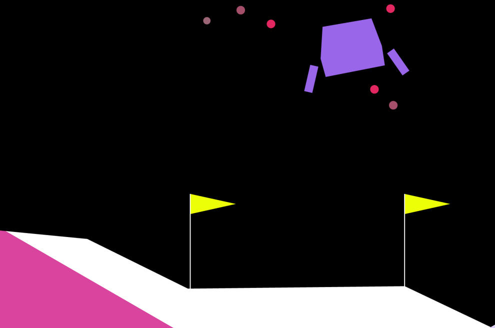


Lunar Lander

RL project

Valentina Blasone
Giulia Marchiori Pietrosanti

July 18, 2022

Overview

1. Introduction and problem statement
2. TD-methods with discretization
3. SARSA with function approximation
4. DQN
5. Final results
6. Conclusions

Introduction and problem statement

Introduction and problem statement
Problem statement

The project tackles the problem of the "LunarLander-v2", an environment of OpenAI gym (https://www.gymlibrary.ml/environments/box2d/lunar_lander/) [3].

The **goal** is to land a shuttle on the moon.

Start

- The lander starts at the top center of the viewport with a random initial force applied to its center of mass.

Termination:

1. The lander's body gets in contact with the moon;
2. The lander gets outside of the viewport.

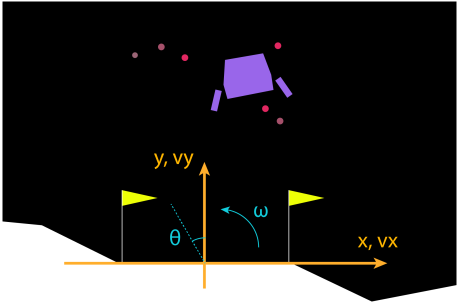
Introduction and problem statement

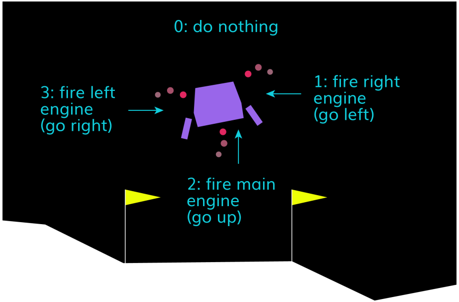
States - **continuous**

- 8 variables: 6 continuous (x, y, v_x, v_y, θ, ω) and 2 booleans.

Actions - **discrete**

- 4 actions: do nothing, fire right (go left), fire main, fire left (go right).





Introduction and problem statement

Reward structure

Positive reward when:

- Moving towards the landing pad;
- Coming to rest ($+100$ points);
- Leg on the ground ($+10$ points. each)

Negative reward when:

- Moving away from the landing pad;
- Lander crashes (-100 points);
- Fire main engine (-0.3 points), fire left/right engine (-0.03 points).

The game is considered solved with 200 points.

Introduction and problem statement

Model-free problem, with continuous states.

To solve the problem different approaches can be used. In the project the following algorithms were implemented:

1. TD-methods with discretization
2. SARSA with non-linear function approximation
3. DQN

TD-methods with discretization

TD-methods with discretization

Discretization

Set $I = [var_{min}, var_{max}]$ for each variable var - the interval is divided in n bins; in additions, two bins collect all the observations that fall outside the minimum/maximum of I, respectively.

Discretization: equally sized bins with with reasonable choice of minimum/maximum values.

Complexity: Huge state space

$$
\begin{aligned}
&O(x \times y \times v_x \times v_y \times \theta \times \omega \times \texttt{bool}_1 \times \texttt{bool}_2) = \\
&= O(6 \times 4 \times 4 \times 4 \times 4 \times 4 \times 2 \times 2) = \\
&= 24576
\end{aligned}
$$

TD-methods with discretization

SARSA (on-policy TD control)

Inputs: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$;
Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$,
arbitrarily except that $Q(\text{terminal}, \cdot) = 0$;
foreach *episode* **do**
Initialize S;
Choose A from S using policy derived from Q (e.g., ε-greedy);
foreach *step of episode* **do**
Take action A, observe R, S';
Choose A' from S' using policy derived from Q (e.g., ε-greedy);
$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \; Q(S', A') \; - Q(S, A)]$;
$S, A \leftarrow S', A'$;
end foreach
until S is terminal
end foreach

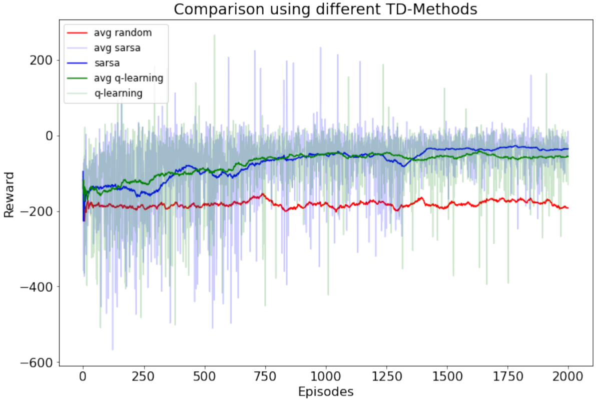
Q-learning (off-policy TD control)

