

Apprentissage profond par renforcement (*Deep RL*)

Laëtitia Matignon

5A - Option Ouverture à la recherche SMA

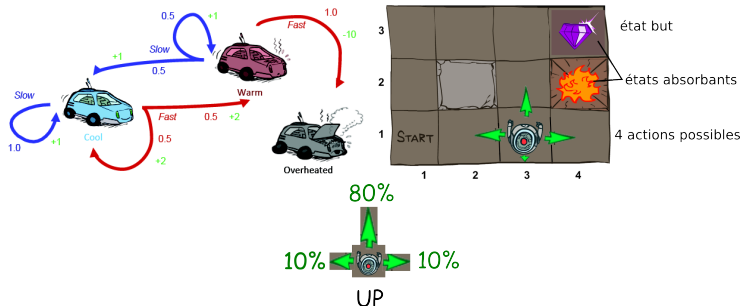
- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
- 4 Application : TP2

- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

Processus décisionnel de Markov (MDP)

Modèle MDP $\langle S, A, T, R \rangle$

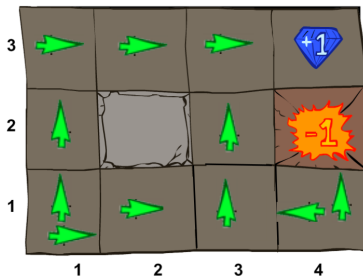
- ensemble fini d'états : S
- ensemble fini d'actions : A ou $A(s)$
- fonction de transition $T : S \times A \times S \rightarrow [0; 1] : T(s, a, s')$ est la probabilité d'atteindre s' lorsque l'on fait a dans s
- fonction de renforcement $R : S \times A \times S \rightarrow \mathbb{R} : \text{récompense}$



Processus décisionnel de Markov (MDP)

Solution au MDP

Politique notée $\pi : S \rightarrow A$ qui associe une (ou plusieurs) action(s)) à exécuter dans tout état.



exemple de politique

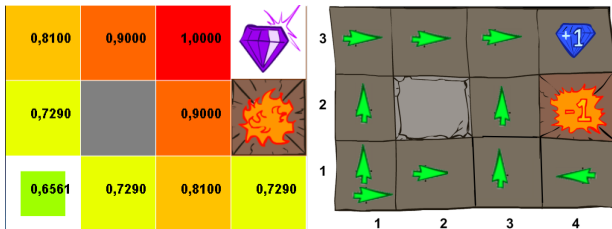
Processus décisionnel de Markov (MDP)

Objectif

Trouver la politique optimale notée π^* , qui donne pour tout état, la ou les action(s) permettant de maximiser les récompenses que l'on espère obtenir à travers la séquence d'états futurs.

π^* maximise la somme des récompenses futures, pondérée par $\gamma \in [0, 1[$:

$$G_t = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$



Fonction de valeur V associée à π^* dans un environnement déterministe

$$V^\pi(s) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid \pi, s_0 = s\right\}$$

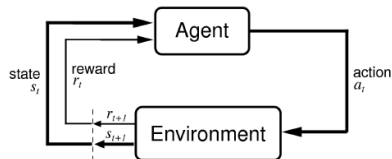
Apprentissage par renforcement

Hypothèses sur le modèle MDP

T et R inconnus de l'agent :

- L'agent ne connaît pas à l'avance les effets de ses actions
- L'agent ne connaît pas à l'avance les actions et/ou états récompensés

