A STUDY OF PRIMAL WASSERSTEIN IMITATION-LEARNING IN GOAL-CONDITIONED RL

SHENGDI CHEN[α]

Abstract. This project, conducted with Prof. Dr. J. Buhmann as supervisor and I. Ovinnikov as advisor, is executed within the framework of a Semester-Thesis under the catalogue identifier `401-3740-01L` at ETH Zürich.

RW/CSE, D-MATH, ETH Zürich: `shenchen␣AT␣ethz␣DOT␣ch | 17-943-358`.

Contents

Part 1. Preliminaries

1. Quick-Start

1.1. Synopsis. This thesis applies the Primal-Wasserstein Imitation-Learning algorithm as introduced in [Dad+20a] to a testing-framework utilizing a separate testing-environment

1.2. Related Researches. Key to the Primal-Wasserstein Imitation-Learning is the distance of its namesake, the Wasserstein distance, the p^{th}-order generalization of which is studied in [Vil08]. Generative Adversarial Networks (GANs) utilizing this distance is introduced in [ACB17].

1.2.1. *On Reinforcement-Learning.*

1.2.2. *On other Learning methodologies.* Behavioral Cloning

I

2. Technical Details

2.1. Programming. The source-code is packaged in one monolithic repository, which includes the testing-environment, training and testing modules for expert-

2.1.1. *General usages.* The programming is performed with the `Python` language with global type-hinting. Python versions no earlier than `Python 3.9` is required[1].

The source-code is consistently processed by the `black`-formatter [Wil+19] and thus implicitly conforms to the `PEP8` standard.

2.1.2. *Required packages.* The framework for the agent-interactive environment is provided by the `gym`-package developed by OpenAI [Bro+16].

Implementation of the Proximal-Policy-Optimization (PPO) algorithm [Sch+17] as provided by stable-baselines3 [Hil+18] (`sb3`), is used as the direct RL algorithms.

2.2. Formatting and Typesetting. This paper is compiled by the LuaTEX typesetting-engine featuring the open-source font-packages `Libertinus` and `Source-Code-Pro`.

2.3. Version-Tracking. The project source-code is distributed under the following repositories:

- The modules for the formulation of various agents, as well as those for training and testing are published under the repository: Link to GitHub;
- All other aspects of the Tetris-game, including the `engine` as described in Section ??, are found at: Link to GitHub;
- This thesis-document is accessible under: Link to GitHub.

Both source-code repositories deploy the »GNU Affero General Public License v3.0« license, else known as `GNU AGPLv3`.

This thesis documents the results obtained until version `V?`, marked with the signed `git-tag` of `#tag-number`.

[1]Version requirement of `Python`'s `typing`-Module: https://docs.python.org/3/library/typing.html#module-contents

Part 2. PWIL Fundamentals

3. Essentials from Primal Wasserstein IL

3.1. The RL problem. The Markov-Decision-Process (MDP) framework as introduced in [REF] characterizes agent-environment interaction over time-horizon $t \in T$ with action a_t drawn from the time-invariant action-space A leading to state transition:

$$S \in s_t \xrightarrow[\text{env}]{a_t} s_{t+1} \in S$$

of the environment, yielding a reward r_t.

The decision for the action a_t of an agent is provided by policy π of policy-space Π:

$$s_\star \xrightarrow{\pi} a_t$$

Central to Reinforcement-Learning (RL) is the search of the optimal policy $\pi^\star$ that maximizes the expected γ-discounted summation of all rewards r, otherwise referred to as the expected »return«.

3.2. The Wasserstein Distance. As discussed in [Vil08], the p^{th}-order Wasserstein[2] distance is defined as follows:

$$fff$$

$$\equiv \inf_{\theta \in \Theta} \left\{ \int \right\}$$

$$\mathcal{W} := \left(\inf_{\theta \in \Theta} \left\{ \mathop{\mathbb{E}}_{(x,y)} \left[d\left(x, y\right) \right] \right\} \right)^{1/p}$$

which, interpreted with the earth's movers analogy (Villani-2008), is equivalently formulated as follows:

$$\pi^\star := \inf_{\theta \in \Theta} \left\{ d\left(\right) \theta_\pi \right\}$$

[Dad+20a] enumerates various desirable properties of the Wasserstein distance: in particular, it is a true distance (in contrast with f-divergences used commonly in ?-Learning (IRL or adversary?)) with guaranteed smoothness.

3.3. The PW-IL algorithm. Solving for the optimal solution conforming to the above definition requires trajectory information of the entire episode (before the environment has to be reset for further agent inputs). This requirement is bypassed by the Primal Wasserstein IL (PW-IL) algorithm in [Dad+20a], where the following optimization problem is solved instead:

$$\pi^\star := \inf_{\theta \in \Theta} \left\{ d\left(\right) \theta_\pi \right\}$$

The optimizer is deliberately non-optimal with respect to the original Wasserstein distance: in fact, the underlying optimization-problem guarantees an upper-bound for the Wasserstein distance.

The non-optimal solution is based on the following »greedy-coupling«

[2]named after Leonid Vaseršteĭn (in Russian: Леонид Нисонович Васерштейн)

Part 3. **Configurations**

4. The environment

A separate testing-environment is implemented where the agent is required to navigate from its starting position within a two-dimensional plane to reach target(s) unknown to the agent itself. This section introduces the »`PointNav`« environment and discusses its configuration options for the purpose of studying the PW-IL algorithm.

