

# Correlation minimizing replay memory in temporal-difference reinforcement learning

Mirza Ramcic<sup>a,\*</sup>, Andrea Bonarini<sup>b</sup>

<sup>a</sup>Artificial Intelligence Center, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic

<sup>b</sup>Artificial Intelligence and Robotics Lab, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy

## ARTICLE INFO

### Article history:

Received 14 June 2019

Revised 31 January 2020

Accepted 5 February 2020

Available online 6 February 2020

Communicated by Dr. Peter. W Vamplew

### Keywords:

Reinforcement learning

Temporal-difference learning

Replay memory

Artificial neural networks

## ABSTRACT

Online reinforcement learning agents are now able to process an increasing amount of data which makes their approximation and compression into value functions a more demanding task. To improve approximation, thus the learning process itself, it has been proposed to select randomly a mini-batch of the past experiences that are stored in the replay memory buffer to be replayed at each learning step. In this work, we present an algorithm that classifies and samples the experiences into separate contextual memory buffers using an unsupervised learning technique. This allows each new experience to be associated to a mini-batch of the past experiences that are not from the same contextual buffer as the current one, thus further reducing the correlation between experiences. Experimental results show that the correlation minimizing sampling improves over Q-learning algorithms with uniform sampling, and that a significant improvement can be observed when coupled with the sampling methods that prioritize on the experience temporal difference error.

© 2020 Elsevier B.V. All rights reserved.

## 1. Introduction

Temporal difference (TD) types of reinforcement learning (RL) such as Q-learning [1] are model-free learning algorithms that differ from a more basic approach as that of Monte Carlo methods [2] in the way they update their estimates about the states. While Monte Carlo methods update their estimates when the final outcome is known, TD learning algorithms *bootstrap*, or build upon the previous estimates, in order to obtain the current one, while sampling states from the environment. As the process of learning continues, the previous estimates become increasingly more reliable because the current estimates are continuously updated.

When dealing with high dimensional spaces it is impractical to keep a table of the value estimates for each agent's state, and update them as they occur. Instead, the value of the state is approximated using an artificial neural network (ANN). In this case, the learning process becomes a sequential update of the neural network weights  $\Theta$  at each learning step in order to obtain a better prediction about the value of that state.

The sequential nature of the TD updates induces correlation among the samples used to train a value approximating ANN, which in turn leads to instability and even divergence of the learn-

ing process [3]. In order to reduce the correlation in the sequence of experiences, some online reinforcement agents use the memory of past experiences, which can be randomly replayed after each learning step in order to make the learning process more stable [3–5]. These learning mechanisms benefit from a *replay memory* structure which is efficient in optimizing the *stochastic gradient descent* (SGD) algorithm used to train the artificial neural network approximator that represents the core of the RL mechanism. The approach presented here further improves the ability of the replay memory to reduce the correlation among the replayed experiences. Instead of using a single memory replay buffer and treating all experiences evenly, the proposed *Correlation Minimizing Memory*, or *COMM*, introduces a finite number of contextual replay memory buffers, each of which holds the experiences corresponding to a specific context. In order to achieve this, each experience of an agent is clustered into the same amount of dimensions or classes as the number of contextual memory buffers, based on their similarity. The more an experience belongs to a specific class the more are the chances of it being sampled into the contextual memory buffer corresponding to that class. The main principle of the *correlation minimizing* algorithm is found in the replay of clustered memories; each experience is supported by replaying the past experiences from memory replay memory buffers that are not belonging to its cluster instead of the random, out of context replay used, e.g., in [3–5]. Experimental results show that *COMM* obtains an improvement in learning performance over

\* Corresponding author.

E-mail addresses: [ramicmir@fel.cvut.cz](mailto:ramicmir@fel.cvut.cz) (M. Ramcic), [andrea.bonarini@polimi.it](mailto:andrea.bonarini@polimi.it) (A. Bonarini).

the standard Q-learning with uniform sampling and shows a very significant improvement when combined with prioritized sampling based on temporal difference error [6].

## 2. Related work

The works by Kretchmar et al. [7], Mnih et al. [8], Ong et al. [9] and Clemente et al. [10] presented asynchronous parallel learning methods which were addressing the correlation problem in replay memory, which scales nicely in the work by Nair et al. [11] and Sutton et al. [12]. The approaches are based on the idea of *multitask learning* [13] and are able to asynchronously execute multiple instances of agents in multiple instances of the environment and use the data from every instance to perform the learning itself. Since the agents were exploring different parts of the environments in parallel, the correlation was removed from the experience stream. These approaches, summarized well in the work of Bellemare et al. [14] have a reputation of being computationally demanding because of its parallelism of agents and environments. They are also limited in their applicability to real-life physical learning problems as opposed to simulations. In the approach presented by Shaul et al. [6] the replay memory mechanism was improved by prioritizing on the experiences with higher temporal-difference or TD error. These experiences are potentially more valuable for training the approximator, since they carry more surprise over the predictions than others.

An incremental improvement on the proposal of Schaul et al. [6] was presented by Ramicic and Bonarini [15], where an additional sampling criteria based on the Shannon's entropy or informational potential of the state space was combined with the TD one in order to induce a more efficient training of the approximator. Ramicic and Bonarini [16] also integrated dynamics of the replay memory sampling in order to improve the performance of the machine learning algorithm. This approach took advantage of a genetic algorithm to evolve an experience filter ANN in charge of deciding whether a single experience will be sampled into replay memory. Clustering techniques similar to the ones in *COMM* have been used to optimize the learning methods of other kind, such as genetic algorithms in the work by Jin et al. [17]. A more general approach of reducing correlation in ANN when using multiple dataset views or modalities is a concept of *correlation neural networks* introduced by Chandar et al. [18] that also uses an *autoencoder* technique to form the different views or modalities. This approach has been also extended to deep neural network architectures in the work by Benton et al. [19] and Cogswell et al. [20].

## 3. Theoretical background

The aim of the learning process is to create a policy  $\pi$  which maps the current state of an agent to a preferred action in order to maximize its reward potential in the long run [2]. A RL agent interacts with its immediate environment in discrete time steps that are defined as transitions of a *Markov Decision Process* or MDP and represented by a tuple  $(s_t, a_t, r_t, s_{t+1})$ . At each time step the agent updates its policy  $\pi$  making it closer to the *optimal policy*  $\pi^*$ , which is represented by an optimal action-value function  $Q^*(s, a)$  shown in Eq. (1) and defined as the maximum expected return while following the policy  $\pi$ .

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \quad (1)$$

Bellman's Eq. (2) provides an expectation of the Q value which is defined as the immediate reward received plus the discounted value of the expected next state.

