

# Apprentissage par renforcement

Philippe Preux  
philippe.preux@univ-lille.fr

École d'Automne en Intelligence Artificielle, 2019

<https://philippe.preux.github.io/talks/IA2.pdf>



# Motivations

- ▶ Comment un agent autonome peut-il apprendre à réaliser une tâche dans un environnement qu'il ne connaît pas parfaitement ?

# Motivations

- ▶ Comment un agent autonome peut-il apprendre à réaliser une tâche dans un environnement qu'il ne connaît pas parfaitement ?
- ▶ Apprendre à faire est plus facile à programmer que faire.  
Illustration : aux échecs, Alpha Zero >> Deep Blue.

# Motivations

## Applications

- ▶ jeux vidéos, jeux de plateau
- ▶ contrôle de robots
- ▶ contrôle de smart-grid
- ▶ contrôle alimentation électrique serveurs
- ▶ gestion de flotte de véhicules
- ▶ ...

L'apprentissage par renforcement est à l'intersection de nombreux domaines :

- ▶ psychologie (analyse scientifique du comportement)
- ▶ informatique
- ▶ mathématiques appliquées (probabilités, statistiques, optimisation)
- ▶ économie
- ▶ automatique
- ▶ neurosciences
- ▶ apprentissage automatique numérique
- ▶ contrôle optimal

## Très bref historique

- ▶ depuis fin XIX<sup>e</sup> siècle : étude de l'adaptation du comportement animal (Thorndike, Pavlov, Skinner, Tolmann, Rescorla-Wagner, Staddon, ...)
- ▶ vers 1955 : naissance de l'IA ; cybernétique ; invention du neurone formel.
- ▶ 1984 : R.S. Sutton défend sa thèse de doctorat où il formalise l'apprentissage du comportement animal avec des modèles algorithmiques. Il y introduit les notions essentielles de l'AR.
- ▶ 1986 : re-découverte d'une méthode permettant d'entraîner un réseau de neurones.
- ▶ vers 1990 : combinaison réseau de neurones et AR ; diverses applications dont Backgammon.
- ▶ vers 2010 : apprentissage profond.
- ▶ 2016 : victoire d'AlphaGo contre Lee Sedol.
- ▶ 2017 : Alpha Zero apprend à jouer tout seul au go.

# Plan

- ▶ Outilage (rappels ou brève introduction) :
  - ▶ régression,
  - ▶ réseaux de neurones
  - ▶ descente de gradient
- ▶ Les problèmes de décision de Markov
- ▶ Planification
- ▶ Apprentissage par renforcement
- ▶ Conclusion et enjeux du domaine



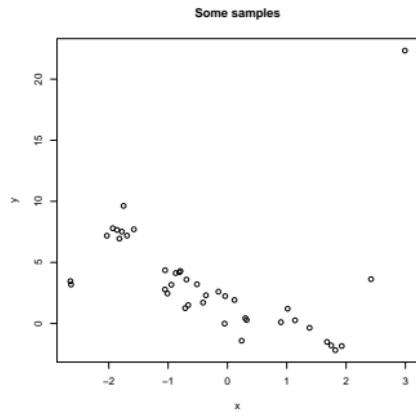
# Le problème de régression

- ▶ On considère un espace de données  $\mathcal{X} = \mathbb{R}^P$ .
- ▶ Chaque donnée  $x \in \mathcal{X}$  est associée à une étiquette  $y(x) \in \mathbb{R}$ .
- ▶ On dispose d'un ensemble de  $N$  paires  $(x, y)_i$  (exemples).
- ▶ Question : trouver un moyen de prédire  $y(x)$  pour tout  $x \in \mathcal{X}$

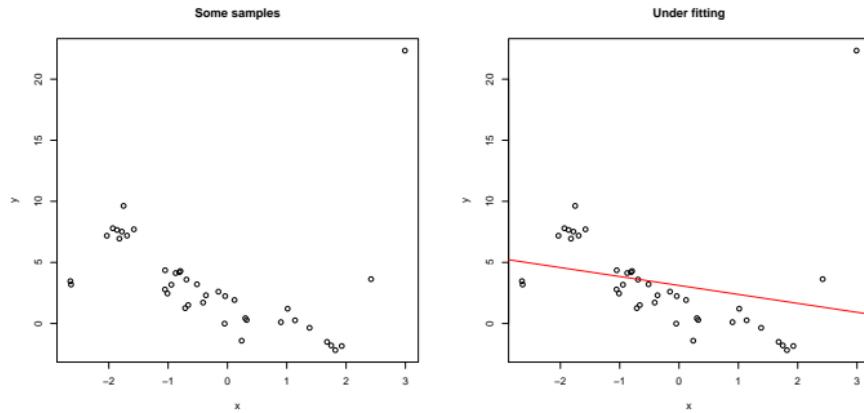
# Le problème de régression

- ▶ On considère un espace de données  $\mathcal{X} = \mathbb{R}^P$ .
  - ▶ Chaque donnée  $x \in \mathcal{X}$  est associée à une étiquette  $y(x) \in \mathbb{R}$ .
  - ▶ On dispose d'un ensemble de  $N$  paires  $(x, y)$ ; (exemples).
  - ▶ Question : trouver un moyen de prédire  $y(x)$  pour tout  $x \in \mathcal{X}$
- 
- ▶ On suppose que  $y = f + \text{bruit}$ .
  - ▶ On cherche un modèle permettant d'estimer ce  $f(x)$ .
  - ▶ Beaucoup de méthodes ont été inventées et continuent à l'être.
  - ▶ Modèle paramétré :
    - ▶ nombre de paramètres fixés (*a priori* ou non);
    - ▶ paramètres de différentes natures : scalaire, vecteur, matrice, fonction.

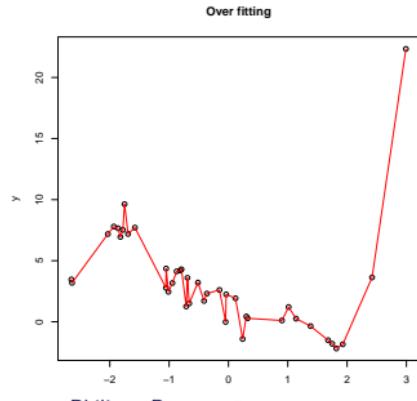
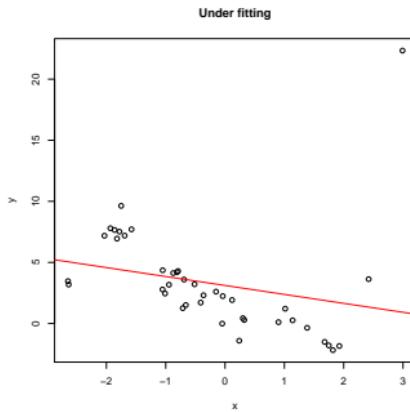
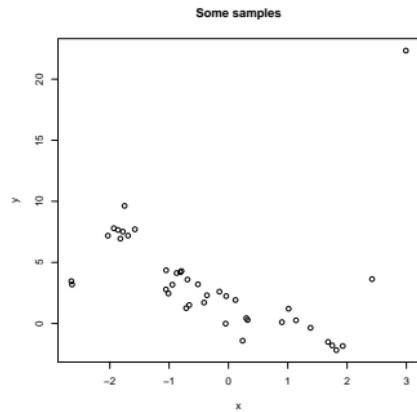
# Le problème de régression



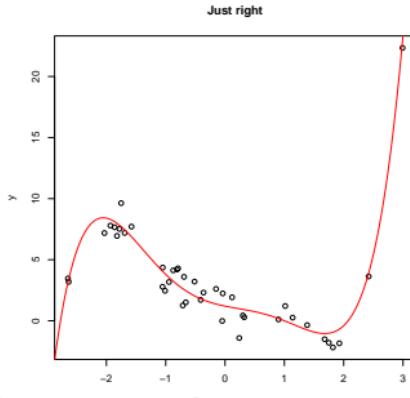
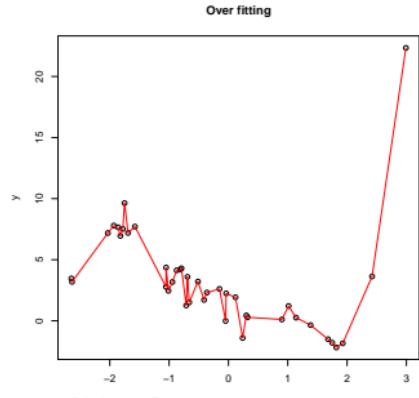
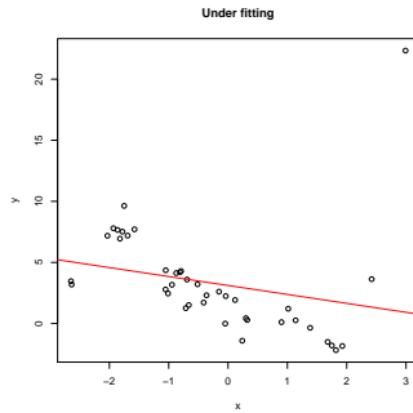
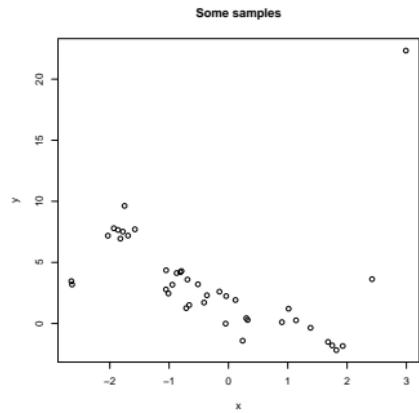
# Le problème de régression



# Le problème de régression



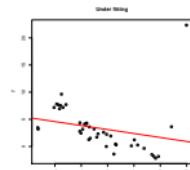
# Le problème de régression



# Le problème de régression

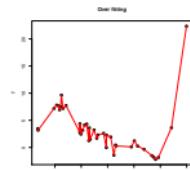
Sous-ajustement/apprentissage (*under-fitting*) : modèle trop simple pour  $y$ .

Le modèle n'a pas assez de paramètres pour s'ajuster à la complexité des données.

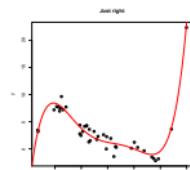


Sur-ajustement/apprentissage (*over-fitting*) : modèle trop complexe pour  $y$ .

Le modèle a trop de paramètres pour la complexité du jeu de données.



Le bon modèle retire juste le bruit.



# Le problème de régression

Parmi les techniques existantes, quelques mots sur les réseaux de neurones formels.

# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$

# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$

# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j\neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .

# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone

# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

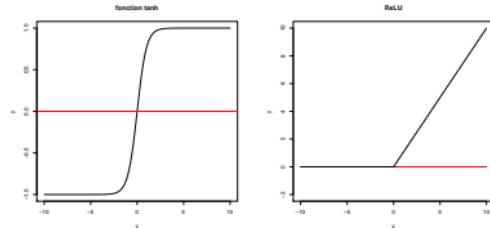
- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi\left(\sum_{j=0}^{j=P} \theta_j x_j\right)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone

# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone
- ▶  $\varphi$  est la fonction d'activation :

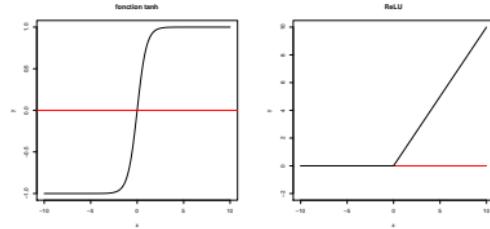


# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone
- ▶  $\varphi$  est la fonction d'activation :
  - ▶ linéaire

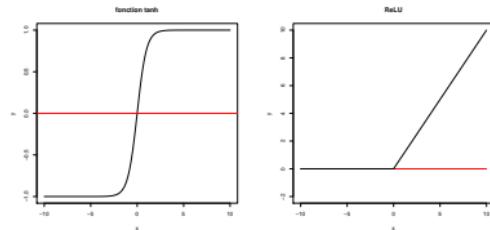


# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone
- ▶  $\varphi$  est la fonction d'activation :
  - ▶ linéaire
  - ▶ logistique ou tanh

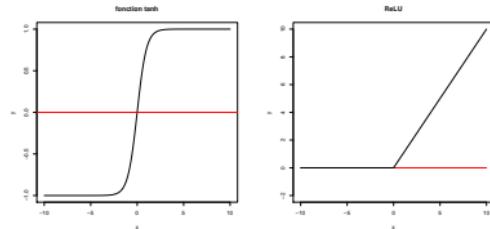


# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone
- ▶  $\varphi$  est la fonction d'activation :
  - ▶ linéaire
  - ▶ logistique ou tanh
  - ▶ fonction à base radiale (RBF)

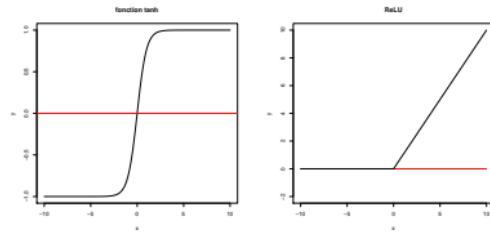


# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone
- ▶  $\varphi$  est la fonction d'activation :
  - ▶ linéaire
  - ▶ logistique ou tanh
  - ▶ fonction à base radiale (RBF)
  - ▶  $\text{ReLU} = \max(0, x)$

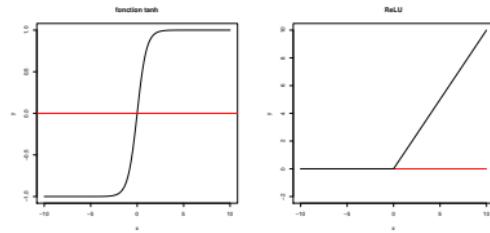


# Réseaux de neurones artificiels / formels

## Le neurone

Unité élémentaire : le neurone formel = le perceptron :

- ▶ une donnée (= un vecteur réel)  $\in \mathbb{R}^P$
- ▶ un neurone est une fonction réelle :  $f : \mathbb{R}^{P+1} \rightarrow \mathbb{R}$
- ▶  $x \mapsto f(x) = \varphi(\sum_{j=0}^{j=P} \theta_j x_j)$   
où  $x_{j \neq 0}$  est le  $j^{\text{th}}$  élément du vecteur  $x$  et  $x_0 = 1$ .
- ▶ les  $x_{j \neq 0}$  : entrées du neurone
- ▶  $\theta_j$  sont des poids (synaptiques) = paramètres du neurone
- ▶  $\varphi$  est la fonction d'activation :
  - ▶ linéaire
  - ▶ logistique ou tanh
  - ▶ fonction à base radiale (RBF)
  - ▶ ReLU =  $\max(0, x)$
  - ▶ ...



# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.
- ▶ (vu plus haut) on suppose que : valeur =  $f(\text{donnée}) + \text{bruit}$ .

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.
- ▶ (vu plus haut) on suppose que : valeur =  $f(\text{donnée}) + \text{bruit}$ .
- ▶ objectif : le perceptron représente aussi précisément que possible  $f$ .

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.
- ▶ (vu plus haut) on suppose que : valeur =  $f(\text{donnée}) + \text{bruit}$ .
- ▶ objectif : le perceptron représente aussi précisément que possible  $f$ .
- ▶ la sortie du perceptron =  $\varphi(\sum_{j=0}^{j=P} \theta_j x_j)$

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.
- ▶ (vu plus haut) on suppose que : valeur =  $f(\text{donnée}) + \text{bruit}$ .
- ▶ objectif : le perceptron représente aussi précisément que possible  $f$ .
- ▶ la sortie du perceptron =  $\varphi(\sum_{j=0}^{j=P} \theta_j x_j)$
- ▶ ↵ ajuster les  $\theta_j$  pour que la sortie du perceptron devienne =  $f(\text{entrée})$ .

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.
- ▶ (vu plus haut) on suppose que : valeur =  $f(\text{donnée}) + \text{bruit}$ .
- ▶ objectif : le perceptron représente aussi précisément que possible  $f$ .
- ▶ la sortie du perceptron =  $\varphi(\sum_{j=0}^{j=P} \theta_j x_j)$
- ▶ ↵ ajuster les  $\theta_j$  pour que la sortie du perceptron devienne =  $f(\text{entrée})$ .
- ▶ ajustement par les moindres-carrés : trouver  $\theta^* = \arg \min_{\theta} \sum_i (\varphi(\sum_{j=0}^{j=P} \theta_j x_{i,j}) - y_i)^2$

# Réseaux de neurones artificiels / formels

## Entraîner un neurone

- ▶ un neurone est entraîné avec un ensemble d'exemples : (donnée, valeur)
- ▶ quand la donnée est placée en entrée du neurone, on veut que sa sortie soit cette valeur.
- ▶ (vu plus haut) on suppose que : valeur =  $f(\text{donnée}) + \text{bruit}$ .
- ▶ objectif : le perceptron représente aussi précisément que possible  $f$ .
- ▶ la sortie du perceptron =  $\varphi(\sum_{j=0}^{j=P} \theta_j x_j)$
- ▶ ↵ ajuster les  $\theta_j$  pour que la sortie du perceptron devienne =  $f(\text{entrée})$ .
- ▶ ajustement par les moindres-carrés : trouver  $\theta^* = \arg \min_{\theta} \sum_i (\varphi(\sum_{j=0}^{j=P} \theta_j x_{i,j}) - y_i)^2$
- ▶ il existe différents algorithmes d'optimisation pour cela. En général, on effectue une descente de gradient.

# Réseaux de neurones artificiels / formels

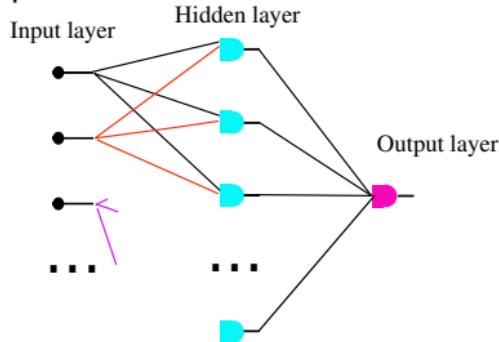
## Du neurone au réseau de neurones

- ▶ un perceptron peut seulement représenter  $\varphi$
- ▶ pour représenter d'autres fonctions, on utilise un ensemble de neurones
- ▶ fondement : théorème : toute fonction réelle, bornée, continue et définie sur un compact peut être approximée avec une précision arbitraire par une combinaison finie de fonctions de base.  
Exemples de fonctions de base : fonction logistique, tanh, RBF.

# Réseaux de neurones artificiels / formels

## Du neurone au réseau de neurones

- prendre  $N$  neurones et sommer leurs sorties pondérées.



- la sortie du réseau de neurones est  $\varphi_o(\sum_{k=0}^{K=N} \theta_k \varphi_k(\sum_{j=0}^{J=P} \theta_j x_j))$
- à nouveau, on cherche les  $\theta_j^*$  et  $\theta_k^*$  minimisant les moindres-carrés.  
Rétro-propagation du gradient de l'erreur.
- « Perceptrons multi-couches »

# Réseaux de neurones artificiels / formels

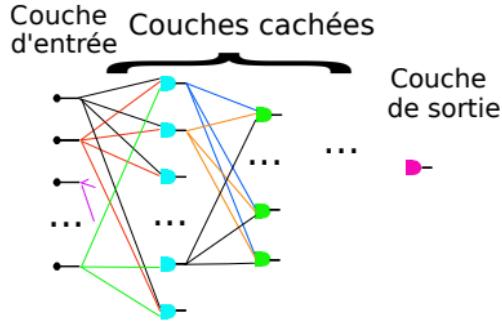
## Du neurone au réseau de neurones

- avoir plus d'une couche cachée ne permet pas de représenter une plus grande variété de fonctions.

# Réseaux de neurones artificiels / formels

## Du neurone au réseau de neurones

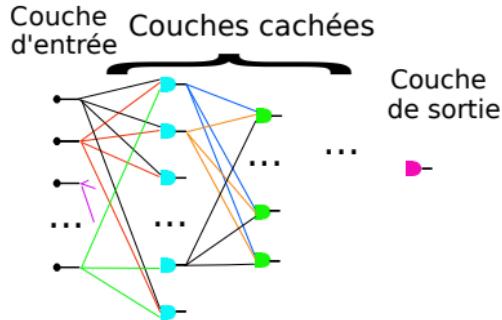
- ▶ avoir plus d'une couche cachée ne permet pas de représenter une plus grande variété de fonctions.
- ▶ mais c'est très utile d'un pdv pratique : au lieu d'utiliser une couche cachée très large, on utilise plusieurs couches cachées plus petites.



# Réseaux de neurones artificiels / formels

## Du neurone au réseau de neurones

- ▶ avoir plus d'une couche cachée ne permet pas de représenter une plus grande variété de fonctions.
- ▶ mais c'est très utile d'un pdv pratique : au lieu d'utiliser une couche cachée très large, on utilise plusieurs couches cachées plus petites.

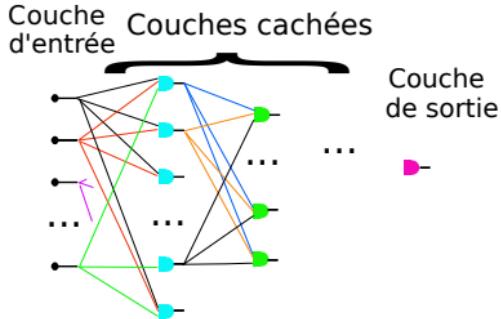


- ▶ Rétro-propagation du gradient de l'erreur.

