Off-Policy Neural Fitted Actor-Critic

Matthieu Zimmer, Yann Boniface, Alain Dutech

Off-Policy Neural Fitted Actor-Critic

Matthieu Zimmer
University of Lorraine
LORIA, UMR 7503
Nancy, F-54000, France
matthieu.zimmer@loria.fr

Yann Boniface
University of Lorraine
LORIA, UMR 7503
Nancy, F-54000, France
yann.boniface@loria.fr

Alain Dutech
INRIA
LORIA, UMR 7503
Nancy, F-54000, France
alain.dutech@loria.fr

Abstract

A new *off-policy*, *offline*, *model-free*, *actor-critic* reinforcement learning algorithm dealing with continuous environments in both states and actions is presented. It addresses discrete time problems where the goal is to maximize the discounted sum of rewards using *stationary* policies. Our algorithm allows to trade-off between *data-efficiency* and *scalability*. The amount of *a priori* knowledge is kept low by: (1) using neural networks to learn both the critic and the actor, (2) not relying on initial trajectories provided by an expert, and (3) not depending on known goal states. Experimental results compare *data-efficiency* to 4 state-of-the-art algorithms on three benchmark environments.

This article largely reproduces a previous work [34] by adding a higher dimensional environment, improving control architectures and provides batch normalization for others state-of-the-art algorithms.

1 Introduction

Reinforcement learning (RL) is a framework for solving sequential decision problems, in which an agent interacts with its environment and adapts its policy based on a scalar reward signal [27]. RL agents can autonomously learn difficult tasks, like playing video games [19]. While the basic setting of RL is currently well established, fully continuous environments for both state and action spaces need new algorithms to solve more real-world problems. In many realistic tasks, like robotics, it is time-consuming and costly to produce data. RL agents should thereby exhibit good data-efficiency, *i.e.* exploiting each sample as best as possible, even at the cost of a longer computational time.

The purpose of this work is to design an RL algorithm that: (1) tackles continuous state and action spaces, (2) is data-efficient, and (3) uses neural networks to be as generic as possible with minimal *a priori* knowledge.

Recently, several RL algorithms for fully continuous environments have been developed with neural networks control architectures [18, 24]. However, they were focused on task performance rather than data-efficiency since they are *model-free* and data were not too costly to produce. Seeking for data-efficiency usually means to use *model-based* algorithms, like Probabilistic Inference for Learning COntrol (PILCO) [3]. However, PILCO lacks scalability [31] and *model-based* algorithms do not always lead to straightforward improvements when using neural networks [8].

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

In this work, we present an *offline*, *model-free*, *off-policy*, *actor-critic* RL algorithm that allows a trade-off between scalability and data-efficiency. It is based on the *fitted actor-critic* family [1, 33] and benefits from the improvements proposed by Deep Deterministic Policy Gradient (DDPG) [18].

2 Background

We are interested in RL problems, modeled as *Markov Decision Processes* (MDP) $\langle S, A, T, R \rangle$, where the state space S and the action space A are continuous. The goal is to seek for an *optimal* policy π^* maximizing the expected discounted reward:

$$\pi^* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \times R(s_t, \pi_t(s_t))\Big], \tag{1}$$

where t denotes a time step and $0 < \gamma < 1$ is the discount factor.

When the state space S is continuous, classical value-function methods like Least-Squares Temporal Difference (LSTD) [2] rely on an estimation of $Q : S \times A \to \mathbb{R}$, the sequential values of actions in each state:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a\Big], \tag{2}$$

where r_t is the reward obtained at time t from R following π. Being *data-efficient* means to search for the best policy given the collected samples. An example of a neural *data-efficient*, *critic-only* algorithm is Fitted Q Iteration (FQI) [5, 21], which updates the Q function several times using the Bellman operator as an approximated version of Value Iteration [12]. Instead of iterating over all states and actions, it relies only on the collected samples $(s_t, a_t, r_{t+1}, s_{t+1})$:

$$Q_{k+1} = \arg\min_{Q \in \mathcal{F}} \sum_{t=1}^{N} \Big[Q(s_t, a_t) - \big(r_{t+1} + \gamma \max_{a' \in A} Q_k(s_{t+1}, a')\big)\Big]^2. \tag{3}$$

