to appear in: *Proceedings of the IEEE Conference on Intelligent Robots and Systems*
to be held Sept. 12-16, 1994, Munich, Germany.

A Lifelong Learning Perspective for Mobile Robot Control

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

Abstract—**Designing robots that learn by themselves to perform complex real-world tasks is a still-open challenge for the field of Robotics and Artificial Intelligence. In this paper we present the robot learning problem as a lifelong problem, in which a robot faces a collection of tasks over its entire lifetime. Such a scenario provides the opportunity to gather general-purpose knowledge that transfers across tasks. We illustrate a particular learning mechanism, explanation-based neural network learning, that transfers knowledge between related tasks via neural network action models. The learning approach is illustrated using a mobile robot, equipped with visual, ultrasonic and laser sensors. In less than 10 minutes operation time, the robot is able to learn to navigate to a marked target object in a natural office environment.**

I. 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 have been carefully designed by hand, using extensive knowledge of the robot, its environment and its task. 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 (*e.g.*, [18]). 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 a novel, unknown environment, or faced with a novel task for which no a priori solution is known, a robot that learns can collect new experiences, acquire new skills, and eventually perform new tasks all by itself. For example, in [5] a robot manipulator is described which learns to insert a peg in a hole without prior knowledge regarding the manipulator or the hole. Maes and Brooks [8] 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 (*e.g.*, the characteristics of noisy sensors). For example, Pomerleau describes a computer system that learns to steer a vehicle driving at 55mph on public highways, based on input sensor data from a video camera [15]. 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 [11] 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.

However, there is a principle shortcoming in most of to date's rigorous learning approaches. Most of the robot control learning

Figure 1: Episode and Q-networks in Q-Learning.

approaches focus on learning to achieve single, isolated performance tasks. If one is interested in learning with a minimum amount of initial knowledge, as is often the case in approaches to robot learning, such approaches have a critical limiting factor: the number of training examples required for successful generalization. The more complex the task at hand and the lesser is known about the problem beforehand, the more training data is necessary to achieve the task. In many robotics domains, the collection of training data is an expensive undertaking due to the slowness of robotics hardware. Hence, it does not surprise that the time required for real-world experimentation has frequently been found to be the limiting factor that prevents rigorous machine learning techniques from being truly successful in robotics.

The task of learning from scratch can be significantly simplified by considering robots that face whole collections of control learning problems over their entire lifetime. In such a *lifelong robot learning scenario* [27], learning tasks are related in that they all play in the same environment, and that they involve the the same robot hardware. Lifelong learning scenarios open the opportunity for the transfer of knowledge across tasks. Complex tasks, which might require huge amounts of training data when faced in isolation, can conceivably be achieved much faster if a robot manages to exploit previously learned knowledge. For example, a lifelong learning robot might acquire general-purpose knowledge about itself and its environment, or acquire generally useful skills that can be applied in the context of multiple tasks. Such functions, once learned, can be applied to speed up learning in new tasks. Hence, a lifelong learning perspective, as proposed in this paper, promises to reduce the number of training examples required for successful learning, and hence to scale machine learning technology to more complex robot scenarios.

In this paper we will present one particular candidate approach to the lifelong learning problem: The explanation-based neural network learning algorithm (EBNN) [13, 26, 27]. EBNN uses a hybrid learning strategy to generalize training data. On the one hand, it allows to learn functions inductively from scratch, just like neural network Backpropagation [17]. On the other hand, EBNN also allows to learn task-independent *domain knowledge*, which applies to multiple tasks. More specifically, in the robot control learning scenarios studied in this paper, domain knowledge is represented by neural network *action models*, which model the effect of the robot's actions. These models are independent of the particular control learning task at hand, and they are acquired over the lifetime of the robot. When learning control, they are used to *analyze* observations, in order to generalize better to unseen situations.

Since in this paper we are interested in learning robot control, we will describe EBNN in the context of Q-Learning [28]. Q-Learning is a popular method for learning to select actions from delayed and sparse reward. To illustrate the appropriateness of the EBNN learning mechanism for robotics problems, we will describe experimental results obtained in an indoor mobile robot navigation domain.

II. Learning Control

Q-Learning [28] is a reinforcement learning technique that has frequently been applied to robot control. The goal of Q-Learning is to learn a policy for generating whole action sequences, which maximize an externally given *reward function*. In general, the reward may be delayed and/or sparse, adding a significant degree of complexity to the learning problem. For example, in the robot navigation experiments reported here, reward is only received upon reaching a particular goal position, or upon total failure (*i.e.*, loosing a target object out of sight, as described in Sect. IV). The goal of learning, thus, is to construct a reactive controller that carries the robot to its goal position in as few steps as possible.