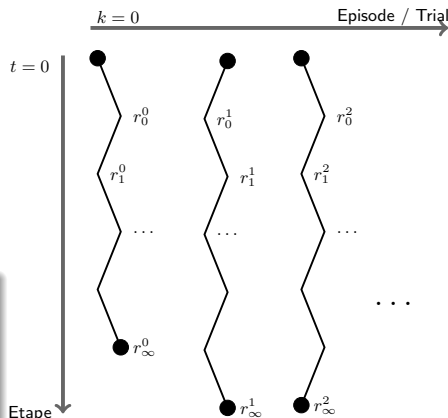
Apprentissage par renforcement



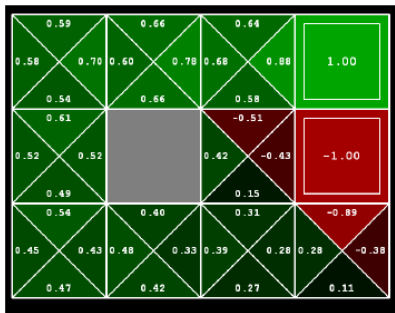
Agent plongé dans un MDP **sans**
connaissance de R et T .

Double temporalité :

- **Étape ou pas** t (une **action** est effectuée)
- **Épisode** k (ensemble d'étapes/actions de l'état initial à un état absorbant)



Apprentissage par renforcement

Fonction de valeur d'action Q

$$Q : S \times A \rightarrow \mathbb{R}$$

- $Q^\pi(s, a)$ évalue le retour espéré lorsque l'on effectue l'action a dans l'état s puis que l'on suit π : $Q^\pi(s, a) = E\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_t = s, a_t = a\}$
- Liens entre V et Q : $\forall s \in S \quad V(s) = \max_a Q(s, a)$

Amélioration de politique en-ligne

Politique gloutonne : action de plus grande valeur dans s : $\pi(s) = \arg \max_{a \in A} Q(s, a)$

Apprentissage par renforcement

Agent glouton avec erreur TD (différence temporelle)

A chaque étape :

- L'agent dans s suit sa politique, en **exécutant l'action** $a = \pi(s)$
- Il **évalue** le couple (s, a) pour mettre à jour $Q(s, a)$:
 - ▶ évaluation itérative de la moyenne des récompenses avec le coefficient d'apprentissage $\alpha \in [0, 1[$:

$$\underbrace{Q(s, a)}_{new_estim} \leftarrow \underbrace{Q(s, a)}_{old_estim} + \alpha \times (\underbrace{\delta - Q(s, a)}_{new_echantillon - old_estim = erreur})$$

- ▶ le nouvel échantillon doit être une évaluation de la somme pondérée des récompenses espérées depuis s : r est la récompense immédiate, $Q(s', a)$ est une évaluation de la somme pondérée des récompenses espérées depuis s'
- ▶ le nouvel échantillon est : $\delta = r + \gamma Q(s', a)$
- ▶ la mise à jour TD est

$$\underbrace{Q^\pi(s, a)}_{new_estim} \leftarrow (1 - \alpha) \underbrace{Q^\pi(s, a)}_{old_estim} + \alpha \times \underbrace{(r + \gamma \max_b Q^\pi(s', b))}_{new_echantillon}$$

- Après chaque mise à jour, il **améliore** sa politique π :

$$\pi(s) = \arg \max_{a \in A} Q^\pi(s, a)$$

Apprentissage par renforcement

Stratégies d'exploration : ϵ -greedy

Explorer avec une probabilité ϵ :

$$\pi_{greedy}(s) =$$

- action d'exploration (aléatoire parmi **toutes** les actions possibles) avec la probabilité ϵ
- action gloutonne ($a = \arg \max_a Q(s, a)$) avec la probabilité $1 - \epsilon$

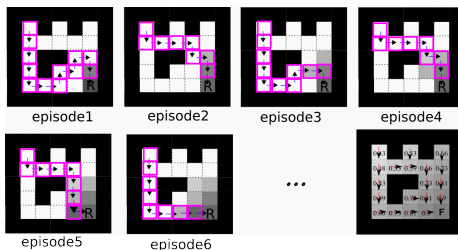
Apprentissage par renforcement

Algorithme du Q-learning

- $s \leftarrow$ état initial
- Répéter pour chaque étape dans l'épisode
 - ▶ Choisir a selon une stratégie d'exploration (**apprentissage actif**)
 - ▶ Exécuter a , observer r et s'
 - ▶ Mettre à jour la Q -valeur de (s, a) (**mise à jour TD**) :

