A Dynamic-Bayesian Network Framework for Modeling and Evaluating Learning from Observation

Santiago Ontañón\textsuperscript{a}, José L. Montaña\textsuperscript{b}, Avelino J. Gonzalez\textsuperscript{c}

\textsuperscript{a}Drexel University, Philadelphia, PA, USA
\textsuperscript{b}University of Cantabria, Santander, Spain
\textsuperscript{c}University of Central Florida, Orlando, FL, USA

Abstract

Learning from Observation (LfO), also known as Learning from Demonstration, studies how computers can learn to perform complex tasks by observing and thereafter imitating the performance of a human actor. Although there has been a significant amount of research in this area, there is no agreement on a unified terminology or evaluation procedure. In this paper, we present a theoretical framework based on Dynamic-Bayesian Networks (DBNs) for the quantitative modeling and evaluation of LfO tasks. Additionally, we provide evidence showing that: (1) the information captured through the observation of agent behaviors occurs as the realization of a stochastic process (and often not just as a sample of a state-to-action map); (2) learning can be simplified by introducing dynamic bayesian models with hidden states for which the learning and model evaluation tasks can be reduced to minimization and estimation of some stochastic similarity measures such as crossed entropy.

Keywords: Learning from Observation, Dynamic Bayesian Networks

1. Introduction

Learning by watching others do something is a natural and highly effective way for humans to learn. It is also an intuitive and highly promising avenue for machine learning. It provides a way for machines to learn how to perform tasks in a more natural fashion. For many tasks, learning from
observation is more natural than providing static examples that explicitly contain the solution, as in the traditional supervised learning approach. It is also easier than manually creating a controller that encodes the desired behavior. Humans typically just perform the task and trust that the observer can figure out how to successfully imitate the behavior.

Although there has been a significant amount of research in learning from observation (LfO), there is no agreement on a unified terminology. Works reported in the literature also refer to learning from demonstration, learning by imitation, programming by demonstration, or apprenticeship learning, as largely synonymous to learning from observation. In learning from demonstration, a human purposely demonstrates how to perform a task or an action, expressly to teach a computer agent how to perform the same task or mission. We consider learning from demonstration to be a specialization of LfO and define the latter as a more general learning approach, where the actor being observed need not be a willing participant in the teaching process.

Specifically, the problem we are trying to address in this paper is the lack of a unified framework to understand existing work in LfO, as well as the lack of standard evaluation metrics to assess the performance of LfO algorithms (which are typically evaluated using metrics designed for standard supervised learning). To that purpose, we present an unified framework for learning from observation based on Dynamic Bayesian Networks (DBNs) [28]. We provide both an intuitive description of the framework, as well as a formal statistical model of LfO. The main contributions of this paper, are:

- A formal statistical model of LfO, that provides a unified vocabulary and theoretical framework for LfO.
- A taxonomy of the different behaviors that can be learned through LfO.
- An explicit formulation of the difference between supervised learning and LfO algorithms. This is important because in most LfO work, standard supervised algorithms (like neural networks, or nearest-neighbor classifiers) are used, yet there are many behaviors to be learned through LfO for which those algorithms are not appropriate.
- A proposal for standard evaluation metrics for agents trained through LfO (currently lacking from the literature). Our framework makes explicit the reason for which standard metrics, such as classification accuracy, do not properly reflect how well an LfO algorithms can learn
complex tasks in some situations. We describe the reasons for this, and propose an alternative evaluation approach based on the Kullback-Liebler divergence.

The remainder of this paper is organized as follows. Section 2 briefly summarizes previous research in the field. After that, Section 3 introduces a common framework and vocabulary for learning from observation, including a statistical formalization of the problem. Section 4 focuses on evaluation metrics for LfO algorithms. Finally, Section 5 presents an empirical validation of two of our claims: a) supervised learning algorithms are not appropriate for some LfO behaviors, and b) our proposed evaluation metric is more accurate than the typical metrics used in the literature of LfO.

2. Learning from Observation Background

LfO is a subfield of machine learning that studies how to learn behavior from observation or demonstration. This has many practical applications. A first set of interesting applications of LfO concern allowing machines to learn how to perform complex behaviors that would be difficult to manually program (e.g. driving vehicles [35], robotics [1], playing videogames [32]). Another interesting set of applications involves understanding and analyzing behavior, i.e., using LfO to generate models of certain behaviors of interest in order to compare them, cluster them or understand them [10].

In this paper, we argue that LfO is fundamentally different from traditional supervised, unsupervised and reinforcement learning approaches. Clearly, LfO differs from unsupervised learning because the examples (demonstrations) contain an implicit indication of correct behavior. Furthermore, LfO clearly differs from reinforcement learning because LfO learns from a collection of traces (or trajectories), rather than from trial and error through a reinforcement signal (although, as we show later, some reinforcement learning approaches use LfO as a way to guide the learning process, e.g., [22]).

LfO can be more readily likened to supervised learning, but two key differences exist. First, in LfO the learning examples are time-based, continuous and not easily separable throughout the duration of the exercise. Furthermore, no explicit linkage between cause and effect is provided, but must be extracted automatically by the learning algorithm. The cause of an action might very well be something perceived in a past instant of time, rather than in the current perceptual state. Thus, the form of supervised learning that is more related to LfO is that of sequential learning [11].
A second key difference is on what needs to be learned. Supervised machine learning techniques (including sequential ones) focus on learning models that minimize the prediction error, i.e., they learn to predict the output of the learning task given the input, on average. However, that is not the goal in LfO. Consider an agent trying to learn by observing the behavior of an actor who, while driving a car, chooses a different random speed each minute with a mean of 100kph and a certain variance. A standard supervised learning method would learn to predict that the speed should always be 100kph (because that is the value that yields minimum prediction error). Therefore, an LfO agent should learn that the speed must be changed when appropriate with a particular variance. LfO aims to replicate the behavior of the actor, rather than to minimize the prediction error. For that reason, supervised learning techniques for LfO have to be employed with care, and a different class of algorithms is required in the general case. However, as our previous work shows, some types of learning from observation tasks can be addressed with supervised learning (for example, as we explain later, when learning state-less deterministic behavior, minimizing prediction error is equivalent to replicating behavior).

Work in learning from observation can be traced back to the early days of AI. Bauer [3] proposed in 1979 to learn programs from example executions, which basically amounts to learning strategies to perform abstract computations by demonstration. This form of learning has been especially popular in robotics [24]. Other early mentions of learning from observation come from Michalski et al. [25] who define it merely as unsupervised learning, and from Pomerleau [35], who developed the ALVINN system that trained neural networks from observation of a road-following automobile in the real world.

More recent work on the more general LfO subject came from Sammut et al [38], Sidani [40] and Gonzalez et al [15]. Fernlund et al. [12] used learning from observation to build agents capable of driving a simulated automobile in a city environment. The neural network approach to learning from observation has remained popular, and contributions are still being made, such as the work of Moriarty and Gonzalez [26], in the context of computer games.

In robotics, learning from demonstration has been extensively used to implement human behavior in humanoid robot movements. Bentivegna and Atkeson [5] used learning from demonstration to teach a humanoid robot to play air hockey using a set of action primitives, each describing a certain basic action. Chernova and Veloso [7, 8] studied the problem of multi-robot
learning from demonstration where a group of agents was shown how to collaborate with each other. Other important work was reported by Schaal [39], Atkeson et al. [2] and Argall et al. [1] among many others.

Könik and Laird [19] studied learning from observation with the SOAR system by using inductive logic programming techniques.

A theoretical approach to LfO is that of Khardon [17], where he proposed to use a systematic algorithm that enumerates all the possible finite-state machines (given the inputs and outputs of a given domain), and rank them according to their probability of achieving the goal, as well as to how consistent they are with the behavior observed from the expert. An important difference between Khardon’s work and most work on LfO is that Khardon assumed that the learning agent has access to a description of the goal (i.e., the learning agent knows, during learning, if a particular behavior would achieve the goal or not in a given scenario). Most work on LfO assumes that the only form of input are samples of behavior from the expert, without any explicit description of the goal to be achieved.

Other significant work done under the label of learning from demonstration has emerged recently in the case-based reasoning (CBR) community. Floyd et al. [13] presented an approach to learn how to play RoboSoccer by observing the play of other teams. Ontaño and associates [31, 32, 30] used learning from demonstration in the context of case-based planning, applied to real-time strategy games. Rubin and Watson [37] used LfO for creating a Poker-playing agent. And LaMontagne et al. [21] studied techniques based on conditional entropy to improve case acquisition in CBR-based LfO.

The main difference between the work based on CBR and the previous work presented in this section is that CBR methods are related to lazy machine learning techniques that do not require any form of generalization during learning. In CBR, thus, learning becomes a pure memorization of new cases, and any kind of generalization is delayed until problem solving time.

Additionally, Inverse Reinforcement Learning (IRL) [29] is closely related to learning from observation, where the focus is on reconstructing the reward function given optimal behavior (i.e., given a policy, or a set of trajectories). One of the main difficulties in IRL is that there might be different reward functions that correspond to the observed behavior, and heuristics need to be devised to bias the learning towards the reward functions of interest. The paradigm of inverse reinforcement learning has recently received increased attention for its applications to LfO [18, 23, 41]. The key difference between the IRL approach to LfO and the supervised learning approaches mentioned
above is that IRL tries to learn the goal (reward function) of the expert, to then use standard reinforcement learning to learn optimal behavior to achieve such goal, while the supervised learning approaches to LfO aims at learning to directly predict the actions of the expert from the observed behavior. These approaches, however, all ignore the fact that the expert might have internal state, e.g., memory of past events.

Finally, the approaches that are most related to the model presented in this paper, are those based on Markov Decision Processes (MDP). For example, Dereszynski et al. [10] describe an approach based on Hidden Markov Models to learn a model of the behavior of agents in real-time strategy games. Pentland and Liu [34] used Hidden Markov Models (HMMs) to infer the internal state of an automobile driver.

An in-depth review of the area is beyond the scope of this paper (the interested reader is referred to [1] for a recent overview of the relevant literature). Nevertheless, we can see that while a significant amount of work has been on-going over the last 20 years, LfO does not enjoy a modicum of formalization, or even of agreement in terminology. For example, in the in-depth recent overview by Argall et al. [1], or in the recent formalization by Billing and Hellström [6], they present a formalization that only covers learning Markovian processes from observation and do not cover the whole spectrum of existing LfO techniques, as we show later. In this paper, we hope to provide a significant step towards this missing formalization.

3. A Framework for Learning from Observation

In this section, we describe a framework that seeks to unify previous work in the area. It does so using the concept of stochastic processes [33]. Additionally, we define three different types of problems (levels) where LfO could be applicable, and describe the above process according to the specific needs of each of these levels.

3.1. Some Definitions and Discussion

Let us start by introducing the different elements appearing in LfO (illustrated in Figure 1.a), and how are they represented in our framework:

- **Task, T**: a task to be learned, which can be either achieving a condition (such as building a tower, or defeating an enemy), maintaining some process over time (such as keeping a car on the road), or maximizing
some value. In general, $T$ can be represented as a reward function, which has to be maximized.

