Reinforcement Learning & Advanced Deep

M2 DAC

TME 8. Soft Actor-Critic (SAC)

Ce TME a pour objectif d'expérimenter l'approche SAC, pour environnements à actions continues, avec maintient d'entropie.

SAC

L'objectif est d'implémenter et expérimenter l'algorithme SAC donné par l'algorithme 1 ci dessous, utilisant dans un premier temps une température α fixe, donnée en hyper-paramètre de l'algorithme (nous verrons la version adaptative dans la section suivante).

Plutôt que de définir un acteur déterministe comme dans DDPG, ce qui pose des problèmes d'instabilités dues aux approximations off-policy sur lesquelles ce genre d'approche repose, n'offre pas de réel contrôle sur le processus d'exploration et ne permet pas d'obtenir des agents efficaces pour diverses applications où la stochasticité de la politique définie est cruciale (e.g., problèmes adverses), l'idée de SAC est de chercher à approcher une politique stochastique maximisant l'entropie à long terme selon les états atteints, en incluant l'entropie comme récompense intrinsèque $\mathcal{H}(\pi_\theta(.|s_t))$:

$$\pi^*_{MaxEnt} = \arg\max_\pi \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(.|s_t))]$$

Lorsque l'on cherche une politique π_θ stochastique, par exemple selon $\pi_\theta(.|s_t) = \mathcal{N}(\mu_\theta(s_t), \sigma_\theta(s_t)I)$ pour tout état s_t comme on considérera dans ce TP, avec $\mu_\theta(s_t)$ et $\sigma_\theta(s_t)$ des sorties vectorielles du réseau de la politique donnant respectivement les paramètres de moyenne et de variance d'une loi normale multivariée sur les actions (on travaille avec une matrice de variance-covariance $\Sigma = \sigma I$ diagonale), il est en effet crucial de contrôler l'entropie de la politique, sans quoi elle tend à s'effondrer rapidement sur son mode (μ), ne lui permettant alors plus aucune exploration (et impliquant même de nombreuses difficultés d'instabilités). Plutôt que d'ajouter à l'objectif PG un simple terme d'entropie ponctuelle comme dans de nombreuses approches (e.g., Reinforce), ce qui mène à conserver de l'incertitude sur chaque état rencontré, l'idée de l'incorporer ici comme récompense intrinsèque, permet de favoriser les actions qui mèneront vers des

zones plus incertaines dans le futur, offrant un contrôle sur le niveau d'exploration long terme de la politique (entropie sur les trajectoires plutôt que sur les actions en chaque état).

Alors que Soft Q-Learning [1] atteint cet objectif en considérant une politique de maximum d'entropie $\pi(a|s) \propto exp(Q^{\pi}_{\text{soft}}(s,a))$ avec:

- $Q^{\pi}_{\text{soft}}(s_t,a_t) = r_t + \gamma \mathbb{E}_{s_{t+1}|s_t,a_t}[V^{\pi}_{\text{soft}}(s_{t+1})]$
- $V^{\pi}_{\text{soft}}(s_t) = \mathbb{E}_{a_t|s_t}[Q^{\pi}_{\text{soft}}(s_t,a_t) - \alpha log \pi(a_t|s_t)]$

et $\pi(a|s)$ une distribution de Bolzman selon $Q^{\pi}_{\text{soft}}(s,a)$, Soft Actor-Critic (SAC) [2] s'intéresse à chercher une politique (actor) proche de cette distribution de maximum d'entropie selon Q_{soft} (critic). Cela permet notamment de traiter des problèmes à actions continues (là où il est difficile d'échantillonner selon $exp(Q^{\pi}_{\text{soft}}(s,a))$), en considérant un acteur paramétrique (e.g., gaussienne multivariée). En outre, cela permet de restreindre l'espace d'exploration de manière régulière autour d'un certain mode pour chaque état, plutôt que de baser la politique d'échantillonnage sur une fonction de valeur possiblement très irrégulière, et à très forte variance, particulièrement problématique dans le cas d'actions en grande dimension.

À chaque itération, l'idée est d'optimiser la politique selon:

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'\left(\cdot|\mathbf{s}_t\right) \| \frac{\exp\left(Q^{\pi_{\text{old}}}_{\text{soft}}\left(\mathbf{s}_t,\cdot\right)\right)}{Z^{\pi_{\text{old}}}\left(\mathbf{s}_t\right)}\right)$$

avec $Q^{\pi_{\text{old}}}_{\text{soft}}$ la valeur Q_{soft} calculée sur la politique à l'itération précédente π_{old} et $Z^{\pi_{\text{old}}}\left(\mathbf{s}_t\right)$ la fonction de partition d'une distribution proportionnelle à $exp(Q^{\pi_{old}}_{\text{soft}}(s,a))$. Notons que cette fonction de partition (intractable) ne dépend pas de la politique à optimiser et n'intervient alors pas dans le gradient.

