Active Preference Learning Using Trajectory Segmentation

Monica Omprakash Patel

Master of Science

School of Computer Science
McGill University
Montreal, Quebec, Canada

July 2019

A thesis submitted to McGill University in partial
fulfillment of the requirements of the degree of
Master of Science

Abstract

Machine learning is currently used heavily to develop robot behaviors, giving great flexibility and power, but there remains significant burden on human designers to specify reward functions or label data for the learning target. Learning from demonstration is a very intuitive way to teach your robot a new skill. This thesis considers two improvements to modern learning from demonstrations.

First, we consider a model-based imitation approach that utilizes a modern form of deep probabilistic model to predict agent behaviors in order to match them to demonstrations. We replace the non-parametric estimator utilized by an existing approach, which has the limitation of poor scaling with the amount of training data. In its place, a learned parametric model is trained to capture inherent uncertainties. Through sampling-based prediction, our approach is able to capture a distribution over likely outcomes of the given policy. This is paired with a probabilistic notion of the difference between the agent's outcome distribution and the distribution of demonstrations, to produce a gradient-based policy improvement approach. Our results show that this method is effective in imitating demonstrations in a range of scenarios.

The second portion of this thesis considers the *temporal credit assignment* problem within learning from demonstration. We propose an active learning framework that uses trajectory segmentation to addresses this issue. Our method uses spatiotemporal criteria to segment the trajectory. These criteria can be based upon speed, heading, or curve of the trajectory which are intuitive properties to understand user's intentions. Thus, not only does our framework make the user query interface more intuitive but the resulting approach also

learns faster. We demonstrate and evaluate our approach by learning a reward function for various driving scenarios and show that our algorithm converges faster.

Abrégé

L'apprentissage automatique est actuellement très utilisé pour développer les comportements des robots, ce qui leur confère une grande flexibilité et puissance, mais il incombe toujours aux concepteurs humains de spécifier des fonctions de récompense ou des données d'étique-tage pour la cible d'apprentissage. Apprendre à partir d'une démonstration est un moyen très intuitif d'enseigner de nouvelles compétences à votre robot. Cette thèse considère deux améliorations de l'apprentissage moderne à partir de démonstrations. Nous examinons d'abord une approche par imitation basée sur un modèle qui utilise une forme moderne de modèle deepprobabilistic pour prédire les comportements des agents afin de les faire correspondre à des démonstrations. Nous remplaçons l'estimateur non paramétrique utilisé par approche existante, qui a pour limite la faible mise à l'échelle avec la quantité de données de formation. À la place, un modèle paramétrique appris est formé pour saisir les incertitudes inhérentes. Grâce à la prédiction basée sur l'échantillonnage, notre approche est capable de capturer une distribution sur les résultats probables de la politique donnée. C'est assorti d'une notion probabiliste de différence entre la distribution des résultats de l'agent et la distribution des démonstrations, afin de produire une approche basée sur l'amélioration de la politique basée sur les gradients. Nos résultats montrent que cette méthode est efficace pour imiter des démonstrations dans différents scénarios. La deuxième partie de cette thèse examine le problème de l'affectation temporaire de crédits avec l'apprent-issage de la démonstration. Nous proposons un cadre d'apprentissage actif qui utilise la segmentation par trajectoire pour résoudre ce problème. Notre méthode utilise des critères spatio-temporels pour segmenter la trajectoire. Ces critères peuvent être basés sur la vitesse, le cap ou la courbe de la trajectoire, propriétés intuitives permettant de compren-

dre les intentions de l'utilisateur. Ainsi, non seulement notre framework rend l'interface de requête utilisateur plus intuitive, mais l'approche résultante apprend également plus rapidement. Nous démontrons et évaluons notre approche en apprenant une fonction de récompense pour différents scénarios de conduite et montrons que notre algorithme converge plus rapidement.

Contributions

The research described in this thesis was carried out by the author, Monica Patel, supervised by Prof. David Meger, and in collaboration with several members of McGill's Mobile Robotics Lab. Monica independently implemented all of the methods described here and carried out the experiments and analysis. Code for previous method in Active learning was completely implemented by her. The thesis is written entirely by Monica, with reviews and corrections from Prof. Meger. In some cases, the implementations build upon existing libraries developed for previous research, which were used through standard interfaces to save time during implementation. These libraries include: Bayesian Neural Network and PILCO policy improvement code base written by Juan Camilo Gamboa Higuera.

This work has appeared through technical reports submitted to Huawei Research Canada, through a collaborative research project between McGill and Huawei. Those reports represent early versions of this public research and were written by Monica, with reviews and corrections from Prof. Meger.

Acknowledgements

Foremost, I would like to acknowledge and express my sincere gratitude to my thesis advisor, Prof. David Meger for his help, support, expert advice and amiable patience throughout all stages of the work. I would also like to thank him for his constant encouragement that not only made this work possible but also introduced me to the field of Medical Technology through graduate certificate in Surgical Innovation.

I would also like to thank my colleague and good friend Juan Camilo Gamboa Higuera for his advise, invaluable discussions and extremely helpful code base. Thank you for always taking time to answer my questions. Beside Juan, I also thank my friend Jana Pavlasek for her shared efforts in building and maintaining the simulation environment used in this work.

A very special gratitude goes to Prof. Joelle Pineau at McGill University and Huawei for generous funding and fruitful discussions on research in Autonomous Driving.

A special thanks goes to Prof. Gregory Dudek for extremely enthusiastic and friendly research environment. The skills I acquired during the field experiments made me a better roboticist.

Finally, my very special thanks to my grandmother, parents and friends in Montreal and back home for their constant love, care, encouragement and support.

Thank you very much all!

Contents

List of Figures

Part I

Introduction

1

Introduction

Finding an optimal sequence of actions to perform a particular task lies at the heart of skill acquisition. Research shows that imitation plays an essential role in the development of these skills in humans and animals [BR63]. Discoveries in developmental psychology have altered theories about the place of imitation learning in human nature. It was first believed that humans gradually learn to imitate over the years, but now it is known that newborns can imitate body movements at birth [MP02] thus revealing an innate link between observed and executed acts. Neuroscientists and experimental psychologists have discovered mechanisms connecting the observation and execution of actions [FR14], [FR15].

Just as skill acquisition involves connecting observations to actions in human beings and animals, for an artificial agent, the problem of learning involves mapping of the world's state to an appropriate action. This mapping is often referred to as a *policy*. Programming such policies by hand using domain knowledge alone is extremely challenging. It requires a large amount of effort for every different task and would not take into consideration changes in the environment. Moreover, many robotic applications involve assisting a human to perform a particular task; thus we want our agent to be able to learn from a human who might not necessarily have knowledge of robotics.

Learning from demonstrations is an approach to solve such skill acquisition problems in an elegant way. In this approach, a human expert's demonstrations are collected for a particular task, and the algorithm helps the agent to find an optimal policy using these demonstrations. There are two major design parameters which influence different approaches to

learning from demonstration: 1) the demonstration approach; and 2) the policy derivation approach. A demonstration approach specifies how and what kind of demonstrations are collected from the expert. For example, many methods apply *batch learning* to the problem where complete data should be provided before the agent can start learning. A classic example of this approach is *behavior cloning* [BS95]. On the other hand, many methods apply interactive learning where data is provided by the expert incrementally, and the policy is updated incrementally with batches of these data point, e.g., DAgger [RBG11]. The policy derivation approach specifies a method by which an agent derives its policy from the demonstration. There are two core approaches to policy derivation: 1) directly using demonstration data to define a approximate *mapping function* between state of the world and an action, e.g., *behavior cloning* [BS95]; and 2) using demonstration data to learn the reward function and optionally system model and learning an optimal policy using this reward function [SBTC16].

Another aspect in which learning from demonstration approaches differ is the type of data provided to the agent. In many applications, it is possible to provide data in the state and action space of the robot. This reduces an extra step of *understanding the data* for an agent. However, in other domains, the nature of the robot or environment may prevent a human from giving direct demonstrations easily. For example, humans are not skilled at controlling each joint of a high dimensional manipulator or all of the individuals that make up a swarm of robots. In such cases, the expert can demonstrate the task in different state space, and the agent learns via observation, e.g., learning by watching a video [RBG18]. This usually involves processing large an amount of complex visual information. Therefore, many approaches do not rely on an expert's demonstration of a task but rather on the expert's *opinion* on the agent's behavior of the task. This *opinion* can be a *ranking* of the behaviors which the agent demonstrate or *preference* over a set of behaviors.

One difficulty in learning from demonstration approaches is dealing with long demonstration trajectories with complex state space. Taking an example of a self-driving problem, if an artificial agent wants to learn an overtaking maneuver, the trajectory in this scenario will be long and involve complex continuous state-action space of the autonomous vehicle. Thus the learning algorithm should be scalable to handle this large amount of data. In this thesis, we improve upon an existing, state of the art imitation learning method [EPPD13]

by proposing the use of a Bayesian Neural Network (BNN) (see section 4.2) to make the method scale better for different scenarios.

Another issue with imitation learning is that it assumes the demonstration given by an expert to be near-optimal, while for applications like training an autonomous vehicle, demonstrations collected from humans may not be very reliable because it has been well documented in popular culture that human beings are poor drivers [Van08], [Dav15]. A user study performed by Basu *et al.* [BYH+17] shows that people often do not want their car to drive like *they* drive but how they *think* they drive. In this context, it is desirable to avoid the demonstrator from having to generate detailed motions.

Thus Sadigh *et al.* [SDSS17] proposed using an active approach to reward learning where the agent car provides the expert with two behaviors, and the expert provides the agent with a preference. The existing work, known as active preference learning, has illustrated the ability of this method for understanding user preferences in an intuitive fashion. However, since the feedback is received over the complete trajectory, the difficulty with the existing approach is that the algorithm needs to determine which states or actions were responsible for the encountered preferences. This is known as the *temporal credit assignment* problem (see section 2.8). In this thesis, we propose to use trajectory segmentation with active preference learning to understand what properties of the trajectories were of most significance for the demonstrator.

As an illustrative example, consider the two scenarios given in figure 1.1. The expert may choose Trajectory A for segment 1 (shown in cyan) since the car shifts into the other lane with greater distance from Non-Player-Character (NPC). On the contrary, for segment 2 (shown in yellow), trajectory B is better than A, since the agent car stays in middle of the road. For segment 3 (shown in magenta) Trajectory A is better than B for the same reason. By segmenting the trajectory and asking for preferences on each portion, we get information about what part of the trajectory was responsible for the user to choose one over the other. In the previous work, the user would have faced a difficult choice in how to accumulate their preferences, and the algorithm would have faced the difficult task if disentangling this feedback.

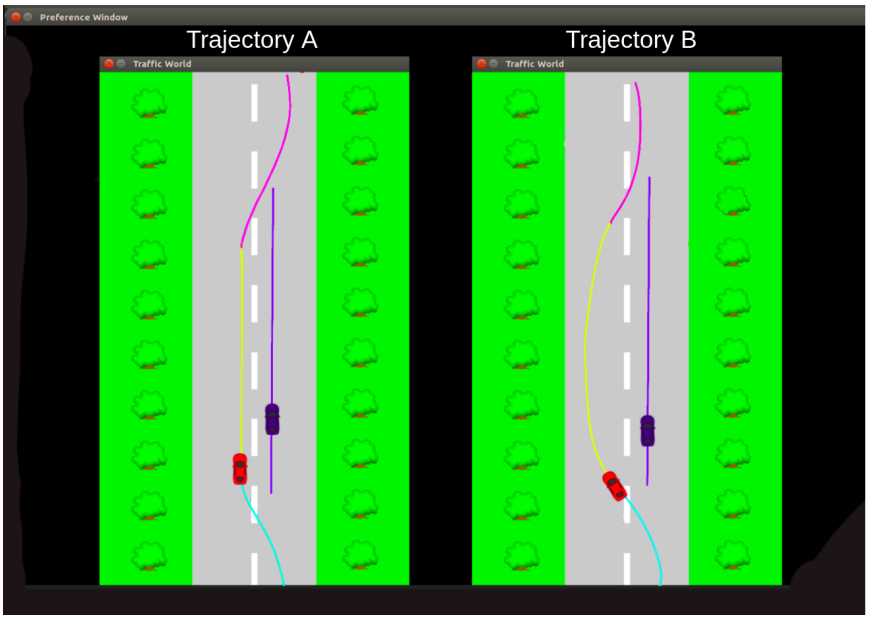


Figure 1.1: Preference interface showing two different overtaking trajectories, with segmentation

1.1 Outline

The thesis is structured as follows. In chapter 2 we provide background on the concepts used in this thesis. This chapter also contains a literature review related to the problems we address. In chapter 3 we describe our experimentation task and environment. It provides a detailed overview of the scenarios over which we test our methods and compare our results. It also describes how human experts can interact with the learning agent and the types of experts used in experiments. Chapter 4 details a method proposed by Englert *et al.* [EPPD13] in previous work for imitation learning and our adaptation of the method to better suit a larger data set. Chapter 5 describes the previous method proposed by Sadigh *et al.* [SDSS17] for Active preference learning of a reward function and our adaptation of this method to incorporate trajectory segmentation. It lastly describes methods we use to get meaningful segments of the trajectory. Chapter 6 shows all the results obtained from active preference learning over various scenarios, their comparison with previous methods and understanding of why segmentation is necessary. Lastly, chapter 7 details the conclusions of our work and discusses future work that can result from this contribution.

2

Related Work

The idea of *Operant conditioning* from psychology was adopted in *Artificial Intelligence* and *engineering (control theory)* to form techniques to program artificial agents to make decisions based on experience. A framework consisting of a collection of these methods is called *Reinforcement Learning* (RL). In RL agent and environment interact continually, where agent selects actions and the environment responds to these actions and present new situations to the agent. Markov Decision Process (MDP) [Bel57], described in section 2.1, are used for a straightforward framing of the problem of learning from interaction to achieve a goal. Many methods in this RL framework are based on this mathematical framework. An MDP is used to describe a *fully observable* environment for RL. Similar to RL algorithms, algorithms in *learning from demonstration* also involves a process where the agent interacts with the environment and chooses an action. Therefore, an MDP framework can also be used to formulate these problems. As mention in the previous chapter, there are many design parameters involved in devising a *learning from demonstration method*. Throughout the thesis, we use the MDP framework to formulate all of our design parameters. In this chapter, we first explain this framework. Then, we give details on a selection of background techniques that are the most required to comprehend the contributions within this thesis.

In section 2.1 we describe MDP framework. In section 2.2 we give overview of a few of the learning algorithms in reinforcement learning. In section 2.3 we give a definition of *Inverse Reinforcement learning* based on the MDP framework and give an overview of the

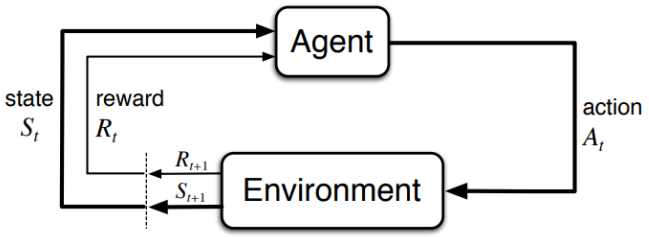


Figure 2.1: The agent - environment interaction in a MDP

algorithms in Imitation learning. One of our main contributions involves model-based RL, and therefore in section 2.4 we briefly explain Gaussian Processes and how they can be used for learning a dynamics model of the environment. This section is based on the definition of the Gaussian process in [RW05]. In section 2.5 we explain how a Bayesian neural network can be learned to represent input and output uncertainty just like the Gaussian Process. This section is based on the work of Gal *et al.* [GG16]. The final contribution of this thesis relates to active Preference learning, and therefore section 2.8 gives an overview of the algorithms in Preference learning and explains the choice of our design parameters in the context of these methods.

2.1 Markov Decision Process

An MDP is defined by tuple $(S, A, R, \delta, \gamma)$, where, S is set of states, A is set of actions, R is a reward function that maps states in set S to a real value, $R : S \rightarrow \mathbb{R}$, δ is transition probability function, $\delta(s'|s, a)$, that gives distribution over next states given current state and action, and γ is the discount factor, $\gamma \in [0, 1)$. A policy for an MDP is function that maps states to action, $\pi(s) = a$. A trajectory τ induced by a policy π in an MDP is a sequence of states and actions upto a finite time horizon H. The goal of an RL agent is to find an optimal policy π^* that maximizes the total accumulated reward of the trajectory for a particular task. Interaction of agent in an MDP in a RL setting is given in figure 2.1.

