DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Coverage of Behavior Spaces using Deep Distributional Reinforcement Learning for Autonomous Driving

Eesha Kumar

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Coverage of Behavior Spaces using Deep Distributional Reinforcement Learning for Autonomous Driving

Abdeckung von Verhaltensräumen mithilfe von Deep Distributional Reinforcement Learning für autonomes Fahren

Author:	Eesha Kumar
Supervisor:	Julian Bernard
Advisor:	Prof. Dr.-Ing. Alois Christian Knoll
Submission Date:	15.03.2021

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.03.2021 Eesha Kumar

Acknowledgments

First, I would like to thank my Supervisor Julian for his guidance and assistance in helping me complete successfully. Throughout the course of my thesis, Julian has helped me stay focused and not lose sight. His inputs have been valuable and have helped me deliver on a challenging research topic.

Second, I would like to thank my family for their support and patience. Their confidence in me has helped me stay strong. They have always been there to cheer me on, along with reading my work and helping me improve upon it.

Last, I am also grateful to my friends for their emotional support. Especially during the Pandemic when things seem extremely monotonous and it is easy to lose morale or feel overwhelmed, I am glad that they could help me unwind and relax. I am extremely happy that they have also taken keen interest in what I do and taken the time with proof-reading my work.

Abstract

Multi-Agent Autonomous Driving involves ad-hoc coordination by assessing and reasoning about other agents' driving behavior. Determining the driving behavior involves learning on some observations from the environment and acting on such observations. Some agents do so by incorporating additional information from the environment. In Behavior Spaces[BK20], this additional data is provided by a set of hypotheses. Existing approaches evaluate Behavior Spaces of classical learning techniques. However, there is currently no data on the effects of using Behavior Spaces with Deep Distributional Reinforcement Learning for better coverage of stochasticity from the environment. Current approaches also do not use the additional data available from past driving or simulation driving to improve training of learners.

This work presents two novel approaches along with an empirical analysis. First, the learning network, an Implicit Quantile Network(IQN)[Dab+18] is adapted and its input is augmented with beliefs for Behavior Spaces. Next, the learning paradigm is expanded by training offline using demonstration information before exploratory training. The expectation here is that some improvements are noticeable in terms of generalization and exploratory capacity.

To provide a comparative estimate, the Standalone IQN is used as a baseline. The preliminary evaluation results show that in certain aspects, generalization could be improved when the scenario gets more complex. However, while solely evaluating on the learning scenario, the Standalone IQN performs better. In terms of exploration, Learning from Demonstrations shows better scope of exploration while compared to an Expert.

Contents

1. Motivation

Multi-agent environments are found across various industries such as production, logistics, medical, surveillance and many more. These environments require coordination between their autonomous agents to complete certain tasks and achieve a goal without causing interference amongst one another. In Autonomous Driving, the aim is to provide adequate decision making capacity to an agent on the road so that it can reason about the environment and navigate without collisions and crashes. Planning behavior in Autonomous driving can be a particularly challenging task as it deals with safety and real-time decision making in complex multi-agent environments. The difficulty arises due to the imperfect state of complex multi-agent environments where stochasticity and uncertainty cause discrepancies between observed and actual events. Additionally, a multi-agent environment requires every such agent to reason about another's behavior in an Ad-Hoc manner which introduces additional uncertainty. Such observation uncertainty in terms of participating agents' behavior complicates decision-making. Further difficulties arise given the nature of such environments which are partially observable but coordination based. That is, the agents are not aware of each other's behavior but also need to drive real-time in the same conditions by reasoning about others' behavior. Autonomous agents are also expected to perform in mixed environments such as environments with human behavior where the costs of collisions are too high. Such interactions are modeled in simulation as close to real-world as possible to mitigate these compromising circumstances. Solutions often look at interaction-based decision making in simulated traffic scenarios so that it is possible to train offline in simulation to achieve a certain confidence before deploying such systems in the real-world.

Consider the scenario shown in Figure 1.1 where the ego agent must merge into its adjacent lane. The ego agent has several points where it can do so and can accurately merge in an observable deterministic environment where it is provided with accurate and real-time information about its environment as well as other agents' behavior. However, in a stochastic environment, the agent does not have accurate information and instead observes the behavior of the platoon in order to reason making a turn to merge at the right time. If done inaccurately, it risks colliding with another vehicle. In order to do so the agent needs to see enough states so as to make a reasonable assumption about its environment. The agent should also possess a certain level of generalization

so as to not overfit to a particular trajectory. This reasoning involves observing other agents' accelerations, distances and other physical measures. Observational uncertainty of such dynamics makes it additionally difficult to learn and solve such a scenario.

Existing approaches that use reinforcement learning use common deep learning techniques but few exist that leverage deep distributional reinforcement learning(DDRL) which is shown to work well with higher uncertainties. Of these, the approaches that do use distributional reinforcement learning techniques solve by modeling types[AS17] for agent behaviors. This has the limitation that human-like behaviors that are more continuous cannot be expressed. There is also no provision for allowing an expert to better configure such a scenario. Behavior Spaces[BK20] account for such continuous behavior but are not evaluated on deep distributional reinforcement techniques to better leverage the stochasticity and potentially improve exploration. Since the state space of traffic scenarios is large, it might also be possible to improve the training paradigm by utilizing existing driving data available in simulation to try learning from demonstrations(LfD) and possibly improve performance for distributional reinforcement learning.

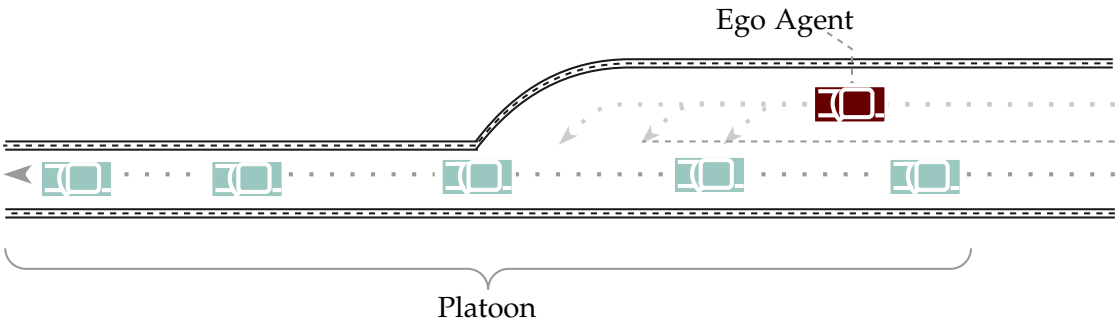


Figure 1.1.: Merging scenario where the Ego Agent is attempting to merge into the lane having a platoon of vehicles. The agent has several options to turn onto the left lane, but only some of them would result in a collision-free maneuver. Such a maneuver can be difficult since the agent first needs accurate information from the environment but cannot expect it since sensor information could be potentially noisy. Further, merging also depends on the agents ability to accurately determine the other cars' actions.

This work attempts to address these shortcomings by taking a three phase approach. First, a distributional reinforcement learner, IQN[Dab+18] is trained on merging scenarios as a baseline. Next, the IQN is trained with Behavior Spaces induced beliefs expecting an improvement in performance and exploration. Last, a learning from demonstrations paradigm [Hes+17](LfD) is established for the IQN and this variant is evaluated. This work finds that the standalone IQN outperforms its variants. However,

in certain aspects of generalization and improved exploration, the results suggest that the variants perform better.

This work is arranged as follows: The next chapter introduces important literature providing the necessary background. In Chapter 3, various other approaches to solve such uncertainties are explored and the limitations of those approaches to certain use-cases are explained. Chapter 4 is an introduction to the proposed approaches after which Chapter 5 continues with establishing the design, implementation and experiment specific parameters. The approaches are evaluated considering various performance as well as quantitative metrics in Chapter 6. Following this, Chapter 7 examines the implications of observed results while also suggesting further actions. At last, Chapter 8 winds up this work with a brief summary of the key findings.

2. Background

2.1. Markov Decision Process

The Reinforcement Learning setting uses a Markov Decision Process(MDP) defined as $< S, A, R, \gamma, T >$ for a state space (S), action space (A), reward function $R : S \times A \times S' \rightarrow \mathbb{R}$ for state transition, a discount factor $\gamma \in [0, 1)$ and transition probabilities $T : S \times A \times S \rightarrow [0, 1]$. In Reinforcement Learning, MDP is used as a model to accommodate the agent interaction with its environment where the agent explores by taking a step based on its action selection policy in its current state upon which the agent observes the reward and next state from the environment. The action-selection policy $\pi : S \rightarrow A$ is greedy and looks to select action that maximize gains at the current state.

2.2. Reinforcement Learning Setting

The goal for an autonomous agent in reinforcement learning is to search for a sequence of states and actions that can maximise its expected returns. The expected returns for an agent at state s taking an action a receiving a reward r can be given by the Q state-action value function,

$$Q_\pi(s, a) = \gamma \mathbb{E}[R(s, a)] + \gamma \mathbb{E}_{T,\pi} max_{a \in A} Q(s', a) \tag{2.1}$$

Action selection is greedy, where the behavior policy selects an action that maximizes the Q value,

$$\pi(a|s) = argmax_{a \in A}(Q(s, a)) \tag{2.2}$$

Euquation 2.1, the Bellman equations assume that the agent iterates and uses the Q value to inform the behavior policy $\pi(a|s)$ while searching for the optimal expected return Q^* until reaching the goal state.

2.3. Distributional Reinforcement Learning

In Distributional Reinforcement Learning [Dab+17], instead of learning $Q_\pi(s, a)$, which is an expectation of some random variable $Z^\pi(s, a)$ representing the rewards distri-

bution, the approach proposes learning the entire distribution instead. This mean Equation 2.1 can be rewritten as,

$$Q_\pi(s,a) = \mathbb{E}[Z^\pi(s,a)] \tag{2.3}$$

where the learning now focuses on the underlying distribution $Z^\pi(s,a)$ instead of $Q_\pi(s,a)$. In Quantile Regression and further QR-based learning methods, this $Z^\pi(s,a)$ is represented by quantiles.

3. Related Work

This work is related to three different research categories: deep distributional reinforcement learning, learning from demonstrations and exploration using behavior spaces. As stated in the motivation, solving uncertainty in dynamic environments with reinforcement learning is a challenging task. This primarily stems from factors involved in establishing a model suitable for representation of the environment when there is uncertainty in establishing behavior. This model is typically a behavior policy that helps an agent perform an action, with only partial information about another agent's action. The behavior policy is required to perform actions that maximize its expected reward while exploring and solving for a large number of states. Driving in a multi-agent environment poses a partially-observable, real-world dynamic problem. Here, the ego vehicle or agent has to sample an action from its behavior policy such that the trajectory traced by the action strategy is conducive to acceptable driving behavior.

Various techniques are employed for solving such uncertainty under partially observable conditions. These involve improving the search for optimal behavior policy using classical search techniques[BK20] or deep learning techniques[Dab+18]. Some methods involve introducing additional information about the environment during learning. Others include making improvements to the training paradigms[Hes+17].

The ego agent must select the best strategy on noisy and sparse information on the possible states. This can be improved by leveraging the large amounts of driving data available from simulations or from real-world driving.

3.1. Ad-Hoc Coordination in Multi-Agent Environments

Many real-world scenarios rely on multiple agents acting both synchronously or asynchronously. In a synchronized setting, agents share information with one another, thereby enabling a communication input for action selection strategy. The agents rely on constant and crucial information feedback and sharing for optimal performance. In [PM20], it is shown that sharing memory can improve coordination, which in some scenarios is critical. However, in the real-world driving environment, these situations are not scalable or computationally feasible since the world is highly dynamic and real-time solutions to a changing and already large state space are challenging. In

addition, since all agents' are required to learn a unified decision model, it can often be restrictive to generalizing such learning [ZL13].

In asynchronous multi-agent systems, the agents each have decentralised control and select an action strategy based on greedy assumptions. This means that the agent decides its own behavior policy to maximize its rewards while performing actions in a partially observable environment. The asynchronous decision making policy is better suited to real-world modeling since in its representation, it includes the partially observable and dynamic environment. Reinforcement learning(RL) is useful for solving such a setting as it deals with continued exploration suitable for early gathering of information when there is high uncertainty with respect to the knowledge available from the environment.

The Reinforcement Learning setting of a problem are usually goal-oriented with some reward/penalty function. Hence it relies on the Markov Decision Process(MDP) as control representation. Since the goal in Reinforcement learning is the maximization of rewards through value function approximation, it relies on a constant 'act, receive feedback and learn' loop. Prior exploration either active(self-generated) or passive(learning on others' data) is necessary for agent navigation in the environment.

3.2. Intrinsic and Parametric Uncertainty

In Reinforcement Learning(RL), two kinds of uncertainty need to be addressed while solving for a behavior policy. The first is intrinsic uncertainty and the other parametric uncertainty. The standard RL setting modeled using an MDP depends on a reward function, transitioning states and actions that inform these transitions. To best symbolize the real world, it is beneficial to model the state with certain transition probabilities to account for uncertainty in the environment. Therefore, as a result of appropriate modeling, an intrinsic uncertainty is introduced in the form of randomness with respect to the transition probabilities for attaining the next state resulting from the stochasticity of a partially observed environment. A model-free MDP [HCS13] cuts the transition probabilities out of the process, thereby forcing the agent to also learn these transitions from the environment. This ensures agent participates in the environment instead of following a rule-based navigation. Parametric uncertainty on the other hand is introduced when a learner that attempts to learn the behavior policy is required to parameterize its environment but can only do so with a finite number of samples, that is act with partial information. This uncertainty introduced with approximation of the environment can cause instability in learning since the newer samples, assumed independent of previous samples can potentially disturb the learned value function, thereby interfering with decision-making.

3.3. Deep Reinforcement Learning

Deep Reinforcement learning uses neural networks to learn approximating the value Function. The Bellman Equation used to represent the Q value function accounts for expecting the future rewards for taking action a at state s. The goal is to maximize this value using a neural network for approximating the underlying reward function. The uncertainty in deep reinforcement learning comes from approximating the value function using neural networks. DQN [Mni+13] uses an uniform-replay buffer to store enough samples from exploring the environment through game-play and perform supervised learning for learning the optimal value function. An improvement over the DQN which is the Double DQN involves a transfer of learned states using a second network to stabilize learning from the environment. Double DQN showed considerable improvement of all 49 games against regular DQN. Prioritized Experience Replay(PER) [Sch+16] shows that accuracy in attaining optimal value function can be improved with a prioritized sampling of the agents' memory. PER showed better performance of the DQN on 41 of 47 tested Atari Games.

The above listed implementations are used for the estimation of the Q value based on the Bellman Equation [Bel54]. From Equation 2.1, it is evident that the Q value is an expectation at any point in time t of the underlying value distribution. [Mav+19a] argues for a different outlook on the Bellman Equation. The argument is based on the observation that the learned Q values, that is the expected Q values are not truly accurate representations of the environment. Instead, they argue for taking a distributional perspective wherein instead of approximating the Q values using deep neural networks, they argue for learning the underlying full distribution of returns i.e. the Value Distribution whose expectation is the Q values. In doing so, their premise is that such approximations are more accurate representations of the underlying value distribution, thereby achieving a more robust behavior policy. The C51 algorithm developed with deep reinforcement learning was compared against the state-of-art DQN networks and showed that C51 significantly outperforms in over 50 of 57 games played.

The C51 uses a distributed operator for representation of solving of the distorted Bellman equation. In contrast, [Dab+17] argue for a solution from economics using Quantile Regression(QR). As a result QR-DQN Developed for the purpose of approximating the underlying distribution of returns. This is done by using QR to adjust the locations of the distribution parameterized by a uniform mixture of dirac deltas. This notion is further extended in Implicit Quantile Networks [Dab+18], where instead of learning a value distribution, a network is used to learn the quantiles that can map a distorted view of of the distribution. An extension of the IQN, Fully Parameterized Quantile Function(FQF) attempts to parameterize with a network, the task of locating

suitable quantiles. This however creates a drawback with respect to speed since the FQF is 20% slower.

Another approach to solving for uncertainty is distributed learning. Here, the multi-agent environment is leveraged for facilitation of knowledge exchange. This means that agents interact to make sense of the environment. The collective experience of participating agents in an environment is leveraged for distributed deep reinforcement learning. [SPB19] show an improved model for planning navigation in drones wherein the networks are trained on their collective experience for collision avoidance with minimum inter-communication. A transfer of learning is also possible in a distributed setting, as proposed by Policy Distillation(PD) [Rus+16] where the policy learning is "distilled" from a teacher model to a student model. This is transfer of knowledge acquired by a teacher network with better information about the environment. A distillation loss is used to control the learned policy between the student model and teacher model. It differs from a Double DQN in that it aims to solve for multi-task policy learning where multiple agent learning is distilled by the best performing agent. In a distributed setting, driven by the interest of preserving privacy of participating agents, federated reinforcement distillation(FRD) attempts to maintain a local as well as global repository of transition information for policy distillation. In this case, the agent accesses a global "proxy" that is periodically updated for determining its policy. FRD when compared with PD showed improved completion time using the Gym Cart-pole experiment. In addition, it also showed a lower payload requirement for effective sharing of memory while compared to PD.

Further approaches aim at leveraging the availability of existing data. Imitation learning [BG12] attempts to cut down on training time by allowing an expert to demonstrate a desirable behavior. Dataset Aggregation(DAGGER) [RGB11], designed to train an agent to learn policy outside its sampled state space by leveraging a supervised approach to imitation learning, thereby showing that regret-free online learning of such a policy is possible. Deeply AggreVaTeD [Sun+17] extends DAGGER to work with deep neural networks, but can only allow imitation and so a learning network cannot imitate to outperform the expert. Reinforcement Learning from Expert Demonstrations(RLED) [PGP14] attempt to solve searching for the optimal policy from fixed expert data. Here, they attempt to minimize the Optimal Bellman Residue which is the distance between estimated and optimal Q functions. The problem is challenging since the minimization is non-convex and non-differentiable. Learning from Demonstrations influenced by RLED attempts to use a supervised loss to transfer the expert transitions to improve learning. In addition, they add a pre-training step where the agent only learns from expert demonstrations. Certain other pre-trained demonstrated learning implementations such as AlphaGo [Sil+16] relies on a pre-existing model for online game play, generated by prior training of the network on demonstration data.

An upcoming research targets using Adverserial Learning with imitation learning called Adverserial Imitation Learning where the transition information that is state-actions provided by the expert are treated as finite samples representative of the target distribution. In a sense, here, imitation attempts to minimize the divergence between two target distributions. Generative networks are used to alternatively generate an estimate of how far the expert state actions provide resemble the target distribution and the other uses these as estimates for training a behavior policy. These on-policy methods require constant interaction with environment for training agents. [KNT20] attempt to train using an expert's offline samples so dependency on constant interaction is eliminated.

3.4. Modeling Complex Behavior in Multi-Agent Driving Environments

Specific to a driving environment are multiple agents making ad-hoc decisions based on a behavior policy. Desirable representation of driving behavior involve techniques that can map some physical state to a quantifiable expectation of that state. The central problem for planning in Autonomous Driving is exploration. The possible state space of tasks like lane-change, turning at an intersection, etc. are large and they increase exponentially with growing number of agents.

In [Mav+19b], the QR-DQN is used with the CARLA [Dos+17] driving simulator to show further suppression of intrinsic uncertainty can help achieve near-optimal safety. [MKH19] use QR-DQN to process camera image data and LiDAR sensor data for action selection in a highway driving environment.

3.4.1. Type based Reasoning of Driving Behavior

The type based approach [AS17] assumes there exists a fixed number of hypotheses about the driving environment to be learnt. The hypotheses include in them the stochasticity of the environment to learn. As described, the types help an agent identify a hypothesis about another agent in the environment by looking to its interaction histories. Here, the hypothesis shows how aggressive or passive an agent is about an action observed. This helps the observing agent reason about the observed agent's action selection policy. An empirical measure is the belief that the observed agent is of a certain type. In general, the type is assumed to be binary, the agent either takes an action passively or aggressively. However, [Cap19] shows that this is not necessarily the case and that an agent can exhibit mixed types. Here, he shows that reasoning the behavior of observed agents based on a mixed-type assumption shows a better

performance in complex scenarios.

