The importance of experience replay database composition in deep reinforcement learning

Tim de Bruin
Delft Center for Systems and Control
Delft University of Technology
t.d.debruin@tudelft.nl

Jens Kober
Delft Center for Systems and Control
Delft University of Technology
j.kober@tudelft.nl

Karl Tuyls[*]
Department of Computer Science
University of Liverpool
K.Tuyls@liverpool.ac.uk

Robert Babuška
Delft Center for Systems and Control
Delft University of Technology
r.babuska@tudelft.nl

Abstract

Recent years have seen a growing interest in the use of deep neural networks as function approximators in reinforcement learning. This paper investigates the potential of the Deep Deterministic Policy Gradient method for a robot control problem both in simulation and in a real setup. The importance of the size and composition of the experience replay database is investigated and some requirements on the distribution over the state-action space of the experiences in the database are identified. Of particular interest is the importance of negative experiences that are not close to an optimal policy. It is shown how training with samples that are insufficiently spread over the state-action space can cause the method to fail, and how maintaining the distribution over the state-action space of the samples in the experience database can greatly benefit learning.

1 Introduction

The combination of Reinforcement Learning (RL) with Artificial Neural Networks (ANN) is an interesting approach to the control of dynamical systems. Reinforcement learning provides a framework that makes it possible to learn complex nonlinear control policies without any prior knowledge of the system to be controlled. When this is combined with the ability of neural networks to represent highly nonlinear mappings and generalize the gained knowledge to previously unseen inputs, it is clear that the combination holds a lot of potential.

One category of methods that is of special interest for control are the model-free off-policy actor-critic methods [11]. These methods allow for continuous actions and use only interaction samples from the system. Although model-free methods require no prior knowledge of the system, this comes at the price of needing more samples to learn a good policy. This is problematic when learning control on physical systems, since extra interaction time will cause wear on the system. Having an off-policy algorithm helps to some extent since it allows for the use of experience replay [14, 21, 22]. This technique reuses experience samples, increasing the sample efficiency.

One recently published model-free off-policy actor-critic method that uses experience replay is the Deep Deterministic Policy Gradient (DDPG) method [1]. In this method, experience replay is used not only to increase the sample efficiency but it is also crucial for the stability (convergence) of the

[*]Author is also affiliated as a part-time professor with Delft Center for Systems and Control.

learning process. In this paper the influence of the properties of the database and the distribution of the samples contained in the database on the stability and speed of the learning process are examined.

In reinforcement learning, the way in which new experiences are obtained presents a well known problem: the exploration-exploitation trade-off. It is important to explore sufficiently far away from the current best policy, to not to get stuck in a local minimum, while still exploiting the current knowledge so that the learning process is as fast as possible [18]. When using neural networks, additional requirements are placed on the distribution of the experiences over the state-action space that also have to be taken into account. These requirements stem from the fact that when the neural network training data are not varied enough, the network is likely to over fit, resulting in poor generalization performance in unexplored parts of the state-space. When this is combined with the fact that neural networks can easily forget about data that they are no longer trained on [16], it becomes apparent that when an experience database of limited size is used, the contents should be maintained with some care.

These issues are investigated in the context of a 2-DOF robot, on which the DDPG method is implemented both in simulation and on the robot. The remainder of this paper is structured as follows. Section 2 will provide background on the DDPG method and the use of experience replay. In Section 3 we give details about the implementation of the experiments and in Section 4 compare different experience collection strategies, the effect of the size of the experience replay database and the effect of retaining experiences that were obtained early on in the learning process.

2 Method

The reinforcement learning method used in this paper is based on the Deep Deterministic Policy Gradient method presented in [1], an off-policy actor-critic algorithm. The actor π attempts to determine the real-valued control action $a^\pi \in \mathbb{R}^n$ that will minimize the expected sum of future costs c based on the current state of the system $s \in \mathbb{R}^m$; $a^\pi = \pi(s)$. The critic attempts to give the expected discounted sum of future costs when taking action $a(k)$ in state $s(k)$ at time k and the policy π is followed for all future time steps:

$$Q^\pi(s,a) = \mathbb{E}\left[\left.\left(c\big(s(k),a,s(k+1)\big) + \sum_{j=k+1}^{K} \gamma^{j-k} c\big(s(j),\pi(s(j)),s(j+1)\big)\right)\right|_{s(k)=s}\right] \tag{1}$$

With $0 \leq \gamma < 1$ the discount factor that ensures the sum is finite and the emphasis is on short term costs.