When the action space A is continuous, the use of an *actor* (*i.e.* a parametric policy) becomes crucial to overcome the complexity of the argmax search. This often leads to *actor-only* methods like Policy Gradient [28], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10] or Trust Region Policy Optimization (TRPO) [24]. The major drawbacks of *actor-only* methods are the high variability of the cost J (because there is no *critic*) and, for gradient-based methods, the plateau effect and local minima that can lead to poor policies [7, 17]. On the other hand, *actor-critic* algorithms try to combine both the advantages of previous methods. The critic learns a value function thus reducing the variability of the approximation of J and the actor learns the parametric policy, allowing the use of continuous actions.

We now present three state-of-the-art *actor-critic* algorithms that we will use for comparison in our experiments (from least to most *data-efficient*):

- Continuous Actor Critic Learning Automaton (CACLA) is a successful *actor-critic* algorithm [30] that uses neural networks for both the critic and the actor. Due to it's online nature and its *on-policy* updates, it cannot achieve good data efficiency (the collected data is used then forgotten). In some environments, CACLA performs better than CMA-ES [29].
- Neural Fitted Actor Critic (NFAC) may achieve a better data efficiency than CACLA since it uses FQI updates [33]. However, the data is forgotten after each end of episode because the actor features *on-policy* update.
- Deep Deterministic Policy Gradient (DDPG) is also an *actor-critic* algorithm [18]. It accomplishes online updates of the policy and Q function, and it reuse previous samples through its *off-policy* update. Based on Neural Fitted Q with Continuous Actions [9], DDPG is more scalable due to online updates, targets networks [19] and batch normalization [13]. The target networks serve to slow down the weights updates to increase the stability of learning, by soft updating a copy of the policy and the value function.

Recently, two new methods have been proposed to increase the efficiency of some RL algorithms.

- When the dimensions of action space A are bounded, instead of limiting the output of the neural policy with a last layer (for instance with a hyperbolic tangent) that squashes the gradient obtained from the critic, it is preferable to have an unbounded last layer with an adapted gradient strategy [11].
- Retrace(λ) is a new strategy to weight a sample for *off-policy* learning [20], it provides low-variance, safe and efficient updates.

3 Algorithm

Our algorithm, that we name Data Efficient Neural Fitted Actor Critic (DENFAC), can be seen as a neural version of a *fitted actor-critic* (FAC) algorithm [1]. It contains an approximated version of both Value and Policy Iteration (for the critic and the actor respectively).

The critic is updated with a FQI update where the argmax operator is replaced by the policy choice. Moreover, the policy is able to change at each update to approximately fit what would be the argmax.

$$Q_{k+1} = \underset{Q \in \mathcal{F}_c}{\text{argmin}} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}) \in \mathcal{D}} c(s_t, a_t) \Big[Q(s_t, a_t) - \Big(r_{t+1} + \gamma Q_k(s_{t+1}, \pi_k(s_{t+1})) \Big) \Big]^2, \quad (4)$$

$$\pi_{k+1} = \underset{\pi \in \mathcal{F}_a}{\text{argmax}} \sum_{s_t \in \mathcal{D}} Q_{k+1}\big(s_t, \pi_k(s_t)\big), \quad (5)$$

where $c(s_t, a_t) = min\big(1, \frac{\pi_{k-1}(a_t|s_t)}{\pi_b(a_t|s_t)}\big)$ is the weight associated to a sample [20], and π_b is the policy that gathered the sample. This coupled optimization can be applied multiple times without acquiring new samples.

DENFAC is an *off-line* algorithm, therefore the execution part of one episode consists only of performing the policy choices and collecting the samples $(s_t, a_t, r_{t+1}, s_{t+1})$ that are added to $\mathcal{D}$ (the replay buffer). The *off-line* part is depicted in Algorithm 1. The algorithm is *data-efficient* because it performs a type of FQI. Furthermore, unlike DDPG, it performs updates over the largest set of data given a computational constraint. This might requires too much computational time so the *data-efficiency* vs *scalability* dilemma can be adjusted through the length of $\mathcal{D}$. Another meta-parameter of Algorithm 1, $reset\_critic$ that reset the weight of the critic to another initial solution, can lead to an even better *data-efficiency* by allowing the critic to get out of local minima.

