Reinforcement Learning for Humanoid Robot Control

Marek Danel[1]

[1]Faculty of Information Technology
Czech Technical University
Thákurova 9, 160 00 Praha, Czech Republic

danelmar@fit.cvut.cz

Abstract. *In this paper, we describe results of application of reinforcement learning on full body control of a humanoid robot. We start with a simple task of achieving vertical position of robot's torso. We use an actor-critic neural network architecture, which is well established approach for reinforcement learning continuous action policies. Our experimental setup includes an instance of the NAO robot in the Webots simulation environment and custom adaptation components for Keras-rl reinforcement learning framework. We present minor modifications of original algorithm and discuss several encountered challenges of applying deep reinforcement learning methods to humanoid robot control.*

Keywords

Reinforcement learning, NAO, humanoid robot, actor-critic, DDPG, neural networks

1. Introduction

Nowadays, there are humanoid robot bodies available with high degree of movement freedom. They are also equipped with sensors, which provide large amount of input from robot's environment in multiple modes as well as input from robot's own body. This immense amount of input data and freedom of actions poses a challenge of fully exploiting robot's potential.

In real-world environment, any movement is subject to unpredictable deviations. When performing fixed sequences of movements, a robot will end up in a very different final position every time due to cumulating inaccuracies of every step in the sequence. Therefore, many subtle and significant changes to originally planned sequence need to be made continuously to compensate for stochasticity of the real-world environment. A feasible control policy needs to translate sensory readings to appropriate action in every time step.

Complexity of such a policy makes it very time consuming to engineer. Thus, machine learning algorithms are suitable tools for this task. Finally, it is impossible to predict all situations a truly autonomous robot can encounter in real-world. Therefore, ability to learn and improve policy is necessary feature of the solution we look for.

We decided to build upon recent advancements of reinforcement learning. In this paper, we describe our results of application of Deep Deterministic Policy Gradient (DDPG) method [1] on a full humanoid body control task. We present several challenges of reinforcement learning (RL) application encountered and discuss possible solutions. The goal is to reach vertical position of robot's torso. The specific device we use in our experiments is robot NAO. We also show some minor improvements over baseline Keras-rl [2] implementation that promote learning stability in our task.

2. Background

In the field of reinforcement learning, we model agent in its environment as a Markov Decision Process (MDP). MDP consists of state space S, action space A, transition dynamics given by probability density $P(s_{t+1}|s_t, a_t)$ and a reward function $R(s, a)$. By small letters s and a we denote state and action respectively and subscript them with t and $t+1$ whenever the distinction between subsequent time steps is necessary. In every time step, agent observes the state of the MDP, picks an action according to its policy π and receives a reward determined by the reward function. The policy π is defined as probability of taking an action given the state agent has observed $\pi = P(a|s)$. Alternatively, policy can be constrained to a deterministic form. Then, it is simply a function of state that returns action to be performed $a_t = \pi(s_t)$. In our work, we use only the deterministic form of a policy.

We assume full observability of the environment. Under this assumption, the *state* and its *observation* are equal. They are often used interchangeably and in the rest of the paper we use both these terms.

The DDPG algorithm we apply to humanoid robot control is a model free approach that learns a policy by function approximation. The function approximation is used to evaluate utility (or value) of actions in given state expressed as a single real value. This evaluation function is called action value function and it is always denoted by $Q(s, a)$. When the utility of actions is known, it is possible to pick the most useful one. Another commonly used term is a state value function $V(s)$, which evaluates utility of a state. The relationship between the action value function and the state value function is:

$$V(s) = max_a Q(s,a) \tag{1}$$

2.1. Actor-critic agent and how it learns

The DDPG algorithm utilizes an actor-critic neural network architecture. The actor network implements the deterministic policy. The critic approximates the action value function $Q(s,a)$ and the actor network is trained to maximize the value of actions it outputs with respect to the action value function the critic approximates. The mathematical formulation of the actor is therefore $argmax_a Q(s,a)$.

The critic network is trained to minimize a mean squared temporal difference error. Temporal Difference (TD) learning is an approach for learning an action value estimation that considers long-term payoff. An action is valuable for the immediate reward received upon taking it and also for all the rewards received in future for which taking that specific action was critical. The TD value of an action is defined as exponentially weighted sum of future rewards:

$$Q(s_t, a_t) = \sum_{i=t}^{\infty} \gamma^i r_{t+i} \tag{2}$$

where $\gamma \in (0,1)$ is a discount factor that controls preference of close or distant outcomes. The action value of this form can be interpreted as an expected discounted cumulative reward or as an expected discounted return.

