Using Neural Networks for Synthesizing Importance Sampler Database in Reinforcement Learning

Zvezdan Lončarević[1,2], **Andrej Gams**[1,2]

[1]*Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia*
[2]*Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia*
zvezdan.loncarevic@ijs.si

Abstract

Reinforcement learning is a widely used method of acquiring new skills in robotics. However, it is usually rather slow and a lot of learning iterations are needed until robot successfully learns the skill. During learning attempts, parameters of the actions together with the corresponding reward are stored and used in the following update. In this paper, we present the possibility of using neural networks for expanding the database containing the knowledge from previous learning iterations. Results of throwing examples show that this can lead to accelerated robot learning with less iterations and real-world repetitions.

1 Introduction

In order for robots to move into unstructured environments, adaptation to the current state of the world is critical. One of the approaches to adaptation is robot learning, i.e., the process of task performance improvement over the course of several repetitions [1]. However, robot learning can be complicated and can take a long time. It might also not be safe for the robot or its vicinity.

Many approaches have been proposed to improve the speed and safety of robot learning. The literature states that a good starting point for learning is really important [2]. For example, the initial task execution or policy can be acquired by demonstration [3] or by generalization [4]. Another key approach is in the reduction of search space, for example with principal component analysis or autoencoder neural networks [5].

Learning approaches themselves can also have a profound effect on the required number of task iterations. Sample efficiency of methods is a known problem of deep reinforcement learning approaches [6]. However, even different reinforcement learning methods require different amounts of samples. For example, gradient-based methods, such as Reinforce, eNAC and CMA-ES require several roll-outs for one iteration (update of policy parameters) [7]. Non-gradient based algorithms, such as PoWER and PI2, do not need to first calculate the gradient but can update the policy based on a few best attempts and, for example random noise.

A few best attempts in reinforcement learning can be classified as those attempts that collect the most rewards.

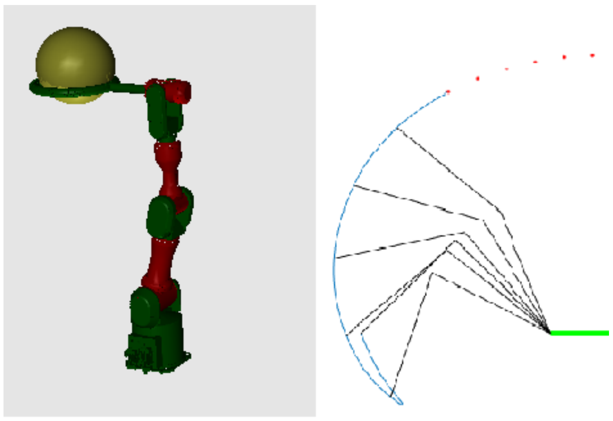

Figure 1: PA10 robot in MuJoCo Dynamic Simulation (left) and 2-DoF planar robot in Matlab (right)

For example, the reward from all task executions is compared, and only the ones with the highest rewards are then used to generate the new iteration. In order to classify the attempts, specially when there is no model available, can also be time consuming.

In this paper we propose a method that reduces the number of required attempts by training a neural network that maps between policy parameters and the expected reward. Before generating a new set of policy parameters, the algorithm predicts their reward, and then uses this virtually acquired reward, together with the rewards of previous attempts, as the input into the importance sampler. The results show that the approach reduces the required amount of needed iterations and the highest number of iterations until the task if learned. We used simulated throwing at a target as the demonstration task. The simulated set-up in MuJoCo [8] for dynamic simulation and planar kinematic simulation are shown in Fig. 1.

2 Search Space Reduction

In order for RL to be applied in robotics, it has to learn fast enough. As neural networks have fixed number of inputs, we need to represent trajectories so that each example has the same number of parameters. For that purpose, we used Dynamic Movement Primitives and autoencoder networks.

