Deep Reinforcement Learning from Expert's Experience Replay

Tal Zadik and Alon Magrafta
Supervisors: Tom Zahavy, Daniel J. Mankowitz, Shie Mannor
CRML Laboratory, Electrical Engineering Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel

April 24, 2016

Abstract

Experience replay (ER) [Lin (1993)] serves as a way for online reinforcement learning agents to remember past experiences, and learn from them in future times. Experience replay plays an important role in state of the art architectures in the field of deep reinforcement learning, such as the vanilla DQN [Mnih et al. (2015)]. In this work, we explore how the ER's attributes, such as the size and amount of exploration within it, affect the learning quality. In addition, we investigate the effect of dynamic data sets, which are common in RL domains, on the learning process. We use ER generated by a teacher, or an expert, agent for instructing a student agent. We conducted our experiments on two domains: *MNIST* digits classification and *Atari 2600*, where we have tested on the games *Road Runner* and *Breakout*. We propose that both, the diversity of policies used for generating the ER and the size of the ER, have strong correlation with the agent's performance, such that larger ER that contain more exploration are preferable. Moreover, we show that the neural network's weights, which represents the agent policy, diverge, after certain amount of learning steps, when using immutable ER. Finally, we show that an RL agent that is taught from an expert's experience can generalize well. |

Introduction

Recent years have seen the emergence of deep reinforcement learning, which has shown that RL agents can achieve human results and exceed them, in domains such as Atari 2600 and the game 'Go' [Silver et al. (2016)]. The deep Q-network (DQN) framework [Mnih et al. (2015)] gets image's pixels and rewards, observes the world state, and chooses an action which maximizes the expected future rewards. In order to estimate Q-function values, the DQN uses a convolutional neural network as a function approximator. One of the key components of the DQN algorithm is the use of experience replay to reduce correlations between consecutive sampled observations.

In this paper we investigate how aspects of the training data affect the learning process and the performance of an RL agent, specifically we investigated the experience replay properties. One of the issues in RL is the existence of rare experiences that are hard for the agent to discover due to insufficient exploration or hierarchical nature of the environment. This causes the revelation of some states only when the agent performs advanced performance. This notion led us to the idea of exploring the influence of dynamic data sets on an agent's performance in terms of score / success rate and training length. These issues haven't been explored yet, to the best of our knowledge.

There have been a few studies relating to the order in which the examples should be presented to the agent. Among them there are some which concern about using prior knowledge to manually define learning curriculum [Bengio et al. (2009), Karpathy and van de Panne (2012)]. Others relate to choosing the order in an automatic fashion, such that the most meaningful examples (under some criterion) are chosen first [Schaul et al. (2015), Kumar, Packer, and Koller (2010)]. Another study deals with a combination of the two mentioned methods [Jiang et al. (2015)].

A lot of aspects of the ER have been left unexplored such as the importance of the amount of exploration within the ER and its size. We used an expert's experience to train a student agent, as in [Rusu et al. (2015)], resulting in an off-policy learning setting. Our main contributions in this work are the following observations:

- The learning is more effective, if the ER contains trajectories from multiple policies, rather than from a single policy (that comes from one trained network). That is to say that it is better to use ER that contains policy exploration, that comes from changes in the network's weights.
- When learning from an immutable (or fixed) ER, the ER size affects the agent's performance. Larger ER brings the agent to better performance. In addition, the networks weights diverge after the exhaustion of the fixed ER's information.
- Efficient learning can be done by following a fixed expert's experience (the agent can generalize well).
- The dynamic nature of the training data available to an RL agent has negligible effect on the agent's performance (given that eventually all data is available for the agent).

Background

- Reinforcement learning (RL) is a field in machine learning in which an **agent** tries to improve in performing a task, based on a reward signal which gives feedback to its decisions. The time domain is discretized, and in each

time step t the agent observes a world **state** s_t, chooses an **action** a_t, and then gets a **reward** r_t and transitions to state s_{t+1} . The mapping from states to actions is called a **policy** and denoted by $\pi = P(a|s)$. The agent's goal is to maximize its expected cumulative future rewards, which is done by minimizing a cost function. The discount factor $\gamma \in [0, 1]$ defines the importance of immediate rewards over future rewards.

