Value Function Approximation and Deep Q Learning

Clément Rambour

Plan

Cours précédent

- Comment apprendre une bonne politique grâce à l'expérience
- Pour l'instant : représentation tabulaire de la fonction de valeur d'état ou d'état-action
- Souvent trop d'états pour être applicable

Aujourd'hui

Motivation :

- Ne pas stocker ou apprendre explicitement pour chaque état
  - Une dynamique
  - Une récompense
  - Valeur d'action-état
  - Politique
- Recherche d'une représentation compacte qui se généralise mieux

Aujourd'hui

Motivation :

- Ne pas stocker ou apprendre explicitement pour chaque état
  - Une dynamique
  - Une récompense
  - Valeur d'action-état
  - Politique
- Recherche d'une représentation compacte qui se généralise mieux
- Réduction de l'impact mémoire
- Réduction du coût en calcul
- Réduit la quantité d'expérience nécessaire

Plan

Principe

Inférence de la valeur d'un état/ d'un couple action-état grâce à un modèle

$$s \longrightarrow \boxed{\quad \boldsymbol{w} \quad} \longrightarrow \hat{V}(s)$$

$$(s, a) \longrightarrow \boxed{\quad \boldsymbol{w} \quad} \longrightarrow \hat{Q}(s, a)$$

Fonctions approximatrices

De nombreuses fonctions nont possible pour calculer la fonction de valeur :

- Combinaison linaire de features
- Réseaux de neurones
- Arbres de décisions
- Plus proches voisins
- Fourier / ondelettes

Fonctions approximatrices

De nombreuses fonctions nont possible pour calculer la fonction de valeur :

- Combinaison linaire de features $\leftarrow$ différentiable
- Réseaux de neurones $\leftarrow$ différentiable
- Arbres de décisions
- Plus proches voisins
- Fourier / ondelettes

Table of Contents

Apprentissage avec un oracle

Intuition : se ramener à un cas supervisé

- Si on connaît la valeur de $v_\pi(s) \forall s$
- Apprentissage supervisé $(s, v_\pi(s))$
- L'ojectif est d'obtenir la meilleure approximation de v_π grâce à la fonction paramétrée $\hat{v}(s, \boldsymbol{w})$

Descente de gradient stochastique

- **But** : trouver les paramètres $\boldsymbol{w}$ qui minimisent la loss $\mathcal{L}(v_\pi(s), \hat{v}(s, \boldsymbol{w}))$
- Généralement, $\mathcal{L} = \text{MSE}$:

$$\mathcal{L}(v_\pi(s), \hat{v}(s, \boldsymbol{w})) = \mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))^2]$$

- Minimum trouvé par descente de gradient :

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} \mathcal{L}$$

- Descente de gradient stochastique : gradient approché par une moyenne sur un nombre finit d'échantillons (souvent un seul)

Model Free Approximation

- Ne nécessite pas forcément d'être calculé avec un oracle/les labels
- Model free policy evaluation :
  - On suit une politique π
  - Convergence vers $\boldsymbol{v}_\pi$ ou $\boldsymbol{q}_\pi$
  - Valeurs d'état / d'actions-états stockées en mémoire
  - Mise à jour après un épisode (MC) ou après chaque étape (TD)
- **Modification** de l'approche existente pour inclure la mise à jour des paramètres du modèle

Table of Contents

Approximation linéaire

- Chaque état s est représenté par un ensemble de descripteurs/*features* : $\boldsymbol{x}(s) = [x_1(s), ..., x_n(s)]$
- L'approximation de la fonction de valeur est obtenue par approximation linéaire : $\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s)$
- La loss est donnée par $\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))^2]$
- Gradient donné par :

$$\nabla_{\boldsymbol{w}} \mathcal{L} = \mathbb{E}_\pi[2(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))] \nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w})$$

Monte Carlo Value function apprixmation

- Monte Carlo : Le retour G_t est un estimateur non biaisé de $v_\pi(s)$
- Apprentissage supervisé : $(s_1, G_1), ..., (s_T, G_T)$
- Le gradient s'écrit :

$$\nabla_{\boldsymbol{w}} \mathcal{L} = -2(G_t - \hat{v}(s, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w})$$

