Department of Computer, Control and Management Engineering

Atari Breakout with $\mathbf{LTL}_f/\mathbf{LDL}_f$ Goals

Elective in Artificial Intelligence: Reasoning Robots

Students:
Ivan Bergonzani
Michele Cipriano
Armando Nania

Professor:
Giuseppe De Giacomo

Academic Year 2017/2018

Contents

1 Introduction

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

2 Reinforcement Learning

Introduction to RL.

2.1 Q-Learning

Q-Learning algorithm.

Algorithm 1: Q-Learning algorithm Python implementation.

```python
class QLearning(TDBrain):
    def __init__(self, observation_space:Discrete, action_space, policy:Policy=EGreedy(),
                 gamma=0.99, alpha=None, lambda_=0):
        super().__init__(observation_space, action_space, policy, gamma, alpha, lambda_)

    def update_Q(self, obs:AgentObservation):
        state, action, reward, state2 = obs.unpack()

        action2 = self.choose_action(state2)
        Qa = np.max(self.Q[state2])
        actions_star = np.argwhere(self.Q[state2] == Qa).flatten().tolist()

        delta = reward + self.gamma * Qa - self.Q[state][action]
        for (s, a) in set(self.eligibility.traces.keys()):
            self.Q[s][a] += self.alpha.get(s,a) * delta * self.eligibility.get(s, a)
            if action2 in actions_star:
                self.eligibility.update(s, a)
            else:
                self.eligibility.to_zero(s, a)

        return action2
```

2.2 SARSA

SARSA algorithm.

Algorithm 2: SARSA algorithm Python implementation.

```python
class Sarsa(TDBrain):
    def __init__(self, observation_space:Discrete, action_space, policy:Policy=EGreedy(),
                 gamma=0.99, alpha=None, lambda_=0.0):
        super().__init__(observation_space, action_space, policy, gamma, alpha, lambda_)

    def update_Q(self, obs:AgentObservation):
        state, action, reward, state2 = obs.unpack()
```

```

        action2 = self.choose_action(state2)
        Qa = self.Q[state2][action2]

        delta = reward + self.gamma * Qa - self.Q[state][
            ↪ action]
        for (s, a) in set(self.eligibility.traces.keys()):
            self.Q[s][a] += self.alpha.get(s,a) * delta *
                ↪ self.eligibility.get(s, a)
            self.eligibility.update(s, a)

        return action2
```

3 LTL$_f$/LDL$_f$ Non-Markovian Rewards

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

Intro.

4.1 Framework

Introduction to the framework.

4.2 Examples

Examples like the one on the website (+CODE).

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

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

Algorithm 3: Robot feature extractor Python implementation.

```python
class BreakoutNRobotFeatureExtractor(BreakoutRobotFeatureExtractor):

    def __init__(self, obs_space):
        robot_feature_space = Tuple((
            Discrete(287),
            Discrete(157),
        ))

        self.prev_ballX = 0
        self.prev_ballY = 0
        self.prev_paddleX = 0
        self.still_image = True

        super().__init__(obs_space, robot_feature_space)

    def _extract(self, input, **kwargs):
        self.still_image = not self.still_image
        if self.still_image:
            return (self.prev_ballX-self.prev_paddleX+143, self.prev_ballY)
        # Extract position of the paddle:
        paddle_img = input[189:193,8:152,:]
        gray = cv2.cvtColor(paddle_img, cv2.COLOR_RGB2GRAY)
        thresh = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)[1]
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        min_distance = np.inf
        paddleX = self.prev_paddleX
        for c in cnts:
```

```
            M = cv2.moments(c)
            if M["m00"] == 0:
                continue
            pX = int(M["m10"] / M["m00"])
            if abs(self.prev_paddleX - pX) < min_distance:
                min_distance = abs(self.prev_paddleX - pX)
                paddleX = pX

        # Extract position of the ball:
        ballX = self.prev_ballX
        ballY = self.prev_ballY
        ballspace_img = input[32:189,8:152,:]
        lower = np.array([200, 72, 72], dtype=np.uint8)
        upper = np.array([200, 72, 72], dtype=np.uint8)
        mask = cv2.inRange(ballspace_img, lower, upper)
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        for c in cnts:
            M = cv2.moments(c)
            # Avoid to compute position of the ball if M["m00"] is zero:
            if M["m00"] == 0:
                continue
            # Calculate the centroid
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            # Check that the centroid is actually part of the ball:
            left_black = False
            right_black = False
            if cX > 3:
                if ballspace_img[cY][cX-3][0] != 200 or \
                    ballspace_img[cY][cX-3][1] != 72 or \
                    ballspace_img[cY][cX-3][2] != 72:
                    left_black = True
            else:
                 if ballspace_img[cY][cX+3][0] != 200 or \
                     ballspace_img[cY][cX+3][1] != 72 or \
                     ballspace_img[cY][cX+3][2] != 72:
                     right_black = True
            if left_black or right_black:
                ballX = cX
                ballY = cY

        self.prev_ballX = ballX
        self.prev_ballY = ballY
        self.prev_paddleX = paddleX

        return (self.prev_ballX - self.prev_paddleX + 143, self.prev_ballY)
```