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right] \quad (2)$$

The expectation value is crucial to the learning process as it can tell how wrong the previous estimate of the Q value was. This difference between the previous estimation and the expectation is called *temporal difference* or TD error and it is indicative of how much the current estimate of the value of the state-action pair is wrong. Knowing this error, it is possible to perform the main learning loop by updating the prediction for the state-action pair so to make it closer to the expectation.

### 3.1. Approximation

When faced with high dimensional state spaces it is highly impractical to keep the estimates of the Q value for each combination of state and action, even more so when we are dealing with continuous state representations. In this case, the best option is to approximate  $Q^*(s, a)$  using a function approximator such as an ANN. A function approximation makes it possible to predict a Q value for each of the possible actions available to the agent by providing the agent's current state as input of the ANN. After each time step, the expected Q value can be computed using Eq. (2) and compared to the estimate that the function approximator provides at its output  $Q(s, a; \Theta) \approx Q^*(s, a)$  by forwarding the state  $s_0$  to its input. The difference between the previous estimate of the approximator and the expectation is the TD error, and it can be backpropagated through the ANN in order to update the current approximation of  $Q^*(s, a)$ . In order to train the ANN to approximate a newly observed transition an expectation of Q value is obtained by Bellman equation Eq. (2) which becomes the target  $y$ . In order to make the current prediction  $\hat{y}$  closer to the target  $y$ , a parameter update on  $\Theta$  is performed in order to minimize the loss function  $L(\Theta)$  shown in Eq. (3).

$$L(\Theta) = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2 \quad (3)$$

The actual backpropagation is performed using *Stochastic Gradient Descent* (SGD) on the loss  $L_i(\Theta_i)$  according to Eq. (4):

$$\nabla_{\Theta_i} L_i(\Theta_i) = (y_i - Q(s, a; \Theta_i)) \nabla_{\Theta_i} Q(s, a; \Theta_i), \quad (4)$$

where  $y_i = r + \gamma \max_{a'} Q(s', a'; \Theta_{i-1})$  is in fact our Bellman equation defining the target value.

Eq. (4) formally represents the full update using SGD for approximating  $Q(s, a, \Theta) = Q^*(s, a)$ . A SGD is a method for solving a minimization problem by updating the parameter vector  $\Theta$  consisting of weights  $w$ . In order to find out how will an update on a specific weight parameter  $w_i$  influence the change in the loss function  $L$ , it is possible to consider its derivative with respect to that parameter as reported in Eq. (5).

$$\Delta L \approx \frac{\partial L}{\partial w_i} \Delta w_i \quad (5)$$

Since the aim is to minimize the loss function  $L$ , the weight  $w_i$  can be modified in the direction that is opposite of the derivative by a small number  $\alpha$  called *learning rate*.

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \quad (6)$$

## 4. Model architecture and learning algorithm

The main part of the learning mechanism represents the *learning loop* outlined in section (b) of Fig. 1. Here, the agent iteratively performs an interaction within its environment and uses the newly obtained experience to update its belief about the best action to take depending on the perceived state. During the transition stage, an agent performs an action  $a_t$  that transitions the agent from the starting state  $s_t$  to the next  $s_{t+1}$  while receiving the immediate



state given by the experience. The  $M$  dimensions or classes are effectively encoded in the  $M$  neuron layers and, in order to find the class of a single experience, the state  $s_t$  is forwarded through the network, and a vector of the  $M$  activation values of the bottleneck layer  $V_C = (C_1, C_2, \dots, C_M)$  is obtained. Since the vector  $V_C$  contains the clusters which the current experience belongs to, we define the probability of the experience being sampled into  $i$ th *contextual replay memory* or  $CRM_i$  as the activation value of the  $i$ th element of the vector  $V_C$  as defined in Eq. (7).

$$P(\text{store}(CRM_i)) = C_i \quad (7)$$

The more the experience belongs to a specific class the higher the probability it will be sampled into the  $CRM$  of that specific class. In order to create a memory replay that would support the minimization of correlation, we sample the required mini-batch according to the different probabilities. Depending on the applications we introduce two modes of sampling that differ in their complexity. *COMM-A* represents a more complex method that reduces the correlation in a twofold way: between current experience and the experiences stored in replay memory, and, additionally, among the replay memory experiences themselves. This method is suited for the Q-learning applications that use a simpler  $Q$  function approximator and thus rely on smaller batches during the training phase. The second method, *COMM-B*, allows the proposed approach to scale up to more demanding problems such as Atari games [3,5], which are characterized by much larger state spaces approximated using a more complex class of deep neural network architectures based on convolutional layers. Because of the increased dimensionality, this type of approximator requires much more training data than the simpler approaches and can additionally benefit from a *pre-training* procedure that is performed on randomly generated state space samples computed before the RL process as shown in Algorithm 2. In order to address this issue the agent's experiences in [3,5] are collected in a replay buffer of larger capacity over millions of learning steps and also sampled in larger batches. The sheer size of the replay buffer batch in complex Atari games problems makes learning from the current experience at each step relatively trivial. *COMM-B* simplifies the sampling in these type of problems by only reducing correlation between the experiences in the replay memory regardless of the current one.

#### 4.1. COMM-A

Finding out which contextual replay memory contains experiences that are not in the context of the current memory is easy as creating an inverted vector  $V_I = (I_1, I_2, \dots, I_M)$  by clamping each value of the  $V_C$  between 0 and 1 and subtracting 1 from it as shown in Eq. (8)

$$I_i = \text{clamp}[0, 1](C_i) - 1 \quad (8)$$

Now the *COMM-A* is able to replay a mini-batch from the contextual replay memories that are least correlated with the current memory using inverted probabilities in vector  $V_I$  as defined in Eq. (9)

$$P(\text{replay}(CRM_i)) = I_i \quad (9)$$

Algorithm 1 showcases the details about the process of *context inhibition* in a way that each experience is supported by the experiences that are the least correlated to replaying experiences from the contextual buffers that are determined by the inverted classes vector  $V_I$ .

#### 4.2. COMM-B

The goal of the simpler version of the sampling, *COMM-B*, is only to minimize the correlation between experiences in memory.