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_b Q(s', b)]$$

- ▶ $s \leftarrow s'$



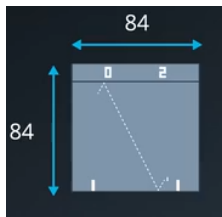
- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

Limites des approches tabulaires

- utilisation d'une table pour stocker les Q -valeurs
- chaque expérience (s, a, s', r) à l'instant t ne permet d'apprendre que pour le couple état-action (s, a) correspondant à cette expérience.

Espace mémoire

- Besoin de stocker $|S| \times |A|$ éléments.



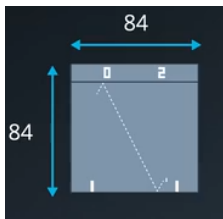
Par ex. si les états sont des images de taille 84×84 , en niveau de gris (256 valeurs) : $|S| = 256^{84 \times 84} = 256^{7056}$

Limites des approches tabulaires

- utilisation d'une table pour stocker les Q -valeurs
- chaque expérience (s, a, s', r) à l'instant t ne permet d'apprendre que pour le couple état-action (s, a) correspondant à cette expérience.

Espace mémoire

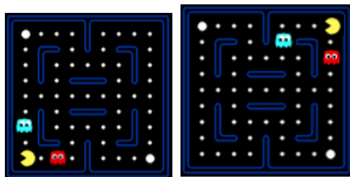
- Besoin de stocker $|S| \times |A|$ éléments.



Par ex. si les états sont des images de taille 84×84 , en niveau de gris (256 valeurs) : $|S| = 256^{84 \times 84} = 256^{7056}$

Lenteur de l'apprentissage

- il faut visiter tous les états un grand nombre de fois
- pas de **généralisation** de l'expérience apprise à des états jamais rencontrés



Si l'état à gauche est mauvais, l'agent ne saura rien sur l'état à droite s'il ne l'a jamais rencontré

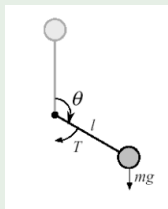
Limites des approches tabulaires : discrétisation

- hypothèse d'espaces d'états **discrets** et finis

Comment choisir une discrétisation de ISI adaptée ?

- si trop détaillée, la discrétisation sera computationnellement ingérable
- si trop faible, la discrétisation pourrait conduire à une politique non pertinente

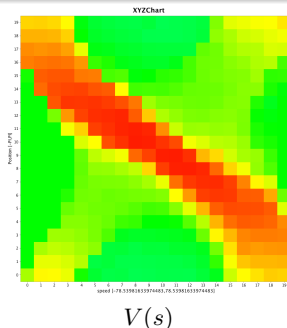
Exemple du pendule inversé



Espace d'états en 2D : $\mathbf{s} = (\theta, \dot{\theta})$

9 actions (*torque*)

