Correlation Minimizing Replay Memory in Temporal-Difference Reinforcement Learning

Mirza Ramicic

Artificial Intelligence Center
Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic
Email: ramicmir@fel.cvut.cz

Andrea Bonarini

Artificial Intelligence and Robotics Lab
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Milan, Italy
Email: andrea.bonarini@polimi.it

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.

Keywords: Reinforcement learning, Temporal-difference Learning, Replay memory, Artificial Neural Networks

1. Introduction

Temporal difference (TD) types of reinforcement learning (RL) such as Q-learning [22] are model-free learning algorithms that differ from a more basic approach as that of Monte Carlo methods [20] 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 Θ at each learning step in order to obtain a better prediction about the value of the 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 learning process [13]. 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 [11, 12, 13]. 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 [11, 12, 13]. Experimental results show that *COMM* obtains an improvement in learning performance over the standard Q-learning with uniform sampling and shows a very significant improvement when combined with prioritized sampling based on temporal difference error [19].

2. Related Work

The works by Kretchmar et al. [10], Mnih et al. [14], Ong et al. [16] and Clemente et al. [6] presented asynchronous parallel learning methods which were addressing the correlation problem in replay memory, which scales nicely in the work by Nair et al. [15] and Sutton et al. [21]. The approaches are based on the idea of *multitask learning* [4] 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. [1] 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. [19] 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. [19] was presented by Ramicic and Bonarini [17], 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 [18] 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. [8]. 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. [5] 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.[2] and Cogswell et al. [7].

3. Theoretical Background

The aim of the learning process is to create a policy π which maps the current state of an agent to a preferred action in order to maximize its reward potential in the long run [20]. 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 π making it closer to the *optimal policy* π^*, which is represented by an optimal action-value function $Q^*(s, a)$ shown in Equation 1 and defined as the maximum expected return while following the policy π.

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

Bellman's Equation 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.

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

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 Equation 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 equationEquation 2 which becomes the target y. In order to make the current prediction $\hat{y}$ closer to the target y, a parameter update on Θ is performed in order to minimize the loss function $L(\Theta)$ shown in Equation 3.

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

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

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

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

Equation 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 Θ 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 Equation 5.

$$\Delta L \approx \frac{\partial L}{\partial w_i}\Delta w_i \tag{5}$$

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 α called *learning rate*.

$$w_i = w_i - \alpha\frac{\partial L}{\partial w_i} \tag{6}$$

4. Model Architecture and Learning Algorithm

The main part of the learning mechanism represents the *learning loop* outlined in section (b) of Figure 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 feedback from its environment in the form of a scalar reward r_t. These parameters form a tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ that fully determines a transition and contains every information is needed to perform a learning step. After an agent has transitioned and the values of the tuple components are obtained, it is possible to apply Equation 2 to calculate the expected value for the state-action pairs $Q(s_t, a_t)$. The update is performed in the module of section (d) of the *learning loop* shown in Figure 1, by adjusting the weights of the approximating neural network using SGD in such a way that it is possible to minimize the squared error of the difference between the newly calculated expected Q-value and its current estimate. After a learning update, the states are updated and a new iteration of the *learning loop* is performed. This loop creates a stream of experiences (c) represented by a sequence of transition tuples e_t. In the basic approach of Q-learning (e.g., [11, 12, 13]), these experiences are sampled in a sliding window *replay memory* buffer from which a number of them are randomly selected to be reused after each iteration in order to perform additional training on approximator neural network.

With $COMM$ approach, instead of having only one *replay memory* buffer, we have *M contextual replay memory* buffers as shown in section (a) of Figure 1, each of which is corresponding to one of the M classes of the experiences coming from the *experience stream*. Before being stored, each experience is clustered using an unsupervised learning method to decide in which *contextual replay memory* it has to be sampled. To determine the class of the experience, an *autoencoder* neural network shown in (f) is used to predict the state s_t part of an experience from the same s_t on the input. The *autoencoder* performs a reduction of the dimension of the perceived state space from its original size to M dimensions or classes using a bottleneck hidden layer in the middle consisting of M fully connected neurons. The process similarly performs SGD in order to make the weights of a neural network a better predictor of itself by minimizing the difference between the predicted and the actual 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 $V_C = (C_1, C_2, ..., C_M)$ 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 Equation 7.

