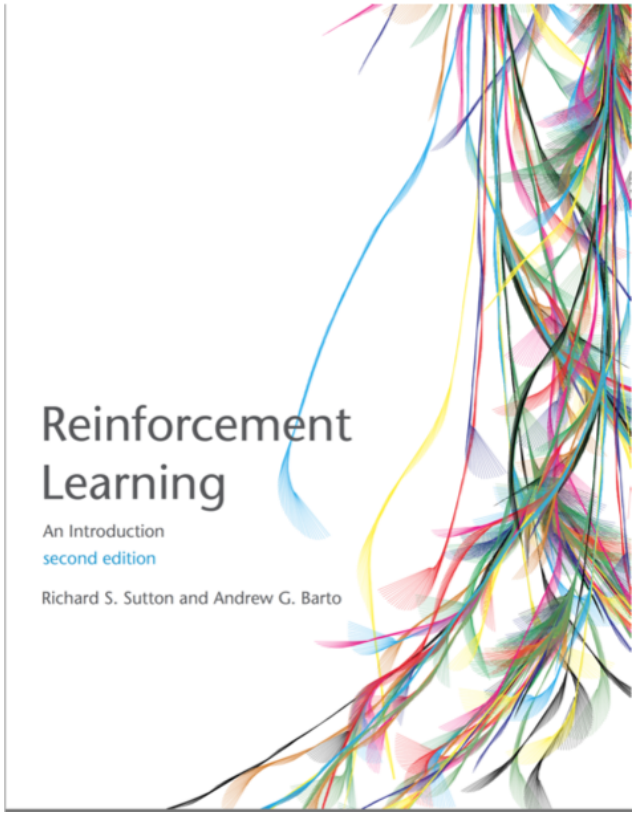
Apprentissage par renforcement

Grégoire Passault

March 31, 2022

1. Introduction
   - Formalisme
   - Programmation dynamique
2. Apprentissage sans modèle
   - Monte Carlo
   - TD et Q-Learning
3. Montée à l'échelle
   - Deep reinforcement learning (DQN)
   - Cas du continu (DDPG)
4. Des problématiques
   - Curriculum learning
   - Sur de vrais robots

Ressources: Le livre de référence



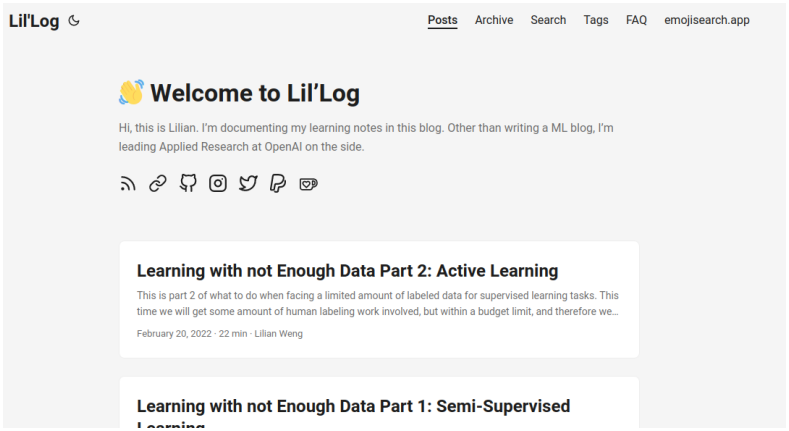
R. S. Sutton and A. G. Barto (1998)
Reinforcement Learning : An Introduction.
MIT Press, 1998

Ressources: Cours sur YouTube

David Silver

Chaîne YouTube "DeepMind"

Ressources: Blog de Lilian Weng



Lil'Log

`https://lilianweng.github.io/`

Blog très complet (revues de litératures)

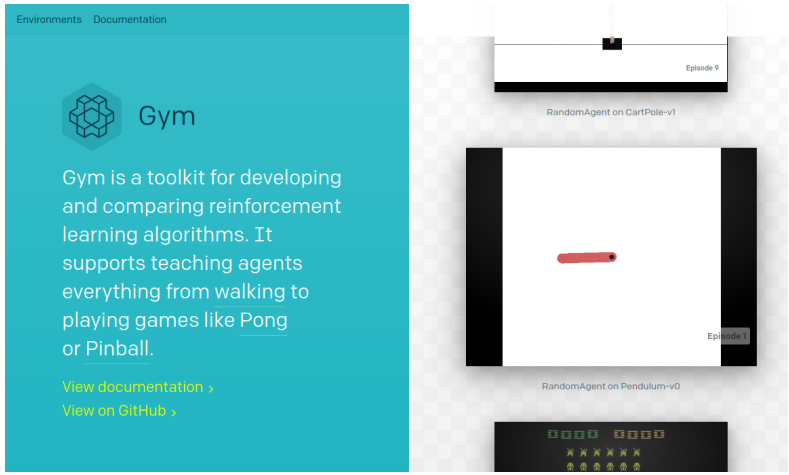
Ressources: Bibliothèques

Stable-Baselines3

`https://stable-baselines3.readthedocs.io/en/master/`

Bibliothèque RL (avec PyTorch)

Ressources: Gym OpenAI



OpenAI gym

`https://gym.openai.com/`

Formaliser des environnements

Motivation

La robotique industrielle est gouvernée par des robots:

- Dotés d'encodeurs très précis,
- De réducteurs sans jeu (par exemple harmonique),
- En somme, très fidèles à leur **modèle physique**.

Motivation

Cependant:

- Il n'est pas toujours facile d'avoir le modèle complet d'un robot (robot à bas coût, jeu, encodeurs imprécis, pièces tordues...)
- Même en connaissant parfaitement le modèle, ça ne résout pas le problème de planification (exploitation) du modèle.

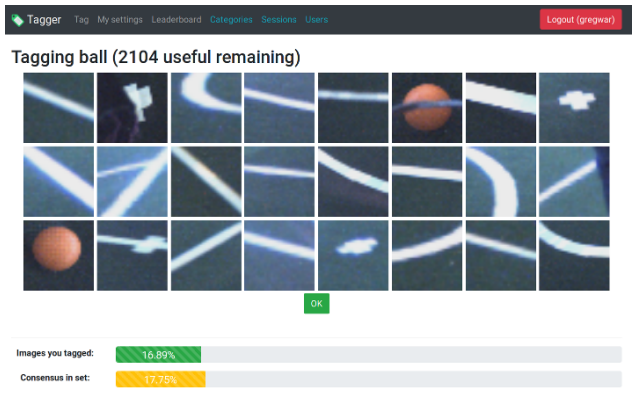
Model-based ou *Data-based*

Un robot peut interagir avec son environnement. On aimerait pouvoir exploiter ces interactions pour améliorer son comportement

Catégories d'apprentissage automatique

Il existe plusieurs catégories d'apprentissage:

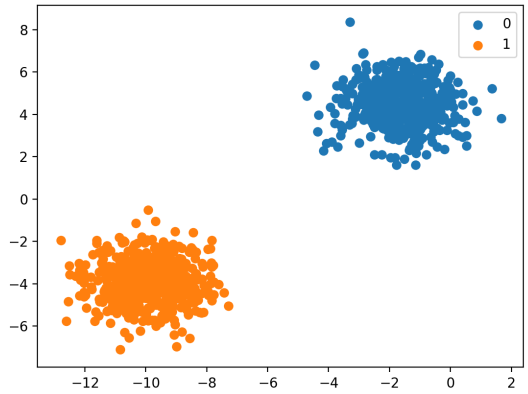
- **Apprentissage supervisé**: On apprend sur des données étiquetées avec une vérité terrain
- **Apprentissage non supervisé**: On dispose uniquement des données, et on essaie d'en comprendre des caractéristiques
- **Apprentissage par renforcement**: On veut maximiser un signal de futures récompenses (nous allons en reparler!)



