IFT 608 / IFT 702
Planification en intelligence artificielle

Méthodes *Policy Gradient*

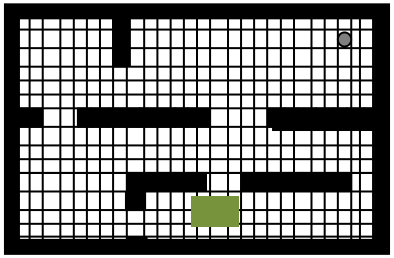
Professeur: Froduald Kabanza

Assistants: D'Jeff Nkashama & Jordan Félicien Masakuna

Université de Sherbrooke

Sujet couvert

- Policy-gradient
- Reinforce
- Actor-Critic



Cadre général

Maximiser $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, avec $0 \leq \gamma \leq 1$

Agent

Politique

$\pi(s,a)$

Action a

$a_1 \rightarrow a_2 \rightarrow \ldots\ a_t$

$$U^{\pi}(s) = \Sigma_{s' \in S} P(s'|s, \pi(s))\, [R(s, \pi(s), s') + \gamma\, U^{\pi}(s')]$$

$$U(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a)\, [\, R(s,a,s') + \gamma\, U(s')\,]$$

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a)\, [\, R(s,a,s') + \gamma \max_{a'} Q(s',a')\,]$$

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} Q(s,a) = \operatorname*{argmax}_{a \in A(s)} \sum_{s' \in S} P(s'|s,a)\, [\, R(s,a,s') + \gamma\, U(s')\,]$$

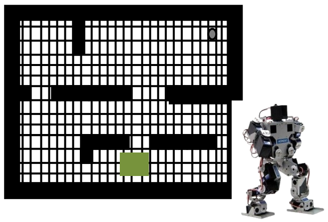
Récompense r

$r_1 \rightarrow r_2 \rightarrow \ldots\ r_t$

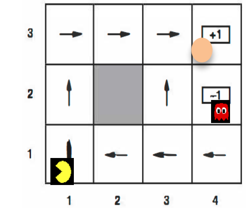
Environnement

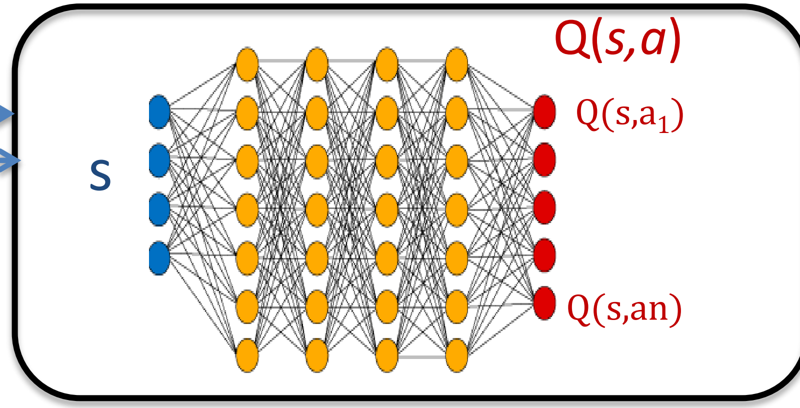
État s

$s_1 \rightarrow s_2 \rightarrow \ldots\ s_t$



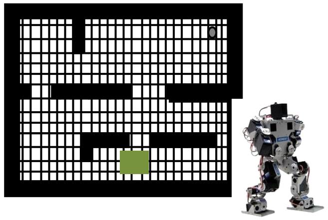
DQN (Deep Q-Network)



