



SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER, CONTROL AND
MANAGEMENT ENGINEERING

Atari Breakout with LTL_f/LDL_f Goals

ELECTIVE IN ARTIFICIAL INTELLIGENCE:
REASONING ROBOTS

Professor:
Giuseppe De Giacomo

Students:
Ivan Bergonzani
Michele Cipriano
Armando Nania

Contents

1	Introduction	2
2	Reinforcement Learning	3
2.1	Q-Learning	3
2.2	SARSA	4
3	LTL_f/LDL_f Non-Markovian Rewards	5
3.1	Theoretical Background	5
3.2	Examples	5
4	OpenAI Gym	6
4.1	Framework	6
4.2	Examples	6
5	Atari Breakout	8
5.1	PyGame Breakout	8
5.2	Arcade Learning Environment	8
5.3	Implementation	8
5.3.1	Robot Features Extractor	8
5.3.2	Goal Features Extractor	10
5.3.3	Temporal Goals	10
5.4	Experiments	12
6	Conclusion	13

1 Introduction

Introduction to the whole project, structure of the report and summary of the work.

2 Reinforcement Learning

Reinforcement learning [1] is an area of machine learning which aims at studying how to develop agents that can interact with their environment maximizing a cumulative reward. The environment can be formally defined as a Markov Decision Process (MDP), which is a tuple $\langle S, A, \delta, R \rangle$ where S is a finite state of states that can represent the environment, A is a finite state of actions that can be performed by the agent in the environment, δ is a probability function modeling the transition from a state to another when performing a certain action and R is a reward function which models the reward received by the environment when performing a certain action which makes the agent move from a state to another.

An interesting property of the MDP is that it satisfies the Markov property, hence, the future states that will be reached by the agent do not depend on the past interaction of the environment, but just on the current state. This makes it possible to define the transition and the reward function depending only on the current state (and of course the action and the future state of interest).

This section considers two common reinforcement learning algorithms, namely Q-Learning and SARSA, which have been used in our experiments in order to train an agent interacting with an Atari Breakout environment (section 5.4).

2.1 Q-Learning

Q-Learning algorithm.

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

Algorithm 1: Q-Learning algorithm Python implementation.

```
1 class QLearning(TDBrain):
2     def __init__(self, observation_space:Discrete,
3         ↪ action_space, policy:Policy=EGreedy(),
4             gamma=0.99, alpha=None, lambda_=0):
5         super().__init__(observation_space, action_space,
6             ↪ policy, gamma, alpha, lambda_)
7
8     def update_Q(self, obs:AgentObservation):
9         state, action, reward, state2 = obs.unpack()
10
11         action2 = self.choose_action(state2)
12         Qa = np.max(self.Q[state2])
13         actions_star = np.argmax(self.Q[state2] == Qa).
14             ↪ flatten().tolist()
```

```

13         delta = reward + self.gamma * Qa - self.Q[state][
14             ↪ action]
14         for (s, a) in set(self.eligibility.traces.keys()):
15             self.Q[s][a] += self.alpha.get(s,a) * delta *
16                 ↪ self.eligibility.get(s, a)
16             if action2 in actions_star:
17                 self.eligibility.update(s, a)
18             else:
19                 self.eligibility.to_zero(s, a)
20
21         return action2

```

2.2 SARSA

SARSA algorithm.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (2)$$

Algorithm 2: SARSA algorithm Python implementation.

```

1 class Sarsa(TDBrain):
2     def __init__(self, observation_space:Discrete,
3         ↪ action_space, policy:Policy=EGreedy(),
4             gamma=0.99, alpha=None, lambda_=0.0):
5         super().__init__(observation_space, action_space,
6             ↪ policy, gamma, alpha, lambda_)
7
8     def update_Q(self, obs:AgentObservation):
9         state, action, reward, state2 = obs.unpack()
10
11         action2 = self.choose_action(state2)
12         Qa = self.Q[state2][action2]
13
14         delta = reward + self.gamma * Qa - self.Q[state][
15             ↪ action]
16         for (s, a) in set(self.eligibility.traces.keys()):
17             self.Q[s][a] += self.alpha.get(s,a) * delta *
18                 ↪ self.eligibility.get(s, a)
19             self.eligibility.update(s, a)
20
21         return action2

```

3 LTL_f/ LDL_f Non-Markovian Rewards

Intro.