- La mise à jour :

$$\begin{aligned} \Delta \boldsymbol{w} &= -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} \mathcal{L} \\ &= \alpha\big(G_t - \hat{v}(s, \boldsymbol{w})\big)\nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w}) \\ &= \alpha\big(G_t - \boldsymbol{x}(s_t)^T \boldsymbol{w}\big)\boldsymbol{x}(s_t) \end{aligned}$$

Monte Carlo Value function approximation

Initialize w $= 0$, $k = 1$
loop
  Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given π
  for $t = 1, \ldots, L_k$ **do**
    if First visit to (s) in episode k **then**
      $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$
      Update weights:
    end if
  end for
  $k = k + 1$
end loop

Table of Contents

TD Learing

- Bootstraping
- Mise à jour après chaque transition (s, a, r, s') :

$$V(s) = V(s) + \alpha\big(\underbrace{r + \gamma V(s')}_{\text{TD Target}} - V(s)\big)$$

- TD Target : estimattion biaisée de $v_\pi(s)$

TD Learing

- Bootstraping
- Mise à jour après chaque transition (s, a, r, s') :

$$V(s) = V(s) + \alpha\big(\underbrace{r + \gamma V(s')}_{\text{TD Target}} - V(s)\big)$$

- TD Target : estimattion biaisée de $v_\pi(s)$
- $\Rightarrow$ Utilisation de la TD Target **approchée** comme supervision : $(s_1, r_1 + \gamma\hat{v}(s_2, \boldsymbol{w})), ..., (s_T, r_{T+1} + \gamma\hat{v}(s_T, \boldsymbol{w}))$
- **But** : Trouver les poids qui minimisent la loss :

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_\pi[(r_{t+1} + \gamma\hat{v}(s_t, \boldsymbol{w}) - \hat{v}(s_t, \boldsymbol{w}))^2]$$

- Pour TD(0), mise à jour cas linéaire :

$$\begin{aligned}\Delta\boldsymbol{w} &= \alpha\big(r + \gamma\hat{v}(s', \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w})\big)\nabla_{\boldsymbol{W}}\hat{v}(s, \boldsymbol{w}) \\ &= \alpha\big(r + \gamma\boldsymbol{x}(s')^T\boldsymbol{w} - \boldsymbol{x}(s)^T\boldsymbol{w}\big)\boldsymbol{x}(s) \qquad (1)\end{aligned}$$

TD(0) Linear value function approximaton

Initialize w = 0, $k = 1$
loop
  Sample tuple (s_k, a_k, r_k, s_{k+1}) given π
  Update weights:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$$

  $k = k + 1$
end loop

Table of Contents

Control et approximation de la valeur

- Approximation de la fonction de valeur état-action :

$$Q_\pi(s,a) \simeq \hat{Q}(s,a;\boldsymbol{w})$$

- $\hat{Q}(s,a;\boldsymbol{w})$ paramétrée par $\boldsymbol{w}$
- Schéma classique possible (légèrement modifié) :
  - Approximation de la fonction de valeur
  - ϵ-greedy amélioration

Control et approximation de la valeur

- Approximation de la fonction de valeur état-action :

$$Q_\pi(s,a) \simeq \hat{Q}(s,a;\boldsymbol{w})$$

- $\hat{Q}(s,a;\boldsymbol{w})$ paramétrée par $\boldsymbol{w}$
- Schéma classique possible (légèrement modifié) :
  - Approximation de la fonction de valeur
  - ϵ-greedy amélioration
- Parfois instable
- Amélioration possible :
  - Approximation bien choisie
  - Bootstrapping
  - Off-policy learning

Contrôle avec un oracle

- $Q_\pi(s,a) \simeq \hat{Q}(s,a;\boldsymbol{w})$ Mean Square Error :

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_\pi[(Q_\pi(s,a) - \hat{Q}(s,a;\boldsymbol{w}))^2]$$

- Minimum obtenu par stoachastic gradient descent :