The actor and critic functions are approximated by neural networks with parameter vectors ζ and ξ respectively. The critic network weights are updated to minimize the squared temporal difference error:

$$\mathcal{L}(\xi) = \Big(Q(s,a|\xi) - c + \gamma Q(s',\pi(s'|\zeta)|\xi)\Big)^2 \tag{2}$$

Where $s = s(k)$, $s' = s(k+1)$ and $c = c(s,a,s')$ for brevity. The actor network is updated in the direction that will minimize the expected cost according to the critic:

$$\Delta\zeta \sim -\nabla_a Q(s,a|\xi)|_{s=s(k),a=\pi(s(k)|\zeta)} \nabla_\zeta \pi(s|\zeta)|_{s=s(k)} \tag{3}$$

It is shown in [4] that this is the true gradient of the expected overall cost with respect to the policy parameters, when the critic has a certain *compatible* form. Although using this compatible form might be too restricting, using (3) was shown in [1] to work for non-compatible forms as well, when certain stability measures are added to the method. These measures include the addition of an experience replay database and the use of separate target networks.

Using (2) and (3) directly can be problematic when using neural networks for the actor and critic functions [1, 2]. During the training of the critic network the parameters ξ are updated to bring the output of the network $Q(s,a|\xi)$ closer to the *target* value $c + \gamma Q(s',\pi(s'|\xi))$. However the target value depends on the same parameters ξ that are being updated to bring the output of the network closer to the target, both directly for the value of Q at the next state and through the policy which is updated based on Q. This coupling can cause the training of the networks to become unstable.

To solve this, the target values are updated using a separate copy of both the actor and the critic networks. A low-pass filter is used to update the network parameters of the copies according to:

$$\xi^- \leftarrow \tau\xi + (1-\tau)\xi^- \tag{4}$$
$$\zeta^- \leftarrow \tau\zeta + (1-\tau)\zeta^- \tag{5}$$

Where the choice of τ presents a trade-off between the speed and stability of the learning process. The target networks are then used to replace (2) with:

$$\mathcal{L}(\xi) = \Big(Q(s,a|\xi) - c + \gamma Q(s', \pi(s'|\zeta^-)|\xi^-)\Big)^2 \tag{6}$$

2.1 Experience replay

The second contribution of [1], based on earlier work [2, 3], that enabled the use of deep neural networks as function approximators for the actor and critic in a stable way is the use of the experience replay database [14]. The experience tuples $< s, a, s', c >$ from the interaction with the system are stored in a database. During the training of the neural networks, the experiences are sampled from this database, allowing experiences to be used multiple times. The addition of experience replay aids the learning in several ways. The first benefit is increased efficiency. Experience replay helps to increase the sample efficiency by allowing samples to be reused. On top of this, in the context of neural networks, experience replay allows for mini-batch updates which helps the computational efficiency, especially when the training is performed on a GPU.

On top of the efficiency gains that experience replay brings, it also improves the stability of the DDPG learning algorithm in several ways. The first way in which the database helps stabilize the learning process is that it is used to break the temporal correlations of the neural network learning updates. Without an experience database, the updates of (2), (3) would be based on subsequent experience samples from the system. These samples are highly correlated since the state of the system does not change much between consecutive time-steps. For real-time control, this effect is even more pronounced with high sampling frequencies. The problem this poses to the learning process is that most mini-batch optimization algorithms are based on the assumption of independent and identically distributed data [1]. Learning from subsequent samples would violate this i.i.d. assumption and cause the updates to the network parameters to have a high variance, leading to slower and potentially less stable learning [12, 13]. By saving the experiences over a period of time and updating the neural networks with mini-batches of experiences that are sampled uniformly random from the database this problem is alleviated.