Q-Learning works as follows. Let S be the set of all possible robot percepts. For simplicity, let us assume that at any (discrete) point in time, the agent can observe the complete state of the world[1], denoted by $s \in S$. Based on this observation, the learner then picks an action a (from a set of actions A). As a result, the world state changes. In addition, the learner receives a scalar *reward value*, denoted by $r(s, a)$ (or r_t, if we refer to

[1] This restrictive assumption is frequently referred to as the *Markov assumption*. See [2, 7, 10, 22] for approaches to reinforcement learning in partially observable worlds.

Figure 2: **EBNN.** General-purpose action models are used to derive slope information for training examples..

the expected reward at a certain time t), which measures its current performance. Throughout this paper we will assume that reward will exclusively be received upon reaching a designated goal location, or upon total failure, respectively.

Formally, in $\mathcal{Q}$-Learning one seeks to find an *action policy* $\pi{:}s{\longrightarrow}a$, *i.e.*, a mapping from environment states s to actions a, which, when applied for action selection, maximize the *cumulative discounted future reward*

$$R \;=\; \sum_{t=t_0}^{\infty} \gamma^{t-t_0} \; r_t. \tag{1}$$

Here γ (with $0{\leq}\gamma{\leq}1$) is a *discount factor* that favors rewards reaped sooner in time. Notice that actions may impact reward many time steps later. Hence, in order to learn to pick reward-maximizing actions, one has to solve a *temporal credit assignment problem.*

The key of $\mathcal{Q}$-Learning is to learn a *value function* for picking actions. A value function, denoted by $\mathcal{Q}{:}S{\times}A{\longrightarrow}\Re$, maps robot percepts $s{\in}S$ and actions $a{\in}A$ to scalar "utility" values. In the ideal case, $\mathcal{Q}(s,a)$ is, after learning, the maximum cumulative, discounted reward one can expect upon executing action a in state s (*cf.* Eq. (1)). Hence, $\mathcal{Q}$ ranks actions according to their reward: The larger the expected cumulative reward for picking action a at state s, the larger its value $\mathcal{Q}(s,a)$. $\mathcal{Q}$, once learned, allows to generate optimal actions by greedily picking the action which maximizes $\mathcal{Q}$ for the current state s:

$$\pi(s) \;=\; \operatorname*{argmax}_{\hat{a}\in A} \mathcal{Q}(s,\hat{a})$$

Initially, all values $\mathcal{Q}(s,a)$ are set to zero. During learning values are refined incrementally, using the following recursive update procedure. Suppose the robot just executed a whole action sequence which, starting at some initial state s_1, led to a final reward $r(s_T, a_T)$. Such an episode is shown in Fig. 1. For all time steps t within this episode, $\mathcal{Q}(s_t,a_t)$ is then updated through mixture of the values of subsequent observation-action pairs, up to the final value at the end of the episode. The exact update equation, combined with a modified temporal differencing rule [24], is:

$$\mathcal{Q}^{\text{target}}(s_t,a_t) = \begin{cases} +100 & \text{if } a_t \text{ final action, robot reached goal} \\ -100 & \text{if } a_t \text{ final action, robot failed} \\ \gamma\cdot\Big[(1{-}\lambda)\cdot \max\limits_{a \text{ action}} \mathcal{Q}(s_{t+1},a) & \\ \qquad +\lambda\cdot\mathcal{Q}^{\text{target}}(s_{t+1},a_{t+1})\Big] & \text{otherwise} \end{cases} \tag{2}$$

Here λ ($0{\leq}\lambda{\leq}1$) is a gain parameter, trading off the recursive component and the non-recursive component in the update equation. If $\lambda{=}0$, update equation (2) estimates the value of picking action a at state s by both the observed immediate reward $r(s,a)$, and the best achievable value at the subsequent states. Notice if $\mathcal{Q}(s,a)$ is represented by a lookup-table, the $\mathcal{Q}$-Learning rule has been shown to converge to a value function that yields optimal policies (under certain conditions concerning the environment, the exploration scheme and the learning rate) [6, 28]. In the experiments reported in this paper, the number of actions is finite, and $\mathcal{Q}$ is represented by neural networks $\mathcal{Q}_a$, one for each action $a{\in}A$.

