Uncertainty Aware Behavioral Cloning using Bayesian Neural Networks

Sanjay Thakur

School of Computer Science
McGill University
Montreal, Canada

August 2019

A thesis submitted to McGill University in partial fulfillment of the requirements for the Master of Science, Computer Science.

2019/08/06

Abstract

Leveraging demonstrations to learn complex maneuvers is a handy technique to teach robots when other techniques like reinforcement learning are difficult to implement. However, the world is diverse and partially observable, and generating demonstrations under all possible situations in hard. As a result, a lot of times the robot's decision on the situation it has not seen earlier is wrong.

One line of potential solutions to handle this problem of incorrect decision-making at previously unseen situations begins with equipping the robots with a sense of confidence in their decisions. In this work, we develop one such technique using a data-driven probabilistic method.

Specifically, we use a Bayesian Neural Network to generate a cheap to obtain quantity for the robot to gauge its confidence of doing well in a given situation. These different situations are more relatable to the real world in the sense that the differences between them are unobservable. We show the scalability and consistency of our approach to such situations in high dimensional simulated and real robotic domains. Also, we show that such a confidence based solution allows making an informed decision about when to invoke a fallback strategy. One fallback strategy is to request more data. We empirically show that providing data only when requested results in increased data-efficiency. This is crucial in the real-world as data is expensive and painstaking to obtain.

Le Résumé

L'utilisation de démonstrations pour apprendre des manoeuvres complexes est une technique pratique pour enseigner aux robots lorsque d'autres techniques telles que l'apprentissage par renforcement sont difficiles à mettre en oeuvre. Cependant, le monde est divers et partiellement observable, et générer des démonstrations dans toutes les situations possibles est dur. Par conséquent, la décision du robot sur la situation qu'il n'a pas vue est souvent erronée.

Une ligne de solutions potentielles pour adresser ce problème de prise de décision incorrecte dans des situations observées pour la première fois commence par donner aux robots une notion de confiance dans leurs décisions. Dans ce travail, nous développons une telle technique en utilisant une méthode probabiliste basée sur les données.

Plus précisément, nous utilisons un réseau de neurones bayésien pour générer une quantité peu coûteuse à obtenir pour que le robot puisse évaluer sa confiance afin de bien réussir dans une situation donnée. Ces différentes situations sont davantage liées au monde réel en ce sens que les différences entre elles ne sont pas observables. Nous montrons l'évolutivité et la cohérence de notre approche sur de telles situations dans des domaines robotiques simulés et réels de grande dimension. En outre, nous montrons qu'une telle solution basée sur la confiance permet de prendre une décision éclairée en temps réel, afin d'ínvoquer potentiellement une stratégie de repli. Une stratégie de repli consiste à demander plus de données. Nous montrons de manière empirique que l'obtention de données uniquement à la demande entraîne une efficacité accrue des données. Ceci est crucial dans le monde réel car les données sont chères et difficilesà obtenir dans le monde réel.

Acknowledgments

I would first like to thank my first co-supervisor Dr. Doina Precup for allowing me to be part of the RLLab and for her constant support throughout during the duration of my thesis.

I would deeply thank my second co-supervisor Dr. David Meger for allowing me the opportunity to work in the Mobile Robotics Lab and for his constant guidance, and feedback on my work. I am also grateful to him for the amount of time he had taken out of his schedule weekly and ease me through the challenging process of doing a fruitful and constructive work. Dr. David Meger has also helped me become a better writer through his feedback on my writing.

My deepest gratitude goes to Dr. Herke Van Hoof who has always been there as a friend, mentor, collaborator all round the clock and who have found ways to help me every single time I needed it even if it meant from a different continent. He helped me tremendously to become a better researcher and a better writer, and provided constant inspiration, feedback and support, as well as a wealth of intriguing ideas.

I would also thank my colleague Juan Camilo Gamboa Higuera for passionate participation, and validation of my ideas. I had a wonderful time working with him on the furuta pendulum swing up that he had made from scratch.

Many thanks to all other colleagues at RLLab, Mila, and MRL who have created such a conducive environment to do effective research with their regular reading groups and brainstorming sessions. I especially want to thank my colleagues Kushal, Sumana, Charles, Fengdi, and Jingyun who had enhanced my joy of working in the lab.

Many thanks for Dr. Joelle Pineau for allowing me to participate in her weekly meetings during the beginning of my program which helped me stimulate my mind for a meaningful research.

Many thanks to my friends Romain, Emilie, Saatvik, Sumaira, Amulya, Manikant, Mangaljit, Ayush, and Avinash for their affection and being my constant source of support.

Finally, I would like to thank my parents, my lovely sister, Utsav Uncle, and Rekha Aunty for their numerous sacrifices, support, and unconditional love to help me reach where I am now.

Contributions

The work described in this thesis is a result of the combined assistance from Dr. Herke Van Hoof, Dr. David Meger, Dr. Doina Precup, and Juan Camilo Higuera.

Dr. Herke Van Hoof has contributed in almost all aspects of this work, like conceiving of the idea, developing the theory, planning the experiments, investigating different aspects of our research findings, correcting mistakes, writing the academic papers related to this work and reviewing the thesis document.

Dr. David Meger has played an important role in giving feedbacks on the idea, theory, and experiments, editing and reviewing the academic papers related to this work and this thesis itself.

Dr. Doina Precup has helped in giving feedbacks on the idea and theory, planning the course of my thesis and reviewing of the academic papers related to this work. She has also helped in funding my thesis.

Juan Camilo Gamboa Higuera has helped in setting up the robotic pendulum experiment and reviewing the academic papers related to this work.

Contents

List of Figures

List of Tables

List of Acronyms

LfD	Learning from Demonstration
MDP	Markov Decision Process
NN	Neural Network
BNN	Bayesian Neural Network
BBB	Bayes-by-Backprop
BC	Behavioral Cloning
MLE	Maximum Likelihood Estimate
MAP	Maximum A Posteriori
GP	Gaussian Process
DNN	Deterministic Neural Network

Chapter 1

Introduction

Being able to extract and leverage useful information from demonstrations is one of the hallmarks of highly evolved intelligence. Learning by observing how others do something is one of the primary ways by which humans learn social constructs, protocols and other pertinent behavioral aspects of life [1]. Hence being able to leverage demonstrations is a critical piece to building a more capable artificial intelligence.

1.1 The Dichotomy of Leveraging Demonstrations for Learning

Learning from Demonstrations (LfD) is an umbrella term used for any technique that leverages demonstrations to build machine intelligence. *Behavioral Cloning* (BC) in particular, is a technique for learning from demonstrations where a mapping is learnt directly from inputs or observations to their corresponding outputs or actions. It has been highly effective for learning decision-making architectures or *controllers* for complex and high-dimensional tasks. One of the success stories of BC in developing such complex controllers is self-driving cars [6, 7]. BC is handy where other techniques such as deep reinforcement learning and optimal control are challenging to implement or are highly prone to failure. This is because learning the right solution is relatively easier to attain using BC as compared to other frameworks such as reinforcement learning. Other benefits of BC and learning from demonstrations in general include opening up a convenient way of communicating an intent between a layman person and an AI system.

However, BC is not a silver bullet for all our AI problems. BC based methods can fail when there is a mismatch between the situations at deployment and situations at training.

We call these diverse situations as *contexts* in our work. Our state of the art learning architectures such as *Neural Network* (NN) is prone to fit to the data from seen contexts in a way that its prediction at previously unseen contexts could be wrongly estimated. This problem is more pronounced when demonstrations are non-exhaustive and the contexts are high-dimensional, extremely diverse in nature, and partially observable. For example, a self-driving car might transfer its knowledge positively to a dry road after training from demonstrations on a wet one but might fail on a slippery icy road. Hence, BC is marred with its own set of problems.

1.2 Contribution of the Work

One potential direction towards mitigating the problems that we describe above would be to have a cheap proxy that reflects the chances of success of a learner in any given situation. Such a proxy should also be easy be obtain and be integrated in a way that facilitates better decision-making abilities for our learner [8]. This proxy would be particularly invaluable in the failure prone contexts which are otherwise hard to know.

Previous works have shown that training on multiple diverse conditions helps [9] but demonstrations are expensive and modelling all possible situations is hard. A few works [10, 11] also show that conditioning on history to learn useful behavior ameliorates the situation but cannot adjust to drastic changes. In this work, I study and analyze integration of probabilistic methods with BC using *Bayesian Neural Networks* (BNN) under diverse high dimensional contexts where the demonstrations are close to optimal. What makes the problem more interesting is the fact that the differences between these contexts can be unobservable. In our self-driving car example given earlier, this unobservability would correspond to the fact that a self-driving car does not have any sensor to tell the difference in the friction coefficient of a dry road and an icy one. Specifically, I investigate whether a relation can be obtained between the predictive uncertainties of the BNN and its performance. The idea is to use the predictive uncertainties as a cheap proxy to gauge the learner success in a given context.

I further go on to develop a framework which can leverage the predictive uncertainties to appropriately time the invocation of a fallback strategy. Such an uncertainty aware learner has the ability to not only effectively make decisions when encountering diverse contexts as in the real-world but also opens up a gamut of possibilities of adopting reasonable fallback strategies in the light of high uncertainty. As an example of such a reasonable fallback strategy, we propose a learner that stops and seeks for context specific demonstrations. Our contributions in this work are

- to design a framework to generate predictive uncertainties over multiple contexts as a confidence measure in a BC setting scalably, and
- to propose an approach for actively requesting context-specific training data where the learner is not confident.

Note that our work is closely related to *Adaptive Control* and *System Identification* as our methodology adjusts its behavior (adaptive) depending on what it infers as the underlying situation (identification) as a latent variable. We first support our claims through a proof-of-concept experiment on a real *pendulum swing up robot*. We then do experiments on simulated environments like OpenAI Gym [12] supported *MuJoCo* tasks of *HalfCheetah* and *Swimmer* where the differences in contexts are unobservable. We first show that BNNs scale up easily to higher-dimensional tasks. We then show that there exists an inverse relation between the predictive uncertainty given by the BNN and the performance of the controller, so, the uncertainty can be used as a confidence to predict its success. Finally, as fallback strategy we propose requesting more demonstrations when the predictive uncertainty is too high. We show that seeking more context specific demonstrations only when the learner is highly uncertain on that particular context leads to data-efficient learning.

1.3 Thesis Outline

In the next chapter (chapter 2), we describe the background relevant to our work such as LfD and learning predictive uncertainties. We describe in detail the intricacies and significance of various methods in each of those categories while positioning similar methods to our work such as active learning from demonstrations and learning across multiple task variants.

We then highlight the scope of our work by formalizing our problem set-up and methodology of how we use the probabilistic methods to represent uncertainty while learning by BC in chapter 3. We describe how do we generate predictive uncertainties using BNN that scales naturally to the high-dimensional tasks and number of data-points where the underlying contexts can vary considerably in an unobservable way. We also describe our fallback strategy to leverage the obtained uncertainty in the context of active learning from demonstrations on multiple diverse contexts. We describe how we ground the confidence of the controller in a way that helps it to discern between the contexts that it can generalize to and the contexts it cannot generalize to satisfactorily.

We explain our experimental set-ups of a physical pendulum swing-up task, as well as several simulated robotics tasks in chapter 4. We explain their observation space, action space, the reward structure, and their respective demonstrators. We also explain briefly what makes these experiments interesting for the solutions to the problem that we care about in this work.

We explain the viability and benefits of our approach in chapter 5 where we show that our methodology can scale up organically, can successfully capture the success in a given context through the predictive uncertainty metric and yield a data-efficient active learning mechanism.

Finally, in chapter 6, we delineate the possible lines of future work based on what can be built on top of our approach and the avenues where our methodology might not be effective.

Chapter 2

Background

In this chapter, we describe the background knowledge needed to fully understand and estimate the scope of our work. We define methodologies and frameworks used in this work mean and explain their scope while also covering their broad categories, advantages and limitations. We also highlight methods that are built on these methodologies and are related to our work in a bid to position our work in the space of similar methods.

2.1 Learning from Demonstrations

Learning by imitation in human history has played a pivotal role in steering our evolution and continues to play a crucial role in shaping intelligent behavioral societies [1]. Children learn most of their daily life routines like cleaning, and cooking by observing their caregivers [13]. Hence the pursuit of a stronger AI involves, in part, equipping the AI agents with the ability to learn useful behavior by observing. Additionally, as more and more AI powered systems will permeate into our societies in the form of self-driving cars, care-giving robots, cobots etc, the need for devising ways for better learning by imitation will be appealing as it opens up the possibility for even a layman person to tweak their robot's behavior by demonstrating their intention.

Learning from Demonstrations (LfD) or *Imitation Learning* is an umbrella term used for any learning methodology that leverages demonstrations to learn any function of interest. The simple nature of many LfD techniques had prompted researchers for quite some time to leverage it for complicated tasks such as navigation avoidance through pedestrian trajectories [14], helicopter control [15], autonomous driving [16], one-shot visual transfer [17],

Fig. 2.1 A bulk of our behavior patterns are motivated by observing others (top figure) [1]. Hence in order to attain more capable AI systems, we have to endow them with the mechanism to learn from demonstrations (bottom figure, taken from [2]).

etc. In this section we will first develop perspective around why LfD is significant today, followed by formalizing LfD and different ways of approaching it, then talk about unique complications arising from the nature of LfD based techniques and finally talk briefly about getting around these unique LfD problems and extensions.