- **Environment**, $E$: simulated or real, where the task is performed.
- **Actor** (or trainer or expert), $C$: who performs the task $T$ in the environment (there can in principle be more than one actor).
- **Learning agent**, $A$: whose goal is to learn how to achieve $T$ in the environment $E$ by observing $C$. Some works in the literature, such as [7] or [?] focus on multiagent learning from observation; in this paper, we will focus on single-agent LfO.

In learning from observation, the learning agent $A$ first observes one or several actors performing task $T$ in the environment $E$, and records their behavior in the form of *traces*. Then, those traces are used by $A$ to learn how to perform $T$ via a learning algorithm. The environment in which the learning agent observes the actor perform the task need not be the same one in which the agent later performs the task. For instance, in the context of a computer game, the learning agent might observe the actor play the game in a particular map, and then try to perform the actions in a different map. Moreover, most LfO work assumes that the agent does not have access to a description of $T$ during learning, and thus, behavior must be learned purely by *unobtrusive observation* of the behavior of the expert.

Let $B_C$ be the *behavior* of an actor $C$. By behavior, we mean the control mechanism, policy, or algorithm that an actor or learning agent uses to determine which actions to execute over time. Our formalization is founded on
the concept that behavior can be modeled as a stochastic process, and the elements shown in Figure 1.a will be modeled as random variables. See, for instance, [33] for the theoretical background related to stochastic processes.

We assume that, in general, actor C might have non-observable internal state. For example, consider the Stratego game, a strategy game similar to Chess, but where the players do not see the types of the pieces of the opponent, only their locations. After certain movements, a player can temporally observe the type of one piece, and must remember this in order to exploit this information in the future. A person playing the game of Stratego might keep an internal record (in her memory) of the type of known pieces that the opponent might have on the board — information not directly observable. Because her behavior will be influenced by this memory, it is necessary to somehow include it in the model. Therefore, our model includes the following variables: $E_t$ represents the (hidden) state of the environment at time $t$, the learning agent observes the environment state via an input random variable $X_t$. The actions executed at time $t$ are represented by a control random variable $Y_t$. Finally, the internal state of the actor at time $t$ is a non-observable random variable $C_t$. We will use the following convention: if $X_t$ is a variable, then we will use a calligraphic $X$ to denote the set of values it can take, and lower case $x \in X$ to denote specific values it takes.

Depending on the type of the task to be learned, the actions of the actor can vary in nature. These could be atomic actions, durative actions, or a set of continuous control parameters. In our framework, we assume that the random variables $X_t$ and $Y_t$ are multidimensional variables that can be either continuous or discrete: that is, the set of values $X$, that $X_t$ can take is either $X = \mathbb{R}^p$, for some $p$, or some discrete set; analogously, either $Y = \mathbb{R}^q$ for some $q$, or $Y$ is a discrete set. Examples in which the environment is a continuous space and the actions are described inside a discrete space are found quite often in the literature (e.g. [35]).

The behavior $B_C$ of the actor $C$ can be interpreted, therefore, as a stochastic process $I = \{I_1, \ldots, I_n, \ldots\}$, with state space $\mathcal{I} = \mathcal{X} \times \mathcal{C} \times \mathcal{Y}$, where $I_t = (X_t, C_t, Y_t)$ is the random variable where $X_t$ and $Y_t$ represent respectively the input and output variables at time $t$, and $C_t$ represents the internal state of the actor at time $t$.

In a particular execution, a behavior $B_C$ is manifested as the series of actions that the actor executes over time, which we call a learning trace, or $LT$. A learning trace $LT = [(x_1, y_1), \ldots, (x_n, y_n)]$ observed by the learning agent $A$ can be seen as the realization – also trajectory or sample path – of
the stochastic process corresponding to the behavior of the actor $C$; where, however, the values $c_t$ of the non-observable variables $C_t$ have been omitted.

The pair of variables $X_t$ and $Y_t$ represent the observation of the learning agent $A$ at time $t$, i.e.: $O_t = (X_t,Y_t)$. Thus, for simplicity, we can write a learning trace as $LT = [o_1,...,o_n]$.

We also assume, as usual in stochastic process theory, that variables $I_t$ are defined over the same probability space $(\mathcal{I},F,\rho)$, where $\mathcal{I}$ is the nonempty state space, $F$ is the $\sigma$-algebra of events and $\rho$ is an unknown probability measure that governs the behavior of the observed actor.

Under this formalization, the LfO problem reduces to estimating the unknown probability measure $\rho$ from a set of traces, or trajectories, $\{LT_j : 1 \leq j \leq k\}$ of the stochastic process $I = \{I_t : t \in T\}$. How the probability distribution $\rho$ can be learned depends on the problem under consideration. We discuss this below starting in Section 3.2.

The task $T$ to perform is defined as a reinforcement signal $T(x,y)$, which assigns a reinforcement value $T(x,y) \in \mathbb{R}$ to each state/action pair $(x,y)$. When the learning agent observes the actor, it is assumed that the actor is trying to maximize $T$. Notice that a reinforcement signal is general enough to capture tasks such as “reaching a terminal state” (by having a positive reward in that state, and 0 in any other state), “staying in a certain set of states”, etc. However, in most work on LfO, only the actor has access to the definition of the task, which is unknown to the learning agent $A$. Thus, we can define the general problem of LfO as:

**Given:** An environment $E$, perceived through a set of input variables $x$, and acted upon by a set of control variables $y$, and a collection of learning traces $LT_1,...,LT_k$

**Learn:** A behavior $B$ that behaves in the same way as the actor does in the environment $E$, given the same inputs. For example, the work of Fernlund et al. [12] falls into this category. Given that no reward function $T$ is available in this case, the main challenge is is how to formally define “in the same way”, i.e. how to define performance metrics for LfO algorithms.

Moreover, measuring the success of a LfO agent is not trivial [1]. In supervised learning, a simple way is to leave some training examples out, and use them later as test. The equivalent in LfO would be to leave some learning
traces out, and use them to verify the learned behavior $B$. However, in LfO we have to compare behavior traces, which is not trivial. For example, if at some point of time $t$ two traces differ, it might be meaningless to compare the rest of the traces, since from that point on, they might completely diverge. Therefore, just counting percentage of actions correctly predicted is not a good approach.

Furthermore, if the behavior to be learned is stochastic, comparing individual actions would not be a good approach either. Imagine that an actor is demonstrating a behavior which consists of picking an action at random from the set $\{-1, 0, 1\}$ at each executing cycle. There are two learning agents, $A_1$ and $A_2$. After observing this actor, the behavior learned by $A_1$ consists of always executing 0 (which gives the best mean-square error prediction error); the behavior learned by $A_2$, however, is the correct one (randomly execute one of the three actions at each cycle). Now, if we were to evaluate the performance of these two agents using a standard loss function, like mean-square error using a leave-one-out method, agent $A_1$ would have an average error of about 0.66, and agent $A_2$ would achieve one of about 0.88. However, it was $A_2$ who properly learned the behavior of the actor. Therefore, simple measures such as classification accuracy or mean-square error are not enough to measure the performance of LfO algorithms and determine whether the learned behavior is similar or not to the observed behavior.

3.2. Levels of Learning from Observation

It is intuitively evident that not all LfO learning algorithms will work for learning all behaviors. Therefore, we next attempt to categorize the types of behaviors that can be learned through LfO. First, we must state that the general type of problems LfO addresses are those that involve learning some type of control function (i.e. a behavior). Such behaviors may be low level, such as motor skills, or higher level, such as strategic decision making.

Several factors determine the complexity of a behavior to be learned through LfO such as whether variables are continuous or discrete, whether the environment is fully observable or not, or whether the task requires memory of past events or not.

In particular two factors (whether the learning task requires generalization, and whether the learning task requires memory or not) define the three major families of behaviors that can be learned through different LfO algorithms, and which correspond to different models in our framework:
Level 1 - Strict Imitation: some behaviors do not require feedback from the environment, and thus require neither generalization nor memory. The learned behaviors are a strict function of time. For that reason, it does not matter whether the environment is observable or not. One can think of these behaviors as requiring open-loop control. Robots in factories are an instance of this type of learning, where the pieces are always in the exact same place each time.

Level 2 - Reactive Behavior: reactive behaviors correspond to input-to-action mappings, without requiring memory of past events. Generalization is desirable when learning these behaviors. Reactive behaviors are Markovian, because they can be represented as a mapping from the perceived state of the environment to the action, i.e., a policy. Learning how to play some video games such as pong and space invaders would fit this category.

Level 3 - Memory-based Behavior: when the current state of the world is not enough to determine which actions to execute, previous states might have to be taken into account. These behaviors are not a situation-to-action mapping, but might have internal state. For example, learning how to play a game like Stratego (described above) is a task of Level 3. Thus, a learning agent has to take into account past events in order to properly understand why the actor executed certain actions.

While level 1 problems can be considered trivial, learning behaviors of levels 2 or 3 from observation constitute the most interesting cases. Traditionally, most work on LfO has focused on level 2 (or has simply ignored the distinction between level 2 and 3 completely). Level 2 (and some special cases of level 3) is the highest that can be directly dealt with standard (i.e. non-sequential) supervised machine learning techniques.

It is important to distinguish level 2 from level 3, because the kind of algorithms required to learn those behaviors are inherently different. This distinction is typically not made in the LfO literature, and standard supervised learning algorithms or reinforcement learning algorithms have been and are used to learn tasks of level 3. Some restricted forms of level 3 behaviors can be reduced to level 2 behaviors by using techniques such as sliding windows [11], but in general this is not always possible.

Finally, it is also important to highlight that not all level 2 behaviors are learnable through any standard supervised learning method. This is because
most supervised learning algorithms focus on learning average behavior, i.e. they focus on minimizing prediction error. However, if the behavior to be learned has a stochastic component, the LfO algorithm must also learn such stochastic component. Therefore, in general, LfO requires the use of learning algorithms that can learn probability distributions, instead of just functions that minimize the prediction error.

3.3. A Dynamic Bayesian Network model of Learning from Observation

The probability distribution $p$ that governs the behavior of a given agent can be studied using *Dynamical Bayesian Networks*. A Bayesian Network (BN) is a modeling tool that represents a collection of random variables and their conditional dependencies as a directed acyclic graph (DAG). In this paper, we are interested in a specific type of BNs called *Dynamic Bayesian Networks* (DBN) [14], which can be used to model stochastic processes. In a DBN, the variables of the network are divided into a series of identical time-slices. A time-slice contains the collection of variables representing the state of the process at a specific instant of time. Variables in a time-slice can only have dependencies with variables in the same or previous time-slices.