A second way in which the experience database helps stabilize the learning process is by ensuring that the experiences used to train the networks are not only based on the most recent policy. The policy that is used during the trials to get new experiences is given by:

$$\mu(s) = \pi(s|\zeta) + \epsilon\mathcal{O} \tag{7}$$

Where $\mathcal{O}$ is a noise process that provides exploration and ϵ is used to scale the amount of exploration. This policy therefore determines the experiences that are added to the database and that the neural networks are trained on. When the Q function changes slightly the policy can change significantly, leading to very different samples that are added to the database. This in turn could lead to the algorithm getting stuck in a local minimum or even diverging [2]. By using a database with past experiences this effect is reduced as the experiences from previous policies are still used for training. Important to note here is the fact that neural networks are global function approximators. When the training data are only from a small part of the state-action space the networks might over-fit to the data and generalize poorly to other parts of the state space. Additionally, even if a network has previously learned to do a task well, it can forget this knowledge completely when learning a new task, even when the new task is related to the old one [16]. In the case of the DDPG algorithm, even if the critic can accurately predict the expected cost for parts of the state space, this ability can disappear when it is no longer trained on data from this part of the state-action space, as the same parameters apply to the other parts of the state-action space as well. These effects are shown to seriously affect the learning performance in Section 4.

Regularization can help prevent neural networks from over-fitting to their training data. However, the regularization of neural networks used for control is currently not as well understood as the regularization of neural networks in other domains [17]. It is therefore important to ensure the training data in the experience database is varied enough for the neural networks to properly generalize their knowledge to the whole state-action space. A simple method that helps with this issue is introduced in Section 4.

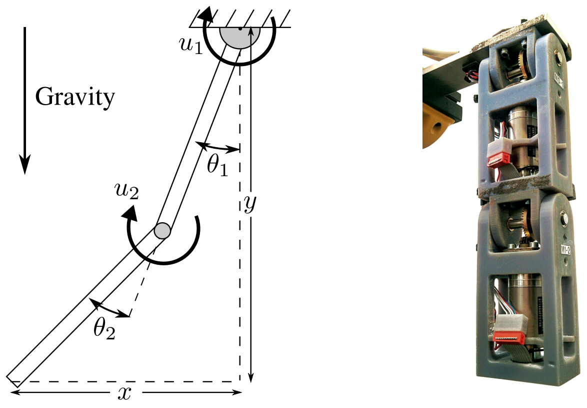


Figure 1: 2-link arm robot schematic (left) and photo (right).

3 Experimental setup

The DDPG method is used to learn the control of a 2-link arm, both in simulation and on a real setup. The system is depicted in Figure 1. It consists of two links that are connected through a motorized joint. The arm hangs down from another motorized joint. The angle between the base and the first link is θ_1 and the angle of the second link with respect to the first link is θ_2. Both joints are constrained to $\theta_1, \theta_2 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. The control signals u_1, u_2 represent scaled versions of the voltages to the motors in the joints. To prevent damage to the physical setup due to jittering, the control signal to the setup, but not the simulator, is filtered with a low-pass filter:

$$u(k) = 0.9u(k-1) + 0.1a(k) \tag{8}$$

To ensure the Markov property is satisfied the reinforcement learning state at time k consists of the angles and angular velocities of the joints and the motor signals at time $k-1$, as well as the reference:

$$s(k) = \begin{bmatrix}\theta_1(k) & \theta_2(k) & \dot{\theta_1}(k) & \dot{\theta_2}(k) & u_1(k-1) & u_2(k-1) & r_x(k) & r_y(k)\end{bmatrix}^T \tag{9}$$

The reference in 9 is given in Cartesian coordinates whereas the state of the arm is given in angular coordinates. It is left up to the neural networks to learn the mapping between the two and to deal with the fact that some reference positions can be reached through multiple arm configurations.

The cost of taking action a in state s at time k is based on the distance between the Cartesian coordinates of the end of the second link at time-step $k+1$, as shown in Figure 1, and the reference position at time-step k:

$$c(s(k), a(k), s(k+1)) = w_1 d^2(k) + w_2(1 - e^{-\alpha d^2(k)}) + w_3\|\dot{\theta}(k)\|_2 \tag{10}$$

with

$$d^2(k) = \left\| \begin{bmatrix} r_x(k) - x(k+1) \\ r_y(k) - y(k+1) \end{bmatrix} \right\|_2^2, \qquad x = \sin(\theta_1) + \sin(\theta_1 + \theta_2), \qquad y = \cos(\theta_1) + \cos(\theta_1 + \theta_2) \tag{11}$$

The first term in (10) ensures that the initial learning is quick. Even when the policy is very bad the quadratic cost ensures that it is clear that moving towards the reference position is better than moving away from it. Using a quadratic term alone will not result in good final policies, however, since such a cost function is very flat close to the reference. The second term adds a steep drop in the cost very close to the reference to solve this problem. The third term discourages the highly oscillatory responses that might otherwise result. The constants in (10) that are used in the experiments are $w_1 = 0.2, w_2 = 0.4, w_3 = 8, \alpha = 100$.