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\mathbb{E}_\pi\left[\big(Q_\pi(s,a) - \hat{Q}(s,a;\boldsymbol{w})\big)\nabla_{\boldsymbol{w}}\hat{Q}(s,a;\boldsymbol{w})\right]$$

SARSA et approximation de la valeur

- Résumé en deux étapes :
  - À partir de s choix d'une action a associée à une politique ϵ-greedy(π) $\rightarrow s'$
  - Mise à jour de la fonction de valeur en fonction du couple (s', a') associé à la prochaine action :

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha\big(r + \gamma Q_\pi(s',a') - Q(s,a)\big)$$

- SARSA avec approximation $Q_\pi(s,a) \simeq \hat{Q}(s,a;\boldsymbol{w})$
- Loss avec ojectif SARSA :

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_\pi[(r + \gamma Q_\pi(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w}))^2]$$

$$\mathcal{L}(\boldsymbol{w}) \simeq (r + \gamma Q_\pi(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w}))^2$$

- gradient de la loss % $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = 2\big(r + \gamma\hat{Q}(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w})\big)\nabla_{\boldsymbol{w}}\hat{Q}(s,a;\boldsymbol{w})$$

- Mise à jour :

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$$

SARSA et approximation de la valeur : cas linéaire

- Couple états actions représentés par un vecteur de features : $\boldsymbol{x}(s,a) = [x_1(s,a), ..., x_n(s,a)]$
- Approximation linéaire :

$$\hat{Q}(s,a;\boldsymbol{w}) = \boldsymbol{w}^t \boldsymbol{x}(s,a)$$

- Loss avec ojectif SARSA :

$$\mathcal{L}(\boldsymbol{w}) \simeq (r + \gamma Q_\pi(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w}))^2$$

- gradient de la loss % $\boldsymbol{w}$ cas linéaire :

$$\begin{aligned}\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) &= 2\big(r + \gamma\hat{Q}(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w})\big)\nabla_{\boldsymbol{w}}\hat{Q}(s,a;\boldsymbol{w}) \\ &= \big(r + \gamma\boldsymbol{x}(s',a')^T\boldsymbol{w} - \boldsymbol{x}(s,a)^T\boldsymbol{w}\big)\boldsymbol{x}(s,a)\end{aligned}$$

- Mise à jour :

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$$

Q-learning et approximation de la valeur

- Résumé en deux étapes :
  - À partir de s choix d'une action a associée à une politique ϵ-greedy$(\pi) \leftarrow s'$
  - Mise à jour de la fonction de valeur en fonction du couple (s', a') associé à la **meilleure** action possible :

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha\big(r + \gamma \max_{a'} Q_\pi(s',a') - Q(s,a)\big)$$

- Q-learning avec approximation $Q_\pi(s,a) \simeq \hat{Q}(s,a;\boldsymbol{w})$
- Loss avec ojectif Q-learning :

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_\pi[(r + \gamma \max_{a'} Q_\pi(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w}))^2]$$

$$\mathcal{L}(\boldsymbol{w}) \simeq (r + \max_{a'} \gamma Q_\pi(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w}))^2$$

- gradient de la loss % $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = 2\big(r + \gamma \max_{a'} \hat{Q}(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w})\big)\nabla_{\boldsymbol{w}} \hat{Q}(s,a;\boldsymbol{w})$$

- Mise à jour :

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

Q-learning et approximation de la valeur : cas linéaire

- Couple états actions représentés par un vecteur de features : $\boldsymbol{x}(s,a) = [x_1(s,a), ..., x_n(s,a)]$
- Approximation linéaire :

$$\hat{Q}(s,a;\boldsymbol{w}) = \boldsymbol{w}^t \boldsymbol{x}(s,a)$$

- Loss avec ojectif Q-learning :

$$\mathcal{L}(\boldsymbol{w}) \simeq (r + \max_{a'} \gamma Q_\pi(s', a'; \boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w}))^2$$

