Upside Down Reinforcement Learning

Iván Hidalgo
Gorka Arrausi
Iñigo Rojo

Index

- Introduction
- Theory
- Practise
- Conclusion

Introduction

When was created?

We are presenting the technical report of Jürgen Schmidhuber on the topic of upside down reinforcement learning presented in the 5th of december of 2019, which was presented days latter at the NeurIPS, the conference in neural Information Processing Systems held annually.

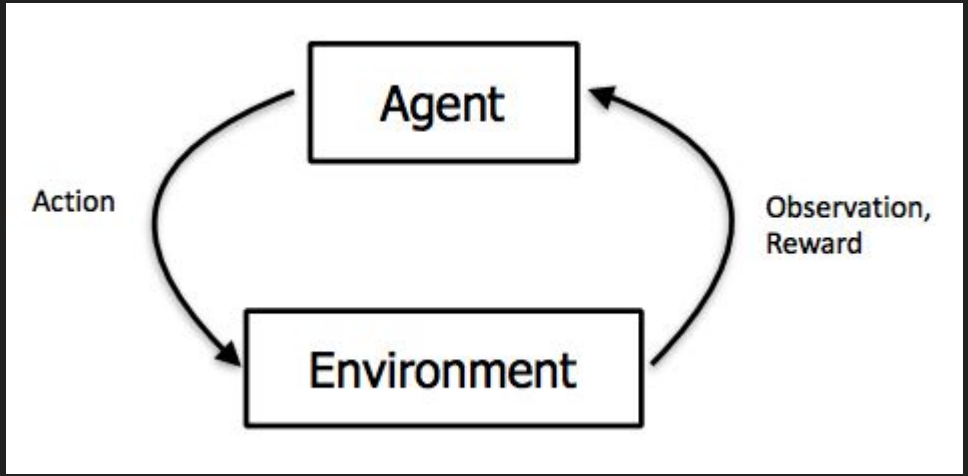
Reinforcement Learning

- Simple concept

- An agent interacts with actions

- The environment answers with a new state and a reward.

- The agent uses the feedback

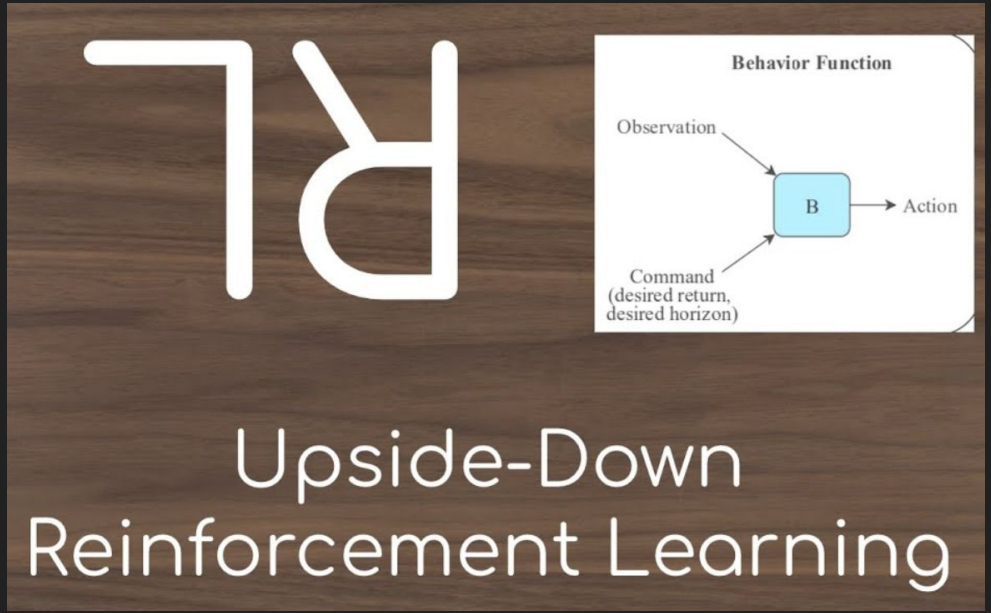


Theory

- Reinforced Learning → Supervised Learning

- Desired reward + state of the environment as input

- Predicts needed actions



Upside Down Reinforcement Learning

- Desired reward as input and predicted action as output

- Desired return + desired number of timesteps(horizon) + state of the environment ⟶ Predicted best action

State	Desired Return	Desired Horizon	Action
s_0	2	1	a_1
s_0	1	1	a_2
s_0	1	2	a_1
s_1	-1	1	a_3

- In short it manages to learn by interacting with the environment using gradient descent to map self-generated commands to corresponding action probabilities with the possibility of using this acquired knowledge to solve new problems.

