Averaging rewards as a first approach towards Interpolated Experience Replay

Wenzel Pilar von Pilchau[1]

Abstract: Reinforcement learning and especially deep reinforcement learning are research areas which are getting more and more attention. The mathematical method of interpolation is used to get information of data points in an area where only neighboring samples are known and thus seems like a good expansion for the experience replay which is a major component of a variety of deep reinforcement learning methods. Interpolated experiences stored in the experience replay could speed up learning in the early phase and reduce the overall amount of exploration needed. A first approach of averaging rewards in a setting with unstable transition function and very low exploration is implemented and shows promising results that encourage further investigation.

Keywords: Experience Replay; Deep Q-Network; Deep Reinforcement Learning; Interpolation; Machine Learning; Organic Computing.

1 Motivation

Reinforcement Learning (RL) [SB18] is a subdomain of Machine Learning (ML) [Mi97] and has got broader attention through recent breakthroughs such as the creation of an artificial Go player that won against the world's top human player 5:0 in a live match [Si16]. Go has been the biggest challenge for ML for many years, because of its enormous search space and the difficulty of evaluating board positions and moves. The next big breakthrough took place in StarCraft 2, a real-time strategy PC game with high speed, hundreds of units which have to be controlled and at the same time the need for managing economic aspects like income or army creation. It was chosen as the next big challenge and in 2019 DeepMind presented AlphaStar[2], the first trained StarCraft 2 program to beat a professional player. Although this happened in a limited and fixed scenario and one aspect of winning was super-human micromanagement, this represents a milestone in the progress of ML and in particular of ML in StarCraft 2. These achievements have been enabled through deep RL of which one important part is the Experience Replay (ER).

A RL agent has to adapt it's policy and believe about the world it is in with every step it takes, in an autonomous, self organizing way. With the combination of neural networks and

[1] Universität Augsburg, Organic Computing Group, Eichleitnerstr. 30, 86159 Augsburg Germany, wenzel.pilar-von-pilchau@informatik.uni-augsburg.de, ORCID: 0000-0001-9307-855X

[2] `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/` [Accesses 09 April 2019]

doi:10.18420/inf2019_ws53

standard RL techniques like Q-learning a powerful nature inspired algorithm has found it's way into this domain.

The ER can be seen as a memory of experienced transitions used to train the learn algorithm. To speed up learning in the earlier phase, it seems possible to interpolate transitions and fill the ER faster. The research goal in the long run therefore defines as follows: How could interpolated synthetic experiences in the ER speed up learning? It is also of great interest, in which environments which interpolation techniques yields the most speed up.

The rest of the paper is organized as follows. Sec. 1.1 arranges the intended research contribution in the context of Organic Computing, then, in Sec 2 a brief introduction of RL, deep RL, the ER and Interpolation is given. Sec. 3 introduces a first simple approach of how an agent could benefit from synthetic experiences in a non deterministic world. An evaluation shows first results, and in Sec. 4 some ideas of how to continue research are presented. Finally, Sec. 5 concludes the paper by summarizing it.

1.1 Organic Computing

Organic Computing (OC) [MT17, TSM17] describes the design of "life-like" technical systems with so-called *self-* properties*. An OC system therefore has the ability to act, based on it's own decisions. A related topic is Autonomic Computing (AC), which uses the biological principle of the autonomic nervous system as paradigm. Each OC system is equipped with sensors (observe the environment) and actuators (manipulate it). It adapts autonomously to the actual state of the environment it received from it's observer. The reaction has to be in a way that the system remains functional. To fulfill this requirement, even in, yet unseen states and unanticipated conditions, an OC system is typically based on (machine) learning.

As the research goal is about machine learning in general, and concrete about RL, this fits very well into the context of OC. Also, is the ER a nature inspired technique and in combination with deep RL and therefore neural networks, it lays in the scope of AC/OC as well.

2 Background

In this chapter a short introduction into the theoretical basics of relevant research areas, namely RL, deep RL and interpolation is given as well as a short overview of work conducted in this area.