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

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

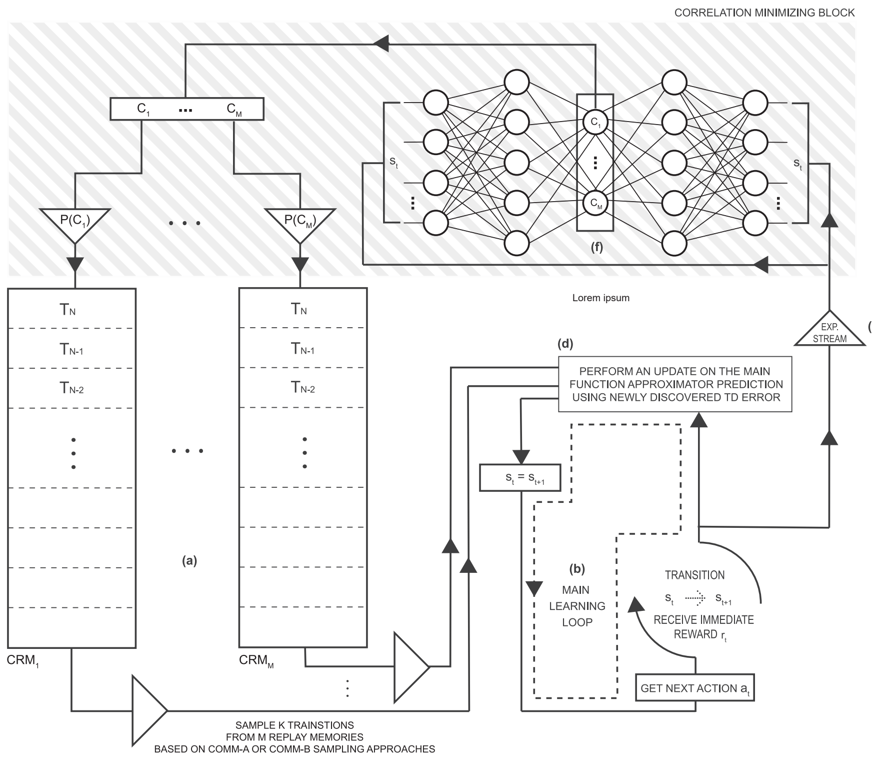

CORRELATION MINIMIZING BLOCK
C1 ... CM
P(C1)
P(CM)
st
(f)
Lorem ipsum
TN
TN-1
TN-2
(a)
CRM1
CRMM
(c)
EXP. STREAM
(d)
PERFORM AN UPDATE ON THE MAIN FUNCTION APPROXIMATOR PREDICTION USING NEWLY DISCOVERED TD ERROR
st = st+1
(b)
MAIN LEARNING LOOP
TRANSITION
st st+1
RECEIVE IMMEDIATE REWARD rt
GET NEXT ACTION at
SAMPLE K TRAINSTIONS FROM M REPLAY MEMORIES BASED ON COMM-A OR COMM-B SAMPLING APPROACHES

Figure 1: General learning model architecture including *correlation minimizing block*: (*a*) Array of M contextual replay memories, each of which stores the last N experiences of its own class in a sliding window buffer for later replay; (*b*) Main learning loop consists of: 1) the transition in which the agent performs an action a_t, receives an immediate reward r_t, and transitions to the next state s_{t+1}; 2) performing an update on main function approximator ANN (*d*) by backpropagating the TD error as a gradient of the a_t output; 3) shifting the states for the next iteration in which the s_t becomes s_{t+1}; 4) forwarding the current state through a function approximator in order to find out the best candidate action a_t based on its Q value for ϵ-greedy policy; (*d*) A block implementing Q-value function approximator taking the starting state s_t in input and predicting Q-values for each of the available actions on its output; (*c*) Raw stream of the experiences that are perceived, representing unfiltered cognition of an agent; (*f*) Context augmented block implemented as an autoencoder neural network performing unsupervised clustering of the experiences into M dimensions or classes, each of which determines the probability of the selected experience being sampled into the corresponding contextual replay memory buffer.

