Reinforcement Learning Upside Down

: Don’t Predict Rewards – Just Map Them to Actions

+ Training Agents using Upside-Down Reinforcement Learning

By Juergen Schmidhuber, Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaskowski

Aug 8, 2023
Sungmin Yoon

Index

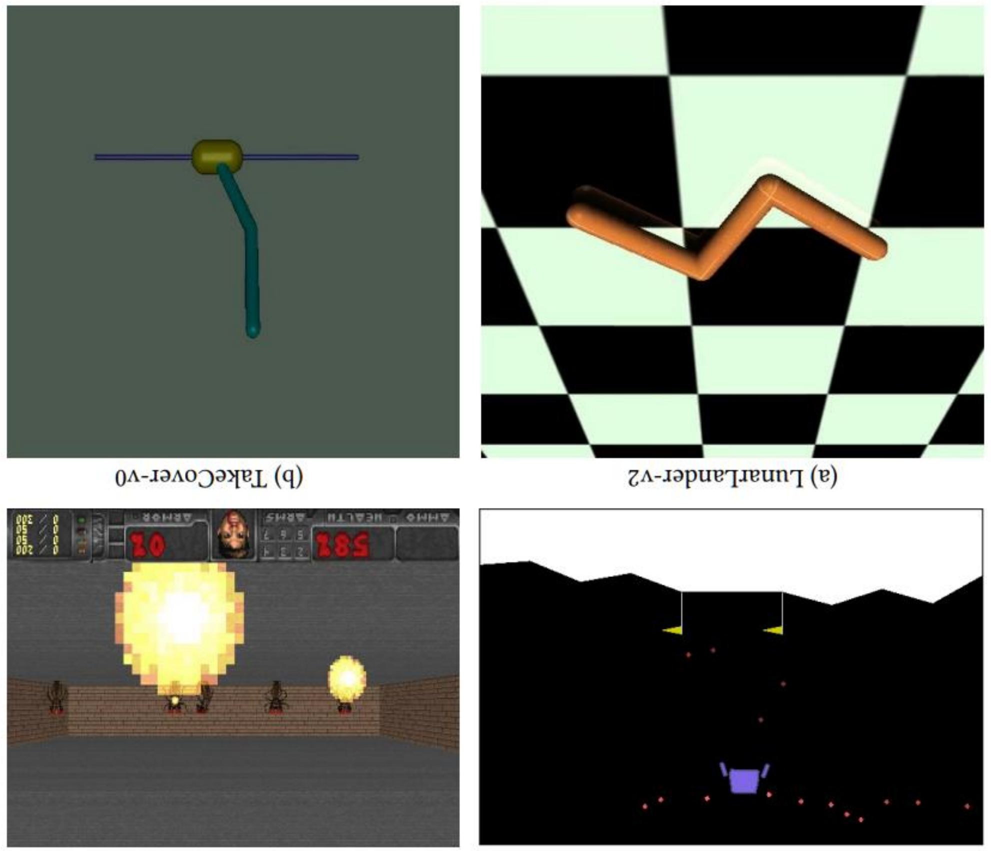
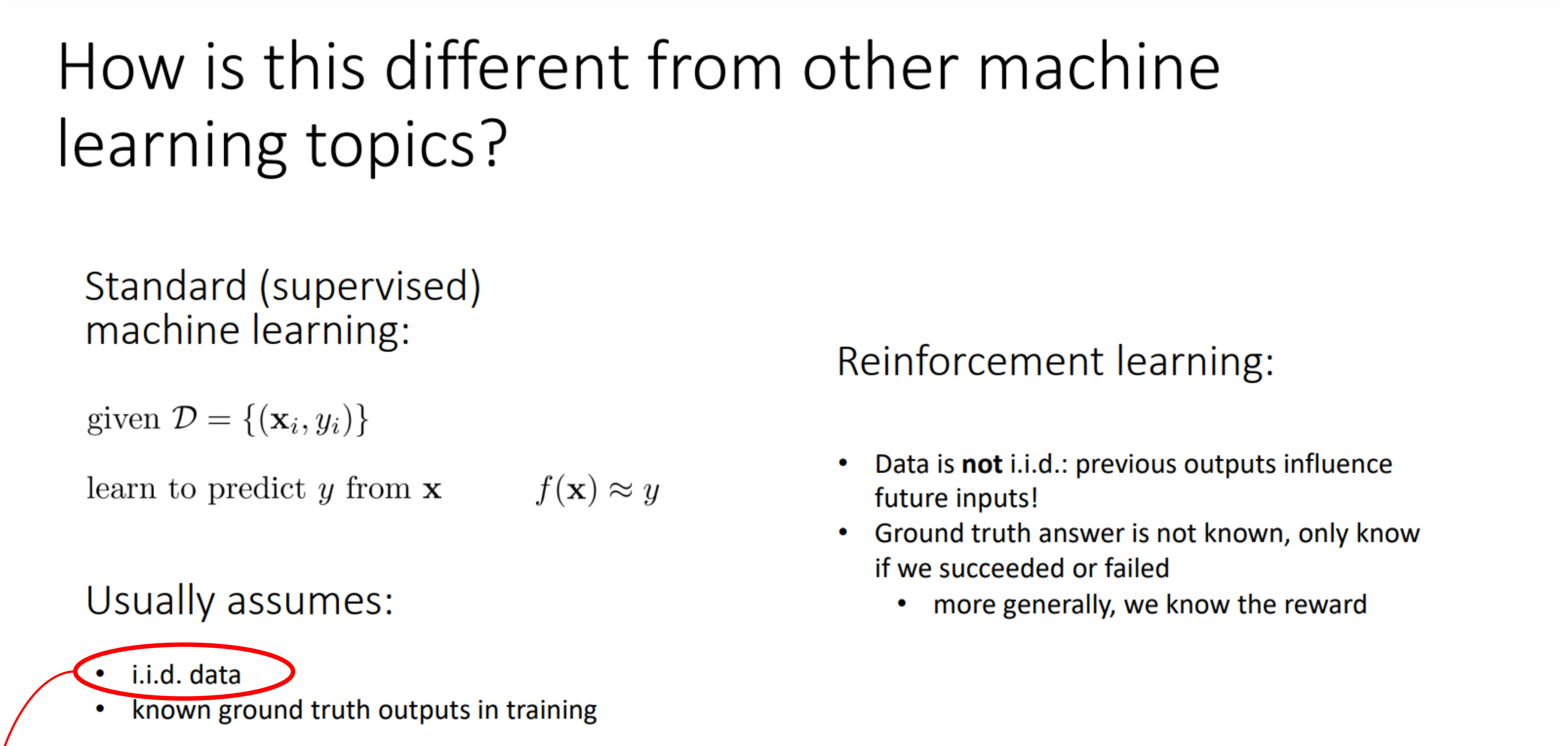


Figure 5. Test environments. In TakeCover-v0, the agent observes gray-scale visual inputs down-sampled to 64×64 resolution. In other environments, the observation is a set of state variables.