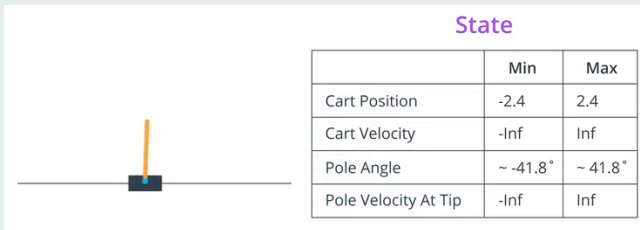
$$R(\mathbf{s}) = \cos(\theta)$$



Fonctionne avec une discrétisation uniforme de S en 30×30

Limites des approches tabulaires : discrétisation

Pendule inversé sur chariot



Espace d'états en 4D : $\mathbf{s} = (x, \dot{x}, \theta, \dot{\theta})$

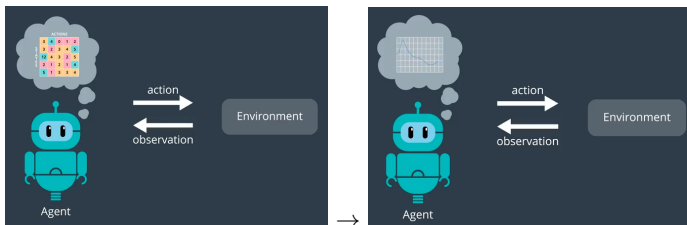
3 actions : $a = \{-F, 0, F\}$

$R(\mathbf{s}) = \cos(\theta) - d$ avec d pénalité si proche du bord

Fonctionne avec un Q-learning et une discrétisation **non-uniforme** de S :

- $x = \{-0.8, 0.8\}$; $\dot{x} = \{-0.5, 0.5\}$
- $\theta = \{-0.105, -0.017, 0, 0.017, 0.105\}$; $\dot{\theta} = \{-0.873, 0, 0.873\}$

Fonctions d'approximation

Approximation **linéaire** de la fonction Q (cf. IA 4A)

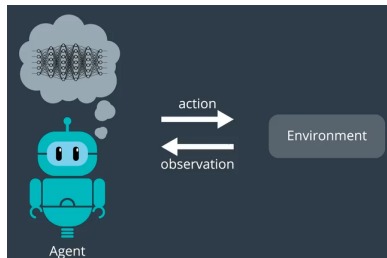
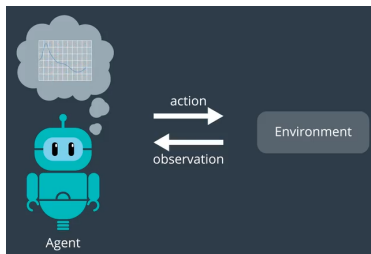
La fonction de Q -valeur est approximée par une fonction linéaire de n fonctions caractéristiques f_i :

$$Q_w(s, a) = \sum_i w_i f_i(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Les *features* $f_i : S \times A \rightarrow R$ sont choisies de façon experte
- Les paramètres/poids w_i sont appris (descente de gradient, apprentissage supervisé) pour avoir la meilleure approximation de la fonction Q

On stocke les poids $w \ll S$ et apprentissage conjoint des Q -valeurs (généralisation)

Fonctions d'approximation



Limites de l'approximation linéaire

- Difficile de choisir les bonnes caractéristiques expertes
- Limité par la linéarité : on ne peut représenter que des relations linéaires entre entrées et sorties

Solution : fonction d'approximation **non-linéaire** (réseau de neurones)

- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

Outline

- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

Approximation non-linéaire de la Q fonction

- Q_ω où ω sont les poids (appris) d'un réseau de neurones
- des *features* non-expertes seront « apprises » par le réseau
- en entrée, version brute des données



Action **Value** Approximation :
 $Q_\omega : S \times A \rightarrow \mathbb{R}$



Action **Vector** Approximation :
 $Q_\omega : S \rightarrow \mathbb{R}^A$

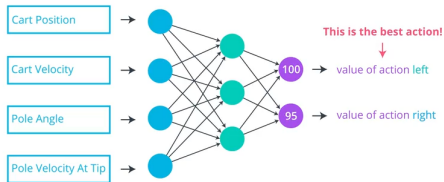
Autant de sortie que d'actions : cela permet de calculer plus facilement les **max** sur les actions.