# Réseaux de neurones artificiels / formels

## Du neurone au réseau de neurones

- ▶ avoir plus d'une couche cachée ne permet pas de représenter une plus grande variété de fonctions.
- ▶ mais c'est très utile d'un pdv pratique : au lieu d'utiliser une couche cachée très large, on utilise plusieurs couches cachées plus petites.

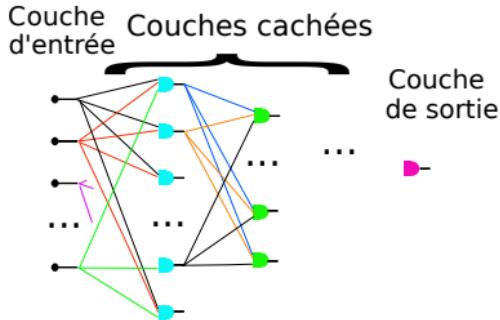


- ▶ Rétro-propagation du gradient de l'erreur.
- ▶ Plusieurs applications très pratiques : reconnaissance des codes postaux, montant des chèques, ...

# Réseaux de neurones artificiels / formels

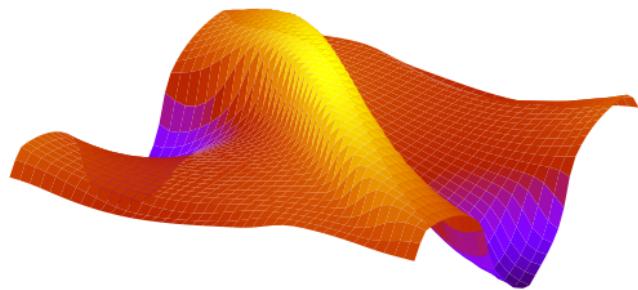
## Du neurone au réseau de neurones

- ▶ avoir plus d'une couche cachée ne permet pas de représenter une plus grande variété de fonctions.
- ▶ mais c'est très utile d'un pdv pratique : au lieu d'utiliser une couche cachée très large, on utilise plusieurs couches cachées plus petites.



- ▶ Rétro-propagation du gradient de l'erreur.
- ▶ Plusieurs applications très pratiques : reconnaissance des codes postaux, montant des chèques, ...
- ▶ Beaucoup de problèmes pratiques si le nombre de couches cachées supérieur à 10. Résolution de ces problèmes  $\leadsto$  révolution de l'apprentissage profond (AP).

# Optimisation de fonctions



# Optimisation de fonctions

Supposons que vous vouliez trouver le minimum d'une fonction à une variable.

- ▶ Vous ne connaissez pas la fonction.

# Optimisation de fonctions

Supposons que vous vouliez trouver le minimum d'une fonction à une variable.

- ▶ Vous ne connaissez pas la fonction.
- ▶ La fonction est continue.

# Optimisation de fonctions

Supposons que vous vouliez trouver le minimum d'une fonction à une variable.

- ▶ Vous ne connaissez pas la fonction.
- ▶ La fonction est continue.
- ▶ Vous pouvez proposer une valeur de la variable et obtenir la valeur de la fonction en ce point.

# Optimisation de fonctions

Supposons que vous vouliez trouver le minimum d'une fonction à une variable.

- ▶ Vous ne connaissez pas la fonction.
- ▶ La fonction est continue.
- ▶ Vous pouvez proposer une valeur de la variable et obtenir la valeur de la fonction en ce point.
- ▶ Comment faites-vous ?

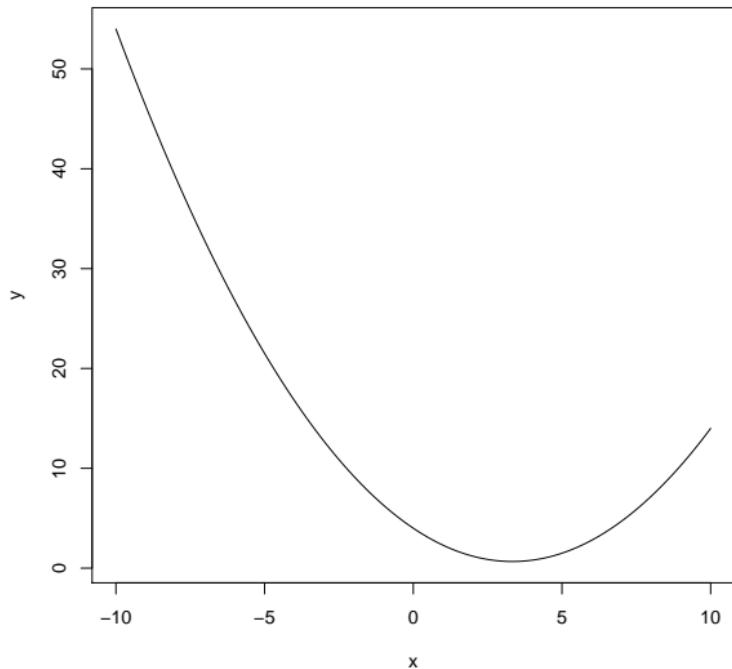
# Optimisation de fonctions

Supposons que vous vouliez trouver le minimum d'une fonction à une variable.

- ▶ Vous ne connaissez pas la fonction.
- ▶ La fonction est continue.
- ▶ Vous pouvez proposer une valeur de la variable et obtenir la valeur de la fonction en ce point.
- ▶ Comment faites-vous ?
- ▶ Il existe des tas de méthodes.

# Optimisation de fonctions

**fonction à minimiser**



# Optimisation de fonctions

## Descente de gradient stochastique

- ▶ Méthode générale de minimisation d'une fonction ayant un nombre de variables quelconque.

# Optimisation de fonctions

## Descente de gradient stochastique

- ▶ Méthode générale de minimisation d'une fonction ayant un nombre de variables quelconque.
- ▶ Trouve un minimum *local*.

# Optimisation de fonctions

## Descente de gradient stochastique

- ▶ Méthode générale de minimisation d'une fonction ayant un nombre de variables quelconque.
- ▶ Trouve un minimum *local*.
- ▶ Équation fondamentale de l'algorithme :

$$x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$$

soit pour une fonction ayant une seule variable :

$$x_{t+1} \leftarrow x_t - \eta f'(x_t)$$

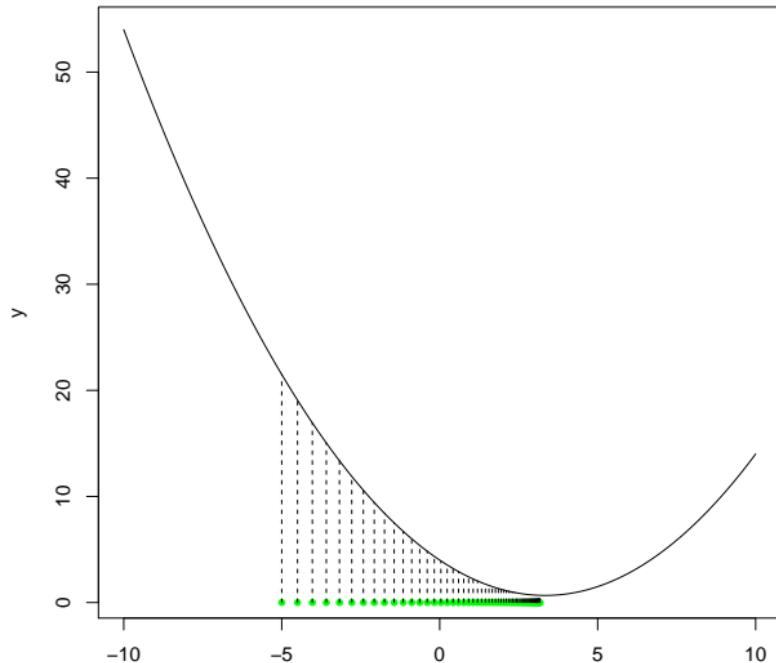
avec

$$f'(x) = \frac{d}{d x} f$$

# Optimisation de fonctions

## Descente de gradient stochastique

Itération finale



# Optimisation de fonctions

## Descente de gradient stochastique

- ▶ Réaliser une DGS sur la fonction  
 $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

- ▶ définir une fonction qui calcule la valeur de  $f$  en  $x$
- ▶ calculer la dérivée de  $f(x)$
- ▶ définir une fonction qui calcule la dérivée de  $f$  en  $x$
- ▶ prendre  $\eta = 0.1$
- ▶ réaliser la DGS

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

- ▶ définir une fonction qui calcule la valeur de  $f$  en  $x$

```
f <- function (x) 0.2 * x^5 - 1.1 * x^3 + .7 * x^2 - x + 1.2
```

- ▶ calculer la dérivée de  $f(x)$
- ▶ définir une fonction qui calcule la dérivée de  $f$  en  $x$
- ▶ prendre  $\eta = 0.1$
- ▶ réaliser la DGS

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

- ▶ définir une fonction qui calcule la valeur de  $f$  en  $x$

```
f <- function (x) 0.2 * x^5 - 1.1 * x^3 + .7 * x^2 - x + 1.2
```

- ▶ calculer la dérivée de  $f(x)$

$$\frac{d f}{d x} = x^4 - 3.3x^2 + 1.4x - 1$$

- ▶ définir une fonction qui calcule la dérivée de  $f$  en  $x$

- ▶ prendre  $\eta = 0.1$

- ▶ réaliser la DGS

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

- ▶ définir une fonction qui calcule la valeur de  $f$  en  $x$

```
f <- function (x) 0.2 * x^5 - 1.1 * x^3 + .7 * x^2 - x + 1.2
```

- ▶ calculer la dérivée de  $f(x)$

$$\frac{df}{dx} = x^4 - 3.3x^2 + 1.4x - 1$$

- ▶ définir une fonction qui calcule la dérivée de  $f$  en  $x$

```
df <- function (x) x^4 - 3.3 * x^2 + 1.4 * x - 1
```

- ▶ prendre  $\eta = 0.1$

- ▶ réaliser la DGS

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

- ▶ définir une fonction qui calcule la valeur de  $f$  en  $x$

```
f <- function (x) 0.2 * x^5 - 1.1 * x^3 + .7 * x^2 - x + 1.2
```

- ▶ calculer la dérivée de  $f(x)$

$$\frac{df}{dx} = x^4 - 3.3x^2 + 1.4x - 1$$

- ▶ définir une fonction qui calcule la dérivée de  $f$  en  $x$

```
df <- function (x) x^4 - 3.3 * x^2 + 1.4 * x - 1
```

- ▶ prendre  $\eta = 0.1$

```
eta <- .1
```

- ▶ réaliser la DGS

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

- ▶ définir une fonction qui calcule la valeur de  $f$  en  $x$

```
f <- function (x) 0.2 * x^5 - 1.1 * x^3 + .7 * x^2 - x + 1.2
```

- ▶ calculer la dérivée de  $f(x)$

$$\frac{df}{dx} = x^4 - 3.3x^2 + 1.4x - 1$$

- ▶ définir une fonction qui calcule la dérivée de  $f$  en  $x$

```
df <- function (x) x^4 - 3.3 * x^2 + 1.4 * x - 1
```

- ▶ prendre  $\eta = 0.1$

```
eta <- .1
```

- ▶ réaliser la DGS

```
repeat {
  xt <- xt - eta * df (xt)
  if (abs (df (xt)) < 0.1)
    break
}
```

# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$

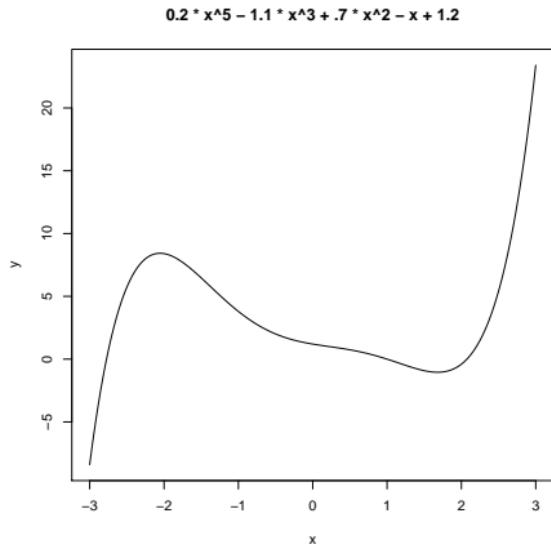
Tout mis bout à bout :

```
f <- function (x) 0.2 * x^5 - 1.1 * x^3 + .7 * x^2 - x + 1.2
df <- function (x) x^4 - 3.3 * x^2 + 1.4 * x - 1
xi <- runif (1, min = -2, max = 3)

eta <- .1
repeat {
  xi <- xi - eta * df (xi)
  if (abs (df (xi)) < 0.00001)
    break
}
```

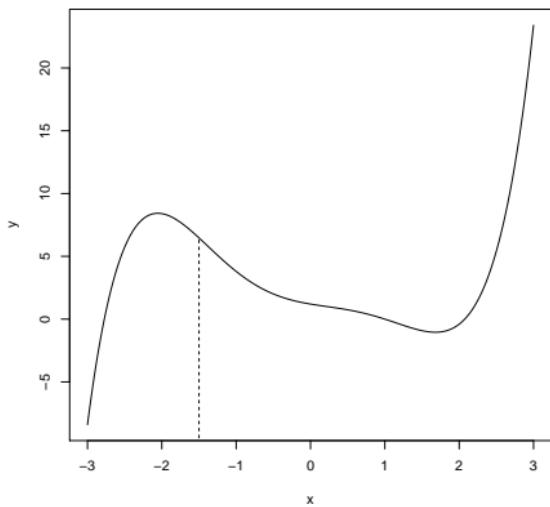
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



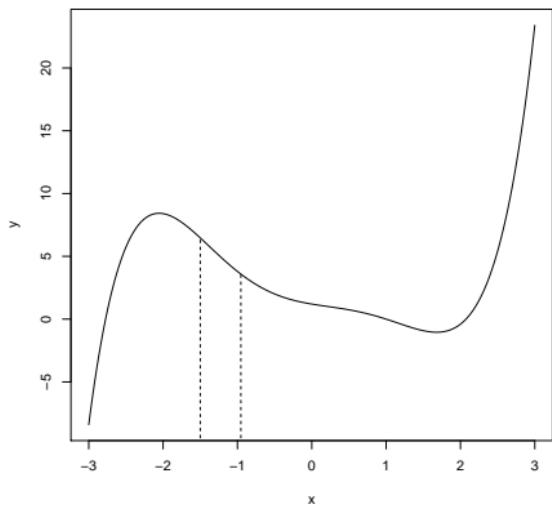
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



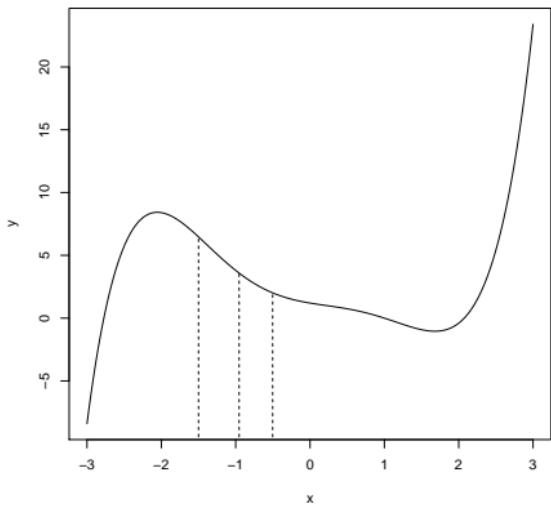
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



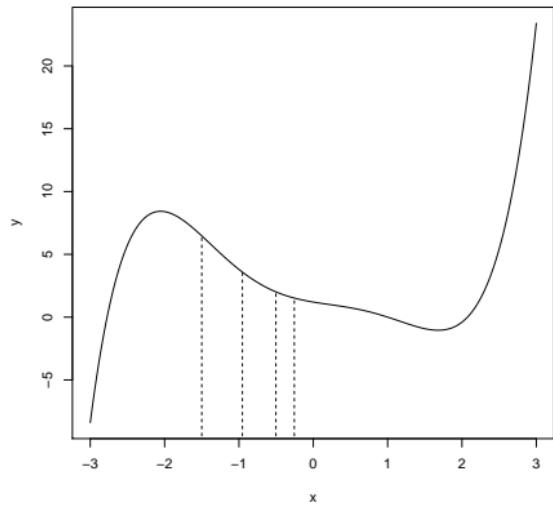
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



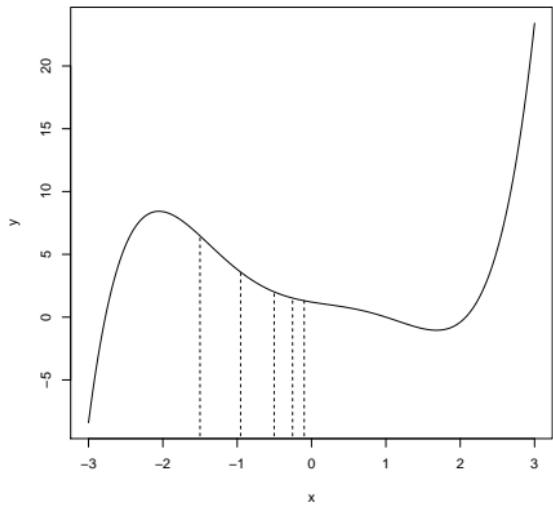
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



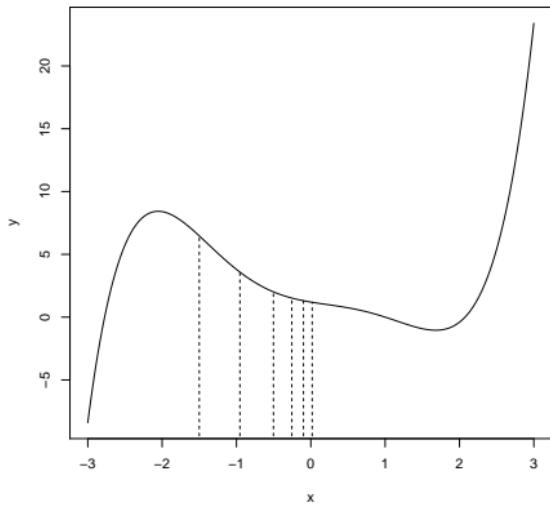
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



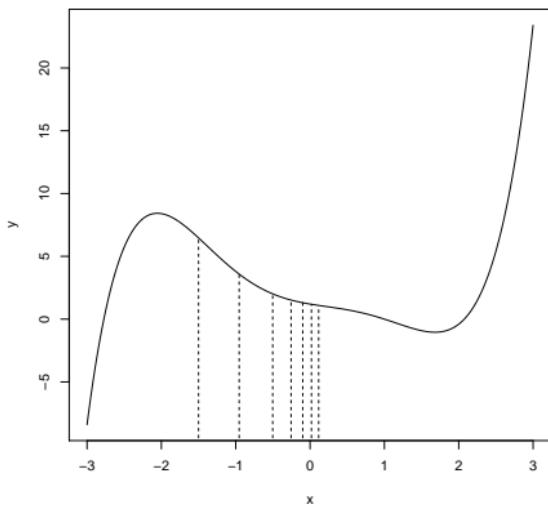
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



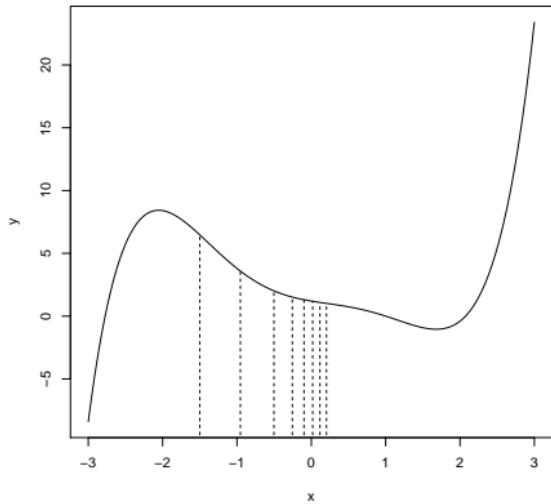
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



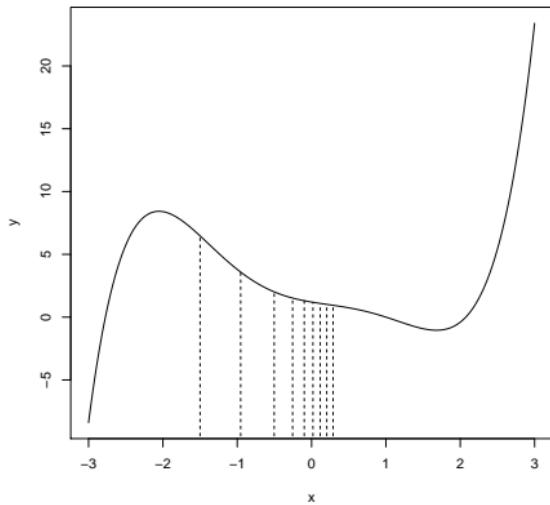
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



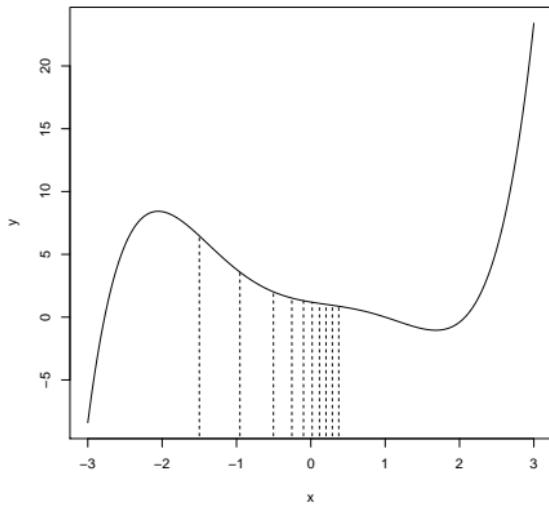
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



