# Heterogeneous Team Deep Q-Learning in Low-Dimensional Multi-Agent Environments

Mateusz Kurek, Wojciech Jaśkowski
Institute of Computing Science, Poznan University of Technology, Poznań, Poland
wjaskowski@cs.put.poznan.pl

***Abstract*—Deep Q-Learning is an effective reinforcement learning method, which has recently obtained human-level performance for a set of Atari 2600 games. Remarkably, the system was trained on the high-dimensional raw visual data. Is Deep Q-Learning equally valid for problems involving a low-dimensional state space? To answer this question, we evaluate the components of Deep Q-Learning (deep architecture, experience replay, target network freezing, and meta-state) on a Keepaway soccer problem, where the state is described only by $13$ variables. The results indicate that although experience replay indeed improves the agent performance, target network freezing and meta-state slow down the learning process. Moreover, the deep architecture does not help for this task since a rather shallow network with just two hidden layers worked the best. By selecting the best settings, and employing heterogeneous team learning, we were able to outperform all previous methods applied to Keepaway soccer using a fraction of the runner-up's computational expense. These results extend our understanding of the Deep Q-Learning effectiveness for low-dimensional reinforcement learning tasks.**

**Keywords:**

## I. Introduction

Deep learning [24] is currently a hot topic in machine learning. Deep neural networks, which are neural networks with many hidden layers [1], have been successfully applied to supervised learning problems involving high-dimensional data in the fields of image processing [13], speech recognition [4], or natural language processing [27].

Up until recently, however, deep neural networks were considered to exhibit unstable dynamics in reinforcement learning settings. Things have changed with the seminal DeepMind's paper [17], which demonstrated how to make deep reinforcement learning obtain human-level performance on a large set of Atari 2600 games. Their method, called Deep Q-Learning involved Q-learning, deep convolutional neural networks, and a set of additional techniques: experience replay with minibatches, target network freezing, a modified version of the RMSProp backpropagation algorithm, and merging subsequent states into a meta-state. The input to the learning system involved high-dimensional visual pixel data.

In this work, we ask the question whether the Deep Q-Learning techniques used for high-dimensional inputs are equally effective for reinforcement learning problems with low-dimensional states. To this aim, we evaluate the components of DeepMind's Deep Q-Learning on a challenging Keepaway soccer task.

Keepaway soccer [31] is a task within the RoboCup Soccer simulator, which goal is to control a team of players to maintain the possession of the ball away from the opposing team. It presents many challenges to machine learning methods which include the continuous state space, uncertainty, multiple agents, and delayed effects of players' actions. The mechanics of the game are physically simulated, which makes the problem complex enough so it cannot be solved trivially. On the other hand, the simulation time is limited thus complete machine learning approaches are computationally feasible. This is why, it has been a popular benchmark for reinforcement learning methods, neuroevolution and, genetic programming.

The contributions of this work include an experimental analysis of the components of the DeepMind's Deep Q-Learning techniques on the Keepaway soccer task. We demonstrate that although experience replay with minibatches indeed improves the agent performance, target network freezing and meta-state slow down the learning process. Our results indicate that deep architectures do not help the agents and shallow networks with only two hidden layers are enough. We hypothesize that this is due to the low-dimensionality of the input data in Keepaway soccer.

In addition, as the Keepaway soccer is a multi-agent problem, we treat it as an opportunity to compare homogeneous and heterogeneous learning. The latter resulted in outperforming all previous methods applied to this task using only a fraction of the computational expense of the runner-up, Symbiotic Bid-based (SBB) GP [10]. The results indicate a considerable potential of (at least some of) the components of Deep Q-learning also when applied to low-dimensional problems.

## II. Related Work

Reinforcement learning [34] has been applied in the past to fields such as robotics [12], operations research [20], or economics [18]. One of the first application of reinforcement learning is Tesauro's TD-Gammon [38], a computer program trained to play Backgammon at super-human level using temporal different learning and self-play. Other successful applications of reinforcement learning to games include Go [25], Othello [36], Chess [26], and 2048 [35].

The Keepaway soccer problem was first proposed by Stone et al. in 2001 [30]. Since then, it has been approached by a number learning methods [6], [31], [32], [41], [29], [9], [22], [5], [2], [10], [11]. Below, we mention only the selected ones.

Stone et al. [29] used SARSA($\lambda$) with different function approximators: linear tile-coding (CMAC), radial basis functions

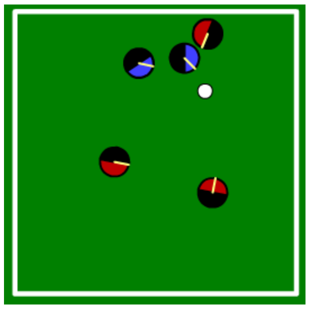

Figure 1. 3 vs. 2 Keepaway soccer

(RBF) and neural networks. Several work [28], [37] treated the problem with NeuroEvolution of Augmenting Topologies (NEAT) or HyperNEAT [40].