2.1.1 Why do we need LfD

Very broadly, LfD has three major advantages over other learning paradigms:

1. **Communication of intent is easy:** A major way of communicating intent in *control* applications is through reward or cost functions but these functions are hard to design. Given the fact that LfD techniques allow communication of intent easily, methods that leverage demonstrations seem more enticing to adopt.

2. **Finding a good control solution is hard:** Optimization is hard. Often, the converged solution is not the best possible achievable answer. However conveying the best solution through demonstrations is an easier task. This phenomenon even exists in the real world, as it had been observed with Fosbury Flop [18], see figure 2.2).

3. **Learning from scratch is time-consuming and dangerous:** Usually learning from scratch necessitates making an catastrophic error to know what was bad. This is unacceptable especially in the real world. However, demonstrations can be used to effectively communicate how to avoid such situations.

Fig. 2.2 Fosbury Flop: Until demonstrated by Dick Fosbury in 1968 Summer Olympics, all elite high jumpers used less effective techniques of high jump. This turn of events has remarkable similarity to finding the optimal policy for problems by current techniques versus by using demonstrations. Image is taken from Wikipedia.

2.1.2 Formalizing LfD

Decision making problems are usually cast in a Markov Decision Process (MDP) setting where an MDP is defined as a set of (S, A, T, R) which corresponds to states space, action space, transition-dynamics function, and reward dynamics function respectively. Let s_t and a_t denote the state and action respectively at time step t and $\tau = (s_0, a_0, s_1, a_1, ..)$ be any trajectory or rollout. The demonstrations are fed by a demonstrator π^* in the form of a dataset $\mathcal{D} = \{(s_i, a_i)\}_{i=0}^{N}$, where $a_i = \pi^*(s_i) \in A$. Let $P(\tau|\pi)$ and $P_t(s|\pi)$ be the trajectory distribution and the state distribution at time t respectively induced by any policy π. Finally, let $P(s|\pi) = (\frac{1}{T}) \sum_t P_t(s|\pi)$ be the overall state distribution induced by policy π. The general goal of LfD is to find a θ parameterized learner policy π_θ such that some loss function L is minimized with respect to the demonstrator policy π^* on the state distribution induced by the learner, i.e.,

$$\arg\min_{\theta} \mathbb{E}_{s\sim P(s|\pi_\theta)}[L(\pi^*(s), \pi_\theta(s))], \tag{2.1}$$

Usually, there is either limited or no access to L and so a surrogate function that uses demonstrations is used instead. For the sake of simplicity here, we will use L to represent the surrogate function. See figure 2.3 for an example.

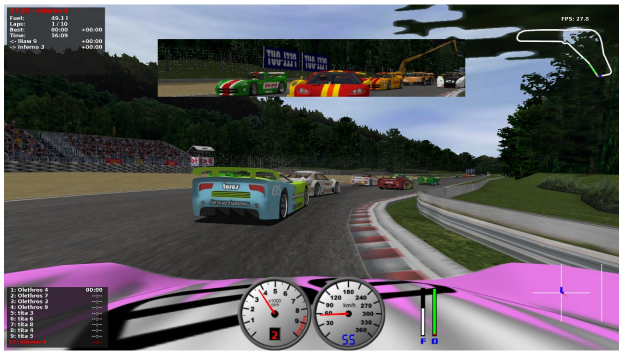


Fig. 2.3 A snapshot of the TORCS simulator [3]. Here s would be the game screen and a would be the amount of throttle and steering angles of the car. π^* can be a human demonstrator controlling a car using a joystick and π_θ can be a neural network policy. τ would be the coupled state-action pairs at every time-step in a game run and the loss (surrogate) function can be mean-squared error on the demonstrated and predicted actions.

2.1.3 Broad kinds of LfD

Very broadly LfD methods can be categorized into two kinds:

1. behavioral cloning (actively or passively with an interactive demonstrator),

2. inverse reinforcement learning

In this subsection, we will cover the above mentioned two categories of LfD. Later in this chapter we will also talk briefly about other important extensions of LfD such as third person imitation learning.

2.1.4 Behavioral Cloning

Behavioral Cloning (BC) is the simplest of all LfD techniques where the learner tries to learn the mapping between the state and action pair in the demonstrations fed to it. It can be done either actively (with an interactive demonstrator) or passively (without an interactive demonstrator). What makes BC interesting is that the fact that it is a *sequential decision making* problem where the future states depend on the previous states and actions. We will describe below the important consequence of this phenomenon as well.

Behavioral Cloning Passively

BC passively means learning the state-action mapping from demonstrations without any feedback from the demonstrator. Another way of putting this is that the learner tries to minimize the loss function L in the demonstrator induced distribution of states. Formally, this is equivalent to

$$\arg\min_{\theta} \mathbb{E}_{s\sim P(s|\pi^*)}[L(\pi^*(s), \pi_\theta(s))], \tag{2.2}$$

This looks clean and intuitive. However, the applicability of behavioural cloning is limited in the real-world.

Notice the difference between the induced state distributions in equations 2.1 and 2.2 that correspond to general LfD and behavioral cloning objectives respectively. Usually there is a per time-step learner error in mimicking the demonstrator, which we will denote as ϵ, i.e. $\epsilon = \mathbb{E}_{s\sim P(s|\pi^*)}(L(\pi^*(s), \pi_\theta(s)))$. As the subsequent states depend on the previous states and actions, this becomes a *sequential decision making* problem that violates the i.i.d assumption of supervised learning. As a result, ϵ keeps accumulating which results in inconsistencies in the learner induced and the demonstrator induced distribution of states. It has been shown that in a T-step horizon of episodes, the accumulated error in passive BC grows in the order of $O(T^2\epsilon)$ [19, 20]. See figure 2.4 for an example. As a result, behavioral cloning techniques work well when the demonstrations exhaustively span across the whole state-space of interest, ϵ remains small and long-term planning is not necessary.

BC can be however, made to do well by gathering additional demonstrator actions from nearby states to the current one. One way of doing this is by maintaining a 3-camera setup to generate demonstrations where the inputs to the right and left camera are annotated automatically with left and right directional controls respectively [6, 21]. Another way

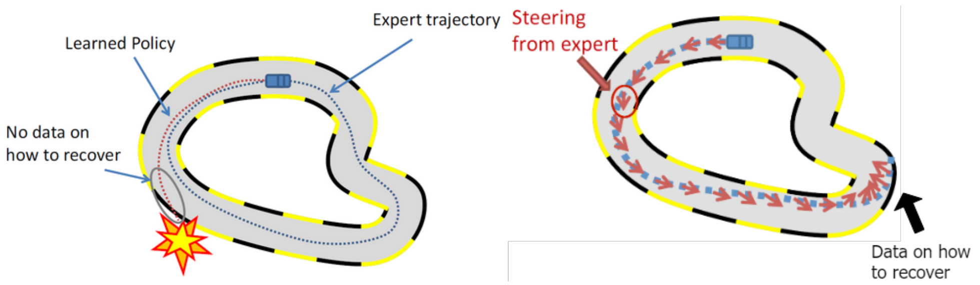


Fig. 2.4 Naive Behavioral Cloning (left figure) leads to gradual away drift from the demonstrations that can lead to potentially unwanted situation. Interactive approaches (right figure) mitigates this problem by having learner use demonstrator feedback on learner induced states. Image used with permission from the course material of Deep RL, 2015 offered at UC Berkeley.

of doing this is by adding a gaussian noise to generate such additional data instead of a 3-camera setup [22].

Active Behavioral Cloning with an Interactive Demonstrator

An interactive Demonstrator in the context of LfD is a demonstrator which can be asked for demonstrations at any state. Methods like *SEARN* [23], *DAGGER* [4] and *SMILe* [20] showed that by repetitively doing behavioral cloning on the interactive demonstrator feedback on the learner induced distribution of states in an online fashion (see figure 2.5) guarantees that the accumulation of errors does not exceed $O(T\epsilon)$. The methods in this category are analyzed using reduction-based approaches [24]. Theoretically , the LfD techniques in this category are equivalent of performing the operation in equation 2.3.

$$\arg\min_{\theta} \mathbb{E}_{s\sim P(s|\pi_\theta)}[L(\pi^*(s), \pi_\theta(s))], \tag{2.3}$$

Figure 2.6 shows how the skeleton of interactive LfD techniques looks like in one of its member techniques called *DAGGER* [4] which works the best almost invariably among all other interactive LfD techniques and serves as an ideal benchmark for newer algorithms in its category. Note that N, p, T are number of iterations, rate to decay demonstrator interjection in a rollout and number of time-steps in an episode respectively.

However, this category of LfD methods are not a silver bullet either. All interactive LfD methods assume the availability of an interactive demonstrator which is not always realistic. Moreover, they also have drawbacks that emanate from not having a notion of

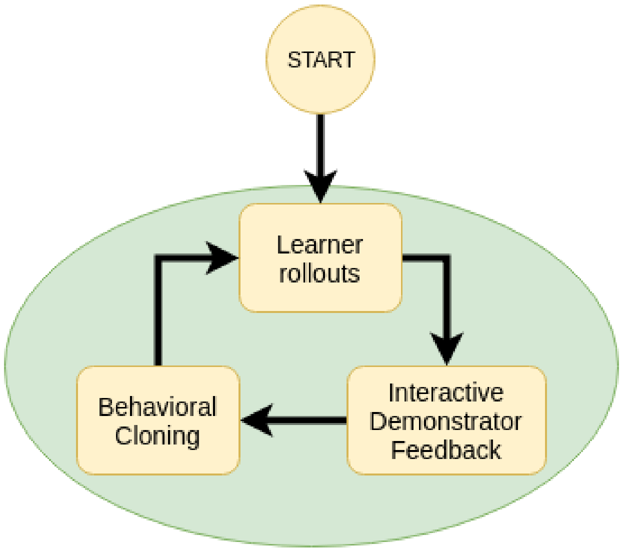
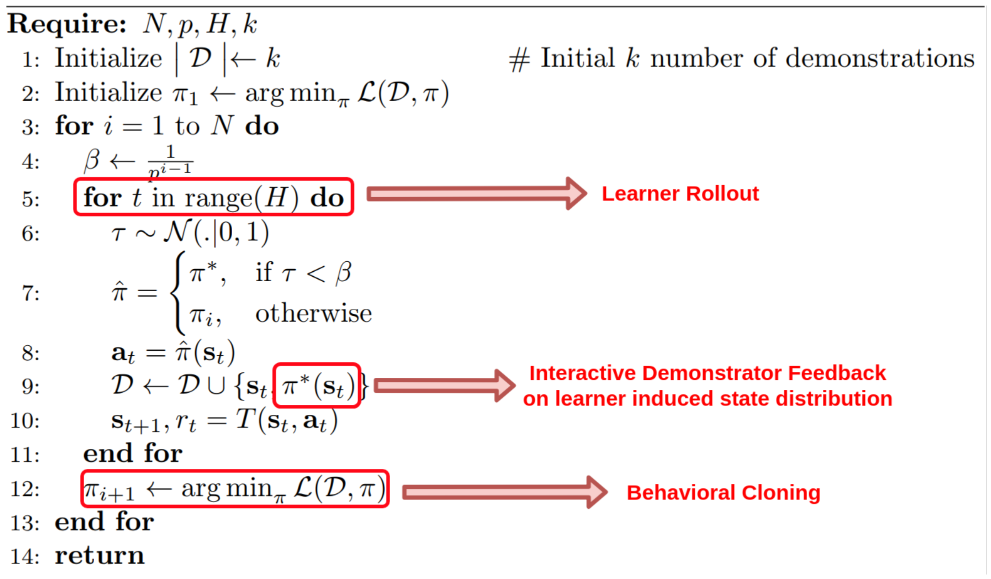


Fig. 2.5 Any interactive LfD technique repetitively keeps training itself on the demonstration feedback received on the learner rollouts.



```
Require: $N, p, H, k$
1: Initialize $|\mathcal{D}| \leftarrow k$                # Initial $k$ number of demonstrations
2: Initialize $\pi_1 \leftarrow \arg\min_\pi \mathcal{L}(\mathcal{D}, \pi)$
3: for $i = 1$ to $N$ do
4:   $\beta \leftarrow \frac{1}{p^{i-1}}$
5:   $\boxed{\text{for } t \text{ in range}(H) \text{ do}}$
6:     $\tau \sim \mathcal{N}(.|0,1)$
7:     $\hat{\pi} = \begin{cases} \pi^*, & \text{if } \tau < \beta \\ \pi_i, & \text{otherwise} \end{cases}$
8:     $\mathbf{a}_t = \hat{\pi}(\mathbf{s}_t)$
9:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \boxed{\pi^*(\mathbf{s}_t)}\}$
10:    $\mathbf{s}_{t+1}, r_t = T(\mathbf{s}_t, \mathbf{a}_t)$
11:  end for
12:  $\boxed{\pi_{i+1} \leftarrow \arg\min_\pi \mathcal{L}(\mathcal{D}, \pi)}$
13: end for
14: return
```

Learner Rollout
Interactive Demonstrator Feedback on learner induced state distribution
Behavioral Cloning

Fig. 2.6 *DAGGER* [4] is an iterative and interactive LfD technique that has demonstrated impressive results on learning policies by directly mimicking the demonstrations. The red boxes highlights the important steps in this method as described in figure 2.5.

planning integrated in the control process such as treating all deviations equally (some of them could be catastrophic) and asking for demonstrations at every visited state even if they might not be required.