The forgetting factor γ is calculated via:

$$\gamma = e^{-\frac{T_s}{\tau_\gamma}} \tag{12}$$

Where T_s is the sampling period and τ_γ is the lookahead horizon in seconds. For this horizon a value of 2.5 seconds was used, which gives $\gamma = 0.961$ for the simulator, which uses a control frequency of 10 Hz and $\gamma = 0.996$ for the real setup which uses a control frequency of 100 Hz.

The networks used for both the actor and the critic are fully connected networks with Rectified Linear Unit (ReLU) [10] nonlinearities. Both networks have two hidden layers of equal size. In the critic network the action inputs come into the network before the second hidden layer. The number of hidden units in both networks is chosen such that they have around 10000 parameters each. This gives the actor network 94 neurons per hidden layer and the critic network 93 neurons per hidden layer. As in [1], batch normalization [9] is used on the inputs to all layers of the actor network and all layers prior to the action input of the critic network. On the critic network L_2 weight decay of $0.5 \cdot 10^{-2}$ was used. The Adam [7] optimization algorithm is used to train the neural networks. This optimization algorithm is appropriate for non-stationary objective functions and noisy gradients, which makes it suitable for reinforcement learning problems.

The experiments in this paper have been run in a growing batch setting [15]. Each experiment consists of n repetitions of a learning run which itself consists of o trials of 60 seconds. During these trials the experiences are added to the experience database. Between trials the neural networks are updated based on the experiences in the complete database. For each separate learning run in the experiment the initialization of the networks and the training references are unique and the database is reset. When the database is full the first experiences are overwritten in a first in first out manner.

The noise process $\mathcal{O}$ in (7) that is used is an Ohrnstein-Uhlenbeck process:

$$\mathcal{O}(k) = \mathcal{O}(k-1) - \alpha\mathcal{O}(k-1) + \beta\mathcal{N}(0,1) \tag{13}$$

With $\alpha = 0.6$, $\beta = 0.4$ and $\mathcal{N}(0,1)$ is Gaussian noise with a mean of 0.0 and standard deviation 1.0. During training the references are determined stochastically and in such a way that they can be reached with both joints $\theta_1, \theta_2 \in [-0.5, 0.5]$. They are changed periodically during the trials. After each learning run the final policy is evaluated with a fixed sequence of predetermined references in the same range.

4 Results

A series of tests were performed to investigate the importance of the size of the experience database and the properties of the experiences contained within the database for the speed and stability of the learning process.

4.1 Data collection strategy

To investigate the effects of the properties of the collected data, three collection strategies are compared. For these tests a database was used that could hold all samples of the learning run.

The first strategy is the default reinforcement learning setting in which the data are collected by following a noisy version of the current policy, where the noise adds exploration (7). The amplitude ϵ of the exploration noise is decayed exponentially per trial.

In the second strategy, the data are collected by following a deterministic policy of decent quality. In this case a PD controller is used which achieved an average cost per trial of 0.35. No exploration is added to this policy, so that the effects of only training the networks on data from one policy become most apparent.

For the final strategy the noise process of (13) was used exclusively to collect the experiences. The average cost per trial of this random policy was 1.1. In all cases the database size was chosen large enough to contain the entire learning run.

The results of this experiment are shown in Figure 2. The results of learning from data obtained with a completely random policy are close to those of learning from the current best policy with exploration. When trying to learn from experiences obtained by following a good but non changing

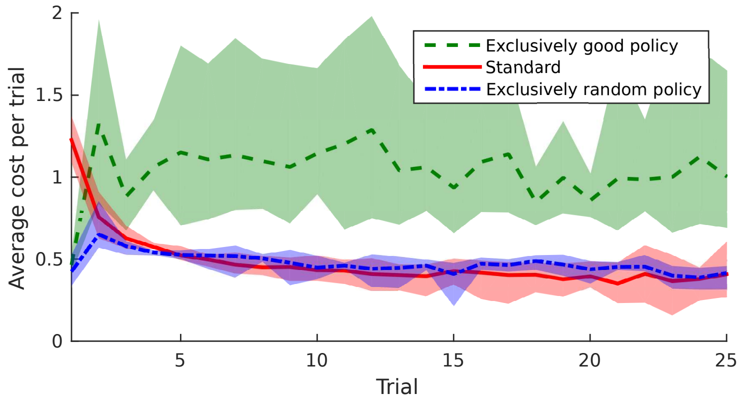


Figure 2: Cost per trial for different experience collection strategies. The curves depict the means over 5 learning run repetitions in the simulator.