Inputs: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$;
Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$,
arbitrarily except that $Q(\text{terminal}, \cdot) = 0$;
foreach *episode* **do**
Initialize S;
foreach *step of episode* **do**
Choose A from S using policy derived from Q (e.g., ε-greedy);
Take action A, observe R, S';
$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \; \max_a Q(S', a) \; - Q(S, A)]$;
$S \leftarrow S'$;
end foreach
end foreach

Pseudocodes taked from [5].

TD-methods with discretization

Results



- Both SARSA and Q-learning **perform poorly**, as expected
- Discretization is **not rich enough** to describe the continuous states
- **Insufficient exploration**

Average results are in terms of average reward over the previous 100 episodes.

SARSA with function approximation

SARSA with function approximation

The general idea is to:

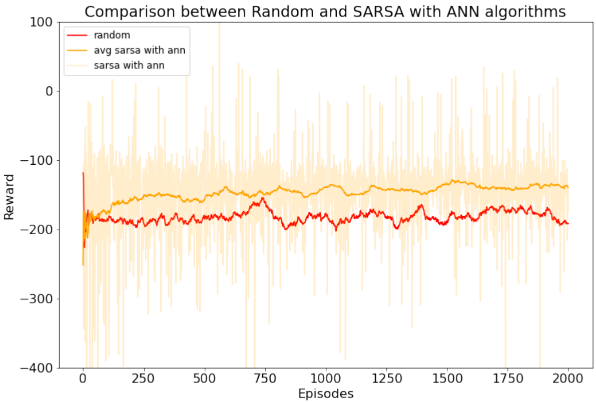
1. **Compute** the Q-values with function approximation instead of using the Q-table
2. **Update** approximation given the past experience

In this project, a semi-gradient SARSA with non-linear function approximation through ANN was implemented.

$$Q(S, a) \approx ANN, \quad a \in A$$

SARSA with function approximation

Results



- **Poor performance**, worse than expected
- Possible **explanation**: update of all weights of an action network combined with highly correlated observations
- Possible **solution**: use batches of observations, not feasible for standard on-policy

Average results are in terms of average reward over the previous 100 episodes.

DQN

DQN

Deep reinforcement learning method proposed by DeepMind [4].

It is a Q-learning algorithm that uses ANN to approximate the state-action value function.

Some techniques are used to improve and stabilize the learning:

1. ε-greedy policy
2. Experience replay
3. Target network

DQN

Experience replay

Problem: sequence of experience tuples (s, a, r, s') derived from the interaction of the agent with the environment is auto-correlated.

Solution: the idea is to store a history of experience tuples in a memory. This allows to:

- use mini-batches of experiences drawn at random from the stored samples, instead of using a single experience tuple. This should lead to more stable training and better convergence.

DQN
Target network

Problem: target values change continuously as the parameters of the network change with each iteration. This can make the learning unstable.

Solution: use an additional target network.

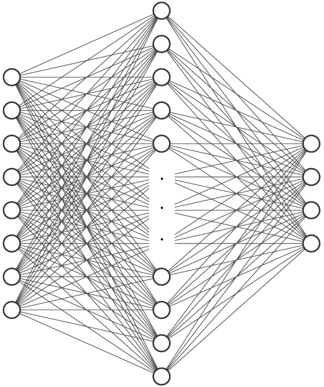
- the **main network** is only used for calculating the Q-values for selecting the next action and is trained every `update_every` step;
- the **target network** is used for computing the target Q-values to update the main model. This network is updated every `update_every` step.
  soft update:

$$w'_{\texttt{target}} = (1 - \tau) \cdot w_{\texttt{target}} + \tau \cdot w_{\texttt{main}}$$

.

