SLAR - A Real World Reinforcement Learning Standard

Johannes Loewe and Alexander Zehetmaier
Radboud University

Submitted: 13 January 2020

Abstract

This paper introduces a new real-world reinforcement learning standard. Training reinforcement learning agents is hard, training them in the real world is even harder. No parallelism of multiple virtual worlds and not a thousand policy updates per second.
Keywords: Robotics, Reinforcement Learning, Neural Networks & Artificial Intelligence

1 Introduction

In the last couple of years reinforcement learning had several breakthroughs in the field of machine learning. Lots of credit goes to DeepMind who developed alphaGo, alphaZero and more recently alphaStar and alphaFold.

Nevertheless in the field of robotics reinforcement learning has not unfold the same amount of acquisition. Robotics hardware is expensive and not accessible to many researchers and engineers around the globe. Furthermore it is harder to train real-world reinforcement learning robots since you can not simulate multiple worlds in parallel. Marc Raibert the CEO of Boston Dynamics lately claimed that they do not use any reinforcement learning in their robots. It is suspected that Boston Dynamics mostly relies on sequential composition of dynamically dexterous robot behaviors. The vision of this project is to push the field of real-world reinforcement learning more into the direction of its boundaries. In the future complicated walking behaviours and motion control may uses more reinforcement learning as these fields are using it right now.

To accelerate the process of reinforcement learning in the field of robotics Jolex-Corp. developed an open-source six legged robot which can be 3D-printed from anyone around the globe. The costs to build a fully working hexapod does not exceed a 100 Euro threshold.

Figure 1 summarizes the eight pillars what the SLAR real-world reinforcement learning project stands for.

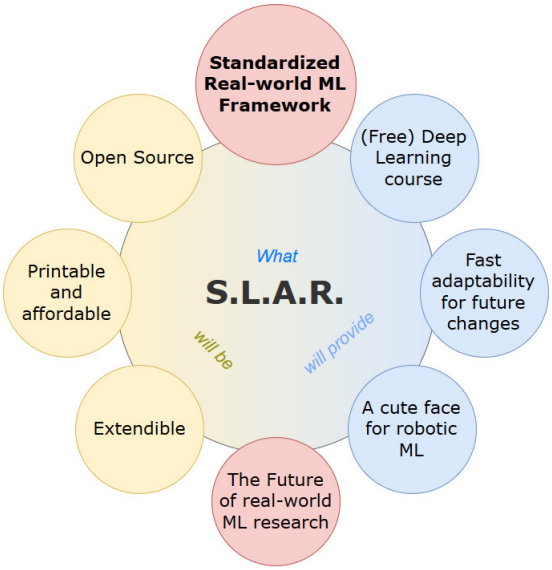

Figure 1: The 8-pillars of SLAR

2 Hardware Specification

One key aspect of this project is to keep the hardware as simple and cheap as possible. Since reinforcement learning is computational expensive this should lead to smarter algorithms instead of more expensive hardware.

Everybody should be able to afford SLAR and everybody should play with the same deck of cards.

2.1 Processing Unit

As processing unit we use a Raspberry Pi 3 since it is available in (every?) country and prominent for its community and cheap prices.

2.2 Servo Motors

2.3 Servo Motor Control

2.4 Power Management

3 Software Specification

In this section the software specifications are discussed. Only free available opensource packages will be used.

3.1 Language and Frameworks

Python3 is the programming language of choice for most machine learning researches. Not only because of it's simple syntax but also because of it's numerous fast numeric computing libraries. These libraries are highly optimized and mostly use C or C++ backend.

3.2 Discretization of Leg Positions

To reduce the size of the sample space we discretize the different positions of all the legs. With this strategy the complexity of the problem gets reduced to a minimum, since it limits the robot's degree of freedom.

Each servo-motor has 4 possible positions and since we have 18 motors we have a problem-space of 4^{18} possible positions.

3.3 Continuous Leg Positions

Continuous positions of the legs are increasing the complexity enormously. (Pseudo Continuous -¿ servo motor 255 positions: 255^{18})

3.4 Natures Inspiration: Leg-Grouping

All insects have six legs, but not everything with six legs is an insect -SLAR.

Within million of years insectual evolution developed a specific grouping of the insect's legs whilst these are walking. This grouping enhanced their locomotive behaviour and can be seen in Figure 2.

