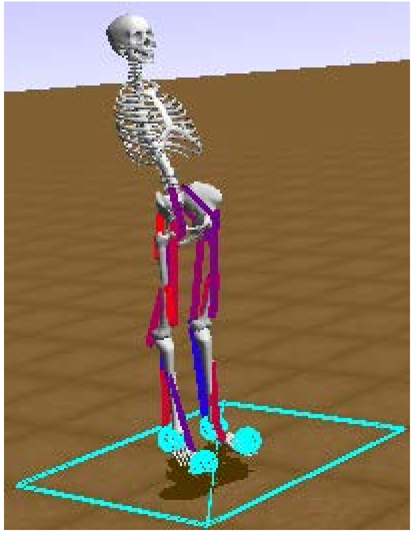
INSTITUT Mines-Télécom



F4BP01B : Projet S5
« Learning to Run »

Endacrants:
M.Fablet
M.Rousseau

MINES Albi-Carmaux MINES Alès MINES Douai MINES Nantes MINES ParisTech Ecole Nationale Supérieure des Mines SAINT-ETIENNE TELECOM Bretagne TELECOM Ecole de Management TELECOM ParisTech TELECOM SudParis EURECOM Sophia Antipolis TELECOM Lille MINES nancy

Plan

- Présentation du sujet
- Etat de l’art
- Implémentation DDPG
- Prochaines étapes

Présentation du sujet

- **NIPS 2017 Challenge. NIPS : conférence et ateliers sur les systemes de traitement de l’information neurale.**
- **But : Apprendre à un model humain à courir avec des obstacles le plus loin et le plus vite possible.**
- **18 entrées (9 muscles par jambe) et 41 variables d’état en sortie**
- **Apprentissage par renforcement**

Etat de l’art

- **Q-learning MAIS espace d’action continu**
- **Problème de stabilité et de convergence**
- **Idée: Utiliser un réseau de neurones tel que propose dans DPG**

Continuous Control with Deep Reinforcement Learning

Theoretical approach

- $\mathcal{S}$: **state space**
- $\mathcal{A}$: **action space,** $\mathcal{A} = \mathbb{R}^N$
- $r(s_t, a_t) \in \mathbb{R}$: **scalar reward for the action** a_t **taken in state** s_t
- $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$: **policy**
- $\mathcal{R}_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$: **sum of discounted future reward**

  with $\gamma \in [0,1]$: **discounting factor**

  and $a_{i+1} \in \pi(s_i, a_i)$

Continuous Control with Deep Reinforcement Learning (2)

- $\mathcal{J}$ **: expected return from the start distribution**

- $\mathcal{J} = \underset{\substack{r_i, s_i, a_i \\ i \geq 1}}{\mathbb{E}} [\mathcal{R}_i]$

- $\mathcal{J} = \mathrm{f}(\pi)$

- **Goal : Learn a policy that maximizes $\mathcal{J}$.**

$$\hat{\pi} = \underset{\pi}{\operatorname{argmax}}\, \mathcal{J}(\pi)$$

Continuous Control with Deep Reinforcement Learning (3)

- Q_π: **action-value function, describes the expected return after taking an action in a state following the policy π**

$$Q_\pi(s_t, a_t) = \mathop{\mathbb{E}}_{\substack{r_i, s_{i+1}, a_{i+1} \\ i \geq t}} \left[\mathcal{R}_t \middle| a_t, s_t\right]$$

Note : In the Q function we consider the expectation criterion rather than the argmax criterion because it is more stable.