Collaboration between agents in a multi-agent environment, each with its own set of types, can cause the complexity of learning a scenario to increase exponentially with types. Further, [Jia+21], observed that collaboration is stronger for agents with a larger overlap of similar types. Therefore, they suggest a hierarchical approach called Type-based Hierarchical Group Communication(THGC) model. THGC model uses some prior into pre-specified group where inter-group cooperation is facilitated by a value decomposition method. [MTA18], given that the number of participating agents can be arbitrary, the authors attempt to fix the number of types attributed to each agent. Then, a Q-learning agent is used to learn a turn-taking policy by learning to coordinate while building towers. [Hay+20] propose an approach to reason behavior by encoding situation-based information i.e. beliefs using a POMCP[SV10] solver. In this case, the history of the situation is considered entirely to improve belief planning. [RAS19] attempt to incorporate the capacity of an agent to dynamically switch types and not stick to a pre-defined type in order to model a real-world setting. Here, a CNN is used to learn the 'change' based on a Change Point Detection method, where the agent task is to detect when a change of type takes place in another agent. [Sha+20] look at the ad-hoc type-based scaling problem differently wherein they allow the decentralized agents to "choose" tasks for interaction thereby reducing the overall complexity of the problem.

3.4.2. Behavior Spaces for Planning in Autonomous Driving

The type based approach is restrictive in that it only defines a finite set of hypothesis conducive to learning. This hypothesis space may inturn be non-representative of physical quantities required for accurate modeling of the environment. To overcome this, behavior spaces was introduced where the range of observable behavior is treated as a continuous space and the resulting hypotheses and beliefs over behavior subspaces as a model for agent to reason about another agent's behavior.

Classical Search Techniques for Behavior Space Coverage

In [BK20], Robust Stochastic Bayesian Games(RSBG) is used to model the reduced complexity of continuous actions for behavior spaces. Here, an MCTS is used as a solver for RSBG to search for the optimal policy. The solver is tested using merging scenarios in BARK[Ber+20]. It is also shown that in the merging scenario, a 1D behavior space is sufficient to solve for higher dimension behavior spaces as well. This is crucial because it highlights that modeling using behavior spaces is useful for real-world problems which typically are of higher dimensionality.

The above approaches handle uncertainty in deep reinforcement learning with application across domains. This work aims to extend the existing literature by presenting approaches that trains a selected Deep distributional reinforcement learner using Behavior Spaces as well as enabling a distributional learner to learn from demonstrations. In doing so, this work aims to provide a broader scope for further analysis of methods that evaluate performance with respect to generalization and improved exploration.

4. Distributional Reinforcement Learning for Behavior Spaces Coverage

The previous chapter discussed past approaches to learning an optimal behavior policy to best determine action selection. The controlling behavior policy is often a greedy selection from the optimal Q value function. The Q value function is a measure of expected rewards for actions taken at states. The optimality criterion for the Q value function is determined by the Bellman Equation. These approaches included improvements to the network inputs as well as improvements to the training process. This chapter looks as the theoretical founding of two approaches (i) augmenting the state space with belief hypothesis from Behavior Spaces (BO-IQN) and (ii) A learning from demonstrations IQN (LfD-IQN) which trains an IQN to learn demonstration input.

4.1. Belief Hypotheses in Implicit Quantile Networks

The Implicit Quantile Network(IQN) architecture is based on learning distorted quantiles representative of the underlying optimal value distribution. The optimal value function is defined as the Bellman equation as specified in Equation 2.1.

Equation 2.1 refers to the Bellman condition for optimality, that is, find a fixed point Q such that value in the system is maximised. In the IQN setting, this is obtained by assuming that this Q value can be obtained from learned Quantiles Z. Here,

$$Q(s,a) = E_{\tau \sim U[0,1]}[Z(s,a)] \tag{4.1}$$

The quantile function randomly selects Q values for given states and actions. Over time, quantiles are learned to best represent the optimal Q values. The standalone implementation relies on sampling observation histories for state action values. The current state is sampled from a transformed physical state to learnable values.

A Behavior Space defines realistic boundaries for desirable behavior of an agent. Intuitively, it defines the space of ideal, physically interpretive values for an agent. An easy example to consider the physical interpretive aspect is as follows: Consider the distance between the vehicles, commonly referred to as Headway. In the real world,

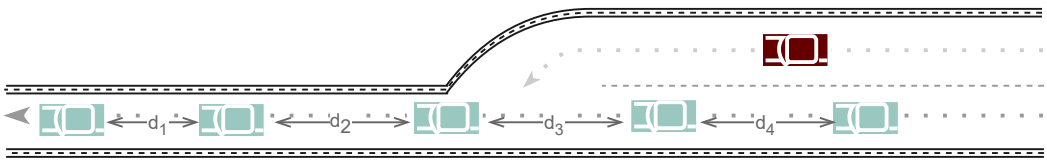


Figure 4.1.: Visualization of the merging scenario limitations in using the existing finite types approach to model varying distances $d_1, d_2, d_3, d_4 \in [d_{min}, d_{max}]$ present in the behavior space of each observed agent.Consider d_1 which can be any value defined in the continuous range $[d_{min}, d_{max}]$. If d_1 can be in continuous space and take different values at different states, existing type-based approaches are limited to only configuring a finite set of behaviors d_1.

given various perception bias in humans as well as noisy sensors, it is evident that the "true" distance is not a constant, but rather continuously varying. This inference can further be understood by considering the physical limitations of a scenario, which in this case is that the vehicles cannot drive with 0 distance between them and crash. At the same time, there is also a limit on the maximum distance to be maintained such that partial observation is still possible. Therefore modeling a behavior space for such a situation involves satisfying such limitations.

As illustrated in Figure 4.1, current limitations do not account for various agents showing different behavior at different times to maintain Headway. In order to build such a model, some physical considerations can be made. That is, there exists a maximum distance h_{max} such that the environment is observable. At the same time, there also exists a $h_{min} = 0$, such that the agents do not collide. Therefore, agents are to achieve maintaining some distance $h \in [h_{min}, h_{max}]$. Given such a behavior space, it should be noted that any value in the range is a valid distance between two agents. However, at a step t, only one such value is possible. Thus at time t, a behavior state $b_j^t \sim B_j$ represents a value in the specified range. A uniform sampling of such values can provide the necessary variance and entropy to ensure sufficient values are explored. In a partially observable multi agent environment, since the observing agent cannot know another agents behavior state, assumptions about the behaviour state can be made easier by exploiting the realistic limitations on an agent's behavior space.

Until now, the physical interpretation is still maintained, that is the behaviour state corresponds to a physically realistic value. However, this value in itself is not a useful input to a learner. This is because only a physical value is not enough to establish the behavior. Determining driving behavior in uncertainty also involves making certain assumptions about the impact of observing such a behavior in a time-

bound environment. Hence, it is necessary to map internally these behavior states to learnable 'beliefs'. Beliefs here signify a mapping from the observed space of state action(histories) and behavior space to a probability distribution based on actions. Such a belief is a 'hypothesis' over a set of actions, encoding potentially learnable information with respect to the impact of observing agent's action. Such behaviour is also continuously tracked because a behavior exhibited at state s does not have to persist when conditions change at s_{t+1}. In addition, the impact of the action a_j by the observing agent can only be after the agent has completed action a_j at time step t. The hypothesis is representative of a hypothetical policy followed by other agents. In addition, depending on the magnitude of possible assumptions, it is also possible to split the Behavior space into subspaces, each equally representative of a belief in policy π_j. Borrowing from the definition by [BK20], such a hypothesis can be represented as π_k

$$\pi_{\theta_k} : H \times B \rightarrow A \tag{4.2}$$

Equation 4.2 formalizes information implicit in interactions. Although this information is also dependant on state-action histories H, it is additional information that can potentially be useful to make better predictions of other agent behavior thereby improving driving behavior. The set of hypothesis or hypothesis sets can influence the training in certain ways.

1. For each hypothesis in a hypothesis set, train a quantile network.
2. Pre-process all hypotheses in a hypothesis set before providing the input space.
3. Concatenate all learnable hypotheses in hypothesis set to allow access to the raw hypothesis.

These values can be incorporated simply by treating them as additional information of the environment and concatenating the existing input state space with the observed hypotheses. In doing so, the learnable input states contain more information about the environment. Therefore the existing approximation learned is now transformed as

$$Z(x, a) \approx Z(s, a) \tag{4.3}$$

where $x = s \bigoplus \pi_k$.

In Chapter 5, the implementation of the architecture provided in Figure 4.2 is discussed along with results from the training performance of such a network.

4.2. Modeling of Hypothesis-based Input Space

In general, the state space is configured based on the agents nearest to the ego agent, thereby permitting an agent's observation space to be configured by a physically interpretable quantity - distance. Each agent's state is defined using certain standard physically observable attributes. Assuming the set of states observed at time step t is S_t, N_A is number of observable agents and L_A is the length of physically observable attributes of each agent, then size and complexity of state space $O(OS_t) \propto O(N_A * L_A)$. The growth of the extended state space is proportional to the dimensionality of physical behavior states, the number of splits of the respective behavior space into subspaces, as well as the number of observable agents. That is, if N_D is the number of dimensions, N_S number of behavior subspaces and N_A number of observable agents, the extended state space at a time step t denoted as ES_t, then the additional complexity of state space $O(ES_t) \propto is O(N_D * N_S * NA)$. The total new input state space at time step t obtained by concatenating the two spaces, $S_t = ES_t \bigoplus OS_t$ is of the complexity $O(S_t) = O(N_A * (L_A + N_D * N_S))$. If the length of observed physical attributes is considered a constant, then the complexity can be reduced to $O(S_t) = O(N_A * N_D * N_S)$. In addition, as shown by [BK20], a 1-dimensional behavior space is sufficient to solve for higher order behavior spaces. Therefore, the complexity is now $O(S_t) = O(N_A * N_S)$. From the final equation, we can see that the complexity of the end state space depends on the number of behavior subspaces or hypothesis generated to solve for adequate coverage.

4.2.1. Thresholding Beliefs

The hypothesis for a behavior state $\pi_k : S \times A \times B \rightarrow [0, 1]$ is a posterior distribution over actions. This hypothesis is provided as input to a neural network. Given the greater entropy of a deep distributional reinforcement learner, it is possible that if the values are of lower magnitude, they are ineffective in providing any information but rather could possibly make trouble during propagation through deeper layers, thereby causing vanishing gradients during back-propagation, where the gradients saturate due to infinitesimal changes caused by low belief values. To overcome this, thresholding can be adapted wherein the hypothesis value $h \in [0, 1]$ is grounded by some thresholding value δ. Thus, hypothesis at time step t,

$$[\pi_k <= \delta] = 0 \tag{4.4}$$

where δ is a small positive constant to ensure that infinitesimal changes with a tendency to saturate are not propagated.

In doing so, it is possible to ensure that small values do not factor as additional noise during learning, thereby decreasing the signal to noise ratio. In Chapter 5, the impact

of using a threshold is evaluated and discussed.

4.2.2. Discretization of Beliefs

The hypothesis for a behavior state $\pi_k : S \times A \times B \rightarrow [0, 1]$ is a continuous value in the range [0, 1] and as such increases the entropy during learning. Further, learning a vector of continuous values might do little in the way of providing information and contribute more to noisy learning. While some amount of noise during learning has been shown to provide farther generalizing capabilities, if the valley is crossed, noise disturbs learning and makes behaviour predictions worse. To overcome this, the magnitude of influence of the continuous beliefs values can be brought down by reducing the complexity with discretization, thereby simplifying learning with minimum loss of information during approximation. The discretization space in this case is assumed as the range of probabilities in [0, 1]. Depending on the complexity of the required discretization, a continuous-discrete map can be obtained. Thus at time step t,

$$\pi_k = \left\lceil b|_{b_{i-1}<b<b_i} \right\rceil, \forall i \in [1, N] \tag{4.5}$$

where N is the number of desired buckets.

This ensures that small fluctuations in the beliefs during training are controlled and smoothed to bucket values. In Chapter 5, the utility of discretization is evaluated and discussed.

4.3. IQN Learning from Demonstrations

The Learning from demonstrations training paradigm involves transfer learning in addition to classical exploratory reinforcement learning. In learning from demonstrations, advantages of simulation is leveraged to pre-train the agent prior to self exploration. A learner called the 'expert' is used to navigate the environment and these experiences of the expert learner are collected as demonstrations determine ideal driver behavior. The learning from demonstrations training algorithm is split into a pre-training(offline) and exploration-training(online) phase. Data is collected from expert experiences in the environment. An experience replay buffer is used to store and sample these experiences based on the annealing bias weighted priority[Sch+16]. Learning from demonstrations is a training paradigm that leverages the use of demonstrable experiences to speed up learning. It also takes advantage of the large amounts of data available from previously learned systems. During the pre-training phase, the goal is to enable the learner to mimic the actions of a demonstrator while satisfying the Bellman Equation. The learner/agent in this phase has access to only previously experienced data and

thus samples these in mini-batches for learning. In the exploratory training phase, the agent retains some of the demonstration samples while discarding a large chunk of the demonstrations data from it's experience buffer to make way for self-generated data.

4.3.1. Selection and Modeling of Losses

The loss function for demonstrated-learning in a double DQN is a weighted sum of 4 different losses - 1 step double DQN learning loss, N step double DQN learning loss, supervised large margin classification loss and regularization loss. Each loss plays a part in minimizing the relevant error - the 1 step and N step losses are used to optimize the learning network, the large margin loss is to ensure the demonstrator action always renders a lower loss than all other actions and finally the regularization loss is to prevent overfitting the network to demonstration data.

This novel implementation makes changes to the existing loss function such that relevant losses from a distributional learning network are used. To recap, the standard compound loss function $J(Q)$ is

$$J(Q) = J_{DQN}(Q) + \lambda_1 J_N(Q) + \lambda_2 J_E(Q) + \lambda_3 J_R(Q) \tag{4.6}$$

The above equation needs to be modified in distributional reinforcement learning to accommodate the changes in network training. The IQN network uses DQN networks to learn quantiles as opposed to a Q value function. Although the output of the IQN is returns upon actions, the 'implicit' learning in the network is quantiles of optimal Q values. Therefore Equation 4.6 is to be changed as

$$J(Q) = J_{IQN}(Q) + \lambda_2 J_{DE}(Q) + \lambda_3 J_R(Q) \tag{4.7}$$

$J_{DQN}(Q)$ is now replaced with $J_{IQN}(Q)$. $J_{IQN}(Q)$ is the quantile Huber Loss[Hub64] and remains unchanged since it is necessary for propagation of gradient based on the temporal difference error $\delta_{\theta,\theta'}$. The supervised classification loss for distributional learning $J_{DE}(Q)$ is to be adapted so that quantiles can be learned based on Q value differences. The regularization loss which is currently absent in Standalone IQN learning can be applied directly by a weighted sum of the learning network parameters.

This novel implementation focuses on changes to the supervised large margin classification loss $J_E(Q)$ for distributional learning with quantiles. The Deep Q-learning implementation for this loss

$$J_E(Q) = max_{a \in A}[Q(s,a) + l(a_E, a)] - Q(s, a_E) \tag{4.8}$$

is designed to propagate learning of the demonstrator actions. This is done by ensuring the penalties for all non-demonstrator actions are non-zero. That is, there

exists some loss incurred for an incorrect action selection. The loss controlling this penalty is

$$l(a_E, a) = \eta|_{a \neq a_E} \tag{4.9}$$

where η is a margin value to ensure all non-demonstrator actions incur additional cost. Factoring for $l(a, a_E)$ and taking a deeper look at the classification loss, it is evident that the difference between the Q values with some additional margin η is propagated to maximize rewarding the network for accurate prediction of a demonstrated transition. In doing so, the demonstrated transitions are expected to be learned accurately. As this can cause the network parameters to overfit to the demonstrator data, an L2 regularization loss, $J_{L2}(Q)$ is used to ensure that the parameters are penalized appropriately.

In order to transfer the supervised large margin classification loss, revisiting the quantile and Q value relationship is necessary. From the IQN literature, the relationship between the Q value and β-distorted risk measure is derived as

$$Q(s, a) = \mathbb{E}_{\tau \sim U[0,1]}[\mathbb{Z}_{\beta(\tau) \sim U[0,1]}(s, a)] \tag{4.10}$$

Equation 4.10 is the interpretation of approximating Q values at random quantiles denoted by τ. Intuitively, this means that if there exists a distorted risk measure $\beta(.)$ under which the quantiles are to be selected at random τ, then the resulting integral weighted by the distortion measures is an approximation of the Q value. In the discrete domain, this integral is the expectation over all such distorted quantiles. As 4.10 shows, the Q value at the end of each learning step is the expected value of the sampled quantiles. Since the quantiles are randomly sampled, it ensures adequate exploration for attaining optimal quantiles and subsequently Q values for optimal behavior policy. Extending this to distributional $J_E(Q)$ in Equation 4.8, and substituting for the Q value from equation 4.10, the resulting supervised large margin loss is

$$J_{DE}(Q) = max_{a \in A}[E_{\tau \sim U[0,1]}[Z_{\tau' \sim U[0,1]}(s, a)] + l(a_E, a)] - E_{\tau \sim U[0,1]}[Z_{\tau' \sim U[0,1]}(s, a_E)] \tag{4.11}$$

4.3.2. Importance Sampling for Experience Replay Buffer

An experience replay buffer is a data storage that holds the explored transition state samples i.e. state, action, next state, reward and additional metadata. The replay buffer is prioritized with sampling probability proportional to the assigned priority. The probability of sampling a particular transition i is given by,

$$P(i) = p_i^\alpha / (\sum_k p_k^\alpha) \tag{4.12}$$

where $p(i)$ is the priority of the transition i calculated as,

$$p(i) = \delta_{(i)} + \epsilon \tag{4.13}$$

where $\delta_{(i)}$ is the temporal difference error obtained from the learning and the target network. ϵ is a small value added to assign a minimum priority to the sample in case the temporal difference is 0. ϵ is also leveraged to assign different priorities to different types of transition. In this case, the transition sampled during exploratory training can belong to either the demonstrating expert or agent. To account for proportional sampling of the memory, that is, to ensure adequate demonstrator data is sampled during exploration, ϵ_d is assigned for the demonstration data and ϵ_α for the agent data where

$$\epsilon = \epsilon_d >> \epsilon = \epsilon_\alpha \tag{4.14}$$

So far, the sampling priorities deal with insertion and retrieval of the transition. However, it is also important to factor in the changing distribution. To do so, weights can be assigned to transitions based on the priority to mark "importance" of the transition samples' updates to the network. This can be done by Importance Sampling[Sch+16] where weight for transition i is calculated as,

$$w_i = (N * P(i))^{-\beta} \tag{4.15}$$

where P(i) is the priority calculated in Equation 4.12. $\beta \in [0, 1]$ is a control introduced where if β is 0, there is no importance and if β is 1, full importance is given to the sample. In practice, β is usually annealed linearly from some starting β_0 to 1. Intuitively, this means that the importance to samples are updated for samples recorded later during training, since by these states, the agent is expected to have partially learned about the environment.

The design of a PER buffer is shown in Figure 4.3. Here, the differences of memory storage are highlighted during the pre-training and exploratory training phases.

4.3.3. Pre-training with Expert Experiences

In the pre-training phase, the experience replay buffer is filled with data from a demonstrator agent. The demonstrations are past states explored by an agent in the same environment. The weights for all samples initially are equal. Based on the annealing bias sampling method, the probability of sampling an experience is directly

proportional to the temporal difference obtained during learning. The idea is to ensure that those samples with a lower Q value estimate or outlook are sampled more often. The probability is proportional to p_i which depends on the temporal difference error as well as the priority based on type of experience - demonstrated or self-generated. In the pre-training phase, the type of experience is irrelevant since all experiences are demonstrated as as such only the temporal difference error factors for sampling. In the pre-training phase, a supervised learning paradigm is established by using the expert experiences as training data for the learning agent. The experiences are stored in an experience replay buffer and sampled according to the annealing bias.

