To appear in: *Robotics And Autonomous Systems*
special issue on Robot Learning, 1995.

An Approach to Learning Mobile Robot Navigation

Sebastian Thrun
Universität Bonn
Institut für Informatik III
Römerstr. 164, 53117 Bonn, Germany
E-mail: thrun@carbon.informatik.uni-bonn.de, thrun@cs.cmu.edu

Abstract

This paper describes an approach to learning a simple indoor robot navigation task through trial-and-error. A mobile robot, equipped with visual, ultrasonic and laser sensors, learns to servo to a designated target object. In less than ten minutes of operation time, the robot is able to navigate to a marked target object in an office environment. The central learning mechanism is the explanation-based neural network learning algorithm (EBNN). EBNN initially learns function purely inductively using neural network representations. With increasing experience, EBNN employs domain knowledge to explain and to analyze training data in order to generalize in a more knowledgeable way. Here EBNN is applied in the context of reinforcement learning, which allows the robot to learn control using dynamic programming.

Keywords: explanation-based learning, mobile robots, machine learning, navigation, neural networks, perception

1 Introduction

Throughout the last decades, the field of robotics has produced a large variety of approaches for the control of complex physical manipulators. Despite significant progress in virtually all aspects of robotics science, most of today's robots are specialized to perform a narrow set of tasks in a very particular kind of environment. Most robots employ specialized controllers that are carefully designed by hand, using extensive

knowledge of the robot, its environment and the task is shall perform. If one is interested in building autonomous multi-purpose robots, such approaches face some serious bottlenecks.

- **Knowledge bottleneck.** Designing a robot controller requires prior knowledge about the robot, its environment and the tasks it is to perform. Some of the knowledge is usually easy to obtain (like the kinematic properties of the robot), but other knowledge might be very hard to obtain (like certain aspects of the robot dynamics, or the characteristics of the robot's sensors). Moreover, certain knowledge (like the particular configuration of the environment or the particular task one wants a robot to do) might not be accessible at all at the design-time of the robot.
- **Engineering bottleneck.** Making domain knowledge computer-accessible, *i.e.*, hand-coding explicit models of robot hardware, sensors and environments, has often been found to require tremendous amounts of programming time. As robotic hardware becomes increasingly more complex, and robots are to become more reactive in more complex and less predictable environments, the task of hand-coding a robot controller will become more and more a cost-dominating factor in the design of robots.
- **Tractability bottleneck.** Even if the robot, its environment and its goals can be modeled in sufficient detail, generating control for a general-purpose device has often been found to be of enormous computational complexity (see for example [13, 31]). Moreover, the computational complexity often increases drastically with the complexity of the mechanical device.

Machine learning aims to overcome these limitations, by enabling a robot to collect its knowledge on-the-fly, through real-world experimentation. If a robot is placed in an unknown environment, or faced with a novel task for which no a priori solution is available, a robot that learns shall collect new experiences, acquire new skills, and eventually perform new tasks all by itself. For example, in [9] a robot manipulator is described which learns to insert a peg into a hole without prior knowledge regarding the manipulator or the hole. Maes and Brooks [15] have successfully applied learning techniques to coordinating leg motion for an insect-like robot. Their approach, too, operates in the absence of a model of the dynamics of the system. Learning techniques have frequently come to bear in situations where the physical world is extremely hard to model by hand. For example, Pomerleau describes a computer system that learns to steer a vehicle driving at 55mph on public highways, based on sensor data from a video camera [26]. Learning techniques have also successfully been applied to *speed-up* robot control, by observing the statistical regularities of "typical" situations (like typical robot and environment configurations), and compiling more compact controllers

for the frequently encountered. For example, Mitchell [19] describes an approach in which a mobile robot becomes increasingly reactive, by using observations to compile fast rules out of a database of domain knowledge.

Generally speaking, approaches to machine learning can be divided into two major categories: *inductive learning* and *analytical learning*. Inductive learning techniques, like decision tree learning [27], spline interpolation [7] or artificial neural network learning [29], generalize sets of training examples via a built-in, domain-independent inductive bias. They typically can learn functions from scratch, based purely on observation. Analytical approaches to learning, like explanation-based learning [5, 18, 20], generalize training examples based on domain-specific knowledge. They employ a built-in theory of the domain of the target function for analyzing and generalizing individual training examples. Both families of approaches are characterized by opposite strengths and weaknesses. Inductive learning mechanisms are more general in that they can learn in the absence of prior knowledge. In order to do so, however, they require large amounts of training data. Analytical learning techniques learn from much less training data, relying instead on the learner's internal domain theory. They hence require the availability of an appropriate domain theory.

In mobile robot domains, large amounts of training data is typically hard to obtain due to the slowness of actual robot hardware. Therefore, analytical learning techniques seem to have a clear advantage. Their strong requirement for accurate domain knowledge, however, has found to be a severe obstacle in applying analytical learning to realistic robotics domains [12, 19]. In this paper we present the explanation-based neural network (EBNN) learning algorithm [22, 37] which integrates both analytical and inductive learning. It smoothly blends both learning principles depending on the quality of the available domain knowledge. This paper also reports experimental results for learning robot navigation in an office environment.

2 Integrating Inductive and Analytical Learning

EBNN is a hybrid learning mechanism, which integrates inductive and analytical learning. Before explaining EBNN, let us briefly consider its components: (inductive) neural network learning and (analytical) explanation-based learning.

2.1 Neural Network Backpropagation

Artificial neural networks (see [10, 30, 40] for an introduction) consist of a set of simple, densely interconnected processing units. These units transform signals in a non-linear

(a) Training examples

is light	*has handle*	*made of Styrofoam*	*upward concave*	*color*	*open vessel*	*flat bottom*	*is expensive*	*is_cup?*
yes	yes	no	no	blue	yes	yes	yes	yes
no	no	yes	yes	red	no	yes	no	no
yes	no	yes	yes	red	yes	no	no	no
no	no	yes	yes	green	yes	yes	no	yes
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(b) Target concept

is_cup? :- *is_liftable, holds_liquid*
is_liftable :- *is_light, has_handle*
is_liftable :- *made_of_Styrofoam, upward_concave*
holds_liquid :- *open_vessel, flat_bottom*

Figure 1: **The cup example.** (a) Some training examples, (b) the unknown target concept.

way. Neural networks are nonparametric estimators which can fit smooth functions based on input-output examples. The internal parameters of a neural network, which are adapted in the process of function fitting (learning), are called weights and biases. The Backpropagation algorithm [29], which is the currently most widely used supervised neural network learning algorithm, learns purely inductively, by observing statistical regularities in the training patterns.

Consider the example depicted in Fig. 1. Suppose we are facing the problem of learning to classify cups. More specifically, imagine we want to train an artificial neural network, denoted by f, which can determine the cupness of an object based on the features *is_light*, *has_handle*, *made_of_Styrofoam*, *color*, *upward_concave*, *open_vessel*, *flat_bottom*, and *is_expensive*. One way to learn the new concept is to collect training examples of cups and non-cups, and employ the Backpropagation procedure to iteratively refine the weights of the target network. Such a learning scheme is purely inductive. It can learn functions from scratch, in the absence of any domain knowledge. The generalization accuracy of the trained neural network depends on the number of training examples, and typically a large set of training examples is required to fit a target function accurately.

2.2 Explanation-Based Learning

To illustrate explanation-based learning (EBL) [5, 20], which is the most widely studied analytical approach to machine learning, imagine one were given a theory of the domain. In the symbolic regime, in which analytical approaches have been studied the most, such a *domain theory* typically consists of a set of rules, which can *explain why* training examples are members of the target function. EBL generalizes training examples via the following three-step procedure:

1. **Explain.** Explain the training example by chaining together domain theory rules.
2. **Analyze.** Analyze the explanation in order to find the *weakest precondition* under which this explanation leads to the same result. Features that play no part in an explanation are not included in this weakest precondition. The generalized explanation forms a rule, which generalizes the training example.
3. **Refine.** Add this generalized explanation to the rule memory.

