions)
- Cross-platform (linux and unix, windows)
- Open source, but can still be used in commercial applications
- Graphical Interfaces
- Bindings to a large number of programming languages
- Widely used (approximately 100 downloads a day)

La licence LGPL de FANN

Rappel sur la **GPL** : GNU General Public License

L'objectif est de garantir à l'utilisateur les droits suivants :

1. liberté d'exécuter le logiciel, pour n'importe quel usage ;
2. liberté d'étudier le programme et de l'adapter à ses besoins, ce qui passe par l'accès aux codes sources ;
3. liberté de redistribuer des copies ;
4. obligation de faire bénéficier à tous des modifications.

La LesserGPL (ou licence GPL « faible ») de FANN

La licence libre LGPL autorise à lier le programme à du code non LGPL, sans pour autant révoquer la licence. Cette Licence LGPL permet donc de s'affranchir du caractère héréditaire de la GPL. C'est donc la clause de copyleft que n'a pas la LGPL.

- Permet à un logiciel propriétaire d'utiliser une librairie libre, sans devenir un logiciel libre ;
- toute modification de code source de la bibliothèque LGPL devra être également publiée sous la licence LGPL ;
- passage sous GPL par simple mise à jour des notifications.

FANN à partir de Python pour apprendre un XOR

On va apprendre la fonction XOR avec un MLP,
il nous faut une base d'apprentissage :

on crée le fichier texte « `xor.data` » :

```
4 2 1
-1 -1
-1
-1 1
1
1 -1
1
1 1
-1
```

FANN à partir de Python pour apprendre un XOR

On va apprendre la fonction XOR avec un MLP,

il nous faut une base d'apprentissage : fichier texte « `xor.data` » →

avec la première ligne comprenant :

- le nombre d'exemples (4)
- le nombre d'entrées (2)
- le nombre de sortie(s) (1)

puis les exemples : les entrées et à la ligne la sortie

Remarque : on a pris les entrées et sorties dans [-1, 1]
mais on aurait pu les prendre dans [0, 1]

xor.data

```
4 2 1
-1 -1
-1
-1 1
1
1 -1
1
1 1
-1
```

Paramètres du réseau

```
1 #!/usr/bin/python
2
3 from fann2 import libfann
4
5 connection_rate = 1
6 learning_rate = 0.7
7
8 input = 2  # nombre de cellules sur la couche d'entree
9 hidden = 4 # nombre de cellules sur la couche cachee
10 output = 1 # nombre de cellules sur la couche de sortie
11
12 error = 0.0001       # erreur cible
13 iterations = 100000 # nombre d'interation maximum
14 reports = 10         # nombre d'interation entre deux affichages
15
```

Création du réseau et apprentissage

On crée le MLP (lignes 16 et 17),

on fixe le taux d'apprentissage (18),

on choisit une fonction sigmoïde dans [-1,1] pour la cellule de la couche de sortie (19),

puis on apprend (21) et on sauve l'architecture et les poids (22) :

```
15
16 ann = libfann.neural_net()
17 ann.create_sparse_array(connection_rate, (input, hidden, output))
18 ann.set_learning_rate(learning_rate)
19 ann.set_activation_function_output(libfann.SIGMOID_SYMMETRIC_STEPWISE)
20
21 ann.train_on_file("xor.data", iterations, reports, error)
22 ann.save("xor.net")
```

Mise en œuvre du réseau et résultat

Dans un autre programme on peut charger le réseau et l'utiliser :

```
1 #!/usr/bin/python
2
3 from fann2 import libfann
4
5 ann = libfann.neural_net()
6 ann.create_from_file("xor.net")
7
8 print ann.run([1, -1])
```

```
dpuzenat@X220T:~/tmp$ python FANN-1.py
Max epochs   100000. Desired error: 0.0000100000.
Epochs            1. Current error: 0.2503619194. Bit fail 4.
Epochs           10. Current error: 0.2487698793. Bit fail 4.
Epochs           20. Current error: 0.1971800178. Bit fail 3.
Epochs           30. Current error: 0.1376865059. Bit fail 2.
Epochs           40. Current error: 0.0189153515. Bit fail 0.
Epochs           50. Current error: 0.0000819563. Bit fail 0.
Epochs           51. Current error: 0.0000000000. Bit fail 0.
dpuzenat@X220T:~/tmp$ python FANN-2.py
[1.0]
```

