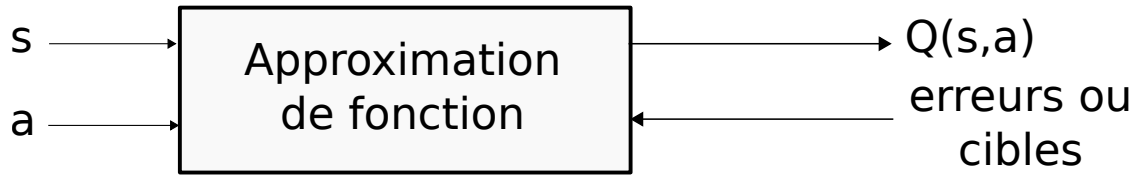
CM3: Apprentissage profond par renforcement: Deep QNetwork (DQN)

Laëtitia Matignon

INSA 5ATC - IA pour la robotique

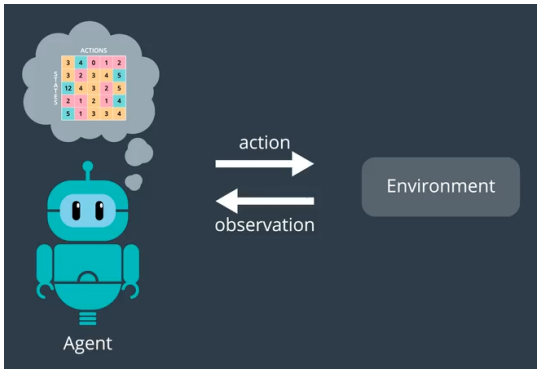
3 façons de représenter la Q fonction :



$Q(s, a) =$

- table
- Approximation linéaire et vecteur de *features* expertes : Approximate Q-learning
- Approximation non-linéaire : réseau de neurones

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.

Problème 1 : espace mémoire

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

Problème 2 : Lenteur de l'apprentissage

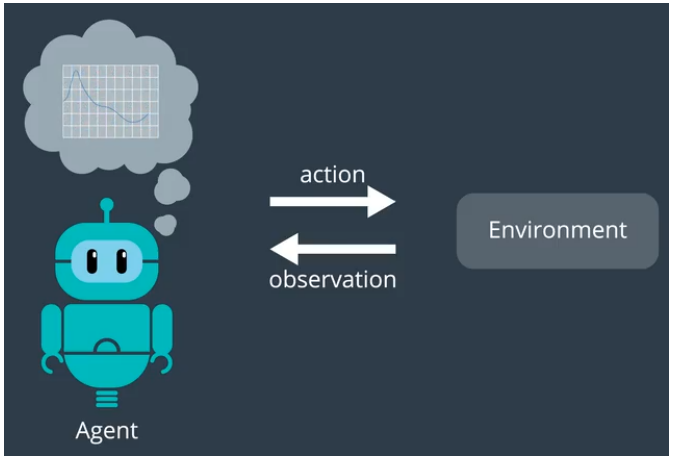
- il faut visiter tous les états un grand nombre de fois
- pas de généralisation de l'expérience apprise à des états jamais rencontrés

Problème 3 : Discrétisation

Hypothèse d'espaces d'états **discrets** et finis

- Comment choisir une discrétisation adaptée ?

Fonctions d'approximation



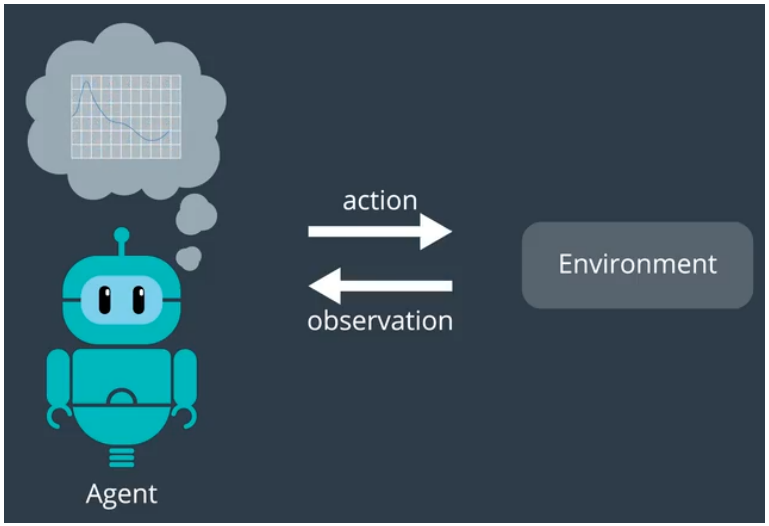
Approximation **linéaire** de la fonction Q

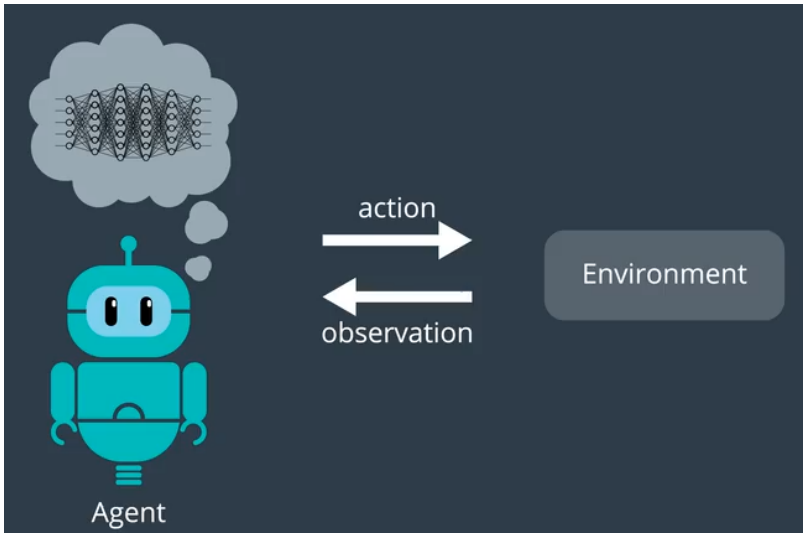
$$Q_w(s,a) = \sum_i w_i f_i(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + ... + w_n f_n(s,a)$$