To illustrate this procedure, consider the cup example depicted in Table 1. Let us assume the learner is given a domain theory which contains the following rules (*cf.* Fig. 2):

is_liftable :- *is_light, has_handle*
is_liftable :- *made_of_Styrofoam, upward_concave*
holds_liquid :- *open_vessel, flat_bottom*
is_cup? :- *is_liftable, holds_liquid*

This domain theory can explain every positive example in the training set. For example, the explanation of the first training example in Table 1 reads: *The object is light and has a handle. Hence is is liftable. It also has an open vessel and a flat bottom, and therefore can hold a liquid. Consequently it is a cup.* As the reader may notice, the color of the cup does not play a part in this explanation. Therefore, any object with a different color but otherwise identical features will also be a cup. This reasoning step illustrates the weakest precondition operator in symbolic EBL, which basically analyzes the explanation to find the weakest precondition under which this very explanation still applies. The complete analysis of the example explanation leads to the following generalized rule:

$$\textit{is\_cup?} \;:-\; \textit{is\_light} \wedge \textit{has\_handle} \wedge \textit{open\_vessel} \wedge \textit{flat\_bottom}$$

Figure 2: **A Symbolic domain theory for explaining cups.** Notice that only the grey shaded features are relevant, since only those occur in a domain theory rule. EBL benefits from rules with sparse dependencies.

Explaining and analyzing the second positive example Table 1 yields the rule

$$\textit{is\_cup?} \quad :- \quad \textit{made\_of\_Styrofoam} \wedge \textit{upward\_concave} \wedge \textit{open\_vessel} \wedge \textit{flat\_bottom}$$

which, together with the first rule, describes already the target concept correctly. The drastically improved generalization rate, when compared with pure inductive techniques, follows from the fact that the learner has been provided with a complete and correct theory of the domain. In its most pure form, EBL can acquire only rules that follow from the initial domain theory – it does not learn at the knowledge level [6]. Therefore, EBL methods have often been applied to *speed-up learning* [18]. Recent work has produced EBL methods can learn from domain knowledge that is only partially correct [3, 24, 25, 28].

2.3 The Explanation-Based Neural Network Learning Algorithm

Most approaches to EBL require that the domain theory be represented by symbolic rules, and, moreover, be correct and complete. In EBNN, the domain theory is represented by artificial neural networks which are learned from training data, hence might be incorrect. To illustrate EBNN, let us assume one had already learned a *neural network domain theory* for the cup example considered here. As in other approaches

to EBL, the domain theory shall allow the learner to reason about training examples. Thus, an appropriate domain theory could, for example, represent each individual inference step in the logical derivation of the target concept by a separate neural network. In our cup example, a collection of three networks for predicting whether an object is liftable (denoted by f_1), for determining if an object can hold a liquid (f_2), and for predicting the cupness of an object as a function of the two intermediate concepts *is_liftable* and *holds_liquid* (f_3) forms an appropriate domain theory (*cf.* Fig. 3), as it suffices to reason about the cupness of an object. It is, however, not necessarily correct, as the domain theory networks themselves may have been learned through previous observations.

Recall the goal of learning is to learn an artificial neural network f, shown in Fig. 4, which shall directly determine if an object is a cup based on the observed object features. How can the neural network domain theory be employed to refine the target network f? EBNN learns analytically by explaining and analyzing training examples via the following three-step procedure:

1. **Explanation.** In EBNN the domain theory is used to *explain* training examples by chaining together multiple steps of neural network inferences. An *explanation* is a post-facto prediction of the training example in terms of the domain theory. In our example, the domain theory network f_1 is used to predict the degree to which the object is liftable, network f_2 is employed to predict if the object can hold a liquid, and finally the cupness of an object is evaluated via network f_3. This neural network inference, which forms the *explanation* of a training example, explains *why* a training example is a member of its class in terms of the domain theory. The explanation sets up the inference structure necessary for analyzing and generalizing this training example.

2. **Analysis.** The explanation is analyzed in order to generalize the training example in feature space. Unlike symbolic approaches to EBL, which employ the weakest precondition operator, EBNN generalizes by extracting *slopes* of the target function. More specifically, EBNN extracts the output-input slopes of the target concept by computing the first derivative of the neural network explanation. These slopes measure, according to the domain theory, how for a given training example infinitesimal small changes in feature space will change the output of the target function. Since neural network functions are differentiable, these slopes can easily be obtained by computing the first derivative of the neural network explanation.

   In the cup example, EBNN extracts slopes from the explanation composed of the three domain theory networks f_1, f_2, and f_3. The output-input derivative of f_1 predicts, how infinitesimally small changes in the input space of f_1 will change the degree as to which f_1 predicts an object to be liftable. Likewise, the

Figure 4: **The target network.** The target network maps input features directly to the target concept.

---

derivatives of f_2 and f_3 predict the effect of small changes in their input spaces on their vote. Chaining these derivatives together results in slopes which measure how infinitesimal small changes of the individual example features will change the cupness of the object.

Slopes guide the generalization of training examples in feature space. For example, irrelevant features, *i.e.*, features whose values are irrelevant for the cupness of an object (*e.g.*, the features *color* and *is_expensive* in the cup example above)

will have approximately zero slopes. On the other hand, large slopes indicate important features, since small changes of the feature value will have a large effect on the target concept according to the neural network domain theory. Hence, if the domain theory is sufficiently accurate, the extracted slopes provide a knowledgeable means to generalize a training example in feature space.

3. **Refinement.** Finally, EBNN refines the weights and biases of the target network. Fig. 5 summarizes the information obtained by the inductive and the analytical component. Inductive learning yields a target *value* for each individual training example. The analytical extraction of slopes returns estimates of the *slopes* of the target function. When updating the weights of the network, EBNN minimizes a combined error function which takes both value error and slope error into account, denoted by E_{values}, and E_{slopes}, respectively:

$$E = E_{\text{values}} + \alpha E_{\text{slopes}}$$

Here α is a gain parameter trading off value fit versus slope fit. Thus, weight updates in EBNN seek to fit both the inductively derived target values, as well as the analytically derived target slopes. Gradient descent in weight space is employed to iteratively minimize E. Notice that in our implementation we used a modified version of the Tangent-Prog algorithm [32] to refine the weights and biases of the target network [17].

2.4 Accommodating Imperfect Domain Theories

Thus far, we have not addressed the potential damage arising from poor domain theories. Because the domain theory itself is learned from training data and thus might be erroneous, the analytically extracted slopes will only be approximately correct. Especially in the early phase of learning the domain theory will be close to random, and as a consequence the extracted slopes may be misleading. EBNN provides a mechanism to prevent from the damaging effects arising from inaccurate slopes. The accuracy of slopes is estimated by the prediction accuracy of the domain theory: The more accurate a domain theory predicts the target value of the training example, the allegedly more accurate are the slopes extracted from this domain theory. Here the accuracy is the root-mean square deviation of the true target value and the prediction by the domain theory, denoted by δ.

When refining the weights and biases of the target network, δ determines the weight of the target slopes α

$$\alpha = \max\left[0, 1 - \frac{\delta}{\delta_{\max}}\right] = \begin{cases} 1 - \dfrac{\delta}{\delta_{\max}}, & \text{if } \delta < \delta_{\max} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Figure 5: **Fitting values and slopes in EBNN:** Assume f is an unknown target function for which three examples $\langle x_1, f(x_1)\rangle$, $\langle x_2, f(x_2)\rangle$, and $\langle x_3, f(x_3)\rangle$ are given. Based on these points the learner might generate the hypothesis g. If the slopes are also known, the learner can do much better: h.

The constant $\delta_{\max}$ denotes the maximum anticipated prediction error, which is used for normalization and which must be set by hand. This weighting scheme ensures that accurate slopes will have a large weight in training, whereas inaccurate slopes will be gradually ignored. We consistently found in our experiments that the arbitration scheme, called LOB* [22], is crucial for successful learning in cases where the domain theory is poor and misleading, since it diminishes the damaging effect arising from misleadingly wrong domain theories [22, 23].

This completes the description of the EBNN learning mechanism. To summarize, EBNN refines the target network using a combined inductive-analytical mechanism. Inductive training information is obtained through observation, and analytical training information is obtained by explaining and analyzing these observations in terms of the learner's prior knowledge. Inductive and analytical learning are traded off dynamically by an arbitration scheme LOB* which estimates the current accuracy of the domain theory.

Both methods of learning, inductive and analytical, play an important role in EBNN. Once a reasonable domain theory is available, EBNN benefits from its analytical component, since it replaces the pure syntactical bias of neural networks gradually by a more knowledgeable, domain-specific bias. The result is an improved generalization accuracy from less training data. Of course, analytical learning requires the availability of a domain theory which can explain the target function. If such a domain theory is weak or not available, EBNN degrades to a purely inductive neural network learner. It is the inductive component of EBNN which ensures that learning is possible even in the total absence of domain knowledge.