RL vs SL

- Different Data's characteristic



How is this different from other machine learning topics?

Standard (supervised) machine learning:

given $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$

learn to predict y from $\mathbf{x}$ $\quad f(\mathbf{x}) \approx y$

Usually assumes:

- i.i.d. data
- known ground truth outputs in training

Reinforcement learning:

- Data is **not** i.i.d.: previous outputs influence future inputs!
- Ground truth answer is not known, only know if we succeeded or failed
  - more generally, we know the reward

(Slide From CS285 (Sergey Levine) Lecture : https://rail.eecs.berkeley.edu/deeprlcourse-fa21/)

Q1. What about Sequence data?

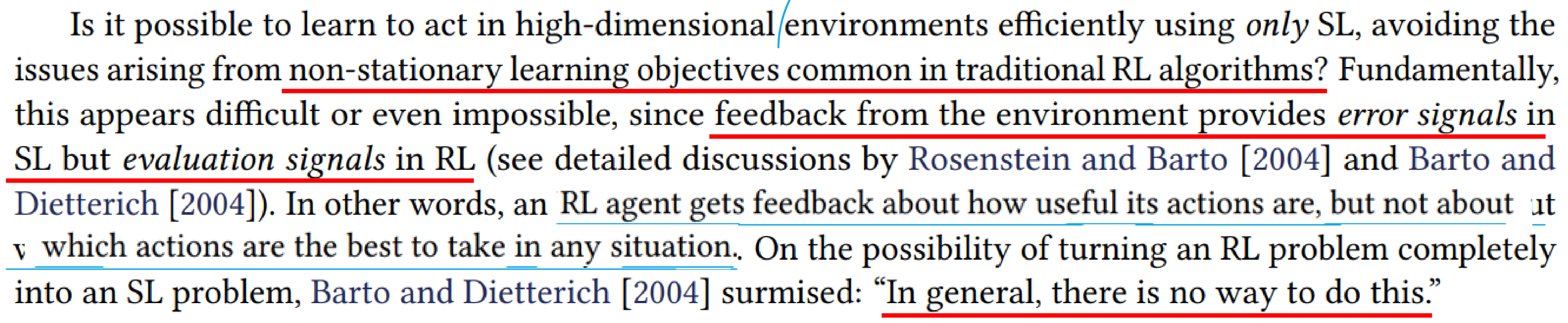
My thought : There's still difference (causality)

RL vs SL

- Different Data's characteristic

Reason to shift RL into SL

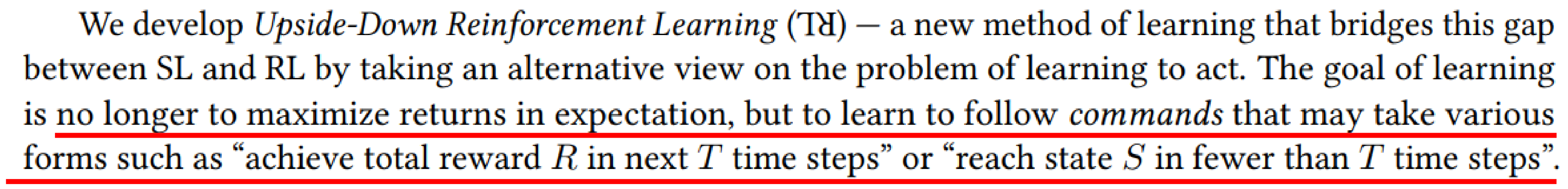
Introduction of RL Upside Down



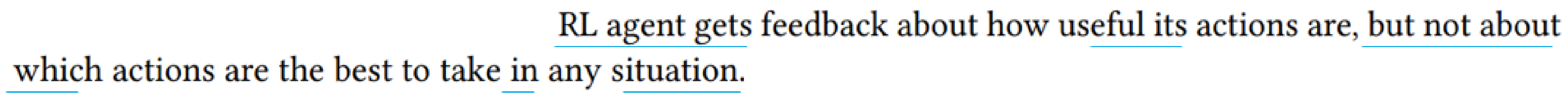
Is it possible to learn to act in high-dimensional environments efficiently using *only* SL, avoiding the issues arising from non-stationary learning objectives common in traditional RL algorithms? Fundamentally, this appears difficult or even impossible, since feedback from the environment provides *error signals* in SL but *evaluation signals* in RL (see detailed discussions by Rosenstein and Barto [2004] and Barto and Dietterich [2004]). In other words, an RL agent gets feedback about how useful its actions are, but not about ut v which actions are the best to take in any situation.. On the possibility of turning an RL problem completely into an SL problem, Barto and Dietterich [2004] surmised: "In general, there is no way to do this."

RL → SL

Trick : Command (Optimization Problem → Decision Problem like NP)



We develop *Upside-Down Reinforcement Learning* (⅂ꓤ) — a new method of learning that bridges this gap between SL and RL by taking an alternative view on the problem of learning to act. The goal of learning is no longer to maximize returns in expectation, but to learn to follow *commands* that may take various forms such as "achieve total reward R in next T time steps" or "reach state S in fewer than T time steps".



RL agent gets feedback about how useful its actions are, but not about which actions are the best to take in any situation.

Concept of ⅂ꓤ

S → A → R → S → A ...

S → R → A → S → R ...