---

#### Algorithm 1 Q-learning with COMM-A.

---

```

Initialize M instances of replay memory D of capacity N
Initialize autoencoder neural network A with M neurons in the
middle layer
Initialize action-value function Q with random weights
for episode = 1, E do
  for t = 1, T do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t = \arg \max_a Q^*(s_t, a; \Theta)$ 
    Execute action  $a_t$ , observe reward  $r_t$  and state  $s_{t+1}$ 
    Train A by backpropagating  $s_t$  with same  $s_t$  on the input
    Forward  $s_t$  through A and obtain a M-dimensional vector
    of activation values from neurons on the middle layer  $(C_1, C_2, \dots, C_M)$ 
    for  $C_n = 1, M$  do
       $I_n = \text{abs}(\text{clamp}[0, 1](C_n) - 1)$ 
    end for
    for i = 1, M do
      Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $i$ th replay memory  $D_i$ 
      according to the probability of the  $i$ th activation  $P(C_i)$ 
    end for
    Sample random batch of transitions  $(s_t, a_t, r_t, s_{t+1})$  from  $i$ th
    replay memory  $D_i$  according to the probability of the  $i$ th
    activation  $P(I_i)$ 

    set  $y_i = \begin{cases} r_i, & \text{terminal } s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \Theta), & \text{nonterminal} \end{cases}$ 

    Perform a gradient descent step on  $(y_i - Q(s_i, a_i; \Theta))^2$  ac-
    cording to Eq. (4)
  end for
end for

```

---

Because the experiences in different memory replay buffers belong to different uncorrelated classes, in this variation the experiences are sampled uniformly from each of the  $M$  buffers as reported in Algorithm 2.

### 5. Experimental setup

The evaluation was performed in a variety of environments with different characteristics which also implemented different types of function approximation. The first set of experiments applied *COMM-A* algorithm, and have used a simpler  $Q$  function approximator with linear network layers only. This set includes the *Waterworld* environment (Section 5.1) as implemented in a ReinforceJS framework [21], and the *Lunar Lander* environment Section 5.2 which is a part of OpenAI Gym [22]. The set of learning problems that require a more complex function approximator implemented using convolutional network layers include a selection of Atari games as reported in Section 5.3.

#### 5.1. Waterworld environment - COMM-A

Waterworld environment [21] as showcased in Fig. 2 consists of moving food pieces, instantiated at random positions with random speed and direction, and capable of bouncing on the walls framing the environment. The food pieces come in two dispositions: "good", which provides a positive reinforcement of +1 when matched by the agent, and "bad", which results in a negative reinforcement value of -1 upon contact with the agent. The environment contains an equal amount of good and bad food. Once a piece of food has been consumed, a new piece of food of the same type is generated in a random position, with random speed,

**Algorithm 2** Q-learning with COMM-B.

---

```

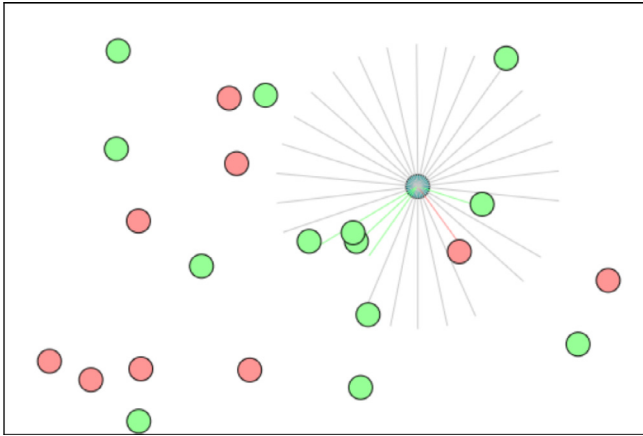
Initialize M instances of replay memory  $D$  of capacity  $N$ 
Initialize and pre-train autoencoder neural network  $A$  with  $M$ 
neurons in the middle layer
Initialize action-value function  $Q$  with random weights
for episode = 1,  $E$  do
  for  $t = 1, T$  do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t = \arg \max_a Q^*(s_t, a; \Theta)$ 
    Execute action  $a_t$ , observe reward  $r_t$  and state  $s_{t+1}$ 
    Train  $A$  by backpropagating  $s_t$  with same  $s_t$  on the input
    Forward  $s_t$  through  $A$  and obtain a  $M$ -dimensional vector
    of activation values from neurons on the middle layer ( $C_1, C_2, \dots, C_M$ )
    for  $M_n = 1, M$  do
      Sample random transition  $(s_t, a_t, r_t, s_{t+1})$  from  $M_n$ th replay
      memory buffer
    end for
    Sample random batch of transitions  $(s_t, a_t, r_t, s_{t+1})$  from  $i$ th
    replay memory  $D_i$  according to the probability of the  $i$ th
    activation  $P(I_i)$ 

    set  $y_i = \begin{cases} r_i, & \text{terminal } s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \Theta), & \text{nonterminal} \end{cases}$ 

    Perform a gradient descent step on  $(y_i - Q(s_i, a_i; \Theta))^2$  according to Eq. (4)
  end for
end for

```

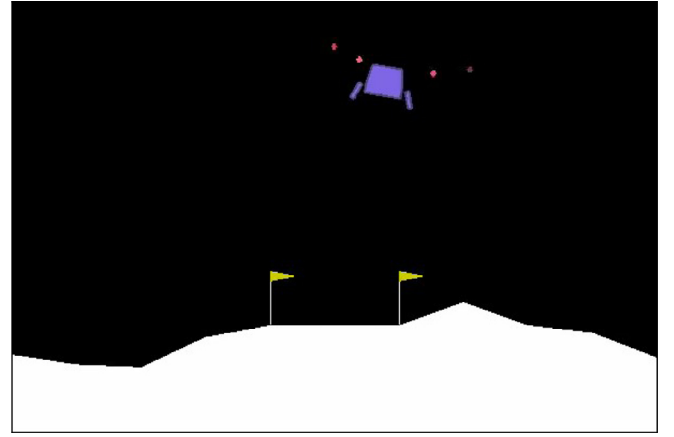
---



**Fig. 2.** A single rendered frame from the Water world environment depicting an agent together with its sensor array learning to consume good (red) and avoid bad (green) food sources.

in order to keep the distribution constant. The agent's goal is to maximize its expected reward in the long run by learning to consume good food pieces, while avoiding the bad ones. The agent can take at each step one out of five possible actions: left, right, up, down, and stay. Its perception of environment is implemented as 30 equally distributed directional sensors, each capable of perceiving five continuous variables: distances and velocities components in  $x$  and  $y$  to good food and bad food, as well as distances to the walls. These, along with the perception of two additional variables for agents own speed give a high dimensional state space of 152 continuous variables.