2.1 Reinforcement Learning

This section is based on the seminal book of Sutton and Barto about RL [SB18].

In general, RL can be seen as learning what to do. In a more precise way, this means learning how to map situations to actions. A RL learner, also called agent, gets a numerical reward signal after executing an action and his task is to maximize this signal. The agent has no initial knowledge about which action it should take, but instead must discover which action returns the highest reward by trying them. This trial-and-error search is called exploring and forms, together with the delayed reward, the most important characteristics of RL. The latter explains the fact that an action not only affects the immediate reward, but also has an impact on the next situation and consequential all successional rewards.

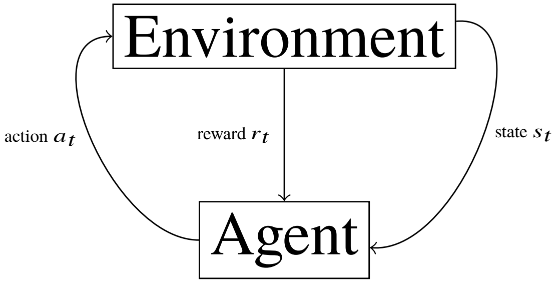

Fig. 1: The Reinforcement Learning Loop

In Fig. 1 a basic RL scenario is shown. It consists of an agent which interacts with an environment. At a time step t the agent gets the information of its actual state s_t from the environment. On basis of this information the agent can choose an action a_t it will execute in this state. After it has decided, it sends the information to the environment which calculates the reward r_t and the next state s_{t+1} the agent ends up in. Both are sent to the agent and it can now perform a learning step with the transition 4-tuple (s_t, a_t, r_t, s_{t+1}). This loop repeats until the agent has converged or a stop criterion is reached.

One challenge in RL, which was already mentioned above, is the explore-exploit dilemma. This describes the fact, that an agent, trying to maximize its reward, must prioritize actions it has already tried in the past and found them to be effective in generating reward. But to discover such actions, it has to try new - never chosen before - actions. The agent has to *exploit* the knowledge he has already gained to gather reward, but it also has to *explore* in order to make better decisions in the future. The *dilemma* is that neither exploration nor exploitation can be executed exclusively without failing at the task. The agent has to combine both in a meaningful way to succeed.

The main subelements - beyond the above discussed agent and environment - of a RL system are the following four: *a policy*, *a reward signal*, *a value function*, and, optionally, a *model* of the environment. We will take a deeper look at the first three ones.

The way a learning agent behaves at a given time is called its *policy*. More precise: the agent's policy decides which action it takes in which state. Roughly speaking one could imagine the policy as a mapping of states to actions. The policy is the core element of a RL agent as in itself it is is sufficient to determine behavior. A policy may be stochastic, specifying probabilities for each action. The *reward signal* has already been mentioned above and defines the goal of a RL problem. To determine which events are good and which are bad, the agent uses the reward received from the environment. The agent's objective over the long run is to maximize the total reward (return) he receives and therefore these signals are the main component for the agent to alter its policy. Contrary to this immediate measure of what is good, the *value function* specifies what is good on the long run. The value of a state is the return an agent can expect to accumulate over the future, starting in the particular state and following its policy. For example, a state with a low immediate reward might have a high value, because it is followed by other states which yield a high reward.

Many of the core algorithms of RL are inspired by biological learning systems, and therefore RL is - compared to other forms of ML - the closest to the kind of learning humans and other animals perform. This property of RL makes it a major component of Organic Computing.