3.1 Theoretical Background

Introduction to the research paper.

3.2 Examples

How it can be used to train a RL model.

4 OpenAI Gym

OpenAI gym [2] is a toolkit for developing and comparing reinforcement learning algorithms, without making assumptions about the structure of the agent interacting with the environment, in order to keep development flexible to updates on both sides.

4.1 Framework

The framework of `gym` allows to interact easily with an environment, giving the developers the tools they need to perform actions and to observe the state of the environment itself. In this way it is possible to focus more on the development of the agent without spending time on the structure of the world.

`gym` makes it possible to interact with multiple kinds of environments. Among these, the authors of the framework developed the support for Arcade Learning Environment [3], which includes all the classic Atari games, including Breakout, which has been used in this project.

4.2 Examples

Let's consider a simple example to understand how `gym` works and how the framework can be used to interact with an environment. The description will follow Algorithm 3.

Algorithm 3: Example of a random interaction with the `gym` environment `BreakoutNoFrameskip-v4`, used also in our experiments of subsection 5.4.

```
1 import gym
2
3 env = gym.make("BreakoutNoFrameskip-v4")
4 env.reset()
5
6 for _ in range(1000):
7     env.render()
8     action = env.action_space.sample() # takes random actions
9     observation, reward, done, info = env.step(action)
10    if done == True:
11        env.reset()
12
13 env.close()
```

Initially (line 1) the framework is imported. Then (line 3-4) an environment is created specifying its name and initializing it. The program makes a random agent interact randomly with the environment for 1000 episodes (lines 6-11) before closing the environment. Line 7 renders the current observation of the

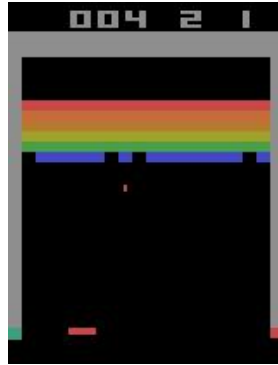


Figure 1: Observation of a frame of the environment `BreakoutNoFrameskip-v4`.

environment on screen, line 8-9 performs a random action between those available in this Brekout version, note that the method `step` return an `observation` (shown in Fig. 1), which is an array of pixels that represent the current state of the environment, a `reward`, which is a value return by the game after performing the specified action `action`, a boolean value `done`, which is `True` is the game is over, `False` otherwise, and `info` which contains extra information about the game. Lines 10-11 handles the case when the game is over, resetting the environment.

5 Atari Breakout

Intro.

5.1 PyGame Breakout

Original implementation of the paper (non-ATARI).

5.2 Arcade Learning Environment

ATARI Breakout (from ALE) and differences from the other one.

5.3 Implementation

Intro to implementation.

5.3.1 Robot Features Extractor

`RobotFeatureExtractor` (OpenCV). Extracts features of the robot (robot and ball positions).