4.1. **Introduction to `PointNav`.**

4.1.1. *Physical components.*

Definition 1. The board

The two-dimensional plane containing the agent and the target(s) is referred to as the »board«. Without loss of generality, its size is chosen as 200×200 and internally represented by an `np.ndarray` of the same shape.

Definition 2. The agent

For this thesis, the starting position of the agent is fixed at $(10, 10)$.

Random initialization, readily configurable through a boolean-flag, is left for future work.

Definition 3. The two targets and the ideal path

In this study, two targets of non-random positions are used, with the first located at the board center $(100; 100)$ and the second at $(190; 190)$. Once the first target is reached, the »current« target is to the second, which, if also found, indicates successful solving of `PointNav`. See Section 4.1.3 for further discussion of termination-criteria.

Given the starting position of the agent at $(10; 10)$, the ideal path thus traverses diagonally across the board: from the starting position in the lower, left corner to first reach the first target in the middle, followed by moving towards second target in the upper, right corner at $(190; 190)$.

4.1.2. *Reward.* The reward at time-step t at is calculated as the negative L^2 distance between the position of the agent and that of the current target:

$$\text{reward}_t^{\texttt{PointNav}} := -\left(\left(x_t^{\text{agent}} - x_t^{\text{curr-target}}\right)^2 + \left(y_t^{\text{agent}} - y_t^{\text{curr-target}}\right)^2\right)^{1/2}$$

Implicitly, the reward-value is positive, with higher values indicating better performance.

4.1.3. *Termination.* An episode is terminated if both targets have been found or if the internal timer of 1000 time-steps has elapsed. Justification for this specific choice of 1000 time-steps as the maximal episode-length is provided below in Note 2.

4.2. **The `PointNav` variants.**

4.2.1. *Target shifts.* For controlled modifications of the environment `PointNav`, the first target at $(100; 100)$ is shifted off-center. In this project, two such target-shift variations are made. Table 4.2 documents all three variants: the default without target-shift is marked blue.

Variant	`PointNav-ID`	Shift of First Target	Position of First Target
Default	0	$(0;\ 0)$	$(100;\ 100)$
`Variation-1`	1	$(0;\ +50)$	$(100;\ 150)$
`Variation-2`	2	$(+50;\ 0)$	$(150;\ 100)$

TABLE 4.2. The three different

Note 1. The actual target

Note that the default variant marked blue in Table 4.2 is the (only) environment that the PW-IL algorithm is trained on and evaluated against. One also observes that Variation-1 and Variation-2 (marked blue) demonstrate symmetry with respect to the shifts.

4.2.2. *Action-Continuity.* Two modes of continuity for action-input are available to agents interacting with `PointNav`: the discrete and the continuous action-space:

Definition 4. Discrete `PointNav`

In the discrete action-mode, 5 possible actions are available to the agent, which are mapped to two-dimensional movement-inputs as follows:

$$A^{\text{discrete}} := \begin{bmatrix} \texttt{gym.spaces.Discrete(5)} \\ \triangleq \begin{bmatrix} 0\ 1\ 2\ 3\ 4 \end{bmatrix}^{\top} \end{bmatrix} \longrightarrow \begin{bmatrix} (0;+2) \\ (0;-2) \\ (+2;0) \\ (-2;0) \\ (0;0) \end{bmatrix}$$

Definition 5. Continuous `PointNav`

For the continuous variant, the action-space spans the range of $[-2.5;+2.5]$ for both dimensions. The x and y inputs are then rounded[3] before applying to the agent's position.

$$A^{\text{cont}} := \begin{bmatrix} \texttt{gym.spaces.Box(-2.5, +2.5)} \\ \texttt{gym.spaces.Box(-2.5, +2.5)} \end{bmatrix} \longrightarrow \texttt{np.round(}\star\texttt{)}$$

Note that the range of the action-space is chosen such that the final, rounded action applied to `PointNav` is uniformly distributed among $\{-2,-1,0,+1,+2\}$ for the random agent, analogous to the discrete counterpart.

Note 2. Episode-Length

Given the maximal (absolute) displacement of 2 in one direction every step and the size of 200×200 of the board, one readily sees that, for the purpose of navigating along the diagonal of the board, the ideal agent would require ~ 200 steps (~ 100 per dimension) to reach both targets in the discrete `PointNav` environment and ~ 100 steps (movement in both dimensions at the same time) in the continuous version, irrespective of the shift values of the first target.

[3]The documented behavior of `np.round()` rounds ± 2.5 to ± 2: exactly desired behavior.

Continuity	`PointNav`-ID	Target-Shift
Discrete	0	$(0;\ 0)$
	1	$(0;\ +50)$
	2	$(+50;\ 0)$
Continuous	0	$(0;\ 0)$
	1	$(0;\ +50)$
	2	$(+50;\ 0)$

Table 4.3. Performance of the experts with respect to the two metrics m^{Length} (lower is better) and m^{Reward} (higher is better), shown as $(\text{avg} \pm \sigma)$.

The episode-length of 1000 time-steps as mentioned in Section 4.1.3 allows a considerable margin of tolerance for non-optimal behavior while guaranteeing baseline performance, as later discussed in Definition 8.

