Learning Agents for Pong: A Survey

Davide Pietrasanta 844824, Giuseppe Magazzù 829612, Gaetano Magazzù 829685

Abstract—The aim of this survey is to explore the state of the art of learning agents for the game of Pong, focusing on reinforcement learning algorithms applied to agents.

1 Introduction

This survey wants to explore the state of the art of the learning agents, focusing on the reinforcement learning as a way to learn the game of Pong. The reward-based learning literature includes reinforcement learning and stochastic search methods such as evolutionary algorithms [1].

Reinforcement learning methods can be particularly useful in domains where penalties or rewards are provided after a sequence of actions performed in the environment [1]. Evolutionary computation is a family of techniques that uses mechanisms inspired by biological evolution (evolutionary algorithms) to refine populations of candidate solutions to a given problem [2].

The survey is organized as follows. Section 2 gives an introduction of learning agents. Next different methods to make the agents learn are presented: Markov Decision Process (section 3), reinforcement learning (section 4), deep reinforcement learning (section 5) and evolutionary computation (section 6). Section 7 cover the state of the art of learning agents specific for Pong. Conclusions are given in section 8.

2 Learning Agents

A learning agent is an agent who can operate in unfamiliar environments and can improve its knowledge by learning from experiences.

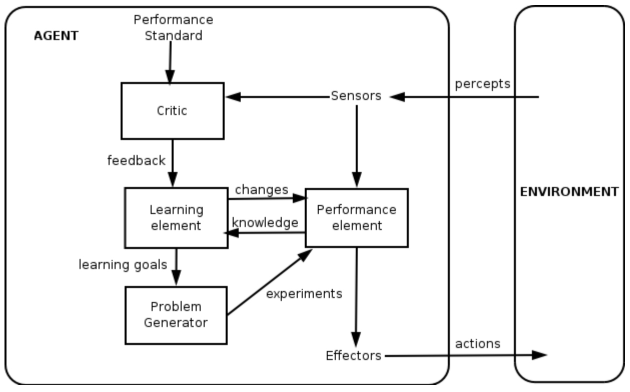

Fig. 1. Learning agent.

A learning agent can be divided into four conceptual elements:

- **Performance element**: Perceives the environment and performs actions.
- **Critic**: Tells the learning element how well the agent is doing with respect to a fixed performance standard [3].
- **Learning element**: Improves the performance element given the feedback from the critic.
- **Problem Generator**: Suggests actions to the agent to explore new experiences.

The performance standard is needed to help establish the quality of agent behavior.

Agent behavior is described by a policy π which decides what action to take in a given state. To evaluate whether the policy is good or not, a utility function U^{π} is used. The policy with the largest utility value is called optimal policy and there are several ways to improve the policy to make the agent learn.

3 Markov Decision Process (MDP)

A sequential decision problem for a fully observable stochastic environment with a Markovian transition model and additive rewards is called a Markov Decision Process or MDP [3]. It consists of a set of states S, a set of actions A(s) for each state, a transition model $P(s'|s, a)$ and a reward function R(s).

Due to the stochastic natures of the environment, the quality of a policy is measured by the expected utility of the possible environmental histories generated by that policy.

This approach is very simple and can lead to several problems. If the space of actions and states is vast, there will be problems in learning. Moreover, if a state has never been seen the next action will be random and this is more likely with large spaces, due to the sparsity of the transition matrix [4]. This method can also bring with it the rewards' sparseness problem, which occurs when many actions take place between one reward and another. This is called Temporal Credit Assignment Problem [5]. However the reward function depends on only the current state, it is therefore difficult to integrate a discounted rewards, which aims to reward/penalize more actions close to victory/defeat, without considering an history and therefore without using Hidden Markov Models [4].

As already mentioned, for unseen states we cannot precisely predict the reward. Q-Learning is an algorithm that can be used to solve MDP problems with unknown reward. It also manages to solve the integration of the discounted rewards easily.

