A Methodology of Experiments

We evaluated the COMPER agent on the Arcade Learning Environment (ALE) [1, 4] [1], which consists of a development platform and a set of challenges for agents with different RL methods, such as non-determinism, stochasticity, and exploration and exploitation tasks. As its Q-target function is an important component of the method, we also evaluated three possible DNN architectures using Long-short Term Memory (LSTM), Multi-layer Perceptron (MLP), and Convolutional Neural Network.

Relevant works in the literature have used different methodologies for the development and evaluation of RL agents, making it challenging to analyze and compare their results. Therefore, in [4] the authors proposed a very well-defined methodology for experiments and agent evaluation using ALE and presented a benchmarking with the experimental results for DQN [5] and SARSA(λ) + blob-PROST [3]. Mostly, we follows their propositions, and highlight the following: (i) Agents should be evaluated over the training data and not in "test mode", because we are interesting to evaluate their learning processes, not their gaming performance; (ii) The agent's performance should be evaluated on different check points of the training considering the last episode scores; (iii) Agents interact with the game environment in an episodic way. An episode begins by putting the environment to its initial state and ends in a natural final state for the game; (iv) The main measure of the agent's effectiveness consists of the undiscounted sum of the rewards for each episode; (v) ALE explore non-determinism and stochasticity using two techniques called *stick actions* and *frame skipping*. We use both with the same configuration as [4]; (vi) Three approaches are commonly used for the environment's states representation: (a) *color averaging*, (b) *frame pooling*, and (c) *frame stacking*. ALE implements color averaging. Frame stacking is not implemented directly by ALE and is an algorithm decision. Color averaging and frame pooling may end up removing the most interesting form of partial observation in ALE, and frame stacking reduces the degree of partial observability [4]. We use color averaging and make experiments using single frames (that is, not staked) and staked frames. (vii) There are differences in the literature on how to summarize and measure the results and on how the agent performance is evaluated based on these results. These questions lead to an important methodological decision, which refers to the moment when each training episode must end. We consider only the signal of "game over" as the end of an episode, as strongly recommended by [4]; (viii) The way the results are measured and summarized can hamper or improve the analysis of an agent's learning progress. It should be carried out on a fixed number of the n last episodes at different checkpoints during an agent training to allow the evaluation of its progress. (ix) Training an agent over a given number of episodes before evaluating it can yield misleading results, as the duration of each episode can vary widely between different games and as a function of the agent's performance. A more interesting approach is to measure the amount of training data in terms of the total number of environ-

[1] Available at https://github.com/mgbellemare/Arcade-Learning-Environment.

ment states (video frames on Atari 2600), which makes reproducing experiments and comparing the results easier. Furthermore, interrupting an ongoing episode as soon as the total number of frames is reached, or waiting for the end of the episode does not represent a clear choice in the literature [4]. This way, our training trials stop condition is: if the total frames count equals the total number of frames to observes in a trial, and the game is over (that is, reached a final state), then we stop the run; (x) To measure the agents' capacity for generalization, we use only a smaller subset of games to fit the hyperparameters and train and evaluate the agents using the defined values (and without further adjustments) over all the games set [4].

A.1 Evaluation Procedures of COMPER

In [4] the authors defined a limit of 200 million frames for each agent training trial and four checkpoints at 10, 50, 100, and 200 million frames. They pointed out that this total number of frames implies a great computational cost and a lot of time in experiment execution and, therefore, their methodology allows any researcher to report results earlier during experiments, minimizing this problem. They have computed the rewards average and standard deviation over the last 100 episodes preceding each checkpoint and performed 5 training trials of a DQN agent on 5 games (Asterix, Beam Rider, Freeway, Seaquest, and Space Invaders) to tune the hyperparameters, which they used for agent training on all the 60 games available. We define a limit of 100 thousand frames for each training trial and log the results at every 100 iterations and end of the episodes. We divided the number of episodes obtained in each trail to split the results into 3 checkpoints and calculated the rewards average and standard deviation for the last 5 episodes preceding each checkpoint. These adaptations were necessary because of the smaller number of episodes resulting from the limit of the total number of frames, which is 0.0005% of the total used in [4]. This way, we also redefined the decay rate of the hyperparameter ϵ, so that it starts at 1.0 and drops to 0.01 in 90 thousand frames (instead of 1 million frames as in [4]). In COMPER the weights Ω in $QT(\tau_t, \Omega)$ are updated through backpropagation over sets of similar transitions every 100 agent steps. In turn, the update of $Q(s_t, a_t, \Theta)$ is performed every 4 agent steps, similar to DQN [4]. Moreover, we start weight updating after the first 100 steps in COMPER and 1,000 steps in DQN (instead of 50,000 as in [4]). Finally, we evaluate COMPER by representing the states of the environment either with a single frame in a matrix with dimensions of $84 \times 84 \times 1$ or with frames stacked in a matrix of $84 \times 84 \times 4$. In both cases, we just used the luminance values for every two frames using the color averaging method of ALE. We implemented our DQN agent as defined in [5] and only adopted new hyperparameters values for the total number of frames, the ϵ value decay, the results log frequency, the learning start, and the target function update frequency to accomplish our experiments.