- gradient de la loss % $\boldsymbol{w}$ cas linéaire :

$$\begin{aligned}\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) &= 2\big(r + \gamma \max_{a'} \hat{Q}(s',a';\boldsymbol{w}) - \hat{Q}(s,a;\boldsymbol{w})\big) \nabla_{\boldsymbol{w}} \hat{Q}(s,a;\boldsymbol{w}) \\ &= \big(r + \gamma \max_{a'} \boldsymbol{x}(s',a')^T \boldsymbol{w} - \boldsymbol{x}(s,a)^T \boldsymbol{w}\big) \boldsymbol{x}(s,a)\end{aligned}$$

- Mise à jour :

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2} \alpha \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

Plan

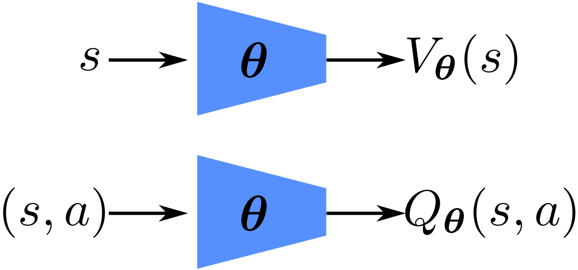
Deep RL

- Utiliser des réseaux de neurones pour approcher
  - La fonction de valeur (v ou Q)
  - La politique
  - Le modèle (de l'environnement)
- Optimisation par descente de gradient stochastique

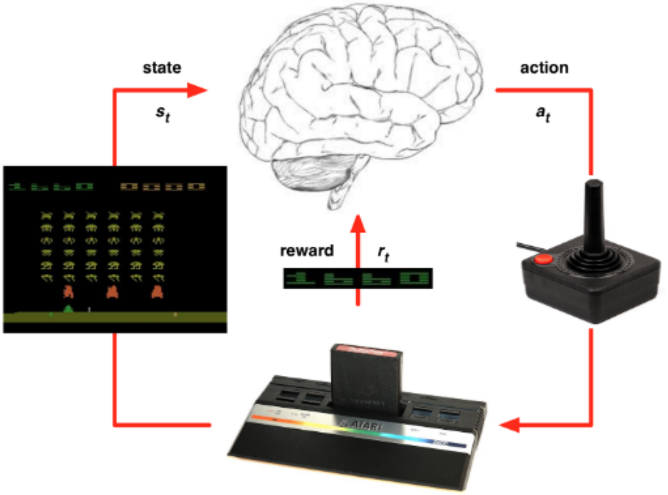
Table of Contents

Deep Q-Networks (DQNs)

Représentation de la table de valeur état-action par un *Q-network* de paramètre $\boldsymbol{\theta}$

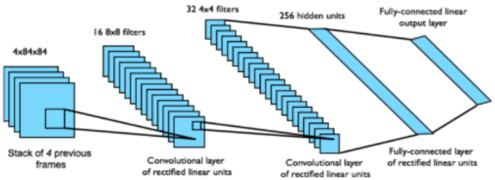


DQNs dans Atari



DQNs dans Atari

- Apprentissage *end to end* des valeurs de $Q(s, a)$ d'après les pixels $\boldsymbol{s} \leftarrow$ vecteur de features décrivant l'état
- Entrée du réseau $\boldsymbol{s}$ concaténation des dernières frames observées (ex : 4 dernières)
- Sortie : $Q(s, a)$ pour les actions réalisables ($\sim$ 18 bouttons)
- Architectures et hyperparmètres fixés pour toute la partie



DQ-learning

- Loss : MSE optimisée par descente de gradient stochastique
- Deux problèmes (principaux) pouvant amener à diverger :
  - Correlation entre les échantillons
  - *Targets*/labels non stationnaires
- Deux solutions proposées pour le Deep Q-Learning (Mnih et Al. 2015)
  - Experience replay
  - Fixed Q-targets

DQNs : Experience replay

- Pour éviter la forte correlation entre échantillons successifs : les données sont stockées dans un *buffer* $\mathcal{D} \leftarrow$ *replay buffer*

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

- *Experience replay* consiste à répéter les étapes suivantes :
  - Échantillonnage d'une expérience $(s, a, r, s' \sim \mathcal{D})$
  - Calcul de la TD *target* pour l'échantillon : $r + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s', a')$
  - Calcul de la loss :
$$\mathcal{L}(\boldsymbol{\theta}) = \left(r + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s', a') - Q_{\boldsymbol{\theta}}(s, a)\right)^2$$
  - Mise à jour des paramètres :
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{2} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \big(r + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s', a') - Q_{\boldsymbol{\theta}}(s, a)\big) \nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(s, a)$$

