Reinforcement Learning Upside Down

Don't predict rewards - Just Map Them to Actions

Matthia Sabatelli
Montefiore Institute, Department of Electrical Engineering and Computer Science, Université de Liège, Belgium

March 18th 2021

Presentation outline

1. Reinforcement Learning
2. Upside-Down Reinforcement Learning
3. Personal Thoughts

Reinforcement Learning

The goal of Reinforcement Learning is to train an agent to interact with an environment which is modeled as a Markov Decision Process (MDP)

- a set of possible states $\mathcal{S}$
- a set of possible actions $\mathcal{A}$
- a reward signal $R(s_t, a_t, s_{t+1})$
- a transition probability distribution $p(s_{t+1}|s_t, a_t)$
- a discount factor $\gamma \in [0, 1]$

Reinforcement Learning

The agent interacts continuously with the environment in the rl-loop

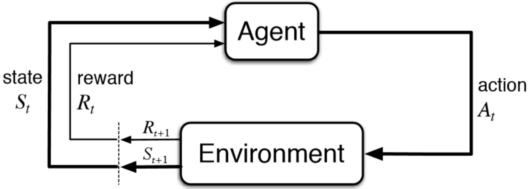


Figure: Image taken from page 48 of Sutton and Barto [2].

Reinforcement Learning

The goal of the agent is to maximize the expected discounted cumulative reward

$$
\begin{aligned}
G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\
&= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.
\end{aligned}
$$

Reinforcement Learning

An agent decides how to interact with the environment based on its policy which maps every state to an action:

$$\pi : \mathcal{S} \to \mathcal{A}$$

- The essence of RL algorithms is to find the best possible policy.
- How to define a good policy?

Reinforcement Learning

We need the concept of value function

- The state value function $V^{\pi}(s)$
- The state-action value function $Q^{\pi}(s, a)$

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, \pi\right]$$

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a, \pi\right].$$

Both value functions can compute the desirability of being in a specific state.

Reinforcement Learning

When maximized both value functions satisfy a consistency condition that allows us to re-express them recursively

$$V^*(s_t) = \max_a \sum_{s_{t+1}} p(s_{t+1}|s_t, a)\Big[\Re(s_t, a, s_{t+1}) + \gamma V^*(s_{t+1})\Big]$$

and

$$Q^*(s_t, a_t) = \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t)\Big[\Re(s_t, a_t, s_{t+1}) + \gamma \max_a Q^*(s_{t+1}, a)\Big],$$

which correspond to the Bellman optimality equations.

Reinforcement Learning

Value functions play a key role in the development of RL

- Dynamic Programming and Value Iteration (if $p(s_{t+1}|s_t, a_t)$ or $\Re(s_t, a_t, s_{t+1})$ are known!)
- Model Free RL:
  - Value based methods
  - Policy based methods

However ...

... when dealing with large MDPs learning these value functions can become complicated since they scale with respect to the state-action space of the environment.

Reinforcement Learning

Why is RL considered to be so challenging?

- We are dealing with a component of time
- The environment is unknown
- There no such a thing as a fixed dataset
- The dataset consists of a moving target

These are all differences that make Reinforcement Learning so different from Supervised Learning!

Upside-Down Reinforcement Learning

This idea has been introduced in 2 papers:

- Schmidhuber, Juergen. "Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions." arXiv preprint arXiv:1912.02875 (2019).
- Srivastava, Rupesh Kumar, et al. "Training agents using upside-down reinforcement learning." arXiv preprint arXiv:1912.02877 (2019).

Upside-Down Reinforcement Learning

The paper starts with two strong claims:

- Supervised Learning (SL) techniques are already incorporated within Reinforcement Learning (RL) algorithms
- There is no way of turning an RL problem into an SL problem

Main Idea

Yet the main idea of Upside-Down RL is to turn traditional RL on its head and transform it into a form of SL.

Upside-Down Reinforcement Learning

- Let's take a look at why the gap between SL and RL is actually not that large
- Consider Deep Q-Networks, a DRL technique which trains neural networks for learning an approximation of the state-action value function $Q(s, a; \theta) \approx Q^*(s, a)$

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \Big[\big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \big)^2 \Big],$$

Upside-Down Reinforcement Learning

- Let's take a look at why the gap between SL and RL is actually not that large
- Consider Deep Q-Networks, a DRL technique which trains neural networks for learning an approximation of the state-action value function $Q(s, a; \theta) \approx Q^*(s, a)$

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \Bigg[\Big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \Big)^2 \Bigg],$$

Upside-Down Reinforcement Learning

- Let's take a look at why the gap between SL and RL is actually not that large
- Consider Deep Q-Networks, a DRL technique which trains neural networks for learning an approximation of the state-action value function $Q(s, a; \theta) \approx Q^*(s, a)$

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \Big[\big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \big)^2 \Big],$$