2.2 Reinforcement Learning Algorithms

In an RL setting, the goal of an agent is formalized in terms of reward signals received from the environment. The agent's goal is to maximize the total amount of reward it receives. This is usually done by finding an optimal policy that maximizes the total expected reward of a trajectory τ. The trajectory is given by sequence of states and actions. Therefore the reward over a trajectory can be given by,

$$R(\tau) = \sum_{t=0}^{H} r(s_t, a_t), \tag{2.1}$$

where, H is a finite horizon. For infinite horizon trajectories equation 2.1 can be written as,

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t), \tag{2.2}$$

where, γ is a discount factor used to bound the return. The objective of an RL algorithm is finding the policy that maximizes this expectation:

$$\pi = \arg\max_{\pi} \mathbb{E}[R(\tau)]. \tag{2.3}$$

As we can see from equation 2.3, the RL objective contains a term for the expected accumulated reward. In the RL framework, a value function or state-value function defines the expected return of an agent in state s given by,

$$V(s_0) = E_{\pi}[\sum_{\tau} r(s_t, u_t)], \tag{2.4}$$

where, s_0 is the starting state for a trajectory. s_t and u_t are the state and action at time t where u_t is obtained from current policy π.

A core task of RL is to estimate value functions from data, and for IRL our goal is to

x	$\phi(x)$
[0, 0]	[0, 0, 0, 0]
[0, 1]	[1, 0, 0, 0]
[1, 0]	[0, 1, 0, 0]
[1, 1]	[0, 0, 0, 1]

Table 2.1: Tabular representation of function ϕ

estimate the user's hidden reward. For MDPs with continuous state and action spaces, such as robots operating in the real physical world, *function approximation* is used to estimate these quantities from finite sampled data. Function approximation is defining a continuous function $f(x) \rightarrow \mathbb{R}^d$ for input x. A function approximator can be either *parametric* or *non - parametric*. A parametric function approximator can be define as function $f(\theta, x)$ that can be described by a finite set of parameters θ, for e.g., a polynomial. Thus, the reward and/or policy can be defined using a continuous functions. A policy defined using parametric function approximator is called a *parametrized policy* and is represented as $\pi(\theta, s) = a$. In many methods, reward is represented using a *linear function approximator*. Linear function approximator assumes that there exists a *feature space* $\phi(x)$ that represents input and function's output as linear function of features with weight **w** given by,

$$f(x) = \mathbf{w}^T \phi(x).$$

An example function ϕ can be seen in table 2.2 that maps two dimensional input space x to four dimensional feature space $\phi(x)$.

A linearly approximated reward function can be given by,

$$R(s) = \mathbf{w}^T \phi(s).$$

Features for the reward function can either be manually designed, extracted, or learned from the data.

Broadly, there are two ways of learning an optimal policy as shown in figure 2.2. 1) Using direct RL and 2) Model based RL. In direct RL, the policy π or the value function

is optimized directly from the experience obtained from environment. One of the classic example of direct RL method is *Value Iteration* [SB18]. It is simple iterative algorithm that finds the optimal value function for an MDP. Initially the value function is initialized arbitrarily. Then it is updated using value function update equation until convergence given by,

$$\begin{aligned} Q(s,a) &= R(s,a) + \gamma \sum_{s' \in S} \delta(s'|s,a) V(s'), \\ V(s') &:= \max_a Q(s,a). \end{aligned} \tag{2.5}$$

As we can see from equation 2.5 value iteration algorithm needs to know transition probability to calculate $Q(s,a)$. A more realistic case for robotics is that the state transition probability is not known and the agent discovers about certain transition and reward only when it lands in that state. This is called the Reinforcement Learning problem and the *Q-learning* algorithm [SB18] can be used. *Q-learning* rule can be given by following equation,

$$Q_t(s,a) := Q_{t-1}(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q(s',a') - Q_t(s,a)), \tag{2.6}$$

where, next state s' is sampled and non-deterministic. We can see from equation 2.6 next action a' is independent of the policy and is chosen based on max Q-value, such methods where learning is independent of the current policy are called *off-policy* methods.

For low-dimentional discrete state and actions, a tabular form of Q-learning can be used but when state and actions are continuous or become high dimensional the tabular method suffers from *curse of dimensionality*. For these types of problems the Q function must be approximated using function approximation explained above. Deep networks are known for their capability of handling large data. Mnih *et al.* proposed a *Deep Q-learning method* [MKS+13] that used a deep network to approximate Q function. Deep Q learning was successfully applied to tasks with continuous state space but its action space was still discrete.

A method for using deep learning method to learn a control policy with both continu-

ous state and action spaces is proposed by Lillicrap *et al.* [LHP+13]. They adapted deep Q-learning for continuous action space and proposed a model-free, off-policy actor-critic algorithm to learn policies. Algorithm uses actor-critic approach based on the Deterministic Policy Gradient (DPG) algorithm where a parameterized *actor function* is maintained which specifies the current policy by deterministically mapping states to a specific action. The Q-function is used as a *Critic* to provide guidance during policy optimization.

A second type of reinforcement learning algorithms are categorized as Model-based RL, where experience are used to learn the model of the environment and policy or value functions are optimized based on data obtained from this dynamics model. The Dyna-Q architecture introduced by Sutton *et al.* includes all of the processes shown in figure 2.2. Planning in Dyna-Q is done using tabular Q-planning (see equation 2.6) and learning is done using tabular Q-learning method. Model learning assumes the environment to be deterministic. After each transition, the model simply adds s_t, a_t and its prediction $r(s,a), s_{t+1}$ in its table. Thus, if the model is queried with a state-action pair that has been experienced before, it simply returns the last-observed next state and next reward as its prediction. Thus algorithm for Dyna-Q can be given by algorithm 1.

Algorithm 1: Dyna-Q algorithm

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A$;
while *True* **do**
  $s \leftarrow$ current state ;
  $a \leftarrow \epsilon$-greedy(s, Q) ;
  Execute action a in state s and observe reward $r(s,a)$ and next state s' ;
  $Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \gamma \max_a Q(s',a) - Q(s,a)]$ (Q-learning step) ;
  $Model(s,a) \leftarrow r(s,a), s'$;
  while *iterations* < *n* **do**
    $s \leftarrow$ random previously observed state ;
    $a \leftarrow$ random action previously taken in s ;
    $r(s,a), s' \leftarrow Model(s,a)$;
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \gamma \max_a Q(s',a) - Q(s,a)]$;
  end
end

One of the recent model-based RL method that is used to learn control policies in con-

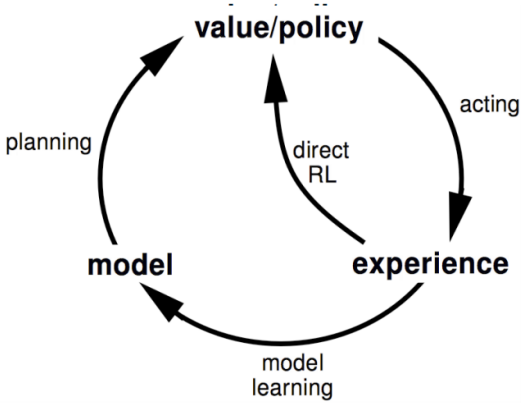


Figure 2.2: Reinforcement learning Algorithms

tinuous state and action space sample efficiently is PILCO [DR11], proposed by Deisenroth *et al.* PILCO uses a Gaussian process to represent the model of the environment and both the model and policy are improved iteratively. The model of the environment is improved by the data collected from actual experience while exploring the state and action space using the current policy. The policy is then improved using synthetic roll-outs of the policy through the learned model. This allows many policy update rounds in between data collection, thus making the algorithm sample efficient. PILCO was able to learn the swing up pendulum task with only handful of trials and a total experience of 17.5 seconds.

2.3 Imitation Learning Algorithm

Now, consider an MDP without the reward function, denoted by MDP\R, given by tuple (S, A, δ, γ). Goal of *learning from demonstration* algorithms is to learn an optimal policy in this MDP. It can broadly be done two ways. 1) The agent directly learns the policy by using the demonstrations. This is usually called *Imitation learning*. Or 2) The agent finds the reward function for the task that can explain the observed behavior, this is called *inverse reinforcement learning* problem. That is, if for a given MDP\R we know a policy π then we find reward function R such that π is optimal.

As we can see algorithm 1 uses tabular form of Q-learning and Q-planning which can-

not be used for continuous state and action space applications. In order to address this issue Schaal *et al.* [Sch97] used learning from demonstration along with Q-learning to solve a swing up pendulum task. The algorithm they proposed learned both the Q function and model incrementally by a non linear function approximator, Receptive Field Weighted Regression. They tested following learning conditions empirically.

1. *Scratch:* Q-function, model and policy π were learned from scratch using trial by trial learning.
2. *Primed Actor:* initially, π was trained from demonstration and later trained by trial by trial.
3. *Primed Model:* Initial training of model using demonstration then trial by trial learning.
4. *Primed Actor and Model:* training both π and model initially by demonstration then by trial by trial learning.

Comparing condition 1 and 3 results showed that learning model from demonstration data did not show significant improvement in speeding up learning. This is true since approximating the model of system requires dense exploration of complete state and action space. While in condition 2 demonstration had significant effect on initial performance.

Many other algorithms in literature use the fact that demonstrations helps in learning a policy for a complex task quicker than trial by trial learning. These algorithms are grouped under *imitation learning.* Imitation as a mechanism to acquire skill is studied widely in both cognitive and computer science. One of the most classic technique in imitation learning is using pure supervised learning to learn the mapping between states and actions [BS95]. But these methods suffer greatly from *covariance shift*. That is as the policy rolls out a trajectory, error compounds with each step. Because in this case distribution of states encountered by policy is different than what was seen in the training data and therefore the policy divergens from intended behaviour. Moreover, these methods assume that training data is independently and identically distributed (i.i.d), which is not the case when training data composed of trajectories where the next step is dependent on the previous state. Many approaches address this issue by using either extensive data set [BTD+16] or by using the interactive aggregation of data set rather than using a fixed data set [RBG11].

Another approach to address this issue is using a policy evaluation and improvement step instead of using direct supervised learning. Policy evaluation and improvement step can use either learned reward function from demonstrated data(IRL) [ZMBD08, AN04] or there can be a direct policy optimization step which uses demonstrated data[HGE16]. One main difficulty with these methods is that policy needs to be run on the real system for evaluation which can be very costly and even damage the system. To address this issue many approaches use *model based* imitation learning[EPPD13] where the policy is not evaluated on that real system but on the dynamics model of the system. This model can be either mathematical dynamics model of the system or can be a learned model. Methods that use deterministic models of the environment rather than probabilistic ones ignore the stochastic nature of realistic environments, thus making them less robust towards small changes. To capture this uncertainty, [RA07] proposed using distribution over reward function rather than a single reward. [EPPD13] proposed using *Gaussian Process* to learn the model of environment (see section 4.2).

Thus for the first part of the thesis, design requirements we work on are, 1) we don't want our approach to reply completely on supervised learning method. 2) We want our system to be sample efficient. 3) We want our strategy to be able to handle uncertainty in environment and noise in expert's demonstrations. 4) And, most importantly, we want our system to scale for longer trajectory demonstrations and sophisticated feature space. These requirements will be addressed one by one in chapter 4.

2.4 Gaussian Process

In section 2.3 we saw initial attempts at using model based imitation learning, which suggested using data from exploration to learn the model of the environment and using demonstration data to improve policy. In chapter 4, we use similar approach to imitation learning problem. But since the environment in real scenarios is very complex and uncertain we need a function approximator that can capture this uncertainty. Englert *et al.* [EPPD13] proposed using Gaussian Process to learn the model of the environment. This section gives a brief introduction to Gaussian Process [RW05].

Definition 1 *A Gaussian process is a collection of random variables, any finite number of*

which have a joint Gaussian distribution [RW05].

For application in the thesis, more intuitive definition can be describing it as a distribution over functions. This distribution can be fully specified by a *mean function* and *covariance function.* The *mean function* is usually defined to be zero or as mean of the available data its trying to model. There are several form of *covariance function* in the literature but most commonly used is *squared exponential* or *Radial basis function (RBF)* given as,

$$k(x, x') = \lambda_f^2 \exp(-\frac{1}{2}(x - x')^T \Lambda_x (x - x')), \tag{2.7}$$

where, two input vectors x, x' are related using a covariance function to measure correlation. $a\lambda_f^2$ and Λ_x are hyper-parameters where, $a\lambda_f^2$, reflects how much variance is present in the mapping itself and Matrix Λ_x is a diagonal matrix, whose elements represent length scale.

According to definition 1 we saw that A Gaussian process is a stochastic process such that any finite subcollection of random variables has a multivariate Gaussian distribution. That is, a collection of random variables $f(x) : x \in X$ is said to be drawn from a Gaussian process with mean function m(·) and covariance function k(·, ·) for finite set of elements $x_i \in X$. Therefore the associated distribution of finite set of random variables $f(x_i)$ can be given by,

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_m) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1), & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{bmatrix} \right), \tag{2.8}$$

where, m is mean function and k is covariance function given in equation 2.7.

2.4.1 Gaussian Process Regression

In many of model based reinforcement learning methods and imitation learning methods we saw in section 2.2 and 2.3 Gaussian process was used as dynamics model of the envi-

ronment. We can consider the prediction of next state using dynamics model as regression problem where input is current state and action and output is either next state or change in state. Let $D = \{x^{(i)}, y^{(i)}\}_{i=1}^{m}$ be the training data set for Gaussian process model. Gaussian process regression model can be given by,

$$y^{(i)} = f(x^{(i)}) + \epsilon^{(i)}, \tag{2.9}$$

where, $\epsilon^{(i)}$ is noise variable with Gaussian distribution $\mathcal{N}(0, \sigma^2)$ for i^{th} element. Now let, $T = \{x_*^{(i)}, y_*^{(i)}\}_{i=1}^{m_*}$ be the test data set. Applying definition of GP with zero mean and covariance $k(.,.)$ to training and test set we get,

$$\begin{bmatrix} \vec{f} \\ \vec{f_*} \end{bmatrix} |X, X* \sim \mathcal{N}(\vec{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}), \tag{2.10}$$

where,

$$\begin{aligned} \vec{f} &= [f(x^{(1)}), \cdots, f(x^{(m)})]^T, \\ \vec{f_*} &= [f(x_*^{(1)}), \cdots, f(x_*^{(m_*)})]^T, \\ (K(X,X))_{i,j} &= k(x^{(i)}, x^{(j)}), \\ (K(X,X_*))_{i,j} &= k(x^{(i)}, x_*^{(j)}), \\ (K(X_*,X_*))_{i,j} &= k(x_*^{(i)}, x_*^{(j)}). \end{aligned}$$

Noise in the equation 2.9 is assumed to i.i.d therefore we can write,

$$\begin{bmatrix} \vec{\epsilon} \\ \vec{\epsilon_*} \end{bmatrix} \sim \mathcal{N}(\vec{0}, \begin{bmatrix} \sigma^2\mathbf{I} & \vec{0} \\ \vec{0}^T & \sigma^2\mathbf{I} \end{bmatrix}). \tag{2.11}$$

The sums of independent Gaussian random variables is also Gaussian, therefore,

$$\begin{bmatrix} \vec{y} \\ \vec{y_*} \end{bmatrix} |X, X* = \begin{bmatrix} \vec{f} \\ \vec{f_*} \end{bmatrix} + \begin{bmatrix} \vec{\epsilon} \\ \vec{\epsilon_*} \end{bmatrix} \sim \mathcal{N}(\vec{0}, \begin{bmatrix} K(X,X) + \sigma^2\mathbf{I} & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) + \sigma^2\mathbf{I} \end{bmatrix}). \tag{2.12}$$

Using the rules for conditioning Gaussians, equation 2.12 can be written as,

$$\vec{y_*}|\vec{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*), \tag{2.13}$$

where,

$$\mu^* = K(X_*, X)(K(X, X) + \sigma^2\mathbf{I})^{-1}\vec{y},$$
$$\Sigma^* = K(X_*, X_*) + \sigma^2\mathbf{I} - K(X_*, X)(K(X, X) + \sigma^2\mathbf{I})^{-1}K(X, X_*).$$

As we can from see while making a prediction using a Gaussian Process, calculating inverse of the kernel matrix of all training data is involved. Therefore Gaussian processes scale poorly as the size and complexity of the data set increases.

2.5 Bayesian Neural Network

As we saw in section 2.4 using Gaussian process for a large data set can be challenging as Model fitting scales with $O(Dn^3)$ where n is the dataset size and D is the number of vector dimensions. Deep learning tools have gain tremendous popularity for such large data application but they do not capture uncertainty. Gal *et al.* showed that show that a neural network with arbitrary depth and non-linearities, with dropout applied before every weight layer, is mathematically equivalent to an approximation to the probabilistic deep Gaussian process [GG16].