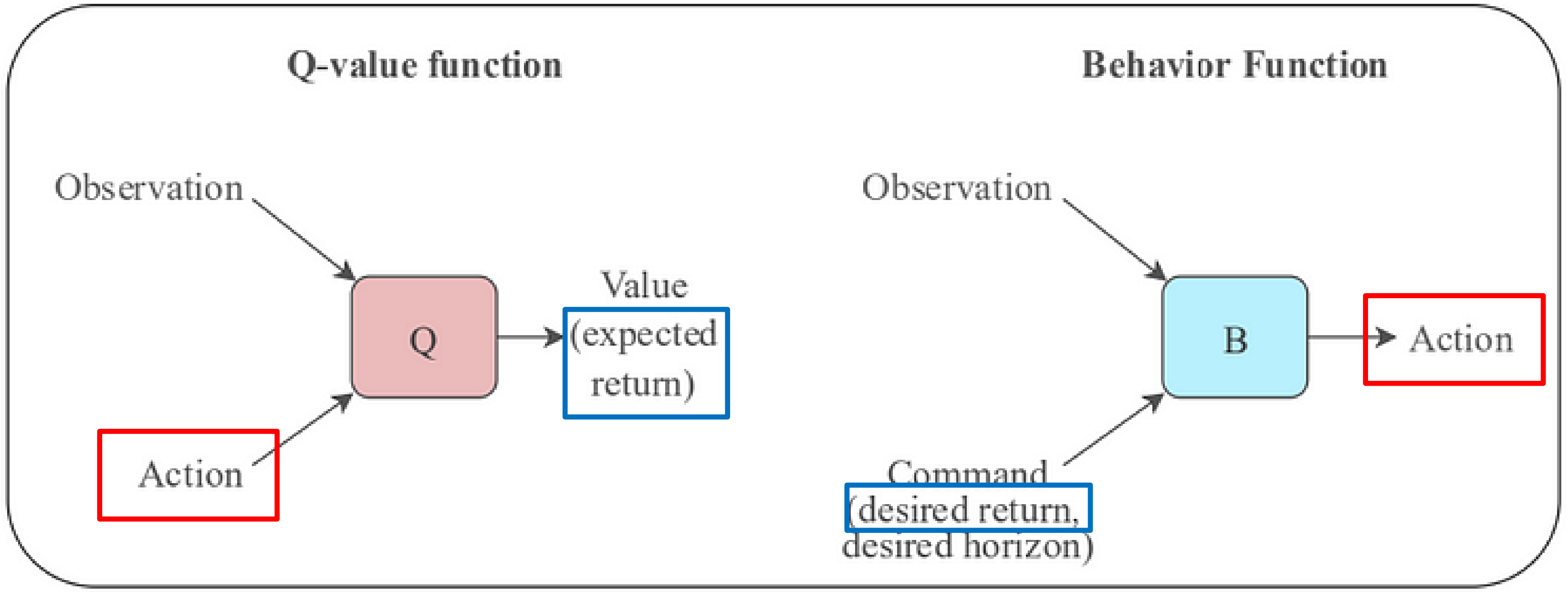


Figure 1. The action-value function (Q) in traditional RL predicts expected return over the long-term future while the behavior function (B) in ⅂ꓤ takes commands such as desired future return as input — conceptually the roles of actions and returns are switched. B directly produces a probability distribution over immediate actions. In addition, it may have other command inputs such as desired states or a desired time horizon.

Concept of ꓶꓤ

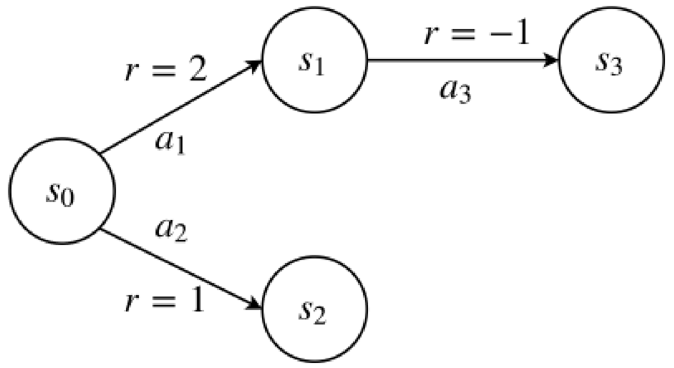


Figure 2: A toy environment with four states.

Table 1. A behavior function based on all unique trajectories for the toy environment (left image).

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

Notation of ꓶꓤ

Desired Return

Desired Horizon

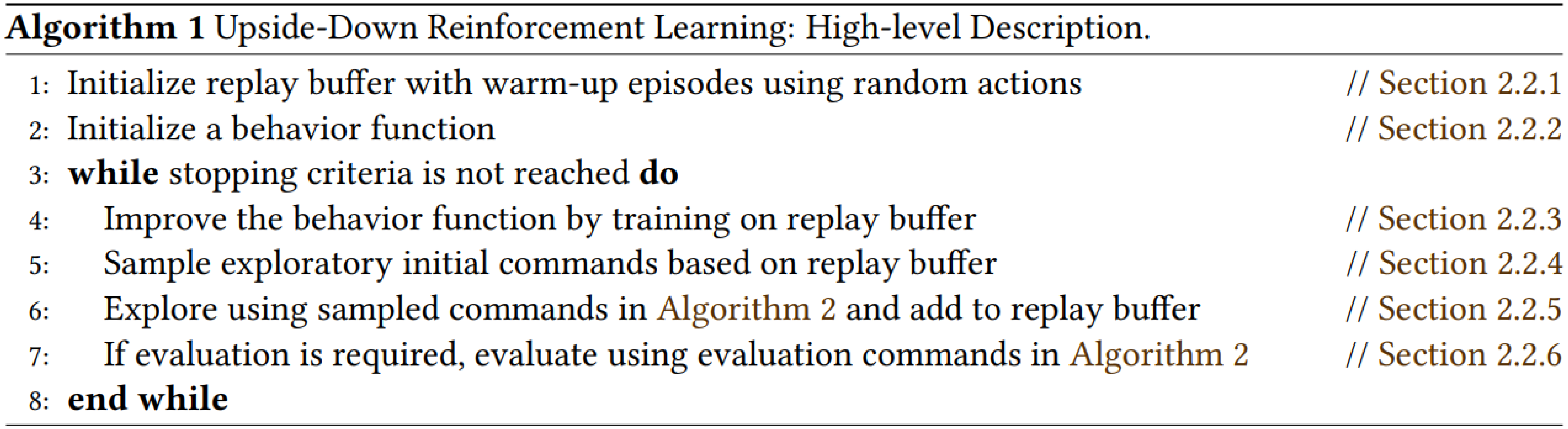
$$B_{\mathcal{T}} = \arg\min_{B} \sum_{(\tau, t_1, t_2)} L(B(a_{t_1}, s_{t_1}, d^r, d^h), a_{t_1}),$$

$$\text{where } 0 \leq t_1 < t_2 \leq \text{len}(\tau) \text{ for } \tau \in \mathcal{T}, d^r = \sum_{t=t_1}^{t_2} r_t \text{ and } d^h = t_2 - t_1.$$