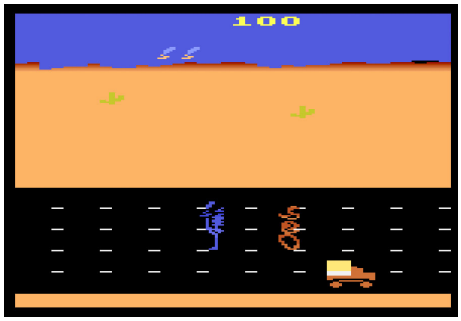

Figure 1: Atari 2600 Road Runner

- Action-value function, which is also known as **Q-function** $Q_t(s, a)$, gives a value for each state-action pair, which is the expected accumulated future rewards from time step t onward given $s_t = s$, if we choose $a_t = a$ and then proceed optimally. It is easy to see that the Q-function defines a policy by the relation

$$\pi^Q(s) = \arg\max_a Q(s, a)$$

Thus, the optimal policy can be derived from the optimal Q-function. **Q-learning** [Watkins and Dayan (1992)] is an iterative algorithm for evaluating the optimal Q-function values based on the following iterative step

$$Q_{t+1}(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} Q_t(s', a')$$

where S is the set of all possible states, A is the set of all possible actions, and p is the transitions probability function.

- **Deep Q-network (DQN)** [Mnih et al. (2013), Mnih et al. (2015)] is a model free approach to reinforcement learning, used in the Atari 2600 domain, that uses deep neural networks as function approximators, with only raw image pixels and reward signal as inputs. The DQN is used to estimate the optimal Q-function [Krizhevsky, Sutskever, and Hinton (2012)]

$$Q^*(s, a) = \max_\pi E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a, \pi]$$

which is the maximum cumulative discounted reward, when starting from state s, and taking action a while following policy π. A convolutional neural network (CNN) weights θ are optimized to estimate the Q-function values. Thus, the Q-function depends also on θ, i.e. $Q(s, a; \theta)$. The agent's policy is then derived by choosing the action with the highest value, given a state. The DQN introduced two key ideas in order to stabilize the learning process and decorrelate the sampled observations sequence:

1. **Experience replay (ER)** [Lin (1993)] is a buffer that stores previous tuples $\{s_t, a_t, r_t, s_{t+1}, term\}$, that represents the agent's experience, where $term$ indicates whether s_{t+1} is terminal state. The ER contains trajectories generated by following a single or multiple policies. During learning the samples are drawn uniformly at random from the ER, thereby reducing temporal correlations in the observation sequence.
2. The DQN maintains two different networks, the current network with weights θ and a target network with weights θ_{target} that is only updated to θ periodically. The current network weights are updated towards the target weights, thereby reducing correlations with the target.

At iteration i the DQN uses a sample from the ER to update the weights towards minimizing the following loss:

$$L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^{target}) - Q(s, a; \theta_i))^2]$$

Method

As mentioned before, dynamic data sets are common in RL domains, and therefore in ER. Hence, in the first part we wanted to check the implications of the dynamic nature of the ER on the learning efficiency. For that purpose, we decided to experiment with a simpler domain of supervised learning. We used MNIST database, and trained a digit classifier. We conducted experiments in which part of the training examples weren't accessible to the agent from the beginning. Then we gradually exposed the agent to the missing data, at different stages.

In the second part we based our work on the Vanilla DQN architecture [Mnih et al. (2015)], with some modifications. We added a mechanism for saving and loading the ER buffer from the disk. We adopted the concept of teacher and student networks presented in [Rusu et al. (2015)]. We also refer to the teacher as 'expert'. The expert network is used for generating experience replay to be later loaded and used by the student network.

Experience replay attributes

The ER stored by the teacher is generated during a phase when the teacher's performance is relatively high. We distinguish between two types of ER, depending on the characteristics of the teacher network that generated it:

1. **Fixed ER**, denoted by ER_f- generated from a trained teacher network, while following a fixed policy (after the completion of the teacher's training).
2. **Varying ER**, denoted by ER_v- generated during the teacher's training, hence the ER is effectively comprised from transitions extracted from different networks (because the network weights keep changing during training). Note that the network weights stand for a certain

policy, therefore this type of ER contains trajectories from different policies.

The ER was divided into train set and test set. During the student's training, it uses the expert's ER exclusively, i.e. the student itself doesn't store tuples to the ER buffer. Note that the ER that is used by the student is fixed during all of it's training, as opposed to the Vanilla DQN where the ER functions as a sliding window for the previous transitions and thus keep changing. Also, note that this is an off-policy learning setting.