Catégories d'apprentissage automatique

Il existe plusieurs catégories d'apprentissage:

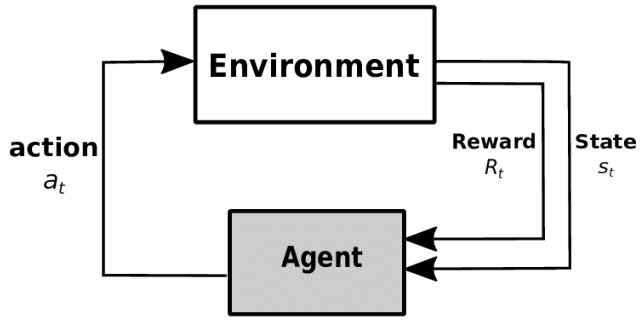
- **Apprentissage supervisé**: On apprend sur des données étiquetées avec une vérité terrain
- **Apprentissage non supervisé**: On dispose uniquement des données, et on essaie d'en comprendre des caractéristiques
- **Apprentissage par renforcement**: On veut maximiser un signal de futures récompenses (nous allons en reparler!)



Catégories d'apprentissage automatique

Il existe plusieurs catégories d'apprentissage:

- **Apprentissage supervisé**: On apprend sur des données étiquetées avec une vérité terrain
- **Apprentissage non supervisé**: On dispose uniquement des données, et on essaie d'en comprendre des caractéristiques
- **Apprentissage par renforcement**: On veut maximiser un signal de futures récompenses (nous allons en reparler!)



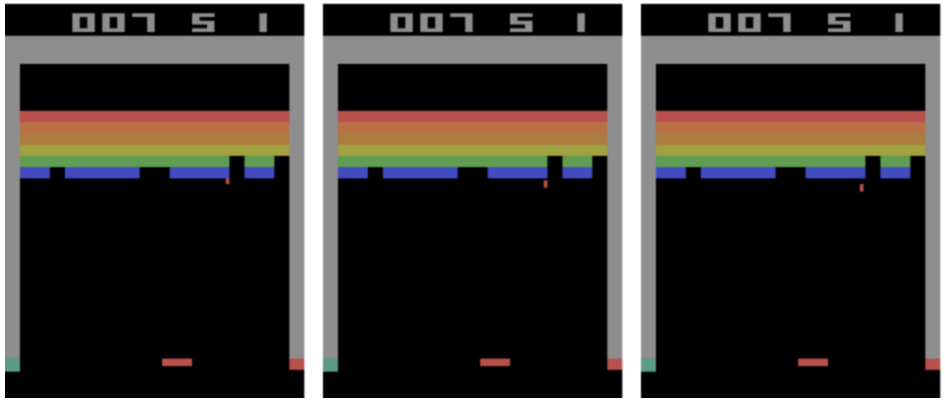
Success story

AlphaGo, puis AlphaGo zero (DeepMind)

Capable de battre le grand maître humain au jeu de Go en utilisant une base de données de parties pour l'entraînement.

Variante "Zero", capable d'apprendre en jouant uniquement contre elle-même.

Success story



Apprentissage automatique de jeux d'Atari (DeepMind)

En utilisant l'apprentissage par renforcement (mariée au deep learning), le même algorithme était capable d'apprendre à jouer à une multitude de jeux, à partir de l'image, la manette et le score.

Success story

Victoire contre des grand maître StarCraft (DeepMind)

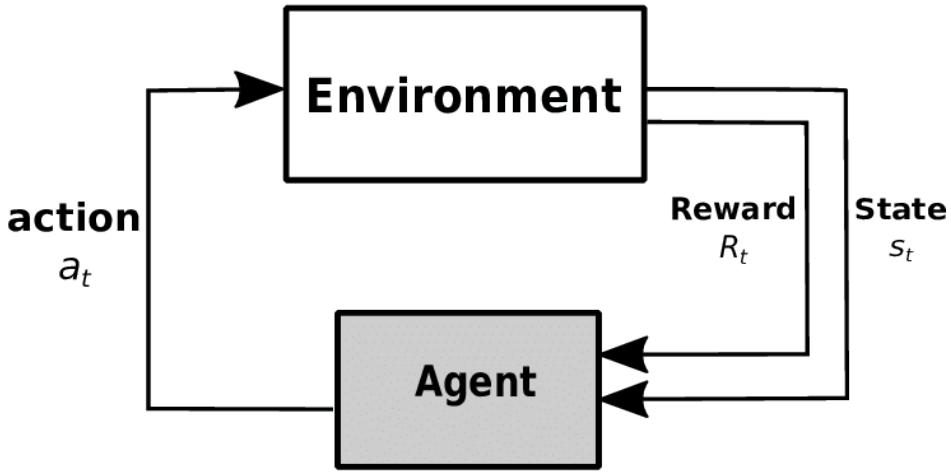
Dans un des jeux les plus compétitifs du monde, l'IA de Google s'est élevée au rang de grand maître.

Success story

Résolution de Rubik's Cube avec une main robotique (OpenAI)

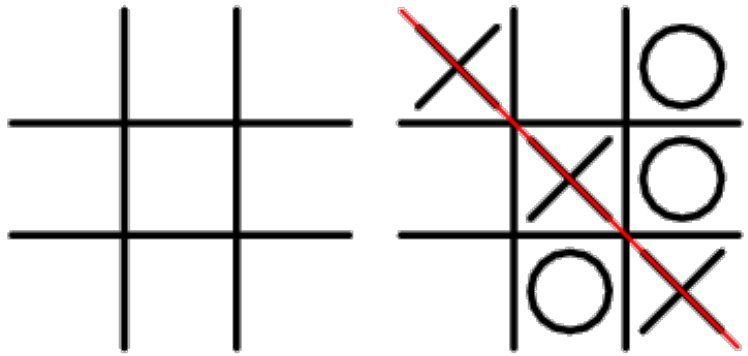
La manipulation du cube est apprise par de l'apprentissage par renforcement, à partir d'abord de simulation, puis sur le vrai robot.