2.1 Dynamic Movement Primitives

The basic idea of Dynamic Movement Primitives (DMPs) [9] is to represent the trajectory with the mass on spring-damper system, with learned external accelerations - the forcing term. For the each joint space coordinate y, DMP is based on the following second order differential equation:

$$\tau^2 \ddot{y} = \alpha_z(\beta_z(g-y) - \tau\dot{y}) + f(x), \tag{1}$$

where τ is the time constant and it is used for time scaling, α_z and β_z are damping constants ($\beta_z = \alpha_z/4$) that make system critically damped and x is the phase variable. The forcing term $f(x)$ encodes the shape of the trajectory from the initial position y_0 to the final configuration g. It is given by:

$$f(x) = \frac{\sum_{i=1}^{N} \psi(x) w_i}{\sum_{i=1}^{N} \psi_i(x)} x, \tag{2}$$

$$\psi_i(x) = \exp\left(-\frac{1}{2\delta_i^2}(x-c_i)^2\right), \tag{3}$$

where c_i are the centers of radial basis functions ($\psi_i(x)$) distributed along the trajectory and $\frac{1}{2\delta_i^2}$ their widths. Phase x makes the forcing term $f(x)$ disappear when the goal is reached as it exponentially converges to 0. Its dynamics are given by

$$\tau\dot{x} = -\alpha_x x, \tag{4}$$

where α_x is a positive constant and x starts from 1 and converges to 0 as the goal is reached. Multiple DoFs are realized by maintaining separate sets of (1–3), while a single canonical system given by (4) is used to synchronize them.

2.2 Autoencoder networks

Autoencoder networks are neural networks that are often used for dimensionality reduction. Autoencoders with nonlinear layers are capable of extracting the most relevant features of robotic movements. They are composed of two parts: encoder and decoder (Fig. 2). An autoencoder neural network is trained so that the output data $\tilde{\boldsymbol{\theta}}^{\mathrm{DMP}}$ matches the input data $\boldsymbol{\theta}^{\mathrm{DMP}}$ as close as possible. The encoder part pushes data to the bottleneck of the neural network called latent space and the decoder part recreates the original data, therefore $\mathbf{F}_{\mathrm{d}} \approx \mathbf{F}_{\mathrm{e}}^{-1}$. An autoencoder is trained on a large set of executable kinematic trajectories represented with DMP parameters $\boldsymbol{\theta}_i^{\mathrm{DMP}}, i = 1, \ldots, m$ by optimizing the following criteria:

$$\boldsymbol{\zeta}^\star = \arg\min_{\boldsymbol{\zeta}} \frac{1}{m} \sum_{i=1}^{m} \left\| \boldsymbol{\theta}_i^{\mathrm{DMP}} - \mathbf{F}_{\mathrm{d}}\left(\mathbf{F}_{\mathrm{e}}\left(\boldsymbol{\theta}_i^{\mathrm{DMP}}\right)\right) \right\|, \tag{5}$$

where $\boldsymbol{\zeta}^\star$ are the autoencoder parameters (weights and biases of neurons in the AE network). Once the network is fully trained, latent space parameters can be computed by applying the encoder part of the network:

$$\boldsymbol{\theta}^{\mathrm{AE}} = \mathbf{F}_{\mathrm{e}}\left(\boldsymbol{\theta}^{\mathrm{DMP}}\right), \tag{6}$$

and the decoder part can return latent space values into DMP:

$$\tilde{\boldsymbol{\theta}}^{\mathrm{DMP}} = \mathbf{F}_{\mathrm{d}}\left(\boldsymbol{\theta}^{\mathrm{AE}}\right). \tag{7}$$

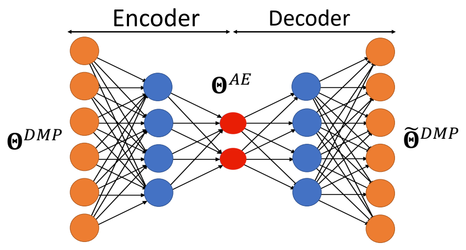

Figure 2: Example of simple autoencoder network

3 Reward Weighted Policy Learning with NN Extended Importance Sampler

As RL algorithm of choice we used reward-weighted policy learning with importance sampling (RWPL), which is a simplified variant of Policy Learning by Weighting Exploration with the Returns (PoWER) method [10]. It uses a parametrized skill policy and a reward function to maximize the expected return of skill performance trials.