The best results for Keepaway soccer to date have been obtained by symbiotic bid-based (SBB) genetic programming [14] by Kelly et al [10]. A control policy is defined by a team of programs that coevolve, each specializing on a subcomponent of the task.

## III. Keepaway Soccer

Keepaway soccer is a problem within the RoboCup Soccer simulator [3], in which the task for one team (the keepers) is to maintain possession of the ball in a rectangular field, preventing another team (the takers) from taking it over [31]. Keepaway soccer is a challenging problem involving a continuous, partially observable state space, multiple agents, and noisy actions.

The most popular configuration in Keepaway soccer, which we focus on in this paper, involve $n = 3$ keepers and $m = n - 1 = 2$ takers playing in $20\,\mathrm{m} \times 20\,\mathrm{m}$ region (see Fig. 1) but other settings have also been considered in the past.

At the beginning of the game, all the players are placed in predefined locations of the playing field. In each turn, they move or kick the ball. The game ends when the ball leaves the playing field or any of the takers take it over.

Each agent acts based on the information about relative distances and angles to the other objects in the world. The state consist of i) distances from the players to the center of the playing region; ii) distances from the keeper possessing the ball to its teammates; iii) distances of each keeper to its closest opponent; and iv) angles between the keeper possessing the ball, some other keeper and an opponent. For the 3 $vs.$ 2 version, the state is described by 13 variables.

It is worth to note that the simulator introduces noise to the state variables.

In the simplified, standardized keepaway player framework, we consider here, only the keeper that currently possesses the ball can be controlled. The rest of the keepers follow a predefined behavior — they either wait for the ball or go towards the ball if none of its teammates is in the possession of the ball. The keeper possessing the ball can make one of $n$ possible actions: it may hold the ball or pass it to one of its teammates. The movement of the keepers is controlled by an arbitrary algorithm.

The framework comes with two fixed policies for the players: *Random* (choose the action randomly) and *Hand-coded* (a.k.a. *All-to-ball,* always go towards the ball). In the standardized Keepaway soccer, the takers behave according to the *Hand-coded* policy.

The Keepaway soccer task is a reinforcement learning problem, in which rewards are provided for each simulator second of ball possession.

## IV. Methods

### A. Reinforcement Learning

Reinforcement learning [34] is a class of Markov Decision Process (MDP). An MDP is a discrete-time, stochastic control process, defined as a quintuple $(S, A, P, R, \gamma)$, in which:

- $S$ is a finite set of states ($s_t \in S$, where $s_t$ is state observed at (discrete) time $t$),
- $A(s_t)$ is a finite set of possible actions in state $s_t$ ($a_t \in A(s_t)$, $A(s) \subseteq A$, where $a_t$ is action executed at time $t$),
- $P(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability, that executing action $a$ in state $s$ at time $t$ will lead to state $s'$ at time $t+1$,
- $R(s, a)$ is function describing the immediate (or expected immediate) reward for taking action $a$ in state $s$,
- $\gamma \in (0, 1]$ is the discount factor, which denotes the difference in priorities between the immediate reward and future rewards.

The solution to the MDP is a policy $\pi : S \to A$. The policy should maximize the expected discounted cumulative reward:

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}],$$

where $r_t$ is a reward obtained in step $t$.

The reinforcement learning problem is an MDP in which the model of the environment ($R$ or $P$) is unknown.

### B. Q-Learning

$Q$-learning is a model-free, off-policy, temporal difference [33] algorithm to solve reinforcement learning problems. It learns the action-value function $Q$ defined as:

$$Q^{\pi}(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a, a_t = \pi(s_t)].$$

The optimal $Q = Q^{\pi^*}$ value fulfills the Bellman equation:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \max_{a' \in A(s')} Q(s', a') \quad (1)$$

for all $s \in S$. The optimal policy is greedy w.r.t to $Q$:

$$\pi^*(s) = argmax_{a \in A(s)} Q(s, a)$$

Due to the search space size, the $Q$ function has often to be approximated, which is done by choosing a family of functions $Q_\theta$, where $\theta$ is a vector of parameters to learn.

To learn $\theta$, the agent gathers training examples by interacting with the environment according to its behavioral policy. One training example is a quadruple $(s, a, r, s')$. The agent iteratively updates $\theta$ to minimize the error on the one-step prediction:

$$L_\theta(s,a) = \frac{1}{2}\left(Q_\theta(s,a) - Q_{target}\right)^2,$$

where

$$Q_{target}(s,a) = R(s,a) + \gamma \max_{a' \in A(s')} Q_\theta(s', a').$$

$Q$-learning minimizes the loss function by following the gradient:

$$\theta \leftarrow \theta - \eta\left(Q_\theta(s,a) - Q_{target}(s,a)\right)\frac{\partial Q_\theta(s,a)}{\partial \theta}.$$

### *C. $\epsilon$-greedy Policy*

During collecting the training examples, the agent follows a behavioral policy. As making actions greedily according to the current policy may lead to a local optima, a common choice is the $\epsilon$-greedy policy, which consists in making a greedy action w.r.t. to the current state-action function with probability 1-$\epsilon$ and a random one with probability $\epsilon$, where $\epsilon \in (0,1)$ is the exploration rate.