Approximation non-linéaire de la Q fonction



State

	Min	Max
Cart Position	-2.4	2.4
Cart Velocity	-Inf	Inf
Pole Angle	$\sim -41.8^\circ$	$\sim 41.8^\circ$
Pole Velocity At Tip	-Inf	Inf



QL avec NN naïf

Forme d'apprentissage supervisée

On veut minimiser l'erreur sur les Q -valeurs :

- A chaque interaction/expérience (s, a, s', r) :
 - ▶ prédiction de la Q -valeur $Q(s, a)$ (*forward pass*) : $\hat{y} = q_{\omega}(s, a)$
 - ▶ **label/valeur cible de $Q(s, a)$ inconnue !!!** :
 → estimation par *amorçage* de la valeur cible $y = q(s, a)$.

Estimation par amorçage :

- $q(s, a)$ est la somme pondérée des récompenses espérées depuis (s, a) soit

$$r + \gamma \underbrace{r(s', a', s'') + \gamma^2 r'', \dots}_{\rho \text{ somme des recompenses esperees depuis } s'}$$

- On peut estimer ρ avec la Q -valeur max de s' qui est la somme pondérée des récompenses depuis s' .
- Ainsi la valeur cible est

$$y = r + \gamma \max_{b \in A} q_{\omega}(s', b)$$

QL avec NN naïf

Forme d'apprentissage supervisée

On veut minimiser l'erreur sur les Q -valeurs.

A chaque interaction/expérience (s, a, s', r) :

- prédiction pour la valeur de (s, a) (*forward pass*) : $\hat{y} = q_{\omega}(s, a)$
- **label/valeur cible de (s, a) inconnue** :
 → estimation par *amorçage* (autre *forward pass*) : $y = r + \gamma \max_{b \in A} q_{\omega}(s', b)$

- une itération de la descente de gradient (i.e. mise à jour des poids ω)

- ▶ fonction de perte MSE^a *loss* avec *erreur* $= y - \hat{y}$:

$$J_{\omega} = (y - \hat{y})^2$$

- ▶ calcul du gradient de la fonction MSE par rapport à ω (**suppose ici que y est indépendant de ω !**) :

$$\nabla_{\omega} J_{\omega} = -2(y - \hat{y}) \nabla_{\omega} \hat{y}$$

- ▶ mise à jour des poids ω selon descente de gradient, avec $\alpha \in [0, 1]$ coefficient d'apprentissage :

$$\omega = \omega - \frac{\alpha}{2} \nabla_{\omega} J_{\omega}$$

$$= \omega + \alpha(r + \gamma \max_{b \in A} q_{\omega}(s', b) - q_{\omega}(s, a)) \nabla_{\omega} q_{\omega}(s, a)$$

a. moyenne des erreurs au carré

QL avec NN *naïf* : instabilités

Corrélations

- corrélations entre expériences (s, a, s', r) envoyées successivement au NN
- corrélations entre valeur cible et paramètres

Outline

- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - **Techniques d'entraînement**
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

Experience replay : objectifs

Problème 1 : Oubli des expériences anciennes

catastrophic forgetting :



By learning how to play on water level, our agent will forget how to behave on the first level

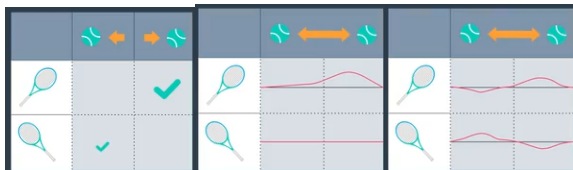
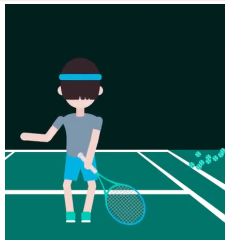
Experience replay : objectifs

Problème 2 : Corrélations entre expériences

Corrélation entre expériences (s, a, s', r) d'une même trajectoire envoyées successivement au NN :

- **difficulté** pour l'apprentissage supervisé, car les données en app. sup. doivent être une réalisation de variables aléatoires *iid*^a
- les expériences sont corrélées car une action d'une expérience affecte les états des expériences suivantes.