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 [12, 13], 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 [12, 13] are collected in a replay buffer of larger capacity over millions of learning steps and also sampled in larger batches. The shear size of the replay buffer batch in complex Atari games problems makes learning from the current experience at each step relatively simple. *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 $V_I = (I_1, I_2, ..., I_M)$ by clamping each value of the V_C between 0 and 1 and subtracting 1 from it as shown in Equation 8

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

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 Equation 9

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

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. 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*, and used a simpler Q function approximator with

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 ( $C_1, C_2 ... C_M$ )
    for $C_n$ = 1, M do
      $I_n = abs(clamp[0,1](C_n) - 1)$
    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 $P(I_i)$

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

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

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 ( $C_1, C_2$ ... $C_M$ )
        **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 $y_i = \begin{cases} r_i, & \text{terminal } s_{i+1} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \Theta), & \text{non terminal} \end{cases}$
        Perform a gradient descent step on $(y_i - Q(s_i, a_i; \Theta))^2$ according to Equation 4
    **end for**
**end for**
```

linear network layers only. This set includes the *Waterworld* environment (subsection 5.1) as implemented in a ReinforceJS framework [9], and the *Lunar Lander* environment subsection 5.2 which is a part of *OpenAI Gym* [3]. 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 insubsection 5.4.

5.1. Waterworld Environment - COMM-A

Waterworld environment [9] as showcased in Figure 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, 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 agent: up, down, left, right, stay, given the state space of 152 dimensions as input. The learning rate of an approximator α 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 ϵ 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 γ 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 yellow flags and set on an uneven terrain under the simulated weak moon gravity as depicted in Figure 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

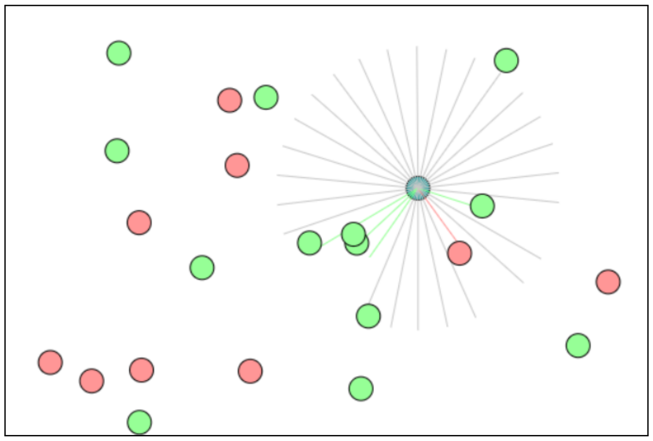


Figure 2: A single rendered frame from the Waterworld environment depicting an agent together with its sensor array learning to consume good (red) and avoid bad (green) food sources.

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.

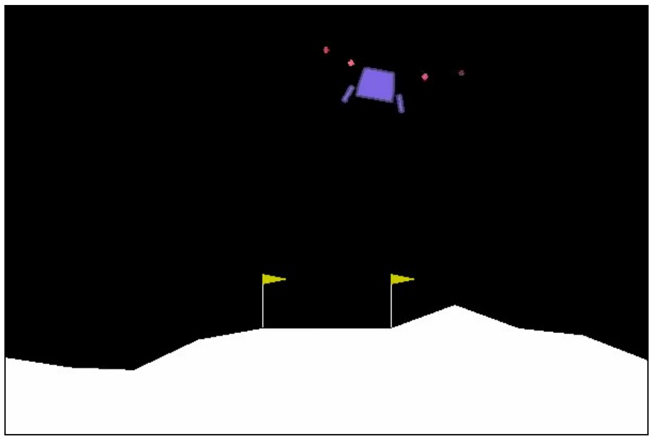