A DBN can model the conditional dependencies between all the variables that play a role in LfO. Figure 1.b shows our proposal for such model, where actions $Y_t$ depend on the current perception, and on the internal state $C_t$, and where $C_t$ represents everything that the agent remembers from previous states. Figure 1.b also shows that in our model the state of the environment $E_t$ at time $t$ only depends on the previous state of the environment $E_{t-1}$ and of the previous action $Y_{t-1}$ (if there are other agents, they are considered as part of the environment). These dependencies are shown with dashed lines, because they are not relevant for the learning task. Note that, given $X$, the actions of the expert $Y$ are conditionally independent of the environment $E$. Therefore, for the purposes of learning from observation, we can simplify the Bayesian model, and obtain the model shown in Figure 2, which we will call the *LfO-DBN model*. The internal state of the actor at time $t$, $C_t$, depends on the internal state at the previous instant of time, $C_{t-1}$, the previous action $Y_{t-1}$ and of the current observation $X_t$. The action $Y_t$ depends only on the current observation, $X_t$, and the current internal state of the actor $C_t$.

Notice, that in some specific LfO settings, where the learning agent has a description of the task $T$ being executed by the actor, it might be interesting for the learning agent to learn a model of the environment (the dashed lines in...
Figure 2: The *LfO-DBN model*, grey variables are observable, white variables are hidden.

Figure 1.b), or at least the dependency between $Y_t$ and $X_{t+1}$. However, this is not relevant in the LfO setting studied in this paper, where $T$ is unknown.

Given the LfO-DBN model, if the learning agent wants to learn the behavior of the expert, it has to learn the dependencies between the variables $C_t$, $X_t$, and $Y_t$, i.e. it has to learn the following conditional probability distributions: $\rho(C_1)$, $\rho(Y_t|C_t, x_t)$, and $\rho(C_t|C_{t-1}, X_t, Y_{t-1})$.

If the learning agent is able to infer the previous conditional probability distributions, it can replicate the behavior of the expert, and thus, we could say that it has successfully learned its behavior from observation. In practice, the main difficulty of this learning task is that the internal state variable $C_t$ is not observable, and thus, if no assumption is made on its structure, in the general case, there is no direct way to learn its probability distribution.

Based on the LfO-DBN model, let us now present four different approaches to LfO based on making different assumptions over the internal state of the actor $C_t$, and on the variable relations, leading to increasingly complex learning algorithms. Each of these four approaches map intuitively to different complexity levels, from the ones we identified in Section 3.2.

### 3.4. Level 1 - Strict Imitation

In this first approach we assume the learning task is a strict imitation task, and thus the variables $Y_t$ do not depend on either $X_t$ or $C_t$. Therefore, they can be removed from the model. In this approach, the probability distribution of the $Y_t$ variables only depend on time (i.e. the distribution of variable $Y_{t_1}$ might be different from the one of $Y_{t_2}$ if $t_1 \neq t_2$).

If what we want is to reproduce the set of actions defined by a behavior trace $BT = [y_1, ..., y_n]$, we just consider $\rho(Y_t = y_t) = 1$. Learning algorithms at this level only requires memorization of the learning trace.
3.5. Level 2 - Reactive Behavior

In this second approach to LfO, we only assume that the expert has no internal state, i.e. that $C_t$ is irrelevant, and thus the action $Y_t$ only depends on the current observation ($X_t$). In this case, we can simplify the LfO-DBN model as shown in Figure 3. Under this assumption, we can use supervised learning techniques to learn the conditional probability $\rho(Y_t|X_t)$.

In this approach, each entry in a learning trace can be treated independently. When the behavior to be learned is not stochastic, or when we can assume that the variations from the mean observed in the training data are only the result of noise, standard supervised learning algorithms such as induction of decision trees, neural networks or SVMs are enough to learn any behavior at this level. Each pair $(X_t, Y_t)$ can be considered as a training example. In the behavior to be learned is stochastic, then supervised learning algorithms that can learn a probability distribution should be used.

This approach cannot learn any behavior that requires the agent to remember past states. The next two approaches alleviate this problem by making less restrictive assumptions about the internal state of the expert.

3.6. Restricted Level 3 - Time Window-based Behavior

In this approach, we assume that the expert internal state is a time window memory that stores the last $k$ observations, i.e., the current state $X_t$, and the last $k-1$ observations $O_{t-1}, \ldots, O_{t-(k-1)}$ (which corresponds to the typical “sliding window” approach [11]). For example, if $k = 2$, the expert internal state is $C_t = (X_t, O_{t-1})$. Under this assumption we can reformulate the LfO-DBN model, as shown in Figure 3.b for $k = 2$. Notice that given $k$, we can ignore $C_t$ in the DBN model, and thus, we still have no hidden variables. In general, for any given $k$, the conditional probability that must be learned is: $\rho(Y_t|X_t, O_{t-1}, \ldots, O_{t-(k-1)})$. 

Figure 3: a) Simplification of the model shown in Figure 2 for LfO tasks of level 2. b) simplification when the internal state of the expert is a time window of size $k = 2$. 
Figure 4: Example extraction from a trace. Internal state is a time window with \( k = 2 \).

Figure 5: A Dynamic Bayesian Network (DBN) representation of Hidden Markov Models and Input-Output Hidden Markov Models.

Notice that this is a restricted version of LfO tasks of Level 3. In this approach, each subsequence of \( k \) entries in a learning trace can be treated independently as a training example, and we can still use supervised learning techniques. Figure 4 shows how training examples for supervised learning can be extracted under this assumption. The main drawback of this approach is that, as \( k \) increases, the number of features in the training examples increases, making the learning task becomes more complex.

3.7. Level 3 - Memory-based Behavior

In this approach, we take into account that the actor \( C \) has internal state. The internal state \( C_t \) and the actions \( Y_t \) at time \( t \) depend on the perceived state of the environment and on the internal state at previous instants of time. In this situation, the complete LfO-DBN model, as shown in Figure 2 must be considered. The main difference between this approach and the previous two is that now, the supervised learning hypothesis and the conditional hypothesis \( p(C_t|x_t, c_{t-1}) = p(C_t|x_t) \), might not be true and therefore learning traces have to be considered as a whole and cannot be divided into collections of training examples \((X_t, Y_t)\) as before.
A particular case is if we assume that the internal state $C_t$ depends only on previous internal state $C_{t-1}$ and observation $X_t$. In this case, interestingly enough, the resulting model shown in Figure 2 corresponds to an *Input-Output Hidden Markov Model* (IOHMM) [4] (in this case the internal state depends only on previous inputs, $X_1, \ldots, X_t$). IOHMMs are an extension of Hidden Markov Models (HMM) [36], that can be used to learn how to map an input sequence to an output sequence (in our case, how to map the sequence of environment observations $X_t$ to actions, $Y_t$). In a standard HMM there are only two variables: a hidden internal state, typically denoted by $x$, and an output variable, typically denoted by $y$. In a IOHMM, there is a third variable $u$, called *input*, and both the internal state and the observation variable depend on $u$ (see Figure 5). In LfO terminology, the internal state of the IOHMM corresponds to the internal state of the actor, $C_t$, the output variable is the action executed by the expert, $Y_t$, and the input variable (upon which the other two depend on) is the observation of the environment state $X_t$. Learning algorithms for IOHMMs exist for restricted cases. The standard learning algorithm [4] is derived from the expectation-maximization (EM) algorithm [9] and is applicable as long as the internal state variable $C_t$ is discrete ($X_t$ and $Y_t$ can be either continuous or discrete).

Another interesting case is that where the input variables $X_t$ are also unobservable, corresponding to the well known HMM, as shown in Figure 5.

The main conclusion of this section is that tasks of different levels impose different restrictions upon the learning algorithm. Tasks of level 1 require simple memorization, some tasks of level 2 correspond to supervised learning (when behaviors are not stochastic), whereas, in general, tasks of level 3 require DBN-style learning algorithms. A specific subclass of behaviors of level 3 (those that require only remembering a fixed amount of time steps) can also be transformed to level 2 by using a sliding-window approach.

Assuming finite and discrete internal state, we can use EM to learn the parameters of the DBN, but there is no known algorithm to learn the general class of problems with unrestricted internal state, to the best of our knowledge. Next section discusses evaluation metrics for LfO algorithms.

4. Evaluation Metrics for LfO

As mentioned before, the evaluation of the performance of LfO algorithms is still an open problem. In the situation where the task $T$ is known, and the learning agent thus has access to a reward function, evaluation is easier,
because we can use the reward function to assess the performance of the agent, in the same way as in reinforcement learning. However, when the task \( T \) is not specified, the performance of an LfO algorithm needs to be measured by how well the learned behavior “resembles” that of the observed actor. Assessing this “resemblance” can be easy or hard, depending on the behavior to be learned. In this section, we present a collection of evaluation metrics that can be used for a range of behaviors.

4.1. Restricted Level 2 - Deterministic Reactive Behavior

When the behavior to be learned is reactive (i.e., it is level 2) and we want to build a deterministic model for this behavior, we can interpret the behavior \( BC \) of the actor \( C \) as a random variable \( I = (X, Y) \) that takes values in \( \mathcal{X} \times \mathcal{Y} \), where \( X \) and \( Y \) represent the input and the output variables respectively. In this case, a learning trace \( LT = [(x_1, y_1), \ldots, (x_n, y_n)] \) observed by the learning agent \( A \) can be seen as a sample in \( (\mathcal{X} \times \mathcal{Y})^n \), that is, \( n \) examples independently drawn according to the unknown distribution \( \rho_C \) of variable \( (X, Y) \) that governs the behavior of the actor, \( BC \).

A deterministic model for the behavior of agent \( C \) is an agent \( A \) that reacts always in the same way if the same input is given. Such a deterministic agent \( A \) can be modeled as an input/output map: \( A : \mathcal{X} \rightarrow \mathcal{Y} \). If we have a loss functions \( l(y, y') \) between actions, then the risk of agent \( A \) is defined as the average value of the loss function over the pairs \( (A(x), y) \):

\[
R(A) := E_{\rho_C}[l(A(x), y)] = \int_{\mathcal{X} \times \mathcal{Y}} l(A(x), y) d\rho_C,
\]

and the goal of learning is to find a model \( A \) minimizing the previous risk function. When a trace \( LT_C \), generated by the behavior of the actor \( BC \), is given the former theoretical quantity is replaced by the empirical risk

\[
R_{LT_C}(A) := \frac{1}{n} \sum_{i=1,\ldots,n} l(A(x_i), y_i)
\]

If the loss function \( l : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R} \) is a 0-1 function, \( l(y, y') = I_{\{y\}}(y') \) (where \( I_{\{y\}} \) is the indicator function), then the (empirical) accuracy of agent \( A \) measured using the learning trace \( LT_C \) is

\[
AC_{LT_C}(A) := 1 - R_{LT_C}(A) = 1 - \frac{1}{n} \sum_{i=1,\ldots,n} l(A(x_i), y_i)
\]
and represents the proportion of actions correctly predicted by agent A. If the space of actions $\mathcal{Y}$ is discrete we shall use the $0-1$ loss function as explained before. If actions are represented by real numbers (or vectors of real numbers) the square error loss function could be more accurate.

