

Continual Robot Learning with Constructive Neural Networks

Axel Großmann and Riccardo Poli

School of Computer Science, The University of Birmingham, UK
{A.Grossmann, R.Poli}@cs.bham.ac.uk

Abstract. In this paper, we present an approach for combining reinforcement learning, learning by imitation, and incremental hierarchical development. We apply this approach to a realistic simulated mobile robot that learns to perform a navigation task by imitating the movements of a teacher and then continues to learn by receiving reinforcement. The behaviours of the robot are represented as sensation-action rules in a constructive high-order neural network. Preliminary experiments are reported which show that incremental, hierarchical development, bootstrapped by imitative learning, allows the robot to adapt to changes in its environment during its entire lifetime very efficiently, even if only delayed reinforcements are given.

1 Introduction

The development of learning techniques for autonomous robots constitutes one of the major trends in the current research on robotics [2]. Adding learning abilities to robots offers a number of benefits. For example, learning is essential when robots have to cope with dynamic environments. Moreover, it can help reduce the cost of programming robots for specific tasks. These and other features of robot learning are hoped to move autonomous robotics closer to real-world applications.

Reinforcement learning has been used by a number of researchers as a computational tool for constructing robots that improve themselves with experience [6]. Despite the impressive advances in this field in recent years, a number of technological gaps remain. For example, it has been found that traditional reinforcement learning techniques do not scale up well to larger problems and that, therefore, ‘we must give up *tabula rasa* learning techniques’ [6, p. 268] and guide the learning process by shaping, local reinforcement signals, imitation, problem decomposition, and reflexes. These techniques incorporate a search bias into the learning process which may lead to speed-ups. However, most of the proposed solutions do not address sufficiently the issue of continual adaptation and development.

If a robot is provided with a method for measuring performance, learning does not need to stop. Robots could adapt to changes in their environment during their entire lifetime. For example, Dorigo and Colombetti [4] have proposed an

incremental development approach in which a learning agent goes through three stages of development during its life: a ‘baby phase’, a ‘young phase’, and an ‘adult phase’. In the ‘adult phase’, a monitoring routine is used, which can reactivate either the trainer (used in the ‘baby phase’) or the delayed-reinforcement modules (used in the ‘young phase’) if the agent’s performance drops.

Our approach to continual learning is slightly different. We believe that incremental, hierarchical development is essential for continual learning. If a robot has to learn a new behaviour during its lifetime, it should make use of the previously learned behaviours. Machine learning mechanism that could allow this include the default hierarchies produced in learning classifier systems [11] and the automatic addition of units and connections in constructive neural networks [10]. Of particular interest in the field of constructive networks are the Temporal Transition Hierarchies introduced by Ring [10] who has shown that they can be used as supervised learning algorithms in a Q-learning reinforcement system.

The aim of this paper is to show that temporal transition hierarchies can be used for the integration of several robot-learning techniques. We propose an approach for combining reinforcement learning, imitation, and incremental hierarchical development. We apply this approach to a simulated mobile robot that learns to perform a navigation task by imitating the movements of a teacher and then continues to learn by receiving reinforcement.

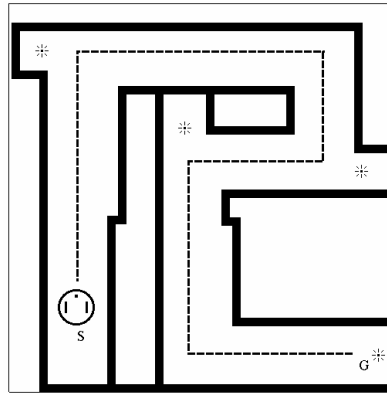
The paper is organised as follows. In Section 2, we describe the learning scenario which we have chosen for our experiments. In Section 3, we introduce the particular connectionist model we are using for learning the robot skills. In Section 4, we explain how the robot can acquire skills hierarchically and incrementally by imitative learning. In Section 5, we explain how skills learned by imitation can be used in a reinforcement-learning system. In Section 6, we demonstrate how additional behaviours can be learned by receiving delayed reinforcement. Finally, we draw some conclusions in Section 7. We explain our ideas using examples obtained from our actual experiments; there is no particular section for the discussion of experimental results. Future papers we be devoted to a fuller validation of our approach.