III. Explanation-Based Neural Network Learning

$\mathcal{Q}$-Learning learns individual policies independently, ignoring the opportunity for the transfer of knowledge across different tasks. An important aspect of successful approaches to the lifelong learning problem is the ability to transfer knowledge to related tasks. In this section we will describe the explanation-based neural network learning algorithm (EBNN), which uses task-independent knowledge to bias generalization when learning robot control. Notice that the EBNN algorithm is a general learning algorithm, albeit the fact that EBNN is described in the context of $\mathcal{Q}$-Learning. A more detailed description can be found in [26, 27].

In order to transfer knowledge, EBNN learns general-purpose *predictive action models*. Action models, denoted by $M{:}S{\times}A{\longrightarrow}A$, model the effect of the robots actions. In EBNN, these models are represented using artificial neural networks, which are trained with observed state transitions using the Backpropagation algorithm [17] (or EBNN itself). Once

Figure 3: **Fitting values and slopes in EBNN:** Let f be the 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 known. Based on these points the learner might generate the hypothesis g. If the slopes (tangents) are also known, the learner can do much better: h.

---

trained, they are used to analyze observed episodes using the following three-step procedure:

1. **Explain.** An explanation is a post-facto prediction of an observed episode. More specifically, when explaining an episode, action models are employed to predict each state s_t from the previously observed state action tuple (s_{t-1}, a_{t-1}), as illustrated in Fig. 2.

2. **Analyze.** The explanation is now analyzed in order to extract the slopes of the target Q function. More specifically, consider the update rule (2). The slope of the target output, $Q^{\text{target}}(s_t, a_t)$, with respect to its input s_t is governed by:

$$\nabla_{s_t} Q^{\text{target}}(s_t, a_t) = \begin{cases} \left.\dfrac{\partial r(M(s,a_t))}{\partial s}\right|_{s_t} & \text{if } a_t \text{ final action} \\ \gamma \cdot \left[(1-\lambda) \left.\dfrac{\partial Q(s, a_{t+1})}{\partial s}\right|_{s_{t+1}} + \right. \\ \left. \lambda \cdot \nabla_{s_{t+1}} Q^{\text{target}}(s_{t+1}, a_{t+1}) \right] \cdot \left.\dfrac{\partial M(s,a_t)}{\partial s}\right|_{s_t} \\ & \text{otherwise} \end{cases} \tag{3}$$

   If both Q and M are correct, Eq. (3) can be viewed as the derivative of Eq. (2) with respect to s with fixed actions. The auxiliary derivatives $\frac{\partial M(s,a_t)}{\partial s}$ and $\frac{\partial Q(s,a_{t+1})}{\partial s}$ are obtained by computing the first derivative of the neural network function which, since neural networks are differentiable functions, is straightforward to obtain. Hence, for each target value of the form $\langle (s_t, a_t), Q^{\text{target}}(s_t, a_t)\rangle$ obtained via Eq. (3), EBNN finds the *slope* of the target function Q at (s_t, a_t), denoted by $\nabla_{s_t} Q^{\text{target}}(s_t, a_t)$. This slope vector informs the learner about the sensitivity of Q with respect to infinitesimal changes in s_t. Clearly, action models M play an integral part in the extraction of slopes.

3. **Learn.** Once the slopes of the target function are known, one can generalize better by incorporating them into the training procedure, as depicted in Fig. 3. In EBNN both the target values $Q^{\text{target}}(s_t, a_t)$ and the target slopes $\nabla_{s_t} Q^{\text{target}}(s_t, a_t)$ are used to refine the weights of the target networks. Hence, a combined error function

$$E \quad = \quad \sum_{\text{patterns}} E_{\text{value}} + \alpha \cdot E_{\text{slope}}$$

   is minimized, which takes both value error, E_{value}, and slope error, E_{slope}, weighted by a gain parameter α into account. Target values are fitted using the Backpropagation algorithm [17]. Target slopes are fitted using the Tangent-Prop algorithm, which is an analogue of Backpropagation for fitting slopes [20].

