Deep Reinforcement Learning Workshop, NIPS 2016

Off-policy experience retention for deep actor-critic 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

When a limited number of experiences is kept in memory to train a reinforcement learning agent, the criterion that determines which experiences are retained can have a strong impact on the learning performance. In this paper, we argue that for actor critic learning in domains with significant momentum, it is important to retain experiences with off-policy actions when the amount of exploration is reduced over time. This claim is supported by simulation experiments with a pendulum swing-up problem and a magnetic manipulation task. Additionally, we compare our strategy to database overwriting policies based on obtaining experiences spread out over the state-action space, and also to using the temporal difference error as a proxy for the value of experiences.

1 Introduction

Deep reinforcement learning is a very powerful machine learning technique. It combines the capabilities of reinforcement learning to solve sequential decision making processes with the generalization abilities of neural networks. Together, these methods make it possible to learn near optimal behavior policies that can even work in previously unseen states. This large potential comes at a price however, as deep reinforcement learning can be a complex problem. The main challenges concern the stability of the learning process and the speed of convergence to a solution.

Many recent advances in the field can be seen as ways of bringing the reinforcement learning problem closer to the easier problem of supervised learning. The reinforcement learning setting differs from supervised learning in that there is no a priori fixed database of *inputs* and corresponding correct *targets* to train the neural networks on. Instead, inputs are collected during the learning process, with the distribution of the inputs being dependent on the process itself. Additionally, these inputs do not come with correct targets. Instead, the correct targets, such as the value of a state or the optimal action for a given state, are the intended outcome of the learning process. As such, the function that is to be learned is changing during the learning process.

For value function based reinforcement learning methods, an important improvement (and step in the direction of supervised learning), is to change the targets more slowly [10, 7]. Additionally, using a large experience replay buffer [8] helps to break the correlations between updates to the network parameters. This restores the i.i.d. assumption that stochastic gradient descent methods are based on. It also means the distribution of the inputs changes less quickly. Other improvements involve

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

calculating the targets more precisely. For value based methods this can be done by calculating them from longer roll-outs (e.g. [12, 9, 5]). For actor only methods the targets can for example be calculated with optimal control methods [2].

In this paper we instead focus on *the distribution of the inputs* over the domain of the functions we are trying to learn. In supervised learning, some care is generally taken in collecting the training data. For classification this would entail having sufficient examples of all classes, whereas for regression it would come down to having the input samples sufficiently spread out over the domain of the function that needs to be learned. By actively curating the contents of the experience replay database, we again move reinforcement learning closer to supervised learning.

We are specifically concerned with actor critic methods in the latter stages of learning. In this phase, the policy is already near optimal, and only slight variations of the policy are tried to further improve it. In the limit, with the policy π converging to the optimal policy, the exploration could decay to zero. For a deterministic policy, this would mean that there is a direct relationship between the state s and the state-action value $Q(s, a)$ for the new experiences, since $a = \pi(s)$. Therefore, even if the critic correctly outputs the target $Q(s, a)$ value, it could forget the dependency on the action input. This breaks the learning method, as the derivative of this function with respect to the action is used as the policy gradient. Continued updates to the policy with the faulty policy gradient will change the policy in unpredictable and almost certainly unfavorable ways causing it to diverge from the optimal policy. Even when the updates are initially correct, changes to the policy can quickly bring it outside of the area of the state-action space that is covered by the experiences in the database. If the Q function is incorrect here, this could quickly lead to instability.

In this paper we will tackle scenarios in which the problem outlined above actually occurs in practice and not just in the limit of decaying the exploration. We will present a method to prevent the problem and show that this method enables using a much smaller experience replay buffer, which can be beneficial when using high dimensional inputs, or when performing deep reinforcement learning on systems with limited memory. We will also compare our method to other methods that change the distribution of the experiences that the neural networks are trained on. Related work is discussed in detail in Section 4.

2 Preliminaries

In this paper we use the Deep Deterministic Policy Gradient (DDPG) [7] actor-critic method as a starting point. The algorithm uses an actor and a critic. The actor π attempts to determine the real-valued control action $a \in \mathbb{R}^n$ that will maximize the expected sum of future rewards r based on the current state of the system $s \in \mathbb{R}^m$; $a = \pi(s)$. The critic Q predicts the expected discounted sum of future rewards when taking action $a(k)$ in state $s(k)$ at time k and the policy π is followed for all future time steps:

$$Q^\pi(s, a) = \mathbb{E}\Big[\big(r\Big(s(k), a, s(k+1)\big) + \sum_{j=k+1}^{\infty} \gamma^{j-k} r\big(s(j), \pi(s(j)), s(j+1)\big)\Big| s(k) = s\Big] \quad (1)$$

with $0 \leq \gamma < 1$ the discount factor, which ensures that the sum is finite.