2 The Learning Task

For a robot to learn a task, it must be given some feedback concerning its performance. In reinforcement learning, the robot is given a scalar performance signal (called reinforcement) telling it how well it is currently doing at the task. The success of the learning process depends very much on how often the robot receives rewards. It may receive reinforcement after each action or after a sequence of actions. For many learning tasks, only delayed reinforcement is available. For example, imagine a robot wandering from a start state to a goal state. It will receive no positive feedback at all during most of its interaction with the environment. Reinforcement will only be given when it actually reaches the goal. This example illustrates why learning using delayed reinforcement is usually difficult

to achieve. If the task is complex and the robot has to learn from scratch, then the robot is unlikely to find any action for which it is given reward.

For our experiments, we have chosen a learning task where only delayed reinforcement is available. A mobile robot has to travel through a maze-like environment. The robot always starts from a predefined position. By reacting correctly to the corners, it has to find its way to a specified goal position. To receive reinforcement, the robot has to reach the goal area and to stop there.



(a) Maze-like environment.

$$\begin{aligned}
 \text{no_light}(t) \wedge \text{light_left}(t - 1) &\rightarrow \text{turn_right} \\
 \text{no_light}(t) \wedge \text{light_right}(t - 1) &\rightarrow \text{turn_left} \\
 \text{no_light}(t) &\rightarrow \text{move_forward} \\
 \text{light_left}(t) &\rightarrow \text{turn_right} \\
 \text{light_right}(t) &\rightarrow \text{turn_left} \\
 \text{light_ahead}(t) &\rightarrow \text{stop}
 \end{aligned}$$

(b) Default hierarchy for maze navigation.

Fig. 1. Learning a navigation task with delayed reinforcement.

The experiments have been performed using a physically realistic simulator for the Khepera robot [9]. Khepera is a miniature mobile robot equipped with eight infrared sensors, six at the front and two at the back. Using its sensors, the robot can detect obstacles and light sources within a range of 40 mm. A typical test environment is shown in Fig. 1(a) where the start and goal positions are

marked with S and G, respectively. In some corners of the maze, there are light sources which can be detected by the robot.

Learning from delayed reinforcement is generally a slow process. Therefore, a straightforward application of reinforcement-learning techniques is often not practical. To speed up the learning process, we have used task decomposition and abstraction. Namely, we have predefined a small set of medium-level motor and pattern-recognition behaviours which enormously reduce the size of the search space the learning algorithm has to explore. We have restricted the number of actions the robot can perform by defining four motor controllers: *move_forward*, *turn_left*, *turn_right*, and *stop*. That is, the robot can follow a path, perform 90-degree turns in front of obstacles, and stop at any position. Moreover, the robot can distinguish the following light-configurations: *light_ahead*, *light_right*, *light_left*, and *no_light*.

The motor controllers have been implemented as hand-written routines whereas the light-detection skills have been learned off-line using a feed-forward neural network and a supervised learning algorithm. Sensor readings for typical light-configurations have been used as training instances.

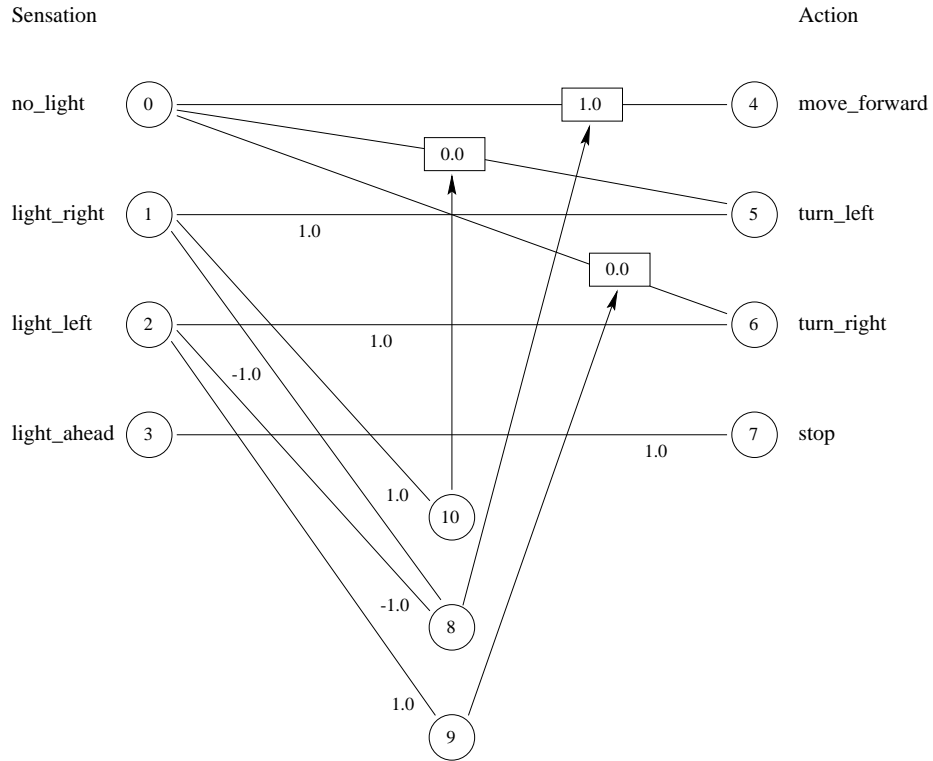
In the given learning task, the robot has four different sensations and four actions. It has to learn to choose the right action each time there is a wall blocking its way. That is, the robot has to solve a sequential decision task. This could be done by developing a set of sensation-action rules. Preferably, the rule set should be minimal. A way of achieving this is to design a default hierarchy in which general rules cover default situations, and specialised rules deal with exceptions like the rule set in Fig. 1(b). In the next sections, we will show how such rule sets can be represented and learned by a constructive neural network.