In this subsection, we described BC. We explained that BC is a straightforward approach which is a promising technique to build complex systems but are susceptible to failure when the data is scarce. This is not an easy problem to solve as generating data is expensive. A part of our work involves reducing the required number of demonstrations by doing active learning. In the next section (section 2.2), we will see how a metric can be generated to automatically base decision on when to seek more demonstrations. In section 2.5, we will describe some techniques similar to our method of active learning.

2.1.5 Inverse Reinforcement Learning

Inverse Reinforcement Learning [25] is the problem setting where the reward dynamics function R is unknown or difficult to formulate and the learner has to estimate it from the demonstrations. It has also been cast as the problem of *Inverse Optimal Control* [26], estimating energy-based models [27], or learning parameters of probabilistic graphical models [28]. Such techniques have a huge applicability mainly because of two reasons

1. they allow for the for demonstrator and the learner to possess different embodiments (different S, A, and T in the definition of their MDPs),

2. the learner can do well even in the unseen part of the state-space.

Most Inverse Reinforcement Learning techniques start with demonstrations and iterate over finding a good reward estimate, evaluating it using a forward RL model, and refining the estimated reward function (see figure 2.7). However, robustly estimating the reward function and evaluating it takes a lot of compute and experiences [8].

One interesting work along the lines of IRL is Maximum Entropy Inverse Reinforcement Learning [28] where the reward function r_θ is estimated in an MDP setting which has deterministic and known dynamics, discrete state and action spaces, and linear rewards over the set of pre-defined features. Although this technique has some success so far [29, 30], it is hard to scale up to higher dimensional problems such as pedestrian detection as it has an expensive forward RL step and an exponential complexity in the number of state and action dimensions. Improvements were made to have Inverse Reinforcement Learning work

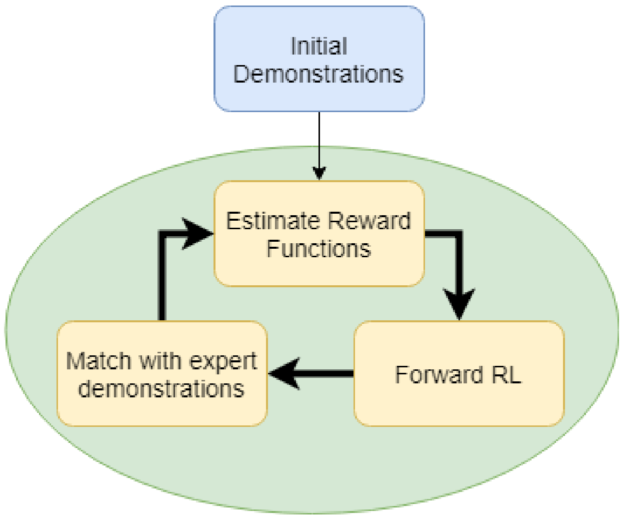


Fig. 2.7 Flow of operations in inverse reinforcement learning

with non-linear rewards [31] which was demonstrated to produce good results in complex tasks such as path planning [32]. Recently, techniques have been devised for doing Inverse Reinforcement Learning with non-linear rewards, continuous state and action spaces, and unknown dynamics [33, 34]. The performance is, however, sensitive to the estimate of the partition function which is evaluated using importance sampling.

2.1.6 Other interesting applications of LfD

Other interesting contexts where reward estimation have been used are

- **Inverse Planning/Structural Estimation in POMDPs**: Inverse Reinforcement Learning has been shown to effectively deduce rewards even in POMDPs [35, 36]. An example would be to estimate the destination of pedestrians [37].
- **Cooperative Inverse Reinforcement Learning** [38]: Here the robot has to learnt to cooperate with people where robots try to reduce the uncertainty of their reward estimation whose complete knowledge only lies with the human.
- **Task Level Inverse Reinforcement Learning** [39, 40]: The objective in this line of work is to segment sub-tasks from demonstrations without having to specify the order of their execution. This has applications where the task at hand can be

completed with a different ordering of sub-tasks which is not explicitly described in the demonstrations.

- **Reward learning from preferences** [41, 42, 43]: Specifying the right reward function is hard [44]. One promising way that has evolved recently to estimate the reward function has been through human preferences over certain options. For example, it has been shown that a complicated reward function to doing a backflip can be learned from a very low number of human feedback [43].

- **Third person imitation learning**: Another interesting line of work using LfD is *third person imitation learning* where the goal of the learner is to estimate the demonstrator policy with the limitation that the observation and action space of the learner is different from the demonstrator. Major ways of doing this has been through domain adaptation [45] and developing viewpoint invariant representations of the demonstrations [46].

2.2 Learning Predictive Uncertainties

By virtue of its ability to approximate any function [47, 48], Neural Network (NN) based architectures have achieved massive success in learning complex input-output mappings from data. However, a mere knowledge of the input-output mapping falls short when it is needed to gauge predictive uncertainty in their predictions. This can be important when the data available is limited. NNs have a propensity to overfit to the data they see. This causes unwarranted extrapolation to the unseen space of interest (see figure 2.8). This problem is glaringly apparent when the seen data does not span the whole space of interest. Hence it would be useful to generate predictive uncertainties that could reflect the confidence of the predictor about its prediction.

The probabilistic explanation behind these downsides of deterministic NN is that they attempt to evaluate the *maximum likelihood point estimates* or MLE. It does that by maximizing the log-likelihood of the seen data (see equation 2.4) given the parameters of the NN (denoted as $\boldsymbol{w}$) which is typically solved by using backpropagation.

$$\boldsymbol{w}^{MLE} = \arg\max_{\boldsymbol{w}} \log P(\mathcal{D}|\boldsymbol{w}). \tag{2.4}$$

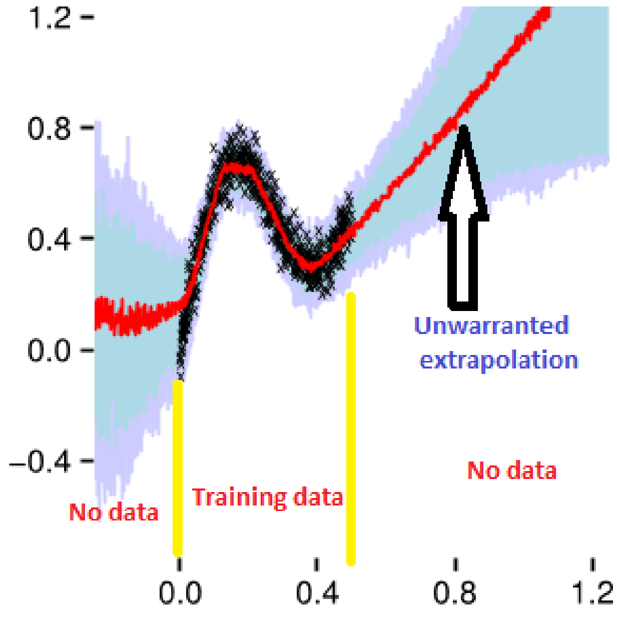


Fig. 2.8 The red line shows the typical extrapolation by any NN. Ideally it should have also predicted an uncertainty measure that should be higher at farther places in the input space. Image used from[5].

Such an optimization leads to overfitting of the NN to the seen data. Hence this fails to generalize [49]. One partial fix to this problem is instead of evaluating the MLE, estimate *maximum a posteriori* point estimates or MAP which makes the NN relatively more resistant to overfitting (see equation 2.5).

$$\boldsymbol{w}^{MAP} = \arg\max_{\boldsymbol{w}} \log P(\boldsymbol{w}|\mathcal{D}) \tag{2.5}$$

Using a Gaussian prior is equivalent to doing L2 regularization while using a Laplace prior is equivalent to L1 regularization. However this does not guarantee against any unwarranted extrapolation.

Ideally, we would like to not just have predictions but also the uncertainty in the predictions in the light of the seen data and prior beliefs. This uncertainty ideally should be higher at points that are far away from the seen region than the points that are closer. In other words, given a dataset of demonstrations $\mathcal{D}$, we would like to estimate the posterior distribution over policies $p(\pi|\mathcal{D})$ to make predictions at possibly unseen states $\boldsymbol{s}_*$ as given in equation 2.6.

$$p(\boldsymbol{a}_*|\boldsymbol{s}_*) = \int p(\boldsymbol{a}_*|\boldsymbol{s}_*, \pi)p(\pi|\mathcal{D})d\pi. \tag{2.6}$$

In this section, we will describe two state of the art methods that follow this approach: *Gaussian Process* (GP) regression and *Bayesian Neural Networks* (BNN).

2.2.1 Gaussian Process

Gaussian Processes are a popular non-parametric method for solving regression problems as shown in equation 2.6. GP regression proceeds by assuming that beliefs over different evaluations of $\pi(\boldsymbol{s})$ for $\boldsymbol{s} \in \mathcal{D}$ are jointly Gaussian. GP regression requires specifying a *mean function* $m(\boldsymbol{s})$ and a *kernel function* $k(\boldsymbol{s}, \boldsymbol{s}')$. The mean functions acts as a prior for π, while the kernel function models the covariance of the outputs $\pi(\boldsymbol{s}), \pi(\boldsymbol{s'})$ given the inputs $\boldsymbol{s}, \boldsymbol{s}'$. In this case, the learnable parameters of the policy $\boldsymbol{\theta}$ correspond to the hyper-parameters of $m(\boldsymbol{s})$ and $k(\boldsymbol{s}, \boldsymbol{s}')$. Fitting is done by optimizing $\boldsymbol{\theta}$ to maximize the marginal likelihood $p(\boldsymbol{a}|\boldsymbol{s})$ for all $\boldsymbol{s}, \boldsymbol{a} \in \mathcal{D}$.

To evaluate the predictive distribution, we construct a *kernel matrix* $\boldsymbol{K}_{\boldsymbol{S_1 S_2}}$, where its entry at i^{th} row and j^{th} column is $k(\boldsymbol{s_1}, \boldsymbol{s_2})$ with $\boldsymbol{s_1} \in \boldsymbol{S_1}$ and $\boldsymbol{s_2} \in \boldsymbol{S_2}$. We define $\boldsymbol{S}$ and $\boldsymbol{S_*}$ as two dimensional matrices of seen inputs ($\boldsymbol{s}$) and inputs to be predicted ($\boldsymbol{s_*}$) where each row correspond to one input. Since the outputs are assumed to be jointly Gaussian, the resulting predictive distribution is a Normal distribution with the mean $\boldsymbol{\mu}(\boldsymbol{S_*})$ and variance $\boldsymbol{\sigma}^2(\boldsymbol{S_*})$ given as in equation 2.7.

$$\begin{aligned} \boldsymbol{\mu}(\boldsymbol{S_*}) &= m(\boldsymbol{S_*}) + \boldsymbol{K}_{\boldsymbol{S_* S}} \boldsymbol{K}_{\boldsymbol{SS}}^{\boldsymbol{-1}}(\boldsymbol{a} - m(\boldsymbol{S_*})) \\ \boldsymbol{\sigma}^2(\boldsymbol{S_*}) &= \boldsymbol{K}_{\boldsymbol{S_* S_*}} - \boldsymbol{K}_{\boldsymbol{S_* S}} {\boldsymbol{K}_{\boldsymbol{SS}}}^{-1} \boldsymbol{K}_{\boldsymbol{S S_*}} \end{aligned} \tag{2.7}$$

We use $\boldsymbol{\mu}(\boldsymbol{s}_*)$ as the action to be applied by the agent and $\boldsymbol{\sigma}^2(\boldsymbol{s}_*)$ as a measure of its uncertainty. Although inference can be done exactly, this method has a higher computational cost than inference with BNNs. We refer the reader to [50] for more details.

2.2.2 Bayesian Neural Networks

Bayesian Neural Networks (BNN) are NN whose weights or parameters are expressed as a distribution rather than a deterministic value and learned using Bayesian inference (see figure 2.9). Their innate potential to simultaneously learn complex non-linear functions from data and express uncertainties have lent them a major role in our pursuit to develop more capable AI. The posterior distribution in this case is defined over the parameters of

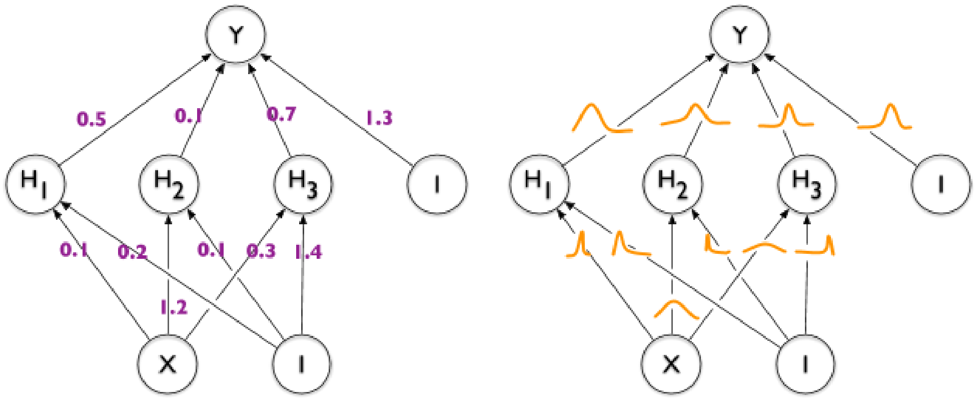


Fig. 2.9 Unlike deterministic Neural Networks (left) that have a fixed value of their parameters, BNN (right) have a distribution defined over them.

the model which also represents our policy, $P(\pi|\mathcal{D}) = P(\boldsymbol{w}|\mathcal{D})$, where $\boldsymbol{w}$ are the weights of the NN model.