3 Reinforcement Learning

In order to learn sequences of actions, as required for controlling a robot, we applied EBNN in the context of $\mathcal{Q}$-Learning [41, 42]. Recently, $\mathcal{Q}$-Learning and other related reinforcement learning and dynamic programming algorithms [2, 34] have been applied successfully to robot manipulation [8], robot control [16] and games [35].

In essence, $\mathcal{Q}$-Learning learns control policies for partially controllable Markov chains with delayed pay-off (penalty/reward) [1, 2]. It does this by constructing a value function $\mathcal{Q}(s,a)$, which maps perceptual sensations, denoted by s, and actions, denoted by a, to task-specific utility values. Such a value function, once learned, ranks actions according to their goodness: The larger the future pay-off for picking an action a when s is observed, the larger its value $\mathcal{Q}(s,a)$.

Formally, the value function approximates the sum of discounted future penalty/reward one is expected to receive upon executing action a when s is observed, and acting optimally thereafter

$$\hat{\mathcal{Q}}(s,a) \;=\; E\left[\sum_{\tau=0}^{\infty} \gamma^{\tau} \cdot r(\tau)\right] \tag{2}$$

Here γ ($0 \leq \gamma \leq 1$) is a discount factor that, if $\gamma < 1$, favors reward reaped sooner in time. The value $r(\tau)$ denotes the pay-off at time τ. Throughout this paper, we will make the restrictive assumption that the goal of learning is to achieve a certain goal configuration. The pay-off r is zero, except for the end of an episode, where it is $+100$, if the robot achieved its goal, and -100 if it failed. Notice that the framework of partially controllable Markov chains implies that the state of the environment is *fully observable*. Hence, knowing the most recent observation is sufficient to predict the effect of actions on future states and pay-offs.

Initially, all values $\mathcal{Q}(s,a)$ are set to zero. $\mathcal{Q}$-Learning approximates $\hat{\mathcal{Q}}(s,a)$ on-line, through real-world experimentation. Suppose, in the course of learning, the learner executes action a_t when s_t is observed, which leads to a new sensation s_{t+1} and, if a_t terminates the learning episode, a penalty of -100 or a reward of $+100$. This observation is used to update $\mathcal{Q}(s,a)$:

$$\mathcal{Q}^{\text{target}}(s_t,a_t) \;\longleftarrow\; \begin{cases} +100 & \text{if } a_t \text{ final action, robot reached its goal} \\ -100 & \text{if } a_t \text{ final action, robot failed} \\ \\ \gamma \cdot \max\limits_{a \text{ is action}} \mathcal{Q}(s_{t+1},a) & \text{otherwise} \end{cases} \tag{3}$$

The scalar α ($0 < \alpha \leq 1$) is a *learning rate*, which is typically set to a small value that is decayed over time. This simple update rule has been shown—under certain conditions

Figure 6: **$\mathcal{Q}$-Learning.** The goal of $\mathcal{Q}$-Learning is to learn a value function which ranks actions according to their goodness. This value function is learned recursively: $\mathcal{Q}(s_1, a_1)$ is updated based on later values $\mathcal{Q}(s_2, a_2), \mathcal{Q}(s_3, a_3), \ldots$ in the episode.

---

concerning the exploration scheme, the environment, the learning rate and the representation of $\mathcal{Q}$—to converge to the optimal value function $\hat{\mathcal{Q}}$, and hence to produce optimal policies that maximize the future, discounted pay-off [11, 41].

$\mathcal{Q}$-Learning propagates knowledge in the value function from the end of an episode gradually to the beginning. This is because at the end of an episode, $\mathcal{Q}$ is learned directly based on the final pay-off. In between, $\mathcal{Q}$ is updated based on later $\mathcal{Q}$-values. This process of backing up values can be slow, since they are propagated in the reverse order of the robot's operation. For this and other reasons, $\mathcal{Q}$-Learning has often been combined with Sutton's *temporal difference learning* [33], which basically mixes multiple value estimates. The extended $\mathcal{Q}$-Learning rule, which was used in all experiments reported in this paper, reads:

$$\mathcal{Q}^{\text{target}}(s_t, a_t) \longleftarrow \begin{cases} +100 & \text{if } a_t \text{ final action, robot reached its goal} \\ -100 & \text{if } a_t \text{ final action, robot failed} \\ \gamma \cdot \left[(1-\lambda) \cdot \max\limits_{a \text{ is action}} \mathcal{Q}(s_{t+1}, a) + \lambda \cdot \mathcal{Q}^{\text{target}}(s_{t+1}, a_{t+1}) \right] & \text{otherwise} \end{cases} \tag{4}$$

Here λ ($0 \leq \lambda \leq 1$) is a gain parameter which trades off the recursive component and the non-recursive component in the update equation. This extended update rule propagates values faster, but the target values may be inaccurate due to non-optimal action choices within the episode. Fig. 6 illustrates a $\mathcal{Q}$-Learning episode.

The application of EBNN to $\mathcal{Q}$-Learning is straightforward. The target function in $\mathcal{Q}$-Learning is $\mathcal{Q}$, and training examples in the standard, inductive regime are of the form

$$\left\langle (s_t, a_t), \quad \mathcal{Q}^{\text{target}}(s_t, a_t) \right\rangle \tag{5}$$

(*cf.* Eqs. (3) and (4)). To explain such training examples, the learner has to have a neural network domain theory which allows to post-facto predict episodes. In our implementation, this domain theory consists of a collection of predictive *action models*, denoted by f_a (one for each action a), which predict the sensation and the penalty/reward value at time $t+1$ based on the sensations at time t. To simplify the notation, predictions of sensations will be denoted by f_a^{sens}, and predictions of pay-off values will be denoted by $f_a^{\text{pay-off}}$.

To see how training examples are explained, consider the first two cases in the plain $\mathcal{Q}$-Learning rule (3). Since the final penalty/reward can directly be predicted by the action models, a single prediction constitutes already the full explanation:

$$f_{a_t}^{\text{pay-off}}(s_t)$$

To explain the third case in Eq. (3), the action model and the $\mathcal{Q}$-network are chained together. Recall the action model f_{a_t} predicts the sensation s_{t+1} based on s_t, and the value function $\mathcal{Q}$ estimates the target value $\mathcal{Q}^{\text{target}}(s_t, a_t)$ based on the sensation s_{t+1}. Hence, the explanation of the third case in Eq. (3) is:

$$\mathcal{Q}(f_{a_t}^{\text{sens}}(s_t), \hat{a}) \qquad \text{with} \quad \hat{a} = \underset{a \text{ is action}}{\operatorname{argmax}} \mathcal{Q}(s_{t+1}, a)$$

Notice that the value function is both the target function and part of the domain theory, as it is required for explaining its own training examples.

The analysis of this explanation, which involves computing the first derivative of neural network functions, is straightforward:

$$\nabla_{s_t} \mathcal{Q}^{\text{target}}(s_t, a_t) \quad \longleftarrow \quad \begin{cases} \left.\dfrac{\partial f_{a_t}^{\text{pay-off}}(s)}{\partial s}\right|_{s_t} & \text{if } a_t \text{ final action} \\ \gamma \cdot \left.\dfrac{\partial \mathcal{Q}(s, \hat{a})}{\partial s}\right|_{s_{t+1}} \cdot \left.\dfrac{\partial f_{a_t}^{\text{sens}}(s)}{\partial s}\right|_{s_t} & \text{otherwise} \end{cases} \tag{6}$$

Here an expression of the type $\left.\frac{\partial g(x)}{\partial x}\right|_{x_0}$ denotes the first derivative of a function g with respect to the input variable x, taken at x_0. The target slope, $\nabla_{s_t} \mathcal{Q}^{\text{target}}(s_t, a_t)$, is an n-dimensional vector with n being the dimension of the perceptual space (*i.e.*, the input space of the action model networks f_a). The second case in (6) involves multiplying a $n \times n$-dimensional matrix with an n-dimensional vector.

Explaining training examples constructed according to the combined $\mathcal{Q}$-Learning and

Figure 7: **EBNN and reinforcement learning.** General-purpose action models are used to derive slope information for training examples.