4.3.4. Exploratory training with Mixed Experiences

In the exploratory training phase, the agent retains some ratio of demonstrator experiences in its replay buffer and performs ϵ greedy exploration to sample actions from the learning online network. Only the self-generated data is overwritten when the replay buffer is full. The demonstrator data on the other hand is left untouched so that there is proportional sampling during mixed-data sampling. The exploratory training after the pre-training phase is to ensure the network is trained with initial knowledge of the environment. Here, the training is not solely supervised training but involves agent taking steps in the environment by sampling from an epsilon-greedy behavior policy. The agent determines if the action is to be random or evaluated on the network to ensure there are adequate instances for the network to generate data for its buffer.

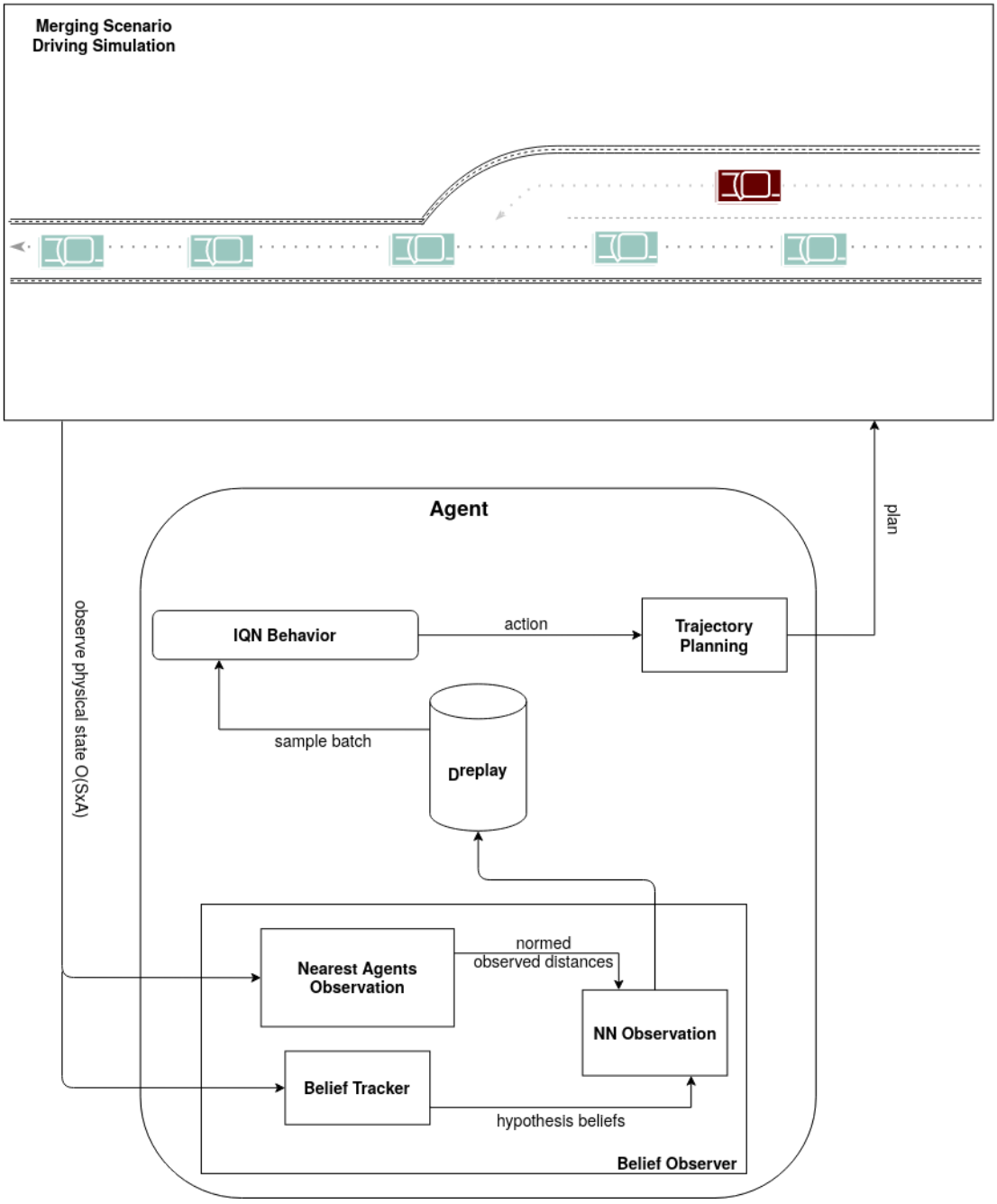


Figure 4.2.: Architecture overview describing the Belief Observing IQN for an agent's interaction with driving environment using beliefs updated online by the Belief Tracker.

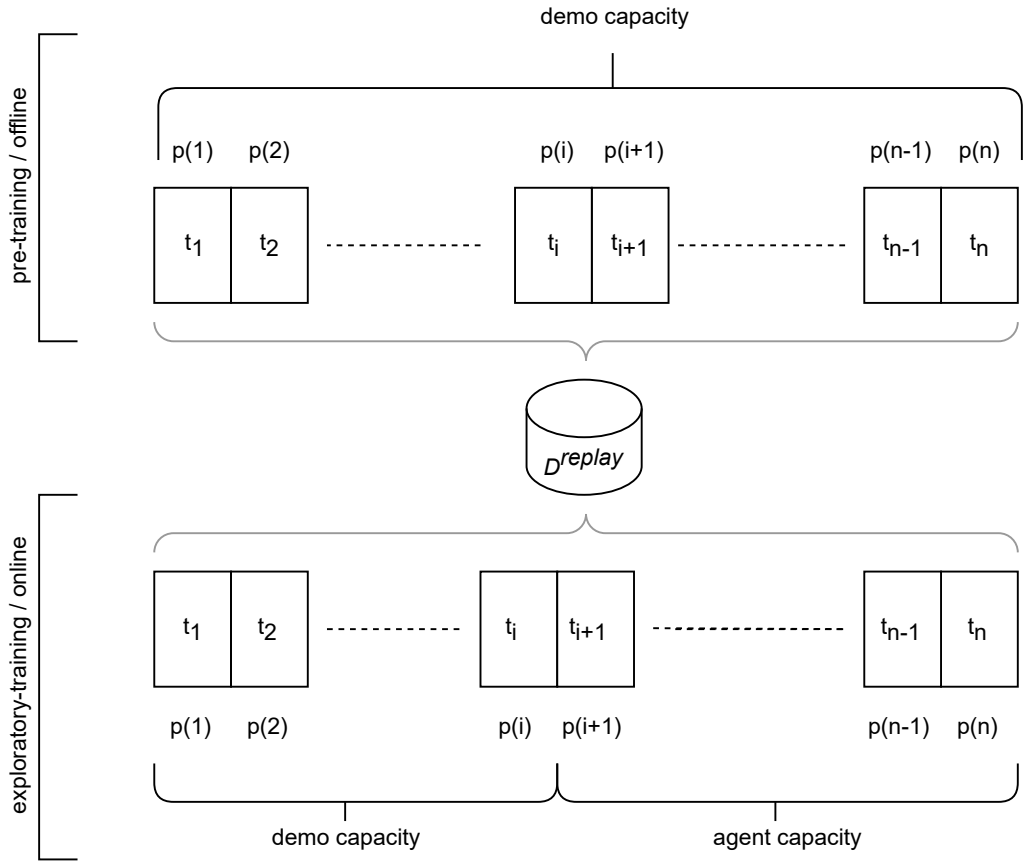


Figure 4.3.: Memory Architecture highlighting the partition differences necessary to allow for sampling the same memory during pre-training(offline) and exploratory training(online) phases.

5. Architecture of Simulated Experiments

The previous chapter gave a detailed explanation of the suggested approaches. In this chapter, the simulation setup and then the experiment setup for each of the variants is detailed. In addition, the training results are also provided to get a preliminary overview of the variant's performance.

5.1. Setup and Implementation

Here, a real-world simulation of a driving scenario is considered and the suggested approaches are implemented to solve driving behavior in the scenario. The setup of the run-time environment is leveraged from the BARK[Ber+20] simulation environment. The agent-specific training setup is borrowed from an open-source PyTorch implementation of IQN[Wat13]. The setup was extended to include bark simulation specific state transition information. An experiment in the context of this work is described as an end-to-end training and evaluation run. First, the overall architecture and experiment design is presented and described. Then, the implementation specifics of training an IQN Agent to learn a Q-value distribution and the behavior policy are discussed. Following this, changes to the observed hypothesis belief space (Belief Observer) are introduced and described. At last, the experiment implementation and design in the learning from demonstrations training paradigm is discussed. The traffic simulation used to run the various experiments is the BARK simulation environment. The BARK environment is open-source, easy-on-boarding and offers broad configurability.

The setup of the experiment including persistence, extraction and usage of agent, environment and accompanying parameters is done by the Experiment Setup in Hythe[Kum13]. The experiment setup controls the execution specific parameters so that it is possible to configure various BARK provisions such as the Environment, Observer, Behavior, Scenario and Evaluator with Distributional Reinforcement Learning Agents.

The experiment execution is configured such that each variant receives adequate time training. All experiments are run for 50000 episodes. At each training step, the agent stores transition information. In specific intervals, the agent tries to learn by sampling from its memory. The agent also has evaluation steps configured for 5000 steps. These

evaluations are repeated every 2000 steps. The return at each step in the evaluation is averaged and the mean of returns is recorded as the evaluation return.

5.1.1. Environment and Scenario

The environment implementation is an OpenAI-Gym wrapper over a bark runtime environment. The gym-bark environment extends the Bark Runtime as well as the Gym environment. The scenario for driving behavior simulation is a highway merging lane scenario where the ego vehicle, let us assume on the right lane must merge into the left lane, as shown in Figure 5.1. The right lane is populated by a platoon of vehicles, one following another separated by any distance $d_i \in [d_1, d_2]$. Here, d_1 and d_2 are expert configured as limits for a vehicle distance sampling range. Control over this variable enables an expert to control the density of traffic. The platoon of vehicles on the left lane are configured randomly, although the agent observes the neighboring vehicle to make assumptions about its behavior based on nearest distance. Hence, at each step, the transition information recorded by the ego vehicle varies by the closest n other vehicles. Each agent in the scenario is configured by its own behavior policy. At each step, every agent in the scenario samples and action based on its behavior policy. This is to maintain consistency while being modeled by the same environment. An episode in this merging scenario terminates when one of three incidents listed below take place:

- The ego vehicle collides with another vehicle or with the driving boundaries.
- The ego vehicle does not reach goal nor does it collide within the allowed steps.
- The ego vehicle reaches the adjacent(left) lane goal.

The reward setting is goal-oriented and therefore categorical. From the three possibilities listed above, it is possible to allocate rewards as $r(s, a) \in \{-1.0, 0, +1.0\}$ where $r(s, a)$ is the reward received by the ego agent at state s upon performing action a. From the representation it is quite evident that the reward setting is sparse where the agent receives 0 points most of the scenario.

The merging scenario is fairly complex since the ego agent must make reasonably accurate assumptions about numerous other agents' driving behavior policy in order to avoid collisions. To make such assumptions, the ego agent learns an observation. An observation here is realistic and physical attributes representative of the participating agents. The physical attributes are part of the observation which is in turn part of the state. The scenario state representation 'observed' by the ego agent is specified in Equation 5.1

$$(x_{ego}, y_{ego}, v_{ego}, \theta_{ego}, x_{1n}, y_{1n} ..... x_{Nn}, y_{Nn}, v_{Nn}, \theta_{Nn}) \tag{5.1}$$

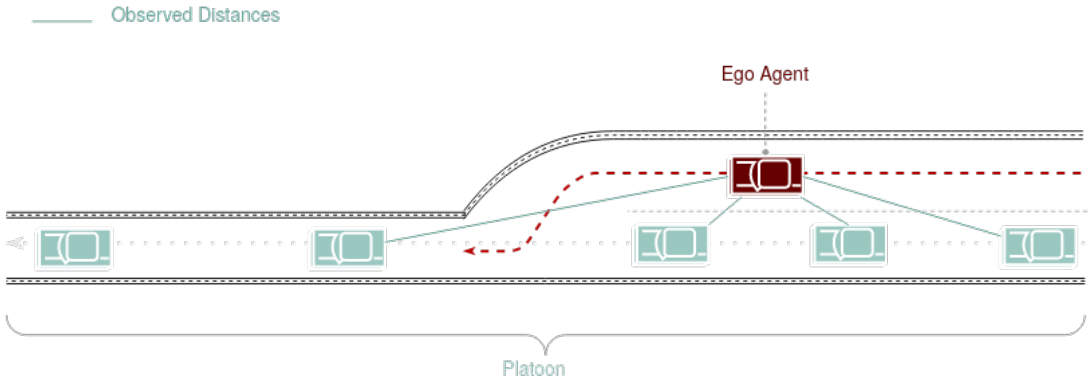


Figure 5.1.: A merging scenario where the Ego Agent in red is attempting to merge into the neighboring lane with a platoon of vehicles. The Ego Agent has a range of observation configured to the 4 nearest agents marked 'Observed Distances'. The trajectory marked for representation in red shows a possible left turn the ego agent could make to enter the left lane and achieve its goal.

where x_{1n}, y_{1n}... are physical attributes of 1-nearest agent and $x_{Nn}, y_{Nn}, v_{Nn}, \theta_{Nn}$ are corresponding attributes of N-nearest agent.

Here, the agent observes the position, velocity and steering angle of its own as well as N nearest vehicles. This information is transformed by some mapping $\phi(x, y, v, \theta..)$ where the agent states are pre-processed before producing a state to be treated as the state input for the learning network.

The standard scenario is driving in a highway, attempting to merge from the right lane to the left lane. The left lane contains a platoon of vehicles driving in the same direction. The goal is to drive without collisions onto the neighboring lane. The space of possible actions is discrete with 5 actions for varying accelerations at $a_{1-5} = 0.0, 1.0, -1.0, -2.0, -5.0$. The next two are *right* and $left$ turn respectively. The last action is one for maintaining the desired minimum gap. The scenarios used to evaluate have lightly dense to medium dense scenarios where the range between the vehicles is varied. The purpose of varying the scenarios is to establish a hypothesis depending on the behavior of the IQN agent is to compare between free vs restricted exploration.

In the light dense scenario, the vehicle maintains a gap distance in the range between 12 and 16 meters and in the medium dense scenario, the gap distance adhered to is 6 and 10 meters. These minor variations in vehicle distance range are adequate to ensure that there is variation in the state space for ego agent learning. The classic IDM is used to establish relevant uniform sampling for the adopted range maintained by the agent. Here, the ego vehicle is trained, evaluated and bench-marked for performance metrics

on the same scenario.

5.1.2. Behavior Model and Sampling with IDM

The action taken by an agent is sampled based on a behavior policy. The behavior policy selects an action that maximizes the Q-value as specified in Equation 2.2 which are obtained from a forward pass on the training network. A behavior model acts and plans agent's driving behavior in the environment by means of its defining behavior. The behavior of an agent is configured based on the scenario and can be continuous or discrete. In the context of training a deep distributional reinforcement learning network however, the possible values for a behavior policy is limited to a set of values. This means that there exists a certain space of categorical values as possible outcomes of a training network. The accompanying mathematical interpretation would be

$$\pi^*(s,a) : S \times A \rightarrow [a_1, ..., a_N] \tag{5.2}$$

that is, there is a set number of discrete actions possible by the agent. The behavior model prescribes mainly behavior with maximum utility to explore and observe a vast number of states. Given that the scenario state space is large, adequate exploration can be controlled by discretizing the action space. For the purpose of this experiment, this action space is limited to the necessary utilities in the merging scenario which are braking and acceleration values, gap keeping action and lane change action. The ϵ-greedy sampling allows for random(exploration) as well as predicted(exploitation) sampling of actions. In an explore step, an action is uniformly sampled by the behavior. In an exploit step, the action that maximises the current Q-value distribution is sampled. Apart from the Act stage, the agent also has a planning stage which updates the trajectory and behavior of the agent. This is necessary since the BARK framework needs past trajectory to update and model the agent in the environment.

The agent actions, that is applied acceleration or deceleration values, are sampled according to the Intelligent Driving Model(IDM). IDM was initially designed to be a car following model but there were considerable drawbacks to such a model. The car following model expects an agent to be driving in front of it, in the same lane. In a merging scenario where the ego agent merges into the goal lane from another lane, this is not possible. It also raises concern of how the behavior model of each agent in the scenario modeled by its own behavior policy is to coordinate when one agent cannot observe another agent since it might be merging into the goal lane. This problem is mitigated by 1) Introducing a 'Coolness Factor' control as deceleration heuristic 2) Allowing the agent observation state to account for other agents position relative to the ego agent. These parameters are desired constants and maintained by agents in the scenario. A behavior configured with such parameters is the Classic IDM. The Classic

IDM model is used as the default Behavior for all vehicles in the scenario experiments. However, in the case with hypothesis-based belief observation space, the Classic IDM is extended as a Stochastic IDM so sampling over a predefined distribution is possible. Here, if needed, a Uniform Sampling over the Behavior Space is enabled and so, it is possible to have varying values as the underlying parameters for the ego agent's behavior. In doing so, it is possible to construct a more realistic setting for driver behavior.

5.2. Standalone IQN Learning

An open-source setup of an IQN network is adapted for the purpose of simulating driving behavior in traffic scenarios. Much of the setup is preserved while adapting the network for traffic simulation use-case. The current and industry-standard focus with IQN networks are based on using games as simulations and images as states in transition information using the Atari Learning Environment(ALE). This however is not useful in the driving behavior simulations since the representation of transition information is different. Particularly the state representation is a 1-dimensional space as opposed to RGB channeled image for games. In practice, training on images involves the use of Convolutional Neural Networks(CNN). However, these networks tend to be large and unnecessary to process 1-dimensional state input. To process 1-dimensional state input, a Fully Connected Network(FCN) is used. In addition, the available implementation was focused on processing images and so optimized to store images as states with industry-standard normalization techniques. These do not apply in the state representation case as the normalization and other related operations are relative to the driving behavior in a multi-agent coordination environment. The normalization techniques suitable for images are eliminated so there is no unidentified tampering with state data potentially causing a vanishing gradient problem.

The IQN learning framework follows closely with the DQN learning. Like in the DQN architecture, IQN also configures two networks, online and target where the online network is trained on all states and the target network is updated every t steps to stabilize learning. The IQN uses a Huber loss over the temporal difference error so as to be less sensitive to outliers.

Reasonable generalization is typically achieved with good exploration. Hence, the Standalone IQN Agent is trained on the merging scenario by varying certain exploration-exploitation specific parameters. In order to improve exploration and subsequently driving success rate during merging, it is necessary to baseline the network using various configurations of the ϵ-greedy parameters. In the ϵ-greedy approach, the agent has ϵ probability of exploration. Since, in the beginning, the agent is to explore more

and gather information about the environment, the ϵ is highest in the start and decays linearly to a specified end value. In the context of this work, the end value is configured with higher and lower limits to evaluate the effects of a small slope of epsilon decay. The resulting training accuracies and returns were compared and the best set of exploration parameters were selected.

The IQN, unlike standard C51 and QR-DQN, does not output a value distribution over actions. Instead the IQN learns quantiles which are randomly transformed at τ locations to obtain points on the value distribution. This is done by using a cosine embedding network to learn and select specific locations more conducive to producing outcomes at higher scores. That is, the cosine network learn to embed or map τ such that these embeddings correspond to selecting τ such that resulting quantiles are maximized. As show in Equation 4.10, the Q value can be obtained using a weighted sum over quantiles.