Formalisme



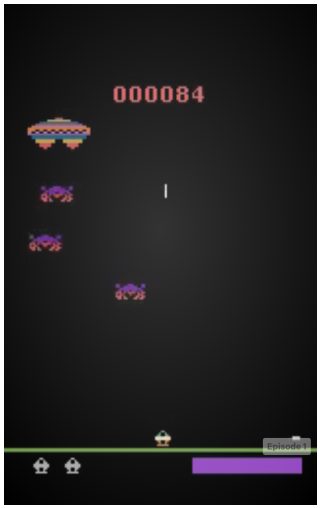
L'apprenant (qu'on appelle agent) est plongé dans un environnement avec lequel il interagit. À chaque étape, l'agent est dans un état s_t, il décide alors de son action a_t et observe une récompense r_{t+1} et un nouvel état s_{t+1}.

Exemple: morpion



- La récompense est latente (gagné / perdu), jouer un coup change uniquement d'état
- Dans les approches classiques (min/max etc.), on explore l'arbre des possibilités futures, mais pourrait-on apprendre juste en jouant?

Exemple: jouer à un jeu vidéo



- Que représentent sémantiquement les éléments à l'écran ?
- Quel effet ont les boutons de la manette sur le jeu ?
- Anticiper les états futurs dans lesquels on souhaite aller ou éviter
- Certains jeux sont stochastiques (ex: jeter un dé)

Exemple: piloter un Kart

- Difficile de connaître tous les paramètres physiques
- On peut modéliser le système par de la physique... mais aussi utiliser des expériences réelles à la place

Motivation

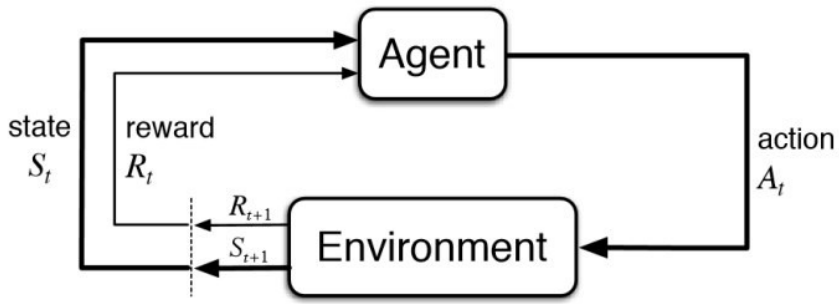
Une des motivations de l'apprentissage par renforcement est la capacité d'apprendre en **interaction avec l'environnement**, et aussi le mélange de l'apprentissage et des actions visant à mieux **découvrir l'environnement** (le compromis exploration/exploitation dont nous avons parlé).

Les problèmes seront décrit selon un certain formalisme (que nous allons détailler).

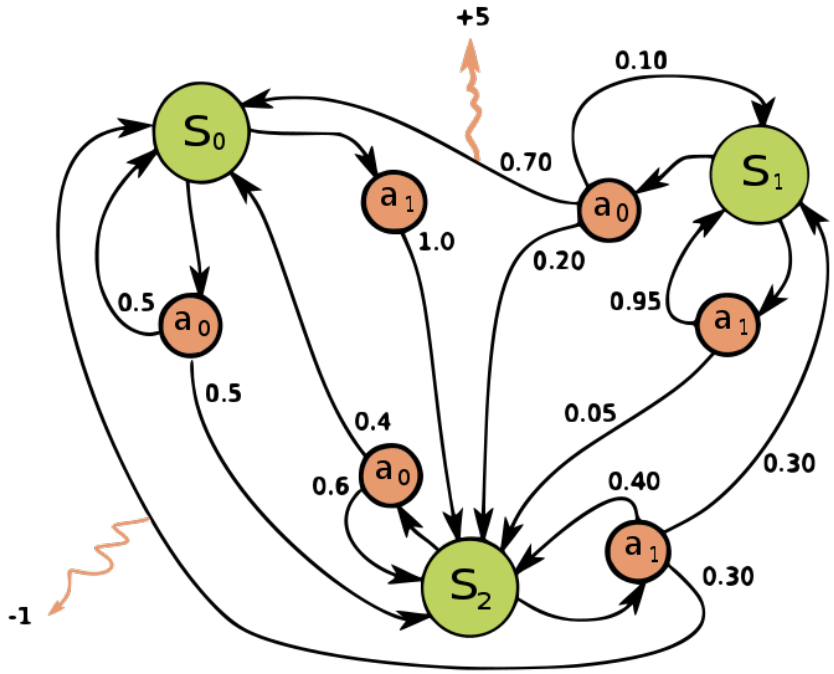
Formalisme: MDP

On note:

- S, un ensemble d'états,
- A, un ensemble d'actions,
- $P(s'|s,a)$, le "modèle" de l'environnement: la probabilité d'arriver dans l'état s' après avoir effectué l'action a depuis l'état s (il peut être connu ou inconnu)
- $R^a_{s,s'}$, une fonction de récompense: obtenue en appliquant l'action a depuis l'état s et d'être arrivé dans l'état s'.



Formalisme: MDP



https://en.wikipedia.org/wiki/Markov_decision_process

Formalisme: politique

On appelle politique π (*policy* en anglais) la "stratégie" de notre agent, qui prend un état s et fournit l'action à effectuer $\pi(s)$.

Notre but est de produire une politique optimale. Pour formaliser "optimal", on veut avoir la plus grande somme de récompenses futures obtenues:

$$J(\pi) = \mathbb{E}_{s\sim E}[\sum_{t=1}^{\infty} R^{a_t}_{s_t,s_{t+1}}]$$

Où:

- $a_t = \pi(s_t)$ est la décision que l'on prend dans l'état s_t,
- s_{t+1} est l'état dans lequel on arrive après avoir pris l'action a_t dans l'état s_t.

Formalisme: politique

On appelle politique π (*policy* en anglais) la "stratégie" de notre agent, qui prend un état s et fournit l'action à effectuer $\pi(s)$.

Notre but est de produire une politique optimale. Pour formaliser "optimal", on veut avoir la plus grande somme de récompenses futures obtenues:

$$J(\pi) = \mathbb{E}_{s\sim E}[\sum_{t=1}^{\infty} R^{a_t}_{s_t,s_{t+1}}]$$

Où:

- $a_t = \pi(s_t)$ est la décision que l'on prend dans l'état s_t,
- s_{t+1} est l'état dans lequel on arrive après avoir pris l'action a_t dans l'état s_t.

Fonction de valeur

On appelle fonction de valeur la fonction V^{π} qui représente toutes les récompenses futures à partir d'un état s en suivant une politique π donnée:

$$V^{\pi}(s) = \mathbb{E}_{s \sim E}[\sum_{t=1}^{\infty} R^{a_t}_{s_t, s_{t+1}} | s_1 = s]$$

Où

- a_t est l'action sélectionnée par la politique à l'étape t
- s_{t+1} est l'état dans lequel on arrive après avoir sélectionné l'action a_t dans l'état s_t.