Given a model $f_{\mathbf{w}}$ with parameter $\mathbf{w}$ and dataset $D = \{X, Y\}$, BNN can be used to find posterior over the parameter, $p(\mathbf{w}|D)$, to make predictions at new test points. Let y be the prediction of Neural Network (NN) model with L layers, $\mathbf{W}_i$ be the NN's weight matrices of dimension $K_i \times K_{i-1}$ and $\mathbf{b}_i$ be the bias vector of dimension K_i for each layer $i = 1, \cdots, L$ then the uncertainty induced at prediction of new point $\mathbf{x}$ can be given by,

$$p(y) = \int p(y|f_{\mathbf{w}}, \mathbf{x})p(\mathbf{w}|D)d\mathbf{w}. \tag{2.14}$$

The posterior distribution $p(\mathbf{w}|D)$ used in equation 2.14 is intractable. A new distribution $q(\mathbf{w})$ is used over matrices whose columns are randomly set to zero, to approximate the intractable posterior. Thus $q(\mathbf{w})$ can be defined as:

$$\mathbf{W}_i = \mathbf{M}_i.diag([\mathbf{z}_{i,j}]_{j=1}^{K_i}), \tag{2.15}$$

where, $\mathbf{z}_{i,j} \sim$ Bernoulli(p_i) for $i = 1..., L; j = 1, ..., K_{i-1}$ given some probability p_i and variational parameter matrices M_i. Variable $z_{i,j}$ when set to zero denotes j^{th} unit in layer $i-1$ dropped out as an input to layer i.The posterior $p(\mathbf{w}|D)$ can be approximated by $q(\mathbf{w})$ by minimizing the KL divergence (see section 2.6) between two distributions.

2.6 Kullback-Leibler Divergence

Kullback-Leibler divergence, or simply, the KL divergence is measure of the difference between two probability distributions over the same variable x. Unlike other distance measures like euclidean distance, KL divergence is non-symmetric measure. That is, KL divergence of probability distribution $p(x)$ from $q(x)$ is not same as divergence of $q(x)$ from $p(x)$. More specifically, KL-divergence of $q(x)$ from $p(x)$, written as $D_{KL}(p(x), q(x))$, is a measure of the information lost when $q(x)$ is used to approximate $p(x)$. For a continuous random variable x, the KL divergence can be given by equation,

$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} \tag{2.16}$$

2.7 Active Learning and Adaptive Submodularity

One of the biggest challenge in reinforcement learning or inverse-reinforcement learning is obtaining labeled data. In reinforcement learning data is obtained from experience. For the system that runs the risk of damage during exploration collecting such data is very expensive. In inverse reinforcement learning labels are provided using expert and therefore involves, human's interaction with the agent. Therefore collecting large amount of data this

way is also very expensive. An important this to notice in both the cases is that all records are not equally important from the perspective of labeling. Active leanring is technique that allows agent to decide which data is point if labeled help it most in the learning process. Thus agent can selectively ask user to label those data points and improve its data efficiency. Many approaches propose problem of selecting the query in active learning as *adaptive sub-modular optimization* [GK10], [GB10], [Gui12].

A submodular function is a set function satisfying a natural diminishing returns property. We call a set function F defined over a ground set V submodular iff for all $A \subseteq B \subseteq V$ and $v \in V \backslash B$

$$F(A + v) - F(A) \geqslant F(B + v) - F(B).$$

That is, adding an element to A, a subset of B, results in a larger gain than adding the same element to B. Consider the following example to understand diminishing returns property. We want to deploy a collection of sensors to monitor certain phenomenon. Each sensor has its own sensing range covering a certain region. Thus, we want to find best subset of location for the sensors. Therefore, intuitively, adding a sensor helps more in region where we have placed few sensors and helps less in region where we have already placed many sensors. Thus the total area covered by the sensors is a submodular function defined over all sets of locations. Adaptive version of this problem can be seen as placing sensor one by one where sensor may fail with some probability. Thus we get to observe which sensor failed and *adaptively* decide the the placement of new sensor based on this observation.

2.8 Preference Learning Algorithm

The final contribution of this thesis focuses on tasks where giving optimal demonstrations is either very challenging or impossible. In such cases, a numerical feedback signal is replaced with the assumption of a preference-based feedback signal, where the expert only needs to determine if the relation holds for a given *object pair*. Here, the object pair can be a pair of states, actions, or trajectory. A preference $z_i > z_j$ indicates z_i is preferred over z_j for object pair (z_i, z_j). Several kind of preference can be defined over object pairs [Kre98]:

- $z_i > z_j$: z_i is *strictly preferred* over z_j
- $z_i \sim z_j$: The choices are *indifferent*, meaning neither $z_i > z_j$ nor $z_j > z_i$ holds.
- $z_i \geqslant z_j$: z_i is *weakly preferred* over z_j

Just like IRL, preference learning problem is also formulated on MDP\R where goal is to learn the reward function R. There are many design parameters on which various approaches differ. The first design parameter is the learning approach of the algorithm. Either agent directly learns the policy from data that maximally comply with the preferences (see block method in figure 2.3)[Aar12, Bus14], other approaches learn the reward function from preference and then run forward RL to find optimal policy [CN17, SDSS17, Aar12]. Steps in learning loops may also differ. The method may first learn the reward function entirely and then find optimal policy [SDSS17] or both reward and policy learning happens iteratively [Aar12, ASS12]. Next parameter is type of query that is generated (see agent block in figure 2.3). Some of the approaches ask for user preference over individual states [RBG11, WF12, WF13], while other approaches ask for preference over trajectories [SDSS17, CN17]. Methods which ask for preference over complete trajectory suffers from *temporal credit assignment problem* [Sut84]. Temporal credit assignment can be seen in two ways, in direct policy learning its inability to tell which action in complete trajectory resulted in the desired behavior. In reward learning, it can be seen as an inability to tell which part of the trajectory was responsible for influencing expert's preference (hight reward region). To address this issue, many methods propose on taking feedback over segments of trajectory rather than complete trajectory [Ols17]. Other parameter is a type of feedback provided (see Expert's block in figure 2.3). Many approaches ask expert to provide ratings over the demonstrated trajectory [Ols17]. Some approaches ask their user to critique the demonstrated trajectory as good or bad [CN17]. Other approaches ask the expert to rank the trajectories [AM10]

Thus just like imitation learning, for the second part of the thesis, we describe our approach with respect to each of above mentioned parameters. The main focus of our approach is to alleviate the credit assignment problem while still keeping the query natural for the expert. We also want our method to be sample efficient in order to reduce interaction time with the expert. The response that user provides should be easy and small, thus

2.8 Preference Learning Algorithm

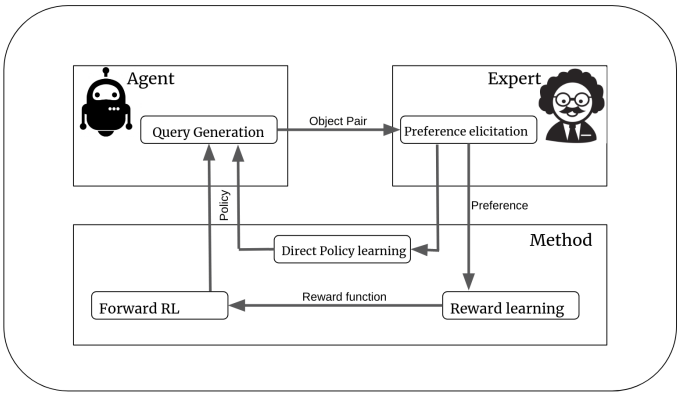


Figure 2.3: Block diagram: Different approaches to Preference learning.

reducing expert overhead.

3

Experiment Task Description

Imitation learning and inverse reinforcement learning are being applied in many fields and applications. Its application in Robotics is specifically challenging because it involves difficulties such as 1) complex and constantly changing environment. 2) Difference in policy space and behavior space of the robot. That is, as a human we know what behavior robot should exhibit, like go in a circle while avoiding the obstacle, but we don't know how to instruct it to do so in it its policy space which involves moving the motor with certain velocity and steering at certain angle. 3) longer behavior sequences and, 4) complex system dynamics, etc.

Many methods in the literature address these problems or a combination of them. In this thesis, we wanted to progressively study all the different aspects of these problems. Starting with the problem of uncertainty associated with the environment and long, complex demons-tration sequences in such environments. This is addressed in our proposed adaptation of the imitation learning method described in chapter 4. We next study the problems of the dependence of the optimal policy of the robot on the performance of the expert, the difference between the behavior and policy spaces as discussed above and the correspondence problem due to the difference in the anatomy and dynamics of the expert and robot. Chapter 4 presents an integrated solution to all of the problems mentioned above. In order to have the same experiment setup for the study which involved all of these difficulties, we chose autonomous driving as our experimentation task.

Learning a driving maneuver for a particular scenario involves a long trajectory with

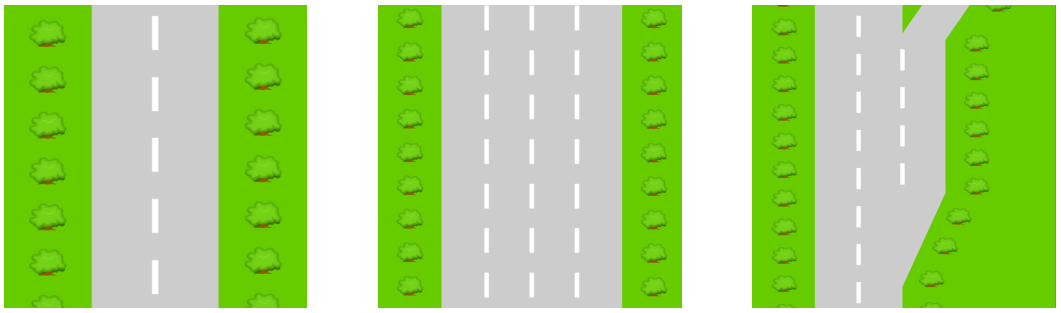
a complex state and action space of the autonomous vehicle. Moreover, the World Health Organization has indicated that about 1.2 million people die in road accidents [Org 15] each year [Org15], thus showing humans are not the ideal demonstrator for this task. There are many different aspects in driving that influence human preference of one driving behavior over another, such as defensive driving vs. aggressive driving, shortest time vs. increasing safety by slow driving or any combination of these. Therefore it is important to understand in what part of the maneuver the user preferred which style. Alternatively, what part of the trajectory most influenced his/her preference. Thus the autonomous driving task best suited our experimental needs. Due to a lack of access to a physical self-driving car, we perform all of our experiments on a simulation environment that we built.

In this chapter, section 3.1 describes the simulator for all the experiments in the thesis. Section 3.2 describes how the expert data is collected for imitation learning experiments and section 3.3 describes how expert's preferences are collected for active preference learning experiments.

3.1 Simulation Environment - Conduite-Simulateur (ConSim)

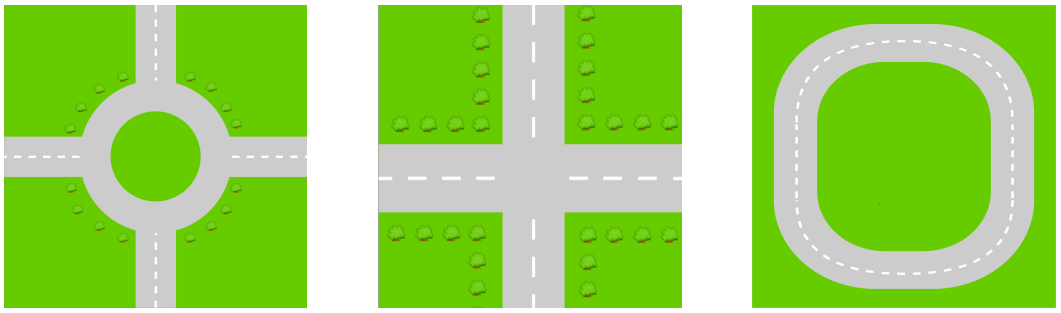
The simulation environment used for all of our experiments is a low-dimensional driving simulator programmed in PyGame. Significant work was put into developing this simulator to create an appropriate and suitable experimental setup.

Our simulator provides a variety of different driving scenarios with the flexibility to configure traffic. For convenient usage and to follow the same conventions as state-of-the-art RL experiment platforms in order to facilitate integration with code developed for those simulators, *ConSim* has the same interface as OpenAI Gym [BCP+16]. It provides flexible control over environment parameters such as traffic and behavior of the NPC to allow for greater control in experimental design. Not only does it provide control over traffic but it can also load various scenario maps, such as those visualized in figures 3.1 and 3.2, to facilitate experimentation on different agent behaviors.



(a) Two Lanes (b) Four lanes (c) Merge

Figure 3.1: ConSim Simulator various scenarios



(a) Round A Bout (b) Intersection (c) Track

Figure 3.2: ConSim Simulator various scenarios

3.2 Expert's Data Collection

The simulator was built not only for forward reinforcement learning but also with inverse reinforcement learning in mind. We mainly focus on two types of learning from demonstration methods: 1) Imitation learning and 2) Active Preference IRL. Therefore, the simulator provides mechanisms for the expert to give learning inputs to the agent. There are two different ways to collect data from the expert. 1) The simulator provides the Robot Operating System (ROS) support with a joystick interface. The expert can drive the car using the joystick in different scenarios, and data is collected in ROS-bags, which can later be used by a learning algorithm. 2) When the user study is not involved, and a large amount of data is to be collected for testing, driving using a joystick every time can be a very time-consuming process. Therefore, the simulator also provides preprogrammed experts for different sce-

narios which automatically drive and collect data. For example, figure 3.3 shows our software expert's trajectory for the overtaking scenario. Similarly figures 3.4, 3.5 and 3.6 show the expert's trajectory for the merging, crossing an intersection and round-a-bout scenarios respectively.

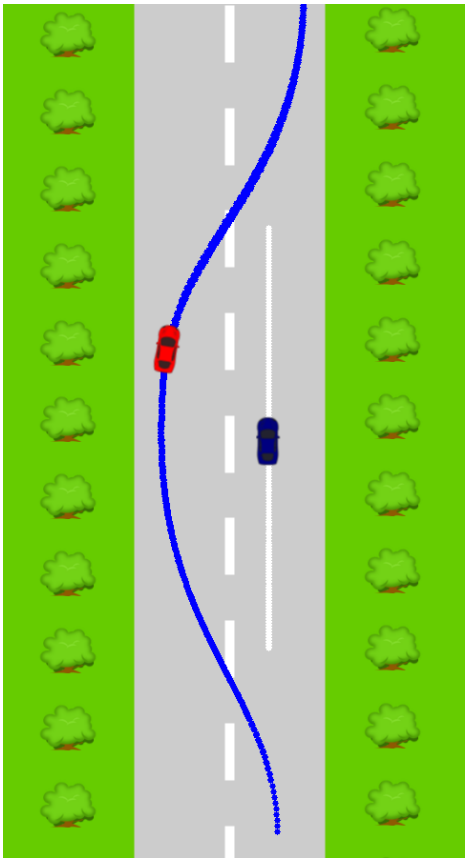


Figure 3.3: Our overtaking scenario where the red car is the agent and the blue car is the NPC. The lines are their respective trajectories.

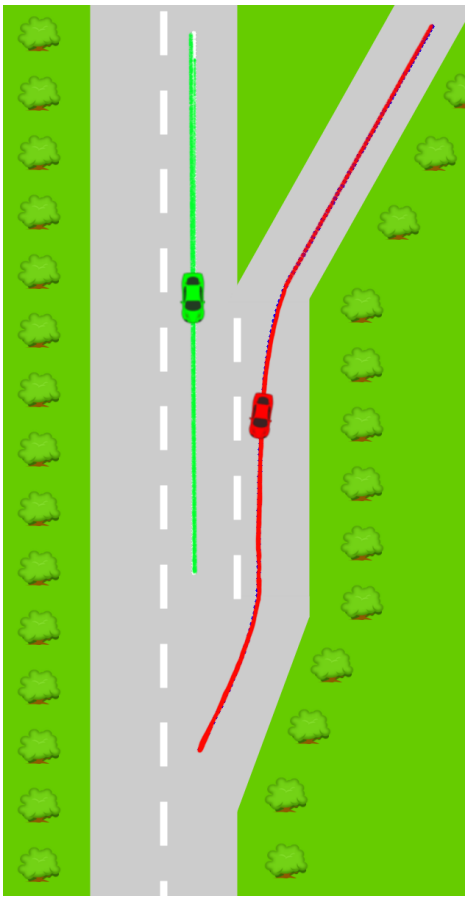


Figure 3.4: Our merging scenario where the red car is our agent car and the green car is the NPC. The lines are their respective trajectories.