Approximation of  $Q(s, a; \Theta) \approx Q^*(s, a)$  is done using an ANN with one hidden fully connected layer of 100 neurons that are producing as output the  $Q$  values of all five actions available to the



**Fig. 3.** A single rendered frame from Lunar Lander environment depicting a craft firing its thrusters in order to land on the designated area marked by two yellow flags.

agent: up, down, left, right, stay, given the state space of 152 dimensions as input. The learning rate of an approximator  $\alpha$  is set to a low 0.05 and the capacity of the each of the 6 contextual replay memory buffers  $N$  was 1000, while the non-COMM configurations were implemented with a single buffer of 6000 experiences. The value of  $\epsilon$  was set to 0.2 at the beginning and adjusted to 0.1 at the mid-point of the set learning period to exploit more the learned behavior. The discount factor  $\gamma$  was set to 0.9.

## 5.2. Lunar lander environment - COMM-A

The second of the simpler environments, Lunar Lander, represents a more realistic example of rocket trajectory optimization, which is a common problem in the optimal control area.

It consists of a craft attempting to land on a designated platform marked by two flags and set on an uneven terrain under the simulated weak moon gravity as depicted in Fig. 3. The craft is equipped by one main thruster pointing downwards and two stabilization thrusters on each side. The craft should be able to manipulate the three thrusters in order to safely land and counterbalance the gravitational pull. This accounts for an agent with four discrete actions: do nothing, fire main thruster, fire left thruster, and fire right thruster. The craft senses the environment using 6 continuous and 2 boolean variables respectively indicating: the  $x$  position of the craft relative to the platform, its  $y$  position, the crafts angle,  $x$  and  $y$  components of velocity, angular velocity together with a variable for each landing leg valued 1 if the specific leg of the craft is in contact with the ground. This creates a 8-dimensional state space.

Reinforcement function provides a reward from 100 to 140 if the lander moves from top of the screen to the landing pad and achieves 0 speed at the landing. The function creates the reward by taking into the account the proximity of the craft to the landing platform, whether the legs are in contact with the ground and how many fuel the craft have consumed so far.

Lunar Lander is an episodic task and a single episode ends if the lander crashes or rests on the ground receiving additional reinforcement of -100 or +100.

The experiments were performed on 1200 episodes which proved to be more than enough for an agent to reach the score of +200 which indicates that the environment is considered solved by the framework. The exploration-exploitation parameter  $\epsilon$  was decayed from the starting value of 1.0 during the simulations by multiplying it by 0.998 after each episode reaching a low value of 0.06 at the end of the final episode and  $\gamma$  was set at 0.99. The  $Q$ -

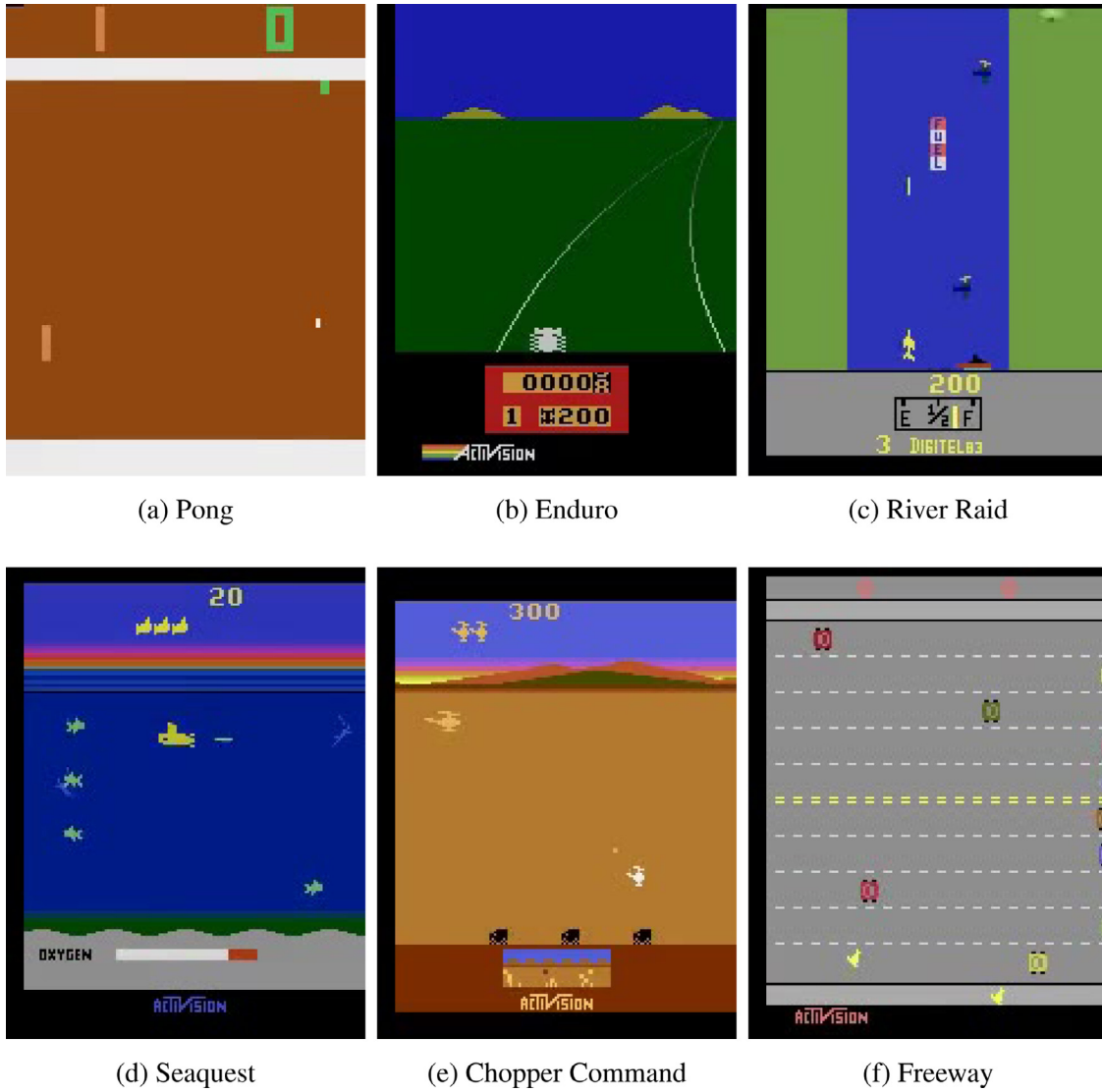


Fig. 4. Rendered frames from variety of tested Atari games also representing the agent's state space used in function approximation during learning.

value approximating ANN was initialized with two hidden layers which consisted of 128 and 64 nodes respectively with the learning rate parameter set at a low 0.0001.

Memory buffer was set to 12,000 for the non-COMM and 2000 x 6 for the COMM implementations making the two configurations identical in memory capacity.