Trajectory

$$\begin{aligned} B_\pi(a, s, d^r, d^h) &= P(A = a \mid R_{d^h} = d^r, S = s; \pi) \\ &= \frac{P(A = a, R_{d^h} = d^r, S = s; \pi)}{P(R_{d^h} = d^r, S = s; \pi)}. \end{aligned}$$

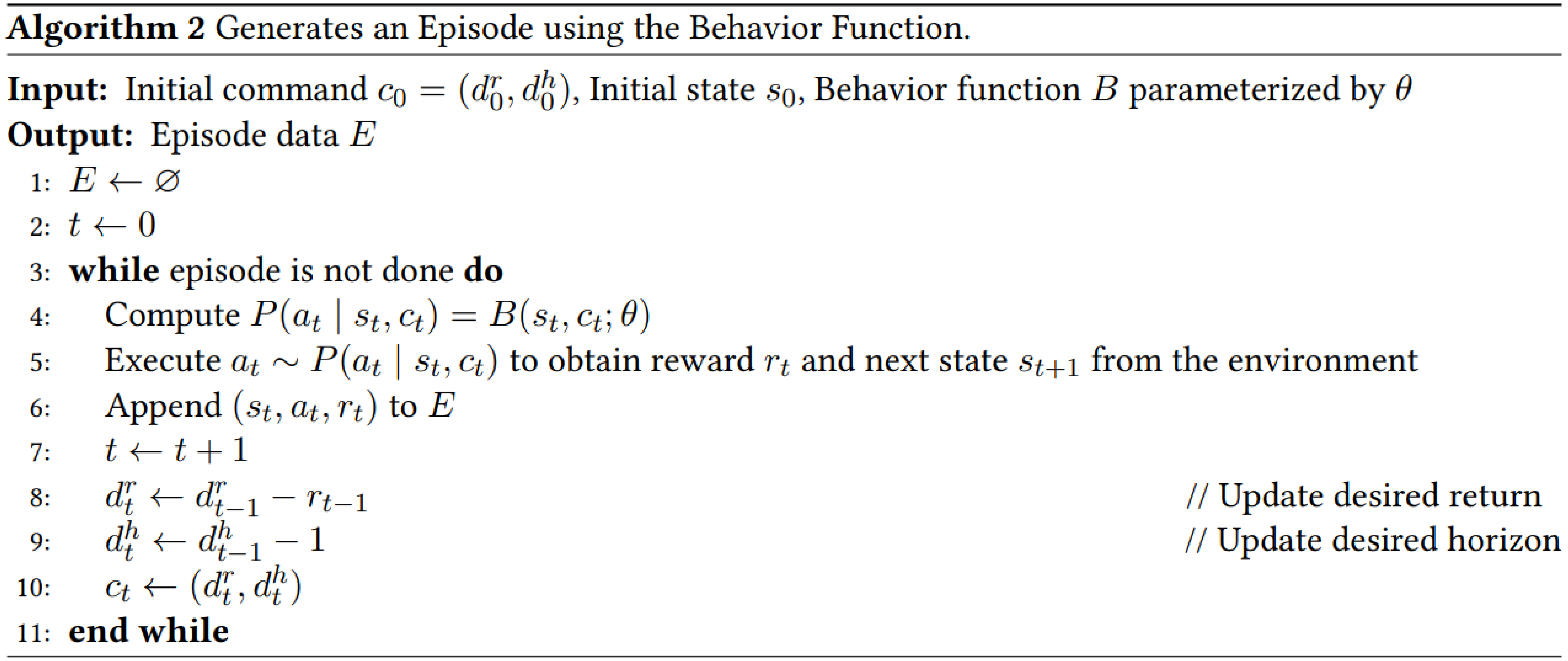
Concept of ⅂ꓤ



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

```
1: Initialize replay buffer with warm-up episodes using random actions            // Section 2.2.1
2: Initialize a behavior function                                                 // Section 2.2.2
3: while stopping criteria is not reached do
4:     Improve the behavior function by training on replay buffer                 // Section 2.2.3
5:     Sample exploratory initial commands based on replay buffer                 // Section 2.2.4
6:     Explore using sampled commands in Algorithm 2 and add to replay buffer     // Section 2.2.5
7:     If evaluation is required, evaluate using evaluation commands in Algorithm 2   // Section 2.2.6
8: end while
```

Concept of ⅂ꓤ



Algorithm 2 Generates an Episode using the Behavior Function.

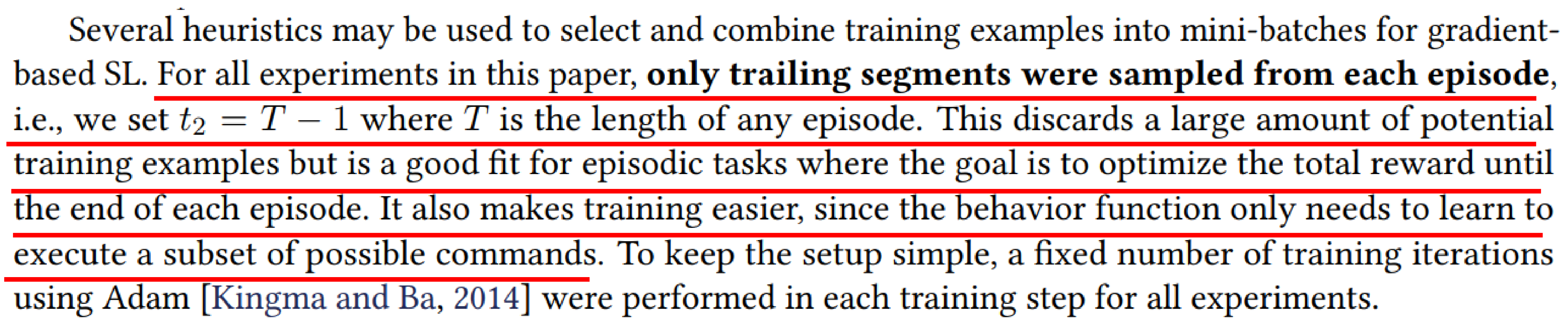
Input: Initial command $c_0 = (d_0^r, d_0^h)$, Initial state s_0, Behavior function B parameterized by θ