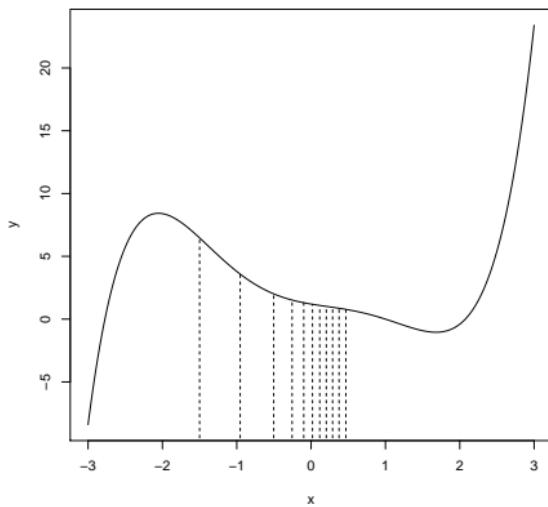
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



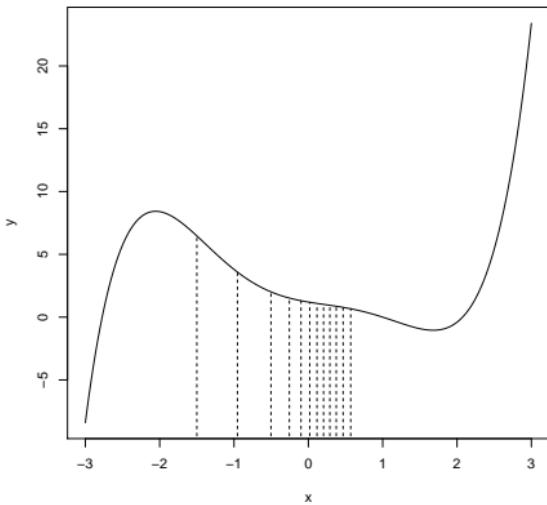
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



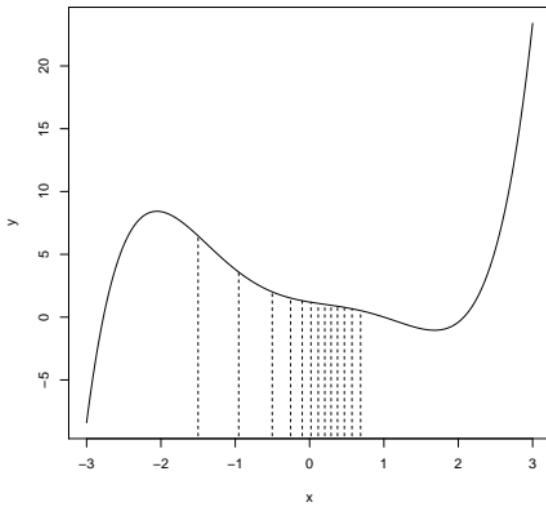
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



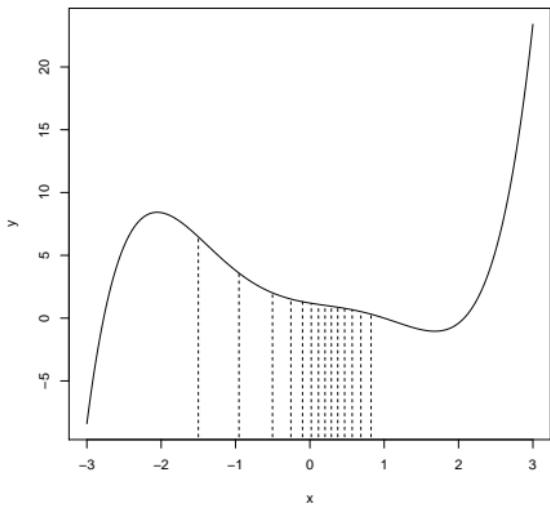
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



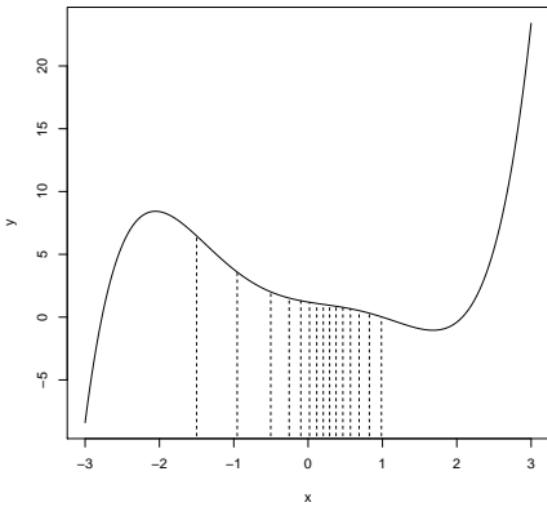
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



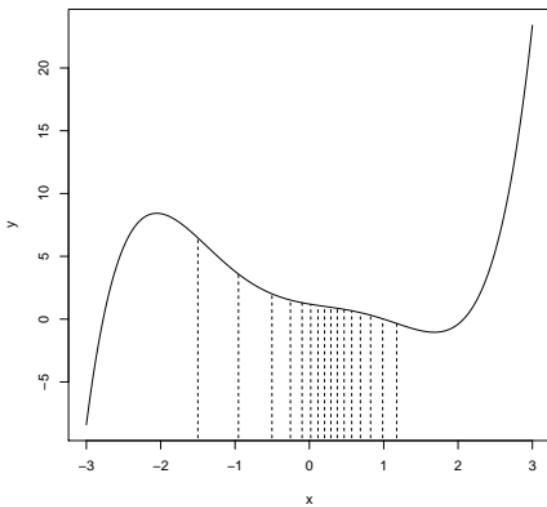
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



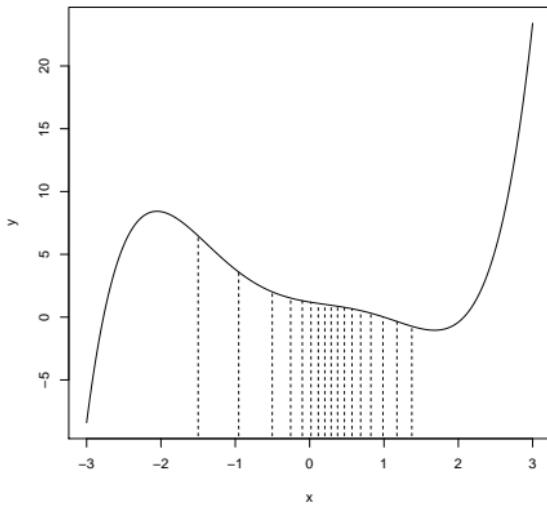
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



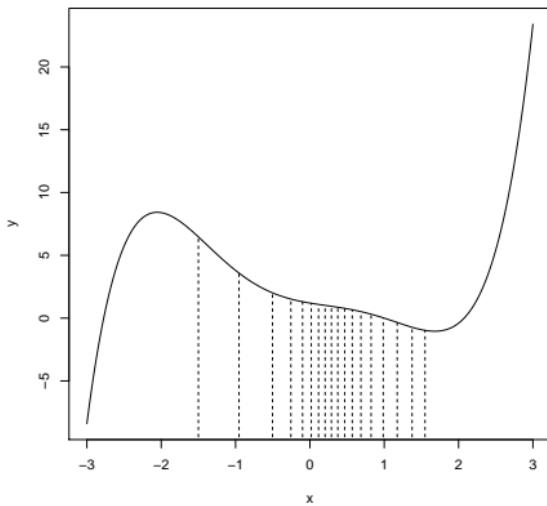
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



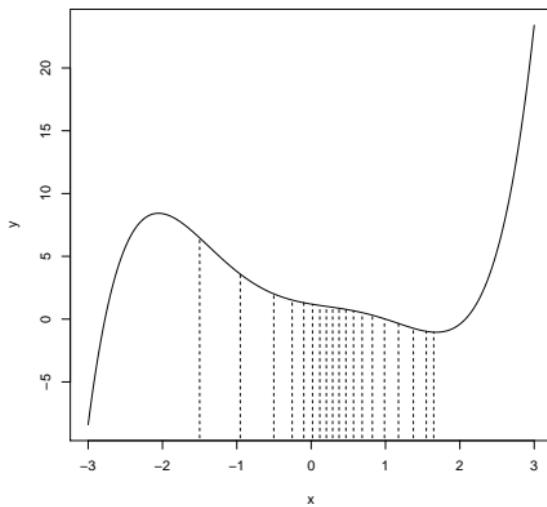
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



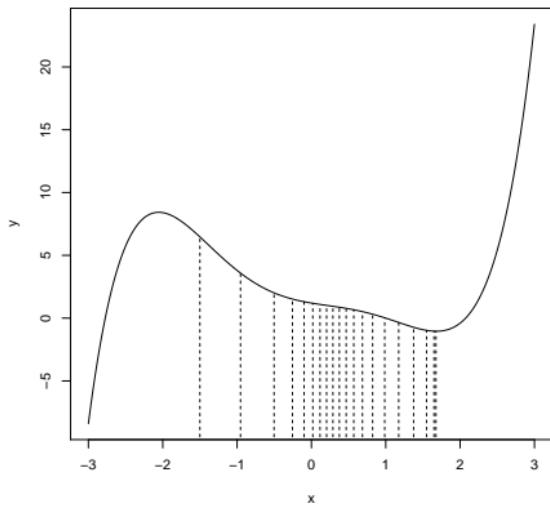
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



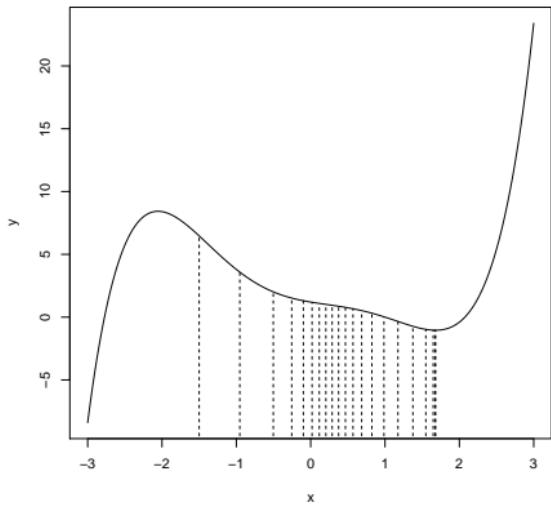
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



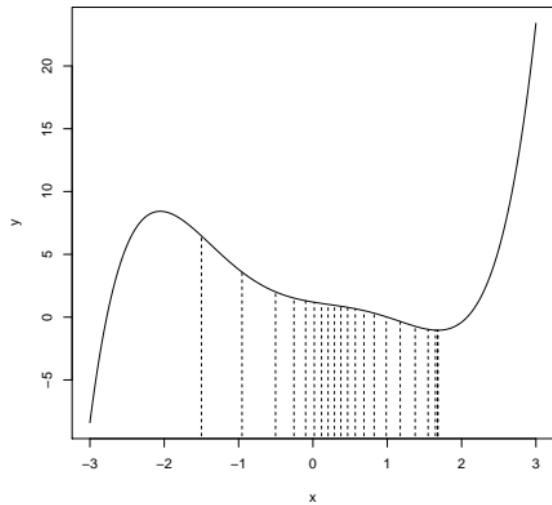
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



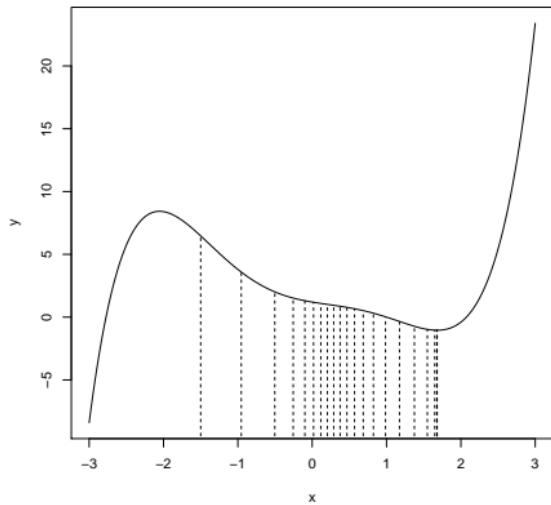
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



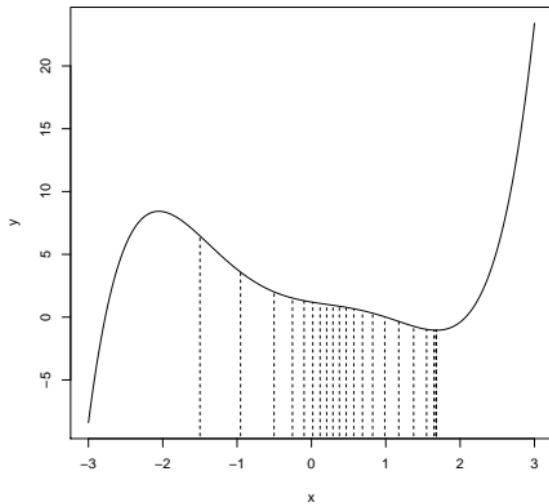
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



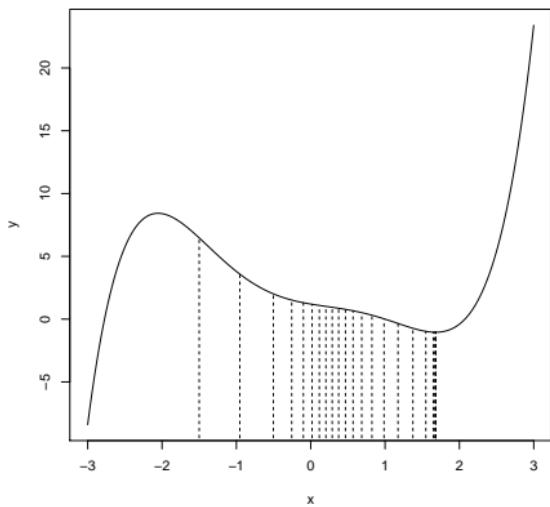
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



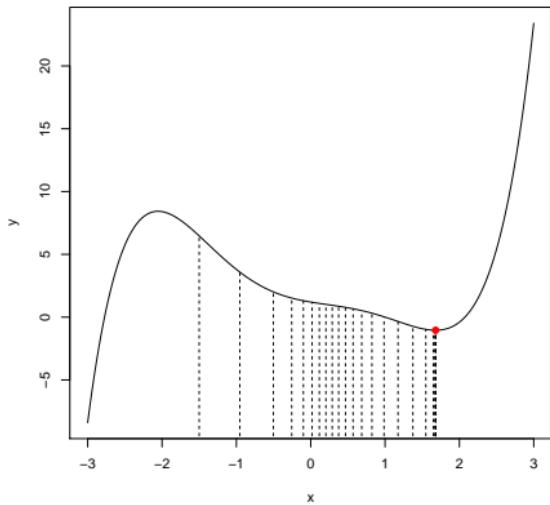
# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



# Optimisation de fonctions

Descente de gradient stochastique : minimiser  $f(x) = 0.2x^5 - 1.1x^3 + .7x^2 - x + 1.2$



# Descente de gradient stochastique

Exercice :

- ▶ Minimiser :

$$g(x) = (x + 4)(x + 2)(x + 1)(x - 1)(x - 3)(x - .5)(x + 1.5)/20$$

soit

$$g(x) =$$

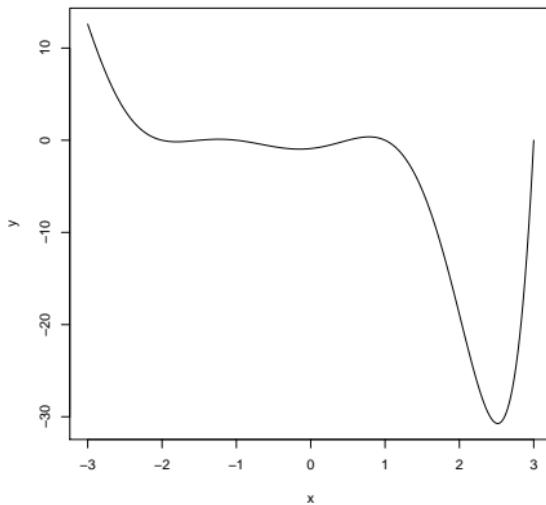
$$\frac{x^7}{20} + \frac{x^6}{5} - 0.4375x^5 - 2.0125x^4 - 0.4375x^3 + 2.7125x^2 + 0.825x - 0.9$$

- ▶ en prenant  $x \in [-3, 3]$

# Optimisation de fonctions

## Descente de gradient stochastique

$$g(x) = (x + 4)(x + 2)(x + 1)(x - 1)(x - 3)(x - .5)(x + 1.5) / 20$$



# Optimisation de fonctions

## Descente de gradient stochastique améliorée

- ▶ l'algorithme DGS vu précédemment trouve le minimum local le plus proche de son point de départ.

# Optimisation de fonctions

## Descente de gradient stochastique améliorée

- ▶ l'algorithme DGS vu précédemment trouve le minimum local le plus proche de son point de départ.
- ▶ Nombreux trucs pour améliorer ce comportement.

# Optimisation de fonctions

## Descente de gradient stochastique améliorée

- ▶ l'algorithme DGS vu précédemment trouve le minimum local le plus proche de son point de départ.
- ▶ Nombreux trucs pour améliorer ce comportement.
- ▶ DGS à départs multiples :  
on exécute plusieurs fois DGS en partant de  $x_0$  différents à chaque fois et on conserve le meilleur minimum trouvé.  
Exercice : programmer ce DGS à départ multiple. Appliquez-le à la fonction  $g$ . Trouvez-vous le bon minimum ?

# Optimisation de fonctions

## Descente de gradient stochastique améliorée

- ▶ l'algorithme DGS vu précédemment trouve le minimum local le plus proche de son point de départ.
- ▶ Nombreux trucs pour améliorer ce comportement.
- ▶ DGS à départs multiples :  
on exécute plusieurs fois DGS en partant de  $x_0$  différents à chaque fois et on conserve le meilleur minimum trouvé.  
Exercice : programmer ce DGS à départ multiple. Appliquez-le à la fonction  $g$ . Trouvez-vous le bon minimum ?
- ▶ C'est un problème qui s'amplifie quand le nombre de variables augmente.

# Descente de gradient stochastique

Pour réseau de neurones

- ▶ 1 perceptron :  $\theta^* = \arg \min_{\theta} \sum_i (\varphi(\sum_{j=0}^{j=P} \theta_j x_{i,j}) - y_i)^2$
- ▶ 1 réseau à une couche cachée :  

$$\theta^* = \arg \min_{\theta} \sum_i (\varphi_o(\sum_{k=0}^{k=N} \theta_k \varphi_k(\sum_{j=0}^{j=P} \theta_j x_j)) - y_i)^2$$
- ▶ 1 MLP : ...
- ▶ cas simple : le perceptron avec  $\varphi = Id$ .  
 Exercice : calculer le gradient de la fonction et appliquer la DGS.
- ▶ MLP : beaucoup plus compliqué : calcul du gradient de l'erreur pour chaque poids  $\theta$   
 ↵ algorithme de rétro-propagation du gradient de l'erreur [Werbos, 1974;  
 Rumelhart et al. 1986 ; Le Cun, 1986]

Nombreux problèmes pratiques résolus dans les années 2000  
 + Très nombreuses idées. } ↵ apprentissage profond.

# Les problèmes de décision de Markov

# Processus de décision de Markov

## Définition

Un processus de décision de Markov décrit la dynamique d'un système.

- ▶ ensemble d'instants :  $t \in \mathcal{T}$
- ▶ ensemble d'états :  $x \in \mathcal{X}$
- ▶ ensemble d'actions :  $a \in \mathcal{A}$
- ▶ fonction de transition :  $\mathcal{P}(x, a, x') = Pr(x'|x, a)$
- ▶ fonction de retour :  $\mathcal{R}(x, a, x') = \mathbb{E}(r(x'|x, a))$

Markov = l'état suivant ne dépend que de l'état et de l'action courants.

# Problèmes de décision de Markov

## Définition

Un problème de décision de Markov est un problème de contrôle d'un système dynamique.

- ▶ ensemble d'instants :  $t \in \mathcal{T}$
- ▶ ensemble d'états :  $x \in \mathcal{X}$
- ▶ ensemble d'actions :  $a \in \mathcal{A}$
- ▶ fonction de transition :  $\mathcal{P}(x, a, x') = Pr(x'|x, a)$
- ▶ fonction de retour :  $\mathcal{R}(x, a, x') = \mathbb{E}(r(x'|x, a))$
- ▶ fonction objectif :  $\zeta$
  
- ▶ **Objectif** : trouver une politique  $\pi^*$  qui optimise  $\zeta$ .
- ▶ Une politique  $\pi$  spécifie l'action à effectuer dans chacun des états.
  - ▶ déterministe :  $\pi(x) = a$
  - ▶ stochastique :  $\pi(x, a) \rightarrow Pr[a_t = a | x_t = x]$

# Problème de décision de Markov

## Autres notions

- ▶ état initial

# Problème de décision de Markov

## Autres notions

- ▶ état initial
- ▶ trajectoire : séquence d'états parcourus au fil du temps

# Problème de décision de Markov

## Autres notions

- ▶ état initial
- ▶ trajectoire : séquence d'états parcourus au fil du temps
- ▶ horizon : fini vs. infini

# Problème de décision de Markov

## Autres notions

- ▶ état initial
- ▶ trajectoire : séquence d'états parcourus au fil du temps
- ▶ horizon : fini vs. infini
- ▶  $\zeta$  les plus courantes :

# Problème de décision de Markov

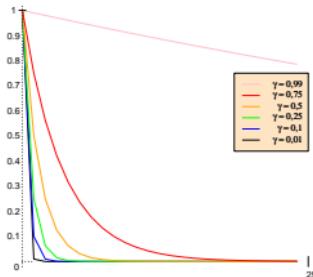
## Autres notions

- ▶ état initial
- ▶ trajectoire : séquence d'états parcourus au fil du temps
- ▶ horizon : fini vs. infini
- ▶  $\zeta$  les plus courantes :
  - ▶ horizon fini :  $\zeta = \sum_t r_t$ .