3 Transition Hierarchy Networks

Ring [10] has developed a constructive, high-order neural network, known as Temporal Transition Hierarchies, that can learn continuously and incrementally. There are two types of units in the network: *primitive* units and *high-level* units. The primitive units are the input and output neurons of the network and represent the sensations and actions of the learning agent. The high-level units enable the network to change its behaviour dynamically. More precisely, a high-level unit l_{ij} is assigned to the synaptic connection between the lower-level units i and j and can modify the strength w_{ij} of that connection.

When no high-level units are present, the transition hierarchy network behaves like a simple feed-forward neural network. The activation of the i -th action unit is the sum of the sensory inputs, multiplied by the high-order weight \hat{w}_{ij} connected to unit i . The activation of the high-order units, if present, is computed in the same way. The high-order weights \hat{w}_{ij} are defined as follows. If no l unit exists for the connection from j to i , then $\hat{w}_{ij} = w_{ij}$. If there is such a unit, its previous activation value is added to w_{ij} , i.e., $\hat{w}_{ij}(t) = w_{ij} + l_{ij}(t - 1)$.

The function of the high-level units will be illustrated with an example. The transition hierarchy network shown in Fig. 2 solves the navigation task in



t=1:	In	1	0	0	0
	Out	1	0	0	0
t=2:	In	0	0	1	0
	Out	0	0	1	0
t=3:	In	1	0	0	0
	Out	0	0	1	0
t=4:	In	0	0	1	0
	Out	0	0	1	0
t=5:	In	0	1	0	0
	Out	0	1	0	0
t=6:	In	1	0	0	0
	Out	0	1	0	0
t=7:	In	0	0	0	1
	Out	0	0	0	1

Fig. 2. Transition hierarchy network to solve the navigation task (above) and the outputs produced at different time steps (below).

Fig. 1. As explained in Sect. 4, it has been created automatically. Until it reaches the goal position, the robot has to change its direction of travel several times. The optimal sequence of decisions can be described through sensation-action rules. Indeed, the network is a computational representation for the rule set in Fig. 1(b). The net outputs are computed each time the robot detects an obstacle in its way. Therefore, we distinguish several time steps t . In time step 2, 4, 5, and 7, the robot can select the action on the basis of the current sensation only. In the absence of light however, the decision on which action to take depends also on the sensation in the previous time step. This ambiguity (called hidden state problem) is solved using the high-level units 8, 9, and 10. For example, the high-level unit 8 inhibits the default rule $\langle \text{no_light}(t) \rightarrow \text{move_forward} \rangle$ if *light_right* or *light_left* was active in the previous time step.

The connection weights of temporal transition hierarchies can be determined using a supervised learning technique. Ring [10] has derived a learning rule that performs gradient descent in the error space. Transition hierarchy nets learn constructively and incrementally; new high-level units are created while learning. A new unit needs to be added when a connection weight should be different in different circumstances, see [10, p. 59]. Therefore, long-term averages are maintained for every connection weight. Ring's experimental results show that the learning algorithm performs very well compared to other neural-network models for temporal processing.