---

temporal difference learning rule (4) is analogous:

$$\nabla_{s_t}\mathcal{Q}^{\text{target}}(s_t,a_t) \longleftarrow \begin{cases} \left.\dfrac{\partial f_{a_t}^{\text{pay-off}}(s)}{\partial s}\right|_{s_t} & \text{if } a_t \text{ final action} \\ \gamma\cdot\left[(1-\lambda)\cdot\left.\dfrac{\partial \mathcal{Q}_a(s,\hat{a})}{\partial s}\right|_{s_{t+1}} + \right. & \\ \qquad \left.\lambda\cdot\nabla_{s_{t+1}}\mathcal{Q}^{\text{target}}(s_{t+1},a_{t+1})\right]\cdot\left.\dfrac{\partial f_{a_t}^{\text{sens}}(s)}{\partial s}\right|_{s_t} & \text{otherwise} \end{cases} \tag{7}$$

Eq. (7) can be understood as the first derivative of the learning rule (4), in which unknown, missing derivatives are replaced by neural network domain theory derivatives. A neural network explanation of an episode is graphically depicted in Fig. 7. As in the cup example, the process of explaining and analyzing individual training episodes produces target slopes for the value function $\mathcal{Q}$. EBNN's training examples for $\mathcal{Q}$ are hence of the type

$$\left\langle\ (s_t,a_t),\ \ \mathcal{Q}^{\text{target}}(s_t,a_t),\ \ \nabla_{s_t}\mathcal{Q}^{\text{target}}(s_t,a_t)\ \right\rangle$$

(*cf.* (5)) which are obtained via Eqs. (4) and (7). As described in the previous section, when updating the $\mathcal{Q}$-network, its weights and biases are refined such as to fit both the target values $\mathcal{Q}^{\text{target}}(s_t,a_t)$ and the target slopes $\nabla_{s_t}\mathcal{Q}^{\text{target}}(s_t,a_t)$. The arbitration factor α (*cf.* Eq. (1)) is adjusted according to LOB*, using the prediction accuracy of the action models as a measure for the accuracy of the slopes.

4 Results

4.1 Experimental Setup

In this section we will present some empirical results obtained in a mobile robot navigation domain. *Xavier*, the robot at hand, is shown in Fig. 8. It is equipped with a ring of 24 sonar sensors, a laser light-stripper (range finder), and a color camera mounted on a pan/tilt unit. Sonar sensors return approximate echo distances, along with noise. Xavier's 24 sonar sensors provide a full 360 degree range of view. The laser light-stripper measures distances more accurately, but its perceptual field is limited to a small range in front of the robot.

In the experiments reported here, the task of the robot was to learn to navigate to a specifically marked target location in a laboratory environment. In some experiments, the location of the marker (a green soda can) was fixed throughout the course of learning, in others it was moved across the laboratory and only kept fixed for the duration of a single training episode. Sometimes parts of the environment were blocked by obstacles. The marker was detected using a visual object tracking routine that recognized and tracked the marker in real-time, making heavily use of the robot's pan/tilt unit. Every 3 seconds the robot could chose one of seven applicable actions, ranging from sharp turns to straight forward motion. In order to avoid collisions, the robot employed a pre-coded obstacle avoidance routine. Whenever the projected path of the robot was blocked by an obstacle, the robot decelerated and, if necessary, changed its motion direction (regardless of the commanded action). Xavier was operated continuously in real-time. Each learning episode corresponded to a sequence of actions which started at a random initial position and terminated either when the robot lost sight of the target object, for which it was penalized, or when it halted in front of the marker, in which case it was rewarded. Penalty/reward was only given at the end of an episode, and $\mathcal{Q}$-Learning with EBNN was employed to learn action policies.

In our implementation, percepts were mapped into a 46-dimensional perceptual space, comprising 24 logarithmically scaled sonar distance measurements, 10 locally averaged laser distance scans, and an array of 12 camera values that indicated the horizontal angle of the marker position relative to the robot. Hence, each action model mapped 46 sensor values to 46 sensor predictions (15 hidden units), plus a prediction of the immediate penalty/reward value. The models were learned beforehand with the Backpropagation training procedure, using cross-validation to prevent from over-fitting. Initially, we used a training corpus of approximately 800 randomly generated training examples for training the action models, which was gradually increased through the course of this research to 3,000 examples, taken from some 700 episodes. These training examples were distributed roughly equally among the seven action model networks.

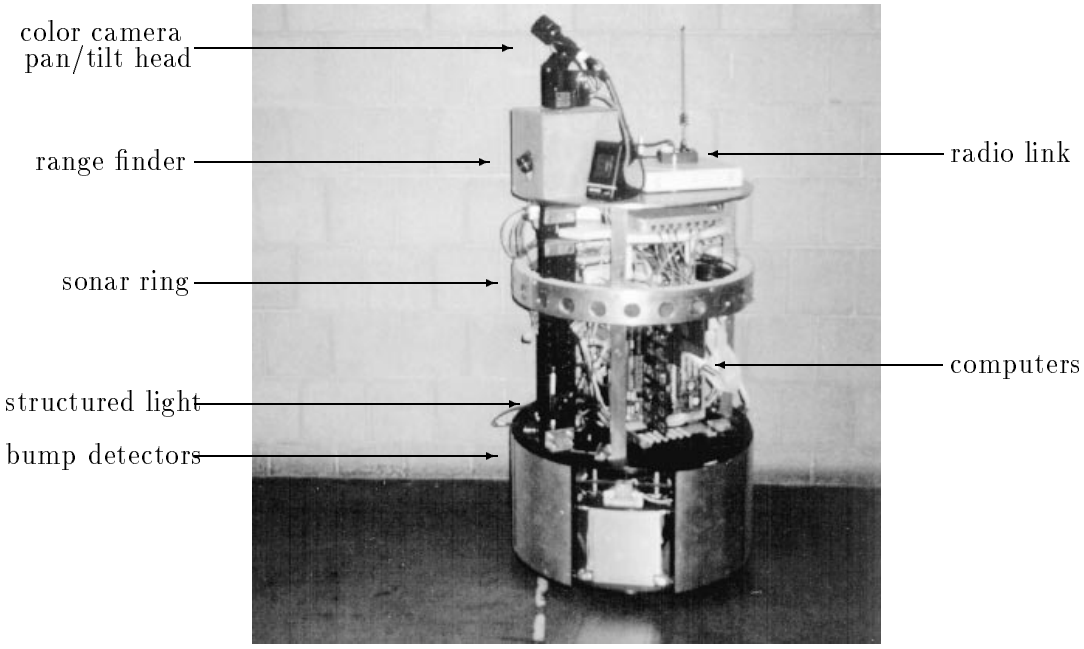

Figure 8: **The Xavier robot.** Xavier has been built by and is property of Carnegie Mellon University, Pittsburgh, USA.

Of course, Xavier's predictive action models face highly stochastic variables. There are many unknown factors which influence the actual sensations. Firstly, sensors are generally noisy, *i.e.*, there is a certain probability that a sensor returns corrupted values. Secondly, the obstacle avoidance routine was very sensitive to small and subtle details in the real world. For example, if the robot faces an obstacle, it is often very hard to predict whether its obstacle avoidance behavior will make it turn left or right. Thirdly, the delay in communication, imposed by the tetherless Ethernet link, turned out to be rather unpredictable. These delays, which extended the duration of actions, were anywhere in the range of 0.1 up to 3 seconds. For all those reasons, the domain theory functions f_a captured only *typical* aspects of the world by modeling the average outcome of actions, but were clearly unable to predict the details accurately. Empirically we found, however, that they were well-suited for extracting appropriate slopes. Fig. 9 gives an example for a slope array extracted from a domain theory network.