- n *fonctions caractéristiques* $f_i : S \times A \to R$ choisies de façon experte
- n paramètres/poids w_i appris (descente de gradient, apprentissage supervisé)

On stocke n poids ($n << S \times A$) et apprentissage conjoint des Q-valeurs (généralisation)

Fonctions d'approximation





Limites de l'approximation linéaire

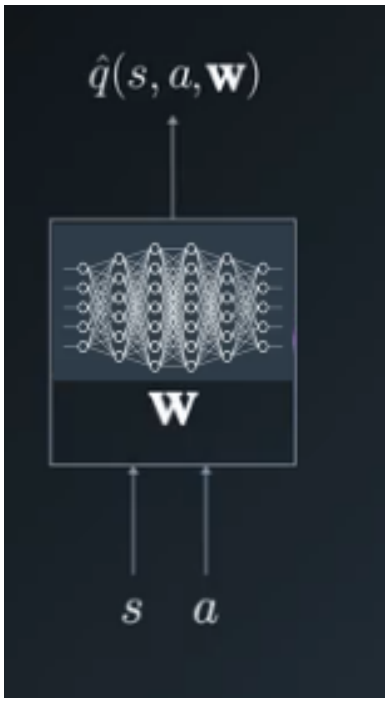
- Difficile de choisir les bonnes caractéristiques expertes
- Limité par la linéarité : on ne peut représenter que des relations linéaires entre entrées et sorties

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

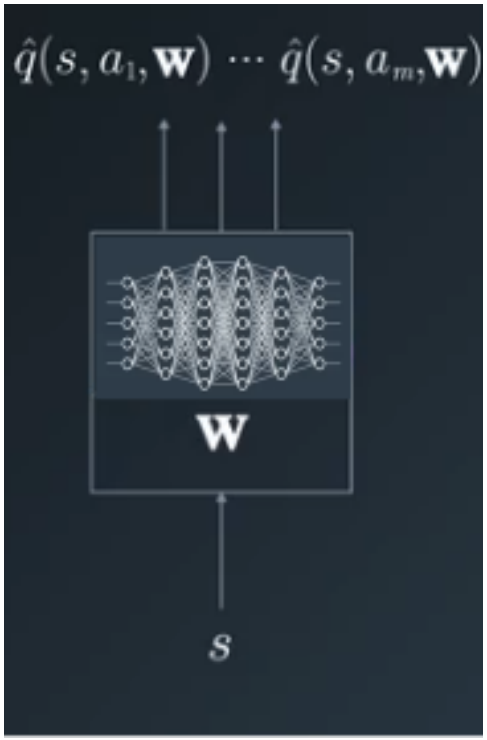
Outline

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 perceptions



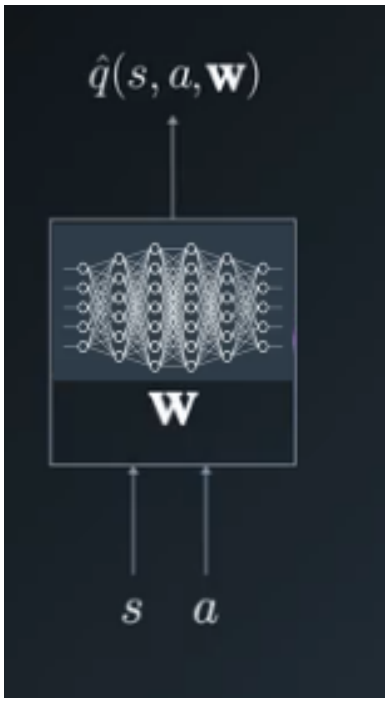
Action **Value** Approximation :
$Q_\omega : S \times A \to \mathbb{R}$



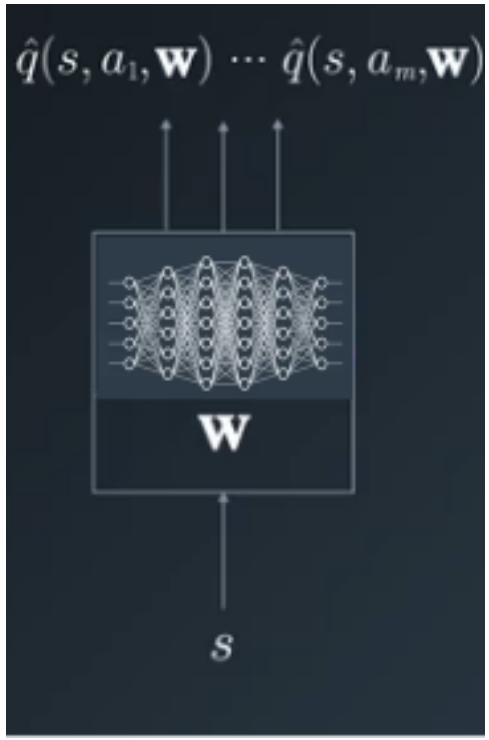
Action **Vector** Approximation :
$Q_\omega : S \to \mathbb{R}^A$

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 perceptions



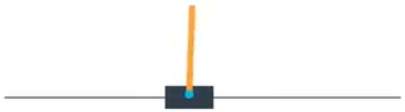
Action **Value** Approximation :
$Q_\omega : S \times A \to \mathbb{R}$



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

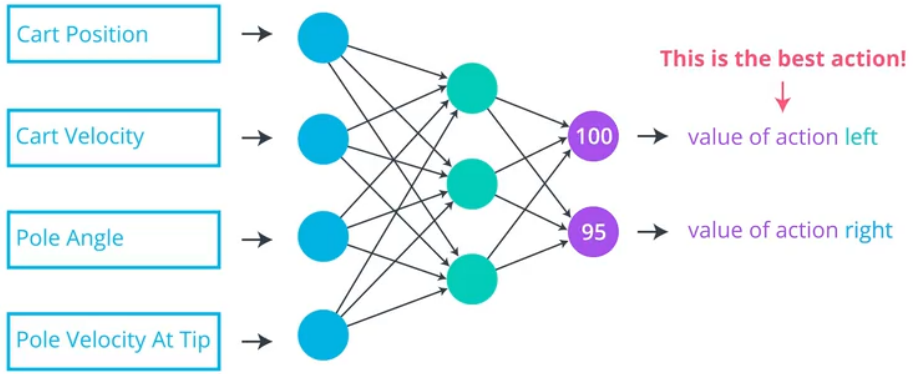
Autant de sortie que d'actions : cela permet de calculer plus facilement les `max` sur les actions.