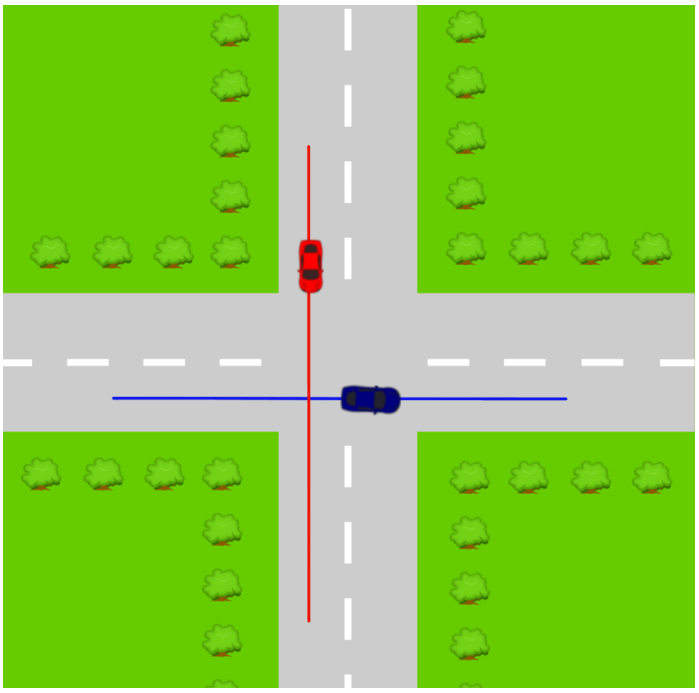


Figure 3.5: Our intersection scenario where the red car is our agent and the blue car is the NPC. The lines are their respective trajectories.

Please note that temporal information is missing from the image, hence the trajectories seem to intersect.

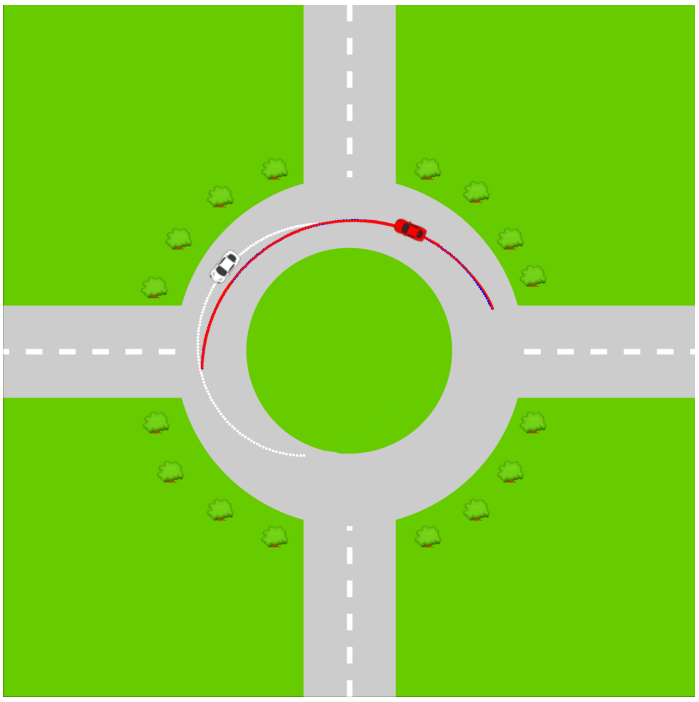


Figure 3.6: Our round-a-bout scenario where the red car is our agent car and the white car is the NPC. The lines are their respective trajectories.

Please note that temporal information is missing from the image, hence the trajectories seem to intersect.

3.3 Expert's Preference Collection

For active learning, our simulator provides a user interface showing two different behaviors to choose from (see figure 3.7). The user can then click on the box of the preferred behavior to provide input. We also provide a modification of this functionality to fit our adaptation of active learning, which requires the user to give input over segments of the trajectory rather than on the complete trajectory. The Active preference box for that method is shown in figure 3.8. In this interface user enters a $k-bit$ sequence of 1 and -1 for k segments. Where 1 represents user prefer segment from trajectory A and -1 represents user prefer segment from trajectory B.

3.3 Expert's Preference Collection

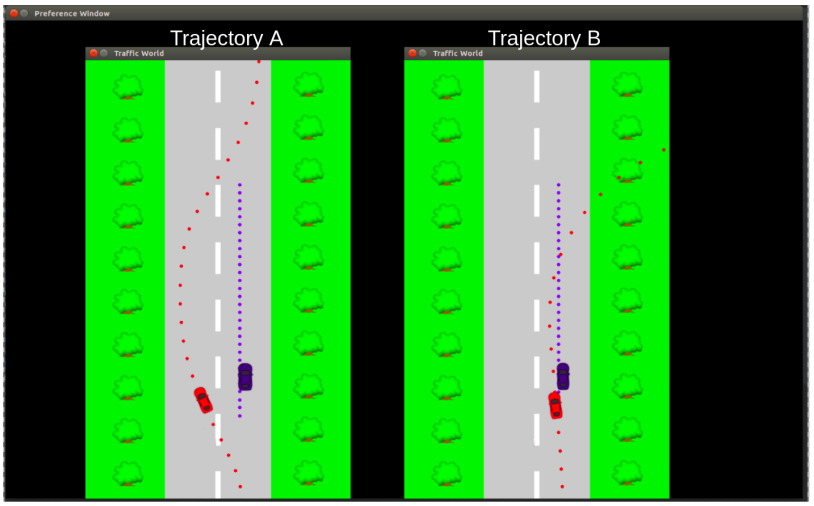


Figure 3.7: Preference interface showing two different overtaking trajectories, without segmentation

Please note that temporal information is missing from the image. Preference box actually shows a moving gif.

3.3 Expert's Preference Collection

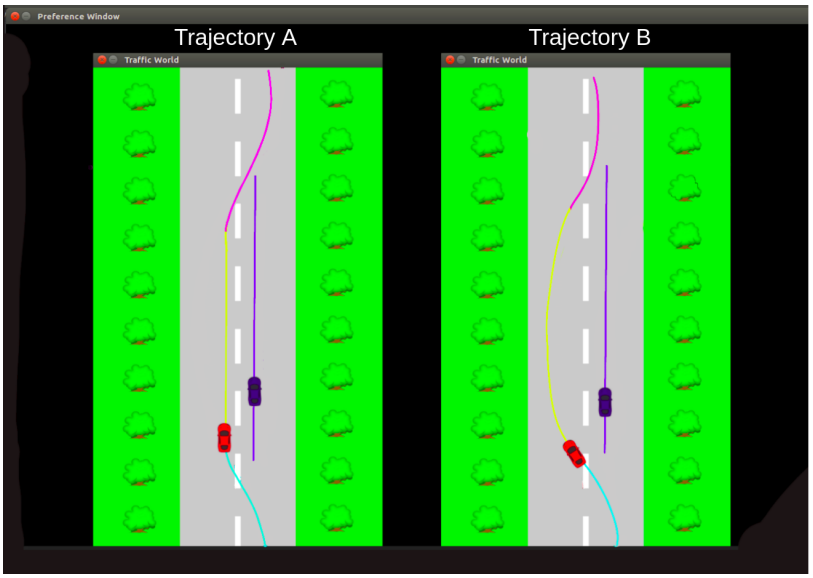


Figure 3.8: Preference interface showing two different overtaking trajectories, with segmentation

Please note that temporal information is missing from the image. Preference box actually shows a moving gif with segments highlighted in various colors.

Part II

Thesis Contributions

4

Imitation Learning

In this chapter we adapt model based imitation learning method proposed by Englert *et al.* [EPPD13] to make it more scalable for longer and complex trajectories. Section 4.1 introduces terminologies used throughout the chapter and describes the problem we are trying to solve. Section 4.2 explains the previously proposed approach to learning the dynamics model of the environment and our adaptation of the method to make it suitable for large data set. This section relies on background given in section 2.4 and 2.5. Section 4.3 describes our Trajectory matching approach to obtain the optimization objective for learning the imitation policy. It also explains how the expert and policy data can be represented as probabilistic distributions. After representing the policy distribution and expert distribution and obtaining the optimization objective, section 4.4 describes how the policy is actually optimized. Finally, section 4.5 and 4.6 describes the experiment we do to test our approach and the results obtained and discuss the benefits and limitation of the approach respectively.

4.1 Problem Statement

We start by introducing the notations that will be used throughout this chapter. The state of the system is denoted by $x \in \mathbb{R}^d$ and the actions are denoted by $u \in \mathbb{R}^e$ respectively, where $d, e \in \mathbb{N}$. Furthermore, the trajectory, denoted by τ, is defined as sequence of states $x_0, x_1, ...x_H$ for a fixed time horizon H. Our goal is to learn a policy π with parameter θ such that $u = \pi(x, \theta)$ where u approximately imitates experts demonstrations.

As an input to the algorithm, we take the expert's demonstrations. We assume that ex-

perts provide near-optimal demonstrations of behavior for each task T. We also assume that the expert is able to provide this demonstration in the robot's state and action space. The input demonstrations from an expert are received in the form of N trajectories with fixed time horizon H. We represent these N trajectories as a probability distribution denoted by $p(\tau^{exp})$. The initial state of the robot while collecting these trajectories is sampled from a distribution over initial state $p(x_0)$. The main objective of the experiment is to find policy π such that distribution over the trajectories predicted by π, denoted by $p(\tau^{\pi})$ matches the expert's trajectory distribution $p(\tau^{exp})$. Using probability distributions has many advantages. For example, it allows us to capture the uncertainty in the system's dynamics. It also allows room for variability in the expert's behavior. Consider the task of overtaking in a two-lane scenario. The expert might choose to overtake the NPC at variable distances with different speeds. We want to learn a policy that will take into account all these variabilities. Thus using a probability distribution allows us room for such differences in the expert's demonstration. For measuring similarity between $p(\tau^{exp})$ and $p(\tau^{\pi})$ we use Kullback-Leibler (KL) divergence [Kul59]. Thus the imitation learning objective of the algorithm is to find a policy such that the following loss is minimized:

$$\pi^* \in \arg\min_{\pi} KL(p(\tau^{exp})||p(\tau^{\pi})). \tag{4.1}$$

Since we want the distribution generated by the policy to get closer to the expert's distribution, that is we want to approximate expert's distribution by trajectories generated by policy we take KL divergence of $p(\tau^{\pi})$ from $p(\tau^{exp})$ (see section 2.6) and not the other way around.

4.2 Probabilistic Forward Model using BNN

In order to collect predicted trajectory data from π we need to execute the policy multiple times. This can be very expensive to run on a real system and can also cause physical damage. Therefore we make use of the forward model of the system dynamics, thus making the process sample efficient and practical for our task. We learn the forward model by collecting initial data by applying random input to the system and collecting the system's response to these actions. The forward model of a system is a function that maps the current

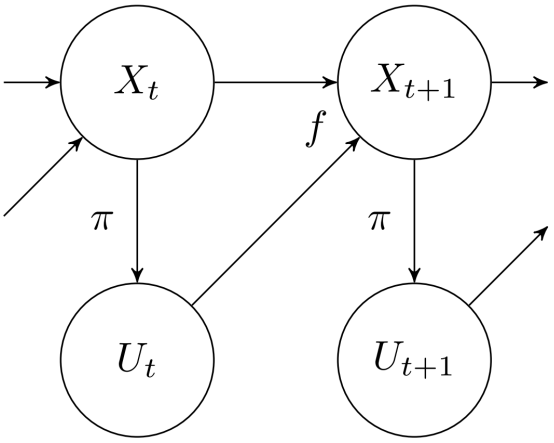


Figure 4.1: Generative model showing generation of predicted trajectories.

state and action of the system, (x_t, u_t), to the next state x_{t+1}.

In our case, we define the dynamics of the system using the probabilistic model since it is able to capture the uncertainty in the environment as well as the noise in demonstration provided by different users. Using this probabilistic dynamics model, the policy generates predicted trajectories from an initial state distribution $p(x_0)$ until time horizon T, $p(x_T)$. This prediction of the state distribution follows a generative model given by figure 4.1 At a certain state x_t, the policy predicts action u_t both along with dynamics model predict samples y_{t+1} for next state which forms a distribution over next state. Next state x_{t+1} is sampled from this distribution. This can be summarized by the following equations:

$$\begin{aligned} x_0 \sim p(x_0), \;\; u_t \sim \pi_\theta(u_t|x_t), \\ f \sim p(f), \;\; y_{t+1} \sim p(x_{t+1}|x_t, u_t, f), \\ \text{where, } \; x_{t+1} \sim N(x_{t+1}|\mu_{yt}, \sigma_{yt}). \end{aligned} \tag{4.2}$$

The main difference between our [EPPD13] and our implementation is the modeling of the dynamics model. Function f is represented by a Gaussian Process (see section 2.4) in implementation of [EPPD13]. With Gaussian process, model fitting scale by $O(Dn^3)$ and simulated roll-out trajectories scale by $O(D^3n^2)$, where n is data set size and D is state dimensions. Thus in a scenario where trajectory horizon is long, and state space of the system

is complex Gaussian processes scale poorly, limiting their use. We represent our dynamics model using Bayesian Neural Network, thus making it scalable to high dimensional observation space.

We use a BNN model learning (see secion 2.5) implementation as described in [HMD18] to learn our dynamics model. Given a function f with parameter θ_m and data $D = (X, Y)$ we find posterior $p(\theta_m|D)$ to make prediction for new point. The uncertainty in this distribution induces uncertainty in model prediction at new point. Using the true posterior for predictions on a neural network is intractable therefore methods based on variational inference are used to approximate posteriors to make prediction. Fitting of the model is done by minimising KL divergence between the true and the approximate posterior by optimising the following objective,

$$L(\theta_m) = -L_D(\theta_m) + D_{KL}(q(\theta_m)||p(\theta_m)), \tag{4.3}$$

where, $L_D(\theta_m)$ is the expected value of the likelihood $p(D|\theta)$, $q(\theta_m)$ is the approximate posterior and $p(\theta_m)$ is a user-defined prior on the parameters. Negative sign in the objective function given by equation 4.3 indicates gradient ascent to maximize a likelihood function. In our case to we build dynamic model using data set which consist of tuple $\{(x_t, y_t), \delta_x\}$ where $\delta_x = x_{t-1} - x_t \in R^E$ is change in state x when applied action u and $(x_t, y_t) \in R^{(D+E)}$ are state action pair at time t.

4.3 Model-based Imitation Learning by Probabilistic Trajectory Matching

The goal of our experiments in this chapter is to find policy parameters θ_p such that policy π^* imitates the expert. That is, the distribution of trajectories as predicted by the policy π^* is similar to expert's demonstration distribution. This is done my minimizing KL divergence between distribution $p(\tau^{exp})$ and $p(\tau^{\pi})$ using equation 4.1.

Recall (from section 2.6) that the KL divergence is a difference measure between two probability distributions and is defined for continuous distribution as:

$$KL(p(x)||q(x)) = \int p(x) \log \frac{p(x)}{q(x)} dx, \tag{4.4}$$

where $p(x)$ and $q(x)$ are both continuous probability distributions.

In the case of Gaussian distributions, where $p(x) \sim \mathcal{N}(x|\mu_0, \Sigma_0)$ and $q(x) \sim \mathcal{N}(x|\mu_1, \Sigma_1)$, the KL divergence equation can be stated as a closed form given by

$$KL(p||q) = \frac{1}{2} \log |\Sigma_1^{-1}\Sigma_0| + \frac{1}{2} tr(\Sigma_1^{-1}((\mu_0 - \mu_1)(\mu_0 - \mu_1)^T + (\Sigma_0 - \Sigma_1)). \tag{4.5}$$

In our case, we use this closed form of the KL divergence equation to measure distance between distributions. Therefore trajectory data collected from the expert and predicted from trajectories are represented as Gaussian distributions. Data is collected as set of trajectories represented by $D_\tau = \tau_0, \tau_1, ...\tau_M$. Therefore, approximate distribution over trajectories $p(\tau) = p(x_0, x_1.....x_H)$ using a Gaussian can be given by

$$p(\tau) \approx \prod_{t=0}^{H} p(x_t) = \prod_{t=1}^{H} \mathcal{N}(\mu_t, \Sigma_t). \tag{4.6}$$

4.3.1 Expert Distribution Representation

In this section we describe how the expert's demonstration is represented as a Gaussian distribution. We collected and time align (each trajectory has the same number of states) the demonstration data which is represented by $D_{\tau_{exp}} = \tau_0, \tau_1, ...\tau_M$. Then we compute the mean and covariance matrix of the marginal distribution $p(x_t)$ given by equation 4.7

$$\widehat{\mu}_t^{exp} = \frac{1}{M} \sum_{i=0}^{M} x_t^i, \;\; \widehat{\Sigma}_t^{exp} = \frac{1}{M-1} \sum_{i=0}^{M} (x_t^i - \widehat{\mu}_t^{exp})(x_t^i - \widehat{\mu}_t^{exp})^T. \tag{4.7}$$

In equation 4.7 x_t^i is state at time t in i^{th} demonstrated trajectory in data set $D_{\tau_{exp}}$. Therefore the Gaussian distribution over the complete set $D_{\tau_{exp}}$ can be given by the equa-

tion 4.8

$$p(\tau_{exp}) = \mathcal{N}(\widehat{\mu}^{exp}, \widehat{\Sigma}^{exp}) = \mathcal{N}\left(\begin{bmatrix} \widehat{\mu}_0^{exp}, \\ \widehat{\mu}_1^{exp}, \\ \vdots \\ \widehat{\mu}_H^{exp} \end{bmatrix}, \begin{bmatrix} \widehat{\Sigma}_0^{exp}, & 0, & \cdots & 0 \\ 0, & \widehat{\Sigma}_1^{exp}, & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \widehat{\Sigma}_H^{exp} \end{bmatrix}\right). \tag{4.8}$$

As we can see from equation 4.8, we assume that Σ_τ is block diagonal without cross-correlation among states at different time steps.