4 Reinforcement Learning

It is believed that reinforcement learning can provide a path towards generally capable agents [6] [7]. It's very useful in today's world indeed it's widely used with self-driving cars, NLP, recommendation systems, financial trades, etc. Reinforcement learning [8] describes a solution to a Markov Decision Process with the goal of finding a policy that maximizes the sum of expected rewards [9]. The agent learns an optimal (or nearly optimal) policy for the environment [3] through a series of reinforcements (rewards or penalties) that provide a quality of its behavior.

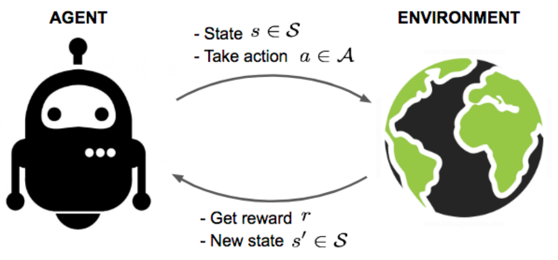
AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Fig. 2. Simple scheme of a reinforcement learning system.

Reinforcement learning is divided in two approaches:

- **Model-based**: The agent uses a transition model of the environment to help interpret the reward signals and to make decisions about how to act [10].
- **Model-free**: The agent learns the consequences of his actions through experience in order to refine his policy.

Reinforcement learning algorithms are divided into several categories:

- **Value-based**: Value-based methods are based on the estimation of a Value-function that estimate the value (expected return) of being in a given state according to a policy π, allowing you to choose the best action in each state.
- **Policy-search**: The policy-search approach describe a parametrized policy, which is often stochastic $\pi_\theta : (a|s) \rightarrow P[a|s]$, and can be seen as an optimization problem where we need to find the best parameters θ to maximize a score function $J(\theta) = E_{\pi_\theta}[\sum \gamma r]$.
- **Actor-critic**: Actor-critic methods use both the value-based and policy-search approaches by combining value functions with an explicit representation of the policy.

4.1 Q-Learning

Q-learning [11] is a model-free reinforcement learning algorithm which estimate a value for each (state, action) pair and with these estimated values compute a policy that can maximize the expected discounted reward.

Given a set of states S, a set of actions A, a policy $\pi : S \rightarrow A$ and a reward function $R : A \times S \rightarrow \mathbb{R}$, the algorithm assigns a quality value to each (state, action) pair using the quality function $Q : S \times A \rightarrow \mathbb{R}$, eq. (1). The quality values are updated iteratively at each agent action a leading to a state s' from a state s.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max_a Q(s',a) - Q(s,a)] \quad (1)$$

where $\alpha \in [0,1]$ is the learning rate and $\gamma \in [0,1]$ is a discount factor which makes futures rewards less valuable than the current ones.

Given infinite exploration time the Q-learning algorithm is able to learn an optimal action-selection policy [12].

A problem of this method is the limitation of the state-action space required, that can be solved with an approximation function. instead of storing each Q-values a mapping function could be learned to map a state-action pair to their respective Q-value.

4.2 SARSA

SARSA algorithm [13] is a variation of the Q-learning algorithm. Its name come from (s, a, r, s', a'), that are (state, action, reward, next state, next action), and they are used to compute the update.

An algorithm has an "Off-Policy" technique if it learns the value function according to the action derived from another policy. On the other hand, it called "On-Policy" if it learns the value function according to the current action derived from the current policy. Q-learning has an Off-Policy technique while SARSA has an On-Policy one.

The update formula of SARSA is similar to the one used by the Q-learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma Q(s',a') - Q(s,a)] \quad (2)$$

This means that SARSA updates the state based on the action taken, while Q-learning updates following the optimal policy. Suppose to be in a "cliff world", where the agent has to walk from the starting cell to the goal cell along the edge of a cliff without falling off. Q-learning, following the optimal policy, would tend to be close to the edge of the cliff while SARSA would prefer a "safer" path.

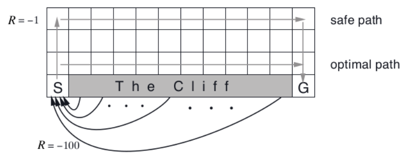

Fig. 3. Image to show the difference between SARSA and Q-learning in a "cliff word".

5 Deep Reinforcement Learning

When an MDP has a large number of states and actions, more memory and more time are required for learning. To solve these problems it is possible to use approximation functions. The function can be linear [14] or non-linear as in the case of neural networks.

