AN EMPIRICAL STUDY OF PRIMAL WASSERSTEIN IMITATION-LEARNING IN GOAL-CONDITIONED RL

SHENGDI CHEN[α]

Abstract. This thesis investigates the performance impact of various configurations options of the Primal Wasserstein Imitation-Learning algorithm by empirically correlating the performance of the resultant agents and the settings of three hyper-parameters of the algorithm to control the optimality and richness of the expert demonstration information. A separate evaluation framework is created with a sample test-environment of a two-dimensional navigation task.

Among the investigated parameter options, utilizing expert-demonstrations of higher quality is found to dramatically increase the probability of obtaining capable agents. Settings of the number of expert episode-trajectories and the sub-sampling frequency for processing the demonstrations are observed to display high variance with respect to the particular cases of application and thus require extensive tuning to retrieve the optimal pairing value for the specific task in question.

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 (PW-IL) algorithm as introduced in [Dad+20a] to a testing-framework utilizing a separate testing-environment referred to as »`PointNav`«. Three hyper-parameters of PW-IL are configured independently to control the quality and data availability of the input demonstration.

1.2. **Related Researches.** Methodologies from Reinforcement Learning (RL) have solved numerous difficult tasks, with notable examples such as [Mni+15]; [Sil+17]; [Tes95]. Representative algorithms of RL include [Mni+16]; [Mni+15]; [Sch+17].

Key to the PW-IL algorithm in question is the Wasserstein distance, the p^{th}-order generalization of which is studied in [Vil08], displaying connections with the optimal transport problem originally coined by [Mon81] and expanded upon by [Kan04b]; [Kan04a]. Generative Adversarial Networks (GANs) utilizing this distance, as referred to the Wasserstein GANs (WGANs), are introduced in [ACB17].

[Bro+19]; [Jac+19] study RL training scenarios performed with non-optimal demonstrations. Other Inverse Reinforcement Learning (IRL) methods are found in [Rus98]; [NR00]. Methodologies of the Behavioral Cloning (BC) paradigm are presented in [Bag+06]; [Pom91]; [RB10].

2. Technical Details

2.1. **Programming.** The source-code is packaged in one monolithic repository, including the testing-environment, statistical and visual analysis modules and an adapated implementation of the PW-IL algorithm based on the original open-source code published by the authors in [Dad+20b].

The programming is performed with the `Python` language with global type-hinting. Python versions no earlier than `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.

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 algorithm.

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 source-code of this project is distributed under the following repositories:

- The main repository contains all code necessary for inspecting and reproducing the training and testing results of this study and is published under: GitLab [ETH-Domain] and GitHub.
- This thesis-document is accessible under: GitHub.

[1] Version requirement of `Python`'s `typing`-Module:

Part 2. PWIL Fundamentals

3. Essentials of the PW-IL Algorithm

3.1. The RL problem.

The Markov-Decision-Process (MDP) framework as introduced in [SB05] 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:

$$s_t \xrightarrow{\pi} a_t$$

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« in reference literature such as [SB05]; [Bel57]; [Ber95].

3.2. The Wasserstein-distance.

3.2.1. *Definitions.* As studied in [Vil08], the p^{th}-order Wasserstein-distance[2] of PW-IL's namesake with respect to metric-space (M, d) is defined as follows:

$$\begin{aligned}\mathcal{W}_p &:= \left(\inf_{\theta \in \Theta(\mu,\nu)} \left\{ \mathop{\mathbb{E}}_{(x,y)\sim\theta} \left[d^p(x,y) \right] \right\}\right)^{1/p} \\ &\equiv \left(\inf_{\theta\in\Theta} \left\{ \int d^p(x,y)\, \mathrm{d}\theta(x,y) \right\}\right)^{1/p}\end{aligned}$$

where μ and ν are two (probability) measures on (M, d), and $\theta(\cdot,\cdot)$ denotes a coupling on $M \times M$:

$$\begin{cases} \int_M \theta(x,y)\, \mathrm{d}y \equiv \mu(x) \\ \int_M \theta(x,y)\, \mathrm{d}x \equiv \nu(y) \end{cases}$$

As a side-note, the Wasserstein-distance is known ([Vil08]) to display connections with the optimal transport problem, originally proposed in [Mon81] and further investigated and refined in [Kan04b]; [Kan04a].

3.2.2. *Application in RL.* [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 the settings of RL) with guaranteed smoothness.

In the settings of the RL problem as defined in Section 3.1, the metric $d(\cdot,\cdot)$ for the Wasserstein-distance is understood as:

$$\begin{bmatrix} S \\ A \end{bmatrix} \times \begin{bmatrix} S \\ A \end{bmatrix} \xrightarrow{d(\cdot,\cdot)} \mathbb{R}^+$$

The PW-IL algorithm [Dad+20a] considers only the first-order Wasserstein-distance $p \equiv 1$ and imposes the restriction of distributions with only finite support. With D state-action pairs as

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

demonstration provided by some expert, the Wasserstein-distance between policy π and the expert itself is found with:

$$\mathcal{W}_{p\equiv 1}\left(\pi, \text{expert}\right) := \inf_{\theta\in\Theta} \sum_{i\in[1;T]} \sum_{j\in[1;D]} \left\{ d\left(\begin{bmatrix} s_i^{\pi} \\ a_i^{\pi} \end{bmatrix}, \begin{bmatrix} s_j^{\text{expert}} \\ a_j^{\text{expert}} \end{bmatrix} \right) \theta\left(i,j\right) \right\}$$

where T denotes the length of time-horizon as seen in Section 3.1. The Wasserstein-optimal policy follows naturally:

$$\pi^{\star} := \inf_{\pi\in\Pi} \left\{ \mathcal{W}\left(\pi, \text{expert}\right) \right\}$$

Detailed discussions on the expert-demonstrations are found later in Section 5.