5.2.1. Network Design

For the purpose of the experiment, the standalone IQN architecture was modified to accommodate for 1-D state inputs as well as scaled down in terms of learning units from a CNN to an FCN. The network also has fewer hidden units as compared to those necessary to train using images. The IQN in turn comprises of a collection of networks as listed below;

- States Embedding Network
- Cosine Tau Embedding Network
- Quantile Network

A visual representation of of the Network Design is provided in Figure 5.2.

The State Embedding Network is the feature learning network where states are received as network inputs and transformed using some embedding function ϕ as is the use in standard DQN. The states are embedded and parameterized in a higher dimensional space for improving representation. The transformation is applied after 3 layers of fully-connected hidden learning units with each followed by a ReLU non-linearity. The final output is embedded input states. The initialization of the network weights is given by He[He+15] initialization. If S is the state space and σ the mapping functional, equation 5.3 shows the mapping functional from input space to the embedded space facilitated by consequent fully connected neurons forming a Multi Layer Perception(MLP).

$$\sigma : S \rightarrow \mathbb{R}^e \tag{5.3}$$

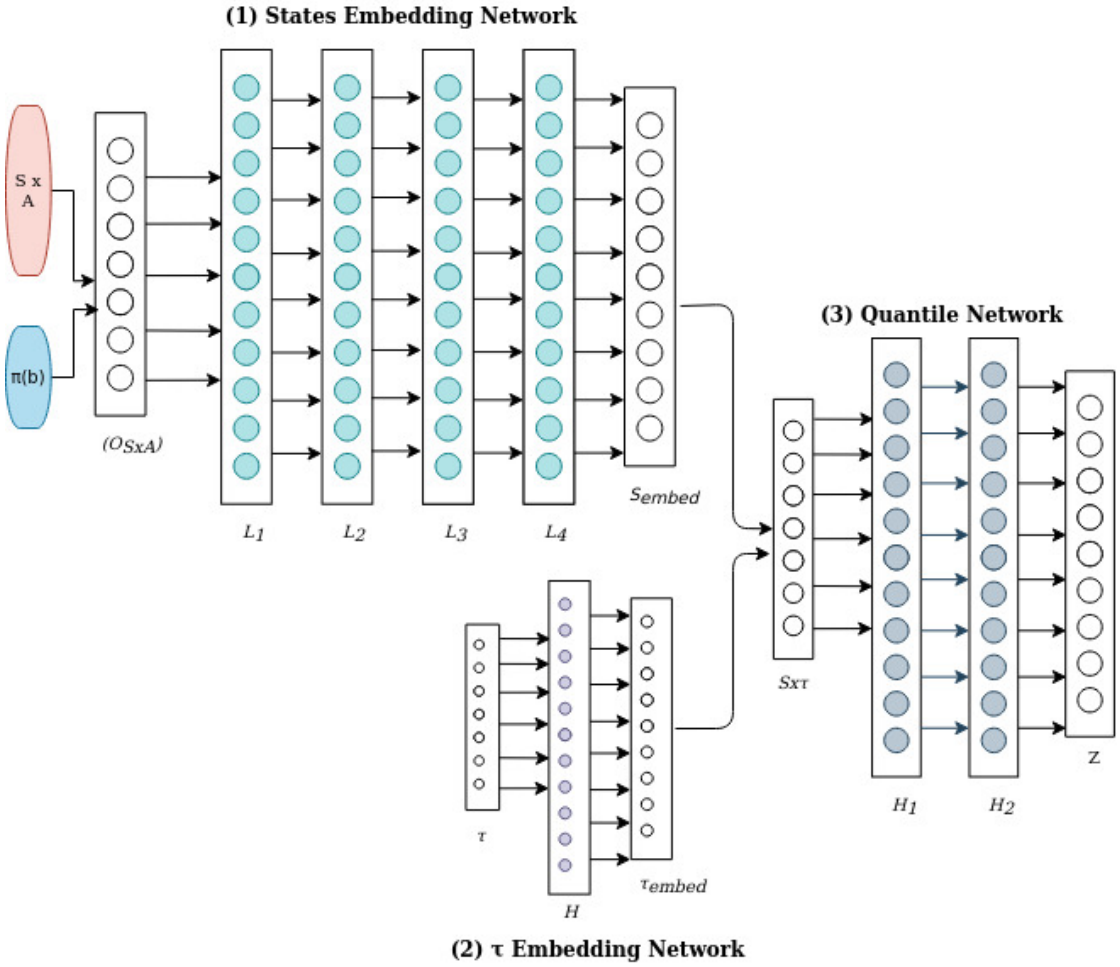


Figure 5.2.: A simplified representation of the Belief Observing IQN Network with fully connected layers $L_1, L_2, L_3, L_4, H, H_1$ and $H_2 \in R^{512}$. The Output of the Final Network, that is the Quantile Network, is the predicted action.

$$s_{embed} = \sigma(s), s \in S \tag{5.4}$$

The Cosine Tau Embedding network is similar to the state embedding network except the transformation embedding the τ values are applied over a cosine transformation of the selected τ points. The embedding range's dimensionality is to be consistent with the state embedding network $\sigma(s)$.

$$\phi : \mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{R}^{e} \tag{5.5}$$

$$\tau_{embed} = \phi(\Sigma_{i=0}^{N-1} cos(i * \pi * \tau)) \tag{5.6}$$

where $\phi(\tau)$ is the mapping applied by the embedding network. e is the size of the resulting embedding dimensions.

Equation 5.6 shows the mapping for τ using cosines prior to transformation by the learning network. The network has no hidden layers but has a ReLU non-linearity. The lack of hidden layers also shows that the network forces multiplicative interaction of τ in a low dimension space. The non-linearity over this cosine promotes better learning as shown in the IQN experiment. The τ mapping is also to embed in a space of the same size as $\sigma(s)$.

The final network in the IQN is the quantile network which combines the result of the previous two networks. The operation for combining the two embedded outputs is a merge function which takes the state and τ embeddings and performs an element-wise multiplication(Hadmard[Kaz19]). This ensures that the two embeddings interact early thereby allowing for a shallow learning in the Quantile Network. Since the Quantile function maps quantiles to actions over τ selected points, this function can be represented as

$$q_{out} : \mathbb{R}^{N} \rightarrow \mathbb{R}^{|\mathfrak{A}|} \tag{5.7}$$

where $|\mathfrak{A}|$ is the configured number of discrete actions. The final output of the IQN is select points on the resulting quantiles approximating a value distribution located by τ for each action in the action space $\mathfrak{A}$. As specified in the IQN literature, this can be represented as

$$z = q_{out}(\sigma(s) \odot \phi(\tau)) \tag{5.8}$$

Since τ is sampled randomly at each learning step, adequate steps need to be taken to ensure that enough values of τ are sampled such that a good enough representation of the underlying value distribution is possible. From the IQN paper, these values, N, N' each for sampling τ from the learning and target network work best at 32 and 64 respectively.

5.2.2. Exploration Tuning

Since the goal of the experiment is improved exploration, various epsilon specific parameters were tried to ensure a strong baseline. ϵ-greedy training parameters were selected with varying slopes for decay and varying end exploration values. Of these, the best was found to be the default set of parameters. Figure 5.3 shows the training and evaluation accuracies for various exploration decay and end exploration value parameters.

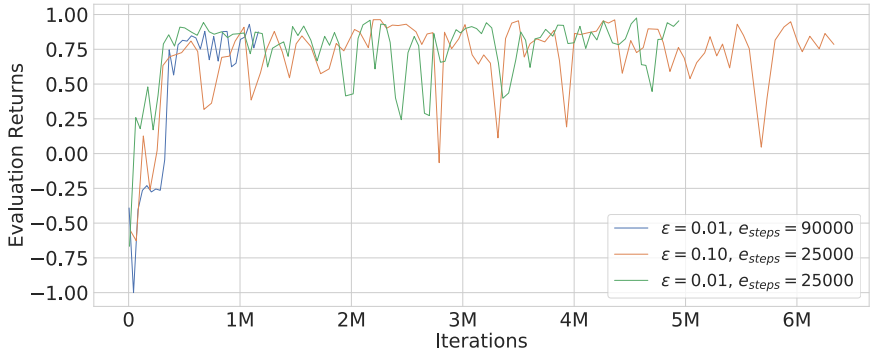
5.3. Belief Observer IQN Learning

As discussed in the previous chapter, the state space of a regular observer is expanded to include a set of hypotheses. These hypotheses are continuous values that reflect some probabilistic 'belief' in certain behaviour states of a behavior space. The expectation of expanding the input state space is to improve exploration of greater number of states, thereby also trying to improve the generalization capacity.

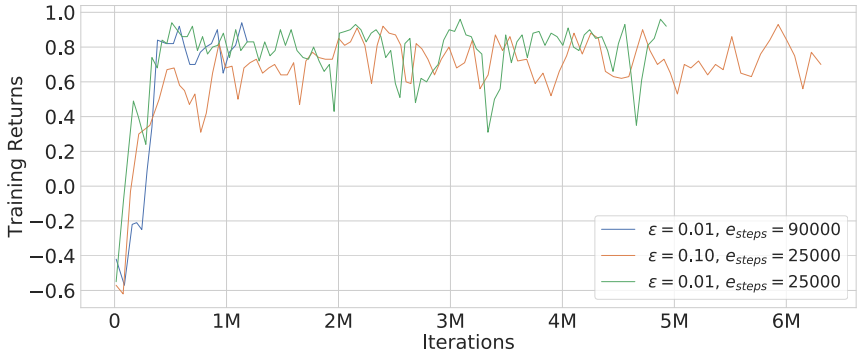
The standard observer implementation in BARK represents states by its location, speed and orientation in the environment. This is extended to accommodate the behavior space sampling using a stochastic representation of IDM. The Stochastic IDM in BARK is setup to extend from the existing classical model with the addition of a distribution accompanying the sampling parameters. Hence, parameters pertaining to certain limits of the behavior space can be uniformly sampled with a Stochastic IDM and give a more accurate representation of real-world driving behavior. The internal mappings of a behavior state to its beliefs are invisible to the network during training and the network is only concerned with the beliefs for learning. There exists a BARK implementation of a Belief Calculator that handles these calculations and can be used to obtain obervations from the environment. As mentioned previously, the dimensionality of beliefs depends on the number of observable agents, the dimensions of Stochastic IDM hypotheses and the number of splits configured for a Behavior Space.

In this work, a Belief Observer is implemented to incorporate beliefs in the ego agent's observations. This is done by extending the existing Nearest Agent implementation and concatenating a beliefs vector to the nearest agents' state vector. Since the belief calculator calculates beliefs for all non-ego agents, it is necessary to obtain only the N-nearest agents. Since this was not present in the existing implementation, changes were made to order and filter N-nearest observable agents. This information is then relayed to the Belief Observer to further filter beliefs depending on N-nearest agents. The resulting observation space is treated as the state input for learning.

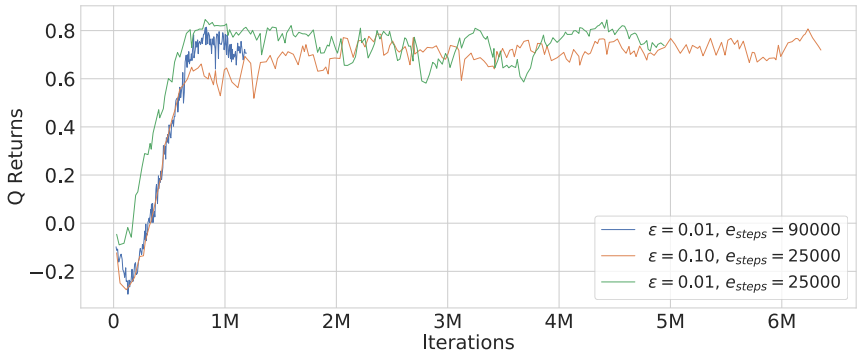
Algorithm 1 illustrates the changes incorporated in expanding the observation space



(a) Evaluation Returns on various exploration parameters



(b) Training Returns on various exploration parameters



(c) Q Returns on various exploration parameters

Figure 5.3.: Performance metrics obtained from varying exploration parameters to gauge influence of exploration decay steps e_{steps} and end decay value ϵ.

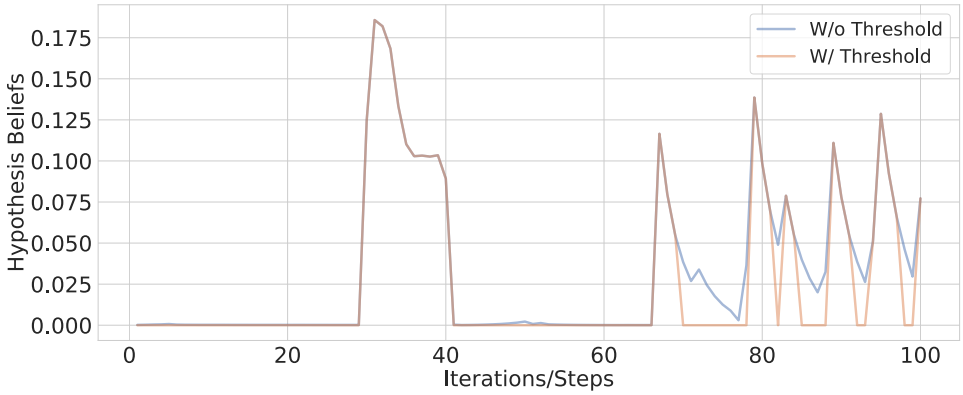
of the ego agent.

Algorithm 1: Belief Observer Algorithm with belief-expanded input observations based on proximity of other agents

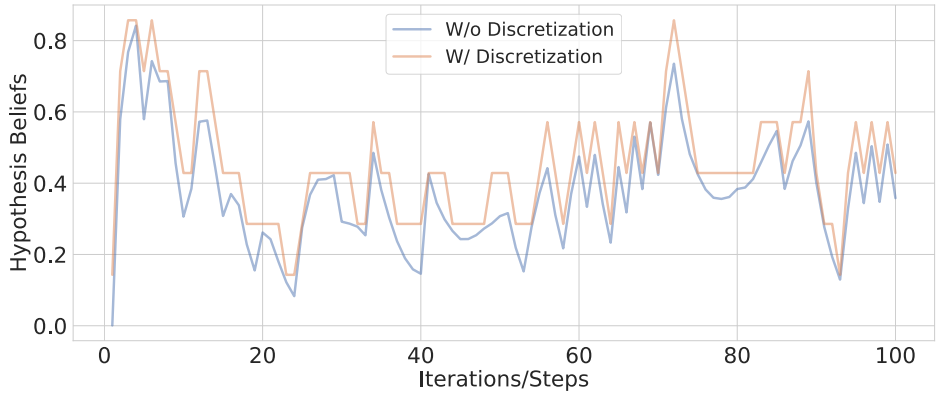
input : td_0, D_0, N_s
output: $beliefs$
$td \leftarrow td_0$
$D \leftarrow D_0$
for $i = 0$ **to** N_s **do**
  $beliefs \leftarrow \pi(s_i, a_i, b_i)$
  `// Get Nearest Agents List`
  $ag \leftarrow [a_1, .., a_N]$
  $beliefs \leftarrow map(ag : beliefs_{ag})$
  for $j \in ag$ **do**
    $beliefs \leftarrow beliefs \odot beliefs_j$
  end
  if *threshold* **then**
    $$b = \begin{cases} 0, & b \leq td \\ b, & otherwise \end{cases} \qquad \forall b \in beliefs$$
  end
  if *discretize* **then**
    $b = d_{i+1}, \qquad d_i <= b < d_{i+1} \qquad \forall b_i \in beliefs$
  end
end
return $beliefs$

5.3.1. Thresholding to Reduce Irrelevant and Noisy Beliefs

Belief Thresholding is introduced as an option to analyze and possibly mitigate the effects of vanishing gradients. Since the learning network is quite deep and the probabilistic belief values small, it is possible that propagation is not sufficient past the initial hidden dimension, which would cause small gradient changes to be observed during the back propagation. Thresholding is therefore used to control the gradient flow and facilitate learning. The thresholding value is set quantitatively by analyzing density of belief values with marginal δ differences. Figure 5.4a shows one such case where marginal differences can introduce unnecessary noise during learning. In Chapter 6, the evaluation also considers the ego agent performance with thresholding applied for belief values against non-thresholded beliefs.



(a) A sample plot visualizing noise thresholding of the input beliefs. Noisy values below 0.05 set threshold are flattened to 0.



(b) A sample plot visualizing discretized beliefs of the continuous hypothesis space. 8 discrete bins are used to fit to the continuous beliefs.

Figure 5.4.: A sample plot of the thresholded and discretized beliefs of the continuous hypothesis space π_k.

5.3.2. Discretization to Reduce Dimensionality and Noise

Since the beliefs are continuous, the complexity of learning such values is increased considerably. As an option to control this complexity, the probability space is discretized using D points located uniformly. The discretization is a pre-processing step and a quantitative approach is used to evaluate the success rates of different discretization values by varying D in the same scenario across different experiments. In Chapter 6, these evaluated results are across different dimensions of discretization in scenarios of varying difficulty. The expectation is that there exists a possible discrete value wherein the complexity of the network is reduced(adequate discretization with reasonably low D) while not compromising on generalization. Figure 5.4b shows discretized values for the underlying continuous hypotheses.

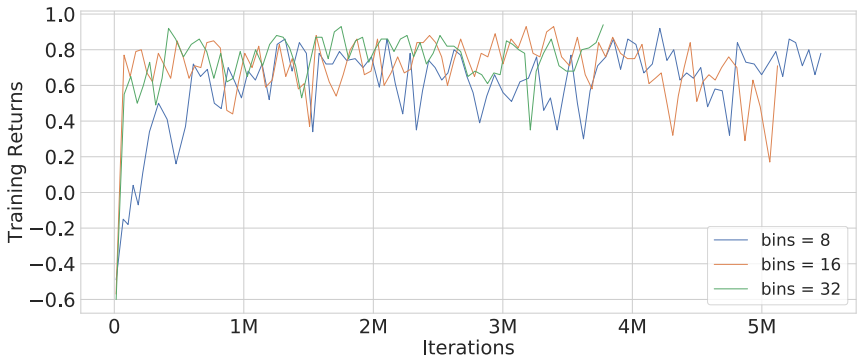
As a result of discretization depending on hyperparameter D, it is necessary to evaluate the training performance. Figures 5.5 and 5.6 show the training and evaluation results of thresholded and discretized Belief Observing IQN.

5.4. Learning from Demonstrations

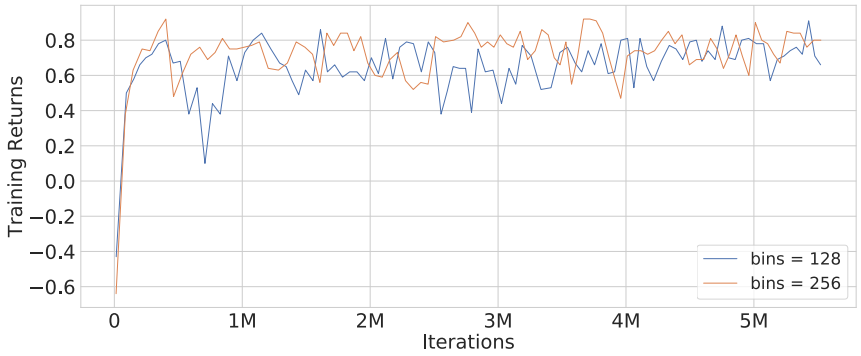
As summarized in the previous chapter, Learning from Demonstrations(LfD) is an algorithm developed to promote training of an agent on expert-generated data before exploring and generating data from the scenario. This expert-generated data is demonstrations which are used as the only training data by the agent as pre-training observation histories of the environment. In this work, a novel approach is attempted where the Learning from Demonstrations algorithm is applied to the IQN network instead of a DQN network to evaluate possible performance improvements of the IQN network. The expert learner in this case is a Multi-Agent Monte Carlo Tree Search(MAMCTS) using both a Nearest Agent-observing IQN as well as a Belief-Observing IQN. The implementation of LfD had to be made from scratch. In the interest of maximizing reusability, as much as possible from the existing implementation was leveraged for Learning from demonstrations.

