

Apprentissage par renforcement

Cours 7: Curriculum RL, Hierarchical RL & Multi-Agents

Sylvain Lamprier

UE RLD - Master DAC

2019

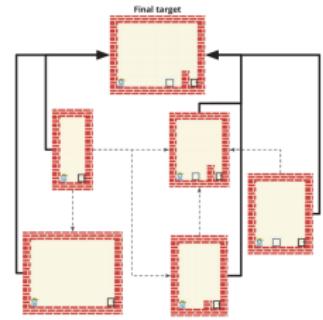
Curriculum Reinforcement Learning (& Rewards Intrinsèques)

Curriculum Learning : Augmentation graduelle de la difficulté des problèmes

- Lorsque le problème est très complexe, il peut être efficace de ne pas l'attaquer de front
- Mais de définir un parcours d'apprentissage adapté
 - ⇒ Définition des problèmes d'entraînement : Souvent Relaxations du problème
 - ⇒ Transfert des compétences apprises sur une sous-tâche à une tâche suivante ?
 - ⇒ Définition d'un graphe de problèmes à résoudre :

Algorithm Curriculum learning

- 1: **Given:** curriculum $C = \langle V, E \rangle$
- 2: $L = \{\}$ set of learned tasks
- 3: $Q = \{\}$ set of value functions learned.
- 4: **while** $V \setminus L \neq \emptyset$ **do**
- 5: pick a task t in $V \setminus L$ such that $X_t \subseteq L$
- 6: $Q_t = \text{learn}(t, \{Q_i | t_i \in X_t\})$
- 7: $L = L \cup \{t\}$, $Q = Q \cup \{Q_t\}$
- 8: Return Q_{tf}



Curriculum Learning : Sous-tâches

Un domaine \mathcal{D} contient l'ensemble des MDPs exprimables [Nar+16] :

- en faisant varier les degrés de liberté du domaine $F = [F_0, \dots, F_M]$
 - Aux échecs un degré de liberté pourrait être la taille de l'échiquier, un autre le nombre de pièces d'un certain type, etc.
 - Chaque $F_i \in F$ possède un intervalle de valeurs possibles $\text{Rnge}(F_i)$, ordonné selon la complexité.
- ou en appliquant des restrictions au MDP (e.g., suppression d'états, d'actions, modification des probabilités initiales, terminales, etc.).

[Nar+16] propose un ensemble de modifications (heuristiques) du problème :

- Task Simplification : on remplace une ou plusieurs caractéristiques F_i par une valeur inférieure
- Promizing initialization : on concentre les probabilité d'état initial autour des zones de fort reward (utile dans le cas de rewards très sparses)
- Action Simplification : retrait des actions qui tendent souvent à des mauvais taux de récompenses dans le problème cible
- Mistake Learning : ajout de chemins pour reconsiderer des mauvais choix constatés dans la tâche cible
- Ajout de sous-tâches, etc.

... Mais toutes ces modifications nécessitent une connaissance experte de la tâche cible !

Curriculum Learning : Transfert

Comment transférer les compétences apprises entre tâches successives ?

⇒ Initialiser π ou Q selon tâches précédentes...

⇒ ...ou Reward shaping [NHR99]

- Ajout d'un signal de reward additionnel, dépendant de la tâche apprise :
 $r'_t = r_t + f_t$, avec f une fonction de "shaping"
- Une fonction de shaping permet de rajouter des feedbacks à la récompense de l'environnement (e.g., pour densifier les rewards ou donner des conseils experts)
- [NHR99] montre qu'avec f_t de la forme d'une différence de potentiels :
 $f_t = \gamma\phi(s_{t+1}) - \phi(s_t)$, on peut retrouver π^* à partir des valeurs optimales Q^F d'une procédure type Q-Learning qui utilise $r_t + f_t$:

À convergence, on a :

$$Q^F(s, a) = \mathbb{E}_{s'}[R(s, a, s') + F(s, a, s') + \gamma \max_{a'} Q^F(s', a')] \text{ (optimalité de Bellman)}$$

Soit $Q'(s, a) = Q^F(s, a) + \phi(s)$, on a alors $Q^F(s, a) = Q'(s, a) - \phi(s)$ et :

$$\begin{aligned} Q'(s, a) &= \mathbb{E}_{s'}[R(s, a, s') + \gamma\phi(s') - \phi(s) + \gamma \max_{a'} Q^F(s', a')] + \phi(s) \\ &= \mathbb{E}_{s'}[R(s, a, s') + \gamma \max_{a'} (Q^F(s', a') + \gamma\phi(s'))] \\ &= \mathbb{E}_{s'}[R(s, a, s') + \gamma \max_{a'} Q'(s', a')] \end{aligned}$$

⇒ Equation d'optimalité de Bellman, donc : $Q'(s, a) = Q^*(s, a)$

⇒ $Q^*(s, a) = Q^F(s, a) + \phi(s)$

$$\pi^*(s) = \arg \max_a Q^*(s, a) = \arg \max_a Q^F(s, a) + \phi(s) = \arg \max_a Q^F(s, a) = \pi^F(s)$$

Curriculum Learning : Reward Shaping

Reward Shaping

- ⇒ La politique π^F déduite des valeurs de Q^F est alors équivalente à la politique optimale qu'on obtiendrait sans reward shaping
- ⇒ ... Mais les valeurs F peuvent être utiles à la convergence de l'algorithme
- On note la similarité avec Actor-Critic basé sur $r_t + \gamma V(s_{t+1}) - V(s_t)$

Mais la forme de $f_t = \gamma\phi(s_{t+1}) - \phi(s_t)$ limitée à des guidages uniquement sur les états

- [WCE03] étend le reward shaping aux "look ahead advices" qui permettent de considérer des conseils etat-action
- Avec $f_t = \gamma\phi(s_{t+1}, a_{t+1}) - \phi(s_t, a_t)$, on peut retrouver π^* à partir des valeurs optimales Q^F d'une procédure type Q-Learning qui utilise $r_t + f_t$ et une politique cible biaisée $\pi^b(s) = \arg \max_a Q(s, a) + \phi(s, a)$:

À convergence, on a (optimalité de Bellman) :

$$Q^F(s, a) = \mathbb{E}_{s'}[R(s, a, s') + F(s, a, s') + \gamma Q^F(s', a')] \text{ avec } a' = \arg \max_a Q^F(s', a) + \phi(s', a)$$

Soit $Q'(s, a) = Q^F(s, a) + \phi(s, a)$, on a alors $Q^F(s, a) = Q'(s, a) - \phi(s, a)$ et :

$$\begin{aligned} Q'(s, a) &= \mathbb{E}_{s'}[R(s, a, s') + \gamma(Q^F(s', a') + \phi(s', a'))] \text{ avec } a' = \arg \max_a Q^F(s', a) + \phi(s', a) \\ &= \mathbb{E}_{s'}[R(s, a, s') + \gamma Q'(s', a')] \text{ avec } a' = \arg \max_a Q'(s', a) \end{aligned}$$

⇒ Equation d'optimalité de Bellman, donc : $Q'(s, a) = Q^*(s, a)$

⇒ $Q^*(s, a) = Q^F(s, a) + \phi(s, a)$

⇒ On peut alors retrouver les valeurs de la politique optimale selon R , tout en se servant des F pour explorer

Curriculum Learning : Parcours

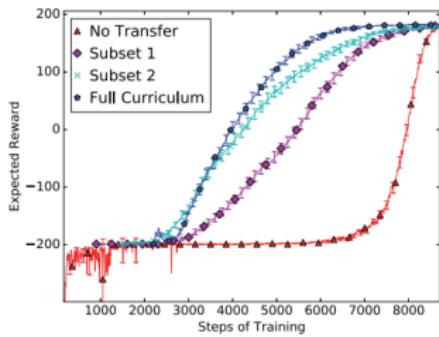
[Sve+17] se sert du reward shaping pour transférer les connaissances dans un parcours de curriculum :

- Domaine d'application de Q_i de la tâche i sur les états S_j de la tâche j : $D_{Q_i}(j) = \{s_j \in S_j \mid \exists a \in A_i \text{ tel que } Q_i(s_j, a) \text{ est défini}\}$
- Potentiel de transfert de i vers j : $v(i, j) = \frac{D_{Q_i}(j)}{1 + |S_i| - |S_j|}$ (heuristique : plus on peut reutiliser les valeurs Q de i , plus on considère un bon transfert)
- Clustering des tâches selon caractéristiques puis création de dépendances entre tâches ou groupes de tâches si potentiel de transfert $\geq \epsilon$
- Reward shaping :

$$\phi_{i,j}(s, a) = \begin{cases} Q_i(s, a) & \text{si } s \in D_{Q_i}(j) \wedge a \in A_i \\ 0 & \text{sinon} \end{cases}$$

$$\phi_j(s, a) = \sum_{i \in X_j} \phi_{i,j}(s, a)$$

avec X_j l'ensemble des tâches parentes de la tâche j dans le graphe de curriculum

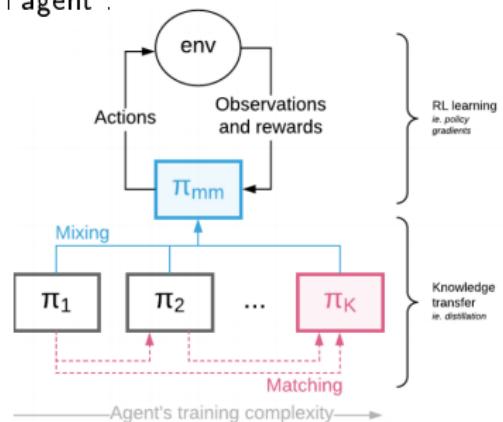


Curriculum Learning : Mix&Match

... Mais établissement des tâches très heuristique (et dépendant de la tâche finale) !

Plutôt que de chercher des environnements plus simples, Mix&Match [Cza+18] propose de travailler avec la complexité croissante de l'agent :

- K agents de politiques respectives π_1, \dots, π_K (éventuellement partage de certains paramètres)
- π_K est l'agent final que l'on souhaite entraîner
- Capacités croissantes de π_1 à π_K (par exemple NN de plus en plus gros ou utilisation de plus d'informations)
- Mixture de politiques :
$$\pi_{mm}(a|s) = \sum_{i=1}^K \alpha_i \pi_i(a|s),$$
 avec α un vecteur de paramètres sommant à 1



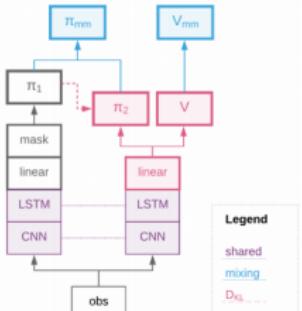
Problème d'optimisation : $\mathcal{L}(\theta) = \mathcal{L}_{RL}(\pi_{mm}; \theta) + \lambda \mathcal{L}_{mm}(\theta)$, avec $\mathcal{L}_{RL}(\pi_{mm}; \theta)$ un coût de renforcement à optimiser selon des trajectoires collectées (par exemple selon A2C) et $\mathcal{L}_{mm}(\theta)$ un coût d'incitation au transfert de connaissances entre politiques (policy distillation) :

$$\mathcal{L}_{mm}(\theta) = \frac{1}{K} \sum_{i=1}^K \frac{1}{|\text{par}(i)|} \sum_{j \in \text{par}(i)} \frac{\alpha_j}{|\mathcal{R}|} \sum_{\tau \in \mathcal{R}} \sum_{t=1}^{|\tau|} D_{KL}(\pi_j(.|s_t) || \pi_i(.|s_t))$$

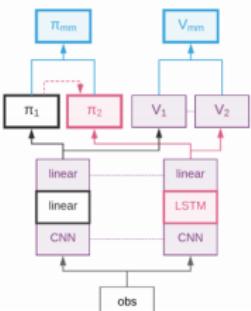
Optimisation de θ et α selon Population Based Training of neural networks[Jad+17]

- Paramétrisation online des hyper-paramètres α selon population de π_{mm}
- En début de process $a_1 = 1$, puis évolution jusqu'à $a_K = 1$

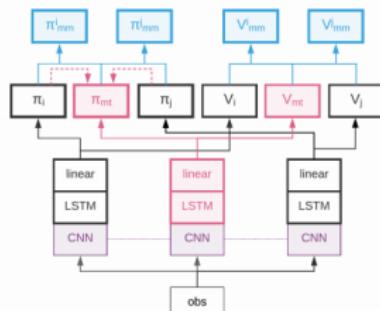
Curriculum Learning : Mix& Match



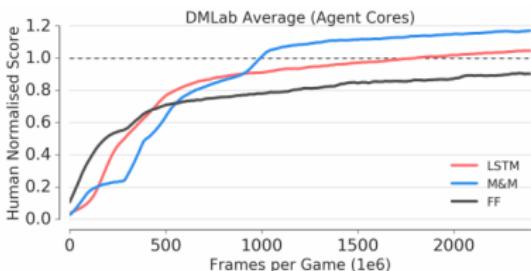
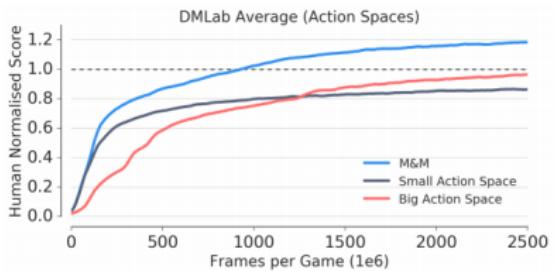
(a) M&M for action spaces progression



(b) M&M for architecture progression



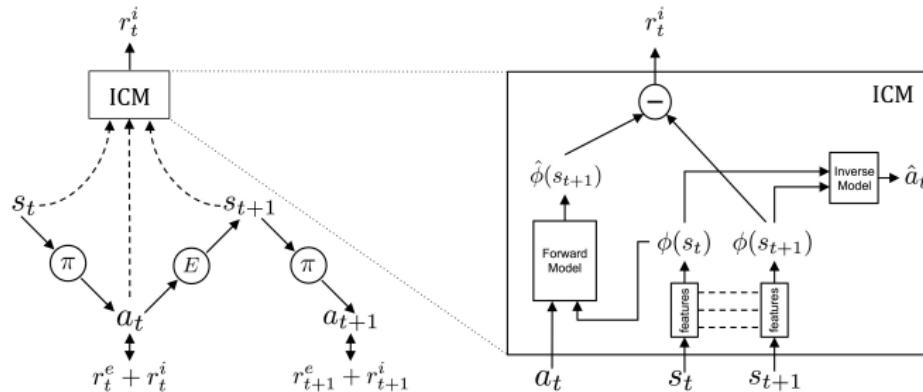
(c) M&M for multitask progression



Curiosity Driven RL

Curiosity Driven RL [Pat+17]