Fonction de valeur (2)

On peut remarquer que la fonction de valeur est récursive (on l'appelle l'équation de Bellman):

$$V^{\pi}(s) = \mathbb{E}_{s_{t+1} \sim E}[R^{\pi(s_t)}_{s_t, s_{t+1}} + V^{\pi}(s_{t+1})]$$

$$V^{\pi}(s) = \sum_{s'} P(s_{t+1} = s' | s_t, a_t)[R^{a_t}_{s_t, s'} + V^{\pi}(s')]$$

Avec $a_t = \pi(s)$ choisi à l'aide de la politique

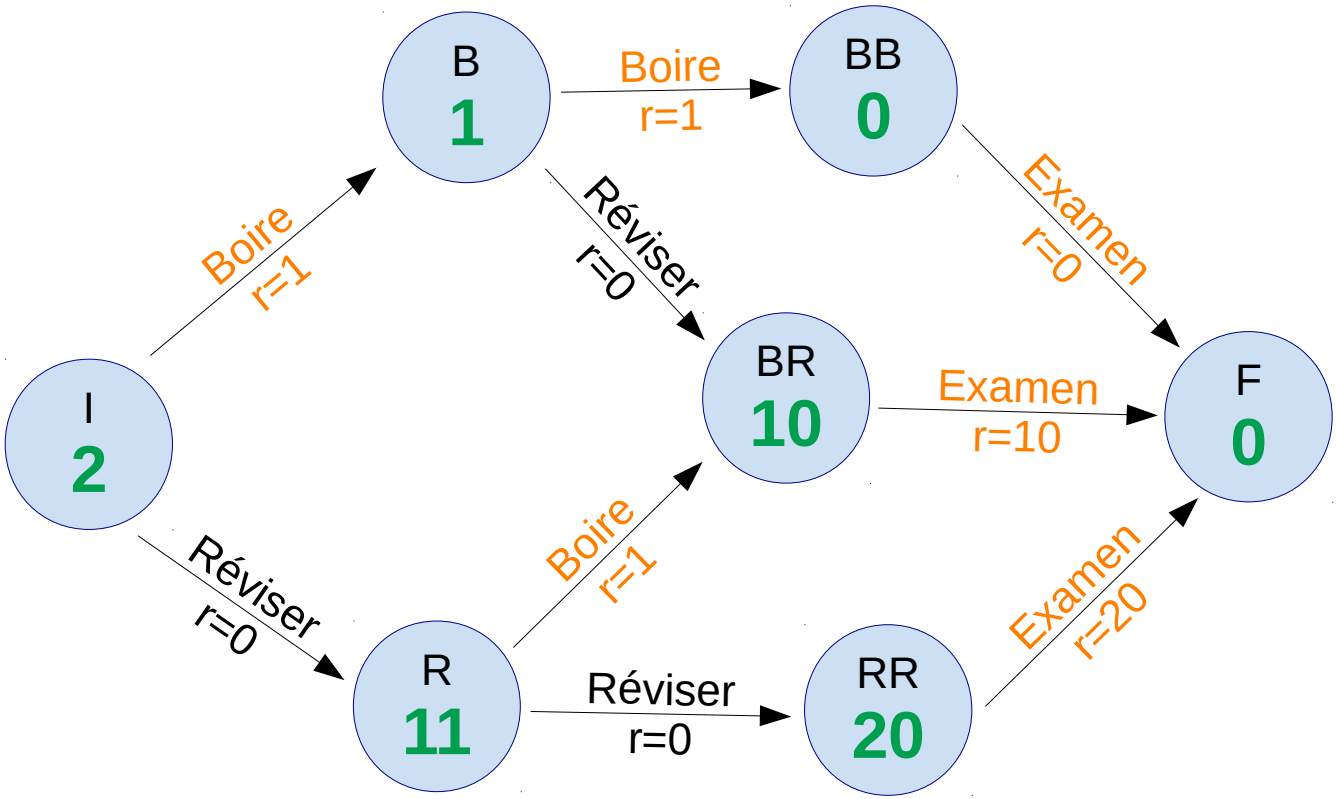
- Problème: on ne donne aucune importance à la récompense court terme
- Problème: dans un épisode sans terminaison, cette valeur peut tendre vers l'infini

Solution: introduire un terme d'escompte $\gamma \in [0, 1]$:

$$V^{\pi}(s) = \sum_{s'} P(s_{t+1} = s' | s_t, a_t)[R^{a_t}_{s_t, s'} + \gamma V^{\pi}(s')]$$

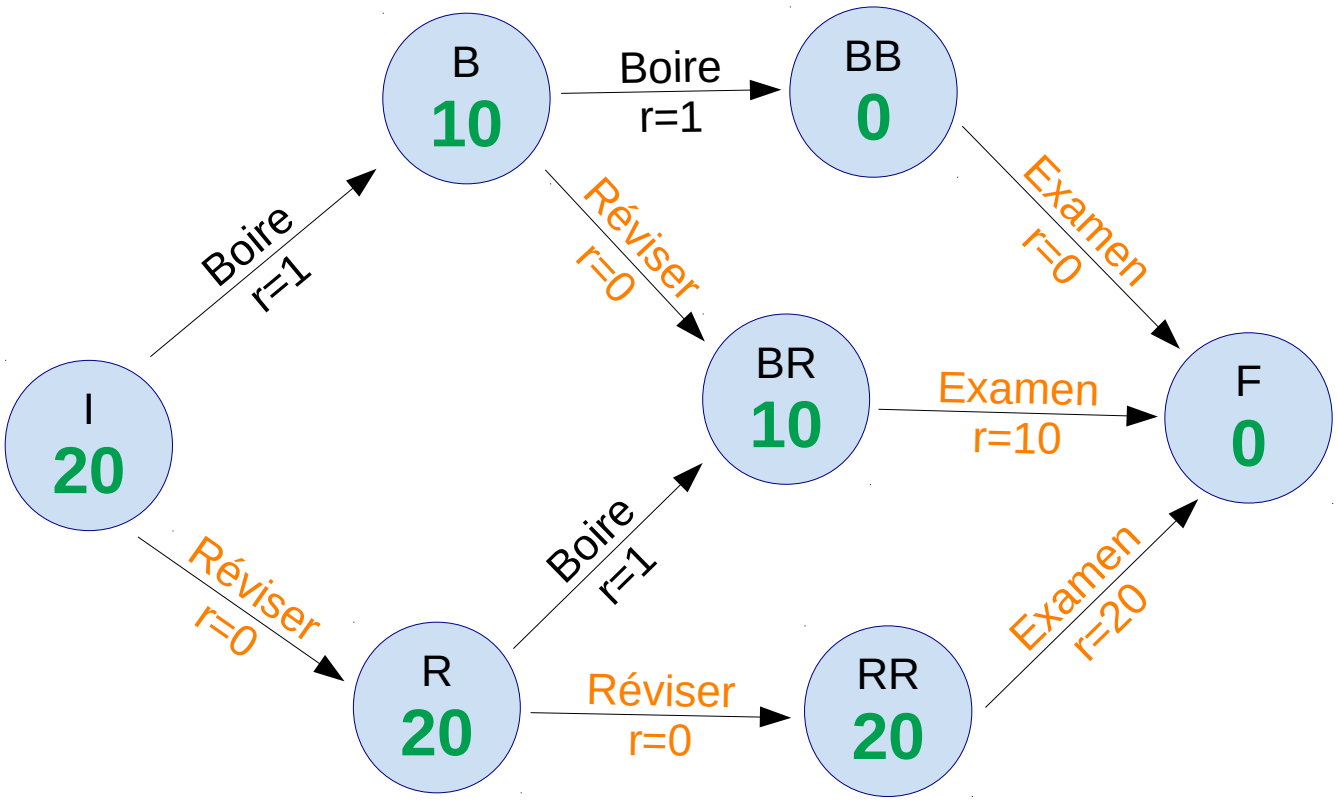
Un exemple

Si $\gamma = 1$, et que la politique suit les actions en orange, la fonction de valeur est:



Un exemple

Pour une autre politique, la fonction de valeur est différente!



