J. Hlaváčová (Ed.): ITAT 2017 Proceedings, pp. 71–77
CEUR Workshop Proceedings Vol. 1885, ISSN 1613-0073, © 2017 M. Danel, M. Skrbek

ITAT

Humanoid Robot Control by Offline Actor-Critic Learning

Marek Danel, Miroslav Skrbek

Faculty of Information Technology
Czech Technical University
Prague, Czech Republic
danelmar@fit.cvut.cz, skrbek@fit.cvut.cz

Abstract: In this paper, we present our results on application of reinforcement learning on full body control of a humanoid robot. The task we try to learn is achieving vertical position of robot's torso from an initial position of laying flat on the ground. Our experimental setup includes an instance of the NAO robot in the Webots simulation environment. We use an actor-critic neural agent. As this is a work in progress, we only train offline from a sample of random movements. We present a series of experiments on a simplified task and a final evaluation on the humanoid robot control task that shows improvement over random policy.

Keywords: reinforcement learning, humanoid robot, actor-critic, offline learning, 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 policy makes it very time consuming to engineer using model-based methods. Also, it is impossible to predict all situations a robot can encounter in real-world. Ability to learn by reinforcement from the environment is necessary for true autonomy. Therefore, we aim to deploy and adapt reinforcement learning algorithms to humanoid robotics.

Deploying reinforcement learning algorithms to a humanoid robot is a difficult task as the dimensionality of state and action space is high. Robot NAO, which we use in our experiments, has 25 degrees of freedom. The robot should learn to control its body by directly applying torque to its actuators. This means that all the dynamics of its body are responsibility of the learning algorithm. As we have shown in our previous work [1], deploying the Deep Deterministic Policy Gradient Algorithm (DDPG) [2] yields very little results out of the box.

In this work, we evaluate applicability of actor-critic agents to humanoid body control. We focus on testing the capability of actor-critic method to infer the correct action from available experience. We first compare multiple common perceptron optimization methods to get best possible results on a simplified task. Then, we use the findings from simplified task to set up an experiment with the robot in a simulated environment, where we show an improvement over random policy baseline.

2 Background

In the field of Reinforcement Learning (RL), 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 neural actor-critic algorithms are model free approaches that learn 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 func-

tion $V(s)$, 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 the error of action value prediction. 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. Most often, the 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 inspiration we draw from the DDPG [2] algorithm by DeepMind, which was our starting point. This algorithm was already applied to control of robotic arms with 7 degrees of freedom by Gu et al. [3].

Our previous work we tried to apply the DDPG algorithm to the humanoid robot control task, which has shown little success [1]. We continue this work by exploring deeper possible causes of failure. A major one is most likely the regression accuracy. The mean error of the critic network was around 0.7. Considering sample standard deviation of reward values of around 0.62 this is likely to seriously hurt the learning process while combined with Bellman style propagation of discounted long-term rewards.

The most closely related work to ours is probably an example of learning a humanoid robot to stand up from sitting on a chair by Iida et al. [4]. They also control directly joint torques and use an actor-critic architecture, but with radial basis networks and Temporal Difference critic updates. This means that their critic network approximates the state value function. Its update rule is stated in equation 5.

$$\nabla V(s_t) = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{5}$$

Unlike Sun and Roos [5] or Tutsoy et al. [6], we do not use animation primitives or inverse kinematics to lower dimensionality of the task. We want to impose no restrictions to the freedom of movements created by learning algorithm. This makes the problem much more challenging and the solution more useful if we are successful. We also do not want to rely on prior knowledge of body dynamics like inverse kinematics. Relying on the model of the specific device makes the method bound to it and makes the solution on the accuracy of the mathematical model. On the other hand, a model-free Reinforcement Learning method has potential to find perfectly working behaviors even for a damaged device with altered dynamics.

Beside the method we explore, there are Evolution Strategies (ES) and Policy Gradient (PG) methods that have also been successful in optimizing policies with continuous state and actions. Our method itself could be called a to PG method as it improves the policy by following its estimated gradient. However, it does so in quite an unusual way compared to classic PG algorithms, as it derives the gradient from a learned action-value $Q(s,a)$ function. Therefore, it is also a Q-Learning approach. The PG and ES methods are all competitive with the DDPG algorithm we started with and they deserve careful evaluation of applicability to NAO too.