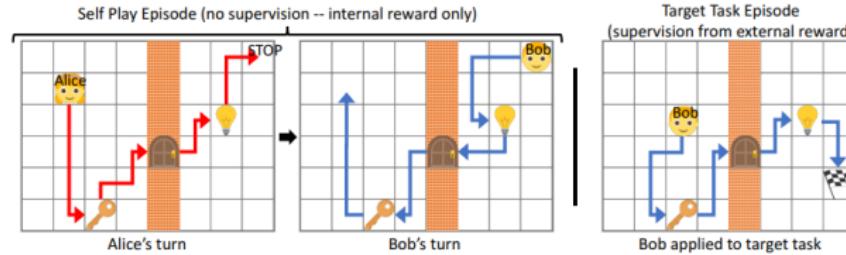
- Ajout de rewards internes r_t^i pour l'exploration
 - Inverse Model vise à apprendre des représentations d'état sans perte d'info
 - Puis comparaison entre représentation du prochain état et sa prédiction par Forward Module
- ⇒ Si l'état d'arrivée est peu prévisible par le modèle : zone peu connue, à explorer plus avant et donc fort reward interne
- ⇒ Prise en compte de la somme des rewards externes (exploitation) et internes (exploration) dans le rollout



Exploration : Automatic Curricula Via Asymmetric Self-Play

Automatic Curricula Via Asymmetric Self-Play [Suk+17]

- Curriculum Learning : Augmentation graduelle de la difficulté des problèmes
 - Application au RL pour les problèmes réversibles ou réinitialisables
 - Un seul agent, deux cerveaux : Alice (l'instructeur) et Bob (l'élève)
 - Objectif de Bob : compléter le problème en un temps minimal $\Rightarrow r_B = -\gamma t_B$, avec t_B le temps mis par Bob pour compléter la tâche
 - Objectif d'Alice : proposer des problèmes adaptés à Bob $\Rightarrow r_A = \max(0, t_B - t_A)$
- \Rightarrow Entraîner Bob à atteindre n'importe quel état à partir de n'importe quel autre



Exploration : Automatic Curricula Via Asymmetric Self-Play

Algorithm 1 Pseudo code for training an agent on a self-play episode

```
function SELFPLAYEPISODE(REVERSE/REPEAT, $t_{\text{MAX}}$ ,  $\theta_A$ ,  $\theta_B$ )
     $t_A \leftarrow 0$ 
     $s_0 \leftarrow \text{env.observe}()$ 
     $s^* \leftarrow s_0$ 
    while True do
        # Alice's turn
         $t_A \leftarrow t_A + 1$ 
         $s \leftarrow \text{env.observe}()$ 
         $a \leftarrow \pi_A(s, s_0) = f(s, s_0, \theta_A)$ 
        if  $a = \text{STOP}$  or  $t_A \geq t_{\text{Max}}$  then
             $s^* \leftarrow s$ 
             $\text{env.reset}()$ 
            break
         $\text{env.act}(a)$ 
         $t_B \leftarrow 0$ 
        while True do
            # Bob's turn
             $s \leftarrow \text{env.observe}()$ 
            if  $s = s^*$  or  $t_A + t_B \geq t_{\text{Max}}$  then
                break
             $t_B \leftarrow t_B + 1$ 
             $a \leftarrow \pi_B(s, s^*) = f(s, s^*, \theta_B)$ 
             $\text{env.act}(a)$ 
         $R_A \leftarrow \gamma \max(0, t_B - t_A)$ 
         $R_B \leftarrow -\gamma t_B$ 
         $\text{policy.update}(R_A, \theta_A)$ 
         $\text{policy.update}(R_B, \theta_B)$ 
    return
```

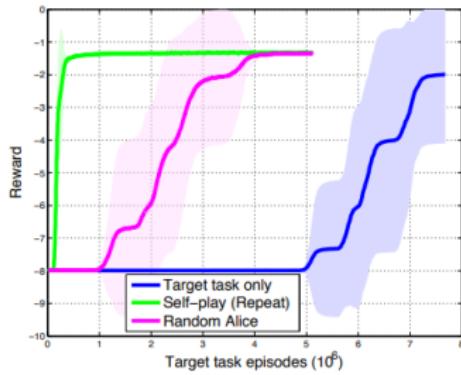
- Si Alice produit une tâche difficile pour elle : t_A proche de t_{Max} et du coup t_B a de grande chances d'être $< t_A$ (car borné par $t_{\text{Max}} - t_A$)
 - ⇒ Alice a intérêt à produire une tâche qu'elle complète le plus rapidement et que Bob ne sait pas résoudre
 - ⇒ Bob apprend progressivement car il a toujours une tâche juste au delà de ses capacités.

Exploration : Automatic Curricula Via Asymmetric Self-Play

⇒ Fine-tuning de Bob sur la tâche cible (son but est alors \emptyset)

Algorithm 2 Pseudo code for training an agent on a target task episode

```
function TARGETTASKEPISODE( $t_{\text{MAX}}$ ,  $\theta_B$ )
     $t \leftarrow 0$ 
     $R \leftarrow 0$ 
    while True do
         $t \leftarrow t + 1$ 
         $s \leftarrow \text{env.observe}()$ 
         $a \leftarrow \pi_B(s, \emptyset) = f(s, \emptyset, \theta_B)$ 
        if env.done() or  $t \geq t_{\text{MAX}}$  then
            break
        env.act( $a$ )
         $R = R + \text{env.reward}()$ 
    policy.update( $R, \theta_B$ )
return
```

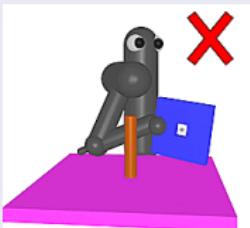


Reverse curriculum generation for reinforcement learning

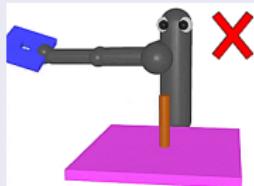
Pour certaines tâches, le reward est très sparse :

- Aucun reward tant qu'on n'a pas atteint le but recherché
- Exemple : tâche de contrôle d'un robot récompensé si il arrive à insérer un anneau sur un bâton

Agent aléatoire a très peu de chances d'arriver à obtenir le moindre feedback si il démarre toujours loin de la cible :



Même si on fait du reward shaping (non expert) pour densifier le reward selon proximité à la cible :



⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Si on est très proche du but, même un agent aléatoire peut réussir la tâche et commencer à apprendre
- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

S^g : goal states we want to reach from everywhere.

s^g : one goal state is provided



- Hypothèse 1 : On connaît au moins 1 des buts finaux du problème $s^g \in S^g$
- Hypothèse 2 : On est capable de faire démarrer l'environnement de n'importe quel état arbitraire
- Hypothèse 3 : À partir de n'importe quel état que l'on pourrait échantillonner comme état de départ, il existe au moins un chemin vers les états finaux

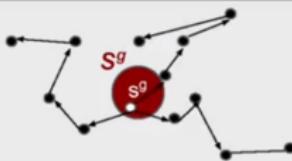
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 1:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- À partir de l'état de départ, marches aléatoires pour trouver des points d'intérêt proches

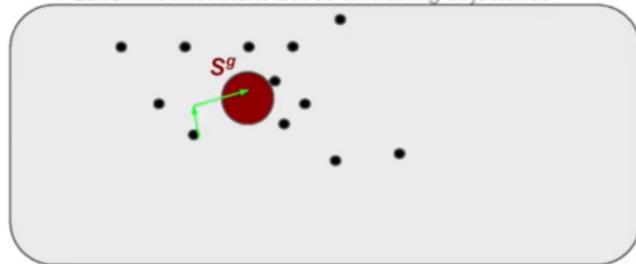
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 1:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- Collecte de trajectoires et Apprentissage RL à partir de tous les points trouvés (exemple TRPO)

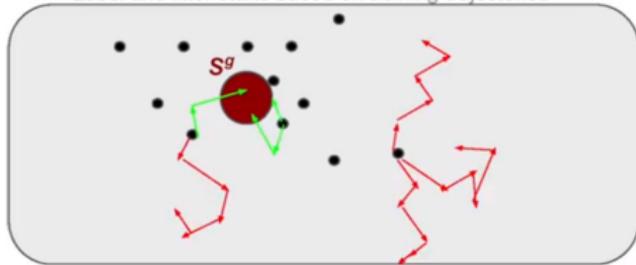
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 1:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- Collecte de trajectoires et Apprentissage RL à partir de tous les points trouvés (exemple TRPO)

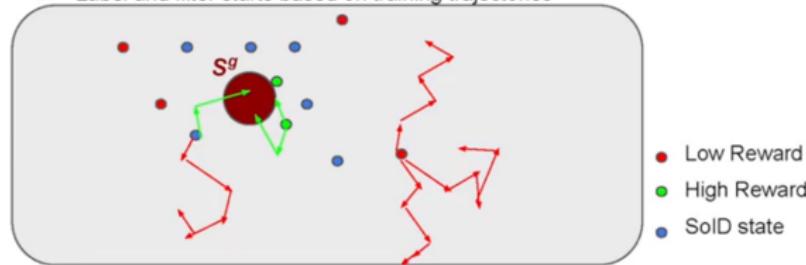
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 1:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- Selection des points d'intérêt (Solid States)
- Un point s_0^i est intéressant comme point de départ si (reward binaire, succès/échec final) :
 - le succès à partir du point est suffisamment incertain (sinon $V(s_0^i) \approx 1$, ce qui donne un gradient PG quasi nul)
 - le succès à partir du point est suffisamment probable (sinon $V(s_0^i) \approx 0$, ce qui donne aussi un gradient PG quasi nul)
- On sélectionne les points dont $V(s_0^i)$ est dans $[R_{min}; R_{max}] = [0.1; 0.9]$
- Estimation de $V(s_0^i)$ selon traces collectées dans la phase d'apprentissage

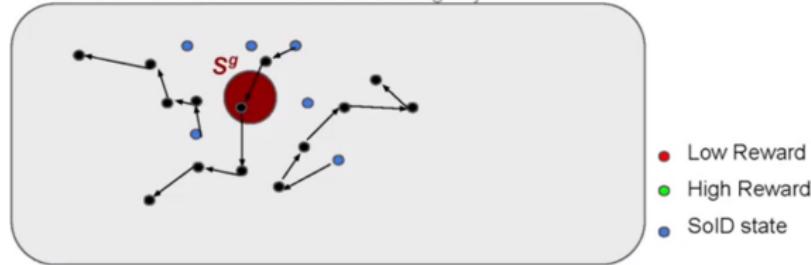
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 2:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- À partir des points retenus, marches aléatoires pour trouver de nouveaux points d'intérêt
- Hypothèse : les points proches des points d'intérêt identifiés sont eux aussi probablement des points d'intérêt

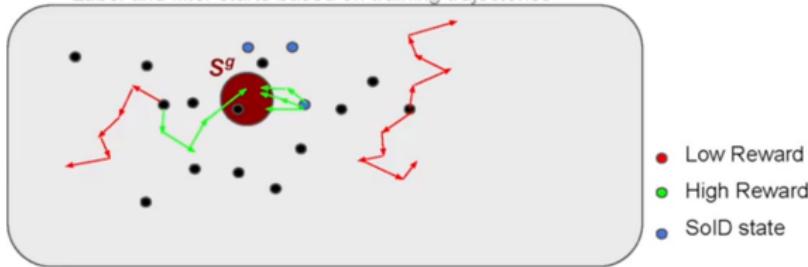
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 2:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- À nouveau apprentissage à partir de l'ensemble des points obtenus (nouveaux et anciens SolID)

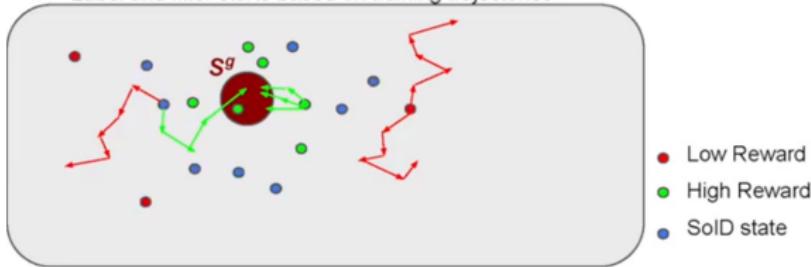
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 2:

- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



- À nouveau apprentissage à partir de l'ensemble des points obtenus (nouveaux et anciens SolID)
- Re-selection, etc...