Propriété de Markov

Propriété de Markov

"Given the present, the future does not depend on the past"

$$P(S_{t+1}|S_t, S_{t-1}, ...S_1) = P(S_{t+1}|S_t)$$

Quelles que soient les actions passées, la politique optimale reste la même.

Politique optimale

Une politique (π') est meilleure qu'une autre (π) si $\forall s, V^{\pi'}(s) \geq V^{\pi}(s)$.

Il existe une politique optimale π^*, qui est meilleure que toutes les autres $\forall \pi, s, [V^{\pi^*}(s) \geq V^{\pi}(s)]$

- Notons V^* la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise $\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V^*(s')]$. Si on connaît V^*, on peut donc trouver π^*.
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

Politique optimale

Une politique (π') est meilleure qu'une autre (π) si $\forall s, V^{\pi'}(s) \geq V^{\pi}(s)$.

Il existe une politique optimale π^*, qui est meilleure que toutes les autres $\forall \pi, s, [V^{\pi^*}(s) \geq V^{\pi}(s)]$

- Notons V^* la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise $\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V^*(s')]$. Si on connaît V^*, on peut donc trouver π^*.
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

Politique optimale

Une politique (π') est meilleure qu'une autre (π) si $\forall s, V^{\pi'}(s) \geq V^{\pi}(s)$.

Il existe une politique optimale π^*, qui est meilleure que toutes les autres $\forall \pi, s, [V^{\pi^*}(s) \geq V^{\pi}(s)]$

- Notons V^* la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise $\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V^*(s')]$. Si on connaît V^*, on peut donc trouver π^*.
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

Politique optimale

Une politique (π') est meilleure qu'une autre (π) si $\forall s, V^{\pi'}(s) \geq V^{\pi}(s)$.

Il existe une politique optimale π^*, qui est meilleure que toutes les autres $\forall \pi, s, [V^{\pi^*}(s) \geq V^{\pi}(s)]$

- Notons V^* la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise $\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V^*(s')]$. Si on connaît V^*, on peut donc trouver π^*.
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

Politique optimale

Une politique (π') est meilleure qu'une autre (π) si $\forall s, V^{\pi'}(s) \geq V^{\pi}(s)$.

Il existe une politique optimale π^*, qui est meilleure que toutes les autres $\forall \pi, s, [V^{\pi^*}(s) \geq V^{\pi}(s)]$

- Notons V^* la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise $\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V^*(s')]$. Si on connaît V^*, on peut donc trouver π^*.
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

Politique gourmande

Pour une fonction de valeur donnée V, on peut définir la politique **gourmande** $\hat{\pi}$ qui consiste à prendre l'action a qui maximise $\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V(s')]$.

$$\hat{\pi}^V(s) = \arg\max_a [\sum_{s'} P(s'|s,a)[R^a_{s,s'} + V(s')]]$$

Value iteration

- On prend V_0, en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande $\hat{\pi}^{V_0}$
- On peut alors calculer $V^{\hat{\pi}^{V_0}}$, la fonction de valeur de $\hat{\pi}^{V_0}$ (souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

Value iteration

- On prend V_0, en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande $\hat{\pi}^{V_0}$
- On peut alors calculer $V^{\hat{\pi}^{V_0}}$, la fonction de valeur de $\hat{\pi}^{V_0}$ (souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

Value iteration

- On prend V_0, en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande $\hat{\pi}^{V_0}$
- On peut alors calculer $V^{\hat{\pi}^{V_0}}$, la fonction de valeur de $\hat{\pi}^{V_0}$ (souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

Value iteration

- On prend V_0, en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande $\hat{\pi}^{V_0}$
- On peut alors calculer $V^{\hat{\pi}^{V_0}}$, la fonction de valeur de $\hat{\pi}^{V_0}$ (souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

Value iteration: algorithme

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
$\Delta \leftarrow 0$
For each $s \in \mathcal{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc $\hat{\pi}^{V^{\pi^*}} = \pi^*$. Autrement dit π^* est un "point fixe" par l'opération $f(\pi) = \hat{\pi}^{V^{\pi}}$.

Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc $\hat{\pi}^{V^{\pi^*}} = \pi^*$. Autrement dit π^* est un "point fixe" par l'opération $f(\pi) = \hat{\pi}^{V^{\pi}}$.

Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc $\hat{\pi}^{V^{\pi^*}} = \pi^*$. Autrement dit π^* est un "point fixe" par l'opération $f(\pi) = \hat{\pi}^{V^{\pi}}$.

Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc $\hat{\pi}^{V^{\pi^*}} = \pi^*$. Autrement dit π^* est un "point fixe" par l'opération $f(\pi) = \hat{\pi}^{V^{\pi}}$.

1 Introduction
- Formalisme
- Programmation dynamique

2 Apprentissage sans modèle
- Monte Carlo
- TD et Q-Learning

3 Montée à l'échelle
- Deep reinforcement learning (DQN)
- Cas du continu (DDPG)

4 Des problématiques
- Curriculum learning
- Sur de vrais robots

Value iteration: limitations

- Que faire si on ne connaît pas $P(s'|s,a)$?
  - On ne sait pas dans quel état on arrivera si on applique l'action a à partir de l'état s
  - Par exemple, si vous apprenez à conduire un vrai Kart, ou jouez à un jeu vidéo
- **Idée**: Sur un problème réel, on aurait plutôt envie de réaliser des expériences, c'est à dire des trajectoires d'états/actions/récompense: $(s_1, a_1, r_1, s_2, a_2, r_2, ..., s_n, a_n, r_n, s_{n+1})$ et d'utiliser ces données pour apprendre!

Value iteration: limitations

- Que faire si on ne connaît pas $P(s'|s,a)$?
  - On ne sait pas dans quel état on arrivera si on applique l'action a à partir de l'état s
  - Par exemple, si vous apprenez à conduire un vrai Kart, ou jouez à un jeu vidéo
- **Idée**: Sur un problème réel, on aurait plutôt envie de réaliser des expériences, c'est à dire des trajectoires d'états/actions/récompense: $(s_1, a_1, r_1, s_2, a_2, r_2, ..., s_n, a_n, r_n, s_{n+1})$ et d'utiliser ces données pour apprendre!