4 Learning by Imitation

Recently, several researchers have proposed to use imitation as a way for robots to learn new skills [1, 3, 5, 7]. The main idea is that the robot learns how to act by perceiving and imitating the actions of a teacher. In that way, the robot can find out what is a good action to perform under which circumstances.

We decided to study the possibilities offered by imitative learning, in particular, the idea that the knowledge of successful action sequences reduces the size of the search space that the robot would need to explore in order to learn new skills using reinforcement learning. Our learning scenario is similar to that used by Hayes and Demiris [5]. Two robots are present in the environment: a teacher and a learner. The teacher robot is travelling through a maze along the optimal path with the learner robot following at a certain distance. The learner detects and records 'significant events'. That is, it notices when the teacher changes its direction of travel in front of a corner. The learner then associates the current light-configuration with the action the teacher has carried out, and imitates that action.

To learn by imitation, the learner robot needs the following additional skills: (1) the learner has to be able to follow the teacher; (2) the learner has to be able to detect changes in the movement of the teacher, specifically, to detect 90-degree turns; and (3) the learner has to associate an action with the light-configuration currently sensed. One can imagine many possible realisations of the teacher-following behaviour and the turn-detection skill. For the former, we

have used a simple pattern-associator neural network which estimates the current distance from the teacher on the basis of the current sensor readings. The latter is performed by a partial recurrent neural network. Both networks have been trained off-line using typical sensor readings.

To detect a significant event, the learner needs, of course, a corresponding recognition behaviour. In our experiments, the segmentation of significant events is performed by the predefined pattern-recognition behaviours (see Sect. 2).

The situation-action association is actually the crucial part in our imitative-learning approach. The selection of an optimal computational mechanism for this task depends on the amount of information to be stored, the structure of the learning tasks, and the duration of the learning process. Given that in our experiments sensations are pre-processed, that the number of possible actions is finite, and that knowledge reuse can help the learning process, then temporal transition hierarchies seemed very well suited to perform this task.

So, the sensation-action association behaviour was implemented as follows. The learner robot is watching the teacher on its way through the maze, recording the actions performed and light-configurations detected. When the tour is finished, i.e., the goal position has been reached, the sensations and actions stored are used to train a transition hierarchy network. For each significant event, there is a sensation-action pair, which is used as a training instance by the supervised learning algorithm. If a light-configuration has been present, then the activation of the corresponding sensory unit is set to 1.0, to 0.0 otherwise. The activation values of all sensory units form the input vector for the network. The target vector contains an activation 1.0 for the action chosen; all other action units are required to be inactive, i.e., their activation is 0.0. The input and target vectors are presented several times to the network. The sequence of training instances is fixed and corresponds to the temporal order of the significant events.

In our experiments, the learning algorithm did converge very quickly (in less than 30 training epochs). The transition hierarchy network shown in Fig. 2 has been obtained using imitative learning (the connection weights have been rounded to the closed integer after learning). Theoretically, it is possible to collect more than one set of associations and learn them altogether using a single network.

5 Bootstrapping Reinforcement Learning

The reproduction of the teacher's actions in itself cannot solve the problem of continual learning as soon as the teacher robot is removed. Reinforcement learning could but it is computationally impractical. We believe that one can overcome the practical problems of traditional reinforcement learning techniques by gaining information about successful action sequences through imitating an expert on one hand, and by reusing previously learned sensation-action rules on the other hand.

To explore these ideas, we have decided to use Q-learning which is probably the most popular and well understood model-free reinforcement-learning algo-

gorithm [6]. The idea of Q-learning is to construct an evaluation function $Q(s, a)$, called Q-function, which returns an estimate of the discounted cumulative reinforcement, i.e., the utility, for each state-action pair (s, a) given that the learning agent is in state s and executes action a . Given an optimal Q-function and a state s , the optimal action is found simply by choosing the action a for which $Q(s, a)$ is maximal. The utility of doing an action a_t in a state s_t at time t is defined as the expected value of the sum of the immediate reinforcement r plus a fraction γ of the utility of the state s_{t+1} , i.e., $Q(s_t, a_t) = r(s_t, a_t) + \gamma(\max_a Q(s_{t+1}, a))$, with $\gamma \in [0, 1]$.