The learning from demonstrations implementation is divided into three self contained components as listed below

- Annealing Bias Prioritized Experience Replay Buffer
- Learning from Demonstration Loss functionality J_Q
- Setup of Online and Offline demonstrated learning



(a) Evaluation Returns on changing discretization parameters

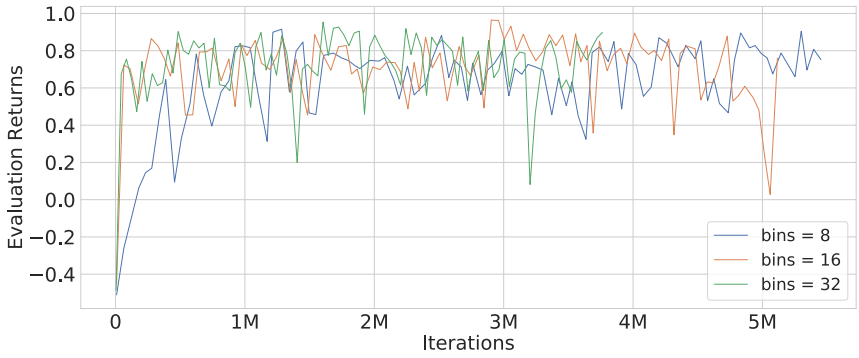


(b) Training Returns on changing discretization parameters

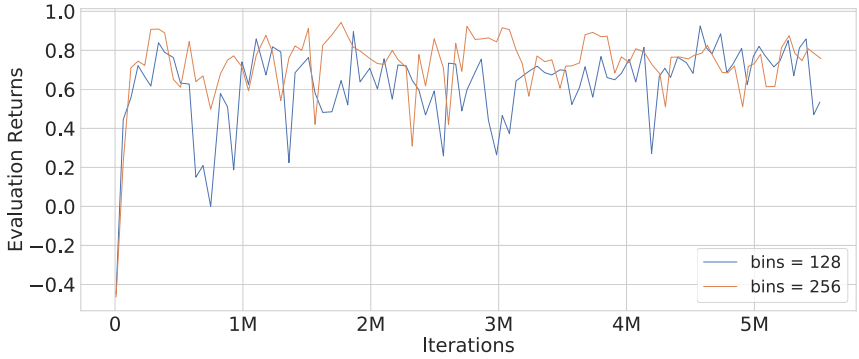
Figure 5.5.: Performance of the Belief Observing IQN with varying Discretization Parameters

5.4.1. Importance Sampling and Prioritized Experience Replay Buffer

The LfD literature specifies the use of importance sampling for assignment of transition information weights. Annealing bias is used for assignment of priorities. These priorities in turn determine the probability of sampling a transition. In addition, the LfD can be thought of as two different training steps executed successively. The first is the pre-training or online training phase where the agent is trained only on demonstrator data. For this purpose, the memory is filled solely with demonstrator-generated samples. The agent learns with mini-batch updates where each learning step is a forward pass through the network for learning the state, τ and quantiles of only demonstrator-generated transition information. Hence in this pre-training phase, the



(a) Q Returns on changing discretization parameters



(b) Q Returns on changing discretization parameters

Figure 5.6.: Q Returns of the Belief Observing IQN with varying discretization parameters with thresholding enabled

memory is only populated with demonstration samples. In the next phase however, the buffer is to retain certain demonstration samples while discarding the remaining to make place for agent-generated transition information.

In the interest of robustness, the memory architecture is designed such that the memory can store both demonstrator-only, agent-only, as well as mixed samples. Minimum variables are used to control indexing of these transitions. Since the demonstrator samples in the online training phase are to remain untouched, the counter is positioned such that the agent can only overwrite agent-generated transition information.

5.4.2. Learning from Demonstration Loss J_Q

The loss function of the standard IQN is modified to learn transition information from an Expert. The convergence of the supervised classification loss depends on establishing a margin suitable for transfer of learning between the quantiles by means of using the Q values. Since the Q values represent the expectation of the quantiles, the propagation of the supervised classification loss allows a gradient flow that equals the maximum margin possible between the most rewarding action Q_{s,a_E} and the closest rewarding action by margin. Since the loss captures the best action, which is not necessarily a demonstration action, it ensures that the addition of $l(a_E, a)$ allows for a propagation of a relatively large gradient.

The pseudo-code of the implemented loss function is provided in Algorithm 2. The pseudo-code focuses on the steps to calculate the supervised classification loss alone. The loss is implemented to facilitate learning of the online network parameterized by θ. The classification loss only deals with sampled transitions that are also demonstrations and is 0 for all agent-generated transitions. The weights attributed to the loss of each transition are determined by the importance sampling weights. During a mixed transitions run, since the number of demonstrations is much smaller than the agent-generated transitions, the weight attributed to demonstrated transitions is higher. This facilitates proportional sampling of demonstrations so that the agent does not "forget" the demonstrations data. The network θ is trained to learn actions taken by the demonstrator at those states which means that the network learns weights that predict higher Q values for demonstrator transitions. This Q value is in turn determined as the expectation of quantiles sampled at τ uniformly. Hence, to ensure the Q values learnt correspond to the quantiles located by τ, the classification loss always considers the same set of τ. Thus, quantiles located by τ used to map the underlying value distribution for the Huber loss is also used for the supervised classification loss.

Since the changing distribution attributing to continuous sampling from the environment can cause considerable instability during learning, the target network is updated with the online network weights periodically to stabilize learning. The remaining losses

are applied as usual. The regularization of learning parameters is only applied on the online network parameterized by θ since this is to prevent overfitting to the demonstrations data. The Huber loss used to propagate gradients based on the temporal differences is preserved in the implementation to control learning from transitions.

Algorithm 2: Loss Calculation Algorithm showing the steps involved in obtaining the IQN Huber Loss and the Supervised Classification Loss

input : N, N', b, s, a, r, s', δ, κ, τ, θ
output: J_Q
`// Sample action by policy` π_{Q_θ}
$a = argmax_{a \in A}[\pi_{Q\theta}(s, a)]$
`// Calculate IQN Huber Quantile Loss`

$$J_{IQN,ijk}(Q) = \begin{cases} \frac{1}{2}\delta_{ijk}^2, & if|\delta_{ijk}| <= \kappa \\ \kappa(|\delta_{ijk}| - \frac{1}{2}\kappa), & otherwise \end{cases}$$

$J_{IQN}(Q) = \mathbb{E}_{i \in b, j \in N, k \in N'}[J_{IQN,ijk}(Q)]$
`// Sample quantiles from online network at` τ
$Z_{\tau i}(s, a) = q_{out}(s, a) \qquad i \in [1, N]$
`// Compute Q Values from Quantiles`
$q_{\tau_i}(s, a) = \mathbb{E}[Z_{\tau_i}(s, a)]$
`// Compute Classification Loss`
$J_{E_i}(Q) = argmax_{a \in A}[q_{\tau_i}(s, a) + l(a \neq a_E)] - q_{\tau_i}(s, a_E)$
$J_E(Q) = \sum_{i=1}^{N}[\mathbb{E}[J_{E_i}(Q)]$
return $J_{IQN}(Q) + J_E(Q)$

5.4.3. Setup of Offline and Online Demonstrated Learning

The Learning from demonstrations training process involves training the agent's learning network θ on demonstration experiences alone prior to exploration. This means, initially, the agent does not take steps in an environment but rather performs a batch gradient descent using the cumulative loss $J(Q)$. As shown in Figure 5.7, the training with demonstrations can be split into two steps:

1. Pre-training only on demonstrations.
2. Exploratory training on mixed experiences.

The pre-training phase involves loading the memory entirely with demonstrations data before training the online network. At each step, a mini-batch of b transitions are sampled from the memory. At the start, the weights given by importance sampling are

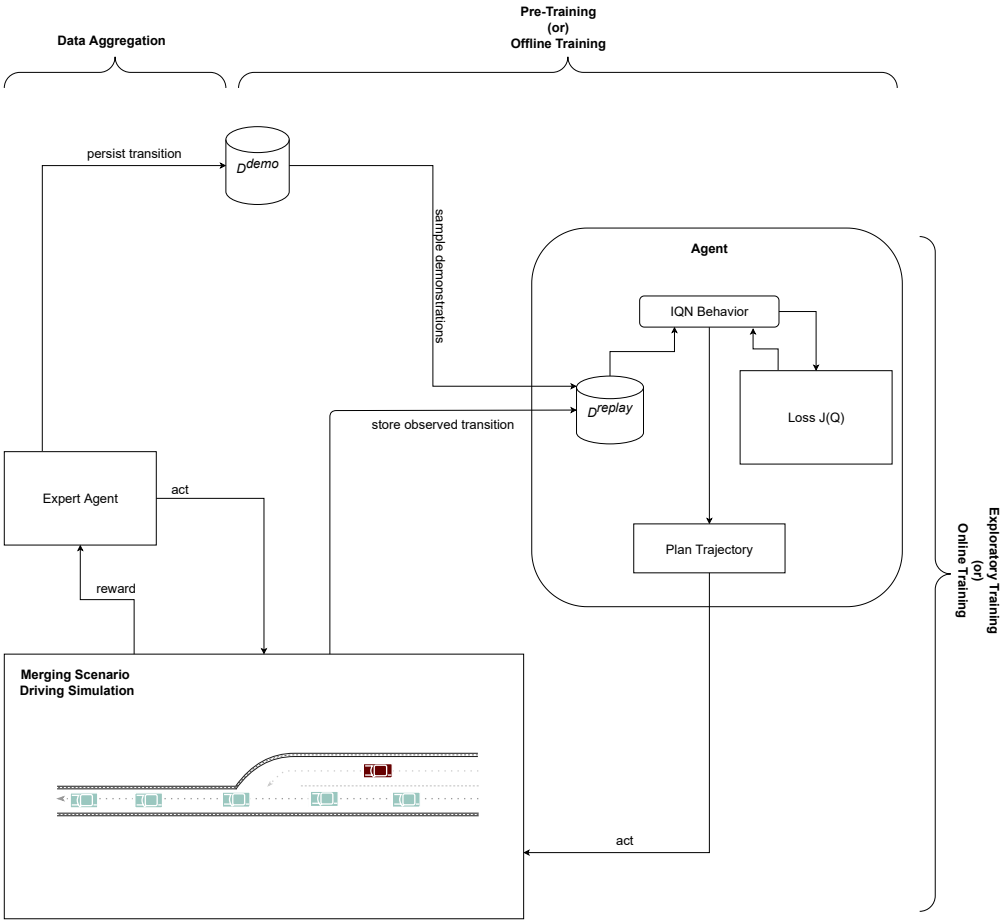


Figure 5.7.: Learning from Demonstrations Architecture detailing the various phases involved. The three main phases are: aggregating demonstrations data, training the agent offline without interaction with the environment and allowing the agent to self-explore the environment.

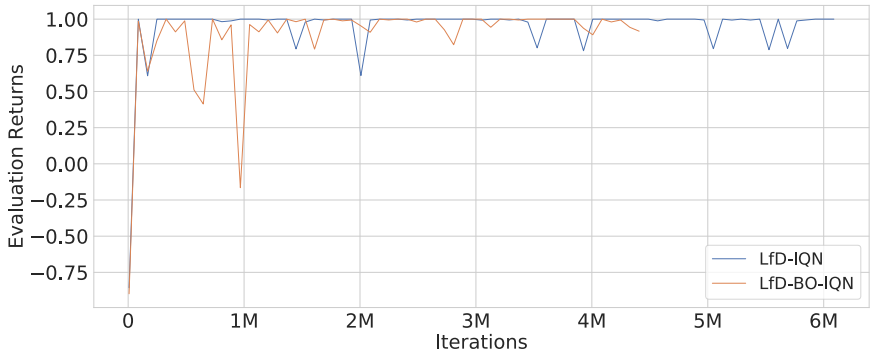
the same and so all demonstrations have the same priority. The sampled demonstrations are then used to calculate the cumulative loss as shown in Equation 4.7. The cumulative loss is used to perform a gradient update step using the online network. In addition, evaluation is included where the agent steps and plans its trajectory at specific intervals during training. This is to provide better insight into the learning during pre-training when the agent has not seen the driving environment.

In the exploratory training phase, the agent memory is first reset to retain a certain number of demonstration samples. This is so there are always demonstrations for the agent to train, as the exploration phase continues and the agent begins generating its own data. The parameter β used to control the amount of importance sampling is also reset. This is because, similarly to the demonstrations case, the agent-generated data towards the beginning of exploration, when the agent has minimal knowledge, is not as important as the agent-generated data once exploration progresses and the agent has sufficient learning. That is, the behavior policy improves with exploration and so experiences sampled later in learning are more important. The behavior policy sampling is ϵ-greedy with respect to the online network. The network samples according to its behavior policy, but learning occurs only during a certain interval. Hence, a distinction needs to be made between pure policy and exploratory learning steps. During a pure policy stage, the agent samples an action and acts accordingly. During a learning step, the agent samples its memory for transitions. These transitions are use to calculate the cumulative loss to propagate as the learning step.

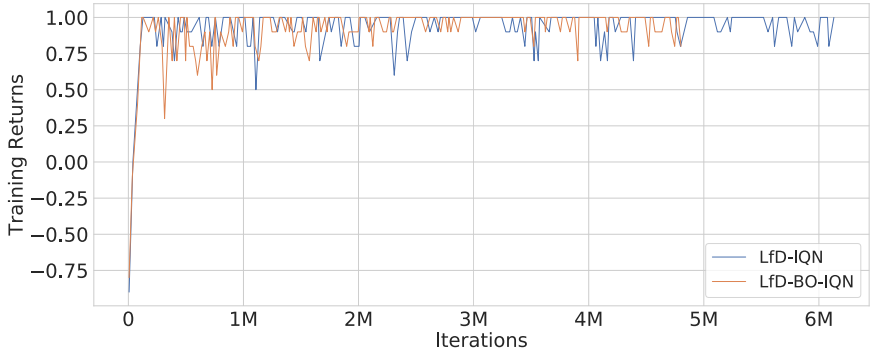
The training and evaluation accuracies are shown in Figure 5.8, where it is noticeable that the LfD-IQN variant is quicker to converge on solving the scenario in comparison to the LfD-BO-IQN variant.

5.4.4. Learning from Demonstrations Convergence Criterion

The sampled quantiles at τ determine the underlying value distribution at step t. There exists a tightly coupled relation between the number of quantiles sampled and the accuracy of the obtained Q values. As shown in the IQN literature[Dab+18], this does not vary much beyond values of $N = 64$ and $N' = 64$, but until those values, it affects long-term learning. Similarly, the accuracy in the LfD Expert Loss is also affected by the number of Q values obtained. The subsequent convergence of the loss continues to depend on the number of τ sampled. In addition, convergence also relies on the batch size. The convergence grows linearly with batch size and number of sampled τ. In a way, this is intuitive in that the number of Q values needed to be sampled to fit a distribution grows with the batch size. Adequate passes of the underlying sampled distribution are also needed. Number of steps linearly affects convergence. Convergence is quick for a small batch size with small samples of τ, but for larger



(a) Evaluation Returns of the LfD variants



(b) Training Returns of the LfD variants

Figure 5.8.: Performance of the LfD-IQN vs. LfD-BO-IQN where one is observing only distances from the ego agent while the other observes beliefs in addition to the distance.

batch sizes, the samples τ need to be larger to sufficiently cover the points in a value distribution in the same number of steps.

Given the complexity of the merging scenario, the state space representative of adequate exploration to reach the goal is also large. A brute-force estimate can be obtained by discretizing the entire space of explorations possible in the scenario by assigning a specific Δ for step intervals at t and $t+1$. Larger Δ implies each step covers more of the physical world, thereby allowing lesser number of next states possible.

6. Evaluation and Key Findings

In this chapter, the performance of the Deep Distributional Reinforcement Learner IQN and its variants is evaluated. The evaluation involves benchmarking to analyse performance using relevant metrics such as collision rate, success rate, out of lane rate and exceeding maximum steps rate. The evaluation also includes an analysis of the learned trajectory of the agent using LfD with IQN variations. The purpose is to ensure the performance of IQN and its variations is evaluated for improvements in exploration, driving behavior as well as generalization.

The generalization analysis can be made by benchmarking performance by slightly varying scenario parameters and configurations. In doing so, it is possible to ensure that there is some overlap between the state space for benchmarking success of driving behavior and the solutions to these scenarios. Evaluating policy improvement can be done simply by ensuring the exact same set of scenarios are seeded and used to evaluate the driving behavior. Further, exploration can be investigated by looking at the transition information generated by an expert in comparison to the LfD variants.

This chapter is arranged as follows: First, the benchmark statistics on the Standalone IQN with varying exploration parameters are evaluated and the greedy policy decision-making is visualized. Second, the impact of varying hypotheses parameters by thresholding, discretization and number of behavior subspaces (splits) is assessed. Third, the impact of learning from demonstrations is evaluated. Following this, the exploration of different agents in LfD is compared to analyze the space of transitions taken to reach the goal. At last, a generalization assessment is made by changing the vehicle distance parameters on all the IQN variations.

The scenarios used to train the ego agent are seeded the same and so each agent sees the same scenario setting during training and evaluation. Every learned variant is benchmarked on 225 unseen merging scenarios. For the generalization analysis, the scenarios are configured from easy or light traffic to difficult or denser traffic by incrementally restricting the distance range between vehicles.

Four key performance indicators are considered for assessing the driving behavior. These are: success rate, collisions, agent position with respect to drivable area and if the agent has exceeded maximum number of steps possible.

6.1. Baseline Performance

The IQN is baselined for the lightly dense scenario. The baseline is given in Table 6.1, showing benchmarked performance metrics. The best performing IQN achieves ∼99.6% success rate in the lightly dense scenario. While the IQN with longest decay $e_{steps} = 90000$ shows the worst performance at 96.89% success. Upon comparing the performance of high end ϵ and slow decay learners, the learner with high end decay appears to be more risky in merging. The learner with slowest decay shows a less risky behavior where it sometimes also exceeds the steps possible and misses merging to the goal. Given the results, the IQN with parameters $\epsilon = 0.01$, $e_{steps} = 25000$ is selected as a baseline for further comparisons.

Figure 6.1 shows a plot of the calculated Q values by the network taken during different phases in a scenario. Here, the greedy action selection policy always selects the action that maximizes the Q value. In figure 6.2 the corresponding Cumulative Distribution Functions (CDF) of the underlying Returns Distributions being learned are shown. Notice that these are the returns distributions that are approximated by the quantiles 'implicitly' using the distorted risk measure. As a result of this approximation, it also appears that the CDF loses its non-decreasing property. This could also potentially indicate that while the underlying returns distribution is not accurately learned by IQN, it can still solve the lightly dense merging scenario.

	Metric			
Exploration Parameters	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
$\epsilon = 0.01$, $e_{steps} = 25000$	**99.56**	0.0	0.44	0.0
$\epsilon = 0.10$, $e_{steps} = 25000$	99.56	0.44	0.0	0.0
$\epsilon = 0.01$, $e_{steps} = 90000$	96.89	2.22	0.0	0.89

Table 6.1.: Exploration Parameters variations with IQN where the number of steps to decay e_{steps} from 1.0 towards end ϵ are configured differently. The e_{steps} highlight the number of steps it takes for ϵ to decay to the end value. The colors are assigned in order to identify them by their relevant training results presented in Figure 5.3.