$\epsilon$ can be fixed during the learning but annealing it often leads to better results. Here, we decrease it linearly from $0.1$ towards $e_f \in (0, 0.1)$, which is reached after the first $K$ learning episodes.

### *D. Deep Q-Networks*

The action-value function can be approximated by a linear weighted function of hand-designed features. This has been a common choice since the Q-learning with a linear function approximator has converge guarantees [34]. Neural networks, which are nonlinear function approximators, are known to be notoriously unstable (or even to diverge) when coupled with Q-learning [39]. Their unstable behavior is especially evident when many hidden layers were used (deep architectures).

Recently, however, in a breakthrough paper, Mnih et al. [17] proposed a set of techniques, called Deep Q-Learning in the following, to stabilize the Q-learning with (deep) neural networks. The techniques include experience replay with minibatches, target network freezing, and Root Mean Squared Gradient (RMSProp) algorithm for backpropagation. We describe the techniques in the following sections.

*1) Experience Replay and Minibatches:* E*xperience replay* is originally due to [15]. In this idea, the data used for learning is randomly sampled at each step from a memory of agent's previous transitions. In result, it removes the correlations in the sequence of training examples and smooths the training distribution over many past behaviors reducing oscillations of the learning process. Another advantage of this mechanism is the reuse of a single experience in many weights updates, which allows for greater data efficiency [17].

To perform the experience replay, agent's transitions at each time step $t$ are stored in replay memory $D$ as a tuple $e_t = (s_t, a_t, r_t, s_{t+1})$, where $s_t$ is state observed at time $t$, $a_t$ is action performed in state $s_t$, $r_t$ is the obtained reward, and $s_{t+1}$ is the resulting state after taking action $a_t$. The replay memory $D = \{e_1, e_2, ..., e_n\}$ stores the last $N$ such tuples.

```
1:  function DEEPQLEARNING
2:      $D \leftarrow \emptyset$                                ▷ Replay memory
3:      $Q \leftarrow$ RANDOMWEIGHTS()                   ▷ initialize $Q$
4:      while not stop condition satisfied do
5:          $s_0 \leftarrow$ INITIALSTATE()
6:          for time step $t$ in current episode do
7:              if RANDOMVALUE() $< \epsilon$ then
8:                  $a_t \leftarrow$ RANDOMACTION()
9:              else
10:                 $a_t = \arg\max_{a'} Q(s_t, a')$
11:             $r_t \leftarrow$ EXECUTEACTION($a_t$)
12:             $s_{t1} \leftarrow$ OBSERVESTATE()
13:             Store $(s_t, r_t, a_t, s_{t+1})$ in $D$
14:             for $(s_i, r_i, a_i, s_{i+1})$ in SAMPLE($M, D$) do
15:                 $y_j = \begin{cases} r_i & \text{if } terminal(s_{i+1}) \\ r_i + \gamma \max_{a'} Q_\theta(s_{i+1}, a') & \text{otherwise} \end{cases}$
16:                 $L \leftarrow \sum_{j=1}^{M}(y_j - Q_\theta(s_j, a_j))^2$       ▷ Backprop
```

Figure 2. Deep Q-Learning with experience replay and minibatches.

Another technique, used together with experience replay, is *minibatch learning*, which consists in learning more than one training example at each step. It makes the learning process less prone to outliers and noises as the gradient computed at each step uses more training examples. At each step of the training, dataset $D$ is sampled uniformly to get a minibatch of experiences of size $M$ ($(s,a,r,s') \sim U(D)$). The Deep Q-Learning algorithm with experience replay and minibatches is presented in Fig. 2.

*2) Target Q-Network Freezing:* Target Q-Network freezing [17] consists in maintaining two separate networks: i) a target network, called $Q_{\theta^-}$ in the following, with a fixed set of 'old' parameters $\theta^-$ for generating target Q-values, used in the Q-learning process, and ii) a network $Q_\theta$ for interacting with the environment, with the current set of parameters $\theta$.

Target Q-values are calculated using the old set of parameters $\theta^-$:

$$Q_{target}(s,a) = R(s,a) + \gamma max_{a'} Q_{\theta^-}(s', a')$$

At every update iteration, the current parameters $\theta$ are updated to minimize the mean-squared Bellman error w.r.t the old parameters $\theta^-$ by optimizing the following loss function:

$$L_\theta = \mathbb{E}_{s,a,r,s',i \sim D}[(R(s,a) + \gamma max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s,a))^2]$$

Every $C$ steps, the parameters from the Q-network are copied to the target Q-network. Generating the learning targets using an old set of weights adds a delay between the time an update to Q is made and the time the update affects the targets, counteracting oscillations and divergence.