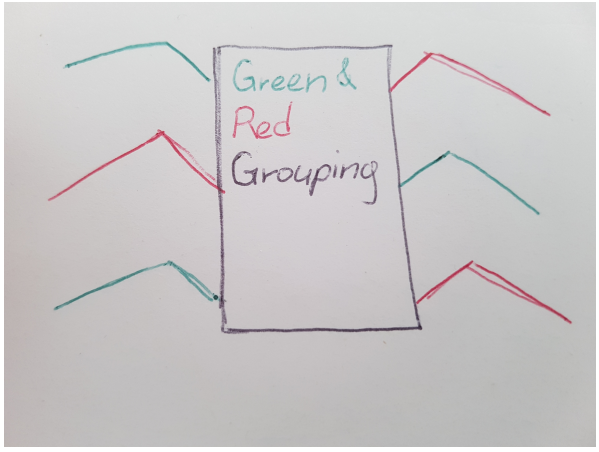

Figure 2: Placeholder for nice image

3.5 Q-Learning

The goal of Q-learning is to update the values of the Q-Table in such a manner that these values represent a plan for the agent. This plan is constructed in a way to maximize the reward in this specific environment.

Q-learning is a reinforcement learning algorithm which has whether the rewards nor the transition-probabilities given. The algorithm consists of a Q-table which is a two-dimensional tensor (matrix). The rows are the states and the columns represent the actions possible at the current state in an environment.

We update the values of the Q-table according to the bellman equations, which can be seen in Equation 1 below.

$$Q(s,a) \leftarrow Q(s,a) \; + \; \alpha\Big[reward + \gamma \max_a Q(next\ s, all\ a) - Q(s,a)\Big] \quad (1)$$

Algorithm 1 Bellman Equations

1: **function** BELLMAN($Q, s, a, rew, max\_next\_a, \gamma, \alpha$):
2: **return** $Q(s,a) + \alpha\Big[rew + \gamma \max_a Q(next\ s, all\ a) - Q(s,a)\Big]$
3: **end function**

3.5.1 Algorithmic Description

1. Initialize Q-Table
   - random
   - all zeros
   - with domain knowledge initialization
2. Explore vs Exploit Action
   - select an action for current state s
3. Execute selected Action → Go to s'
4. Select Action in s'
   - action with highest q-value
5. Update Q-Table according to Equation 1
6. Update the Current State to be the Next State
7. If Goal-State → New Episode

3.5.2 Pseudo-Code

Algorithm 2 Find representative Q-Values

```
function Q-Learning(Q, env, ε, α, γ):
    Q(#states, #actions) ← initialization
    for each episode do
        state ← env.reset()
        done ← false
        while not done do
            action ← explore_vs_exploit(ε)
            s', rew, done ← env.step(action)
            max_next_action ← max(Q(s'))
            Q(state, action) ← bellman(Q, state, action, rew, max_next_action, γ, α)
            state ← s'
        end while
    end for
end function
```

3.5.3 Q-Table with Discrete Leg Postions

Constraining the Servomotors according to section 3.2 leads to discrete motor positions. Consequently this reduces the amount of rows and columns at the Q-Table.

18 motors with 3 position's each (left, middle, right) leads to $18 \cdot 3 = 54$ actions at every state. If we now additionally group the legs according to section 3.4 we can divide this number by 2 which leads to 27 actions at every state. We divide by 2 since we split the motors in 2 different groups. The first group of motors is performing an action and afterwards the second group of motors is performing an action.

The Q-Table's rows correspond to the possible states. Since we have ...

Summarizing this section with 3 possible discrete positions of every motor a Q-Table with the following properties is constructed: A Q-Table with 27-columns and XX-rows evolves when grouping the legs according to section 3.4. Not grouping the legs would correspond to a Q-Table with 54-columns and XX-rows.

3.6 Neural Network meets Q-Learning

Deep Q-Learning is a Neural Network getting the states of a Q-Table as the input and trying to predict an appropriate action as output.

These technique lead to many advances in the last couple of years and was an key aspects, leading to the breakthrough of reinforcement learning in Atari Games [1].

4 Previous Work

5 Discussion

6 Conclusion

References

[1] Deepmind, Nature, Human-level control through deep reinforcement learning, Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al.