Remarques : le résultat (valeur réelle 1)
aurait pu être récupérée dans une variable

Et à présent en C :-)

Pour ceux qui préfèrent le langage C :

```c
#include "fann.h"

int main()
{
    const unsigned int num_input = 2;
    const unsigned int num_output = 1;
    const unsigned int num_layers = 3;
    const unsigned int num_neurons_hidden = 3;
    const float desired_error = (const float) 0.001;
    const unsigned int max_epochs = 500000;
    const unsigned int epochs_between_reports = 1000;

    struct fann *ann = fann_create_standard(num_layers, num_input,
        num_neurons_hidden, num_output);

    fann_set_activation_function_hidden(ann, FANN_SIGMOID_SYMMETRIC);
    fann_set_activation_function_output(ann, FANN_SIGMOID_SYMMETRIC);

    fann_train_on_file(ann, "xor.data", max_epochs,
        epochs_between_reports, desired_error);

    fann_save(ann, "xor_float.net");

    fann_destroy(ann);

    return 0;
}
```

```c
#include <stdio.h>
#include "floatfann.h"

int main()
{
    fann_type *calc_out;
    fann_type input[2];

    struct fann *ann = fann_create_from_file("xor_float.net");

    input[0] = -1;
    input[1] = 1;
    calc_out = fann_run(ann, input);

    printf("xor test (%f,%f) -> %f\n", input[0], input[1], calc_out[0]);

    fann_destroy(ann);
    return 0;
}
```

Et à présent en C :-)

Pour la compilation il faut lier aux bibliothèques lfann et lm :

```
dpuzenat@X220T: ~/tmp
dpuzenat@X220T:~/tmp$ gcc FANN-1.c -lfann -lm -o FANN-1
dpuzenat@X220T:~/tmp$ ./FANN-1
Max epochs   500000. Desired error: 0.0010000000.
Epochs            1. Current error: 0.2505110204. Bit fail 4.
Epochs           27. Current error: 0.0007874089. Bit fail 0.
dpuzenat@X220T:~/tmp$ gcc FANN-2.c -lfann -lm -o FANN-2
dpuzenat@X220T:~/tmp$ ./FANN-2
xor test (-1.000000,1.000000) -> 0.946073
dpuzenat@X220T:~/tmp$
```

Remarques : cette fois j'ai demandé moins de précision (et un affichage plus léger)

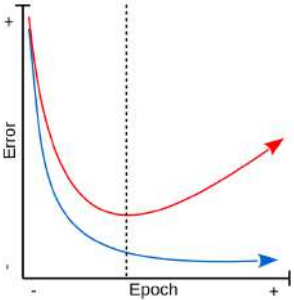
Éviter le sur-apprentissage

Courbe bleu : erreur en apprentissage

→ si le réseau a assez de cellules, il peut apprendre parfaitement !

Courbe rouge : erreur en généralisation

→ si on apprend « par cœur », on généralise moins bien.



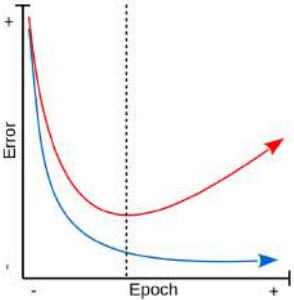
Éviter le sur-apprentissage

Courbe bleu : erreur en apprentissage

→ si le réseau a assez de cellules, il peut apprendre parfaitement !

Courbe rouge : erreur en généralisation

→ si on apprend ≪ par cœur ≫, on généralise moins bien.



Stratégie :

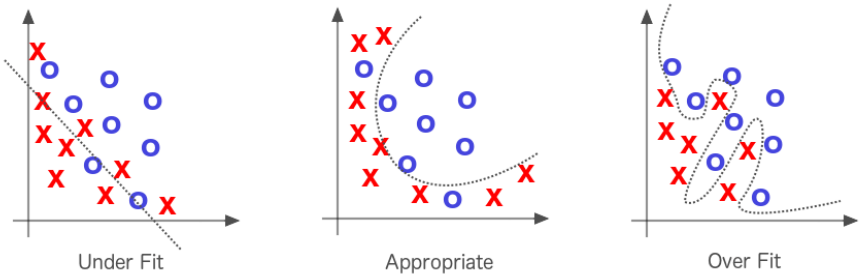
- tester en généralisation après chaque époque,
- arrêter l'apprentissage
  si la généralisation se dégrade.