# Problème de décision de Markov

## Autres notions

- ▶ état initial
- ▶ trajectoire : séquence d'états parcourus au fil du temps
- ▶ horizon : fini vs. infini
- ▶  $\zeta$  les plus courantes :
  - ▶ horizon fini :  $\zeta = \sum_t r_t$ .
  - ▶ horizon fini ou infini :  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .

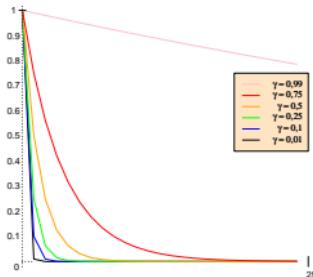


$\gamma^t$  garantit que  $|\sum_t \gamma^t r_t| < \infty$ .  
 Selon la valeur de  $\gamma$ , les retours futurs sont pris en compte sur un plus ou moins long terme.

# Problème de décision de Markov

## Autres notions

- ▶ état initial
- ▶ trajectoire : séquence d'états parcourus au fil du temps
- ▶ horizon : fini vs. infini
- ▶  $\zeta$  les plus courantes :
  - ▶ horizon fini :  $\zeta = \sum_t r_t$ .
  - ▶ horizon fini ou infini :  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .



$\gamma^t$  garantit que  $|\sum_t \gamma^t r_t| < \infty$ .  
 Selon la valeur de  $\gamma$ , les retours futurs sont pris en compte sur un plus ou moins long terme.

- ▶ retour immédiat :  $r_t$  vs. retour long terme  $\sum_t \dots$

# Modélisation d'une situation comme un PDM

## Problème du chauffeur de taxi

- ▶  $\mathcal{X} = \{A, B, C\}$
- ▶  $\mathcal{A} = \{$ 
  - ▶  $a_1 = \text{marauder}$
  - ▶  $a_2 = \text{stationner à une station}$
  - ▶  $a_3 = \text{stationner et attendre un appel radio}$ $\}$
- ▶ dans la ville B, il n'y a pas de station de taxi.

# Modélisation d'une situation comme un PDM

## Problème du chauffeur de taxi

$\mathcal{P}$	état courant $x_t$ (ville courante)	action $a_t$	état suivant $x_{t+1}$ ville suivante		
			A	B	C
A		$a_1$	1/2	1/4	1/4
		$a_2$	1/16	3/4	3/16
		$a_3$	1/4	1/8	5/8
B		$a_1$	1/2	0	1/2
		$a_3$	1/16	7/8	1/16
C		$a_1$	1/4	1/4	1/2
		$a_2$	1/8	3/4	1/8
		$a_3$	3/4	1/16	3/16

$\mathcal{R}$	état courant $x_t$ (ville courante)	action $a_t$	état suivant $x_{t+1}$ ville suivante		
			A	B	C
A		$a_1$	10	4	8
		$a_2$	8	2	4
		$a_3$	4	6	4
B		$a_1$	14	0	18
		$a_3$	8	16	8
C		$a_1$	10	2	8
		$a_2$	6	4	2
		$a_3$	4	0	8

Exo : en terme de retour immédiat espéré, quelle est la meilleure action dans A ? (dans B, dans C ?)

# Problème de décision de Markov

Énoncé du problème de planification :

- ▶ Donnée : un PDM.
- ▶ Question : en calculer la/une politique optimale.
- ▶ Solutions :
  - ▶ résolution d'un système d'équations linéaires
  - ▶ programmation dynamique
  - ▶ programmation linéaire

# Problème de décision de Markov

## Quelques propriétés

« Markov » signifie que toute l'information utile est contenue dans l'état courant : toute l'information concernant le passé de l'agent est présente dans l'état.

Markov implique que  $\pi^*$  ne dépend que de l'état courant (pas des états passés, des actions passées, ...).

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).
  - ▶  $\mathcal{X}$  est fini et discret.

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).
  - ▶  $\mathcal{X}$  est fini et discret.
  - ▶  $\mathcal{A}$  est fini.

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).
  - ▶  $\mathcal{X}$  est fini et discret.
  - ▶  $\mathcal{A}$  est fini.
- ▶ Théorème de Blackwell : sous ces hypothèses,  $\pi^*$  existe et est stationnaire et déterministe.

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).
  - ▶  $\mathcal{X}$  est fini et discret.
  - ▶  $\mathcal{A}$  est fini.
- ▶ Théorème de Blackwell : sous ces hypothèses,  $\pi^*$  existe et est stationnaire et déterministe.
- ▶ Remarques :

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).
  - ▶  $\mathcal{X}$  est fini et discret.
  - ▶  $\mathcal{A}$  est fini.
- ▶ Théorème de Blackwell : sous ces hypothèses,  $\pi^*$  existe et est stationnaire et déterministe.
- ▶ Remarques :
  - ▶ pour un  $\gamma$ -PDM donné, il existe en général plusieurs politiques optimales équivalentes ;

# Problème de décision de Markov

## Quelques propriétés

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On suppose :
  - ▶ que le PDM est stationnaire.
  - ▶  $\gamma \in [0, 1[$
  - ▶ que les retours sont bornés ( $|\mathcal{R}(x, a, x')| < +\infty$ ).
  - ▶  $\mathcal{X}$  est fini et discret.
  - ▶  $\mathcal{A}$  est fini.
- ▶ Théorème de Blackwell : sous ces hypothèses,  $\pi^*$  existe et est stationnaire et déterministe.
- ▶ Remarques :
  - ▶ pour un  $\gamma$ -PDM donné, il existe en général plusieurs politiques optimales équivalentes ;
  - ▶  $\pi^*$  dépend de  $\gamma$  en général.

# Problème de décision de Markov

## Valeur d'un état

### Intuition

Si on est capable de calculer l'intérêt d'être dans un état quelconque, on peut ensuite chercher à atteindre les états les plus intéressants.

# Problème de décision de Markov

## Valeur d'un état

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .

# Problème de décision de Markov

## Valeur d'un état

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On définit la **valeur** d'un état  $x$  selon une politique  $\pi$ ,  $V^\pi(x)$  :  
$$V^\pi(x) = \mathbb{E}[\sum_{k \geq 0} \gamma^k r_{t+k} | x_t = x].$$

# Problème de décision de Markov

## Valeur d'un état

- ▶ On considère un PDM défini par  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$  et la fonction objectif  $\zeta = \sum_t \gamma^t r_t, \gamma \in [0, 1[$ .
- ▶ On définit la valeur d'un état  $x$  selon une politique  $\pi$ ,  $V^\pi(x)$  :  
$$V^\pi(x) = \mathbb{E}[\sum_{k \geq 0} \gamma^k r_{t+k} | x_t = x].$$
- ▶ Question : comment calculer  $V^\pi(x), \forall x \in \mathcal{X}$  ?

# Problème de décision de Markov

Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$

# Problème de décision de Markov

Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :

$$\sum_{k \geq 0} \gamma^k r_{t+k} =$$

# Problème de décision de Markov

Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :

$$\sum_{k \geq 0} \gamma^k r_{t+k} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :

$$\sum_{k \geq 0} \gamma^k r_{t+k} = r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots)$$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :

$$\sum_{k \geq 0} \gamma^k r_{t+k} = \frac{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots}{r_t + \gamma (\sum_{k \geq 0} \gamma^k r_{t+1+k})}$$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :
- ▶  $\sum_{t \geq 0} \gamma^k r_{t+k} = \underbrace{r_t}_{\text{retour immédiat}} + \gamma \underbrace{\left( \sum_{k \geq 1} \gamma^k r_{t+k} | x_t = x \right)}_{\text{ensuite}}$
- ▶ examinons maintenant ce  $\mathbb{E}$ .
  - ▶ si on effectue  $a$  dans  $x$ , on va dans l'état  $x'$  avec la probabilité  $\mathcal{P}(x, a, x')$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :
- ▶  $\sum_{t \geq 0} \gamma^k r_{t+k} = \underbrace{r_t}_{\text{retour immédiat}} + \gamma \underbrace{\left( \sum_{k \geq 1} \gamma^k r_{t+k} | x_t = x \right)}_{\text{ensuite}}$
- ▶ examinons maintenant ce  $\mathbb{E}$ .
  - ▶ si on effectue  $a$  dans  $x$ , on va dans l'état  $x'$  avec la probabilité  $\mathcal{P}(x, a, x')$
  - ▶ la probabilité d'effectuer  $a$  dans  $x$  est  $\pi(x, a)$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :
- ▶  $\sum_{t \geq 0} \gamma^k r_{t+k} = \underbrace{r_t}_{\text{retour immédiat}} + \gamma \underbrace{\left( \sum_{k \geq 1} \gamma^k r_{t+k} | x_t = x \right)}_{\text{ensuite}}$
- ▶ examinons maintenant ce  $\mathbb{E}$ .
  - ▶ si on effectue  $a$  dans  $x$ , on va dans l'état  $x'$  avec la probabilité  $\mathcal{P}(x, a, x')$
  - ▶ la probabilité d'effectuer  $a$  dans  $x$  est  $\pi(x, a)$
  - ▶ donc la probabilité d'atteindre  $x'$  depuis  $x$  est  $\sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x')$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :
- ▶  $\sum_{t \geq 0} \gamma^k r_{t+k} = \underbrace{r_t}_{\text{return immédiat}} + \gamma \underbrace{\left( \sum_{k \geq 1} \gamma^k r_{t+k} | x_t = x \right)}_{\text{ensuite}}$
- ▶ examinons maintenant ce  $\mathbb{E}$ .
  - ▶ si on effectue  $a$  dans  $x$ , on va dans l'état  $x'$  avec la probabilité  $\mathcal{P}(x, a, x')$
  - ▶ la probabilité d'effectuer  $a$  dans  $x$  est  $\pi(x, a)$
  - ▶ donc la probabilité d'atteindre  $x'$  depuis  $x$  est  $\sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x')$
  - ▶ donc  $\sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x')$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :
- ▶  $\sum_{t \geq 0} \gamma^t r_{t+k} = \underbrace{r_t}_{\text{return immédiat}} + \gamma \underbrace{\left( \sum_{k \geq 1} \gamma^k r_{t+k} | x_t = x \right)}_{\text{ensuite}}$
- ▶ examinons maintenant ce  $\mathbb{E}$ .
  - ▶ si on effectue  $a$  dans  $x$ , on va dans l'état  $x'$  avec la probabilité  $\mathcal{P}(x, a, x')$
  - ▶ la probabilité d'effectuer  $a$  dans  $x$  est  $\pi(x, a)$
  - ▶ donc la probabilité d'atteindre  $x'$  depuis  $x$  est  $\sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x')$
  - ▶ donc  $\sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x')$
  - ▶  $\mathbb{E}[\sum_{t \geq 0} \gamma^t r_t] = V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$

# Problème de décision de Markov

## Équation de Bellman [Bellman, 1957]

- ▶  $V^\pi(x) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_{t+k} | x_t = x]$
- ▶ examinons le terme dont on veut calculer l'espérance :
- ▶  $\sum_{t \geq 0} \gamma^t r_{t+k} = \underbrace{r_t}_{\text{retour immédiat}} + \gamma \underbrace{\left( \sum_{k \geq 1} \gamma^k r_{t+k} | x_t = x \right)}_{\text{ensuite}}$
- ▶ examinons maintenant ce  $\mathbb{E}$ .
  - ▶ si on effectue  $a$  dans  $x$ , on va dans l'état  $x'$  avec la probabilité  $\mathcal{P}(x, a, x')$
  - ▶ la probabilité d'effectuer  $a$  dans  $x$  est  $\pi(x, a)$
  - ▶ donc la probabilité d'atteindre  $x'$  depuis  $x$  est  $\sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x')$
  - ▶ donc  $\mathbb{E}[\sum_{t \geq 0} \gamma^t r_t] = V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$
  - ▶ Équation de Bellman

# Problème de décision de Markov

## Calcul de $V^\pi$

$$\blacktriangleright V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$$

# Problème de décision de Markov

## Calcul de $V^\pi$

- ▶ 
$$V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$$
- ▶  $\pi$ ,  $\mathcal{P}$  et  $\mathcal{R}$  sont connus.

# Problème de décision de Markov

## Calcul de $V^\pi$

- ▶ 
$$V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$$
- ▶  $\pi$ ,  $\mathcal{P}$  et  $\mathcal{R}$  sont connus.
- ▶  $V^\pi(x)$  est inconnu,  $\forall x$ .

# Problème de décision de Markov

## Calcul de $V^\pi$

- ▶ 
$$V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$$
- ▶  $\pi$ ,  $\mathcal{P}$  et  $\mathcal{R}$  sont connus.
- ▶  $V^\pi(x)$  est inconnu,  $\forall x$ .
- ▶ Première approche : c'est donc un système de  $N$  équations linéaires à  $N$  inconnues.  
...

# Problème de décision de Markov

## Calcul de $V^\pi$ II

- ▶  $V^\pi(x) = \sum_{x' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(x, a) \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + V^\pi(x')]$
- ▶  $\pi$ ,  $\mathcal{P}$  et  $\mathcal{R}$  sont connus.
- ▶  $V^\pi(x)$  est inconnu,  $\forall x$ .
- ▶ Deuxième approche : algorithme d'évaluation d'une politique :

**Nécessite:** un  $\gamma$ -PDM  $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ , une politique  $\pi$ , un seuil  $\epsilon$   
 initialiser  $V_0^\pi \leftarrow 0$

$i \leftarrow 0$

**répéter**

**pour** tout état  $x \in \mathcal{X}$  **faire**

$$V_{i+1}^\pi(x) \leftarrow \sum_{a \in \mathcal{A}(x)} \pi(x, a) \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V_i^\pi(x')]$$

**fin pour**

$i \leftarrow i + 1$

**jusque**  $\|V_i^\pi - V_{i-1}^\pi\|_\infty \leq \epsilon$

# Planification

## Application au problème du chauffeur de taxi

Calcul de  $V^\pi$  pour une politique uniformément aléatoire et  $\gamma = 0,9$  avec différentes méthodes :

- ▶ calcul de  $V^*$  par résolution de systèmes linéaires :  
$$V^*(A) = \frac{156420}{1789} \approx 87,43, V^*(B) = \frac{5113540}{51881} \approx 98,56, V^*(C) = \frac{13602460}{155643} \approx 87,40.$$
- ▶ Estimation de  $V^\pi$  par algorithme d'évaluation d'une politique : avec  $\epsilon$  suffisamment petit :  
$$V^*(A) \approx 87,43, V^*(B) \approx 98,56, V^*(C) \approx 87,40.$$

# Planification

## Améliorer $\pi$

- ▶ On peut définir un ordre (partiel) sur les politiques :  
 $\pi \leq \pi' \iff V^\pi(x) \leq V^{\pi'}(x), \forall x$

# Planification

## Améliorer $\pi$

- ▶ On peut définir un ordre (partiel) sur les politiques :
$$\pi \leq \pi' \iff V^\pi(x) \leq V^{\pi'}(x), \forall x$$
- ▶ On peut améliorer une politique en la rendant gloutonne : politique

gloutonne = dans chaque état, on choisit la meilleure action.

$$\pi'(x) \leftarrow \arg \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^\pi(x')], \forall x$$

entraîne  $\pi \leq \pi'$

# Planification

## Itération sur les politiques (IP) [Howard, 1960]

Partant d'une politique  $\pi_{k=0}$  quelconque, alterner évaluation et amélioration :

- ▶ évaluation de  $V^{\pi_k}$
- ▶ calculer  $\pi_{k+1}$  par « gloutonnisation » de  $V^{\pi_k}$
- ▶  $k++$
- ▶ itérer tant que  $\pi_{k-1} \neq \pi_k$

# Planification

## Itération sur les politiques (IP) [Howard, 1960]

Partant d'une politique  $\pi_{k=0}$  quelconque, alterner évaluation et amélioration :

- ▶ évaluation de  $V^{\pi_k}$
- ▶ calculer  $\pi_{k+1}$  par « gloutonnisation » de  $V^{\pi_k}$
- ▶  $k++$
- ▶ itérer tant que  $\pi_{k-1} \neq \pi_k$
  
- ▶ Algorithme très efficace

# Planification

## Itération sur les politiques (IP) [Howard, 1960]

Partant d'une politique  $\pi_{k=0}$  quelconque, alterner évaluation et amélioration :

- ▶ évaluation de  $V^{\pi_k}$
  - ▶ calculer  $\pi_{k+1}$  par « gloutonnisation » de  $V^{\pi_k}$
  - ▶  $k++$
  - ▶ itérer tant que  $\pi_{k-1} \neq \pi_k$
- 
- ▶ Algorithme très efficace
  - ▶ Converge vers  $\pi^*$  sous quelques conditions ... délicates.

# Planification

## Itération sur les valeurs (IV)

- ▶ Équation d'optimalité de Bellman :

$$V^*(x) = \max_{\pi} V^{\pi}(x) = \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^*(x')]$$

équation caractérisant la valeur de la politique optimale  $\pi^*$ .

# Planification

## Itération sur les valeurs (IV)

- ▶ Équation d'optimalité de Bellman :

$$V^*(x) = \max_{\pi} V^{\pi}(x) = \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^*(x')]$$

équation caractérisant la valeur de la politique optimale  $\pi^*$ .

- ▶  $V^*$  peut être calculé par l'algorithme d'itération sur les valeurs.

# Planification

## Itération sur les valeurs (IV)

- ▶ Équation d'optimalité de Bellman :

$$V^*(x) = \max_{\pi} V^{\pi}(x) = \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^*(x')]$$

équation caractérisant la valeur de la politique optimale  $\pi^*$ .

- ▶  $V^*$  peut être calculé par l'algorithme d'itération sur les valeurs.
- ▶ Pour un  $\gamma$ -PDM, cet algorithme converge asymptotiquement vers  $V^*$ .

# Planification

## Itération sur les valeurs (IV)

- ▶ Équation d'optimalité de Bellman :

$$V^*(x) = \max_{\pi} V^{\pi}(x) = \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^*(x')]$$

# Planification

## Itération sur les valeurs (IV)

- ▶ Équation d'optimalité de Bellman :

$$V^*(x) = \max_{\pi} V^{\pi}(x) = \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^*(x')]$$

- ▶ Nécessite: un  $\gamma$ -PDM, un seuil de précision  $\epsilon$

initialiser  $V_0 \leftarrow 0, k \leftarrow 0$

**répéter**

**pour** tout état  $x \in \mathcal{X}$  **faire**

$$V_{k+1}(x) \leftarrow \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V_k(x')]$$

**fin pour**

$$k \leftarrow k + 1$$

**jusque**  $\|V_k - V_{k-1}\|_{\infty} \leq \frac{\epsilon(1-\gamma)}{2\gamma}$

**pour** tout état  $x \in \mathcal{X}$  **faire**

$$\pi(x) \leftarrow \arg \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V_k(x')]$$

**fin pour**

# Planification

## Itération sur les valeurs (IV)

- ▶ Équation d'optimalité de Bellman :

$$V^*(x) = \max_{\pi} V^\pi(x) = \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V^*(x')]$$

- ▶ Nécessite: un  $\gamma$ -PDM, un seuil de précision  $\epsilon$

initialiser  $V_0 \leftarrow 0, k \leftarrow 0$

**répéter**

**pour** tout état  $x \in \mathcal{X}$  **faire**

$$V_{k+1}(x) \leftarrow \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V_k(x')]$$

**fin pour**

$$k \leftarrow k + 1$$

**jusque**  $\|V_k - V_{k-1}\|_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$

**pour** tout état  $x \in \mathcal{X}$  **faire**

$$\pi(x) \leftarrow \arg \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V_k(x')]$$

**fin pour**

- ▶ Propriété :  $\|V_k - V^*\|_\infty \leq \epsilon$

# Planification

## Application au problème du chauffeur de taxi

Calcul de  $\pi^*$  pour  $\gamma = 0,9$  avec différentes méthodes :

- ▶ estimation de  $V^*$  par IV :

avec  $\epsilon$  suffisamment petit :

$$V^*(A) \approx 121,65, V^*(B) \approx 135,31, V^*(C) \approx 122,84$$

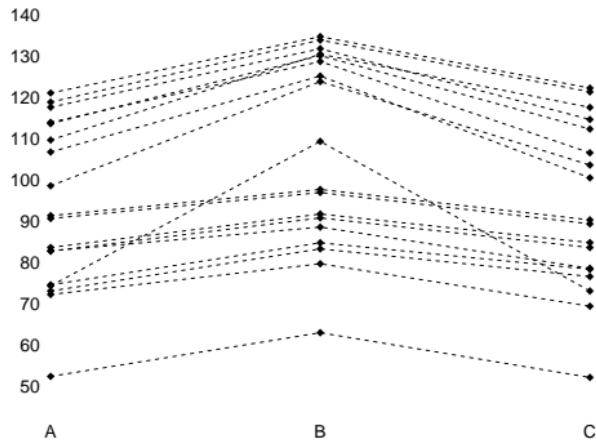
$$\text{Gloutonnisation} \rightsquigarrow \pi^*(A) = a_2, \pi^*(B) = a_3, \pi^*(C) = a_2$$

- ▶ estimation de  $\pi^*$  par IP :

$$\pi^*(A) = a_2, \pi^*(B) = a_3, \pi^*(C) = a_2$$

# Planification

## Application au problème du chauffeur de taxi



Fonction valeur pour toutes les politiques déterministes stationnaires pour le problème du chauffeur de taxi pour  $\gamma = 0,9$ .