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

$$\mathcal{L}(\xi) = \big(\big[r + \gamma Q\left(s', \pi(s'|\zeta^-)\big|\, \xi^-\right)\big] - Q(s, a|\xi)\big)^2 \quad (2)$$

Where $s = s(k)$, $s' = s(k+1)$ and $r = r(s, a, s')$ for brevity. The parameter vectors ζ^- and ξ^- are copies of ζ and ξ that are updated with a low-pass filter to slowly track ζ and ξ:

$$\xi^- \leftarrow \tau\xi + (1-\tau)\xi^- \quad (3)$$

$$\zeta^- \leftarrow \tau\zeta + (1-\tau)\zeta^-. \quad (4)$$

with the low-pass filter parameter $\tau \in (0, 1)$, $\tau \ll 1$. This makes the training targets more stable and thus moves the reinforcement learning algorithm closer to supervised learning.

The actor network is updated in the direction that will maximize the expected reward according to the critic:

$$\Delta\zeta \sim \nabla_a Q(s, a|\xi)|_{s=s(k), a=\pi(s(k)|\zeta)} \nabla_\zeta \pi(s|\zeta)|_{s=s(k)} \quad (5)$$

During the learning trials, the experience tuples $\langle s, a, s', r \rangle$ from the interaction with the system are stored in a database $\mathcal{D}$. During the training of the neural networks, the experiences are sampled uniformly at random from this database.

To obtain the experience tuples, learning trials are performed. These trials consist of E episodes of T time-steps each. During all training trials, the initial state of each episode is fixed for each environment; $s(0) = s_0$. Subsequent states depend on the deterministic environment dynamics function: $s(k+1) = f(s(k), a(k))$. The actions $a(k)$ are chosen according to $a(k) = \pi(s(k)|\zeta) + \mathcal{E}(k)$.

Here, $\mathcal{E}$ is the exploration term. We use an Ohrnstein-Uhlenbeck process, which is beneficial for dynamical systems with momentum [7]:

$$\mathcal{E}(k) = \mathcal{E}(k-1) - S_e \alpha_0 \mathcal{E}(k-1) + S_e \beta_0 \mathcal{N}(0,1) \tag{6}$$

With $\mathcal{N}(0,1)$ a Gaussian noise process with mean 0 and standard deviation 1, α_0 and β_0 constants that are chosen to excite the system at relevant frequencies, and S_e a scaling factor.

The scaling factor S_e is used to decay the exploration during the learning trial. We start all trials with $S_e = 1$ and decay linearly per episode until some minimum amount of exploration $S_{min} > 0$.

3 Method

When the amount of exploration is reduced too far, reinforcement learning can break down [1]. This is because the newly sampled experiences do not sufficiently cover the domain of the functions we are learning. Even when a function was previously learned that holds over the relevant domain, this function can be forgotten when training is continued with different data [4].

The problem we are specifically interested in here concerns the critic in actor-critic learning methods such as DDPG. The critic needs to learn the mapping from the states and actions to the expected future sum of rewards when taking action a in state s and following the policy for all time-steps after the initial one $Q(s, \pi(s) + \mathcal{E})$. When we reduce the amount of exploration $\mathcal{E}$, the actions in the new experiences are going to be closer to the policy actions, so the influence of the action in this mapping is going to diminish.

In the limit, the on-policy experiences would only contain information about the state-value function $V(s)$, not the state-action value function $Q(s, a)$, since $a = \pi(s)$. The critic network might then learn to return the correct $Q(s, a)$ values for the samples we train it on based only on the states s. This would mean that the derivative with respect to the actions $\nabla_a Q(s, a|\xi)$, used in (5) would not contain any useful information. This could completely break the reinforcement learning method. Additionally, even if the approximation of the effect of a on $Q(s, a)$ is accurate near the policy, changes to the policy can quickly move it outside of the coverage of the recent data. If the approximation of Q is inaccurate for these new actions, instability can quickly occur as the policy is further changed based on the incorrect derivative of Q here.

