TECHNISCHE UNIVERSITÄT
CHEMNITZ

Training a deep policy gradient-based neural network with asynchronous learners on a simulated robotic problem

Winfried Lötzsch, Julien Vitay and Fred Hamker

Chemnitz University of Technology
Department of Computer Science
Professorship for Artificial Intelligence

May 8, 2019

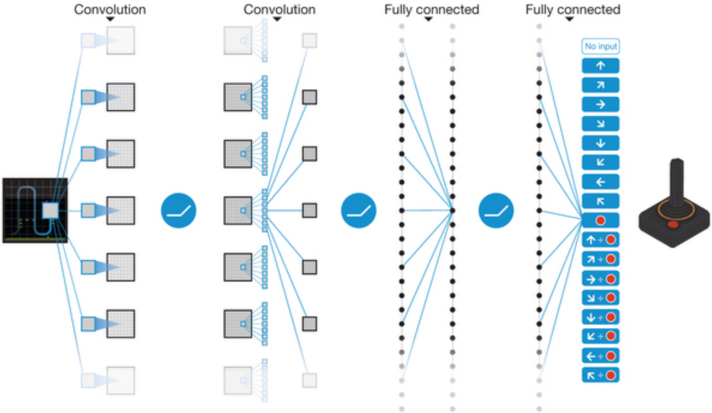
Outline

Outline

Introduction

- Reinforcement learning (RL, Sutton and Barto (1998)) is an important learning method used since decades in many control problems, including robotics, to map states into actions, in order to maximize the amount of reward received on the long-term.
- Non-linear approximators such as deep neural networks suffer from highly temporally correlated training samples and non-stationary objective functions, which are inherent to RL.
- Mnih et al. (2015) proposed a solution to this problem by introducing:
  1. an *experience replay memory*, where episodes are stored and randomly fed in mini-batches to the neural network
  2. *target networks* to increase the stationarity of the objective function.
- This forms a **Deep Q-network** which is able to learn multiple Atari games.

Deep Q-network (Mnih et al., 2015)



Continuous control

- End-to-end deep RL approaches called *policy gradient* methods allow to directly learn continuous policies from high dimensional state spaces.
- Deep Deterministic Policy Gradient methods (DDPG, Silver et al., 2014; Lillicrap et al., 2015) are *actor-critic* architectures, where:
  - the **actor** generates a deterministic policy $\mu_\theta(s)$.
  - the **critic** evaluates an action by estimating its Q-value.
- However, the sample complexity of DDPG is high.
  $\rightarrow$ need for parallel asynchronous learning.

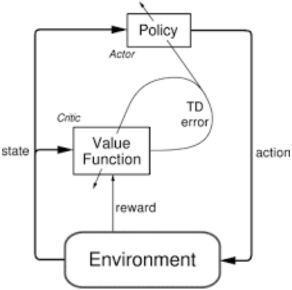
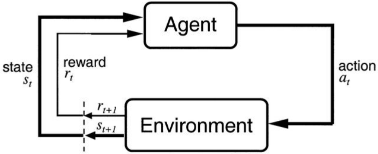


Figure taken from Sutton and Barto (1998), `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html`

Outline

Outline

The agent-environment interface



- The agent and the environment interact at discrete time steps: t=0, 1, ...
- The agent observes its state at time t: $s_t \in \mathcal{S}$
- It produces an action at time t, depending on the available actions in the current state: $a_t \in \mathcal{A}(s_t)$
- It receives a reward according to this action at time t+1: $r_{t+1} \in \Re$
- It updates its state: $s_{t+1} \in \mathcal{S}$

Policies

- The policy is defined as a mapping of the state space into the action space:
  - a **stochastic policy** $\pi_\theta : \mathcal{S} \rightarrow P(\mathcal{A})$ defines the probability distribution $P(\mathcal{A})$ of performing an action.
  - a **deterministic policy** $\mu_\theta(s_t)$ is a discrete mapping of $\mathcal{S} \rightarrow \mathcal{A}$.
- $\theta \in \Re^n$ is a vector of parameters defining the policy, typically the weights of a neural network when using function approximators.
- The policy can be used to explore the environment and generate trajectories of states, rewards and actions.
- The performance of a policy is determined by calculating the *expected discounted return*, i.e. the sum of all rewards received from time step t onwards:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $0 < \gamma < 1$ is the discount rate and r_{t+1} represents the reward obtained during the transition from s_t to s_{t+1}.