Note 3. Practical duration of target reaching

The careful reader would protest that a stricter minimal number-of-steps to retrieve the positions of the targets could be provided. Indeed, as introduced previously, the agent's position is initialized to $(10; 10)$ and that of the second agent to $(190; 190)$, thus not the entire distance of the diagonal need be traversed to conclude an episode.

Also, for the purpose of visualization, the implementation of `PointNav` attaches an icon-image to the agent and the targets. The target-reaching criteria is such that the overlapping of the outer edge of these images signals attainment of the current target. Effectively, this further reduces the minimal number of steps required to reach the targets.

The interested reader is referred to the source-code repository to inspect the implementation details and work out the actual best-case episode-length. For the sake of providing an estimate for the ideal model and defining the baseline of 1000 steps per episode, the approximation provided in Note 2 is sufficient.

4.2.3. *The* 6 *variants.*

5. Expert-Demonstration Setup

5.1. **The expert-demonstrations.** As setup for deploying the PW-IL method, the »demonstration« from the expert-agent must first be generated. This section defines such demonstrations and describe the hyper-parameters available to the process of the generation.

Definition 6. Episode-Trajectories of an agent

An »episode-trajectory«, or simply »trajectory«, is an iterable collection of per-time-step recording of all state-action pairs generated by an agent during one entire episode, where such a state-action pair s-a is the concatenation of the environments state s and the agent's action a.

$$\texttt{traj} := \left(\texttt{sa}_1, \text{s-a}_2, \cdots \text{s-a}_{\text{episode-over}}\right)$$

Per [Dad+20a], the »demonstration« for PW-IL transports information from expert trajectories.

Definition 7. Demonstration for PW-IL

For practical implementation, [Dad+20b] encodes this as an iterable collection of state-action pairs, as previously seen in the definition of the episode-trajectories.

The above definition hints at the usage of multiple expert-trajectories to generate PW-IL's demonstration. Indeed, the number of trajectories forming the pool for the demonstration constitutes a configurable hyper-parameter $h^{\#\text{-traj}}$. Inspired by the choice of 1 and 10 as values of $h^{\#\text{-traj}}$ in the original paper of PW-IL [Dad+20a], this thesis expands upon this with the addition of an intermediate value 5:

$$h^{\#\text{-traj}} \in \{1, 5, 10\}$$

For each of these trajectories, sub-sampling at frequency $h^{\text{s-smp}}$ is performed by selecting state-action pairs once every certain number of time-steps. Assume thus that within a selected episode-trajectory, the state-action pair at the k^{th} time-step is chosen, the next state-action to pick is the one at the $(k + h^{\text{s-smp}})^{\text{th}}$ time-step.

While the original thesis [Dad+20a] uses the uniform sub-sampling frequency of 20 to simulate data scarcity, this study deploys the following value-range

$$h^{\text{s-smp}} \in \{1, 2, 5, 10, 20\}$$

The sub-sampled state-action pairs of all $h^{\#\text{-traj}}$ are finally concatenated to form the demonstration as input for executing the PW-IL algorithm.

5.2. **The source.**

Besides the number of trajectories $h^{\#\text{-traj}}$ and the sub-sampling frequency $h^{\text{s-smp}}$, the quality of the demonstration itself is configurable, as also studied in [Bro+19]; [Jac+19]. In this thesis, the quality of the expert-demonstration is controlled by adding experts to the trajectory pool trained on *variations* of the `PointNav`. Allowing the presence of such variation trajectories in the pool constitutes the non-optimal demonstration quality. One further distinguishes between the »mixed« pool, where expert-trajectory on the default, non-shifted variation-0 is included and the »distant« pool, where only the shifted variations are taken into consideration.

By subsequently varying the specific constellation within the optimal, the mixed and non-optimal pools, one observes the 7 demonstration-sources, identified with the hyper-parameter h^{id} of integer-values 0 to 6 defined in Table 5.2.

5.3. **Summary of hyper-parameters.**

Conclusion 1. Configuration-Space of PW-IL

Demo-ID h^{id}	Traj-Pool of Experts ID	Demo-Quality
0	$\{0\}$	optimal
1	$\{0,1\}$	non-optimal: Mixed
2	$\{0,2\}$	
3	$\{0,1,2\}$	
4	$\{1\}$	non-optimal: distant
5	$\{2\}$	
6	$\{1,2\}$	

TABLE 5.2. Time-stamps of key activities

To conclude this chapter, the three configurable hyper-parameters of PW-IL as discussed above are presented as follows:

$$h^{\texttt{PW-IL}} := \begin{bmatrix} h^{\text{id}} \\ h^{\text{\#-traj}} \\ h^{\text{s-smp}} \end{bmatrix} \in \begin{bmatrix} \{0,1,\cdots,6\} \\ \{1,5,10\} \\ \{1,2,5,10,20\} \end{bmatrix}$$

With the distinct and continuous variants of the action-space, a total of

$$\begin{aligned} 2 \times |h| &:= 2 \times \left(|h^{\text{source}}| \times \left|h^{\text{\#-traj}}\right| \times h^{\text{s-smp}}\right) \\ &:= 2 \times (7 \times 3 \times 5) \\ &\equiv 210 \end{aligned}$$

The upcoming sections analyze the performance of the PW-IL algorithm under varying configurations of the

Assessment	Ep-Length m^{Length}	Reward m^{Reward}	Assessment
insufficient	> 1000	$< -1.2\,\text{e}5$	insufficient
baseline	< 1000	$< -7.0\,\text{e}4$	»meaningful«

TABLE 6.1. tab: The two performance measures for agents interacting with `PointNav`: the episode-length m^{Length} sets the actual baseline, while the reward m^{Reward} presents an alternative unofficial metric for casual inspection of training effect.