3.3. **The PW-IL algorithm.** [Dad+20a] observes that solving for the optimal solution of the Wasserstein-distance necessitates trajectory information of the entire episode, posing difficulties for online learning tasks and environments of prolonged episodes. This requirement is bypassed by the PW-IL algorithm in [Dad+20a], where, instead of the optimal coupling $\theta^{\star}$ (with respect to the true Wasserstein-distance), the following greedy coupling θ^{grd} is deployed:

$$\theta^{\text{grd}} := \underset{\theta[i,T]\in\Theta_i}{\arg\min} \sum_{j\in[1;D]} \left\{ d\left(\begin{bmatrix} s_i^{\pi} \\ a_i^{\pi} \end{bmatrix}, \begin{bmatrix} s_j^{\text{expert}} \\ a_j^{\text{expert}} \end{bmatrix} \right) \theta\left(i,j\right) \right\}$$

with the set $\Theta_i := \left\{ \theta\left[i,T\right] \in \mathbb{R}^{D,+} \right\}$ satisfying the constraints:

$$\Theta_i \triangleq \begin{cases} \displaystyle\sum_{j'\in[1;D]} \theta\left(i,j'\right) = \frac{1}{T} \\ \forall k \in [1;D], \displaystyle\sum_{i'\in[1;i-1]} \left\{ \theta^{\text{grd}}\left(i',k\right) + \theta\left(i,k\right) \right\} \leq \frac{1}{T} \end{cases}$$

[Dad+20a] further shows that the greedy-coupling is non-optimal and thus by-definition serves as an upper-bound for the true Wasserstein-distance. Using the greedy coupling θ^{grd} instead of attempting to find the optimal $\theta^{\star}$, the PW-IL algorithm thus computes its reward in an *offline* manner, without re-evaluation throughout the agent's interaction with the underlying environment. The minimizer with this greedy-coupling θ^{grd} is finally used to calculate the per-time-step reward:

$$r_i^{\texttt{PW-IL}} := f\left(\sum_{j\in[1;D]} \left\{ d\left(\begin{bmatrix} s_i^{\pi} \\ a_i^{\pi} \end{bmatrix}, \begin{bmatrix} s_j^{\text{expert}} \\ a_j^{\text{expert}} \end{bmatrix} \right) \theta^{\text{grd}}\left(i,j\right) \right\} \right)$$

with f chosen as:

$$f\left(c\right) := 5 \, \exp\left(-\frac{5T}{\left(|S|+|A|\right)^{1/2}} \, c \right)$$

3.3.1. *Further details.* Per [Dad+20a], the complexity of computing the reward-value after one time-step is $\mathcal{O}\left(\left(|S|+|A|\right) D + {}^{D^2}/_{T}\right)$. Thus, the total runtime estimate across the entire time-horizon of T steps as:

$$\mathcal{O}\left(\left(|S|+|A|\right) DT + D^2\right)$$

The interested reader shall inspect the original paper [Dad+20a] for further derivation details and (non-)optimality discussions with respect to the greedy coupling and the resultant PW-IL algorithm.

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 the 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 the second target in the upper, right corner at $(190; 190)$.

4.1.2. *Reward.* The reward at time-step t 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 in Note 2.

4.2. The `PointNav` variants.

4.2.1. *Target shifts.* For controlled modifications of the environment `PointNav`, the first target at the default position $(100; 100)$ is shifted off-center. In this project, two such target shiftings are made, leading to two new variants respectively. Table 4.2 documents all three versions of `PointNav`: the default (without shifting) is identified as Variant-0, while the other two

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

TABLE 4.2. The three different variants of `PointNav`.

variants (of ID 1 and 2) symmetrically perform shifting of $+50$ in the two movement-directions respectively.

Note 1. The actual target for PW-IL

Note that the default Variant-0 in Table 4.2 is the (only) environment that the PW-IL algorithm is trained on and evaluated against. The usage of the two shifted variants is illustrated in Section 5.2.3 and Table 5.2.

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, corresponding to the desired behavior.

Action-Space Continuity	`PointNav` Variant-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. The six variants of `PointNav`

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 10.

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 six variants.* To summarize this introductory section of `PointNav`, the six variants of `PointNav` are enlisted in Table 4.3.

5. Demonstration Setup

As briefly touched upon in the introductory Section 3.2.2, the »demonstration« from the expert-agent must be provided before deploying the PW-IL method. This section defines such demonstrations and describes the hyper-parameters available to the process of their generation.

5.1. **Preliminaries.** Per [Dad+20a], the demonstration for PW-IL transports information of episode-trajectories of the expert model, with the episode-trajectories defined as follows:

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 $\texttt{sa}_t$ is the concatenation of the environments state s and the agent's action a at time-step t:

$$\texttt{traj} := \left[\texttt{sa}_1, \texttt{sa}_2, \cdots, \texttt{sa}_{\text{episode-over}}\right]$$

Based on the expert's trajectories as described above, the demonstration of PW-IL is understood as follows:

Definition 7. Demonstration for the PW-IL algorithm

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

$$\texttt{demo} := [\texttt{sa}_1, \texttt{sa}_2, \cdots]$$

5.2. **The hyper-parameters.**

5.2.1. *Number of trajectories.* The above definition hints at the usage of multiple expert-trajectories to generate PW-IL's demonstration. This notion is developed as follows:

Definition 8. The candidate-trajectories of PW-IL

The candidate-trajectories used to generate the demonstration is given as:

$$\texttt{candidates-traj}^{\text{PW-IL}} := \left\{\texttt{traj}_1, \texttt{traj}_2, \cdots, \texttt{traj}_{\text{\#-traj}}\right\}$$

where the number of candidate-trajectories $h^{\text{\#-traj}}$ constitutes a configurable hyper-parameter of the PW-IL algorithm:

$$h^{\text{\#-traj}} := \left|\texttt{candidates-traj}^{\text{PW-IL}}\right|$$

Definition 9. Choices of $h^{\text{\#-traj}}$

While the original paper of PW-IL [Dad+20a] uses $\{1, 4, 11\}$ for $h^{\text{\#-traj}}$, this study deploys the following values:

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

One notes that the [Dad+20a] reports improved training results with larger trajectory-pools, i.e., higher $h^{\text{\#-traj}}$ values. This claim is investigated in subsequent sections on performance analysis of PW-IL.