An optimal policy picks the most valuable action in each state. For such a policy, the Bellman Equation holds:

$$Q(s_t, a_t) = r_t + \gamma Q(s_{t+1}, a_{t+1}) \tag{3}$$

We take this relationship as an objective function for training critic network. The approximation error Y_t in a time step t can be obtained by a simple modification:

$$Y_t = Q(s_t, a_t) - (r_t + \gamma Q(s_{t+1}, a_{t+1})) \tag{4}$$

Since calculation of the error includes the approximated function $Q(s,a)$ itself, the training bootstraps from $Q(s,a) = 0$ or a random distribution of values close to zero.

3. Related Work

The most relevant work is the DDPG [1] algorithm by DeepMind, which we apply to a new task. It is a combination of actor-critic network architecture with replay buffer and target network techniques taken over from the previous success of Deep Q-Network (DQN) approach [3]. In other words, it is an extension of DQN to continuous action domain.

A very close alternative to DDPG is the approach called Normalized Advantage Functions (NAF) [4]. It is another extension of DQN to continuous action tasks, but it works in a different way. In every state DQN outputs value per action. However, in continuous action space possible actions are uncountable, so output must take form of a function. This is roughly what NAF does. NAF decomposes action-value function $Q(s,a)$ to a sum of state value $V(s)$ and advantage function $A(s,a)$. The advantage function further decomposes to $aL(s)L(s)a$ where L is a lower-triangular matrix obtained as output from the Q-Network. Both algorithms have been applied to robotic arms with 7 degrees of freedom both in simulation and on real devices by Gu et al. [5]. The attempted tasks were door opening and pick & place. Experiments show successful learning of both algorithms with NAF yielding slightly better results, both in simulation and real devices. However, collection of sufficient experience for learning in the real-world is very time consuming.

To speed up collection of experience in the real-world environment, authors set up several devices to attempt the same task at once. Every device acts in its own copy of the environment. This approach to dealing with experience collection bottleneck has worked well also for Levine et al. [6] for the object grasping task. For our simulated NAO environment, this method is also relevant, as the simulation speed is fixed to real-time.

4. Approach

4.1. Experimental setup

Our experimental setup consists of the Webots simulator [7] hosting a single NAO robot instance. We start with Keras-rl implementation [2] of the DDPG algorithm [1]. To integrate the robot simulation with learning backend framework we define custom environment class compatible with OpenAI Gym [8] environment's interface. A block diagram of the architecture and data flow is presented on figure 1.

Observation of the environment includes:

- 25 joint angles that are to be reached and maintained by servomotors.
- 25 actual measured joint angles
- 3 real values of gyroscope
- 3 real values of accelerometer
- 8 feet force sensors readings

Some more sensory data are supported but are not available in the Webots simulator (e.g. joint absolute current values). Image from two cameras is also available, but we do not use this input yet.

The action vector $a \in (-1,1)^{25}$ is meant to define the percentage of maximum torque to be applied to rotating each of 25 joints in positive or negative direction. However, NAO's API does not allow to manipulate the current let to its servomotors directly. There is a feedback control mechanism built-in that regulates the current let into servomotors so that they maintain a specified position of the joint. Our