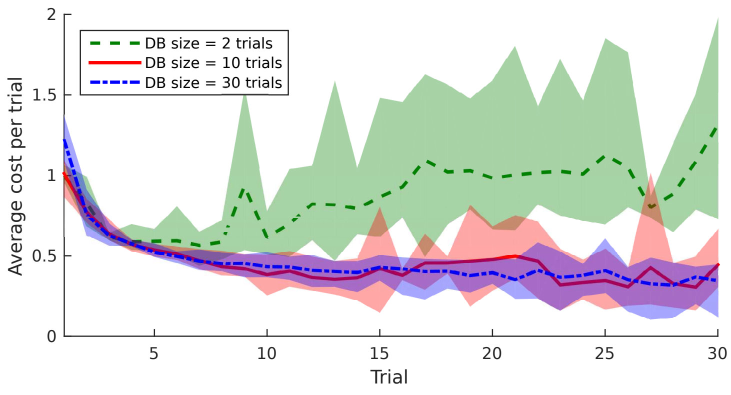


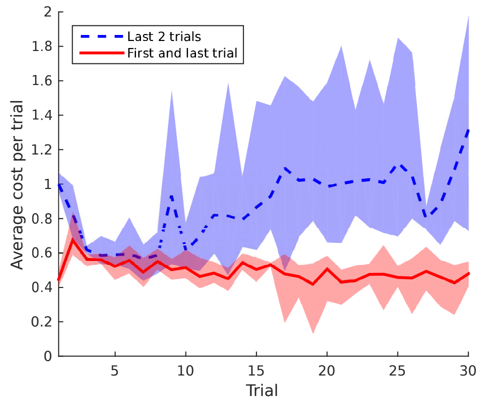
Figure 3: Cost per trial for different database sizes. The curves depict the means over 5 learning run repetitions in the simulator, the shaded areas contain all runs.

deterministic policy the method completely fails. This is likely to be the result of the neural networks over-fitting to this very limited sampling of the state-action space. It is therefore very important to ensure that the samples are diverse enough for the networks to be able to generalize to the whole state-action space with acceptable accuracy.

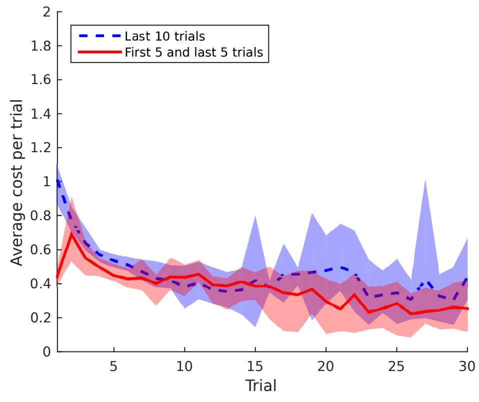
4.2 Database size

From Figure 2 it can be seen that the standard learning strategy of sampling the state-action space according to the current policy with exploration works when the database is large enough to contain all the samples that have been collected so far. It is interesting to see if these results also hold when the amount of exploration is decaying and the database is not large enough to hold all samples. In that case, the database will initially contain experiences from an exploratory policy. As the exploration decays the new samples will be closer to the policy and less varied. Eventually, the initial exploratory samples will be overwritten by the new samples, reducing the variety in the database. When this happens the networks will already contain knowledge from the early exploration. However, this knowledge might be lost after repeated training on the new experiences.

To test this, the learning performance that results from using an experience replay database large enough to contain all experiences is compared to the performance that results from using a database that contains only a small number of trials. In Figure 3 the results are reported for experience databases that contain the last 2 trials and the last 10 trials.



(a) Database size = 2 trials



(b) Database size = 10 trials

Figure 4: Effects of keeping early samples. The curves depict the means over 5 learning run repetitions on the simulator, the shaded areas contain all runs.