*3) Root Mean Squared Gradient (RMSProp):* The most common learning method is stochastic gradient descent (SGD). SGD assumes that the learning rate is the same for each parameter being learned, which works poorly when the gradient values vary since for some components, it leads to large changes and tiny changes for the others. RProp [21] solves this problem by i) using only the sign of the gradient and ii) adapting the step size separately for each weight. Unfortunately, RProp does not work well with minibatches [7].

RMSProp [7] is an extension to RProp and SGD, which combines the robustness of RProp, efficiency of minibatches, and effective averaging of gradients over minibatches (unlike RProp). RMSProp keeps track of the previous gradients and divide the updates by the average magnitude of the gradient over the last several updates, which leads, in practice, to maintaining separate learning rate $\eta_i$ for each weight. This allows modifying each parameter $\theta_i$ according to its previous magnitudes, preventing from taking it into account with too large or too small weight. RMSProp is] parametrized by gradient moving average decay factor $\rho$ (0.9 by default), overall learning rate $\eta$ and $\varepsilon$ ($10^{-6}$ by default) to maintain the numerical stability. The learning rate $\eta_i$ at time step $t$ is calculated as follows:

$$g_i^{(t)} = \rho g_i^{(t-1)} + (1-\rho)(\frac{\partial L}{\partial \theta_i}^{(t)})^2$$
$$\eta_i^{(t)} = \frac{\eta}{\sqrt{g_i^{(t)} + \varepsilon}}$$

Mnih et al. [17] introduced a slightly different version of the RMSProp algorithm, which, besides of using the squared gradients from the past, keeps track of the regular gradients as well:

$$h_i^{(t)} = \rho h_i^{(t-1)} + (1-\rho)\frac{\partial L}{\partial \theta_i}^{(t)}$$
$$\eta_i^{(t)} = \frac{\eta}{\sqrt{g_i^{(t)} - (h_i^{(t)})^2 + \varepsilon}}$$

## V. Experiments in Deep Q-Network

In this section, we evaluate the components of the DQN framework in the context of the Keepaway Soccer task.

### A. Experimental Setup

In all experiments, we use the classical Keepaway setup with 3 $vs.$ 2 players in a $20\,\mathrm{m} \times 20\,\mathrm{m}$ region, with the *hand-coded* policy for takers. Learning time has been limited to 20 000 episodes. The controller *score* denotes the time of a single episode in simulator seconds. By controller *performance* we mean the expected score, which we estimate by averaging the duration of a number of episodes. To monitor the learning progress, every 500 learning episodes, we play 100 testing episodes. The performance of the final controller is evaluated on 1000 testing episodes.

The l*earning time* is the time of the learning process measured in real-time units (minutes). The experiments were conducted with Intel® Core™ i7-950 3.7GHz. The Keepaway simulator is quite sophisticated thus it is the computational bottleneck of the experiment (the time required for evaluating the network or updating its weights is negligible).

Each experimental run was repeated 6 times unless stated otherwise.

| Parameter | Tested values |
|---|---|
| Discount factor $\gamma$ | 0.99, **1.0** |
| Learning rate $\eta$ | 0.001, 0.0005, **0.0001**, 0.00005, 0.00001 |
| Exploration time $K$ | 5000, 10000, **15000**, 20000 |
| Final exploration $e_f$ | **0.01**, 0.1, 0.05 |

Table I
THE PARAMETER VALUES TESTED IN THE PRELIMINARY EXPERIMENTS. BEST FOUND VALUES WERE MARKED BOLD.

| Memory size | Minibatch size | Mean score [s] | Learning time [min] |
|---|---|---|---|
| 1 | 1 | $8.3 \pm 1.5$ | $192 \pm 17$ |
| 10000 | 1 | $6.2 \pm 0.7$ | $154 \pm 12$ |
| 10000 | 4 | $14.2 \pm 3.3$ | $293 \pm 6$ |
| 10000 | 8 | $14.3 \pm 3.0$ | $344 \pm 14$ |
| 10000 | 16 | $15.4 \pm 1.5$ | $405 \pm 10$ |
| 10000 | 32 | $17.8 \pm 1.8$ | $500 \pm 15$ |
| 10000 | 64 | $18.1 \pm 1.3$ | $750 \pm 31$ |
| 10000 | 128 | $17.7 \pm 1.0$ | $620 \pm 21$ |

Table II
INFLUENCE OF EXPERIENCE REPLAY AND MINIBATCH SIZE ON AGENT'S FINAL PERFORMANCE

### B. Preliminary Parameter Search

In preliminary trial-and-error experiments, we evaluated several numerical parameters of the learning framework using a 13-30-100-50-3 (three full-connected hidden layers) network. The network's weights were uniformly initialized by the He's method [8]. The tested and best-found parameter values are shown in Table I. These values are used in subsequent experiments.