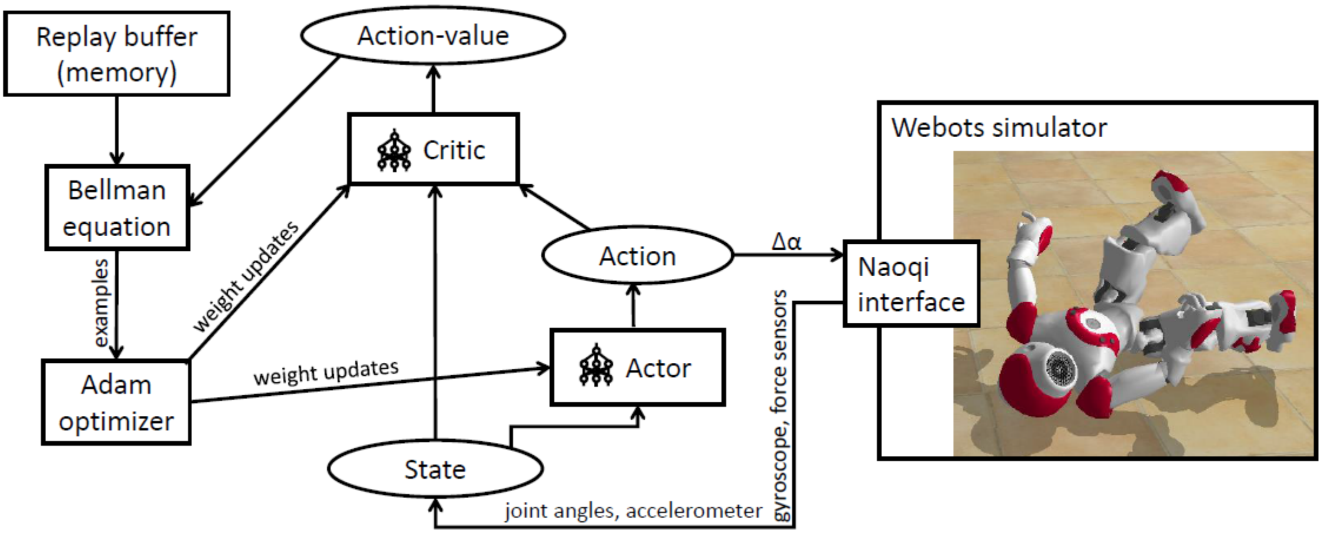

Fig. 1: Block diagram of experimental setup. Boxes denote software components, ellipses denote data.

policy can control the angles maintained and also a parameter of every joint called stiffness. Stiffness is a value in range $[0, 1]$ and its meaning is a portion of maximum current allowed to be used by thefeedback control for maintaining join's desired position.

To simulate direct control of the torque in the joints, we set the stiffness of an i-th joint to $|a_i|$ and set a new angle to be reached and maintained by the built-in feedback controller. We determine a change in the maintained angle $\Delta\alpha_i$ by following relationship:

$$\Delta\alpha_i = \omega_{max,i} * a_i * \Delta t \qquad (5)$$

Where the $\omega_{max,i}$ is the maximum rotation velocity of the i-th joint (with full torque) and Δt is a duration of the action. This way we make sure that joint is moving during the whole interval Δt and reaches its destination angle right at the end of the action time frame.

4.2. Task

The goal is to achieve a vertical posture of robot's torso. We measure how much a posture is vertical by readings from accelerometer located in torso. In a vertical posture the acceleration on Z axis of the accelerometer is approximately -9.8 ms^{-2} which equals gravitational acceleration of the earth. The reward function $r(s_t, s_{t+1})$ is defined for every transition from state s_t to state s_{t+1} as the difference of the vertical acceleration (See equation 6). We denote an acceleration measured on axis Z in state s_t as $AccelZ(s_t)$.

$$r(s_t, s_{t+1}) = -(AccelZ(s_{t+1}) - AccelZ(s_t)) \qquad (6)$$

A reward function like this makes the get up task particularly suitable for testing reinforcement learning algorithms, since non-zero rewards are available for nearly all transitions. This makes this task significantly easier compared to tasks with sparse rewards, where non-zero rewards are many steps away from each other. Temporal difference learning then requires more iterations to propagate action-values through distant states.

Although the chosen task is in many aspects well suited for early experiments, it is still quite difficult. The goal state is distant and the sequence of correct actions needed to achieve it is long. The robot needs to support its weight by his limbs through major part of the movement. This introduces problem of choosing reasonable balance between exploration and exploitation. An agent can learn effects of actions not yet examined by taking random actions. On the other hand, it can make progress in reaching the goal by following the learned deterministic policy.

4.3. Exploration

We use ϵ-*greedy* exploration policy with dynamically managed ϵ parameter. In every time step, ϵ-*greedy* agent performs random action with probability ϵ and follows learned policy with probability $1-\epsilon$. We propose to set probability by following formula:

$$\epsilon = \frac{1}{max(1, V(s))} \qquad (7)$$

The idea behind this formula is that when predicted discounted reward gain is high, then it pays off to follow the learned policy. The agent will either complete the task or make partial progress and proceeds with exploration when the predicted future reward gain decreases. There is a possibility that following the learned policy does not yield the expected cumulative reward. This can happen when the generalization is incorrect due to insufficient sampling of state and action space. Then, following the learned policy is still beneficial as an exploration mechanism.