One way to generate uncertainty in the prediction in the light of seen data and prior beliefs is to estimate the whole posterior distribution by doing a Bayesian Inference. Doing a Bayesian Inference uses Bayes rule in the light of seen data $\mathcal{D}$ to estimate a posterior distribution of the parameters. This is the underlying concept of the BNN training (see equation 2.8).

$$
\begin{aligned}
P(\boldsymbol{w}|\mathcal{D}) &= \frac{P(\mathcal{D}|\boldsymbol{w}).P(\boldsymbol{w})}{P(\mathcal{D})} \\
\text{where,} & \\
P(\boldsymbol{w}|\mathcal{D}) &\implies \text{Posterior parameter distribution} \\
P(\mathcal{D}|\boldsymbol{w}) &\implies \text{Data likelihood} \\
P(\boldsymbol{w}) &\implies \text{Prior parameter distribution} \\
P(\mathcal{D}) &\implies \text{Evidence}
\end{aligned}
\tag{2.8}
$$

The prediction step to compute output of the new samples, say $\boldsymbol{s}_*$ is done by taking an expectation of the output over the optimized posterior parameter distribution, say $P(\boldsymbol{w}_*|\mathcal{D})$

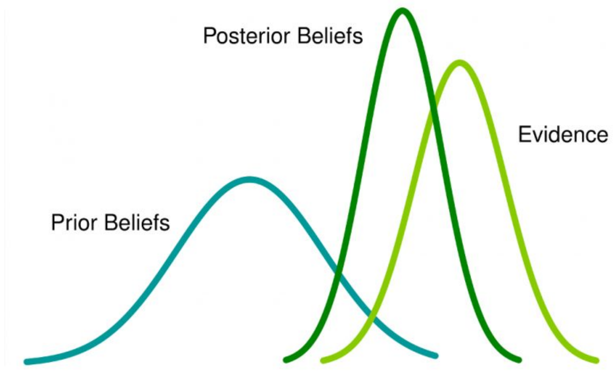


Fig. 2.10 Bayesian inference adjusts the beliefs about a distribution in the light of data or evidence.

as given in equation 2.9.

$$P(\boldsymbol{a_*}|\boldsymbol{s}_*) = \mathbb{E}_{P(\boldsymbol{w}_*|\mathcal{D})} P(\boldsymbol{a_*}|\boldsymbol{s}_*, \boldsymbol{w}_*) \tag{2.9}$$

This expectation is equivalent to predicting using ensembles of an infinite number of NNs which leads to model averaging and hence resistance to noise.

However, both the exact computation of the posterior in the form in equation 2.8 and the prediction step as shown in equation 2.9 are computationally intractable. Computing the denominator term in equation 2.8 needs marginalizing the evidence over all the possible parameters in our parameter space which is hard. Also, computing the exact Bayesian inference on the weights of a neural network is intractable as the number of parameters is very large and the functional form of a neural network does not lend itself to exact integration [5]. Hence various ways to approximate this in the context of BNNs have been developed which yields us a wide variety of BNNs today [5, 51, 52, 53, 54, 55, 56, 57]. Also, finding a form to differentiate with respect to parameters of distributions is not possible which is indispensable for backpropagation.

Note that BNNs should be seen as different from NNs that have distribution estimation applied to stochastic hidden units rather than over the parameters. The former is a way to choose suitable NNs (hence regularization and model averaging) while the latter is about expressing uncertainty about a particular observation [5].

In conclusion, BNNs are useful for integrating and modelling uncertainties. Furthermore, they have also been shown to improve predictive performance [58, 5] and carry out systematic exploration [59]. Recent advances in the field of deep learning and hardware allow us to approximate the relevant quantities scalably using off-the-shelf optimizers. The fundamental problems in developing BNNs or any probabilistic model are computations of the posterior distribution and their expectations. Hence we have to resort to their approximation. There are broadly two categories of methods of doing this approximation - stochastic (eg. Markov Chain Monte Carlo) and deterministic (eg. variational inference). For readers interested in knowing more about them, I would point to chapters 10 and 11 of the book[60].

Bayes-by-Backprop: A Bayesian Neural Network

As explained in the subsection 2.2.2 BNNs provide a framework to capture uncertainty in

- Parameters through equation 2.8
  $P(\boldsymbol{w}|D) = \frac{P(D|\boldsymbol{w})P(\boldsymbol{w})}{P(D)}$
- Predictions through equation 2.9
  $P(\boldsymbol{a}_*|\boldsymbol{s}) = \mathbb{E}_{P(\boldsymbol{w}_*|\mathcal{D})} P(\boldsymbol{a}_*|\boldsymbol{s}, \boldsymbol{w}_*)$

However, these two operations are intractable in general because of the following:

- **Intractable Posterior Updates**: The shape of the parameter distribution makes it difficult to compute the evidence term $[P(D) = \sum_{\boldsymbol{w}} P(D|\boldsymbol{w})P(\boldsymbol{w})]$ in equation 2.8. Hence, computing the posterior is impossible.
- **Non-integrable**: Functional form of NN does not lend itself to exact integration which makes making prediction as in equation 2.9 impossible.

Bayes-by-Backprop [5, BBB] is a Variational-Inference [61] based BNN technique where a probability distribution is defined over the weights of the network. It approximates the intractable posterior distribution of the weights of the network by Variational Inference. Monte-Carlo samples are used to evaluate and update the variational approximation. This whole set-up is made amenable for back-propagation using Gaussian reparameterization [62, 5]. The re-parametrization makes all randomness an input to the model, making the network deterministic for differentiation (see figure 2.11).

The approximating (variational) distribution q over the policy $\boldsymbol{w}$ is given by a Gaussian with diagonal co-variance, $q(\boldsymbol{w}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. This Gaussian is parametrized with $\boldsymbol{\theta} =$

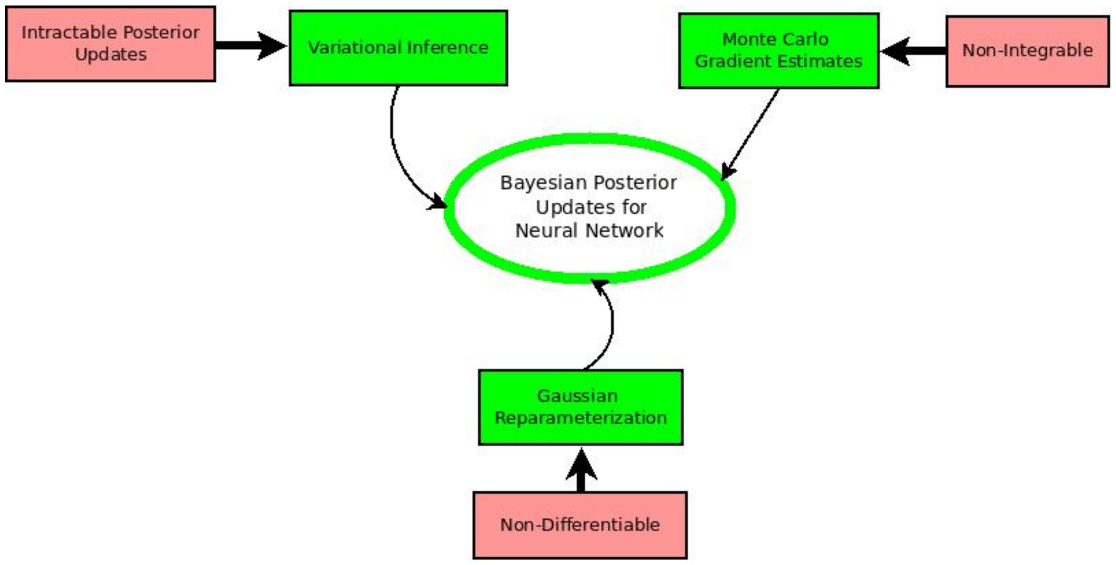

Intractable Posterior Updates
Variational Inference
Monte Carlo Gradient Estimates
Non-Integrable
Bayesian Posterior Updates for Neural Network
Gaussian Reparameterization
Non-Differentiable

Fig. 2.11 BBB gets around the problem of intractable posterior updates, intractable prediction, and unsuitable form for backpropagation by using the techniques of Variational Inference, Monte-Carlo sampling, and Gaussian Reparameterization.

$[\boldsymbol{\mu}^T, \boldsymbol{\rho}^T]^T$, where $\boldsymbol{\sigma}_{ii} = \log(1 + \exp(\boldsymbol{\rho}_i))$. $\boldsymbol{\sigma}_{ii}$ is computed in this way to make sure that it is always positive.

Fitting the model by a mere use of Variational Inference is done by minimizing the Kullback-Leibler divergence between the true and approximate posterior, which also has an information theoretic justification by means of a bits-back argument [63]. Here, $\boldsymbol{\theta}_* = [\boldsymbol{\mu}_*^T, \boldsymbol{\rho}_*^T]^T$ is the optimized variational posterior (See 2.10).

$$\begin{aligned}\boldsymbol{\theta}_* &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}\left[q(\boldsymbol{w}|\boldsymbol{\theta})||P(\boldsymbol{w}|D)\right] \\ &= \arg\min_{\boldsymbol{\theta}} \int q(\boldsymbol{w}|\boldsymbol{\theta}) \log \frac{q(\boldsymbol{w}|\boldsymbol{\theta})}{P(\boldsymbol{w}|D)} \\ &= \arg\min_{\boldsymbol{\theta}} -\mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{w})}\left[P(\mathcal{D}|\boldsymbol{w})\right] + \mathrm{KL}\left[q_{\boldsymbol{\theta}}(\boldsymbol{w})||P(\boldsymbol{w})\right]\end{aligned} \tag{2.10}$$

However, it is intractable to compute the terms in equation 2.10. Hence we approximate it using Monte-Carlo samples which may be expressed with an objective of the form

$$\mathcal{L}(D, \boldsymbol{\theta}) \approx \frac{1}{M}\sum_{i=1}^{M} \log q(\boldsymbol{w}^{(i)} \mid \boldsymbol{\theta}) - \log\left(P(\boldsymbol{w}^{(i)})P(D \mid \boldsymbol{w}^{(i)})\right), \tag{2.11}$$

where $\boldsymbol{w}^{(i)}$ is the i^{th} Monte-Carlo sample from $q(\boldsymbol{w}|\boldsymbol{\theta})$. We use a zero-mean Gaussian as prior $P(\boldsymbol{w}^{(i)})$.

The inference on a new input $\boldsymbol{s}_*$ is done via Monte Carlo sampling as shown in equation 2.12.

$$p(\boldsymbol{a}_*|\boldsymbol{s}_*) \approx \frac{1}{M}\sum_{i=1}^{M} p(\boldsymbol{a}_*|\boldsymbol{s}_*, \pi_{\boldsymbol{w}_i}), \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w}|\boldsymbol{\theta}_*) \tag{2.12}$$

Predictive uncertainties have been shown to be useful for multiple purposes. For example, PILCO [64] uses Gaussian processes for learning a probabilistic dynamics model and long-term planning. In Deep-PILCO [65, 66] a dropout training approach [67] was used to quantify the uncertainty of a learned neural network dynamics model. As an alternative, a heuristic measure has been used to quantify trust on the demonstrations in [68]. However, unlike in our work, all these methods focused on a single task, and most of them focus on model-uncertainty and not policy-uncertainty.

In our work, we use predictive uncertainties to help the controller distinguish familiar from non-familiar contexts. However, the world can be partially observable where its partial observability can can show up in forms of changes in the reward function or the transition function of the MDP. One way to identify such partially observable changes is by conditioning the policies on the history or the sequences of previous observations. In the next section (section 2.3), we will see the different ways to learn from sequential data.

2.3 Learning from sequential data

There are four broad classes of methods for capturing sequential information [69] as explained below.

1. **sliding window:** The easiest to use with any off-the-shelf supervised learning methods, which we also employ in our work is using a sliding temporal windows on sequences [70].

2. **recurrent models:** The second method is using recurrent models such as LSTM networks which are relatively difficult to train and work with evenly sampled sequences [71].

3. **regression based extrapolation:** The third technique which works well with unevenly sampled data is regression-based extrapolation where training data is fitted to

a curve [72, 73].

4. **convolutional models:** Lately Convolution based NN techniques have been shown to perform at least as good as recurrent models for learning from sequences [74], [75], [76], [77], [78]. Some of its exclusive advantages include allowing more support for more parallelism, controllable length of history intended to be learnt, stabler gradients than recurrent models, and lower memory requirements [74].

2.4 Adaptive Control

Adaptive Control [79] is the umbrella term for techniques that allow controllers to adapt their control law based on the inputs they receive. Note that it is different from a similar class of techniques called *robust control* [80] that uses the same control law for a range of values of inputs, i.e. they do not change their control law. For example, adaptive control based techniques change the control law for an aircraft system depending on the amount of fuel it has in its tank during its flight. Usually, parameters of the system are estimated which makes use of the update laws to update the control laws. These update laws are derived using *Lyapunov stability*. Estimation of the parameters uses techniques from *System Identification* [81] that uses statistical methods to estimate data-driven mathematical models of dynamical systems.

