Axel Grossmann

A Constructive Connectionist Approach Towards Continual Robot Learning

School of Computer Science
The University of Birmingham

Department of Computer Science
Dresden University of Technology

Diploma Thesis

About

A Constructive Connectionist Approach Towards Continual Robot Learning

By Axel Großmann
Born on 29th December 1971 in Halle

Department of Computer Science
Dresden University of Technology

Submitted on 29th August 1997

Supervisor: Prof. Steffen Hölldobler
External Supervisor: Dr Riccardo Poli

This work is also available as Technical Report CSRP-97-20, School of Computer Science, The University of Birmingham, UK.

Abstract

This work presents an approach for combining reinforcement learning, learning by imitation, and incremental hierarchical development. The approach is used in 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. The experimental results indicate that default hierarchies of sensation-action rules can be constructed solely by connectionist learning.

Keywords: Mobile robotics, continual learning, reinforcement learning, learning by imitation, constructive neural networks.

Contents

1 Introduction **1**

2 Robot Learning Techniques **5**

2.1 Challenges 5

2.2 Main Types of Learning 7

2.3 Robot-Environment Interaction 8

2.3.1 Robotics Environments 8

2.3.2 Learning Tasks 9

2.4 Main Methods of Learning 11

2.4.1 Reinforcement Learning Approaches 11

2.4.2 Evolutionary Approaches 13

2.4.3 Connectionist Approaches 14

3 Ontogenic Neural Networks **17**

3.1 Topology Adaptation 17

3.2 Supervised Ontogenic Networks 18

3.3 Unsupervised Ontogenic Networks 19

3.4 Constructive Networks for Temporal Processing 20

3.4.1 Hidden State Problem 20

3.4.2 Learning Gap Tasks 21

4 Continual Robot Learning **23**

4.1 Related Learning Approaches 23

4.1.1 Skill Acquisition in Psychology 23

4.1.2 Operator Acquisition 24
4.1.3 Bottom-up Skill Learning 24
4.2 Ingredients of Continual Learning 25

5 Learning of Sequential Decision Tasks 29
5.1 The Learning Task 29
5.2 Transition Hierarchy Networks 31

6 Experiments on Continual Learning 39
6.1 The Experimental Environment 39
6.2 Learning by Imitation 40
6.3 Bootstrapping Reinforcement Learning 43
6.4 Learning Additional Behaviours 46

7 Discussion and Conclusions 51

Bibliography 53

Chapter 1

Introduction

The goal of autonomous robotics is to build physical systems that accomplish useful tasks without human intervention in real-world environments. The development of learning techniques for autonomous robot operation constitutes one of the major trends in the current research on robotics [10]. Two motivations underlie this trend. First, current industrial robots lack flexibility and autonomy. Typically, these robots perform pre-programmed sequences of operations in highly constrained environments, and cannot deal with unexpected situations. Second, there is a clear emerging market for truly autonomous robots. Possible applications range from intelligent service robots in offices, hospitals, and factory floors to maintenance robots in hazardous environments.

Adding learning abilities to mobile 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 [23]. 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' [23, 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 [12] 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.

In this work, we want to follow a slightly different life-long learning approach, which is known as continual learning [34]. We will argue that incremental, hierarchical development is essential for life-long learning. When a robot has to learn a new behaviour, it should make use of previously learned behaviours. Moreover, the behaviours should be learned in a bottom-up process whereby the old behaviours are used as constituents of newly created behaviours.

We believe that continual learning can only be useful in practice if it can incorporate different kinds of learning. The robot, if possible, should use several means of learning, such as performance feedback from the environment, the observable behaviour of a teacher, or the experiences of other learning agents. Therefore, continual learning can also be seen as an attempt in closing the traditional gap between learning techniques developed for single agents and multi-agent learning approaches.

To explore the usefulness of this approach to continual learning, three problems have to be solved. First, one has to develop a learning mechanism that enables the robot to learn incrementally and hierarchically. Second, one has to find a way to integrate continual learning with traditional robot learning techniques such as learning by imitation and reinforcement learning. Finally, one has to investigate the specific conditions or circumstances when continual learning gives an advantage compared to the other forms of learning.