A challenging problem in this algorithm is how to handle the growth of $\mathcal{D}$. In this work, we consider $\mathcal{D}$ as a First-In First-Out (FIFO) queue. So the agent accesses to its memory of the latest episodes. It can be enough in some simple environments but it is clearly sub-optimal in a *data-efficiency* point of view. Defining a weight associated to each data of $\mathcal{D}$ from the δ-error might be a solution [23]. However, in contrast to [23] the δ-error depends not only on the critic but also of the actor. It is not clear yet if the sampling from $\mathcal{D}$ should be the same for the actor and the critic.

DENFAC learns a deterministic policy, thus during the execution part an exploration strategy must be used. It can greatly influence the *data-efficiency*. We do not address this issue in this work. In the experimental setup, each algorithm uses the same exploration strategy : a Gaussian noise is added to the actor choice.

	FAC	DDPG	NFAC	DENFAC
Offline & Batch	×		×	×
Off-policy	×	×		×
Fitted Critic	×		×	×
Actor updated through ∇Q	×	×		×
Learn Q	×	×		×
Reset Networks			×	×
Retrace				×
Target Networks		×		
Batch Normalization		×		×

Figure 1: Properties of the nearest actor-critic algorithms : FAC [1], DDPG [18] and NFAC [33].

Data: $\mathcal{D}$ replay buffer of N samples, Q_0 value-function, π_b previous policies, K number of fitted iteration, G number of gradient descent for actor updates, $inverting\_gradient$ strategy, $reset\_critic$ strategy
Result: π_K the next policy to play, Q_K the next value function
for $k \leftarrow 1$ **to** K **do**
  for $(s_t, a_t, u_t, r_{t+1}, s_{t+1}) \in \mathcal{D}$ **do**
    $q_{k,t} \leftarrow \begin{cases} r_{t+1}, & \text{if } s_{t+1} \in S^* \\ r_{t+1} + \gamma Q_{k-1}(s_{t+1}, \pi_{k-1}(s_{t+1})), & \text{otherwise} \end{cases}$
  end
  $Q_k \leftarrow$ randomly initialize critic network **if** $reset\_critic$ **else** Q_{k-1}
  Update critic by minimizing the loss:

$$L = \frac{1}{N} \sum_{t=1}^{N} min\big(1, \frac{\pi_{k-1}(a_t|s_t)}{\pi_b(a_t|s_t)}\big) \Big(q_{k,t} - Q_k(s_t, a_t)\Big)^2$$

  Randomly initialize actor network π_k
  for $g \leftarrow 1$ **to** G **do**
    Update the actor policy using the batch gradient:
    if *inverting_gradient* **then**

$$\nabla_a = \nabla_a . \begin{cases} (a_{max} - a)/(a_{max} - a_{min}) & \text{if } \nabla_a < 0 \\ (a - a_{min})/(a_{max} - a_{min}), & \text{otherwise} \end{cases}$$

    end

$$\nabla_{\theta^{\pi_k}} \pi_k = \frac{1}{N} \sum_{t=1}^{N} \nabla_a Q(s_t, a)|_{a=\pi_k(s_t)} \nabla_{\theta^{\pi_k}} \pi_k(s_t)$$

  end
end

Algorithm 1: Data Efficient Neural Fitted Actor Critic (DENFAC)

4 Experimental Setup

An experimental comparison of DENFAC, DDPG, CMA-ES, NFAC and CACLA is done into three environments: Acrobot [26], Cartpole [22] and Half-Cheetah [32].

In Acrobot (double swing-up), the reward function is defined as (1) +1 if the goal is reached (arm straight up), (2) the normalized max height of end effector if 500 steps are reached, and (3) 0 otherwise.

In Cartpole (inverted pendulum), the reward function is defined as (1) 0 when the cart position is between $[-0.05; 0.05]$ and the pole angle between $[-\frac{\pi}{60}, \frac{\pi}{60}]$, (2) $-2 \times (500 - last\_step)$ if it exits at $last\_step$ (pole angle $\notin [-\frac{\pi}{6}, \frac{\pi}{6}]$ or cart position $\notin [-2.4; 2.4]$), and (3) -1 otherwise.