2.2 Experience Replay

The Experience Replay [Li92, Li93] was introduced by Lin to increase sample efficiency and speed up convergence in RL. It is a biological inspired mechanism and can be understood as a collection of previous experiences the agent made. One experience is therefore defined as $e_t = (s_t, a_t, r_t, s_{t+1})$ and at each time step t the agent stores its recent experience in a data set $D_t = \{e_1, \ldots, e_t\}$. This procedure is repeated over many episodes, where the end of an episode is defined by a terminal state. In the training phase, the agent uses every transition, including a policy action, stored in the ER for training. A policy action is thereby defined as an action the agent would choose, in the same state, following the current policy, with a certain probability above a threshold P_l. This technique was used together with so-called "connectionistïmplementations of Q-Learning [SB18] and Adaptive Heuristic Critic [BSW89], which describes an implementation using non-deep neuronal networks. The use of an ER increases the sample efficiency and therefore the learning speed. It is very easy to implement in it's basic form and the cost of using it is mainly specified by the storage space needed to store it. [Li92]

2.3 Deep Reinforcement Learning

The following section is based on the publication of Mnih et al. [Mn13] and the dedicated journal article [Mn15].

Classical RL agents achieved some good results in a variety of domains, but are limited to their potential of representing the state. The domains of speaking are such in which useful features can be handcrafted, or which are fully observable and of low-dimension. Recent advances in deep learning, especially in computer vision, made it possible to extract high-level features from raw sensory data and produced better representations than handcrafted features. These breakthroughs led to a combination of RL and a class of artificial neural network, called deep neural network which operated directly on RGB images called deep Q-network (DQN) [Mn13]. This algorithm was able to successfully learn policies from high-dimensional sensory inputs using end-to-end reinforcement learning.

The DQN of Mnih et al. consists of a Convolutional Neural Network (CNN) [Le98], which uses hierarchical layers of tiled convolutional filters to extract relevant features of the input data. Then, after the convolution part, the network is used as an approximation of the optimal action-value function, which is also known as Q function [WD92]:

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

This function represents the maximum expected sum of rewards r_t discounted by γ at each time step t, achievable by a policy $\pi = P(a|s)$, after making an observation s and taking an action a.

A neural network is said to be a nonlinear function approximator and unfortunately RL on the other hand is known to be unstable or even diverge when the Q function is approximated by such [TVR97]. This is because of the following reasons:

1. the correlations present in the sequence of observations
2. the fact that small updates to Q may significantly change the policy and therefore change the data distribution
3. the correlations between the action-values and the target values

To address these issues an ER is used. In contrast to the basic ER from Lin described above, where all experiences with policy actions were used, Q-learning updates are applied on samples (or minibatches) of experience $(s,a,r,s') \sim U(D)$ which are drawn uniformly at random from the ER . This randomization breaks the correlations of the observations and succeeds in reducing variance of the updates. Another advantage is the fact that each step of experience is potentially used in more than one update, and therefore enables a greater data efficiency. Finally, in an on-policy [SB18] learning scenario, current parameters determine the next data sample that the parameters are trained on. This can result in the occurrence of unwanted feedback loops and the parameters getting stuck in a poor local minimum, or even diverging catastrophically. ER prevents that through averaging the behaviour distribution over many of its previous states and smoothing out learning.

A deep CNN is used to parameterize an approximated value function $Q(s,a;\theta)$ where θ stands for the parameters which are the weights of the network. The performed Q-learning update uses the following loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\Big[\big(r + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta)^2\big)\Big] \tag{1}$$

This corresponds to the squared error of the Q value minus the target value where the target value is expressed as $r + \gamma \max_{a'} Q(s', a'; \theta)$. As can be seen easily, the same parameters are used for the computation of the Q and also the target value. In every weight update the parameters change and thus increasing $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a which increases the target value. This could lead to oscillations or divergence of the policy. To overcome this issue a second network, called the target network, is introduced which is only used for calculating the target values. A copy of the Q network is frozen every C steps and generates the target network. This procedure adds a delay between the time an update to Q is made and the time the update affects the targets, making divergence or oscillations much more unlikely.