Part 4. Evaluation and Comparison

6. Analysis

6.1. **Evaluation Framework.**

The following *a priori* performance-metrics are used to evaluate the performance of an agent interacting with the testing-environment `PointNav`.

Definition 8. Performance metrics and baselines

The baseline is defined as reaching both targets before episode terminates, i.e., the episode-length $m^{\text{Length}} \leq 1000$, where lower m^{Length} values indicate less time required to attain the target positions, thus better performance. See Note 2 for justification of the baseline episode-length value of 1000.

Alternatively, the reward m^{Reward} returned by the environment is used to gauge the agent's performance. However, as the reward of `pointnav` is calculated as the negative L^2 distance of the agent to the target (see Section 4.1.2), achieving higher reward value does not necessarily indicate actually reaching the targets: an agent »hovering« around the target in close proximity will effectively reduce the L^2 distance close to 0, thus leading to high reward-values. Thus, instead of a baseline, the reward m^{Reward} constitutes a loose guideline for training effectiveness.

Empirical observations suggest that a random agent displays episode-reward $m^{\text{Reward}} \lessapprox -1.2\,\text{e}5$, whereas signs of »meaningful« training are found for reward-values greater than $-7.0\,\text{e}4$.

Table 6.1 summarizes the value thresholds for different performance assessments of an agent with respect to the two metrics nominated above.

In the following sections, evaluations of an agent are performed on the playback information of 10 episodes of interacting with the `PointNav` environment.

6.2. **The experts.**

Using the PPO-algorithm as introduced in [Sch+17] and implemented by the reference RL-library `sb3` [Hil+18], 6 expert-agents are trained: 3 on the discrete-action `PointNav` environment, 3 on the continuous counterpart. It shall be noted that other algorithms from RL, such as A2C [Mni+16] or DQN [Mni+15] can be utilized.

The experts are trained with a largely sufficient step-count of 1 million. After every 10 thousand steps, reward-evaluation on the respective, optionally shifted `PointNav` environment variant is performed. The model yielding the best evaluation-reward respectively is saved as the final resultant agent when the training period has elapsed.

Continuity	`PointNav`-ID	Ep-Length m^{Length}	Reward m^{Reward}
Discrete	0	239.1 ± 44.73	$-1.284\,\text{e}4 \pm 573.6$
	1	257.8 ± 22.86	$-1.427\,\text{e}4 \pm 526.3$
	2	406.1 ± 75.37	$-1.918\,\text{e}4 \pm 2016$
Continuous	0	88.20 ± 0.400	$-5.221\,\text{e}3 \pm 329.3$
	1	244.2 ± 57.40	$-1.194\,\text{e}4 \pm 498.2$
	2	188.3 ± 38.26	$-9.065\,\text{e}3 \pm 321.5$

TABLE 6.2. Performance of the experts with respect to the two metrics m^{Length} (lower is better) and m^{Reward} (higher is better), shown as $(\text{avg} \pm \sigma)$.

6.2.1. *Evaluations.* Data generated by the final 6 expert agents is collected. As described above, the average and the standard-deviation across 10 episodes with respect to the two performance-metrics m^{Length} and m^{Reward} are then calculated and enlisted in Table 6.2.

Solution 1. Analysis of Expert performance on `PointNav`

Per the assessment criteria nominated in Definition 8, the experts evidently solve the `PointNav` problem in all 3 configurations of both the discrete and the continuous version.

Further more, despite being ultimately discretized with rounding of the action, the continuous `PointNav` is solved more robustly than the discrete version. One observes, in particular, the Variation-2 of the discrete model demonstrates high standard-deviation with respect to episode-length m^{Length}.

It is also evident that the experts solve the default variant of both the continuous and discrete environment better than the shifted ones respectively.

Solution 2. Visualization of a `PointNav` agent

Figure 6.1 provides various graphs for studying the an agent interacting with `PointNav`. Here, the expert-agent of the continuous, Variation-0 `PointNav` as seen in Table 6.2 is shown.

- The two plots on the left demonstrate the agent's position across the 10 sampled episodes. One observes near perfect performance of the Expert-agent, as hardly any deviation is visible from the optimal, diagonal route. The histogram also depicts the ideal, close to uniform distribution across the trajectory, indicating little hovering around any particular location.
- The upper-right plot provide the locations of the two targets as described previously. The first target is unsurprisingly located at the non-shifted center of $(100; 100)$.
- The lower-right plot delineates the agent's action. One observes the high concentration around the maximal allowed value of 2.5 for both dimensions, indicating fast movement in both directions across the board.

The analysis shows desirable decision-making from this particular Expert-agent: no wondering its m^{Length}-value is near optimal (88.20).