We tuned the hyperparameters over only 3 training trials on the game Space Invaders both for COMPER and DQN. Then we used the best values to run the experiments over all the other games. To build the Transitions Memory Index

$\mathcal{TMI}$ we used the class library named Faiss [2], which was developed for efficient similarity search and clustering of dense vectors. According to the authors, it searches in sets of vectors of arbitrary size, up to ones that possibly do not fit in RAM. We used the library's class IndexFlatL2, that applies Euclidean distance and return a matrix with the distances between the input (querying) vector and other n vectors in the set. We defined the distance threshold (maximum distance) through a simple parameter, that we named δ. We used strict similarity with $\delta = 0$ to closely verify our hypotheses about the possibility of identifying similar transitions sets.

B Experimental Setup and Parameters

There are several sets of specific hyperparameters for different purposes. Some of them are independent of the methods being evaluated while others address aspects inherent to each of them, as we present in the next sections.

B.1. General Parameters of ALE and the Evaluation Protocol

In Table 1, we present the parameters used to set up the Arcade Learning Environment (ALE), to evaluate the agent's learning, and to summarize the results. The other parameters used to configure the environment have the same values define by [4]. Those that refer to summarizing the results were defined as discussed in Subsection A.1. In Table 2, we present the parameters related to the exploration (and exploitation) rate, and the discount factor.

Hyperparameter	**Value**	**Description**
Action set	Full	18 actions are always available to the agent.
Frame skip	5	Each action lasts 5 time steps.
Stochasticity (ς)	0.25	We used $\varsigma = 0.25$ for all experiments.
Lives signal used	False	We did not use the game-specific information about the number of lives the agent has at each time step.
Number of episodes used for evaluation	3 or 10	We report the average score obtained in the last 3 or 10 episodes used for learning. See Subsection A.1 for details.
Number of episodes used for evaluation in the continual learning	3 or 5	In the continual learning setting, we report the average score obtained in the last 3 or 5 episodes at the end of each tertile. See Subsection A.1 for details.
Number of frames used for learning (T)	100,000	We report the scores after 100,000 frames and at the end of each tertile of episodes.
Number of trials	5	COMPER and DQN were evaluated in 5 trials.

Table 1: Parameters used to set up the environment and evaluate the methods.

Hyperparameter	Value	Description
Discount factor (γ)	0.99	The discount factor applied in the update rule on the expected long-term reward.
Initial exploration rate (ϵ)	1.0	Probability of a random action will be taken at each time step.
Final exploration rate (ϵ)	0.001	Probability of a random action will be taken at each time step.
Final exploration frame	90,000	Number of frames over which ϵ is linearly annealed.

Table 2: Discount factor and exploration rate used by COMPER and DQN.

B.2. Specific parameters of COMPER

COMPER uses different models and architectures of deep neural networks to approximate their value functions, two transition memories, and a reduction process from one to another, requiring specific sets of parameters, which are presented in Tables 3, 4, and 5.

Hyperparameter	Value	Description
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0.95	Gradient momentum used by RMSProp.
Squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
Min squared gradient	0.01	Constant added to the denominator of the RMSProp update.

Table 3: Parameters used in CNN training by COMPER.

Hyperparameter	Value	Description
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0	Gradient momentum used by RMSProp.
Decay	0.9	Discounting factor for the history and coming gradient used by RMSProp.
Epsilon	1e-10	Constant added to the denominator of the RMSProp update.