# Planification

## Résolution par programmation linéaire

- ▶  $V^*$  est la solution du PL suivant :

$V \in \mathbb{R}^N$ , où  $N = |\mathcal{X}|$  :

$$\min \sum_{x=1}^{x=N} V[x]$$

vérifiant  $\forall (x, a) \in \mathcal{X} \times \mathcal{A}$

$$V[x] - \sum_{x'} \gamma \mathcal{P}(x, a, x') V[x'] \geq \sum_{x'} \mathcal{P}(x, a, x') \mathcal{R}(x, a, x')$$

# Planification

## Résolution par programmation linéaire

- ▶  $V^*$  est la solution du PL suivant :

$$V \in \mathbb{R}^N, \text{ où } N = |\mathcal{X}| :$$

$$\min \sum_{x=1}^{x=N} V[x]$$

vérifiant  $\forall (x, a) \in \mathcal{X} \times \mathcal{A}$

$$V[x] - \sum_{x'} \gamma \mathcal{P}(x, a, x') V[x'] \geq \sum_{x'} \mathcal{P}(x, a, x') \mathcal{R}(x, a, x')$$

- ▶ Le problème dual est :

$$\max \sum_{(x,a)} \sum_{x'} \mathcal{P}(x, a, x') \mathcal{R}(x, a, x') \xi(x, a)$$

vérifiant

$$\sum_a \xi(x', a) - \sum_{x,a} \gamma \mathcal{P}(x, a, x') \xi(x, a) \leq 1, \forall x' \in \mathcal{X}$$

et  $\xi(x, a) \geq 0, \forall (x, a) \in \mathcal{X} \times \mathcal{A}$

La solution du dual spécifie  $\pi^* : \forall x \in \mathcal{X}, \xi(x, a) \neq 0 \iff a \in \pi^*(x)$ .

# Planification

## Application au problème du chauffeur de taxi

Calcul de  $\pi^*$  pour  $\gamma = 0,9$  par PL :

- ▶  $V(A) = 121.6535, V(B) = 135.3063, V(C) = 122.8369$

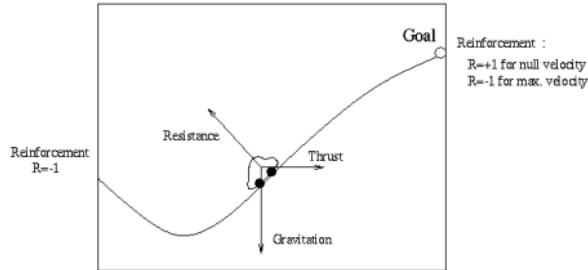
- ▶  $\xi = \begin{pmatrix} 0 \\ 2.866906 \\ 0 \\ 0 \\ 0 \\ 23.94366 \\ 0 \\ 3.189432 \\ 0 \end{pmatrix} \rightsquigarrow \pi^*(A) = a_2, \pi^*(B) = a_3, \pi^*(C) = a_2$

# Planification

Les limites :  $\mathcal{X}$  grand

# Planification

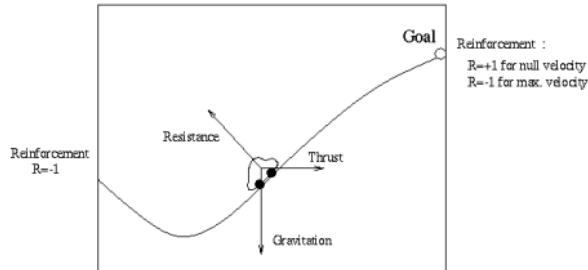
Les limites :  $\mathcal{X}$  grand



Voiture pas assez puissante pour atteindre le sommet de la colline : doit prendre de l'élan puis accélérer !

# Planification

Les limites :  $\mathcal{X}$  grand

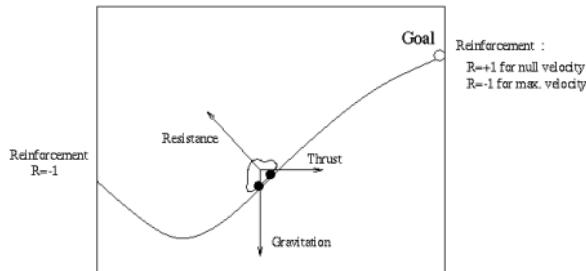


Voiture pas assez puissante pour atteindre le sommet de la colline : doit prendre de l'élan puis accélérer !

- ▶  $\mathcal{X} = [-1,2; 0,5] \times [-0,07; 0,07]$  : position, vitesse ;  
 $x_0 \in ([-0,6; -0,4]; 0)$
- ▶  $\mathcal{A} = \{-1, 0, +1\}$  : accélération
- ▶  $\mathcal{P}$  : déterministe : dynamique Newtonienne approximative :  
 $x_{t+1} \stackrel{\text{def}}{=} \text{bound}(\dot{x}_t + 0,001a_t - \cos(3x_t))$  et  $x_{t+1} \stackrel{\text{def}}{=} \text{bound}(x_t + x_{t+1})$
- ▶  $r_t = -1$  à chaque pas de temps
- ▶ Objectif : atteindre le sommet de la colline en moins de 200 pas de temps.

# Planification

Les limites :  $\mathcal{X}$  grand

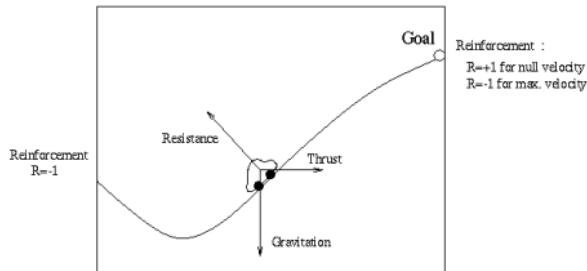


Partitionnement progressif et adaptatif [Munos & Moore, 2002]

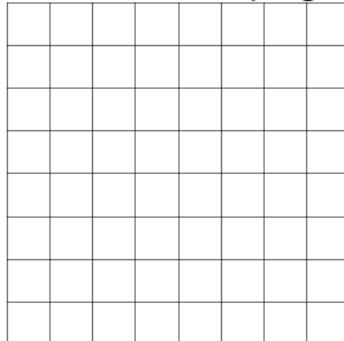
Ce type de techniques est limité à des espaces de petites dimensions :  $\mathbb{R}^{\leq 5}$ .

# Planification

Les limites :  $\mathcal{X}$  grand



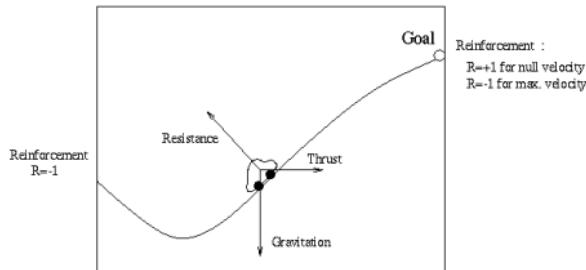
Partitionnement progressif et adaptatif [Munos & Moore, 2002]



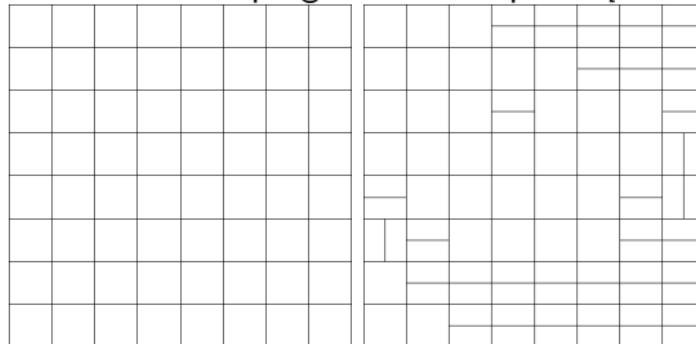
Ce type de techniques est limité à des espaces de petites dimensions :  $\mathbb{R}^{\leq 5}$ .

# Planification

Les limites :  $\mathcal{X}$  grand



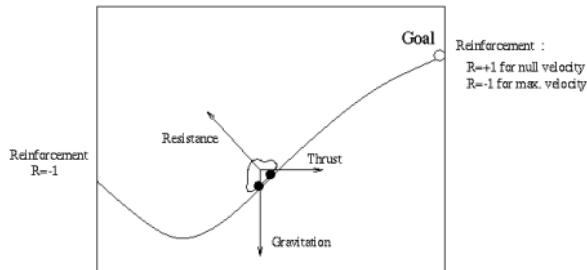
Partitionnement progressif et adaptatif [Munos & Moore, 2002]



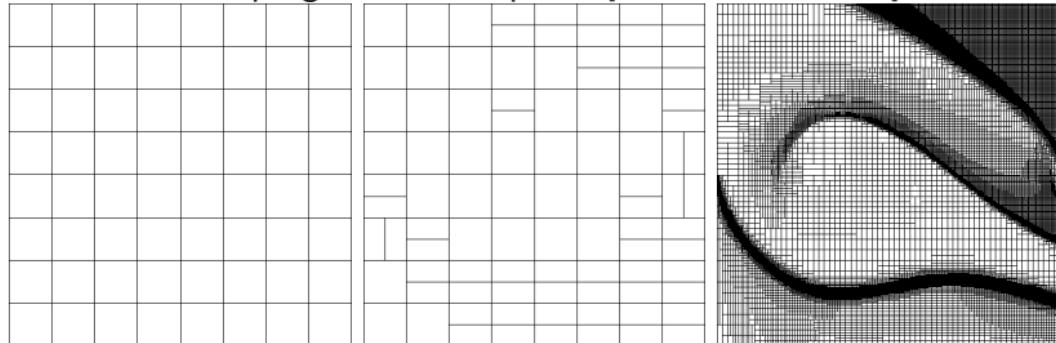
Ce type de techniques est limité à des espaces de petites dimensions :  $\mathbb{R}^{\leq 5}$ .

# Planification

Les limites :  $\mathcal{X}$  grand



Partitionnement progressif et adaptatif [Munos & Moore, 2002]



Ce type de techniques est limité à des espaces de petites dimensions :  $\mathbb{R}^{\leq 5}$ .

# Planification

Techniques approchées : programmation dynamique approchée

- ▶ Quand  $\mathcal{X}$  est trop grand, on utilise une approximation de  $V$ .  
« Programmation dynamique approchée »  
Voir e.g. W. Powell, Castlelab, Princeton.

# Planification

## Techniques approchées : programmation dynamique approchée

- ▶ Quand  $\mathcal{X}$  est trop grand, on utilise une approximation de  $V$ .  
« Programmation dynamique approchée »  
Voir e.g. W. Powell, Castlelab, Princeton.
- ▶  $\mathcal{A}$  peut aussi être trop grand, voire infini.  
Exemple : jeu de go : technique arborescente (MCTS).

# Planification

## Techniques approchées : programmation dynamique approchée

- ▶ Quand  $\mathcal{X}$  est trop grand, on utilise une approximation de  $V$ .  
« Programmation dynamique approchée »  
Voir e.g. W. Powell, Castlelab, Princeton.
- ▶  $\mathcal{A}$  peut aussi être trop grand, voire infini.  
Exemple : jeu de go : technique arborescente (MCTS).
- ▶ Il y a beaucoup moins de garanties théoriques que dans les approches à la Bellman.

# Planification

## Techniques approchées : programmation dynamique approchée

- ▶ Quand  $\mathcal{X}$  est trop grand, on utilise une approximation de  $V$ .  
« Programmation dynamique approchée »  
Voir e.g. W. Powell, Castlelab, Princeton.
- ▶  $\mathcal{A}$  peut aussi être trop grand, voire infini.  
Exemple : jeu de go : technique arborescente (MCTS).
- ▶ Il y a beaucoup moins de garanties théoriques que dans les approches à la Bellman.
- ▶ Idée que l'on va détailler dans la partie suivante.

# Apprentissage

# Apprentissage

- ▶ Maintenant, on suppose connus :
  - ▶ l'espace d'états  $\mathcal{X}$ ,
  - ▶ l'espace d'actions  $\mathcal{A}$ ,
  - ▶ la fonction objectif  $\zeta$ ,
- ▶ et inconnues :
  - ▶ la fonction de transition  $\mathcal{P}$
  - ▶ la fonction de retour  $\mathcal{R}$

# Apprentissage

- ▶ Maintenant, on suppose connus :
  - ▶ l'espace d'états  $\mathcal{X}$ ,
  - ▶ l'espace d'actions  $\mathcal{A}$ ,
  - ▶ la fonction objectif  $\zeta$ ,
- ▶ et inconnues :
  - ▶ la fonction de transition  $\mathcal{P}$
  - ▶ la fonction de retour  $\mathcal{R}$

Objectif : trouver  $\pi^*$  qui optimise  $\zeta$ .

# Apprentissage

- ▶ Maintenant, on suppose connus :
  - ▶ l'espace d'états  $\mathcal{X}$ ,
  - ▶ l'espace d'actions  $\mathcal{A}$ ,
  - ▶ la fonction objectif  $\zeta$ ,
- ▶ et inconnues :
  - ▶ la fonction de transition  $\mathcal{P}$
  - ▶ la fonction de retour  $\mathcal{R}$

Objectif : trouver  $\pi^*$  qui optimise  $\zeta$ .

✓ Certaines quantités indispensables à cette optimisation sont inconnues.

# Apprentissage

- ▶ Maintenant, on suppose connus :
  - ▶ l'espace d'états  $\mathcal{X}$ ,
  - ▶ l'espace d'actions  $\mathcal{A}$ ,
  - ▶ la fonction objectif  $\zeta$ ,
- ▶ et inconnues :
  - ▶ la fonction de transition  $\mathcal{P}$
  - ▶ la fonction de retour  $\mathcal{R}$

Objectif : trouver  $\pi^*$  qui optimise  $\zeta$ .

- ✓ Certaines quantités indispensables à cette optimisation sont inconnues.
- ✓ Ces quantités définissent l'environnement : en **interagissant** avec l'environnement, on n'a pas besoin de les connaître.

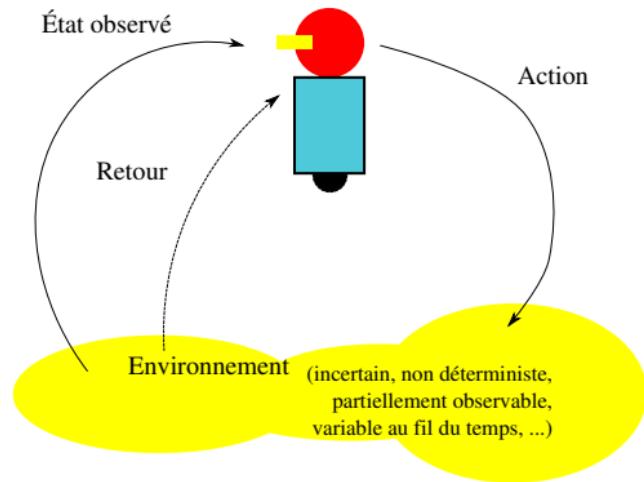
# Apprentissage

- ▶ Maintenant, on suppose connus :
  - ▶ l'espace d'états  $\mathcal{X}$ ,
  - ▶ l'espace d'actions  $\mathcal{A}$ ,
  - ▶ la fonction objectif  $\zeta$ ,
- ▶ et inconnues :
  - ▶ la fonction de transition  $\mathcal{P}$
  - ▶ la fonction de retour  $\mathcal{R}$

Objectif : trouver  $\pi^*$  qui optimise  $\zeta$ .

Tout ce qui suit s'appuie donc sur l'**interaction** entre l'agent et son environnement.

# Apprentissage par renforcement = apprentissage par interaction



# Apprentissage

## Méthode de Monte Carlo

Intuition

# Apprentissage

## Méthode de Monte Carlo

### Intuition

Méthode de Monte Carlo  $\rightsquigarrow$  estime une quantité liée à un processus stochastique.

# Apprentissage

## Méthode de Monte Carlo

### Intuition

Méthode de Monte Carlo  $\rightsquigarrow$  estime une quantité liée à un processus stochastique.  
(Ou de trop grande taille pour être simulé exactement.)

# Apprentissage

## Méthode de Monte Carlo

- ▶ on suppose disposer d'un simulateur de l'environnement :  
 $(x_t, a_t) \mapsto (x_{t+1}, r_t)$
  - ▶ on peut estimer  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$  par simulation :
- sélectionner  $x_0 \in \mathcal{X}$
- $t \leftarrow 0$
- $W \leftarrow 0$
- tant-que**  $\gamma^k \geq \epsilon$  **faire**
- choisir  $a_t$  en fonction de  $\pi$
- $(x_{t+1}, r_t) \leftarrow \text{simule}(x_t, a_t)$
- $W \leftarrow W + \gamma^t r_t$
- $t++$
- fin tant-que**

# Apprentissage

## Méthode de Monte Carlo

- ▶ on suppose disposer d'un simulateur de l'environnement :  
 $(x_t, a_t) \mapsto (x_{t+1}, r_t)$
- ▶ on peut estimer  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$  par simulation :
  - sélectionner  $x_0 \in \mathcal{X}$
  - $V[x_0] \leftarrow 0$
  - pour**  $i \in \{1,..N\}$  **faire**
  - $t \leftarrow 0$
  - $W \leftarrow 0$
  - tant-que**  $\gamma^k \geq \epsilon$  **faire**
  - choisir  $a_t$  en fonction de  $\pi$
  - $(x_{t+1}, r_t) \leftarrow \text{simule}(x_t, a_t)$
  - $W \leftarrow W + \gamma^t r_t$
  - $t++$
  - fin tant-que**
  - $V[x_0] \leftarrow V[x_0] + W$
  - fin pour**
  - $V[x_0] \leftarrow V[x_0]/N$

# Apprentissage

## Méthode de Monte Carlo

- ▶ on suppose disposer d'un simulateur de l'environnement :  
 $(x_t, a_t) \mapsto (x_{t+1}, r_t)$
- ▶ on peut estimer  $V^\pi(x) = \mathbb{E}[\sum_k \gamma^k r_{t+k} | x_t = x]$  par simulation :
  - sélectionner  $x_0 \in \mathcal{X}$
  - $V[x_0] \leftarrow 0$
  - pour**  $i \in \{1,..N\}$  **faire**
  - $t \leftarrow 0$
  - $W \leftarrow 0$
  - tant-que**  $\gamma^k \geq \epsilon$  **faire**
  - choisir  $a_t$  en fonction de  $\pi$
  - $(x_{t+1}, r_t) \leftarrow \text{simule}(x_t, a_t)$
  - $W \leftarrow W + \gamma^t r_t$
  - $t++$
  - fin tant-que**
  - $V[x_0] \leftarrow V[x_0] + W$
  - fin pour**
  - $V[x_0] \leftarrow V[x_0]/N$
- ▶  $V[x_0]$  contient une estimation non biaisée de  $V^\pi(x_0)$ .

# Apprentissage

## Méthode de Monte Carlo : illustration sur le problème du chauffeur de taxi

- ▶ Pour  $\pi^*$ ,  $\epsilon = 2 \cdot 10^{-10}$ ,  $\gamma = 0,9$ , on obtient :  
 $V^*(A) = 120,18 \pm 1,06$ ,  $V^*(B) = 133,24 \pm 0,90$ ,  
 $V^*(C) = 122,03 \pm 0,98$ .  
Écart-type obtenu sur 100 exécutions.  
À comparer avec le résultat d'IV :  
 $V^*(A) \approx 121,65$ ,  $V^*(B) \approx 135,31$ ,  $V^*(C) \approx 122,84$

# Apprentissage

## Méthode de Monte Carlo : illustration sur le problème du chauffeur de taxi

- ▶ Pour  $\pi^*$ ,  $\epsilon = 2 \cdot 10^{-10}$ ,  $\gamma = 0,9$ , on obtient :  
 $V^*(A) = 120,18 \pm 1,06$ ,  $V^*(B) = 133,24 \pm 0,90$ ,  
 $V^*(C) = 122,03 \pm 0,98$ .  
Écart-type obtenu sur 100 exécutions.  
À comparer avec le résultat d'IV :  
 $V^*(A) \approx 121,65$ ,  $V^*(B) \approx 135,31$ ,  $V^*(C) \approx 122,84$
- ▶ Utilisation de cette méthode pour évaluer une politique, puis l'améliorer (IP).

# Apprentissage

## Méthode de Monte Carlo

- ▶ Idée et algorithme très simples.
- ▶ Idée qui peut s'appliquer dans de très nombreux cas.
- ▶ Estimateur converge ... lentement.
- ▶ Idée à la base de nombreux algorithmes beaucoup plus subtils.