Fonction de valeur état/action

- Si on ne connaît pas le modèle, même en connaissant V^*, on ne peut pas trouver la politique optimale, car la politique gourmande cherche à maximiser $\sum_{s' \in S} P(s', r|s, a)[r + \gamma V(s')]$
- **Solution**: apprendre plutôt une fonction qui prend en paramètre s l'état actuel et a la prochaine action et vaut $Q^\pi(s, a) = R^a_{s,s'} + \gamma V(s')$ (toutes les récompenses que l'on obtiendra dans le futur à partir de cette action).
- On appelle cette fonction Q^π la fonction valeur état/action. Si on appelle Q^* la fonction de valeur état/action optimale, la politique optimale produit donc le a qui maximise $Q^*(s, a)$!

Fonction de valeur état/action

- Si on ne connaît pas le modèle, même en connaissant V^*, on ne peut pas trouver la politique optimale, car la politique gourmande cherche à maximiser $\sum_{s' \in S} P(s', r|s, a)[r + \gamma V(s')]$
- **Solution**: apprendre plutôt une fonction qui prend en paramètre s l'état actuel et a la prochaine action et vaut $Q^\pi(s, a) = R^a_{s,s'} + \gamma V(s')$ (toutes les récompenses que l'on obtiendra dans le futur à partir de cette action).
- On appelle cette fonction Q^π la fonction valeur état/action. Si on appelle Q^* la fonction de valeur état/action optimale, la politique optimale produit donc le a qui maximise $Q^*(s, a)$!

Fonction de valeur état/action

- Si on ne connaît pas le modèle, même en connaissant V^*, on ne peut pas trouver la politique optimale, car la politique gourmande cherche à maximiser $\sum_{s' \in S} P(s', r|s, a)[r + \gamma V(s')]$
- **Solution**: apprendre plutôt une fonction qui prend en paramètre s l'état actuel et a la prochaine action et vaut $Q^\pi(s, a) = R^a_{s,s'} + \gamma V(s')$ (toutes les récompenses que l'on obtiendra dans le futur à partir de cette action).
- On appelle cette fonction Q^π la fonction valeur état/action. Si on appelle Q^* la fonction de valeur état/action optimale, la politique optimale produit donc le a qui maximise $Q^*(s, a)$!

La fonction Q comme tableau à double entrées

Dans le monde discret, on peut simplement représenter Q à l'aide d'un tableau.

states	actions			
	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...

Politique gourmande et valeur état/action

Une politique gourmande pour une fonction Q donnée est la politique qui choisit l'action maximisant $Q(s, a)$ pour s l'état actuel.

La politique optimale π^* est également une politique gourmande sur Q^*. Dans la politique optimale, on a donc l'égalité suivante:

$$Q^*(s_t, a_t) = R^{a_t}_{s_t, s_{t+1}} + \gamma \max_a Q^*(s_{t+1}, a)$$

Politique gourmande et valeur état/action

Une politique gourmande pour une fonction Q donnée est la politique qui choisit l'action maximisant $Q(s, a)$ pour s l'état actuel.

La politique optimale π^* est également une politique gourmande sur Q^*. Dans la politique optimale, on a donc l'égalité suivante:

$$Q^*(s_t, a_t) = R^{a_t}_{s_t, s_{t+1}} + \gamma \max_a Q^*(s_{t+1}, a)$$

Exploration et exploitation

Comment faire les expériences?

- Essayer des actions au hasard ? Efficace pour tout explorer mais très long!
- Suivre la politique optimale ? On risque d'être très vite "bloqué" sur un optimum local!

Il faut donc un compromis entre les deux, le plus simple étant ϵ-greedy: suivre la politique optimale avec une probabilité $1 - \epsilon$, et choisir une action au hasard avec une probabilité ϵ.

Monte Carlo: principe

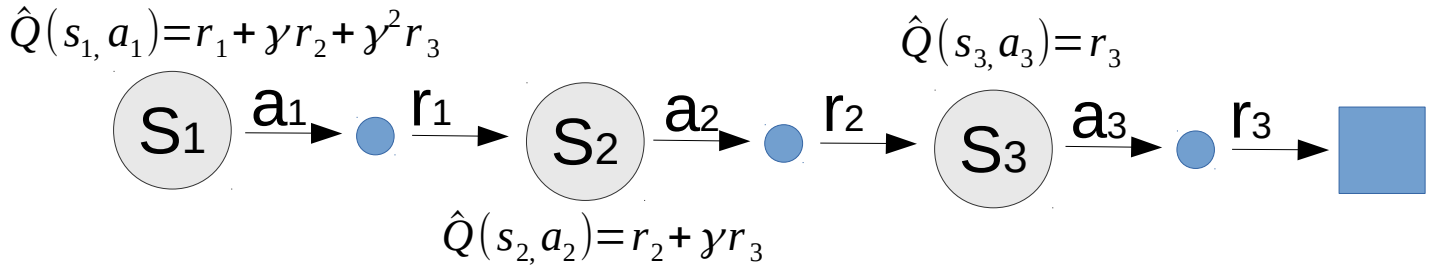
La politique optimale sélectionne l'action a qui maximise $Q^*(s, a)$, où Q^* est la fonction état/valeur optimale.

Sur la même idée que value iteration, on peut donc utiliser une politique ϵ-gourmande pour notre estimation actuelle de Q, et mettre à jour Q.

Monte Carlo: principe

Pour une politique donnée (par exemple ϵ-greedy avec Q) , on peut réaliser des expériences, jusqu'à atteindre un état terminal.

On peut alors, pour chaque couple (s, a) dans l'épisode, calculer la somme des récompenses obtenues à partir de cet état/action.



Notre estimation de $Q(s, a)$ sera alors une moyenne des expériences qu'on a effectuées.

Monte Carlo

Ce qui nous donne l'algorithme suivant:

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
    Generate an episode starting from S_0, A_0, following π
    For each pair s, a appearing in the episode:
        $G \leftarrow$ return following the first occurrence of s, a
        Append G to $Returns(s,a)$
        $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
    For each s in the episode:
        $\pi(s) \leftarrow \text{argmax}_a\, Q(s,a)$

Avantages et limitations de Monte Carlo

- Monte Carlo fournit des vrais échantillons de $Q(s, a)$, puisqu'on va jusqu'au bout de l'épisode pour l'obtenir
- Il faut justement attendre la fin de l'épisode pour l'obtenir
- Ne fonctionne pas dans des environnements non-épisodiques