5.2.2. *Sub-sampling.* For each of the candidate-trajectories, sub-sampling at frequency $h^{\text{s-smp}}$ is performed by selecting state-action pairs once every $h^{\text{s-smp}}$ time-step(s). Assume thus that within a certain 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 unique sub-sampling frequency of 20 to simulate data scarcity, this study expands upon this to empirically study the impact of $h^{\text{s-smp}}$ on PW-IL performance:

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

The sub-sampled state-action pairs of all $h^{\text{\#-traj}}$ expert candidate-trajectories are finally concatenated to form the demonstration as input for deploying the PW-IL algorithm, formatted as shown in Definition 7.

Pool-ID h^{pool}	Expert-IDs in Pool	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. The IDs and constituents of the seven expert-pools.

5.2.3. *The expert-pools.* Besides the number of candidate-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]. This section explains tuning of the demonstration quality for `PointNav`.

As explained in Note 1, the PW-IL is trained and evaluated against the default Variant-0 of `PointNav`. By including trajectories generated by experts trained on the shifted, non-default Variant-1 and 2 in the set of candidate-trajectories, one may actively control the quality of the demonstration as input for PW-IL.

To simplify nomenclature, experts trained on the default Variant-0 of `PointNav` is referred to as the »default-expert« or »expert-0«, whereas experts of the shifted variants are identified as »expert-1« and »expert-2« respectively. The presence of the non-default experts in the expert-pool thus leads to non-optimal demonstrations. Furthermore, one distinguishes between the »mixed« pool, where expert-0 is included and the »distant« pool, where only the non-default expert-1 and expert-2 are taken into consideration.

By subsequently varying the specific constellation within the non-optimal pools, one observes in total seven different expert-pools deployed in this study, identified with the hyper-parameter $h^{\text{pool}} \in \{0, \cdots, 6\}$ in Table 5.2.

Remark 1. Symmetry-heuristics of the expert-pools

Attention shall be paid to pool-IDs 0, 3 and 6: due to the symmetry of the underlying `PointNav` Variant-1 and 2 as introduced in Table 4.2, an expert-pool containing both or neither of these two shifted variants shall generate heuristically unshifted trajectories with respect to the main diagonal of the board, which is the ideal path for the default variant-0 of `PointNav`. This specific property of these expert-pools is studied later in Section 7.2.

5.2.4. *Summary of hyper-parameters.*

Conclusion 1. Configuration-Space of PW-IL

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

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

With the distinct and continuous variants of the action-space, the total number of agents trained with the PW-IL algorithm is thus calculated as follows:

$$\begin{aligned}\underbrace{2}_{\text{disc\&cont}} \times \left|h^{\texttt{PW-IL}}\right| &:= 2 \times \left(\left|h^{\text{pool}}\right| \times \left|h^{\#\text{-traj}}\right| \times \left|h^{\text{s-smp}}\right|\right) \\ &:= 2 \times (7 \times 3 \times 5) \\ &\equiv 210\end{aligned}$$

The upcoming sections analyze the performance of the PW-IL algorithm under varying configurations of the three hyper-parameters nominated above.

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. 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 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 10. 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 justifications of the baseline episode-length value of 1000.

Alternatively, the reward m^{Reward} returned by the environment could be 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 efficacy. Empirical observations suggest that a random agent displays reward-values of approximately $\lessapprox -1.2\,\text{e}5$, whereas signs of »meaningful« training are found for 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 without difficulty.

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 10, 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 through action-rounding, the continuous `PointNav` is solved more robustly than the discrete version. One observes that, in particular, the Variation-2 of the discrete model demonstrates high standard-deviation for episode-length m^{Length}.

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

Solution 2. Visualization of a `PointNav` agent

Figure 6.1 provides various graphs for studying 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 provides 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 wonder its m^{Length}-value is near optimal (88.20).

