Atari Breakout with LTL$_f$/LDL$_f$ Goals

Ivan Bergonzani, Michele Cipriano, Armando Nania

Professor: Giuseppe De Giacomo

Elective in Artificial Intelligence: Reasoning Robots
Department of Computer, Control and Management Engineering
Sapienza University of Rome

The main goal of the project is to train an agent acting on a non-Markovian reward decision process (NMRDP) environment.

Rewards are define through LTL_f/LDL_f formulas, defining complex behaviour made up of multiple steps over time.

The learning is done using classical reinforcement learning algorithm working on MDP environments.

Q-Learning

Q-Learning is a temporal difference (TD) reinforcement learning algorithm.

The Q-function is approximated through a Q-table that stores the expected rewards for taking action *A* in state *S*.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

Off-policy algorithm: action at time $t + 1$ independent from current policy.

Converges to optimal policy if all state-action pairs are continuously updated.

SARSA is a TD reinforcement learning algorithm which uses a quintuple $\langle S_t, A_t, R_t, S_{t+1}, A_{t+1} \rangle$ from which takes name.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

On-policy algorithm: action A_{t+1} chosen following current policy.

Converge to optimal policy if each state-action pair is visited infinitely often.

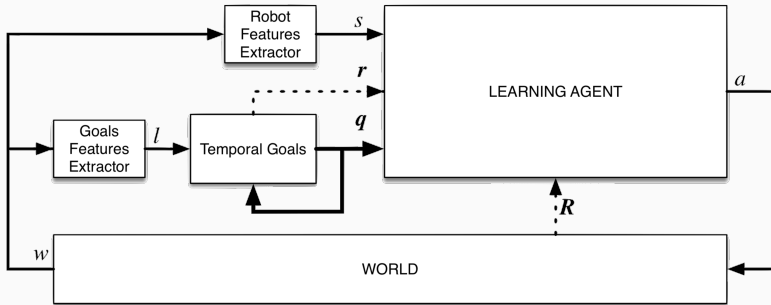LTL$_f$/LDL$_f$ non-Markovian rewards + integration in our project.

Figure 1: Pipeline describing how the agent is interacting with the world and how the robot features extractor and the goal features extractor are used in order to handle non-Markovian rewards.

Reward for destroying the block rows in order from top to bottom.

$$\langle(\neg\varphi_0 \wedge \neg\varphi_1 \wedge \neg\varphi_2)^*; (\varphi_0 \wedge \neg\varphi_1 \wedge \neg\varphi_2); (\varphi_0 \wedge \neg\varphi_1 \wedge \neg\varphi_2)^*;$$
$$(\varphi_0 \wedge \varphi_1 \wedge \neg\varphi_2); (\varphi_0 \wedge \varphi_1 \wedge \neg\varphi_2)*; (\varphi_0 \wedge \varphi_1 \wedge \varphi_2)\rangle tt$$

where φ_i indicates the *i*-th row.

$(\neg\varphi_0 \wedge \neg\varphi_1 \wedge \neg\varphi_2)^*$ all rows not destructed for an indefinite amount of tin

$(\varphi_0 \wedge \neg\varphi_1 \wedge \neg\varphi_2)$ first row destructed, other still there.

LTL$_f$/LDL$_f$ transformed to an equivalent automaton.

Automaton representation given in input to the learning agent.

Learning agent input: ⟨environment state, automaton state⟩

PyGame Breakout

Starting from the work presented in [4] the agents were trained on a modified PyGame Breakout with a 6×18 grid.

Game state already stored inside the class instance (ball position, paddle position, blocks status).

Atari Breakout

Breakout from OpenAI Gym: toolkit for comparing learning algorithm. Provides easy access to state and reward. No assumption on the agent.

Class stores pixels status. Need to extract information about the game state from it.