4.3.2 Policy Distribution Representation

In the above section, we have described the representation of expert data using a Gaussian distribution. In this section, we describe how to represent trajectories predicted by the policy as the Gaussian distribution. Our implementation in this section differs from that of [EPPD13] because we use a Bayesian Neural Network to represent our forward model rather than a Gaussian process. The advantages of this are described in section 4.2.

The learned BNN model as described in section 4.2 is iteratively used to predict the state distribution $P(x_1), ...P(x_H)$ for a given policy π and distribution over initial state $P(x_0)$.

To predict a complete roll-out of the trajectory, the dynamics model must pass uncertain dynamics outputs from a given time step as uncertain input into the dynamics model in the next time step. In a Gaussian process, this can be handled analytically [EPPD13]. In-order to handle input uncertainty in BNN dynamics model we use the particle method with PEGASUS evaluation given in algorithm 2 as described in [HMD18] (see figure 4.2).

As seen from algorithm 2 the mean μ_t and standard deviation Σ_t are calculated for each time step of the roll out trajectory. This μ_t and Σ_t along with mean $\widehat{\mu}_t^{exp}$ and standard deviation $\widehat{\Sigma}_t^{exp}$ of the expert (see equation 4.7) are used to calculate the imitation cost at every time step given by equation 4.5.

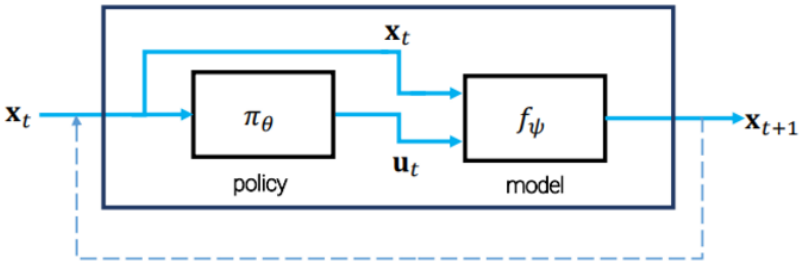


Figure 4.2: Agent's trajectory roll-out generation

Algorithm 2: Predict agent's trajectory roll-outs

Define time horizon H;
Initialize set of K particles $x_0 \sim P(x_0)$;
Sample noise for dynamics $\{\mathbf{z}_w^{(k)} | 1 \leqslant k \leqslant K\}$;
Sample state noise $\{\mathbf{z}_t^{(k)} | 1 \leqslant k \leqslant K, 1 \leqslant t \leqslant H\}$;
Sample noise for policy $\{\mathbf{z}_\theta^{(k)} | 1 \leqslant k \leqslant K\}$;
while $t < H$ **do**
  while $k < K$ **do**
    Sample BNN dynamics model weights $\mathbf{w}^{(k)} = g1(w, \mathbf{z}_w^{(k)})$;
    Sample θ for policy $\theta^{(k)} = g2(\theta, \mathbf{z}_\theta^{(k)})$;
    Evaluate policy $\mathbf{u}_t^{(k)} = \pi_{\theta^{(k)}}(x_t^{(k)})$;
    Propagate state through dynamics model $x_{t+1}^{(k)} = f_{w^{(k)}}(x_t^{(k)}, u_t^{(k)})$
  end
  Fit mean $\mu_{x_{t+1}}$ and covariance $\Sigma_{x_{t+1}}$;
  for $k < K$ **do**
    $x_{t+1}^{(k)} = \mu_{x_{t+1}} + \Sigma_{x_{t+1}} \mathbf{z}_t^{(k)}$
  end
end

4.4 Deep PILCO Learning

Now we have all of the elements for policy optimization. Deep-PILCO uses back-propagation through time (BPTT) to estimate the policy gradients $\nabla_\theta J(\theta)$. The objective function is calculated at each time step given by,

$$J^\pi(\theta) = \sum_{t=0}^{t=H} \mathrm{KL}(p(x_t^{exp})||p(x_t^\pi)) = \sum_{t=0}^{t=H} \mathrm{KL}(\mathcal{N}(\mu_t^{exp}, \Sigma_t^{exp})||\mathcal{N}(\mu_t^\pi, \Sigma_t^\pi)). \tag{4.9}$$

We use implementation of Deep PILCO as proposed by [HMD18] for their two improvements made over traditional Deep-PILCO algorithm. 1) Reducing variance and improving convergence by drawing all the random numbers needed for simulating trajectories at the beginning of the policy optimization and keeping them fixed as the policy parameters are updated. 2) Dealing with vanishing and exploding gradients when computing them via BPTT when trajectories with longer time horizons are involved by using ReLU activations for the policy and dynamics model,and clipping the gradients to have a norm at most equal to pre-decided clipping values.

The complete algorithm for imitation learning using gradient clipping can be given by algorithm 3.

4.5 Experiments and Results

We tested our algorithm on the various autonomous driving scenarios mentioned in chapter 3. Results for each scenario are given in different subsections. Each subsection starts by showing the expert's demonstrated data (for example see figure 4.3). It then shows policy roll-outs collected by applying actions to the real environment. We can see that the behavior of the policy gets closer to the expert's behavior as learning iterations progress (for example, see figure 4.4). Roll-outs of action on the environment are also shown which match the expert's action roll outs as iterations progress (for example, see figure 4.5). Next, we also show trajectory roll outs over dynamics model that demonstrate both the policy and the model get better with the number of iterations (for example, see figure 4.6).

Algorithm 3: Model based imitation learning using BNN

Define time horizon H;
Initialize set of K particles $x_0 \sim P(x_0)$;
initialize iterations for optimization N_{opt} ;
Sample noise for dynamics $\{\mathbf{z}_w^{(k)} | 1 \leqslant k \leqslant K\}$;
Sample state noise $\{\mathbf{z}_t^{(k)} | 1 \leqslant k \leqslant K, 1 \leqslant t \leqslant H\}$;
Sample noise for policy $\{\mathbf{z}_\theta^{(k)} | 1 \leqslant k \leqslant K\}$;
while *n* < N_{opt} **do**
  while *t* < *H* **do**
    while *k* < *K* **do**
      Sample BNN dynamics model weights $\mathbf{w}^{(k)} = g1(w, \mathbf{z}_w^{(k)})$;
      Sample θ for policy $\theta^{(k)} = g2(\theta, \mathbf{z}_\theta^{(k)})$;
      Evaluate policy $\mathbf{u}_t^{(k)} = \pi_{\theta^{(k)}}(x_t^{(k)})$;
      Propagate state through dynamics model $x_{t+1}^{(k)} = f_{w^{(k)}}(x_t^{(k)}, u_t^{(k)})$;
    end
    Fit mean $\mu_{x_{t+1}}$ and covariance $\Sigma_{x_{t+1}}$;
    for *k* < *K* **do**
      $x_{t+1}^{(k)} = \mu_{x_{t+1}} + \Sigma_{x_{t+1}} \mathbf{z}_t^{(k)}$
    end
    ;
  end
  Evaluate objective given in equation 4.9 and average over each particle. ;
  Compute gradient estimate $\nabla_\theta J(\theta)$;
  if $\nabla_\theta J(\theta) > \epsilon$ **then**
    $\nabla_\theta J(\theta) \leftarrow \epsilon \frac{\nabla_\theta J(\theta)}{||\nabla_\theta J(\theta)||}$
  end
  Update θ by stochastic gradient descent step.
end

4.5.1 Two Lane Over Taking

To represent the state for two-lane overtaking scenarios, the features are the x and y position of the agent, as well as its heading and speed and the y position of the NPC. These are given by tuple, $[x, y, \theta, v, y_{npc}]$. The actions used by the policy to direct the agent are the continuous value of acceleration and steering angle. Expert data is collected using pre-programmed expert and variability in demonstrations are introduced by injecting noise in acceleration value of the agent. Figure 4.3 shows the trajectory state and action data for horizon $H = 148$. The goal of the agent is to try to generate trajectories that are similar to the expert's.

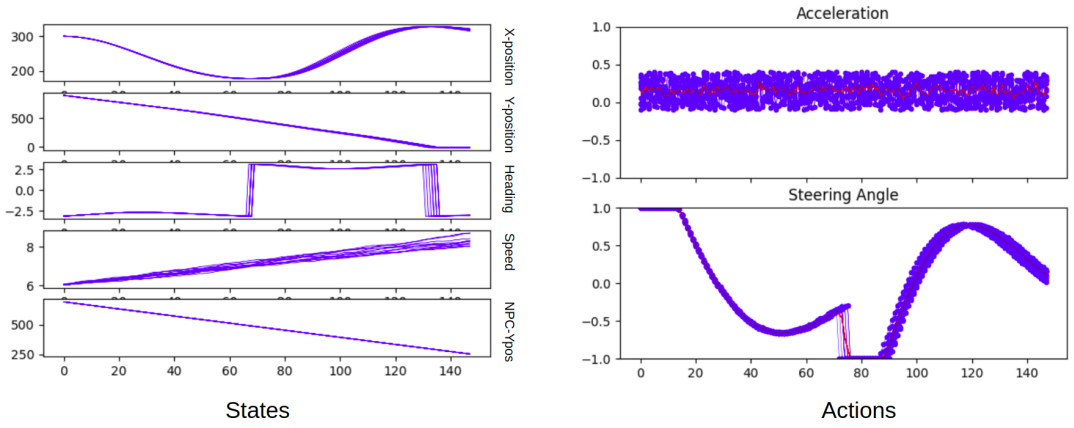


Figure 4.3: Overtaking Expert's Demonstration State Action Roll-outs

Trajectory roll-outs from Expert's Demonstration (H = 148). Agent will try to roughly match these roll-outs by minimizing KL divergence loss.

In figure 4.4 we can see that when an initial policy with random weights (iteration 0) is run on the environment, the behavior of the policy (trajectory in red) is far off from expert's behavior (trajectory in green). As the iterations proceed, the trajectory roll out of the policy gets closer to the expert's roll-out. Similarly, in figure 4.5, we can see that at with each iteration, actions output by the policy get closer to the expert's actions.

4.5 Experiments and Results

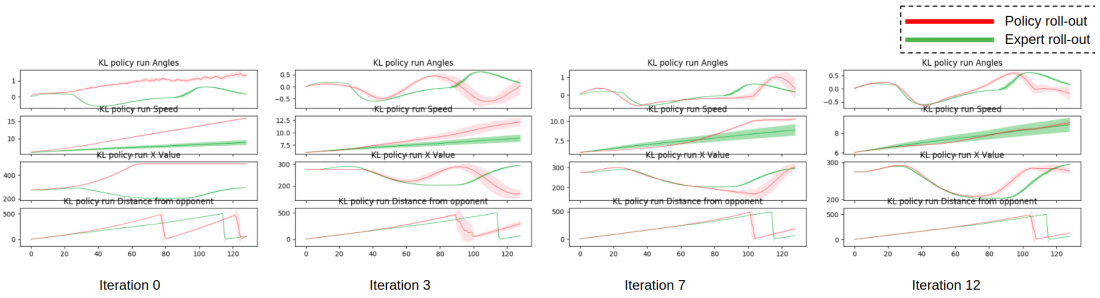


Figure 4.4: Overtaking Policy roll-outs on Environment

We can see the roll outs of policy on environment progressively gets closer to expert's roll-outs.

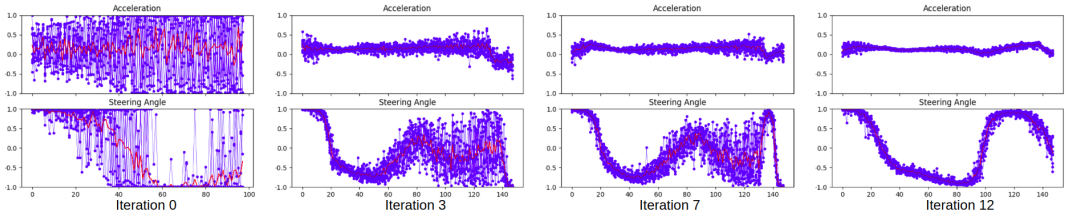


Figure 4.5: Overtaking Policy Action on Environment

We can see the actions of policy on environment progressively gets closer to expert's actions.

In the proposed method, the weights of both the policy as well as the dynamics model are updated with each iteration. With each iteration, more data is collected from the environment, and the dynamics model is retrained with this data. Therefore as iterations progress, the behavior of the dynamics model should get closer to the real environment and roll-out of the policy on the model should become increasingly similar to the expert's roll-out on the environment. In figure 4.6, at iteration zero, we can see that the model of the environment has huge uncertainty. That is, for the same initial state and action, the model predicts a different next stat. Therefore, there is a huge variability in roll-outs (high variance). As iterations progress, the uncertainty of the model decreases and policy roll-out on model gets similar to the expert's roll-out on the environment.

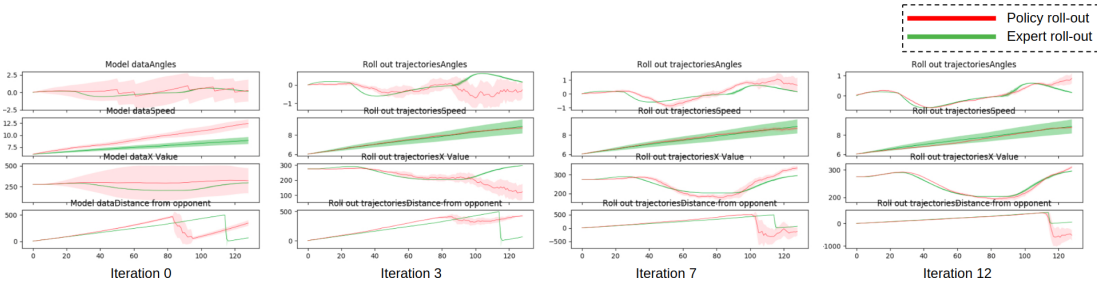


Figure 4.6: Overtaking Policy roll-outs on Dynamics Model

We can see that model uncertainty progressively reduces with each iteration and agent's behavior gets closer to expert's behavior. Showing that both model and policy improve with iterations.

Similar results are given for different driving scenarios in the following section.

4.5.2 Lane Merging

Lane merging scenario uses same features as in section 4.5.1. Expert's data is collected using a pre-programmed expert and is shown in figure 4.7. Incrementally updating Policy state and action roll outs on the environment is shown in figure 4.8 and 4.9 respectively. Figure 4.10 show model of the environment incrementally getting closer to the real environment.

4.5 Experiments and Results

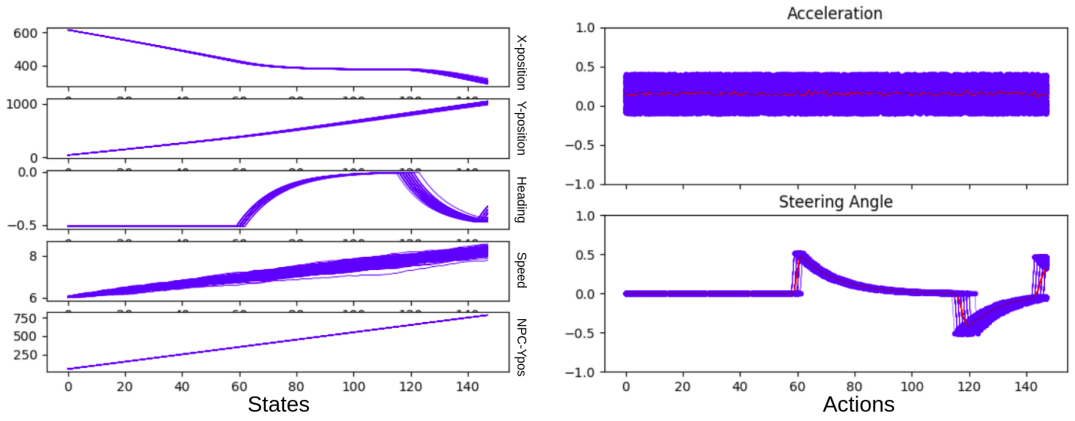


Figure 4.7: Merging Expert's Demonstration State Action Roll-outs

Trajectory roll-outs from Expert's Demonstration (H = 148). Agent will try to roughly match these roll-outs by minimizing KL divergence loss.