In all our experiments the action models were trained first, prior to learning $\mathcal{Q}$, and frozen during learning control. When training the $\mathcal{Q}$ networks (46 input, eight hidden and one output unit), we explicitly memorized all training data, and used a replay technique similar to "experience replay" described in [14]. This procedure memorizes all past experiences explicitly. After each learning episode, it re-estimates the target values of the $\mathcal{Q}$ function by recursively replaying past episodes, as if they had just

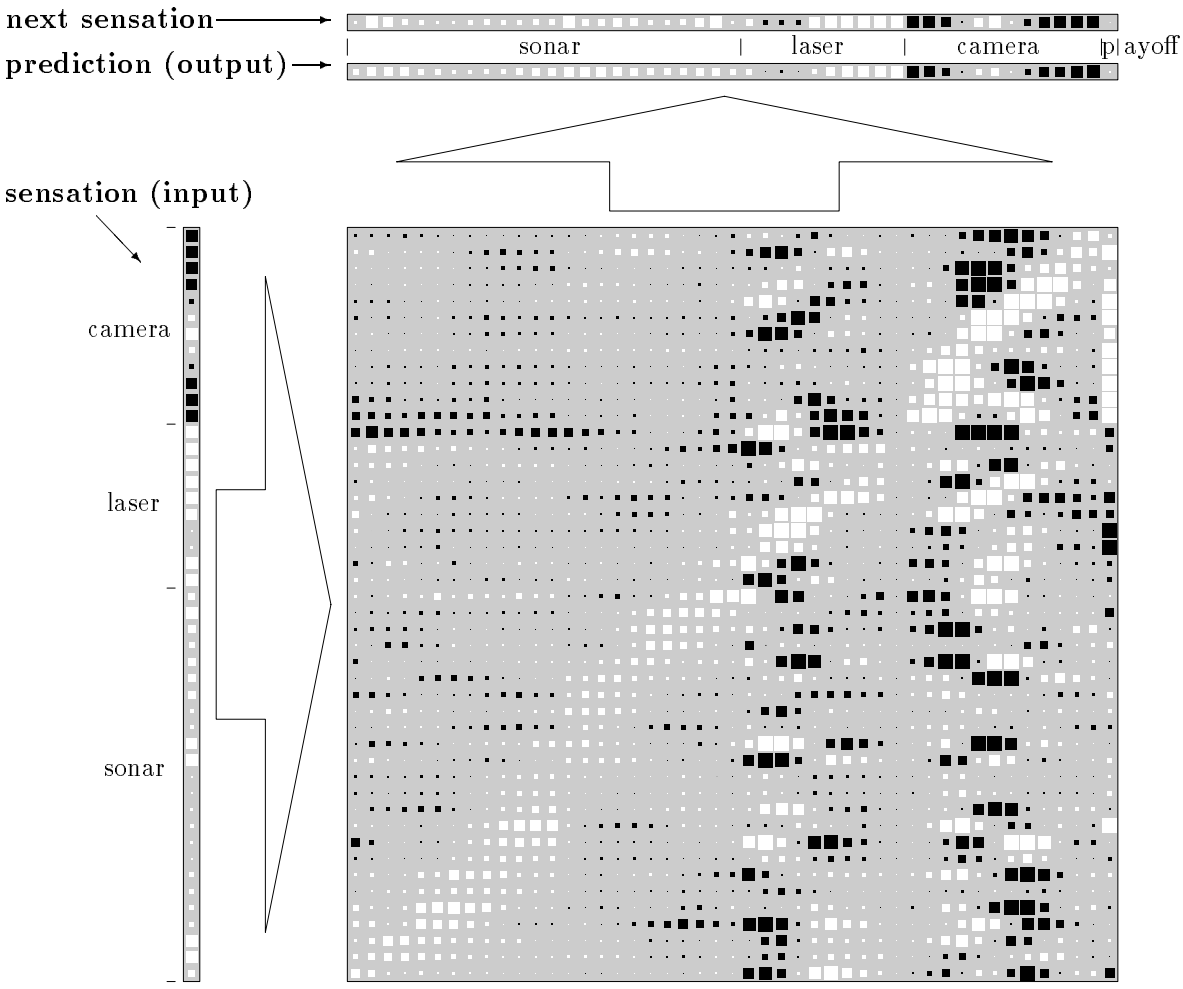

Figure 9: **Prediction and slopes.** A neural network action model predicts sensations and penalty/reward for the next time step. The large matrix displays the output-input slopes of the network. White boxes refer to positive and black boxes to negative values. Box sizes indicate absolute magnitudes. Notice the bulk of positive gradients along the main diagonal, and the cross-dependencies between different sensors.

---

been observed. Experience replay makes extensive use of the training examples, hence allowing for more accurate evaluations of the minimum requirements on data volume. In all our experiments the update parameter γ was set to 0.9, and λ was set to 0.7, which was empirically found to work best. We performed a total of seven complete learning experiments, each consisting of 25-40 episodes.

4.2 Experimental Results

To evaluate the importance of the action models for rapid learning, we ran two sets of experiments: one in which no prior knowledge was available, and one where Xavier had access to the pre-trained action models. One of the latter experiments is summarized in Fig. 10. In this figure, we visualized several learning episodes seen from a bird eye's view, using a sonar-based technique for building occupancy maps described in [4, 38]. In all cases Xavier learned to navigate to a static target location in less than 19 episodes (with action models) and 24 episodes (without action models). Each episode lasted for between two and eleven actions. Xavier consistently learned to navigate to arbitrary target locations (which was required in five out of seven experiments) always in less than 25 (35, respectively) episodes. The reader should notice the small number of training examples required to learn this task. Although the robot faced a high-dimensional sensation space, it always managed to learn the task in less than 10 minutes of robot operation time, and, on average, less than 20 training examples per $\mathcal{Q}$-network. Of course, the training time does not include the time for collecting the training data of the action models. Almost all training examples for the action models, however, were obtained as a side effect when experimenting with the robot.

When testing our approach, we also confronted Xavier with situations which were not part of its training experience, as shown in Fig. 11. In one case, we kept the location of the marker fixed and moved it only in the testing phase. In a second experiment, we blocked the robot's path by large obstacles, even though it had not experienced obstacles during training. It was here that the presence of appropriate action models was most important. While without prior knowledge the robot consistently failed to approach the marker under these new conditions, it reliably ($>90\%$) managed this task when it was trained with the help of the action model networks. Obviously, this is because in EBNN the action models provides a knowledgeable bias for generalization to unseen situations.

4.3 Simulation Results

In order to investigate EBNN more thoroughly, and in particular to study the robustness of EBNN to errors in the domain theory, we ran a more systematic study in a simulated robot domain. Simulation has the advantage that large sets of experiments are easy to perform, under repeatable and well-controlled conditions.

The task, which bears close resemblance to the navigation task described above, is depicted in Fig. 12a. The robot agent, indicated by the small circle, has to navigate to the fixed goal location (large circle) while avoiding collisions with the obstacle and the surrounding walls. Its sensors measure the distances and the angles to both the center

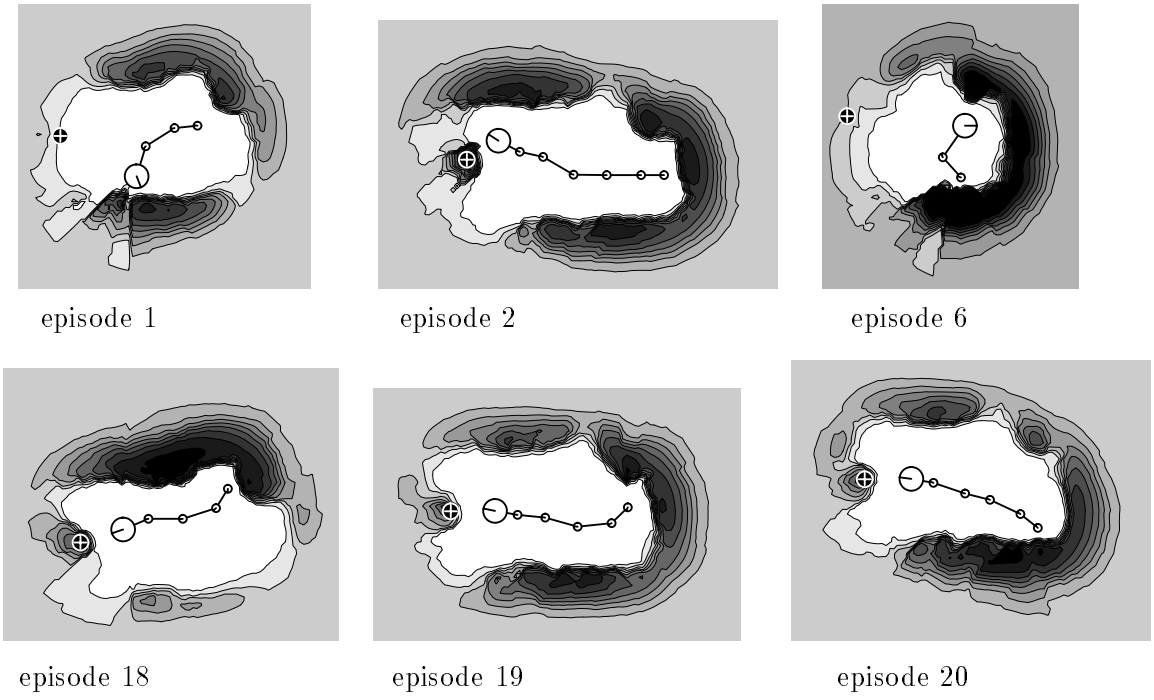

Figure 10: **Learning navigation.** Traces of three early and three late episodes are shown. Each diagram shows a two-dimensional occupancy maps of the world, which have been constructed based on sonar information. Bright regions indicate free-space and dark regions indicate the presence of obstacles. Note that the location of the target object (marked by a cross) is held constant in this experiment.