### 5.3. Atari games - COMM-B

Atari 2600 games [3,5] represent a radically different challenge to RL as their state space consists of the rendered pixels of the actual game: in this case the machine learning algorithm is presented with an input that is created or “optimized” for a human player senses. Under the Gym framework [22] the Atari games state space is defined as a RGB image of the screen as shown in Fig. 4 and represented by an array of dimension (210,160,3), while the action space is game dependent.

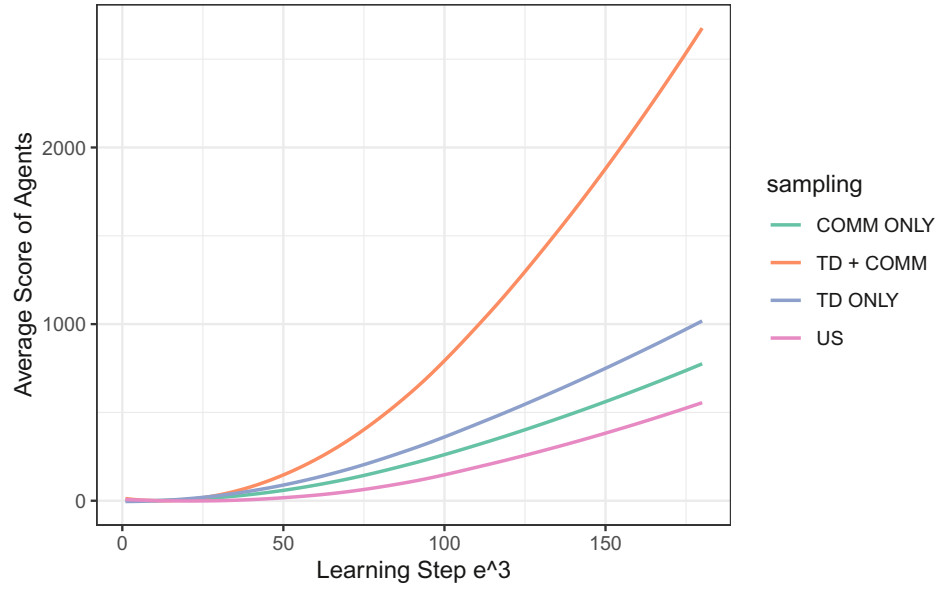
The learning problems of this type are usually solved within several million learning steps. For this reason the memory capacity of a single replay buffer in the baseline was set to a rather high 72,000 experiences while the COMM approaches used a capacity of  $8 \times 9000$ . The epsilon parameter  $\epsilon$  was decayed using a simple

exponential function from 0.7 to 0.05 thorough the training steps while  $\gamma$  was 0.9.

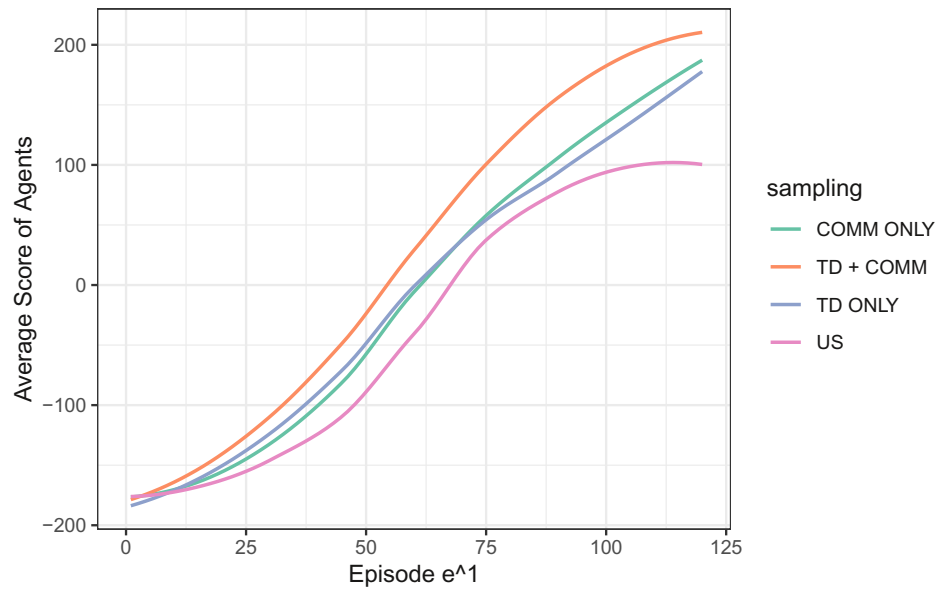
The main difference compared to the previous environments is that Atari games rely on a more complex convolutional neural network architecture (CNN, or ConvNet) in order to approximate the Q value function. The raw image input was pre-processed into a (1,84,84) array by cropping and converting the RGB channels into grayscale. The approximator itself was implemented with a convolutional input layer with kernel size of 8 followed by two hidden convolutional layers implementing gradually decreasing kernel sizes. The output from the convolutional layers was channeled through a hidden intermediate linear layer to an output layer, whose outputs were an approximation of Q values, one output for each of the actions available to the agent. Because of the larger availability of learning data and considering the CNN architecture the meta-parameter learning rate  $\alpha$  was set to the rather low value of 0.000015.

### 5.4. Unsupervised learning

In order to decide in which mini-batch the sampled experience has to be inserted, an autoencoder neural network was used. The autoencoder used for the lower complexity environments such as



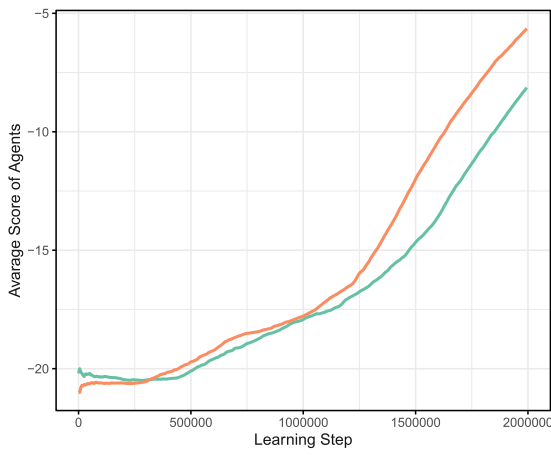
**Fig. 5.** Average total reinforcement received over 50 trials with agents with different sampling dynamics during first 180,000 learning steps in Waterworld environment. *COMM* represents the proposed method, *TD + COMM* represents a combination of *COMM* method and temporal difference prioritization, *TDONLY* is temporal difference prioritization without *COMM* sampling while *US* is the baseline uniform sampling.