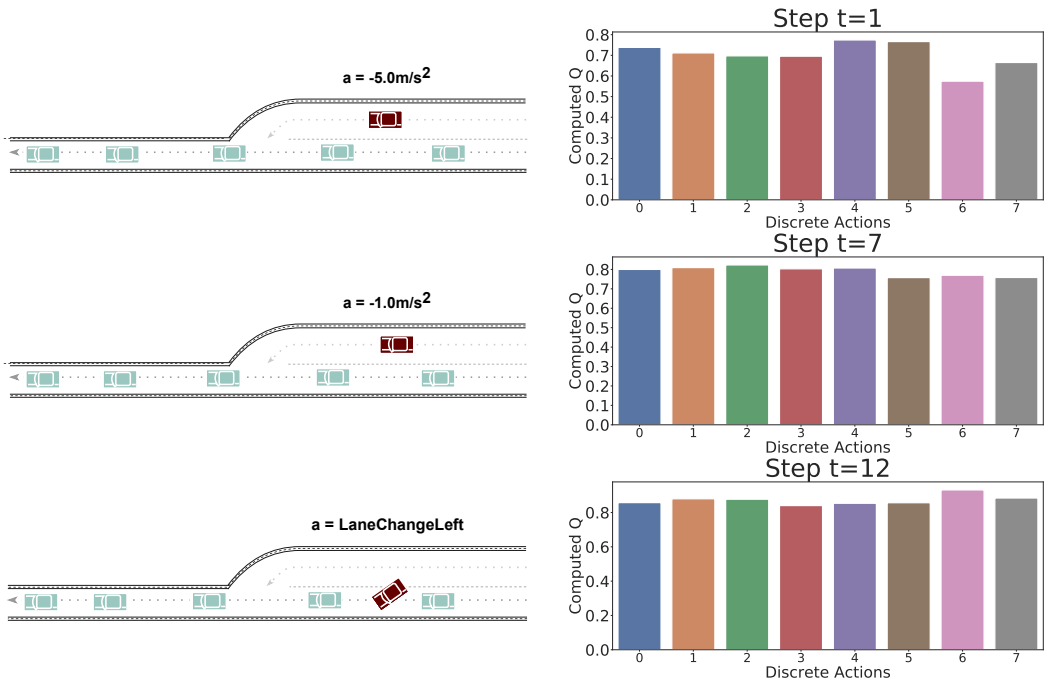


Figure 6.1.: Reconstruction of the merging scenario's steps with a visualization of the Q values given by a Standalone IQN. The action taken is selected by a greedy behavior policy. Here, the policy selects an action that maximizes the Q value at the selected state or time. For example, in the case of the last row, the agent decides that it is time to merge into the adjacent lane, which is shown by the maximum Q value predicted for action A_6 and so the Agent executes action A_6 thereby making a left turn.

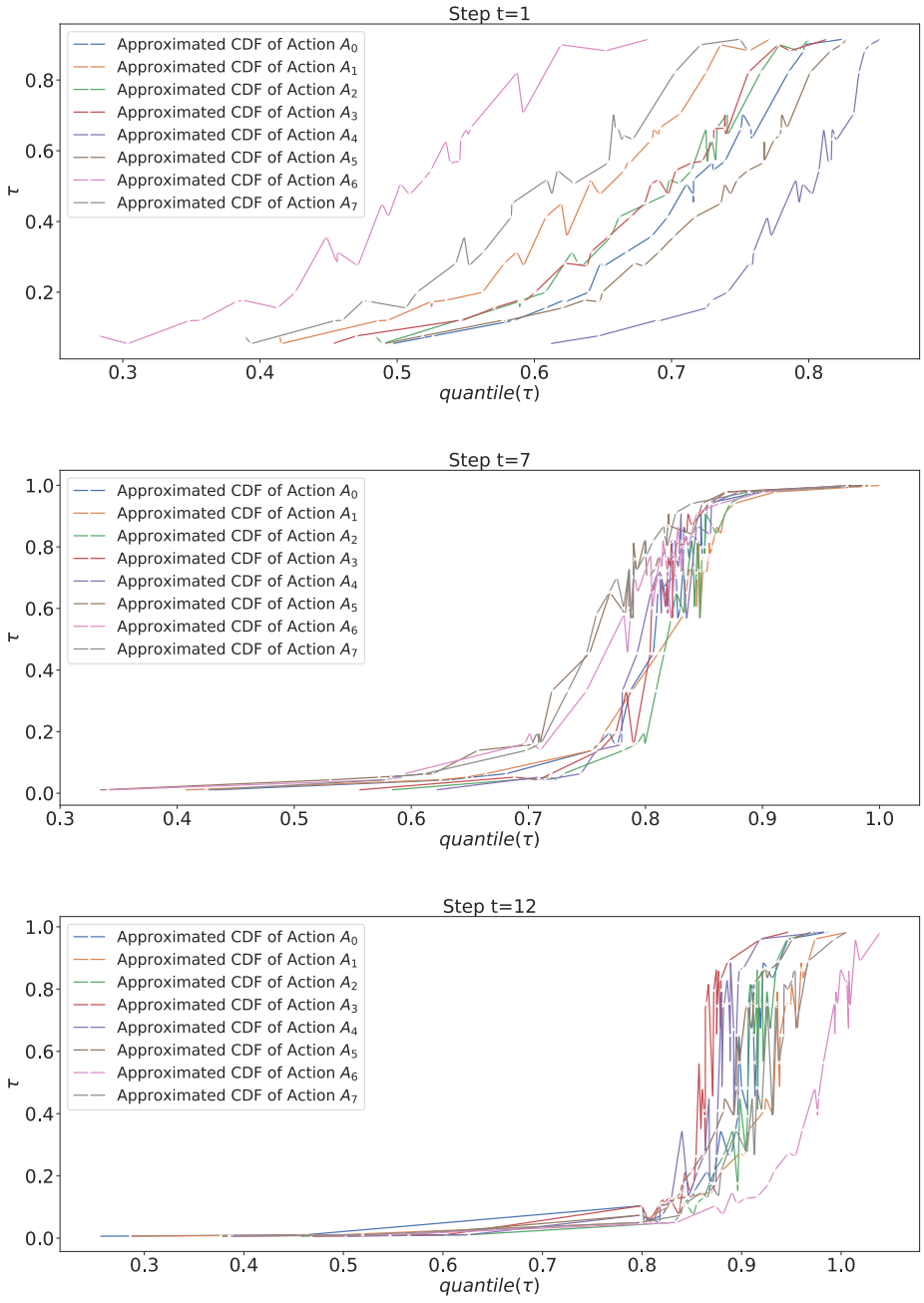


Figure 6.2.: Visualization of the cumulative distribution function of the underlying returns distribution learned for solving the scenario.

6.2. Belief Observing IQN Performance

The Belief Observing IQN(BO-IQN) is evaluated for lightly dense to medium dense scenarios, taken as a baseline and compared with its pre-processing configurations to further assess the effects of thresholding, discretization and dimensionality of hypothesis subspaces. Experiments with changing hypothesis splits from $k = 2$ (Table 6.2) to $k = 4$ (Table 6.3) show that learning worsens as there is a sharp decline from a best of 96% to 76% for solving the same scenario. Considering a regular BO-IQN as a baseline, performance improvements are observed while the belief hypothesis space is discretized and thresholded.

6.2.1. Analysis of Thresholding

Comparing the thresholded BO-IQN and regular BO-IQN in Tables 6.2 and 6.3 show that thresholding can improve performance by a recognizable margin where at $k = 2$, the performance improvement from a regular BO-IQN upon thresholding saw a big jump from $\sim$84% to 96% and at $k = 4$, the improvement is subtle but noticeable from 72.44% to 75.11%. It is interesting to note that the thresholded BO-IQN is more likely to drive out of the lane with a rate of 0.89% in comparison to the regular IQN with a out of lane rate at 0.44% as shown in Table 6.2.

6.2.2. Analysis of Discretization

From Figure 6.3, it is visible that discretization shows improvement over the regular BO-IQN. The number of discretized bins appears to affect the learning in a subtle way, in that learners with 64 and 8 bins outperform the other learners. In Table 6.2 ($k = 2$), pure discretization of beliefs appears to outperform pure thresholding by a $\sim$1% margin. In the same table, pure discretization shows a substantial improvement over regular BO-IQN where a $\sim$17% increase is noticed in the total success rate. Further, in Table 6.3 where a performance comparison with 4 behavior subspaces is made, discretization achieves 76% success whereas the regular BO-IQN achieves 72.44% success. These metrics show that (i) a relatively low discretization could possibly be sufficient to represent the continuous beliefs space and (ii) discretizing can improve performance irrelevant of the number of hypothesis subspaces needed to solve the scenario.

6.2.3. Analysis of Increasing Hypotheses Subspaces

The number of splits of the behavior space also show the possible degree of variations encountered by the ego agent for that particular behavior space. Consider the space used in these experiments which is the headway spacing for the agent to maintain

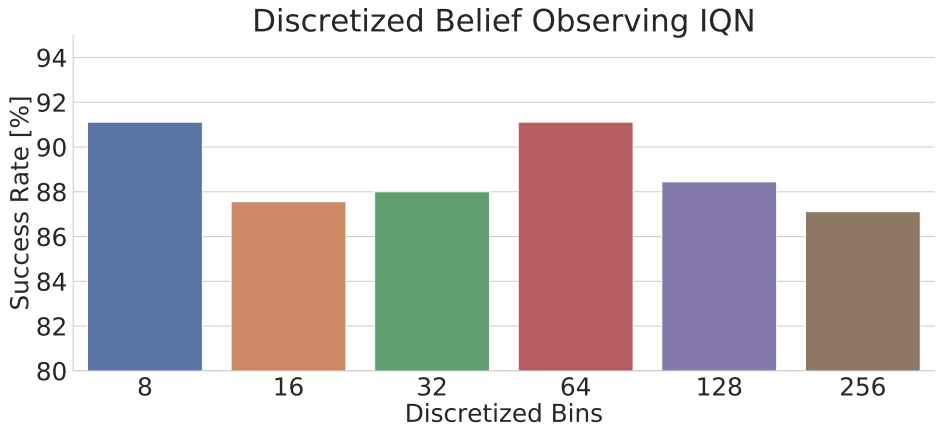


Figure 6.3.: The thresholded Belief Observing IQN learning based on discretization(DISC) - the number of discrete bins used to approximate the continuous hypothesis space. The number of behavior subspaces (hypothesis splits) is kept constant at k=2. The effects of these changes from the metrics suggest that the performance does depend on discretization, with 8 and 64 bins showing better results.

Table 6.2.: A Comparison of pre-processing techniques in Belief Observing IQN configured with 2 subspaces. The number of behavior subspaces (hypothesis splits) is kept constant at k=2. The effects of these changes by means of using success and collision metrics suggest that while both approaches and their combination improve on the baseline, thresholding on its own is most effective in this scenario.

	Metric			
Belief Behaviors, k=2	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
BO-IQN$_{DISC(64)}$	94.22	5.33	0.89	0.44
BO-IQN$_{TH}$	**96.0**	2.67	0.89	0.44
BO-IQN$_{TH,DISC(64)}$	91.11	7.11	0.89	1.33
BO-IQN	84.44	11.56	0.44	3.56

a suitable "gap" in front. If the behavior space is split into only two subspaces, the assumption is that this particular parameter has two possible values in which all agent's behavior policy can be determined. From the Behavior Space literature [BK20], it is shown that 1D input is enough to solve for increasing dimensionality of hypotheses as long as sufficient subspaces are explored. Hence, this work explores the implications of varying parameters along a single dimension, namely the headway in front of the ego vehicle.

In the case of Belief Observing IQN, the network appears to be greatly sensitive to noise and condensing information with a slightly increased state space of observations. Tables 6.2 and 6.3 shows that learning deteriorates as the number of splits of the behavior spaces grow. An average of ∼20% decrease is seen in performance when the number of hypothesis splits to solve the scenario is increased from 2 to 4. It is surprising that the performance drops so sharply between an agent with 4 splits in its behavior space as opposed to having just 2 splits. This could potentially mean that the slightest bit of additional information interferes with the learning of an IQN. The observed decrease in success with increasing splits also show that it is not necessary to use more sub-spaces to solve the problem in the lightly dense traffic scenario.

Table 6.3.: A Comparison of pre-processing techniques in Belief Observing IQN configured with 4 subspaces. The number of behavior subspaces (hypothesis splits) is kept constant at k=4. The effects of these changes by means of using success and collision metrics suggest that while both approaches and their combination improve on the baseline, discretization on its own is most effective in this scenario.

	Metric			
Belief Behaviors, k=4	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
BO-IQN$_{DISC(64)}$	**76.0**	19.11	1.33	3.56
BO-IQN$_{TH}$	75.11	21.78	0.44	3.56
BO-IQN$_{TH,DISC(64)}$	75.56	23.56	0.0	1.78
BO-IQN	72.44	23.56	0.89	4.0

6.3. Learning from Demonstrations IQN Performance

The Learning from Demonstrations IQN Performance is evaluated on the lightly dense to medium dense traffic scenario. The evaluation also considers the changing transition

data available to the learner during the pre-training phase and the exploratory training phase. The benchmarking is performed on the agent learning during both phases since it is necessary to analyse the efficacy of learning even when the agent has not self-observed the environment.

From Table 6.4, reasonably good success rates show that the agent manages to learn reasonably well just by looking at the demonstrations data. The best performing of these is the thresholded LfD-BO-IQN with an impressive accuracy of 93% without any self-exploration. The poorest performer is the regular IQN with a lower tan expected success rate of 64.44%. The supervised learning loss plays a crucial role in propagating information from demonstrations for solving the scenario. The success considered here could possibly inform on how tightly-coupled learning is to expert demonstrations before exploratory training. As shown in [Hes+17], the elimination of supervised loss does not promote a better performance even if the agent is exposed to the same demonstrations. The range of success rates of learned behavior is wide with a total $\sim$30% gap in the resulting performance. Overall, the LfD-BO-IQN shows to benefit the most from the pre-training phase with all pre-trained LfD-BO-IQN showing better performance.

Table 6.4.: Comparative performance metrics of the network after the pre-training phase in LfD-IQN and LfD-BO-IQN. These preliminary results show that LfD-BO-IQN potentially benefits more from the pre-training stage in comparison to LfD-IQN.

	Metric			
Pre-Training Behavior	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
LfD-IQN	64.44	4.44	0.0	31.56
LfD-BO-IQN$_{DISC(64)}$	89.33	7.11	0.89	3.56
LfD-BO-IQN$_{TH}$	**93.33**	4.89	0.0	1.78
LfD-BO-IQN$_{TH,DISC(64)}$	71.11	14.22	0.89	13.78
LfD-BO-IQN	80.0	7.11	0.44	12.44

In the Exploratory learning phase, however, the IQN does not manage to improve learning by a large margin. Comparing the Standalone IQN in Table 6.1 with LfD-IQN in 6.5, it appears that overall performance worsens while learning from demonstrations. This also seems to hold for the Belief Observing variants where the success rate reduces for the thresholded BO-IQN. However, while comparing the performance of a discretized BO-IQN with discretized LfD-BO-IQN, it appears that learning from

demonstrations has helped improve performance. Further, it appears to be the best performing of all the Belief Observing variants with a success rate of 97.33%.

Table 6.5.: Comparative performance metrics of the network in the exploratory training phase in LfD-IQN and BO-LfD-IQN. In the case of only LfD-IQN, it is noticeable that performance is relatively worse from demonstrated learning in comparison to LfD-BO-IQN. For the LfD-BO-IQN, the number of behavior subspaces(hypothesis splits) is kept constant at k=2 during these experiments.

	Metric			
Online Training Behavior	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
LfD-IQN	96.89	1.78	0.89	0.44
LfD-BO-IQN$_{DISC(64)}$	**97.33**	1.78	0.0	0.89
LfD-BO-IQN$_{TH}$	88.44	9.33	0.0	2.22
LfD-BO-IQN$_{TH,DISC(64)}$	89.33	7.11	0.0	4.0
LfD-BO-IQN	92.44	5.33	0.0	2.22

6.4. Analysis of Improvements of Exploration

An analysis of exploration is made by comparing the states learned by the IQN variants. For this analysis, the Expert MCTS states data is used as a reference. Therefore, the agents are selected as MCTS demonstrator with a Nearest Agent Observer, MCTS with a Belief Observer, LfD IQN and LfD-BO-IQN.

In Figures 6.5 and 6.4, the last few explored states are visualized. Here, the Agent which always starts on the upper right traverses to a lane in the lower region. Comparing (a) and (b) in Figure 6.4, the demonstrator appears to learn taking the left turn earlier on in the scenario and exploring little by driving further. Here, the demonstrator also appears to take a softer left turn. Whereas, the self-exploring IQN appears to take sharper left turns while soon reasoning that such a turn could have greater chances of collision and so decides to turn right again. Similarly in the LfD-BO-IQN Figures 6.5 (a) and (b), the visualized steps show that the agent anticipates collision and quickly tries to exit the lane. Further, it is interesting to note that in comparison to the LfD-IQN, the LfD-BO-IQN learns more the shortest paths to merge while the tendency to drift further is shown by LfD-IQN. Comparing the demonstrator states in Figures 6.5 and 6.4,

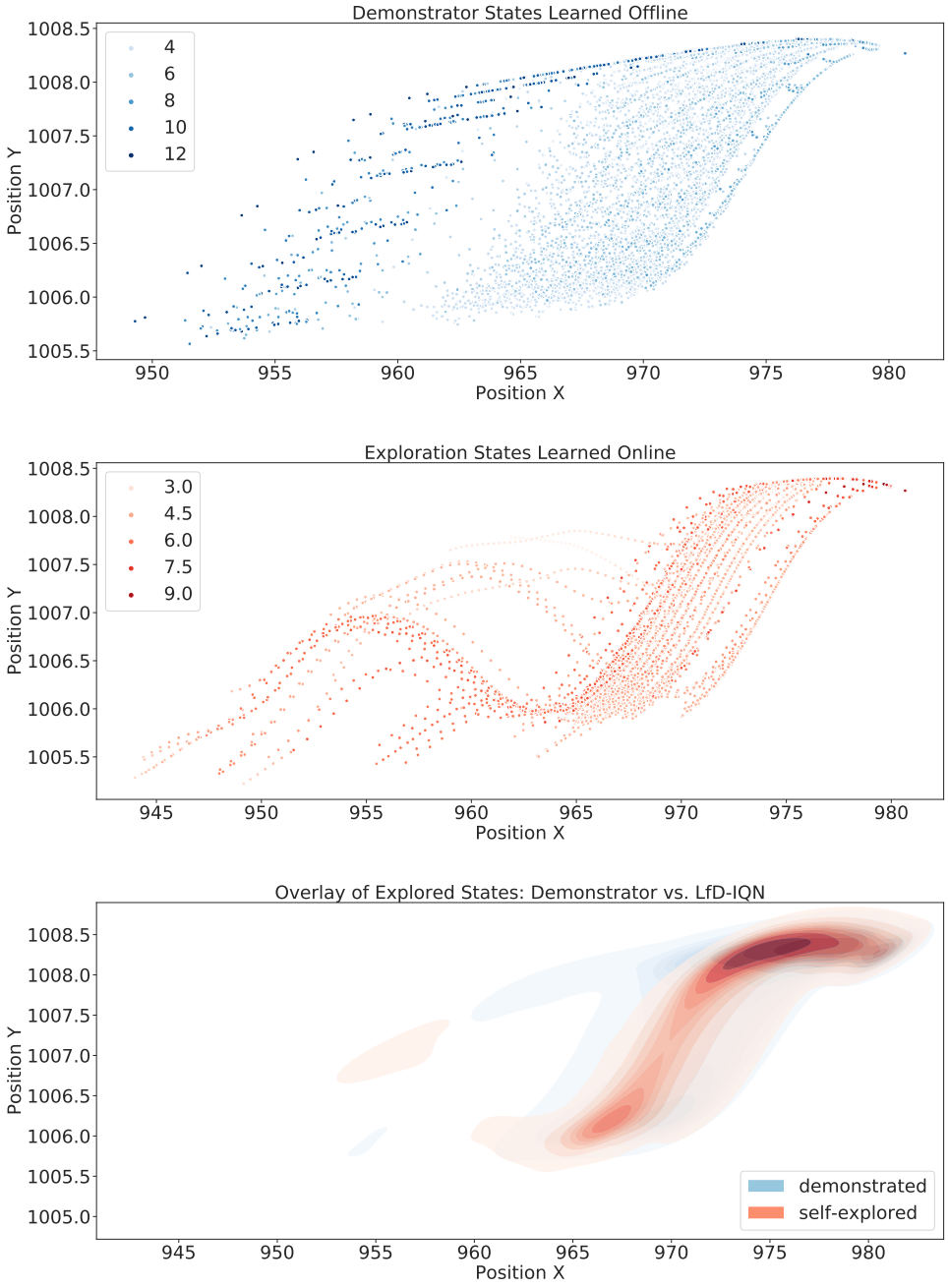


Figure 6.4.: Reconstructed visualization tracing the last 10000 steps learned by the LfD-IQN agent along with velocities applied represented in different colors. The agent starts in every scenario in the top right and has to get to the goal in the neighboring lane by making a left.