Approximation non-linéaire de la Q fonction : exemple



State

	Min	Max
Cart Position	-2.4	2.4
Cart Velocity	-Inf	Inf
Pole Angle	~ -41.8°	~ 41.8°
Pole Velocity At Tip	-Inf	Inf



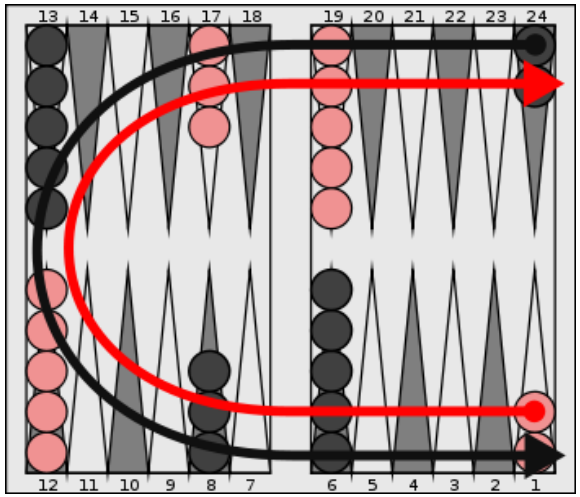
QL avec NN *naïf*

On a une fonction Q_ω paramétrée par les poids du NN ω.
On va **minimiser l'erreur** sur les Q-valeurs (forme d'apprentissage supervisé) :
A chaque interaction (s, a, s', r) :

- prédiction de la Q-valeur $Q(s, a)$ (*forward pass*) : $\hat{y} = Q_\omega(s, a)$
- valeur cible de $Q(s, a)$ **inconnue** ! ! ! :
  $\rightarrow$ estimation par *amorçage* : $y = r + \gamma \max_{b \in A} \underbrace{Q_\omega(s', b)}_{\text{2nd } \textit{forward pass}}$
- une itération de la descente de gradient (i.e. mise à jour des poids ω)
  - fonction de perte *MSE*[1] : $J_\omega = (y - \hat{y})^2$
  - 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 avec $\alpha \in [0, 1]$ coefficient d'apprentissage :
    $$\begin{aligned} \omega &= \omega - \frac{\alpha}{2} \nabla_\omega J_\omega \\ &= \omega + \alpha (r + \gamma \max_{b \in A} q_\omega(s', b) - q_\omega(s, a)) \nabla_\omega q_\omega(s, a) \end{aligned}$$

1. moyenne des erreurs au carré

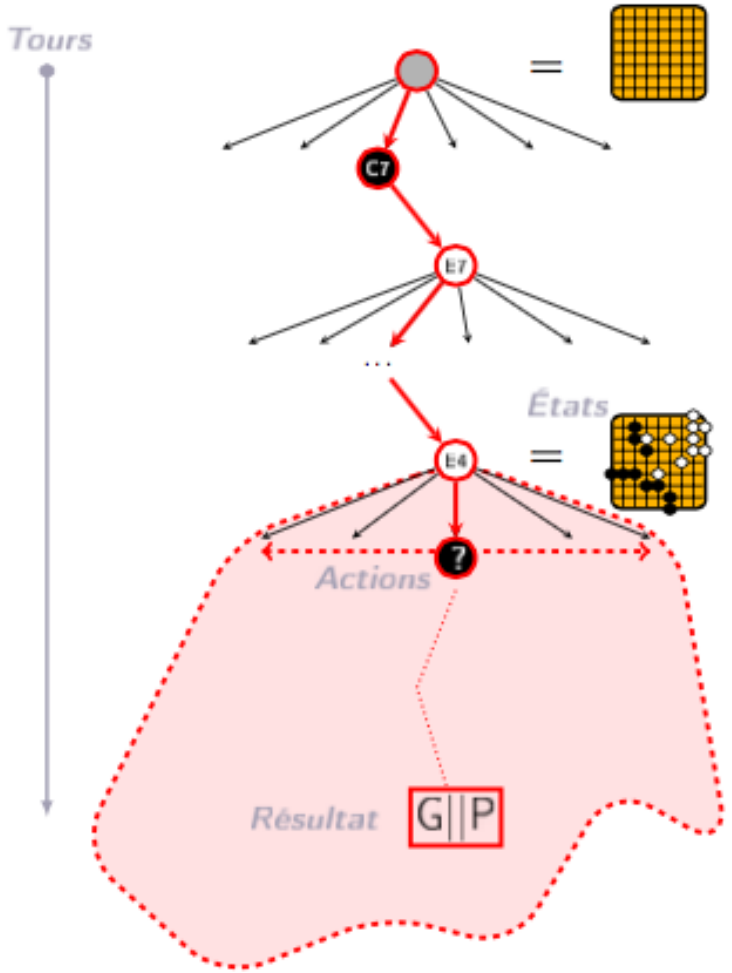
Application : TD-Gammon [Tesauro, 1992]



Backgammon

- jeu à 2 joueurs sur plateau, avec dés
- 2 ensembles de 15 dames
- joueur gagne lorsqu'il sort toutes ses dames du plateau (pour cela, il faut amener toutes ses dames dans le dernier quadran)
- à chaque jet de dés, le joueur choisit parmi un grand nombre d'options pour déplacer ses dames (20 en moyenne)

Application : TD-Gammon [Tesauro, 1992]