In addition, we also have verified that increasing the number of learning episodes over 20 000 does not improve the score. Also, although decreasing the learning rate $\eta$ over time is a common choice in Q-learning, we found out that this does not improve the results for this task.

### C. Experience Replay and Minibatches

In the first experiment, we investigated whether learning with experience replay and minibatches is more effective than on-line learning (with no experience replay and minibatch size equal to 1), how the size of the minibatch affects the overall performance, and whether the claims made in Section IV-D1 that using experience replay with minibatches increase the stability of the learning process are correct.

The results of the experiment are shown in Table II and Fig. 3. The baseline algorithm did not use experience memory

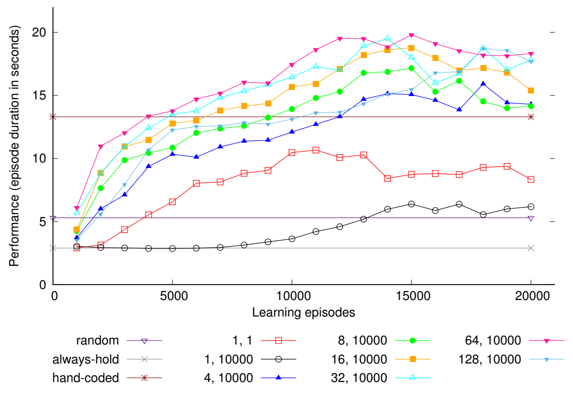

Figure 3. Influence of the experience replay and minibatches on the average episode duration.

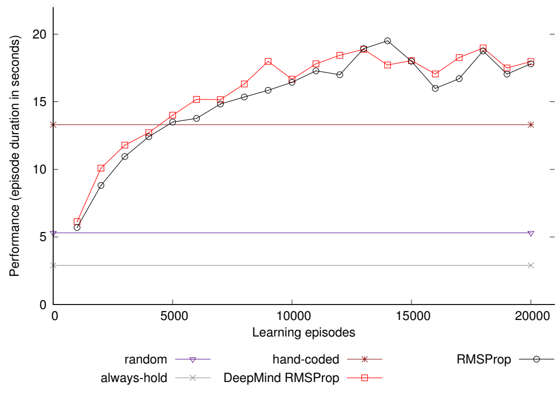

Figure 4. Comparison of backpropagation algorithms.

| Target network update frequency | Mean score [s] |
|---|---|
| no freezing (baseline) | $17.9 \pm 2.1$ |
| 100 | $14.5 \pm 2.6$ |
| 1000 | $16.9 \pm 3.7$ |
| 10000 | $16.1 \pm 1.4$ |

Table III
TARGET NETWORK FREEZING.

(and thus experience replay). We can see that experience replay alone makes the learning process significantly slower. However, coupled with learning from minibatches, the results improve considerably. Already for minibatches of size 4, we observe a 70% performance improvement over the baseline. The larger the minibatch, the higher is the controller score, however, increasing the minibatch size from 64 to 128 does not help further. The difference in the learning time are mostly due to the episode durations of the experiments.

Importantly, except the minibatch of size 1, increasing the size of the minibatch decreases the standard deviation of the performance. It indicates that the minibatches stabilize the learning process.

Since the difference between the results with 32 and 64 sizes of the minibatch is minor while, at the same time, the experiments with minibatch size of 64 take approximately 1.5 times longer to run, in the subsequent experiments minibatch of size 32 with memory of size 10000 will be used (the minibatch of size 64 will be used only to learn champion policy).

### D. Backpropagation Algorithms

The Deep Q-Learning used by the DeepMind team used its version of the Hinton's RMSProp. Does this backpropagation algorithm also pay off in our settings? Fig. 4 shows the results of the experiment conducted to answer this question. The experiment was performed using experience replay with minibatch of size 32 and the replay memory of size 10000. The results are not conclusive. The DeepMind's version seems slightly better and it is also characterized by a slightly smaller variance, but we found no statistically significant difference between the two versions of the RMSProp.

We will use the DeepMind's version of RMSProp in further experiments.

### E. Target Network Freezing

In this experiment, different target network update frequencies were compared: every 100, 1000, and 10000 steps. Earlier research has shown (see Section IV-D2) that freezing weights of the target $Q$-network leads to higher stability of the learning process and lower divergence between the consecutive results returned by the neural network. The experiments were performed with minibatch of size 32, the replay memory of size 10000 and used DeepMind's RMSProp.