Pour V_{soft} et Q_{soft}, on considère des approximations optimisées selon les coûts:

$$J_V(\psi) = \mathbb{E}_{s_t \sim D}\left[\frac{1}{2}\left(V_\psi\left(\mathrm{s}_t\right) - \mathbb{E}_{\mathrm{a}_t \sim \pi_\phi}\left[Q_\theta\left(\mathrm{s}_t,\mathrm{a}_t\right) - \alpha \log \pi_\phi\left(\mathrm{a}_t|\mathrm{s}_t\right)\right]\right)^2\right]$$

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta\left(\mathbf{s}_t,\mathbf{a}_t\right) - \hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right)^2\right]$$

avec $\hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right) = r(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1}|s_t,a_t}[V_{\hat{\psi}}(s_{t+1})]$, où $V_{\hat{\psi}}(s_{t+1})$ est un réseau cible mis à jour périodiquement en fonction de V_ψ (moyenne exponentielle)

Pour la politique on considère:

$$\begin{aligned} J_\pi(\phi) &= \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\mathrm{D}_{\mathrm{KL}}\left(\pi_\phi\left(\cdot|\mathbf{s}_t\right)|\frac{\exp\left(Q_\theta\left(\mathbf{s}_t,\cdot\right)\right)}{Z_\theta\left(\mathbf{s}_t\right)}\right)\right] \\ &= \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\mathbb{E}_{\mathbf{a}_t\sim\pi_\phi(.|s_t)}\left[\log \pi_\phi\left(\mathbf{a}_t|\mathbf{s}_t\right) - Q_\theta\left(\mathbf{s}_t,\mathbf{a}_t\right)\right] + \log Z_\theta\left(\mathbf{s}_t\right)\right] \end{aligned}$$

Plutôt que d'utiliser le log-trick pour optimiser selon une action samplée (REINFORCE

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, V-function parameters ψ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\psi_{\text{targ}} \leftarrow \psi$

repeat

  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

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

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for j in range(however many updates) **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets for Q and V functions:

2 réseaux Q pour éviter les sur-estimations

$$y_q(r, s', d) = r + \gamma(1 - d)V_{\psi_{\text{targ}}}(s')$$
$$y_v(s) = \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s), \qquad \tilde{a} \sim \pi_\theta(\cdot|s)$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi,i}(s, a) - y_q(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update V-function by one step of gradient descent using

$$\nabla_\psi \frac{1}{|B|} \sum_{s \in B} \left(V_\psi(s) - y_v(s)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \Big(Q_{\phi,1}(s, \tilde{a}_\theta(s)) - \quad \log \pi_\theta \left(\tilde{a}_\theta(s)| \, s\right) \Big),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.

      Update target value network with

$$\psi_{\text{targ}} \leftarrow \rho\psi_{\text{targ}} + (1 - \rho)\psi$$

    end for

  end if

until convergence

assez inefficace dans le cas continu), on considère une re-paramétrisation $a_t = f_\phi(\epsilon_t; \mathbf{s}_t)$, avec $\epsilon_t \sim p(\epsilon_t)$, tel que pour tout h: $\mathbb{E}_{a_t|s_t}[h(s_t, a_t)] = \mathbb{E}_{\epsilon_t \sim p(\epsilon_t)}[h(s_t, f_\phi(\epsilon_t; \mathbf{s}_t))]$ (**Reparameterization-trick**). Si on considère $\pi_\phi(.|s_t) = \mathcal{N}(\mu_\phi(s_t), \sigma_\phi(s_t)^2)$, on peut choisir $\epsilon_t \sim \mathcal{N}(0, 1)$ et définir $a_t = f_\phi(\epsilon_t; s_t) = \epsilon_t \times \sigma_\phi(s_t) + \mu_\phi(s_t)$ (en notant que la génération d'une VA $X \sim \mathcal{N}(\mu, \sigma^2)$ se fait en prenant $X = \sigma \times \epsilon + \mu$, avec ϵ issu d'une loi normale standard). On peut alors calculer le gradient de manière classique, en samplant ce bruit selon une gaussienne standard. Sous Pytorch, cela se fait automatiquement selon la fonction `rsample()` de la loi normale (package distributions), qui permet d'échantillonner selon cette reparamétrisation et ainsi être à même de rétropropager selon les paramètres de la distribution visée.