Solution 3. General analysis strategy of `PointNav` agents

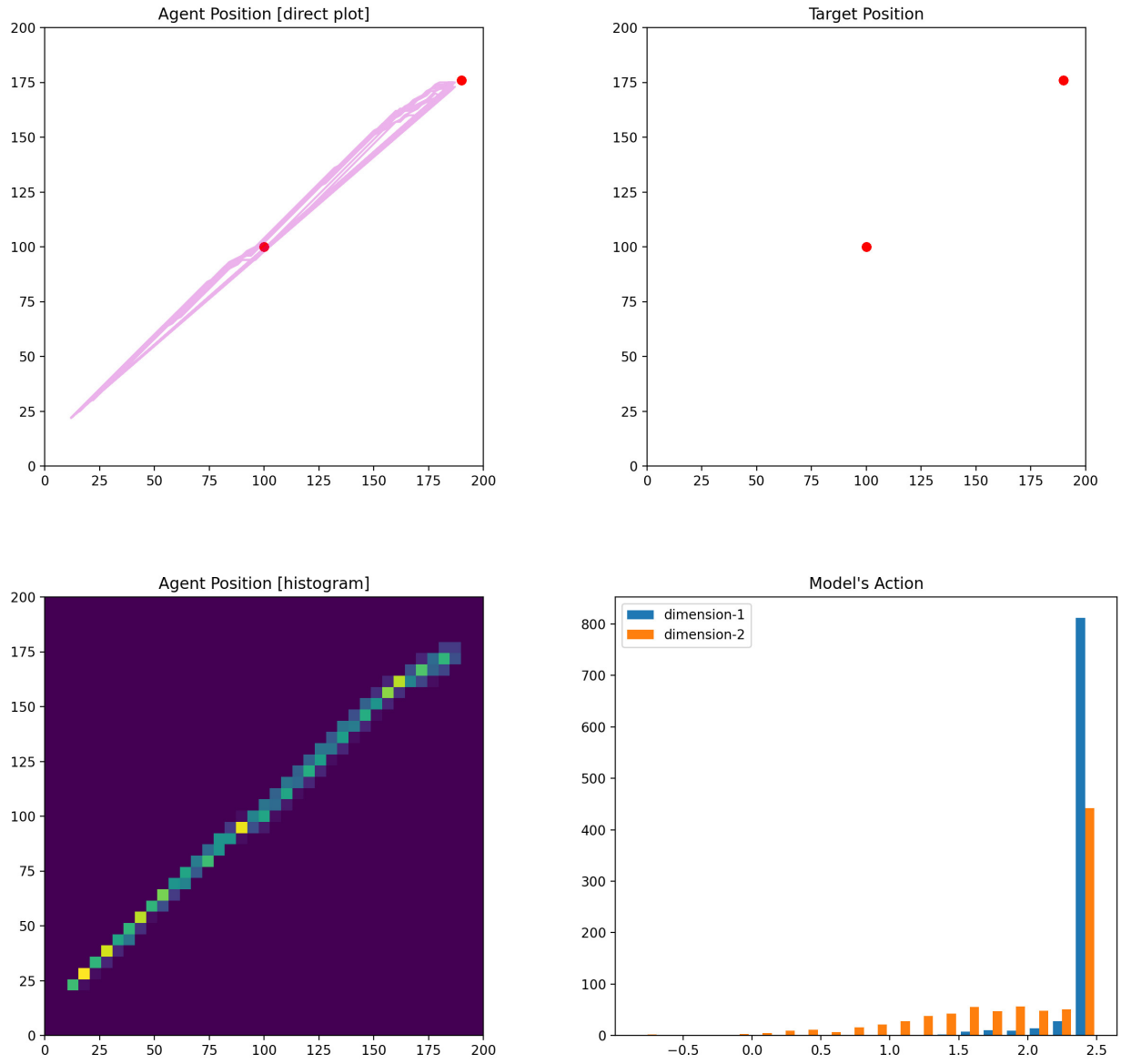

FIGURE 6.1. Various visualization schemes of the Expert agent of the continuous default `PointNav` ID $= 0$. Upper and lower left: direct-plot and histogram plot of the agent's position. Upper right: positions of the two targets. Lower right: distribution of the agent's action.

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:

- GitHub: Outputs of the Experts
- GitHub: Outputs of PW-IL

7. Results of PW-IL

This section presents the result of agents trained with the PW-IL algorithm, sorted by the optimality of the input expert-demonstrations. Due to the elevated number of models trained, instead of the separate study of individual models as seen in Table 6.2 and Figure 6.1, the performance of multiple models are grouped together by hyper-parameter choices for visual inspection.

Note that all PW-IL agents are trained for 500 thousand steps with the same intermediate evaluation frequency of 10 thousand steps as for the six previously introduced Experts to select the best performing model.

7.1. **Optimal demonstrations.**

To commence, the performance of the PW-IL agents with the »optimal« pool are presented in Figure 7.1, where the upper row shows the episode-length m^{Length}, whereas the lower the reward-values m^{Reward} as performance measure. On the left, the discrete-action `PointNav` is solved, and its continuous counterpart on the right.

Solution 4. Analysis of PW-IL with optimal demonstration $h^{\text{pool}} = 0$

At first glance, one observes that for the majority of the $(h^{\text{\#-traj}}, h^{\text{s-smp}})$ configurations, the baseline of episode-length $m^{\text{Length}} \leq 1000$ is met, indicating successful solving of the `PointNav` environment. Cases also abound where the expert performance is almost fully recovered. One further draws the following conclusions for the discrete and the continuous variants of `PointNav` respectively:

For the discrete `PointNav` environment, the couplings of $(h^{\text{\#-traj}}, h^{\text{s-smp}}) = (1, 10)$ and $(5, 20)$ are the only two setups leading to unsatisfied baseline. In particular, one observes from the upper-left plot that PW-IL agents with a higher number of trajectories $h^{\text{\#-traj}} = 10$ (marked green) uniformly out-perform the agents with lower trajectories-count $h^{\text{\#-traj}} = 1$ (marked blue), conforming to the expectation of the author of PW-IL in [Dad+20a]. However, for $h^{\text{\#-traj}} = 5$ (marked orange), the agents do not consistently out-perform or under-perform the cases of 1 and 10, and at sub-sampling $h^{\text{s-smp}} = 20$, the corresponding agent does not even achieve the baseline. For the sub-sampling frequency $h^{\text{s-smp}}$, larger values degrade the performance of the case where $h^{\text{\#-traj}} = 5$, and appears to have little to even the opposite impact for the configurations $h^{\text{\#-traj}} \in \{10, 1\}$.

For the continuous counterpart, the highest trajectories-number $h^{\text{\#-traj}} = 10$ (marked green) displays high variance (upper-right plot): with three cases of $h^{\text{s-smp}}$ failing to meet the baseline. One further concludes that, contrast to the continuous case, lower $h^{\text{\#-traj}}$ values lead to better performance, where, with only one trajectory $h^{\text{\#-traj}} = 1$ (marked blue), the expert performance is completely recovered for $h^{\text{s-smp}} \leq 5$. Concerning the sub-sampling parameter $h^{\text{s-smp}}$, larger values lead to poorer performance for the stable $h^{\text{\#-traj}}$ values of 1 (marked blue) and 5 (marked orange). However, as stated previously, no immediate correlation is visible for $h^{\text{\#-traj}} = 10$ due to high fluctuations.