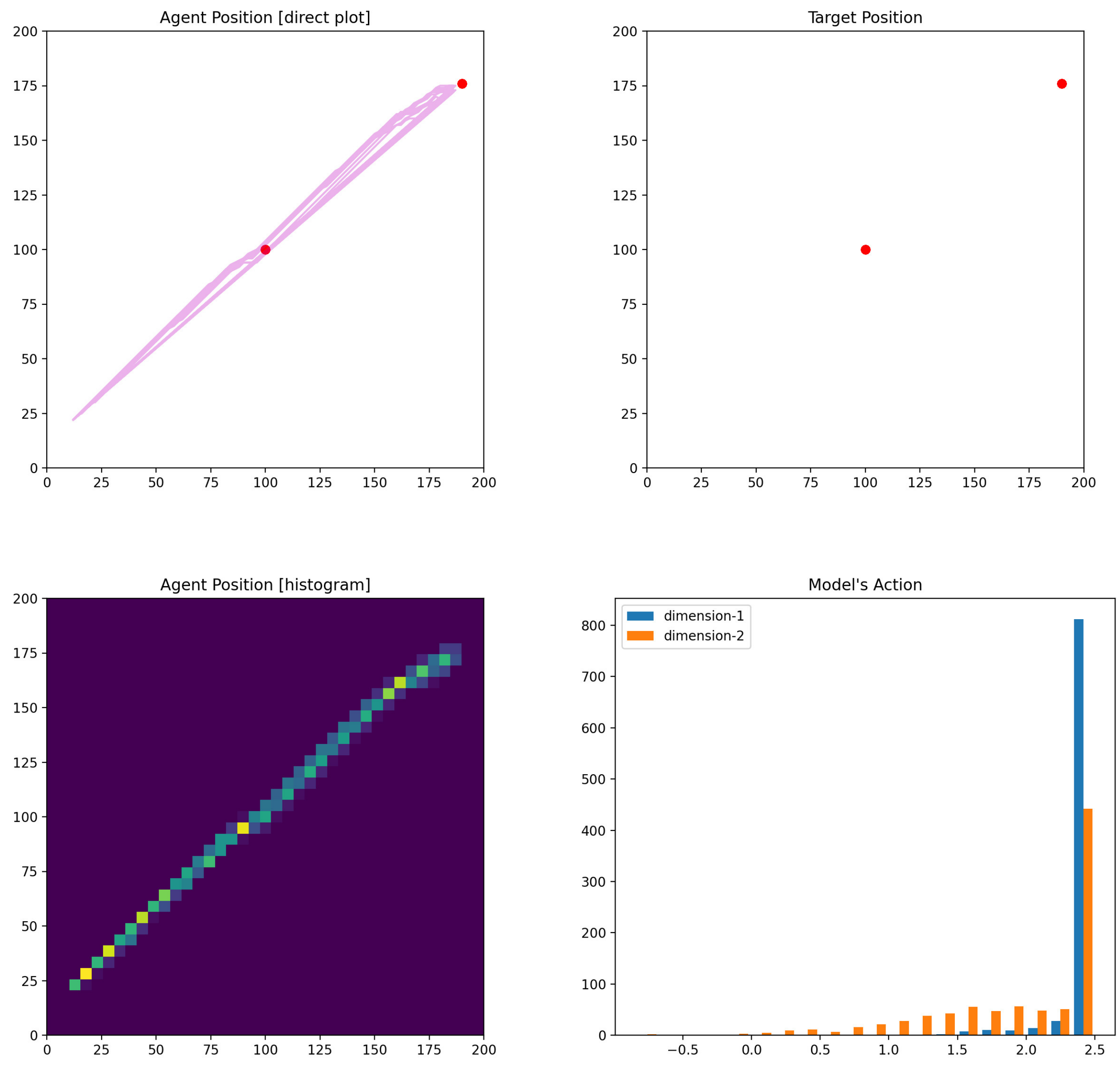

Figure 6.1. Various visualization schemes of the Expert agent of the continuous default `PointNav` ID $= 0$. Upper and lower left: direct-plot and histogram plot of the agent's position. Upper right: positions of the two targets. Lower right: distribution of the agent's action.

Solution 3. General analysis strategy of `PointNav` agents

The study of the Expert agents above extends towards the general procedure of studying any model trained on the `PointNav`: with the statistics of episode-length m^{Length} as baseline and the reward-value m^{Reward} as auxiliary metric, one decides on the general assessment of success or fail for the agent in question.

Subsequently, the plotting utilities offer further insight to the real-time playback of the expert with respect to the actual position of the agent and the distribution of its actions.

Both the stats and the plotting outputs are saved for all the Experts above and the PW-IL agents introduced in the upcoming section.