Output: Episode data E

```
1: $E \leftarrow \varnothing$
2: $t \leftarrow 0$
3: while episode is not done do
4:     Compute $P(a_t \mid s_t, c_t) = B(s_t, c_t; \theta)$
5:     Execute $a_t \sim P(a_t \mid s_t, c_t)$ to obtain reward $r_t$ and next state $s_{t+1}$ from the environment
6:     Append $(s_t, a_t, r_t)$ to $E$
7:     $t \leftarrow t + 1$
8:     $d_t^r \leftarrow d_{t-1}^r - r_{t-1}$        // Update desired return
9:     $d_t^h \leftarrow d_{t-1}^h - 1$        // Update desired horizon
10:    $c_t \leftarrow (d_t^r, d_t^h)$
11: end while
```

Concept of ⅂ꓤ : Detail

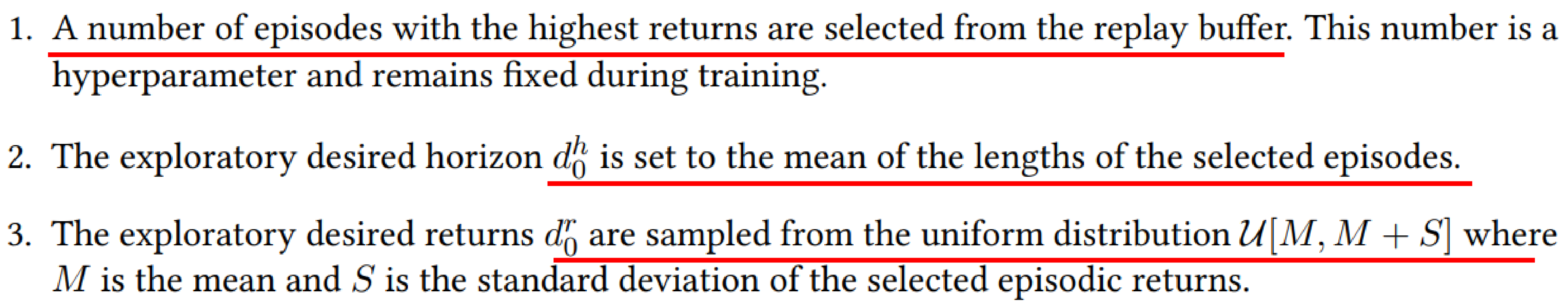
- **Training the Behavior Function B**



Several heuristics may be used to select and combine training examples into mini-batches for gradient-based SL. For all experiments in this paper, **only trailing segments were sampled from each episode**, i.e., we set $t_2 = T - 1$ where T is the length of any episode. This discards a large amount of potential training examples but is a good fit for episodic tasks where the goal is to optimize the total reward until the end of each episode. It also makes training easier, since the behavior function only needs to learn to execute a subset of possible commands. To keep the setup simple, a fixed number of training iterations using Adam [Kingma and Ba, 2014] were performed in each training step for all experiments.

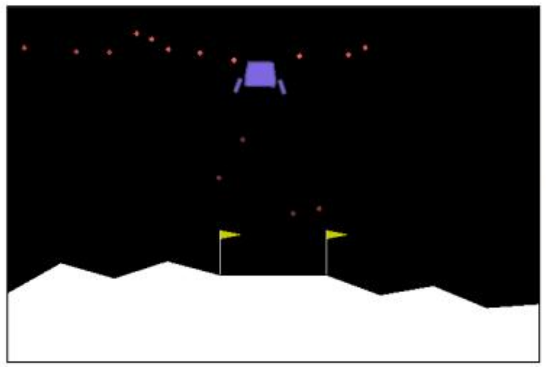
Concept of ⅂ꓤ : Detail

- **Sampling Exploratory Commands**



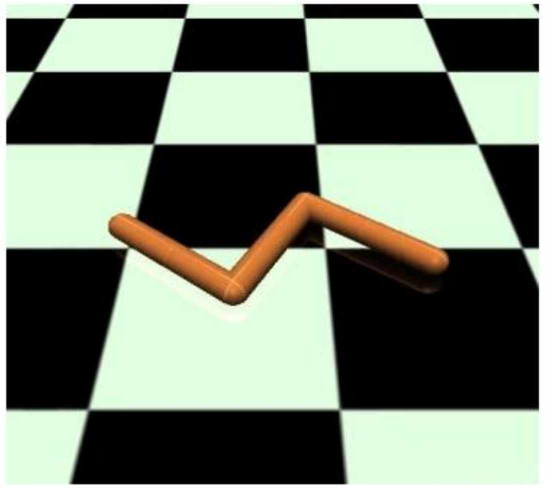
1. A number of episodes with the highest returns are selected from the replay buffer. This number is a hyperparameter and remains fixed during training.
2. The exploratory desired horizon d_0^h is set to the mean of the lengths of the selected episodes.
3. The exploratory desired returns d_0^r are sampled from the uniform distribution $\mathcal{U}[M, M + S]$ where M is the mean and S is the standard deviation of the selected episodic returns.