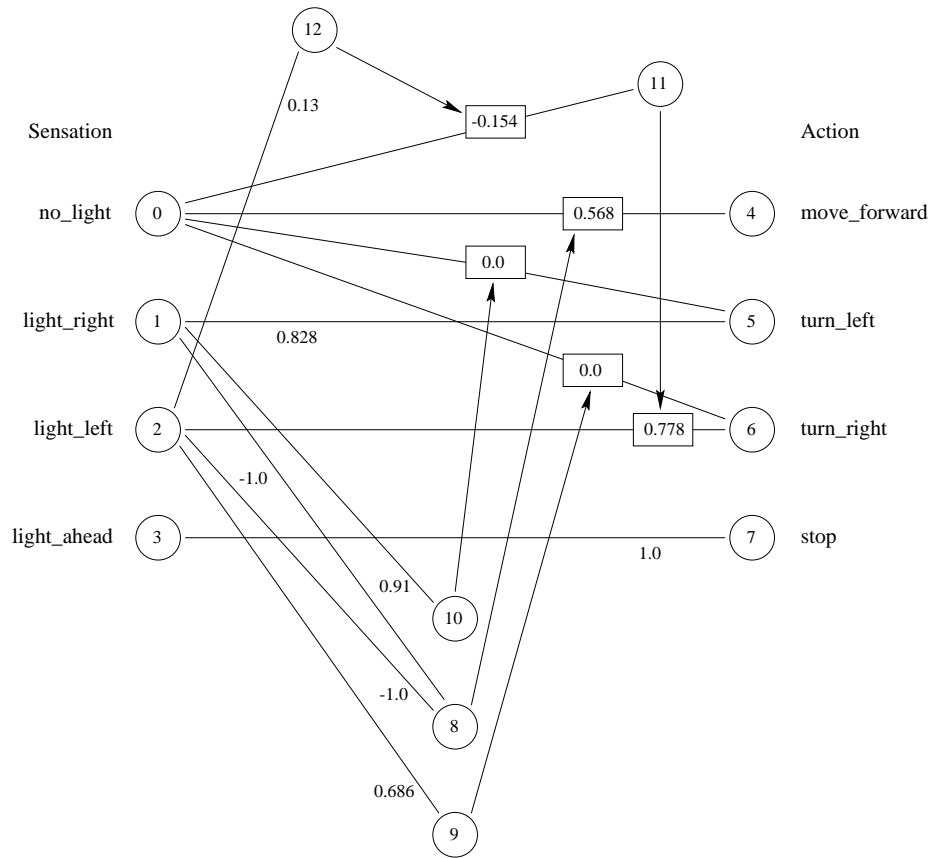
A transition hierarchy network can be used to approximate the utility function $Q(s, a)$. The states s can be represented by the activation pattern of the sensory units, whereas the utility $Q(s, a)$ can be represented by the activation value of the corresponding action unit in the network. After learning by imitation, the outputs of the network represent actions and not utilities. Nevertheless, we can use the outputs as initial utilities, i.e., as starting points for the computation of the correct utility values. To use the net output as utilities, the output of the action units has to be discounted according the action's contribution to achieving reinforcement. One way to perform this transformation is to let the robot receive reinforcement from the environment and to apply Lin's algorithm for connectionist Q-learning [8]. The algorithm uses the following rule to update the stored utility values: $\Delta Q(s_t, a_t) = r(s_t, a_t) + \gamma(\max_a Q(s_{t+1}, a) - Q(s_t, a_t))$ where $\Delta Q(s_t, a_t)$ is the error value corresponding to the action just performed.

Generally, a transition hierarchy network will grow while learning the Q-values since a single sensation-action rule can have different utilities values at different time steps if utilities are discounted, i.e., if $\gamma < 1$. The effect of Lin's algorithm on the network in Fig. 2 for $\gamma = 0.91$ can be seen in Fig. 3. Two high-level units have been added during the learning process to distinguish the utility values of the rule $(\text{light_left}(t) \rightarrow \text{turn_right})$ in time step 2 and 4. Since Lin's algorithm changes the utility values only for actions which have actually been performed, the output of some action units becomes negative. However, these values could be changed into 0.0 if necessary. Provided that the environment has not changed, the Q-learning algorithm converges very quickly in less than 15 trials, a trial being a repetition of the navigation task followed by receiving reinforcement each time at the goal position.

6 Learning Additional Behaviours

After a robot has learned an optimal behaviour, the environment might change. If the robot is not able to adapt to the new situation, it will keep trying formerly effective actions without any success. The objective of our continual-learning approach is to allow the robot to adapt to environmental changes while using as much previously learned knowledge as possible to find a solution for the current task.