- **Bellman equation :**

$$Q_\pi(s_t, a_t) = \mathop{\mathbb{E}}_{r_t, s_{t+1}} \left[r(s_t, a_t) + \gamma \mathop{\mathbb{E}}_{a_{t+1}} [Q_\pi(s_{t+1}, a_{t+1})]\right]$$

Note : We got rid of the dependencies on terms indexed with $i > t + 1$

Continuous Control with Deep Reinforcement Learning (4)

Bellman equation proof:

$$\mathcal{R}_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$$

$$\Rightarrow \mathcal{R}_t = \mathrm{r}(s_t, a_t) + \gamma \sum_{i=t+1}^{T} \gamma^{(i-(t+1))} r(s_i, a_i)$$

$$\Longrightarrow \mathcal{R}_t = \mathrm{r}(s_t, a_t) + \gamma \mathcal{R}_{t+1}$$

$$\Longrightarrow \mathrm{Q}_\pi(s_t, a_t) = \mathop{\mathbb{E}}_{\substack{r_i, s_{i+1}, a_{i+1} \\ i \geq t}} [\mathrm{r}(s_t, a_t) + \gamma \mathcal{R}_{t+1}]$$

$$\Longrightarrow \mathrm{Q}_\pi(s_t, a_t) = \mathop{\mathbb{E}}_{r_t, s_{t+1}} \left[\mathrm{r}(s_t, a_t) + \gamma \mathop{\mathbb{E}}_{\substack{r_i, s_{i+1}, a_i \\ i \geq t+1}} [\mathcal{R}_{t+1}] \right]$$

$$\Longrightarrow \mathrm{Q}_\pi(s_t, a_t) = \mathop{\mathbb{E}}_{r_t, s_{t+1}} \left[\mathrm{r}(s_t, a_t) + \gamma \mathop{\mathbb{E}}_{a_{t+1}} [\mathrm{Q}_\pi(s_{t+1}, a_{t+1})] \right]$$

Continuous Control with Deep Reinforcement Learning (4)

- **Since the target policy μ is deterministic:**
  $\mu : \mathcal{S} \to \mathcal{A}$ **(and not** $\mu : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ **)**

- **We can now simply the Bellman equation by removing the inner expectation** $\mathbb{E}_{a_{t+1}}[Q_\pi(s_{t+1}, a_{t+1})]$ **:**

$$Q_\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}\left[r(s_t, a_t) + \gamma Q_\mu(s_{t+1}, a_{t+1})\right]$$

Note: We can learn Q_μ **using transitions generated from different stochastic behaviour policy. This is why DDPG is an off-policy algorithm.**

Continuous Control with Deep Reinforcement Learning (5)

- **We approximate μ with function approximators parameterized by θ_Q.**
- **The optimisation of the parameters is done by minimizing the loss function defined as follows:**

$$L(\theta_Q) = \mathop{\mathbb{E}}_{s_t, a_t, r_t}\left[\left(Q\left(s_t, a_t \middle| \theta_Q\right) - y_t\right)^2\right]$$

where $y_t = r(s_t, a_t) + \gamma Q\left(s_{t+1}, \mu(s_{t+1}) \middle| \theta_Q\right)$

Note: The dependency of y_t on θ_Q is typically ignored (because we assume that y_t is an observation).

Continuous Control with Deep Reinforcement Learning (5)

- **We approximate μ with function approximators parameterized by θ_Q.**
- **The optimisation of the parameters is done by minimizing the loss function defined as follows:**

$$L(\theta_Q) = \mathbb{E}_{s_t, a_t, r_t}\left[\left(Q\left(s_t, a_t \middle| \theta_Q\right) - y_t\right)^2\right]$$

where $y_t = r(s_t, a_t) + \gamma Q\left(s_{t+1}, \mu(s_{t+1}) \middle| \theta_Q\right)$

Note: The dependency of y_t on θ_Q is typically ignored (because we assume that y_t is an observation).