Fixed Q-Targets

- Pour améliorer la stabilité, on peut fixer les poids du réseaux pour évaluer la TD *target*
- Deux réseaux : un réseaux cible et un réseaux apprenant la fonction de valeur
- $\boldsymbol{\theta}^-$: les paramètres du réseau cible
- $\boldsymbol{\theta}$: les paramètres du réseaux mis à jour
- Léger changement dans le schéma précédent :
  - Échantillonnage d'une expérience $(s, a, r, s') \sim \mathcal{D}$
  - Calcul de la TD *target* pour l'échantillon : $r + \gamma \max_{a'} Q_{\boldsymbol{\theta}^-}(s', a')$
  - Calcul de la loss :

$$\mathcal{L}(\boldsymbol{\theta}) = \left(r + \gamma \max_{a'} Q_{\boldsymbol{\theta}^-}(s', a') - Q_{\boldsymbol{\theta}}(s, a)\right)^2$$

  - Mise à jour des paramètres :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{2} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Big(r + \gamma \max_{a'} Q_{\boldsymbol{\theta}^-}(s', a') - Q_{\boldsymbol{\theta}}(s, a)\Big) \nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(s, a)$$

DQN Pseudocode

Input $\color{blue}{C}$, $\color{blue}{\alpha}$, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
Get initial state s_0
loop
    Sample action a_t given ϵ-greedy policy for current $\hat{Q}(s_t, a; \boldsymbol{w})$
    Observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D
    Sample random minibatch of tuples (s_i, a_i, r_i, s_{i+1}) from D
    for j in minibatch **do**
        if episode terminated at step $i + 1$ **then**
            $y_i = r_i$
        else
            $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \boldsymbol{w}^-)$
        end if
        Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))^2$ for parameters $\boldsymbol{w}$: $\Delta\boldsymbol{w} = \alpha(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{Q}(s_i, a_i; \boldsymbol{w})$
    end for
    $t = t + 1$
    if mod(t,C) == 0 **then**
        $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
    end if
end loop

DQN results dans Atari

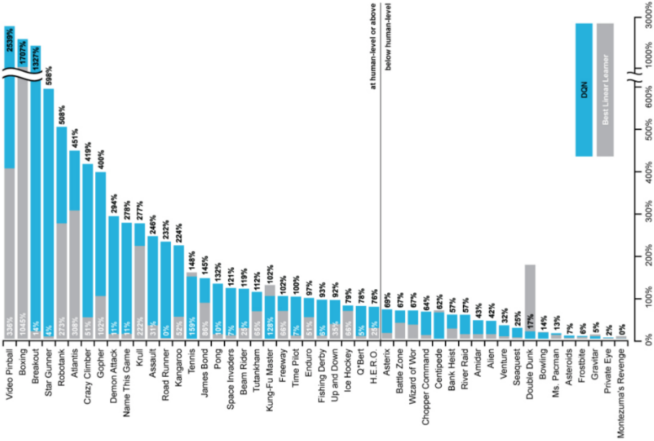


Figure – Human-level control through deep reinforcement learning, Mnih et al, 2015

Vanilla DQN vs. Fixed Target and experience replay

Game	Linear	Deep Network	DQN w/ fixed Q	DQN w/ replay	DQN w/replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Seaquest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

Table of Contents

Double DQN

- Q-learning : biaisé vers l'estimation du maximum dans la table Q
- Combinaison d'estimateur peut réduire le biais ($\sim$ bagging)
- Double Q-Learning :
  - Combinaison de deux estimations de Q
  - Probabilité non nulle de sélectionner la meileure option pour l'une ou pour l'autre des estimations
  - réduction du biais