Backgammon : Difficultés

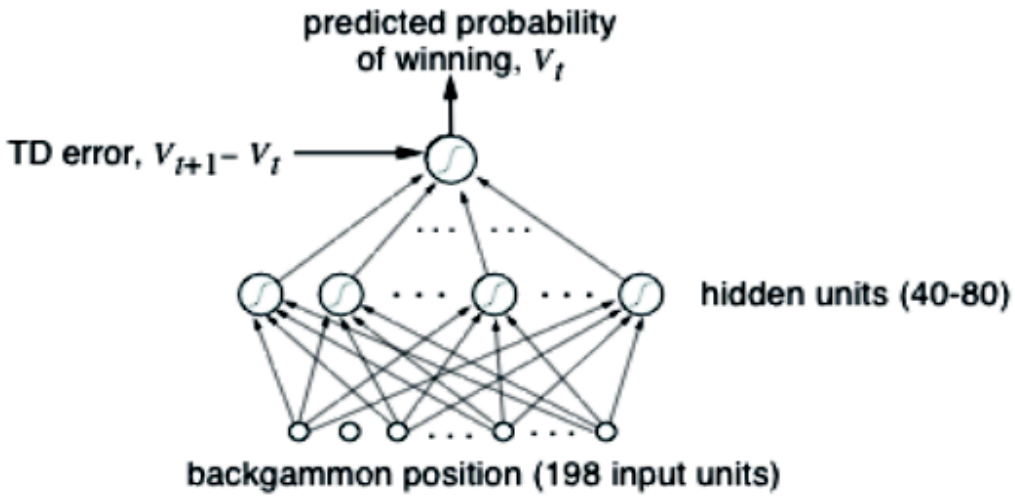
- Grand nombre de positions sur le plateau (30 pièces et 26 cases)
  $\rightarrow$ exclut les méthodes tabulaires
- Beaucoup d'options pour chaque jet de dés
- Pour considérer les mouvements futurs de l'adversaire, il faut considérer les jets de dés possibles.
  $\rightarrow$ Facteur de branchement d'environ 400

Application : TD-Gammon [Tesauro, 1992]

AR appliqué au Backgammon

- TD(λ) avec retro propagation dans un réseau de neurones
- une couche d'entrée avec 198 entrées qui codent des informations sur la position des dames sur le plateau
- une couche cachée de 40 neurones
- une couche de sortie avec 3 valeurs pour estimer la probabilité de gagner de façon ordinaire, gammon ou backgammon.

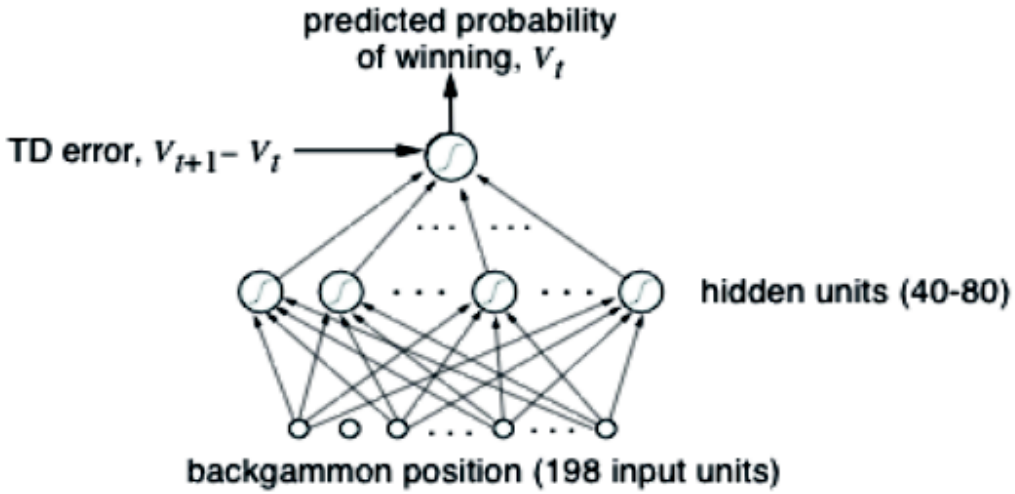
$V_t(s)$ utilisé pour estimer la probabilité de gagner depuis s.



TD-Gammon [Tesauro, 1992]

AR appliqué au Backgammon

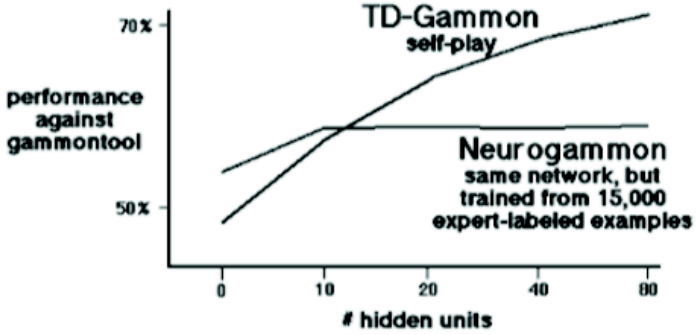
- récompenses immédiates : +100 si gagne, -100 si perd, 0 sinon
- après chaque jet de dés, le réseau évalue toutes les positions possibles depuis la position courante. Celle de plus forte valeur est choisie. La récompense reçue est alors rétro-propagée dans le réseau.
- Apprend en jouant contre soi-même 300000 parties
  $\rightarrow$ joue aussi bien que le meilleur joueur du monde



TD-Gammon [Tesauro, 1992]

Plusieurs versions

- Neurogammon : apprentissage supervisé sur corpus expert + informations caractéristiques en entrées
- TD Gammon 0.0 : apprentissage contre soi-même en mode TD
- TD Gammon 1.0 : apprentissage contre soi-même en mode TD + informations caractéristiques en entrées
- TD Gammon 2 : *2-ply search* : prise en compte des mouvements possibles de l'adversaire
- TD Gammon 3 : *3-ply search*



Tesauro, 1992

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	other programs	tied for best
TD-Gam 1.0	80	300,000	Robertie, Magriel, . . .	−13 points / 51 games
TD-Gam 2.0	40	800,000	various Grandmasters	−7 points / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	−1 point / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 points / 20 games