2.4 Interpolation

In the process of approximating a function f, one tries to find another function φ which minimizes the distance relative to a norm $|| \cdot ||$. As there exist several functions φ which approximates f unequally well, $\hat{\varphi}$ is denoted as the best approximation for which holds:

$$||f - \hat{\varphi}|| \leq ||f - \varphi|| \text{ for all } \varphi.$$

To find an approximation some knowledge about the function f is required. This knowledge is represented by a set of already seen data instances, more often denoted as sampling points $s_i := (x_i, f(x_i))$. The function f is either defined on a finite interval $s_i \in [a, b]$ or a finite set of points $s_i \in SP$. [Ga12]

If the best approximation $\hat{\varphi}$ in the discrete case holds the following condition: $||f - \hat{\varphi}|| = 0$, then it also satisfies $\hat{\varphi}(s_i) = f(s_i)$ for $i = 1, 2, \ldots N$. This special case is called *Interpolation* and describes the state in which the best approximation $\hat{\varphi}$ is equal to the function f in every sampling point. To map a value to a new and unknown point, called query point x_q, one can now approximate the interpolant between surrounding sampling points using low-degree polynomials. This is reasonable because a sufficiently small interval can be approximated arbitrarily well by low degree polynomials, even degree 1, or zero. Following this approach, not every sampling point has to be taken into consideration for calculating the query point's function value $\hat{\varphi}(x_q)$. It is nevertheless also possible to include all sampling points for calculation. This leads to a classification into two categories of interpolation: global methods which satisfy the latter and local methods which refer to the former. [Ga12, St17]

2.5 Related Work

The classical ER, introduced in Sect. 2.3, is a basic and not optimized technique, which has been improved in many further publications. One prominent improvement is the so called Prioritized Experience Replay [Sc15] which replaces the uniform sampling with a weighted sampling in favor of samples which might influence the learning process most. This modification of the distribution in the replay induces bias and to correct this importance-sampling has to be used. The authors show that a prioritized sampling leads to great success. Because of the fact that replays store a lot of experiences and the sampling occurs in every training step, it is crucial to reduce the computation cost to a minimum.

Another publication from de Bruin et al. [De15] investigates the composition of samples in the ER. They discovered that for some tasks it is important, that transitions, made in an early phase when exploration is high, are important to prevent overfitting. Therefore they split the ER in two parts, one with samples from the beginning and one with actual samples. They also show that the composition af the data in an ER is vital for the stability of the learning process and at all times diverse samples should be included. Furthermore in [de16] they investigate the impact of a uniform sample distribution achieved through synthetic samples and discover that the distribution obtained by the agent complemented with the synthetic experiences sampled uniformly performs best.

Jiang et al. investigated Experience Replays combined with model-based RL and implemented a tree structure to represent the transition and reward function [JHL19]. In their research they learned a model of the problem and invented a tree structure to represent it. Using this model they could simulate ”virtual experiences” which they used in the planning phase to support learning. To increase sample efficiency samples are stored in an ER. This approach has some similarities to interpolation but addresses other aspects.

An interpolation of on-policy and off-policy model-free deep reinforcement learning techniques present Gu et al. [Gu17]. In this publication an approach of interpolation between on- and off-policy gradient mixes likelihood ratio gradient with Q-Learning which provides unbiased but high-variance gradient estimations. This approach interpolates the gradient for the learning update and thus sets another focus than this paper.

Stein et al. use interpolation in combination with eXtended Classifier Systems to speed up learning in single-step problems [St17, St16b]. Their approach was applied to an Organic Traffic Control System [St16a]. In their work they introduce a so-called Interpolation Component which this publication uses as basis for it’s interpolation tasks.

3 Interpolated Experience Replay

This section outlines the idea of combining deep RL, especially the ER, with interpolation and then introduces a first simple approach.

3.1 Idea

As shown by de Bruin et al. [De15, de16] the composition in an ER is important for the learning stability. The stored samples are the ones which the agent has already seen before.

Depending on the problem, some states/transitions are rarely or very late discovered. At this point interpolation could help and create virtual experiences from states/transitions never seen before. These synthetic samples could be interpolated from surrounding, already visited sampling points.