4.2. Level 2 - Stochastic Reactive Behavior

When the behavior to be learned is stochastic, standard empirical risk as defined in Section 4.1 (or classification accuracy) could not be representative of the performance of a LfO agent. For example, an agent trying to replicate an actor that executes actions at random would, by definition, have a very low classification accuracy, even if its behavior is exactly the same as that of the actor (executing actions at random). For this reason, we need to use an evaluation metric that compares probability distributions over actions, rather than specific actions. Several metrics exist in the literature of Information Theory. Perhaps the most common strategy for density estimation is based on likelihood principle. These algorithms try to minimize Kullback-Leibler divergence [20] in the presence of empirical data. For this reason the metrics that we propose are based on the Kullback-Leibler (KL) divergence.

Let $\rho_C$ and $\rho_A$ be the probability measures on $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$ that govern the actor behavior $B_C$ and the learning agent behavior $B_A$ respectively. Let’s assume that $\rho_C$ is absolutely continuous respect to $\rho_A$ and that $\frac{d\rho_C}{d\rho_A}$, the density function of measure $\rho_C$ with respect to the measure $\rho_A$, exists (then, its Radon-Nikodym derivative). Under these conditions the risk of agent $A$ is defined as the KL-divergence of $\rho_A$ from $\rho_C$, that is, the expectation of the logarithmic difference between $\rho_C$ and $\rho_A$, $\log \frac{d\rho_C}{d\rho_A}(x,y)$.

$$KL(\rho_C, \rho_A) = E_{\rho_C} \left[ \log \frac{d\rho_C}{d\rho_A} \right]$$  \hspace{1cm} (4)

From know on we assume that the space $\mathcal{Z}$ is either discrete or $\mathbb{R}^k \times \mathbb{R}^l$ (for some natural numbers $k$ and $l$). We also assume that $\rho_C$ and $\rho_A$ are Borel measures (in the continuous case) and, if we take as reference the uniform distribution on $\mathcal{Z}$ (the Lebesgue measure in the continuous case), then $\rho_C$ and $\rho_A$ have density functions, $f_C(x,y)$ and $f_A(x,y)$ respectively. In this case $\frac{d\rho_C}{d\rho_A}$ is just $\frac{f_C(z)}{f_A(z)}$. Hence, the risk of agent $A$ can be written as:

$$\int_{\mathcal{Z}} f_C(z) \log \frac{f_C(z)}{f_A(z)} dz.$$  \hspace{1cm} (5)
Because of the general properties of the KL-divergence, we have $KL(\rho_C, \rho_A) \geq 0$, and $KL(\rho_C, \rho_A) = 0$ if and only if $\rho_C = \rho_A$. Moreover, we know that:

$$KL(\rho_C, \rho_A) = \int_Z f_C(z) \log f_C(z) \, dz - \int_Z f_C(z) \log f_A(z) \, dz = H(\rho_C) + H(\rho_C, \rho_A)$$

The first term of previous formula $H(\rho_C)$ is the negative entropy of probability measure $\rho_C$ and is constant (only depends on the actor behavior $B_C$). The second term $H(\rho_C, \rho_A)$ is the crossed entropy between $\rho_C$ and $\rho_A$. Notice therefore, that in order to use the KL-divergence, we need to know the probability distribution that governs the behavior of the expert. Because we can only approximate this, we propose an alternative way of assessing performance, based on the observation that $\rho_C$ is a constant. Our final choice to define the risk of agent $A$ is the quantity $R(A)$ defined by the crossed entropy:

$$R(A) := H(\rho_C, \rho_A) = E_{\rho_C}[f_A(z)] = - \int_Z f_C(z) \log f_A(z) \, dz \quad (6)$$

This former definition – sometimes called the Vapnik risk in the related field of density estimations [42] – has the advantage that, once we have learned agent $A$, the risk of $A$ can be approximated using a Monte Carlo estimation on empirical data, without requiring to know the probability distribution that governs the expert’s behavior. If we have a trace $LT_C = [z_1, \ldots, z_n]$ generated according to the actor’s behavior $B_C$, the Monte Carlo approximation of Equation 6 is defined as:

$$R_{LT_C}(A) = -\frac{1}{n} \sum_{i=1}^{n} \log f_A(z_i) \quad (7)$$

Note that, on the other hand, minimizing the previous expression over empirical data, corresponds just to the maximum likelihood principle.

It is interesting also to note that sometimes risk estimation can be solved without estimating the densities first (as suggested in Equation 7). For instance if the distributions are discrete, this intermediate step is unnecessary and the KL-divergence or the risk can be estimated using directly the empirical distributions. To do this it is enough to substitute the distributions $\rho_C$ (corresponding to the actor behavior) and $\rho_A$ (corresponding to the learning agent) by their corresponding empirical distributions. Let $\rho_{LT_C}$ and $\rho_{LT_A}$ the
corresponding empirical distributions obtained from learning traces $LT_C$ and $LT_A$ respectively. Recall that if $\rho$ is a discrete probability measure on $\mathbb{Z}$, the empirical probability measure associated to a learning trace $LT = [z_1, \ldots, z_n]$ of size $n$ is defined as:

$$\rho_{LT}(z) := \frac{1}{n} \sum_{i=1}^{n} \delta_{z_i}(z),$$  

where $\delta_{z_i}$ is the Dirac distribution. If $f$ is a real valued function on $\mathbb{Z}$ then:

$$\int_{\mathbb{Z}} f(z) d\rho_{LT} = \frac{1}{n} \sum_{i=1}^{n} f(z_i)$$

Next, using a trace $LT_C = [z_1, \ldots, z_n]$ of size $n$ (generated according to the actor behavior) and a trace of size $m$ $LT_A = [z'_1, \ldots, z'_m]$ (generated according to the learning agent behavior), we estimate the risk $R(A)$ defined in Equation 6 by the risk between the empirical distribution of the learning agent $\rho_{LT_A}$ and and the empirical behavior of the actor $\rho_{LT_A}$. This leads to the following definition:

$$R_{LT_A,LT_C}(A) = -\int_{\mathbb{Z}} \log \rho_{LT_A}(z) d\rho_{LT_C}$$

(10)

If the distributions are discrete, using Equations 8 and 9 we get the following expression:

$$R_{LT_A,LT_C}(A) = -\frac{1}{n} \sum_{i=1}^{n} \log \left[ \frac{1}{m} \sum_{j=1}^{m} \delta_{z'_j}(z_i) \right]$$

(11)

Strong consistency for the estimator $R_{LT_A,LT_C}(A)$ is because of the strong law of large numbers. Moreover, to improve estimations when the available traces are too short, or do not cover the entire $\mathbb{Z}$, we can use Laplace smoothing, and modify the equation like this (which is the actual equation we used in the experiments reported in the next section):

$$R_{LT_A,LT_C}(A) = -\frac{1}{n} \sum_{i=1}^{n} \log \left[ \frac{1 + \sum_{j=1}^{m} \delta_{z'_j}(z_i)}{|\mathbb{Z}| + m} \right]$$

(12)

The previous method can be generalized to continuous probability measures using a empirical measure based on partitions. The statistical background of this approach is beyond the scope of this paper.
4.2.1. Estimating Risk from Conditional Distributions

When dealing with reactive behaviors, sometimes it is easier to work with the conditional probabilities \( \rho_C(Y|x) \) and \( \rho_A(Y|x) \) than with the joint distribution. In this case, if we have access to a method for generating environmental states (from \( \mathcal{X} \)) following some distribution, then we know the marginal distribution on the the space \( \mathcal{X} \), call it \( \rho_C(X) \). The question that naturally arises is which risk function should be used in this situation. An intuitive answer to this question is to use the expectation of the Kulback-Leibler divergence between the conditional distributions, that is,

\[
E_{\rho_C(X)}[KL(\rho_C(Y|x), \rho_A(Y|x))],
\]

where \( \rho_C(Y|x) \) and \( \rho_A(Y|x) \) are the conditional distributions of actions \( Y \) given the states \( X = x \) of the actor and the learning agent respectively. The previous expression can be decomposed into two parts:

\[
-E_{\rho_C(X)}[H(\rho_C(Y|x))] + E_{\rho_C(X)}[H(\rho_C(Y|x), \rho_A(Y|x))].
\]

Because of the previous decomposition, given that the first term is constant (i.e. it does not depend on the learning agent), our choice for the risk function in this situation is:

\[
R^*(A) := E_{\rho_C(X)}[H(\rho_C(Y|x), \rho_A(Y|x))] = -\int_X f_{C,X}(x) \left[ \int_Y f_{C,Y|x}(y) \log f_{A,Y|x}(y) \, dy \right] \, dx
\]

where \( f_{C,X} \) is the density function of \( \rho_C(X) \) and \( f_{C,Y|x} \) is the density function of \( \rho_C(Y|x) \). We call \( R^*(A) \) the conditional distribution risk. We note this risk measure with an asterisk, to indicate it is computed from conditional probabilities, rather than from absolute probabilities, as were the risk functions defined before. The precise relationship between the quantities \( R(A) \) (defined in Equation 6) and \( R^*(A) \) (defined in Equation 13) can be established using Bayes and Fubini’s theorems as follows.

**Proposition 1.** Let \( B_C \) and \( B_A \) be the actor and the learning agent behaviors respectively. Let \( R(A) \) and \( R^*(A) \) be as before. Then the following holds:

\[
R(A) = H(\rho_C(X), \rho_A(X)) + R^*(A)
\]

where \( H(\rho_C(X), \rho_A(X)) \) denotes the cross entropy between \( \rho_C(X) \) and \( \rho_A(X) \).

A proof for Proposition 1 is given in Appendix A, and an interpretation of Proposition 1 is given in next Corollary.
Corollary 1. Assume that the environmental states \( x \in X \) are generated according to the same probability measure on \( X \), \( \rho = \rho_C(X) = \rho_A(X) \), both for the actor \( C \) and for the learning agent \( A \). Then \( R(A) \) and \( R^*(A) \) differ only by an additive constant, namely, the entropy \( H(\rho) \) of the marginal measure \( \rho \).

4.3. Restricted Level 3 - Time Window-based Behavior

Previous metrics (classification accuracy, and the way we employed KL-divergence or the risk notion) assume reactive behavior. However, they can also be applied to non-reactive behavior if we know that the behavior to be learned only depends on a fixed set of past observations by using the idea of time-windows.