Hyper parameters

Some of the DQN hyper parameters have been adjusted for the student use case:

1. $learn\_start$ was set to zero, because there is no need to fill the ER, which is provided by the expert.
2. ep, which controls the amount of the exploration the agent performs, was set to a fixed value of 0.1, instead of decreasing gradually from 1 to 0.1.
3. $replay\_memory$, which defines the ER size, was set to values between 100K to 1 million.
4. We experimented with different $seed$ and lr (learning rate) values.

Learning methods

We used two types of cost functions for the learning step:

1. Q-learning, as in the Vanilla DQN.
2. Supervised learning. In that case, we used the expert's Q-function values as target values.

Evaluation methods

The student's performance was evaluated in two ways:

1. Game score, achieved during a period when the student interacts with the simulator, without performing any learning, as in the vanilla DQN paper. We compared the student's score as opposed to the score achieved by the expert.
2. We measured the MSE of the student's Q-function versus the expert's Q-function values. The states that were used for this measurement were taken, separately, from both the train ER set and the test ER set.

Experiments

The first domain for our experiments was based on MNIST database of handwritten digits. MNIST is often used by the machine learning community as a "toy" dataset, as in our case, due to its relatively small size and simple nature. We used an of-the-shelf CNN for training a digit classifier with supervised learning. The second domain used for the experiments was Atari 2600 platform, in particular we experimented on the games *Road runner* and *Breakout*.

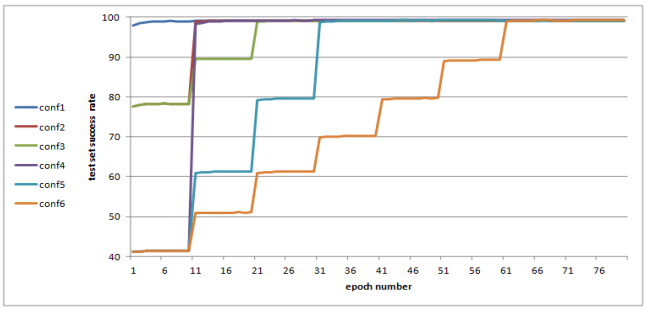

Figure 2: MNIST tests results

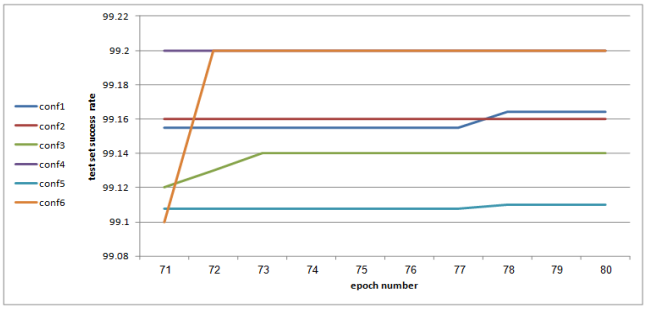

Figure 3: MNIST last epoch's tests results

MNIST

On MNIST domain we examined the effect of dynamic data sets on an agent's learning performance, by gradually exposing the agent to data in a selective way. We evaluated the agent's performance w.r.t speed and value of convergence (of classification success rate). Each test used different configuration that defined what part of the data is available to the agent at what stage. The definition of each configurations is given in Table 1. The results are shown in Figure 2 and Figure 3.

There are a few things to notice in these figures:

1. The lack of data causes a significant reduction in the digits classification success rate. Every digit's data is adding about 10 percent to the success rate, which corresponds to the frequency of each digit within the database.
2. Eventually, all the configurations converge to a close percentage of success in digits categorizing.

Epoch	1	11	21	31	41	51	61
conf1	0-9						
conf2	0,1,3,4,5,6,8,9	2,7					
conf3	0,1,3,4,5,6,8,9	2	7				
conf4	2,4,5,8	0,1,3,6,7,9					
conf5	2,4,5,8	0,1	3,6	7,9			
conf6	2,4,5,8	0	1	3	6	7	9

Table 1: MNIST tests configurations. Each line describes a configuration- which digits become available to the agent at what epoch.