A typical ER is of remarkable size and filled step by step. This early phase could be a point where interpolation could be of great assistance. Every time when a real sample is attached to the ER, several interpolated samples could be attached as well, leading to a faster filling and therefore a better distribution of the samples over the state space. This could speed up the early learning phase.

Another approach could be the existence of a small set of interpolated samples in the ER during the whole learning phase. With better knowledge of the problem - more sampling points, better policy and/or model - the interpolated samples get more accurate and yield more impact. Synthetic experiences are spread over the whole state space and thus could help to achieve a diverse enough distribution of samples in the ER. Depending on what the agent learns, it can create synthetic samples with smaller error over time. A trained state transition function for example would generate next states and with converging of this function the predictions would get better and synthetic samples using these would also benefit.

3.2 What to interpolate?

The question arises what exactly can be interpolated. An experience $e = (s, a, r, s')$ consists of the following components: state s, action a, reward r and the next state s'. The state should be drawn randomly from a defined state space S. The action which belongs to the action space A, could also be drawn randomly, despite the existence other possible choices, like using the action which promises the most impact on learning. The reward seems like a good choice to interpolate. As the model of a RL problem by definition persists of a reward function, it seems promising to interpolate this function. The last component is the next state, and this part comes with the most uncertainties. For the computation of the TD-error it is of some relevance that the next state s' in the equation:

$$r + \gamma V(s') - V(s) \tag{2}$$

is the actual next state of s. In Eq. 2 the V represents the value function, which could be replaced with the Q-function and then looks like Eq. 1. This raises the problem of how to get this unknown next state and is it maybe interpolable. One suggestion might be learning a state transition function like [JHL19], another approach could be ignoring this fact and therefore focusing on single-step problems like [St17].

3.3 A first simple approach

To start off the combination of RL and ER, a first, simplified approach was created.

3.3.1 Environment

As environment the so-called *FrozenLake-v0* from the OpenAI Gym [**?**] was chosen. This environment consists of a grid representing a frozen lake with some holes in it. Fig. 2 shows the 4x4 grid-world of the environment. The agent starts in the cell to the upper left with the big S in it and its goal is to reach the lower right F. The action set consists of 4 actions: walk up, walk left, walk right and walk down. Another constraint is the fact, that some piles represent holes in the ice, denoted with H, and if the agent walks upon one of them, he looses and has to restart. Additionally, because of the grid representing a frozen lake, the agent might slide and end up in another direction as he intended. By definition the agent gets a reward of 1 if he reaches the goal and in any other case a reward of 0. For this paper, a reward of -1, if the agent falls into a hole, was granted as well. This modification is crucial for averaging rewards described in the next chapter.

S			
	H		H
			H
H			F

Fig. 2: The FrozenLake-v0 environment

3.3.2 Averaging rewards

Because of the slippery ice, the transition function has to be taken into account for learning and this leads to the following assumption: If the agent starts in a state s and chooses an action a he might end up in four different next states s'_i with $i = 1, 2, 3, 4$, most of the time the agent will end up in the intended next state $s'_{intended}$ and over a lot of time the learn algorithm will realize that. If one or more states of s'_i are holes, the reward of -1 will threaten this action, but if this action leads towards the goal in most of the times, the action should be altogether positively rewarded (compared to actions which leads directly into the hole most of the time). An interpolated sample uses as its reward r the average reward of all sampling points with the same s and a, this synthetic sample represents the average risk of taking the action a in state s. To overcome the problem of simulating a next state, a synthetic sample for every distinct next state $s'_i \in SP$ which has been reached from state s in the past is created. This technique is more an averaging then a real interpolation, but it serves as a first approach.