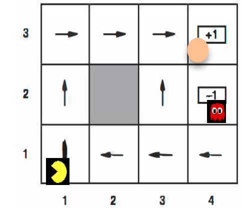


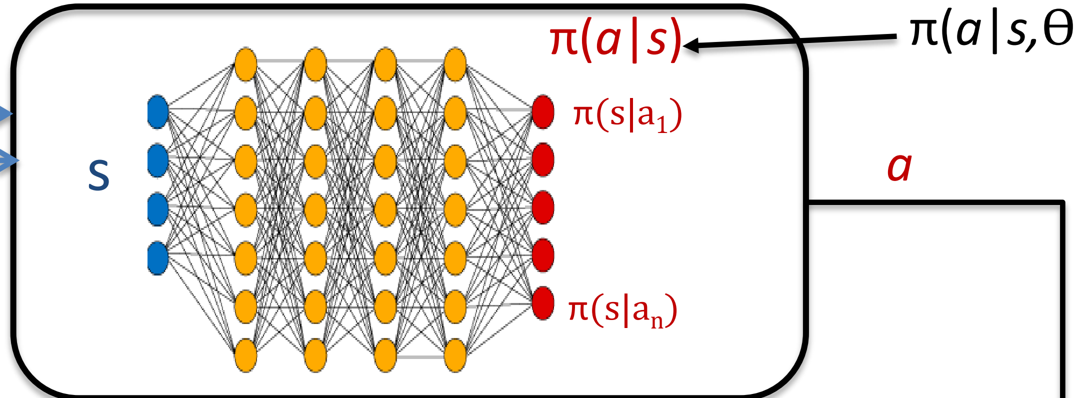
Deux reseaux : Q_{Target} et Q

$$\text{Loss} = \sum_{(s,a)\,\epsilon\,\text{Batch}} (\text{Predicted Q-Values} - \text{Target Q-Values})^2 \;/\; \text{BatchSize}$$



Méthodes *Policy-Gradient*





Politique π

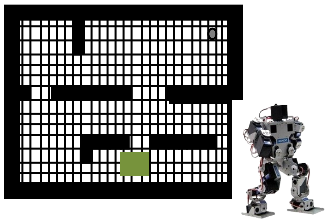
- Probabiliste (mixte)
- Approximée par un réseau de neurones

Une famille de méthodes

- *Reinforce*
- *Actor-Critic*

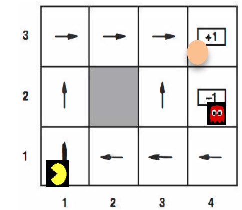
r

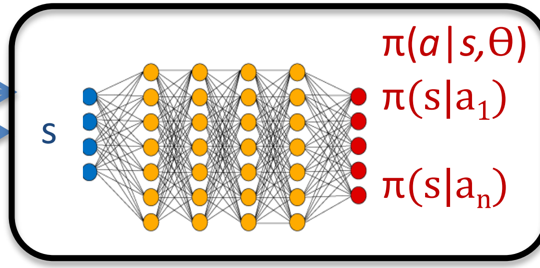
s



Algorithme *Reinforce*

Monte-Carlo Policy Gradient

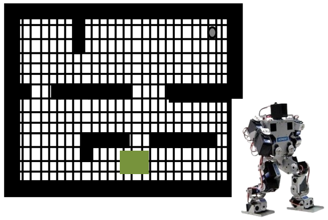




Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

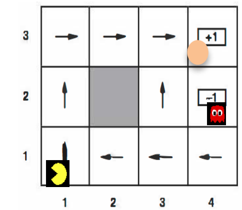
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ *(G_t)*
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

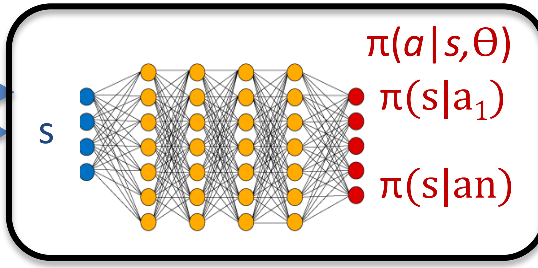
Exemple d'échantillon: $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$



Algorithme *Reinforce*

Monte-Carlo Policy Gradient





a

Algorithme en texte …

Répéter sans fin (pour chaque episode)

Génère un échantillon en utilisant la politique courante

À chaque transition de l'échantillon:

Calcule la recompense cumulée escomptée $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

Calcule le log de la politique probabiliste $\ln \pi(A_t|S_t, \theta)$ ← Rétropapagation du gradient

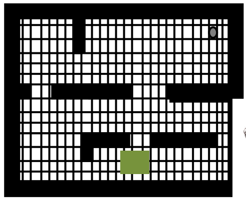
Calcule le gradient de la politique $\nabla \ln \pi(A_t|S_t, \theta)$

Mets à jour les poids du réseau $\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \theta)$

Exemple d'échantillon: $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$

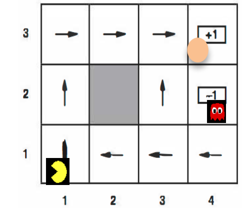
r

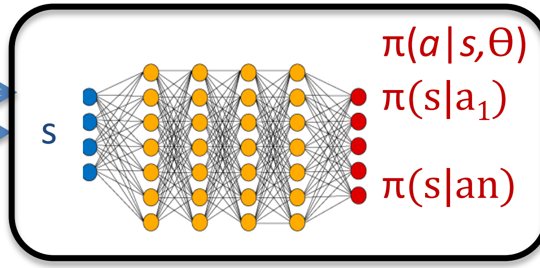
s



Algorithme *Reinforce*

Monte-Carlo Policy Gradient





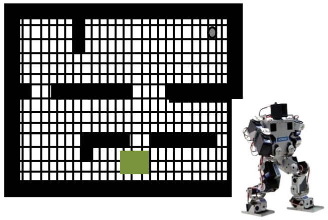
a

Dérivation mathématique de la règle d'apprentissage

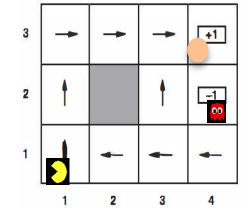
- Sutton & Barto, Sections 13.1 à 13.3
- [Chris Youn, Medium,]*"Deriving Policy Gradients and Implementing REINFORCE"*