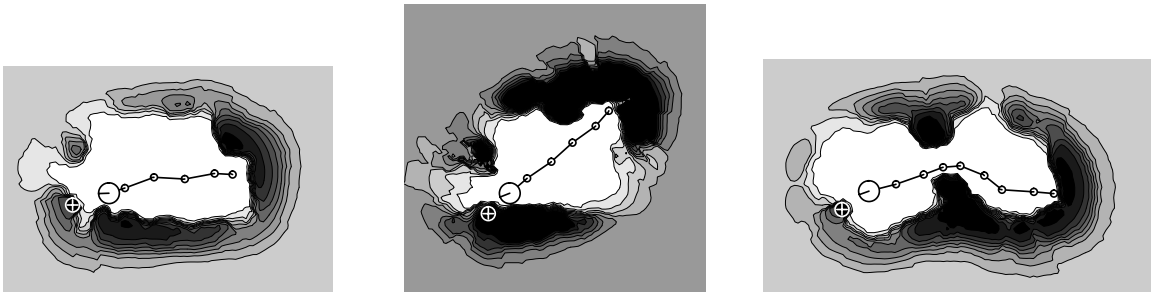

Figure 11: **Testing navigation.** After training, the location of the target object was moved. In some experiments, the path of the robot was also blocked by obstacles. Unlike plain inductive neural network learning, EBNN almost always manages these cases successfully.

---

of the obstacle and the center of the goal, relative to the agent's view. Five different actions are available to the robot, as depicted in Fig. 12b. Note that this learning task

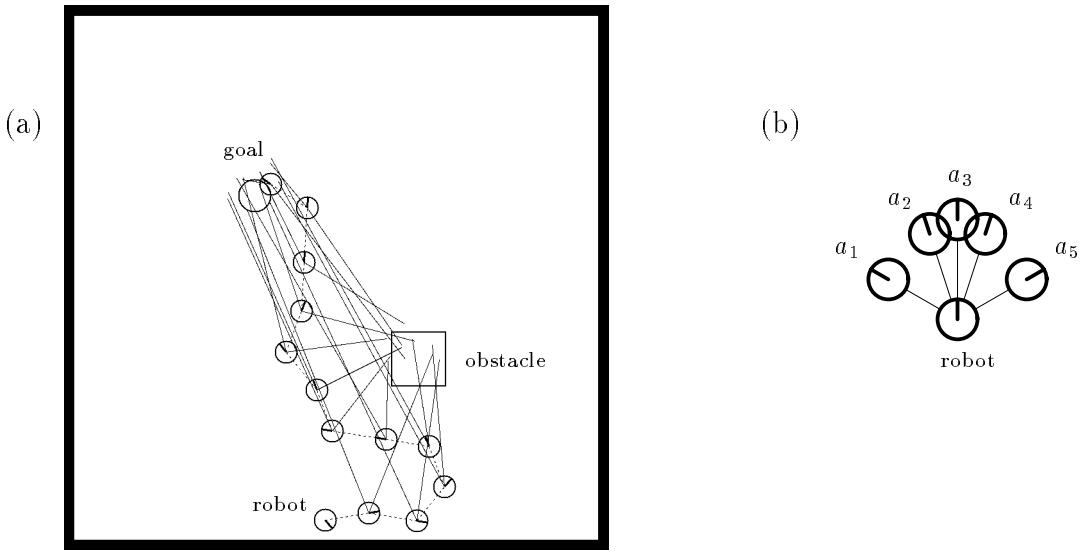

Figure 12: **The simulated robot world.** (a) The agent has to learn a function which maneuvers the robot to the goal location starting at arbitrary initial location, without colliding with walls or the obstacle. The sensations S are the distance and relative angle to goal and obstacle, respectively. The thin lines indicate the 1-step predictions of the sensations by well-trained neural action models. (b) shows the action space.

is completely deterministic, lacking any noise in the agent's percepts or control. This task also differed from the task described above in that only 4 perceptual values were provided, instead of 46, making the learning problem slightly harder.

The learning setup was analogous to the robot experiments described in the previous section. Before learning a value function, we trained 5 neural network action models, one for each individual action. Subsequently, 5 $\mathcal{Q}$-functions were trained to evaluate the values of each action. In this implementation we used a real-valued approximation scheme using nearest-neighbor generalization for the $\mathcal{Q}$-functions [22]. This scheme was empirically found to outperform Backpropagation. Parameters were otherwise identical to the navigation task described above.

Experiment 1: The role of analysis. In a first experiment, we were interested in the overall merit of the analytical learning component of EBNN. Thus, the action models were trained in advance, using 8192 randomly generated training examples per network. Fig. 13 shows results for learning control with and without explanation-based learning. The overall-performance is measured by the cumulative probability of success, averaged on an independent, randomly drawn testing set of 20 starting locations, and

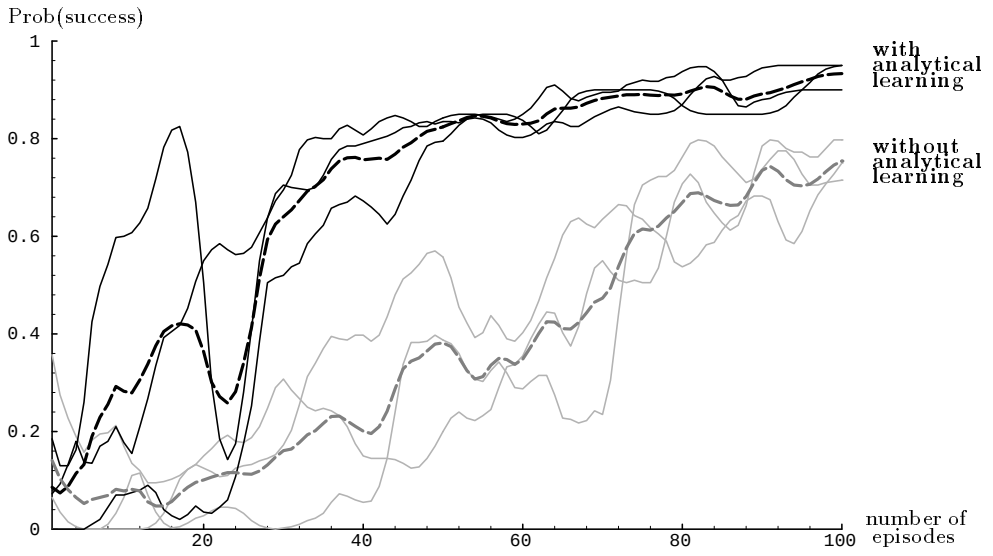

Figure 13: **Performance.** Performance curves for EBNN with (black) and without (gray) analytical training information (slope fitting) for three examples each, measured on an independent set of problem instances. The dashed lines indicate average performance. In this experiment, the agent used well-trained predictive action models as its domain theory.

summed over the first 100 episodes. Both techniques exhibit asymptotically the same performance and learn the desired control function successfully. However, there is a significant improvement in generalization accuracy using EBNN when training data is scarce, similar to the effect noted in the previous sections.

Experiment 2: Weak domain theories. Obviously, EBNN outperformed pure inductive learning because it was given a domain theory trained with a large number of training examples. But how does EBNN perform if the domain theory is weak, and the extracted slopes are misleading? In order to test the impact of weak domain theories, we conducted a series of experiments in which we trained the domain theory networks using 5, 10, 20, 35, 50, 75, 100, 150, and 8192 training examples per action network. Obviously, the less training data, the less accurate the domain theory. Fig. 14 shows the overall accuracy of the individual domain theories as a function of number of training patterns supplied. Fig. 15 displays the learning performance resulting from EBNN when it used these different domain theories. As can be seen, EBNN degraded gracefully to the performance of a pure inductive learner as the accuracy of the domain theory decreases. Even if the domain theory was close-to-random and therefore delivered misleading slopes, the overall performance of EBNN did not drop beyond that of the purely inductive approach. This finding illustrates that EBNN can operate

robustly over a broad range of domain theories, from strong to random. The reader may notice that this effect was also observed in other experiments involving mobile robot perception, described in [23].