Remarque : il peut être bon de ≪ patienter ≫.

Éviter le sur-apprentissage

C'est quoi en fait...

- « sur-apprendre » *(to overfit)* ?
- apprendre « par cœur » ?



Augmenter la base d'exemples

Solutions pour éviter le sur-apprentissage :

- s'arrêter à temps (comme déjà vu) ;
- ne pas utiliser un modèle sur-dimensionné (par rapport au problème et/ou à la quantité d'exemples) ;
- ajouter des exemples.

Augmenter la base d'exemples

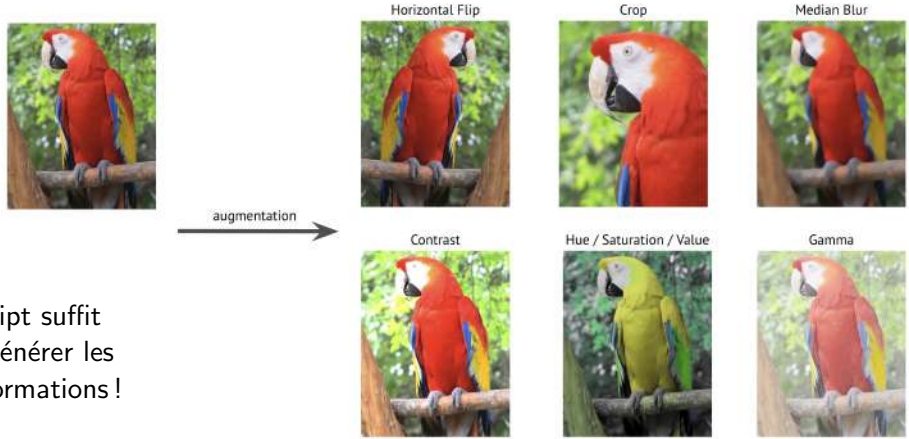
Solutions pour éviter le sur-apprentissage :

- s'arrêter à temps (comme déjà vu) ;
- ne pas utiliser un modèle sur-dimensionné (par rapport au problème et/ou à la quantité d'exemples) ;
- ajouter des exemples.

Si on n'a pas assez d'exemples, on peut :

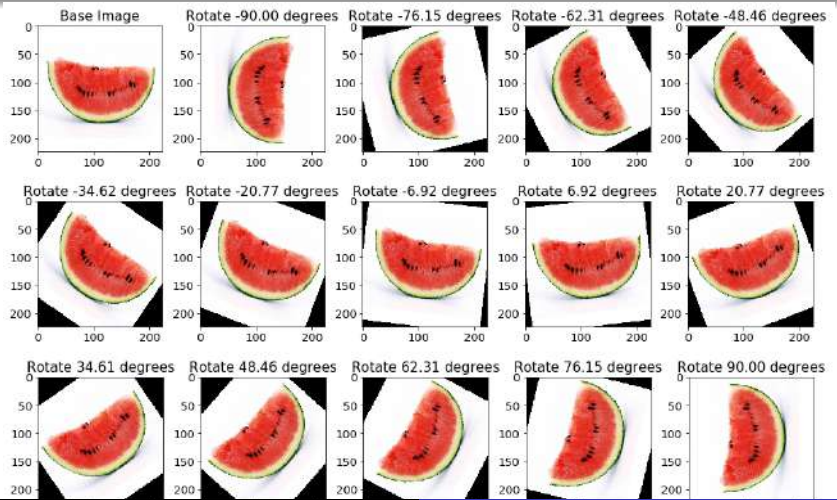
- en ajouter, mais étiqueter peut être très coûteux ;
- en créer par « augmentation », par exemple en inversant ou inclinant ou (...) une image ;
- en créer par synthèse ;
- ne pas « gaspiller » d'exemples pour la généralisation.

Augmenter la base d'exemples par transformations



Un script suffit pour générer les transformations !

Augmenter la base d'exemples par transformations



Augmenter la base d'exemples par synthèse

Exemple : détection (segmentation + reconnaissance) d'images réelles

Augmenter la base d'exemples par synthèse

Idem sur des images 3D générées :

Avantages :

- les images sont générées par synthèse
  $\Rightarrow$ on en a autant qu'on le souhaite !
- la *bounding box* est déjà connue
  $\Rightarrow$ pas besoin de les dessiner à la main.

Remarque :
on peut faire de l'apprentissage « assisté » :

1. le réseau segmente et reconnaît ;
2. l'opérateur confirme les cas douteux.