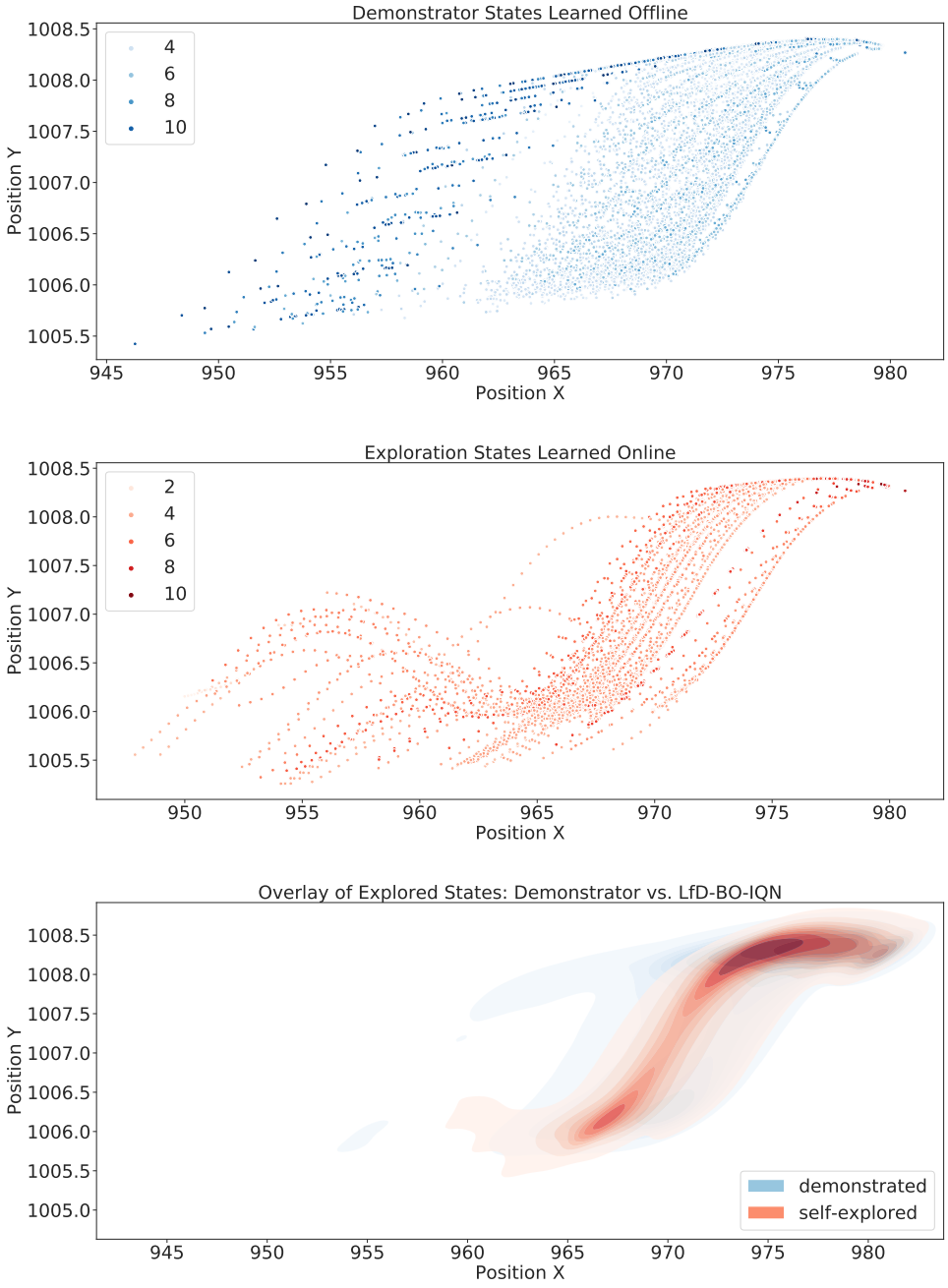


Figure 6.5.: Reconstructed visualization tracing the last 10000 steps learned by the LfD-BO-IQN agent with velocities indicated by different colors. Here too the agent starts at the top right and traverses these states to make it to the neighboring lane with a left turn.

although they observed different data, it does not appear to have influenced the final traced paths. An interesting note are the overlays of the traced states which suggest that although IQN learns on the demonstrated data before self-exploring, in comparison to the Demonstrator, its exploration appears to seek shorter paths to the goal.

6.5. Analysis of Improvements for Generalization

Generalization is important given the epistemic uncertainty intrinsic to reinforcement learning. Since the agent cannot visit all possible states, and since it is possible for large networks to overfit to the training scenario at hand, evaluating capacity of the learned networks to generalize is also necessary. In this section, a generalizability analysis is made by considering the best learners in each variant and evaluating their performances on scenarios with parameters possibly more restrictive than the scenarios used to train the agent. Three variations are considered where the vehicle distance range is gradually narrowed for evaluation.

From Table 6.6, where the traffic is makde slightly denser from configuring the vehicle distance range as [11m, 15m], the best performing is the Standalone IQN learner. However, from Table 6.7, where this gap was only slightly narrowed to [10m, 14m] range, it looks like most of the IQN variants are unable to adapt and perform poorly with the highest success at 6.22% by the discretized BO-IQN. This suggests that exploration and general adaptability of the networks is absent when the network has to try and solve more dense traffic scenarios. However, quite surprisingly, in Table 6.8, where the traffic is now of medium density with a distance range of [6m, 10m], it appears that LfD-IQN shows the best performance amongst the variants with 16% accuracy. Put togeather, these results show that while the IQN and variants learn lightly dense scenarios well, their capacity to generalize this learning appears absent.

Table 6.6.: A comparison of the IQN learner and its variants when benchmarked on a scenario with different parameters where the applied distance range was shortened to lie between 11m to 15m. The Standalone IQN shows to be the best learner for the scenario which closely represents the scenarios it was trained on.

	Metric			
Driving Behavior	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
Standalone IQN	**96.44**	0.44	0.44	2.67
LfD-IQN	88.00	11.11	0.44	0.44
BO-IQN$_{DISC(64)}$	92.00	5.33	0.89	1.78
BO-IQN$_{TH}$	91.11	6.67	0.89	1.33
BO-IQN$_{TH,DISC(64)}$	80.89	10.67	1.33	7.56
BO-IQN	73.33	17.78	3.56	6.67
LfD-BO-IQN$_{DISC(64)}$	90.67	8.89	0.0	0.89
LfD-BO-IQN$_{TH}$	81.33	14.67	0.44	4.00
LfD-BO-IQN$_{TH,DISC(64)}$	85.78	11.56	0.44	2.22
LfD-BO-IQN	87.56	10.22	0.0	2.22

Table 6.7.: A Comparison of the IQN learner and its variants when benchmarked on a scenario with different parameters where the applied distance range was shortened to lie between 10m to 14m. The best result in this scenario is achieved by the discretized Belief Observing IQN.

	Metric			
Driving Behavior	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
Standalone IQN	4.44	97.78	0.0	0.0
LfD-IQN	1.33	15.56	0.89	82.22
BO-IQN$_{DISC(64)}$	**6.22**	71.56	2.22	21.33
BO-IQN$_{TH}$	1.78	62.22	0.44	36.0
BO-IQN$_{TH,DISC(64)}$	4.44	74.67	4.44	18.67
BO-IQN	5.33	87.11	1.78	6.22
LfD-BO-IQN$_{DISC(64)}$	4.0	86.22	4.89	6.67
LfD-BO-IQN$_{TH}$	5.78	96.44	0.0	0.0
LfD-BO-IQN$_{TH,DISC(64)}$	3.56	82.67	1.33	13.78
LfD-BO-IQN	3.56	81.33	9.33	8.0

Table 6.8.: A comparison of the IQN learner and its variants when benchmarked on a scenario with different parameters where the applied distance range was shortened to lie between 6m to 10m. Here the comparison shows that the LfD-IQN achieves the best result.

	Metric			
Driving Behavior	success [%]	collision rate [%]	out of drivable rate [%]	exceeded steps rate [%]
Standalone IQN	8.44	92.00	0.0	1.33
LfD-IQN	**16.44**	44.44	0.89	39.56
BO-IQN$_{DISC(64)}$	6.67	84.44	3.56	7.11
BO-IQN$_{TH}$	15.56	48.89	1.78	36.44
BO-IQN$_{TH,DISC(64)}$	10.22	74.67	4.0	14.67
BO-IQN	10.67	83.11	3.11	3.56
LfD-BO-IQN$_{DISC(64)}$	6.67	93.33	0.0	1.78
LfD-BO-IQN$_{TH}$	10.67	93.33	0.0	0.0
LfD-BO-IQN$_{TH,DISC(64)}$	8.0	81.33	1.78	11.56
LfD-BO-IQN	5.33	89.33	0.44	5.33

7. Discussion and Future Work

In the previous chapter, the performance of a Belief Observing IQN and the Learning from Demonstrations IQN was compared against the established baseline of a Standalone IQN. An extensive comparison was also made amongst the Belief Observing IQN with various configurations of thresholding and discretization applied across behavior subspaces. Training and evaluating performances also considered different scenarios so as to provide extensive data on the generalization.

Recall that the research taken up in this work was to identify ways to improve the existing baseline of a Standard IQN network in terms of benchmarking performance, capacity for generalization and improving exploration. Solutions for such involved looking at improving approaches to solving uncertainty by leveraging additional information or adding more steps during learning. Following this, a hypothesis is assumed that better planning behavior could be achieved in a Deep Reinforcement Learning Agent by considering augmentations in the observed state space as well as improvements in the underlying training paradigm. The former was achieved by considering an extended state space with belief information from the environment and the latter by introducing learning from demonstrations to realign the IQN training.

Preliminary results show that the standalone IQN outperforms these suggested improvements in the scenarios that it was originally trained on. However, when evaluated on more complex scenarios, the Standalone performs significantly worse when compared to the Belief Observing IQN.

This work further considered two other factors that influence planning behavior in the real-world. These are improvements in terms of generalization and improvement with respect to exploration. Generalization is where the network is expected to broaden its scope to scenarios where slight re-configurations with respect to some physical aspect of the environment are made. Improvements in terms of exploration capacity is where the network is not expected to saturate and continue optimizing for a suitable exploration strategy especially when training only requires learning a specific type of scenario. Generalization is important for solving problems in real-world since all states cannot possibly be learned during training and that the network must possess a certain level of robustness to navigate unseen states. To generalize learning in an Agent, in the presence of sufficient additional information, it was suggested that the Agent's network parameters could be trained on additional beliefs such that its capacity to

predict behavior on unseen states improves.

To make an argument for improved generalization, three different kinds of scenarios are considered for evaluating the agents performance. The first being the scenario that the agent trained on where IQN performs the best. However, with difficult scenarios, results show that the standalone IQN's performance does not hold up to the Belief Observing IQN's performance on average. That is, with the addition of beliefs, it is noticeable that the performance of Standalone IQN worsens far more than that of the Beliefs Observing IQN for scenarios not seen during training of the network. This indicates that training using beliefs in addition to the observed physical state allows the network to grasp better generic attributes and prevent overfitting to certain scenario specifications. Similarly, in the learning from demonstrations case, the standalone IQN's performance deteriorates far quicker than the LfD variant of IQN. This suggests that the LfD IQN trained on expert information sees more useful states in comparison while avoiding overfitting given that the states finally learnt come from two learners. However, it must be noted that for the denser scenarios, all variants performed poorly. The success rates dropped significantly and at best only highlight a tendency for better performance with Belief Observing IQN and LfD-IQN.

In particular, thresholding and discretization appear to help improve performance over the regular Belief Observing IQN. This also holds for increasing the number of behavior subspaces from 2 to 4 where there is a 100% jump in the number of beliefs to be learned by the network. In the case of improvement with discretization, if adequate bins are used to approximate the underlying continuous space, and noise in bin is reduced, it might be enough to nudge the gradient to learn noisy scenarios thereby potentially improving generalization. Whereas in the regular BO-IQN, it could be that continuous values are too noisy and cause oscillations during the gradient update. As the scenario gets difficult, however, discretization shows poorer performance to the Belief Observing IQN variants where a shift in performance on part of thresholding can be noticed. While on average, the Belief Observing IQN appears to generalize better than the LfD-BO-IQN, the performance is still poor and need further evaluations with a different method of using the beliefs. Surprisingly, while thresholding and discretization are applied together, although performance is improved over the regular BO-IQN, performance is poorer in comparison to only thresholding and only discretization. This could potentially be affected due to the ordering where the beliefs are thresholded before discretization, thereby causing potential loss of beliefs that might otherwise have been discretized to a higher value.

In terms of improving exploration, the comparison considers the expert used to generate demonstrations and the IQN variations. The expert used in the light dense traffic scenario is a MCTS solver with full information about the universal state. This means, in an ideal scenario, with full information, the MCTS solver's states are what

the optimal solution would look like. The Belief Observing IQN traces more states taking shorter paths to the goal in the same number of episodes when compared to the IQN. This could be due to the beliefs mapping some "confidence" in an agent following a particular behavior, and when this behavior happens to be a range, the ego agent must explore more states to solve such behavior. The beliefs force the agent to consider that an observed agent is acting in a range of physical attributes. This behavior of the observed agent suggests a level of uncertainty which the ego agent must solve to learn which value amongst these ranges most influence the observed agent's behavior policy. In order to determine such behavior, the ego agent relies on beliefs tracked in real-time and takes steps to observe a greater number of unique states. In the case of comparing the demonstrated steps to the IQN variants, it clearly shows that uncertainty in partially observable environments can cause considerable confusion, even when the scenario is relatively simple to solve. Meanwhile Figures 6.5 and 6.4 show that the agent drives into the goal lane and out of it, there shows a trade-off in the behavior policy between reaching the goal whilst avoiding collisions.

The results above show that IQN's performance is best for the scenario it is trained on. Further, it also show that the approaches suggested do no increase benchmarking performance. However, further improvements can be made to better understand why this might have been the case. The current number of scenarios used to benchmark and evaluate are 225. This can be further increased to 500 and 1000 to check for completeness in terms of scenarios learned. This work also considers scenarios with less dense traffic. To qualitatively test the limits of network learning, various other scenarios could be trained on and evaluated. One option would be to use ensemble learning with agents trained in traffic scenarios of different density. In addition, the current implementation of the beliefs involves augmenting input space. This is not prudent if the number of dimensions to solve a very difficult scenario were to increase. Instead, considerations can be made to combine beliefs prior to training network so that the size of the final input space is kept small and ideally can grow linearly. From the network point of view, this work evaluates the performance of one Distributional Reinforcement Learner, IQN. There are further distributional learners that can also be used to gain a better understanding of impact of Distributional Reinforcement Learning for covering Behavior Spaces. In addition, improvements like Policy Distillation[Rus+16] and Adverserial Learning[KNT20] can further be made to explore other avenues that could potentially work better for such a setting.

8. Conclusion

In this work, a Belief Observing IQN and a novel LfD-IQN algorithm are introduced. The Belief Observing IQN shows signs of better generalization and exploration over the Standalone IQN. The LfD-IQN algorithm also appears to perform better in certain situations when evaluated for generalization and exploration of further states. However, when considered for performance on the training scenario, the Standalone IQN performs sufficiently well and using the Belief Observing IQN does not appear to show improvements. That is, the driving behavior is not vastly affected as long as the IQN can learn a good enough strategy. As the scenario gets restrictive, the IQN is affected by its inability to learn further states, thereby causing its performance to worsen when compared to the Belief Observing IQN and LfD-IQN.

The Belief Observing IQN, modeled using Behavior Spaces [BK20] shows performance over the Standalone IQN does not improve if the comparison is made by evaluating the benchmarked performance on scenarios trained. However, as the traffic gets denser and the scenario more complex to solve, the Belief Observing IQN shows better capacity to generalize. The Belief Observing IQN also shows the capacity to learn more states to find shortest distances to reach the goal in comparison to the regular IQN where both learners are trained in the LfD paradigm. Both these factors suggest some interesting results. Although the Belief Observing IQN does not learn as well as the Standalone IQN, the Belief Observing IQN does appear to learn good enough strategies to help in solving scenarios that are not previously seen. The comparison between the Belief Observing IQN variants further show that pre-processing steps such as thresholding and discretization improve performance irrelevant of the Behavior subspaces needed to solve the scenario. In most Belief Observing IQN experiments, it seems discretization can also help increase the success rate. Data in this analysis also further suggests that there appears a limit to the the number of discrete quantities required to improve on a regular Belief Observing IQN.

The LfD-IQN shows relatively good performance on evaluation after pre-training. This is when the Agent has not explored the environment, but can still reason about it simply by learning from demonstrations data. Although this performance does not match that of the Standalone IQN, further training shows that this continues to improve. In addition, the LfD-IQN shows that it can better generalize on difficult scenarios where traffic is considerably denser and more complex to solve. Combined,

this suggests a tendency to retain both the demonstration information as well as self-explored information. The LfD-IQN also covers substantially different states, varying from the initial demonstrated states that it was trained on. This shows that the LfD-IQN does not fit to the demonstration data during exploratory learning and further explores independently along different states without attempting to fit learning to demonstrated states. The Belief Observing variant of the LfD appears to benefit more from demonstrated learning in comparison to LfD-IQN. However, this needs to be further evaluated to better understand the impact of Behavior Spaces in LfD.

These results show that although benchmarking and evaluating performance based on tangible physical impacts such as success when reaching goal, collision when crashing vehicle or curb are necessary, it is also required to consider certain other relevant criterion such as generalization and exploratory capacity. Together, they can provide a broader and thorough analysis of learned driver behavior.

Since the Belief Observer aims to augment the input state space, it need not be restricted to the IQN and can further be analysed with other distributional reinforcement learning techniques. Similarly, the LfD modification for IQN need not be limiting and can be explored with modifications according to other distributional learning techniques. In addition, the training paradigm of LfD-IQN can also be further extended to other driving-based scenarios where the observed states have additional dimensionality like in the case of drones, animistic robots etc. Results from such analysis could potentially further inform on how LfD can be improved for general navigation of environments and shed light on the impact of changing training paradigms.

A. BARK and Hythe Environments

A.1. Hythe Environment

The Hythe/cite environment handles extending this BARK environment with OpenAI's gym so that environments can be used as the base for training various open-source implementations. This is because most open-source standard learning agents are modeled using the Gym environment capabilities to launch and run games or other simulations.

A.1.1. Architecture

The Hythe architecture is designed primarily to leverage Gym environments for Bark Utility. This involves allowing the overload of the environment Step function as well as extending the functionality of scenario setup. The Bark Database Configuration allows generation and control of scenarios online. Once a scenario is generated, the environment is reset so all agents are brought back to the start of the scenario.

A.1.2. Experiment Overview

Each experiment in Hythe configures an agent training run. This run can be of training or evaluating the agent or for the purpose of setup and configuration of a suitable scenario for the Agent by means of a Scenario Generator. The experiment can be run locally or remotely depending on preference. If remote, a gluster capacity is used to leverage parallelization capacities. The Hythe experiment for training and evaluation is deployed remotely for execution.