Policy Gradient methods use a stochastic policy to generate a batch of randomized trajectories, then propagate discounted returns to all recorded actions and update the policy parameters in the ascent direction of likelihood of greater returns. The Trust Region Policy Optimization algorithm (TRPO) [7], a state of the art PG algorithm, uses Fisher information matrix for parameter scaling independency [8] and Kullback–Leibler divergence to normalize the magnitude of gradient ascent steps.

Salimans et al. [9] achieve comparable results to TRPO in MuJoCo physics simulator locomotion environments. Their approach is an Evolution Strategy, that represents the population by a factored gaussian distribution over weights of the neural network, which implements the policy. As black box optimization methods, Evolution Strategies work only with parameters perturbations and sampled values of the objective function, which is the result of an episode. The lack of per-transition evaluation of the deci-

sions made by the policy is the main difference from PG methods [8]. It also makes them generally less sample efficient. However, they are more suitable for parallelization [9].

4 Approach

In following experiments, we use actor-critic neural network architecture with offline learning from fixed sample of random trajectories. We focus on testing the capability of actor-critic method to infer the correct action from available experience. A sample of trajectories generated by random policy should be enough to infer first few moves from the initial state. A random policy is a policy that samples random actions uniformly over the action space. Having the trajectories pre-generated also speeds up our experiments.

We train without propagating long-term effects by bootstrapping critic with the Bellman equation, as it could be an undesirable source of variance. Therefore, the critic in this case implements a straightforward regression of rewards from state and action pairs, which should be sufficient for tasks in this paper. We implement both networks as multilayer perceptrons with one hidden layer. We compare results of stochastic gradient descend, ADAM optimizer [10] and LBFGS optimizers and some hyperparameters to find a setup that yields stable results.

The expected outcome is a policy that makes a *reliable* partial progress towards moving the robot to the upright posture. The fixed sample of random movements makes it unlikely to accomplish the whole task, as the samples of state and action are concentrated around the starting point and will be missing further along the unknown optimal trajectory.

4.1 Artificial Task

We start parameter exploration with a simpler artificial task to lower computational costs, and make it possible to perform more exhaustive search. More benefits of evaluating on an artificial task include elimination of noise and availability of known optimal policy.

The artificial task is defined as follows:

- *state* - real vector $s \in \mathbb{R}^d$. Represents agents position in a d-dimensional space. Dimensionality d is a parameter of the task.
- *action* - a real vector $a \in \mathbb{R}^d$ of values in range $\in [-1,1]$. Represents a direction and distance in which the agent wants to move.
- *transition* - $s_{t+1} = s_t + a_t$.
- *reward function* - $r(s,a) = ||s|| - ||s+a||$.

An episode ends when $\exists c \in s; c > 3$, which is considered a failure. The agent has breached the boundary of allowed space. As a successful completion of the task is considered when the agent gets to the zero-position closer than a threshold value: $||s|| < 0.001$.

4.2 Getting up 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. To complete this task the robot does not need to stand on its feet necessarily. Any stable sitting posture is enough. 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 $AccelZ(s_t)$.

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

The location of accelerometer in robot's body is shown at the figure 2.

A specific characteristic of this task is no implicit division to episodes. The robot can make some progress towards the goal. Whenever it makes a wrong move, it will fall back to the initial position or some state along the way. This effect of gravity exposes the agent to new states and it is desirable to let the agent go on trying to learn independence of initial position.

We suppose that this nature of the task makes it unfavorable for propagation of discounted return by bootstrapping critic with Bellman equation. After a few rewarding actions, the robot will be in partially upright position, where any imprecise movement can cause fall. While exploring such states with random policy, there will be a strong imbalance between subsequent actions that bring more progress and rewards, and actions that cause the whole progress to be lost. Bootstrapping too early, before the subsequent correct actions have been sampled, will cause incorrect inference of long-term action utility. Therefore, we have decided to train only the immediate reward signal prediction with the critic network.