Continuous Control with Deep Reinforcement Learning (6)

- **To compute large, non-linear function approximators, we use neural networks as described in the DDPG algorithm.**

- **In fact, it is impossible to apply straight forwardly Q-learning because the action space is continuous.**

- **So we use an actor-critic approach based on DPG.**

Continuous Control with Deep Reinforcement Learning (7)

<u>DPG:</u> Deterministic Policy Gradient

- **Critic** $\mathrm{Q}(s, a)$ **is learned using Bellman equation.**
- **Actor** $\mu\left(s \middle| \theta_\mu\right)$ **is updated by following the chain rule:**

$$\nabla_{\theta_\mu} \mathcal{J} \approx \mathbb{E}_{s_t}\left[\nabla_{\theta_\mu} \mathrm{Q}\left(s, a \middle| \theta_Q\right)\Big|_{s=s_t, a=\mu\left(s_t | \theta_\mu\right)} \right]$$

$$\nabla_{\theta_\mu} \mathcal{J} \approx \mathbb{E}_{s_t}\left[\nabla_a \mathrm{Q}\left(s_t, a \middle| \theta_Q\right)\Big|_{a=\mu\left(s_t | \theta_\mu\right)} \nabla_{\theta_\mu} \mu\left(s_t \middle| \theta_\mu\right) \right]$$

Continuous Control with Deep Reinforcement Learning (8)

Resolving stability issues

- **Replay buffer for independent identically distributed samples.**
- **Batch whitening and normalization.**

- $Q\left(s, a \middle| \theta_Q\right)$ **is prone to divergence (Loss equation)**

<u>Solution</u>: Target actor $\mu'\left(s|\theta_{\mu'}\right)$ **and critic** $Q'\left(s, a \middle| \theta_{Q'}\right)$

- **Updated as follows :** $\theta' \leftarrow \boldsymbol{\tau\theta} + (\mathbf{1} - \boldsymbol{\tau})\boldsymbol{\theta}'$

with $\boldsymbol{\tau} \ll \mathbf{1}$

Continuous Control with Deep Reinforcement Learning (9)

Proposed algorithm

- **Exploration policy $\mu_{\mathbf{e}}$ construction :**

$$\mu_{\mathbf{e}}(\boldsymbol{s_t}) = \mu\left(\boldsymbol{s_t}|\theta_{\mu,\boldsymbol{t}}\right) + \mathcal{N}$$

with $\mathcal{N}$: a noise process chosen to suit the environment

- **Expectation approximated by** $\mathbb{E}[X] \approx \frac{1}{N}\sum_{i=1}^{N} X_i$

Continuous Control with Deep Reinforcement Learning (10)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta_Q)$ and actor $\mu(s|\theta_\mu)$ with weights θ_Q and θ_μ
Initialize target network Q' and μ' with weights $\theta_{Q'} \leftarrow \theta_Q$, $\theta_{\mu'} \leftarrow \theta_\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta_\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_t = r_i + \gamma Q_t\left(s_{i+1}, \mu_t\left(s_{i+1}|\theta_\mu\right)|\theta_Q\right)$
    Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta_Q)\right)^2$$

    Update the actor policy using the sampled policy gradient

$$\nabla_{\theta_\mu}\mathcal{J} \approx \frac{1}{N}\sum_i \nabla_a Q(s_i, a|\theta_Q)\big|_{a=\mu\left(s_i|\theta_\mu\right)} \nabla_{\theta_\mu}\mu\left(s_i|\theta_\mu\right)$$

    Update the target networks:

$$\theta_{Q'} \leftarrow \tau\theta_Q + (1-\tau)\theta_{Q'}$$
$$\theta_{\mu'} \leftarrow \tau\theta_\mu + (1-\tau)\theta_{\mu'}$$

  end for
end for

Implémentation DDPG

- Implémentation proposée avec le papier
- Utilisation de Python et Tensorflow
- Récupération du code de Tomàs Völker
- Problème installastion environnement de travail

Prochaines étapes

A moyen terme :

- **Faire fonctionner le programme proposé**
- **Etudier l'influence des paramètres**
- **Etudier la fonction de récompense**

Prochaines étapes

A long terme, deux options :

- **Apprendre à courir avec d’autres algorithmes**

- **Etudier les performance du DDPG sur d’autres applications**

Merci pour votre attention!

Bonnes vacances et joyeuses fêtes!