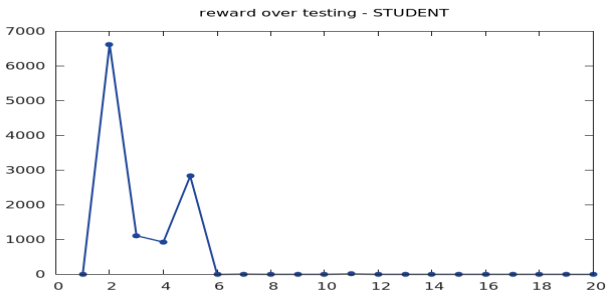

Figure 4: RL student's reward over 100K ER ($seed$=3, lr=0.00025)

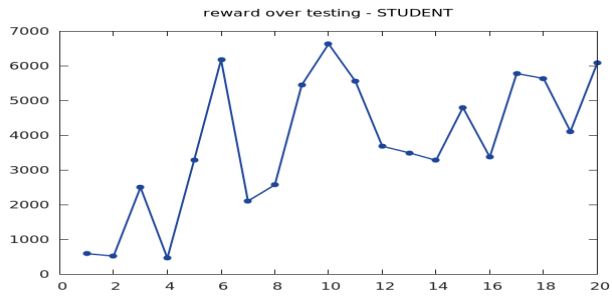

Figure 5: SV student's reward over 100K ER ($seed$=1, lr=0.00025)

Atari 2600

On Atari 2600 domain, we investigated the effect of different properties of the experience replay on an agent's learning performance. We tested different working premises:

First setting An *expert* agent was set to produce an ER of 100K tuples. The ER was then divided into two sets: *train set* and *test set*. A second agent, a *student* agent, was set to learn the task of playing *Road Runner* from the *train set*. We used reinforcement learning (RL) and supervised learning (SV). The learning curves (rewards) of the students are shown in Figure 4 and Figure 5. As shown in Figure 4, the RL student did not manage to do a significant learning process, compared to the teacher's score of approximately 20K. Moreover, the student's network weights diverged (reflected on the drop in the reward). Figure 5 shows that the SV student has succeeded to do some learning, but still not to get close to the teacher's reward level. To try and get the student to learn, we tried different $seed$ and lr values and evaluated the student's performance again:

- lr - changed by factor of [10, 25, 100]
- $seed$ - we used the values of 1, 3 and 11.

Changing the lr and the $seed$ did not bring the network to converge and the student's performance remained low. The results were similar to those shown on Figure 4.

Second setting As before, a *teacher* agent was set to produce an ER, this time with a size of 1M tuples, which were divided, as well, into (*train set* and *test set*). We used default values for the lr and $seed$ parameters on all the simulations. We then tested the following configurations:

- Two **game domains** - *Road Runner* and *Breakout*.
- Two ***teacher* networks** [*A,B*] per domain- for credibility issues.
- Two **ER types** - fixed ER, ER_f, and varying ER, ER_v.
- Two **cost functions** - for reinforcement learning (RL) and for supervised learning (SV).

In each test, a *student* agent was set to learn a *game* from an *ER* according to a *cost function*. The results are shown on Figure 6 and Figure 7.

From Figure 6 we deduce the following hypotheses:

1. Exploration is needed to achieve significant learning.
2. The RL student is capable of achieving good performance.
3. The weights of the RL student network diverge after certain number of steps and the reward drops to zero (as it did in the first setting).
4. In all four configurations, the student managed to do generalize well over the test set. This is reflected in the MSE plots.

From Figure 7 we deduce the following insights:

1. None of the configurations were able of achieving a significant reward.
2. Only the SV method managed to generalize over the test set.

Third setting In order to understand how does the size of the ER affect the learning process we tested a student agent using varying ER in different sizes, produced by the same expert agent, on the task of playing the game of Road Runner. We set an expert agent to produce five different sizes (number of tuples) of ER: 200K, 400K, 600K, 800K, 1M. For each different size we tested the reward and the MSE of the Q-function.We noticed that, qualitatively, all of the simulations behaved in a similar way. Shortly after the reward reached a peak value, the student's network weights has diverged and the reward dropped to zero. The divergence point is reached later and the maximum reward gets higher as the ER size gets bigger. In addition, from the size of 600K onward the student's performance is very close to the expert's, and even better. Hence, we propose that 600K is a big enough ER size, for effective learning.

ER Size	Max reward	Approx. divergence point (#steps)	Overall #steps
1M	24901.11	3M	5M
800K	24845.87	3.5M	5M
600K	21406.59	2.25M-2.75M	5M
400K	18336.72	1.25M-1.75M	5M
200K	9212.96	1M-1.5M	3.5M

Table 2: Road Runner different ER sizes results. The expert reached maximum reward of 21712.75 after 3.5M steps.