The environment is more stochastic.
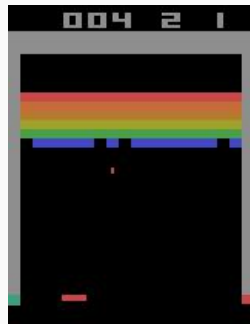
Implementation

Atari Wrappers

Atari wrappers from OpenAI baselines:

- `EpisodicLifeEnv`: make an "end of life" be the end of the episode resetting the environment only on the true game over, this helps in value estimation
- `FireResetEnv`: use "Fire" as starting action in order to launch the ball
- `MaxAndSkipEnv`: returns only the skip-th frame, this reduces the amount of frame the agent has to deal with (set to 4 in the main experiments)

Robot Features Extraction

- extracting **paddle position**
- extracting **ball position**
- features can be increased with **ball direction**
- **previous positions** could help the training too
- more features increase the state space
- OpenCV

Extracting **paddle position** from the observation: tensor of dimension $(210, 160, 3)$ where each value represents a color of the pixel.

Considering only the **lower part** of the observation which contains the paddle.

Converting the image to **gray-scale** in order to reduce the number of channels.
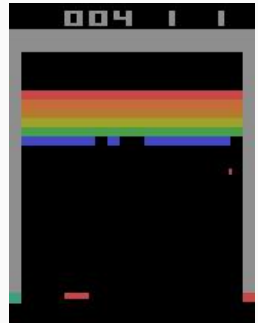This makes it easier to apply a threshold function.

Determine the position of the paddle:

- apply a **threshold** function to obtain a black and white image
- find **contours**
- extract **centroid** of the paddle
- $paddle_x = 35$

Similarly, extract **ball position** from initial observation.

Considering only the **upper part** of the observation which contains the ball.

Robot Features Extraction

- `inRange` function to consider only pixels with the same color of the ball, obtain a black and white image
- remove the **remaining bricks** by checking sorrounding pixels, width of the ball is much smaller

Determine the position of the ball:

- find **contours**
- extract **centroid** of the ball
- $\text{ball}_x = 135, \text{ball}_y = 77$

Robot Features Extraction

Return the difference between the position of the paddle and the ball on the *x* axis and the position of the ball on the *y* axis:

$$(\text{ball}_x - \text{paddle}_x + 143, \text{ball}_y) = (243, 77)$$

Extract a binary representation of the bricks from the observation:

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0
```

It will be used to evaluate fluents in temporal goals.
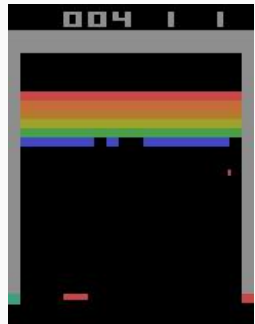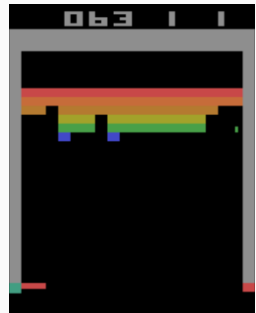
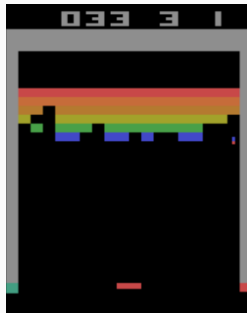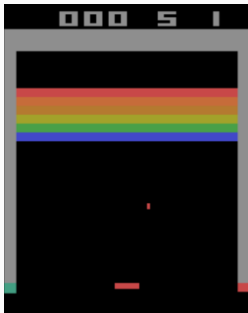- string formula to break rows from top to bottom:

```
<(!l0 & !l1 & ... & !l5)*;
 ( l0 & !l1 & ... & !l5);
              ...
 ( l0 &  l1 & ... & !l5)*;
 ( l0 &  l1 & ... &  l5)>tt
```

- parsed with `LDLfParser` (FLLOAT)
- `LDLfFormula` passed to abstract class `TemporalEvaluator`
- automaton is being created (RLTG, DFA built upon `Pythomata`)
  in order to create the extended MDP

Experiments

Experiments

- Example of gameplay in the Atari Environment
- This gameplay is the one that gave the best results, with a score of 63
- It's worth noting that the ball changes color when it's inside the brick area

- Example of gameplay in the PyGame Environment
- This gameplay gave a score of 101
- In this case the ball does not change color, which is a further simplification compared to the Atari environment
- The window width/height ratio is much greater, but the ball is much slower

Results

Environment	Epochs	Explored States	Total Reward
Atari	100,000	26,291	46
Atari	200,000	26,271	63
PyGame	50,000	91,821	1010

- Comparison of the results obtained
- All the tests shown above were performed with the SARSA algorithm
- It is important to note that the reward is assigned differently in the two environments
- PyGame returns a reward of 10 for the destruction of any brick
- Atari returns a reward equal to 1, 4 or 7 for the destruction of a brick
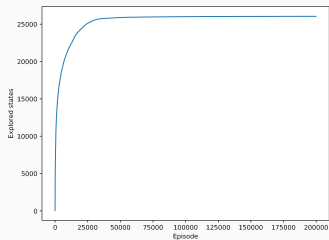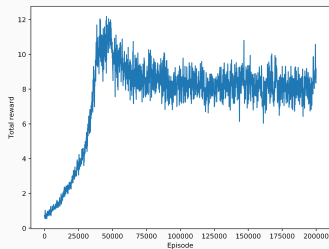
Atari Environment results
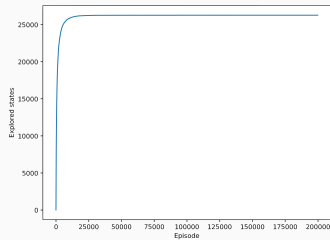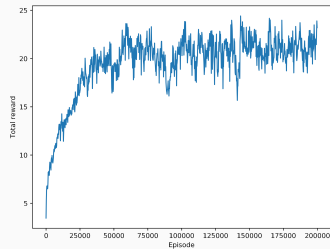

Figure 2: Q-Learning algorithm

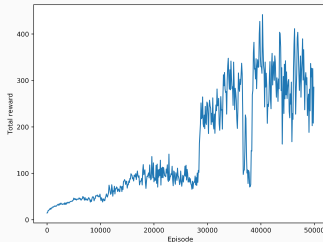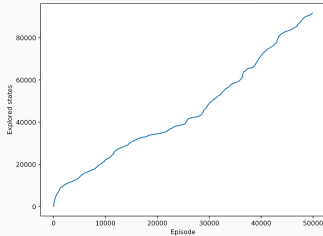**Figure 3:** SARSA algorithm

- The Total Reward in the PyGame Environment is much higher
- In the Atari Environment, the number of states explored has stabilized after 25,000 epochs
- In the Pygame Environment, the number of states explored shows an increasing linear trend along the entire sample of 50,000 epochs
- The number of states explored in PyGame exceeds 90,000, a sign that the game continues to move forward, exploring more and more states

The Modified PyGame

- We tried to investigate what was going on
- The paddle-bricks distance is covered in 35 frames in PyGame, only 15 in Atari
- In 1500 frames, the agent manages to destroy 10 bricks in PyGame, 29 in Atari
- In the Atari Environment there are 5 lives, but only 1 in PyGame
- Given alle the differences, we changed the two environments to make them as similar as possible, to compare them better
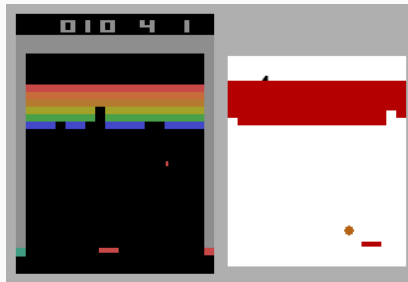
The pygame environment thus modified was able to learn the destruction of an entire line already from 5,000 epochs.

The Atari environment, on the other hand, could not achieve this goal even after 200,000 eras.

But these tests produced interesting results

Tests

Ball X	Ball Y	Angle	Paddle	Move	Lives Count	Frame Skip	Score
144	157	10	144	49	1	4	6
144	157	10	144	49	5	4	4-20
8	18	10	8	49	1	4	5
144	157	10	144	10	1	4	7
144	157	10	144	0	1	4	7
144	157	10	144	0	5	4	2-16
8	18	10	8	10	1	4	6
8	18	10	8	0	1	4	2
8	18	10	8	0	5	4	5-15
144	157	10	144	13	5	14	19
144	157	10	144	13	5	10	24
144	157	10	144	13[1]	5	10	16
144	157	10	144	121	5	10	24

Table 1: [1] This test was performed with the same hyperparameters as the previous test, but with a different calculation of the move variable: instead of distributing the 13 states of the move variable from -60 to +60 motion pixels, they were distributed over the range [-40 , +40], cropping excess values.