An interesting scaling benchmark of EBNN is the reduction in the number of training examples required to achieve a certain level of performance. Fig. 16 displays this average speed-up factor for EBNN relative to plain, inductive Backpropagation. The speed-up factor is the quotient of the average number of training examples required by EBNN with and without the analytical learning component. This is shown for the varying target performance accuracies $20\%, 30\%, ..., 80\%$. As is easily seen, this speed-up increases with the accuracy of the domain theory. For the strongest domain theories here, the speed-up factor is approximately 3.4, which compares well to what we would expect given that EBNN produces 5 times as many constraint on the target function as plain Backpropagation.

Experiment 3: The role of LOB*. As pointed out above, EBNN has been found to degrade gracefully to pure inductive learning, as the quality of its domain theory decreases. This finding can be attributed to the LOB* mechanism, which heuristically weights the analytical and inductive learning components based on the observed accuracy of explanations. In order to evaluate the importance of this mechanism, we repeated the previous experiment, replacing the dynamic weighting scheme for trading off values and slopes (*cf.* Eq. (1)) with a single, static value for α. Fig. 17 shows the resulting performance graphs.

By comparing Fig. 17 with Fig. 15, one can see that for well-trained domain theories the performance of EBNN without LOB* approximately equals its performance with LOB*. When the domain theory is poor, however, performance degrades much more dramatically when LOB* is not used. In some cases (when domain theories were trained with 5 or 10 training examples), performance was even much worse than that of plain inductive learning. These results indicate the importance of effective mechanisms which mediate the effects of severe domain theory errors. They also illustrate that in cases where the domain knowledge is poor, a pure analytical learner would be hopelessly lost. EBNN recovers from poor domain theories because of its inductive component, which enables it to overturn misleading bias extracted from inaccurate prior knowledge. These findings match our experimental results reported in [23].

5 Conclusion

In this paper we have reported results obtained for applying the explanation-based neural network learning algorithm to problems of indoor robot navigation. Despite the high-dimensional sensor spaces faced by the Xavier robot, it consistently managed

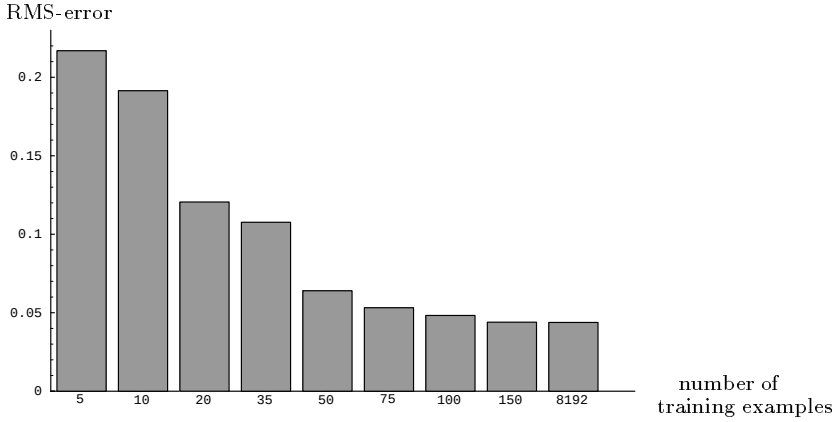

Figure 14: **Accuracy of different domain theories.** The root mean square error of the domain theory networks, measured on a large, independent testing set, is shown as a function of training set size.

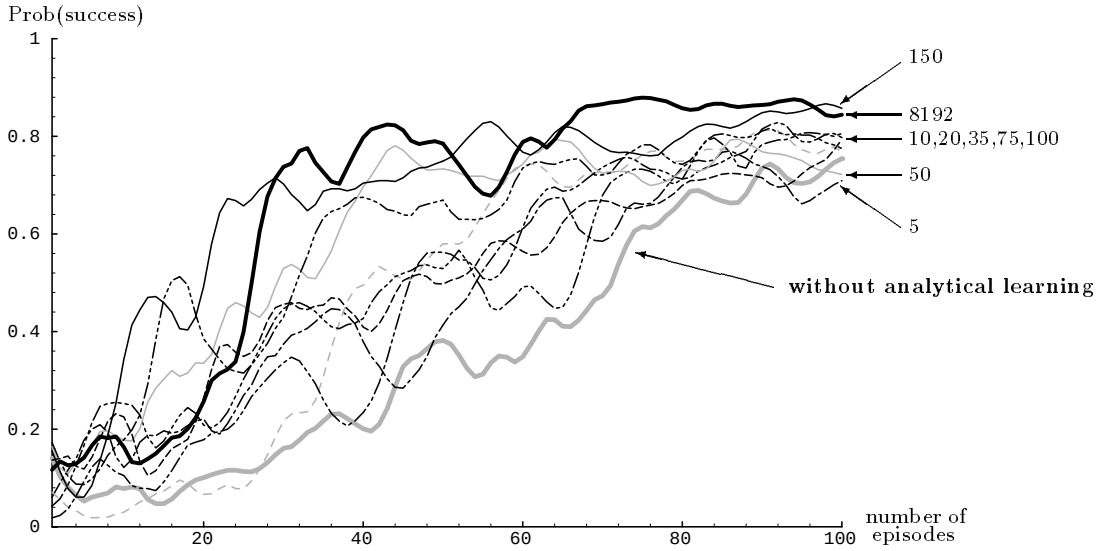

Figure 15: **Scaling.** How does domain knowledge improve generalization? Averaged results for EBNN domain theories of differing accuracies, pre-trained with 5 to 8,192 training examples. In contrast, the bold gray line reflects the learning curve for pure inductive Q-Learning. (b) Same experiments, but without dynamically weighting the analytical component of EBNN by its accuracy. All curves are averaged over 3 runs and are also locally window-averaged. The performance (vertical axis) is measured on an independent test set of starting positions.

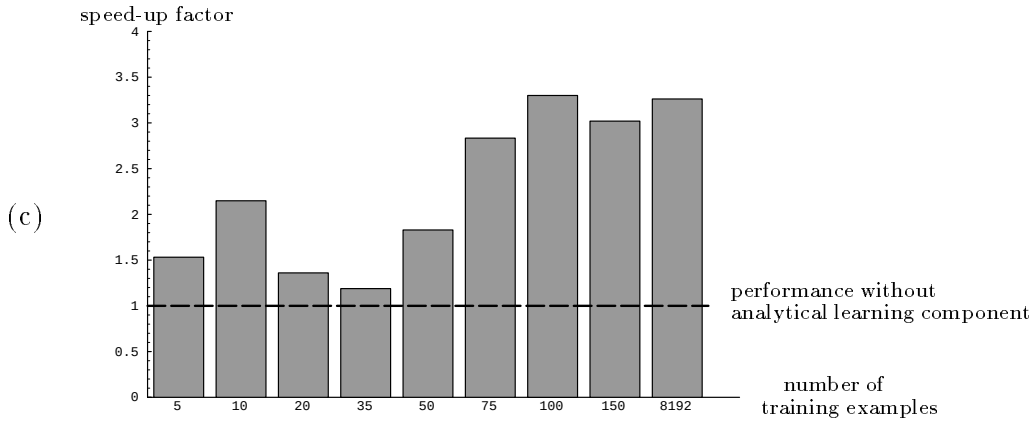

Figure 16: **Speed-up due to learning analytically.** Speed-up factor as a function of the different domain theories described in the text. The speed-up factor is computed relative to pure inductive learning in the same domain.