Figure 3: A single rendered frame from Lunar Lander environment depicting a craft firing its thrusters in order to land on the designated area marked by two yellow flags.

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 ϵ 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 γ was set at 0.99. The Q-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 12000 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

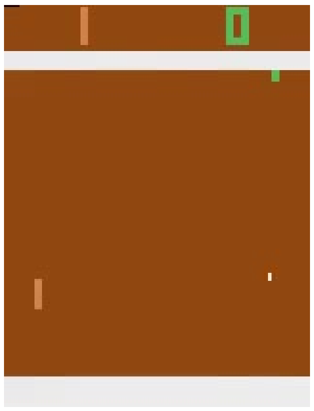
Atari 2600 games [12, 13] represent a radically different challenge to RL as their state space consists of the rendered pixels of the actual game: in this case the machine learning algorithm is presented with an input that is created or "optimized" for a human player senses. Under the Gym framework [3] the Atari games state space is defined as a RGB image of the screen as shown in Figure 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 72000 experiences while the $COMM$ approaches used a capacity of 8 x 9000. The epsilon parameter ϵ was decayed using a simple exponential function from 0.7 to 0.05 thorough the training steps while γ was 0.9.

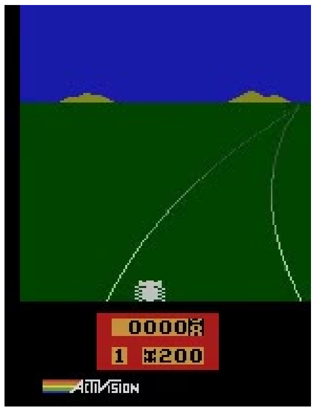
The main difference compared to the previous environments is that Atari games rely on a more complex convolutional neural network architecture (CNN, or ConvNet) in order to approximate the Q value function. The raw image input was pre-processed into a $(1, 84, 84)$ array by cropping and converting the RBG 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 α was set to the rather low value of 0.000015.

5.4. Unsupervised learning

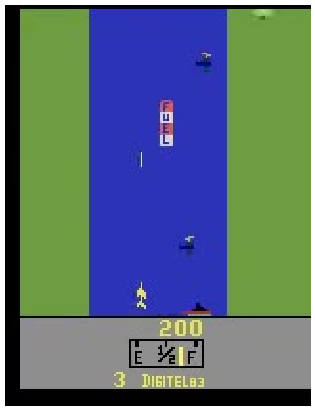
In order to decide in which mini-batch the sampled experience has to be inserted, an autoencoder neural network was used. The autoencoder used for the lower complexity environments such as was a simple linear neural network able to cluster up to 152 dimensional state into $M = 6$ dimensional array using 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



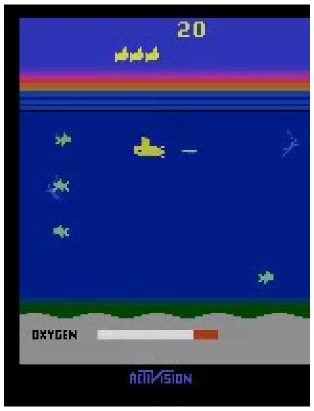
(a) Pong



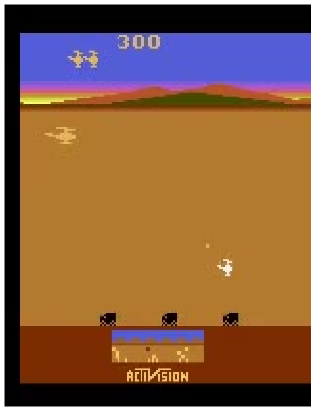
(b) Enduro



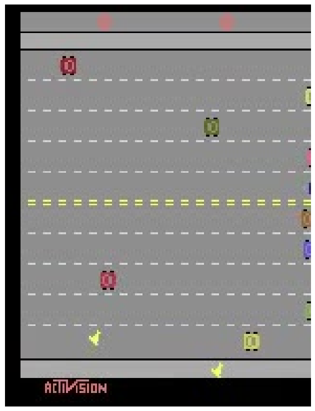
(c) River Raid



(d) Seaquest



(e) Chopper Command



(f) Freeway

Figure 4: Rendered frames from variety of tested Atari games also representing the agent's state space used in function appoximation during learning

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 architecure of the autoencoder itself: it contained a greater number of neurons which in turns 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 Figure 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 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 Figure 5 and Figure 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 [19].

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.

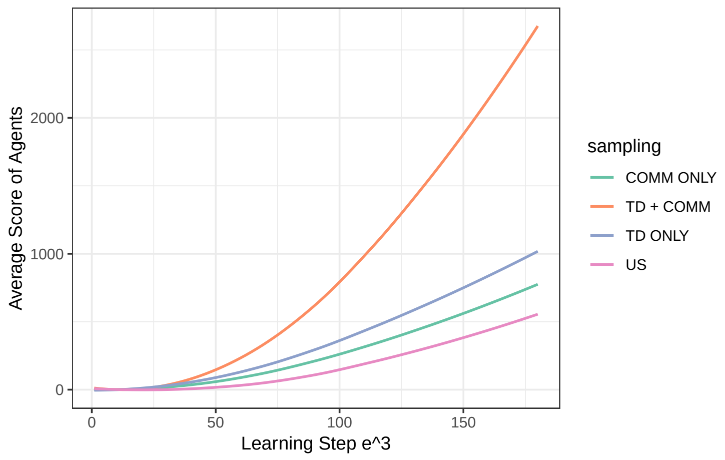