Augmenter la base d'exemples par synthèse

Exemple de la conduite autonome :

- on commence à apprendre dans un simulateur (un jeu vidéo), on peut ainsi faire des milliards de km en toute sécurité ;
- on peaufine l'apprentissage avec une vraie voiture (dès qu'on estime que c'est raisonnablement sûr).

Augmenter la base d'exemples par synthèse

(ou comment se faire offrir *Grand Theft Auto* par son laboratoire)

Ne pas gaspiller d'exemples pour la généralisation

On a déjà vu qu'il faut des exemples distincts :

- pour apprendre : la base d'apprentissage
- pour tester la capacité à généraliser : la base de généralisation

$\Rightarrow$ on garde jusqu'à 40% des exemples étiquetés disponibles pour la généralisation...

... ce qui est dommage si on manque déjà d'exemples !

Ne pas gaspiller d'exemples pour la généralisation

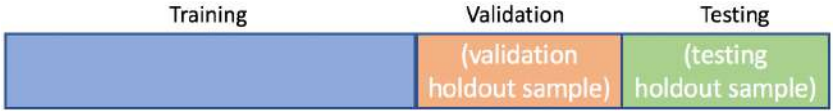
On a déjà vu qu'il faut des exemples distincts :

- pour apprendre : la base d'apprentissage
- pour tester la capacité à généraliser : la base de généralisation

$\Rightarrow$ on garde jusqu'à 40% des exemples étiquetés disponibles pour la généralisation...

... ce qui est dommage si on manque déjà d'exemples !

Et il faut une troisième base dite base de test, *à cause de fuites !*



Ne pas gaspiller d'exemples, pourquoi des fuites ?

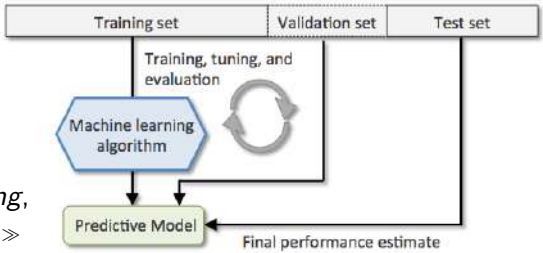
On a vu qu'il faut régler de nombreux hyperparamètres

$\Rightarrow$ on cherche les hyperparamères qui minimisent l'erreur en généralisation

$\Rightarrow$ le réseau devient optimisé pour notre base de généralisation :-(
... l'information a *fuité !*

Il faut une troisième base,
appelée « base de test »
pour évaluer le réseau.

Remarque : à l'ère du *machine learning*,
la mode est de parler de « validation »
plutôt que de « généralisation ».



Ne pas gaspiller d'exemples pour la généralisation

Pour ne pas gaspiller d'exemples pour la base de généralisation,
on ne va y mettre qu'un seul exemple !

$\Rightarrow$ on fait de la validation croisée d'un contre tous,
ou en anglais « *leave-one-out cross-validation* » (LOOCV).

Problème : le résultat est trop dépendant de l'exemple testé en généralisation.

Solution :

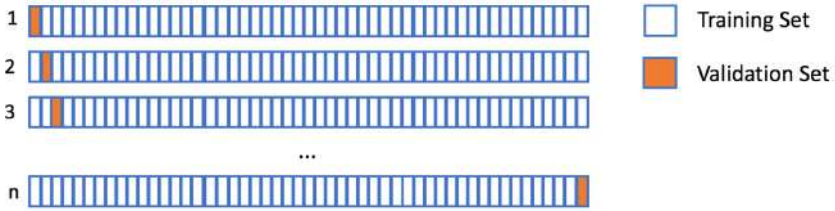
- on va recommencer pour chaque exemple,
- on fera la moyenne des résultats obtenus.

Bilan : on aura appris et testé sur tous les exemples sans danger.

Ne pas gaspiller d'exemples grâce à la LOOCV

Pour n exemples, on fait donc :

1. n apprentissages sur $n-1$ exemples,
2. n généralisations sur 1 exemple.



Problème : ça risque de prendre du temps !

Remarque : évidemment on a toujours la base de test !

Ne pas gaspiller d'exemples sans prendre trop de temps

Pour aller plus vite on va faire de la validation croisée à k blocs,
« *k-fold cross-validation* en anglais :

- on apprend sur $k-1$ bloc,
- on généralise sur un bloc.