Table 4: Parameters used in LSTM training.

B.3. Specific parameters of DQN

In Table 6, we present the parameter used on DQN. Those related to the memory of experiences and the target network update frequency (**in bold**) were adjusted

Hyperparameter	**Value**	**Description**
Transition memory size	100,000	The transition memory size is defined to stores up to 100,000 sets of last similar transitions. However, it is reduced every when these sets are sampled.
Replay start size	100	Number of frames over which a random policy is executed to first populate the transition memory.
History length (single frame)	1	Number of most recent frames the agent observed that are given as input to the CNN when using single frame.
History length (stacked frames)	4	Number of most recent frames the agent observed that are given as input to the CNN when using stacked frames.
Train Frequency (TF)	4	Number of iterations between the CNN model updates.
Transitions batch size (K)	32	The number of transitions sampled from the reduced transitions memory to update the CNN.
Similar transitions batch size	1,000	The number of sets of similar transitions taken from the transition memories to produce the reduced transition memory.
Similarity Threshold δ	0.0	Maximum distance to consider two transitions as similar
Update Frequency (UTF)	100	The frequency with which the target network (LSTM) is trained and the reduced transition memory is updated by the QLSTM algorithm.
Color averaging	True	The observation received is the average between the previous and the current frame.
Number of different colors	8	NTSC is the color palette in which each screen is encoded but only the luminance channel is used

Table 5: General parameters used by COMPER.

due to the smaller number of frames used in the experiments. All others have the same values defined by [4].

C. Deep Neural Network Architectures used in COMPER

The convolutional neural network (CNN) used in COMPER to approximate the Q-Value function has the same general architecture as proposed by [5] and used in [4]. The difference is in the input layer because when COMPER is using a single frame it represents the frames as a matrix of $84 \times 84 \times 1$ and when using stacked frames its shape is $84 \times 84 \times 4$. This representation is produced by a preprocessing step that reduces the original dimensionality of the game frames and extracts the luminance channel. Another difference is in the output layer. As recommended by [4] and discussed in Section A.1, the COMPER agent has access to the complete set of actions on ALE, independently from the game, so the output layer is a fully-connected layer with a single output for each of the 18

Hyperparameter	**Value**	**Description**
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0.95	Gradient momentum used by RMSProp.
Squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
Min squared gradient	0.01	Constant added to the denominator of the RMSProp update.
Replay memory size	100,000	The samples used in the algorithm's updates are drawn from the last 100 thousand recent frames.
Replay start size	1,000	Number of frames over which a random policy is executed to first populate the replay memory.
History length (stacked frames)	4	Number of most recent frames the agent observed that are given as input to the CNN.
Update frequency	4	Number of actions the agent selects between successive updates.
Update frequency of target network	1,000	Number of actions the agent selects between successive updates of the target CNN parameters.
Frame pooling	True	The observation received consists of the maximum pixel value between the previous and the current frame.
Number of different colors	8	NTSC is the color palette in which each screen is encoded but only the luminance channel is used.

Table 6: Parameters used by DQN

valid actions. The hidden layers are three convolutional and one fully-connected. The first convolutional layer applies 32 filters of 8×8 with stride 4 on the input frame image representation, followed by a ReLU nonlinearity. The second layer convolves 64 filters of 4×4 with stride 2 and also applies a ReLU nonlinearity. The third hidden layer convolves 64 filters of 3×3 with stride 1, again followed by a ReLU nonlinearity. The final hidden layer consists of 512 units. The RMSProp optimizer uses the parameters shown in Table 3 and this architecture is detailed in Table 7.