4.4. Level 3 - Memory-based Behavior

In general, the method employed above to measure the difference between the behavior of the actor and of the learning agent is not enough for behaviors of level 3, since it does not take into account the internal state of the actor. However, we can use the same idea, and exploit our LfO-DBN model. In the graphical model formalism, a LfO-DBN is represented as a chain structure as shown in Figure 2. The state space for the stochastic process represented by the LfO-DBN structure is now \( Z := X \times C \times Y \). Throughout this paper we have assumed that the set of internal states \( C \) is finite and each variable \( C_t \) is a multinomial random variable. We also assume that the transition probability distributions \( a(C_t|C_{t-1}, X_{t-1}, Y_{t-1}) \) and the output probabilities \( b(Y_t|X_t, C_t) \) are time homogenous (i.e. do not depend on time). Let \( \pi \) be the initial state probability. Given the previous assumptions, and a learning agent \( A \) adopting the LfO-DBN model, the probability distribution \( \rho \) of the sequence of length \( n \) \( [(y_1, c_1), ..., (y_n, c_n)] \) given \([x_1, ..., x_n]\) is:

\[
\rho_A(\{Y_{1..n}\}, \{C_{1..n}\}|\{X_{1..n}\}) = \\
= \pi(C_1|X_1) \prod_{t=2}^{n} a(C_t|C_{t-1}, X_{t-1}, Y_{t-1}) \prod_{t=1}^{n} b(Y_t|X_t, C_t)
\]

where \( Y_{1..n} \) is a short for \([y_1, ..., y_n]\). Summing this probabilities in Equation 14 over the hidden values \( C_t \) we can compute \( \rho_A(\{Y_{1..n}\}|\{X_{1..n}\}) \). Next, we define the \( n \)-level conditional distribution risk of agent \( A \) (following the LfO-DBN model) with respect to an actor \( C \) as:

\[
R^*_n(A) =
\]
Remark 1. Note that the previous definition is the natural extension of the expression proposed for $R^*(A)$ for reactive behaviors (Equation 13), but in this case dealing with the (conditional) distributions over sequences of observations $(\mathcal{X} \times \mathcal{Y})^n$, rather than individual observations $(\mathcal{Z} = \mathcal{X} \times \mathcal{Y})$.

Our assumptions about transition and output probabilities, in particular those assuming the time homogeneous property, implies that the LfO-DBN defines a stationary process. In this case, the risk rate, defined as:

$$R^*(A) := \lim_{n \to \infty} \frac{1}{n} R^*_n(A),$$

is finite. The risk rate can be used as a length-independent risk measure.

The LfO-DBN is a 2D-lattice with inhomogeneous field terms (the observed data). It is well known that such kind of structures are intractable using exact calculations. It is not difficult to write down the EM algorithm giving the recursions for the calculations of posterior probabilities in the E-step. These calculations are not too time-consuming for practical use (for $n$ time steps, $k$ values per node the algorithm scales as $O(k^2n)$).

Another important question is how to approximate the risk rate $R^*(A)$ from empirical data. To perform a Monte Carlo approximation of expression 15 we note that a version of Fubini’s theorem states that for any measure $\rho$ on a product space $\mathcal{X} \times \mathcal{Y}$ if we consider the factorization of $\rho$ as the marginal measure $\rho_X$ on $\mathcal{X}$ and the conditional measure $\rho_{Y|X}$ then for any integrable function $\phi(x, y)$ the following holds: $\int \phi(x, y) d\rho = \int \left[ \int \phi(x, y) d\rho_{Y|X} \right] d\rho_X$. Using Fubini’s theorem in this form, the $n$-level risk can be written as:

$$R^*_n(A) = -\int \log \rho_A(\{Y_{1:n}\}|\{X_{1:n}\})d\rho(\{X_{1:n}\}|\{Y_{1:n}\})$$

Because this last integral is over the joint distribution $\rho_C(\{X_{1:n}\}|\{Y_{1:n}\})$, given a set of traces $T_C$ of length $n$ generated by the actor $C$, the Monte Carlo approximation of $R^*_n(A)$ is defined as:

$$R^*_n(A) = -\frac{1}{|T_C|} \sum_{T \in T_C} \frac{1}{n} \log \rho_A(\{T.Y_{1:n}\}|\{T.X_{1:n}\})$$ (17)
Classification Accuracy

\[ AC_{LT_c}(A) := 1 - \frac{1}{n} \sum_{i=1,...,n} l(A(x_i), y_i) \]

Applicable for deterministic reactive behaviors

Monte Carlo approximation of Vapnik’s Risk:

\[ R_{LT_A,LT_c}(A) = -\frac{1}{n} \sum_{i=1}^{n} \log \left( \frac{1}{m} \sum_{j=1}^{m} \delta_{z_j}(z_i) \right) \]

Applicable for deterministic or stochastic reactive behaviors

Monte Carlo approximation of Vapnik’s rate:

\[ R_{T_c}^*(A) = -\frac{1}{|T_C|} \sum_{T \in T_C} \frac{1}{n} \log \rho_A(\{T.Y_{1...n}\}|\{T.X_{1...n}\}) \]

Applicable for deterministic or stochastic memory-based behaviors

Figure 6: Summary of the different metrics presented in Section 4, where \( A(x_i) \) represents the action that the learning agent \( A \) would execute when observing \( x_i \); \( \delta_{z'_j}(z) = 1 \) if \( z = z'_j \) and 0 otherwise); and \( \rho_A(\{T.Y_{1...n}\}|\{T.X_{1...n}\}) \) is the probability that the learning agent \( A \) executed the sequence of actions \( \{T.Y_{1...n}\} \) when observing the sequence of observations \( \{T.X_{1...n}\} \) (this probability is estimated by learning an LfO-DBN using traces generated by the learning agent).

where \( T.Y_{1...n} \) represent the sequence of actions the actor executed in trace \( T \), and \( T.X_{1...n} \) is the set of observations the actor observed in trace \( T \).

4.5. Summary

Summarizing the definitions and results presented in the previous subsections, all the metrics we proposed follow the same general idea: 1) learn a probability distribution for the behavior of each of the agents, 2) use a metric based on the KL-divergence to determine how similar those probability distributions are. The more similar, the better the learning agent has learned. Moreover, instead of using directly the KL-divergence, we use Vapnik’s risk, which does not require to know the probability distribution that governs the behavior of the expert. For behaviors of level 2, these probability distributions take the form of either absolute distributions of the pair of variables \( (X,Y) \), or conditional probabilities \( P(Y|X) \). For behaviors of level 3, these probability distributions take the form of a dynamic Bayesian network, such as the LfO-DBN. Figure 6 presents the list of the evaluation methods.
metrics for LfO that we propose, and that is evaluated in the next section.

5. Experimental Evaluation

This section presents an experimental validation of the theoretical concepts put forward in this paper. Specifically we want to evaluate the two main hypotheses in this paper:

**H1:** Standard supervised learning metrics for learning algorithm performance do not correlate well with the performance of LfO algorithms. The proposed metrics put forward in Section 4 better evaluate how close the behavior of two agents is.

**H2:** Standard supervised learning algorithms (typically used in many approaches to LfO) can only handle tasks of level 2 that are deterministic. For learning stochastic behaviors, or behaviors of level 3, we need a different kind of algorithms.

The following two sections present a collection of experiments designed to validate these two hypotheses respectively. We used simple domains in our evaluation, because those are enough to evaluate whether H1 and H2 hold.

5.1. Performance Evaluation in Learning from Observation

In order to evaluate our proposed metrics, and to show the weaknesses of standard metrics (such as classification accuracy) for evaluating how similar is the behavior learned by the learning agent to that of the actor, we designed the following experiment. We defined a toy domain as follows:

- There are only three possible states: $X = \{x_1, x_2, x_3\}$.
- The agent can only execute two actions: $Y = \{y_1, y_2\}$, with deterministic effect. Action $y_1$ corresponds to moving to the “right” (goes from state $x_1$ to $x_2$, from $x_2$ to $x_3$ and when executed in $x_3$, makes the state go back to $x_1$). Action $y_2$ corresponds to moving to the “left” (from state $x_1$ the action changes the state of $x_3$, from state $x_2$ to $x_1$ and from state $x_3$ to $x_2$).

We defined eight different agents of several levels with the following behaviors:
SEQ: FixedSequence. (level 1) This agent always repeats the same, fixed, sequence of actions (the sequence is six actions long). Once the sequence is over, it restarts from scratch.

RND: Random. (level 1/2) This agent executes actions at random.

L2DetA: Level 2 Deterministic A. (level 2) This agent produces “right” for states $x_1$ and $x_2$ and “left” otherwise.

L2DetB: Level 2 Deterministic B. (level 2) This agent produces “left” for states $x_1$ and $x_2$ and “right” otherwise.

L2StoA: Level 2 Stochastic A. (level 2) This agent produces, with a probability 0.75 “right” for states $x_1$ and $x_2$ and “left” for $x_3$. It has a probability of 0.25 of producing the opposite action.

L2StoB: Level 2 Stochastic B. (level 2) This agent produces, with a probability 0.75 “left” for states $x_1$ and $x_2$ and “right” for $x_3$. It has a probability of 0.25 of producing the opposite action.

RNDS: RandomStraight. (level 3) This agent executes one action at random when in state $x_1$. When in states $x_2$ and $x_3$, it executes the same action as in the previous execution cycle with probability 5/6 and the opposite action with probability 1/6.

INT: InternalState. (level 3) This agent has two internal states: $i_1$ and $i_2$ (initially, it starts at $i_1$). When at $i_1$, the agent executes action $y_1$; when at $i_2$, it executes action $y_2$. When the agent observes state $x_1$, it switches states.

We then generated two traces ($LT^1_A$ and $LT^2_A$) of 1000 time steps for each agent $A$. We then used performance metrics to determine how well does the behavior of one agent resemble the behavior of another using these traces. We then compared the behavior of $A$ against $C$ in the following way:

- When using classification accuracy, we take trace $LT^2_C$ and feed the perception state step by step to agent $A$, asking $A$ to generate the corresponding action $Y$. We then use Equation 3 to determine classification accuracy.

- When using Vapnik’s Risk, we use the 1000 time steps in traces $LT^1_A$ and $LT^2_C$ to compute the Monte Carlo approximation in Equation 12.
Table 1: Comparison of the behavior of different agents using classification accuracy ($AC_{LTc}(A)$) ($A$ is the column agent and $C$ is the row agent). The higher, the better performance. The highest value in each row is highlighted.

<table>
<thead>
<tr>
<th>SEQ</th>
<th>RND</th>
<th>L2DetA</th>
<th>L2DetB</th>
<th>L2StoA</th>
<th>L2StoB</th>
<th>RNDS</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>1.00</td>
<td>0.52</td>
<td>0.66</td>
<td>0.33</td>
<td>0.59</td>
<td>0.40</td>
<td>0.48</td>
</tr>
<tr>
<td>RND</td>
<td>0.53</td>
<td>0.50</td>
<td>0.51</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.51</td>
</tr>
<tr>
<td>L2DetA</td>
<td>0.33</td>
<td>0.53</td>
<td>1.00</td>
<td>0.00</td>
<td>0.74</td>
<td>0.24</td>
<td>0.50</td>
</tr>
<tr>
<td>L2DetB</td>
<td>0.33</td>
<td>0.52</td>
<td>0.00</td>
<td>1.00</td>
<td>0.25</td>
<td>0.76</td>
<td>0.50</td>
</tr>
<tr>
<td>L2StoA</td>
<td>0.50</td>
<td>0.48</td>
<td>0.75</td>
<td>0.25</td>
<td>0.63</td>
<td>0.39</td>
<td>0.54</td>
</tr>
<tr>
<td>L2StoB</td>
<td>0.49</td>
<td>0.50</td>
<td>0.25</td>
<td>0.75</td>
<td>0.25</td>
<td>0.76</td>
<td>0.50</td>
</tr>
<tr>
<td>RNDS</td>
<td>0.50</td>
<td>0.51</td>
<td>0.50</td>
<td>0.50</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
</tr>
<tr>
<td>INT</td>
<td>0.33</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.49</td>
<td>0.49</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Table 2: Comparison of the behavior of different agents using Vapnik’s Risk ($R_{LT_A,LT_C}(A)$) ($A$ is the column agent and $C$ is the row agent). The lower, the better performance. The lowest value in each row is highlighted.