- An agent may interact with its environment throughout a single prolonged period of time. At a given time, the history of actions and vector-valued costs like time, energy, pain and reward signals,etc will contain all the agent can know about the present state of itself and the environment. Its task then will be to obtain a lot of reward before some period of time elapses.

- For all past sub periods of time it can evaluate and implement additional, consistent, vector-valued command inputs for itself trying to achieve similar reward with less cost or to try to surpass that possible reward that happened in the past.

- It's possible now to use the gradient based supervised learning to assign our computer to map sensory inputs alongside our desired commands of rewards and the timeframe(horizons) to the known action sequences. If we find different but equally costly actions between our starting point and towards some goal we can train our computer to approximate the expected value of the appropriate actions.

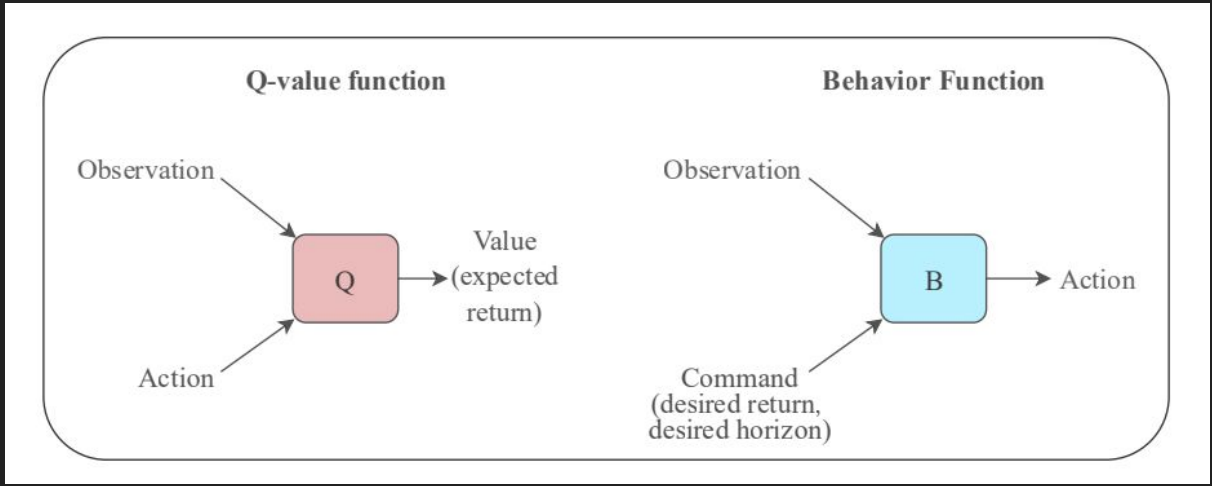
Through our single long period of time our algorithm will learn to solve problems where there are logistical constraints of costs.

With this done we should be able to define commands of our own specific to problems affecting the user and hopefully our trained computer will be able to generalize based on what it has learned. This will not only help our problems, but also increase the experience of the computer which is ultimately what machine learning means: To learn from experience with respect to some task as measured by some method and to improve this performance as the experience grows.