   Clearly, slopes extracted by EBNN can be wrong. This is because they are computed using artificial neural networks, which themselves are constructed from training examples. Consequently, target slopes can mislead the generalization. EBNN provides a simple but effective mechanism to recover from malicious slopes. Whenever a model is used for predicting the next state, EBNN measures its root mean square prediction error, denoted by δ_t, which is obtained by comparing the real state with the model prediction. This error indicates the correctness of the action model, and hence the accuracy of the slopes. Consequently, it is used to weight the slope information when refining the weights of the target network. More specifically, the *weighting factor* of the tth slope, α_t, when refining the weights, is governed by

$$\alpha_t \quad = \quad \prod_{\tau=t}^{T-1} \left(1 - \frac{\delta_\tau}{\delta_{\max}}\right)$$

   Here δ_t is the RMS error of the model M at (s_t, a_t), and $\delta_{\max}$ is an appropriate normalization factor such that $0 \le \delta_t \le \delta_{\max}$. When refining the weights of the Q-network, the learning rate for the tth slope is multiplied with α_t. This mechanism, which is called LOB*, has empirically been found to be important for successfully learning from weak action models [26].

Notice that EBNN employs a neural network version of explanation-based learning [4, 12]. To summarize, EBNN employs action models to analyze training episodes, and to derive slopes that are used for generalizing these episodes. These slopes generalize training instances in input space, since they indicate how small changes will affect the target function, Q. They are extracted from general-purpose action models, which is acquired and used over the entire lifetime of the robot. Hence, in a lifelong learning context, EBNN transfers knowledge by virtue of its action models.

IV. Results in a Robot Navigation Domain

In this section we will present empirical results obtained in a mobile robot navigation domain. *Xavier*[2], the robot at hand, is shown in Fig. 4. 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. The laser light-stripper measures distances more accurately, but its perceptual field is limited to a small range in front of the robot. The task of the robot was to learn to navigate to a specifically marked target location in a natural laboratory environment. In some experiments, the location of the marker 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 possible actions, ranging from sharp turns to straight 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 Q-Learning with EBNN was employed to determine optimal 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 for the immediate reward. The models were learned beforehand using

[2]Xavier has been built by and is property of Carnegie Mellon University, Pittsburgh, USA.

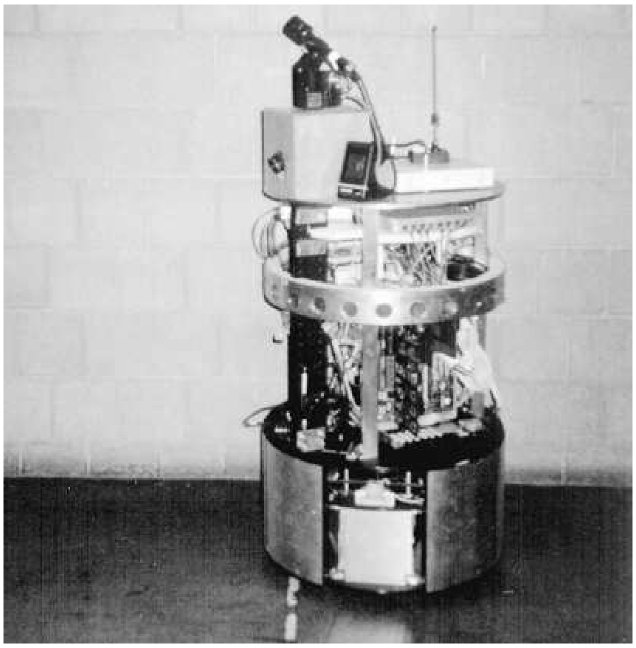

Figure 4: The CMU Xavier robot.

the Backpropagation training procedure, employing a cross-validation scheme to prevent over-fitting the data. Initially, we used a training corpus of approximately 800 randomly generated training examples, 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. In all our experiments the action models were trained first, prior to learning Q, and frozen during learning control. When training the 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 [7]. This procedure memorizes all past experiences explicitly. After each learning episode, it re-estimates the target values of the Q function by recursively replaying past episodes, as if they had just been observed. Experience replay makes extensive use of the training instances, 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.

In order to evaluate the importance of the action models for rapid learning in a lifelong learning setting, 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 experiments is summarized in Fig. 6. In this figure, we visualized several learning episodes viewed from