Figure 5: Average total reinforcement received over 50 trials with agents with different sampling dynamics during first 180.000 learning steps in Waterworld environment. $COMM$ represents the proposed method, $TD + COMM$ represents a combination of $COMM$ method and temporal difference prioritization, $TDONLY$ is temporal difference prioritization without $COMM$ sampling while US is the baseline uniform sampling.(Higheri s better)

Lunar Lander results shown in Figure 6, although having a more modest improvement of the best performing configuration, *TD+COMM* shows a behavior consistent with the Waterworld one depicted in Figure 5. From Figure 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 ϵ 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.

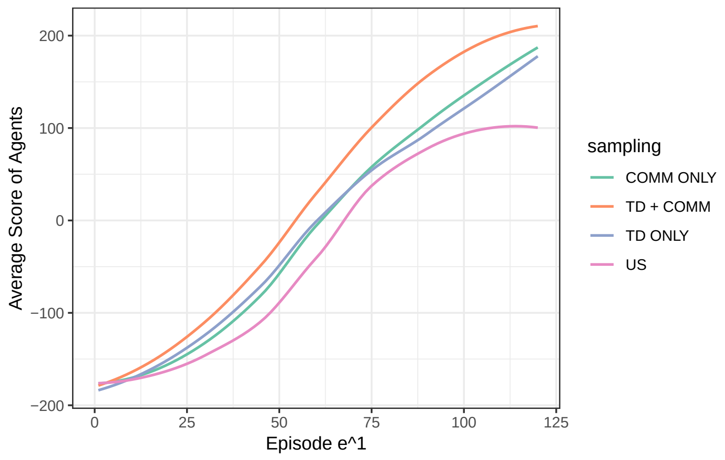


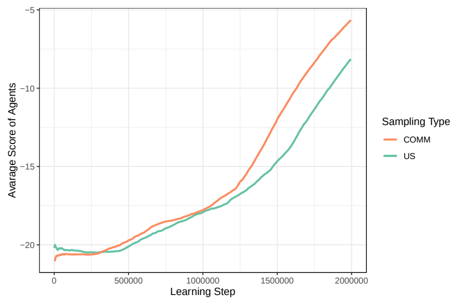
Figure 6: Average total reinforcement received over 30 trials with agents with different sampling dynamics during first 1500 episodes in Lunar Lander environment.

6.2. *Evaluations of COMM-B*

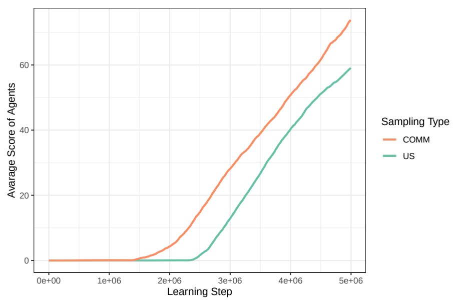
The results presented in Figure 7 show that the proposed *COMM-B* approach enabled the agents to continuously improve their learning performance throughout the learning process in simpler games such as Pong shown in Figure 7a as well as the more complex ones such as Enduro and Freeway presented in Figure 7b, Figure 7e. 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 in Figure 7c, Figure 7f and Figure 7d respectively. From Figure 7d 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 Figure 7c.
Out of all of the evaluated Atari environments the Seaquest game haven't seemed to support the full benefits of $COMM$ algorithm with respect to the baseline: it can be noticed from Figure 7f that $COMM$ made the learning process more consistent, but eventually the baseline performed much better towards the convergence at the last third of the learning steps.

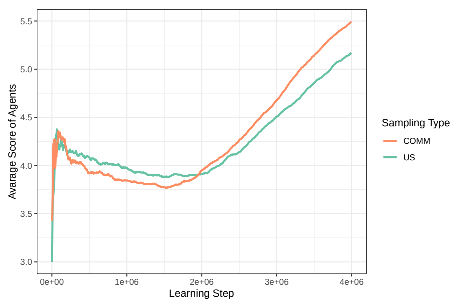
From all of the results showcased in Figure 7 we can see that for most of the environments $COMM$ approach offers an improvement in the convergence of the reinforcement learning algorithm. We can also observe that the increase of performance depends on the specific game; $COMM$ achieves the most significant 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.



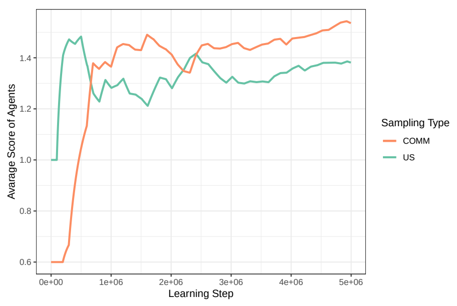
(a) Pong



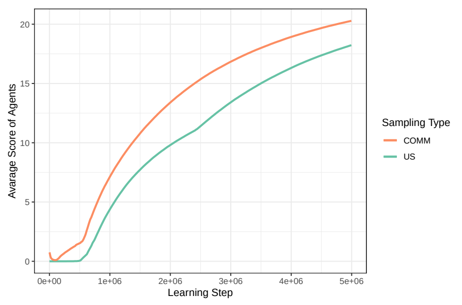
(b) Enduro



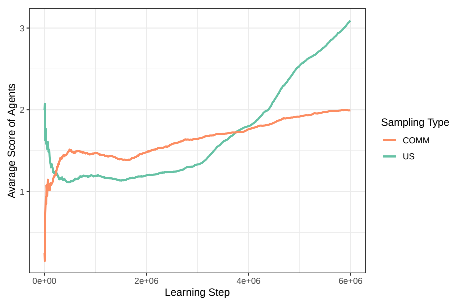
(c) River Raid



(d) Chopper Command



(e) Freeway



(f) Seaquest

Figure 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.

7. Conclusion

The presented approach is aimed at further improving the dynamics of sampling in RL exploiting experience replay, by grouping the uncorrelated memories together 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 with the current experience. As shown in Section 6, the proposed $COMM$ algorithm shows a performance better than that of different learning settings in both vanilla and TD prioritization configurations.