The sensation-action rules represented in the transition hierarchy network need revision after changes in the environment or in the learning task. Some



t=1:	In	1.00	0.00	0.00	0.00
r=0	Out	0.57	0.00	0.00	0.00
t=2:	In	0.00	0.00	1.00	0.00
r=0	Out	0.00	0.00	0.62	0.00
t=3:	In	1.00	0.00	0.00	0.00
r=0	Out	-0.43	0.00	0.69	0.00
t=4:	In	0.00	0.00	1.00	0.00
r=0	Out	0.00	0.00	0.75	0.00
t=5:	In	0.00	1.00	0.00	0.00
r=0	Out	0.00	0.83	0.00	0.00
t=6:	In	1.00	0.00	0.00	0.00
r=0	Out	-0.43	0.91	0.00	0.00
t=7:	In	0.00	0.00	0.00	1.00
r=1	Out	0.00	0.00	0.00	1.00

Fig. 3. Transition hierarchy network representing Q-values (above) and the outputs produced at different time steps (below).

of the rules might need to be retracted and possibly replaced by new rules. The revision should fulfil the following three criteria: (1) the rule set should be minimal; (2) the amount of information lost in the revision should be minimal; and (3) the least useful rules should be retracted first, if necessary. These criteria could be used to evaluate the performance of any continual learning algorithm.

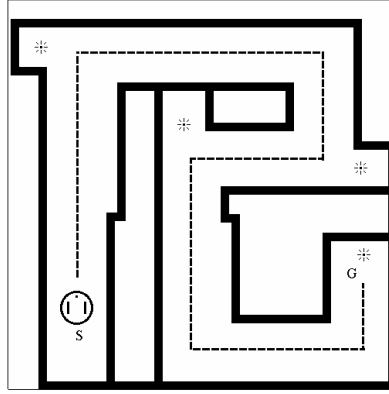
Ring has performed an experiment with temporal transition hierarchies to investigate their capability of reusing previously learned knowledge [10]. He found that the network was actually capable of taking advantage of and building onto results from earlier training. These features, however, have not been tested in real-world applications. For example, we have found that transition hierarchy networks are very sensitive to noise while learning. Under these circumstances, more units are created than actually necessary.

In our experiments with changing environments, we have adopted a rather pragmatic approach which differs from traditional reinforcement-learning techniques. To keep the size of the network as small as possible, the learning algorithm is made to learn correct utility values only for action sequences which have been found to be useful.

-
1. Create a network TTH_{STM} as a copy of TTH_{LTM} .
 2. Set robot to start position; reset the activations of TTH_{STM} and TTH_{LTM} ; clear the memory from experiences.
 3. Provide the pre-processed sensations as input s to TTH_{STM} and TTH_{LTM} ; propagate the activations.
 4. Select an action a probabilistically based on the output of either TTH_{STM} or TTH_{LTM} .
 5. Perform the action a ; get reinforcement r ; store the experience (s, a, r) .
 6. If $(r > 0)$ then replay stored experiences for TTH_{LTM} until its outputs have converged and suspend learning process.
 7. Adjust TTH_{STM} by back-propagating $\Delta Q(s, a)$.
 8. If $(a = stop)$ go to 2 else go to 3.
-

Fig. 4. Algorithm for continual reinforcement learning with temporal transition hierarchies.

To keep the changes in the rule set minimal, two networks are used during the search for a successful action sequence. One transition hierarchy network, denoted as TTH_{LTM} , serves as long-term memory and another network, denoted as TTH_{STM} , is used as a short-term memory. TTH_{LTM} learns from positive experience only, whereas TTH_{STM} keeps track of unsuccessful actions produced by the network. The steps performed during the search for a solution are given in Fig. 4. The robot has to find a solution for the changed learning task by trial-and-error interaction with the environment. It starts a trial always at the predefined start position (step 2). The actions are chosen probabilistically (step 4) as follows.