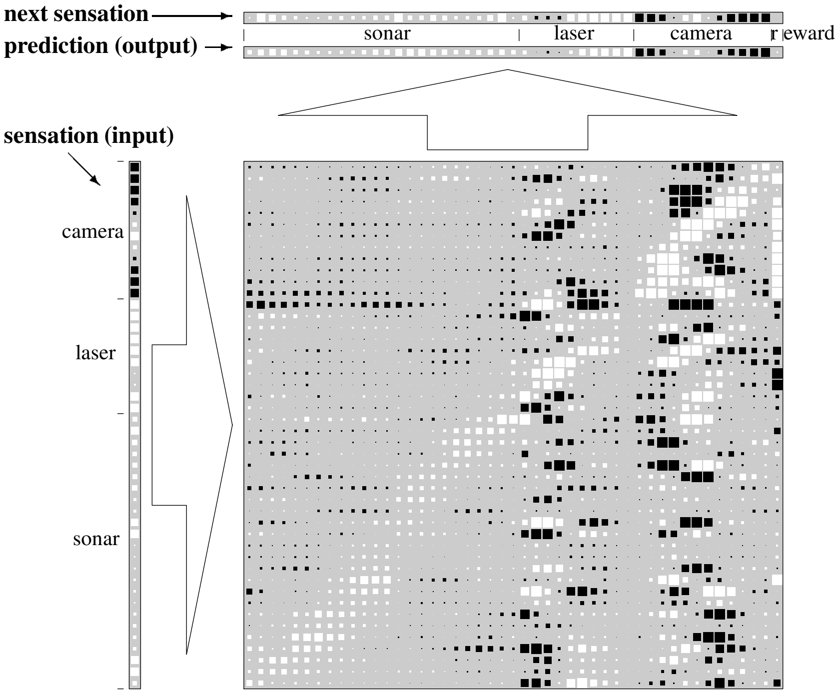

Figure 5: **Prediction and slopes.** A neural network action model predicts sensations and reward for the next time step. The large matrix displays the output-input slopes of the network. White boxes refer to negative and black boxes to positive values. Box sizes indicate absolute magnitudes. Notice the bulk of positive gradients along the main diagonal, and the cross-dependencies between different sensors.

a bird eye's view, using a sonar map techniques described in [27]. 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 was between two and eleven actions in length. It 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 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. 7. 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.

V. Conclusion

In this paper we have described an approach to the lifelong robot learning problem, based on Q-Learning and EBNN. We have empirically illustrated rapid learning capabilities in a mobile robot navigation domain, demonstrating the appropriateness of Q-Learning and EBNN for robot navigation with unstructured, high-dimensional perceptual spaces.

The reader should notice that others have studies the transfer of knowledge. For example, in the context of reinforcement learning, models have been successfully employed for the transfer of knowledge via planning [9, 25] or replay [7]. Others proposed hierarchical approaches, in which the building blocks, once learned, can be applied to multiple tasks [3, 7, 21]. A third way for the transfer of knowledge is concerned with the construction of better internal representations, which improve generalization across multiple tasks [1, 16, 19, 23]. While this list is clearly incomplete, it nevertheless illustrates the impor-

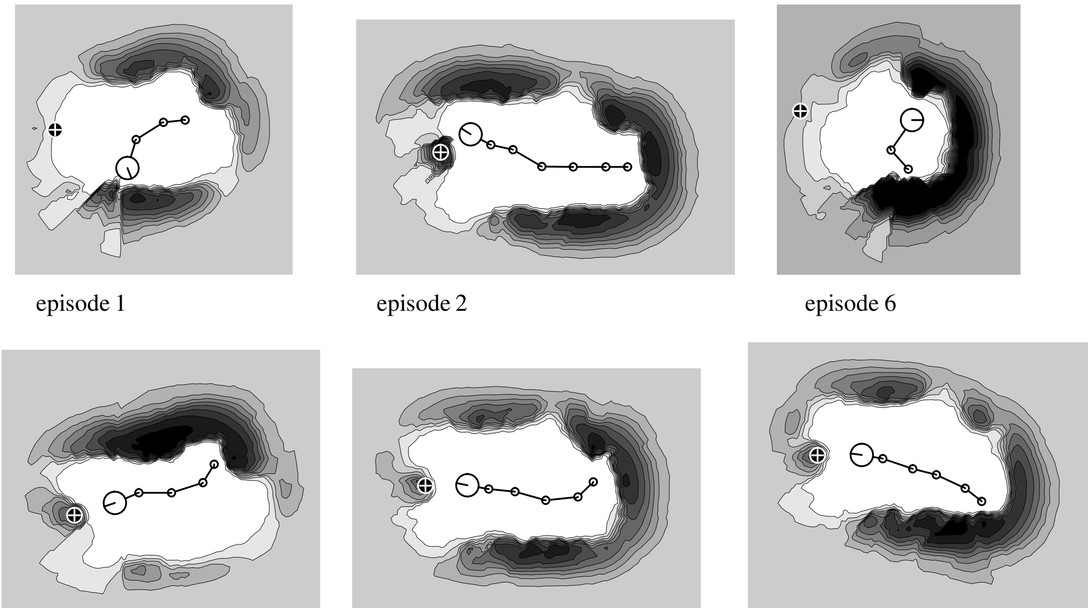

Figure 6: **Learning navigation.** Traces of three early and three late episodes are shown. Each diagram shows a two-dimensional occupancy maps of the world, which have been constructed based on sonar information. Bright regions indicate free-space and dark regions indicate the presence of obstacles. Note that the location of the target object (marked by a cross) is held constant in this experiment.