# Problèmes de décision de Markov (suite)

## Valeur d'une paire état-action

- ▶ Outre la valeur d'un état  $V^\pi(x)$  (resp. valeur optimale  $V^*(x)$ ) , on peut aussi définir la valeur d'une paire état-action  $Q^\pi(x, a)$  (resp.  $Q^*(x, a)$ ) :  
$$Q^\pi(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi)).$$

# Problèmes de décision de Markov (suite)

## Valeur d'une paire état-action

- ▶ Outre la valeur d'un état  $V^\pi(x)$  (resp. valeur optimale  $V^*(x)$ ) , on peut aussi définir la valeur d'une paire état-action  $Q^\pi(x, a)$  (resp.  $Q^*(x, a)$ ) :
- $Q^\pi(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t|a_t = a, \pi))$ .
- ▶ Équation de Bellman pour  $Q$  :

$$Q^\pi(x, a) = \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma \sum_{a' \in \mathcal{A}} \pi(x', a') Q^\pi(x')]$$

# Problèmes de décision de Markov (suite)

## Valeur d'une paire état-action

- ▶ Outre la valeur d'un état  $V^\pi(x)$  (resp. valeur optimale  $V^*(x)$ ) , on peut aussi définir la valeur d'une paire état-action  $Q^\pi(x, a)$  (resp.  $Q^*(x, a)$ ) :
- $Q^\pi(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t|a_t = a, \pi))$ .
- ▶ Équation de Bellman pour  $Q$  :

$$Q^\pi(x, a) = \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma \sum_{a' \in \mathcal{A}} \pi(x', a') Q^\pi(x')]$$

- ▶ Équation d'optimalité de Bellman pour  $Q^*$  :

$$Q^*(x, a) = \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma \max_{a' \in \mathcal{A}} Q^*(x', a')]$$

# Problèmes de décision de Markov (suite)

## Valeur d'une paire état-action

- ▶ Outre la valeur d'un état  $V^\pi(x)$  (resp. valeur optimale  $V^*(x)$ ) , on peut aussi définir la valeur d'une paire état-action  $Q^\pi(x, a)$  (resp.  $Q^*(x, a)$ ) :
- $Q^\pi(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t|a_t = a, \pi))$ .
- ▶ Équation de Bellman pour  $Q$  :

$$Q^\pi(x, a) = \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma \sum_{a' \in \mathcal{A}} \pi(x', a') Q^\pi(x')]$$

- ▶ Équation d'optimalité de Bellman pour  $Q^*$  :

$$Q^*(x, a) = \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma \max_{a' \in \mathcal{A}} Q^*(x', a')]$$

- ▶ Relation entre  $V^*$  et  $Q^*$  :

$$V^*(x) = \max_{a \in \mathcal{A}(x)} Q^*(x, a), \forall x \in \mathcal{X}$$

# Problèmes de décision de Markov (suite)

$V$  vs.  $Q$

- ▶ Avec  $V$ , trouver l'action qui emmène dans le meilleur état suivant  $x$  nécessite une optimisation :

$$\arg \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V(x')]$$

# Problèmes de décision de Markov (suite)

*V vs. Q*

- ▶ Avec  $V$ , trouver l'action qui emmène dans le meilleur état suivant  $x$  nécessite une optimisation :

$$\arg \max_{a \in \mathcal{A}(x)} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x') [\mathcal{R}(x, a, x') + \gamma V(x')]$$

- ▶ Si on connaît  $Q$ , c'est simplement :

$$\arg \max_{a \in \mathcal{A}(x)} Q(x, a)$$

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ éq. Bellman se réduit à :  $V^\pi(x) = \mathcal{R}(x, \pi(x), x') + \gamma V^\pi(x')$

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ éq. Bellman se réduit à :  $V^\pi(x) = \mathcal{R}(x, \pi(x), x') + \gamma V^\pi(x')$
- ▶ au long d'une trajectoire, cette équation s'instancie à l'instant  $t$  :  
$$V^\pi(x_t) = r_t + \gamma V^\pi(x_{t+1})$$

Soit  $r_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t) = 0$

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ éq. Bellman se réduit à :  $V^\pi(x) = \mathcal{R}(x, \pi(x), x') + \gamma V^\pi(x')$
- ▶ au long d'une trajectoire, cette équation s'instancie à l'instant  $t$  :  
$$V^\pi(x_t) = r_t + \gamma V^\pi(x_{t+1})$$

Soit  $r_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t) = 0$
- ▶  $V^\pi$  est inconnue. On l'estime par  $\widehat{V}^\pi$ , donc :  
$$r_t + \gamma \widehat{V}^\pi(x_{t+1}) - \widehat{V}^\pi(x_t) \neq 0$$

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ éq. Bellman se réduit à :  $V^\pi(x) = \mathcal{R}(x, \pi(x), x') + \gamma V^\pi(x')$
- ▶ au long d'une trajectoire, cette équation s'instancie à l'instant  $t$  :
 
$$V^\pi(x_t) = r_t + \gamma V^\pi(x_{t+1})$$
 Soit  $r_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t) = 0$
- ▶  $V^\pi$  est inconnue. On l'estime par  $\widehat{V}^\pi$ , donc :
 
$$r_t + \gamma \widehat{V}^\pi(x_{t+1}) - \widehat{V}^\pi(x_t) \neq 0$$
- ▶ on peut utiliser ce terme pour corriger  $\widehat{V}^\pi$  :
 
$$\widehat{V}_{k+1}^\pi(x_t) \leftarrow \widehat{V}_{k+1}^\pi(x_t) + \alpha[r_t + \gamma \widehat{V}_k^\pi(x_{t+1}) - \widehat{V}_k^\pi(x_t)]$$

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ éq. Bellman se réduit à :  $V^\pi(x) = \mathcal{R}(x, \pi(x), x') + \gamma V^\pi(x')$
- ▶ au long d'une trajectoire, cette équation s'instancie à l'instant  $t$  :
 
$$V^\pi(x_t) = r_t + \gamma V^\pi(x_{t+1})$$
 Soit  $r_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t) = 0$
- ▶  $V^\pi$  est inconnue. On l'estime par  $\widehat{V}^\pi$ , donc :
 
$$r_t + \gamma \widehat{V}^\pi(x_{t+1}) - \widehat{V}^\pi(x_t) \neq 0$$
- ▶ on peut utiliser ce terme pour corriger  $\widehat{V}^\pi$  :
 
$$\widehat{V}_{k+1}^\pi(x_t) \leftarrow \widehat{V}_{k+1}^\pi(x_t) + \alpha[r_t + \gamma \widehat{V}_k^\pi(x_{t+1}) - \widehat{V}_k^\pi(x_t)]$$
- ▶ **Différence temporelle.**

# Apprentissage par renforcement

Différence temporelle [Sutton, 1984]

- ▶ Supposons que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ éq. Bellman se réduit à :  $V^\pi(x) = \mathcal{R}(x, \pi(x), x') + \gamma V^\pi(x')$
- ▶ au long d'une trajectoire, cette équation s'instancie à l'instant  $t$  :
 
$$V^\pi(x_t) = r_t + \gamma V^\pi(x_{t+1})$$
 Soit  $r_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t) = 0$
- ▶  $V^\pi$  est inconnue. On l'estime par  $\widehat{V}^\pi$ , donc :
 
$$r_t + \gamma \widehat{V}^\pi(x_{t+1}) - \widehat{V}^\pi(x_t) \neq 0$$
- ▶ on peut utiliser ce terme pour corriger  $\widehat{V}^\pi$  :
 
$$\widehat{V}_{k+1}^\pi(x_t) \leftarrow \widehat{V}_{k+1}^\pi(x_t) + \alpha[r_t + \gamma \widehat{V}_k^\pi(x_{t+1}) - \widehat{V}_k^\pi(x_t)]$$
- ▶ Différence temporelle.
- ▶ La même idée s'applique aussi dans un environnement stochastique.

# Apprentissage par renforcement

Évaluation d'une politique dans un environnement inconnu : TD(0) [Sutton, 1984]

- ▶ dans l'algorithme de Monte Carlo, remplacer la somme des  $\gamma^t r_t$  par l'utilisation de la différence temporelle :

sélectionner  $x_0 \in \mathcal{X}$

$V[x_0] \leftarrow 0$

**pour**  $i \in \{1,..N\}$  **faire**

$t \leftarrow 0$

**tant-que**  $\gamma^k \geq \epsilon$  **faire**

choisir  $a_t$  en fonction de  $\pi$

$(x_{t+1}, r_t) \leftarrow \text{simule}(x_t, a_t)$

$V[x_t] \leftarrow V[x_t] + \alpha(r_t + \gamma V[x_{t+1}] - V[x_t]))$

$t++$

**fin tant-que**

**fin pour**

Algorithme TD(0)

# Apprentissage par renforcement

## Trace d'éligibilité : TD( $\lambda$ )

- ▶ Supposons à nouveau que tout soit déterministe  $(\pi, \mathcal{P}, \mathcal{R})$
- ▶ Bellman à  $t$  :  $V^\pi(x_t) = r_t + \gamma V^\pi(x_{t+1})$
- ▶ à  $t-1$ , on peut écrire :  $V^\pi(x_{t-1}) = r_{t-1} + \gamma V^\pi(x_t)$
- ▶ et combiner  $V^\pi(x_{t-1}) = r_{t-1} + \gamma[r_t + \gamma V^\pi(x_{t+1})]$
- ▶ autre formulation de la TD :
 
$$r_{t-1} + \gamma[r_t + \gamma V^\pi(x_{t+1})] - V^\pi(x_{t-1})$$
 nulle pour  $V^\pi$ , non nulle pour  $\widehat{V}^\pi$ .
- ▶ correction possible de  $\widehat{V}^\pi(x_{t-1})$  :
 
$$\widehat{V}_{k+1}^\pi(x_{t-1}) \leftarrow \widehat{V}_{k+1}^\pi(x_{t-1}) + \lambda \alpha [r_{t-1} + \gamma \widehat{V}_k^\pi(x_{t+1})] - \widehat{V}_k^\pi(x_{t-1})]$$
- ▶ et on peut continuer à remonter le temps  
et mettre à jour tous les états par lesquels on est passé précédemment.

## Algorithme TD( $\lambda$ )

# Apprentissage par renforcement

## Trace d'éligibilité : TD + *rollouts*

- ▶ dans la mise à jour de TD(0), TD( $\lambda$ ) :  
 $V[x_t] \leftarrow V[x_t] + \alpha(r_t + \gamma V[x_{t+1}] - V[x_t]))$   
on peut remplacer  $V[x_{t+1}]$  par une estimation Monte Carlo :  
on effectue un certain nombre de trajectoires aléatoires pour « aller voir au-delà de l'état suivant ce qui va nous arriver ».  
Ces trajectoires se nomment des *rollouts*.
- ▶ idée simple mais puissante.

# Apprentissage par renforcement

## Apprentissage d'une politique optimale

- ▶ pour  $\hat{Q}$ , TD s'exprime par :  
$$r_t + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(x_{t+1}, a') - \hat{Q}(x_t, a_t)$$
- ▶ ↵ suivre une trajectoire et corriger  $\hat{Q}$  :  
$$\hat{Q}(x_t, a_t) = \hat{Q}(x_t, a_t) + \alpha[r_t + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(x_{t+1}, a') - \hat{Q}(x_t, a_t)]$$
- ▶ idée de l'algorithme Q-Learning

# Apprentissage par renforcement

Q-Learning [Watkins, 1989]

initialiser  $\hat{Q}(x, a)$ ,  $k \leftarrow 0$

**répéter**

$t \leftarrow 0$ ,  $n(x, a) \leftarrow 0, \forall (x, a)$

    Initialiser l'état de l'agent  $x_t$

**tant-que** épisode non terminé **faire**

        sélectionner  $a_t$

        effectuer cette action et observer  $r_t$  et  $x_{t+1}$

$\hat{Q}(x_t, a_t) \leftarrow \hat{Q}(x_t, a_t) + \alpha(x_t, a_t)[r_t + \max_{a'} \hat{Q}(x_{t+1}, a') - \hat{Q}(x_t, a_t)]$

$n(x_t, a_t) ++$ ,  $t ++$

**fin tant-que**

$k ++$

**jusque** ...

$\lim_{k \rightarrow \infty} \hat{Q} = Q^*$  et  $\pi^*(x) = \arg \max_a \hat{Q}(x, a), \forall x$ .

# Apprentissage par renforcement

## Convergence du Q-Learning

Pour un  $\gamma$ -PDM, sous les hypothèses vues plus haut,  $\hat{Q}$  converge asymptotiquement vers  $Q^*$  si  $\sum \alpha = +\infty$  et  $\sum \alpha^2 < +\infty$ .

# Apprentissage par renforcement

## Sélectionner $a_t$

Q-Learning :

initialiser  $\hat{Q}(x, a)$ ,  $k \leftarrow 0$

**répéter**

$t \leftarrow 0$ ,  $n(x, a) \leftarrow 0$ ,  $\forall (x, a)$

Initialiser l'état de l'agent  $x_t$

**tant-que** épisode non terminé **faire**

sélectionner  $a_t$

effectuer cette action et observer  $r_t$  et  $x_{t+1}$

$\hat{Q}(x_t, a_t) \leftarrow \hat{Q}(x_t, a_t) + \alpha(x_t, a_t)[r_t + \max_{a'} \hat{Q}(x_{t+1}, a') - \hat{Q}(x_t, a_t)]$

$n(x_t, a_t) ++$ ,  $t ++$

**fin tant-que**

$k ++$

**jusque** ...

# Apprentissage par renforcement

## Sélectionner $a_t$

Idée générale :

- ▶ initialement, l'agent ne sait rien : il doit essayer (explorer).
- ▶ petit à petit, l'agent découvre les bonnes actions : il doit refaire les actions identifiées comme bonnes (exploiter).



- ▶ Apprentissage par essai-erreur.

# Apprentissage par renforcement

## Sélectionner $a_t$

- ▶  $\epsilon$ -decreasing greedy :  $\epsilon \in [0, 1]$ ,

$$a_t = \begin{cases} \arg \max_a \hat{Q}(x_t, a) \text{ avec proba } 1 - \epsilon \\ \text{action choisie aléatoirement sinon} \end{cases}$$

Au fil des itérations,  $\epsilon$  diminue : l'exploration diminue  $\rightsquigarrow$  choix glouton.

# Apprentissage par renforcement

## Sélectionner $a_t$

- ▶  $\epsilon$ -decreasing greedy :  $\epsilon \in [0, 1]$ ,

$$a_t = \begin{cases} \arg \max_a \widehat{Q}(x_t, a) \text{ avec proba } 1 - \epsilon \\ \text{action choisie aléatoirement sinon} \end{cases}$$

Au fil des itérations,  $\epsilon$  diminue : l'exploration diminue  $\rightsquigarrow$  choix glouton.

- ▶ proportionnel à  $\widehat{Q}$  :

probabilité de sélectionner chacune des actions  $\propto \widehat{Q}(x_t, a)$  :

$$Pr[a_t = a | x_t] = \frac{\widehat{Q}(x_t, a)}{\sum_b \widehat{Q}(x_t, b)}$$

# Apprentissage par renforcement

## Sélectionner $a_t$

- ▶  $\epsilon$ -decreasing greedy :  $\epsilon \in [0, 1]$ ,

$$a_t = \begin{cases} \arg \max_a \hat{Q}(x_t, a) \text{ avec proba } 1 - \epsilon \\ \text{action choisie aléatoirement sinon} \end{cases}$$

Au fil des itérations,  $\epsilon$  diminue : l'exploration diminue  $\rightsquigarrow$  choix glouton.

- ▶ proportionnel à  $\hat{Q}$  :

probabilité de sélectionner chacune des actions  $\propto \hat{Q}(x_t, a)$  :

$$Pr[a_t = a | x_t] = \frac{\hat{Q}(x_t, a)}{\sum_b \hat{Q}(x_t, b)}$$

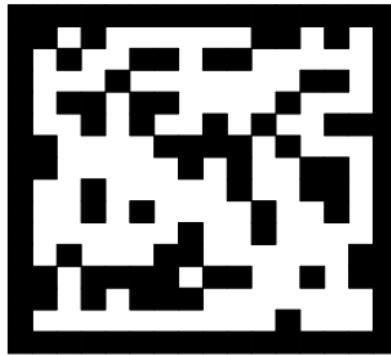
- ▶ softmax, Boltzmann = proportionnelle + contrôle de l'exploration

$$Pr[a_t = a | x_t] = \frac{e^{\frac{\hat{Q}(x_t, a)}{\tau}}}{\sum_b e^{\frac{\hat{Q}(x_t, b)}{\tau}}}$$

$\tau$  grand : exploration ;  $\tau$  proche de 0 : exploitation (glouton).

# Apprentissage par renforcement

## Q-Learning dans un labyrinthe

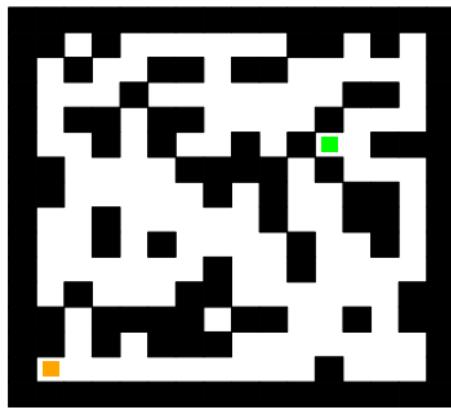


- ▶ les cases du labyrinthe sont numérotées
- ▶  $x$  : le numéro de la case où se trouve l'agent
- ▶  $\mathcal{X}$  : l'ensemble des numéros de case
- ▶  $\mathcal{A} = \{\leftarrow, \uparrow, \downarrow, \rightarrow\}$ .
- ▶ On suppose que les actions ont un effet déterministe
- ▶  $\mathcal{R}(x, a, \text{sortie}) = 1, \mathcal{R}(x, a, \neq \text{sortie}) = 0$

# Apprentissage par renforcement

## Q-Learning en action

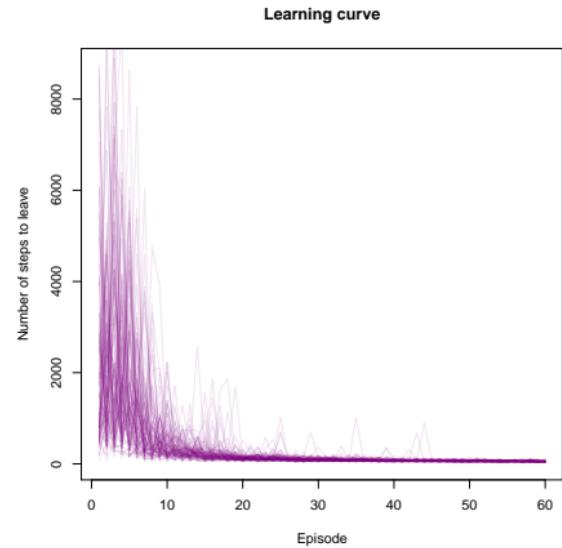
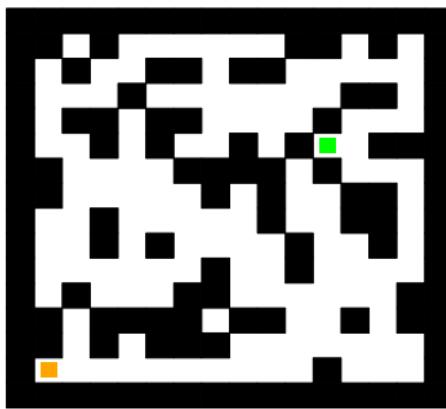
On utilise une implantation très élémentaire du Q-Learning.



# Apprentissage par renforcement

## Q-Learning en action

On utilise une implantation très élémentaire du Q-Learning.