Deep reinforcement learning [15] combines reinforcement learning and deep learning and represents policies as a neural network. Deep learning enables RL to scale to decision-making problems that were previously intractable [15]. Instead of a manual engineering representation of the state space, deep RL can use a very large and unstructured input data.

A very big advantage that deep RL offers instead of other input representation, is that it can handle unseen states well [16] [17]. This is because a large portion of pixels could be similar to an image already seen and on which the model has been trained, so the network is likely to produce a similar prediction to the image seen previously. Instead, algorithms like MDP behave randomly in these cases.

Despite the benefits of deep learning, reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function [18]. So new techniques were introduced to solve this instability.

5.1 Deep Q-Network

A Deep Q-Network (DQN) [16] is a CNN adapted for RL used as a function approximator to estimate the Q-values, where the inputs are images and the outputs depends on the number of actions.

A variant of Q-learning is used that combines experience replay and a target network in order to have a more stable target during training.

The approximation of the Q-values is obtained by minimizing the following loss at each iteration i:

$$L(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i - Q(s,a;\theta_i))^2] \tag{3}$$

where $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$ is the target and θ are the network parameters.

Two networks are used: a Q-Network for estimating the Q-values and a target network which is a snapshot of the network at the previous iteration with fixed parameters θ_{i-1}. Experience replay collects transitions (s, a, r, s') to a fixed-size buffer, called replay buffer, and during training compute the loss using a batch of transitions sampled from the buffer. Thanks to these techniques, the DQNs were able to alleviate training instability and achieve remarkable performance.

6 Evolutionary Computation

Evolutionary computation is a family of techniques that uses mechanisms inspired by Darwin evolution theory to refine populations of candidate solutions to a given problem [2]. It uses mechanisms such as reproduction, mutation, recombination and selection.

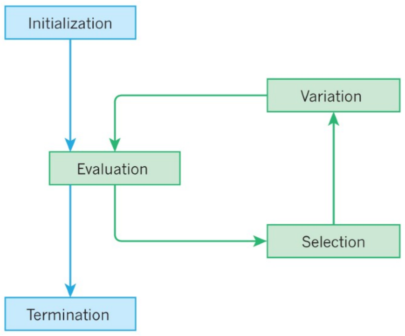

Fig. 4. Evolutionary Algorithm.

An evolutionary algorithm (EA) begins with the initialization of a population of randomly generated individuals. Each member of this population is then evaluated. The EA then uses a fitness-oriented procedure to select, breed, and mutate individuals to produce children which are then added to the population, replacing older individuals. One evaluation, selection, and variation (also known as breeding or reproduction) cycle is known as a generation. Successive generations continue to refine the population until time is exhausted or a sufficiently fit individual is discovered. Evolutionary computation methods include genetic algorithms (GA) and evolution strategies (ES), both of which are usually applied to the search of multidimensional parameter spaces; and genetic programming (GP) [1] [2].

Evolutionary algorithms can be used from learning agent to learn Solo Pong [19] and Pong [20]. It's also possible to designing neural networks through the use of NeuroEvolution [21], an evolutionary algorithm that optimize neural networks.

7 Learning Pong

The most widely used approach to implementing a learning agent that plays Pong is deep reinforcement learning. Since the introduction of Deep Q-Networks (DQNs) in 2013, standard methods like Q-learning have been used less and less as they cannot easily handle large inputs.

One of the first works using Q-learning is [22] in 2002, which describe a way to apply the Q-learning algorithm to the game of Pong. Moreover, it wants to use the moves made by experts as observations in order to speed up the learning process. The pong board is a 10x12 rectangle and the agent controls a paddle of width 2, which can move one position to the right or left. An integral horizontal velocity ranging from -2 to +2 units is added every time step. Hitting the ball yields a +1 reward; missing it yields a -1 penalty.