In Half-Cheetah, the reward function is $R(s, a) = v_x(s) - 0.05 \cdot \|a\|_2^2 - 1 \cdot g(s)$ where $v_x(s)$ is the speed of the cheetah on x axis and $g(s)$ is 1 if the heel, the or 0 otherwise.

The discount factor is fixed to $\gamma = 0.9$ (Acrobot) and $\gamma = 0.99$ (Cartpole and Half-Cheetah). States are composed of the joint positions/angles and joint position/angle velocities. Dimensions of $S \times A$ are 4×1 (Acrobot), 4×1 (Cartpole) and 20×6 (Half-Cheetah).

The neural networks use (1) Adam learning algorithm [16], (2) the leaky rectified linearity (ReLU) [6], and (3) batch normalization [13]. Critic networks contain 2 hidden layers of 50 and 7 neurons. The structure of the actor networks is fixed to : $1 \times 5 \times 1$ units (Acrobot), $1 \times 20 \times 1$ units (Cartpole), and $20 \times 20 \times 10 \times 6$ units (Half-Cheetah). The last layer of the critic and actor networks is linear. The actor policy is a truncated Gaussian policy between $[a_{min} = -1, a_{max} = 1]$ with $\sigma = 0.05$ where the mean is determined by the output of the last linear layer. Each weight is initialized from a normal distribution $\mathcal{N}(0, 0.01)$. Batch normalization is applied on each layer for both the actor and the critic, except on the last two layers.

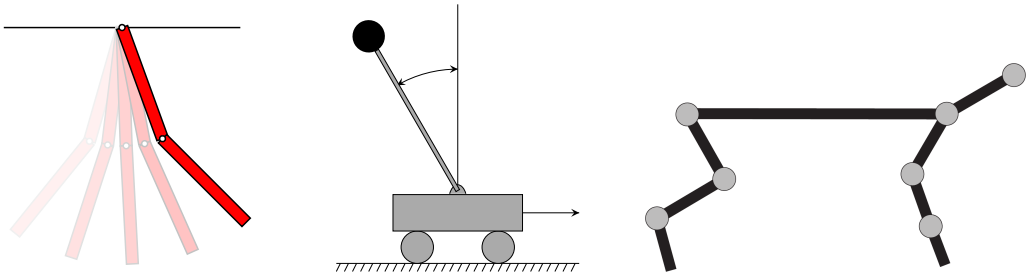

Figure 2: Illustration of Acrobot (left), Cartpole (middle) and Half-Cheetah (right) environments (reproduced from Wikipedia and [32]).

For each experimental setup, we first optimize all the meta-parameters of DDPG and then apply them to DENFAC. To obtain a fair comparison, we also optimized the number of updates performed by DDPG, and we applied the inverting gradient strategy (when it was better) to make it more data-efficient. NFAC and CACLA algorithms are improved with batch normalization, denoted as NFAC+ and CACLA+ in Figure 3. Only CMA-ES do not use batch normalization for its policy as it does not rely on the gradient.

