Experimental Evaluation of Deep Q-Learning Applied on Pendulum Balancing

Zvezdan Lončarević, Rok Pahič, Gregor Papa, Andrej Gams

All authors are with the Jožef Stefan Institute,
and with the Jožef Stefan International Postgraduate School,
Jamova cesta 39, 1000 Ljubljana, Slovenia
e-mail: zvezdan.loncarevic@ijs.si

Abstract

Autonomy is one of the central issues for the future robots that are expected to operate in continuously changing environments. Reinforcement learning is one of the main approaches for learning in contemporary robotics. With the rise of neural networks in recent studies, the idea of incorporating neural networks with classic Q-learning algorithm for learning policies was introduced in a form of deep Q-Learning algorithm. While supervised and unsupervised learning became widely spread within community, deep Q-Learning still remains a black-box in a sense of parameter tuning as well as neural network architecture and training.

In this paper we explore and compare training performance using different parameters and different neural network architectures on a simple use-case of pendulum balancing.

1 Introduction

Reinforcement learning (RL) is a popular way of solving optimization problems in robotics through trial-and-error interaction with the environment. This relieves humans from tedious programming. Planning of actions is possible for solving decision making problems with known and determined dynamics as shown in [1, 2]. However, as this is not always the case, RL is applied to help in finding solutions without having detailed description of the problem and is useful for systems with complex dynamics where it is not possible for all the disturbances and external forces to be modelled [3]. This model-free reinforcement algorithms were successfully applied on different types of problems [4] and with the expansion of neural networks extended variety of its application [5, 6]. However, architecture of the neural network, training strategy and high number of parameters that need to be tuned for each specific task diminish benefits of theoretically reduced need for manual engineering.

In real-world domains experience must be collected on real physical systems. By using simulations and understanding the influence of parameters and training strategies as well as possibilities of RL algorithms, it would be possible to optimize the real world systems to learn optimal policies in less iterations thus causing minimal wear of the equipment and reducing the needed time.

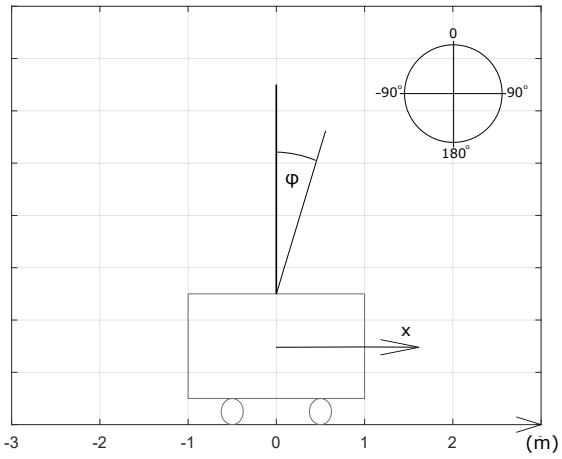

Figure 1: Simulated cart pole used as the experimental environment in MATLAB

The goal of this paper is to show the influence of parameters on the learning process so we used simple inverted pendulum attached to the cart pole (Figure 1) that was powered by discrete accelerations.

The paper is organized as follows: In the next section, we briefly present Deep Q-learning algorithm. In section 3 simulation setup and parameters of the system are presented. Section 4 presents obtained results. The paper concludes with a short outlook on the obtained results and suggestion for the future work.

2 Deep Q-Learning

Reinforcement learning deals with control policies for agents that interact with unknown environments. Environments can be formalized as a Markov Decision Processes (MDPs) with only four values describing them. At each time-step the agent changes its state from the current state s_t to a new state s_{t+1} by performing an action a_t and based on the new state gets the reward r_t. Based on this values, Q-learning algorithm [7] approximates the long term reward known as Q-value if the particular action is performed in given state. Values are iteratively updated by the equation:

$$Q_{new}(s,a) = Q_{old}(s,a) + \\ + \alpha\big[r + \gamma \max_{a'} Q_{old}(s',a') - Q_{old}(s,a)\big] \quad (1)$$

where Q_{old} is an approximate before and Q_{new} after the update, α is learning rate, γ is discount factor and $\max_{a'} Q(s', a')$ is the maximal approximated value over all actions a' in the resulting state s'. However, this way of updating the Q-value means that actions and states need to be discretized thus leading to the Q table of size $S \times A$ where S is the number of possible states and A is the number of possible actions. Instead of this, with the Deep Q-learning algorithm, Q-values are approximated by the neural network (parametrized by weights and biases collectively denoted by θ). With the use of neural network, Q-values approximates, denoted by $Q(s, a|\theta)$, are estimated by making a forward pass when an input is the current state of the system. By using neural network, discretization of the states is not required because it generalizes beyond the states that it was trained on. To avoid divergence and oscillations in learning [8], experiences of transitions are stored in replay memory D as $d_t = \{s_t, a_t, r_t, s_{t+1}\}$ and uniformly sampled in mini-batches containing examples for each training pass. Adam optimizer [9] was used to optimize learning momentum. General deep Q-learning is given below in Algorithm 1.