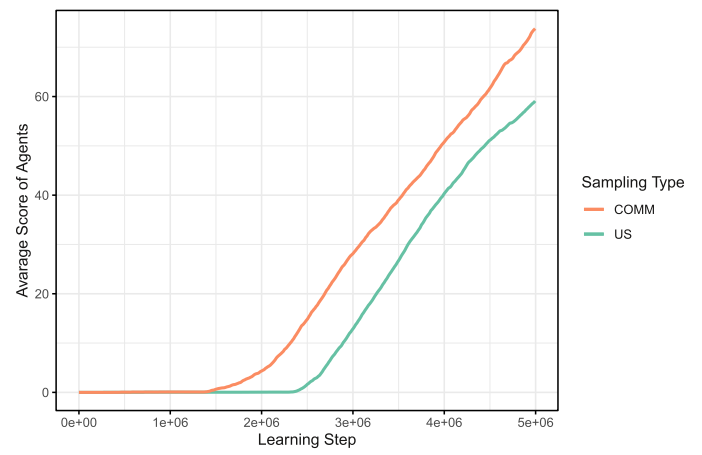
**Fig. 6.** Average total reinforcement received over 30 trials with agents with different sampling dynamics during first 1500 episodes in Lunar Lander environment.

was a simple linear neural network able to cluster up to 152 dimensional state into  $M = 6$  dimensional array using an architecture with three hidden layers. The input and output layers were implemented using 152 neurons capable of handling the high-dimensional state array on both sides. The hidden part consisted of a bottleneck middle layer containing  $M = 6$  neurons surrounded by two additional fully connected layers implemented with 100 neurons each, able to smooth out the transition from high to low dimensional space. The learning rate of the autoencoder was set to  $\alpha = 0.1$ , much higher than the Q-value approximator very low  $\alpha = 0.000015$ . The drastic change in the learning rate was implemented because of the architecture of the autoencoder itself: it contained a greater number of neurons which in turn made its training process significantly slower. For classification of the states of Atari games images under the *COMM-B* variation a more complex convolutional autoencoder was used to cluster the images such

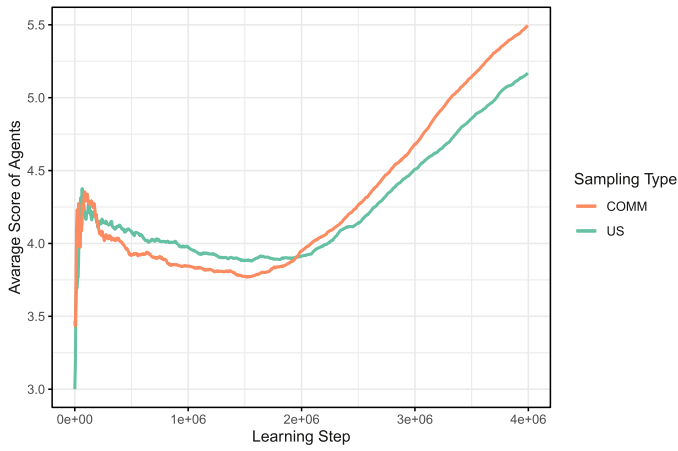
as the ones shown in Fig. 4. The reason for selecting a convolutional autoencoder was the same as for choosing a convolutional architecture for the main Q-value approximator network: the visual nature and increased dimensionality of the Atari game input source. Using regular, fully connected, dense layers to approximate a high-dimensional input matrix of (210,160,3) would be very impractical as it would produce an enormous amount of connections and weights, significantly increasing the training time. On the other hand, convolutional neural network architectures can have the same effect with significantly less inter-neuron connections than fully connected ones: this makes them as perfect candidates for high dimensional inputs. Furthermore, convolutions can take the advantage of the spatial relationship among data, which is particularly useful when detecting common feature shapes in visual data, as needed in applications such as Atari games. The bottleneck in the convolutional architecture case also featured a hidden linear



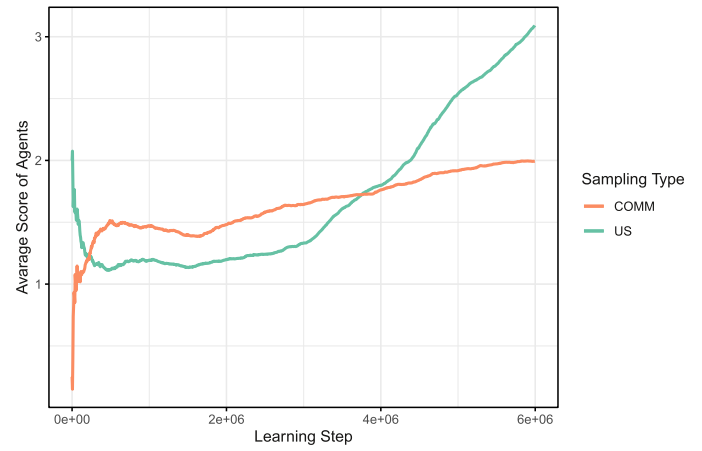
(a) Pong



(b) Enduro



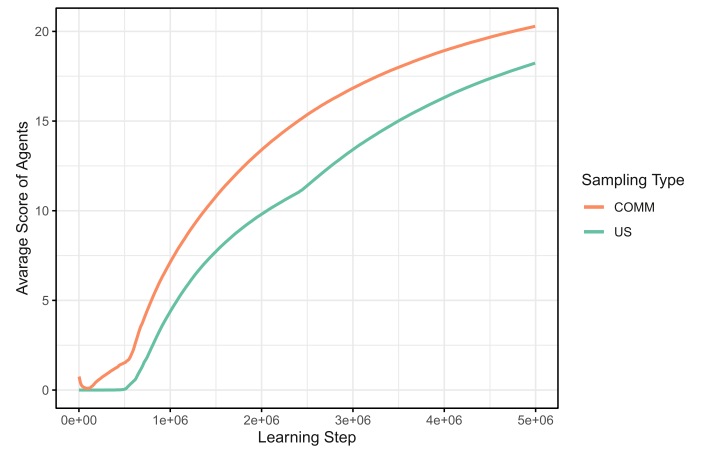
(c) River Raid



(d) Seaquest



(e) Chopper Command



(f) Freeway

**Fig. 7.** Average cumulative score received by agents in a total of 5 trials during millions of learning steps in various Atari games and sampling methods. COMM represents the proposed method, while US is the baseline uniform sampling.

layer, this time with  $M = 16$  nodes surrounded by a collection of convolutional layers on both the input and output side. Each of the output activation values of the bottleneck layer was paired with  $M$  CRM buffers.