a. variables indépendantes et identiquement distribuées

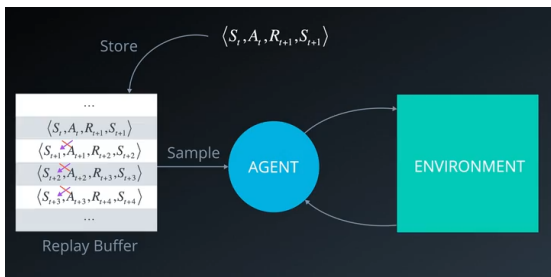


Experience Replay : principe

Replay Buffer

2 processus entrelacés :

- mémorise les expériences rencontrées en suivant la politique courante (sans apprentissage)
- tire aléatoirement un mini-batch d'expériences dans le buffer pour la mise à jour des poids du NN (apprentissage, politique courante mise à jour)



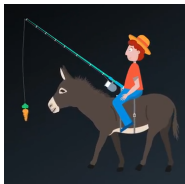
Experience Replay : intérêts

- ➊ réduction de l'oubli, car réutilise des expériences rares ou anciennes.
- ➋ accélère l'apprentissage car mini-batch et chaque expérience est potentiellement utilisée dans plusieurs mises à jour des poids.
- ➌ réduit la corrélation entre expériences successives.

Neural Fitted Q Iteration, Martin Riedmiller, Proceedings of the 16th European conference on Machine Learning, 2005



Target Network : objectifs



Corrélation entre valeur cible y et paramètres ω

- le calcul du gradient de la loss MSE suppose que la cible y est indépendante de ω , ce qui n'est pas le cas ici car on utilise l'erreur TD :

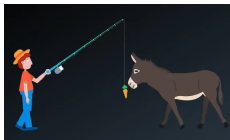
$$y = r + \gamma \max_{b \in A} q_{\omega}(s', b)$$

- la cible y est calculée avec le réseau de neurones q_{ω} , que l'on met à jour à chaque étape en mettant à jour les poids ω :

$$\omega = \omega + \alpha (r + \gamma \max_{b \in A} q_{\omega}(s', b) - q_{\omega}(s, a)) \nabla_{\omega} q_{\omega}(s, a)$$

la valeur cible est mouvante !

Target Network : principe



Figer la cible

Découpler les paramètres utilisés pour calculer la cible ω^- et ceux mis à jour ω :

$$\omega = \omega + \alpha (r + \gamma \max_{b \in A} q_{\omega^-}(s', b) - q_{\omega}(s, a)) \nabla_{\omega} q_{\omega}(s, a)$$

Target network :

- utilisation d'un second NN cible, de poids ω^- , pour calculer la valeur cible
- mise à jour des poids ω^- avec les poids appris par descente de gradient ω selon une fréquence plus faible

Intérêts

- stabilisation de la cible
- réduction des corrélations entre cible et paramètres changeants

Outline

- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - **Deep QLearning algorithm**
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

Deep QLearning algorithm (2013)

- Initialize replay memory D with capacity N
- Initialize action-value function \hat{q} with random weights \mathbf{w}
- Initialize target action-value weights $\mathbf{w}^- \leftarrow \mathbf{w}$
- **for** the episode $e \leftarrow 1$ to M :
 - Initial input frame x_1
 - Prepare initial state: $S \leftarrow \phi(\langle x_1 \rangle)$
 - **for** time step $t \leftarrow 1$ to T :
 - Choose action A from state S using policy $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}(S, A, \mathbf{w}))$
 - Take action A , observe reward R , and next input frame x_{t+1}
 - **SAMPLE** Prepare next state: $S' \leftarrow \phi(\langle x_{t-2}, x_{t-1}, x_t, x_{t+1} \rangle)$
 - Store experience tuple (S, A, R, S') in replay memory D
 - $S \leftarrow S'$
 - **LEARN** Obtain random minibatch of tuples (s_j, a_j, r_j, s_{j+1}) from D
 - Set target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, \mathbf{w}^-)$
 - Update: $\Delta \mathbf{w} = \alpha (y_j - \hat{q}(s_j, a_j, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(s_j, a_j, \mathbf{w})$
 - Every C steps, reset: $\mathbf{w}^- \leftarrow \mathbf{w}$