Although in all experiments a very basic policy is learned in the first few trials, the learning subsequently becomes instable for all learning runs that use a database containing the experiences from only the last two trials, confirming the hypothesis that knowledge that was gained earlier on is simply forgotten. Additionally, with the amount of exploration decaying, these learning runs never recover and without exception result in unusable policies.

When a database containing the most recent 10 trials is used the learning performance is on average quite close to when all trials are stored. However, it can be seen from Figure 3 that the stability of the learning process is not as good. Although a deteriorating policy is in all cases recovered in subsequent trials, this is still an unwanted effect.

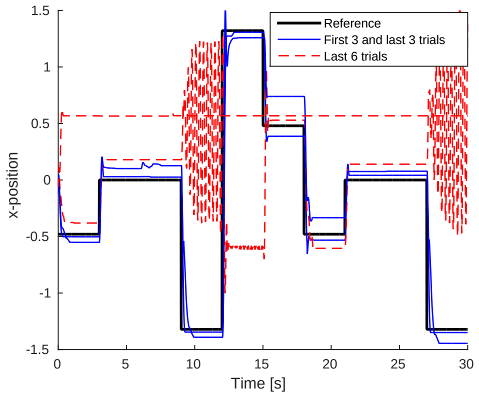
4.3 Retaining early memories

To investigate whether the stabilizing effect observed in Figure 3 for larger databases is indeed related to the presence of the early samples from exploratory policies or simply to the presence of more data, new tests are performed with databases that can hold experiences from 2 and 10 trials respectively. This time however, when the databases are full, the new samples will overwrite only the second half of the database. This way, the experiences from the initial trials are kept indefinitely while the database also always contains the experiences from the most recent trials. The results are compared to the standard strategy of keeping only the most recent trials in Figure 4.

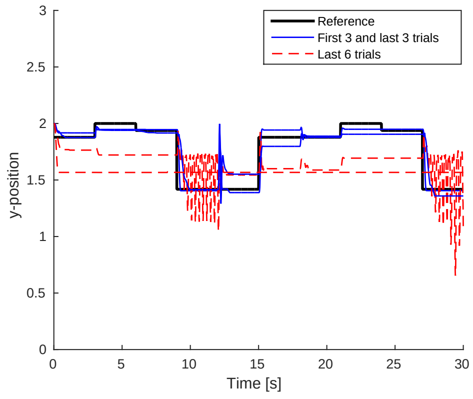
From Figure 4a it can be seen that retaining the experiences from the first trial ensures that the method becomes stable for a database size of 2 trials. For the experiment with a database that holds 10 trials the results show that keeping the initial exploratory experiences in the database not only helps to make the learning process more stable, but also results in better policies than when all trials are kept in memory. Therefore, it might be necessary to make an exploration-exploitation trade-off not only for the policy by which new experiences are obtained, but also for choosing the size of the database and which experiences they replace in the database. Care should be taken however when keeping experiences in the database indefinitely as the networks might start to over-fit on these particular experiences.

4.4 Experiments on the real setup

Experiments have been conducted on the real setup to verify that the phenomena described in this paper apply to real physical systems in a similar way as they do in simulation. A database large enough to hold 6 trials was used in four learning runs. For two learning runs the standard method of overwriting the entire database with new experiences in a FIFO way was used. For the two other learning runs the first three episodes were kept in memory and the second half of the database was overwritten in a FIFO way by new experiences.



(a) x-position



(b) y-position

Figure 5: Effects of keeping early samples on the real setup. Responses when using the policies after 40 trials of 30 seconds are shown. A database that can hold experiences from 6 trials was used.

The policies that resulted after 30 trials were tested on the same sequence of reference positions. The response of the system for the four tests is shown in Figure 5. It can be seen that although the two learning runs which kept the initial episodes in the database produced useful policies, the method completely failed on both runs where the six most recent episodes were kept in memory.

5 Conclusion

This work investigates the effects of the properties of the experience replay database on the stability and speed of learning when using the Deep Deterministic Policy Gradient method in the context of control of 2-DOF robot control.

It is shown that the experience replay database is vital for the stability of the learning process. Most important in this respect is that the database should at all times contain experience samples that are diverse enough. Even after the neural networks have already converged to represent a decent policy and the value function of that policy, they can still loose this knowledge when trained on experiences that are too limited in their coverage of the state-action space.