<table>
<thead>
<tr>
<th>SEQ</th>
<th>RND</th>
<th>L2DetA</th>
<th>L2DetB</th>
<th>L2StoA</th>
<th>L2StoB</th>
<th>RNDS</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>1.33</td>
<td>1.81</td>
<td>4.61</td>
<td>6.91</td>
<td>1.75</td>
<td>2.25</td>
<td>1.79</td>
</tr>
<tr>
<td>RND</td>
<td>3.23</td>
<td>1.79</td>
<td>4.66</td>
<td>4.89</td>
<td>1.95</td>
<td>1.98</td>
<td>1.79</td>
</tr>
<tr>
<td>L2DetA</td>
<td>1.79</td>
<td>1.85</td>
<td>0.70</td>
<td>6.91</td>
<td>1.24</td>
<td>2.55</td>
<td>1.8</td>
</tr>
<tr>
<td>L2DetB</td>
<td>6.91</td>
<td>1.74</td>
<td>6.91</td>
<td>0.70</td>
<td>2.65</td>
<td>1.23</td>
<td>1.79</td>
</tr>
<tr>
<td>L2StoA</td>
<td>2.31</td>
<td>1.81</td>
<td>3.15</td>
<td>6.06</td>
<td>1.62</td>
<td>2.28</td>
<td>1.8</td>
</tr>
<tr>
<td>L2StoB</td>
<td>4.69</td>
<td>1.77</td>
<td>5.97</td>
<td>3.18</td>
<td>2.31</td>
<td>1.61</td>
<td>1.79</td>
</tr>
<tr>
<td>RNDS</td>
<td>3.25</td>
<td>1.79</td>
<td>4.71</td>
<td>4.86</td>
<td>1.97</td>
<td>1.96</td>
<td>1.79</td>
</tr>
<tr>
<td>INT</td>
<td>3.27</td>
<td>1.79</td>
<td>4.73</td>
<td>4.84</td>
<td>1.97</td>
<td>1.96</td>
<td>1.79</td>
</tr>
</tbody>
</table>

- When using Vapnik’s rate, we learn an LfO-DBN from trace $LT_A$, and then compute Vapnik’s Risk (Equation 17) with respect to $LT_C$. Thus, the Monte Carlo approximation in Equation 17 is done with a single trace (i.e. $T_C = \{LT_C^2\}$).

Ideally, a good performance metric should result in a high score when comparing an expert with herself, and result in a low score when comparing different experts. Thus, the goal is to validate hypotheses H1 above, and show that the metrics presented in Section 4 better reflect when the behavior of two agents is similar than standard supervised learning metrics.

For comparison purposes, we also compared the traces using a standard statistical test, based on the $\chi^2$ distance. The $\chi^2$ distance is the underlying measure used in the Pearson statistical hypothesis test. We use it here as a measure of dissimilarity between probability distributions. For this purpose, we compute the empirical probability distributions $\rho_{LT_C}$ and $\rho_{LT_A}$ for $\mathcal{Z}$ from
Table 3: Comparison of the behavior of different agents using the $\chi^2$ distance ($A$ is the column agent and $C$ is the row agent). The lower, the better performance. The lowest value in each row is highlighted.

<table>
<thead>
<tr>
<th></th>
<th>SEQ</th>
<th>RND</th>
<th>L2DetA</th>
<th>L2DetB</th>
<th>L2StoA</th>
<th>L2StoB</th>
<th>RNDS</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>0.00</td>
<td>62.07</td>
<td>1.98</td>
<td>498.01</td>
<td>11.63</td>
<td>172.48</td>
<td>55.66</td>
<td>55.28</td>
</tr>
<tr>
<td>RND</td>
<td>0.66</td>
<td>0.00</td>
<td>1.88</td>
<td>2.04</td>
<td>0.28</td>
<td>0.35</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>L2DetA</td>
<td>166.11</td>
<td>102.47</td>
<td>0.00</td>
<td>1.97</td>
<td>0.71</td>
<td>2.12</td>
<td>2.01</td>
<td>13.96</td>
</tr>
<tr>
<td>L2DetB</td>
<td>277.22</td>
<td>105.56</td>
<td>497.51</td>
<td>0.00</td>
<td>2.26</td>
<td>0.53</td>
<td>0.53</td>
<td>0.53</td>
</tr>
<tr>
<td>L2StoA</td>
<td>0.84</td>
<td>0.56</td>
<td>0.7</td>
<td>6.99</td>
<td>0.00</td>
<td>2.26</td>
<td>0.53</td>
<td>0.53</td>
</tr>
<tr>
<td>L2StoB</td>
<td>1.63</td>
<td>0.45</td>
<td>6.87</td>
<td>0.67</td>
<td>0.17</td>
<td>0.01</td>
<td>0.48</td>
<td>0.51</td>
</tr>
<tr>
<td>RNDS</td>
<td>0.66</td>
<td>0.01</td>
<td>1.94</td>
<td>2.01</td>
<td>0.31</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>INT</td>
<td>0.66</td>
<td>0.00</td>
<td>1.97</td>
<td>1.98</td>
<td>0.32</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4: Comparison of the behavior of different agents using Vapnik’s rate ($A$ is the column agent and $C$ is the row agent). The lower, the better performance. The lowest value in each row is highlighted.

<table>
<thead>
<tr>
<th></th>
<th>SEQ</th>
<th>RND</th>
<th>L2DetA</th>
<th>L2DetB</th>
<th>L2StoA</th>
<th>L2StoB</th>
<th>RNDS</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>1.33</td>
<td>2.12</td>
<td>36.01</td>
<td>36.04</td>
<td>1.88</td>
<td>2.59</td>
<td>2.01</td>
<td>13.96</td>
</tr>
<tr>
<td>RND</td>
<td>36.04</td>
<td>1.86</td>
<td>36.04</td>
<td>35.97</td>
<td>2.09</td>
<td>2.24</td>
<td>2.16</td>
<td>29.04</td>
</tr>
<tr>
<td>L2DetA</td>
<td>1.81</td>
<td>2.31</td>
<td>0.71</td>
<td>36.04</td>
<td>1.16</td>
<td>3.50</td>
<td>3.72</td>
<td>40.26</td>
</tr>
<tr>
<td>L2DetB</td>
<td>36.04</td>
<td>1.79</td>
<td>36.04</td>
<td>0.69</td>
<td>2.12</td>
<td>1.66</td>
<td>3.62</td>
<td>22.45</td>
</tr>
<tr>
<td>L2StoA</td>
<td>36.04</td>
<td>2.05</td>
<td>36.04</td>
<td>36.01</td>
<td>1.65</td>
<td>2.80</td>
<td>2.42</td>
<td>28.55</td>
</tr>
<tr>
<td>L2StoB</td>
<td>36.04</td>
<td>1.86</td>
<td>36.04</td>
<td>35.87</td>
<td>2.39</td>
<td>1.64</td>
<td>2.56</td>
<td>30.86</td>
</tr>
<tr>
<td>RNDS</td>
<td>35.83</td>
<td>1.86</td>
<td>36.01</td>
<td>36.04</td>
<td>2.12</td>
<td>2.16</td>
<td>1.66</td>
<td>19.43</td>
</tr>
<tr>
<td>INT</td>
<td>36.04</td>
<td>1.89</td>
<td>36.04</td>
<td>36.01</td>
<td>2.05</td>
<td>2.14</td>
<td>1.45</td>
<td>1.10</td>
</tr>
</tbody>
</table>

traces of both agents, $A$ and $C$, respectively. Given that our experts have discrete action space $\mathcal{Y}$, then the $\chi^2$ distance is:

$$
\chi^2(\rho_{LT_A}, \rho_{LT_C}) = \sum_{y \in \mathcal{Y}} \frac{(\rho_{LT_A}(z) - \rho_{LT_C}(z))^2}{\rho_{LT_C}(z)}
$$

where quantities $\rho_{LT_A}(z)$ and $\rho_{LT_C}(z)$ denote the probability of the pair observation/action $z$ under measure $\rho_{LT_A}$ and the probability of $z$ under measure $\rho_{LT_C}$. We use Laplace smoothing to prevent $\rho_{LT_C}(z)$ being zero.

Tables 1, 2, 3, and 4 show the results obtained using each of the metrics we used. For each metric, we show a matrix containing one agent per row, and also one agent per column. The number shown in row $i$ and column $j$ represents how good agent $j$ approximates the behavior of agent $i$ according to each metric. Table 1 shows the results computed using standard classi-
fication accuracy. For computing classification accuracy, we feed the set of observations that the row agent observed when generating the test trace to the column agent, recorded the actions the column agent generates and compared them against the actions the row agent actually generated. Ideally, we would like to have numbers close to 1.00 in the diagonal (100% of actions predicted correctly) and lower anywhere else (since an agent should be able to properly predict its own actions, but not the actions of other agents). However, we observe that the only elements of the diagonal that are 1.00 are for agents SEQ, L2DetA, L2DetB, and INT (the agents that are deterministic). When agents are not deterministic, classification accuracy is not a good measure of how similar their behaviors are.

Table 2 shows the results using a Monte Carlo approximation of Vapnik’s risk ($R_{LT_C,LT_A}$, shown in Equation 12). Here we expect the values in the diagonal to be lower than all the other values in the same row (notice that we cannot expect zeroes in the diagonal, since the minimum value that Vapnik’s risk can obtain is the entropy of the distribution of the expert $H(\rho_C)$). We can see that this metric works very well for all the agents of levels 1 and 2. As expected, it is not able to properly distinguish behaviors of level 3 (RNDS and INT). For example, in the rows for the RNDS and INT agents, the values obtained by RND, RNDS and INT are almost identical. Thus, if we know behaviors of interest are level 2, this indicates that this metric seems to more accurately determine whether an agent’s behavior is similar to another one than using classification accuracy.

Table 3 shows the results obtained by using a $\chi^2$ test. In this case, we would like the numbers in the diagonal to be close to 0, and the rest of numbers to be high. Notice that indeed the numbers in the diagonal are all 0 or very close, but some of the elements outside of the diagonal for the rows RNDS and INT are also close to 0. Thus, similarly as for Vapnik’s Risk, a $\chi^2$ test over the distributions $\rho_{LT_C}$ and $\rho_{LT_A}$ can only distinguish the behavior of agents when they are level 2, but not when they are level 3, as expected.