The Dimension Space Problem

- Many tests have been carried out to reduce the space of states in Atari
- Many tests have been carried out to reduce the space of states in Atari
- Some even showed a worsening effect
- The algorithm fails to learn from the states it has visited
- There is therefore a deep problem in the Atari Environment

The Atari Breakout Difficulties II

The Local Maximum Problem

- There are many states from which it is difficult to escape
- If the paddle is in the corner it will be very difficult to get out of it
- Increasing the number of frameskip alleviates this problem, but increases the problem of non-determinism
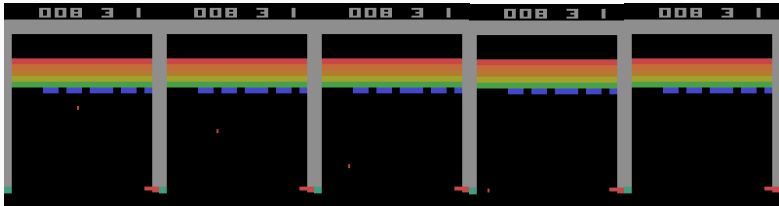

Figure 4: Sequence of sampled frames

The Non-Determinism Problem

- The real problem that has constituted the main obstacle is the non-determinism of the paddle
- The paddle movement consists of two parts: a random part and an inertial part
- A variable called move in the statespace was added to alleviate the inertial part
- The random part can not be resolved, and is the cause of the last problem:

The Atari Breakout Difficulties IV

The State Collision Problem

- Because of non-determinism, some different situations share the same states
- This means that when two similar situations happen in the same game, conflicts can occur
- The best action in the first of two states may not be the best action in the second state
- When the agent learns the best action of the second state, he overwrites the first state, losing all the progress made up to that point
- This makes training for reinforcement learning on the Atari Environment extremely difficult

- The presented project is an extension of the previous project on the Atari Environment
- The best performance did not manage to correctly solve the Atari Breakout game
- Fundamental problems have been identified, in particular non-determinism, in the environment
- Thoose problems prevent algorithms such as Q-Learning or SARSA from achiving satisfactory results
- Future work must be done to solve such problems, especially the introduction of a neural network

Q&A

References I

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*.
The MIT Press, second ed., 2018.

G. De Giacomo, L. Iocchi, M. Favorito, and F. Patrizi, "Reinforcement Learning for LTLf/LDLf Goals," *CoRR*, vol. abs/1807.06333, 2018.

📄 V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

📄 "Montezuma's Revenge Solved by Go-Explore, a New Algorithm for Hard-Exploration Problems (Sets Records on Pitfall, Too)." https://eng.uber.com/go-explore/.

References III

📄 S. A. McIlraith and K. Q. Weinberger, eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, AAAI Press, 2018.

📄 "A Python Implementation of the FLLOAT library."
https://github.com/MarcoFavorito/flloat.

📄 "A library for generating automata from LTL and LDL formulas with finite-trace semantics in Python."
https://github.com/MarcoFavorito/pythomata.

📄 "Framework for Reinforcement Learning with Temporal Goals defined by LDLf formulas.."
https://github.com/MarcoFavorito/rltg.

G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "OpenAI Baselines." https://github.com/openai/baselines, 2017.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.