2.5 Active Learning

In the learning from demonstration paradigm, the learner usually waits passively for the demonstrator to produce a dataset. Active learning, on the other hand, aims to reduce time-consuming data generation and training by seeking to select the most informative learning experiences [82, 83]. Active learning is more effective in learning from demonstration frameworks where judicious use of demonstrations is vital but a lack of exhaustive dataset could result in compounding errors [4].

Active learning for control has been widely studied while doing LfD. Confidence-Based Autonomy [84] is an example where rich interactions between an agent, aware of its own uncertainty, and a benevolent teacher lead to rich learning interactions. This work uses additional supervised models to detect unfamiliar and ambiguous states. Unlike using such supervised learning techniques, our work uses the same network that does control to detect such unfamiliar and ambiguous states. The approaches in [85], and [86] are similar to the approach in Confidence-Based-Autonomy. While the former was vague in terms of the mechanism to detect the bad states, the latter used a set of pre-conditions and post-conditions to identify the mismatch in the resulting states which is hard to scale up. In our work, we will show how we can leverage the predictive standard deviation as uncertainty to invoke demonstrations only when their absence could lead to sub-par performance in a manner that scales up naturally.

In the next section (section 2.6), we explain *transfer learning* (TL). TL addresses how policies can be transferred positively [87, 88] when there are subtle changes between train-time and test-time conditions which might cause controllers to fail silently [8, 89, 90, 87]. We call these different settings as different *contexts* in our work. Our methodology can be seen as one such method. Hence we find it relevant to explain TL.

2.6 Transfer Learning

Transfer learning is an umbrella term for methods that seek to develop high performing learners on a target task in a target domain by previously learning source tasks in source domains [91]. In the real-world, source tasks are the tasks where it is easy to obtain data or experience. Effective transfer learning is important as training separately on each task is impossible given the data-inefficiency of our current data-driven AI algorithms. Performance of these transfer learners are usually compared with learners that learn from scratch on the target task. The gained improvements are generally captured through metrics such as jumpstart, asymptotic performance, total reward, transfer ratio, time to the threshold and so on [91]. Transfer learning also takes place in our living world as crucial information is passed from generation to generation through genes [92]. There have been attempts like allowing the simulator policy to initialize the real world policy [93], to allow policy adjustment model that compensates for simulator-real world discrepancies [94], training a RNN across multiple MDPs, to obtain useful biases that will aid the learning of other tasks in the future [10, 95]. However, these approaches lack an easily scalable way to detect training-testing distribution discrepancies and quantify confidence in a transfer.

What makes transfer learning complicated is the fact that the training conditions (source) and the deployment conditions (target) can vary in multiple different ways. A few of them are task distributions mismatch, state space mismatch, feature space mismatch, action space mismatch, a discrepancy in the optimal (true) underlying mapping between input and output etc. This creates many subtypes of transfer learning problems such as *Continual Learning*, *Multitask Learning*, *Lifelong Learning*, *Domain Adaptation*, learning under *Co-Variate Shift* which we will cover in the following subsections. In the context of our work, we will restrict ourselves to describing transfer learning only in *control* or *Reinforcement Learning*.

2.6.1 Multitask Learning

Unlike transfer learning where the source and the target tasks can come from different distributions, in multitask learning they always come from the same distribution. This constraint makes working under multitask learning relatively easier than transfer learning. For example unlike some problems in transfer learning, the action and observation dimensions in multitask learning set up are always consistent between training and deployment settings. Also usually, training times on the source tasks are assumed to be free of cost. Some popular multitask learning techniques include [96], [97], and [98].

2.6.2 Lifelong/Continual Learning

Artificially intelligent agents in the real world are bound to encounter a stream of data from different tasks sequentially that can vary both temporally and spatially. Learning under such settings are referred to as Lifelong or Continual Learning. Learning under continual learning setting is harder than transfer learning as knowledge has to be transferred positively with limited system resources while being resistant to *catastrophic forgetting* [99]. The field of lifelong learning is still not mature and even the right metrics to evaluate these systems are yet to be established.

2.6.3 Domain Adaptation

Domain adaptation is a way of doing transfer learning where the goal is to learn to represent a source domain into another domain which can be the same or different from the target domain in a way that the learner does well on the target domain. The most successful methods in this category are based on finding a common representation space using adversarial learning [100] techniques which seek to find indistinguishable feature representation in the source and the target domains.

2.6.4 Learning under Covariate-Shift

Covariate shift refers to the problem where the distribution of the independent variables/features also known as covariates change while the underlying true mapping function remains unchanged. Usually, the task stays the same during training and validation/deployment. It can also be seen as looking at an unexplored area of the input space. Covariate shift is different from the problems that can be handled by the domain adaptation approach as unlike covariate shift, such problems can have their underlying true mapping function changed on deployment. This problem has been intensively studied in LfD domains.

In this chapter, we explained in detail the various methodologies that we draw upon to build our work like LfD (section 2.1), techniques to generate predictive uncertainties (section 2.2). We also describe some related work in the space of active LfD in section 2.5. At the end, we briefly touched upon TL to help the readers position our work with respect to TL. We highlighted various ways of approaching them while anchoring them around their advantages and disadvantages. In the next chapter (chapter 3), we will describe our methodology that is built using techniques described in this chapter.

Chapter 3

Design and Methodology

In this chapter, we formalize and explain how we set our test-beds and our motivation behind setting them up in the way we did. Similarly, we also explain our proposed methodology that is designed to work well under diverse situations as posed by our set-up. By the end of this chapter, the reader will know the kind of problems our methodology is aimed to solve and the exact way it goes about doing so.

3.1 Formalizing the set-up

In a lot of situations, the way a robot faces the situations in the world is online in the sense that observations appear one after another in a temporally linear fashion. This online world is also structured as future observations depend on the decisions made at earlier points of time. Moreover, these observations can come from different unobservable underlying functions and yet look similar. Such diverse underlying functions can be formalized as different *contexts*. As a result, there are a huge number of problems out in the wild real world that necessitate their solutions to be equipped with the ability to deal with such structured streaming incoming data from multiple contexts with the guarantees of performance and safety. Such an ability has to be learnt in a data-driven fashion given the complex and high-dimensional nature of our world. Hence, we set our set-up as an incoming sequence of diverse contexts.

We denote each context in a domain as an MDP defined as $d = (S, A, R_d, T_d)$ where S is the state-space, and A is the action space. $R_d(\mathbf{s}_t, \mathbf{a}_t)$ and $T_d(\mathbf{s}_t, \mathbf{a}_t)$ are the reward and transition dynamics function which take as arguments an action $\mathbf{a}_t$ and state $\mathbf{s}_t$. All contexts in a domain share S and A but can have different R_d and T_d.

To test the capability of our proposed methodology (section 3.2) to invoke a fallback policy in a timely manner, we set D as the set of diverse contexts that a robot might encounter one after another sequentially. In our example of a self-driving car given earlier, this sequential nature of multiple contexts would correspond to a situation where the car drives on a dry road (a context) before driving on a icy road (another context). To test one such example of an effective fallback strategy, we let the controller actively seek demonstrations of any context d. Every d has an expert demonstrator π_d^* which provides demonstrations upon request. These new requested demonstrations are merged with the complete dataset $\mathcal{D}$. A good way to see the effectiveness of our approach would be to see if such an active learner can bring such requests for context-specific demonstrations to a minimum while not compromising on performance(see figure 3.1).

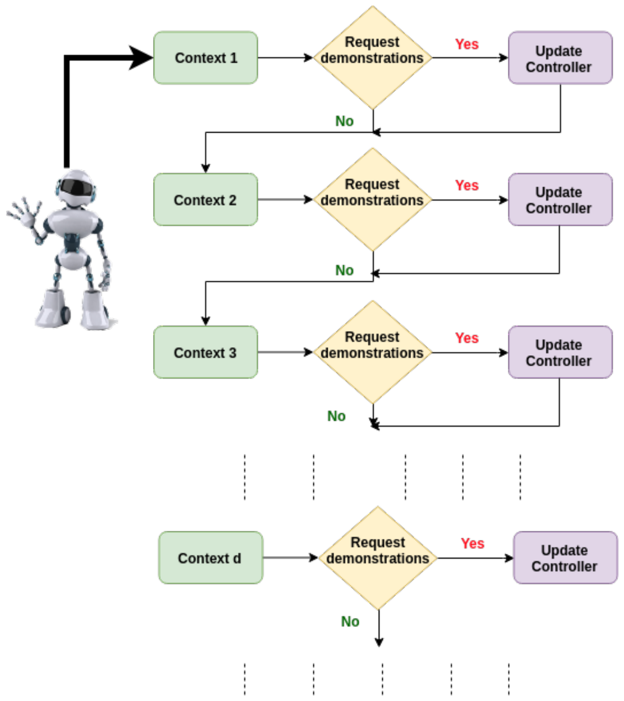


Fig. 3.1 Set up for validating effectiveness of our proposed active learner. The controller faces contexts sequentially where at each context it has to decide whether or not to ask for more demonstrations.

3.2 Proposed Methodology

In this section, I will walk the readers through our proposed methodology for *BC on multiple contexts using BNNs*. We use a *BNN* called Bayes-by-Backprop [5] (BBB) to quantify uncertainty over the unseen situations. We then build on this to perform active learning on multiple-contexts in the BC framework. In short, we use the predictive standard deviation of BBB to quantify the confidence in the controller performing well on the current task. High uncertainty means low confidence of the policy that it would be successful.

To detect unfamiliar dynamics rather than just unfamiliar states, we use BBB to learn policies for which the inputs are temporal windows of the interaction history (see subsection 3.2.1). The associated uncertainty is used to decide whether the agent is confident enough that it can perform well in the current situation, or if it should request demonstrations for additional training on the current context (subsection 3.2.3).

The novelty in our mechanism lies in three parts that interact with each other, and together allow the application of Bayesian methods to quantify uncertainty in imitation learning policies: temporal-windows for input representation, learning policies for generating both actions for control and uncertainty over a given situation, and the detector that decides whether the current policy generalizes well to the current situation. These three elements are described in detail in the sections below.

3.2.1 Temporal Windows

In order to implicitly identify differences in dynamics and adapt to those changes, we use short histories of k time-steps or *temporal windows* instead of just a single state as input to the policy. For example, if the task dynamics depends on the mass of a system, the greater inertia can only be identified when comparing two subsequent states while a known force is applied.

The dataset $\mathcal{D}$ for training is extracted from the demonstrations with such a temporal window, with a stride of one time-step. The overlapping windows $\boldsymbol{x}_t$ consist of the last k states $\boldsymbol{s}_{t-k:t}$ paired with their associated actions $\boldsymbol{a}_{t-k:t-1}$ and rewards $r_{t-k:t-1}$ (see figure 3.2). The mapping to $\boldsymbol{x}_t$ in $\mathcal{D}$ is kept as $\boldsymbol{a}_t$. On deployment/validation of the learned controller, a similar process is followed with updating of the temporal window taking place in an online fashion.

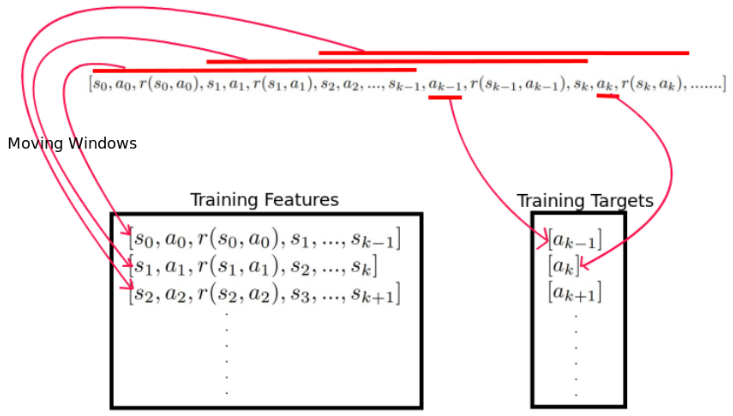

$[s_0, a_0, r(s_0, a_0), s_1, a_1, r(s_1, a_1), s_2, a_2, ..., s_{k-1}, a_{k-1}, r(s_{k-1}, a_{k-1}), s_k, a_k, r(s_k, a_k), ...]$
Moving Windows
Training Features
$[s_0, a_0, r(s_0, a_0), s_1, ..., s_{k-1}]$
$[s_1, a_1, r(s_1, a_1), s_2, ..., s_k]$
$[s_2, a_2, r(s_2, a_2), s_3, ..., s_{k+1}]$
Training Targets
$[a_{k-1}]$
$[a_k]$
$[a_{k+1}]$

Fig. 3.2 The process of creating the training-dataset: A temporal window of size of k time-steps moves over the history of the demonstration history and captures the transitions. This set-up gives the policy learning BNN the opportunity to infer the underlying latent dynamics.