The LSTM network that predicts values for the target function has the input layer with shapes of 1×14114 or 1×56450, which represents a transition τ for single frame and stacked frames, respectively. The batch size is 16. The output is a fully-connected linear layer with only a single output for the predicted Q-Value to the input transition. The hidden layers are three LSTM and one fully connected. The first LSTM layer contains 64 internal units, applies a hyperbolic tangent activation function and sequence return. The second layer contains 32 internal units, and also applies a hyperbolic tangent activation function and sequence return. The third LSTM layer also contains 32 internal units and applies a hyperbolic tangent activation function, but has no sequence return. The last

Layer (type)	Definition	Output Shape	Parameters

Conv1 (Conv2D)	Filters = (32, (8, 8))
	Strides= (4, 4)
	Input = (84, 84, 1)* or
	Input = (84, 84, 4)**
		(None, 20, 20, 32)	2,080*
			or
			8,192**

Conv2 (Conv2D)	Filters = (64, (4, 4))
	Strides= (2, 2)
		(None, 9, 9, 64)	32,832

Conv3 (Conv2D)	Filters = (64, (3, 3))
	Strides= (1, 1)
		(None, 7, 7, 64)	36,928

Flatten	Flatten	(None, 3136)	0
Dense1 (Dense)	512 Units	(None, 512)	1,606,144
Dense2 (Dense)	18 Units	(None, 18)	9,234

Table 7: CNN architecture used in COMPER with single frame. Note that * indicates the values for single frame and ** the values for stacked frames.

hidden layer contains 18 fully connected units and applies a ReLU nonlinearity. The RMSProp optimizer uses the parameters shown in Table 4 and this architecture is detailed in Table 8.

C. Relevant Aspects of Agent Evaluation in ALE

The Arcade Learning Environment (ALE) [1] consists of an Atari 2600 game emulator and a set of challenges that includes non-determinism, stochasticity, and environments with different values and ranges for awarding rewards, allowing the development and evaluation of reinforcement learning agents. In [4] the authors presented a revision of ALE and introduced features such as the support for different difficulty game levels. They also discussed the methodological differences between relevant works in the literature regarding the agent evaluation procedure, which makes the analysis and comparison of results difficult. Therefore, the authors proposed a very well-defined methodology for agents evaluation using

Layer (type)	Definition	Output Shape	Parameters

Lstm1 (LSTM)	Units = 64
	Return Sequences = True
	Input = (1, 14114)* or
	Input = (1, 56450)**
		(None, 1, 64)	3,629,824* or
			14,467,840**

Lstm2 (LSTM)	Units = 32 Return Sequences = True	(None, 1, 32)	12,416
Lstm3 (LSTM)	Units = 32	(None, 32)	8,320
Dense1 (Dense)	Units = 18	(None, 18)	594
Dense2 (Dense)	Units = 1	(None, 1)	19

Table 8: LSTM architecture used in COMPER. Note that * indicates the values for single frame and ** the values for stacked frames.

ALE and presented a benchmarking with the experimental results for DQN [5] and SARSA(λ)+ blob-PROST [3].

An agent should interact with as many games as possible without using any specific information from these games, acting based just on its perceptions from the video streaming, as these games present multiple and different tasks, are challenging even for human beings, and are free of possible biases introduced by researchers. This interaction with the environment happens episodically, and each episode starts in the initial game state and finishes when the game is over at a final state, as when the agent runs out of all its attempts, exceeds a time limit, or reaches its goal. The undiscounted sum of the agent rewards at each episode is the principal measure of its performance, but this can make it difficult to obtain more detailed information about its learning progress. The agent may have learned to focus on actions that lead to subsequent and small rewards, or it may have simply learned to survive in the game by increasing its chances of receiving some reward. Even so, besides being a common main objective of reinforcement learning agents, the evaluation of its total rewards allows to verify if a method leads the agent to an effective learning process, how stable is its behavior between different training trials, and how the different methods affect that evaluation considering the distinct challenges represented by the diversity of games [4].

In most games, an agent has a finite number of "lives", which are lost one by one each time it fails to fulfill its goal, causing the game to ends when the available number of lives runs out. On the other hand, the success of an agent also leads to the game ends. Many factors can cause the game to ends: a time limit that ends regardless of the number of lives lost or an agent decision that takes you to a point in the game from which there is no way out. In ALE, it is possible to end a training episode as soon as an agent loses its first life or only when the game effectively ends. The second option can prevent an agent from learning the meaning of losing a life, but ending the game right after the agent loses its first life can make it just learns not to die rather than accumulate rewards. This way, while both approaches are present in the literature, ending an episode when an agent loses a life is detrimental to its performance. Thinking about the main objective of an agent, (i.e. learning a policy that leads to the highest discounted accumulated reward in the long run) learning "just" not to die may not lead to the best choice of actions. Therefore, it is strongly recommend that only the game is over signal be used to end an episode [4]. Therefore,the differences in the literature regarding the results measurement and the agent's performance evaluation based on these results imply a relevant methodological decision about when each training episode ends.