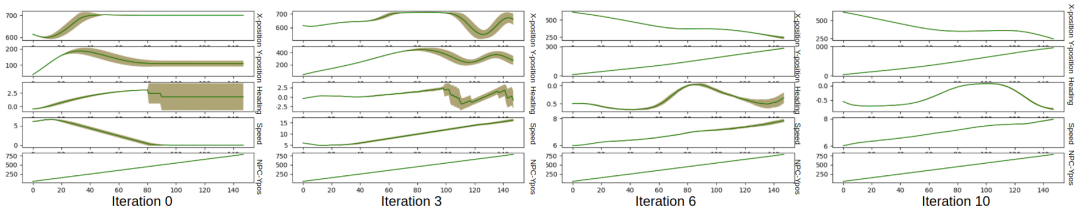


Figure 4.8: Merging Policy roll-outs on Environment

We can see the roll outs of policy on environment progressively gets closer to expert's roll-outs.

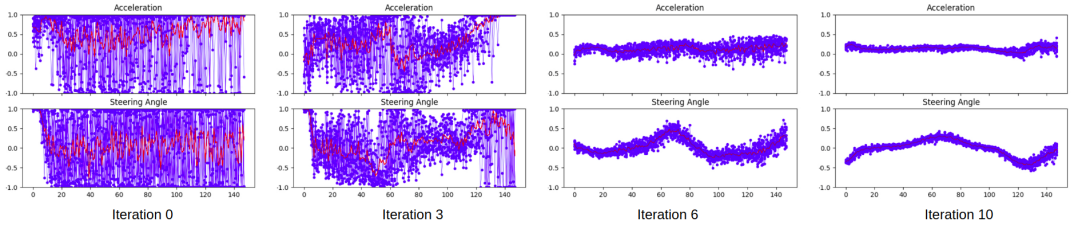


Figure 4.9: Merging Policy Action on Environment

We can see the actions of policy on environment progressively gets closer to expert's actions.

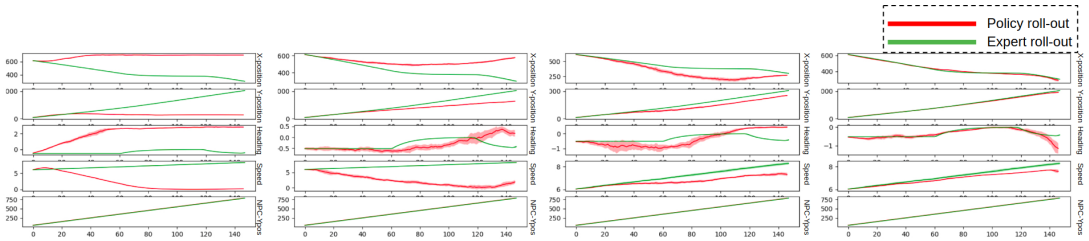


Figure 4.10: Merging Policy roll-outs on Dynamics Model

We can see that model uncertainty progressively reduces with each iteration and agent's behavior gets closer to expert's behavior. Showing that both model and policy improve with iterations.

4.5.3 Round-A-Bout

In the case of round-a-bout scenarios, the features used to represent the state are the x and y positions of the agent, its heading and speed and the distance from the NPC, given by tuple, $[x, y, \theta, v, d_{npc}]$. The actions that the policy uses to control the agent are the continuous value of acceleration and steering angle. Similar analysis is done for this scenario as well like Merge and Two lane over take where, Expert's data is collected using a pre-programmed expert and is shown in figure 4.11. Incrementally updating Policy state and action roll outs on the environment is shown in figure 4.12 and 4.13 respectively. Figure 4.14 shows the model of the environment incrementally getting closer to the real environment.

4.5 Experiments and Results

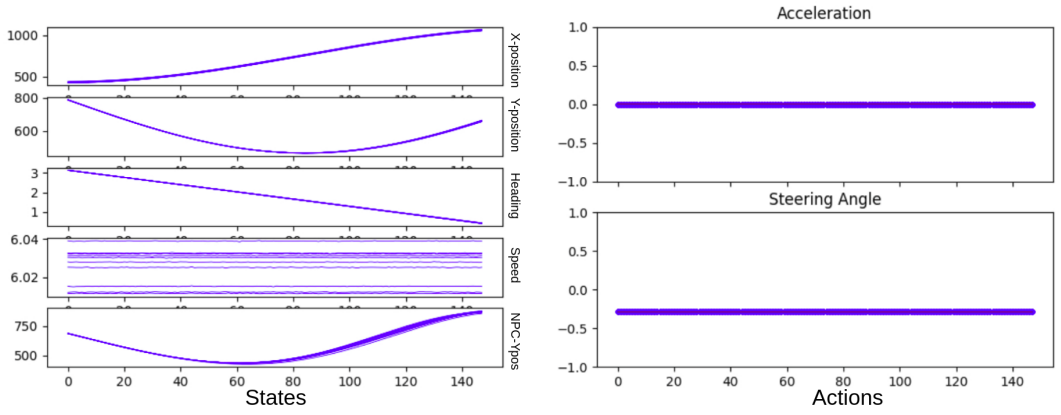


Figure 4.11: Round-a-bout Expert's Demonstration State Action Roll-outs

Trajectory roll-outs from Expert's Demonstration (H = 148). Agent will try to roughly match these roll-outs by minimizing KL divergence loss.

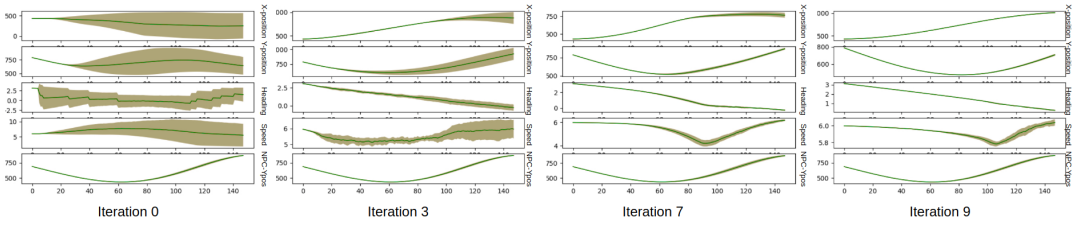


Figure 4.12: Round-a-bout Policy roll-outs on Environment

We can see the roll outs of policy on environment progressively gets closer to expert's roll-outs.

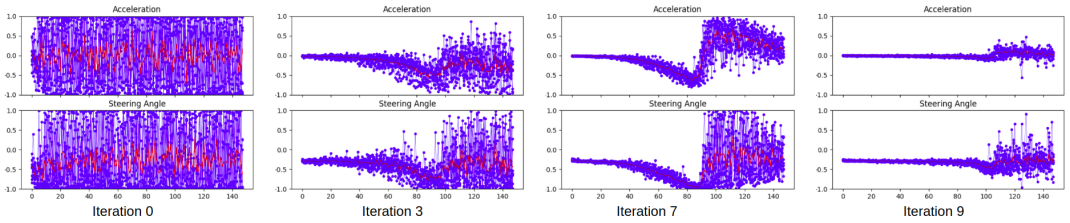


Figure 4.13: Round-a-bout Policy Action on Environment

We can see the actions of policy on environment progressively gets closer to expert's actions.

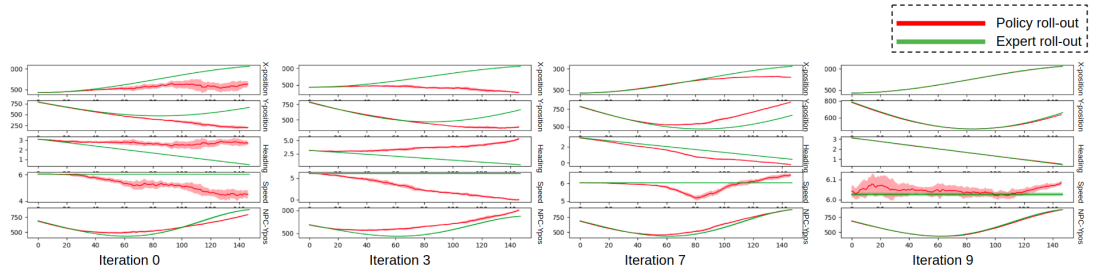


Figure 4.14: Round-a-bout Policy roll-outs on Dynamics Model

We can see that model uncertainty progressively reduces with each iteration and agent's behavior gets closer to expert's behavior. Showing that both model and policy improve with iterations.

4.6 Discussion

From above results, we can see that proposed method can handle both difference in expert's demonstration by making use of distribution over trajectory, and uncertainty in predicting the next state given current state.

The main difficulty in using this method is that to use the KL-divergence as cost function, we have to assume that distribution over trajectory is Gaussian in nature with a block diagonal covariance matrix. This is not always true for trajectories collected in many tasks. It also assumes that an expert is able to provide *near optimal* demonstrations in state and action space of agent. This is not possible in many application, for example, a manipulator arm with many degrees of freedom. Even if the expert is able to move the arm using a joystick, the demonstrations will not be optimal.

Therefore in order to deal with these difficulties, in the next chapter we investigate the inverse reinforcement learning method which learns the user's preferred reward function for the task instead of using a designed reward. Moreover, this method does not require an expert's demonstration in the agent's state space, thus making the user interface more natural for humans.

5

Active Preference Learning

In this chapter we present a segmentation-based Active preference learning framework for reward learning. We introduce a method that helps the robot to learn the reward function for a task in a sample efficient manner. We also present the theoretical aspects of the method and explain how it alleviate the temporal credit assignment problem.

Section 5.1 introduces the terminology used throughout the chapter and describes the problem we are trying to solve. Section 5.2 introduces the approach proposed by Sadigh *et al.* [SDSS17]. It explains the format of the query generated by the agent and the format of feedback given by the user. Section 5.3 describes the initial distribution over reward functions and explains how this distribution is updated from the feedback of the user. Section 5.4 describes how the current distribution over the reward function is used to generate queries. Section 5.5 gives limitation of current method and explains how temporal segmentation can be used to overcome those limitation. Section 5.7 describes method we use for segmenting the trajectory and formate of query after segmentation. Section 5.9 and 5.10 describes how the method of weight distribution is adapted to for feedback over segmented trajectories.

5.1 Problem Statement

The goal of this chapter is to learn the reward function that matches expert's preference, in a sample efficient manner. Finding an optimal policy from this reward function is done

using the same method as described in section 4.4. For the scope of this chapter we focus on only learning the reward function. We first describe in detail the Active Preference Learning algorithm as described in [SDSS17], we then present our adaption of the method, which is described in sections 5.6 - 5.10. The same notation as chapter 4 is used throughout this chapter. A trajectory is generated when the agent interacts with the environment and is denoted by τ. τ is a finite horizon sequence of state and action pairs given by $\tau = \{(x^0, u^0), (x^1, u^1), ...(x^H, u^H)\}$ where H is the horizon. The set of trajectories that satisfies the differential constraints of the system is called the Feasible Set and is denoted by Ω. Figure 5.1 shows Ω visually. Note that trajectories in Ω satisfy the differential constraints of the system but may not be the safe trajectories. For example, crashing the car is physically plausible but not safe. The state of the system and of other agents in the environment are converted to some global features and the mapping function is denoted by $\phi(x) \in \mathbb{R}^l$. This function is used to convert the raw sensor data of robot, for example, LIDAR and camera data, to meaningful global features, such as the distance from the border of the lane etc.

The reward function being learned is assumed to be a linear combination of these global features. Thus the reward function of the system at a particular time t can be written as

$$r(x^t, u^t) = \mathbf{w}^T \phi(x^t, u^t), \tag{5.1}$$

where $\mathbf{w}$ is vector of weights for feature function $\phi(x^t, u^t)$ evaluated at every time step t. Therefore for trajectory τ with finite horizon H, the reward function R can be given by

$$R(\tau) = \sum_{t=0}^{t=H} r(x^t, u^t). \tag{5.2}$$

For simplification of notation we factor the $H + 1$ elements of ϕ such that $\Phi = \sum_{t=0}^{t=H} \phi(x^t, u^t)$. Therefore reward function in equation 5.2 can be written as,

$$R(\tau) = \mathbf{w}^T \Phi(\tau). \tag{5.3}$$

The goal of preference learning is to learn weight vector $\mathbf{w}$ for linear reward R.

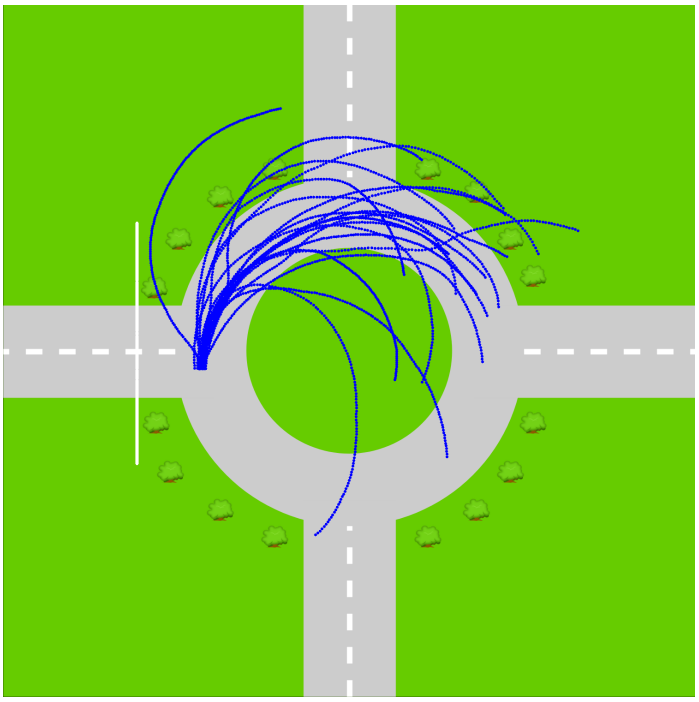


Figure 5.1: Example Feasibility Set for Round-a-bout Scenario

5.2 Learning Reward from Preferences

Unlike imitation learning where training data is given as set of demonstrations, the idea of preference learning is to provide training data in the form of pair-wise comparison between two behaviors. For a particular scenario, the agent iteratively generates two trajectories and presents them to the expert who then selects the one trajectory that they prefer. The response is assumed to be *strictly preferred*, i.e. one trajectory is strictly better than the other, for example $(\tau_A > \tau_B)$ (see figure 3.7). This response provides a preference learning method with information about the reward function for that particular scenario. Active learning deals with generating these queries such that the information required for approximating the unknown function is maximized, according to some measure. In this chapter, we are trying to approximate a probability distribution over the weight vector $\mathbf{w}$. In the following section, we describe how this distribution over the reward function's weight vector is updated based upon the expert's preference response and explain the information measure that is used to generate the next query.

5.3 Distribution Update Based on Feedback

As mentioned in the previous section, we are trying to approximate the distribution over weight vectors **w**. This can also be seen as having various hypotheses over weight vectors **w** and we want a mechanism that help us shrink the remaining version space (the set of consistent hypotheses) as quickly as possible. Golovin *et al.* showed that the reduction in version space probability mass is *adaptive sub-modular* (see section 2.7) and can have a adaptive greedy algorithm that is near optimal querying policy [GK10]. This section shows how the version space shrinks given the response of the query.

We start by having a uniform distribution over the space of **w**. Since the scale of **w** does not change the preference, we can constraint $||\mathbf{w}|| < 1$. Therefore, the distribution over **w** lies in the interior of a *unit hyper-sphere* with coefficients w of feature function ϕ as its axes.

This section assumes that the algorithm is at iteration i and has already synthesized two trajectories, τ_A, τ_B to query from. As shown in figure 3.7, the two trajectories are presented to the expert. She then gives her preference over these trajectories. If the expert's response is $\tau_A > \tau_B$, the input is recorded as +1 and if it is $\tau_B > \tau_A$ then input is -1. This response gives us information about w. User is more likely to say +1 if total reward of τ_A is greater than that of τ_B.

The distribution over **w** is updated using a Bayesian update rule given the preference as evidence. Therefore, the new distribution over **w** can be given by,

$$p(\mathbf{w}|I_t) \propto p(\mathbf{w})p(I_t|\mathbf{w}), \tag{5.4}$$

where, $p(\mathbf{w})$ is prior distribution and $p(I_t|\mathbf{w})$ is likelihood function.

The expert's preferences are not assumed to be perfect. Therefore, to deal with the noise associated with the response, $p(I|\mathbf{w})$ is modeled as a noisy preference with respect to the reward function, R, given by the following equation:

$$p(I_t|\mathbf{w}) = \begin{cases} \frac{exp(R(\tau_A))}{exp(R(\tau_A))+exp(R(\tau_B))} & I_t = +1 \\ \frac{exp(R(\tau_B))}{exp(R(\tau_A))+exp(R(\tau_B))} & I_t = -1. \end{cases} \tag{5.5}$$

Let φ be the difference between the trajectories in feature space given by,

$$\varphi = \Phi(\tau_A) - \Phi(\tau_B). \tag{5.6}$$

Then, the update function (likelihood) $p(I_t|\mathbf{w}) = f_\varphi$ for the Bayesian update is given by,

$$f_\varphi(\mathbf{w}) = \frac{1}{1 + exp(-I_t \mathbf{w}^T \varphi)}. \tag{5.7}$$

Thus, every Bayesian update can be seen as removing the undesired volume of the hyper-sphere that represents the distribution of **w**. Therefore, we want our query to be such that the maximum volume is removed. This is our information measure for generating the next query.