## 6. Experimental results

Experiments compared the average performance of the agents using four different types of memory replay sampling in multiple environments with different dynamics that are solved using different function approximation techniques. The first of the more simpler environments, Waterworld, was evaluated in 50 learning trials consisting of 180 learning steps each, while the second one Lunar Lander being an episodic task was run for 1200 episodes in 30 trials. The more advanced environments of Atari games were evaluated in a range from 4 to 10 million learning steps depending on their complexity while taking an average of the agent's reward.

### 6.1. Evaluations of COMM-A

The results reported in Fig. 5 and 6 show how the novel approach of the proposed COMM – A performed in two different environments, compared with the baseline Q-learning with uniform memory sampling and replay US. Along the baseline further comparisons were done combining the baseline and the COMM configurations with the common method for replay improvement: prioritization based on temporal difference or TD error [6].

In the Waterworld environment we can see that the COMM-A sampling method shows slightly better performance over the baseline uniform sampling. We can also notice a very significant better performance of COMM algorithm combined with the prioritized sampling based on TD error labeled TD+COMM over all sampling methods.

Lunar Lander results shown in Fig. 6, although having a more modest improvement of the best performing configuration, TD+COMM shows a behavior consistent with the Waterworld one depicted in Fig. 5. From Fig. 6 we can also notice that the vanilla approach of COMM-A, even slightly outperforms the vanilla TD in the second portion of the plot which is characterized by a lower  $\epsilon$  value. The curves for COMM ONLY and TD ONLY have similar tendencies which are linearly inclined while TD COMM, and US seem to experience over-fitting and saturation since their gradients lower towards the end of learning period.

The evaluated environments differed to a high degree in the nature of the learning problem to be solved, which, in turn, demanded different agent configurations, yet, the COMM algorithm performance was consistent along them. Waterworld agent was characterized by a very high state dimension space of 152 variables interfaced as an omnidirectional array including 30 directional sensors, which allowed it to detect the food pieces in all directions. Lunar Lander agent, on the contrary, required much less information from the environment in order to learn how successfully land the craft. For this learning problem the craft needed to be equipped with only 8 sensors, which were all proprioceptors measuring the craft positional and velocity data. Furthermore, they differed in implementation of reinforcement functions: Waterworld environment had a simple, crisp function which provided an agent with a reward only when the food was eaten while the Lunar Lander's reinforcement function provided a constant reward which was constantly re-calculated based on the craft's current behavior.

### 6.2. Evaluations of COMM-B

The results presented in Fig. 7 show that the proposed COMM-B approach enabled the agents to continuously improve their learn-

ing performance throughout the learning process in simpler games such as Pong shown in Fig. 7a as well as the more complex ones such as Enduro and Freeway presented in Fig. 7b, f. In these cases the COMM's learning curve steepness is indicative of a significant improvement in agent's performance compared to the baseline uniform sampling method US.

Some of the more complex Atari games such as Chopper Command, River Raid and Seaquest yielded more interesting learning curves as shown from Fig. 7e, c and d, respectively. From Fig. 7e we can see that COMM falls below the uniform sampling at the very beginning in Chopper Command environment but quickly outperforms the baseline significantly and keeps its consistency towards the end. In River Raid, COMM gains an advantage in the second half of the learning process and then increases its performance continuously over the baseline as evident from Fig. 7c.

Out of all of the evaluated Atari environments the Seaquest game haven't seemed to support the full benefits of COMM algorithm relative to the baseline: It is noticeable from Fig. 7d that an implementation of COMM made a learning process a more consistent one, but eventually the baseline performed much better towards the convergence at the last third of the learning steps.

From all of the results showcased in Fig. 7 we can see that for most of the environments, the COMM approach offers an improvement in the convergence of the reinforcement learning algorithm. We can also observe that the increase of performance depends on the specific game; COMM achieves the most significant and stable performance increase throughout the learning process in highly dynamical environments with elements moving mostly independently from the action taken, such as Freeway, which is characterized by a high number of moving elements (cars), and Enduro, a racing game where the dynamics of changing track along with the passing cars make the variance of the perceived state observations high.

## 7. Conclusion

The presented approach is aimed at further improving the dynamics of sampling in RL exploiting experience replay, by storing and replaying the correlated memories separately in order to increase the learning potential of the sequential experience stream. With the proposed correlation minimizing contextual memory retrieval an artificial learning agent is able to support and amplify the effect that experience replay brings to the learning process by selecting memories that are least correlated. As shown in Section 6, the proposed COMM algorithm shows a better performance than that of different learning settings in both vanilla and TD prioritization configurations.

While prioritization methods based on the TD-error [6,23] are a very efficient way to improve the speed of convergence of TD based algorithms they also induce additional correlation of the experiences stored in the replay memory. In this case, contrary to the uniform sampling, the buffer is filled with experiences that are correlated with the value of TD error, and since this error is a function of the agent's sensed state, we infer that this correlation is propagated for the experiences overall. This could justify the high performance gains of the TD-COMM configuration because the potential correlation increase by the TD approach was further minimized by the COMM algorithm. Experimental results outlined in Fig. 7 support the hypothesis that COMM's advantage is highest in the environments that are highly stochastic in nature such as Enduro and Freeway games.

COMM ability to reduce the correlation between experience samples is not limited just to RL applications. It can also be applied to different types of general unsupervised and supervised learning that use a fixed dataset in order to train a ANN. This gives COMM approach a more general scope as an optimized sampling tech-

nique that can be applied to any given dataset, not only the one that is created dynamically by the reinforcement examples presented in this work.

### Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

### Supplementary material

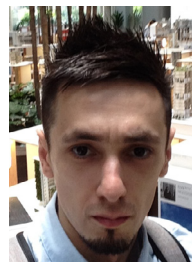
Supplementary material associated with this article can be found, in the online version, at doi:[10.1016/j.neucom.2020.02.004](https://doi.org/10.1016/j.neucom.2020.02.004).

### CRediT authorship contribution statement

**Mirza Ramicic:** Conceptualization, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing - original draft, Writing - review & editing. **Andrea Bonarini:** Methodology, Validation, Project administration, Resources, Supervision, Writing - review & editing.