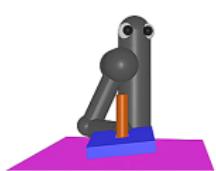
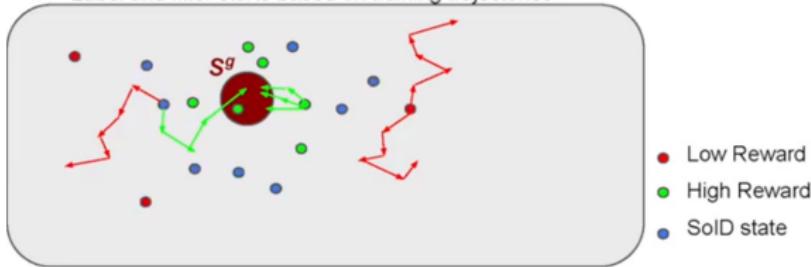
Reverse curriculum generation for reinforcement learning

⇒ Reverse curriculum generation for reinforcement learning [Flo+17]

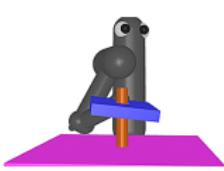
- Idée : faire redémarrer l'agent de plus en plus loin progressivement

Iteration 2:

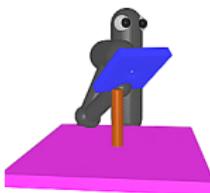
- Run Brownian motion
- Obtain trajectories from collected starts to train policy
- Label and filter starts based on training trajectories



Start



Middle



End

- Multi-goals : on considère une tâche où le but peut changer à chaque épisode
- Exemple : prendre un objet et l'emmener à une position précise
- Universal Value Function : $Q : S \times A \times G \rightarrow \mathbb{R}$ où G est l'ensemble des buts envisageables
- Politique dirigée par un but : $\pi(a|s, g)$
- Tâches complexes lorsque rewards sparses pour algos classiques type DDPG

Exemple de tâche jouet

On cherche un vecteur binaire en effectuant des permutations successives de bits :

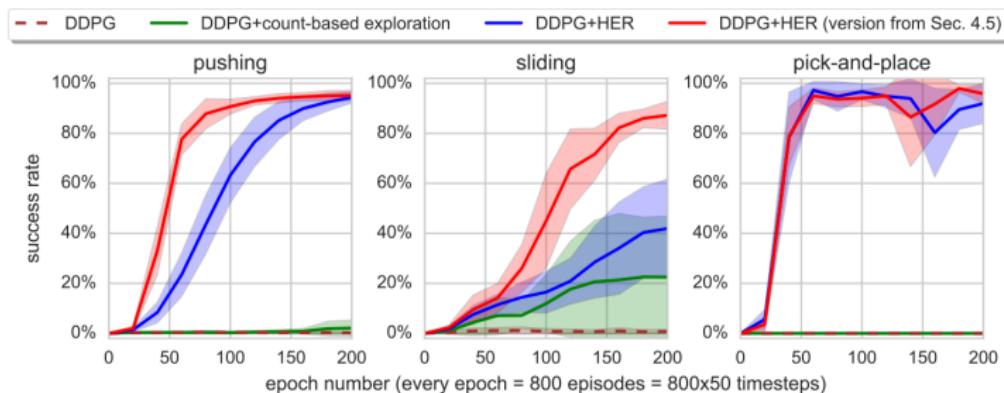
- Espace des états-but très large
- Rewards très peu informatifs
(-1 à chaque step)
- ⇒ [And+17] montre que DQN est incapable de résoudre le problème si taille des vecteurs > 13 (après quoi le taux de succès tombe à 0)

```
Initial State: [0 0 0 0 0 1 1 0 0 1 1 1 1 1]
Goal: [1 0 0 0 0 0 0 1 1 1 1 0 0 0]
State at step 0: [1 0 0 0 0 0 1 1 0 0 1 1 1 1 1]
State at step 1: [1 0 0 0 0 0 1 1 0 1 1 1 1 1 1]
State at step 2: [1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 1]
State at step 3: [1 0 0 0 0 0 1 0 1 1 1 1 0 1 1]
State at step 4: [1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1]
State at step 5: [1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0]
State at step 6: [1 0 0 0 0 0 0 0 1 1 1 1 0 0 0]
Success!
Press enter... □
```

- HER [And+17] tire profit des expériences passées en spécifiant les états rencontrés comme buts des actions réalisées dans l'épisode

HER : Hindsight Experience Replay

- Plutôt que de dire seulement à l'agent que ce qu'il fait est bon au mauvais selon le but spécifié, on l'informe de ce qu'il fait bien :
 - il a réussi à atteindre les états qu'il a rencontrés au cours de l'épisode
 - Dans un épisode, toutes les actions a_i prises avant t ont permis d'atteindre s_t :
 - On peut se servir de s_t pour apprendre $Q(s_i, a_i, s_t)$ et $\pi(a_i | s_i, s_t)$
 - HER : ajout de transitions $(s_t^\tau, a_t^\tau, s_{t+1}^\tau, s_x^\tau)$, avec $x > t$, en plus des transitions $(s_t^\tau, a_t^\tau, s_{t+1}^\tau, g^\tau)$ dans le buffer de replay, avec g^τ le but initial de l'épisode τ (apprentissage DDPG)
- ⇒ On apprend à explorer l'espace efficacement, Rewards plus denses
- ⇒ Agent capable de résoudre des tâches à buts multiples

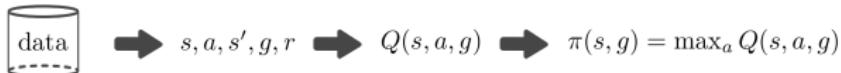


- ⇒ Efficace même pour des problèmes à buts uniques

RIG [Nai+18] exploite le même genre d'idée mais avec des buts imaginés

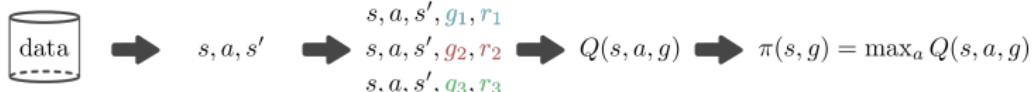
Q-Learning classique :

1. Collect
2. Sample
3. Train
4. Extract



Q-Learning plus efficace, si on avait accès à un moyen de générer des buts et des rewards correspondants à chaque transition :

1. Collect
2. Sample
3. Relabel
4. Train
5. Extract

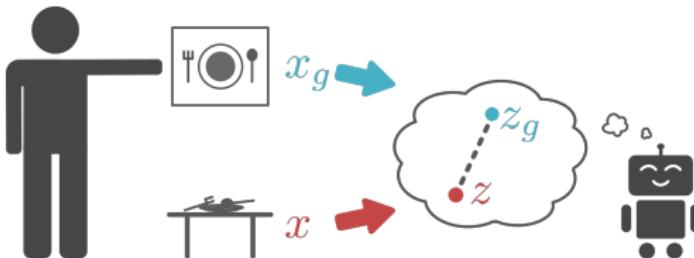


Problèmes (dans le cadre de buts visuels dans RIG) :

- Comment spécifier les sous-buts ?
- Comment attribuer des récompenses correspondantes ?

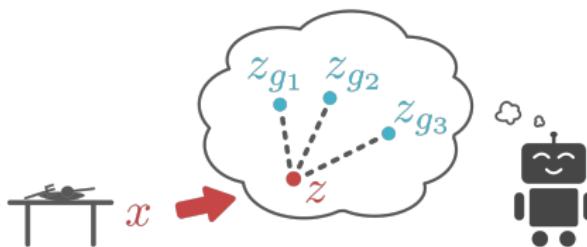
RIG : Visual Reinforcement Learning with Imagined Goals

Encodage des images (correspondant aux différents états) dans un espace latent de plus faible dimension (par exemple par VAE)

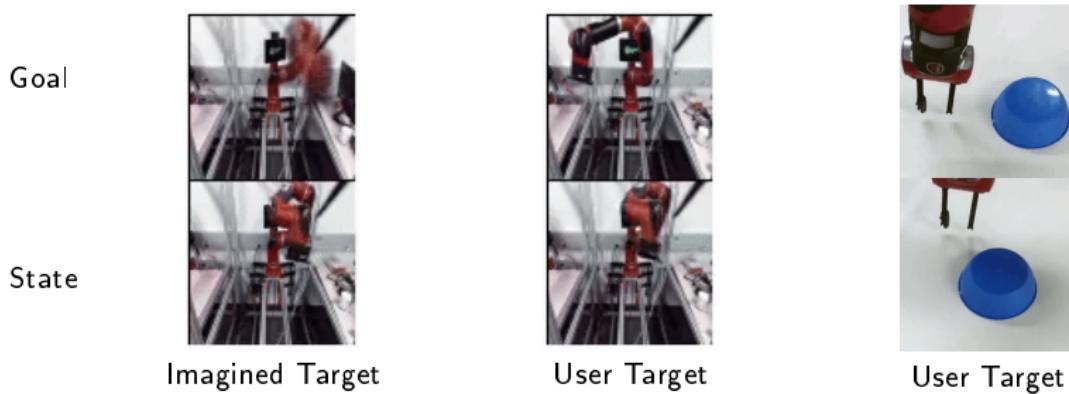
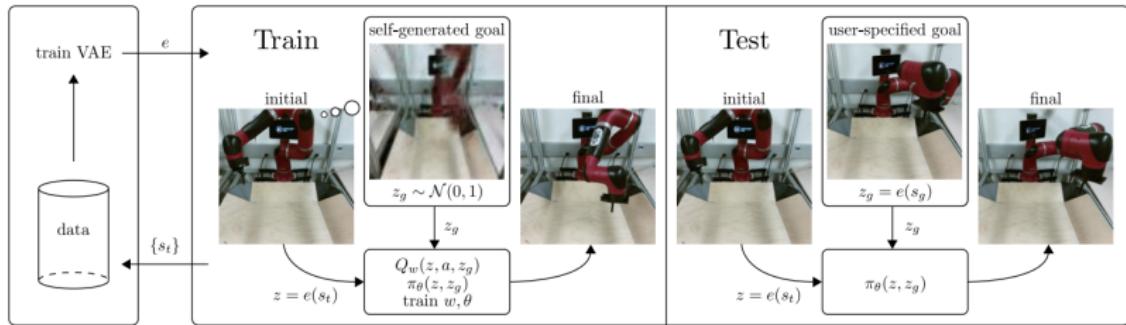


⇒ Reward peut être défini comme une distance au but dans cet espace

Imagination de différents sous-but, selon les distributions dans l'espace latent :



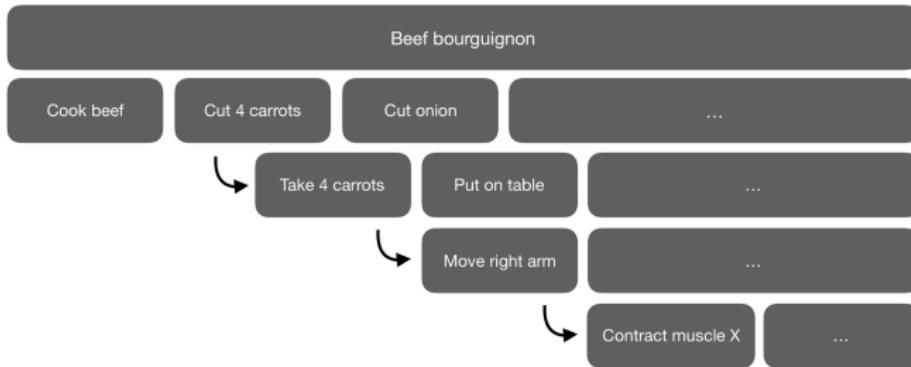
RIG : Visual Reinforcement Learning with Imagined Goals



Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning

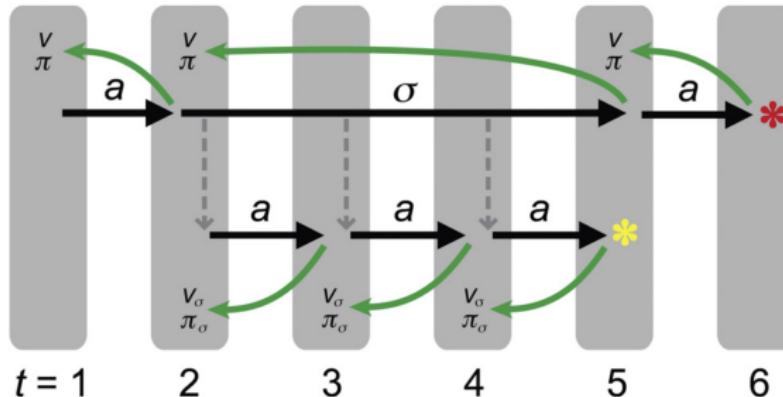
Différents niveaux de granularité d'actions :



- ⇒ Hierarchical RL : représentation explicite de ces différents niveaux
 - ⇒ chaque niveau $l < L$ dépend d'un niveau supérieur $l + 1$ qui correspond à une prise de décision plus large
 - ⇒ Regroupement de séries d'actions plus fines pour accomplir des sous-but plus larges

Hierarchical Reinforcement Learning

Politiques et sous-politiques :

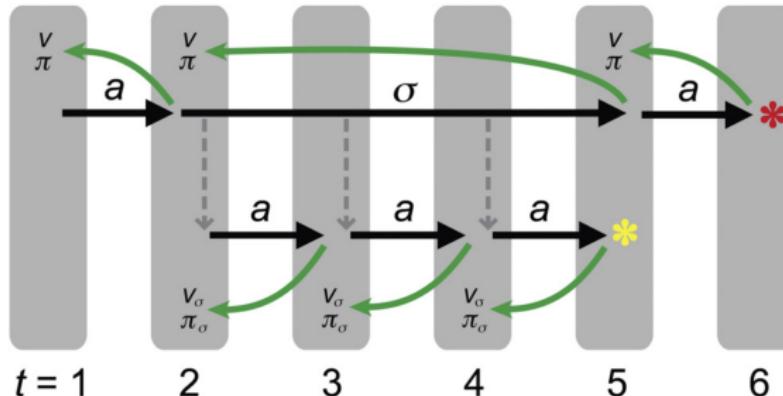


Objectifs :

- Long-term credit assignment : apprentissage plus rapide et meilleure généralisation
- Structured exploration : exploration via sous-politiques plutôt qu'avec actions primitives
- Transfer learning : tâches différentes => planifications différentes de sous-tâches (mais sous-tâches communes)

Hierarchical Reinforcement Learning

Politiques et sous-politiques :



Formulation selon Semi-Markov Decision Process (SMDP) :

- Différentes résolutions temporelles entre niveaux : Les niveaux supérieurs attendent la terminaison des procédures des niveaux inférieurs
- On n'a alors pas une formulation MDP classique avec transitions définies selon $P(s_{t+1}|s_t, a_t)$
- SMDP pour HRL : au niveau I , $P(s'_I, N_I|s_I, a_I)$, avec N_I le nombre d'actions primitives prises par le niveau inférieur depuis la décision a_I au niveau I