# Apprentissage par renforcement

## Q-Learning en action

1ère atteinte



# Apprentissage par renforcement

Q-Learning en action

1ère atteinte



10è atteinte



## Apprentissage par renforcement

## Q-Learning en action

## 1ère atteinte



10è atteinte



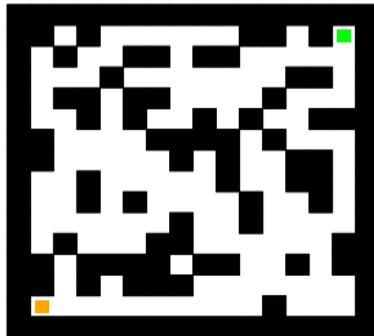
60è atteinte



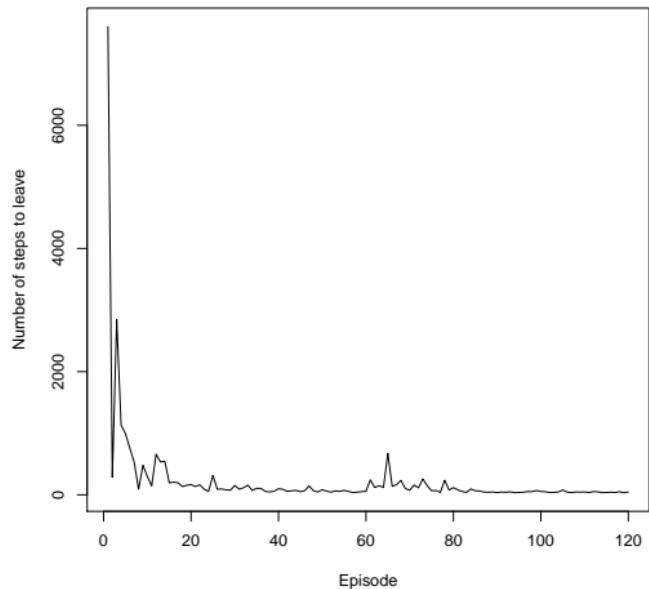
# Apprentissage par renforcement

Le Q-Learning s'adapte continuellement à son environnement

La cellule cible bouge un peu :



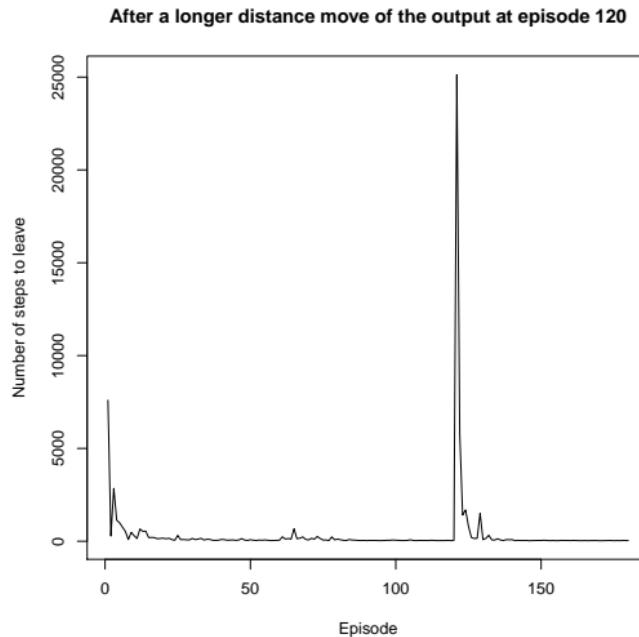
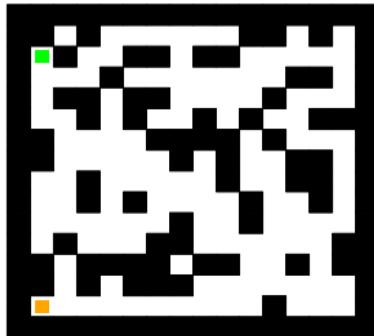
After a small distance move of the output at episode 60



# Apprentissage par renforcement

Le Q-Learning s'adapte continuellement à son environnement

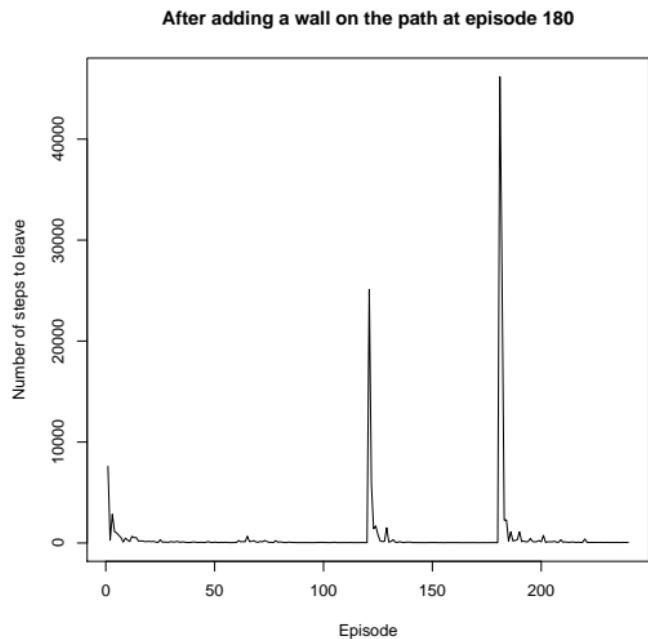
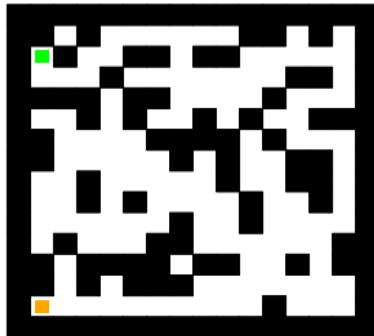
La cellule cible bouge plus loin :



# Apprentissage par renforcement

Le Q-Learning s'adapte continuellement à son environnement

Blocage du chemin trouvé :



# Apprentissage par renforcement

## Au-delà de la version tabulaire

- ▶ Version « tabulaire » du Q-Learning :  $\hat{Q}$  est représenté dans une table.

# Apprentissage par renforcement

## Au-delà de la version tabulaire

- ▶ Version « tabulaire » du Q-Learning :  $\hat{Q}$  est représenté dans une table.
- ▶  $\mathcal{X}$  grand ?

# Apprentissage par renforcement

## Au-delà de la version tabulaire

- ▶ Version « tabulaire » du Q-Learning :  $\hat{Q}$  est représenté dans une table.
- ▶  $\mathcal{X}$  grand ?
- ▶ Impossible de stocker  $Q$  dans une table.

# Apprentissage par renforcement

## Au-delà de la version tabulaire

- ▶ Version « tabulaire » du Q-Learning :  $\hat{Q}$  est représenté dans une table.
- ▶  $\mathcal{X}$  grand ?
- ▶ Impossible de stocker  $Q$  dans une table.
- ▶ Utilisation d'un approximateur de fonctions : remplacement de la table  $Q$   $[x, a]$  par une fonction  $Q(x, a)$ .

# Apprentissage par renforcement

## Au-delà de la version tabulaire

- ▶ Version « tabulaire » du Q-Learning :  $\hat{Q}$  est représenté dans une table.
- ▶  $\mathcal{X}$  grand ?
- ▶ Impossible de stocker  $Q$  dans une table.
- ▶ Utilisation d'un approximateur de fonctions : remplacement de la table  $Q[x, a]$  par une fonction  $Q(x, a)$ .
- ▶  $Q(x, a)$  retourne  $\hat{Q}(x, a)$ .

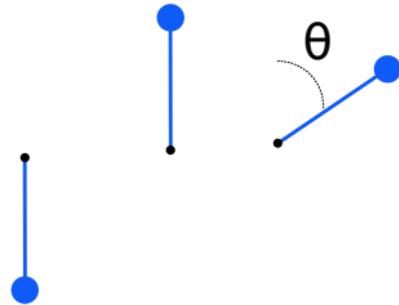
# Apprentissage par renforcement

## Au-delà de la version tabulaire

- ▶ Version « tabulaire » du Q-Learning :  $\hat{Q}$  est représenté dans une table.
- ▶  $\mathcal{X}$  grand ?
- ▶ Impossible de stocker  $Q$  dans une table.
- ▶ Utilisation d'un approximateur de fonctions : remplacement de la table  $Q$   $[x, a]$  par une fonction  $Q(x, a)$ .
- ▶  $Q(x, a)$  retourne  $\hat{Q}(x, a)$ .
- ▶ Cette estimation est mise à jour/améliorée itérativement par apprentissage.  
La fonction est paramétrée : mise à jour = modification de la valeur de ces paramètres  $\theta$ .

# Apprentissage par renforcement

## Fonction valeur

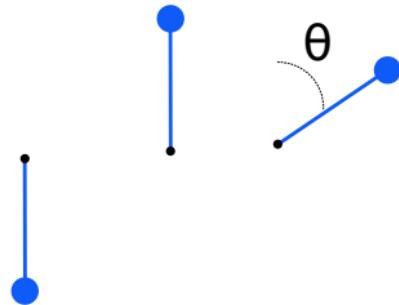


$$x = (\theta, \dot{\theta})$$
$$a = \ddot{\theta}$$

$\zeta$  : tenir l'équilibre le plus longtemps possible

# Apprentissage par renforcement

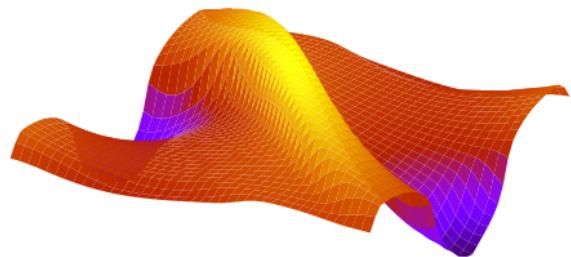
## Fonction valeur



$$x = (\theta, \dot{\theta})$$

$$a = \ddot{\theta}$$

$\zeta$  : tenir l'équilibre le plus longtemps possible



$$\text{plan } (\theta, \dot{\theta})$$

$$z \text{ est } V(x)$$

Maximiser la valeur  $\rightsquigarrow$   
atteindre le sommet de  $V$

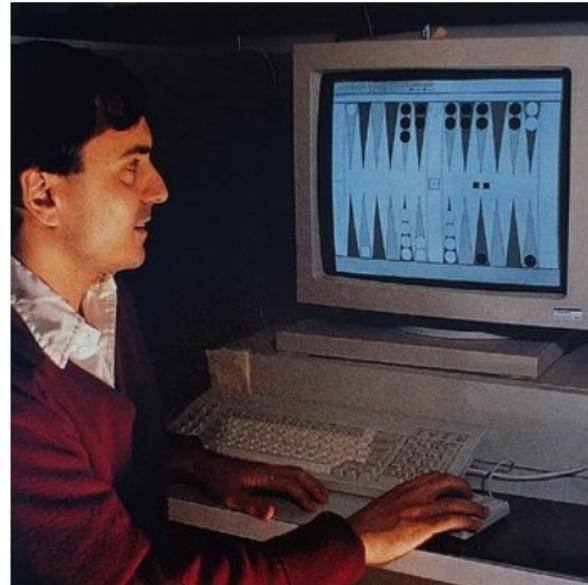
# Apprentissage par renforcement

## Grands espaces d'états : le zoo des approximatrices de fonctions

- ▶ table avec raffinement progressif.
- ▶ forêts aléatoires [Geurts et al., 2006],
- ▶ SVM et machines à noyau,
- ▶ réseaux de neurones [Lin, 1991 ; Tesauro, 1991, 1992, 1994, 1995 ; Riedmiller, 2005 ; ...],
- ▶ ...

# Apprentissage par renforcement

Un premier résultat remarquable : TD-Gammon



# Apprentissage par renforcement

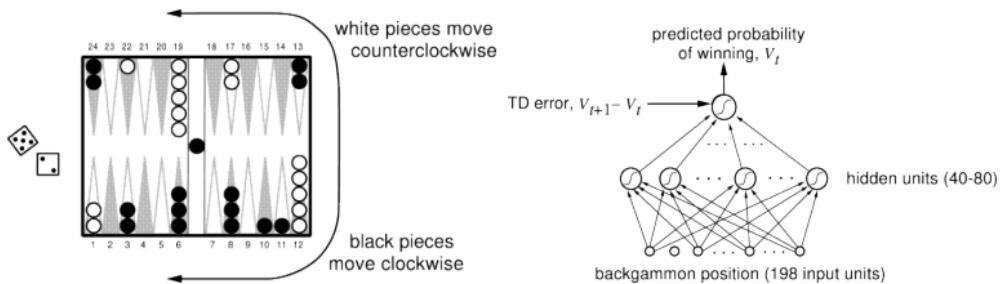
## TD-Gammon

- ▶ Backgammon étudié depuis au moins 1974.
- ▶ Facteur de branchement : 800.
- ▶ TD-Gammon : « successeur de Neurogammon, entraîné par apprentissage supervisé.
- ▶ Représentation de la configuration du plateau de jeu.
- ▶ Entraîné par  $\text{TD}(\lambda)$ .
- ▶ Apprend en jouant contre lui-même.
- ▶ Attributs conçus à la main (pas d'autre connaissance experte).
- ▶ Examine trois coups en avant dans la v3.

Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, 1995

# Apprentissage par renforcement

## TD-Gammon



# Apprentissage par renforcement

DQN : Q-learning avec réseau de neurones

# Apprentissage par renforcement

## DQN : Q-learning avec réseau de neurones

- ▶ Dans le Q-learning, on remplace la table  $Q$  par un réseau de neurones.
- ▶ Ce réseau prend en entrée un état  $x$  et une action  $a$  et produit en sortie  $\hat{Q}(x, a)$

# Apprentissage par renforcement

## DQN

initialiser les paramètres  $\theta$  de la fonction  $Q(x, a)$ ,  $k \leftarrow 0$

**répéter**

$t \leftarrow 0$ ,  $n(x, a) \leftarrow 0$ ,  $\forall (x, a)$

Initialiser l'état de l'agent  $x_t$

**tant-que** épisode non terminé **faire**

**pour**  $k \in \{1, \dots, K\}$  **faire**

sélectionner  $a_t$

effectuer cette action et observer  $r_t$  et  $x_{t+1}$

transition  $[k] \leftarrow (x_t, a_t, r_t, x_{t+1})$

$t++$

**fin pour**

**mettre à jour les**  $\theta$  **pour minimiser l'erreur TD sur les**  $K$  **transitions**

**fin tant-que**

$k++$

**jusque ...**

Permet d'entraîner le réseau sur un lôt de données.

# Apprentissage par renforcement

## DQN : avec *replay buffer*

initialiser les paramètres  $\theta$  de la fonction  $Q(x, a)$ ,  $k \leftarrow 0$

**répéter**

$t \leftarrow 0$ ,  $n(x, a) \leftarrow 0$ ,  $\forall(x, a)$

    Initialiser l'état de l'agent  $x_t$

**tant-que** épisode non terminé **faire**

**pour**  $k \in \{1, \dots, K\}$  **faire**

            sélectionner  $a_t$

            effectuer cette action et observer  $r_t$  et  $x_{t+1}$

**ajouter**  $(x_t, a_t, r_t, x_{t+1})$  aux transitions

$t++$

**fin pour**

    mettre à jour les  $\theta$  pour minimiser l'erreur TD mesurée sur **un échantillon** des transitions

**fin tant-que**

$k++$

**jusque** ...

Rend les données d'entraînement du réseau plus i.i.d.

# Apprentissage par renforcement

## DQN : avec *target network*

```

initialiser deux jeux de paramètres  $\theta'$  et  $\theta''$  pour la fonction  $Q(x, a, \theta)$ ,  $k \leftarrow 0$ 
répéter
     $t \leftarrow 0$ ,  $n(x, a) \leftarrow 0, \forall (x, a)$ 
    Initialiser l'état de l'agent  $x_t$ 
    tant-que épisode non terminé faire
        pour  $k \in \{1, \dots, K\}$  faire
            sélectionner  $a_t$  en utilisant  $\theta'$ 
            effectuer cette action et observer  $r_t$  et  $x_{t+1}$ 
            ajouter  $(x_t, a_t, r_t, x_{t+1})$  aux transitions
             $t++$ 
        fin pour
        mettre à jour les  $\theta''$  pour minimiser l'erreur TD mesurée sur un échantillon des transitions
    fin tant-que
     $k++$ 
    de temps en temps, échanger  $\theta'$  et  $\theta''$ 
jusque ...

```

Rend l'apprentissage plus stable.

# Apprentissage par renforcement

## DQN : avec *target network*

```

initialiser deux jeux de paramètres  $\theta'$  et  $\theta''$  pour la fonction  $Q(x, a, \theta)$ ,  $k \leftarrow 0$ 
répéter
     $t \leftarrow 0$ ,  $n(x, a) \leftarrow 0, \forall (x, a)$ 
    Initialiser l'état de l'agent  $x_t$ 
    tant-que épisode non terminé faire
        pour  $k \in \{1, \dots, K\}$  faire
            sélectionner  $a_t$  en utilisant  $\theta'$ 
            effectuer cette action et observer  $r_t$  et  $x_{t+1}$ 
            ajouter  $(x_t, a_t, r_t, x_{t+1})$  aux transitions
             $t++$ 
        fin pour
        mettre à jour les  $\theta''$  pour minimiser l'erreur TD mesurée sur un échantillon des transitions
    fin tant-que
     $k++$ 
    de temps en temps, échanger  $\theta'$  et  $\theta''$ 
jusque ...

```

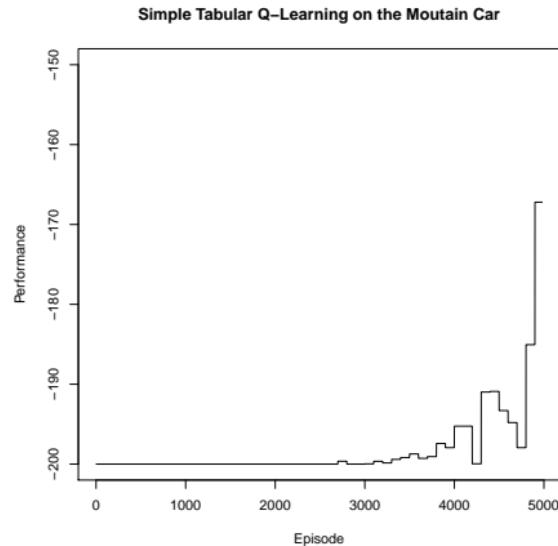
Rend l'apprentissage plus stable.

DQN : + beaucoup d'autres trucs.

# Apprentissage par renforcement

## DQN et la voiture sur la colline

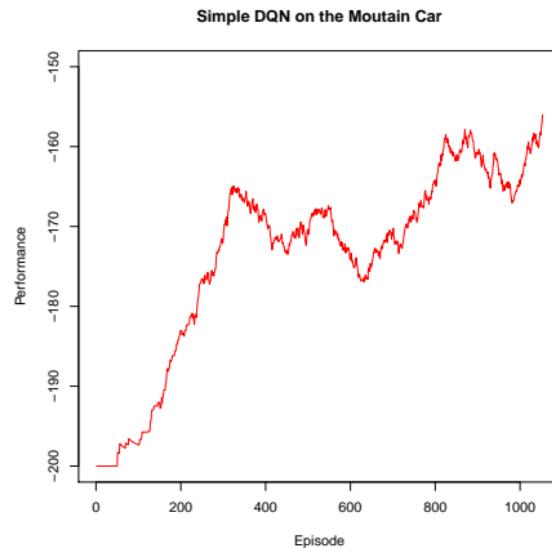
DQN avec un réseau ayant une couche cachée (ReLU).



# Apprentissage par renforcement

## DQN et la voiture sur la colline

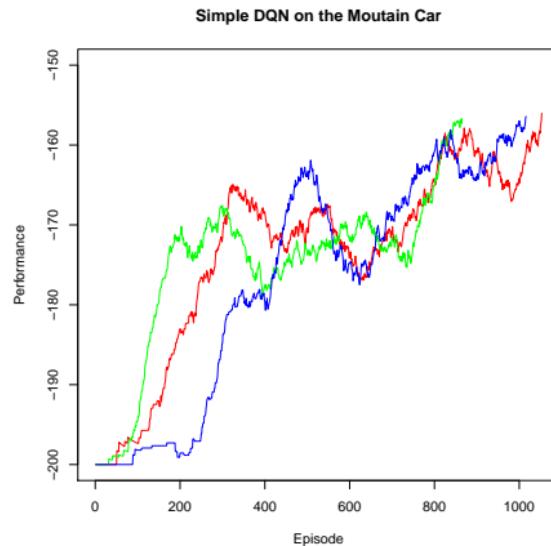
DQN avec un réseau ayant une couche cachée (ReLU).



# Apprentissage par renforcement

## DQN et la voiture sur la colline

DQN avec un réseau ayant une couche cachée (ReLU).

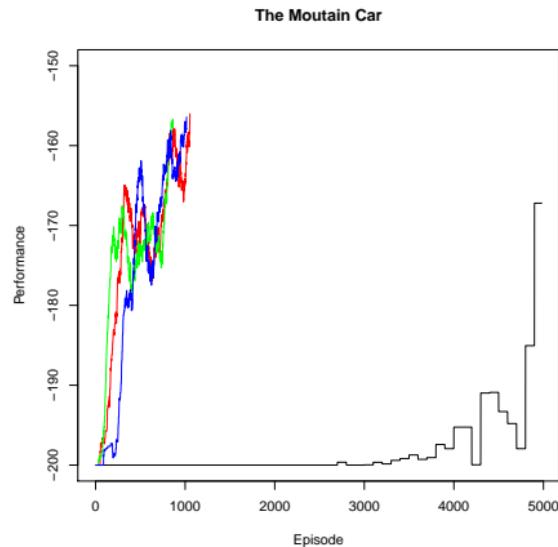


bleu : 32 neurones cachés ; vert : 64 ; rouge : 128

# Apprentissage par renforcement

## DQN et la voiture sur la colline

DQN avec un réseau ayant une couche cachée (ReLU).



# Q-Learning

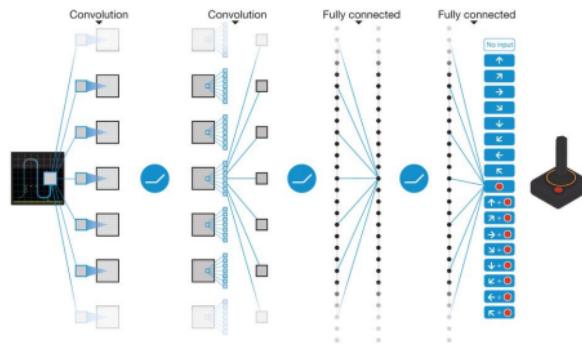
- ▶ la convergence du Q-Learning tabulaire n'est pas facile à montrer.
- ▶ La convergence du Q-Learning avec approximateur de fonctions n'est pas garantie.
- ▶ Avec un approximateur de fonctions,  $\pi^*$  n'est pas forcément déterministe.
- ▶ Les fondements théoriques du Q-Learning sont encore à l'étude ; on peut espérer que la théorie apportera un algorithme plus efficace.

# Apprentissage par renforcement

DQN apprend à jouer aux jeux Atari [Mnih et al. 2015]

Approche de bout-en-bout :

- ▶ état = écran du jeu vidéo (84x84 pixels x4 canaux) et le score
- ▶ sortie = commande du jeu

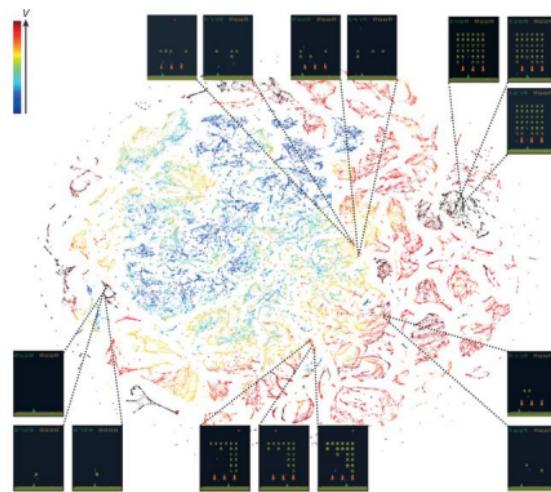


Entraîné sur 49 jeux vidéos différents en même temps.