Although reducing the exploration to zero while learning continues would not make sense, problems can already start to occur long before this point. This is especially true for dynamical systems with considerable momentum, where the next state is largely determined by the current state and the influence of the action on the next state is small. An example of a domain where these conditions are present are low-level control applications. There, the high sampling frequencies that are required for accurate reference tracking or disturbance rejection [3] mean the effect of taking a random action for only a single time-step are very small. The use of the Ohrnstein-Uhlenbeck exploration noise is beneficial for exactly this reason; taking a random action for only a single time-step is not informative enough; it will not enable escaping stable equilibria for example.

When the amount of exploration is reduced, the already small effects of the action a on the $Q(s, a)$ value become even smaller. At some point, especially when regularization is used on the critic network as suggested in [7], there might no longer be sufficient incentive to correctly model the dependency of $Q(s, a)$ on a.

Given these problems, it might seem beneficial to simply not reduce the amount of exploration during the trial. However, the exploration decay might be crucial for at least two reasons:

The first is a practical one and applies when using deep reinforcement learning to learn control policies for physical systems. These systems often cannot sustain long periods of rough exploration without suffering from increased wear or damage.

The second reason to decay the amount of exploration is that we are often interested in controlling these systems close to an unstable equilibrium state. This means that our control policy needs to be very precise around this state. The same amount of precision is not needed further away from the desired state. By reducing the exploration when a policy has been found that will bring us close to the desired state, we ensure that more samples are obtained around the desired state. This will speed up convergence towards an optimal policy. In addition, it might improve the final policy as the networks can specialize in the relevant parts of the state-space.

The need to reduce the amount of exploration needs to be combined with the need to have the experiences in the experience replay database maximally informative about the effects of actions in relevant states. As in our earlier work [1], we will influence the distribution of the contents of the experience database over the state-action space by overwriting it in a nonstandard way. This makes it possible to use smaller databases than otherwise possible, which makes the method especially relevant on domains with high dimensional inputs such as images, where saving all experiences might be impossible.

The default implementation of experience replay is to save the experience tuples $\langle s, a, s', r \rangle$ to an experience database $\mathcal{D}$ until the database is full. Then, we simply start overwriting the database from the beginning in a First In First Out (FIFO) way.

In the methods we consider in this paper, we still always write the most recent experiences to the database. This is done to ensure we change at least some of the data in the databases, to reduce the risk of over-fitting to individual samples. The new experiences will, as the exploration decays, have actions that are close to the policy action in their state. We want to make sure that the effect of taking different actions in those states can be deduced from the samples in the database. Therefore we will try to keep sufficient experiences in memory with *off-policy* actions; preferably actions that maximally differ from the policy actions. To do this we will overwrite the experiences in the database that have the most *on-policy* actions with new experiences.

The metric we use to determine which experiences to overwrite is the squared two norm of the distance between the action a_i of experience i in the database $\mathcal{D}$ and the current policy action for the state s_i of experience i:

$$\text{OFFPOL score}(i) = \|a_i - \pi(s_i|\zeta)\|_2^2 \tag{7}$$

We propose two alternatives methods that use this principle. The first is to split the database $\mathcal{D}$ into two smaller databases $\mathcal{D}_F$ and $\mathcal{D}_O$, each half the size of $\mathcal{D}$. After each episode e, the T new experiences obtained during the episode are written to both databases. The database $\mathcal{D}_F$ is overwritten in a FIFO way and as such contains the most recent (increasingly on-policy) experiences. The database $\mathcal{D}_O$ is the off-policy database. When full, the T experiences with the lowest score according to (7) are overwritten with the new experiences from the latest episode.

When training the neural networks, experiences are sampled with probability Ω from the off-policy database $\mathcal{D}_O$ and with probability $1 - \Omega$ from the FIFO database $\mathcal{D}_F$.

The second method we consider is to simply only have the off-policy database; $\mathcal{D} = \mathcal{D}_O$. This does not allow for any control over the mixing of on- and off-poly experiences, as these are now a result of the capacity of $\mathcal{D}$ and the number of experiences per episode T. We do however prevent writing the same experiences to two databases, which might be wasteful for small databases.