Prioritized experience replay

- Soit i l'index du i-eme tuple dans le buffer d'expérience (s_i, a_i, r_i, s_{i+1})
- Échantillonnage des tuples pour mettre à jour suivant une distribution p
- Probabilité p_i de sélectionner le tuple i proportionelle à l'erreur DQN e_i :

$$e_i = |r + \max_{a'} Q_{\boldsymbol{\theta}}(s_{i+1}, a') - Q_{\boldsymbol{\theta}}(s_i, a_i)|$$

- Mise à jour des p_i à chaque mise à jour de $Q_{\boldsymbol{\theta}}$
- une méthode :

$$p_i = \frac{p_i^\beta}{\sum_k p_k^\beta}$$

- β facteur de température

Table of Contents

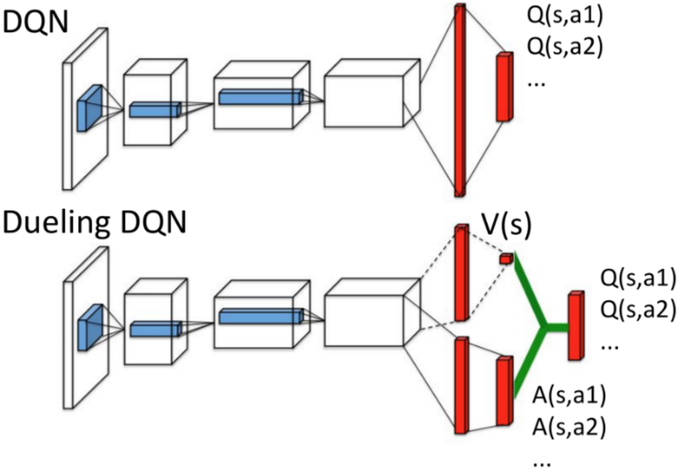
Fonctions valeur et avantage

- **Intuition** : représentations pour décrire les états pas forcément adaptées pour décrire les actions
- Exemple : le score dans un jeu indique l'intérêt d'un état s : $v(s)$ mais pas l'intérêt entre deux action $Q(s, a_1) \lesseqgtr Q(s, a_2)$
- Fonction d'avantage / *Advantage function* :