Under the assumption that there is only terminal reward and that only a single basis function is active at any given time (note that this is only approximately true for DMPs), the policy parameters $\boldsymbol{\theta}_n$ are updated using:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \frac{\langle(\boldsymbol{\Theta}_n - \boldsymbol{\theta}_n)R(\boldsymbol{\Theta}_n)\rangle_{w(\boldsymbol{\pi}_k)}}{\langle R(\boldsymbol{\Theta}_n)\rangle_{w(\boldsymbol{\pi}_k)}}, \tag{8}$$

where $w(\boldsymbol{\pi}_k)$ denotes the policy parameters of k-th iteration. $\boldsymbol{\Theta}_n = \{\boldsymbol{\theta}_k^*\}_{k=1}^{n}$ denotes the set of all policy parameters $\boldsymbol{\theta}_k^*$ executed until the n-th iteration and $R > 0$ the terminal reward received at the end of each rollout. $\langle\cdot\rangle_{w(\boldsymbol{\pi}_k)}$ denotes importance sampling [11]. Its role is to select a predefined number of best trials to compute the update in order to reduce the number of iterations until optimal policy is learnt.

In order to increase the convergence rate, we used neural network (NN) to artificially augment our dataset of parameters-reward pairs. It was trained so that it takes policy parameters as input and gives the approximated reward as an output. It was retrained after each iteration and the training dataset used for this consisted of n already executed trajectory parameters $\boldsymbol{\Theta}_n$ and corresponding rewards $R(\boldsymbol{\Theta}_n)$. After the network is trained, we have chosen p new random sets of the parameters and afterwards used NN to approximate the rewards. This way, we generated extended dataset of parameters $\boldsymbol{\Theta}_n'$ and corresponding rewards $R(\boldsymbol{\Theta}_n')$.

The update rule specified by Eq. (8), together with using augmented dataset is equivalent to

$$\boldsymbol{\theta}_{n+1} = \frac{\sum_{i=1}^{m} R_{\mathrm{in}(n',i)} \boldsymbol{\theta}_{\mathrm{in}(n',i)}^*}{\sum_{i=1}^{m} R_{\mathrm{in}(n',i)}}, \tag{9}$$

where function $\mathrm{in}(n', i)$ selects the trial with the i-th highest reward from the extended trial set $\{\boldsymbol{\theta}_k^*, R_k\}_{k=1}^{m}$, m is the number of best trials selected by importance sampling, and the exploration parameters are computed by adding exploration noise to the current estimate $\boldsymbol{\theta}_n$

$$\boldsymbol{\theta}_n^* = \boldsymbol{\theta}_n + \boldsymbol{\varepsilon}_n. \tag{10}$$

Here $\boldsymbol{\varepsilon}_n$ is zero-mean Gaussian noise. Its variance $\boldsymbol{\Sigma}$ is usually a diagonal matrix and needs to be specified by the user. In general, high variance does a more thorough exploration of the parameter space but may cause oscillations around the solution, while small variance can get stuck in a local minimum.

4 Experimental Evaluation

As the use-case scenario, we used learning of robotic throwing of a ball into the basket. Throwing was the chosen task because it has been already widely studied in RL settings and because it is easy to define the reward. Our method for accelerating RL algorithms was evaluated in two different experimental setups. In Matlab we used a kinematic model of a 2-DoF planar robot that was throwing a ball at the target (Fig. 1 - right). In this simulation, air drag and friction were neglected. With this model, we tested the increase in the performance of the RL algorithm when we use NN for approximating the reward with randomly given DMP parameters to increase the number of examples. As learning in more reduced space is faster [12], for learning in the latent space of neural network, we used MuJoCo simulation with complete robot and ball dynamics (Fig. 1 - left). In this case NN was approximating reward for the corresponding latent space values. Experiments in the dynamic simulation were repeated for 30 different randomly chosen targets and results together with the spreading of the data are reported.