3.3.3 Interpolation Component

For implementing the interpolation, the Interpolation Component from Stein et al. [St17, St16b] was used as a basis. It consists of a Machine Learning Interface MLI, an Interpolant, an Adjustment Component, an Evaluation Component and the Sampling Points SP as shown in Fig. 3. If the MLI decides it wants to alter its collection of sampling points, the appropriate sample $s*$ is handed to the Adjustment Component, there, following a decision function A, it is added to or removed from SP. If an interpolation is required, the Interpolation Component fetches required sampling points from SP and computes, depending on an interpolation technique I, an output o_{int}. The Evaluation Component provides a metric E to track a so-called trust-level T_{IC}.

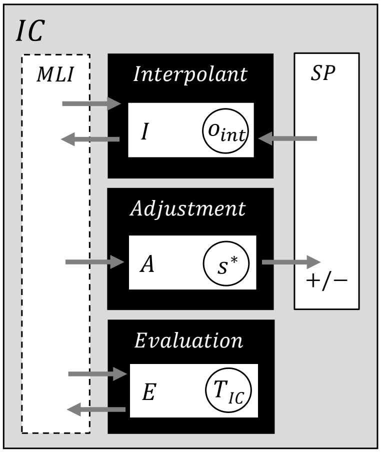

Fig. 3: The Interpolation Component from Stein et al. [St17]

In the implementation of this work, the ER replaces SP and consists of a queue with a maximum length and FiFo insert policy. This queue represents the standard ER and is filled only with real experiences, to store the synthetic samples another queue, a so-called *ShrinkingMemory*, is introduced. This second storage is of decreasing size, starting at a maximum it gets smaller depending on the length of the real valued queue. The *Interpolated Experience Replay* (IER) has a total size of the added length of both queues as can be seen in Fig. 4 and also a defined maximum size. If this size is reached, the length of the ShrinkingMemory is decreased and the oldest items are removed, until either the real valued queue gets to its maximum length and there is some space left for interpolated experiences or the IER fills up with real experiences.

In the Interpolant the averaging is executed at every insertion of a real valued experience. In every averaging process a query point x_q is randomly drawn from the state space and then all sampling points S_{match} whose first entry matches the queried state are collected. For every action $a \in A$ all sampling points that matches this action are selected from S_{match} and stored in S^a_{match}. Next the average reward of all samples in S^a_{match} is computed. At

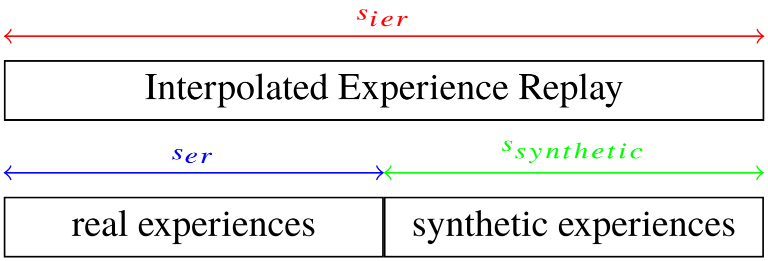

Fig. 4: The Interpolated Experience Replay

last a synthetic sample with this averaged reward is created for every distinct next state in S^a_{match}. This approach is very expensive in computation and acts as a first attempt which has to be improved. A tree-structured storage space for example could decrease the search time for S_{match}, if the leaf nodes hold samples with similar states. Another improvement for discrete states and actions would be the calculation of the averaged rewards for every state-action pair from the beginning on and then only updating the appropriate values with every new sample added to the replay.

3.3.4 Evaluation

An experimental setup, consisting of a neural network with 1 hidden layer of 10 nodes, 16 input nodes, one for each state of the grid and four output nodes, one for every action and parameters shown in Tab. 1 was trained for 1300 episodes. The experiment was executed with two different agents and 50 repetitions each. One agent used a standard ER and the other used the IER with similar sizes of s_{er} and $s_{syntehtic}$ and a total size $s_{ier} = s_{er}$.