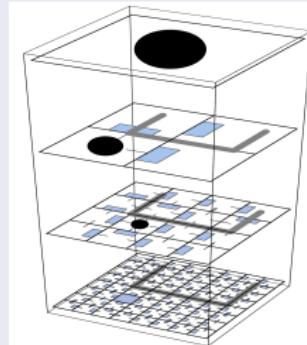
Feudal HRL

Le type d'approche HRL "féodal" [DH93] crée un système hiérarchique dans lequel

- Les lords (ou managers) apprennent à assigner des tâches (ou sous-buts) à leurs serfs (ou sous-managers)
- Chaque niveau ne se soucie que de ses propres buts, pas de la manière dont les niveaux inférieurs répondent à la demande, ni des éventuelles mauvaises commandes passées par les niveaux supérieurs
- **Multi-résolution** : résolution différente de l'information à chaque niveau

Exemple sur un problème de labyrinthe

- Objectif : atteindre la case bleutée (3,3) dans le labyrinthe complet tout en bas
- Niveau inférieur : autant d'états que de positions
- + on monte dans les niveaux, + on regroupe les états
- Politiques supérieures décident si elles doivent rester dans une zone ou chercher à en sortir
- Q-values à convergence des actions :
 - de déplacement \Rightarrow carrés bleus
 - de rester sur zone \Rightarrow sphères noires



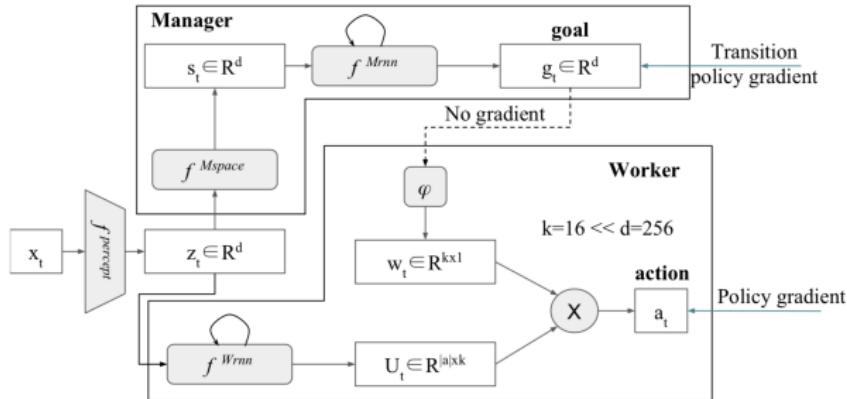
Malheureusement :

- Requiert la capacité de découper l'espace d'états et spécifier les actions à chaque niveau efficacement (expertise du domaine)
- Perte de temps si les niveaux supérieurs passent des commandes abhérentes : apprentissage de sous-politiques inutiles

FUN : FeUdal Networks for HRL

FUN [Vez+17] relaxe l'aspect multi-resolution des états des approches féodales, mais en garde le passage de commande explicite :

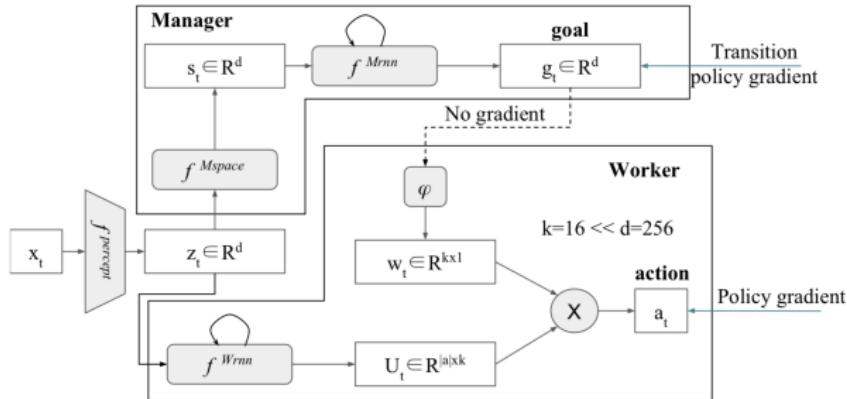
- Master génère des buts continus g_t pour le Worker sous la forme de directions dans l'espace latent s
 - Plus facile de demander au Worker de faire en sorte de se déplacer dans une certaine direction plutôt que d'atteindre une position absolue de l'espace latent
 - Meilleure généralisation : un même angle peut être invoqué selon différents contextes
- ⇒ Demande au Worker de faire en sorte que la représentation latente de l'état au temps $t + c$ soit dans la direction g_t en partant de s_t , avec c la résolution temporelle du manager (g_t est un but à accomplir dans c steps)



FUN : FeUdal Networks for HRL

FUN [Vez+17] relaxe l'aspect multi-resolution des états des approches féodales, mais en garde le passage de commande explicite :

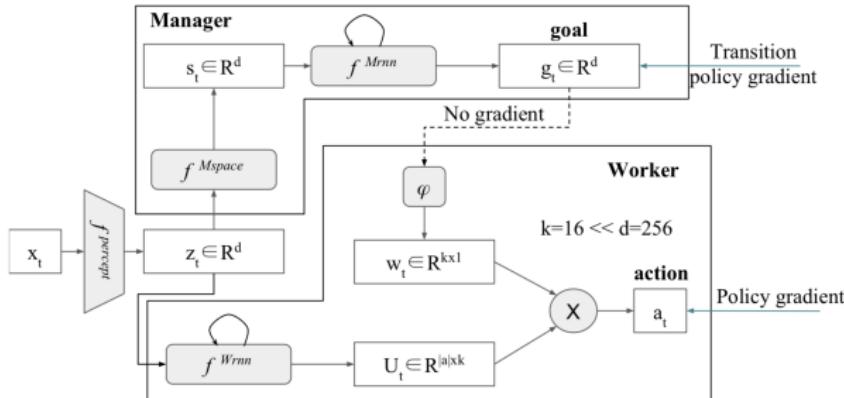
- Worker tente de maximiser $R_t + \alpha R_t^I$, avec $R_t^I = \frac{1}{c} \sum_{i=1}^c d_{cos}(s_t - s_{t-i}, g_{t-i})$
 - Worker récompensé si il prend des actions menant à des déplacements de même direction que les c dernières directions données par le manager
 - Forme de régularité dans les déplacements : plutôt que de regarder uniquement $d_{cos}(s_t - s_{t-c}, g_{t-c})$
- Worker mis à jour selon policy gradient classique avec rewards $R_t + \alpha R_t^I$



FUN : FeUdal Networks for HRL

FUN [Vez+17] relaxe l'aspect multi-resolution des états des approches féodales, mais en garde le passage de commande explicite :

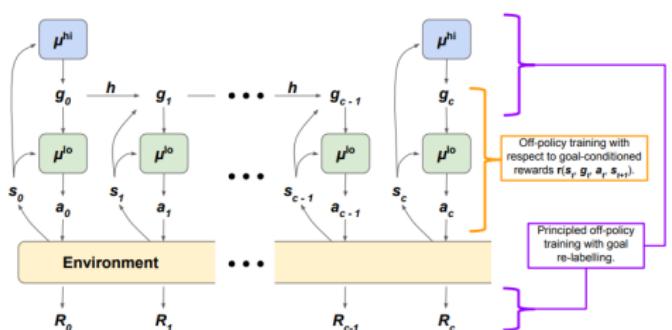
- Pas de gradient entre Worker et Master car cela retirerait tout sens sémantique aux buts du Manager (représentations internes vides de sens)
 - Manager plutôt mis à jour selon PG sur des transitions temporelles de taille c : $\mathbb{E}[A_t^M \nabla_\theta \log p(s_{t+c} | s_t, u_t)]$, avec $u_t = \mu_\theta(s_t)$ et $A_t^M = R_t - V^M(x_t, \phi)$.
 - FUN suppose que les directions dans l'espace latent suivent une distribution von Mises-Fisher de moyenne $g_t = g(u_t) : p(s_{t+c} | s_t, u_t) \propto e^{d_{cos}(s_{t+c} - s_t, g_t)}$
- ⇒ On prend alors : $\nabla g_t = A_t^M \nabla_\theta d_{cos}(s_{t+c} - s_t, g_t)$, où l'on ne considère pas la dérivée selon s pour éviter les solutions triviales.



HIRO : Data-Efficient Hierarchical Reinforcement Learning

Dans le même esprit que FUN, HIRO [Nac+18]. Mais :

- Goals = transitions dans l'espace des états
- Off-Policy PG du niveau inférieur selon transitions effectuées et commandées
- Off-Policy PG avec goal relabelling au niveau supérieur
 - ⇒ Prise en compte des possibles erreurs du niveau inférieur en re-spécifiant le but pour le faire correspondre à ce qui a été effectué et les rewards associés pour PG du niveau supérieur
- Fonction de transition entre buts h



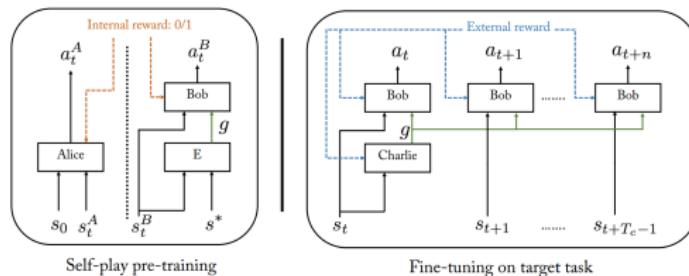
1. Collect experience $s_t, g_t, a_t, R_t, \dots$
2. Train μ^{lo} with experience transitions $(s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1})$ using g_t as additional state observation and reward given by goal-conditioned function $r_t = r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||_2$.
3. Train μ^{hi} on temporally-extended experience $(s_t, \tilde{g}_t, \sum R_{t:t+c-1}, s_{t+c})$, where \tilde{g}_t is relabelled high-level action to maximize probability of past low-level actions $a_{t:t+c-1}$.
4. Repeat.

Figure 2: The design and basic training of HIRO. The lower-level policy interacts directly with the environment. The higher-level policy instructs the lower-level policy via high-level actions, or goals, $g_t \in \mathbb{R}^{d_s}$ which it samples anew every c steps. On intermediate steps, a fixed goal transition function h determines the next step's goal. The goal simply instructs the lower-level policy to reach specific states, which allows the lower-level policy to easily learn from prior off-policy experience.

Learning Goal Embeddings via Self-Play for HRL

Asymetric self-play pour HRL [Suk+18] :

- Politiques haut-niveau dirigent les politiques bas-niveau via des vecteurs continus de sous-but à atteindre
- Challenges pour apprendre un espace de représentation des sous-but :
 - L'espace doit être suffisamment général pour encoder diverses sortes de sous-but
 - L'espace ne doit encoder que les informations utiles pour la tâche
 - Les sous-but générés doivent être adaptés aux capacités des sous-politiques



The Hierarchical Self-Play model in pre-training (left) and fine-tuning (right) phases.

Pendant le fine-tuning (et l'exécution) sur la tâche :

- La politique du contrôleur Charlie consiste en un mapping d'un but g_t à partir de s_t : $g_t = \pi(s_t)$. Charlie émet un nouveau but tous les $T_C = T_A$ steps
- Charlie (ainsi que fine-tuning de Bob) est appris selon les rewards externes

Learning Goal Embeddings via Self-Play for HRL

Approche de pré-entraînement similaire à [Suk+17] mais :

- Alice réalise un nombre d'étapes fixées T_A (T_A pas dynamique pour limiter la complexité), que Bob doit accomplir en moins de T_B étapes (avec T_B légèrement $> T_A$)
- Reward d'Alice $R_A = 1 - R_B$, avec R_B le reward de Bob, qui vaut 1 si il arrive à approcher l'état cible d'une distance $< \epsilon$, 0 sinon (plutôt que rewards basés sur temps)
- Alice et Bob repartent avec s_0 inclus dans l'episode d'Alice précédent pour éviter de ne se concentrer qu'autour d'un même s_0
- Bob appris selon l'accomplissement de la tâche et un coût d'imitation d'Alice. Pour une trajectoire τ_A de fin $s_{\tau_A}^*$ et de point de départ s_0 :

$$L_B = -\mathbb{E}_{\tau_B \sim \pi_B(\cdot | \cdot, s_{\tau_A}^*)}[R_B | s_0] - \alpha \sum_{t=0}^{|\tau_A|} \log \pi_B(a_t^A | s_t^A, s_{\tau_A}^*)$$

- Alice apprise selon un coût d'entropie H pour garantir une variété de tâches :

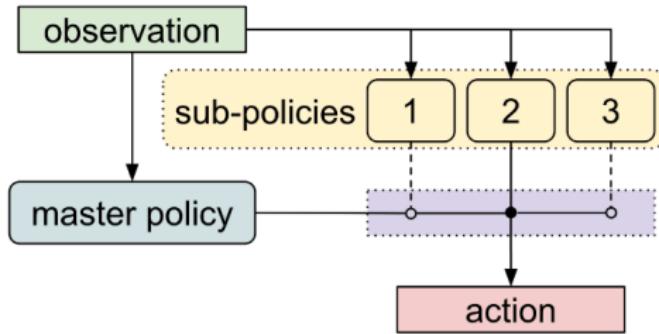
$$L_A = -\mathbb{E}_{\tau_A \sim \pi_A(\cdot, s_0)} [\mathbb{E}_{\tau_B \sim \pi_B(\cdot | \cdot, s_{\tau_A}^*)}[1 - R_A | s_0] + \beta \sum_{t=0}^{|\tau_A|} H(\pi_A(\cdot | s_t^A, s_0))]$$

- Bob utilise un encodeur de buts E : $\pi_B(\cdot | s_t, s^*) = \pi_B(\cdot | s_t, E(s_t, s^*))$. E permet de rendre plus compact l'expression du but et rend la génération de buts plus simple pour Charlie (phase d'exécution)

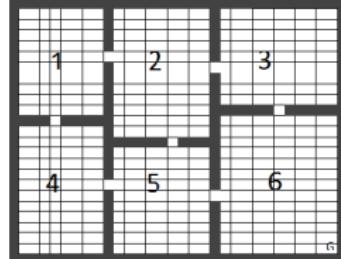
Hierarchical Reinforcement Learning : Options

Autre framework HRL : Options [SPS99]

- Option=sous-politique $\langle \pi_\omega, \beta_\omega \rangle$, avec :
 - $\pi_\omega : S \times A \rightarrow [0; 1]$ la politique de l'option ω
 - $\beta_\omega : S \rightarrow [0; 1]$ son critère de terminaison
- Plutôt que de spécifier des buts aux niveaux inférieurs, on sélectionne une sous-politique lorsque la précédente a terminé