In the equation 7 we have shown how we compute the random action probability ϵ for a state evaluated by $V(s)$ for simplicity. However, with an actor-critic agent the explicit

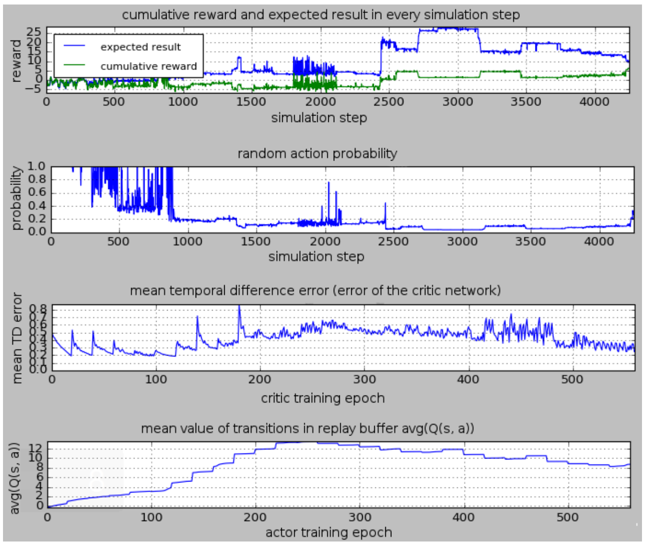

Fig. 2: Figure shows metrics of a learning process. The first plot shows how vertical position robot was in at every time step of training (cumulative reward). It also shows the predicted outcome of learned policy (expected result). The second plot shows the current exploration rate. Third plot shows absolute error of training critic. The last two plots show training metrics of the actor and critic networks.

implementation of the value function $V(s)$ is not available. The available functions of state are action-value function $Q(s, a)$, which is approximated by critic network, and the policy function $argmax_a Q(s, a)$ implemented by actor network. By composing these two terms we can get an equivalent formula that is ready to be implemented.

$$\epsilon = 1/max(1, Q(s, argmax_a Q(s, a))) \tag{8}$$

5. Results

Figure 2 shows metrics of one learning process. As we can see on the first plot, the learned policy manages to maintain score slightly above zero. The goal of vertical position was not achieved yet. In following paragraphs, we discuss issues discovered in learning record and suggest possible solutions.

The record of initial learning attempt on Figure 2 shows unsatisfactory learning schedule. As the number of collected experience grows, the critic network clearly does not converge. A quick fix could be rising the fixed number of learning iterations. However, rising this number too much could result in excess iterations and waste of time when learning on new evidence converges quickly. Choosing an optimal value would take multiple attempts and the chosen value would be suitable for only one specific task. More sophisticated and reliable approach is to stop training when the approximation error stops improving. This can be done by watching sliding mean and variance of the first derivative of learning curve. Learning can be stopped when mean tangent of learning curve is higher than given threshold and when change of variance in two subsequent sliding windows is below given threshold.

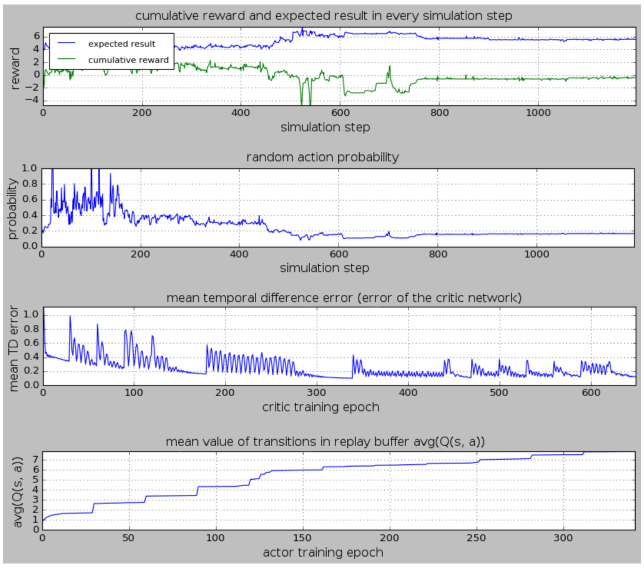

Fig. 3: Overestimation of impossible action seriously restricts exploration.

5.1. Overestimation

Another issue apparent on Figure 2 is overestimation of the expected cumulative reward. This can be seen well on first plot. For example, on 3000-th step you can see predicted outcome of 25, which is even impossible to achieve while gravitational acceleration of earth is about 9.8.

Overestimation is usually credited to inaccuracy of approximation function and it is hurting policy performance [9]. This phenomenon occurs consistently even with much more fine trained networks then on Figure 2. In certain learning configurations, the overestimation even amplifies by the critic-actor training loop. Error then grows to infinity and learning process fails.