4 Related work

This work is related to our earlier work [1] which attempts to influence the distribution of the samples in the experience database over the state-action space by changing the way the database is overwritten when it is full. In contrast to the earlier work, we do not attempt to obtain a uniform spread of the samples over the state-action space, but instead strive for having sufficient *off-policy actions*, ideally in the *on-policy states*. Our method however only performs a test in the action space, which means we avoid having to calculate distances in the state-space. This is beneficial since the state-space is, more often than the action space, very high dimensional and structured. In addition, since we are

interested in distances as opposed to densities, the computational demands of the method presented in this paper scale linearly with the size of the experience database, as opposed to quadratically in [1]. This is because for each experience in the database we only check the distance between the action and the policy action. For the density based method, the distance of every sample to every other sample in the database is checked. This change makes it possible to use our method in combination with much larger databases.

In [11], another method is presented that changes the distribution of the training samples for the neural networks. This method saves *all* experiences in a database and bases the probability of using a sample in a training batch on the temporal difference error made by the critic when trained on that sample. Doing this indirectly implies that the learned functions should be equally accurate over the whole (explored) state-space. Our method instead tries to solve the problem that in environments characterized by large momentum, the dependency of the $Q(s, a)$ values on the actions a might be forgotten when the amount of exploration is reduced. This could theoretically happen even when the $Q(s, a)$ values themselves are correctly predicted based on the states s alone. In this case, the temporal difference error would not be a good proxy for the usefulness of experiences as the error would be low in spite of the fact that the dependency on the action is not correct.

5 Experiments

We test our method on two benchmarks: a pendulum swing-up task and a magnetic manipulation problem.

Both problems are simulated with a sampling frequency of 50 Hz. The dynamics of both problems are defined as differential equations, which we use to calculate the next state s' as a function of the current state s and action a using the (fourth order) Runge-Kutta method. The reward is in both cases given by:

$$r(s, a) = -(W_1|s - s_{\text{ref}}| + W_2|a|) \tag{8}$$

In both cases a fixed reference state s_{ref} is used.

5.1 Pendulum swing up

For the pendulum swing up task, the state s is given by the angle $\theta \in [-\pi, \pi]$ and angular velocity $\dot{\theta}$ of a pendulum, which starts out hanging down under gravity $s_0 = [\theta\ \dot{\theta}]^T = [0\ 0]^T$. The action space is one dimensional and is the voltage applied to a motor that exerts torque on the pendulum $a \in [-3, 3]$ V. The angular acceleration of the pendulum is given by:

$$\ddot{\theta} = \frac{-Mgl\sin(\theta) - (b + K^2/R)\dot{\theta} + (K/R)a}{J} \tag{9}$$

Where $J = 9.41{\cdot}10^{-4}$, $M = 5.5{\cdot}10^{-2}$, $g = 9.81$, $l = 4.2{\cdot}10^{-2}$, $b = 3{\cdot}10{-}6$, $K = 5.36{\cdot}10^{-2}$ and $R = 9.5$ are respectively the pendulum inertia, the pendulum mass, the gravity constant, pendulum length, viscous damping coefficient, the torque constant and the rotor resistance. For this task $W_1 = [5\ 0.1]$, $W_2 = 1$ and $T = 200$ (4 seconds). The pendulum needs about a second to swing up by first moving to one side and then swinging in the other direction. It then needs to stabilize around the unstable upright equilibrium position.

5.2 Magnetic manipulation

The second problem we consider is a magnetic manipulation task. Here, a metal ball needs to be accurately positioned on a 1-D track by dynamically changing a magnetic field. This is done by controlling the current through four electromagnets under the track; $a_i \in [0,\ 0.6]$ for $i = 1, 2, 3, 4$. The state of the problem is defined as the position $x \in [-0.035,\ 0.105]$ of the ball relative to the center of the first magnet and the velocity $\dot{x}$ of the ball: $s = [x\ \dot{x}]^T$. When the position of the ball exceeds the bounds, the position is set to the bound and the velocity is set to zero. The acceleration of the ball is given by:

$$\ddot{x} = -\frac{b}{m}\dot{x} + \frac{1}{m}\sum_{i=1}^{4} g\,(x, i)\,a_i \tag{10}$$

with

$$g(x,i) = \frac{-c_1\ (y - 0.025i)}{\left((y - 0.025i)^2 + c_2\right)^3}. \tag{11}$$

Here, $g(y,i)$ is the nonlinear magnetic force equation, $m = 3.200 \cdot 10^{-2}$ [kg] the ball mass, and $b = 1.613 \cdot 10^{-2}[\frac{Ns}{m}]$ the viscous friction of the ball on the rail. The parameters c_1 and c_2 were empirically determined to be $c_1 = 5.520 \cdot 10^{-10}\mathrm{Nm}^5\mathrm{A}^{-1}$ and $c_2 = 1.750 \cdot 10^{-4}\mathrm{m}^2$.