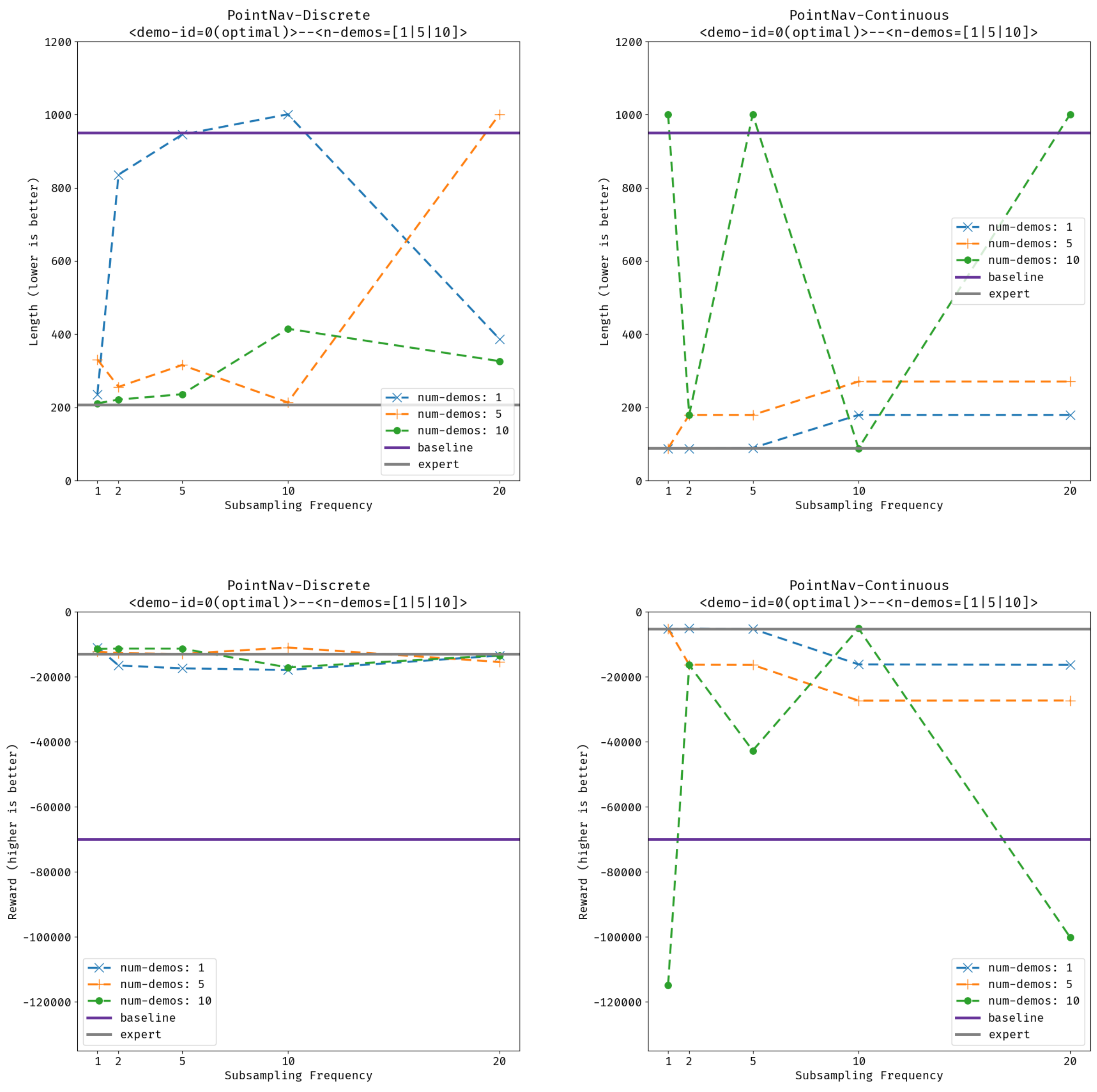

FIGURE 7.1. Performance of PW-IL with demo-id 0 (optimal quality). Upper row: ; lower row: . Left column: discrete `pointnav`; right column: continuous `pointnav`.

7. RESULTS OF PW-IL

7.1. **Optimal demonstration.**

The optimal performance of the PW-IL agents are presented in Figure 7.1.