Solution 5. Relationship between length and reward

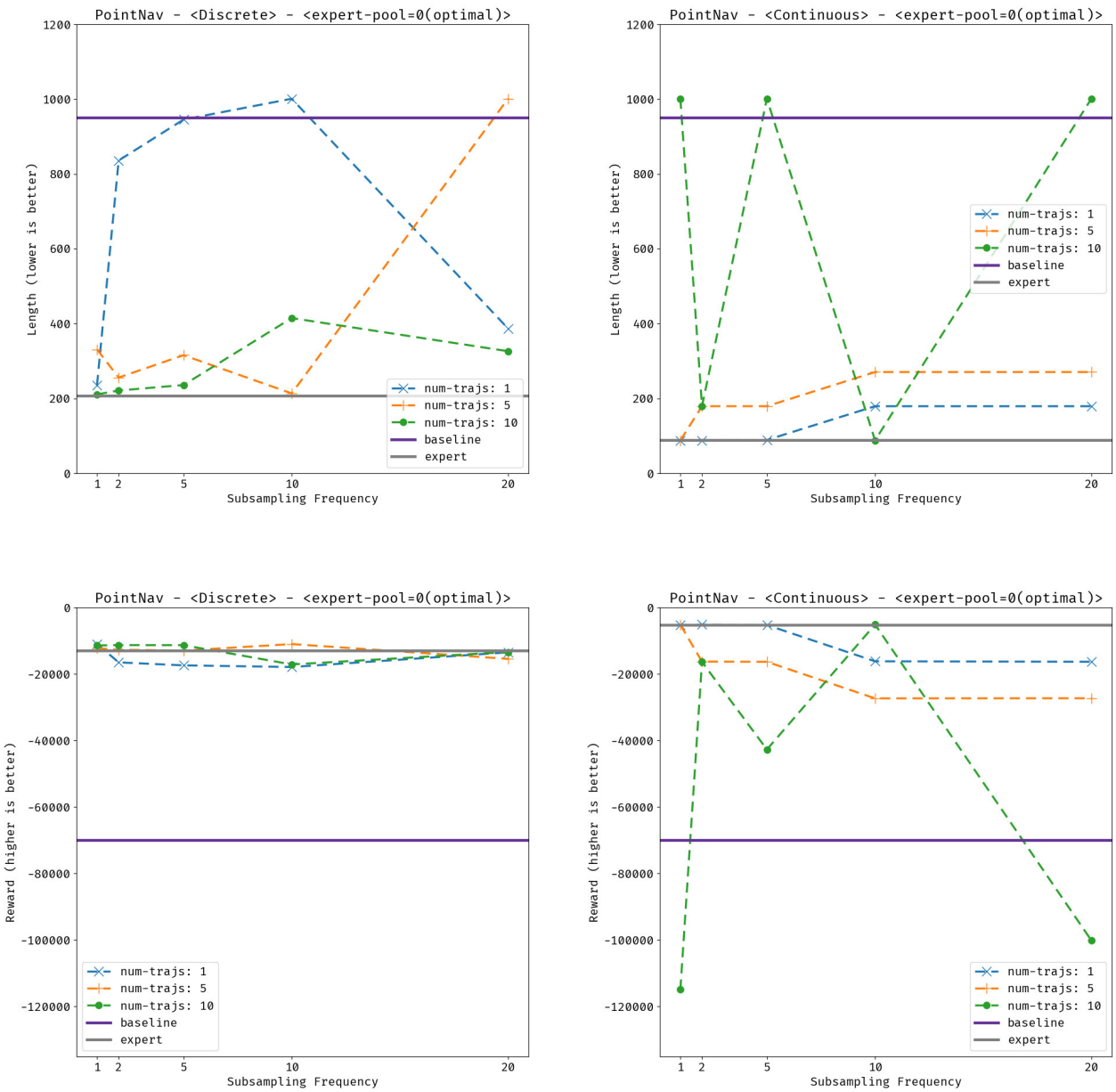

FIGURE 7.1. Performance of PW-IL with expert-pool $h^{\text{pool}} = 0$ (optimal demonstrations).

Studying the lower-left reward-plot of the discrete environment, one observes almost constant reward values for all three $h^{\#\text{-traj}}$ values despite large fluctuations in the upper-left episode-length plot. As discussed in Definition 10, this indicates extended hovering of the agent in close proximity to a target without actually reach its position. Despite not passing the baseline, these agents nevertheless demonstrate positive training impact.

On the other hand, the lower-right reward-plot of the continuous `PointNav` is almost the mirror image of the plot of episode-length, proving low training impact in the cases of unmet baseline. Recalling from Definition 10 that a purely random agent displays reward values of approximately $-1.2\,\text{e}5$, one decides that the agent configured with $(h^{\#\text{-traj}}, h^{\text{s-smp}}) = (10, 1)$ barely displays any improvement over an random actor: surprising outcome, as this

configuration includes the most amount of trajectories with the lowest frequency of sub-sampling, and thus, per heuristic reasoning of the original authors [Dad+20a], should possess the highest amount of replay information from the expert and out-perform other configurations.

7.2. **Non-optimal demonstrations.** For the non-optimal trajectory-pools, i.e., $h^{\text{pool}} \in \{1, \cdots, 6\}$, only the graphs with respect to the metric episode-length m^{Length} are shown. The interested reader is invited to inspect the parallel performance visualizations with the reward-value as metric in the GitHub repository.

7.2.1. *The discrete-action* `PointNav`. The episode-reward values m^{Length} of the PW-IL agents trained on the discrete `PointNav` environment are plotted in Figure 7.2, where the pool-IDs $\{1, 2, 3\}$ of the »mixed« expert-pool are plotted on the left from top to bottom, and the IDs $\{4, 5, 6\}$ of the »distant« pool are depicted on the right.

Solution 6. Analysis of PW-IL agents on discrete `PointNav` with »mixed« expert-pools

It is evident that for the »mixed« expert-pool, the configurations with 1 trajectory $h^{\#\text{-traj}} = 1$ (marked blue) consistently under-perform the cases of $h^{\#\text{-traj}} \in \{5, 10\}$, although for $h^{\text{pool}} \in \{1, 2\}$, the higher value 5 for $h^{\#\text{-traj}}$ (marked orange) surprisingly out-performs the settings with $h^{\#\text{-traj}} = 10$ (marked green). Also, considerable fluctuations are observed for these particular expert-pools of ID 1 and 2. One concludes that for higher sub-sampling frequencies $h^{\text{s-smp}} \geq 5$, higher $h^{\#\text{-traj}}$ values most likely exceed the performance of the case with only one trajectory ($h^{\#\text{-traj}} = 1$). One shall also note that for the configurations of $\left(h^{\text{pool}}, h^{\#\text{-traj}}, h^{\text{s-smp}}\right) \in \{(2, 10, 1), (2, 5, 5)\}$, *super*-expert performance is achieved.

For the pool-ID $h^{\text{pool}} = 3$ (lower-left), disregarding the sub-sampling frequency of $h^{\text{s-smp}} = 1$, one discovers improvements in performance with higher values of trajectories-numbers $h^{\#\text{-traj}}$ and lower frequencies of sub-sampling $h^{\text{s-smp}}$. This observation aligns with the statements and the findings of the original paper [Dad+20a], as larger $h^{\#\text{-traj}}$ and smaller $h^{\text{s-smp}}$ values both hint at richer expert-demonstration data for the PW-IL algorithm.