For the magnetic manipulation problem we take $W_1 = [10\ 0.5]$, $W_2 = [0\ 0\ 0\ 0]$, $T = 100$ (2 seconds), $s_0 = [0\ 0]^T$ and $s_{\text{ref}} = [0.035\ 0]$.

5.3 DDPG implementation

For both problems, we use an actor network with two hidden layers of 50 units with ReLU non-linearities, and respectively 1 or 4 $\tanh$ output units. The outputs of these units are scaled to the correct range. The critics have, for both problems, three hidden layers with 50, 50 and 20 units respectively. The action inputs are fed into the network after the first hidden layer. In both problems we use $\gamma = 0.95$ for the forgetting factor.

To train the networks, ADAM [6] is used with a learning rate of $1 \cdot 10^{-4}$ for the actor and $1 \cdot 10^{-2}$ for the critic. The lowpass filter for the target network parameter updates is set to $\tau = 1 \cdot 10^{-2}$. A batch size of 16 is used.

6 Results

First, we would like to know if the proposed method indeed improves the performance for smaller databases over the default FIFO method, specifically when the amount of exploration is reduced over time. To test this we perform experiments on the magnetic manipulation and pendulum swing-up benchmarks. In these experiments, the exploration scaling S_e is linearly reduced from 1 to 0.1 over the first 500 episodes. It is then kept at 0.1 for the remainder of the trial.

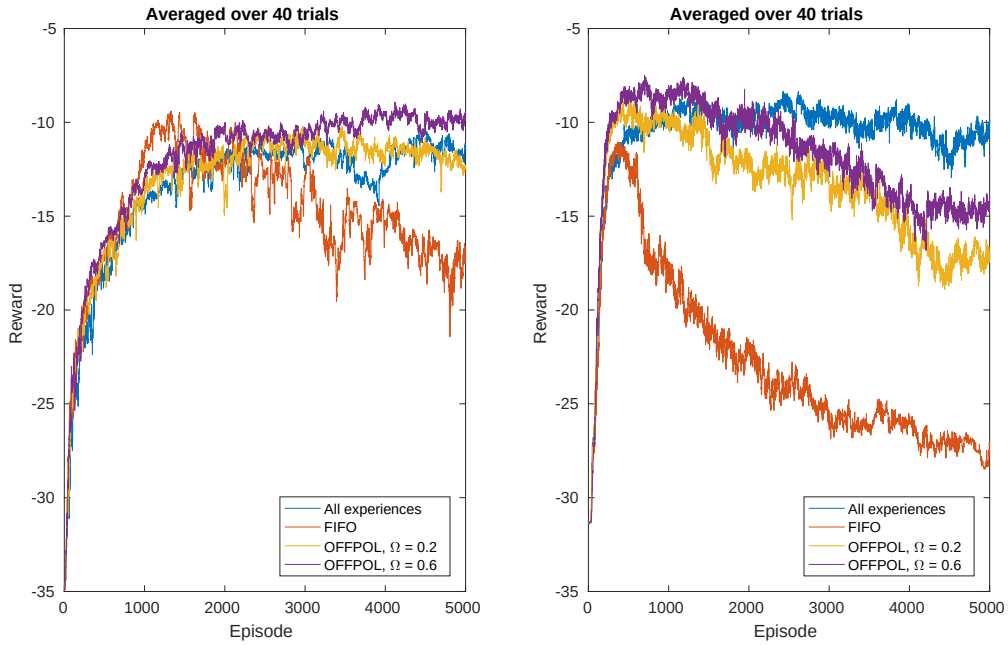
In the test we compare using a database that is large enough to contain all experiences with using a smaller database $\mathcal{D}$ with a total capacity of $5 \cdot 10^4$ experiences. On the smaller database we compare a FIFO strategy with our strategy of splitting it in a FIFO database $\mathcal{D}_F$ and an off-policy database $\mathcal{D}_O$.