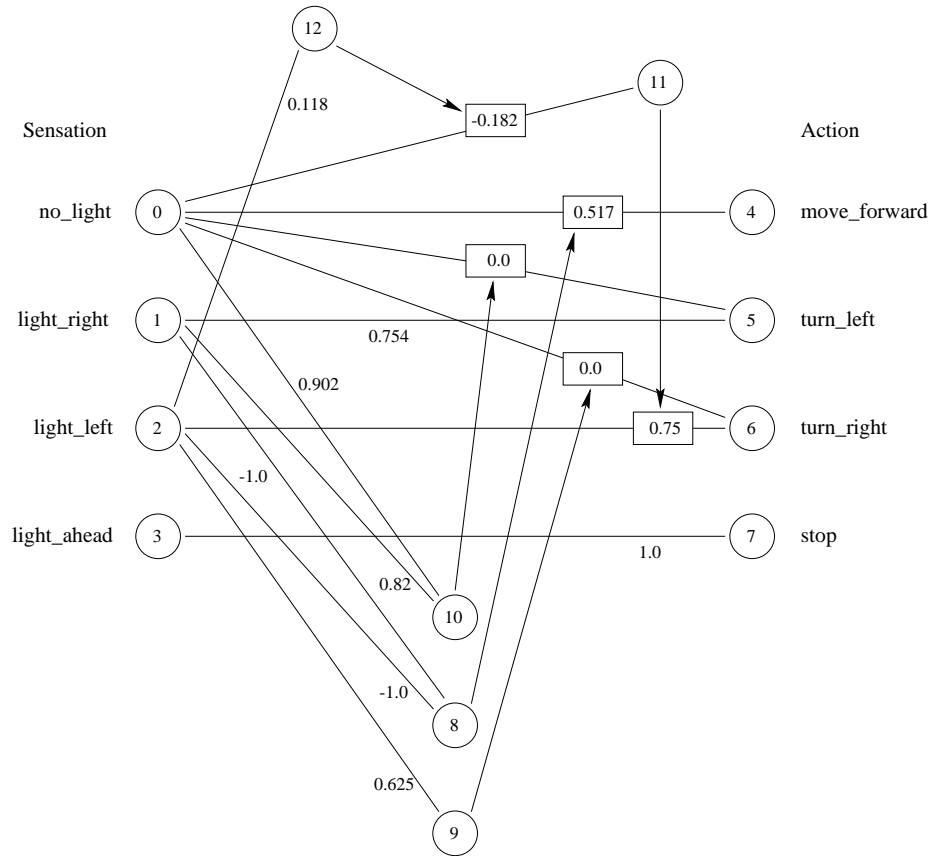
$$\text{no_light}(t) \wedge \text{no_light}(t - 1) \rightarrow \text{turn_left}$$

Fig. 5. Changed test environment requiring to learn a new rule for the final turn.

With a certain probability the robot takes the utility values provided by either TTH_{LTM} or TTH_{STM} . There is always a minimal probability for any action to be chosen. The probability of selecting an action is proportional to its utility value. The utility values are updated (steps 6 and 7) according to the rule described in the previous section with the following exception. The utility of the selected action must not be smaller than the utilities of the other actions. Therefore, the utility values of non-selected actions are reduced if necessary. Each time a solution has been found, TTH_{LTM} is trained (step 6) using the successful action sequence which has been recorded (step 5), and TTH_{STM} is replaced with a copy of the new long-term memory network.

We have performed several experiments in which the robot had to adapt its behaviour to changes in the environment. We have created new environments by changing the position of walls, or just by modifying the start or goal position of the robot. After finding a solution, the robot had to revise the utilities stored in the network. A simple example is shown in Fig. 6 where the robot had to learn an additional rule to respond to the changed goal position in Fig 5. By applying Ring’s supervised learning algorithm to the TTH_{LTM} network (step 6 in Fig. 4), a new link between the units 0 and 10 was added and the weights were adjusted to represent the new Q-values.

In our experiments, we used a neural network with a fixed number of input and output neurons which correspond respectively to the predefined pattern-recognition and motor behaviours mentioned in Sect. 2. We believe that these basic behaviours can be learned as well, for example, using unsupervised learning techniques. In this way, the predefined behaviours would correspond to the behaviours built up in Piaget’s exploration mechanism, namely, a set of ‘base behaviours’ from which to draw on to produce imitative behaviours, and a set of ‘registrations’ with which to compare and recognise behaviours, see [1]. More-



t=1:	In	1.00	0.00	0.00	0.00
r=0	Out	0.52	0.00	0.00	0.00
t=2:	In	0.00	0.00	1.00	0.00
r=0	Out	0.00	0.00	0.57	0.00
t=3:	In	1.00	0.00	0.00	0.00
r=0	Out	-0.48	0.00	0.62	0.00
t=4:	In	0.00	0.00	1.00	0.00
r=0	Out	0.00	0.00	0.69	0.00
t=5:	In	0.00	1.00	0.00	0.00
r=0	Out	0.00	0.75	0.00	0.00
t=6:	In	1.00	0.00	0.00	0.00
r=0	Out	-0.48	0.82	0.00	0.00
t=7:	In	1.00	0.00	0.00	0.00
r=0	Out	0.52	0.90	0.00	0.00
t=8:	In	0.00	0.00	0.00	1.00
r=1	Out	0.00	0.00	0.00	1.00