$$A_\pi(s, a) = Q_\pi(s, a) - v_\pi(s, a)$$

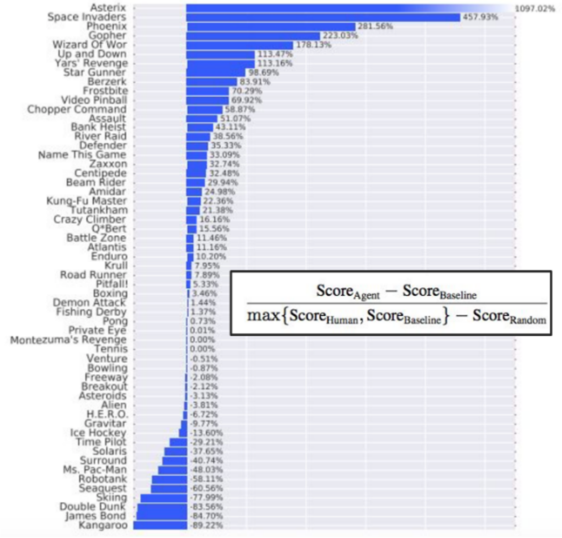
- A_π donne la valeur d'une action

Dueling DQN



Wang et.al., ICML, 2016

Dueling DQN


Asterix 1097.02%
Space Invaders 457.93%
Phoenix 281.56%
Gopher 223.03%
Wizard Of Wor 178.13%
Up and Down 113.47%
Yars' Revenge 113.16%
Star Gunner 98.69%
Berzerk 83.91%
Frostbite 70.29%
Video Pinball 69.92%
Chopper Command 58.87%
Assault 51.07%
Bank Heist 43.11%
River Raid 38.56%
Defender 35.33%
Name This Game 33.09%
Zaxxon 32.74%
Centipede 32.48%
Beam Rider 29.94%
Amidar 24.98%
Kung-Fu Master 22.36%
Tutankham 21.38%
Crazy Climber 16.16%
Q*Bert 15.56%
Battle Zone 11.46%
Atlantis 11.16%
Enduro 10.20%
Krull 7.95%
Road Runner 7.89%
Pitfall! 5.33%
Boxing 3.46%
Demon Attack 1.44%
Fishing Derby 1.37%
Pong 0.73%
Private Eye 0.01%
Montezuma's Revenge 0.00%
Tennis 0.00%
Venture -0.51%
Bowling -0.87%
Freeway -2.08%
Breakout -2.12%
Asteroids -3.13%
Alien -3.81%
H.E.R.O. -6.72%
Gravitar -9.77%
Ice Hockey -13.60%
Time Pilot -29.21%
Solaris -37.65%
Surround -40.74%
Ms. Pac-Man -48.03%
Robotank -58.11%
Seaquest -60.56%
Skiing -77.99%
Double Dunk -83.56%
James Bond -84.70%
Kangaroo -89.22%
$\frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Baseline}}}{\max\{\text{Score}_{\text{Human}}, \text{Score}_{\text{Baseline}}\} - \text{Score}_{\text{Random}}}$

Non unicité de l'avantage

- Q est reconstruit à partir de l'avantage et la valeur d'état :

$$Q_{\boldsymbol{\theta}}(s,a) = V_{\boldsymbol{\theta}}(s,a) + A_{\boldsymbol{\theta}}(s,a)$$

- Infinité de V et A donnant le même Q : solution à une constante prêt
- Nécessité de "punaiser" V et A en rajoutant une constante

Non unicité de l'avantage

- Q est reconstruit à partir de l'avantage et la valeur d'état :

$$Q_{\boldsymbol{\theta}}(s,a) = V_{\boldsymbol{\theta}}(s,a) + A_{\boldsymbol{\theta}}(s,a)$$

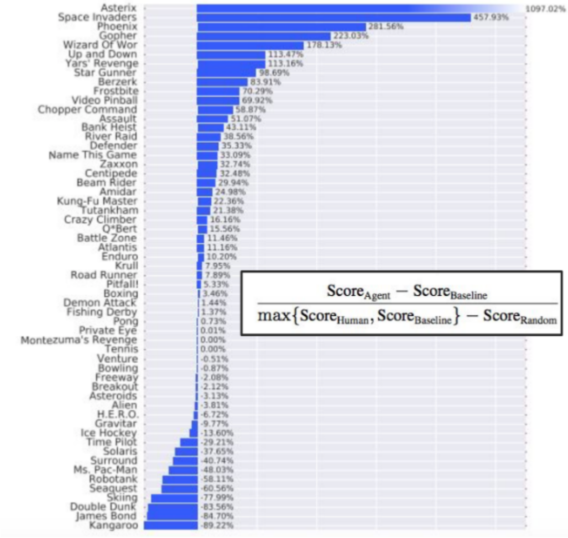
- Infinité de V et A donnant le même Q : solution à une constante prêt
- Nécessité de "punaiser" V et A en rajoutant une constante
- Méthode 1 : Forcer $Q_{\boldsymbol{\theta}}(s,a) = V_{\boldsymbol{\theta}}(s)$ pour la meilleure action possible

$$Q_{\boldsymbol{\theta}}(s,a) = V_{\boldsymbol{\theta}}(s,a) + \left(A_{\boldsymbol{\theta}}(s,a) - \max_{a'} A_{\boldsymbol{\theta}}(s,a')\right)$$

- Méthode 2 : Utiliser la moyenne sur les actions $\overline{A}_{\boldsymbol{\theta}}(s,a')$ comme origine (plus stable)

$$Q_{\boldsymbol{\theta}}(s,a) = V_{\boldsymbol{\theta}}(s,a) + \left(A_{\boldsymbol{\theta}}(s,a) - \overline{A}_{\boldsymbol{\theta}}(s,a')\right)$$

Dueling DQN vs. Double DQN



Conseils pratiques

- Huber Loss sur l'erreur de Bellman (= l'erreur TD)
- Double DQN (peu d'effort par rapport à simple DQN)
- Essayer plusieurs seeds pendant les experiences
- Learning rate scheduling. Haut learning rate au départ pour l'exploration

Plan

Bibliographie

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)