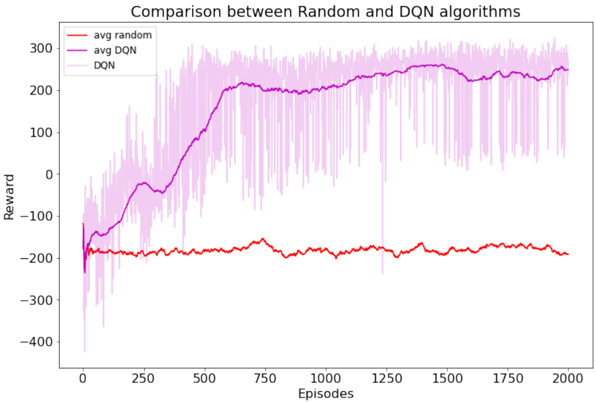
DQN

Networks and hyperparameters



- $lr = 5e^{-4}$
- $\gamma = 0.99$
- ε decays over episodes
- $\texttt{update\_every} = 4$
- $\texttt{buffer\_size} = 1e5$
- $\texttt{batch\_size} = 64$
- $\tau = 1e - 3$

DQN
Results

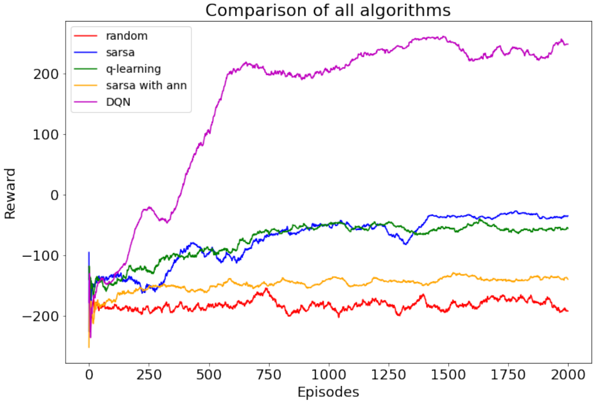


- **Good performance** as expected;
- The winning condition is reached;
- To reach the winning condition only $\sim$ 600 episodes are needed.

Average results are in terms of average reward over the previous 100 episodes.

Final results

Final results
Comparison



	Max Avg Reward
SARSA	-26.62
Q-learning	-39.73
ANN SARSA	-128.54
DQN	261.36

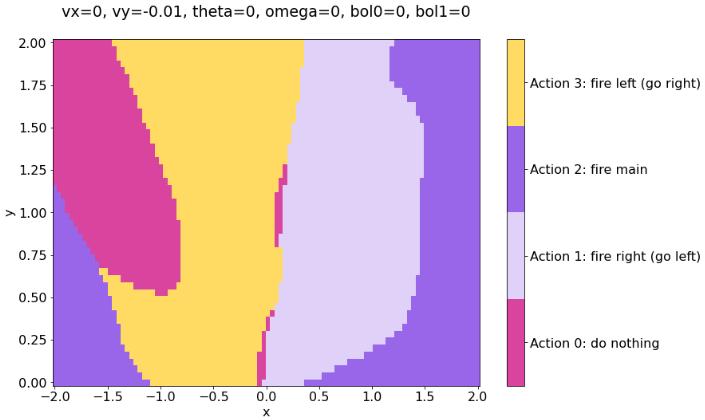
- DQN largely outperforms the other methods, as expected

All results are in terms of average reward over the previous 100 episodes.

Final results

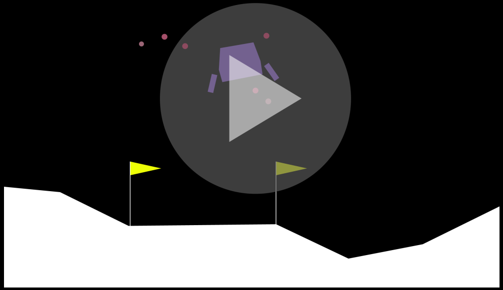
DQN policy visualization

Can we interpret the learned optimal policy?



Final results

DQN: watch the lunar lander successfully landing on the moon!



Conclusions

Conclusions

To sum up:

Different model-free critic-only methods have been implemented and compared:

- **SARSA** and **Q-learning** need discretization to be applied. Performance is low as expected, discretization is neither rich enough to describe the continuous states, nor feasible for a good exploration;
- **SARSA with ANN function approximation** works directly on continuous states. Performance is even worse, a guess is that the weights update with highly correlated observations and without batches negatively affects the learning;
- **DQN** works directly on continuous states. Very good performance, it solved the environment in a small number of epochs.

Conclusions

What else we could have done, which was outside the scope of the project:

- Experiment with richer discretization;
- Perform a more in-depth parameters tuning, in all algorithms but especially in the ones based on ANN;
- Use some stochastic policy methods, which are a common alternative in the case of continuous states.

Thank you!

References

Vincent Bons.
Solving lunar lander using dqn in keras.
https://wingedsheep.com/lunar-lander-dqn/, 2020.

Chanseok Kang.
Deep q-network (dqn) on lunarlander-v2.
https://goodboychan.github.io/python/reinforcement_learning/pytorch/udacity/2021/05/07/DQN-LunarLander.html, 2021.

Oleg Klimov.
Lunar lander.
https://www.gymlibrary.ml/environments/box2d/lunar_lander/#credits.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.
Playing atari with deep reinforcement learning.
arXiv preprint arXiv:1312.5602, 2013.

Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
MIT Press, 1998.