Initialize replay memory D
Initialize neural network for approximating
 Q-value with a random weights and biases θ
for $i \in [1, number\ of\ episodes]$ **do**
  Initialize state s_t
  for $t \in [1, number\ of\ steps]$ **do**
    With ε probability select random action a_t,
     otherwise select $a_t = \max_{a'}(s, a')$
    Execute a_t, observe the next state s_{t+1} and
     get reward r_t
    Store transition $d_t = (s_t; a_t; r_t; s_{t+1})$ in D
    Set $s_t = s_{t+1}$
    Sample mini-batch from D
    for $j \in [1, mini - batch\ size]$ **do**
      if *s'= terminal state* **then**
        $y_j = r_t$
      else
        $y_j = r_t + \gamma \max_{a'} Q(s', a'|\theta)$
    end
    Perform one step of training using
     $(y_j - Q(s, a|\theta))^2$ as a cost function
  end
end

Algorithm 1: Deep Q-learning algorithm [10]

3 Experimental evaluation

In order to test the robustness and speed of learning, we modelled the example of the cart pole with the pendulum in Matlab Environment as shown in the Figure 1.
Simulated cart pole mass was $M = 1$kg, mass of pendulum was $m = 0.1$kg, length was 0.5m and it could be moved left or right by applying the force of -10N or 10N respectively. For the state of the system to be fully defined, we used two generalized coordinates: x-axis and the displacement angle ϕ. The cart was moving along x-axis and it had to stay within the range of $x \in (-2.6m, 2.6m)$ for balancing to be counted as successful. The displacement angle of the pole (ϕ) is the second generalized coordinate and it was in the range $\phi \in [-180°, 180°]$ as shown in Figure 1. The pendulum was set to initial position of $\{x, \dot{x}, \phi, \dot{\phi}\} = \{0\ m, 0\ m/s, 0°, 1°/s\}$ so that in initial position equilibrium state was disturbed. The number of possible actions yields the size of neural network output layer at two neurons (for -10N and 10N) and number of states needed to fully describe the system $(x, \dot{x}, \phi, \dot{\phi})$ sets the input layer size to four neurons.

The goal of learning algorithm was to learn how to balance the pendulum. In order to accomplish that task, we tested three different neural networks with the architecture shown in Figure 2. With all three networks we tested different combinations of reinforcement learning parameters (exploration rate ε and discount factor γ). After finding the combination that was able to find the balancing policy most efficiently, we added uncertainty to the angle measurement to simulate sensors in a real world environments and measured the number of iterations that the policy successfully managed to balance the pendulum.

4 Results

To find optimal learning strategy, we tested the learning efficiency with the different combinations of ε and γ parameters. Our results have shown that the choice of the neural network is crucial for the performance policy learning. We tested learning algorithm on three different networks formed of $4 \times 16 \times 2$ (Network A), $4 \times 1024 \times 256 \times 2$ (Network B), $4 \times 16 \times 32 \times 16 \times 8 \times 2$ (Network C) fully connected layers as shown in Figure 2. Results have shown that it is crucial for the task to find the smallest possible network to achieve good speed of learning and resistance to external perturbations. Bars in Figure 3 show the episode of learning in which algorithm successfully managed to balance the pendulum for at least 300 steps for shown pairs of parameters for the Network A (Figure 2-left) and Network B (Figure 2-middle). The deepest network (Network C) (Figure 2-right) did not manage to find any balancing policy for any pairs of parameters in 10000 episodes.

With analyzing the results we found out that fastest learning occurred with the parameters $\{\varepsilon, \gamma\} = \{0.05, 0.8\}$ for the case of Network A (in 20 iterations) and with the $\{\varepsilon, \gamma\} = \{0.05, 0.9\}$ for the case of Network B (in only 7 iterations).

For aforementioned cases, the training was stopped after the first success and resistance of learned policy was analyzed by adding simulated sensor noise on the reading of the state of the angle ϕ. Balancing was considered to be successful for the angle $\phi \in [-12°, 12°]$ and that is why maximal allowed noise on our simulated sensors was set to the same values.