3.2.2 Learning BNN Policies

In order to use BBB to train on temporal-windows to output predictive uncertainties that reflects the level of generalizability of the learner to a given situation along with the actions, we train BBB on the dataset $\mathcal{D} = \{(x_i, a_i)\}_{i=0}^{N}$ created as described in subsection 3.2.1. The BBB learner is trained by minimizing Eq. 2.11. Hence, the parameters $\boldsymbol{\theta}^*$ of the learned controller $\hat{\pi}_{\boldsymbol{\theta}^*}$ are defined as in equation 3.1.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}, \boldsymbol{\theta}), \tag{3.1}$$

where, $\mathcal{L}$ is defined in equation 2.11. After optimization, the controller computes its prediction of the expert action and its uncertainty about the current situation with $\boldsymbol{x}_t$ as input according to equation 3.2, i.e.

$$\boldsymbol{a}_t, \boldsymbol{\sigma}_t = \hat{\pi}_{\boldsymbol{\theta}^*}(\boldsymbol{x}_t), \text{ with } \boldsymbol{a}_t = \mathbb{E}\left[\hat{\boldsymbol{a}}_{\boldsymbol{t}}|\boldsymbol{x}_t\right], \boldsymbol{\sigma}_t = \sqrt{\mathrm{Var}\left[\hat{\boldsymbol{a}}_{\boldsymbol{t}}|\boldsymbol{x}_t\right]}, \tag{3.2}$$

which are estimated using Monte-Carlo samples. The vector $\boldsymbol{\sigma}_t$ is then converted to a scalar σ_t by taking the average over the standard deviations obtained for each action-dimension.

Algorithm 1 Select action and train if necessary

```
Require: $\boldsymbol{x}_t, \mathcal{D}, \boldsymbol{\theta}, d, c$
1: $\boldsymbol{a}_t, \sigma_t \leftarrow \hat{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$
2: if smoothed$(\sigma_t) > c\Omega$ and $t > t_{start}$ then
3:     $\mathcal{D} \leftarrow \mathcal{D} \cup$ GetDemonstrations$(d)$
4:     $\boldsymbol{\theta}^* \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$          #relearn
5:     $t \leftarrow 0$          # re-start episode in current context
6:     $\Omega =$ AverageUncertainty()          # adapt query threshold
7:     $\boldsymbol{a}_t, \sigma_t = \hat{\pi}_{\boldsymbol{\theta}^*}(\boldsymbol{x}_t)$          # re-select action
8: end if
9: return $\boldsymbol{a}_t, \mathcal{D}, \boldsymbol{\theta}^*$
```

3.2.3 Detector

The controller will likely encounter both situations with dynamics that are sufficiently similar to allow good policy generalization, as well as dynamics that are quite different. Based on the controller's confidence, it decides whether to continue with the task, or to stop and ask for demonstrations of the current task.

Particularly, we first smooth σ_t with a moving average over m time steps. The controller deems itself to be not confident enough to proceed if the average uncertainty exceeds a scaled adaptive threshold $c\Omega$ where c is a scaling factor and Ω is the adaptive threshold. The value of adaptive threshold Ω is automatically set to the average uncertainty obtained over all the contexts the controller has trained itself on so far. Both the constants c and m can be tuned to yield more or less sensitive learners, which we show in subsection 5.3.2.

3.2.4 Connecting the dots

Algorithm 1 describes the steps of our proposed approach, with Figure 3.3 showing how our method is used in our set up. Using a BNN instead of a DNN allows us to obtain the predictive standard deviation along with the action for each history window. The temporal windows contain the controller's state information over consecutive time-steps, which allows the BNN to implicitly infer the underlying dynamics. The uncertainty is passed to a detector that smooths this signal and compares it to a threshold. When the uncertainty is high, continuing will likely lead to poor performance. Hence, task execution is paused and more training is done using new demonstrations solicited on the task at hand. The value of the adaptive threshold Ω is initialized with 0 so that it always trains on

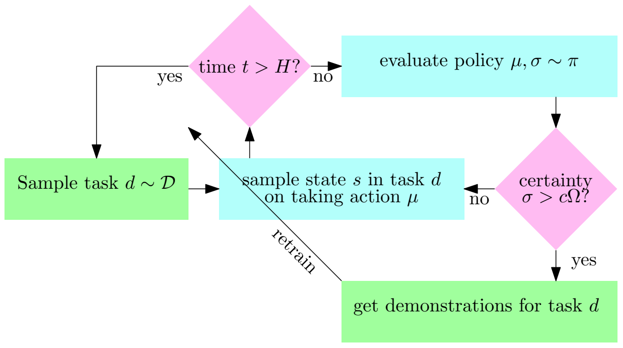

time $t > H$?
yes
no
evaluate policy $\mu, \sigma \sim \pi$
Sample task $d \sim \mathcal{D}$
sample state s in task d on taking action μ
no
certainty $\sigma > c\Omega$?
yes
retrain
get demonstrations for task d

Fig. 3.3 The proposed active learner: The controller has to perform well on all tasks it faces sequentially with limited requests for task-specific demonstrations. A certainty threshold on the predictive variance is used to decide when to ask for demonstrations.

requested demonstrations during the first task it faces. Following each training episode, the threshold Ω is adapted according to the uncertainty obtained in the most recent re-learning step.

In the next chapter (chapter 4), we will explain the experiments that we set up to highlight the exclusive features of our proposed methodology. We would also explain the unique characteristics of these experiments that make them a good fit for our proposed methodology.

Chapter 4

Experiments

In this chapter, we walk the readers through the experiments that we have set up to gauge the viability and effectiveness of our approach in real-world like high-dimensional and complex tasks. Our experiments consist of training robust controllers for different *contexts*. These closely related contexts are obtained by varying parameters of the environment, for example the masses and lengths of parts of the system to be controlled. These parameters are not directly observable by the learning agent, but do have an effect on the outcome of applying an action. Since the learner is not told the context explicitly, it has to infer it from the past transitions within the temporal window. For all experiments, we require *expert* demonstrations. While our system can in principle operate with a human demonstrator, for purposes of reproducible science, we base our experiments on previously optimized agents.

Note that our experiments are set in a way that the ϵ as explained in subsection 2.1.4 within a context is not too bad and long-term planning is not necessary. As a result, there is no significant compounding loss observed by the learner in a context after it trains on it. However, our method can indeed be used to mitigate some of the effects of the compounding of errors in BC learning settings. We have described it in detail in chapter 6.

4.1 Double Integrator experiment of sliding block on ice

We did some simpler experiments on a low-dimensional task: a double integrator domain of a point mass sliding on a friction-less surface, where the goal is to apply forces so as to come to an halt exactly at coordinate $x = 0$. The demonstrator here is an LQR based controller obtained by solving the Riccati equation. The intention behind setting up this experiment is to highlight the ability of our methodology to scale up to high-dimensional tasks organically.

4.2 Real Robotic Pendulum Swing up

We tested parts of our method on a physical Furuta pendulum ([101]). The goal in this environment is to swing-up and balance the pendulum upright, by applying a torque to the base joint. This is analogous to the standard cart-pole swing-up task, with 4 continuous state dimensions and 1 continuous action dimension. To obtain data from different contexts, we varied the pole mass. This was done by attaching weights of varying sizes to the pole. We used the model-based method described in [66] to obtain expert policies for each context. Precisely, 7 contexts are generated by varying the mass of the tip of the pendulum as

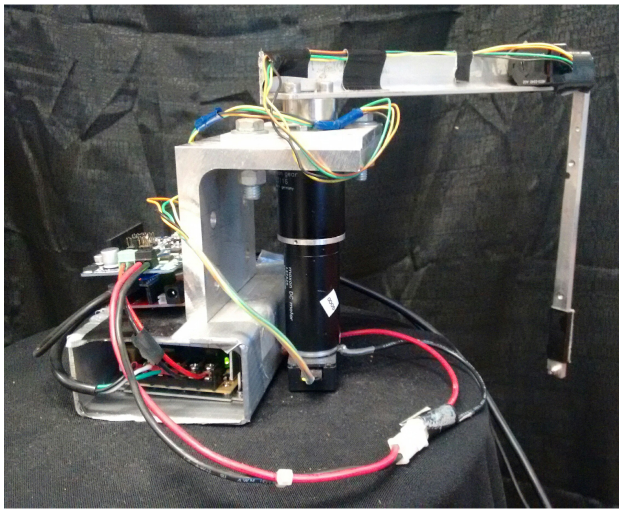


Fig. 4.1 Illustration of the experimental set-up for the real robotic pendulum swing up experimental platform.

specified in the following table (4.1).

The goal behind doing this real-robot experiment is two-fold. The first goal is to prove viability of our approach for real-robots which are hard to control in general. The second goal is to show that it is possible to generate data-driven uncertainty in decision-making that consistently reflects the degree of success that a given policy can achieve.

Context Identity	Pole Mass
0	70.
1	91.0
2	105.0
3	149.0
4	104.0
5	121.0
6	252.0

Table 4.1 Masses of the poles of different contexts of the real robotic pendulum

4.3 MuJoCo

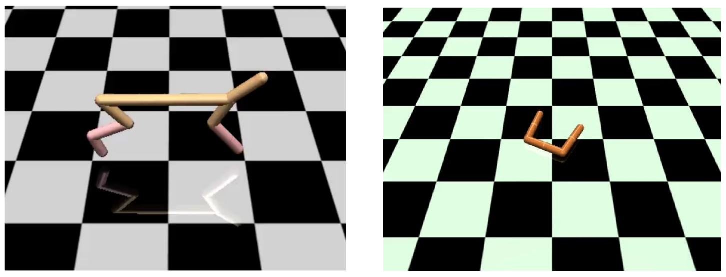


Fig. 4.2 Illustration of the HalfCheetah(left) and Swimmer(right) MuJoCo experiments that we employed in our work.

We tested our method on two modified OpenAI Gym environments ([12]) by creating 10 contexts on both HalfCheetah and Swimmer. In both environments, the goal is to find locomotion controllers to more forward as fast as possible. To simulate different contexts, we changed either or both the lengths and masses of various body parts like torso, middle, and back. The MuJoCo version used is *mjpro131* with the domains identified as *HalfCheetah-v1* and *Swimmer-v1*. For these experiments, we used proximal policy optimization (PPO) ([102]) to obtain expert policies for each context.

A family of related contexts was created for the swimming environment by changing the mass of body segments (for the first five contexts) or the length of body segments (for the last five contexts. The default body lengths of the Swimmer is a 3-dimensional vector of value $(1, 1, 1)$. The exact configuration details are specified in tables 4.2 and 4.3.

The motive behind setting this experiments is to show the viability of our approach to

Context Identity	Mass dimension(s)	Original Mass	Changed Mass
0	No change	0.	No change
1	$(1, 2)$	$(6.3, 1.5)$	$(10., 5.)$
2	(1)	(6.3)	(0.5)
3	(2)	(1.5)	(4.5)
4	(2)	(1.5)	(3.0)
5	(1)	(6.3)	(0.8)
6	$(1, 2)$	$(6.3, 1.5)$	$(0.8, 0.5)$
7	$(1, 2)$	$(6.3, 1.5)$	$(0.5, 0.5)$
8	$(1, 2)$	$(6.3, 1.5)$	$(10.0, 4.5)$
9	$(1, 2)$	$(6.3, 1.5)$	$(10.0, 6.0)$

Table 4.2 Masses of body segments of different tasks of HalfCheetah domain

Context Identity	Length dimension	Mass dimension	Changed Value
0	No change	No change	No change
1	0	No change	1.5
2	1	No change	1.5
3	0	No change	4.0
4	0	No change	10.0
5	NA	2	28.0
6	NA	2	32.0
7	NA	3	32.0
8	NA	2	25.0
9	NA	1	30.0

Table 4.3 Masses of body segments of different contexts of Swimmer domain. Note that the default body lengths of the Swimmer is a 3-dimensional vector of value $(1, 1, 1)$ and default body mass is a 4-dimensional vector of value $(0, 34.558, 34.558, 34.558)$.

efficiently navigate through a sequence of tasks that are complex and high dimensional. Note that we build our method to deal with this sequence of tasks on top of our empirically proven method on a real-robotic pendulum swing-up that successfully captures the degree of success through its generated data-driven uncertainty.

Chapter 5

Results

In this chapter, we walk the reader through how well our methodology worked on experiments that we had set up. In this chapter, we will first show that our approach is more scalable and hence has the potential to do well even in high-dimensional tasks. We then evaluate whether the confidence of our policy as its predictive uncertainty is a good proxy for task success. Then, we will show how we can leverage the confidence to devise any fallback policy when the situation in hand in non-generalizable, in particular we show how to reduce the number of requests for demonstrations when the fallback strategy is to actively train from demonstrations on the non-generalizable situation that is captured by the confidence metric of the controller. We finally investigate the effect of the main tunable parameters where these tunable parameters can tune the degree of conservativeness of our controller.

5.1 BNNs outperform GPs in high-dimensional tasks

We compare our BNN based proposed mechanism with GP in figure 5.1. Although GPs can do as well as BNNs in lower-dimensional domains, they fare significantly worse in higher-dimensions. In contrast, the number of input dimensions has an insignificant effect on the BNN's ability to learn. This is critical in most of the interesting real-world problems as they are high-dimensional. The hyperparameter values used in this experiment BBB is reported in table 5.1 and the initial values of trainable GP parameters is listed in table 5.2.

Settings	Swimmer	Sliding Block
Temporal Window size(k)	1	2/5
Mini-Batches	20	20
Activation Unit	RELU	RELU
Optimizer	Adam	Adam
Learning Rate	0.001	0.001
Hidden Units	[90, 30, 10]	[90, 30, 10]
M	25	25
β	0.01	0.01
Prior	$\mathcal{N}(0, 0.4)$	$\mathcal{N}(0, 0.4)$

Table 5.1 Hyperparameters values of BBB used in our experiments to compare ability of BNNs of GPs to scale up to higher dimensions.