Mnih, Volodymyr, et al. [16] in 2013 introduce the concept of DQNs, a convolutional neural network, trained with a variant of Q-learning. Seven popular ATARI games are considered in this work, including Pong. Grayscale game frames are captured at 210x160 resolution, downsampled to a 110x84 image and cropped to an 84x84 patch. The network has an input shape of 84x84xk, where k is the number of skipped frame and has a separate output unit for each possible action, the number of which depends on the game considered. For all games considered all positive rewards are fixed at 1 and all negative rewards at -1. The network is trained with an ϵ-greedy annealed strategy and a replay memory of most recent frames is used to ease the training.

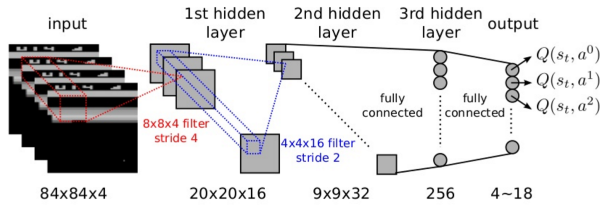

Fig. 5. Deep Q-Network architecture.

Another work that uses DQNs focusing on Pong is [23] in 2017, in which screenshots are taken at 32 FPS binarized and rescaled to 80x80, and the reward considered is the same as in [16]. The agent actions considered are: move paddle up, move paddle down and paddle stays at the same place. This work implement an episodic control technique [24] to reuse successful strategies instead of randomly selected transition from the replay buffer.

In 2017 Diallo et al. [25] discuss the emergence of cooperative and coordinated between joint and concurrent learning agents using deep Q-learning. As in [16] in the case of Pong, the images are resized to 84x84 and a frame skip of 4 is used, and the actions considered are the same as in [23]. In the considered scenario, two agents form a team to win against an hard-coded AI, and learn how to cooperate and how to divide the field. The reward scheme for the two agents is specified in (table 1).

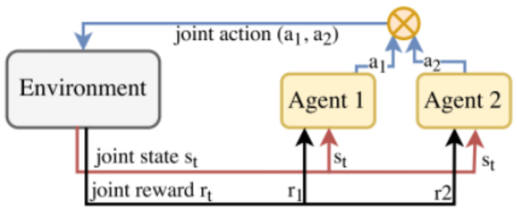

Fig. 6. Multi-agent concurrent DQN [25].

TABLE 1
Reward scheme adopted by [25].

Events	Agent 1 reward	Agent 2 reward
Left player loses	+1	+1
One agent loses	-1	-1
Collision	-1	-1

Unlike previous works where the agent learns to play Pong against a hard-coded AI, Tampuu, Ardi, et al. [26] in 2017 consider both as learning agents and shows that different rewarding schemes lead them towards competition or collaboration. Multiple agents controlled by autonomous DQNs learn to cooperate and compete while sharing a high-dimensional environment and being fed only raw visual input [26]. Each of the two agents can take four actions: move up, move down, stand still, and fire (to relaunch the ball or to start the game). Three rewarding schemes are studied (table 2). The results show that DQNs can become a practical tool for the decentralized learning of multiagent systems living a complex environments [26].

TABLE 2
Rewarding schemes to explore the transition from competitive to the cooperative strategy [26]. Fully competitive $\rho = 1$, fully cooperative $\rho = -1$, intermediate $\rho = range(-1, 1, 0.25)$.

	Left player scores	Right player scores
Left player scores	ρ	-1
Right player scores	-1	ρ

In 2020 McBrien et al. [20] propose a different approach which uses a neural network automatically generated with genetic algorithms to allow agents to learn how to play Pong. A custom version of Pong is considered where the goals are smaller and the players have the ability to move in both the x and y dimensions. The state is represented as eight inputs: the paddle's X and Y position, the opponent's X and Y position, the ball's X and Y velocity, and the ball's X and Y position. The output of the neural network has four output nodes for each possible action: up, down, left and right. Two algorithms are used: NeuroEvolution (NE) and NeuroEvolution of Augmenting Topologies (NEAT). The first start from a predefined neural net structure and iteratively tune its parameters, while the second adds the possibility to modify the network topology by adding nodes and connections randomly.