4.3 Experimental Setup

All experiments take place in Webots simulator with single instance of robot NAO. Figure 1 shows a block diagram of the whole experimental setup for robotic task. 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

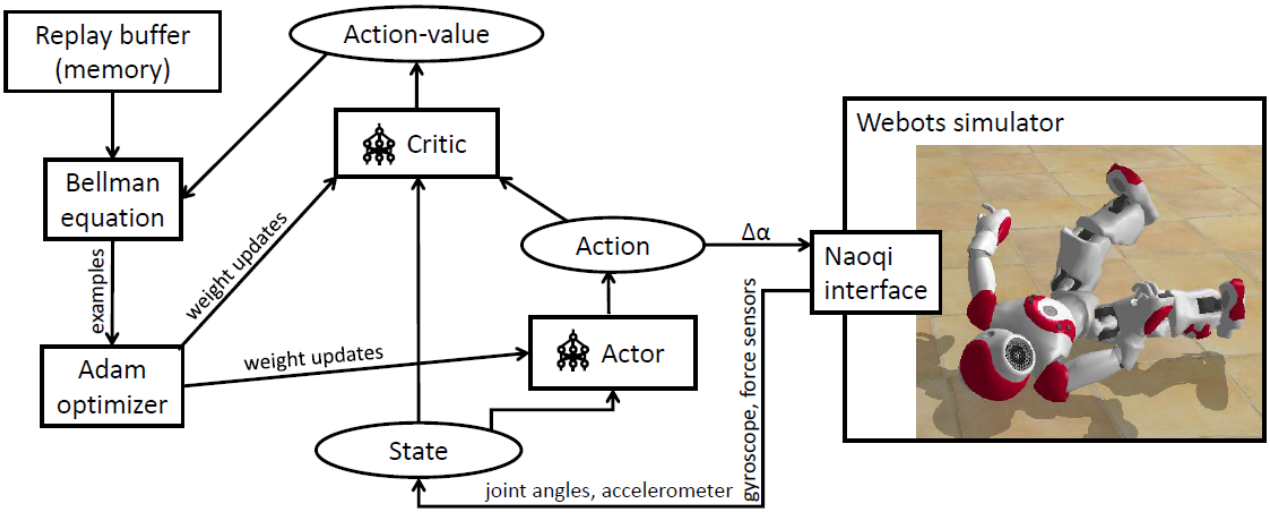

Figure 1: Block diagram of experimental setup. Boxes denote software components, ellipses denote data.

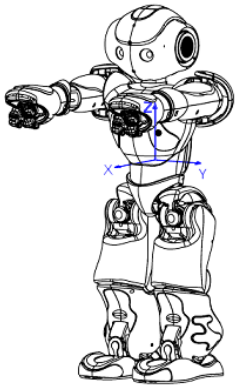

Figure 2: Location of accelerometer in robot's body and its coordinate system.

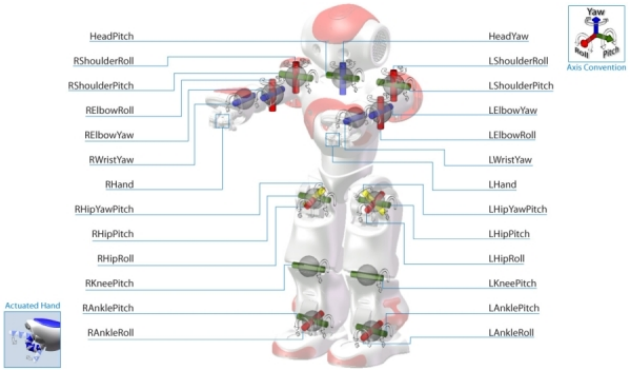

Figure 3: Joints with their degrees of freedom marked.

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.

In this environment, the policy controls directly the torque applied to all 25 actuators in 13 joints. An illustration of robot's joints and their degrees of freedom presented on the figure 3. The action vector $a \in (-1,1)^{25}$ specifies the fraction of maximum torque to be used in positive or negative direction.