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

Algorithm 4: Goal feature extractor Python implementation.

```python
class BreakoutGoalFeatureExtractor(FeatureExtractor):
    def __init__(self, obs_space, bricks_rows=6, bricks_cols=18):
        self.bricks_rows = bricks_rows
        self.bricks_cols = bricks_cols
        output_space = Box(low=0, high=1, shape=(bricks_cols, bricks_rows), dtype=np.uint8)
        super().__init__(obs_space, output_space)

    def _extract(self, input, **kwargs):
        bricks_features = np.ones((self.bricks_cols, self.bricks_rows))
        for row, col in itertools.product(range(self.bricks_rows), range(self.bricks_cols)):
            # Pixel of the observation to check:
            px_upper_left  = int( 8 + 8 * col)
            py_upper_left  = int(57 + 6 * row)
            px_upper_right = int(15 + 8 * col)
            py_upper_right = int(57 + 6 * row)

            # Checking max because the input has 3 channels:
            if max(input[py_upper_left][px_upper_left]) == 0 or \
                max(input[py_upper_right][px_upper_right]) == 0:
                bricks_features[col][row] = 0

        return bricks_features
```

*Ext used to improve implementation.

$\text{LTL}_f/\text{LDL}_f$ implementation (with Marco Favorito libraries).

Algorithm 5: $\text{LTL}_f/\text{LDL}_f$ formulas Python implementation.

```python
def get_breakout_lines_formula(lines_symbols):
    # Generate the formula string
    # E.g. for 3 line symbols:
    # "<(!l0 & !l1 & !l2)*;(l0 & !l1 & !l2);(l0 & !l1 & !l2)*;(l0 & l1 & !l2); (l0 & l1 & !l2)*; l0 & l1 & l2>tt"
    pos = list(map(str, lines_symbols))
    neg = list(map(lambda x: "!" + str(x), lines_symbols))

    s = "(%s)*" % " & ".join(neg)
    for idx in range(len(lines_symbols)-1):
        step = " & ".join(pos[:idx + 1]) + " & " + " & ".join(neg[idx + 1:])
        s += ";({0});({0})*".format(step)
    s += ";(%s)" % " & ".join(pos)
    s = "<%s>tt" % s

    return s
```

```python

class BreakoutCompleteLinesTemporalEvaluator(TemporalEvaluator):
    """Breakout temporal evaluator for delete columns from left to right"""

    def __init__(self, input_space, bricks_cols=3, bricks_rows=3, lines_num=3, gamma=0.99, on_the_fly=False):
        assert lines_num == bricks_cols or lines_num == bricks_rows
        self.line_symbols = [Symbol("l%s" % i) for i in range(lines_num)]
        lines = self.line_symbols

        parser = LDLfParser()


        string_formula = get_breakout_lines_formula(lines)
        print(string_formula)
        f = parser(string_formula)
        reward = 10000

        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)

    @abstractmethod
    def fromFeaturesToPropositional(self, features, action, *args, **kwargs):
        """map the matrix bricks status to a propositional formula
        first dimension: columns
        second dimension: row
        """
        matrix = features
        lines_status = np.all(matrix == 0.0, axis=kwargs["axis"])
        result = set()
        sorted_symbols = reversed(self.line_symbols) if kwargs["is_reversed"] else self.line_symbols
        for rs, sym in zip(lines_status, sorted_symbols):
            if rs:
                result.add(sym)

        return frozenset(result)

class BreakoutCompleteRowsTemporalEvaluator(BreakoutCompleteLinesTemporalEvaluator):
```

```python
    """Temporal evaluator for complete rows in order"""

    def __init__(self, input_space, bricks_cols=3, bricks_rows=3, bottom_up=True, gamma=0.99, on_the_fly=False):
        super().__init__(input_space, bricks_cols=bricks_cols, bricks_rows=bricks_rows, lines_num=bricks_rows, gamma=gamma, on_the_fly=on_the_fly)
        self.bottom_up = bottom_up

    def fromFeaturesToPropositional(self, features, action, *args, **kwargs):
        """complete rows from bottom-to-up or top-to-down, depending on self.bottom_up"""
        return super().fromFeaturesToPropositional(features, action, axis=0, is_reversed=self.bottom_up)
```

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