Remarque : si $k = n$ on fait du LOOCV !

Réseaux neuronaux convolutifs

Réseau de neurones convolutifs = « réseau de neurones à convolution »
= « *convolutional neural network* »
= « CNN » = « ConvNet »

- réseau acyclique, ie *feed-forward*
- connexion entre les neurones inspirée du cortex visuel

→ empilage multicouche de perceptrons
dont le but est de pré-traiter de petites quantités d'informations

Applications : – reconnaissance d'image et vidéo,
– traitement du langage naturel, ...

Champs récepteurs du cortex visuel (Hubel et Wiesel, années 60)

Étude du cortex visuel de chats :

- projection de modèles de lumière sur un écran placé devant l'animal ;
- enregistrement de l'activité de neurones du cortex visuel ;
- écoute de l'activité sur un haut parleur.

Constatations :

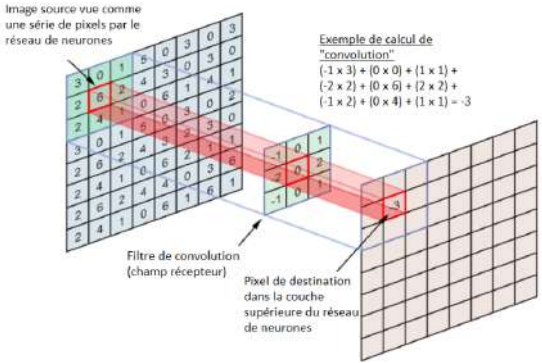
- neurones excités par des motifs de lumière (points, cercles, barres) ;
- neurones sensibles à l'orientation de la forme et au mouvement (4 min 22 s).

Champs récepteurs pour un réseau de neurones

Les champs récepteurs existent depuis longtemps en traitement d'image (notion de filtres).

La nouveauté avec les CNN est que le champ récepteur est trouvé par apprentissage.

Ainsi on a des champs parfaitement adaptés au domaine, qu'un humain n'aurait pas pu trouver sans une connaissance parfaite du problème.



Avantage des CNN par rapport aux MLP

Les avantages des CNN :

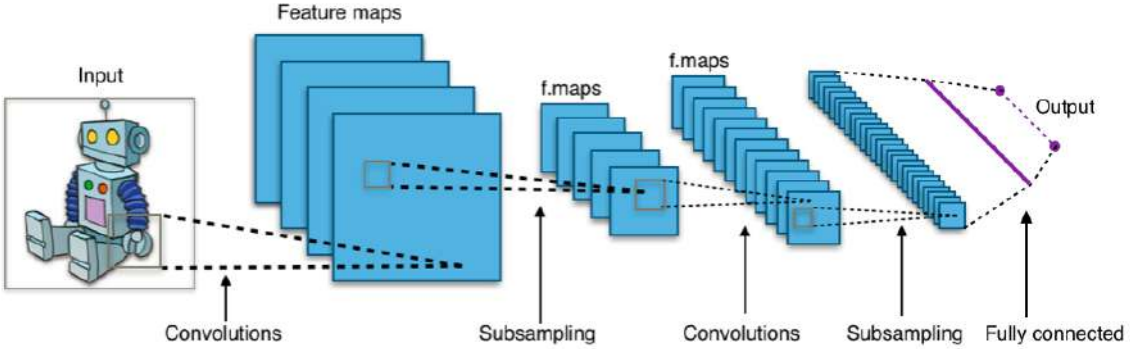
- invariance du traitement par translation
- relativement peu de pré-traitement
  $\Rightarrow$ moins d'intervention humaine (programmation)
  $\Rightarrow$ adaptation des filtres par apprentissage (non supervisé)

Comme pour le cortex visuel, on va détecter différentes « choses »
$\Rightarrow$ on va avoir besoin de plusieurs champs récepteurs
$\Rightarrow$ on va créer autant de « cartes » que l'on veut de champs différents.

Ces « cartes » sont appelées des « *feature maps* ».