Experiment

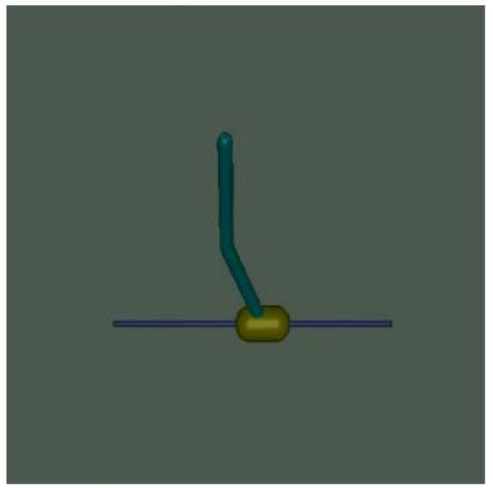


(a) LunarLander-v2

(b) TakeCover-v0



(c) Swimmer-v2

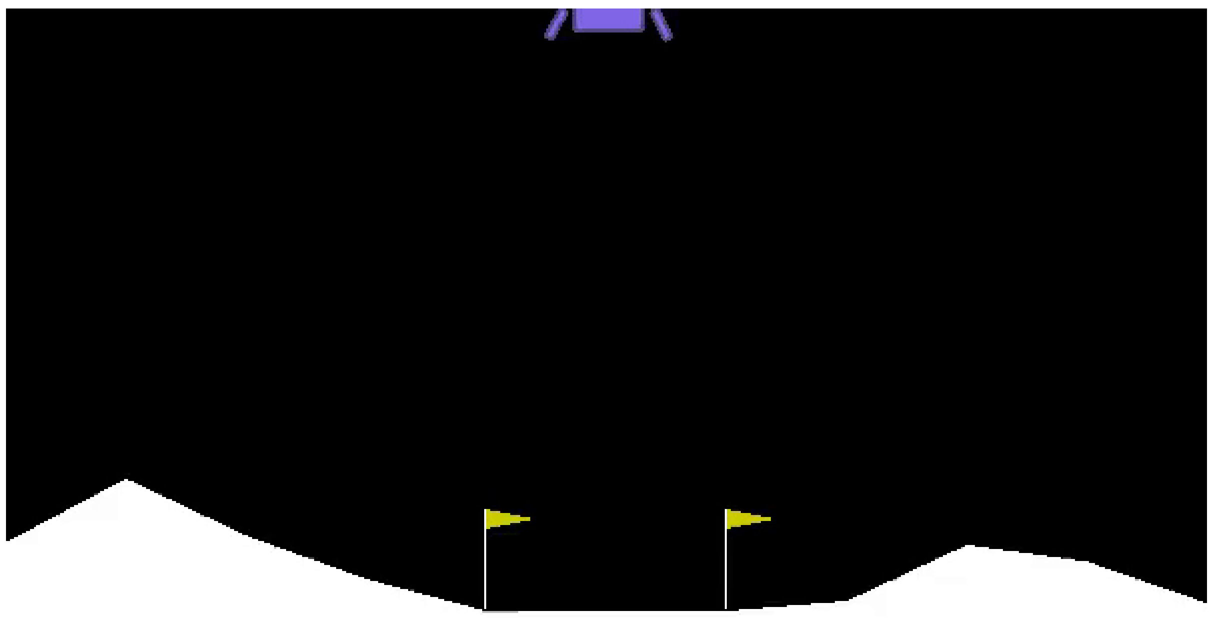


(d) InvertedDoublePendulum-v2

Figure 5. Test environments. In TakeCover-v0, the agent observes gray-scale visual inputs down-sampled to 64×64 resolution. In other environments, the observation is a set of state variables.