Algorithm 4: Robot feature extractor Python implementation.

```
1 class BreakoutNRobotFeatureExtractor(  
    ↪ BreakoutRobotFeatureExtractor):  
2  
3     def __init__(self, obs_space):  
4         robot_feature_space = Tuple((  
5             Discrete(287),  
6             Discrete(157),  
7         ))  
8  
9         self.prev_ballX = 0  
10        self.prev_ballY = 0  
11        self.prev_paddleX = 0  
12        self.still_image = True  
13  
14        super().__init__(obs_space, robot_feature_space)  
15  
16        def _extract(self, input, **kwargs):  
17            self.still_image = not self.still_image  
18            if self.still_image:  
19                return (self.prev_ballX-self.prev_paddleX+143,  
20                    ↪ self.prev_ballY)  
21            # Extract position of the paddle:  
22            paddle_img = input[189:193,8:152,:]  
23            gray = cv2.cvtColor(paddle_img, cv2.COLOR_RGB2GRAY)  
            thresh = cv2.threshold(gray, 60, 255, cv2.  
                ↪ THRESH_BINARY)[1]
```

```

24     cnts = cv2.findContours(thresh.copy(), cv2.
    ↪ RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
25     cnts = cnts[0] if imutils.is_cv2() else cnts[1]
26     min_distance = np.inf
27     paddleX = self.prev_paddleX
28     for c in cnts:
29         M = cv2.moments(c)
30         if M["m00"] == 0:
31             continue
32         pX = int(M["m10"] / M["m00"])
33         if abs(self.prev_paddleX - pX) < min_distance:
34             min_distance = abs(self.prev_paddleX - pX)
35             paddleX = pX
36
37     # Extract position of the ball:
38     ballX = self.prev_ballX
39     ballY = self.prev_ballY
40     ballspace_img = input[32:189,8:152,:]
41     lower = np.array([200, 72, 72], dtype=np.uint8)
42     upper = np.array([200, 72, 72], dtype=np.uint8)
43     mask = cv2.inRange(ballspace_img, lower, upper)
44     cnts = cv2.findContours(mask.copy(), cv2.
    ↪ RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
45     cnts = cnts[0] if imutils.is_cv2() else cnts[1]
46     for c in cnts:
47         M = cv2.moments(c)
48         # Avoid to compute position of the ball if M["
    ↪ m00"] is zero:
49         if M["m00"] == 0:
50             continue
51         # Calculate the centroid
52         cX = int(M["m10"] / M["m00"])
53         cY = int(M["m01"] / M["m00"])
54         # Check that the centroid is actually part of
    ↪ the ball:
55         left_black = False
56         right_black = False
57         if cX > 3:
58             if ballspace_img[cY][cX-3][0] != 200 or \
59                 ballspace_img[cY][cX-3][1] != 72 or \
60                 ballspace_img[cY][cX-3][2] != 72:
61                 left_black = True
62         else:
63             if ballspace_img[cY][cX+3][0] != 200 or \
64                 ballspace_img[cY][cX+3][1] != 72 or \
65                 ballspace_img[cY][cX+3][2] != 72:
66                 right_black = True
67         if left_black or right_black:
68             ballX = cX
69             ballY = cY
70
71     self.prev_ballX = ballX
72     self.prev_ballY = ballY
73     self.prev_paddleX = paddleX

```

```

74 |
75 |         return (self.prev_ballX - self.prev_paddleX + 143,
               ↪ self.prev_ballY)

```

5.3.2 Goal Features Extractor

GoalFeatureExtractor (OpenCV). Extracts 6x18 table representation of the bricks in order to evaluate a formula.

Algorithm 5: Goal feature extractor Python implementation.

```

1 | class BreakoutGoalFeatureExtractor(FeatureExtractor):
2 |     def __init__(self, obs_space, bricks_rows=6,
               ↪ bricks_cols=18):
3 |         self.bricks_rows = bricks_rows
4 |         self.bricks_cols = bricks_cols
5 |         output_space = Box(low=0, high=1, shape=(
               ↪ bricks_cols, bricks_rows), dtype=np.uint8)
6 |         super().__init__(obs_space, output_space)
7 |
8 |     def _extract(self, input, **kwargs):
9 |         bricks_features = np.ones((self.bricks_cols, self.
               ↪ bricks_rows))
10 |         for row, col in itertools.product(range(self.
               ↪ bricks_rows), range(self.bricks_cols)):
11 |             # Pixel of the observation to check:
12 |             px_upper_left = int( 8 + 8 * col)
13 |             py_upper_left = int(57 + 6 * row)
14 |             px_upper_right = int(15 + 8 * col)
15 |             py_upper_right = int(57 + 6 * row)
16 |
17 |             # Checking max because the input has 3 channels
               ↪ :
18 |             if max(input[py_upper_left][px_upper_left]) ==
               ↪ 0 or \
19 |                 max(input[py_upper_right][px_upper_right])
               ↪ == 0:
20 |                 bricks_features[col][row] = 0
21 |
22 |         return bricks_features

```

*Ext used to improve implementation.

5.3.3 Temporal Goals

LTL_f/LDL_f implementation (with Marco Favorito libraries).