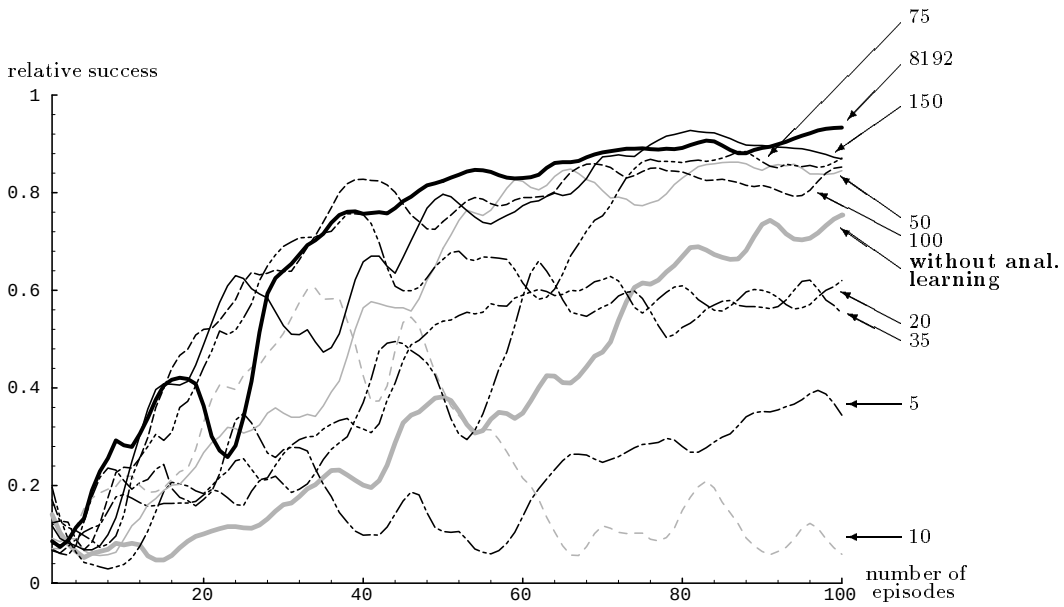

Figure 17: **EBNN without LOB*.** If inductive and analytical learning are always weighted equally, weak domain theories hurt the performance, dropping well below that of pure inductive learning.

to learn to navigate to a designated target object in less than 10 minutes operation time. More elaborate simulation studies illustrate the scaling properties of the EBNN learning algorithm, and the importance of the previously learned domain theory for successful generalization.

There are many open questions that warrant future research, the most important of which are: How does EBNN scale to more complex tasks, involving much longer sequences of actions? Will EBNN be able to effectively transfer large chunks of knowledge across robot control learning tasks? How stable and robust is control learned by EBNN? Can we effectively employ action models with variable temporal resolution in robot control learning problems? How can EBNN utilize domain knowledge provided by human designers?

A key advantage of the approach taken is its generality. No Xavier-specific domain knowledge (like knowledge of specific sensors, the motion dynamics, or the environment) has been encoded manually. Rather, the robot has the ability to collect its domain knowledge by itself. As a consequence, the program can adapt to unforeseen changes, and can adapt to a whole variety of tasks. Tabula-rasa learning approaches, which lack any domain-specific knowledge, seem to be little likely to scale to complex robotic tasks. EBNN, thus, allows to collect domain knowledge on-the-fly, and to employ it for biasing generalization in future control learning tasks. The experimental results reported here indicate that an appropriate domain theory can reduce training time significantly—an effect which has also been demonstrated in different studies in computer perception [21, 23] and vision [39] domains, and chess [36].

Acknowledgment

The author thanks Tom Mitchell, Ryusuke Masuoka, Joseph O'Sullivan, Reid Simmons, and the CMU mobile robot group for their invaluable advice and support. This work was supported in part by grant IRI-9313367 from the US National Science Foundation to Tom Mitchell.

References

[1] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report COINS 91-57, Department of Computer Science, University of Massachusetts, MA, August 1991.

[2] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, to appear.

[3] Francesco Bergadano and Aiello Giordana. *Guiding Induction with Domain Theories*, pages 474–492. Morgan Kaufmann, San Mateo, CA, 1990.

[4] Joachim Buhmann, Wolfram Burgard, Armin B. Cremers, Dieter Fox, Thomas Hofmann, Frank Schneider, Jiannis Strikos, and Sebastian Thrun. The mobile robot Rhino. *AI Magazine*, 16(1), to appear.

[5] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[6] Thomas G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1:287–316, 1986.

[7] Jerome H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, March 1991.

[8] Vijaykumar Gullapalli. *Reinforcement Learning and its Application to Control.* PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1992.

[9] Vijaykumar Gullapalli, Judy A. Franklin, and Hamid Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems*, 272(1708):13–24, February 1994.

[10] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation.* Addison-Wesley Pub. Co., Redwood City, California, 1991.

[11] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Technical Report 9307, Department of Brain and Cognitive Sciences, Massachusetts Institut of Technology, July 1993.

[12] J. Laird, E. Yager, C. Tuck, and M. Hucka. Learning in tele-autonomous systems using Soar. In *Proceedings of the 1989 NASA Conference of Space Telerobotics*, 1989.

[13] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[14] Long-Ji Lin. *Self-supervised Learning by Reinforcement and Artificial Neural Networks.* PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1992.

[15] Pattie Maes and Rodney A. Brooks. Learning to coordinate behaviors. In *Proceedings Eighth National Conference on Artificial Intelligence*, pages 796–802, Cambridge, MA, 1990. AAAI, The MIT Press.

[16] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. December 1990.

[17] Ryusuke Masuoka. Noise robustness of EBNN learning. In *Proceedings of the International Joint Conference on Neural Networks*, October 1993.

[18] Steven Minton. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, 1988.

[19] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of 1990 AAAI Conference*, Menlo Park, CA, August 1990. AAAI, AAAI Press / The MIT Press.

[20] Tom M. Mitchell, Rich Keller, and Smadar Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[21] Tom M. Mitchell, Joseph O'Sullivan, and Sebastian Thrun. Explanation-based learning for mobile robot perception. In *Workshop on Robot Learning, Eleventh Conference on Machine Learning*, 1994.

[22] Tom M. Mitchell and Sebastian Thrun. Explanation-based neural network learning for robot control. In S. J. Hanson, J. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 287–294, San Mateo, CA, 1993. Morgan Kaufmann.

[23] Joseph O'Sullivan, Tom M. Mitchell, and Sebastian Thrun. Explanation-based neural network learning from mobile robot perception. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*. Oxford University Press, to appear.

[24] Dirk Ourston and Raymond J. Mooney. Theory refinement with noisy data. Technical Report AI 91-153, Artificial Intelligence Lab, University of Texas at Austin, March 1991.

[25] Michael J. Pazzani, Clifford A. Brunk, and Glenn Silverstein. A knowledge-intensive approach to learning relational concepts. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 432–436, Evanston, IL, June 1991.

[26] Dean A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. Technical Report CMU-CS-89-107, Computer Science Dept. Carnegie Mellon University, Pittsburgh PA, 1989.

[27] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[28] Paul S. Rosenbloom and Jans Aasman. Knowledge level and inductive uses of chunking (EBL). In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 821–827, Boston, 1990. AAAI, MIT Press.

[29] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II.* MIT Press, 1986.

[30] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. The basic ideas in neural networks. *Communications of the ACM*, 37(3):87–92, March 1994.

[31] Jacob T. Schwartz, Micha Scharir, and John Hopcroft. *Planning, Geometry and Complexity of Robot Motion.* Ablex Publishing Corporation, Norwood, NJ, 1987.

[32] Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. Tangent prop – a formalism for specifying selected invariances in an adaptive network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

[33] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

[34] Richard S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 471–478, San Mateo, 1991. Morgan Kaufmann.

[35] Gerald J. Tesauro. Practical issues in temporal difference learning. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 259–266, San Mateo, CA, 1992. Morgan Kaufmann.

[36] Sebastian Thrun. Learning to play the game of chess. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, San Mateo, CA, 1995. Morgan Kaufmann. (to appear).

[37] Sebastian Thrun and Tom M. Mitchell. Integrating inductive neural network learning and explanation-based learning. In *Proceedings of IJCAI-93*, Chamberry, France, July 1993. IJCAI, Inc.

[38] Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 1993. (to appear). Also appeared as Technical Report IAI-TR-93-7, University of Bonn, Dept. of Computer Science III.

[39] Sebastian Thrun and Tom M. Mitchell. Learning one more thing. Technical Report CMU-CS-94-184, Carnegie Mellon University, Pittsburgh, PA 15213, September 1994.

[40] Philip D. Wasserman. *Neural computing: theory and practice.* Von Nostrand Reinhold, New York, 1989.

[41] Christopher J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England, 1989.

[42] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.