While prioritization methods based on the TD-error [19, 23] are a very efficient way to improve the speed of convergence of temporal-difference algorithm 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 Figure 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 technique that can be applied to any given dataset, not only the one that is created dynamically by the reinforcement examples presented in this work.

References

[1] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[2] A. Benton, H. Khayrallah, B. Gujral, D. A. Reisinger, S. Zhang, and R. Arora. Deep generalized canonical correlation analysis. *arXiv preprint arXiv:1702.02519*, 2017.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[4] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[5] S. Chandar, M. M. Khapra, H. Larochelle, and B. Ravindran. Correlational neural networks. *Neural computation*, 28(2):257–285, 2016.

[6] A. V. Clemente, H. N. Castejón, and A. Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

[7] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[8] Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In *Genetic and Evolutionary Computation Conference*, pages 688–699. Springer, 2004.

[9] A. Karpathy. Reinforcejs framework. `https://github.com/karpathy/reinforcejs`, 2013. Accessed: 2018-09-06.

[10] R. M. Kretchmar. Parallel reinforcement learning. In *The 6th World Conference on Systemics, Cybernetics, and Informatics*. Citeseer, 2002.

[11] L.-J. Lin. *Reinforcement learning for robots using neural networks*. Number CMU-CS-93-103. 1993.

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

[13] 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):529–533, 2015.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[15] 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. *arXiv preprint arXiv:1507.04296*, 2015.

[16] H. Y. Ong, K. Chavez, and A. Hong. Distributed deep q-learning. *arXiv preprint arXiv:1508.04186*, 2015.

[17] M. Ramicic and A. Bonarini. Entropy-based prioritized sampling in deep q-learning. In *Image, Vision and Computing (ICIVC), 2017 2nd International Conference on*, pages 1068–1072. IEEE, 2017.

[18] M. Ramicic and A. Bonarini. Selective perception as a mechanism to adapt agents to the environment: An evolutionary approach. *IEEE Transactions on Cognitive and Developmental Systems*, pages 1–1, 2019. ISSN 2379-8920. doi: 10.1109/TCDS.2019.2896306.

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

[20] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[21] R. S. Sutton, J. Modayil, M. D. T. Degris, P. M. Pilarski, and A. White. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. 2019.

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

[23] J. Zhai, Q. Liu, Z. Zhang, S. Zhong, H. Zhu, P. Zhang, and C. Sun. Deep q-learning with prioritized sampling. In *International Conference on Neural Information Processing*, pages 13–22. Springer, 2016.