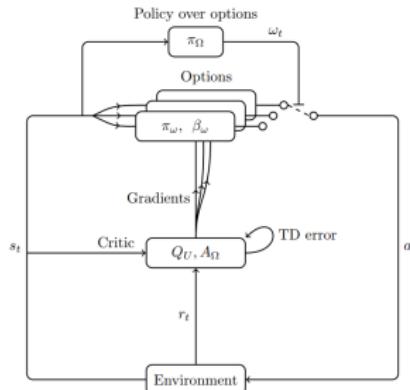
Exemple :
1 option par pièce dans
un Gridworld



Hierarchical Reinforcement Learning : Options

L'architecture Option-Critic [BHP16] est composée :

- D'un manager qui choisit une sous-politique lorsque nécessaire (à la fin de la précédente) selon $\pi_\Omega(\cdot|s)$
- D'un ensemble d'options $\langle \pi_\omega, \beta_\omega \rangle$
- De réseaux de critiques Q_U et Q_Ω



$$Q_U(s, \omega, a) = \mathbb{E}_{s' \in S} p(s'|s, a)(R(s, a, s') + \gamma U(\omega, s'))$$

avec : $U(\omega, s) = (1 - \beta_\omega(s))Q_\Omega(s, \omega)$

$$+ \beta_\omega(s) \sum_{\omega \in \Omega} \pi_\Omega(\omega|s)Q_\Omega(s, \omega)$$

et : $Q_\Omega(s, \omega) = \sum_{a \in A} \pi_\omega(a|s)Q_U(s, \omega, a)$

$$A_\Omega(s, \omega) = Q_\Omega(s, \omega) - \sum_{\omega \in \Omega} \pi_\Omega(\omega|s)Q_\Omega(s, \omega)$$

Algorithm 1: Option-critic with tabular intra-option Q-learning

$s \leftarrow s_0$
Choose ω according to an ϵ -soft policy over options
 $\pi_\Omega(s)$

repeat
Choose a according to $\pi_{\omega, \theta}(a | s)$
Take action a in s , observe s', r

1. Options evaluation:

$\delta \leftarrow r - Q_U(s, \omega, a)$
if s' is non-terminal **then**
 $\delta \leftarrow \delta + \gamma(1 - \beta_{\omega, \vartheta}(s'))Q_\Omega(s', \omega) + \gamma\beta_{\omega, \vartheta}(s') \max_{\bar{\omega}} Q_\Omega(s', \bar{\omega})$
end

$Q_U(s, \omega, a) \leftarrow Q_U(s, \omega, a) + \alpha\delta$

2. Options improvement:

$\theta \leftarrow \theta + \alpha_\theta \frac{\partial \log \pi_{\omega, \theta}(a | s)}{\partial \theta} Q_U(s, \omega, a)$
 $\vartheta \leftarrow \vartheta - \alpha_\vartheta \frac{\partial \beta_{\omega, \vartheta}(s')}{\partial \vartheta} (Q_\Omega(s', \omega) - V_\Omega(s'))$

if $\beta_{\omega, \vartheta}$ terminates in s' **then**
choose new ω according to ϵ -soft($\pi_\Omega(s')$)
 $s \leftarrow s'$

until s' is terminal

Hierarchical Reinforcement Learning : Options

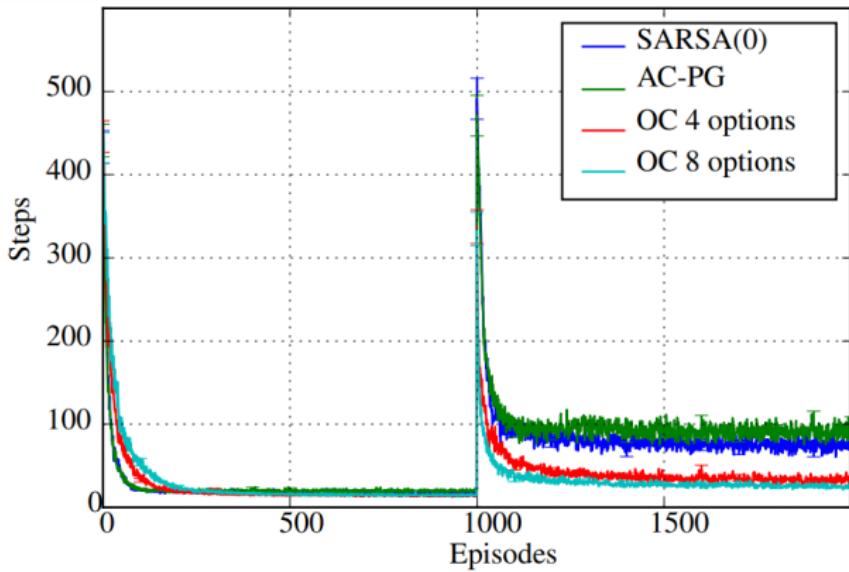


Figure : After a 1000 episodes, the goal location in the four-rooms domain is moved randomly. Option-critic (“OC”) recovers faster than the primitive actor-critic (“AC-PG”) and SARSA(0). Each line is averaged over 350 runs.

Meta Learning of Shared Hierarchies

MLSH [Fra+17] : RL multi-tâches avec options

- un meta-controlleur apprend à choisir les sous-politiques selon la tâche en cours (période de warm-up)
- des sous-politiques que l'on adapte à chaque nouvelle tâche (joint update)

Algorithm 1 Meta Learning Shared Hierarchies

```
Initialize  $\phi$ 
repeat
    Initialize  $\theta$ 
    Sample task  $M \sim P_M$ 
    for  $w = 0, 1, \dots, W$  (warmup period) do
        Collect  $D$  timesteps of experience using  $\pi_{\phi, \theta}$ 
        Update  $\theta$  to maximize expected return from  $1/N$  timescale viewpoint
    end for
    for  $u = 0, 1, \dots, U$  (joint update period) do
        Collect  $D$  timesteps of experience using  $\pi_{\phi, \theta}$ 
        Update  $\theta$  to maximize expected return from  $1/N$  timescale viewpoint
        Update  $\phi$  to maximize expected return from full timescale viewpoint
    end for
until convergence
```

Avec ϕ un ensemble de politiques (phi_1, \dots, phi_k)

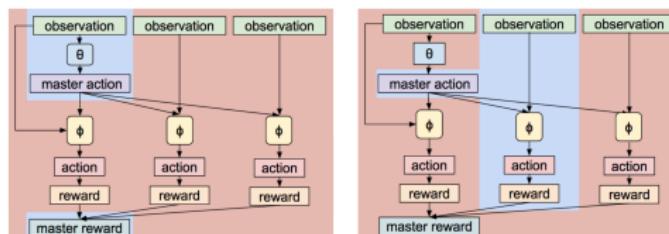


Figure 2: Unrolled structure for a master policy action lasting $N = 3$ timesteps. Left: When training the master policy, the update only depends on the master policy's action and total reward (blue region), treating the individual actions and rewards as part of the environment transition (red region). Right: When training sub-policies, the update considers the master policy's action as part of the observation (blue region), ignoring actions in other timesteps (red region)

Modulated Policy Hierarchies

MPH [Pas+18] : hiérarchie de contrôleurs qui prennent en entrée l'état du système et les vecteurs binaires des contrôleurs supérieurs

- Optimisation selon PPO
- Utilisation d'un reward intrinsèque de curiosité [Pat+17]
- ⇒ Objectifs : Stabilité, Modularité, Exploration

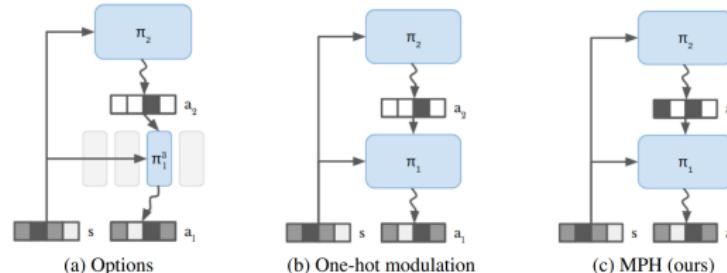
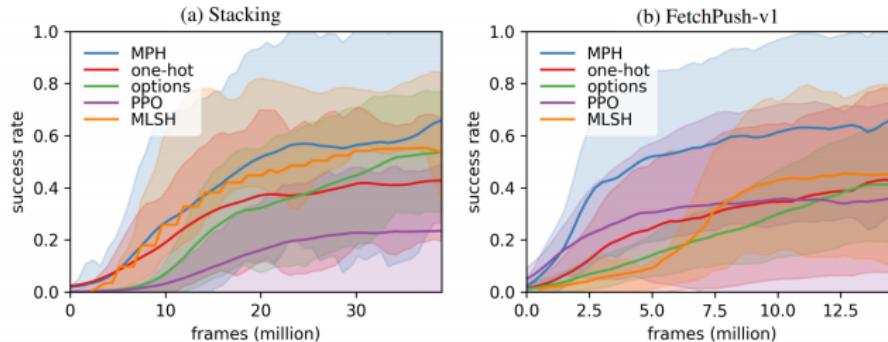


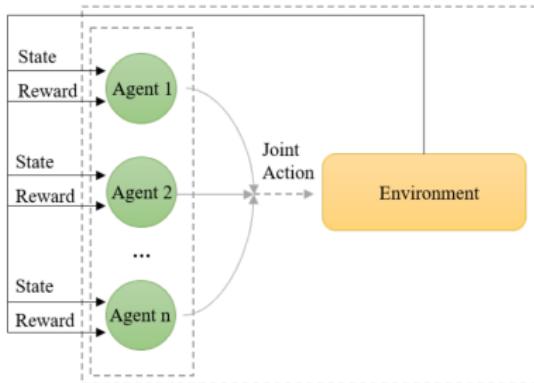
Figure 2: Overview of hierarchical policies. (a) The options agent selects between separate skill networks using a categorical master policy. (b) The one-hot agent combines the skills into a single network and is modulated by a 1-hot signal. (c) Our modulated policy hierarchy sends a binary vector, allowing for richer communication and mixing of skills.



Multi-Agents Reinforcement Learning

Multi-Agents Reinforcement Learning

Problèmes de décision séquentiels où plusieurs agents interagissent simultanément avec le même environnement :



- Ensemble d'actions jointes pour N agents : $A = (A_1, A_2, \dots, A_N)$
- Transitions $P : S \times A \times S \rightarrow [0, 1]$ définies selon l'ensemble d'actions A
- Rewards $R : S \times A \times S \rightarrow \mathbb{R}^N$ selon actions communes
- $V^\pi : S \rightarrow \mathbb{R}^N$ défini selon l'ensemble des politiques $\pi = (\pi_1, \pi_2, \dots, \pi_N)$
- $Q^\pi : S \times A \rightarrow \mathbb{R}^N$ défini selon l'ensemble des politiques $\pi = (\pi_1, \pi_2, \dots, \pi_N)$ et des actions A
- En général, observations partielles pour chaque agent : $O^t = (O_1^t, \dots, O_N^t)$

Multi-Agents Reinforcement Learning

En théorie des jeux, un équilibre de Nash est une situation où :

- Chaque joueur prévoit correctement le choix des autres
- Chaque joueur maximise son gain, compte tenu de cette prévision.

Dilemme du prisonnier (D=dénonciation du complice, C=ne rien faire) :

- Si seul A (resp. B) dénonce, A (resp. B) est libre, B (resp. A) écoppe de 10 ans
- Si les deux dénoncent, les deux écoppent de 5 ans
- Si aucun ne dénonce, les deux écoppent de 6 mois

		D		C
D	😊😊	😊😊	😊😊	😊😊
C	😊😊	😊😊	😊😊	😊😊

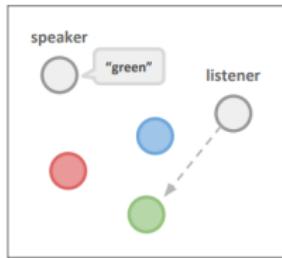
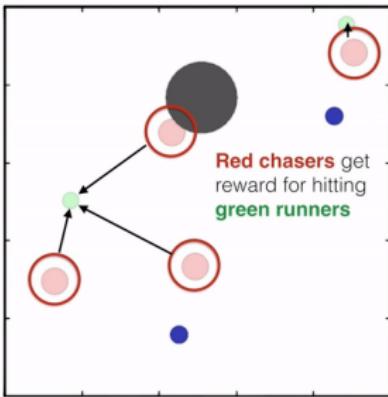
Équilibre de Nash si aucun des joueurs ne regrette son choix/n'aurait pas pu faire mieux compte tenu de ce qu'ont fait les autres (case D,D)

Appliqué au RL Multi-Agents, équilibre de Nash en π_*^1, \dots, π_*^N si pour tout i , pour tout $\pi^i \in \Pi^i$, on a pour chaque s [Kön03] :

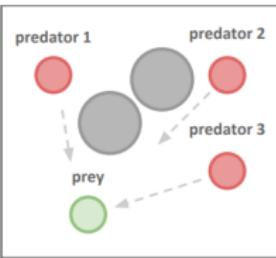
$$V_{\pi_*^1, \dots, \pi^i, \dots, \pi_*^N}^i(s) \leq V_{\pi_*^1, \dots, \pi_*^N}^i(s)$$

avec Π^i l'ensemble des politiques possibles pour l'agent i .

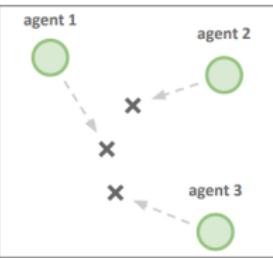
Multi-Agents Cooperative/Competitive tasks



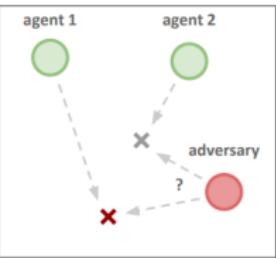
Cooperative Communication



Predator-Prey



Cooperative Navigation



Physical Deception

Challenges :

- Non-stationnarité des transitions
 - ⇒ dépendant des actions des autres agents :
 $P(s'|s, a_i, \pi_1, \dots, \pi_N) \neq P(s'|s, a_i, \pi'_1, \dots, \pi'_N)$
- Design du but / des rewards
- Hétérogénéité des agents :
 - différentes capacités
 - différents rôles
 - différents buts
- Scalabilité : nombre de paramètres peut exploser si nombreux agents avec paramètres individuels
- Problèmes de communication
- Observations partielles
- Agents asynchrones

Multi-Agents Reinforcement Learning

Première Approche : Exécution centralisée

- Agrégation de l'ensemble des informations (observations) récoltées par les différents agents
- Définition d'un reward global ou somme des rewards individuels
- Un seul contrôleur dont la politique retourne un ensemble d'actions à chaque pas de temps $\pi(a_t|o_t)$, avec $a_t = (a_t^1, \dots, a_t^N)$ et $o_t = (o_t^1, \dots, o_t^N)$ l'ensemble des actions et des observations au temps t .