Experiment

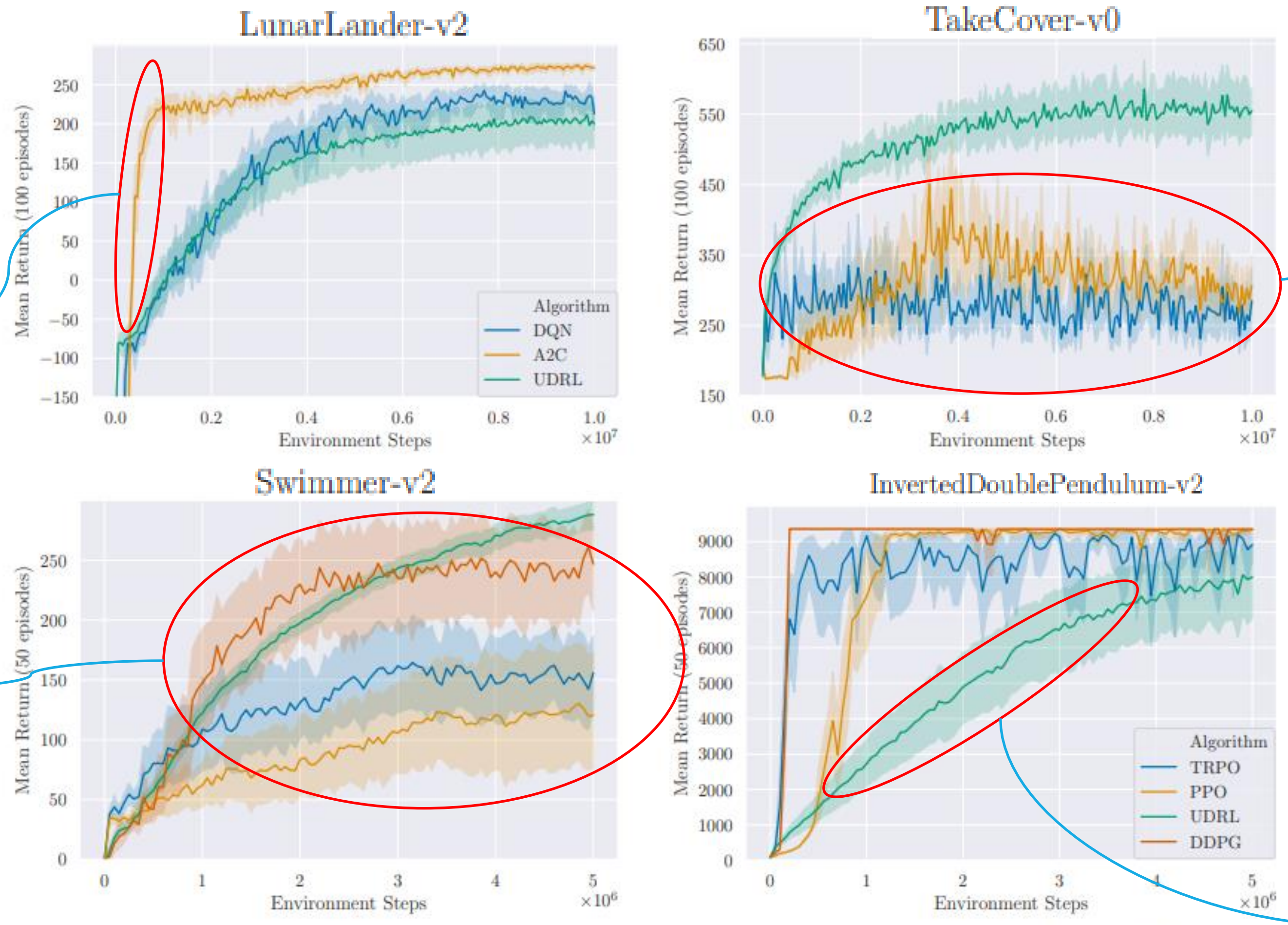
Before Training



Experiment

After Training

Result



Multi-step returns

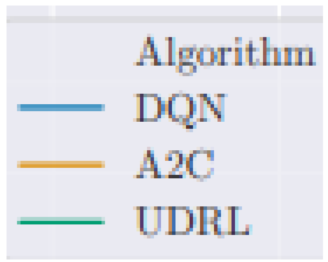
unstable

Fluctuating!

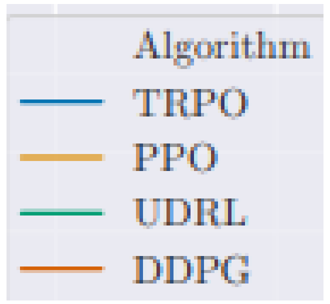
Poor performance…

Figure 3. Results on tasks with discrete-valued actions (top row) and continuous-valued actions (bottom row). 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. TЯ is competitive with or outperforms traditional baseline algorithms on all tasks except InvertedDoublePendulum-v2.

Discrete-valued actions



Continuous-valued actions



But it work better in challenging environment

Result

retained much of its performance in this challenging setting without modification

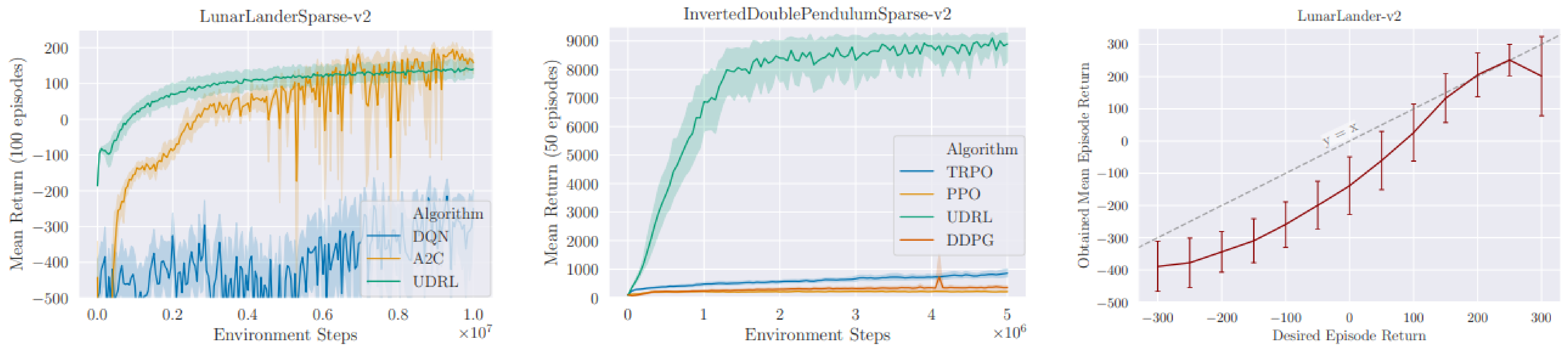
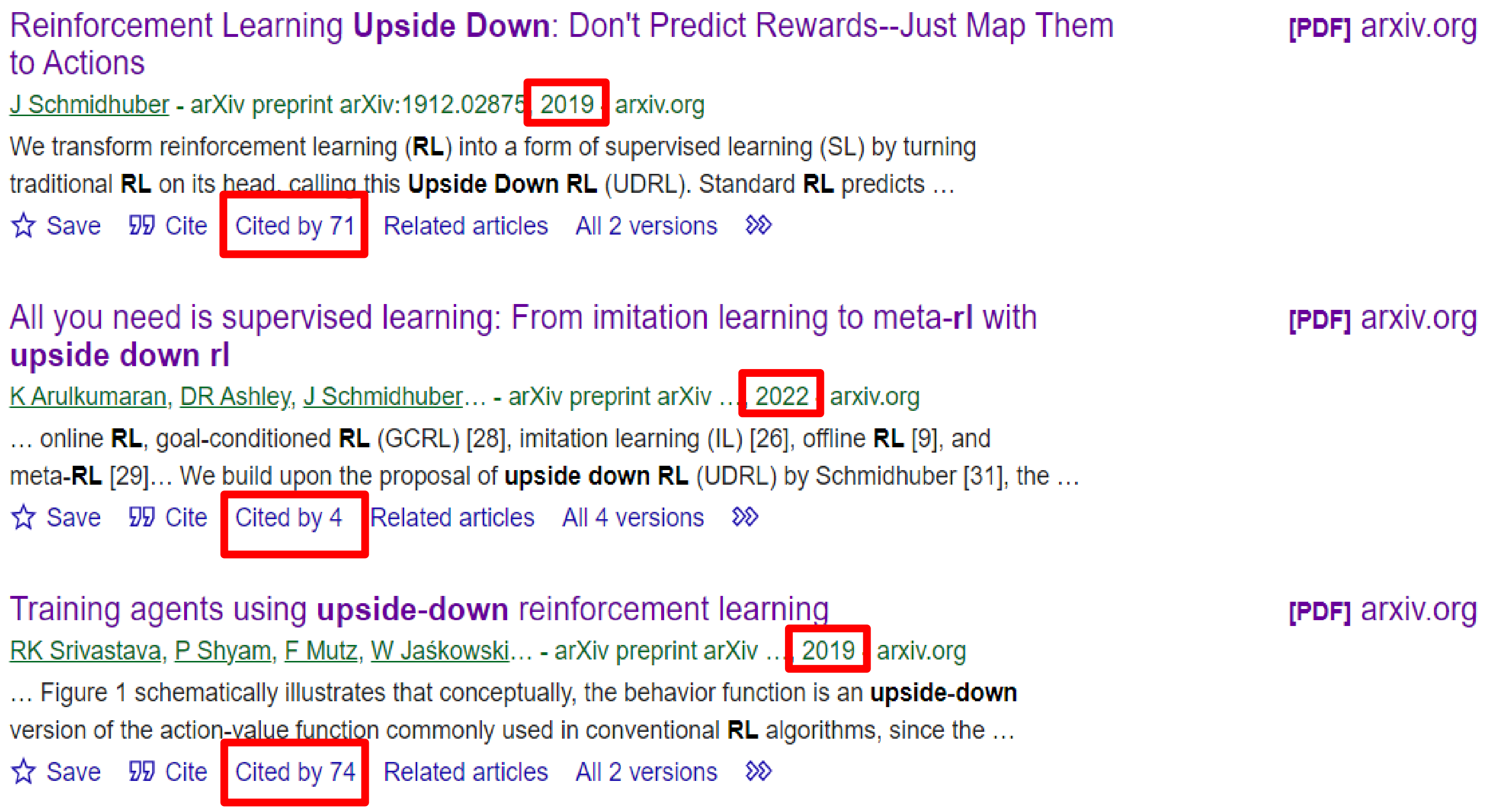