Solution 7. Analysis of the »distant« pools and the symmetry-heuristics

For the distant expert-pool, the performance of the PW-IL agents appear bleak at best. Almost all models fail to achieve the baseline, with the rare exception found in pool-ID 6. Although this particular agent just about met the baseline requirement, the symmetry-heuristics of pool-ID 6 in Remark 1 might nevertheless be taken into account: In particular, one observes comparatively better performance of agents for $h^{\text{pool}} = 6$ (heuristically symmetric) in relation to 4 and 5 (both heuristically asymmetric), as well as the stabler, less fluctuating result of $h^{\text{pool}} = 3$ (symmetric), as discussed previously compared to the instances of 1 and 2 (both asymmetric).

7.2.2. *The continuous-action* `PointNav`. Similar to the discrete counterpart, performance graphs of the PW-IL agents trained on the continuous-action `PointNav` are presented in Figure 7.3.

Solution 8. Analysis of PW-IL agents on continuous `PointNav` with »mixed« expert-pools

High variations are observed for PW-ILs agents with mixed expert-pools. In particular, only the configuration of $\left(h^{\text{pool}}, h^{\#\text{-traj}}\right) = (3, 1)$ (blue line of lower-left plot) meets the baseline `PointNav` across all sub-sampling frequencies.

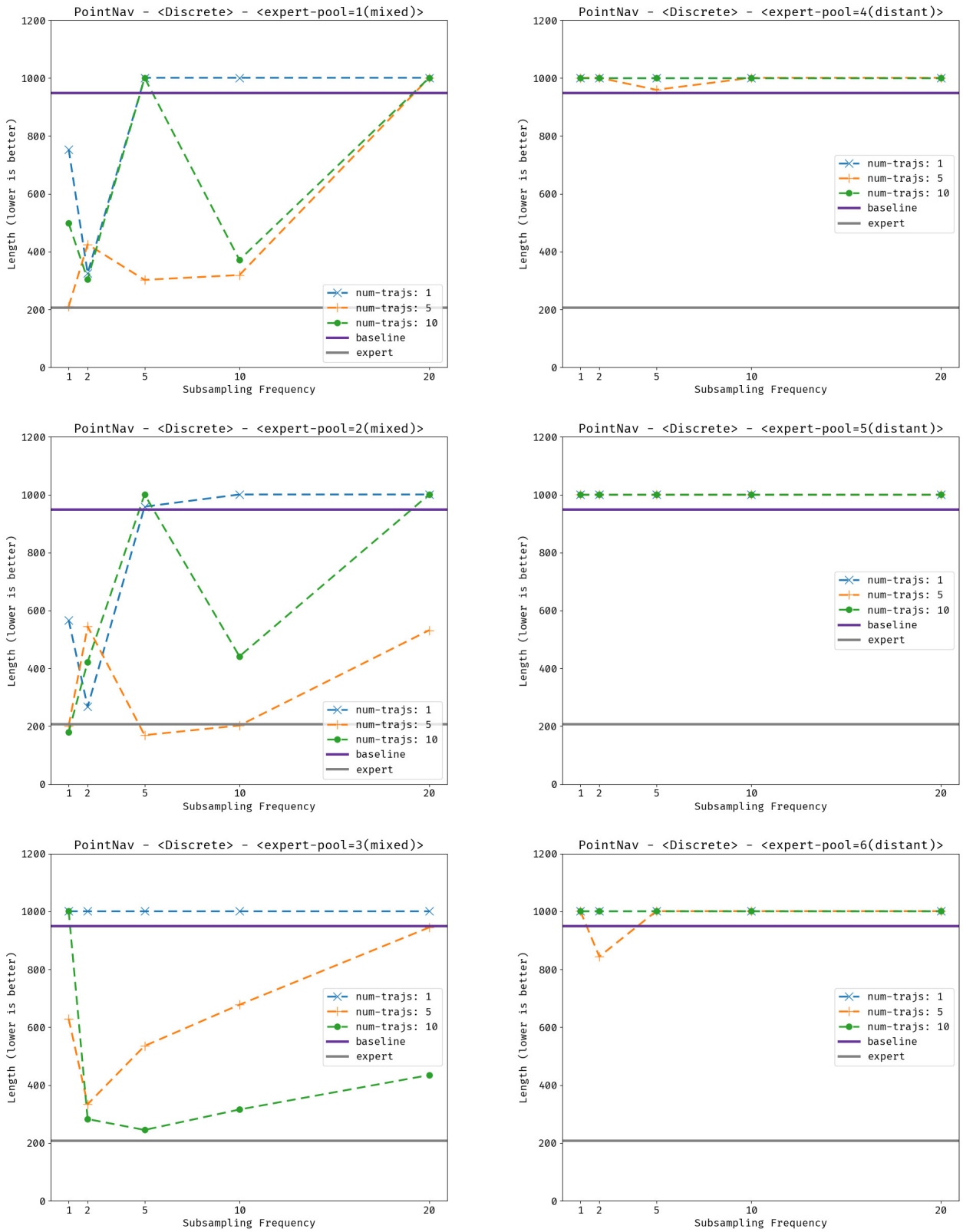

FIGURE 7.2. Episode-length m^{Length} Performance of PW-IL on discrete `PointNav` with non-optimal demonstration-pools.