Upside-Down Reinforcement Learning

In order to successfully train DRL algorithms we are already exploiting SL

- We need to collect a large dataset of RL trajectories $\langle s_t, a_t, r_t, s_{t+1} \rangle$

Upside-Down Reinforcement Learning

In order to successfully train DRL algorithms we are already exploiting SL

- We need to collect a large dataset of RL trajectories $\langle s_t, a_t, r_t, s_{t+1} \rangle$
- Networks are trained with common SL loss functions (MSE)

Upside-Down Reinforcement Learning

In order to successfully train DRL algorithms we are already exploiting SL

- We need to collect a large dataset of RL trajectories $\langle s_t, a_t, r_t, s_{t+1} \rangle$
- Networks are trained with common SL loss functions (MSE)
- Not visited states are dealt with data-augmentation

Upside-Down Reinforcement Learning

In order to successfully train DRL algorithms we are already exploiting SL

- We need to collect a large dataset of RL trajectories $\langle s_t, a_t, r_t, s_{t+1} \rangle$
- Networks are trained with common SL loss functions (MSE)
- Not visited states are dealt with data-augmentation
- Policy and Value function regularization

Upside-Down Reinforcement Learning

DRL algorithms come with a lot of issues that go back to tabular RL

- It is not easy to learn value functions, i.e they can be biased

Upside-Down Reinforcement Learning

DRL algorithms come with a lot of issues that go back to tabular RL

- It is not easy to learn value functions, i.e they can be biased
- When approximated, the algorithms that learn these functions diverge

Upside-Down Reinforcement Learning

DRL algorithms come with a lot of issues that go back to tabular RL

- It is not easy to learn value functions, i.e they can be biased
- When approximated, the algorithms that learn these functions diverge
- Same issues hold for policy gradient methods: extrapolation error

Upside-Down Reinforcement Learning

DRL algorithms come with a lot of issues that go back to tabular RL

- It is not easy to learn value functions, i.e they can be biased
- When approximated, the algorithms that learn these functions diverge
- Same issues hold for policy gradient methods: extrapolation error
- The RL set-up is distorted: see the role of γ

Upside-Down Reinforcement Learning

We consider the same setting as the one that characterizes classic RL

- We are dealing with Markovian environments

Upside-Down Reinforcement Learning

We consider the same setting as the one that characterizes classic RL

- We are dealing with Markovian environments
- We have access to states, actions and rewards (s, a, r)

Upside-Down Reinforcement Learning

We consider the same setting as the one that characterizes classic RL

- We are dealing with Markovian environments
- We have access to states, actions and rewards (s, a, r)
- The agent is governed by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$

Upside-Down Reinforcement Learning

We consider the same setting as the one that characterizes classic RL

- We are dealing with Markovian environments
- We have access to states, actions and rewards (s, a, r)
- The agent is governed by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$
- We deal with RL episodes that are described by *trajectories* τ in the form of $\langle s_t, a_t, r_t, s_{t+1} \rangle$

Upside-Down Reinforcement Learning

Even if the setting is the same one as in RL the core principle of Upside-Down Reinforcement Learning is different!

- Value based RL algorithms predict rewards

Upside-Down Reinforcement Learning

Even if the setting is the same one as in RL the core principle of Upside-Down Reinforcement Learning is different!

- Value based RL algorithms predict rewards
- Policy based RL algorithms search for a π that maximizes a return

Upside-Down Reinforcement Learning

Even if the setting is the same one as in RL the core principle of Upside-Down Reinforcement Learning is different!

- Value based RL algorithms predict rewards
- Policy based RL algorithms search for a π that maximizes a return
- Upside-Down RL predicts actions

Upside-Down Reinforcement Learning

In order to predict actions we introduce two new concepts that are not present in the classic RL setting

- A behavior function B
- A set of commands c that will be given as input to B

Upside-Down Reinforcement Learning

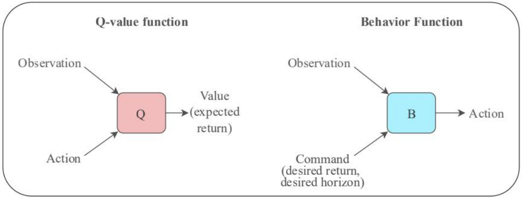
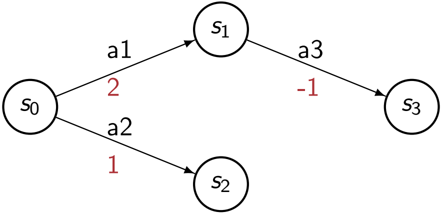


Figure 1: A key distinction between the action-value function (Q) in traditional RL (e.g. Q-learning) and the behavior function (B) in ⅂ꓤ is that the roles of actions and returns are switched. In addition, B may have other command inputs such as desired states or the desired time horizon for achieving a desired return.