The results of the experiment are shown in Table III and Fig. 5. Contrary to the expectations, the results indicate that target network freezing does not provide any improvement over the baseline version, for which only one neural network is maintained. The differences between different update schemes are statistically insignificant. Moreover, as shown in Fig. 5, the neural network with the update frequency of 10 000 steps is learning significantly slower than the other ones. That is why

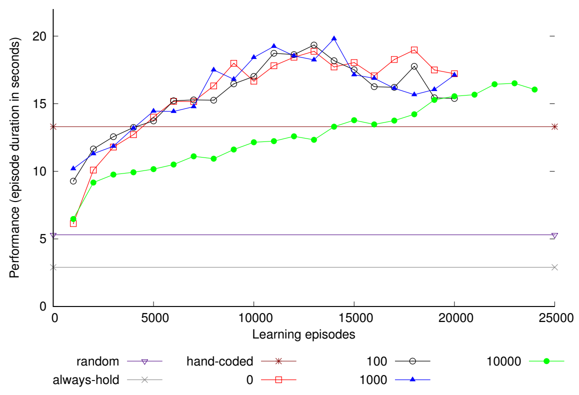

Figure 5. How target network update frequency influence on average episode duration.

| Neural network architecture | Mean score [s] |
|---|---|
| 13-400-200-100-50-25-3 | $16.2 \pm 5.0$ |
| 13-200-100-50-3 | $15.8 \pm 0.5$ |
| 13-200-100-50-3 | $17.1 \pm 1.0$ |
| 13-100-200-50-3 | $16.9 \pm 2.0$ |
| 13-30-100-30-3 | $17.9 \pm 2.1$ |
| 13-200-30-3 | $17.8 \pm 1.6$ |
| 13-200-50-3 | $18.1 \pm 2.2$ |

Table IV
NEURAL NETWORK ARCHITECTURES FOR KEEPAWAY SOCCER.

| Meta-state size | Mean score [s] |
|---|---|
| 1 (baseline) | $18.1 \pm 2.2$ |
| 2 | $12.0 \pm 2.2$ |
| 4 | $9.4 \pm 2.1$ |

Table V
META-STATE LEARNING

| Team learning | Mean score [s] | Median score [s] |
|---|---|---|
| homogeneous | $18.48 \pm 1.30$ | 18.23 |
| heterogeneous | $19.6 \pm 1.1$ | 19.49 |

Table VI
HOMOGENEOUS AND HETEROGENEOUS TEAM LEARNING (THE FINAL EVALUATION OF 25 RUNS).

this experiment was extended over 20 000 episodes.

### *F. Neural Network Architectures*

The goal of this experiment was to choose the best neural networks architecture for the Keepaway Soccer problem. During this experiment, six neural networks with two to five hidden layers with different numbers of nodes in the hidden layers were tested. All of these networks have the rectifier activation function in hidden layers and no activation function in the output layer. The input layer of the neural network always has 13 units (the size of Keepaway's state) while the output layer consists of 3 neurons, each corresponds to the number of possible actions available to the keeper that currently possesses the ball.

All experiments were performed with experience replay with minibatch of size 32, the replay memory of size 10000, DeepMind's RMSProp and no target network freezing.

The results of this experiment are shown in Table IV. The differences between the architectures of neural networks are relatively small. We also have found that in the initial phase of learning, deep neural networks have slightly better performance than the shallow ones, but the overall trend shows no advantage of using deep architectures. The best result was obtained by the smallest neural network 13-200-50-3 consisting of only two hidden layers.

No advantage of deep architectures caused by the low dimensionality of the input state. The results suggest that there is no need to model complex non-linear relationships in a problem with such simple input as Keepaway Soccer or the learning algorithms used are not able to effectively make use of it.

### *G. Meta-State Learning*

Since the Keepaway Soccer is partially observable, in the next experiment, we check whether combining multiple consecutive states into a single *meta-state* leads to better performance. A similar technique was effective for deep reinforcement learning in the video game playing domain [17].

To this aim, consecutive states are merged into a single vector of size $13n$, where $n$ is the number of recent states (meta-state size). This should, theoretically, provide the agent more information about the actual state of the environment. For example, the agent could reason whether the takers are moving towards him.

Meta-states of sizes 2 and 4 were tested. All trials were performed with minibatch of size 32, the memory of size 10000, DeepMind's RMSProp, no target network freezing and with the 13-200-50-3 neural network architecture.

The results of the experiment are shown in Table V. Unexpectedly, combining the consecutive states into a single meta-state, significantly decreases the agent's performance compared to the baseline. This might be caused by the agent's inability to either extract valuable information from the meta-state, or relate consecutive values of variables.

## VI. TEAM LEARNING

In the previous experiments, we learned a single policy for the three keepers (homogeneous team learning). They were using the same experience replay memory and were modifying weights of the single neural network. In the final experiment, we ask the question whether it pays off to learn a separate policy for each agent, which is often called *heterogeneous team learning* [19]. In such settings, each agent has its own experience replay memory and its own neural network. Theoretically, the results should be the same as when using homogeneous team, since the optimal policy is the same for each agent. Nevertheless, when function approximation is used, the theoretical optimal policy is, in general, not achievable, thus learning three neural networks might, in practice, improve the performance of the team. Note that the agents have no possibility of (direct) communication except observing the behavior of the other keepers.

For the experiment, we use the best settings found in the previous sections, that is, the minibatch of size 64, the replay memory of size 10000, DeepMind's RMSProp backpropagation, no target network freezing, no meta-state, and the 13-200-30-3 network architecture. To more precisely evaluate the learning method, this time, 25 individual learning runs were performed.