Exemple d'architecture d'un réseau convolutif

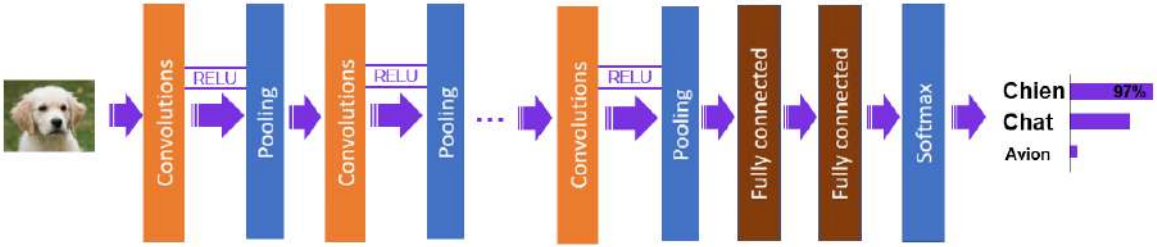
Exemple :



Attention : beaucoup de paramètres (poids) et d'hyperparamètres.

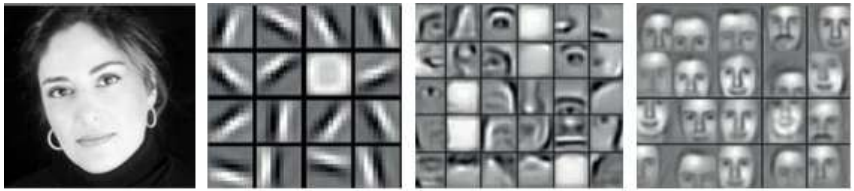
Exemple d'architecture d'un réseau convolutif

Il y a beaucoup de couches ⇒ on utilise une représentation plus synthétique :



Remarques : Les couches dites « *fully connected* » sont des couches de MLP classiques ; on verra les couches « *pooling* » et « *softmax* » plus loin.

Exemple de convolution pour l'apprentissage profond pour les visages

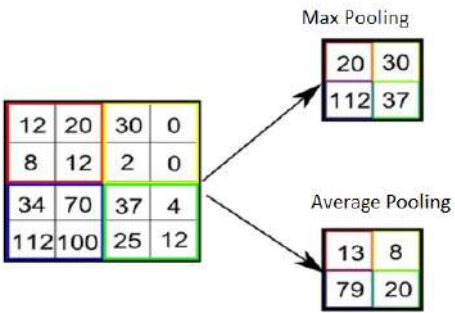


Attention :

- chacune des 3 images de droites représentent un ensemble de « *feature maps* », par exemple 16 « *feature maps* » pour « *edges* » ;
- tous les champs récepteurs d'une même « *feature maps* » sont identiques ce qui fait d'autant moins de paramètres (ie de poids à trouver).

Couche de « *polling* » aka « *subsampling* » aka échantillonnage

Il faut réduire la taille pour ne pas faire exploser le nombre de paramètres, par exemple en prenant la valeur maximale d'une zone, ou valeur la moyenne :



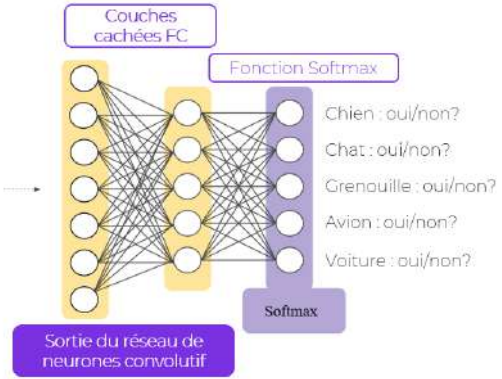
Détails des calculs :
max(12, 20, 8, 12) = 20
max(30, 0, 0, 2, 0) = 30
max(34, 70, 112, 100) = 112
max(37, 4, 25, 12) = 30

Détails des calculs :
(12 + 20 + 8 + 12) / 4 = 13
...