Upside-Down Reinforcement Learning



State	d_r	d_h	a
s_0	2	1	a1
s_0	1	1	a2
s_0	1	2	a1
s_1	-1	1	a3

Upside-Down Reinforcement Learning

While simple and intuitive learning B is not enough for successfully tackling RL tasks

- We are learning a mapping $f : (s, d^r, d^h) \rightarrow a$

Upside-Down Reinforcement Learning

While simple and intuitive learning B is not enough for successfully tackling RL tasks

- We are learning a mapping $f : (s, d^r, d^h) \rightarrow a$
- B can be learned for any possible trajectory

Upside-Down Reinforcement Learning

While simple and intuitive learning B is not enough for successfully tackling RL tasks

- We are learning a mapping $f : (s, d^r, d^h) \rightarrow a$
- B can be learned for any possible trajectory

We are missing two crucial RL components

- Improvement of π over time

Upside-Down Reinforcement Learning

While simple and intuitive learning B is not enough for successfully tackling RL tasks

- We are learning a mapping $f : (s, d^r, d^h) \rightarrow a$
- B can be learned for any possible trajectory

We are missing two crucial RL components

- Improvement of π over time
- Exploration of the environment

Upside-Down Reinforcement Learning

Yet there exists one algorithm that is able to deal with these issues and that trains Upside-Down agents (sort of) successfully. Its main components are:

- An experience replay memory buffer E which stores different τ while learning progresses
- A representation of a state s_t and a command tuple $c_t = (d_t^r, d_t^h)$
- A behavior function $B(s_t, c_t; \theta)$ that predicts an action distribution $P(a_t|s_t, c_t)$

Upside-Down Reinforcement Learning

Algorithm 1 Upside-Down Reinforcement Learning: High-level Description.

```
Initialize replay buffer with warm-up episodes using random actions     // Section 2.3.1
Initialize a behavior function                                          // Section 2.3.2
while stopping criteria is not reached do
    Improve the behavior function by training on replay buffer          // Exploit; Section 2.3.3
    Sample exploratory commands based on replay buffer                  // Section 2.3.4
    Generate episodes using Algorithm 2 and add to replay buffer        // Explore; Section 2.3.5
    if evaluation required then
        Evaluate current agent using Algorithm 2                        // Section 2.3.6
    end if
end while
```

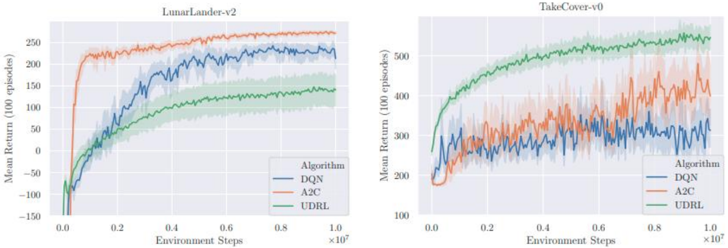
Upside-Down Reinforcement Learning

Algorithm 2 Generates an Episode using the Behavior Function.

Input: Initial command $c_0 = (d_0^r, d_0^h)$, Initial state s_0, Behavior function $B(;\theta)$
Output: Episode data E

$E \leftarrow \varnothing$
$t \leftarrow 0$
while episode is not over **do**
  Compute $P(a_t|s_t, c_t) = B(s_t, c_t; \theta)$
  Execute $a_t \sim P(a_t|s_t, c_t)$ to obtain reward r_t and next state s_{t+1} from the environment
  Append (s_t, a_t, r_t) to E
  $s_t \leftarrow s_{t+1}$ // Update state
  $d_t^r \leftarrow d_t^r - r_t$ // Update desired reward
  $d_t^h \leftarrow d_t^h - 1$ // Update desired horizon
  $c_t \leftarrow (d_t^r, d_t^h)$
  $t \leftarrow t + 1$
end while

Upside-Down Reinforcement Learning



(a) On LunarLander-v2, ⅂ꓤ is able to train agents that land the spacecraft, but is beaten by traditional RL algorithms.

(b) On TakeCover-v0, ⅂ꓤ is able to consistently yield high-performing agents, while outperforming DQN and A2C.