We tested how the number of the iterations that pendulum was balanced was affected by this noise in both

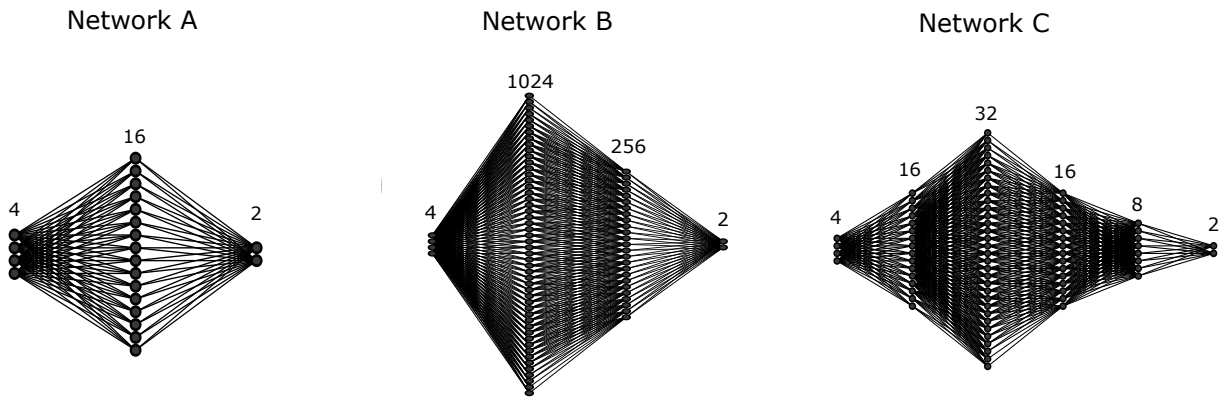

Figure 2: Neural network architectures used for approximating Q-value.

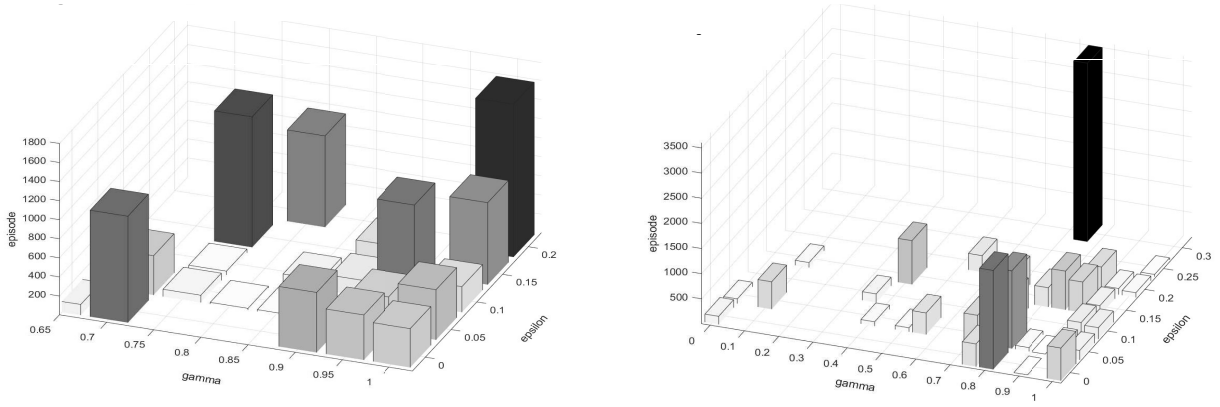

Figure 3: Pairs of γ and ε parameters that found the control policy for the networks A (left) and B (right) at the iteration shown by the bars. Only parts of the graphs where solution is found are shown. Neural network C did not manage to find the policy with any parameters.