The implementation of action in this form for NAO was not straightforward as NAO's API has a built-in feedback control mechanism that maintains specified angles of robot's joints. The API allows to set new posture to be maintained and a torque limit. To simulate direct torque control, we set the torque limit for every joint and compute the change in the maintained angle of i-th joint $\Delta\alpha_i$ by following relationship:

$$\Delta\alpha_i = \omega_{max,i} * a_i * \Delta t \tag{7}$$

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 a joint is being moved with the specified force during the whole interval Δt and reaches its destination angle right at the end of the action time frame.

4.4 Performance Metrics

There are several ways to measure how successful the learned policy is. The more important of all metrics is observation by a human expert. Only human expert is able to tell whether what the robot is doing makes sense. However, it is impossible to evaluate many trained models this way as the time of human expert is precious. Therefore, numerical measures are necessary to compare policies with each other to select a few suitable for evaluating further.

A simple sum of collected rewards is not very informative metric for this task. Recall, that the reward function returns the difference in accelerometer readings between subsequent states. Then, the sum of collected rewards up to timestep t is equal to accelerometer reading in time

Table 1: Action errors achieved with different optimizer combinations

algorithm	critic optimizer	actor optimizer	mean action error mean	mean action error sd	mean maximum action error	maximum action error sd
optimal policy			0.000		0.000	
actor-critic	lbfgs	sgd	0.073	0.2250	0.263	0.3524
actor-critic	lbfgs	adam	0.096	0.1001	0.337	0.2335
actor-critic	adam	sgd	0.134	0.2181	0.381	0.3890
actor-critic	lbfgs	lbfgs	0.188	0.1571	0.532	0.2282
actor-critic	adam	adam	0.199	0.2073	0.558	0.4048
actor-critic	adam	lbfgs	0.272	0.2175	0.648	0.3215
random policy			0.886	0.04277	1.494	0.2632
worst possible policy			1.670	0.01607	2.000	

step t plus the accelerometer reading in the initial state. Suppose there are two policies π_1 and π_2. The policy π_1 reliably achieves half of rewards that lead to target states and falls 10 times during the testing episode. The policy π_2 reliably achieves half of rewards that lead to target states and falls 100 times during the testing episode. Then policy π_1 is obviously better and it is likely to end up with higher score than π_2. However, it is certainly not guaranteed as the final posture of both policies is random variable.

The maximum cumulative reward is also not very informative measure. More precisely, by maximum cumulative reward we mean the highest sum of rewards collected up to some time step of a testing episode. See equation 8 for formal specification.

$$max_t(\sum_{i=0}^{t} r(s_i, a_i)) \tag{8}$$

This metric does not capture well the stability of learning. Poorly trained models that result in very variable policies can lead the robot near to the target posture occasionally without really learning anything. Therefore, we expect high variance of measurements of this metric.

The metric we consider most informative for the getting up task is mean cumulative reward over all time steps of the test episode. We define the mean cumulative reward as:

$$\frac{\sum_{t=0}^{n}(\sum_{i=0}^{t} r(s_i, a_i))}{n} \tag{9}$$

is number of time steps where n is number of time steps of test episode. In other words, this metric is an average verticality of robot's posture weighted by time spent in that posture. This metric is favorable to policies that can make progress and also can maintain it or recover quickly after fall.

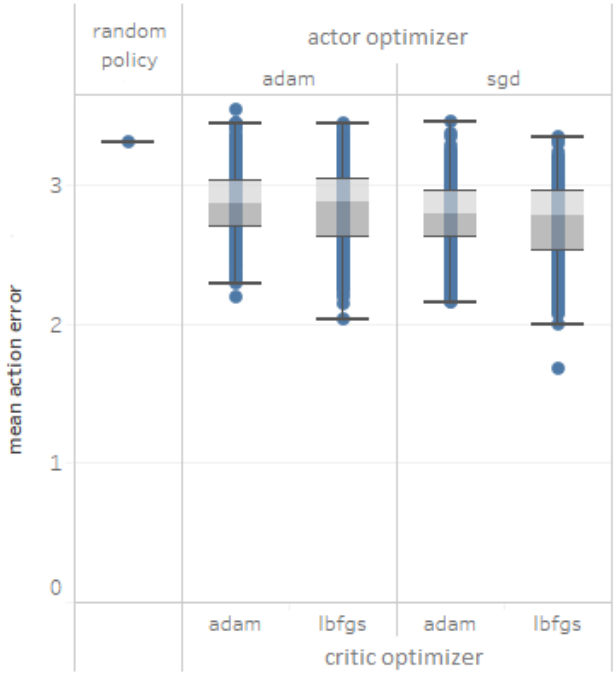

Figure 4: Comparison of action errors achieved with different optimizers for critic and actor respectively for artificial task in 10 dimensions. Random policy as a baseline.