Although the samples from early trials, which include thorough exploration, are valuable for the learning performance of the algorithm, the standard DQN and DDPG methods will eventually replace these samples with new experiences that might be sampled closer to the policy. This can result in a loss of stability and learning performance when these new samples are not diverse enough.

Even when a database is used that is large enough to retain all experience samples, the influence of the early samples on the neural network training will eventually diminish, as the ratio of the exploratory experiences compared to those close to the policy drops. This suggests that there is an exploration-exploitation trade-off not just for generating new experiences, but also for deciding which experiences in the database to replace.

It is shown in this work that retaining the experiences from the initial exploratory trials in the database can help prevent a loss of learning stability for very small databases both in simulation and on a physical setup and that using a relatively small database with this method can improve the performance of the learning algorithm compared to storing all experiences. However, care should be taken to prevent over-fitting to experiences that are kept in the database for too long.

A more sophisticated method to determine which experiences to overwrite in the database, such that a sufficient coverage of the state-action space is enforced, remains future work. The considerations could for example be based on closed loop system identification theory [19] or the temporal difference error of the experiences [20].

References

[1] Lillicrap, Timothy P., et al. ”Continuous control with deep reinforcement learning.” arXiv preprint arXiv:1509.02971 (2015).

[2] Mnih, Volodymyr, et al. ”Human-level control through deep reinforcement learning.” Nature 518.7540 (2015): 529-533.

[3] Mnih, Volodymyr, et al. ”Playing atari with deep reinforcement learning.” arXiv preprint arXiv:1312.5602 (2013).

[4] Silver, David, et al. ”Deterministic policy gradient algorithms.” ICML. 2014.

[5] van Hasselt, Hado, Arthur Guez, and David Silver. ”Deep Reinforcement Learning with Double Q-learning.” arXiv preprint arXiv:1509.06461 (2015).

[6] Hafner, Roland, and Martin Riedmiller. ”Reinforcement learning in feedback control.” Machine learning 84.1-2 (2011): 137-169.

[7] Kingma, Diederik, and Jimmy Ba. ”Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[8] Engel, Jan-Maarten, and Robert Babuška. ”On-line Reinforcement Learning for Nonlinear Motion Control: Quadratic and Non-Quadratic Reward Functions.” World Congress. Vol. 19. No. 1. 2014.

[9] Ioffe, Sergey, and Christian Szegedy. ”Batch normalization: Accelerating deep network training by reducing internal covariate shift.” arXiv preprint arXiv:1502.03167 (2015).

[10] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. ”Deep sparse rectifier neural networks.” International Conference on Artificial Intelligence and Statistics. 2011.

[11] Degris, Thomas, Martha White, and Richard S. Sutton. ”Off-policy actor-critic.” arXiv preprint arXiv:1205.4839 (2012).

[12] Bengio, Yoshua. ”Practical recommendations for gradient-based training of deep architectures.” Neural Networks: Tricks of the Trade. Springer Berlin Heidelberg, 2012. 437-478.

[13] LeCun, Yann A., et al. ”Efficient backprop.” Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

[14] Lin, Long-Ji. ”Self-improving reactive agents based on reinforcement learning, planning and teaching.” Machine learning 8.3-4 (1992): 293-321.

[15] Lange, Sascha, Thomas Gabel, and Martin Riedmiller. ”Batch reinforcement learning.” Reinforcement Learning. Springer Berlin Heidelberg, 2012. 45-73.

[16] Goodfellow, Ian J., et al. ”An empirical investigation of catastrophic forgeting in gradient-based neural networks.” arXiv preprint arXiv:1312.6211 (2013).

[17] Levine, Sergey. ”Exploring deep and recurrent architectures for optimal control.” arXiv preprint arXiv:1311.1761 (2013).

[18] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.

[19] Van den Hof, Paul. ”Closed-loop issues in system identification.” Annual reviews in control 22 (1998): 173-186.

[20] Stadie, Bradly C., Sergey Levine, and Pieter Abbeel. ”Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models.” arXiv preprint arXiv:1507.00814 (2015).

[21] Adam, Sander, Lucian Buşoniu, and Robert Babuška. ”Experience replay for real-time reinforcement learning control.” Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 42.2 (2012): 201-212.

[22] Wawrzyński, Paweł. ”Real-time reinforcement learning by sequential actor–critics and experience replay.” Neural Networks 22.10 (2009): 1484-1497.