Solution 4. Analysis of PW-IL with optimal demonstration

One observes that PW-IL

7.2. **Non-optimal demonstration.**

7.2.1. *The Discrete version.*

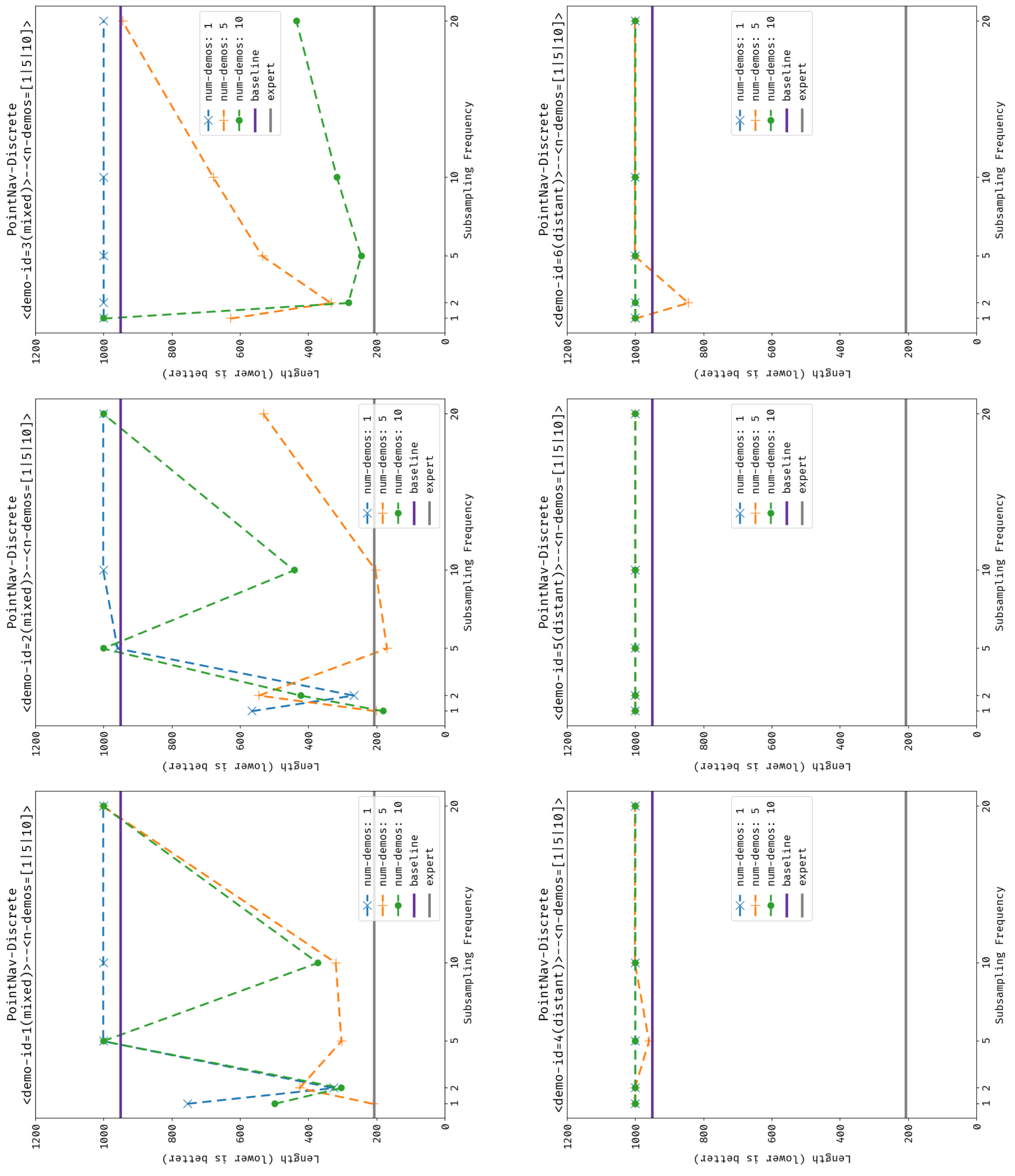

FIGURE 7.2. Performance wrt episode-length m^{Length} of PW-IL on **discrete** `PointNav` with non-optimal demonstration-pools. Upper row from left to right: demo-id $\{1, 2, 3\}$ (of quality »mixed«); Lower row from left to right: demo-id $\{4, 5, 6\}$ (of quality »distant«).