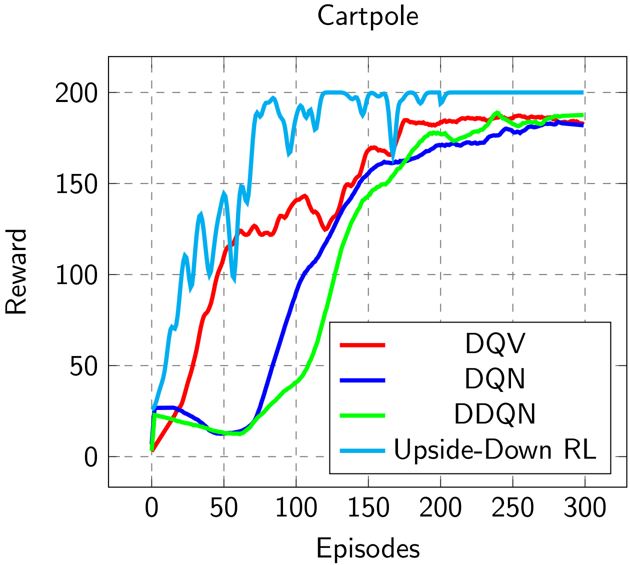
Figure 4: Evaluation results for LunarLander-v2 and TakeCover-v0. Solid lines represent the mean of evaluation scores over 20 runs using tuned hyperparameters and experiment seeds 1–20. Shaded regions represent 95% confidence intervals using 1000 bootstrap samples. Each evaluation score is a mean of 100 episode returns.

Personal Thoughts

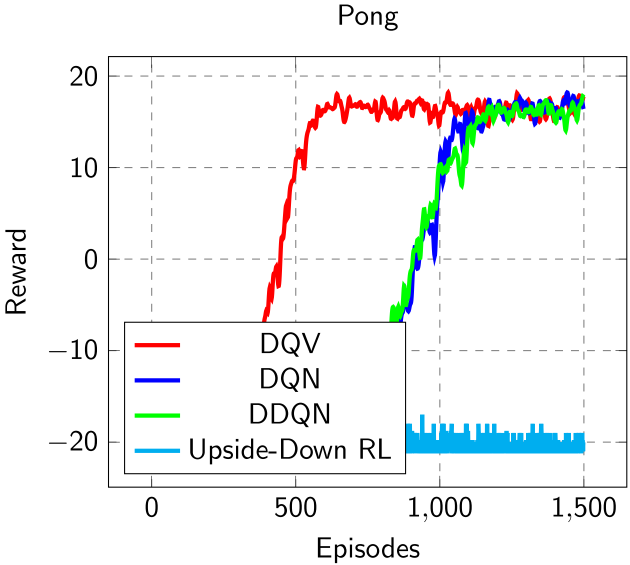
A critical analysis of Upside-Down RL ...

- I started with re-implementing the main ideas of both Upside-Down RL papers
- Is Upside-Down RL a potential breakthrough?
- What are the pros & cons compared to more common RL research?

Personal Thoughts



Personal Thoughts



Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- ✓ It is possible to train agents with the Upside-Down RL setting

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- ✓ It is possible to train agents with the Upside-Down RL setting
- ✓ When it works performance is better than traditional DRL methods

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- ✓ It is possible to train agents with the Upside-Down RL setting
- ✓ When it works performance is better than traditional DRL methods
- ✓ The algorithm allows for more efficient implementations than standard DRL methods

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- ✓ It is possible to train agents with the Upside-Down RL setting
- ✓ When it works performance is better than traditional DRL methods
- ✓ The algorithm allows for more efficient implementations than standard DRL methods
- ✓ For example the capacity of the memory buffer can significantly be reduced

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- × Requires networks with more parameters

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- × Requires networks with more parameters
- × It seems that it does not scale too more complex environments

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- × Requires networks with more parameters
- × It seems that it does not scale too more complex environments
- × Does not deal well with the exploration-exploitation trade-off

Personal Thoughts

Some personal insights about the experiments on the `Cartpole` environment

- × Requires networks with more parameters
- × It seems that it does not scale too more complex environments
- × Does not deal well with the exploration-exploitation trade-off
- × There are no theoretical guarantees: no value functions no Bellmann optimality equations neither

Personal Thoughts

To conclude ...

- Schmidhuber's claims are certainly well motivated

Personal Thoughts

To conclude ...

- Schmidhuber's claims are certainly well motivated
- We should rethink the way we are doing DRL

Personal Thoughts

To conclude ...

- Schmidhuber's claims are certainly well motivated
- We should rethink the way we are doing DRL
- The entire field does not need to fully start over, yet it is true that some concepts do need revision

Personal Thoughts

To conclude ...

- Schmidhuber's claims are certainly well motivated
- We should rethink the way we are doing DRL
- The entire field does not need to fully start over, yet it is true that some concepts do need revision
- Time will tell :)

References

- Bellman, Richard. "Dynamic programming." Science 153.3731 (1966): 34-37.
- Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.
- Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.
- Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 30. No. 1. 2016.
- Sabatelli, Matthia, et al. "The deep quality-value family of deep reinforcement learning algorithms." 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 2020.
- Schmidhuber, Juergen. "Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions." arXiv preprint arXiv:1912.02875 (2019).
- Srivastava, Rupesh Kumar, et al. "Training agents using upside-down reinforcement learning." arXiv preprint arXiv:1912.02877 (2019).