5.4 Generating Queries

This section describes how the two trajectories are synthesized for the query. In the previous section, we saw how the weight distribution for **w** is updated at iteration i. A good query will help in maximal undesired volume removal from the distribution. This fact is used to formulate query generation as a constraint optimization problem. Trajectories are synthesized by maximizing the volume removed under the feasibility constraints of φ given as follows:

$$\begin{aligned} \max_{\varphi} \min\{\mathbb{E}[1 - f_\varphi(\mathbf{w})], \mathbb{E}[1 - f_{-\varphi}(\mathbf{w})]\} \\ \text{subject to } \varphi \in \text{F}. \end{aligned} \tag{5.8}$$

The constraint in this optimization problem requires φ to be in the feasible set F where,

$$\mathrm{F} = \{\varphi : \varphi = \Phi(\tau_A) - \Phi(\tau_B), \tau_A, \tau_B \in \Omega\}. \tag{5.9}$$

As mentioned in section 5.1, Ω is the set of trajectories which satisfy the differential constraints of the system. Therefore F is the set of differences of features over all feasible trajectory pairs (τ_A, τ_B) such that $\tau_A, \tau_B \in \Omega$.

The above optimization given in equation 5.8 maximizes the minimum difference between the two spaces preferred by either choice of the user. Each term in the minimum function represents a volume removed based on the input. It is very difficult to enforce feasibility constraints in the global feature difference space of the environment. Therefore the feasibility of trajectories are enforced by directly optimizing over the states and actions of the agent. Therefore, the optimization equation can be given by:

$$\max_{x^0, u} \min\{\mathbb{E}[1 - f_\varphi(\mathbf{w})], \mathbb{E}[1 - f_{-\varphi}(\mathbf{w})]\}, \tag{5.10}$$

where φ is a function of x^0, u. This optimization is solved using a Quasi-Newton method (L-BFGS) as described in [Gal11]. The volume removed is given by $\mathbb{E}[1-f_\varphi(\mathbf{w})], \mathbb{E}[1-f_{-\varphi}(\mathbf{w})]$ depending on user's input.

The expectation in equation 5.10 is taken with respect to the distribution over $\mathbf{w}$. However $p(\mathbf{w})$ can be very complex thus making it difficult to compute the volume removed by the Bayesian update and to differentiate through it. Therefore, a sampling method is used as a surrogate to approximate the full objective. Subsequently, this sampled estimate is optimized.

Assume that we draw M independent samples from the distribution $p(\mathbf{w})$. Then the distribution $p(\mathbf{w})$ can be approximated using an empirical distribution consisting these M samples. Therefore, the volume removed by the update $f_\varphi(\mathbf{w})$ can be approximated by,

$$\mathbb{E}[1 - f_\varphi(\mathbf{w})] \approx \frac{1}{M} \sum_{i=1}^{M} (1 - f_\varphi(\mathbf{w}_i)). \tag{5.11}$$

Thus the objective is now differentiable with respect to φ which is differentiable with

respect to the starting state and controls.

Since distribution $p(\mathbf{w})$ is very complex to sample from, for sampling its assumed to be log-concave function. The update function is also log-concave, therefore posterior distribution remains log concave. Due to log-concativity assumption it is possible to use efficient polynomial time algorithms for sampling. Hence, the samples are obtained using Metropolis Markov Chain methods.

5.5 Limitation and Assumption

As in most reinforcement learning problems with sparse and delayed reward, one of the main difficulties with this method is *Temporal credit assignment problem*. That is, while the user reports their preference over the complete trajectory, the algorithm does not get information about which states and actions or sequence of states and actions were responsible for the user's choice of one trajectory over the other. Consider the case given in figure 5.2 for example. Even though trajectory B is bad for most of its duration because the car is too close to the border and trajectory A is good for the most part, our user might still choose $\tau_B > \tau_A$ because of the safer merging maneuver in τ_B highlighted by a red circle. Therefore, the segment of the trajectory within the red circle is the most salient segment for the reward function, such information is missed in the current preference learning framework. We will continue by describing a solution based on trajectory segmentation during preference elicitation.

5.6 Our Approach to Preference Learning

In order to alleviate the temporal credit assignment problem explained in the previous section, we propose a framework that asks users for their preference over segments of trajectories rather than complete trajectories. Consider the same case as in the previous section. If the trajectories are segmented based on change in heading, the segments will look something like given in figure 5.3.

Thus, in our framework, our user can choose $\tau_A > \tau_B$ for segments highlighted in cyan and yellow whereas $\tau_B > \tau_A$ for segment in magenta. Thus response provided per-segment

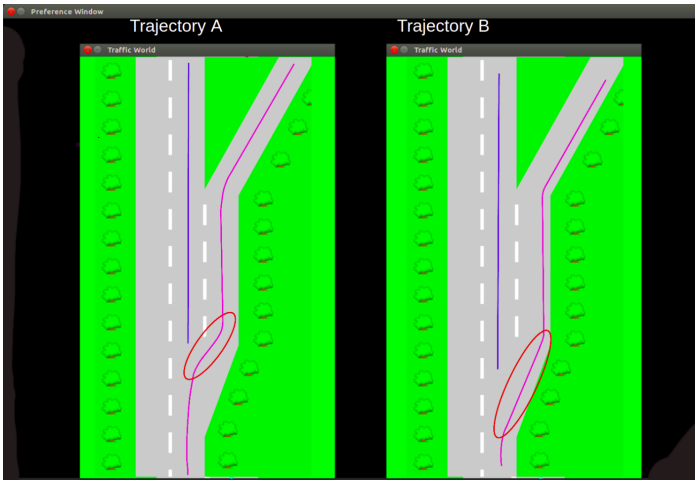


Figure 5.2: Preference interface showing two different Merging trajectories

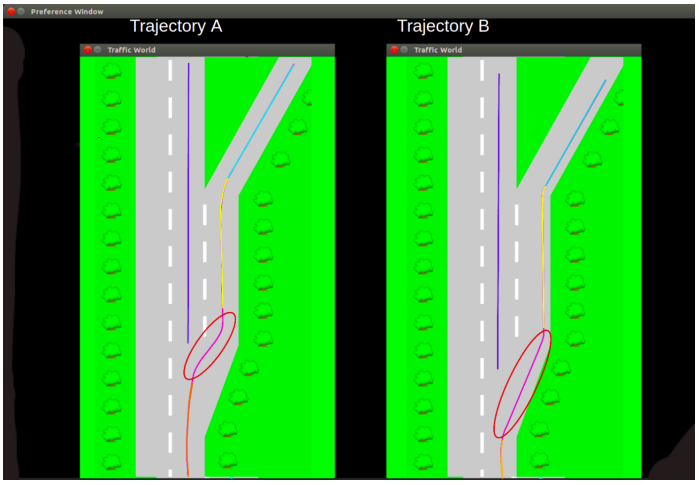


Figure 5.3: Preference interface showing two different Merging trajectories with segmentation

are more informative to the preference learning update. This helps in reducing no of times the optimization process is run to generate queries and the method converges to the reward function with fewer iterations.

As we can see, even though this method has the advantage of requiring fewer iterations, the amount of input feedback that our user must give is increased. In order to deal with this issue, we further evaluate each segment pair based on expected information gain (change in reward function if a certain segment was queried) to update the reward function and ask user for preference only over the segment with most information gain. Intuitively, in case of figure 5.3 that segment would be one highlighted with red circle.

5.7 Temporal Segmentation of Trajectories

In order to gain the most advantage out of this method the segments generated within the trajectory should be *meaningful*. One key goal of the preference learning algorithm is to determine the user's intention and find the policy that is most consistent with it. Therefore it is important to segment the trajectory in way that is naturally understood by the user. We adopt the idea of an *attribute function* and *criterion* described in [Mai11] to segment the trajectory. An *attribute function* specifies a value at every point in time where the trajectory is defined. For example, attributes can be speed, heading, and curvature etc. A *criterion* specifies a rule for judging when to split a segment, so that the attribute values at all points within the segments are sufficiently similar. For example a segment where the attribute value changes more than a certain threshold would be split. An example of segmentation can be seen in figure 5.3 where the attribute function is the heading of the robot and criterion is $\nabla\theta > \epsilon$.

5.8 Query and Response With Segmented Trajectories

The process of generating the two trajectories for a query using the Quasi Newton method for optimization remains the same as described in section 5.4. The trajectories are then segmented using the method described in previous section (5.7). The response of the user is collected over all the segments. If, for a certain segment k, the user prefers $\tau_A^k > \tau_B^k$ the response is collected as $+1$ while if for certain segment k user prefer $\tau_A^k < \tau_B^k$ the response

is collected as -1. Where, τ_A^k, τ_B^k are k^{th} segment in trajectory A and B respectively. Thus, if the algorithm outputs total K number of segments, the input feedback signal will be K-bit vector of $+1$ and -1.

5.9 Weight Distribution Updates With Segmented Trajectories

Our approach uses *Multiple evidence* Bayesian update to update the weight distribution over the reward function. For K segments the update equation can be given by,

$$p(\mathbf{w}|I_t^1, ...I_t^K) \propto p(\mathbf{w}) \prod_{k=1}^{K} p(I_t^k|\mathbf{w}), \tag{5.12}$$

where, I_t^k is user's response for k^{th} segment of the trajectory. Since preference over one segment is independent of other segments, above equation 5.12 assumes conditional independence.

5.10 Smart Segment Query and Distribution Update

The method described in section 5.9 helps in reducing the number of iterations for query generation but increases the need of feedback from the user. In order to deal with this, we further process the segments obtained from queried trajectories based on their information score. We calculate information gain between $\mathbf{w}$ and φ in order to evaluate which segment gives us the most information about the reward function and ask for our user's preference only on the segment with the most information gain. This reduces number of iterations for optimization and keeps the amount of input the same as the previous method. The weight distribution update of this method is the same as described in section 5.3.

Results for our methods, their comparison with the previous method and metrics of comparison are described in detail in the next chapter.

6

Experiments and Results

6.1 Results

We evaluate our method using the different autonomous scenarios mentioned in chapter 3. To evaluate and compare our method with the previously proposed method we use the true hidden reward weights $\mathbf{w}_{true}$ and at every Bayesian update we calculate the *goodness* of the learned reward using following equation;

$$m = \mathbb{E}[\frac{\mathbf{w}.\mathbf{w}_{true}}{|\mathbf{w}||\mathbf{w}_{true}|}], \tag{6.1}$$

where, m computes the average heading of the current distribution of $\mathbf{w}$ with respect to $\mathbf{w}_{true}$. Since the prior distribution of w is symmetric, this expectation starts at 0, and moves closer to 1 at every step of the iteration.

6.2 Over Taking in Two-Lanes

The raw-features use to represent the state for two-lane overtaking scenarios are, x, y position of the agent, heading, speed and y position of NPC given by tuple, $[x, y, \theta, v, y_{npc}]$. These are converted to global features where feature 1 penalizes the agent if it gets too close to the border given by, $f_1 \propto c_1 . \exp(-c_2 . d^2)$, where d is distance from the border and c_1, c_2 are appropriate scaling factors. A similar feature is used to keep the agent in the center of the lane. Higher speed is encourage by $f_3 = (v - v_{max})^2$, where v is the velocity of

the agent and v_max is its speed limit. Feature four encourages the agent's heading along the road, given by, $f_4 = \theta_{agent} \cdot \overrightarrow{n}$, where $\overrightarrow{n}$ is normal vector along the road. Feature 5 corresponds to collision avoidance given by nonspherical Gaussian over the distance of agent and NPC, whose major axis is along the robot's heading. Actions to the agent are the continuous value of acceleration and steering angle.

6.2.1 Comparison of Weight Distribution Update

Figure 6.1 shows $p(\mathbf{w})$ approximated with samples at different iterations. We can see that the distribution which is updated using segmentation converges quicker than the distribution updated without segmentation.

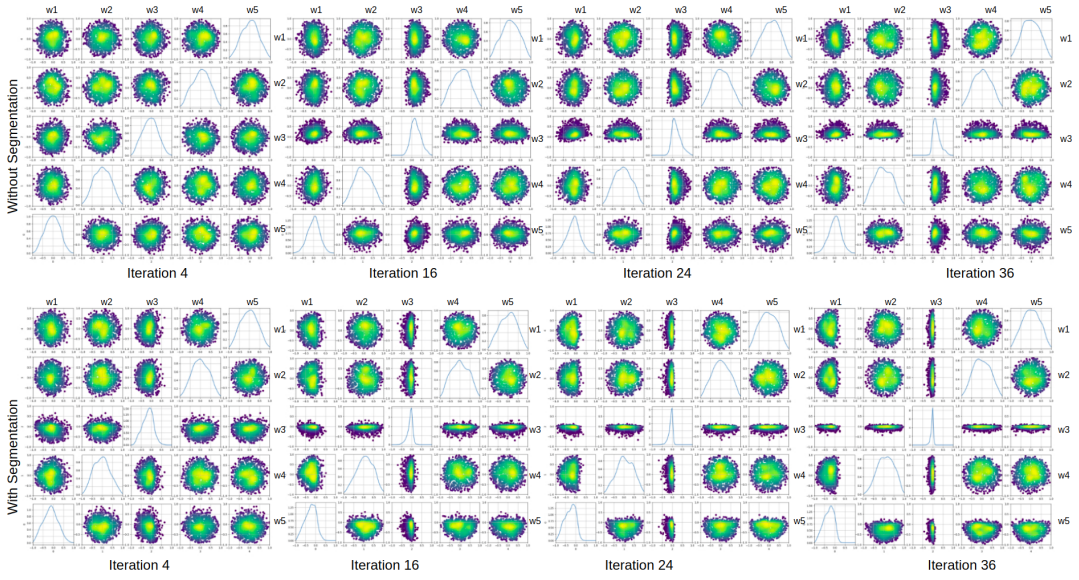


Figure 6.1: Update of weight distribution **w** for Overtaking Scenario

Row 1 shows the distribution update for the reward function without Trajectory Segmentation. Row 2 shows the distribution update for the reward function using Trajectory Segmentation.

6.2.2 Comparison of Goodness Metric

Figure 6.2 shows a comparison of three methods using the metric described in section 6.1. We can see that both methods with segmentation (shown in green and blue) get closer to true reward function in fewer iterations than the method without segmentation (shown in red).

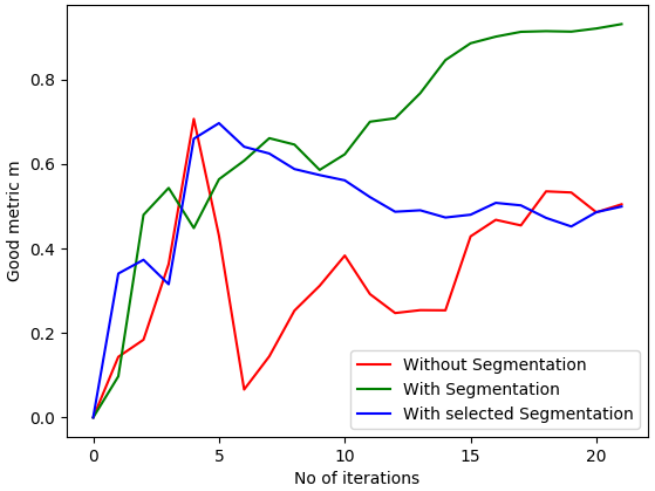


Figure 6.2: Goodness Metric comparison for Overtaking Scenario

The goodness of learned reward for: (red) original method without Trajectory Segmentation; (green) our method with Trajectory Segmentation; and (blue) our method with Selective Segmentation Queries.

6.2.3 Policy Learning From Reward Function

A policy can be developed to optimize the reward function learned using the method in this chapter. This policy optimization is done using the same method as described in 4.4. Figure 6.3 shows trajectory roll-outs from Ideal Expert's Reward (H = 148). The agent will try to roughly match these roll-outs by learning a reward function.