Q-value of an action

- The Q-value of an action a is defined as the expected discounted reward if the agent takes a from a state s and follows the policy distribution π_θ thereafter:

$$Q^{\pi_\theta}(s,a) = \mathbf{E}_{\pi_\theta}(R_t | s_t = s, a_t = a)$$

- The goal of the agent is to find the optimal policy maximizing the expected return from every state.
- *Value-based* methods (such as DQN) achieve that goal by estimating the Q-value of each state-action pair.
- The Q-values can be approximated by a deep neural network, by minimizing the quadratic error between the predicted Q-value $Q^{\pi_\theta}(s,a)$ and an estimation of the real expected return R_t after that action:

$$\mathcal{L}(\theta) = \mathbf{E}_{\pi_\theta}[r_t + \gamma Q^{\pi_\theta}(s_{t+1}, a_{t+1}) - Q^{\pi_\theta}(s_t, a_t)]^2$$

Policy gradient methods

- *Policy gradient* methods directly learn to produce the policy (stochastic or not).
- The goal of the neural network is to maximize an objective function

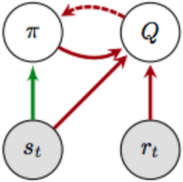
$$J(\theta) = \mathbf{E}_{\pi_\theta}(R_t)$$

- The *stochastic policy gradient theorem* (Sutton et al., 1999) provides a useful estimate of the gradient that should be given to the neural network:

$$\nabla_\theta J(\theta) = \mathbf{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

- A second neural network can be used to learn to approximate the Q-value. Such algorithms are called *actor-critic* architectures, as the actor learns to produce a policy π_θ based on the state alone, while the critic learns to evaluate the Q-value of an action and sends this value to the actor to improve the policy.

Deep deterministic policy gradient - DDPG



- The actor learns a deterministic policy $a_t = \mu_\theta(s_t)$.
- The critic learns the Q-value $Q^{\mu_\theta}(s_t, a_t)$.
- The critic is trained using Q-learning:

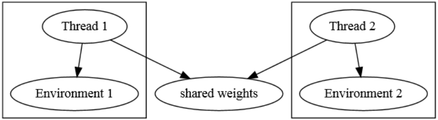
$$\mathcal{L}(\theta_Q) = \mathbf{E}[r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1})) - Q(s_t, a_t)]^2$$

- The actor is trained using the *deterministic policy gradient theorem*:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \mathbf{E}_{\pi_\theta}[\nabla_\theta Q(s, a|\theta)|_{s=s_t, a=\mu_\theta(s_t)}] \\ &= \mathbf{E}_{\pi_\theta}[\nabla_a Q^{\pi_\theta}(s, a)|_{s=s_t, a=\mu_\theta(s_t)} \times \nabla_\theta \mu_\theta(s)|_{s=s_t}]\end{aligned}$$

Lillicrap, T., Hunt, J., Pritzel, A., et al. (2015). Continuous control with deep reinforcement learning. CoRR.

Deep deterministic policy gradient - DDPG



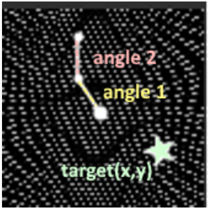
- DDPG requires a lot of samples to converge.
- Mnih et al. (2016) introduced **asynchronously parallel actor-critic** algorithms (A3C) to speed up training.
- Different threads explore the environment in parallel while writing weight updates asynchronously to a single master network.
- They showed that if the threads have different exploration rates, one can use HogWild! weight updates.

Mnih, V., Badia, A., Mirza, M., et al. (2016). Asynchronous Methods for Deep Reinforcement Learning. In Proc. ICML.

Outline

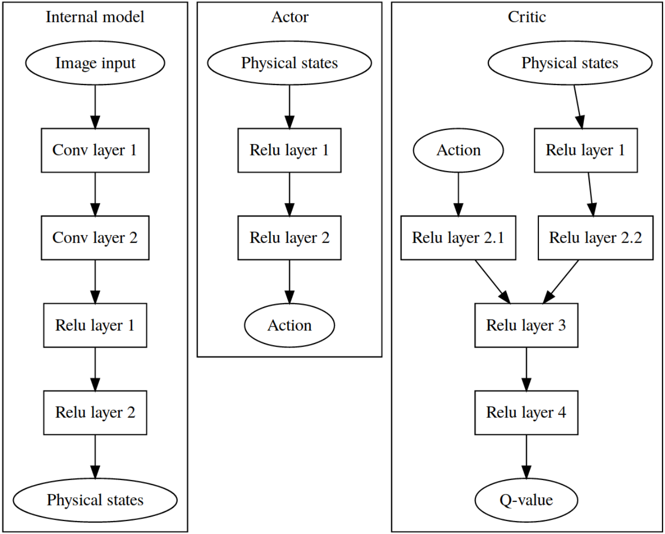
Outline

Simulated robotic arm

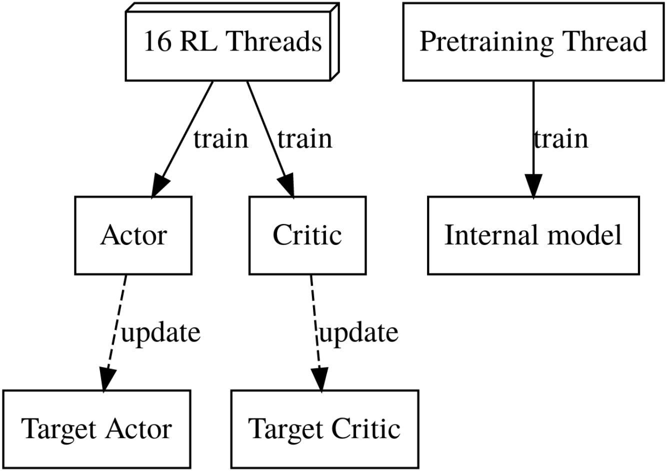


- We designed a simple robotic arm with 2 DOF.
- It is control by two continuous variables θ_1 and θ_2 for the two joints.
- The goal of the agent is to bring the endpoint to a randomly positioned target.
- Some background noise has been added to the image.
- The input to the system is the raw image.