Another shortcoming brought by overestimation is getting stuck in position constrained by joint's angle limitations. Figure 3 shows example of learning process that has converged to policy that overestimates an impossible move. Usual way to deal with constraints in motion through state space in reinforcement learning is restarting often to the initial position and performing many rather short episodes. However, natural learning of humans and animals happens without any restarts. Therefore, we see exploring alternative approaches as a suitable course of research.

Overestimation is a common issue of Q-learning algorithms in general. Hasselt et al. [10] have proposed solution called Double Q-learning, which was also applied for Deep Q-network algorithm [9]. Double Q-learning decouples action evaluation from selection during learning by using separate parameter sets θ and θ' for both tasks.

$$Y_t = r_{t+1} + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a; \theta); \theta') \tag{9}$$

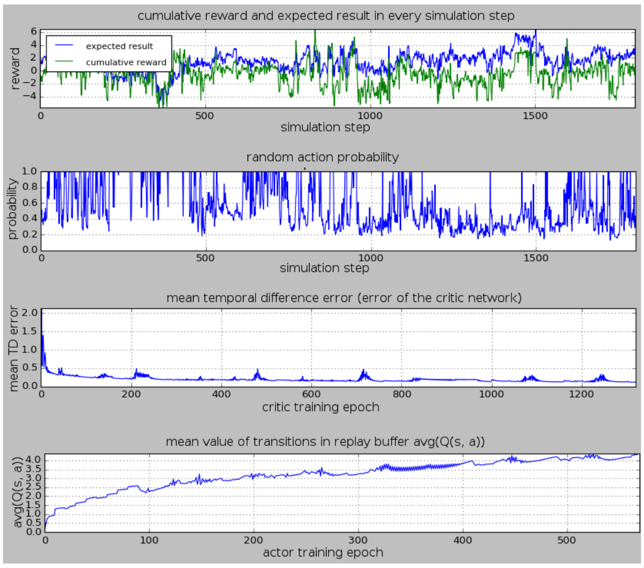

Fig. 4: Learning process metrics of the same agent with larger networks.

This variation of Bellman equation describes reference value Y_t for training a Q-network. The adaptation for actor-critic network may be very straightforward, since actor network actually approximates the $argmax_a Q(s_{t+1}, a)$ portion of formula. However, this approach needs to be experimentally evaluated.

Much quicker way to reduce overestimations is learning with larger networks. Figure 4 shows learning metrics with network size increased to 4 layers of 200 neurons using tanh activations instead of relu. This learning episode also starts from pre-collected experience of roughly 4500 steps. We can see convergence to much less residual error of critic network and significantly lower overestimations. Although, they did not disappear completely yet. Nevertheless, new learned policy was able to achieve maximum cumulative reward of about 6, which is a significant improvement.

The frequent oscillations in random action probability (ϵ) are a good sign, as they correspond well with expected gain for following learned policy. The robot follows the learned policy and reaches a more vertical position. At the boundary of sampled state space the predicted reward gain gets low. Then, the robot takes random action, usually falls and the scenario repeats.

6. Conclusion

We have built an experimental setup for deploying deep reinforcement learning algorithms on humanoid robot in real-world simulation environment. We have conducted several experiments and addressed encountered challenges of applying reinforcement learning to a new task. Our results indicate, that successful learning of humanoid body control requires more experience collected together with either carefully picked network size, or an implementation of Double Q-learning.

Acknowledgment

I thank Miroslav Skrbek for supervision of this work. This research has been supported by CTU grant SGS17/213/OHK3/3T/18.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[2] M. Plappert, “keras-rl,” https://github.com/matthiasplappert/keras-rl, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[4] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q-learning with model-based acceleration,” *CoRR*, vol. abs/1603.00748, 2016. [Online]. Available: http://arxiv.org/abs/1603.00748

[5] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation,” *CoRR*, vol. abs/1610.00633, 2016. [Online]. Available: http://arxiv.org/abs/1610.00633

[6] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” *CoRR*, vol. abs/1603.02199, 2016. [Online]. Available: http://arxiv.org/abs/1603.02199

[7] Webots, “http://www.cyberbotics.com,” commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[9] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[10] H. V. Hasselt, “Double q-learning,” in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf

About Authors...

Marek DANEL

Finished bachelor degree in 2014 and master degree in software engineering in 2016, both at FIT, CTU in Prague. In 2016 he started his doctoral study on behavior modeling for robotics and started working in IBM Watson R&D Prague the same year.

DANEL Marek.: *Dialogový systm využívající znalostní báze*. Master’s thesis, Czech Technical University, Faculty of Information Technology, 2016.