4.1 Extending the database in DMP space

For accelerating the learning in DMP space, in each iteration we were using NN with 45-50-10-1 neurons respectively. Size of NN was determined empirically. Parameters of the input layer were weights describing the shape of trajectory for two joints of the robot. This sets the input of the neural network to:

$$\boldsymbol{\Theta}_n' = \left\{\boldsymbol{w}_j, y_{0,j}, g_j, \tau\right\}_{j=1}^{l}, \tag{11}$$

where $l = 2$ is number of active joints, τ is time scaling factor, $\boldsymbol{w}_j = \{w_i\}_{i=1}^N$ where $N = 20$ are weights describing the trajectory profile and $y_{0,j}$, g_j are initial and final points of the trajectory for each joint. Output of the neural network was predicted normalized reward of the shot given based on the distance of the ball landing spot from the desired target. After each trial, NN was used to approximate $p = 10000$ new examples and add them to the dataset of executed trajectories. Length of the importance sampler was set to $m = 5$. In this experiment, we were shooting at the (same) randomly chosen target with our approach with extended database for importance sampler and with regular RWPL algorithm.

4.2 Extending the database in latent space

For accelerating the learning in the latent space of AE, we used NN with 3-10-7-5-1 neurons respectively. In order to obtain latent space values, a large dataset of executable trajectories was calculated with respect to the kinematic properties of the used Mitsubishi PA10 robot as described in [5]. In order to obtain latent space values, we used autoencoder with 67-15-10-3-10-15-67 neurons. Input and output of the AE network were DMP parameters for 3 active DoF of the robot. With setting $l = 3$ in Eq. (11), 67 input/output values ($\boldsymbol{\theta}^{\mathrm{DMP}}$) were obtained and AE network was trained using Eq. (5). Using Eq. (6) we obtained latent space of the AE ($\boldsymbol{\theta}^{\mathrm{AE}}$). This parameters were used as an input to our neural network for dataset augmentation. Since in our case, there was only 3 latent space values, input parameters to this neural network were:

$$\boldsymbol{\Theta}_n' = \{\theta_i\}_{i=1}^3, \tag{12}$$

and the output was predicted normalized reward of the shot with that parameters. Same as in the experiment in DMP space, importance sampler length was set to $m = 5$ and NN approximated $p = 10000$ examples. Because learning is much faster in the reduced (latent) space, in this experiment we were able to use MuJoCo dynamic simulation. For this experiment we have chosen 30 (same) random targets and learned throwing with our approach and with regular RWPL algorithm.

5 Results

Figure 3 shows the results of throwing in kinematic simulation. Blue line shows convergence of the error and reward for the case of regular RWPL algorithm while red line shows convergence when NN for approximating reward based on DMP parameters was used.

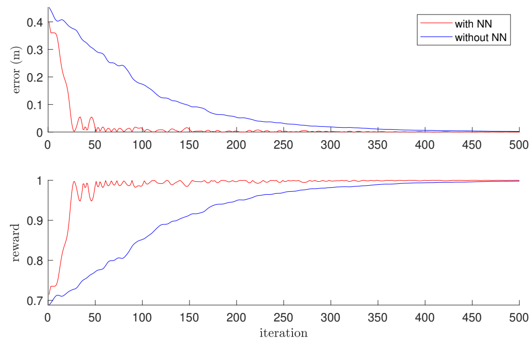

Figure 3: Error and reward convergence for the kinematic simulation with neural network synthesized examples (red line) and without neural network synthesized examples (blue line). Neural network connects DMP parameters with the corresponding reward.

The average convergence of the error and reward together with spreading of the data for the case when NN was used for connecting latent space parameters with corresponding reward is shown in Fig. 4. Blue line shows the results for regular RWPL and red line shows the results of our approach. Since learning in the latent space is anyway significantly faster than learning in DMP space, the results show less difference for this case. However, from Fig. 5 it is visible that our approach still managed to achieve some increase in the convergence rate since the average number of required iterations as well as highest number of iterations both reduced when compared with regular RWPL approach.

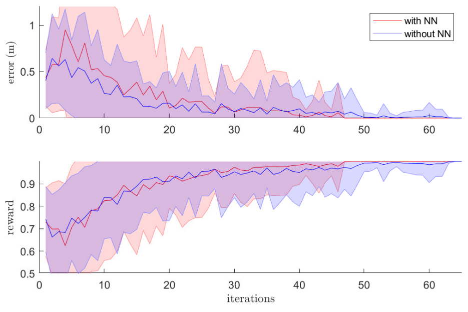

Figure 4: Mean error and reward convergence for the dynamic simulation with neural network synthesized examples (red line) and without neural network synthesized examples (blue line). Neural network connects latent space values with the corresponding reward. Shaded area shows the spreading of the data among 30 different targets.