On the magnetic manipulation task this means that the FIFO strategy database contains the experiences of the 500 most recent episodes. Using our OFFPOL strategy, $\mathcal{D}_F$ contains the experiences of the 250 most recent episodes and $\mathcal{D}_O$ contains 250 episodes worth of experiences in which the actions should differ from the current policy. On the pendulum task, for which the episodes are twice as long, the small databases contain the experiences of half these numbers of episodes.

Figure 1 shows the results of these experiments. For both tasks, the strong loss of performance of the FIFO method as its contents become *too* on-policy is clearly visible. For both benchmarks our OFFPOL method, by more selectively overwriting a database of the same size, produced much better results. It can also be seen that the method is not overly sensitive to Ω, the one hyper-parameter that we introduce.

In the magnetic manipulation task it can be seen how a good mix of experiences with on-policy and off-policy actions can aid the reinforcement learning method performance. While keeping only a tenth of the experiences in memory, our method outperforms having all experiences in memory. Also interesting here is the performance of the FIFO method. Our method attempts to maintain a certain distribution of on- and off-policy experiences. In contrast, the distribution in the database that uses the FIFO method starts to rapidly shift towards on-policy after 500 episodes, as the old exploratory experiences are starting to be overwritten by the new experiences with actions that only deviate slightly from the policy actions. Before this shift results in the problems predicted in Section 3 however, the performance actually starts to increase relative to the performance of the other methods. During this phase the distribution of on- and off-policy experiences might be closer to being optimal than that resulting from using the other methods.

For the magnetic manipulation task our method manages to prevent the loss of performance resulting from overwriting the database naively. For the pendulum task, better performance compared to saving



(a) Performance on the magnetic manipulation task (b) Performance on the pendulum swing-up task

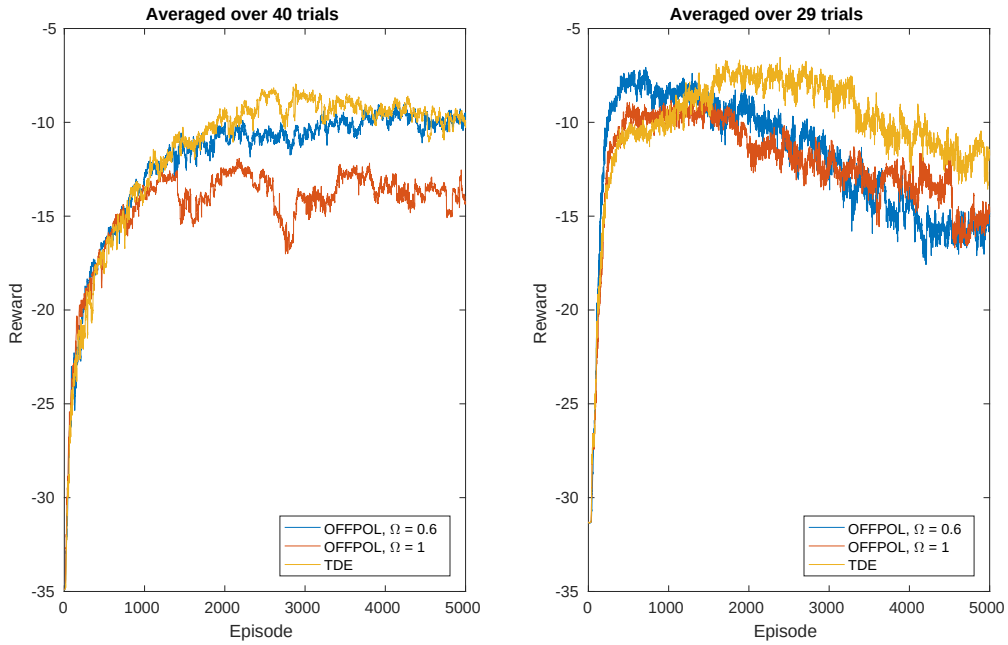
Figure 1: Performance of the proposed method with a database of $5 \cdot 10^4$ experiences. The total number of experiences (5000 episodes) are $5 \cdot 10^5$ for the magnetic manipulation task and $1 \cdot 10^6$ for the pendulum swing-up task.

all experiences is initially obtained. The method, however, cannot prevent a drop in performance over time, although the drop is much less significant than the drop resulting from the use of the FIFO method. A reason for this might be sought in the fact that our focus on the action space is more appropriate for the magnetic manipulation problem. There, the action has four components whose influence is highly dependent on the location in the state space. For the pendulum problem, the effect of actions is more predictable and possibly easier to learn.