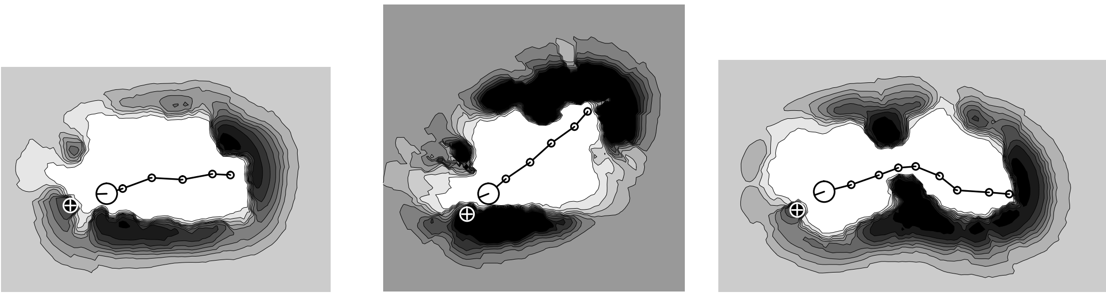

Figure 7: **Testing navigation.** After training, the location of the target object was moved. In some experiments, the path of the robot was also blocked by obstacles. Unlike plain inductive neural network learning, EBNN almost always manages these cases successfully.

tance for transferring knowledge for scaling machine learning to more complex domains [27].

From a lifelong learning perspective, much of the work presented in this paper is preliminary. While we have not yet studied robot control in the context of multiple tasks in practice, in experiments described here and elsewhere [13, 26, 27] we consistently found that EBNN outperforms pure inductive neural network learning, which does not employ background knowledge and hence learns from scratch. In a related paper we have illustrated superior generalization due to EBNN in a robot perception task [14]. Learning mechanisms that allows for the effective knowledge transfer, like EBNN, is a necessary prerequisite for successful approaches to the lifelong robot learning problem.

Acknowledgment

The author thanks Tom Mitchell and the CMU robot learning group for their invaluable feedback. He also thanks CMU for granting him access to the Xavier robot, without which this research would not have been possible.

This research is sponsored in part by the National Science Foundation under award IRI-9313367 to Tom Mitchell. Views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of NSF.

REFERENCES

[1] Richard Caruana. Multitask learning: A knowledge-based of source inductive bias. submitted for publication, 1993.

[2] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinction approach. In *Proceedings of 1992 AAAI Conference*, Menlo Park, CA, July 1992. AAAI Press / The MIT Press.

[3] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

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

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

[6] 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.

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

[8] 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.

[9] Sridhar Mahadevan. Enhancing transfer in reinforcement learning by building stochastic models of robot actions. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 290–299. Morgan Kaufmann, 1992.

[10] R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In Paul E. Utgoff, editor, *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, San Mateo, CA, 1993. Morgan Kaufmann.

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

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

[13] Tom M. Mitchell and Sebastian B. 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.

[14] Joseph O'Sullivan. Xavier manual. Carnegie Mellon University, Learning Robot Lab Internal Document - contact josullvn@cs.cmu.edu, January 1994.

[15] D. 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.

[16] Lori Y. Pratt. Discriminability-based transfer between neural networks. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

[17] 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.

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

[19] Noel E. Sharkey and Amanda J.C. Sharkey. Adaptive generalization and the transfer of knowledge. In *Proceedings of the Second Irish Neural Networks Conference*, Belfast, 1992.

[20] 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.

[21] Satinder P. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, 8, 1992.

[22] Satinder Pal Singh, Tommi Jaakkola, and Micheal I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the Eleventh Machine Learning Conference*, 1994. (to appear).

[23] Steven C. Suddarth and Y. L. Kergosien. Rule-injection hints as a means of improving network performance and learning time. In *Proceedings of the EURASIP Workshop on Neural Networks*, Sesimbra, Portugal, Feb 1990. EURASIP.

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

[25] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning, June 1990*, pages 216–224, 1990.

[26] Sebastian B. 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.

[27] Sebastian B. 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.

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