QL avec NN *naïf* : instabilités

Corrélations

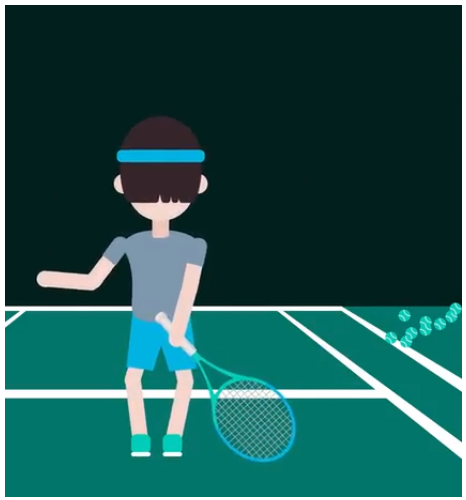
- corrélations entre interactions (s, a, s', r) envoyées successivement au NN
  $\rightarrow$ Experience replay
- corrélations entre valeur cible y et paramètres ω
  $\rightarrow$ Réseau cible

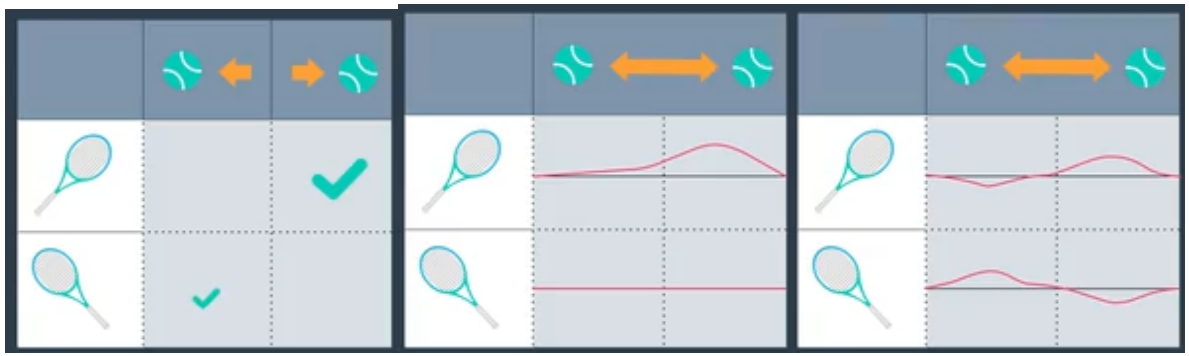
Outline

Problème : Corrélations entre interactions

- interactions (s, a, s', r) envoyées dans ordre chronologique au NN : une action d'une interaction affecte les états des interactions suivantes.
- **difficulté** pour l'apprentissage supervisé : données ne sont pas une réalisation de variables aléatoires *iid* [a]

a. variables indépendantes et identiquement distribuées





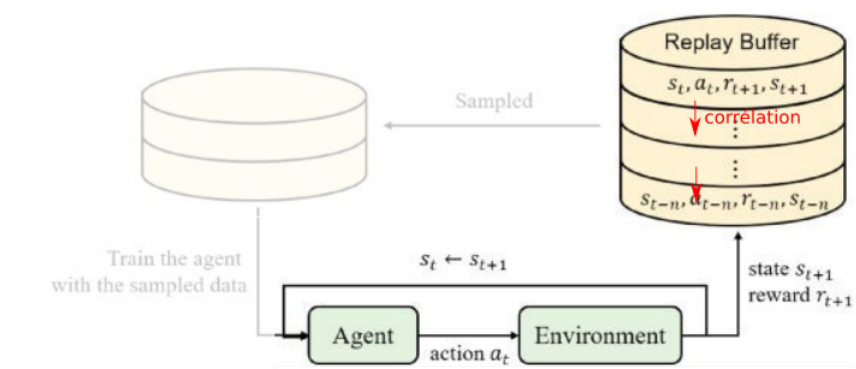
Apprentissage "global" et corrélations $\rightarrow$ mise à jour d'un (s, a) peut entraîner des modifications incontrôlées d'autres (s, a)
Risque de lenteur d'apprentissage ou de divergence.

Experience Replay : principe

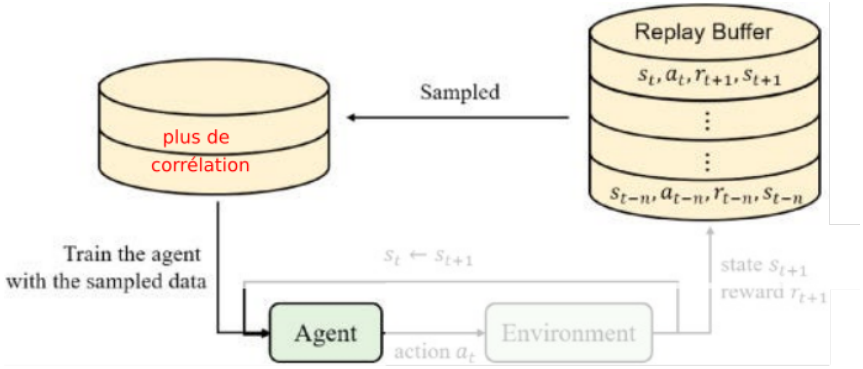
Replay Buffer

2 processus entrelacés :

- Échantillonnage : mémorise dans un buffer les interactions rencontrées en suivant la politique courante (sans apprentissage)
- Apprentissage : tire **aléatoirement** un mini-batch d'interactions dans le buffer pour mettre à jour les poids ω



échantillonage (sampling)



apprentissage

Deep Q-Learning avec *Experience Replay* : algorithme

Initialiser le replay buffer R de capacite N;
Initialiser la fonction Q avec des poids aleatoire θ;
for *episode* $e = 1..M$ **do**
  Recuperer l'etat initial s_1;
  for *etapes* $t = 1..T$ **do**
    Choisir a_t selon une strategie ϵ-greedy;
    Executer a_t et recuperer s_{t+1} et r_t;
    Ajouter la transition (s_t, a_t, r_t, s_{t+1}) dans R;
    $s_t \leftarrow s_{t+1}$;
    if $t\%LEARNSTEP == 0$ **then**
      Recuperer un minibatch de taille d dans R: (s_j, a_j, r_j, s_{j+1});
      Calculer la prediction $\hat{y}_j = Q_\theta(s_j, a_j)$;
      Calculer la cible

$$y_j = \begin{cases} r_j & \text{si e se termine} \\ r_j + \gamma max_b Q_\theta(s_{t+1}, b) & \text{sinon.} \end{cases} ;$$