### References

- [1] C.J. Watkins, P. Dayan, Q-Learning, *Mach. Learn.* 8 (3–4) (1992) 279–292.
- [2] R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction*, 1, MIT Press, Cambridge, 1998.
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, *Nature* 518 (7540) (2015) 529–533.
- [4] L.-J. Lin, *Reinforcement learning for robots using neural networks*, Doctoral Dissertation, Carnegie-Mellon University School of Computer Science, Pittsburgh PA, 1993.
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, *CoRR abs/1312.5602*, arXiv:[1312.5602](https://arxiv.org/abs/1312.5602) (2013).
- [6] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, *ICLR 2016*, arXiv:[1511.05952](https://arxiv.org/abs/1511.05952) (2015).
- [7] R.M. Kretschmar, Parallel reinforcement learning, in: *Proceedings of the 6th World Conference on Systemics, Cybernetics, and Informatics*, Citeseer, 2002.
- [8] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: *Proceedings of the International conference on machine learning*, 2016, pp. 1928–1937.
- [9] H.Y. Ong, K. Chavez, A. Hong, Distributed deep q-learning, *CoRR abs/1508.04186*, arXiv:[1508.04186](https://arxiv.org/abs/1508.04186) (2015).
- [10] A.V. Clemente, H.N. Castejón, A. Chandra, Efficient parallel methods for deep reinforcement learning, *CoRR abs/1705.04862*, arXiv:[1705.04862](https://arxiv.org/abs/1705.04862) (2017).
- [11] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, et al., Massively parallel methods for deep reinforcement learning, *CoRR abs/1507.04296* arXiv:[1507.04296](https://arxiv.org/abs/1507.04296) (2015).
- [12] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '11*, page 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0982657161.
- [13] R. Caruana, Multitask learning, *Mach. Learn.* 28 (1) (1997) 41–75.
- [14] M.G. Bellemare, W. Dabney, R. Munos, A distributional perspective on reinforcement learning, in: *Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR.org*, 2017, pp. 449–458.
- [15] M. Ramicic, A. Bonarini, Entropy-based prioritized sampling in deep q-learning, in: *Proceedings of the 2nd International Conference on Image, Vision and Computing (ICIVC)*, 2017, IEEE, 2017, pp. 1068–1072.
- [16] M. Ramicic, A. Bonarini, Selective perception as a mechanism to adapt agents to the environment: an evolutionary approach, *IEEE Trans. Cogn. Dev. Syst.* (2019), doi:[10.1109/TCDS.2019.2896306](https://doi.org/10.1109/TCDS.2019.2896306). 1–1
- [17] Y. Jin, B. Sendhoff, Reducing fitness evaluations using clustering techniques and neural network ensembles, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, Springer, 2004, pp. 688–699.
- [18] S. Chandar, M.M. Khapra, H. Larochelle, B. Ravindran, Correlational neural networks, *Neural Comput.* 28 (2) (2016) 257–285.
- [19] A. Benton, H. Khayrallah, B. Gujral, D. A. Reisinger, S. Zhang, and R. Arora. Deep generalized canonical correlation analysis. In *Proceedings of the 4th Workshop on Representation Learning for NLP (ReL4NLP-2019)*, pages 1–6, Florence, Italy, Aug. 2019. Association for Computational Linguistics. doi:[10.18653/v1/W19-4301](https://doi.org/10.18653/v1/W19-4301).
- [20] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, D. Batra, Reducing overfitting in deep networks by decorrelating representations, *ICLR'15*, arXiv:[1511.06068](https://arxiv.org/abs/1511.06068) (2015).
- [21] A. Karpathy, Reinforcejs framework, 2013, (<https://github.com/karpathy/reinforcejs>). (Accessed 6 September 2018).
- [22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, *Openai gym*, 2016.
- [23] J. Zhai, Q. Liu, Z. Zhang, S. Zhong, H. Zhu, P. Zhang, C. Sun, Deep q-learning with prioritized sampling, in: *Proceedings of the International Conference on Neural Information Processing*, Springer, 2016, pp. 13–22.



**Mirza Ramicic** Born in Banja Luka, Bosnia and Herzegovina on August 25th 1983 where he also obtained Bachelor and Master degree in Information Technology Engineering. He obtained his Ph.D. title at Politecnico di Milano in the area of Information Technology under the supervision of Professor Andrea Bonarini. He is a member of AI and Robotics Lab at Politecnico di Milano. Currently employed as a postdoctoral researcher at Artificial Intelligence Center (aic.fel.cvut.cz) as a part of the Faculty of Electrical Engineering of Czech Technical University in Prague.



**Andrea Bonarini** (Milano, 1957). Laurea (Master) in Electronics Engineering (Computer Engineering area), 1984. Ph.D. in Computer Engineering in 1989 from Politecnico di Milano. Master in Neuro-Linguistic Programming in 1993, from IIPNL. Full professor and Chair of the Ph.D. Program in Information Technology at Politecnico di Milano, Department of Electronics, Information, and Bioengineering. Since 1990 he is coordinating the AI and Robotics Lab at Politecnico di Milano (AIRLab <http://www.deib.polimi.it/eng/deib-labs/details/21>). He has been nominated Fellow of the Alta Scuola Politecnica (<http://www.asp-poli.it>) in 2012. He is among the founders of the Italian Association for Artificial Intelligence (AI\*IA) and the Italian Regional Interest Group of the IEEE Neural Network Council, now Italian Chapter of the IEEE Computational Intelligence Society (Chair from 2008 to 2010). He has been a coordinator of the Working Group on Robotics of the AI\*IA from 2003 to 2006. He participated since 1997 to the Robocup initiative (member of the Executive Committee from 2002 to 2010 ([www.robocup.org](http://www.robocup.org))). He is currently in charge of "Informatics", "Artificial Intelligence", "Robotics and Design", and "Soft Computing" courses at the Politecnico di Milano. He has given and gives courses about "Uncertainty", "Fuzzy Logic", "Soft Computing" and "Designing Interaction" within the Ph.D. program of Politecnico. He has tutored more than 150 Laurea (Master) Theses, some ERASMUS Theses, Alta Scuola Politecnica theses, and 12 Ph.D. Theses in the AI, Machine Learning, and Robotics fields. He has participated and leaded several EU, national, and industrial projects. He was also representative for Politecnico di Milano in the EU Networks of Excellence on Fuzzy Systems, and on Qualitative Reasoning, and he is currently National Representative in the COST Action LUDI - Play for Children with disabilities. Since 1989, he has realized with his collaborators and students more than 50 autonomous robots. Among the recent recognitions, he won the Kazuo Tamie Award for a robot to support therapy with autistic children. His research interests are focusing on Human-Robot Interaction, but still include Intelligent Data Interpretation, Autonomous Robotic Agents (in particular Edutainment, Entertainment and Robogames), Affective Computing, Reinforcement Learning, and Fuzzy Systems. He has published more than 150 peer-reviewed papers in international journals, books, and proceedings of international congresses.