> A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.
>
> — Tom M. Mitchell

Practice

Difference between RL and UDRL

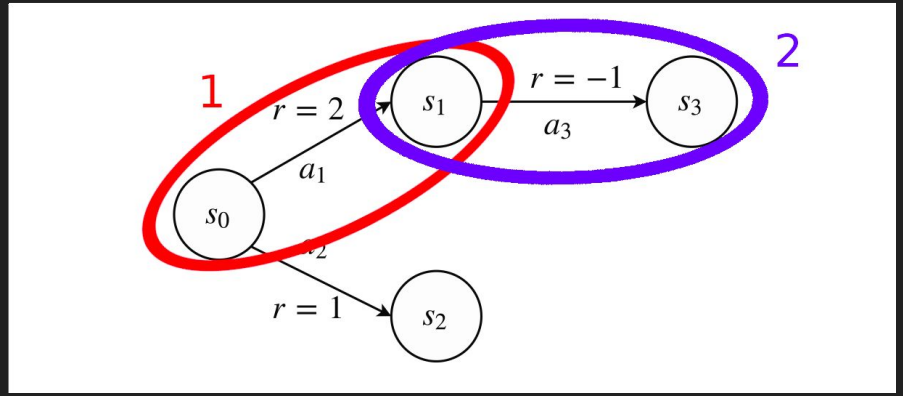


Action-value function
(Q) in traditional RL

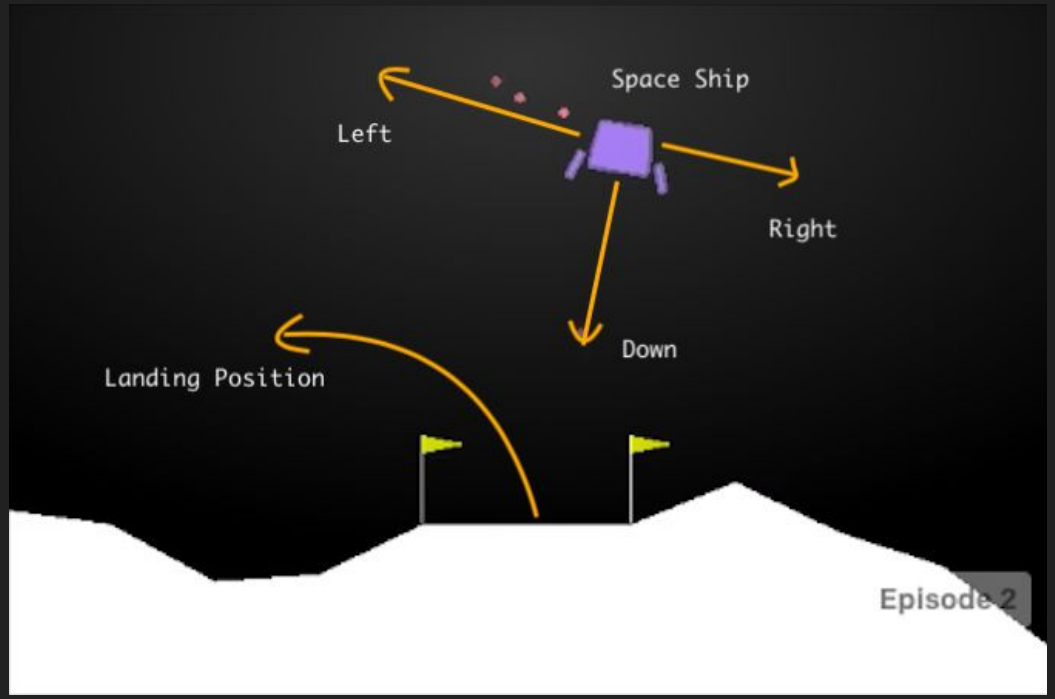
Behavior function
(B) in UDRL

How do commands work?

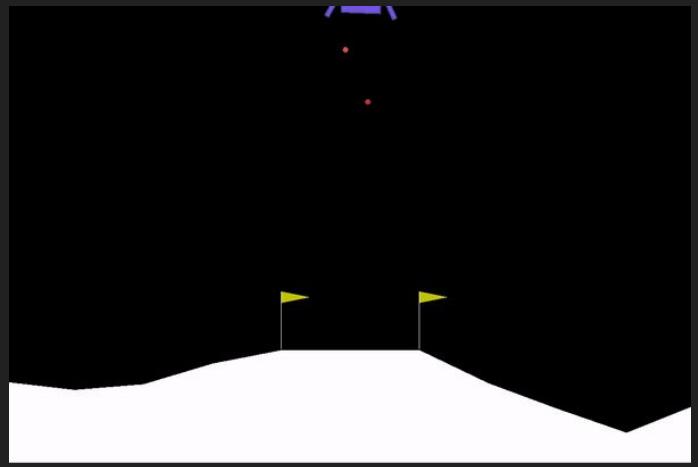
State	Desired Return	Desired Horizon	Action
s_0	2	1	a_1
s_0	1	1	a_2
s_0	1	2	a_1
s_1	-1	1	a_3