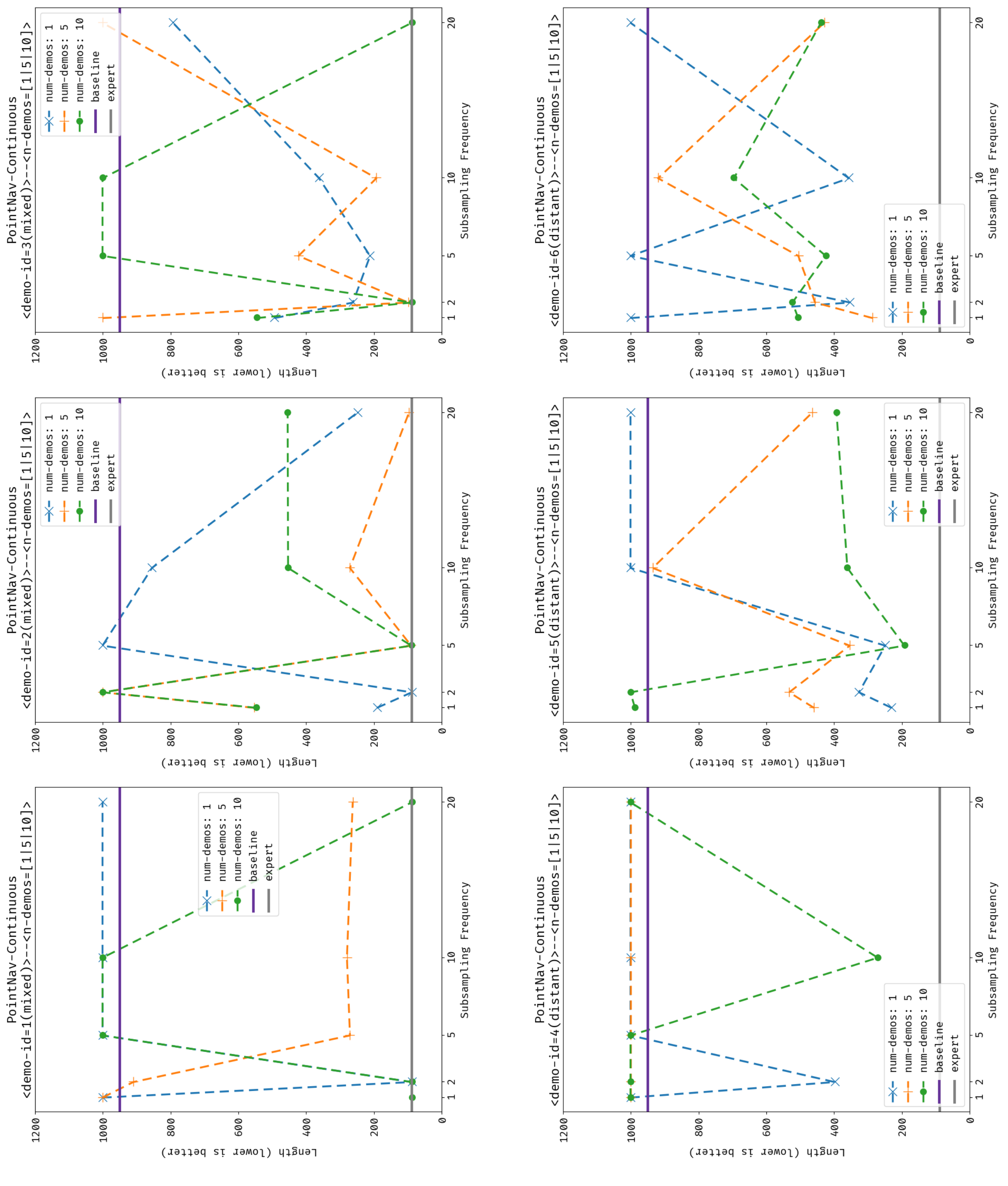

FIGURE 7.3. Performance wrt episode-length m^{Length} of PW-IL on **continuous** `PointNav` with non-optimal demonstration-pools. Upper row from left to right: demo-id $\{1, 2, 3\}$ (of quality »mixed«); Lower row from left to right: demo-id $\{4, 5, 6\}$ (of quality »distant«).

7.2.2. *The Continuous.*

7.3. **Analysis.**

Activities	Duration (×week)
literature survey; project structuring	0.5
testing of relevant source-code and algorithm	1
development of testing framework	1.5
source-code refactoring; expansion of testing	1
collection of results; composition of thesis	1

Table 8.2. Time-stamps of key activities

Part 5. Epilogue

8. Executive Summary

8.1. **The gist.** This project studies the performance of the Primal Wasserstein IL as introduced in [Dad+20a] under varying conditions of:

(1) Quality of demonstrations by expert;
(2) Number of demonstrations available to the algorithm
(3) Sample scarcity with sub-sampling frequency

To this end, a testing-framework targeted at varying these essential configurations is created. A

8.2. **Timeline.** The administrative details of this thesis can be retrieved under the course-catalogue of ETH Zürich under the registration number `401-3740-01L`. Its total duration is estimated at approximately 5 weeks of full-time work, including the drafting of the thesis and the composing of the source-code. Specifics are found in Table 8.2.

8.3. **Future work.**

Definition 9. Part 5

(1) Future work:
  (a) more environments with this framework (cartpole)
  (b) further testing of PWIL
    (i) with IRL (AIRL)

9. Acknowledgements

To the supervisor of this thesis, Prof. Dr. J. Buhmann: my deepest appreciation for the acceptance of the original idea of this thesis in its earliest infancy, for the inspiring and philosophical approach to the field of machine-learning as a whole, and, of course, for granting the much needed extension for completion;

To my advisors, Ivan, Ami and Eugene: my heart-felt thanks to the extended sessions of talks, the continuous support and the constructive suggestions;

To my family and my friends: my sincere gratitude for the unyielding support and care;

Finally, to J. Neubauer: all my respect for his natural mastery of the game and brilliant deliveries in the most competitive environments. Rest in peace among these beloved pieces, *maestro*.

References

1. Arjovsky, M., Chintala, S. & Bottou, L. *Wasserstein GAN* 2017. https://arxiv.org/abs/1701.07875.
2. Bellman, R. *Dynamic Programming* 1st ed. (Princeton University Press, Princeton, NJ, USA, 1957).
3. Bertsekas, D. P. *Dynamic programming and optimal control* (1995).
4. Brockman, G. *et al. OpenAI Gym* eprint: arXiv:1606.01540.
5. Brown, D. S., Goo, W., Nagarajan, P. & Niekum, S. *Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations* 2019. https://arxiv.org/abs/1904.06387.
6. Chen, S. *This Thesis: A Study of Primal Wasserstein Imitation-Learning in Goal-Conditioned RL* https://github.com/shengdichen/sem_tss/blob/main/src/main.pdf.
7. Dadashi, R., Hussenot, L., Geist, M. & Pietquin, O. *Primal Wasserstein Imitation Learning* 2020. https://arxiv.org/abs/2006.04678.
8. Dadashi, R., Hussenot, L., Geist, M. & Pietquin, O. *Primal Wasserstein Imitation Learning Source-Code Repository* https://github.com/google-research/google-research/tree/master/pwil.
9. Dulac-Arnold, G., Mankowitz, D. & Hester, T. *Challenges of Real-World Reinforcement Learning* 2019. https://arxiv.org/abs/1904.12901.
10. Hill, A. *et al. Stable Baselines*
11. Jacq, A., Geist, M., Paiva, A. & Pietquin, O. *Learning from a Learner* in *International Conference on Machine Learning* (2019).
12. Mnih, V. *et al.* Asynchronous Methods for Deep Reinforcement Learning. https://arxiv.org/abs/1602.01783 (2016).
13. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).
14. OpenAI. *OpenAI Spinning Up in Deep RL* eprint: arXiv:1606.01540.
15. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. *Proximal Policy Optimization Algorithms* 2017. https://arxiv.org/abs/1707.06347.
16. Silver, D. *et al.* Mastering the game of Go without human knowledge. *Nature* **550,** 354–359 (2017).
17. Sutton, R. S. & Barto, A. G. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks* **16,** 285–286 (2005).
18. Tesauro, G. Temporal Difference Learning and TD-Gammon. *J. Int. Comput. Games Assoc.* **18,** 88 (1995).
19. Villani, C. in, xxii+973 (Jan. 2008).
20. Willing, C. *et al. The uncompromising Python code formatter*