Figure 4. Left/Middle: Results on sparse delayed reward versions of benchmark tasks, with same semantics as Figure 3. A2C with 20-step returns was the only baseline to reach good performance on LunarLanderSparse-v2 (see main text). SwimmerSparse-v2 results are included in the appendix. **Right:** Desired vs. obtained returns from a trained ꓤL agent, showing ability to adjust behavior in response to commands.

Discussion : Why are these paper's citation so low?



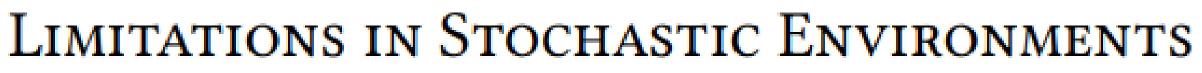
Reinforcement Learning **Upside Down**: Don't Predict Rewards--Just Map Them to Actions
[PDF] arxiv.org
J Schmidhuber - arXiv preprint arXiv:1912.02875, 2019 - arxiv.org
We transform reinforcement learning (**RL**) into a form of supervised learning (SL) by turning traditional **RL** on its head, calling this **Upside Down RL** (UDRL). Standard **RL** predicts …
Save Cite Cited by 71 Related articles All 2 versions

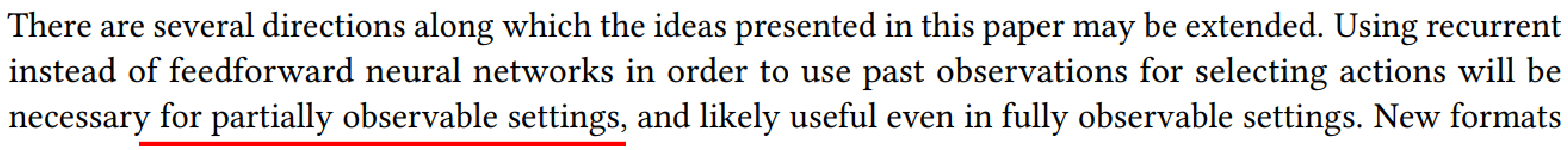
All you need is supervised learning: From imitation learning to meta-**rl** with **upside down rl**
[PDF] arxiv.org
K Arulkumaran, DR Ashley, J Schmidhuber… - arXiv preprint arXiv …, 2022 - arxiv.org
… online **RL**, goal-conditioned **RL** (GCRL) [28], imitation learning (IL) [26], offline **RL** [9], and meta-**RL** [29]… We build upon the proposal of **upside down RL** (UDRL) by Schmidhuber [31], the …
Save Cite Cited by 4 Related articles All 4 versions

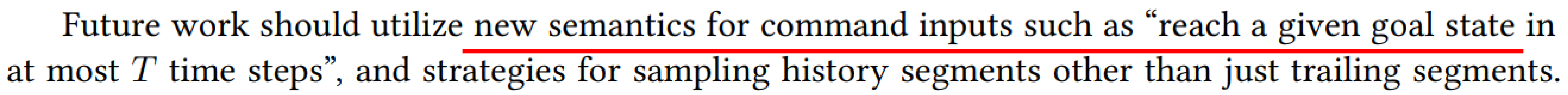
Training agents using **upside-down** reinforcement learning
[PDF] arxiv.org
RK Srivastava, P Shyam, F Mutz, W Jaśkowski… - arXiv preprint arXiv …, 2019 - arxiv.org
… Figure 1 schematically illustrates that conceptually, the behavior function is an **upside-down** version of the action-value function commonly used in conventional **RL** algorithms, since the …
Save Cite Cited by 74 Related articles All 2 versions

- Famous and powerful author in this field.
- Novelty : interesting
  - Bridge between SL and RL
- Good performance

Discussion : Future work


LIMITATIONS IN STOCHASTIC ENVIRONMENTS


There are several directions along which the ideas presented in this paper may be extended. Using recurrent instead of feedforward neural networks in order to use past observations for selecting actions will be necessary for partially observable settings, and likely useful even in fully observable settings. New formats


Future work should utilize new semantics for command inputs such as "reach a given goal state in at most T time steps", and strategies for sampling history segments other than just trailing segments.

Discussion : Further readings

- All You Need Is Supervised Learning : From Imitation Learning to Meta-RL With Upside Down RL
- Decision Transformer: Reinforcement Learning via Sequence Modeling
- Diversity Is All You Need : Learning Skills Without A Reward Function
- Offline Reinforcement Learning : Tutorial, Review, and Perspectives on Open Problem