⇒ Revient à un POMDP classique mais...

 Croissance exponentielle des espaces d'actions et d'observations avec le nombre d'agents (problème combinatoire)

 Limité à des problèmes où l'on peut contrôler conjointement l'ensemble des agents

- Confrontation d'un sous ensemble d'agents avec d'autres que l'on ne contrôle pas impossible
- Suppose une communication parfaite entre les agents et le contrôleur
- Ajout de nouveaux agents difficile
- Définition reward global pas toujours évident (pas adapté aux problèmes adverses par exemple)

 Dans les problèmes collaboratifs, tend à converger vers certains agents qui prennent le contrôle et d'autres qui deviennent "fainéants" [Sun+17]

Multi-Agents Reinforcement Learning

Première Approche : Exécution centralisée

- Possibilité : Factorisation des actions $\pi(a_t|o_t) = \prod_{i=1}^N \pi_i(a_t^i|o_t)$
 - ⇒ Perte de dépendances entre actions des différents agents
 - ⇒ Apprentissage de $\pi_i(a_t^i|o_t)$ pas forcément facilité car doit prendre en compte les distributions sur les autres agents mais...
- thumb up Reduit la complexité de la politique qui ne doit plus mapper qu'entre l'espace des observations jointes et l'espace des actions individuelles :
 - Complexité de π selon les actions passe de $|\mathcal{A}|^N$ à $N|\mathcal{A}|$, si \mathcal{A} est l'ensemble d'actions possible pour un agent
- hand pointing down Croissance de l'espace des observations toujours exponentielle en fonction de N
- hand pointing down Toujours limité à des problèmes où l'on a accès à l'ensemble des observations
- hand pointing down Toujours problématique pour les environnements collaboratifs / concurrentiels
- ⇒ On préfère généralement des approches à exécution décentralisée : chaque agent (ou groupe d'agents) est contrôlé individuellement

Multi-Agents Reinforcement Learning

Approche concurrente :

- Chaque agent est contrôlé individuellement selon ses propres observations, avec chacun une politique différente $\pi_i(a_t^i|o_t^i)$
- Apprentissage simultané ou alterné des différents agents indépendants [Tam+17]
- Dans le cadre des PG, pour chaque agent i , on peut considérer (avec $\pi(\tau)$ la probabilité de τ selon l'environnement et tous les agents) :

$$\theta_i^{t+1} = \theta_i + \mathbb{E}_{\tau \sim \pi(\tau)} \sum_{t=1}^{|\tau|-1} \nabla_{\theta_i} \log \pi_{\theta_i}(a_t^i | o_t^i) \sum_{t'=t}^{|\tau|-1} \gamma^{t'-t} r_t^i$$

- Ou dans le cas d'un reward global avec r_t à la place de r_t^i (+ de garanties de convergence, mais pas toujours plus rapide en pratique [BN06] et pas toujours possible)
 - 👍 Permet le contrôle d'agents hétérogènes mais...
 - 👎 Pas de partage d'expérience entre agents
 - 👎 Dynamiques de l'environnement non stationnaires (si on considère les autres agents comme appartenant à l'environnement)
 - 👎 Instabilité, Expérience Replay impossible

Multi-Agents Reinforcement Learning

Approche décentralisée avec Partage de paramètres :

- Chaque agent est contrôlé selon ses propres observations
- Mais partage de paramètre entre agents : agent m contrôlé selon $\pi_\theta(a_t^m | o_t^m, m)$
- Exemple : PS-TRPO [GEK17]

Algorithm 1 PS-TRPO

Input: Initial policy parameters Θ_0 , trust region size Δ

for $i \leftarrow 0, 1, \dots$ **do**

 Rollout trajectories for all agents $\vec{\tau} \sim \pi_{\theta_i}$

 Compute advantage values $A_{\pi_{\theta_i}}(o^m, m, a^m)$ for each agent m 's trajectory element.

 Find $\pi_{\theta_{i+1}}$ maximizing Eq. (1)

 subject to $\overline{D}_{KL}(\pi_{\theta_i} \| \pi_{\theta_{i+1}}) \leq \Delta$

Avec Eq.(1) : $\mathcal{L}(\pi_{\theta_{i+1}}) = \sum_{m=1}^N \mathbb{E}_{(o,a) \sim p_{\theta_i}^m(o,a)} \left[\frac{\pi_{\theta_{i+1}}(a|o,m)}{\pi_{\theta_i}(a|o,m)} A_{\theta_i}(o, m, a) \right],$

où $(o, a) \sim p_{\theta_i}^m(o, a)$ correspond à un échantillonnage d'un couple observation-action pour l'agent m dans l'ensemble des trajectoires obtenues à l'étape $i - 1$.

Partage d'expérience, meilleure stabilité |

Différents comportements peuvent être appris car différentes observations et index m donné en entrée du réseau |

Centralized training with decentralized execution

Dans divers cas, il est possible d'accéder à plus d'informations pendant l'entraînement que pendant l'exécution :

- Execution décentralisée (basée sur observations locales) mais apprentissage centralisé
- Par exemple entraînement sur simulateur permet de connaître l'état courant (voire les dynamiques du monde), ou Oracle en situation réelle (caméra qui observe le monde par exemple)
- ... Mais les agents n'ont pas directement accès à ces informations supplémentaires

Si on connaît les dynamiques du monde, il est possible de définir dans le cadre des Dec-POMDPs [KB16] :

$$Q_i^*(h_i, a_i) = \sum_{s, a_{-i}, h_{-i}} \pi(a_{-i} | h_{-i}) P_i(s, h_{-i} | h_i) \left[R(s, \vec{a}) + \sum_{\omega_i \in \Omega_i} P_i(\omega_i | s, \vec{a}) \max_{b \in \mathcal{A}_i} Q_i^*((h_i, a_i, \omega_i), b) \right]$$

avec ω_i une observation pour l'agent i , $h_i = (\omega_i^0, a_i^1, \omega_i^1, \dots, a_i^t, \omega_i^t)$ l'historique des actions-observations de l'agent i au temps t , a_{-i} l'ensemble des observations sauf celle de l'agent i .

- $\pi(a_{-i} | h_{-i})$ retourne 1 seulement si toutes les actions sauf celle de i correspondent à celles choisies par les agents selon leurs historiques respectifs, i.e. $\pi(a_{-i} | h_{-i}) = 1$ si $\forall j \neq i, a_j = \text{argmax}_{a' \in \mathcal{A}_j} Q_j^*(h_j, a')$.
- $P_i(\omega_i | s, \vec{a}) = \sum_{s'} P(s' | s, \vec{a}) O_i(\omega_i | s', \vec{a})$, avec $O_i(\omega_i | s', \vec{a})$ la probabilité d'observer ω_i dans l'état s' si les actions précédentes étaient \vec{a} .

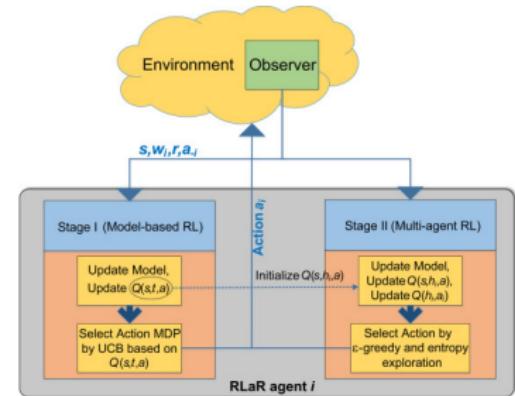
Centralized training with decentralized execution

Objectif à maximiser : $\sum_{a_i \in \mathcal{A}_i} \pi(a_i, \omega_i^0) Q_i^*(h_i^0, a_i)$

- Malheureusement, planification stochastique impossible dans la plupart des cas (problème au moins NEXP-complet).
⇒ RL même lorsque les dynamiques du monde sont connues

RLaR [KB16] :

- Model-based MARL : Chaque agent apprend un modèle $\hat{P}(s'|s, \vec{a})$, $\hat{R}(s, \vec{a})$, $\hat{O}_i(\omega_i|s', \vec{a})$, $\hat{P}_i(a_i|s, h_i)$, $\hat{P}_i(s|h_i)$
- Deux phases d'apprentissage :
 - 1 $Q(s, t, \vec{a}) = \hat{R}(s, \vec{a}) + \sum_{s' \in S} \hat{P}(s'|s, \vec{a}) \max_{\vec{a}' \in A} Q(s', t+1, \vec{a}')$
 - 2 $Q_i(h_i, a_i) = \sum_{s, a_{-i}} \hat{P}_i(a_i|s, h_i) \hat{P}_i(s|h_i) \tilde{Q}_i(s, h_i, \vec{a})$, avec
 $\tilde{Q}_i(s, h_i, \vec{a}) = \hat{R}(s, \vec{a}) + \sum_{\omega_i \in \Omega_i} \hat{P}_i(\omega_i|s, \vec{a}) \max_{a'_i \in \mathcal{A}_i} Q_i^*((h_i, a_i, \omega_i), a'_i)$
- Phase 1 sert à apprendre un modèle global du monde (et des politiques) en bénéficiant de toutes les informations à disposition
- Phase 2 permet d'adapter les agents aux informations partielles dont ils disposeront lors de l'exécution
- Après phase 1, $\tilde{Q}_i(s, h_i, \vec{a})$ initialisé avec $Q(s, t, \vec{a})$



VDN : Value-Decomposition Networks For Cooperative Multi-Agent Learning

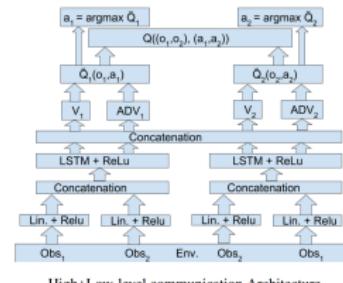
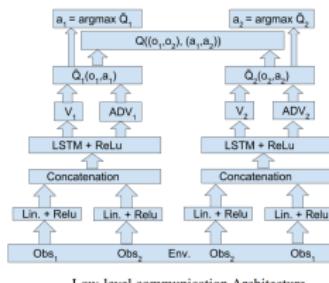
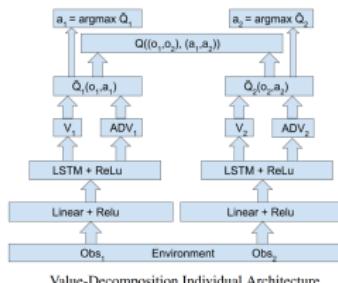
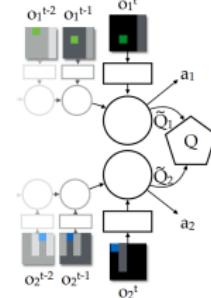
Mais modèle très complexe... Plus simplement, VDN [Sun+17] propose une approche à exécution décentralisée avec apprentissage centralisé basée sur la décomposition de fonctions de Valeur Q en valeurs individuelles, en supposant :

$$Q((h^1, h^2, \dots, h^N), (a^1, a^2, \dots, a^N)) \approx \sum_{i=1}^N \tilde{Q}_i(h^i, a^i)$$

avec h_i l'historique d'observations de l'agent i .

VDN se base sur DQN, en considérant un réseau par agent, dont les sorties sont sommées et comparées à $r_t + \gamma \max_{\vec{a}} Q(h_{t+1}, \vec{a})$.

Plusieurs architectures sont considérées :



Mais VDN limité à une certaine classe additive de fonctions de valeur et impossible d'intégrer des informations supplémentaires sur l'état pendant l'apprentissage.

QMIX [Ras+18] relaxe ces contraintes en affirmant qu'il suffit que :

$$\arg \max_{\boldsymbol{u}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{pmatrix} \arg \max_{u_1} Q_1(\tau_1, u_1) \\ \dots \\ \arg \max_{u_N} Q_N(\tau_N, u_N) \end{pmatrix}$$

Cela correspond à la classe des fonctions de valeur Q monotones, c'est à dire respectant :

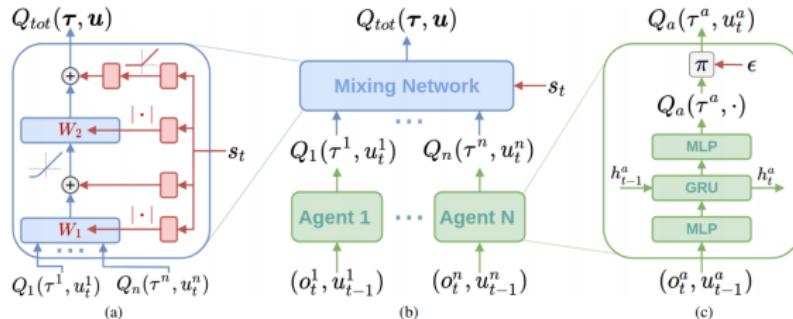
$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a \in \{1, \dots, N\}$$

En plus d'élargir la classe de fonctions, cela permet de simplifier grandement le choix de $\arg \max_{\boldsymbol{u}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u})$ en fonction des valeurs individuelles.