      Mettre a jour θ: effectuer une etape de descente de gradient pour minimiser $(y_j - \hat{y}_j)^2$ (MSE loss);
    end
  end
end

Algorithm 1: Algorithme DQN

interaction avec envi
sampling_step

phase de d'échantillonage (sampling)

phase d'apprentissage (train_step)

Experience Replay : intérêts

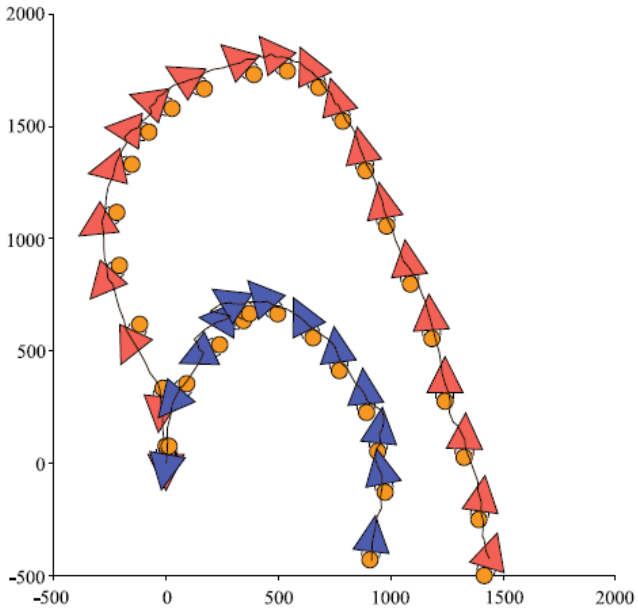
1. réduit la corrélation entre interactions successives.
2. accélère l'apprentissage : une interaction est potentiellement utilisée dans plusieurs mises à jour des poids.
3. réduction de l'oubli (*catastrophic forgetting*) : réutilise des interactions rares ou anciennes.

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

Applications de l'Experience Replay : Robocup soccer

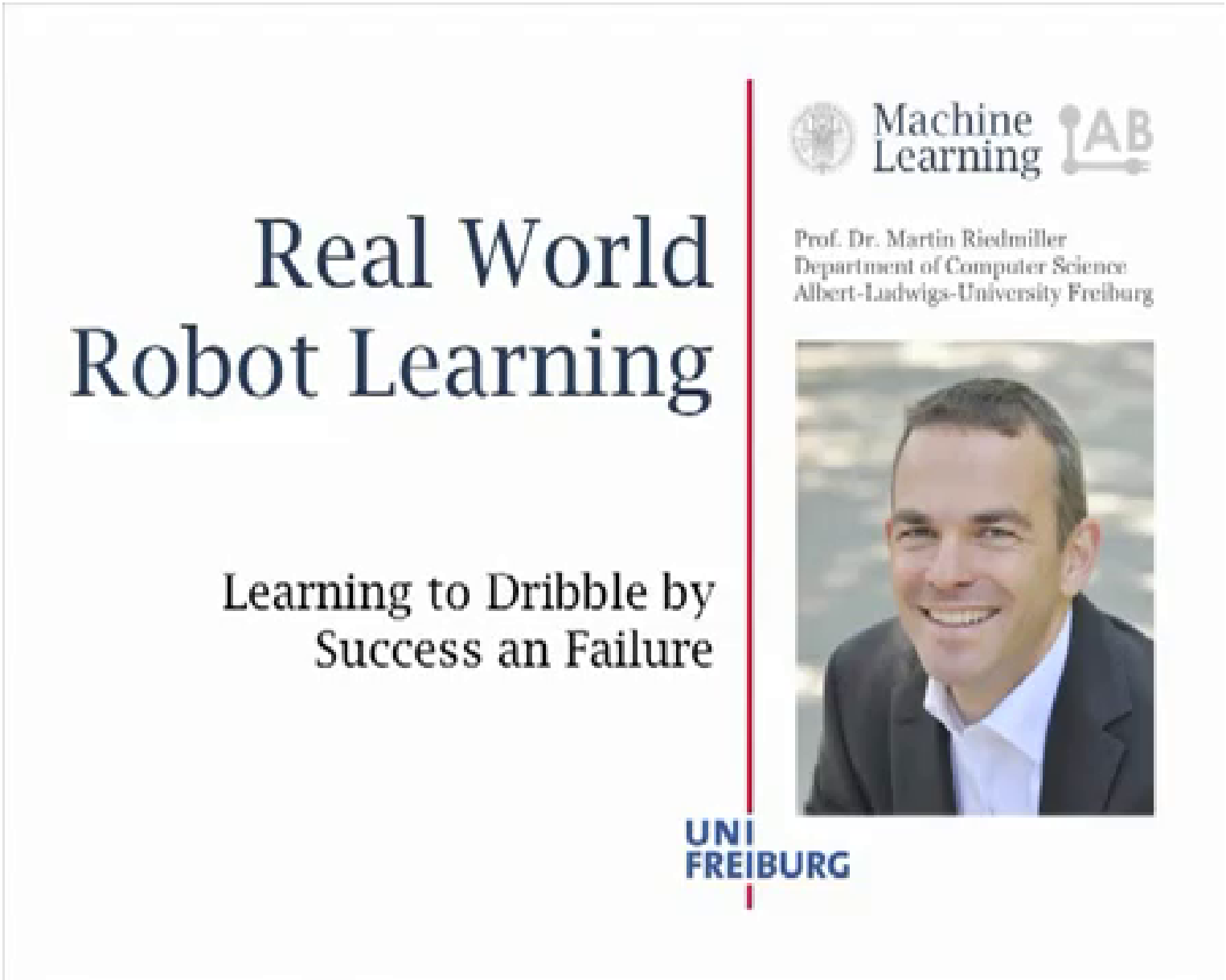
Neural Fitted Q-learning [Stone et al., Riedmiller et al., 2005]

- Robot footballeur qui apprend différentes techniques pour jouer au foot
- Deep Q-learning avec experience replay : MLP avec 9 entrées (6 pour l'état, 3 pour les actions), 2 couches cachées de 20 neurones chacune, un neurone de sortie.
- Phase d'apprentissage d'environ 1h30.