Parameter	Value
Learning rate α	0.01
Discount factor γ	0.95
Stop exploration in step	2000
Epsilon start	1
Epsilon min	0.01
Update target net interval τ	150
Size of Experience Replay s_{er}	10,000
Size of Interpolated Experience Replay $s_{syntehtic}$	10,000
Start Learning at size of ER/IER	300
Minibatch size	32

Tab. 1: Parameters of the FrozenLake-v0 experiment

In Fig. 5 the result of the FrozenLake-v0 experiment is shown. Over the 50 repetitions, the reward (although the agent receives a reward of -1, if he falls into a hole, for visualization this case refers to a reward of 0) of every data point - every episode - was averaged and then a moving average with a window size of 100 was computed. At last all 50 data points were aggregated through calculating the average of it. The same procedure was executed with the epsilon values, this was necessary because the exploration decreased until step 2000, this results in about the 250th episode. It can be seen, that the green line, representing the agent with the IER, is reaching faster and even slightly overall a higher reward than the other agent. Both curves show a very high standard deviation which results from the agents learning badly in some runs. This can be explained with the short exploration phase which in some cases is not enough for the agent to converge. The use of synthetic experiences helps to compensate this and assists the agent in its learning. In real world applications when exploring the environment is expensive (i. e. robots) this approach can be useful.

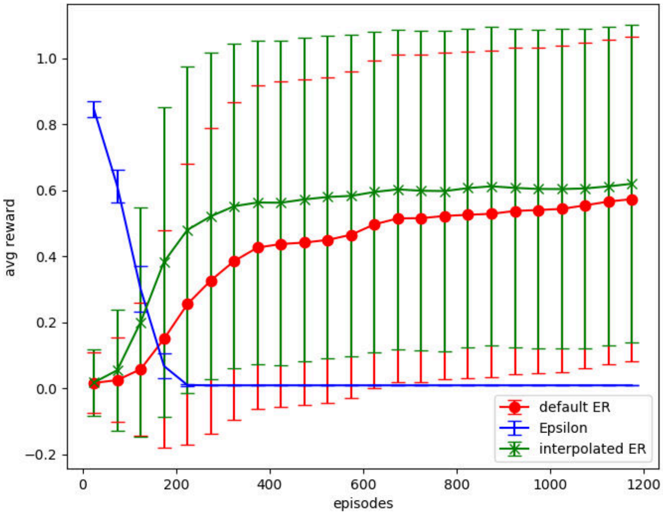

Fig. 5: Experiment 1

4 Future work

Further research in this area is required. One starting point would be the averaging in the FrozenLake environment. Instead of one input node for every state, only two nodes, one for the x- and one for the y-coordinate, could be enough and in this case the neural network could make greater use of its property to generalize. Furthermore, to test this in a larger environment the FrozenLake8x8-v0 (8x8 grid) could be used. Another approach would be the application of a network without a hidden layer which corresponds to a standard Q-table.

Also the basic structure of the ER represents a bottleneck with a search time of $O(n)$ for calculating the averages. A more efficient structure is needed here.

The use of real interpolation in the MountainCar environment with a linear reward function seems promising, too. Here several problems must be solved, like the choice/interpolation of the corresponding next state for the experience and which (more complex) interpolation methods to employ.

5 Conclusion

This publication demonstrated the methods of RL, deep RL and interpolation and how they could be combined in a meaningful way. A first simple approach of averaging rewards in an environment with an unstable transition function and short exploration phase shows that the use of synthetic experiences can assist learning in the early phase.

References

[BSW89] Barto, Andrew G.; Sutton, R. S.; Watkins, C. J. C. H.: Learning and Sequential Decision Making. In: LEARNING AND COMPUTATIONAL NEUROSCIENCE. MIT Press, pp. 539–602, 1989.

[De15] De Bruin, Tim; Kober, Jens; Tuyls, Karl; Babuška, Robert: The importance of experience replay database composition in deep reinforcement learning. In: Deep reinforcement learning workshop, NIPS. 2015.

[de16] de Bruin, T.; Kober, J.; Tuyls, K.; Babuška, R.: Improved deep reinforcement learning for robotics through distribution-based experience retention. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3947–3952, Oct 2016.