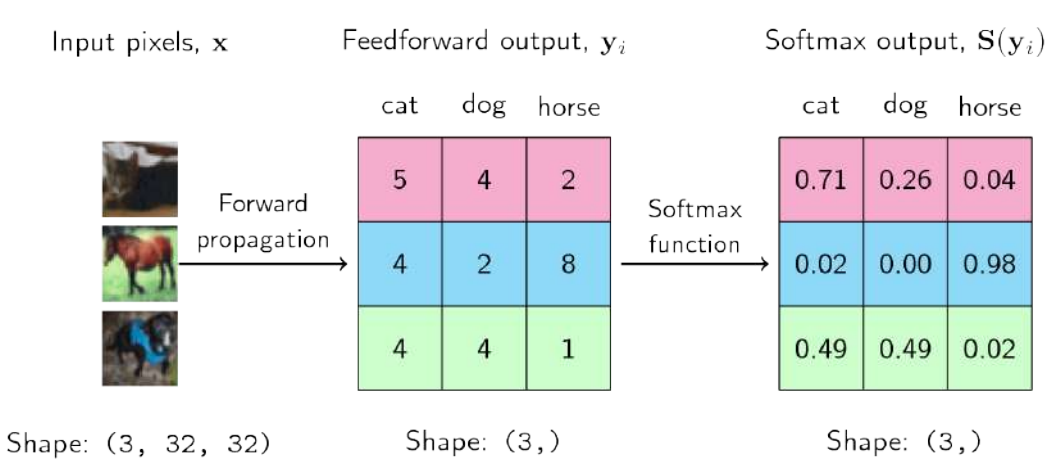
Couche de « *softmax* » : principe

La fonction « *softmax* » permet d'associer à chaque sortie du réseau un score, ensuite transformé en probabilité d'appartenir à une classe, par exemple :

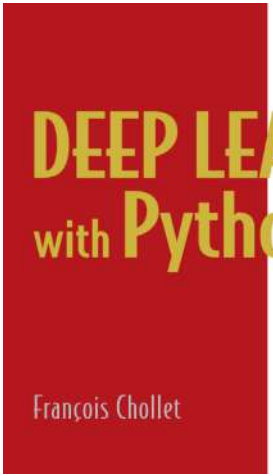
Classe	Probabilité
Chien	0,001
Chat	0,04
Grenouille	0,008
Avion	0,95
Voiture	0,001



Couche de « *softmax* » : exemple



Un bon livre sur Keras par François Chollet



Exemple sur la base MNIST (approche basique sans CNN)

```
from keras import layers
from keras import models

from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Exemple de programme sur les chiens et chats (CNN)

```
from keras import layers
from keras import models

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Topologie correspondant au code précédent

```
>>> model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
maxpooling2d_1 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
maxpooling2d_2 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
maxpooling2d_3 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
maxpooling2d_4 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dense_2 (Dense)	(None, 1)	513

```
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Exemple de photos pouvant être traitées

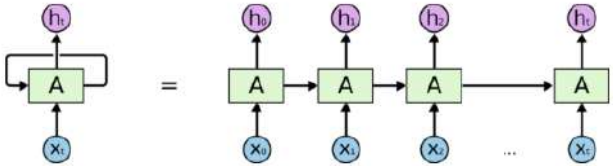






Réseaux profonds récurrents

Pour apprendre des séquences :

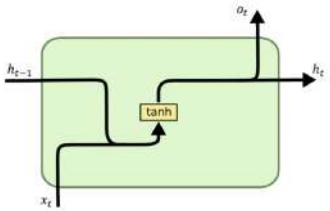


Principaux modèles :

- Recurrent Neural Network (RNN) avec 3 matrices de poids ;
- Long-Short Term Memory (LSTM) avec 4 matrices de poids ;
- Gated Recurrent Unit (GRU) avec 3 matrices de poids ;

Recurrent Neural Network / RNN

x_t : input vector $(m \times 1)$.
h_t : hidden layer vector $(n \times 1)$.
o_t : output vector $(n \times 1)$.
b_h : bias vector $(n \times 1)$.
U, W : parameter matrices $(n \times m)$.
V : parameter matrix $(n \times n)$.
σ_h, σ_y : activation functions.

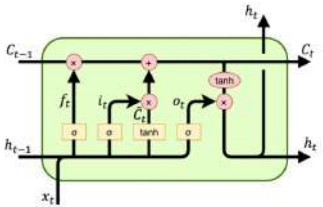


$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h)$$

Long-Short Term Memory / LSTM

h_t , C_t : hidden layer vectors.
x_t : input vector.
b_f , b_i , b_c , b_o : bias vector.
W_f , W_i , W_c , W_o : parameter matrices.
σ , $\tanh$: activation functions.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

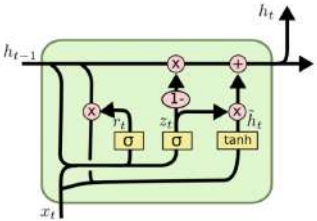
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

Gated Recurrent Unit (GRU)

h_t : hidden layer vectors.
x_t : input vector.
b_z , b_r , b_h : bias vector.
W_z , W_r , W_h : parameter matrices.
σ , tanh : activation functions.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$