Fig. 6. Transition hierarchy network adapted to the environment in Fig. 5 (above) and the outputs produced at different time steps (below).

over, it should be noted that new input and output neurons can be added during the learning process, if necessary. Adding sensory and action units does not affect previously learned behaviours.

In the experiments so far, we were mainly interested in the changes of the network structure during the learning process. Therefore, a simple learning task has been chosen which can be solved using a rather simple set of rules. The robot learns to find its goal position by solving a sequential decision task. Despite the simplicity of the learning task and the absence of contradictory rules, the convergence of the framework is provided by the constructive character of the learning algorithm. On the one hand, a contradicting rule will be revised, if necessary, in step 6 of the learning algorithm (see Fig. 4). On the other hand, as much as possible of the previously learned behaviours is kept by adding a new unit which makes use of the temporal context, see [10, ch. 6].

7 Conclusions and Future Work

We have proposed an approach in which a mobile robot learns skills hierarchically and incrementally by imitative learning. The skills are represented as sensation-action rules in a constructive high-order neural network. We have demonstrated that incremental, hierarchical development, bootstrapped by imitative learning, allows the robot to adapt to changes in its environment during its entire lifetime very efficiently, even if only delayed reinforcements are given.

We think that the usefulness of incremental learning might depend on the specific learning tasks to be carried out. For example, it might only be useful if the learning tasks are correlated. The learning tasks used in our experiments were simple but typical for autonomous robots. In the future, we will apply our approach to more complex learning tasks on a real robot and we will investigate in which situations incremental learning is beneficial.

Acknowledgements

The authors wish to thank Olivier Michel, who has developed the Khepera Simulator [9] used in the experiments, and the members of the EEBIC (Evolutionary and Emergent Behaviour, Intelligence and Computation) group for useful discussions and comments. Also, thanks to an anonymous reviewer for many useful suggestions in improving this paper.

Bibliography

- [1] Paul Bakker and Yasuo Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *Proceedings of the AISB'96 Workshop on Learning in Robots and Animals*, Brighton, UK, 1996.
- [2] Jonathan H. Connell and Sridhar Mahadevan, editors. *Robot Learning*. Kluwer Academic, Norwell, MA, USA, 1993.
- [3] John Demiris and Gillian Hayes. Imitative learning mechanisms in robots and humans. In Volker Klingspor, editor, *Proceedings of the Fifth European Workshop on Learning Robots*, Bari, Italy, 1996.
- [4] Marco Dorigo and Marco Colombetti. The role of the trainer in reinforcement learning. In S. Mahadevan *et al.*, editors, *Proceedings of the Workshop on Robot Learning held as part of the 1994 International Conference on Machine Learning (ML'94) and the 1994 ACM Conference on Computational Learning Theory (COLT'94)*, pages 37–45, 1994.
- [5] Gillian Hayes and John Demiris. A robot controller using learning by imitation. In *Proceedings of the Second International Symposium on Intelligent Robotic Systems*, Grenoble, France, 1994.
- [6] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Artificial Intelligence Research*, 4:237–285, 1996.
- [7] Yasuo Kuniyoshi, Masayuki Inaba, and Hirochika Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6), 1994.
- [8] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3/4):293–321, 1992. Special Issue on Reinforcement Learning.
- [9] Olivier Michel. *Khepera Simulator Package 2.0*. University of Nice Sophia-Antipolis, Valbonne, France, 1996. Available via the URL <http://www.wi3s.unice.fr/~om/khep-sim.html>.
- [10] Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas, Austin, TX, USA. Available via the URL <http://www-set.gmd.de/~ring/Diss/>, 1994.
- [11] R. L. Riolo. The emergence of default hierarchies in learning classifier systems. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 322–327. Morgan Kaufmann, 1989.