5 Results

5.1 Artificial Task

A first experiment is presented in Table 1. We have compared different optimization methods of actor and critic networks for artificial task of 1 dimension. The presented measures are mean and maximum action error during testing episode. All results are averaged over 40 training attempts. The error of an action is computed as a magnitude of difference from known optimal action. For critic and actor 4 hidden units were sufficient for this task. Adding more does not improve performance. The compared optimization methods are ADAM, Limited memory Broyden–Fletcher–Goldfarb–Shanno (lbfgs) and stochastic gradient descend with annealed learning rate.

We include optimal policy, random policy and worst possible policy for comparison. Optimal policy can be easily derived from the artificial task definition. The worst

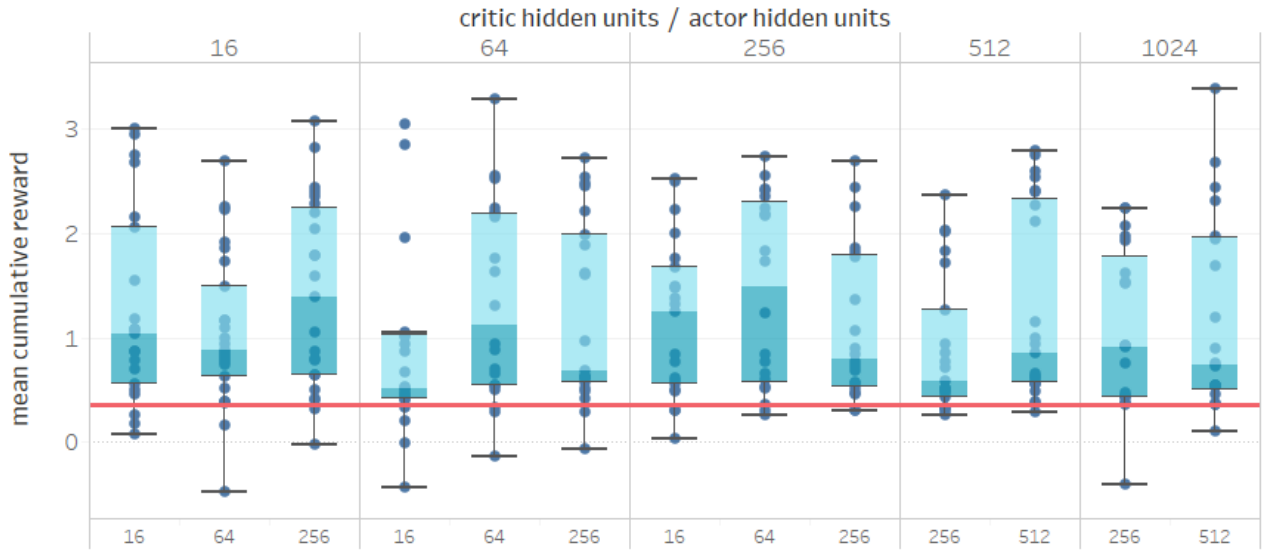

Figure 5: Comparison of results achieved with different critic and actor hidden layer sizes. The red line marks average performance of random policy.

policy can be derived from the same way as the optimal policy and it is the exact opposite of it. The worst policy computes the worst action available in each state. Random policy samples actions uniformly over the action space.

We can see that all configurations produce policies strongly biased towards the optimal policy. In further experiments, we use the top two configurations.