Avantages et limitations de Monte Carlo

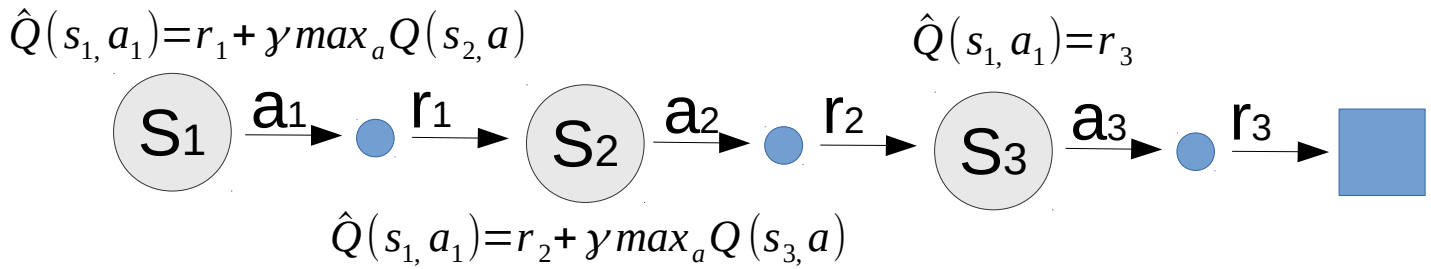
- Monte Carlo fournit des vrais échantillons de $Q(s, a)$, puisqu'on va jusqu'au bout de l'épisode pour l'obtenir
- Il faut justement attendre la fin de l'épisode pour l'obtenir
- Ne fonctionne pas dans des environnements non-épisodiques

Avantages et limitations de Monte Carlo

- Monte Carlo fournit des vrais échantillons de $Q(s, a)$, puisqu'on va jusqu'au bout de l'épisode pour l'obtenir
- Il faut justement attendre la fin de l'épisode pour l'obtenir
- Ne fonctionne pas dans des environnements non-épisodiques

Temporal Differences

Idée: Au lieu de prendre un "vrai" échantillon (jusqu'à bout de l'épisode), utiliser immédiatement le tuple $(s_t, a_t, r_t, s_{t+1}$ pour faire la mise à jour à partir de notre estimation actuelle.



Temporal Differences

Notre nouvelle estimation de $Q(s_t, a_t)$ devient alors:

$$r_t + \gamma \max_a Q(s_{t+1}, a)$$

On peut donc mettre à jour $Q(s_t, a_t)$ légèrement dans cette direction:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Où $\alpha \in [0, 1]$ est appelé le taux d'apprentissage (souvenez vous du calcul incrémental de la moyenne!)

Temporal Differences

Notre nouvelle estimation de $Q(s_t, a_t)$ devient alors:

$$r_t + \gamma \max_a Q(s_{t+1}, a)$$

On peut donc mettre à jour $Q(s_t, a_t)$ légèrement dans cette direction:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Où $\alpha \in [0, 1]$ est appelé le taux d'apprentissage (souvenez vous du calcul incrémental de la moyenne!)

Q-Learning

Ce qui nous donne l'algorithme du Q-Learning:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

Note: si on enleve le max_a, on obtient un autre algorithme nommé SARSA.

1 Introduction
- Formalisme
- Programmation dynamique

2 Apprentissage sans modèle
- Monte Carlo
- TD et Q-Learning

3 Montée à l'échelle
- Deep reinforcement learning (DQN)
- Cas du continu (DDPG)

4 Des problématiques
- Curriculum learning
- Sur de vrais robots

DQN

Playing Atari with Deep Reinforcement Learning (2013) `arXiv:1312.5602`
Human-level control through deep reinforcement learning (2015)
`https://www.nature.com/articles/nature14236`

DQN

Algorithme similaire au Q-Learning, mais la fonction $Q(s, a)$ est approximée par un réseau de neurone.

Deux modifications importantes apportées afin que l'algorithme fonctionne

Modification 1: Experience replay buffer
Si on utilise les expériences faites "en direct" pour entraîner Q, on utilisera des données trop corrélées temporellement. Aussi, il est dommage de "jeter" les expériences passées, alors que faire des expériences coûte plus cher que d'entraîner un réseau de neurones.

Les expériences (s_t, a_t, r_t, s_{t+1}) sont stockées dans une "mémoire" appellée le *replay buffer*.

Des batchs du replay buffer sont utilisées pour entraîner Q

Modification 2: Target networks

Lors de l'entraînement, $Q(s_t, a_t)$ est mis à jour en direction de $r_t + max_a Q(s_{t+1}, a)$.

Cette mise à jour a tendance a changer aussi $Q(s_{t+1}, a_{t+1})$ pour des états proches, rendant l'optimisation instable.

La solution trouvée est d'utiliser deux réseaux, un $\hat{Q}$ pour l'estimation de la cible et un réseau qui est effectivement mis à jour, et les poids sont recopiés toutes les C étapes.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Continu

DQN donne de très bons résultats malgré de très gros états (l'écran d'une console pixels par pixels), mais n'est praticable qu'avec une petite liste d'action discrète.

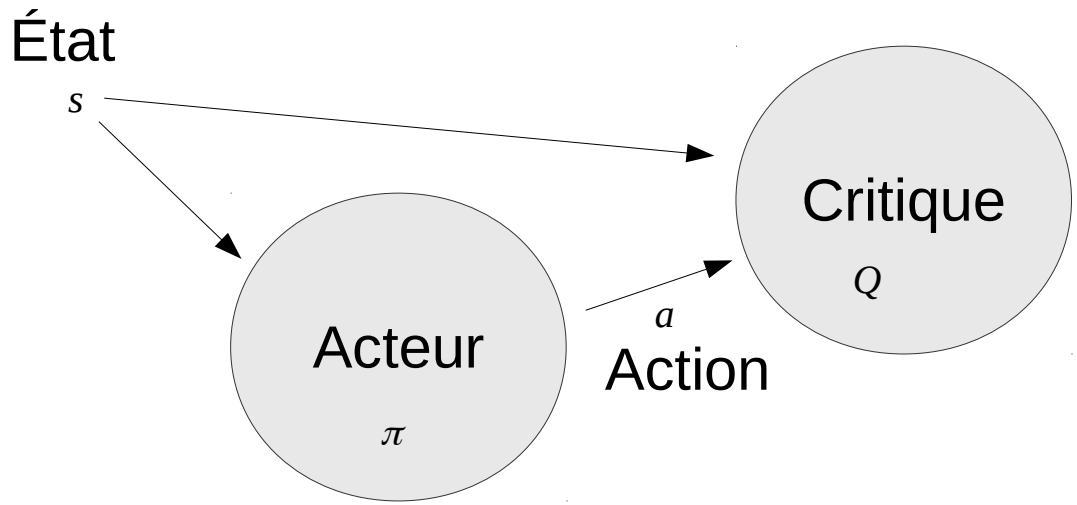
Ceci à cause du $r_j + max_a \hat{Q}(...)$, ici le max_a nécessite de "scanner" toutes les actions possibles.

Cela ne pose pas de problème pour choisir sur quel bouton appuyer sur un manette de jeu par exemple, mais ne fonctionne pas si il y a beaucoup d'actions, ou si les actions sont continues par exemple.

DDPG

Continuous control with deep reinforcement learning (2015)
`arXiv:1509.02971`

Acteur-critique



On utilise alors ici approches acteur-critique, où on sépare la politique π et la critique Q en deux parties (qui pourront par exemple être implémentées à l'aide de réseaux de neurones).