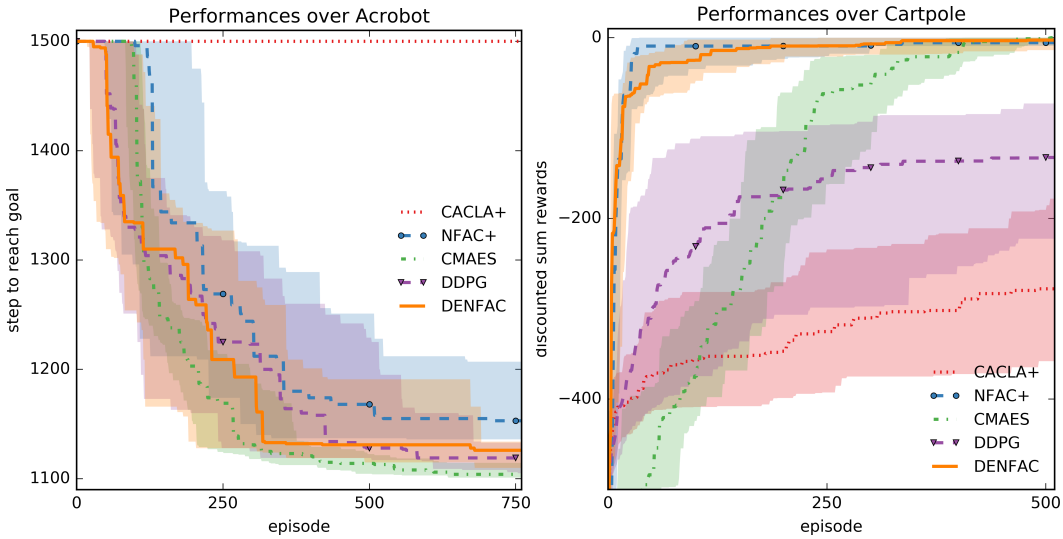

Figure 3: Median and quartiles of the best registered performance in Acrobot (the lower, the better), Cartpole (the higher, the better) and Half-Cheetah (the higher, the better) environments during RL learning with each algorithm. Each experiment has been run 40 times for statistical results.

In order to characterize *data-efficiency* on Figure 3, we plot the best performance the agent has done since the beginning. It can only improve therefore it does not represent the exploration made by each algorithm. The number of data collected is correlated to the number of episode.

Figure 3 shows that DENFAC quickly develops good policies on each tasks compared to DDPG (even if DDPG is online). However, unexpectedly DENFAC, which is off-policy, has the same order of performance that NFAC+, which is on-policy. It let us think that either $\mathcal{D}$ should not be a simple FIFO queue, or that the policy update of NFAC+ allows a better exploration.

Surprisingly, the *actor-only* method CMA-ES achieves very good *data-efficiency* on those environments. In higher dimensional environments, it can lack scalability [4] but here it outperforms online algorithms like CACLA or DDPG even if they have access to the reward information faster (online algorithms).

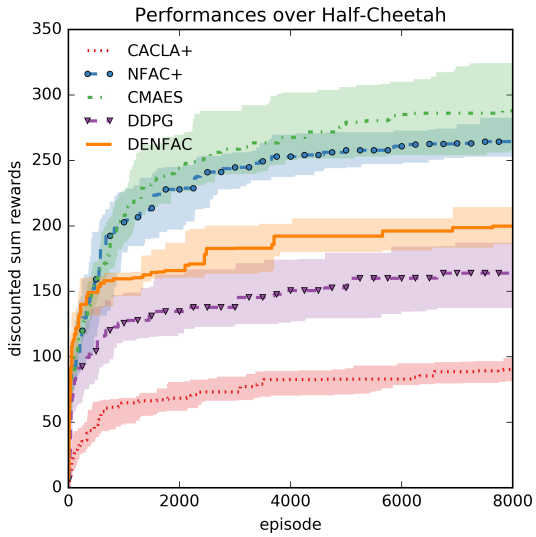

Figure 4: Median and quartiles of the best registered performance Half-Cheetah (the higher, the better) environments during RL learning with each algorithm.

		α_a α_c	inverting gradient	mini batch size	τ	additional updates	K	G	batch size $\mathcal{D}$	reset critic
Acrobot	DDPG	0.1	No	64	0.001	8				
	DENFAC	0.1	No				10	25	6000	Yes
Cartpole	DDPG	0.1	Yes	64	0.1	8				
	DENFAC	0.1	Yes				10	25	3000	No
Half-Cheetah	DDPG	0.1	Yes	64	0.001	8				
	DENFAC	0.1	Yes				10	25	3000	No

Figure 5: Best meta-parameters found for DDPG and DENFAC.

CACLA or CACLA+ cannot reach the goal in only 750 episodes on Acrobot. On those environments, it's the less *data-efficient* algorithm but also the faster in computational time. It proves that having a replay buffer like NFAC, DDPG and DENFAC helps to improve *data-efficient*.