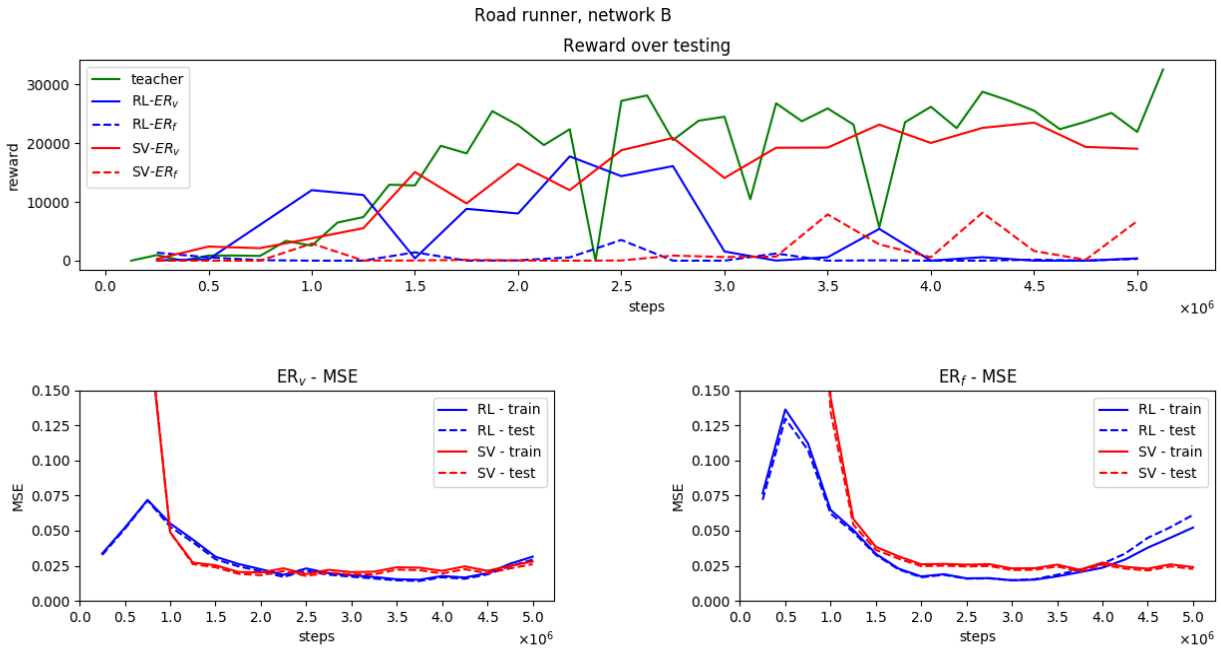

Figure 6: Road Runner second section comparison
(ER of 1M, network B, RL vs. SV, fixed vs. varying ER)