The way results are measured and summarized can hamper or improve the analysis of an agent learning process. When evaluating an agent on a small set of problems, it is possible to describe its learning curves in detail, measuring important information such as its learning speed, its best-presented performance, the stability of its algorithm, and how much its performance improves while it observes more and more data. But this process becomes difficult on the set of all games present in ALE. To facilitate comparative analysis, many authors present their results by numerically summarizing the agents' performance for each game, but with different approaches, and it impairs the direct comparison of results. To machado18arcade, the analysis of the results should be carried out on a fixed number of the last episodes at different times of agent training, allowing the evaluation of its learning evolution. In this way, unstable methods (that show a great deal of variation on the reward values between different episodes) will present a low evaluation compared to those that improve steadily and continuously. Although the performance at the end of the training is of most interest, this approach allows one to evaluate agent evolution through stated points without tracing the entire learning curve, allowing researchers to report results earlier during experiments, minimizing the problem of agents' evaluation associated with the computational cost. In this sense, and according to the authors, agents should be evaluated over the training data and not in "test mode" because we are interested in evaluating its learning and not its in-game performance.

Training an agent over a given number of episodes before evaluating it can yield misleading results, as the duration of each episode can vary widely between different games. Furthermore, the better or worse the agent's performance, the longer or shorter the training episodes could be for the same game in distinct runs. Thus, agents that learn good policies earlier tend to receive more training data than those that learn more slowly, significantly increasing the magnitude of the difference between them for a specific number of episodes. Therefore, a more interesting approach is to measure the amount of training data in terms of the total number of observed frames by an agent, which facilitates results reproducibility, analysis, and comparison. Even so, interrupting an ongoing episode as soon as the total number of frames is reached or waiting for the end of this episode does not represent a clear choice in the literature. So, machado18arcade recommended ending the agent training based on a total number of frames without interrupting the episode in which this total is reached.

Non-determinism and stochasticity consist of two fundamental aspects of reinforcement learning since an agent needs to learn to deal with partially observable and dynamic environments. Therefore, machado18arcade introduced a form of stochasticity named "stick actions" that consists of repeating the last action performed by an agent with a given probability (ignoring its current choice), and an approach named "frame skipping" that consists of limiting the agent's ability to decide on the behavior of the environment through the repetition of its last action on the next frames. Both are defined by specific parameters of the ALE.

Regarding environment states representation, three main approaches are present in the literature: (i) *color averaging*, (ii) *frame pooling*, and (iii) *frame stacking*. The first two consist of composing a representation of successive frames in a single one to reduce visual issues resulting from the hardware limitation of Atari 2600, and ALE implements the first in a parametric way. In turn, frame stacking consists of concatenating a certain number of frames already observed with the most recent frame, seeking to obtain a more informative observation space for the agent and making it easier to infer trajectories of moving objects in the game environment. It is not implemented directly by ALE and is an algorithm decision but. According to machado18arcade, color averaging and frame pooling may end up removing the most interesting form of partial observation in ALE, and frame stacking reduces the degree of partial observability.

Finally, [4] also pointed out divergences in the literature regarding the processes of finding out the values of hyperparameters. Some research works use all of the games, while others make specific adjustments for each game. To evaluate the agents' generalization capacity, the authors recommended splitting the set of games analogously to supervised learning. Therefore, one should only use a smaller subset of games to adjust the values of the hyperparameters and then use those values to train the agent, without further adjustment, across the entire game set.

References

1. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research **47**, 253–279 (jun 2013)
2. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. arXiv preprint arXiv:1702.08734 (2017)
3. Liang, Y., Machado, M.C., Talvitie, E., Bowling, M.: State of the art control of atari games using shallow reinforcement learning. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS '16). pp. 485–493 (2016)
4. Machado, M.C., Bellemare, M.G., Talvitie, E., Veness, J., Hausknecht, M.J., Bowling, M.: Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. Journal of Artificial Intelligence Research **61**, 523–562 (2018)
5. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (Feb 2015), http://dx.doi.org/10.1038/nature14236