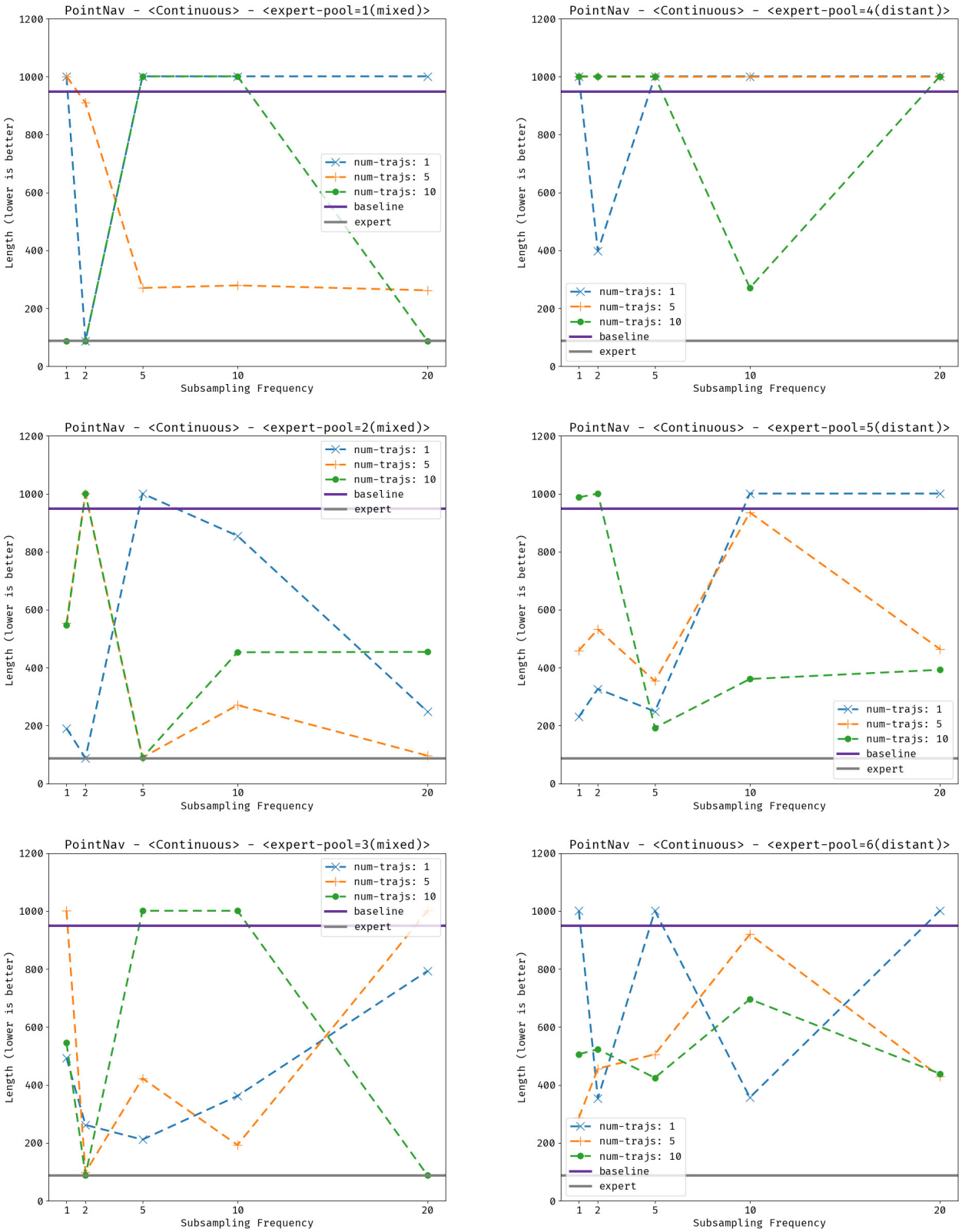

FIGURE 7.3. Episode-length m^{Length} performance of PW-IL on continuous `PointNav` with non-optimal demonstration-pools.

However, despite the fluctuations, a substantial amount of the agents perform reasonably well despite the non-optimal nature of the demonstrations. In particular, one notes that the expert performance is fully recovered in multiple instances. Evidently, these high-performing agents are mostly trained with larger trajectories-numbers $h^{\text{\#-traj}} \in \{5, 10\}$, conforming to the findings of [Dad+20a].

Solution 9. Analysis of the »distant« pools

Unlike the discrete case, where the distant pools lead to hardly any agents meeting the baseline, the continuous `PointNav` produces multiple successfully trained PW-IL models. In particular, for $h^{\text{pool}} \in \{5, 6\}$ (bottom two graphs on the right) and trajectories-number $h^{\text{\#-traj}} = 5$ (marked orange), the baseline is met for every choice of sub-sampling frequency, and for all sub-sampling frequencies of the configurations $(h^{\text{pool}}, h^{\text{\#-traj}}) = (6, 10)$ (green line in lower right plot), the trained agents manage to surpass the baseline by a comfortable margin.

The significance of this lies in the fact that the »distant« pools ($h^{\text{pool}} \in \{4, 5, 6\}$) are defined as experts-trajectories on strictly non default environments (see Table 5.2). In other words, given completely skewed demonstrations for continuous `PointNav`, the PW-IL algorithm still manages to recover the correct behavior for the true, unshifted target environment.

Similar training using suboptimal expert-demonstrations are presented in researches of the Inverse Reinforcement-Learning (IRL) methodology such as [Jac+19]; [Bro+19].

Remark 2. Further symmetry discussions

It shall be finally noted that the symmetry-heuristics in Remark 1 no longer appears to fully apply in the continuous setting, as the agents of Pool-ID 3 do not seem to collectively out-perform Pool-ID 2, nor does Pool-ID 6 consistently surpass ID 5. Also, the symmetry is lost between Pool-ID 4 and 5, as the latter displays visibly better statistics.

8. Executive Summary

From the previous analyses of PW-IL agents trained on both the discrete and the continuous variants of `PointNav`, the final conclusions regarding the method itself and its various hyper-parameter configurations are presented as follows:

8.1. **The overview.** In general, the efficacy of the PW-IL algorithm is validated through this study: with a wide range of reasonable configuration of its hyper-parameters, the PW-IL agents manage to achieve the baseline requirements. Furthermore, the expert performance could be fully recovered even if the demonstration contains expert-trajectories of another variant of the actual environment in question (see discussions 6 and 8). It is even possible to successfully train PW-IL agents even if the provided demonstrations are completely generated by experts of a different, modified environment (explained in Solution 7 and 9).

Predicting the best hyper-parameter configuration of PW-IL in an *a priori* manner is challenging. As shown in Solution 5, the theoretically optimal configuration of high trajectories-number $h^{\text{\#-traj}}$ and low sub-sampling frequency $h^{\text{s-smp}}$ with optimal demonstrations could still lead to unsuccessfully trained agents. Configurations for PW-IL is also highly environment-dependent: by simply manipulating the continuity of the action-space of the `PointNav` environment, one obtains vastly different training results and, in turn, distinct optimal hyper-parameter

values. Therefore, for every specific usage instance of PW-IL, one shall perform separate hyper-parameter search to configure the ideal number of trajectories $h^{\text{\#-traj}}$ and the sub-sampling frequency $h^{\text{s-smp}}$.