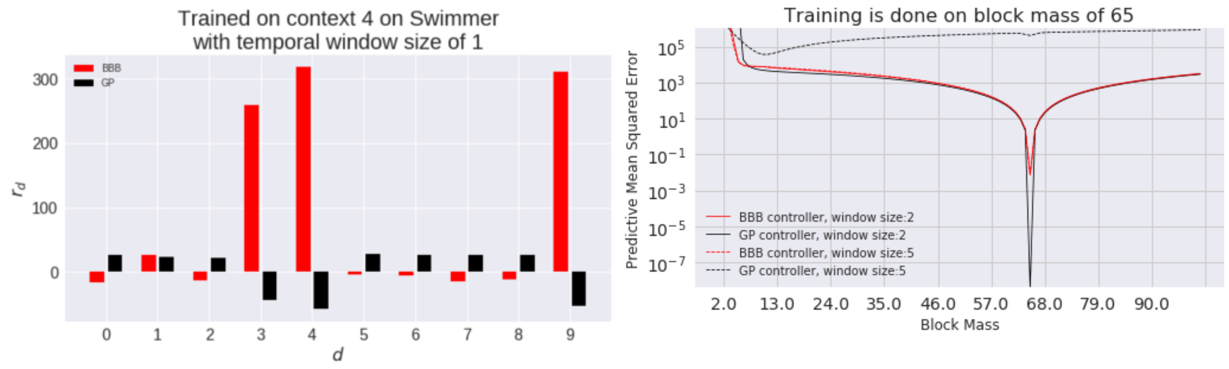


Fig. 5.1 GPs do not do as well as BBB when the number of input dimensions increases. Left figure shows how the performance of GP deteriorates with the increase in dimensions where the number of input dimensions corresponding to window sizes of 2 and 5 is 6 and 18 respectively. As a result, they never work with tasks that are already high dimensional (right figure) even with a temporal window size of 1, let alone use of deeper histories.

Settings	Values
Kernel	RBF
Kernel Variance	1.0
Kernel Lengthscales	0.01*Std($Data$)
Likelihood Variance	0.0001

Table 5.2 We use the same initial values of trainable parameter of GPs in both Swimmer and sliding block on ice in our experiments to compare ability of BNNs of GPs to scale up to higher dimensions.

5.2 Uncertainty as a predictor of success

In our work, the predictive uncertainty σ_d is used to detect and possibly preempt the execution on contexts the policy is unlikely to do well on. To test whether σ_d is indeed a good predictor for success, we compared the obtained σ_d and its corresponding empirical episodic reward r_d. We performed this experiment on the physical Furuta pendulum. Although the imitation learner does not attempt to optimize the reward, the demonstration policy does. Thus, high rewards are indicative of the learner closely following the demonstrator.

Figure 5.2 shows that σ_d is indeed related directly to the performance measurement r_d, so it can be used to predict whether the learner will do well. In particular, if the learner has been trained on contexts that are similar to the new context, it will tend to have a low predictive uncertainty and generalize well. If the new context is quite different, the policy will tend to have a high predictive uncertainty, and instead of attempting to perform the new context, the learner will first solicit new demonstrations for this context and then re-trains its policy on the full data-set. The hyperparameter values are listed in table 5.3. Videos for the corresponding can be found at `https://goo.gl/aCaHir`.

Settings	Values
k	2
t_{start}	0
#Mini-Batches	20
Activation Unit	RELU
Optimizer	Adam
Learning Rate	0.001
Hidden Units	50, 25, 10
M	8
β	0.01
Prior	$\mathcal{N}(0, 0.4)$

Table 5.3 Hyperparameters used in the robotic pendulum swing up experiment to show that task success is captured by the BBB.

5.3 Data efficiency by deciding when to query

One way of validating if our proposed active learner invokes the fallback strategy effectively is by comparing the number of requests it makes for context-specific demonstrations with

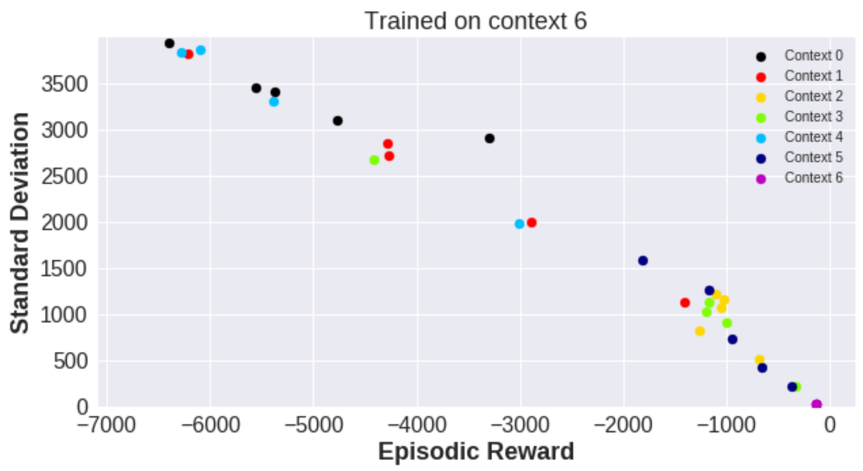


Fig. 5.2 For each swing-up task context, σ_d and r_d obtained during five independent runs are plotted against each other. Note the strong relationship between these quantities: higher σ_d is correlated with lower r_d. Thus, σ_d can be used as a predictor for task success. Training was performed on task context 6 in this case.

a *naive baseline* that solicits such context-specific demonstrations on every context it faces and a *random baseline* that never seeks any demonstration. We do two kinds of analysis for evaluating our active learner:

1. **Active LfD using predictive uncertainty sanity check:** Can the learner effectively ground its adaptive threshold in tandem with generating predictive uncertainty in any situation in order to correctly predict whether the controller can generalize to that context?

2. **Data-efficiency and tuning learner conservativeness:** How judiciously the learner seeks demonstration and how does it affect the overall performance with respect to the *naive* and *random* baselines?

Answering these questions entails gathering data and evaluating policies over multiple task contexts. To keep experiments doable, we will do so in simulation rather than on the real robot system. All hyperparameter values used in the experiments described in this subsection are reported in table 5.4

5.3.1 Active LfD using predictive uncertainty sanity check

We first do a sanity check to see if our active learner has the ability to adapt its threshold to actively request context-specific demonstrations when it is most necessary. For this we purposefully arrange the sequence of contexts in a way that the first two contexts are similar and different from the third. While the first two demonstration request for *naive* controller would be on the first and second context, our active learner's first two requests should correspond to the first and the third context. Figure 5.3 shows that our proposed active learner achieves higher cumulative rewards over all the contexts in D than the *naive* controller after 2 requests for demonstrations which is a reflection of the fact that our active learner learns to perform well across a larger number of contexts than a naive controller with the same number of requests for demonstrations. The exact details of the ordering of the task can be found in the provided code.

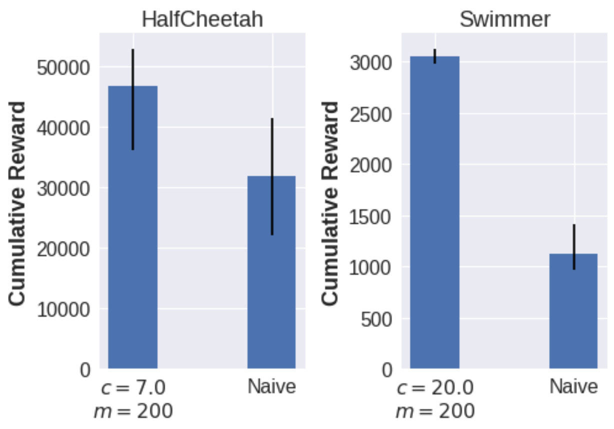


Fig. 5.3 Cumulative reward over all contexts in D after making 2 requests for demonstrations. Our active learner outperforms the naive learner that requests demonstrations in every context. Bars show average rewards over 10 episodes with error bars showing minimum and maximum values.

In order to visualize how our controller handles the familiar and non-familiar contexts, we randomize the ordering to contexts, then let the controller solve them sequentially while validating its performance on a diverse variety of contexts, and visualize the progression as snapshots at a few consecutive iterations of facing contexts. Figure 5.4 visualizes a portion from such a progression of our proposed mechanism on the Swimmer domain. At the first subfigure, the controller is finds itself having low confidence on all shown contexts except context 3 (predictive uncertainties on these contexts exceeded the adaptive thresholds). So our actively learning controller trains on demonstrations specific to context 4 after facing it. As a result, the respective uncertainty goes down in the next subfigure. Another interesting aspect to note here is how the predictive uncertainty on context 9 dropped despite the fact that controller never experienced this context earlier(second subfigure). This is because the controller is able to generalize to context 9 from the training performed on context 4. At the next step, the controller seeks for demonstrations specific to context 0 as its uncertainty exceeding the adaptive threshold again. Also note that the value of adaptive threshold adapts itself post training on any context. Note that generating the plots by validating the controller on all the diverse set of contexts after facing any context is done only for gaining insight into the working of the mechanism. This is in no way a non-compliance with our described problem setup (3.1) where the controller faces tasks one at a time sequentially in tandem with making decisions on whether or not to seek task specific demonstrations on the current task based on the value of the obtained predictive uncertainty.

5.3.2 Data-efficiency and tuning learner conservativeness

After validating the potential of our active learner to make informed decisions about where to request for demonstrations, we now randomize the ordering of contexts in D to analyze data requirements for successfully solving all contexts in D. Figure 5.5 shows various configurations of our active learner using combinations of c and m to solve all D contexts. With a moderate number of requests for demonstrations, our proposed agents learn to solve the task adequately - appreciably better than *random* and close to the *naive* learner. hence attaining data-efficiency (minimizing the number of, possibly expensive, demonstrations). Note that the *naive* learner here solicits demonstrations on all the contexts it faces.

We would also like to note the effect of hyperparameters c (threshold scale) and m (amount of smoothing) on the conservativeness (asking for more demonstrations in case of doubt). Note that lower c and lower m leads to the agent asking for more demonstrations. These additional demonstrations offer greatly diminished returns, as the average reward attained did not go up appreciably as a result.

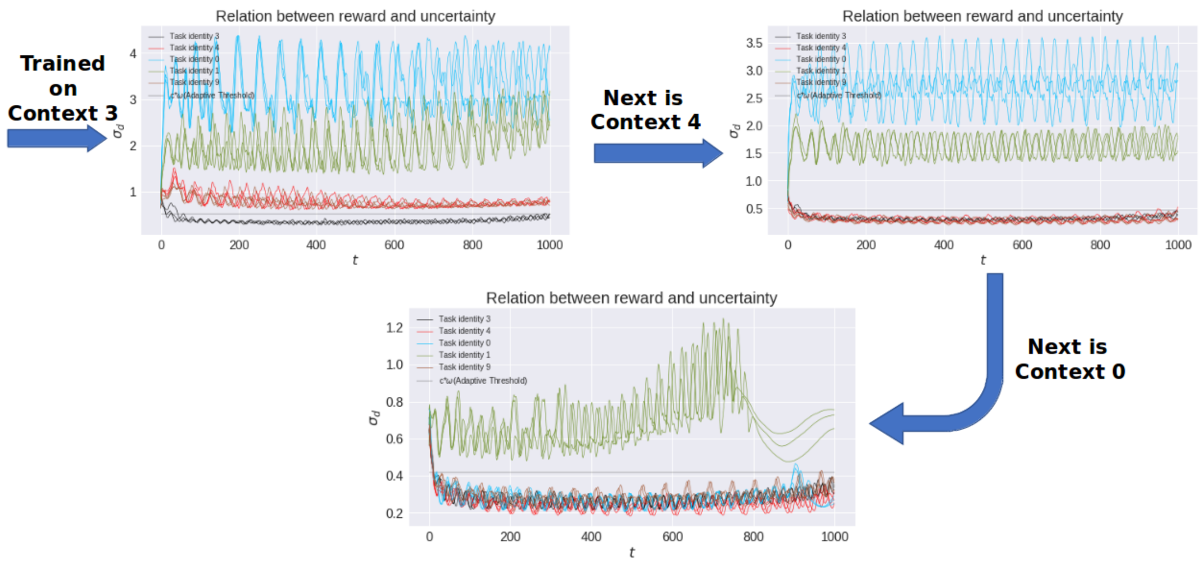


Fig. 5.4 Visualization of how the predictive uncertainties and adaptive thresholds change after facing contexts 4 and 0 sequentially. The controller decides to train on these contexts as their predictive uncertainties exceeded the adaptive thresholds. As a result the predictive uncertainties dropped on these contexts and other controller generalizable context (context 9) while the controller also adapts its adaptive threshold. The controller then goes on to request demonstrations specific to context 0 in the next step. Note that, the set-up where all the tasks are available for validation at every step, is purposefully set to get insights into the working of our methodology. All our other experiments are done in a set-up where the controller has no idea about the tasks other than the task it is dealing with at any moment.