[Ga12] Gautschi, Walter: Approximation and Interpolation. In: Numerical Analysis, pp. 55–158. Birkhäuser Boston, Boston, 2012.

[Gu17] Gu, Shixiang Shane; Lillicrap, Timothy; Turner, Richard E; Ghahramani, Zoubin; Schölkopf, Bernhard; Levine, Sergey: Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3846–3855, 2017.

[JHL19] Jiang, W.; Hwang, K.; Lin, J.: An Experience Replay Method based on Tree Structure for Reinforcement Learning. IEEE Transactions on Emerging Topics in Computing, pp. 1–1, 2019.

[Le98] LeCun, Yann; Bottou, Léon; Bengio, Yoshua; Haffner, Patrick et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[Li92] Lin, Long-Ji: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 8(3):293–321, May 1992.

[Li93] Lin, Long-Ji: Reinforcement learning for robots using neural networks. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1993.

[Mi97] Mitchell, Tom M et al.: Machine learning. 1997. Burr Ridge, IL: McGraw Hill, 45(37):870–877, 1997.

[Mn13] Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; Riedmiller, Martin A.: Playing Atari with Deep Reinforcement Learning. CoRR, abs/1312.5602, 2013.

[Mn15] Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Rusu, Andrei A; Veness, Joel; Bellemare, Marc G; Graves, Alex; Riedmiller, Martin; Fidjeland, Andreas K; Ostrovski, Georg et al.: Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.

[MT17] Müller-Schloer, Christian; Tomforde, Sven: Organic Computing - Technical Systems for Survival in the Real World. Birkhäuser, 2017.

[SB18] Sutton, Richard S; Barto, Andrew G: Reinforcement learning: An introduction. MIT press, 2018.

[Sc15] Schaul, Tom; Quan, John; Antonoglou, Ioannis; Silver, David: Prioritized Experience Replay. CoRR, abs/1511.05952, 2015.

[Si16] Silver, David; Huang, Aja; Maddison, Chris J.; Guez, Arthur; Sifre, Laurent; van den Driessche, George; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda; Lanctot, Marc; Dieleman, Sander; Grewe, Dominik; Nham, John; Kalchbrenner, Nal; Sutskever, Ilya; Lillicrap, Timothy; Leach, Madeleine; Kavukcuoglu, Koray; Graepel, Thore; Hassabis, Demis: Mastering the game of Go with deep neural networks and tree search. Nature, 529:484, January 2016.

[St16a] Stein, A.; Tomforde, S.; Rauh, D.; Hähner, J.: Dealing with Unforeseen Situations in the Context of Self-Adaptive Urban Traffic Control: How to Bridge the Gap? In: Proc. of International Conference on Autonomic Computing (ICAC 2016). Würzburg, Germany, July 2016. accepted for oral presentation, to appear.

[St16b] Stein, Anthony; Rauh, Dominik; Tomforde, Sven; Hähner, Jörg: Architecture of Computing Systems – ARCS 2016: 29th International Conference, Nuremberg, Germany, April 4-7, 2016, Proceedings. Springer International Publishing, Cham, chapter Augmenting the Algorithmic Structure of XCS by Means of Interpolation, pp. 348–360, 2016.

[St17] Stein, Anthony; Rauh, Dominik; Tomforde, Sven; Hähner, Jörg: Interpolation in the eXtended Classifier System: An architectural perspective. Journal of Systems Architecture, 75:79–94, 2017.

[TSM17] Tomforde, Sven; Sick, Bernhard; Müller-Schloer, Christian: Organic Computing in the Spotlight. CoRR, abs/1701.08125, 2017.

[TVR97] Tsitsiklis, John N; Van Roy, Benjamin: Analysis of temporal-diffference learning with function approximation. In: Advances in neural information processing systems. pp. 1075–1081, 1997.

[WD92] Watkins, Christopher J. C. H.; Dayan, Peter: Q-learning. Machine Learning, 8(3):279–292, May 1992.