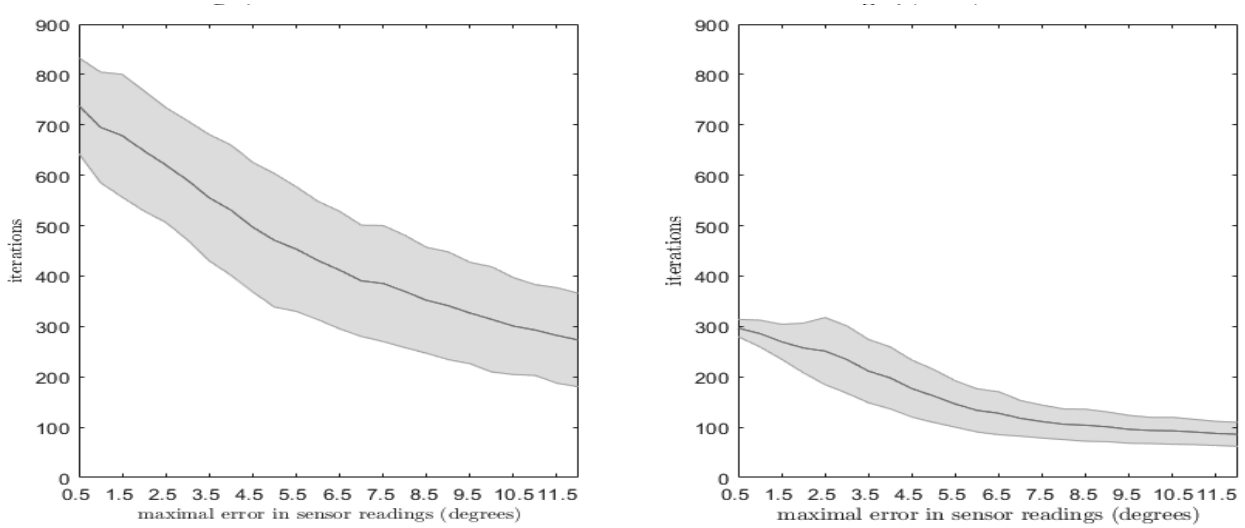

Figure 4: Resistance based on the number of iterations in which the pendulum satisfied the stabilizing criteria using control policy learned by networks A (left) and B (right) with applied sensory noise.

cases (Networks A and B) and the results are shown in the Figure 4. The graphs show the mean and standard deviation for the number of iterations in which balancing was successfully performed (tests were done 1000 times for statistics). As expected, bigger noise reduced the number of successful balancing iterations. The results show that smaller network is much more robust to the wrong readings from the sensors and that it manages to find a better

policy.

5 Conclusion

Results show that the architecture of the neural network is crucial for the success of the task. Size of the neural network should be smallest possible for the solution to be found in reasonably small number of episodes. For the simple problem such as balancing the pendulum from initial state in upright position, high complexity of the neural network negatively affects the speed of a learning process. Our tests show the performance might get more degraded by extending the depth than extending the width of the network. With the control problems in almost linear space as in presented use-case, there is no need for big exploration noise to be added as proven by the cases with the fastest learning ($\varepsilon \approx 0.05$ for the cases with both networks). Our tests with the different parameters have shown that the best choice for γ is to choose values within the range $\left[0.8, 1\right]$. With the success occurring in small number of learning episodes, deep Q-learning seems to be promising approach in making the controllers for the systems found in real world.

In the future work, we plan to test the balancing on the real world system and with using the convolutional neural networks (CNN) as used in [11, 5] and to try to extend the problem complexity to finding the strategy that would be able to swing up and balance pendulum with using the same network for both problems (swing-up and balance) or play ball-in-a-cup game as it was done using the regular Q-learning algorithm in [12]. We also want to check the possibilities of improvement using the adaptive learning rate method such as RMSProp [13] or ADADELTA [14]. We are planning to perform the analysis of vanishing gradient [15, 16] to check if there are methods that would help us to make deeper network architectures learn the policies. With making deep neural networks more robust with optimization methods, we would be able to train the policies for more complicated tasks as it was done with the recurrent neural networks [17, 18].

Acknowledgement: The corresponding author is supported by the Fund for Public Scholarship, Development, Disability and Maintenance Fund of the Republic of Slovenia with Ad futura Scholarship for Postgraduate Studies of Nationals of Western Balkan States for Study in the Republic of Slovenia (226. Public Call).

References

[1] Moravčik Matej, S. Martin, B. Neil, V. Lisy, M. Dustin, B. Nolan, D. Trevor, W. Kevin, J. Michael, and B. Michael, "DeepStack : Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[2] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

[3] R. Pahič, Z. Lončarević, A. Ude, B. Nemec, and A. Gams, "User feedback in latent space robotic skill learning," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 270–276, Nov 2018.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics : A Survey," *Learning Motor Skills. Springer Tracts in Advanced Robotics*, vol. 97, no. Springer, Cham, 2013.

[5] 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, pp. 529–533, 2015.

[6] M. J. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *CoRR*, vol. abs/1511.04143, 2016.

[7] P. Dayan, "Technical Note Q-Learning," *Machine Learning (MLJ)*, vol. 8, pp. 279–292, 1992.

[8] J. N. Tsitsiklis and B. V. Roy, "An Analysis of Temporal-Difference Learning with Function Approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[9] D. Kingma and J. Ba, "Adam: a method for stochastic optimization (2014)," *arXiv preprint arXiv:1412.6980*, vol. 15, 2015.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[11] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, and L. Sifre, "Mastering the game of Go without human knowledge," *Nature Publishing Group*, vol. 550, no. 7676, pp. 354–359, 2017.

[12] B. Nemec, M. Zorko, and L. Zlajpah, "Learning of a ball-in-a-cup playing robot," *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*, pp. 297–301, 2010.

[13] Y. N. Dauphin, H. Vries, J. Chung, and Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for non-convex optimization," *arXiv*, vol. 35, 02 2015.

[14] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[15] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012.

[16] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München," 1991.

[17] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 819–825, IEEE, 2017.

[18] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Fall Symposia*, 2015.