QMIX : Monotonic Value Function Factorisation for MARL

Pour s'assurer de la contrainte de monotonie, QMIX emploie 3 types de réseaux :

- **Agent Network** : produit la valeur Q_i pour chaque agent i en fonction de son historique (mémoire RNN), de son observation courante et de sa dernière action
- **Mixing Network** : produit Q_{tot} à partir des valeurs individuelles. Pour garantir la monotonie, les poids (mais pas les biais) de ce réseau sont tous non-négatifs.
- **Ensemble d'HyperNetworks** : chacun d'entre eux prend l'état en entrée (ou toute autre information additionnelle pendant l'apprentissage) et produit les poids et les biais de chaque couche du Mixing Network (activation valeur absolue pour les poids)



- ⇒ Q_{tot} peut dépendre de l'état de manière non-monotone. Le passer entrée du Mixing Network conduirait à contraindre trop fortement l'apprentissage.
- ⇒ Apprentissage par différence temporelle selon Q_{tot}^θ et un réseau cible $Q_{tot}^{\theta^-}$
- ⇒ Après l'apprentissage, seul l'Agent Network est utilisé. Et les actions sont choisies individuellement (complexité linéaire).

QMIX : Monotonic Value Function Factorisation for MARL

Problème à 2 agents, 2 étapes :

		Agent 2	
		A	B
Agent 1	A	7	7
	B	7	7
State 2A		Agent 2	
Agent 1	A	0	1
	B	1	8
State 2B			

Rewards selon les actions des deux agents à l'étape 2 selon l'action choisie par A à l'étape 1 (agent 2 n'a pas d'effet sur l'étape 1)

(a)	State 1		State 2A		State 2B	
	A	B	A	B	A	B
A	6.94	6.94	6.99	7.02	-1.87	2.31
	6.35	6.36	6.99	7.02	2.33	6.51

(b)	State 1		State 2A		State 2B	
	A	B	A	B	A	B
A	6.93	6.93	7.00	7.00	0.00	1.00
	7.92	7.92	7.00	7.00	1.00	8.00

Q_{tot} on the two-step game for (a) VDN and (b) QMIX.

- 👍 Plus efficace que VDN pour de nombreux problèmes (moins contraint, plus d'info pendant l'apprentissage)
- 👎 Pas adapté aux problèmes non-monotones, où les actions de valeur maximale pour chaque agent dépendent des actions des autres

COMA : Counterfactual Multi-Agent Policy Gradients

COMA [Foe+17] : Policy Gradient avec Centralized training et Decentralized execution :

- Actor-Critic avec critique centralisée (pendant l'apprentissage on peut disposer de toutes les observations, voire de l'état, pour former la critique, par exemple sur un simulateur)
- Reward global (tâches avec coopération nécessaire pour atteindre un but commun)
- Chaque agent dispose d'un historique d'action-observations τ^a sur lequel il définit une politique $\pi(u|\tau^a)$

Une approche naïve serait de considérer le gradient de politique suivant pour chaque acteur a à chaque temps t :

$$\nabla_{\theta_a} \pi_{\theta_a}(u_t | \tau_t^a) (r_t + \gamma V(s_{t+1}) - V(s_t))$$

avec $\pi_{\theta_a}(u_t | \tau_t^a)$ la probabilité de l'action u_t selon la politique de l'agent a et l'historique d'observations-actions τ_t^a au temps t .

⇒ Problème de credit assignment : difficile de distribuer le feedback sur les agents avec une critique globale, gradient très bruité

COMA : Counterfactual Multi-Agent Policy Gradients

COMA [Foe+17] :

- Counterfactual baseline : Difference de rewards espérés selon les possibles actions de chaque agent
- Basé sur [WT02] qui introduit une baseline qui permet de comparer l'action choisie à une action défaut c^a : $r(s, u) - r(s, (u^{-a}, c))$
 - Mais cela nécessite d'interagir de trop nombreuses fois avec le simulateur (+ requiert la possibilité de défaire une action)
- COMA intègre cette idée dans une fonction d'avantage (avec Q compatible) :

$$A^a(s, u) = Q(s, u) - \sum_{u'^a} \pi_{\theta_a}(u_t | \tau_t^a) Q(s, (u^{-a}, u'^a)) \quad \text{Gradient ?}$$

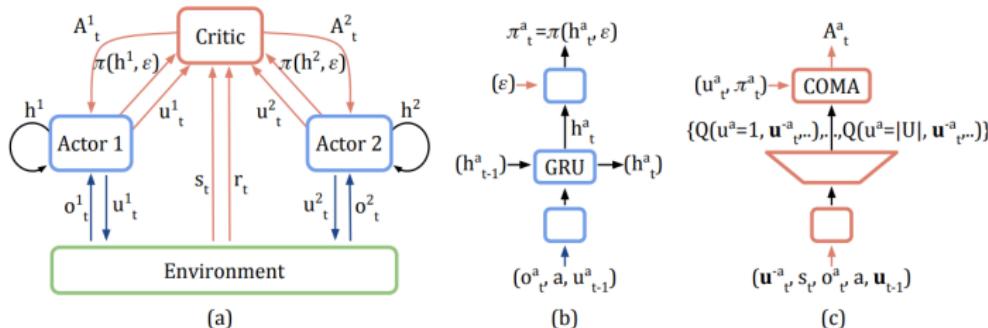
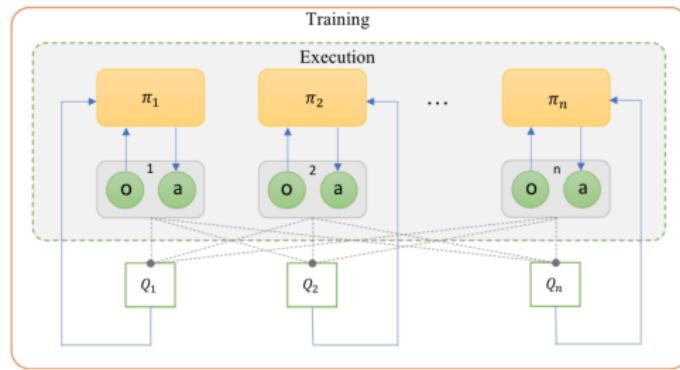


Figure 1: In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

MADDPG : Multi-agent actor-critic for mixed cooperative-competitive environments

MADDPG [Low+17] :

- Rewards individuels permettant des environnements compétitifs
- Actions continues (basé sur DDPG)
- Critiques individuelles centralisées



Gradient pour l'agent i selon un buffer d'expérience replay \mathcal{D} :

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,a \sim \mathcal{D}} [\nabla_{\theta_i} \mu_i(o_i) \nabla_{a_i} Q_i^\mu(x, a_1, \dots, a_N) |_{a_i=\mu_i(o_i)}]$$

avec x contenant le contexte d'action (état si disponible, observations des agents) et Q_i^μ appris selon des politiques différentes μ' et un réseau cible $Q_i^{\mu'}$:

$$\min \mathbb{E}_{x,a,x',r \sim \mathcal{D}} [(Q_i^\mu(x, a_1, \dots, a_N) - y)^2] \text{ avec } y = r_i + \gamma Q_i^{\mu'}(x, \mu'_1(o'_1), \dots, \mu'_N(o'_N))$$

MADDPG : Multi-agent actor-critic for mixed cooperative-competitive environments

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode = 1 to M **do**

 Initialize a random process \mathcal{N} for action exploration

 Receive initial state \mathbf{x}

for $t = 1$ to max-episode-length **do**

 for each agent i , select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration

 Execute actions $a = (a_1, \dots, a_N)$ and observe reward r and new state \mathbf{x}'

 Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer \mathcal{D}

$\mathbf{x} \leftarrow \mathbf{x}'$

for agent $i = 1$ to N **do**

 Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from \mathcal{D}

 Set $y^j = r_i^j + \gamma Q_i^{\mu'}(\mathbf{x}'^j, a'_1, \dots, a'_N)|_{a'_k=\mu'_k(o'_k)}$

 Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S} \sum_j (y^j - Q_i^{\mu}(\mathbf{x}^j, a_1^j, \dots, a_N^j))^2$

 Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j) \nabla_{a_i} Q_i^{\mu}(\mathbf{x}^j, a_1^j, \dots, a_i^j, \dots, a_N^j)|_{a_i=\mu_i(o_i^j)}$$

end for

 Update target network parameters for each agent i :

$$\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$$

end for

end for

⇒ On établit la cible en fonction de $\mu'_1(o'_1), \dots, \mu'_N(o'_N)$ plutôt que des actions suivantes enregistrées dans le buffer pour prendre en compte l'évolution des politiques

⇒ $\mu'_1(o'_1), \dots, \mu'_N(o'_N)$ plutôt que $\mu_1(o'_1), \dots, \mu_N(o'_N)$ afin d'éviter les changements trop brutaux pour l'apprentissage de Q

MADDPG : Multi-agent actor-critic for mixed cooperative-competitive environments

L'idée derrière MADDPG (et l'évaluation de Q selon l'ensemble des actions des agents) est de se dire que si l'on connaît les actions prises par l'ensemble des agents, le problème devient stationnaire même si les politiques des agents évoluent :

$$P(s'|s, a_1, \dots, a_N, \pi_1, \dots, \pi_N) = P(s'|s, a_1, \dots, a_N) = P(s'|s, a_1, \dots, a_N, \pi'_1, \dots, \pi'_N)$$

Cependant l'évaluation de Q telle que définie précédemment nécessite de connaître l'ensemble des politiques. Si ce n'est pas le cas (agents ne communiquent pas), il est possible de faire apprendre à chaque agent i un modèle de chaque agent $j \neq i$:

$$\phi_i^j \leftarrow \phi_i^j + \alpha \mathbb{E}_{o_j, a_j} [\nabla_{\phi_i^j} \log p_{\phi_i^j}(a_j | o_j) - \lambda \nabla_{\phi_i^j} H(p_{\phi_i^j}(\cdot | o_j))]$$

avec par exemple $p_{\phi_i^j}(a_j | o_j) = \mathcal{N}(a_j; \mu_{\phi_i^j}(j), \sigma_{\phi_i^j}^2(o_j)I)$ et H l'entropie de la distribution.

On peut alors considérer :

$$\min \mathbb{E}_{x, a, x', r \sim \mathcal{D}} [(Q_i^\mu(x, a_1, \dots, a_N) - y)^2] \text{ avec } y = r_i + \gamma Q_i^{\mu'}(x, \mu_{\phi_i^1}(o'_1), \dots, \mu_{\phi_i^N}(o'_N))$$

Encore faut-il supposer que l'on ait accès aux observations des différents agents, mais cela peut se faire lors de la transmission au buffer de replay (moins contraignant que la transmission des paramètres de politique qui devrait être réalisée en temps réel).

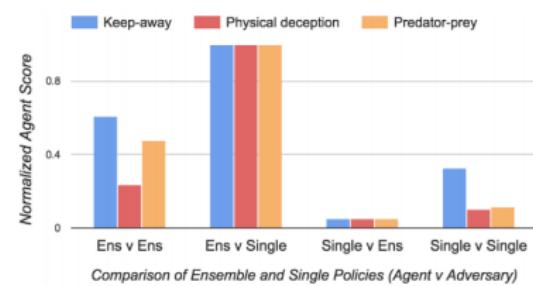
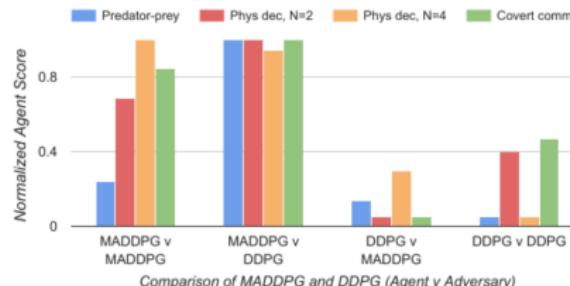
MADDPG : Multi-agent actor-critic for mixed cooperative-competitive environments

Pour obtenir des politiques robustes aux changements des autres agents (e.g., éviter de sur-apprendre sur les politiques des agents concurrents), MADDPG propose également de considérer des ensemble de politiques par agent : chaque agent possède K politiques $\mu_i^{(1)}, \dots, \mu_i^{(K)}$.

On vise alors à maximiser pour chaque agent i : $\mathbb{E}_{k \sim \text{unif}(1, K), s \sim d^\mu, a_i \sim \mu_i^{(k)}} [R_i(s, a_i)]$

Gradient correspondant (avec k replay buffers par agent) :

$$\nabla_{\theta_i^{(k)}} J(\mu_i) = \mathbb{E}_{x, a \sim D_i^{(k)}} [\nabla_{\theta_i^{(k)}} \mu_i^{(k)}(o_i) \nabla_{a_i} Q_i^\mu(x, a_1, \dots, a_N) |_{a_i = \mu_i^{(k)}(o_i)}]$$



- 👍 Bonnes performances sur diverses tâches collaboratives/competitives
- 👍 Ensemble de politiques meilleur que politiques uniques
- 👉 Taille de l'entrée de Q croît linéairement avec le nombre d'agents (on pourrait envisager de ne s'intéresser qu'à un voisinage d'agents pour limiter)

Inequity aversion improves cooperation in MARL

Aucune de ces approches ne garantit qu'un agent ne puisse prendre trop de pouvoir

- Particulièrement vrai avec rewards individuels
- Mais aussi avec les rewards collectifs, pour lesquels on peut converger vers des solutions sous-optimales où certains agents sont négligés

Pour bien des tâches, c'est problématique : Si les agents considérés sont considérés, c'est a priori pour qu'ils servent à quelque chose (sinon problème mal-dimensionné)

Lorsque les moyennes de rewards cumulés par les agents devraient être comparables, on peut définir des récompenses qui modélisent l'aversion à l'inégalité [Hug+18] :

$$u_i(s_i^t, a_i^t) = r_i(s_i^t, a_i^t) - \frac{\alpha_i}{N-1} \sum_{i \neq j} \max(e_j^t(s_j^t, a_j^t) - e_i(s_i^t, a_i^t), 0) \\ - \frac{\beta_i}{N-1} \sum_{i \neq j} \max(e_i^t(s_i^t, a_i^t) - e_j(s_j^t, a_j^t), 0)$$

avec $e_j^t(s_j^t, a_j^t)$ le reward lissé de l'agent j , mis à jour à chaque pas de temps t selon :

$$e_j^t(s_j^t, a_j^t) \leftarrow \lambda \gamma e_j^{t-1}(s_j^{t-1}, a_j^{t-1}) + r_i(s_i^t, a_i^t)$$

Dans [Hug+18] : agents A3C qui ont accès aux valeurs e_j^t de tous les agents j .
 α_i et β_i contrôlent les caractéristiques de l'agent i :

- Un fort α_i diminue le score de l'agent i si un autre agent a un meilleur reward extrinsèque que lui (envie)
- Un fort β_i diminue le score de l'agent i si un autre agent a un moins bon reward extrinsèque que lui (culpabilité)

FEN : Fairness in Multi-Agents Reinforcement Learning

FEN [JL19] va plus loin que ce type de modèle d'envie et de culpabilité, en garantissant un équitable partage de ressources dans les problèmes sur lesquels ce genre de contrainte a un intérêt.