The results of the experiment are shown in Fig. 6, and Table VI. As expected, the final results of both approaches are

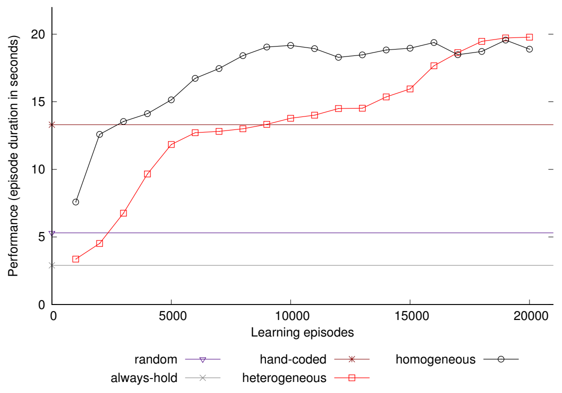

Figure 6. Dynamics of the team learning.

| Method | Mean score [s] | Learning hours |
|---|---|---|
| Always Hold [29] | $2.9 \pm 1.0$ | - |
| Random [29] | $5.3 \pm 1.8$ | - |
| SARSA with Neural Network [29] | $10.1 \pm 0.3$ | 30 |
| Hand-coded [29] | $13.3 \pm 8.3$ | - |
| NEAT [37] | $14.1 \pm 1.8$ | 800 |
| SARSA with RBF [29] | $14.2 \pm 3.1$ | 30 |
| EANT [16] | $14.9 \pm 1.3$ | 200 |
| HyperNEAT [40] | $15.4 \pm 1.3$ | 50-200 |
| SARSA with CMAC [29] | $15.7 \pm 2.8$ | 30 |
| SBB with diversity [10] | $18.5 \pm 1.9$ | 1739 |
| (heterogeneous) Deep Q-Learning | $19.6 \pm 1.1$ | 56.5 |

Table VII
COMPARISON OF THE APPROACHES TO KEEPAWAY SOCCER. THE "$\pm$" SIGN PRECEDES THE STANDARD DEVIATION.

close to each other, but statistically, the heterogeneous team learning is performing significantly better than homogeneous team learning (t-test, $\alpha = 0.05$). What is worth noticing is the learning speed (see Fig. 6). The heterogenous learning is significantly slower than in the homogeneous one. This is because heterogenous teams need to learn 3 times more parameters.

## VII. Comparison with Other Approaches

The comparison of all the previous results in the Keepaway soccer domain together with our Deep Q-Learning approach is shown in Table VII. It indicates that our method outperforms all of the previously published results in terms of the average keepers' possession time (the mean score). Notice that our method is also characterized by relatively a low variance.

Comparing Deep Q-Learning to the runner up, the best published to date method (SBB with diversity [10]), we observe that, although the presented approach is only slightly better in terms of the mean score, importantly, it needed 30 times less learning time (the simulator's hours) to achieve this score. What is more, SBB is conceptually more sophisticated than the deep reinforcement learning and, unlike the deep Q-Learning, it requires providing some domain knowledge, i.e., designing the genetic programming operators.

## VIII. Summary and Conclusions

In this paper, we evaluated the components of Deep Q-Learning on a challenging multi-agent task of the Keepaway soccer, which involves, in contrast to the original general video game playing application of Deep Q-Learning, low-dimensional states.

The results of the experiments showed that some of the deep learning techniques, indeed, increase the performance of the agent, while the others have negative effect. In particular, experience replay and minibatch learning significantly improve the results. RMSProp makes Q-learning for this problem possible since SGD did not converge at all. However, the DeepMind's RMSProp was found not better than the classical one. Target network freezing did not improve the results. On the contrary, when the target network update is rare, the learning process slows down considerably. Also, composing several subsequent states into a single meta-state (to alleviate the effect of partial observability) does not pay of for Keepaway. Finally, we found out that shallow, two-hidden-layer neural networks are enough for this task. Deep architectures do not improve the results. We speculate that it is due to the low-dimensionality of the Keepaway's state space.

In addition, for the Keepaway soccer it is profitable to use heterogeneous team learning. Despite making the learning initially slower than the homogeneous one, it lead to better results, eventually.

The results of the experiments demonstrate that Deep Q-Learning applied for Keepaway soccer performs better than any other previously published method. Not only it is better in terms of the average score but it also uses significantly less computation effort to learn its policy. Compared to the best to date result, achieved by genetic programming (SBB) agent [10], the Deep Q-Learning agent outperforms it using only 1/30 computation time required by SBB.

The experiments confirmed that deep reinforcement learning is a suitable and effective method not only for highly-dimensional problems (that need to utilize convolutional neural networks) but also for low-dimensional problems such as Keepaway soccer.