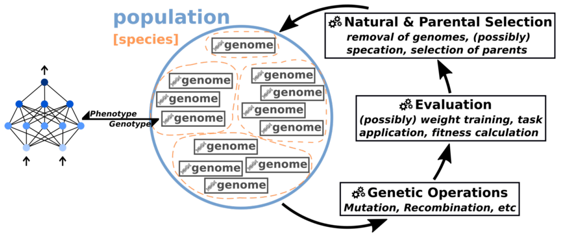
population
[species]
genome
Phenotype
Genotype
Natural & Parental Selection
removal of genomes, (possibly) specation, selection of parents
Evaluation
(possibly) weight training, task application, fitness calculation
Genetic Operations
Mutation, Recombination, etc

Fig. 7. NeuroEvolution algorithm schema.

8 Conclusion

In this survey, different methods of reward-based learning were covered, focusing more on reinforcement learning. Then the state of the art of learning agents for Pong was described.

The models that work best for Pong are those that exploit neural networks and are therefore part of the deep reinforcement learning family. DQNs have provided important help in learning an agent how to play games.

Most of the state of the art has studied the learning of a single agent. Recently some works have explored collaboration and competition behaviors between learning agents and have shown that they can improve results.

There are still many open problems in reinforcement learning. Training these models takes a lot of time and many episodes (millions of frames) are needed to learn a good policy. Exploration-exploitation dilemma is always present in any learning problem. A problem that does not occur with Pong, but which is very problematic for other types of games is the Temporal Credit Assignment Problem. As already mentioned, it refers to the fact that rewards can occur terribly temporally delayed. To solve the problem, the only solution that seems to give results is to perform many steps of the reinforcement-learning algorithm to propagate the influence of delayed reinforcement to all states and actions that have an effect on that reinforcement [5].

It is still difficult to automate the reward structure and often need to be manually defined. Reward design is an important factor that decides the robustness of an RL system.

RL algorithms have also been shown to be incredibly sensitive to hyperparameters and architectures of deep neural networks. AutoRL is a new branch born with the intention of solving this problems [7].

REFERENCES

[1] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, 2005.

[2] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International conference on global trends in signal processing, information computing and communication (ICGT-SPICC)*. IEEE, 2016, pp. 261–265.

[3] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," *Artificial Intelligence*, 2002.

[4] D. Silver, "Lectures on reinforcement learning," https://www.davidsilver.uk/teaching/, 2015.

[5] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, University of Massachusetts Amherst, 1984.

[6] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021.

[7] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust *et al.*, "Automated reinforcement learning (autorl): A survey and open problems," *arXiv preprint arXiv:2201.03916*, 2022.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Q. J. Huys, A. Cruickshank, and P. Seriès, "Reward-based learning, model-based and model-free," in *Encyclopedia of Computational Neuroscience*. Springer New York, 2014, pp. 1–10.

[10] S. Russell and P. Norvig, "Artificial intelligence: a modern approach, global edition 4th," *Foundations*, vol. 19, p. 23, 2021.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[12] F. S. Melo, "Convergence of q-learning: A simple proof," *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[13] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*. IEEE, 2011, pp. 1143–1146.

[14] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 664–671.

[15] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[16] 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.

[17] A. Karpathy, "Deep reinforcement learning: Pong from pixels," *url: http://karpathy. github. io/2016/05/31/rl*, 2016.

[18] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] W. B. Langdon and R. Poli, "Evolutionary solo pong players," in *2005 IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 2005, pp. 2621–2628.

[20] M. McBrien, A. Melehan, and M. Munns, "Learning pong," *Nan*, 2020.

[21] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[22] D. Dahlstrom, E. Wiewiora, G. Cottrell, and C. Elkan, "Imitative policies for reinforcement learning," Technical Report, Department of Computer Science and Engineering, University . . . , Tech. Rep., 2002.

[23] I. Makarov, A. Kashin, and A. Korinevskaya, "Learning to play pong video game via deep reinforcement learning." in *AIST (Supplement)*, 2017, pp. 236–241.

[24] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis, "Model-free episodic control," *arXiv preprint arXiv:1606.04460*, 2016.

[25] E. A. O. Diallo, A. Sugiyama, and T. Sugawara, "Learning to coordinate with deep reinforcement learning in doubles pong game," in *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 2017, pp. 14–19.

[26] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.