Policy gradient theorem

$$J(\theta) = \sum_{s \in S} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_{\theta}(a|s) Q^{\pi}(s,a)$$

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} Q^{\pi}(s,a) \pi_{\theta}(a|s)$$

$$\propto \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} Q^{\pi}(s,a) \nabla_{\theta} \pi_{\theta}(a|s)$$

Policy gradient theorem (stochastic)

$$\nabla_\theta J(\theta) \propto \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

$$= \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$$

$$= \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta ln(\pi_\theta(a|s))]$$

Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems 12 (1999).

Policy gradient theorem (deterministic)

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\mu}[\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)]$$

Silver, David, et al. "Deterministic policy gradient algorithms." International conference on machine learning. PMLR, 2014.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

TRPO

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta)$$
$$\text{s.t. } \bar{D}_{KL}(\theta||\theta_k) \leq \delta$$

Trust Region Policy Optimization (2015) `arXiv:1502.05477`

Idée: limiter les mises à jour des paramètres de la politique à l'aide d'une métrique sur les distributions de probabilité (la divergence de Kullback-Leibler)

A2C

Asynchronous Methods for Deep Reinforcement Learning (2016)
`arXiv:1602.01783`

Idée: Au lieu d'utiliser le replay buffer, lancer des agents en paralèlle

SAC

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot\,|\mathbf{s}_t)) \right].$$

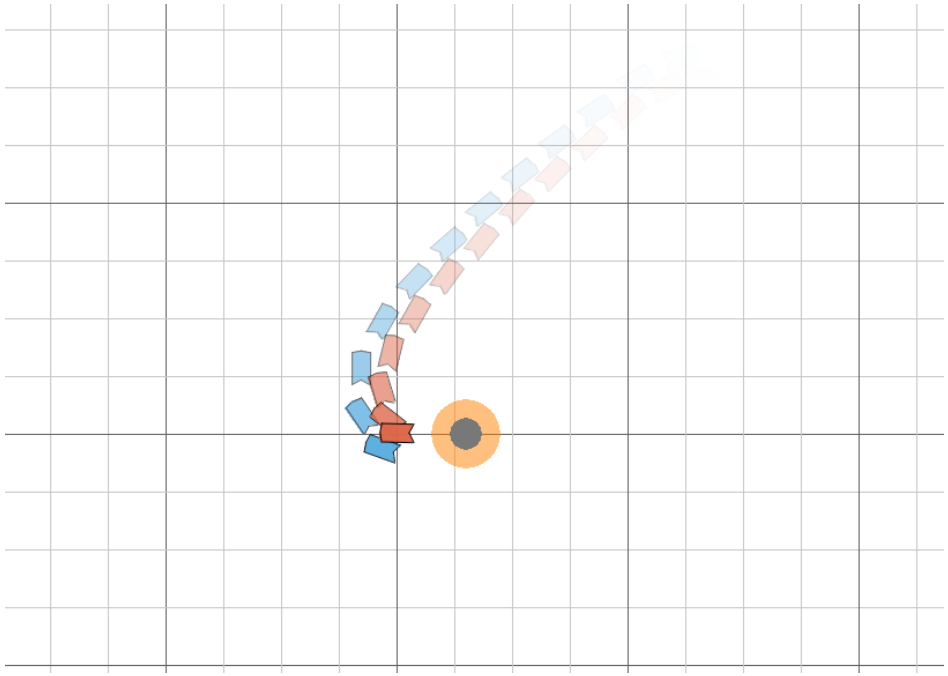
Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor (2018) `arXiv:1801.01290`

Idée: utiliser une politique stochastique et introduire l'entropie de la politique dans le score (de cette manière, à score égal, l'entropie favorise le choix d'une action aléatoire et encourage l'exploration).

1. Introduction
   - Formalisme
   - Programmation dynamique
2. Apprentissage sans modèle
   - Monte Carlo
   - TD et Q-Learning
3. Montée à l'échelle
   - Deep reinforcement learning (DQN)
   - Cas du continu (DDPG)
4. Des problématiques
   - Curriculum learning
   - Sur de vrais robots

Curriculum learning

Certaines tâches très dures sont impossibles à apprendre à partir de rien, mais on peut "guider" l'apprentissage



Exemple: faire le moins de pas possible pour arriver à une position donnée. Récompense de -1 par pas et 0 dans un état terminal lorsque l'épisode se termine.

Curriculum learning

- **Adapter l'état initial/final**: En fonction de la réussite de l'agent Exemple: faire partir l'agent de plus en plus loin de la cible tant qu'il y parvient, ou lui faire d'abord apprendre à atteindre une cible sans précision, puis augmenter la précision
- **Reward shaping**: Tenter de régler la fonction de récompense pour guider l'apprentissage Exemple: $-1 - \epsilon||\delta||$ où δ est l'erreur en position et orientation
- **Mode adversarial**: Lorsque le problème le permet, l'agent peut "jouer" contre lui-même, auquel cas le curriculum s'adapte naturellement (exemple: jeu de Go...)

Sur de vrais robots

Encore peu de RL déployée sur des vrais robots, les raisons étant:

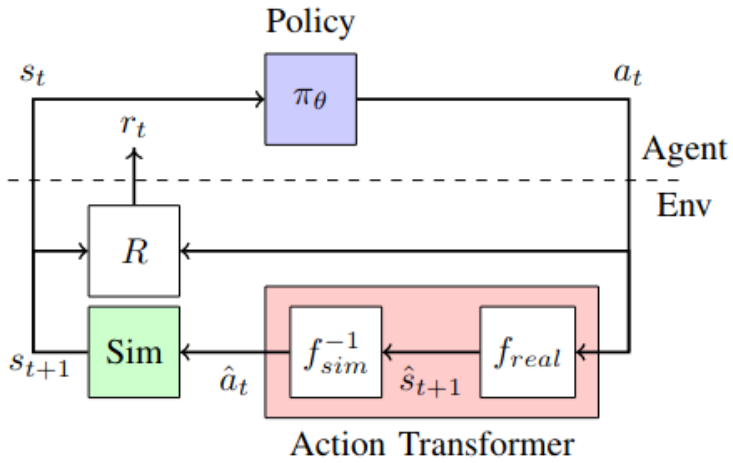
- **Difficultés de transferabilité (Sim2Real)**: ce qui est fait dans le simulateur ne fonctionne pas en général sur le vrai robot $\rightarrow$ besoin d'exploiter les données du monde réel
- **Algorithmes très gourmands en données**: la plupart des algorithmes sont impraticables sur des exemples réels

Des pistes

Solving Rubik's Cube with a Robot Hand (2019) `arXiv:1910.07113`

Idée: entraîner l'agent en simulation afin qu'il fonctionne malgré une diversité de paramètres d'environnement (curriculum + robustesse)

Des pistes



Reinforced Grounded Action Transformation for Sim-to-Real Transfer
`arXiv:2008.01279`

Idée: Apprendre à transformer les actions faites dans le simulateur pour qu'elles aient un effet plus proche de celui observé dans la réalité