Technique de *dribble* : *hand-coded* (rouge) vs. neural FQ (bleu)

Robocup soccer



Robocup soccer

	'00	'01	'02	'03	'04	'05	'06	'07	'08
Simulation League									
NeuroKick	●	●	●	●	●	●	●	●	●
NeuroIntercept	●	●	●	●		○	○		
NeuroGo2Pos	●	●	●	●	●				
NeuroADB								●	●
NeuroAttack	○	●	●	●	●		●	●	●
NeuroPenalty				●	●	●	●	●	●
Rank	2	2	3	3	2	1	2	1	1
MidSize League									
NeuroMotorSpeed								○	○
NeuroGo2Pos							○	○	○
LmapIntercept							●	●	●
NeuroDribble								●	●
Rank							1	1	3

Techniques apprises dans l'équipe Brainstormers Tribots

Outline

Problème

Corrélation entre valeur cible y et paramètres ω

- La valeur cible estimée par amorçage dépend de ω :

$$y = r + \gamma \max_{b \in A} \underbrace{Q_\omega(s', b)}_{\text{2nd } \textit{forward pass}}$$

- Mais le calcul du gradient de l'erreur par rapport à ω suppose que y est indépendant de ω ! :

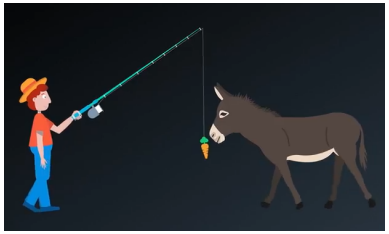
$$\nabla_\omega J_\omega = -2(y - \hat{y}) \nabla_\omega \hat{y}$$

- On réalise une mise à jour des poids ω avec une valeur cible y non-stationnaire :

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

$\rightarrow$ instabilité dans l'apprentissage

Réseau cible : principe



Figer la cible

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

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

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

Intérêts

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

Deep Q-Network (DQN) : algorithme (2013)

Deep Q-Learning avec expérience replay et réseau cible

Initialiser le replay buffer R de capacite N;
Initialiser la fonction Q avec des poids aleatoire θ;
Initialiser la fonction cible $\hat{Q}$ avec des poids $\theta^- = \theta$;
for *episode* $e = 1..M$ **do**
  Recuperer l'etat initial s_1;
  for *etapes* $t = 1..T$ **do**
    Choisir a_t selon une strategie ϵ-greedy;
    Executer a_t et recuperer s_{t+1} et r_t;
    Ajouter la transition (s_t, a_t, r_t, s_{t+1}) dans R;
    $s_t \leftarrow s_{t+1}$;
    if $t\%LEARNSTEP == 0$ **then**
      Recuperer un minibatch de taille d dans R: (s_j, a_j, r_j, s_{j+1});
      Calculer la prediction $\hat{y}_j = Q_\theta(s_j, a_j)$;
      Calculer la cible

$$y_j = \begin{cases} r_j & \text{si e se termine} \\ r_j + \gamma max_b \hat{Q}_{\theta^-}(s_{t+1}, b) & \text{sinon.} \end{cases} ;$$

      Mettre a jour θ: effectuer une etape de descente de gradient pour minimiser $(y_j - \hat{y}_j)^2$ (MSE loss);
      Mettre a jour la cible $\hat{Q}$ (tous les C pas ou en soft update);
    end
  end
end

Algorithm 2: Algorithme DQN avec reseau cible

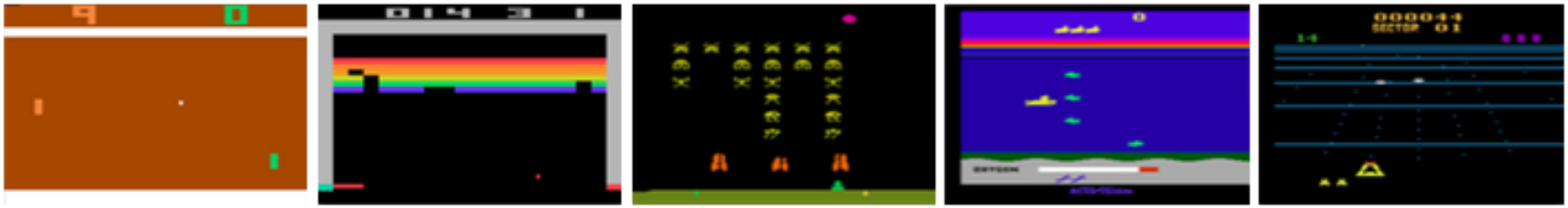
interaction avec envi
sampling_step

phase de d'échantillonage (sampling)

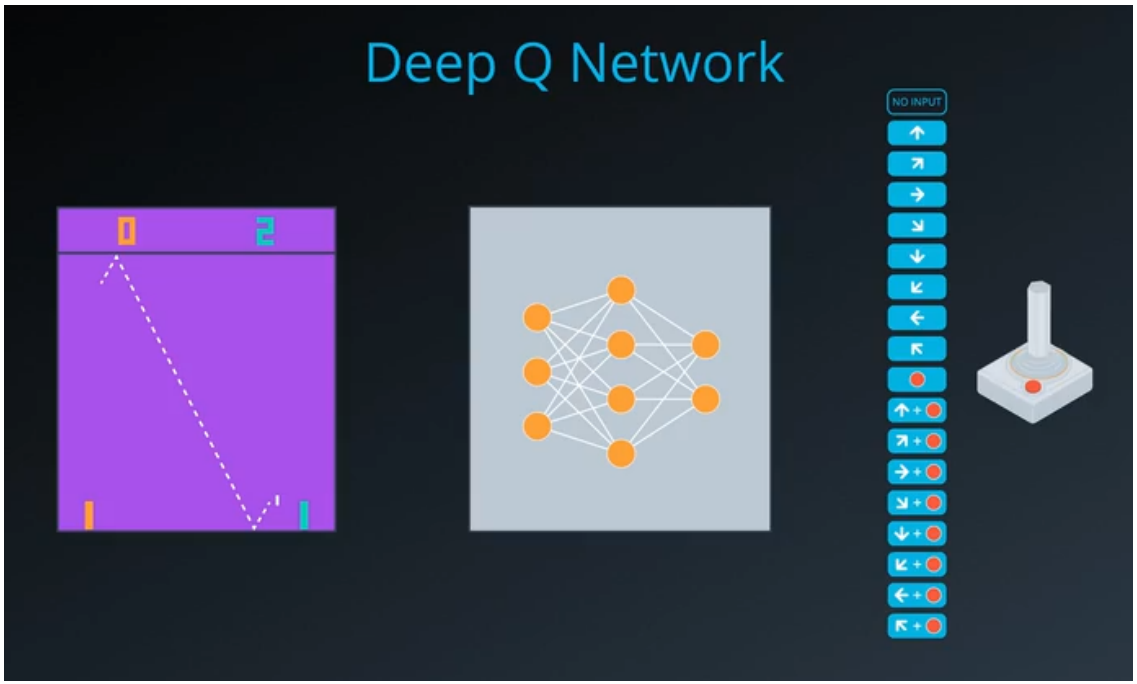
phase d'apprentissage (train_step)