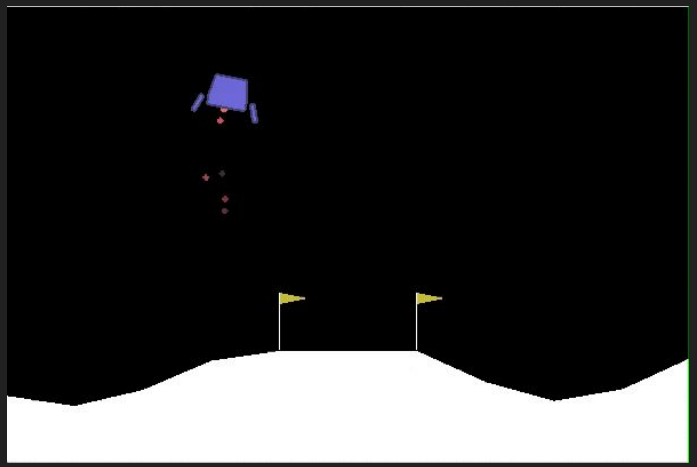


LunarLander - working environment



LunarLander - working environment





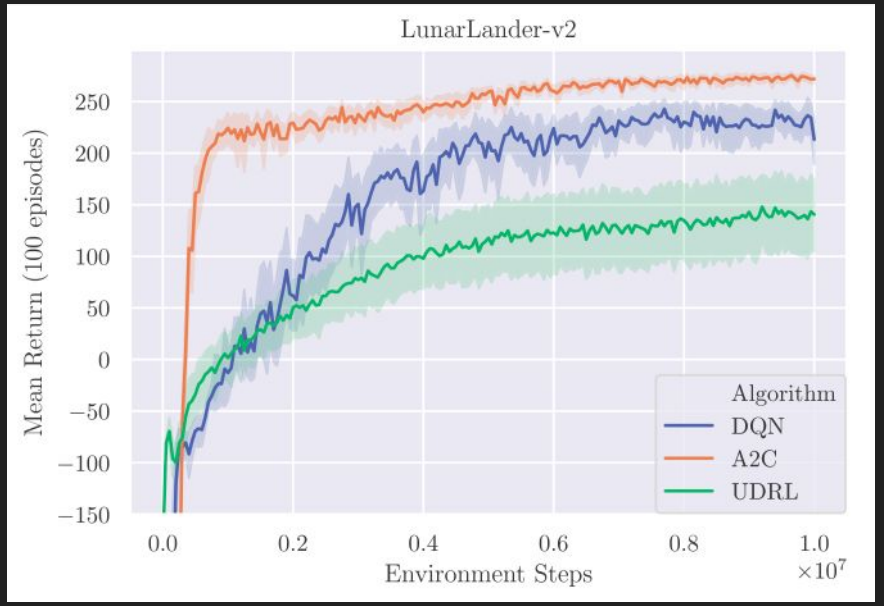
Settings and results of LunarLander

This was the first set up:

- All agents were implemented using Artificial NN.
- The behaviour function of UDRL has been implemented with Fully-Connected Feed-Forward Networks.
- The command inputs suffered a small modification.

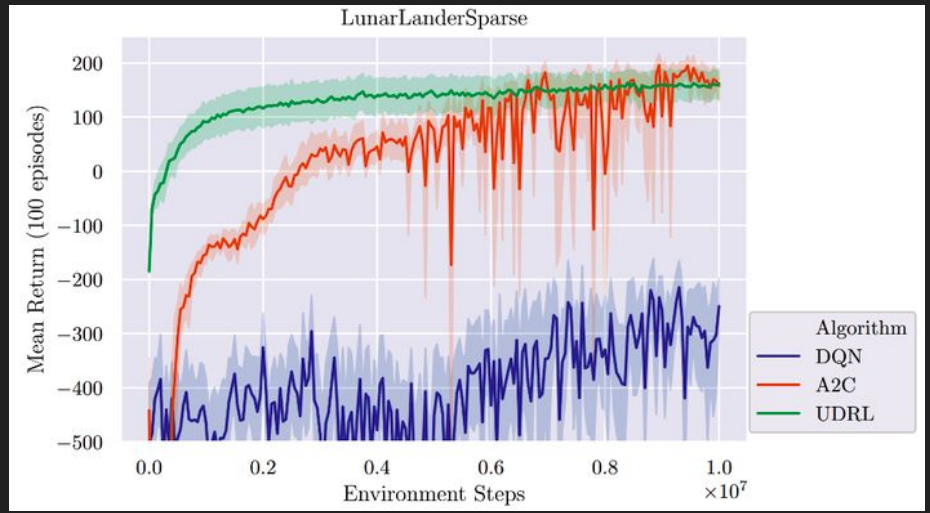
Then 20 seeds were used for each environment and algorithm and were sampled like this:

- [1M, 10M) for training.
- [0.5M, 1M) for evaluation during hyperparameters tuning.
- [1, 0.5M) for final evaluation with best hyperparameters.



Settings and results of LunarLander

With a simple change
in reward structure



Conclusion

To summarize, upside down reinforced learning bring us a new tool that while it's no miracle, not working better in every scenario, for some concrete tasks where reinforced learning doesn't yield very good results this bridge between supervised learning and reinforced learning might do the trick.

Thanks for your attention