# Apprentissage par renforcement

DQN apprend à jouer aux jeux Atari [Mnih et al. 2015]

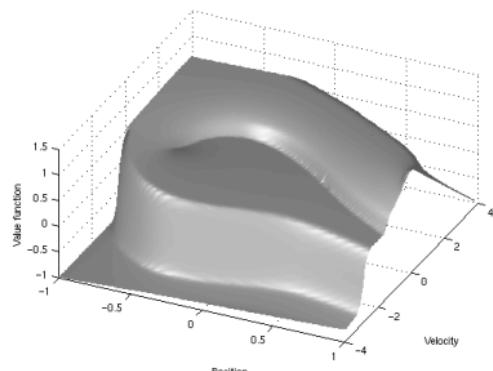
Représentation interne :



Couleur = valeur de l'état.

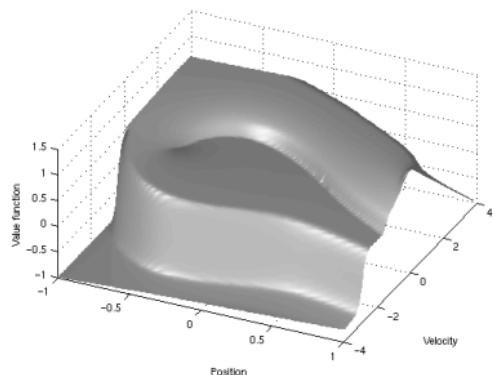
# Apprentissage par renforcement

## Apprentissage direct de politique

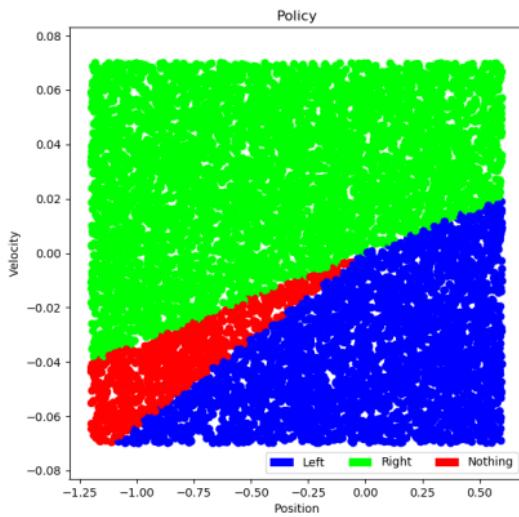
 $V^*$  $\pi^*$

# Apprentissage par renforcement

## Apprentissage direct de politique



$$V^*$$



$$\pi^*$$

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶ Un réseau représente une politique  $\pi(x, a) : \theta \rightsquigarrow \pi_\theta$ .
  - ▶ Entrée du réseau = état  $x$ .
  - ▶  $|\mathcal{A}|$  sorties du réseau = probabilités d'émettre chacune des actions  $a$  dans  $x$

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶ Un réseau représente une politique  $\pi(x, a) : \theta \rightsquigarrow \pi_\theta$ .
  - ▶ Entrée du réseau = état  $x$ .
  - ▶  $|\mathcal{A}|$  sorties du réseau = probabilités d'émettre chacune des actions  $a$  dans  $x$
- ▶ Comment mettre à jour  $\theta$  avec des informations de transitions ?

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶ Un réseau représente une politique  $\pi(x, a) : \theta \rightsquigarrow \pi_\theta$ .
  - ▶ Entrée du réseau = état  $x$ .
  - ▶  $|\mathcal{A}|$  sorties du réseau = probabilités d'émettre chacune des actions  $a$  dans  $x$
- ▶ Comment mettre à jour  $\theta$  avec des informations de transitions ?
- ▶ Réponse : comme précédemment : DGS :  $\theta_{t+1} \leftarrow \theta_t + \alpha \widehat{\nabla_\theta \zeta(\theta_t)}$

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶ Un réseau représente une politique  $\pi(x, a) : \theta \rightsquigarrow \pi_\theta$ .
  - ▶ Entrée du réseau = état  $x$ .
  - ▶  $|\mathcal{A}|$  sorties du réseau = probabilités d'émettre chacune des actions  $a$  dans  $x$
- ▶ Comment mettre à jour  $\theta$  avec des informations de transitions ?
- ▶ Réponse : comme précédemment : DGS :  $\theta_{t+1} \leftarrow \theta_t + \alpha \widehat{\nabla_\theta \zeta(\theta_t)}$
- ▶ Théorème du gradient de politique [Sutton et al., 2000] :  
$$\widehat{\nabla_\theta \zeta(\theta_t)} \propto \sum_x \mu(x) \sum_a Q^\pi(x, a) \nabla_\theta \pi(a|s, \theta)$$

# Apprentissage par renforcement

REINFORCE [Williams, 1991 ; Sutton et al. 2000]

Apprentissage direct de politique pour une tâche épisodique :

initialiser les poids  $\theta$  du réseau  $\pi_\theta$

**répéter**

effectuer un épisode et collecter les  $(x_t, a_t, r_t, x_{t+1})$

**pour** chaque pas  $t$  de l'épisode **faire**

$$G_t \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} r_k$$

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln(\pi(x_t, a_t))$$

**fin pour**

**jusque ...**

# Apprentissage par renforcement

## Apprentissage direct de politique vs. DQN

- ▶ REINFORCE optimise directement la politique

# Apprentissage par renforcement

## Apprentissage direct de politique vs. DQN

- ▶ REINFORCE optimise directement la politique
- ▶ REINFORCE nécessite de nouvelles trajectoires (interactions avec l'environnement) pour chaque mise à jour des  $\theta$ .

# Apprentissage par renforcement

## Apprentissage direct de politique vs. DQN

- ▶ REINFORCE optimise directement la politique
- ▶ REINFORCE nécessite de nouvelles trajectoires (interactions avec l'environnement) pour chaque mise à jour des  $\theta$ .
- ▶ pas vraiment de raison théorique pour dire que REINFORCE ou DQN est meilleur d'un point de vue pratique : il faut essayer.

# Apprentissage par renforcement

## Apprentissage direct de politique vs. DQN

- ▶ REINFORCE optimise directement la politique
- ▶ REINFORCE nécessite de nouvelles trajectoires (interactions avec l'environnement) pour chaque mise à jour des  $\theta$ .
- ▶ pas vraiment de raison théorique pour dire que REINFORCE ou DQN est meilleur d'un point de vue pratique : il faut essayer.
- ▶ dans REINFORCE, le gradient est proportionnel à  $G$  qui peut varier beaucoup d'un épisode à un autre.  
Cela rend l'apprentissage instable.  
Solution classique consiste à soustraire une valeur constante aux termes  $G_t \rightsquigarrow (G_t - b)$

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶  $G_t \rightsquigarrow (G_t - b)$  : choix de  $b$  ?

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶  $G_t \rightsquigarrow (G_t - b)$  : choix de  $b$  ?
- ▶  $V(x_t)$

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶  $G_t \rightsquigarrow (G_t - b)$  : choix de  $b$  ?
- ▶  $V(x_t)$
- ▶ Nécessite une représentation de  $V$  paramétrée par des  $w$

# Apprentissage par renforcement

## Apprentissage direct de politique

- ▶  $G_t \rightsquigarrow (G_t - b)$  : choix de  $b$  ?
- ▶  $V(x_t)$
- ▶ Nécessite une représentation de  $V$  paramétrée par des  $w$
- ▶ Algorithme REINFORCE avec ligne de base :

initialiser les poids  $\theta$  du réseau  $\pi_\theta$  et  $w$  pour  $V_w$

**répéter**

effectuer un épisode et collecter les  $(x_t, a_t, r_t, x_{t+1})$

**pour** chaque pas  $t$  de l'épisode **faire**

$$G_t \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} r_k$$

$$\delta_t \leftarrow G_t - V_w(x_t)$$

$$w \leftarrow w + \alpha_w \delta_t \nabla V_w(x_t)$$

$$\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta_t \nabla_\theta \ln(\pi(x_t, a_t))$$

**fin pour**

**jusque** ...

qui a une variance réduite.

# Apprentissage par renforcement

## Apprentissage direct de politique

Algorithme REINFORCE avec ligne de base :

initialiser les poids  $\theta$  du réseau  $\pi_\theta$  et  $w$  pour  $V_w$

**répéter**

effectuer un épisode et collecter les  $(x_t, a_t, r_t, x_{t+1})$

**pour** chaque pas  $t$  de l'épisode **faire**

$$G_t \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} r_k$$

$$\delta_t \leftarrow G_t - V_w(x_t)$$

$$w \leftarrow w + \alpha_w \delta_t \nabla V_w(x_t)$$

$$\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta_t \nabla_\theta \ln(\pi(x_t, a_t))$$

**fin pour**

**jusque ...**

# Apprentissage par renforcement

## Apprentissage direct de politique

Algorithme acteur-critique :

apprendre  $V$  par différence temporelle :

initialiser les poids  $\theta$  du réseau  $\pi_\theta$  et  $w$  pour  $V_w$

**répéter**

initialiser  $x_0$ ,  $t \leftarrow 0$

**répéter**

choisir  $a_t \sim \pi_\theta(x_t)$ , l'émettre et collecter  $(x_t, a_t, r_t, x_{t+1})$

$\delta_t \leftarrow r_t + \gamma V_w(x_{t+1} - V_w(x_t))$

$w \leftarrow w + \alpha_w \delta_t \nabla V_w(x_t)$

$\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta_t \nabla_\theta \ln(\pi(x_t, a_t))$

$t \leftarrow t + 1$

**jusque**  $x_t$  est terminal

**jusque** ...

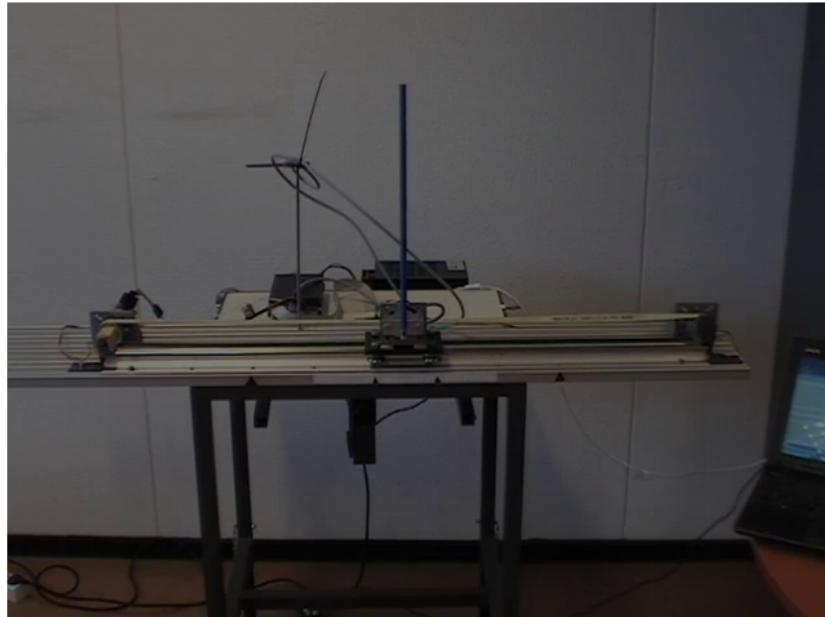
# Apprentissage par renforcement

## Avec modèle

- ▶ pour apprendre, l'AR est lent.
- ▶ L'AR n'offre aucune garantie de comportement : inacceptable pour de nombreuses applications (robotique, ...).
- ▶ intégration d'information « experte »
- ▶ utilisation d'un modèle (imparfait) de l'environnement
- ▶ apprentissage d'un modèle (imparfait) de l'environnement

# Apprentissage par renforcement

## Application en robotique



Riedmiller, Neural reinforcement learning to swing-up and balance a real pole, *Proc. 2005 IEEE International Conference on Systems, Man and Cybernetics*

# Apprentissage par renforcement

## Application en robotique



Lauer et al., Cognitive Concepts in Autonomous Soccer Playing Robots, *Cognitive Systems Research*, 11(3), 287 :309, September, 2010  
(Pas d'apprentissage profond ! Un simple MLP.)

# Apprentissage par renforcement

## Application : Guesswhat ? !

- ▶ Objectif : apprendre à dialoguer en langage naturel.
- ▶ Généralement, les systèmes de dialogue en langage naturel sont basés sur des règles ou entraînés de manière supervisée.  
Ici, on le fait par AR.

# Apprentissage par renforcement

## Application : Guesswhat ? !

- ▶ Jeu à 2 joueurs : un oracle et un joueur
- ▶ Oracle : choisit un objet dans une image
- ▶ Joueur : doit localiser l'objet dans l'image en posant des questions oui-non en langage naturel, et l'oracle doit y répondre.



Is it a person? **No**  
 Is it an item being worn or held? **Yes**  
 Is it a snowboard? **Yes**  
 Is it the red one? **No**  
 Is it the one being held by the person in blue? **Yes**



Is it a cow? **Yes**  
 Is it the big cow in the middle? **No**  
 Is the cow on the left? **No**  
 On the right? **Yes**  
 First cow near us? **Yes**

- ▶ Entraîné sur 70k images, 134 k objets uniques, 800k paires Q&R.

# Apprentissage par renforcement

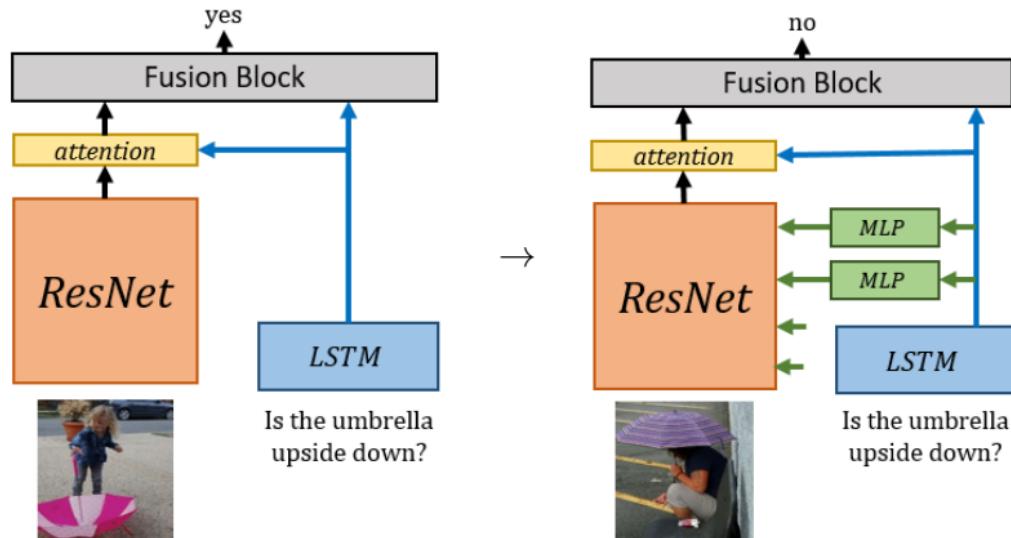
Application : Guesswhat ? !

- ▶ De bout-en-bout : pngs  $\mapsto$  dialogues
- ▶ AR a de meilleures performances que l'apprentissage supervisé
- ▶ Apprendre à dialoguer en langage naturel au travers d'une image
- ▶ Apprentissage par renforcement profond
- ▶ Combine vision et langage : Resnet + LSTM

# Apprentissage par renforcement

Application : Guesswhat ?! : modulation de la vision par le langage

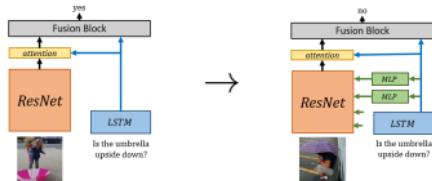
- ▶ La vision est modulée par le langage :



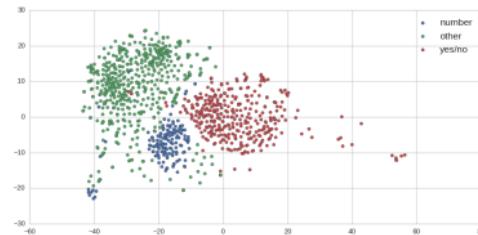
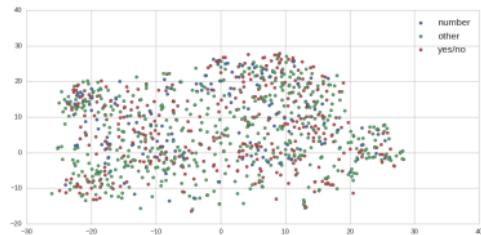
# Apprentissage par renforcement

Application : Guesswhat ?! : modulation de la vision par le langage

- ▶ La vision est modulée par le langage :



- ▶ La modulation améliore la représentation interne :



<https://guesswhat.ai/>

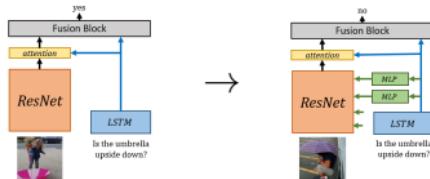
Financé en partie par  IGLU  
Interactive Grounded Language Understanding

en collaboration avec  MILA, et des chercheurs de Deepmind et de Google Brain.

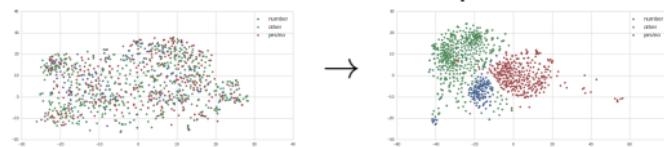
# Apprentissage par renforcement

Application : Guesswhat ?! : modulation de la vision par le langage

- ▶ La vision est modulée par le langage :



- ▶ La modulation améliore la représentation interne :



- ▶ Modulation may be used beyond vision : any “signal” might be modulated.

<https://guesswhat.ai/>

Financé en partie par  IGLU  
Interactive Grounded Language Understanding

en collaboration avec  MILA, et des chercheurs de Deepmind et de Google Brain.

# Apprentissage par renforcement

## Applications au développement durable

- ▶ contrôle de *smartgrids*
- ▶ apprentissage de pratiques en agriculture

# Queqlues sujets chauds en RL

- ▶ apprentissage hiérarchique ; apprentissage d'options
- ▶ apprentissage de représentation
- ▶ généralisation
- ▶ environnements variant au fil du temps (non stationnaires)
- ▶ transfert de compétences
- ▶ apprentissage continu
- ▶ explicabilité/transparence d'un comportement appris
- ▶ la théorie est en retard par rapport à la pratique mais quand elle s'applique, la théorie a un impact significatif sur la conception et l'implantation d'algorithmes

# Take home message

- ▶ De nombreux problèmes peuvent bénéficier d'un point de vue « prise de décision séquentielle dans l'incertain ».



Pas seulement les jeux.

# Take home message

- ▶ De nombreux problèmes peuvent bénéficier d'un point de vue « prise de décision séquentielle dans l'incertain ».



Pas seulement les jeux.

- ▶ L'apprentissage par renforcement est plus performant que l'apprentissage supervisé.

# Bibliographie

- ▶ Bertsekas, *Neurodynamic Programming*, Athena Scientific, 1996
- ▶ Bertsekas, *Dynamic programming and optimal control*, Athena Scientific, 2017
- ▶ Bertsekas, *Reinforcement Learning and Optimal Control*, 2019
- ▶ Goodfellow et al., *The Deep Learning Book*, MIT Press, 2016, [thedeeplearningbook.org](http://the-deep-learning-book.org)
- ▶ François-Lavet et al., *An Introduction to Deep Reinforcement Learning*, Foundations and Trends in Machine Learning : Vol. 11, No. 3-4.,  
<https://arxiv.org/pdf/1811.12560.pdf>
- ▶ Powell, *Approximate Dynamic Programming*, John Wiley and Sons, 2nd edition
- ▶ Puterman, *Markov decision processes*, Wiley, 1994
- ▶ Sutton, Barto, *Reinforcement Learning*, 2nd ed, MIT Press, 2018,  
<http://incompleteideas.net/book/the-book-2nd.html>
- ▶ Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, 1995

# Bibliographie

- ▶ [Bellman, 1957] R.E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.
- ▶ [Howard, 1960] R.A. Howard, *Dynamic Programming and Markov Processes*, The M.I.T. Press, 1960.
- ▶ [Le Cun, 1987] Y. Le Cun, *Modeles connexionnistes de l'apprentissage*, thèse de doctorat, U. Paris 6
- ▶ [Mnih et al., 2015] V. Mnih et al., Human-level control through deep reinforcement learning, *Nature*, Février 2015
- ▶ [Munos, Moore, 2002] R. Munos, A. Moore, Variable Resolution Discretization in Optimal Control, *Machine Learning Journal*, **49**, 291 :323.
- ▶ [Rumelhart et al. 1986] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning Internal Representations by Error Propagation, In *Parallel Distributed Processing : Explorations in the Microstructure of Cognition. Volume 1 : Foundations*, chapitre 8. Cambridge : MIT Press.
- ▶ [Sutton, 1984] R.S. Sutton, *Temporal credit assignment in reinforcement learning*, Ph.D. dissertation, Department of Computer Science, University of Massachusetts.
- ▶ [Sutton et al. 2000] R.S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems 1999*, pp. 1057-1063.
- ▶ [Werbos, 1974] P. Werbos, *Beyond regression : new tools for prediction and analysis in the behavioral sciences*, PhD dissertation, Harvard University
- ▶ [Williams, 1991] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine Learning*, **8**(3-4) :229–256.