En considérant deux réseaux Q pour éviter les sur-estimations des valeurs, cela donne lieu à l'algorithme 1 donné ci-dessus.

Appliquer l'algorithme aux 3 problèmes suivants:

- MountainCarContinuous-v0
- LunarLanderContinuous-v2
- Pendulum-v0

Une difficulté provient du fait que pour ces environnements, l'ensemble des actions dans $\mathbb{R}^d$, avec d la dimension des actions, ne sont pas valides. Il faut alors projeter les actions échantillonnées selon la politique gaussienne (qui appartiennent donc à $\mathbb{R}^d$) dans l'ensemble des actions valides pour l'environnement considéré $\mathcal{A} = \prod_{i=1}^{d}[low_i; high_i]$. Proposer une méthode de projection des actions échantillonnées basée sur la fonction tanh, impliquant l'adaptation correspondante des distributions de probabilités associées, sachant que pour ce genre de fonction de transformation inversible, on peut employer pour cela la formule de changement de variable suivante:

$$\log p(f(x)) = \log p(x) - \log |det(\frac{\partial f(x)}{\partial x})|$$

avec $f : \mathbb{R}^d \to \{low_i, high_i\}_{i=1}^d$ la fonction de transformation employée et *det* le déterminant d'une matrice. Ici la transformation travaillant élément par élément, on a simplement $\log |det(\frac{\partial f(x)}{\partial x})| = \sum_{i=1}^{d} \log \frac{\partial f_i(x_i)}{\partial x_i}$. Attention à passer par le log pour le calcul de la jacobienne, pour éviter les instabilités numériques.

Implémenter une version de SAC avec température α fixe et un réseau V implicite: on ne s'embête pas à avoir une réseau spécifique pour calculer V comme à la ligne 14 de l'algorithme, il est tout aussi simple d'utiliser une cible TD(0) avec entropie pour les mises à jour des réseaux Q, en échantillonnant à chaque itération une nouvelle action selon l'état destination s'. Il est recommandé de coder pour cela une fonction $getAction(etat)$ qui échantillonne une nouvelle action en fonction de l'état passé en paramètre et la retourne associée à sa probabilité, qui sera utilisée à la fois dans *act*

pour choisir une nouvelle action à jouer dans l'environnement et également dans la fonction *learn* pour obtenir des actions selon s' ainsi que les probabilités courantes des transitions du buffer. C'est dans cette fonction que l'on implémentera la transformation discutée ci-dessus (et elle aussi qui utilise $rsample$).

SAC avec température adaptative

Sachant qu'il est difficile de régler efficacement le paramètre de température α pour avoir un réel contrôle sur le niveau d'entropie tout au long du processus d'apprentissage, une deuxième version de SAC définit une température adaptative, en considérant le problème sous contrainte:

$$\max_{\pi_0,\dots,\pi_T} \mathbb{E}\Big[\sum_{t=0}^{T} r(s_t, a_t)\Big] \text{s.t. } \forall t,\ \mathcal{H}(\pi_t) \geq \mathcal{H}_0$$

avec $\mathcal{H}_0$ un seuil d'entropie à assurer. Cela revient à considérer le coût d'optimisation $J(\alpha)$ suivant, obtenu selon le Lagrangien de ce problème sous contrainte (α est le coefficient de Lagrange):

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t}[-\alpha \log \pi_t(a_t \mid s_t) - \alpha \mathcal{H}_0]$$

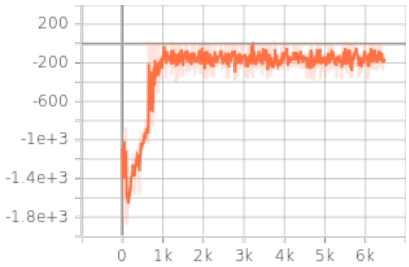
Implémenter cette nouvelle version de SAC, en suivant la procédure donnée dans l'algorithme:

Algorithm 1 Soft Actor-Critic

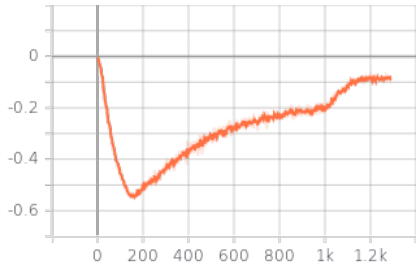
Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