There are many aspects of the proposed approach that are worth further investigation. First, experience replay is limited in some respects — the memory does not differentiate the important transitions from the unimportant ones. A more sophisticated sampling strategy such as prioritized experience replay might be considered [23]. Second, the presented agent is trained to play on the $20\,\mathrm{m} \times 20\,\mathrm{m}$ region and it is evaluated in the same environment. Further research can investigate how well does the learned agent generalize to other sizes of the playing field [10]. Finally, some of the plots shown in Section V contain traces of unstable or cycling dynamics around 12000-16000 learning episodes. This effect requires further research.

## Acknowledgment

W. Jaśkowski acknowledges the support from the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932.

## References

[1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:437–478, 2012.

[2] Luiz A. Celiberto, Jackson P. Matsuura, Ramón López De Màntaras, and Reinaldo A C Bianchi. Reinforcement learning with case-based heuristics for RoboCup soccer keepaway. *Proceedings - 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium, SBR-LARS 2012*, pages 7–13, 2012.

[3] Mao Chen, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, and Xiang Yin Yi Wang. The robocup soccer simulator. http://sourceforge.net/projects/sserver/, 1997-2015. [Online; accessed 12-September-2015].

[4] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Mike Seltzer, Geoffrey Zweig, Xiaodong He, Julia Williams, et al. Recent advances in deep learning for speech research at microsoft. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8604–8608. IEEE, 2013.

[5] Sam Devlin, Marek Grzes, and Daniel Kudenko. Multi-agent, reward shaping for robocup keepaway. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 1227–1228, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[6] Anthony Di Pietro, Lyndon While, and Luigi Barone. Learning in RoboCup Keepaway using Evolutionary Algorithms. 2002.

[7] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. http://www.cs.toronto.edu/t̃ijmen/csc321/slides/lecture_slides_lec6.pdf, 2012.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[9] Tobias Jung and Daniel Polani. Learning RoboCup-Keepaway with kernels. *Journal of Machine Learning Research - Proceedings Track*, 1:33–57, 2007.

[10] Stephen Kelly and Malcolm I. Heywood. Genotypic versus behavioural diversity for teams of programs under the 4-v-3 keepaway soccer task. pages 3110–3111, 2014.

[11] Stephen Kelly and Malcolm I Heywood. On Diversity , Teaming , and Hierarchical Policies : Observations from the Keepaway Soccer Task. *EuroGP*, pages 75–86, 2014.

[12] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[14] Peter Lichodzijewski and Malcolm I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 363–370, New York, NY, USA, 2008. ACM.

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

[16] Jan H Metzen, Mark Edgington, Yohannes Kassahun, and Frank Kirchner. Analysis of an evolutionary reinforcement learning method in a multiagent domain. *{AAMAS} '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, (Aamas):291–298, 2008.

[17] 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.

[18] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *Neural Networks, IEEE Transactions on*, 12(4):875–889, 2001.

[19] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[20] Scott Proper and Prasad Tadepalli. Scaling model-based average-reward reinforcement learning for product delivery. In *Machine Learning: ECML 2006*, pages 735–742. Springer, 2006.

[21] Martin Riedmiller. Rprop-description and implementation details technical report, january 1994.

[22] Toru Sawa and Toshihiko Watanabe. Learning of keepaway task for RoboCup soccer agent based on Fuzzy Q-Learning. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pages 250–256, 2011.

[23] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[25] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[26] Ioannis E. Skoulakis and Michail G. Lagoudakis. Efficient Reinforcement Learning in Adversarial Games. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, pages 704–711. IEEE, nov 2012.

[27] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank, 2013.

[28] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[29] Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway Soccer: From Machine Learning Testbed to Benchmark. *Lncs*, 4020:93–105, 2006.

[30] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward RoboCup soccer. *Icml*, (June):537–544, 2001.

[31] Peter Stone and Richard S Sutton. Keepaway Soccer: A Machine Learning Testbed. *Lecture Notes in Computer Science*, 2377:207–237, 2002.

[32] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, 13:165–188, 2005.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement Learning : An Introduction*. 2012.

[34] R.S. Sutton and A.G. Barto. *Reinforcement learning*, volume 9. MIT Press, 1998.

[35] Marcin Szubert and Wojciech Jaskowski. Temporal difference learning of n-tuple networks for the game 2048. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.

[36] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. On scalability, generalization, and hybridization of coevolutionary learning: a case study for othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):214–226, 2013.

[37] Matthew E Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. *Proceedings of the 8th annual conference on Genetic and evolutionary computation GECCO 06*, (July):1321, 2006.

[38] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.

[39] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Technical report, IEEE Transactions on Automatic Control, 1997.

[40] Phillip Verbancsics and Kenneth O Stanley. Evolving Static Representations for Task Transfer. *Journal of Machine Learning Research*, 11:1737–1769, 2010.

[41] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving keepaway soccer players through task decomposition. *Genetic and Evolutionary Computation Conference 2003, Proceedings of the*, 59(1):356–368, 2005.