We also compare our method to other methods that change the distribution of the experiences presented to the networks for training. The first method, based on the prioritized experience replay method presented in [11], uses the temporal difference error as a proxy for the value of experiences. Since we are interested in methods that work with smaller databases, we do not save all experiences and sample according to the temporal difference error as in [11]. Instead, we use the TDE to determine which experiences in the database to overwrite with new ones. We then sample uniformly from this database. We use the same tests as discussed previously, and compare this to using our method with two databases and our method with only an off-policy database. The results are shown in Figure 2.

It can be seen from Figure 2 that the temporal difference error does in fact offer a very competitive guide on which experiences to retain in memory. On the magnetic manipulation test, the maximum performance is sightly better for the TDE based method, while on the pendulum task the maximum performance is about equal. On both tasks, a drop in performance after the initial peak can be observed for the TDE method, which offers a motivation to further investigate how to evaluate which experiences are needed to keep deep reinforcement learning methods stable.

Finally, we compared our new method to earlier work [1], in which the database is overwritten in a way that ensures the experiences that are retained are maximally spread out over the state-action space. Due to the computational complexity of this method we only performed the comparison on a very small database of 1000 samples. On a database of this size, the method of [1] performed best. However, it failed to reach the same level of performance as in the experiments of Figure 1.



(a) Performance on the magnetic manipulation task (b) Performance on the pendulum swing-up task

Figure 2: Performance of the proposed method compared to using the temporal difference error. Both use a database of $5 \cdot 10^4$ experiences. The total number of experiences (5000 episodes) are $5 \cdot 10^5$ for the magnetic manipulation task and $1 \cdot 10^6$ for the pendulum swing-up task.

7 Conclusion

In this paper, a specific manner in which deep actor-critic learning can break down was discussed. We hypothesized that, especially for problems that are characterized by both large momentum and small experience replay databases, the critic might unlearn the dependency of the state-action value function on the action input when the amount of exploration is reduced. To address this issue, we proposed to overwrite the experience database in such a way as to explicitly keep experiences with off-policy actions in memory. When compared to naively overwriting the experience database in a First In First Out (FIFO) manner, we showed that our method indeed yields significant improvements. On both of the benchmarks used, the maximum performance of our method, which kept at most one tenth of the total number of experiences in memory, even exceeded that of retaining all experiences.

We also compared our method to using the temporal difference error as a guide for which experiences to retain in memory. In theory, using the temporal difference error for this purpose is problematic, as a temporarily low error in a part of the state-action space could lead to losing all samples in this region. Once the value approximation for this region becomes worse due to the lack of training data, there is no way to reclaim the lost experiences. In practice we found that using the temporal difference error worked at least as well as our method, although a more thorough comparison is still to be made.

Given these results, a more optimal strategy for small databases might be to retain experiences based on some stable criterion such as diversity in the (state)-action space, while replaying the experiences based on their temporal difference error. This way, the stability improvements of diversity based methods can be combined with the overall learning performance improvements of prioritized experience replay.

Next steps are investigating the effects of several different experience retention strategies, and their interplay with experience sampling strategies. We will attempt to link their applicability to the presence of noise, the momentum of the environment and the required precision of the controller.

Acknowledgments

This work is part of the research programme Deep Learning for Robust Robot Control (DL-Force) with project number 656.000.003, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

References

[1] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Improved deep reinforcement learning for robotics through distribution-based experience retention. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[2] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. arXiv:1509.06113 [cs.LG], 2015.

[3] Gene F Franklin, J David Powell, Abbas Emami-Naeini, and J David Powell. *Feedback control of dynamic systems*, volume 2. Addison-Wesley Reading, 1994.

[4] I Goodfellow, M Mirza, X Da, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. arXiv:1312.6211 [stat.ML], 2013.

[5] Matthew Hausknecht and Peter Stone. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI Workshop*, New York, July 2016.

[6] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Timothy P Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[8] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

[9] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. arXiv:1602.01783 [cs.LG], 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv:1511.05952 [cs.LG], 2015.

[12] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. arXiv:1506.02438 [cs.LG], 2015.