- buffer de taille fini

Outline

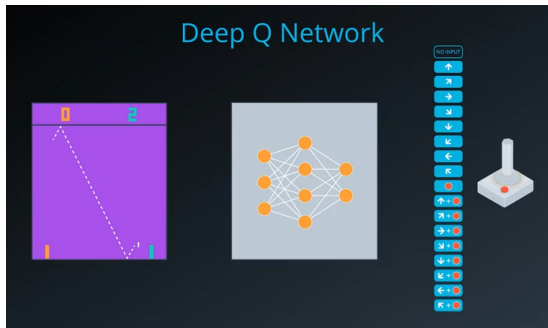
- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

DQN pour les jeux Atari



video

Deep Q Network

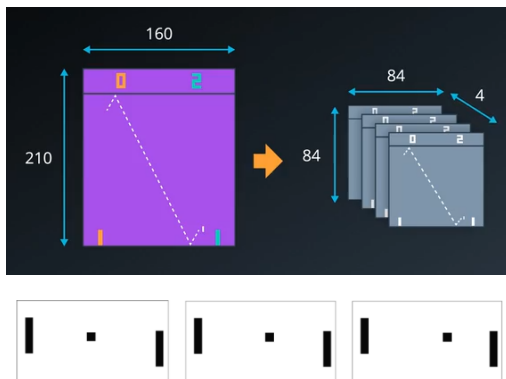


- entrées : image
- sorties : Q -valeurs de chaque action discrète (4 à 18)
- récompense : changement de score à chaque étape
- experience replay : stockages des 100 000 dernières interactions
- target network

Playing Atari with Deep Reinforcement Learning, V. Mnih et. al, 2013

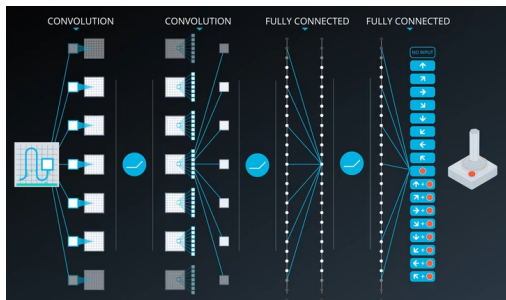
Human-level control through deep reinforcement learning, V. Mnih et. al, Nature, 2015

DQN pour les jeux Atari

Traitement des images en entrée $S = 84 \times 84 \times 4$

- réduction des images en niveau de gris de taille 84×84 pour réduire le temps de calcul de l'apprentissage sans perte d'information
- envoi au NN une agrégation (*stack*) des 4 dernières images, pour réduire le problème de limitation temporelle, et donner à l'agent une information sur les mouvements : direction, vitesse, etc.

DQN pour les jeux Atari



Architecture du NN

- 2 CNNs avec ReLU pour extraire les relations spatiales et temporelles entre les 4 images
- 1 couche fully-connected avec ReLU
- 1 couche fully-connected de sortie produit le vecteur des valeurs d'action
- même architecture pour tous les jeux Atari

- 1 Rappels : Apprentissage par renforcement
- 2 Limites des approches tabulaires
- 3 Deep Q Learning
 - QLearning avec NN naïf
 - Techniques d'entrainement
 - Deep QLearning algorithm
 - Exemple : Deep Q Network pour les jeux Atari
- 4 Application : TP2

TP2

- implémenter en PyTorch l'algorithme du Deep QLearning
- le tester sur un environnement d'*OpenAI Gym* : LunarLander
- lire un article au choix présentant une extension du Deep QLearning
- rédiger une analyse de l'article
- bonus : implémenter cette extension