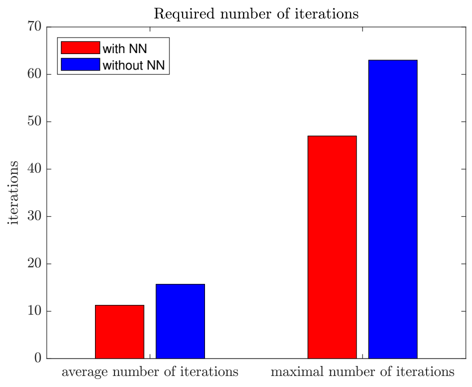

Figure 5: Average (left) and maximal (right) number of iterations needed to hit the target with and without neural network synthesized examples. Blue bars present the results of the regular RWPL algorithm and red bars present the results of our approach.

6 Conclusion

Results show that neural networks can accelerate RL by artificially expanding dataset of known trajectories. Increase in convergence rate in both DMP and AE space shows that our approach has potential to be applied in robotic tasks where the each learning iteration saves time and reduces wear of the equipment. Much bigger increase in performance is noticed in the less reduced DMP space where algorithms need to find more parameters. This was expected since the AE search space reduction already significantly increases RL performance. However, it should be noted that it can be used only when the large dataset of trajectories is available which is usually not the case. Other benefit of this approach is that less parameters need to be tuned for RL because NN gets retrained after each trial and there is less possibility of RL algorithm getting stuck in some local minimum.

In the future, we plan to combine this approach with another RL algorithm and use the additional neural network to predict the parameters of the initial trajectory and reward for each following iteration of learning. We also plan to implement this algorithm on the physical robot instead of only simulation and to check whether trajectories generated with our approach are smoother and safer for execution than the ones found by using only RL algorithm.

References

[1] F. Stulp, E. A. Theodorou, and S. Schaal, “Reinforcement learning with sequences of motion primitives for robust manipulation,” *IEEE Transactions on Robotics*, vol. 28, no. 6, pp. 1360–1370, 2012.

[2] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[3] M. Deniša, A. Gams, A. Ude, and T. Petrič, “Learning compliant movement primitives through demonstration and statistical generalization,” *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 5, pp. 2581–2594, 2016.

[4] Z. Lončarević, R. Pahič, A. Ude, and A. Gams, “Generalization-based acquisition of training data for motor primitive learning by neural networks,” *Applied Sciences*, vol. 11, p. 1013, 2021.

[5] R. Pahič, Z. Lončarević, A. Gams, and A. Ude, “Robot skill learning in latent space of a deep autoencoder neural network,” *Robotics and Autonomous Systems*, vol. 135, p. 103690, 2021.

[6] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, “Improving sample efficiency in model-free reinforcement learning from images,” 2020.

[7] Z. Lončarević, A. Gams, S. Reberšek, B. Nemec, J. Škrabar, J. Skvarč, and A. Ude, “Specifying and optimizing robotic motion for visual quality inspection,” *Robotics and Computer-Integrated Manufacturing*, vol. 72, p. 102200, 2021.

[8] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[9] A. Ijspeert, J. Nakanishi, and S. Schaal, “Movement imitation with nonlinear dynamical systems in humanoid robots,” *Proceedings 2002 IEEE International Conference on Robotics and Automation*, vol. 2, no. May, pp. 1398–1403, 2002.

[10] J. Kober and J. Peters, “Policy search for motor primitives in robotics,” *Machine learning*, vol. 84, p. 171–203, July 2011.

[11] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta, “Learning the skill of archery by a humanoid robot iCub,” in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 417–423, 2010.

[12] Z. Lončarević, R. Pahič, M. Simonič, A. Ude, and A. Gams, “Reduction of trajectory encoding data using a deep autoencoder network: Robotic throwing,” in *Advances in Service and Industrial Robotics*, (Cham), pp. 86–94, Springer International Publishing, 2020.