À titre indicatif, voici ce qu'on peut obtenir sur Pendulum avec SAC utilisant un processus de réglage adaptatif de la température α pour une entropie visée $\mathcal{H}_0 = 0.2$, effectuant une optimisation de 10 étapes tous les 1000 évènements, selon deux target networks Q_1 et Q_2 (mis à jour de manière "soft" à chaque optimisation selon des

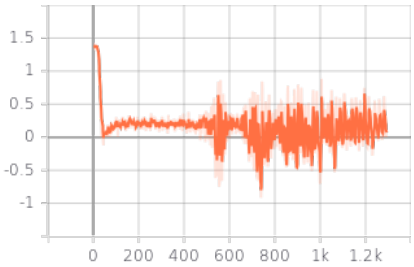
paramètres $\rho = 0.9$), pas de réseau V (calculée de manière implicite en fonction de Q), un discount de 0.95, des récompenses divisées par 100, un pas d'apprentissage de 0.001 pour l'acteur, de 0.01 pour les critiques et de 0.001 pour α, un batch de 1000 transitions échantillonnées dans un buffer de capacité 1000000 à chaque pas d'optimisation, une taille d'épisode maximale de 200 évènements et selon des réseaux de neurones à 2 couches cachées de 30 neurones chacune, avec activation leaky_relu sur les couches cachées et batch_normalisation avant chaque couche :



(a) Reward en apprentissage (abscisse: nombre d'episodes)



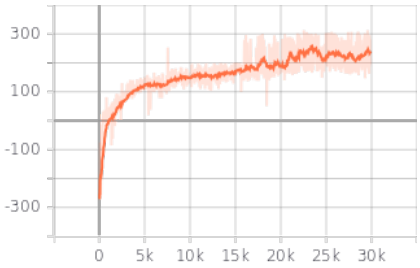
(b) Valeur cible moyenne (abscisse: nombre d'optimisations)



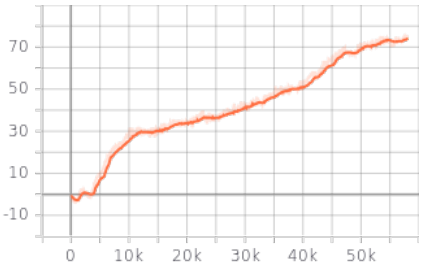
(c) Entropie de l'acteur (abscisse: nombre d'optimisations)

On note un processus légèrement plus lent que DDPG (voir résultats du TP précédent), mais convergeant vers des récompenses moyennes plus élevées. L'adaptativité du poids de l'entropie α permet de maintenir le niveau de stochasticité désiré, soit une entropie moyenne de 0.2.

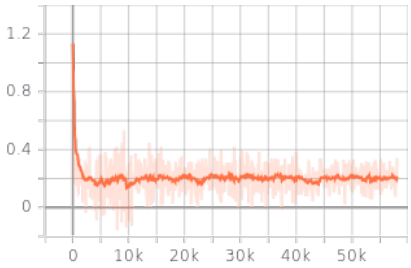
On donne également ci-dessous ce qu'on peut obtenir sur LunarLander avec SAC, toujours utilisant un processus de réglage adaptatif de la température α pour une entropie visée $\mathcal{H}_0 = 0.2$, effectuant une optimisation de 5 étapes tous les 100 évènements, selon deux target networks Q_1 et Q_2 (mis à jour de manière "soft" à chaque optimisation selon des paramètres $\rho = 0.99$), sans réseau V, un discount de 0.99, un pas d'apprentissage de 0.001 pour l'acteur, de 0.003 pour les critiques et de 0.001 pour α, un batch de 1280 transitions échantillonnées dans un buffer de capacité 100000 à chaque pas d'optimisation, une taille d'épisode maximale de 200 évènements en apprentissage (également en test) et selon un réseau de neurones à 4 couches cachées de 64 neurones pour les deux premières et 32 pour les deux suivantes (seulement deux couches de 64 pour les deux réseaux Q_1 et Q_2), avec activation leaky_relu sur les couches cachées et batch_normalisation avant chaque couche, mais sans normalisation des récompenses:



(a) Reward en apprentissage (abscisse: nombre d'episodes)



(b) Valeur cible moyenne (abscisse: nombre d'optimisations)



(c) Entropie de l'acteur (abscisse: nombre d'optimisations)

On peut faire les mêmes analyses que pour Pendulum: plus lent que DDPG mais bien meilleure performance moyenne à partir de 20k episodes d'apprentissage, grâce au maintien d'une entropie moyenne aux alentours de 0.2.

References

[1] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *CoRR*, vol. abs/1702.08165, 2017. [Online]. Available: http://arxiv.org/abs/1702.08165

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290