Finally, Table 4 shows the results obtained using a Monte Carlo approximation to Vapnik’s rate computed using LfO-DBNs (Equation 17). Here we expect the values in the diagonal to be lower than all the other values in the same row (we cannot expect zeroes in the diagonal for the same reason as for Vapnik’s risk). As we can see, this metric works well for all behaviors, and the minimum value for each row falls exactly in the diagonal. This means that this measure is properly able to determine that the behavior of a given expert is most closely predicted by that expert itself in our toy example.
The summary is that, in our experiments classification accuracy was only accurate to compare behaviors that were deterministic, Vapnik’s risk and $\chi^2$ over the empirical distributions $\rho_{LT_C}$ and $\rho_{LT_A}$ over $\mathcal{Z}$ worked for any behavior of Level 2 (or for behaviors that depend only on a fixed set of past states). For behaviors of level 3, since they require memory, we had to learn a dynamical model accounting for the internal state of the expert (such as an LfO-DBN) for each agent and then use Vapnik’s rate.

5.2. The Vacuum Cleaner Domain

In order to illustrate the limitations of traditional LfO algorithms, we designed a domain that simulates an automatic vacuum cleaner navigating a room, and removing dirt spots. A scenario is defined as a rectangular grid, where each cell in the grid can be either an empty space, a wall, an obstacle, or dirt. The vacuum cleaner is a 2 by 2 square that navigates the grid looking for dirt spots. If the vacuum cleaner enters in contact with a dirt spot, the dirt spot disappears. The goal is to remove all the dirt spots in the map. For these experiments, all the obstacles are static, and the only moving object in the simulation is the vacuum cleaner.

The simulation time is discrete, and at each time step, the vacuum cleaner can take one out of these five actions: up, down, left, right and stand still. The effect of the first four actions is to move the vacuum cleaner one cell in each of the four directions, unless there is an obstacle preventing the move (in this case, the vacuum cleaner simple won’t execute the action). Thus, there is a single control variable $y$, which can take five different values: \{up, down, left, right, stand\}.

The vacuum cleaner perceives the world through 8 different variables, illustrated in Figure 7. Basically, there are two variables per direction (up, down, left, right), one of them identifying what can the vacuum cleaner see in each direction (obstacle or dirt), and the other determining whether the object being seen is close (touching) or far. All the variables are thus binary. The state space of this domain is thus small, but the complexity of it comes from the range of behaviors that can be defined, some of which cannot be described as a perception to action mapping.

We hand crafted a collection of seven different maps, of different sizes, from 8x8 to 32x32 and with different configuration of obstacles and dirt. Next section presents our experimental results in this domain.
5.3. Comparison of LfO Algorithms

In order to test hypothesis H2, we created eight different agents by hand, that were used as the experts to learn from:

**WF: Wall Follower.** (deterministic, level 2) This agent always follows the left wall. If there is no wall being touched, then it moves right. The agent is completely deterministic, and has no internal state.

**SWF: Smart Wall Follower.** (deterministic, level 2) Like the WF agent, but if it sees dirt in one direction, it will move straight for it.

**RD: Random.** (stochastic, level 2) This agent just executes one action at random at each cycle.

**SRD: Smart Random.** (stochastic, level 2) This agent just executes one action at random at each cycle, except if it sees dirt in one of the four directions, in which case it will move straight for the dirt.

**SL: Straight Line.** (stochastic, level 3) This agent picks one direction at random and moves in a straight line until colliding with an obstacle or a wall. Then it picks a different direction at random, and restarts its behavior. Notice that this behavior has internal state, since the direction the vacuum cleaner last moved is not included amongst the perception variables $X$.

**SSL: Smart Straight Line.** (stochastic, level 3) Like SL, but if it sees dirt in one of the four directions, it will move straight for the dirt. This behavior also has internal state, like SL.
**ZZ: Zig-Zag.** (deterministic, level 3) This agent moves in zig-zag, it moves all the way to the right, until colliding, then moves down and moves to the left until colliding. When it cannot go down any further, it repeats the behavior, but going up, and so on. To perform this behavior, an agent must remember whether it is going left or right, and also whether it must go down or up after colliding with an object.

**SEQ: Fixed Sequence.** (deterministic, level 1/3) This agent always loops over the same, fixed, sequence of actions (15 actions long), independently of the perception.

We generated a total of 56 learning traces (one per expert per map). We will refer to $LT^m_C$ to the learning trace generate for agent $C$ in map $m$. Therefore, the learning agents have seven learning traces per expert. Each learning trace is 1000 steps long. We compared the performance of the following algorithms:

**Supervised Learning Algorithms (level 2):** We used NN (Neural Networks), widely used in the LfO literature [35, 26], and BN (Bayesian Networks), a direct implementation of the simplified Bayesian Network shown in Figure 3. Specifically, we trained Neural Networks with eight input nodes (one per perception variable), 10 nodes in one hidden layer, and five output nodes (one per possible action), and were trained using backpropagation\(^1\). For Bayesian Networks, our networks have nine variables (the eight perception variables, and one for the output variable, which depends on the other eight)\(^2\).

**Supervised Learning Algorithms (with time-window-based internal state, restricted level 3):** We also experimented with two algorithms in this case: NNk2 (Neural Networks) and BNk2 (a direct implementation of the Bayesian Network shown in Figure 3.b). For both algorithms, we used a time window of size $k = 2$, i.e. they learn to predict the expert actions based on the current state of the world, and the state and action in the previous instant of time. For example, this means

\(^1\)We used the Levenberg-Marquardt backpropagation implementation in Matlab’s Neural Network Toolbox: [http://www.mathworks.com/help/nnet/index.html](http://www.mathworks.com/help/nnet/index.html)

\(^2\)The maximum-likelihood estimation of the parameters of this network were estimated using the Bayesian Network Toolbox: [https://code.google.com/p/bnt/](https://code.google.com/p/bnt/)
Table 5: Classification accuracy of different LfO algorithms in predicting the actions of the expert. The bottom row shows the average of all the other rows (higher is better; best performer per row is highlighted). Right-most column shows the accuracy that the expert itself would achieve in predicting its own actions.

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>NNk2</th>
<th>BN</th>
<th>BNk2</th>
<th>LfO-DBN</th>
<th>IOHMM</th>
<th>Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF</td>
<td>0.99</td>
<td>0.85</td>
<td>0.92</td>
<td>0.86</td>
<td>0.88</td>
<td>0.88</td>
<td>1.00</td>
</tr>
<tr>
<td>SWF</td>
<td>1.00</td>
<td>0.86</td>
<td>0.94</td>
<td>0.88</td>
<td>0.93</td>
<td>0.93</td>
<td>1.00</td>
</tr>
<tr>
<td>RD</td>
<td>0.20</td>
<td>0.20</td>
<td>0.21</td>
<td>0.20</td>
<td>0.19</td>
<td>0.19</td>
<td>0.20</td>
</tr>
<tr>
<td>SRD</td>
<td>0.36</td>
<td>0.34</td>
<td>0.35</td>
<td>0.33</td>
<td>0.36</td>
<td>0.36</td>
<td>0.35</td>
</tr>
<tr>
<td>SL</td>
<td>0.49</td>
<td>0.93</td>
<td>0.49</td>
<td>0.91</td>
<td>0.79</td>
<td>0.73</td>
<td>0.95</td>
</tr>
<tr>
<td>SSL</td>
<td>0.49</td>
<td>0.94</td>
<td>0.49</td>
<td>0.92</td>
<td>0.90</td>
<td>0.83</td>
<td>0.94</td>
</tr>
<tr>
<td>ZZ</td>
<td>0.52</td>
<td>0.96</td>
<td>0.48</td>
<td>0.88</td>
<td>0.83</td>
<td>0.65</td>
<td>1.00</td>
</tr>
<tr>
<td>SEQ</td>
<td>0.53</td>
<td>0.66</td>
<td>0.40</td>
<td>0.47</td>
<td><strong>0.88</strong></td>
<td>0.65</td>
<td>1.00</td>
</tr>
<tr>
<td>Avg.</td>
<td>0.57</td>
<td><strong>0.72</strong></td>
<td>0.54</td>
<td>0.68</td>
<td><strong>0.72</strong></td>
<td>0.65</td>
<td>0.81</td>
</tr>
</tbody>
</table>

that the Neural Networks trained for NNk2 had 21 input nodes (eight for perception variables in the current time step, eight for perception variables in the previous time step, and five boolean variables, each representing one of the five possible actions that could have been executed in the previous time step), one hidden layer with 10 nodes and five output nodes. BNk2 Bayesian Networks had 18 variables (eight perception variables, eight perception variables from the previous time step, one for the action in the previous time step, and one for the output variable, which depends on the other 17). We trained them using the same learning algorithms as explained above.

Dynamic-Bayesian Networks Algorithms (level 3): We experimented with using the EM algorithm to learn the parameters of both our proposed LfO-DBN model (Figure 2) and a IOHMM. We ran 20 iterations of the EM algorithm (we used the EM implementation of the BNT software package [27]), and we assumed the internal state of the expert can take \( l = 4 \) different values.

For testing how well a learning method \( M \) learns the behavior of an agent \( C \), we used a leave-one-out technique: for each map \( m \), we use method \( M \) to learn behavior from the traces of agent \( C \) in all maps except for \( m \). This results in a learned agent \( A \). We then compared the behavior of \( A \) against \( C \) in the following way:
Table 6: Vapnik’s risk between the action distribution predicted by each LfO algorithm and that of the expert, computed based on the current perceptual state $x_t$ (lower is better; best performer per row is highlighted). Right-most column shows the Vapnik’s risk with respect to another set of traces generated by the expert itself.

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>Nnk2</th>
<th>BN</th>
<th>Bnk2</th>
<th>LfO-DBN</th>
<th>IOHMM</th>
<th>Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF</td>
<td>3.18</td>
<td>4.88</td>
<td><strong>2.93</strong></td>
<td>4.60</td>
<td><strong>2.93</strong></td>
<td>2.94</td>
<td>2.41</td>
</tr>
<tr>
<td>SWF</td>
<td>2.51</td>
<td>4.95</td>
<td>2.34</td>
<td>4.01</td>
<td><strong>2.15</strong></td>
<td>2.34</td>
<td>2.12</td>
</tr>
<tr>
<td>RD</td>
<td>8.34</td>
<td>8.56</td>
<td><strong>4.53</strong></td>
<td>6.65</td>
<td><strong>4.55</strong></td>
<td>4.57</td>
<td>4.42</td>
</tr>
<tr>
<td>SRD</td>
<td>7.37</td>
<td>7.88</td>
<td><strong>3.34</strong></td>
<td>3.93</td>
<td><strong>3.34</strong></td>
<td>3.36</td>
<td>3.31</td>
</tr>
<tr>
<td>SL</td>
<td>6.35</td>
<td>7.10</td>
<td>3.41</td>
<td>4.65</td>
<td><strong>3.33</strong></td>
<td><strong>3.35</strong></td>
<td>3.29</td>
</tr>
<tr>
<td>SSL</td>
<td>6.71</td>
<td>7.70</td>
<td>3.07</td>
<td>4.19</td>
<td><strong>3.08</strong></td>
<td><strong>3.11</strong></td>
<td>3.00</td>
</tr>
<tr>
<td>ZZ</td>
<td>5.65</td>
<td>7.23</td>
<td>3.34</td>
<td>4.25</td>
<td><strong>3.25</strong></td>
<td>3.30</td>
<td>3.16</td>
</tr>
<tr>
<td>SEQ</td>
<td>7.90</td>
<td>7.93</td>
<td>4.13</td>
<td>5.19</td>
<td><strong>4.13</strong></td>
<td>4.17</td>
<td>3.45</td>
</tr>
<tr>
<td>Avg.</td>
<td>6.00</td>
<td>7.03</td>
<td>3.39</td>
<td>4.68</td>
<td><strong>3.34</strong></td>
<td>3.39</td>
<td>3.15</td>
</tr>
</tbody>
</table>

- When using classification accuracy, we take trace $LT^m_C$ and feed the perception state step by step to agent $A$, asking $A$ to generate the corresponding action $Y$. We then use Equation 3 to determine classification accuracy.