Architecture of the network



Learning procedure


16 RL Threads

Pretraining Thread

train

train

train

Actor

Critic

Internal model

update

update

Target Actor

Target Critic

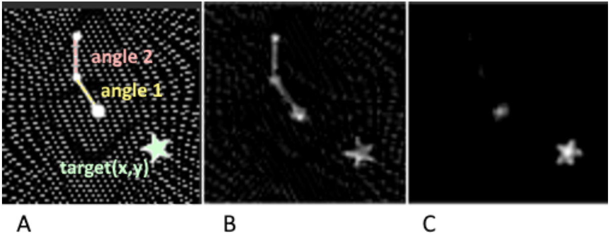
Algorithm for one thread

```
while t < 1000 do
    Execute action $a_t$ according to policy $\mu(s_t) + \epsilon\mathcal{N}$
    Receive reward $r_{t+1}$ and observe new state $s_{t+1}$
    Compute $R_t = \begin{cases} r_{t+1} + \gamma \cdot Q'(s_{t+1}, \mu'(s_{t+1})) \text{ if not terminal} \\ r_{t+1} \text{ otherwise.} \end{cases}$
    Store $(s_t, a_t, R_t)$ in buffer
    if t mod 5 == 0 then
        Update critic: $\mathcal{L}(\theta_Q) = \sum_{i=1}^{5} (R_i - Q(s_i, a_i)]^2$
        Update actor: $\nabla_{\theta_\mu} J(\theta_\mu) = \sum_{i=1}^{5} \nabla_{\mu(s_i)} Q(s_i, \mu(s_i))) \times \nabla_{\theta_\mu} \mu(s_i)$
        Update target critic: $\theta_{Q'} \leftarrow \tau\theta_Q + (1 - \tau)\theta_{Q'}$
        Update target actor:  $\theta_{\mu'} \leftarrow \tau\theta_\mu + (1 - \tau)\theta_{\mu'}$
        Empty buffer
    end if
end while
```

Outline

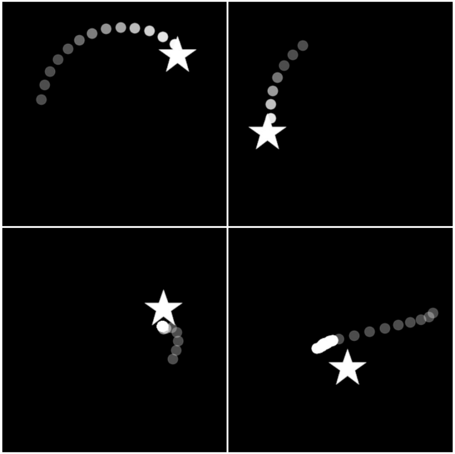
Outline

Activation in the convolutional layers



- The background noise is progressively eliminated in the second convolutional layer.

Examples of successful and unsuccessful trajectories



- Unsuccessful trajectories are mainly due to a wrong estimation of the internal state by the internal model.

Experimental results

Table: Experimental results

Experiment	Amount of successful episodes
without background	57 %
without background, noise added	77.3 %
with background	55 %
only one convolutional layer	38 %
no convolutional layers	27 %

Outline

Outline

Discussion

- We proposed to extend the deep deterministic policy gradient algorithm (Lillicrap et al., 2015) with asynchronous parallel learners (Mnih et al., 2016) to allow end-to-end learning in continuous action spaces from raw pixels.
- We applied this novel algorithm to a simplified continuous control task, with a simulated 2-DOF robotic arm and showed that it is able to achieve a satisfying performance.
- We also further reduced the sample complexity of the algorithm by pretraining an internal model whose role is to transform images into abstract representations.
- The algorithm therefore combines a model-free actor-critic architecture with a model-based internal model.
- However, analysis of failed trials shows that wrong state estimations by the internal model are the main source of failure in our setup.
- Future work will address improving the state representation needed by the actor-critic: intermediate representations could improve the performance of the algorithm.

References

Lillicrap, T., Hunt, J., Pritzel, A., et al. (2015). Continuous control with deep reinforcement learning. *CoRR*.

Mnih, V., Badia, A., Mirza, M., et al. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proc. ICML*.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. In Xing, E. P. and Jebara, T., editors, *Proceedings of ICML*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Beijing, China. PMLR.

Sutton, R. and Barto, A. (1998). *Reinforcement learning: An introduction*, volume 28. MIT press.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Inf. Process. Syst. 12*, pages 1057–1063.