Figure 4 shows the same comparison with artificial task in 10 dimensions. The number of hidden units we needed for these results was 160 for critic and 80 for actor. Adding more hidden units brought only little improvement for considerable computation time cost. Also, number of random action samples was increased from 4000 to 12000. Results on 10-dimensional artificial task also show improvement over random policy. However, the bias towards the optimal policy is significantly weaker compared to 1 dimension. Therefore, we expect even smaller difference between random policy and learned policy for the robot control task.

5.2 Getting up Task

Figure 5 show comparison of results achieved with different number of units in one hidden layer of critic and actor networks. For every configuration, we show mean cumulative reward over testing episode of 20 trained models. This is a very early experiment, so we start from small neural networks to save computation time. All configurations achieved average score much higher than random movements policy. However, the variance between policies trained with the same configuration is high and there can be found policies that perform worse than random. For the best configuration of actor of size 64 and critic of size 256 the improvement over random policy can be confirmed by two sample t-test (p-value: 0.00026).

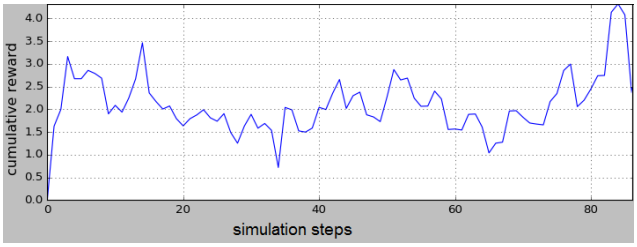

Figure 6: Cumulative reward per simulation step during an example testing episode.

An example of progress made by a trained policy during a testing episode is shown in the Figure 6. This policy achieved mean cumulative reward score of 2.17. The robot made a quick progress in getting off the ground and then was able to maintain it with oscillations. The posture usually reached with the well-trained policy is shown in Figure 7. The robot was able to support its weight with arms and push itself from the ground a bit. Since the first few moves seem to be learned well, it looks reasonable to gather another batch of experience by performing random actions after those few successful steps. An exploration schedule like this would almost certainly end up in a local optimum. Nevertheless, it would be an interesting result in this environment we suspect to be very noisy.

6 Conclusion

We have shown a partial progress in learning humanoid robot control by actor-critic approach. The improvement over random movements policy is clear but rather small. Experiments with artificial task of varying dimensionality show strong performance decrease in higher dimensions despite more data provided. Results show, that more

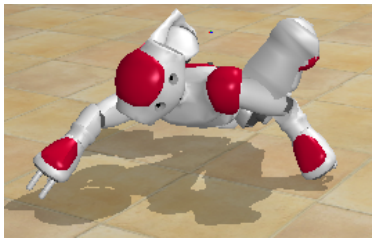

Figure 7: A reliably reached posture along the path to getting up from the ground.

precise hyperparameter search could bring some improvement of unknown magnitude. Therefore, feasibility of offline actor-critic learning for humanoid robot body control remains an open question.

Acknowledgment

This research has been supported by CTU grant SGS17/213/OHK3/3T/18. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures"

References

[1] M. Danel, "Reinforcement learning for humanoid robot control," in *POSTER*, May 2017.

[2] 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

[3] 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

[4] S. Iida, S. Kato, K. Kuwayama, T. Kunitachi, M. Kanoh, and H. Itoh, "Humanoid robot control based on reinforcement learning," in *Micro-Nanomechatronics and Human Science, 2004 and The Fourth Symposium Micro-Nanomechatronics for Information-Based Society, 2004.*, Oct 2004, pp. 353–358.

[5] Z. Sun and N. Roos, "An energy efficient dynamic gait for a nao robot," in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, May 2014, pp. 267–272.

[6] O. Tutsoy, D. E. Barkana, and S. Colak, "Learning to balance an nao robot using reinforcement learning with symbolic inverse kinematic," *Transactions of the Institute of Measurement and Control*, vol. 0, no. 0, p. 0142331216645176, 0. [Online]. Available: http://dx.doi.org/10.1177/0142331216645176

[7] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[8] F. Stulp and O. Sigaud, "Policy improvement: Between black-box optimization and episodic reinforcement learning," 2013.

[9] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," *ArXiv e-prints*, Mar. 2017.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980