- When using Vapnik’s Risk, we generate a trace of length 1000 using the learned agent $A$ in map $m$ ($T^m_A$), and then compare it with the expected behavior of the expert, $LT^m_C$, using Equation 12.

- When using Vapnik’s rate, we generate a trace of length 1000 using the learned agent $A$ in map $m$ ($T^m_A$), and then learn an LfO-DBN using this trace. Then compute Vapnik’s Risk (Equation 17) of this LfO-DBN with respect to trace $LT^m_C$.

Looking at the classification accuracy results (Table 5), we can see that Neural Networks (NN), and Bayesian Networks (BN) can predict the behavior of WF and SWF (which are deterministic and that do not require internal memory) very well. Also, we see that when moving to algorithms based on time-windows (Nnk2, and Bnk2), the predictive accuracy improves significantly (except for the simple WF and SWF, where Nnk2 and Bnk2 have some overfitting). Nnk2 and Bnk2 can predict the behavior of the SL, SSL and ZZ experts with very high accuracy. Overall, we see that Nnk2 and LfO-DBN achieve the highest classification accuracy overall.
Table 7: Vapnik’s rate between the action distribution predicted by different LfO algorithms and that of the expert, computed based on an LfO-DBN (lower is better). Right-most column shows the Vapnik’s rate with respect to another set of traces generated by the expert itself.

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>Nnk2</th>
<th>BN</th>
<th>BNk2</th>
<th>LfO-DBN</th>
<th>IOHMM</th>
<th>Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF</td>
<td>87.10</td>
<td>184.58</td>
<td>207.02</td>
<td>251.18</td>
<td>85.72</td>
<td>118.63</td>
<td>49.63</td>
</tr>
<tr>
<td>SWF</td>
<td>148.49</td>
<td>151.48</td>
<td>80.77</td>
<td>228.98</td>
<td>148.13</td>
<td>146.95</td>
<td>43.01</td>
</tr>
<tr>
<td>RD</td>
<td>252.15</td>
<td>252.22</td>
<td>242.90</td>
<td>252.42</td>
<td>241.50</td>
<td>246.82</td>
<td>220.17</td>
</tr>
<tr>
<td>SRD</td>
<td>251.54</td>
<td>251.09</td>
<td>235.74</td>
<td>246.48</td>
<td>211.39</td>
<td>243.45</td>
<td>157.17</td>
</tr>
<tr>
<td>SL</td>
<td>250.45</td>
<td>249.86</td>
<td>240.16</td>
<td>251.38</td>
<td>180.06</td>
<td>181.34</td>
<td>76.46</td>
</tr>
<tr>
<td>SSL</td>
<td>250.75</td>
<td>249.79</td>
<td>250.83</td>
<td>252.02</td>
<td>184.81</td>
<td>249.69</td>
<td>77.23</td>
</tr>
<tr>
<td>ZZ</td>
<td>250.83</td>
<td>251.83</td>
<td>247.97</td>
<td>249.73</td>
<td>241.20</td>
<td>215.60</td>
<td>73.97</td>
</tr>
<tr>
<td>SEQ</td>
<td>252.10</td>
<td>251.85</td>
<td>97.44</td>
<td>221.10</td>
<td>126.07</td>
<td>64.23</td>
<td>24.95</td>
</tr>
<tr>
<td>Avg.</td>
<td>217.93</td>
<td>230.34</td>
<td>200.35</td>
<td>244.16</td>
<td>177.36</td>
<td>183.34</td>
<td>90.32</td>
</tr>
</tbody>
</table>

Thus, looking at those results, we might believe that Nnk2 (neural networks) performs identically than an LfO-DBN. However, as we showed above, classification accuracy is not representative of how well an LfO agent has learned a stochastic behavior. Table 6 (showing results using Vapnik’s risk), gives us a better idea of how well the stochastic behaviors (RD, SRD, SL and SSL) were learned. Here we can see that only LfO-DBNs, IOHMMs and BNs learn action distributions that resemble those of the expert.

Table 7 shows a comparison of all the Bayesian models we used in our experiments using the Vapnik’s rate (which basically learns an LfO-DBN from traces of the learning agent and determines the likelihood of traces generated by the expert according to that LfO-DBN). Vapnik’s rate shows that there is a significant difference in the performance of BN approaches (BN, BNk2) and DBN ones (LfO-DBN, IOHMM) when learning behaviors of level 3 (SL, SSL, ZZ). Moreover, LfO-DBNs can learn the behaviors in our experiment in a more accurate way than other common DBN models such as IOHMMs. Moreover, we would like to point out that Vapnik’s rate depend on learning an LfO-DBN from traces generated from the learning agent. This process in itself depends on running the EM algorithm, which might fall into local minima. Thus, in order to get a more reliable estimator of Vapnik’s rate in practice, it should be computed several times, and keep the minimum value obtained. Values in Table 7 represent the minimum obtained after computing Vapnik’s rate twice.
In summary, we can conclude the following: algorithms like Neural Networks (typically used in the LfO literature) can only learn deterministic behaviors that do not require memory of past states. Neural Networks can be augmented using time windows, which can expand the range of behaviors that can be learned, but they are still restricted to deterministic behaviors. Methods like Bayesian Networks can capture stochastic behaviors, but not behaviors that require memory of past events. Dynamic Bayesian Networks can learn a wider spectrum of behaviors than other algorithms.

6. Conclusions

Despite the considerable amount of interest and work on learning from observation, the field lacks a unified framework for comparing different approaches. The main goal of this paper is to put forward a proposal to fill in that gap, and present a unified framework for learning from observation, consisting of three main components: a) a statistical model based on Dynamic Bayesian Networks (DBNs) that sheds some light on the formal differences between LfO and other forms of learning, b) a classification of the different levels of difficulty of the tasks attempted by LfO, and c) a collection of evaluation metrics for different LfO tasks. Finally, we have presented an empirical evaluation of two of the main claims underlying our model. Namely, that supervised learning algorithms are not appropriate for some LfO tasks, and that standard metrics such as classification accuracy cannot always accurately determine whether two behaviors are similar or not when these behaviors are stochastic or require memory of past states.

Our experimental evaluation suggests that our evaluation metrics better reflect the performance of LfO algorithms than typical metrics used in the LfO literature: our results show that standard metrics such as classification accuracy cannot accurately determine whether two behaviors are similar or not when these behaviors are stochastic or require memory of past states. The alternative metrics proposed in this paper, based Vapnik’s risk or Vapnik’s rate can better determine behavior similarity in these situations.

The proposed DBN framework makes explicit which are the key challenges in LfO: learning algorithms that can handle the hidden internal state of the expert, or that can learn dependencies between the behavior of the expert and not just the current state, but also past states.

Moreover, note that the DBN framework presented in this paper is not to be seen as a practical approach to LfO but as a theoretical model to under-
stand LfO and its differences with respect to traditional supervised machine learning. In other words, our model is one step forward in understanding LfO, on making explicit which are the key challenges, on providing a unified methodology for evaluating the performance of LfO algorithms, but does not include practical algorithms to solve the LfO problems, which are beyond the scope of this paper.

As part of our future work, we want to explore the creation of new algorithms to LfO that can learn the same range of behaviors as our LfO-DBN model, but that are practical and offer better scalability. Additionally, we will continue our study of performance metrics for LfO, with the goal of reaching agreement in the LfO community over a set of standard metrics. For example, our current proposed metric for behaviors of level 3 requires training an LfO-DBN, which is a computationally expensive process for complex behaviors. As part of our future work, we would like to explore the possibility of defining metrics that do not require training complex DBNs, but that are still able to accurately measure the performance of LfO algorithms when learning tasks of level 3 (memory-based behavior).

Acknowledgements. Research partially supported by Spanish grant TIN-2011-27479-C04-04.

Appendix A. Proof of Proposition 1

Proposition 1 in Section 4.2.1 stated that given two agents, \( B_C \) and \( B_A \), then the following holds:

\[
R(A) = H(\rho_C(X), \rho_A(X)) + R^*(A)
\]

where \( R(A) \) is the risk measure of the behavior of agent \( B_A \) with respect to the behavior of \( B_C \) when computed from the probability distribution functions \( \rho_A \) and \( \rho_C \) that govern their behaviors, and \( R^*(A) \) is the risk when computed from the conditional probability distributions. Let us present a proof to such statement.

\[
R(A) = - \int_{X \times Y} f_C(x, y) \log(f_A(x, y)) \, dx \, dy =
\]

\[
= - \int_{X \times Y} f_C(x, y) \log(f_{A,X}(x)f_{A,Y|X}(y)) \, dx \, dy =
\]
\[
= - \int_{X \times Y} f_C(x, y) \log(f_{A,X}(x)) \, dx \, dy - \int_{X \times Y} f_C(x, y) \log(f_{A,Y|x}(y)) \, dx \, dy
\]

Let’s write
\[
R(A) = R1 + R2,
\]
where
\[
R1 = - \int_{X \times Y} f_C(x, y) \log(f_{A,X}(x)) \, dx \, dy
\]
and
\[
R2 = \int_{X \times Y} f_C(x, y) \log(f_{A,Y|x}(y)) \, dx \, dy
\]
Let’s analyze each term separately. First,
\[
R1 = - \int_{X \times Y} f_C(x, y) \log(f_{A,X}(x)) \, dx \, dy =
\]
\[
= - \int_X \left[ \int_Y f_C(x, y) \log(f_{A,X}(x)) \, dy \right] \, dx =
\]
\[
= - \int_X \log f_{A,X}(x) \left[ \int_Y f_C(x, y) \, dy \right] \, dx =
\]
\[
= - \int_X f_{C,X}(x) \log f_{A,X}(x) \, dx = -E_{\rho_C(X)}[f_{A,X}]
\]
This means that term R1 is the risk of the marginal distribution on component X, \(\rho_A(X)\) (with respect to the marginal distribution \(\rho_C(X)\) of the actor). The second term \(R2\) can be written as:
\[
R2 = - \int_{X \times Y} f_C(x, y) \log f_{A,Y|x}(y) \, dx \, dy =
\]
\[
- \int_{X \times Y} f_{C,X}(x) f_{C,Y|x}(y) \log f_{A,Y|x}(y) \, dx \, dy =
\]
\[
- \int_X f_{C,X}(x) \left[ \int_Y f_{C,Y|x}(y) \log f_{A,Y|x}(y) \, dy \right] \, dx = R^*(A)
\]
\[
\square
\]
References