- Utilité considérée : $u_t^i = \frac{1}{t} \sum_{j=0}^t r_j^i$, avec r_t^i uniquement lié à l'utilisation des ressources en t
- Critère d'équité considéré = coefficient de variation (CV) : $\sqrt{\frac{1}{N-1} \sum_{i=1}^N \frac{(u^i - \bar{u})^2}{\bar{u}^2}}$
- ⇒ Reward intrinsèque : $\hat{r}_t^i = \frac{\bar{u}_t/c}{\epsilon + |u_t^i/\bar{u}_t - 1|}$, avec c le reward extrinsèque maximal atteignable sur un pas de temps
- ⇒ Objectif : maximiser pour chaque agent i , selon sa politique individuelle apprise de manière décentralisée, $F_i = \sum_{t=0}^{\infty} \gamma \hat{r}_t^i$

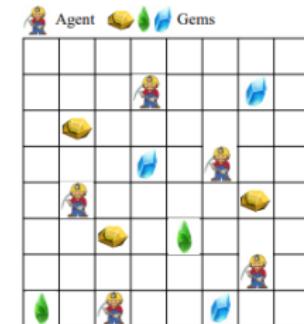
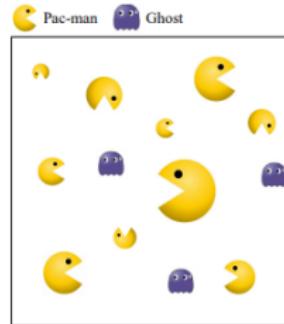
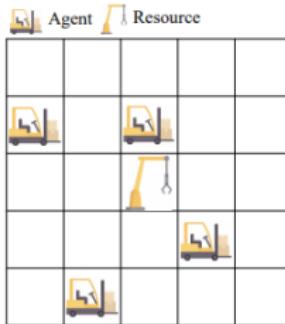
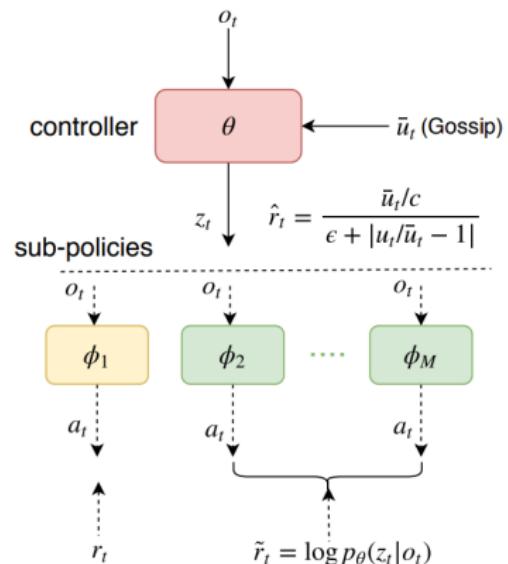


Illustration of experimental scenarios: job scheduling (*left*), the Matthew effect (*mid*), manufacturing plant (*right*).

FEN : Fairness in Multi-Agents Reinforcement Learning

Après avoir montré qu'à l'optimum, on est garanti d'avoir un partage équitable des ressources avec ce genre de reward, FEN [JL19] définit la politique d'un agent comme un réseau hiérarchique (difficile de traiter les deux objectifs contradictoires fairness-efficiency sur un seul niveau) :

- Niveau supérieur : contrôleur qui choisit tous les T pas de temps entre les sous-politiques à appliquer
- Niveau inférieur : sous-politiques qui interagissent avec l'environnement
 - 1 qui choisit une action en fonction de l'état et reçoit les récompenses de l'environnement.
Objectif : maximiser l'efficacité
 - M-1 autres qui visent à être le plus diverses possibles (grâce à un terme d'entropie) et le plus favorables au contrôleur (grâce à un reward dépendant de leur probabilité d'être choisie selon l'observation en courante).
Objectif : trouver des stratégies qui servent la fairness



FEN architecture.

Exécution et Apprentissage décentralisés

Algorithm 1 FEN training

```

1: Initialize  $u_i, \bar{u}_i$ , the controller  $\theta$  and sub-policies  $\phi$ 
2: for episode = 1, ...,  $\mathcal{M}$  do
3:   The controller chooses one sub-policy  $\phi_z$ 
4:   for  $t = 1, \dots, \text{max-episode-length}$  do
5:     The chosen sub-policy  $\phi_z$  acts to the environment
       and gets the reward  $\begin{cases} r_t & \text{if } z = 1, \\ \log p_\theta(z|o_t) & \text{else} \end{cases}$ 
6:     if  $t\%T = 0$  then
7:       Update  $\phi_z$  using PPO
8:       Update  $\bar{u}_i$  (with gossip algorithm)
9:       Calculate  $\hat{r}^i = \frac{\bar{u}_i/c}{\epsilon + |u^i/\bar{u}_i - 1|}$ 
10:      The controller reselects one sub-policy
11:    end if
12:  end for
13:  Update  $\theta$  using PPO
14: end for

```

⇒ Agents coordonnés par un consensus moyen d'utilité

⇒ Si communication limitée à un certain voisinage

⇒ Algorithme de Gossip pendant S itérations pour atteindre un consensus moyen sur le voisinage (algo de type Metropolis-Hastings) [XBK07] :

$$\bar{u}_i(s+1) \leftarrow \bar{u}_i(s) + \sum_{j \in \mathcal{N}_i^t} w_{i,j} (\bar{u}_j(t) - \bar{u}_i(s)) \text{ avec } \bar{u}_i(0) = u_i^t \text{ et}$$

$$w_{i,j} = 1 / (\max(|\mathcal{N}_i^t|, |\mathcal{N}_j^t|) + 1) \text{ et } \mathcal{N}_i^t \text{ le voisinage de } i \text{ à l'instant } t.$$

Multi-Agents Reinforcement Learning

Tableau de synthèse des principaux challenges et algorithmes associés
(source : [NNN18])

Challenges	Value-based	Actor-critic	Policy-based
Partial observability	DRQN [36]; DDRQN [24]; RIAL and DIAL [23]; Action-specific DRQN [121]; MT-MARL [85]; PSDQN [30]; RL as a Rehearsal (RLaR) [55]	PS-DDPG and PS-A3C [30]; MADDPG-M [48]	DPIQN and DRPIQN [42]; PS-TRPO [30]; Bayesian action decoder (BAD) [26]
Non-stationarity	DRUQN and DLCQN [12]; Multi-agent concurrent DQN [18]; Recurrent DQN-based multi-agent importance sampling and fingerprints [25]; Hysteretic-DQN [85]; Lenient-DQN [86]; WDDQN [120]	MADDPG [68]; PS-A3C [30]	PS-TRPO [30]
Continuous action spaces		Recurrent DPG [39]; DDPG [66]	TRPO [101]; PS-TRPO [30]
Multi-agent training schemes	Multi-agent extension of DQN [111]; RIAL and DIAL [23]; CommNet [109]; DRON [38]; MS-MARL [53, 54]; Linearly fuzzified joint Q-function for MAS [69]	MADDPG [68]; COMA [27]	
Transfer learning in MAS	Policy distillation [96]; Multi-task policy distillation [117]; Multi-agent DQN [20]	Progressive networks [97]	Actor-Mimic [87]

Sources

- Yannis Flet-Berliac : <https://thegradient.pub/the-promise-of-hierarchical-reinforcement-learning/>
- Open-ai : <https://openai.com/blog/learning-to-cooperate-compete-and-communicate/>
- OpenAi :
<https://openai.com/blog/learning-a-hierarchy/>

References |

- [And+17] Marcin Andrychowicz et al. « Hindsight Experience Replay ». In : *CoRR abs/1707.01495* (2017). arXiv : 1707.01495.
- [BHP16] Pierre-Luc Bacon, Jean Harb et Doina Precup. « The Option-Critic Architecture ». In : *CoRR abs/1609.05140* (2016). arXiv : 1609.05140.
- [BN06] Drew Bagnell et Andrew Y Ng. « On local rewards and scaling distributed reinforcement learning ». In : *Advances in Neural Information Processing Systems*. 2006, p. 91-98.
- [Cza+18] Wojciech Marian Czarnecki et al. « Mix&Match - Agent Curricula for Reinforcement Learning ». In : *CoRR abs/1806.01780* (2018). arXiv : 1806.01780.
- [DH93] Peter Dayan et Geoffrey E Hinton. « Feudal reinforcement learning ». In : *Advances in neural information processing systems*. 1993, p. 271-278.
- [Flo+17] Carlos Florensa et al. « Reverse curriculum generation for reinforcement learning ». In : *arXiv preprint arXiv :1707.05300* (2017).
- [Foe+17] Jakob Foerster et al. « Counterfactual Multi-Agent Policy Gradients ». en. In : *arXiv :1705.08926 [cs]* (mai 2017). arXiv : 1705.08926.
- [Fra+17] Kevin Frans et al. « Meta Learning Shared Hierarchies ». In : *CoRR abs/1710.09767* (2017). arXiv : 1710.09767.

References II

- [GEK17] Jayesh K Gupta, Maxim Egorov et Mykel Kochenderfer. « Cooperative multi-agent control using deep reinforcement learning ». In : *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2017, p. 66-83.
- [Hug+18] Edward Hughes et al. « Inequity aversion resolves intertemporal social dilemmas ». In : *CoRR abs/1803.08884* (2018). arXiv : 1803.08884.
- [Jad+17] Max Jaderberg et al. « Population Based Training of Neural Networks ». In : *CoRR abs/1711.09846* (2017). arXiv : 1711.09846.
- [JL19] Jiechuan Jiang et Zongqing Lu. « Learning Fairness in Multi-Agent Systems ». In : *Thirty-Third Annual Conference on Neural Information Processing Systems (NIPS)*. 2019.
- [KB16] Landon Kraemer et Bikramjit Banerjee. « Multi-agent reinforcement learning as a rehearsal for decentralized planning ». In : *Neurocomputing* 190 (2016), p. 82 -94.
- [Kön03] Ville Könönen. « Asymmetric Multiagent Reinforcement Learning ». In : *IAT*. 2003.
- [Low+17] Ryan Lowe et al. « Multi-agent actor-critic for mixed cooperative-competitive environments ». In : *Advances in Neural Information Processing Systems*. 2017, p. 6379-6390.
- [Nac+18] Ofir Nachum et al. « Data-Efficient Hierarchical Reinforcement Learning ». In : *CoRR abs/1805.08296* (2018). arXiv : 1805.08296.

References III

- [Nai+18] Ashvin Nair et al. « Visual Reinforcement Learning with Imagined Goals ». In : *CoRR abs/1807.04742* (2018). arXiv : 1807.04742.
- [Nar+16] Sanmit Narvekar et al. « Source Task Creation for Curriculum Learning ». In : *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. AAMAS '16. Singapore, Singapore : International Foundation for Autonomous Agents et Multiagent Systems, 2016, p. 566-574.
- [NHR99] Andrew Y. Ng, Daishi Harada et Stuart J. Russell. « Policy Invariance Under Reward Transformations : Theory and Application to Reward Shaping ». In : *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999, p. 278-287.
- [NNN18] Thanh Thi Nguyen, Ngoc Duy Nguyen et Saeid Nahavandi. « Deep reinforcement learning for multi-agent systems : a review of challenges, solutions and applications ». In : *arXiv preprint arXiv :1812.11794* (2018).
- [Pas+18] Alexander Pashevich et al. « Modulated Policy Hierarchies ». In : *CoRR abs/1812.00025* (2018). arXiv : 1812.00025.
- [Pat+17] Deepak Pathak et al. « Curiosity-driven exploration by self-supervised prediction ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, p. 16-17.

References IV

- [Ras+18] Tabish Rashid et al. « QMIX : Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning ». In : *CoRR abs/1803.11485* (2018). arXiv : 1803.11485.
- [SPS99] Richard S Sutton, Doina Precup et Satinder Singh. « Between MDPs and semi-MDPs : A framework for temporal abstraction in reinforcement learning ». In : *Artificial intelligence* 112.1-2 (1999), p. 181-211.
- [Suk+17] Sainbayar Sukhbaatar et al. « Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play ». In : *CoRR abs/1703.05407* (2017). arXiv : 1703.05407.
- [Suk+18] Sainbayar Sukhbaatar et al. « Learning Goal Embeddings via Self-Play for Hierarchical Reinforcement Learning ». In : *CoRR abs/1811.09083* (2018). arXiv : 1811.09083.
- [Sun+17] Peter Sunehag et al. « Value-Decomposition Networks For Cooperative Multi-Agent Learning ». In : *CoRR abs/1706.05296* (2017). arXiv : 1706.05296.
- [Sve+17] Maxwell Svetlik et al. « Automatic curriculum graph generation for reinforcement learning agents ». In : *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
- [Tam+17] Ardi Tampuu et al. « Multiagent cooperation and competition with deep reinforcement learning ». In : *PloS one* 12.4 (2017), e0172395.

- [Vez+17] Alexander Sasha Vezhnevets et al. « FeUDal Networks for Hierarchical Reinforcement Learning ». In : *CoRR abs/1703.01161* (2017). arXiv : [1703.01161](https://arxiv.org/abs/1703.01161).
- [WCE03] Eric Wiewiora, Garrison W Cottrell et Charles Elkan. « Principled methods for advising reinforcement learning agents ». In : *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, p. 792-799.
- [WT02] David H Wolpert et Kagan Tumer. « Optimal payoff functions for members of collectives ». In : *Modeling complexity in economic and social systems*. World Scientific, 2002, p. 355-369.
- [XBK07] Lin Xiao, Stephen Boyd et Seung-Jean Kim. « Distributed average consensus with least-mean-square deviation ». In : *Journal of parallel and distributed computing* 67.1 (2007), p. 33-46.