Outline

Application DQN : jeux Atari



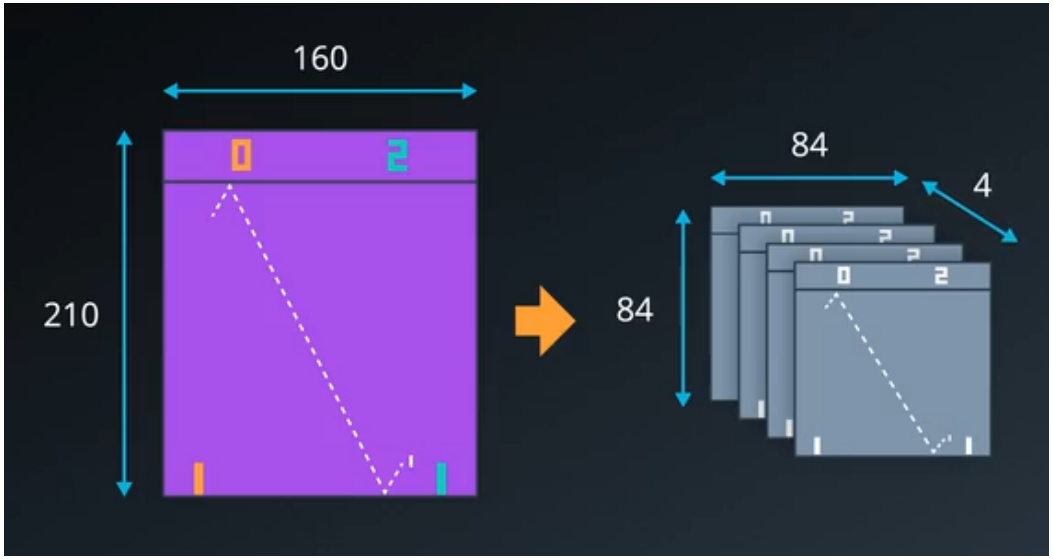
video



- 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
- réseau cible

Playing Atari with Deep Reinforcement Learning, V. Mnih et. al, 2013
Human-level control through deep reinforcement learning, V. Mnih et. al, Nature, 2015

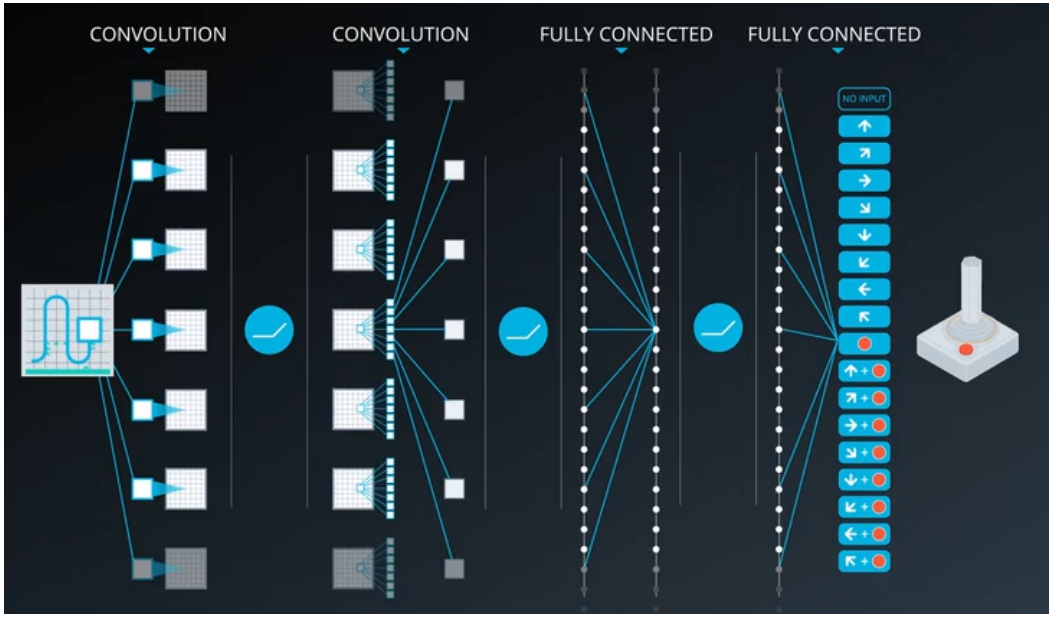
Application DQN : jeux Atari



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
- agrégation (*stack*) des 4 dernières images : réduit le problème de limitation temporelle, donne 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

DQN : Conclusion

- fonction d'approximation non-linéaire pour fonction Q, avec autant de sorties du NN que d'actions
- apprentissage supervisée avec cible mouvante
- *experience replay* pour réduire la corrélation entre interactions successives dans un même épisode
- réseau cible (*target network*) pour réduire les corrélations dues à la cible y mouvante

Limitation principale de DQN : impossible d'avoir des actions continues ...