Finally, in the particular case of `PointNav`, symmetries within the environment and subsequently the (non-)optimal trajectories, as delineated in Remark 1, are sometimes subtly exploited by PW-IL to provide superior agents, as seen in Solution 7. However, the advantages brought forward by the symmetry-heuristics are not consistently taken advantage of, as observed in Remark 2.

8.2. **Unofficial guidelines for hyper-parameters choosing.** Despite the difficulties in soundly choosing PW-IL's hyper-parameters without tuning, several trends have nonetheless emerged throughout the analysis process:

Of the three hyper-parameters investigated in this study, as introduced in Section 5.2.4, the quality of the expert-demonstration, as controlled by h^{pool} in this thesis, significantly impacts the resultant agent's performance. Although non-optimal demonstrations might also lead to successful training (see analysis 6, 8 and 9), deploying high-quality demonstrations leads to dramatically improved probability of meeting the baseline and further achieving excellent performance, as seen in Solution 4.

From analyses in Solution 6, one also concludes that higher expert trajectories-numbers generally lead to better performance at larger sub-sampling values. Given sufficiently high-quality expert-demonstrations, if the state-action pairs of the expert-trajectories are known to exhibit non-trivial time-step gaps in-between, one might consider compensating this by collecting more trajectories.

Ultimately, the ideal method of retrieving the optimal configurations for PW-IL would rely on extensive tuning and parameter searches.

Activities	Duration ($\times$weeks)
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 9.2. Time-stamps of key activities

Part 5. **Epilogue**

9. Review and Projection

9.1. **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 9.2.

9.2. **Future work.** Some immediate prospects for further researches include, but are not limited to:

- Expansion of the evaluation framework to include other testing-environments such as the classic control-problems of the Cartpole and the Pendulum [Bro+16];
- Extension of the PW-IL algorithm with adaptive regularization and parameter tuning;
- Comparison of the PW-IL algorithm with other state-of-the-art algorithms, in particular, IRL methods such as Adversarial Inverse Reinforcement Learning (AIRL) [FLL17] to investigate sample-efficiency and solution robustness.

The list is definitely non-exhaustive: the reader is encouraged to continue upon the findings of this thesis to explore follow-up possibilities building upon this powerful algorithm.

10. Acknowledgements

To the supervisor of this thesis, Prof. Dr. J. Buhmann: my deepest appreciation for the encouragement of pursuing research of this project and for the inspiring and philosophical approach to the realm of machine-learning as a whole;

To my advisor, Ivan: my heart-felt thanks for patiently and gently introducing me to the fascinating algorithm of PW-IL, for guiding me with expert knowledge in the field and for providing me with countless constructive suggestions and hours of extended exchange sessions;

Finally, to my family and my friends: my sincere gratitude for the unyielding support and care during the course of my studies.

References

1. Arjovsky, M., Chintala, S. & Bottou, L. *Wasserstein GAN* 2017. https://arxiv.org/abs/1701.07875.
2. Bagnell, J., Chestnutt, J., Bradley, D. & Ratliff, N. *Boosting Structured Prediction for Imitation Learning* in *Advances in Neural Information Processing Systems* (eds Schölkopf, B., Platt, J. & Hoffman, T.) **19** (MIT Press, 2006). https://proceedings.neurips.cc/paper/2006/file/fdbd31f2027f20378b1a80125fc862db-Paper.pdf.
3. Bellman, R. *Dynamic Programming* 1st ed. (Princeton University Press, Princeton, NJ, USA, 1957).
4. Bertsekas, D. P. *Dynamic programming and optimal control* (1995).
5. Brockman, G. *et al. OpenAI Gym* eprint: arXiv:1606.01540.
6. 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.
7. 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.
8. Dadashi, R., Hussenot, L., Geist, M. & Pietquin, O. *Primal Wasserstein Imitation Learning* 2020. https://arxiv.org/abs/2006.04678.
9. 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.
10. Dulac-Arnold, G., Mankowitz, D. & Hester, T. *Challenges of Real-World Reinforcement Learning* 2019. https://arxiv.org/abs/1904.12901.
11. Fu, J., Luo, K. & Levine, S. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. *CoRR* **abs/1710.11248.** arXiv: 1710.11248. http://arxiv.org/abs/1710.11248 (2017).
12. Hill, A. *et al. Stable Baselines*
13. Jacq, A., Geist, M., Paiva, A. & Pietquin, O. *Learning from a Learner* in *International Conference on Machine Learning* (2019).
14. Kantorovich, L. On a problem of Monge. eng. *Journal of Mathematical Sciences (New York)* **133,** 15–16. http://eudml.org/doc/223721 (2004).
15. Kantorovich, L. V. *On mass transportation* in (2004).
16. Mnih, V. *et al.* Asynchronous Methods for Deep Reinforcement Learning. https://arxiv.org/abs/1602.01783 (2016).
17. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).
18. Monge, G. *Mémoire sur la théorie des déblais et des remblais* (De l'Imprimerie Royale, 1781).
19. Ng, A. Y. & Russell, S. J. *Algorithms for Inverse Reinforcement Learning* in *Proceedings of the Seventeenth International Conference on Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000), 663–670. ISBN: 1558607072.
20. OpenAI. *OpenAI Spinning Up in Deep RL* eprint: arXiv:1606.01540.

21. Pomerleau, D. A. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation* **3,** 88–97 (1991).
22. Ross, S. & Bagnell, J. A. *Efficient Reductions for Imitation Learning* in *International Conference on Artificial Intelligence and Statistics* (2010).
23. Russell, S. J. *Learning agents for uncertain environments (extended abstract)* in *COLT' 98* (1998).
24. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. *Proximal Policy Optimization Algorithms* 2017. https://arxiv.org/abs/1707.06347.
25. Silver, D. *et al.* Mastering the game of Go without human knowledge. *Nature* **550,** 354–359 (2017).
26. Sutton, R. S. & Barto, A. G. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks* **16,** 285–286 (2005).
27. Tesauro, G. Temporal Difference Learning and TD-Gammon. *J. Int. Comput. Games Assoc.* **18,** 88 (1995).
28. Villani, C. *Optimal transport – Old and new* xxii+973 (Jan. 2008).
29. Willing, C. *et al. The uncompromising Python code formatter*