A.1.3. Setup and Dispatch

Since this work involved using fairly large networks, a gluster capacity is used for training networks using GPU. In order to run code on the Gluster, a singularity image is used to repository level discrepancies as well as execute the experiment.

A.2. BARK Environment

The BARK[Ber+20] environment is used for simulating a driving scenario. The simulated scenario uses agents each configured to act based upon their own behavior policy. The ego agent in the Bark simulation can be configured with a behavior to inform such a policy. Bark also configures its own environment. A Reinforcement Learning environment is one that models the the learn setting. Such a learn setting is defined by few key characteristics:

1. Every agent progresses from one state to another by taking a **Step** in the environment.
2. At every step, a **scenario** is generated by the environment in which the agent must act. The scenario defines a setting for the goal-driven agent.
3. Every sequence of steps until a terminal stage is an **Episode** after which the environment **resets**, where the agents are brought to start and counter begins again.
4. At every step, the agent receives some reward or penalty from the environment calculated by the Evaluator.
5. At every step, the agent observes the transition with information from the Observer about the environment.

A.2.1. Observers and Evaluators

The BARK environment also provides an Observer and an Evaluator that can be used to retrieve information with respect to agent learning. The Observer helps convert the observed physical world to relevant Neural Network Inputs. This is done by acquiring the world state from the environment and applying some operations on the world state. The world state input for each agent is In the context on this work, two kinds of Observers are used: one is the BARK Nearest Agent Observer and the other Hythe Belief Observer. The Nearest Agent Observer is one which configures the observations based on N nearest agents driving by the ego agent. The end state configured by this observer is a range-normalized list of physical attributes. In case of the belief observer, this normalized list is extended to accommodate belief values and so the Belief Observer provides the output of the nearest agent observer concatenated with nearest agent beliefs. Further, the Evaluator informs the agent if the action a resulted in collision rewarding -1, goal reached +1 or 0 otherwise.

A.2.2. Belief Tracker

The Belief Tracker is used to track beliefs of the world state in real time. The belief tracker relies on the observed state of the physical world to do so. The tracker tracks posterior beliefs for different behavior subspaces to generate hypotheses. This is done because the Belief Tracker responsible for online tracking of beliefs can track beliefs of all agents in the world. However, the beliefs of only the agent with close proximity are important and so need to be filtered accordingly. Also, consistence of information needs to be preserved since the beliefs of agents not in the frame of observation are irrelevant. Therefore beliefs need to added to the observed state for only those agents within the range of observation at that time.

A.2.3. Benchmark Runner, Database and Result

The Benchmarking Database provided by bark allows launching scenarios for the purpose of evaluating performance of the Agent. The Benchmarking Runner can be configured based on separate criterion to retrieve performance information. The Benchmarking database in turn acts as a configuration engine for management of scenarios used in evaluation or training.

A.2.4. Demonstration Collection

For the purpose of generating expert demonstrations, a full information MCTS Demonstrator behavior is used. The implementation of such a behavior already exists and is used to collected observed data to act as demonstrations. The Demonstrator Collector uses a Benchmark Runner to evaluate successful scenarios and use these as demonstration information.

A.2.5. Agent

An open-source PyTorch Agent implementation is leveraged. The code-base used is fairly straightforward and comfortable for use since the Agent Behavior is decoupled from the Neural Network implementation. As a result of this decoupling, it is easy to plug-in a new network with different configurations. For the merging scenario experiments, this was done to use a DQN Base network with a Fully Connected Network instead of a Convolutional Neural Network. For the purpose of learning from demonstrations, the loss function had to be reconfigured to accommodate for the supervised large margin classification loss, which involved an additional layer of decoupling as defined in the agent. The IQNAgent used is therefore modified for

1. Accommodating Changes to the Core Architecture of the Learning Network.
2. Calculating the cumulative loss Equation4.8 for Learning from Demonstrations

Step 1 involved establishing that the depth of the network makes a bigger difference. The kind of information the network deals with is 1D state information. In addition, this information can potentially saturate in too deep layers. However, shallow layers would not be enough to for the weights to learn as well as cause information loss during back propagation.

Step 2 deals with extending the loss calculation. Since the Learn from Demonstrations loss depends on the obtained Q values. The calculation of these Q values depend on the random Variable Z as discussed in Chapter 5. The intuition of using this random variable Z represented by quantiles remains the same in QR-DQN, IQN and FQF. However, these methods differ in the techniques adopted to map these quantiles to obtain the underlying value distribution. Therefore, the crux of calculating this loss is abstracted so that in the future, should these loss be extended for evaluation on other related networks, it is easy to plugin the calculation of a specific loss and combine them rather than rework the entire calculation.

A.2.6. Prioritized Experience Replay Memory Architecture

The Prioritized Experience Replay buffer was implemented to account for separation of two types of data. The first is demonstrator generated data and the other agent-generated data. This was done keeping in mind that the code base should be as reusable as possible. The structure of retrieving and storing samples in preserved from the open source implementations. Changes are however made with respect to persistence of transition samples, indexing based on type of transitions, calculation of priorities based on types of transitions and calculation of importance sampling weights.

Two approaches were considered. One, total abstraction where memory is two instances, one to serve purely the demonstrator transitions and the other agent-generated transitions. However, this would compromise on the integrity of importance sampling as in the Learning from Demonstrations literature, the sampling priorities are artificially boosted for demonstration samples against agent generated samples. This means that if a separate instance were to be used to hold on demonstration sampling, the proportionally of demonstration samples would be deterministic and always the ratio of demonstrations configured. Instead, the proportionality should be a stochastic variant where the same memory holds some demonstrations samples prescribed by the demonstrations ratio necessary and the other accommodate agent generated data. In doing so, there is some noise introduced while persisting and sampling the memory for training transitions.

With such considerations, a single memory is used to store both demonstrator and agent samples. The demonstrator samples ratio is given beforehand and so the first d locations of the total memory capacity is always reserved for demonstration information. The remaining is indexed as agent generated information. Since at the time of persistence, the memory stores weights according to importance sampling criterion, which in-turn depends on different priorities for demonstrations and agent data, the priorities of both types of transitions are preserved for when the agent has to sample memory during a learning step. Therefore indexing difficulties only deal with the exploration phase and need to be configured for access accordingly. Figure A.2 shows the component-wise dependencies of the constructed PER Memory.

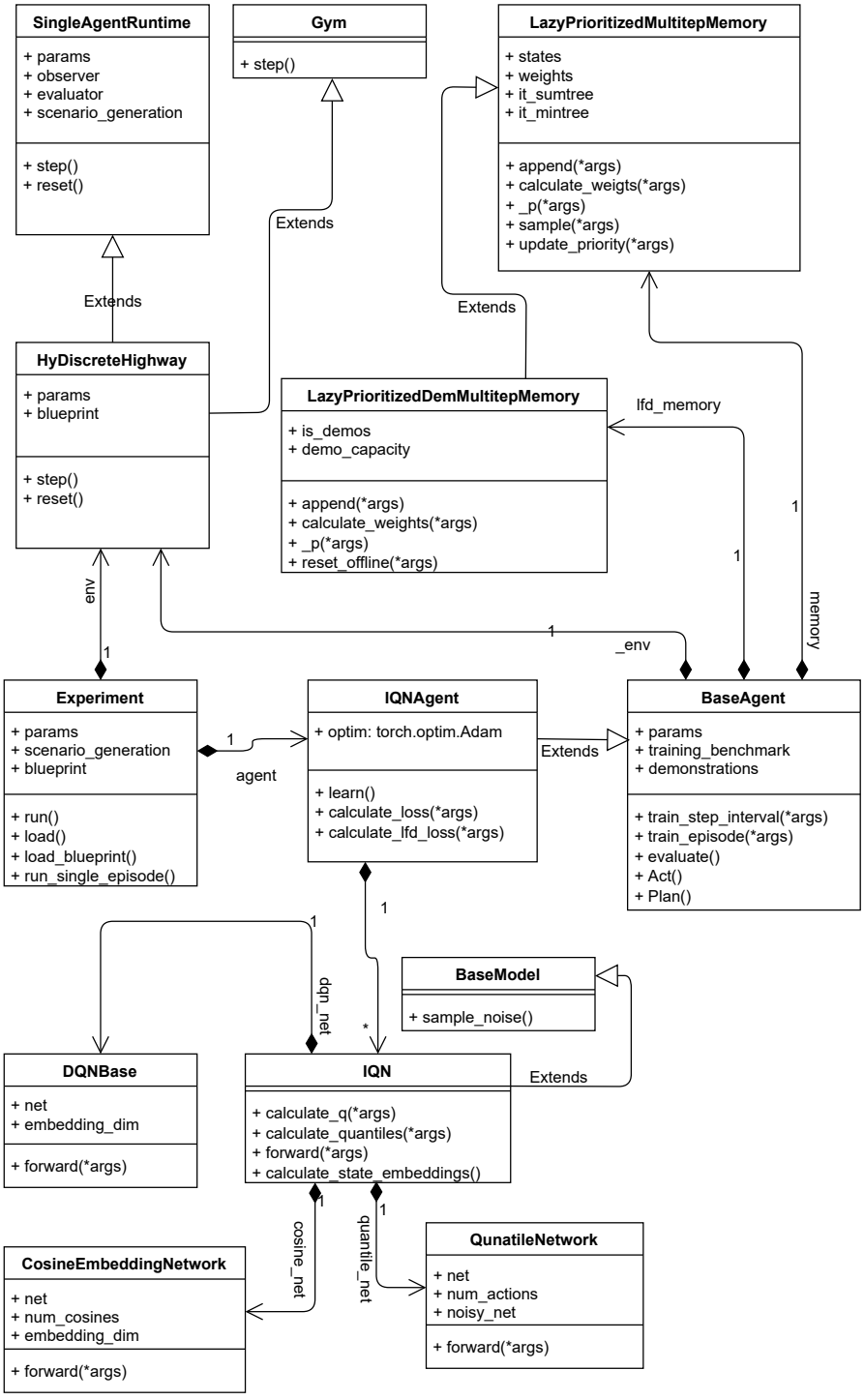


Figure A.1.: Class Diagram of the Hythe and Bark environment showing participating components

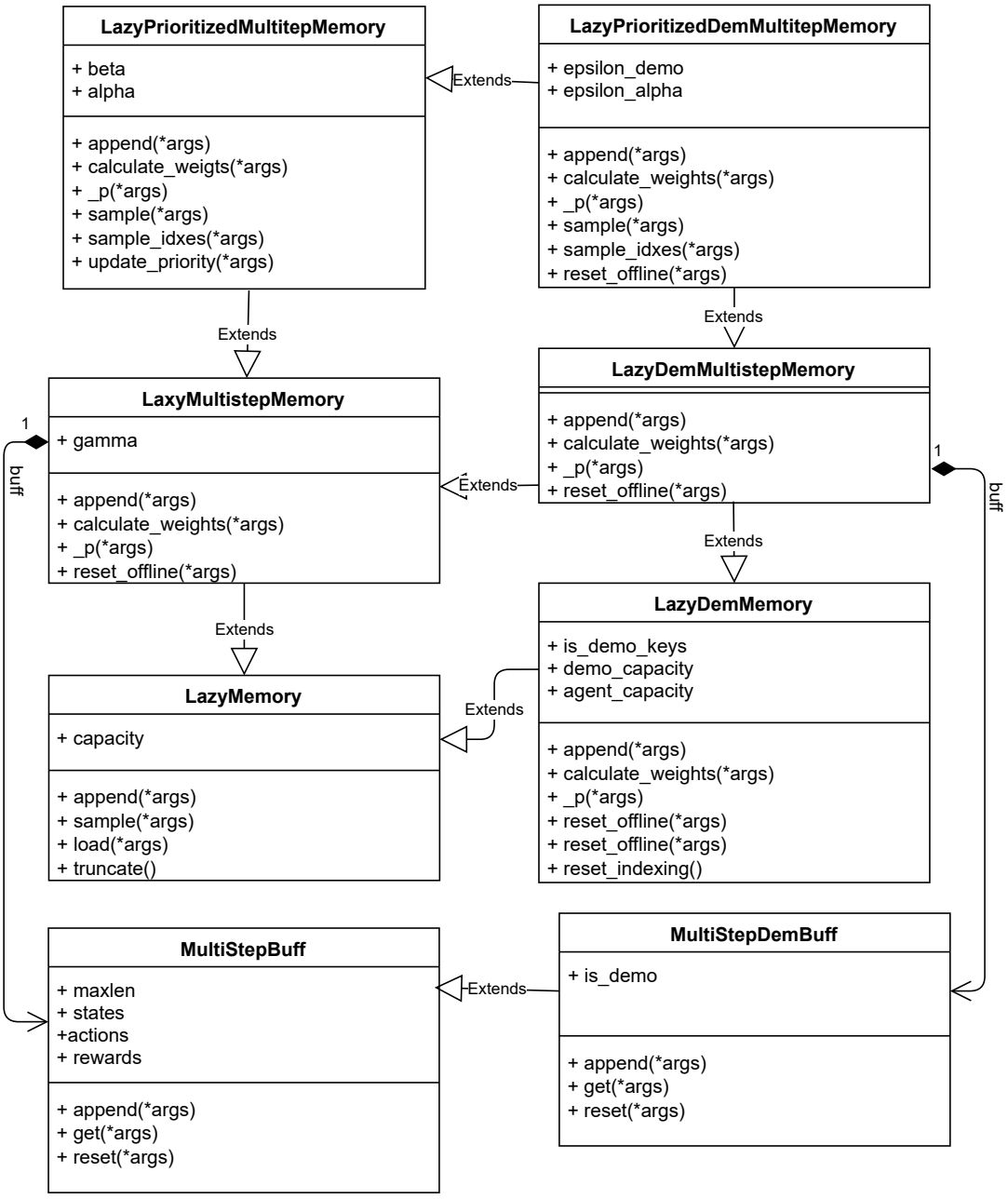


Figure A.2.: Class Diagram visualizing the memory hierarchy with component level dependencies and attributes.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AS17] S. V. Albrecht and P. Stone. "Reasoning about Hypothetical Agent Behaviours and their Parameters." In: *AAMAS*. 2017.

[Bel54] R. Bellman. "The theory of dynamic programming." In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515. DOI: bams/1183519147.

[Ber+20] J. Bernhard, K. Esterle, P. Hart, and T. Kessler. "BARK: Open Behavior Benchmarking in Multi-Agent Environments." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[BG12] A. Billard and D. Grollman. "Imitation Learning in Robots." In: *Encyclopedia of the Sciences of Learning*. Ed. by N. M. Seel. Boston, MA: Springer US, 2012, pp. 1494–1496. ISBN: 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_758.

[BK20] J. Bernhard and A. Knoll. *Robust Stochastic Bayesian Games for Behavior Space Coverage*. 2020. arXiv: 2003.11281 [cs.MA].

[Cap19] C. Caprano. "Bayesian Decision Making for Automated Driving using Distributional Reinforcement Learning." In: (2019).

[Dab+17] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. *Distributional Reinforcement Learning with Quantile Regression*. 2017. arXiv: 1710.10044 [cs.AI].

[Dab+18] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. *Implicit Quantile Networks for Distributional Reinforcement Learning*. 2018. arXiv: 1806.06923 [cs.LG].

[Dos+17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. *CARLA: An Open Urban Driving Simulator*. 2017. arXiv: 1711.03938 [cs.LG].

[Hay+20] A. Hayashi, D. Ruiken, T. Hasegawa, and C. Goerick. "Reasoning about uncertain parameters and agent behaviors through encoded experiences and belief planning." In: *Artificial Intelligence* 280 (2020), p. 103228. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2019.103228.

[HCS13] Q. J. M. Huys, A. Cruickshank, and P. Seriès. "Reward-Based Learning, Model-Based and Model-Free." In: *Encyclopedia of Computational Neuroscience*. Ed. by D. Jaeger and R. Jung. New York, NY: Springer New York, 2013, pp. 1–10. ISBN: 978-1-4614-7320-6. DOI: 10.1007/978-1-4614-7320-6_674-1.

[He+15] K. He, X. Zhang, S. Ren, and J. Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[Hes+17] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys. *Deep Q-learning from Demonstrations*. 2017. arXiv: 1704.03732 [cs.AI].

[Hub64] P. J. Huber. "Robust Estimation of a Location Parameter." In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. DOI: 10.1214/aoms/1177703732.

[Jia+21] H. Jiang, D. Shi, C. Xue, Y. Wang, G. Wang, and Y. Zhang. "Multi-agent deep reinforcement learning with type-based hierarchical group communication." English. In: *Applied Intelligence* (2021).

[Kaz19] S. Kazenas. *The Hadamard product and recursively defined sequences*. 2019. arXiv: 1911.01175 [math.CO].

[KNT20] I. Kostrikov, O. Nachum, and J. Tompson. "Imitation Learning via Off-Policy Distribution Matching." In: *ArXiv* abs/1912.05032 (2020).

[Kum13] E. Kumar. *Hythe*. https://github.com/eeshakumar/hythe. 2013.

[Mav+19a] B. Mavrin, S. Zhang, H. Yao, and L. Kong. "Exploration in the Face of Parametric and Intrinsic Uncertainties." In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '19. Montreal QC, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2117–2119. ISBN: 9781450363099.

[Mav+19b] B. Mavrin, S. Zhang, H. Yao, L. Kong, K. Wu, and Y. Yu. *Distributional Reinforcement Learning for Efficient Exploration*. 2019. arXiv: 1905.06125 [cs.LG].

[MKH19] K. Min, H. Kim, and K. Huh. "Deep Distributional Reinforcement Learning Based High Level Driving Policy Determination." In: *IEEE Transactions on Intelligent Vehicles* PP (May 2019), pp. 1–1. DOI: 10.1109/TIV.2019.2919467.

[Mni+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[MTA18] K. Metcalf, B.-J. Theobald, and N. Apostoloff. *Learning Sharing Behaviors with Arbitrary Numbers of Agents*. 2018. arXiv: 1812.04145 [cs.LG].

[PGP14] B. Piot, M. Geist, and O. Pietquin. "Boosted Bellman Residual Minimization Handling Expert Demonstrations." In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by T. Calders, F. Esposito, E. Hüllermeier, and R. Meo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 549–564. ISBN: 978-3-662-44851-9.

[PM20] E. Pesce and G. Montana. "Improving coordination in small-scale multi-agent deep reinforcement learning through memory-driven communication." In: *Machine Learning* 109.9-10 (Jan. 2020), pp. 1727–1747. ISSN: 1573-0565. DOI: 10.1007/s10994-019-05864-5.

[RAS19] M. Ravula, S. Alkoby, and P. Stone. "Ad Hoc Teamwork With Behavior Switching Agents." In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 550–556. DOI: 10.24963/ijcai.2019/78.

[RGB11] S. Ross, G. J. Gordon, and J. A. Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. 2011. arXiv: 1011.0686 [cs.LG].

[Rus+16] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. *Policy Distillation*. 2016. arXiv: 1511.06295 [cs.LG].

[Sch+16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG].

[Sha+20] E. Shafipour Yourdshahi, M. Do Carmo Alves, L. Soriano Marcolino, and P. Angelov. "On-line Estimators for Ad-hoc Task Allocation." In: (2020).

[Sil+16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529 (2016), pp. 484–489.

[SPB19] H. Shiri, J. Park, and M. Bennis. *Massive Autonomous UAV Path Planning: A Neural Network Based Mean-Field Game Theoretic Approach*. 2019. arXiv: 1905.04152 [cs.SY].

[Sun+17] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell. "Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction." In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 3309–3318.

[SV10] D. Silver and J. Veness. "Monte-Carlo Planning in Large POMDPs." In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc., 2010.

[Wat13] T. Watanabe. *FQF, IQN and QR-DQN in PyTorch*. https://github.com/ku2482/fqf-iqn-qrdqn.pytorch. 2013.

[ZL13] C. Zhang and V. Lesser. "Coordinating Multi-Agent Reinforcement Learning with Limited Communication." In: *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS '13. St. Paul, MN, USA: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1101–1108. ISBN: 9781450319935.