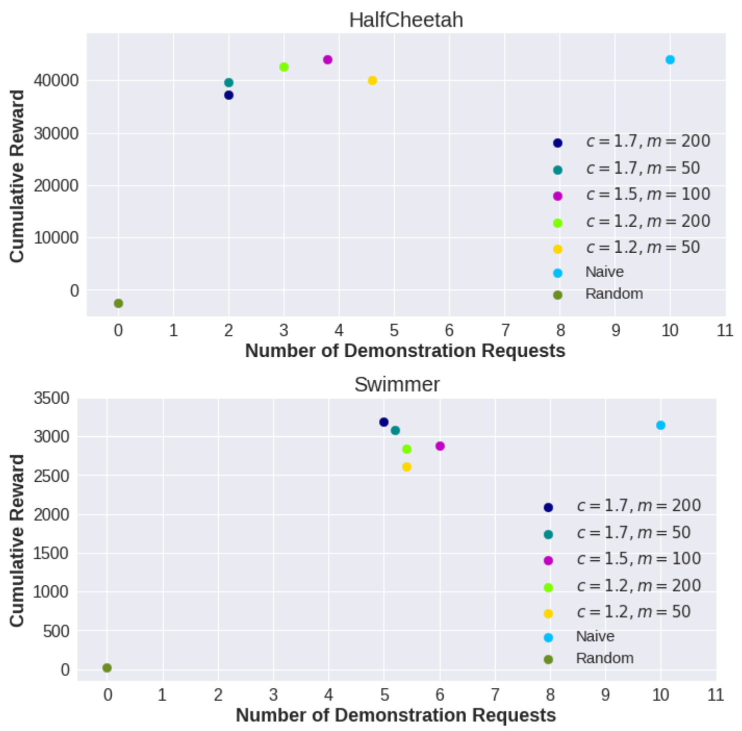


Fig. 5.5 The plots above show that lower c and m leads to seeking more context-specific demonstrations or more conservative behavior without much gain in reward. The plots also show that performance close to a *naive* learner that seeks demonstrations on every context, can be obtained by lesser number of such requests. The results were obtained by averaging over 5 simulation runs with different randomized orderings of contexts in D.

Settings	Values
k	2
t_{start}	200
#Mini-Batches	20
Activation Unit	RELU
Optimizer	Adam
Learning Rate	0.001
Hidden Units	90, 30, 10
M	25
β	0.01
Prior	$\mathcal{N}(0, 0.4)$

Table 5.4 Hyperparameters values used in both HalfCheetah and Swimmer experiments to show the ability of our actively learning controller to be data-efficient.

Chapter 6

Conclusion and Future Work

In this work, we analyzed integration of probabilistic methods in a Learning from Demonstration based technique called Behavioral Cloning through a Bayesian Neural Network technique called Bayes-by-Backprop.

Unlike deterministic neural networks which only yield point estimates, Bayesian Neural Network allows to model of predictive distribution scalably. This predictive distribution allows to gauge the confidence of the controller in its predictions. Through our experiments on a real robotic pendulum swing up and a simulated double integrator set up, we showed that given demonstrations of optimal behavior, it is possible to have a cheap estimate of the confidence of the controller to do well in a given situation. We also showed that such a technique to obtain a confidence measure scales up easily even in the high dimensional tasks. Furthermore, we showed that this confidence measure can be exploited to learn in a data-efficient way. By only querying for demonstrations where the confidence is low, the amount of demonstrations needed is reduced. In other words, our proposed confidence measure can help in identifying satisfactorily transferable and non-transferable tasks. We demonstrate this on MuJoCo tasks of HalfCheetah and Swimmer. Such a data-efficient technique has a lot of utility in areas where demonstrations are painstaking to obtain and hence have to be sought judiciously while ensuring that performance is not compromised.

This work motivates to further develop theoretical bounds and guarantees when an imitation learner is given a probabilistic treatment, especially when an interactive demonstrator is available. Another interesting line of work would be to find better fallback strategies that can integrate well with our proposed probabilistic Behavioral Cloning, such as a back-up controller or gracefully coming to a stop. Another useful addition could be an way to automatically adjust to suboptimal demonstrations.

Note that one downside of our methodology is that it is designed in a way that it works where either there is no *covariate shift* or if the *covariate shift* has no effect on the performance in the context on which the controller has trained itself. Hence, another line of work would be under the circumstances where there could be a possibility of *covariate shift* even on the trained context. An interesting approach in this direction would be to analyze the effect of probabilistic treatment of BC algorithms that have an access to an interactive demonstrator such as in DAGGER ([4]) or AggreVaTe ([103]). These algorithms are specifically designed to reduce the effect of covariate shifts when doing BC.

References

[1] C. A. Ellwood, "The theory of imitation in social psychology," *American Journal of Sociology*, vol. 6, no. 6, pp. 721–741, 1901.

[2] K. Yamane and J. Hodgins, "Simultaneous tracking and balancing of humanoid robots for imitating human motion capture data," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 2510–2517, IEEE, 2009.

[3] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs. sourceforge. net*, vol. 4, p. 6, 2000.

[4] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," in *AISTATS*, Citeseer, 2011.

[5] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," in *ICML*, 2015.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," *Proceedings of Robotics: Science and Systems. Pittsburgh, Pennsylvania*, 2018.

[8] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," Tech. Rep. 1606.06565, CoRR, 2016.

[9] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[10] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[11] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," Tech. Rep. 1606.01540, CoRR, 2016.

[13] H. Over and M. Carpenter, "The social side of imitation," *Child Development Perspectives*, vol. 7, no. 1, pp. 6–11, 2013.

[14] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 3931–3936, IEEE, 2009.

[15] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proceedings of the 25th international conference on Machine learning*, pp. 144–151, ACM, 2008.

[16] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, pp. 305–313, 1989.

[17] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *Advances in neural information processing systems*, pp. 1087–1098, 2017.

[18] J. Dapena, "Mechanics of translation in the fosbury-flop.," *Medicine and Science in Sports and Exercise*, vol. 12, no. 1, pp. 37–44, 1980.

[19] U. Syed and R. E. Schapire, "A reduction from apprenticeship learning to classification," in *Advances in neural information processing systems*, pp. 2253–2261, 2010.

[20] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *AISTATS*, pp. 661–668, 2010.

[21] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[22] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," *arXiv preprint arXiv:1703.09327*, 2017.

[23] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine learning*, vol. 75, no. 3, pp. 297–325, 2009.

[24] A. Beygelzimer, V. Dani, T. Hayes, J. Langford, and B. Zadrozny, "Error limiting reductions between classification tasks," in *Proceedings of the 22nd international conference on Machine learning*, pp. 49–56, ACM, 2005.

[25] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning.," in *ICML*, pp. 663–670, 2000.

[26] C. Sammut and G. I. Webb, eds., *Inverse Optimal Control*, pp. 554–554. Boston, MA: Springer US, 2010.

[27] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," *Predicting structured data*, vol. 1, no. 0, 2006.

[28] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *AAAI*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[29] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *European Conference on Computer Vision*, pp. 201–214, Springer, 2012.

[30] F. Shkurti, N. Kakodkar, and G. Dudek, "Model-based probabilistic pursuit via inverse reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7804–7811, IEEE, 2018.

[31] M. Wulfmeier, P. Ondruska, and I. Posner, "Deep inverse reinforcement learning," *CoRR, abs/1507.04888*, 2015.

[32] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[33] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning*, pp. 49–58, 2016.

[34] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

[35] J. Choi and K.-E. Kim, "Inverse reinforcement learning in partially observable environments," *Journal of Machine Learning Research*, vol. 12, no. Mar, pp. 691–730, 2011.

[36] V. Karasev, A. Ayvaci, B. Heisele, and S. Soatto, "Intent-aware long-term prediction of pedestrian motion," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 2543–2549, IEEE, 2016.

[37] C. L. Baker, R. Saxe, and J. B. Tenenbaum, "Action understanding as inverse planning," *Cognition*, vol. 113, no. 3, pp. 329–349, 2009.

[38] D. Hadfield-Menell, S. J. Russell, P. Abbeel, and A. Dragan, "Cooperative inverse reinforcement learning," in *Advances in neural information processing systems*, pp. 3909–3917, 2016.

[39] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg, "Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning," in *Robotics Research*, pp. 91–110, Springer, 2018.

[40] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[41] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, "Active preference-based learning of reward functions," in *R:SS*, 2017.

[42] C. Basu, M. Singhal, and A. D. Dragan, "Learning from richer human guidance: Augmenting comparison-based learning with feature queries," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pp. 132–140, ACM, 2018.

[43] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, pp. 4299–4307, 2017.

[44] A. Irpan, "Deep reinforcement learning doesn't work yet." `https://www.alexirpan.com/2018/02/14/rl-hard.html`, 2018.

[45] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," *arXiv preprint arXiv:1802.01557*, 2018.

[46] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1134–1141, IEEE, 2018.

[47] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[48] G. Cybenko, "Approximations by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 183–192, 1989.

[49] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[50] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*, pp. 63–71, Springer, 2004.

[51] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, 2016.

[52] T. D. Bui, J. M. Hernández-Lobato, Y. Li, D. Hernández-Lobato, and R. E. Turner, "Training deep gaussian processes using stochastic expectation propagation and probabilistic backpropagation," *arXiv preprint arXiv:1511.03405*, 2015.

[53] T. P. Minka, "Expectation propagation for approximate bayesian inference," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pp. 362–369, Morgan Kaufmann Publishers Inc., 2001.

[54] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *International Conference on Machine Learning*, pp. 1861–1869, 2015.

[55] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[56] D. J. MacKay, "A practical bayesian framework for backpropagation networks," *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[57] P. Jylänki, A. Nummenmaa, and A. Vehtari, "Expectation propagation for neural networks with sparsity-promoting priors," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1849–1901, 2014.

[58] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian model-agnostic meta-learning," in *Advances in Neural Information Processing Systems*, pp. 7342–7352, 2018.

[59] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks," *arXiv preprint arxiv.1605.09674*, 2016.

[60] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[61] M. J. Wainwright, M. I. Jordan, *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[62] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems*, 2015.

[63] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *COLT*, pp. 5–13, 1993.

[64] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *ICML*, pp. 465–472, 2011.

[65] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, 2016.

[66] J. C. Gamboa Higuera, D. Meger, and G. Dudek, "Synthesizing neural network controllers with probabilistic model based reinforcement learning," *arXiv preprint arXiv:1803.02291*, 2018.

[67] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, pp. 1050–1059, 2016.

[68] Z. Wang and M. E. Taylor, "Improving reinforcement learning with confidence-based demonstrations," in *IJCAI*, 2017.

[69] M. S. Gashler and S. C. Ashmore, "Training deep fourier neural networks to fit time-series data," in *International Conference on Intelligent Computing*, pp. 48–55, Springer, 2014.

[70] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," *Journal of intelligent and robotic systems*, vol. 31, no. 1-3, pp. 91–103, 2001.

[71] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus, "Training recurrent neural networks: Why and how? an illustration in dynamical process modeling," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 178–184, 1994.

[72] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in neural information processing systems*, pp. 155–161, 1997.

[73] L. B. Godfrey and M. S. Gashler, "Neural decomposition of time-series data for effective generalization," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 2973–2985, 2018.

[74] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018.

[75] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio.,” in *SSW*, p. 125, 2016.

[76] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” *arXiv preprint arXiv:1404.2188*, 2014.

[77] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” *arXiv preprint arXiv:1612.08083*, 2016.

[78] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, “A convolutional encoder model for neural machine translation,” *arXiv preprint arXiv:1611.02344*, 2016.

[79] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[80] K. Zhou and J. C. Doyle, *Essentials of robust control*, vol. 104. Prentice hall Upper Saddle River, NJ, 1998.

[81] L. Ljung, “System identification,” *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2001.

[82] B. Settles, “Active learning literature survey,” Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[83] A. L. Blum and P. Langley, “Selection of relevant features and examples in machine learning,” *Artificial intelligence*, vol. 97, no. 1-2, pp. 245–271, 1997.

[84] S. Chernova and M. Veloso, “Interactive policy learning through confidence-based autonomy,” *J. Artificial Intelligence Research*, vol. 34, no. 1, p. 1, 2009.

[85] D. H. Grollman and O. C. Jenkins, “Dogged learning for robots,” in *ICRA*, pp. 2483–2488, 2007.

[86] S. Tellex, R. A. Knepper, A. Li, T. M. Howard, D. Rus, and N. Roy, “Assembling furniture by asking for help from a human partner,” in *Collaborative manipulation workshop at human–Robot interaction*, 2013.

[87] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowledge Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.

[88] A. Torralba and A. A. Efros, “Unbiased look at dataset bias,” in *CVPR*, pp. 1521–1528, IEEE, 2011.

[89] X. Zhu, "Semi-supervised learning literature survey," Tech. Rep. 1530, University of Wisconsin—Madison, 2005.

[90] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *ICLR*, 2016.

[91] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[92] W. S. Klug, M. R. Cummings, *et al.*, *Essentials of genetics.* No. Ed. 2, Prentice-Hall Inc., 1996.

[93] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," Tech. Rep. 1610.04286, CoRR, 2016.

[94] J. C. Gamboa Higuera, D. Meger, and G. Dudek, "Adapting learned robotics behaviours through policy adjustment," in *ICRA*, pp. 5837–5843, 2017.

[95] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," Tech. Rep. 1611.02779, CoRR, 2016.

[96] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *arXiv preprint arXiv:1703.03400*, 2017.

[97] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

[98] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," *arXiv preprint arXiv:1511.06342*, 2015.

[99] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[100] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7167–7176, 2017.

[101] K. Furuta, M. Yamakita, and S. Kobayashi, "Swing-up control of inverted pendulum using pseudo-state feedback," *Journal of Systems and Control Engineering*, vol. 206, pp. 263 – 269, 1992.

[102] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” Tech. Rep. 1707.06347, CoRR, 2017.

[103] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, “Deeply aggregated: Differentiable imitation learning for sequential prediction,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3309–3318, JMLR. org, 2017.