Algorithm 6: LTL_f/LDL_f formulas Python implementation.

```

1 | def get_breakout_lines_formula(lines_symbols):
2 |     # Generate the formula string
3 |     # E.g. for 3 line symbols:

```

```

4      # "<(!10 & !11 & !12)*;(10 & !11 & !12);(10 & !11 & !12
      ↪ )*(10 & 11 & !12); (10 & 11 & !12)*; 10 & 11 &
      ↪ 12>tt"
5      pos = list(map(str, lines_symbols))
6      neg = list(map(lambda x: "!" + str(x), lines_symbols))
7
8      s = "(%s)*" % " & ".join(neg)
9      for idx in range(len(lines_symbols)-1):
10         step = " & ".join(pos[:idx + 1]) + " & " + " & ".
            ↪ join(neg[idx + 1:])
11         s += ";({0});({0})".format(step)
12     s += ";(%s)" % " & ".join(pos)
13     s = "<%s>tt" % s
14
15     return s
16
17 class BreakoutCompleteLinesTemporalEvaluator(
    ↪ TemporalEvaluator):
18     """Breakout temporal evaluator for delete columns from
    ↪ left to right"""
19
20     def __init__(self, input_space, bricks_cols=3,
    ↪ bricks_rows=3, lines_num=3, gamma=0.99,
    ↪ on_the_fly=False):
21         assert lines_num == bricks_cols or lines_num ==
            ↪ bricks_rows
22         self.line_symbols = [Symbol("l%s" % i) for i in
            ↪ range(lines_num)]
23         lines = self.line_symbols
24
25         parser = LDLfParser()
26
27
28         string_formula = get_breakout_lines_formula(lines)
29         print(string_formula)
30         f = parser(string_formula)
31         reward = 10000
32
33         super().__init__(BreakoutGoalFeatureExtractor(
            ↪ input_space, bricks_cols=bricks_cols,
            ↪ bricks_rows=bricks_rows),
                set(lines),
                f,
                reward,
                gamma=gamma,
                on_the_fly=on_the_fly)
34
35
36
37
38
39
40     @abstractmethod
41     def fromFeaturesToPropositional(self, features, action,
    ↪ *args, **kwargs):
42         """map the matrix bricks status to a propositional
    ↪ formula
43         first dimension: columns
44         second dimension: row

```

```

45         """
46         matrix = features
47         lines_status = np.all(matrix == 0.0, axis=kwards["
48             ↪ axis"])
49         result = set()
49         sorted_symbols = reversed(self.line_symbols) if
50             ↪ kwards["is_reversed"] else self.line_symbols
50         for rs, sym in zip(lines_status, sorted_symbols):
51             if rs:
52                 result.add(sym)
53
54         return frozenset(result)
55
56 class BreakoutCompleteRowsTemporalEvaluator(
57     ↪ BreakoutCompleteLinesTemporalEvaluator):
57     """Temporal evaluator for complete rows in order"""
58
59     def __init__(self, input_space, bricks_cols=3,
60         ↪ bricks_rows=3, bottom_up=True, gamma=0.99,
61         ↪ on_the_fly=False):
60         super().__init__(input_space, bricks_cols=
62             ↪ bricks_cols, bricks_rows=bricks_rows,
63             ↪ lines_num=bricks_rows, gamma=gamma,
64             ↪ on_the_fly=on_the_fly)
61         self.bottom_up = bottom_up
62
63     def fromFeaturesToPropositional(self, features, action,
64         ↪ *args, **kwargs):
64         """complete rows from bottom-to-up or top-to-down,
65             ↪ depending on self.bottom_up"""
65         return super().fromFeaturesToPropositional(features
66             ↪ , action, axis=0, is_reversed=self.bottom_up
67             ↪ )

```

Atari wrappers (OpenAI).

5.4 Experiments

Results with 6x18 non-ATARI Breakout (+CODE).

Results with our experiments (+CODE).

6 Conclusion

Why it does not work.

Summary + differences between the two environments.

Future works (neural networks and parallel computation).

References

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.
- [2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” 2016.
- [3] 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.