We did not notice that adding a L2 regularization term in the critic, as done in the original version of DDPG, improves it in those environments. In some experiments, we also run our algorithm in an *online* setting or with target networks, but this did not improve the data-efficiency, while requiring more computations (results not shown here).

5 Conclusions and further work

We investigated the *data-efficency* vs *scalability* dilemma in three fully continuous environments. *Data-efficency* often implies more computational time spent on each data impeding the scalability. In some cases, resetting the weights of the neural networks shows even more *data-efficency*. All those additional costs must be negligible compared to the cost of producing data in the environment otherwise such methods are not appropriate. DENFAC is more data-efficient than 4 state-of-the-art *actor-critic* algorithms in some environments but comes at a higher computational cost. To further improve DENFAC, it should be analyzed how to replace the FIFO queue for $\mathcal{D}$ and if uniform sampling could be improved [23]. Moreover, DENFAC lacks stability in learning, target networks did not help, slowing down the change in the policy might increase his stability [24].

Acknowledgments

The data has been numerically analyzed with the free software package GNU Octave [15]. We used Caffe as neural network library [14] and Open Dynamic Engine (ODE) as physic engine [25]. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] András Antos, Rémi Munos, and Csaba Szepesvari. Fitted Q-iteration in continuous action-space MDPs. 2008.

[2] Steven J. Bradtke, Andrew G. Barto, and Pack Kaelbling. Linear least-squares algorithms for temporal difference learning, 1996.

[3] Marc Deisenroth and Carl E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pages 465–472, 2011.

[4] Yan Duan, Xi Chen, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *arXiv preprint arXiv:1604.06778*, 2016.

[5] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Aistats*, volume 15, page 275, 2011.

[7] Ivo Grondman, Lucian Buşoniu, Gabriel AD. Lopes, and Robert Babuška. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man and Cybernetics*, 42(6):1291–1307, 2012.

[8] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv preprint arXiv:1603.00748*, 2016.

[9] Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84(1-2):137–169, 2011.

[10] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[11] Matthew Hausknecht and Peter Stone. Deep Reinforcement Learning in Parameterized Action Space. *arXiv preprint arXiv:1511.04143*, 2016.

[12] Ronard A. Howard. Dynamic Programming and Markov Processes. 1960.

[13] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[15] S\oren Hauberg John W. Eaton David Bateman and Rik Wehbring. *{GNU Octave} version 4.0.0 manual: a high-level interactive language for numerical computations*. 2015.

[16] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2015.

[17] Vijay R. Konda and John N. Tsitsiklis. Actor-Critic Algorithms. *Neural Information Processing Systems*, 13:1008–1014, 1999.

[18] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Others. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[20] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. *arXiv preprint arXiv:1606.02647*, 2016.

[21] Martin Riedmiller. Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method. In *Lecture Notes in Computer Science*, volume 3720 LNAI, pages 317–328, 2005.

[22] Martin Riedmiller, Jan Peters, and Stefan Schaal. Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 254–261, apr 2007.

[23] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *arXiv*, pages 1–23, 2015.

[24] John Schulman, Sergey Levine, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *International Conference on Machine Learning*, page 16, 2015.

[25] Russell Smith. Open dynamics engine. 2005.

[26] Mark W. Spong. Swing up control problem for the acrobot. *IEEE Control Systems Magazine*, 15(1):49–55, 1995.

[27] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998.

[28] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, pages 1057–1063, 1999.

[29] Hado Van Hasselt. Reinforcement Learning in Continuous State and Action Spaces. In *Reinforcement Learning*, pages 207–251. Springer Berlin Heidelberg, 2012.

[30] Hado Van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

[31] Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From Pixels to Torques: Policy Learning with Deep Dynamical Models. *arXiv preprint arXiv:1502.02251*, 2015.

[32] Paweł Wawrzyński. Learning to control a 6-degree-of-freedom walking robot. In *International Conference on Computer as a Tool*, pages 698–705, 2007.

[33] Matthieu Zimmer, Yann Boniface, and Alain Dutech. Neural Fitted Actor-Critic. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2016.

[34] Matthieu Zimmer, Yann Boniface, and Alain Dutech. Toward a data efficient neural actor-critic. In *13th European Workshop on Reinforcement Learning*, 2016.