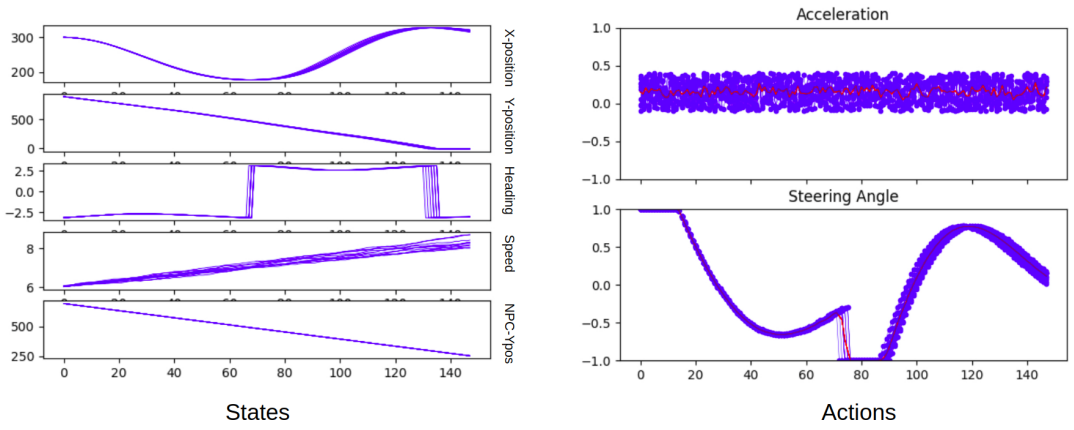


Figure 6.3: Overtaking Experts State Action Roll-outs

In figure 6.4 we can see that when an initial policy with random weights (iteration 0) is run on the environment, the behavior of the policy (trajectory in red) is far off from expert's behavior (trajectory in green). As the iterations proceed trajectory roll out of policy gets closer to expert's roll-out. Similarly, we can see that at with each iteration actions output by policy gets close to expert's actions.

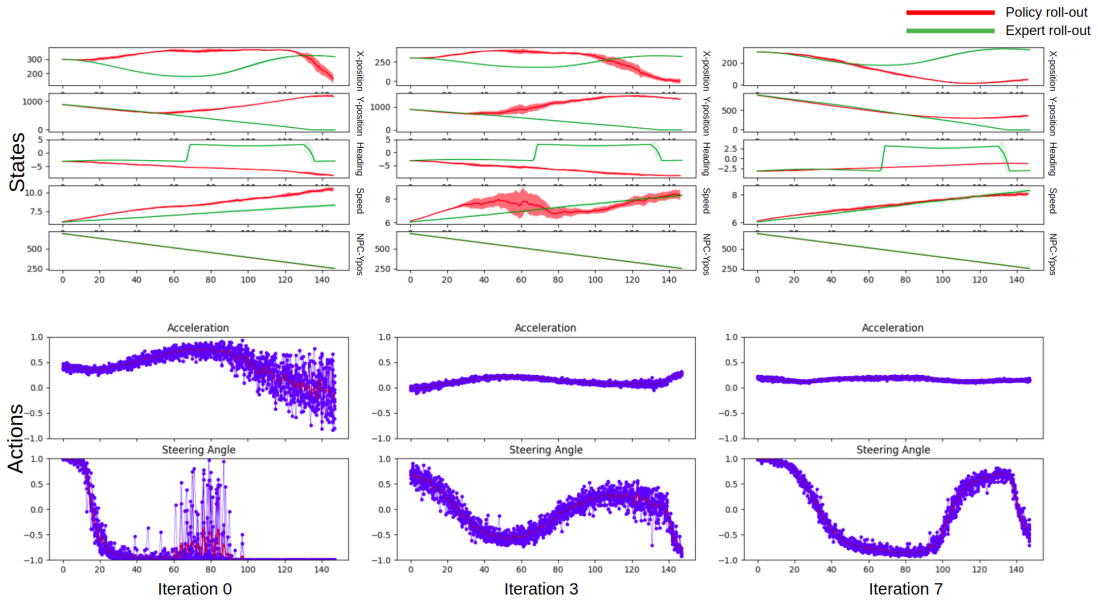


Figure 6.4: Overtaking Policy State Action Roll-outs

Trajectory Roll-outs from Learned Reward Function (H = 148).

6.3 Driving in Round-a-bout

The raw-features to represent the state for round-a-bout scenarios are, x, y position of the agent, heading, speed and x and y position of NPC given by tuple, $[x, y, \theta, v, x_{npc}, y_{npc}]$. These are converted to global features where feature 1 penalizes the agent if it gets too close to the border given by, $f_1 \propto c_1 . \exp(-c_2 . d^2)$, where d is distance from the border and c_1, c_2 are appropriate scaling factors. Distance is calculated using assuming that the car is moving in perfect circle with certain radius and at certain point a tangent can be drawn from its position. Similar feature is used to keep agent in the center of the lane. Higher speed is encourage by $f_3 = (v - v_{max})^2$, where v is velocity of agent and v_{max} is speed limit. Feature four encourages agent's heading along the road, given by, $f_4 = \theta_{agent} . \overrightarrow{n}$, where $\overrightarrow{n}$ is normal vector along the road. Feature 5 corresponds to collision avoidance given by non spherical Gaussian over the distance of agent and NPC, whose major axis is along the robot's heading. Actions to the agent are the continuous value of acceleration and steering angle.

Similar analysis is done for this scenario as in section 6.2 where figure 6.5 shows the distribution update for our method with segmentation is faster than the existing method that did not use segmentation. Figure 6.6 shows comparison of the goodness metric for the three methods. We can see that results are similar between methods in this case. This is possible due to two reasons. 1) Because the segments do not provide any more information about the choice made by the user and 2) the attribute function is chosen such that segments are not created. For example, if the change in heading is chosen as attribution function for this scenario, in round-a-bout the steering is ideally kept constant. Therefore the output of the segmentation will still be a complete trajectory. Even in this less ideal experiment, we can see that our approach either out performs the previous method or at least matches the proposed method, but does not underperform.

6.3.1 Comparison of Weight Distribution Update

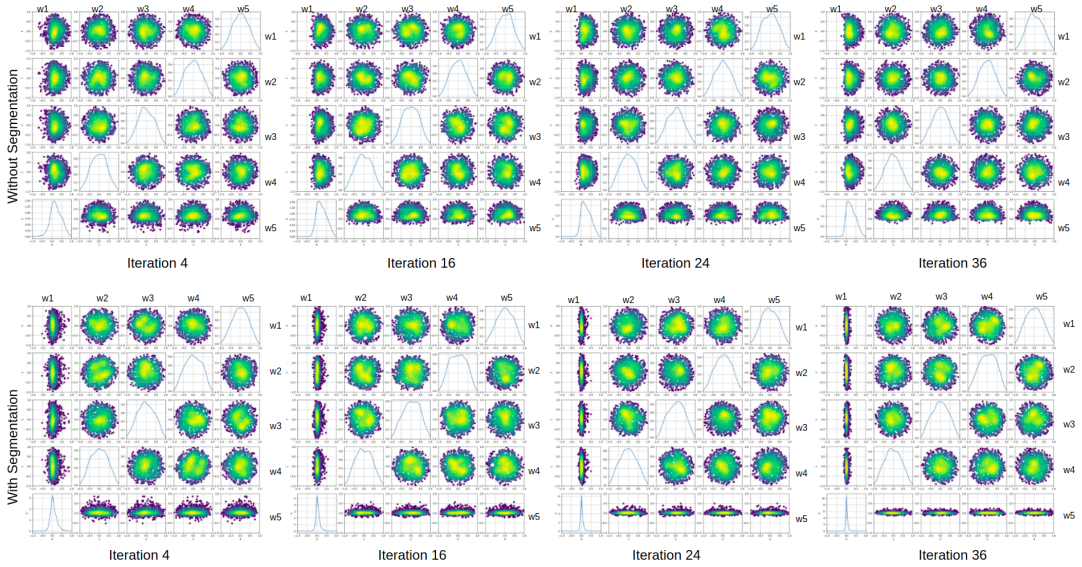


Figure 6.5: Update of weight distribution **w** for Round-a-bout Scenario

Row 1 shows the distribution update for the reward function without Trajectory Segmentation. Row 2 shows the distribution update for the reward function With Trajectory Segmentation

6.3.2 Comparison of Goodness Metric

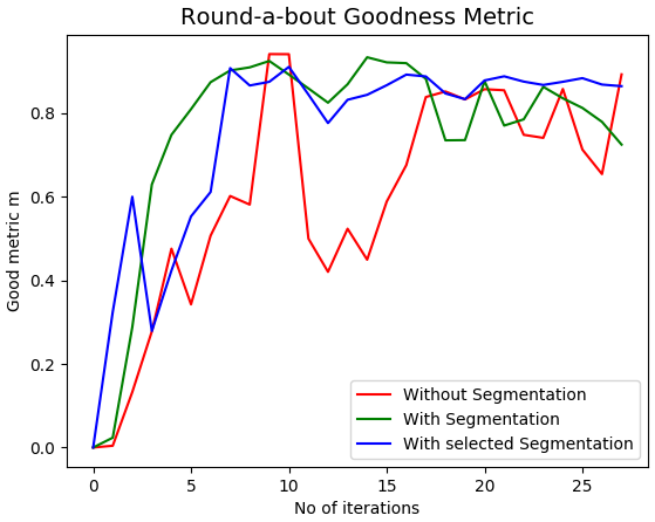


Figure 6.6: Goodness Metric comparison for Round-a-bout Scenario

Goodness metric curves over the learning process for learning: (red) without Trajectory Segmentation; (green) with Trajectory Segmentation; and (blue) with Selective Segmentation Queries.

Part III

Final Conclusion & Future Work

7

Final Conclusion & Future Work

In this thesis, we made two contributions. 1) We presented an improvement on probabilistic model-based imitation learning approach to make it scalable for longer and more complex demonstrations. The main component of this contribution was using a Bayesian Neural network for probabilistic modeling of the dynamic environment and policy roll-outs. Thus the method was better able to handle the uncertainty of the environment. 2) We presented an improvement on active preference learning by introducing a trajectory segmentation framework. This helped in alleviating the temporal credit assignment problem. We proposed two approaches to address the credit assignment using trajectory segmentation. The first approach asked for preference over all the segments in a trajectory. This helped in reducing the number of iterations for which query was made to the user but increased the user feedback inputs (as 3 inputs were needed each iteration). Therefore, in the second approach, we calculated which segment contained the most information about the reward function and asked for preference selectively, over only that segment. The experiments have shown that both our approaches show improvement on the previous method by reducing the number of iterations required to converge to the true reward. The comparison of all three approaches was done by using the goodness function mention in section 6.1.

7.1 Limitations and Future Work

Our framework of trajectory segmentation for active preference learning is currently limited in many ways. 1) While segmenting a trajectory, the point of segmentation is the same

in both the trajectories which are generated for querying. In this case, we choose the trajectory with the most segments and segment both the trajectories using those points. Currently, the segmentation is only helping in updating the reward distribution but is not helping in the optimization process. The framework can be further improved and can be made more general by working on these two points. Furthermore, the current framework considers only linear rewards, which are a function of some global features. For many task, linear reward functions might not work. Moreover, the features are designed by the programmer and not learned by the algorithm. The method can be made adaptive for different and more complex tasks by incorporating these improvements.

Bibliography

[Aar12] Aaron Wilson and Alan Fern and Prasad Tadepalli. *A Bayesian Approach for Policy Learning from Trajectory Preference Queries*. Curran Associates, Inc., 2012.

[AM10] Nir Ailon and Mehryar Mohri. Preference-based Learning to Rank. *Machine Learning*, 80(2-3):189–211, 2010.

[AN04] Pieter Abbeel and Andrew Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. *Proceedings of the Twenty-first International Conference on Machine Learning*, 2004.

[ASS12] Riad Akrour, Marc Schoenauer, and Michèle Sebag. APRIL: Active Preference Learning-based Reinforcement Learning. pages 116–131, 2012.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs.LG]*, 2016.

[Bel57] Richard Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

[BR63] Albert Bandura and Walters R.H. *Social learning and personality development*. Holt Rinehart and Winston, New York, 1963.

[BS95] Michael Bain and Claude Sammut. A Framework for Behavioural Cloning. *Machine Intelligence 15*, 1995.

[BTD+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller,

Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. *CoRR*, abs/1604.07316, 2016.

[Bus14] Busa-Fekete, Róbert and Szörényi, Balázs and Weng, Paul and Cheng, Weiwei and Hüllermeier, Eyke. Preference-based reinforcement learning: evolutionary direct policy search using a preference-based racing algorithm. *Machine Learning*, 97(3):327–351, 2014.

[BYH+17] Chandrayee Basu, Qian Yang, David Hungerman, Anca Dragan, and Mukesh Singhal. Do you want your autonomous car to drive like you? . *12th ACM/IEEE International Conference on Human-Robot Interaction (HRI).*, 2017.

[CN17] Yuchen Cui and Scott Niekum. Active learning from critiques via bayesian inverse reinforcement learning. 2017.

[Dav15] Alex Davies. Oh look, more evidence humans shouldn't be driving. *Online, https://www.wired.com/2015/05/oh-look-evidence-humans-shouldnt-driving/*, 2015.

[DR11] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. *Proceedings of the 28 th International Conference on Machine Learning*, 2011.

[EPPD13] Peter Englert, Alexandros Paraschos, Jan Peters, and Marc P Deisenroth. Model-based Imitation Learning by Probabilistic Trajectory Matching. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013.

[FR14] Pier Francesco Ferrari and Giacomo Rizzolatti. Mirror neuron research: the past and the future. *Phil. Trans. R. Soc. B*, page 369, 2014.

[FR15] Pier Francesco Ferrari and Giacomo Rizzolatti. *New Frontiers in Mirror Neurons Research*. Oxford University Press, 2015.

BIBLIOGRAPHY

[Gal11] Galen Andrew and Jianfeng Gao. Scalable training of l 1-regularized log-linear models. *In Proceedings of the 24th international conference on Machine learning*, pages 33–40, 2011.

[GB10] Andrew Guillory and Jeff Bilmes. Interactive submodular set cover. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 415–422, 2010.

[GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation:Representing Model Uncertainty in Deep Learning. *Proceedings of the 33rd International Conference on Machine Learning*, 48, 2016.

[GK10] Daniel Golovin and Andreas Krause. Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization. *CoRR*, abs/1003.3967, 2010.

[Gui12] Andrew Guillory. *Active Learning and Submodular Functions*. PhD thesis, 2012.

[HGE16] Jonathan Ho, Jayesh K. Gupta, and Stefano Ermon. Model-Free Imitation Learning with Policy Optimization. *CoRR*, abs/1605.08478, 2016.

[HMD18] Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek. Synthesizing Neural Network Controllers with Probabilistic Model based Reinforcement Learning. *International Conference on Intelligent Robots and Systems*, 2018.

[Kre98] David Kreps. *Notes On The Theory Of Choice*. Routledge, 1998.

[Kul59] Solomon Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.

[LHP+13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

[Mai11] Maike Buchin and Anne Driemel and Marc van Kreveld and Vera Sacristan. Segmenting trajectories: A framework and algorithms using spatio-temporal criteria. *JOURNAL OF SPATIAL INFORMATION SCIENCE*, 3:33–63, 2011.

BIBLIOGRAPHY

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with Deep Reinforcement Learning. *ArXiv*, abs/1312.5602, 2013.

[MP02] Andrew Meltzoff and Wolfgang Prinz. The Imitative Mind: Development, Evolution, and Brain Bases. 01 2002.

[Ols17] R.M. Olsthoorn. *Segmented Active Reward Learning*. PhD thesis, 2017.

[Org15] W H Organization. Global Status Report on Road Safety 2015. *World Health Organization*, 2015.

[RA07] Deepak Ramachandran and Eyal Amir. Bayesian Inverse Reinforcement Learning. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2586–2591, 2007.

[RBG11] Stephane Ross, Drew Bagnell, and Geoffrey J Gordon. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *14th International Conference on Artificial Intelligence and Statistics* , 15, 2011.

[RBG18] Stephane Ross, Drew Bagnell, and Geoffrey J Gordon. Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2005.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[SBTC16] Halit Bener Suay, Tim Brys, Matthew E. Taylor, and Sonia Chernova. Learning from Demonstration for Shaping Through Inverse Reinforcement Learning. *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, pages 429–437, 2016.

[Sch97] Stefan Schaal. Learning From Demonstration. *Advances in Neural Information Processing Systems*, 9, 1997.

[SDSS17] Dorsa Sadigh, Anca Dragan, Shankar Sastry, and Sanjit Seshia. Active Preference-Based Learning of Reward Functions. *Robotics Science and Systems*, 2017.

[Sut84] Richard Stuart Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, 1984. AAI8410337.

[Van08] Tom Vanderbilt. *Traffic: Why we drive the way we do (and what it says about us)*. Alfred a. knopf, New York, 2008.

[WF12] Christian Wirth and Johannes Fürnkranz. First Steps Towards Learning from Game Annotations. *Proceedings of the ECAI Workshop on Preference Learning: Problems and Applications in AI*, pages 53–58, 2012.

[WF13] Christian Wirth and Johannes Fürnkranz. A Policy Iteration Algorithm for Learning from Preference-Based Feedback. pages 427–437, 2013.

[ZMBD08] Brian D Ziebart, Andrew Maas, Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, pages 1433–1438, 2008.

Acronyms

NPC Non Player Character

RL Reinforcement Learning

IRL Inverse Reinforcement Learning

BNN Bayesian Neural Network

NN Neural Network

GP Gaussian Process

BPPT Back Propagation Through Time

DQN Deep Q-Learning

DDPG Deep Deterministic Policy Gradients