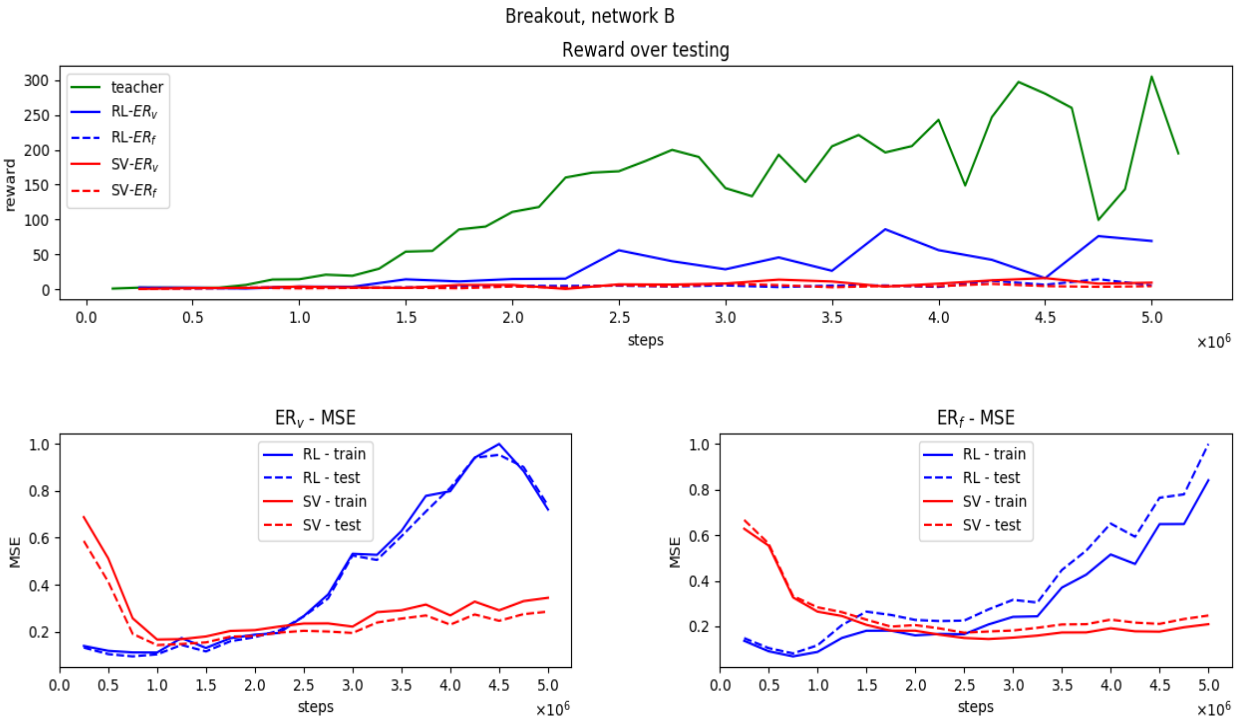

Figure 7: Breakout second section comparison
(ER of 1M, network B, RL vs. SV, fixed vs. varying ER)

For each configuration, we examined the *Max reward* and the approximated *divergence point*. The results are summarized in Table 2. An example of such single test result is shown in Figure 8.

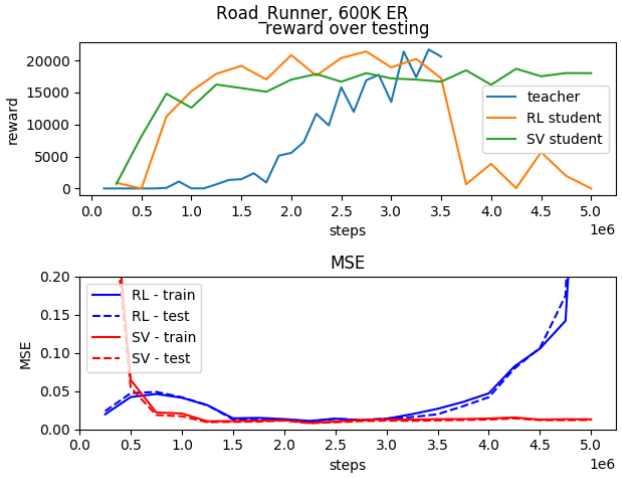

Figure 8: Reward and MSE, 600K ER

Discussion

In this work, we have explored the effect of the training examples over the learning efficiency of an RL agent, mostly using a replay memory, in settings where an expert's experience is used as training data for a student agent. From the tests we performed on MNIST domain we conclude that the influence of dynamic data sets on the final performance of an agent is insignificant, given that all data is, eventually, available to the agent. On the Atari 2600 domain, we showed that as the size of the ER gets bigger, the student's performance gets better, and that qualitatively, the pattern of the student's learning curve stays the same, diverging at some later point, after a peak reward is reached. The divergence cause is unknown, and we speculate that it's the result of overfitting, after the exhaustion of the immutable ER information. We propose that 600K sized ER is sufficient to achieve effective learning and that a decent generalization can be achieved by RL methods, when learning from expert's experience. Finally, our results apply that the use of varying ER, generated from multiple teacher networks, which stands for multiple policies, is superior to using ER produced by only one policy. That is to say that some amount of exploration is vital to achieve efficient learning. However, our results are not conclusive, what calls for further investigation.

References

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. *ICML*.

Jiang, L.; Meng, D.; Zhao, Q.; Shan, S.; and Hauptmann, A. G. 2015. Self-paced curriculum learning. *AAAI*.

Karpathy, A., and van de Panne, M. 2012. Curriculum learning for motor skills. *Advances in Artificial Intelligence. Springer* 325–330.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 1097–1105.

Kumar, M. P.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. *NIPS*.

Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. Technical report, DTIC Document.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2015. Policy distillation.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.