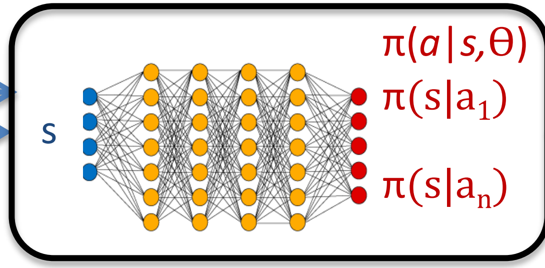
r

s



Reinforce with Baseline





a

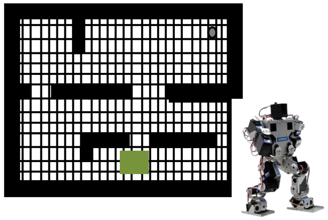
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ *(G_t)*
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

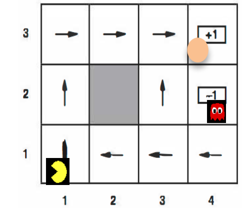
Exemple d'échantillon: $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$

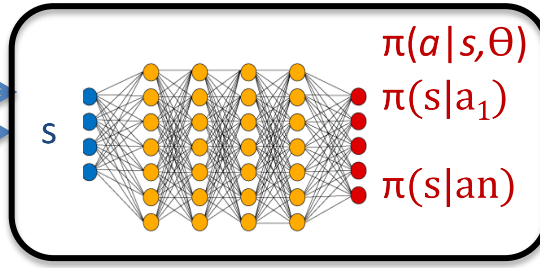
r

s



Actor-Critic





a

L'idee de Acto-Critic découle de *Reinforce with Baseline:* comme Baseline, estimer la valeur de l'état

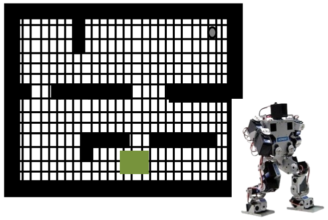
Ainsi:

- La "Critique" estime la valeur de l'état (*Q-Value* ou *Value Fonction*) à l'inage de Reinforce avec Baseline
- L' "Acteur" mets à jour les poids de la (distribution de la) politique comme dans Reinforce, en suivant la direction suggérée par la "Critique"

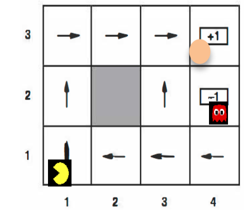
Les deux sont paramétrés par des reseaux de neurones

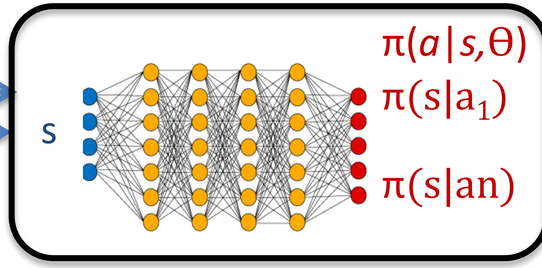
r

s



Actor-Critic





a

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Chris Youn, Towards Data Science
Understanding Actor-Critic Methods

r

s

Imitation Learning

Comme lecture personnel ou sujets avancés pour les étudiants gradués:

- **Learning from Demonstration (Behaviour Cloning):** Apprendre la politique d'un expert par renforcement
- **Inverse Reinforcement Learning**: Apprendre la fonction de récompense et la politique à partir des observations (ou démonstrations)

Références:

- Zoltan Lorincz, Medium: A brief overview of Imitation Learning
- James Teddy, Medium: OpenAI's new approach for one-shot imitation learning, a peek into the future of AI
  - **One-Shot Imitation Learning (**https://arxiv.org/abs/1703.07326)
    Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, Wojciech Zaremba

Vous devriez être capable de...

- Expliquer et implémenter les algorithmes suivants
  - Reinforce
  - Reinforce with Baseline
  - Actor-Critic