Learning mechanisms that allow a system to learn in a hierarchical and incremental fashion include the default hierarchies produced in learning classifier systems [35] and the automatic addition of units and connections in constructive neural networks [34]. Of particular interest in the field of constructive networks are the temporal transition hierarchies introduced by Ring [34] who has shown that they can be used as supervised learning algorithm in a Q-learning reinforcement system.

In this work, we will show that temporal transition hierarchies can be used for continual robot learning. We propose a learning approach that combines reinforcement learning, imitation, and incremental hierarchical development. The approach is applied to a simulated mobile robot which learns to perform a navigation task by imitating the movements of a teacher and then continues to learn by receiving reinforcement.

Chapter 2 presents a brief introduction to robot learning. It describes several types of learning and gives an overview about popular learning techniques.

Chapter 3 is devoted to constructive neural networks. After describing basic principles, their ability to solve temporal problems is investigated.

Chapter 4 discusses issues of continual adaptation and development. It presents some relevant theories from cognitive science as well as computational approaches. The chapter concludes with a definition of continual learning.

Chapter 5 is devoted to temporal transition hierarchies. It describes how this network model can be used to represent and learn default hierarchies of sensation-action rules in an incremental and hierarchical fashion. This and the following chapter describe the main contributions of this work.

Chapter 6 presents experiments which show that incremental, hierarchical development, bootstrapped by imitative learning, allows a robot to adapt to changes in its environments very efficiently, even if only delayed reinforcements are given.

Chapter 7 concludes the work by discussing the results and by proposing future work.

Acknowledgements

I wish to thank Riccardo Poli, Jeremy Wyatt, and John Demiris for stimulating and helpful discussions. Thanks also to Olivier Michel, who developed the Khepera^{TM}Simulator [28] used in the experiments. Moreover, I would like to acknowledge the support by a scholarship of the School of Computer Science at The University of Birmingham.

Birmingham, August 1997

Axel Großmann[1]

[1]E-mail: axg@cs.bham.ac.uk

Chapter 2

Robot Learning Techniques

2.1 Challenges

Building a robot that learns to perform a task is a difficult business. Robot learning forces us to deal with the issue of *embedded systems* [22], where the learner is situated in an unknown, dynamic environment. The issues that arise in robot learning are quite different from those that may arise in classical AI applications, e.g., expert systems.

Traditionally, it has often been assumed in robotics research that accurate *a priori* knowledge about the robot, its sensors, and most important its environment is available. Unfortunately, often only inaccurate models of the world and the robot are available, and the kinds of environments such a robot can operate in, and consequently the kinds of tasks such a robot can solve, are limited.

According to Thrun and Mitchell [42], the usefulness of traditional engineering methods for building robots is limited by a number of factors:

- *Knowledge bottleneck.* A human designer had to provide accurate models of the world and the robot.
- *Engineering bottleneck.* Even if sufficiently detailed knowledge is available, making it computer-accessible, i.e., hand-coding explicit models of robot hardware, sensors, and environments, has often been found to require unreasonable amounts of programming time.
- *Tractability bottleneck.* It was early recognised that many realistic robot domains are too complex to be handled efficiently. Computa-

tional tractability turned out to be a severe obstacle for designing control structures for complex robots in complex domains.

- *Precision bottleneck.* The robot device must be precise enough to accurately execute plans that were generated using the internal models of the world.

Considering these bottlenecks, there are three types of knowledge that it would be useful for a robot to automatically acquire:

- *Hard to program knowledge.* In any given task, it is usually possible to distinguish between information that can be easily hardwired into the robot from that which would involve a lot of human effort.

- *Unknown information.* Sometimes, the information necessary to program the robot is simply not readily available, e.g., in the exploration of unknown terrain.

- *Knowledge of changing environments.* The world is a dynamic place. Objects move around from place to place, or appear and disappear.

Not surprisingly, there are a number of non-trivial real world issues that must be faced to build robots that acquire knowledge automatically. For example, Connell and Mahadevan [9] identified the following problems, which may vary with technical equipment, experimental environment, and learning task:

- *Sensor noise.* Most robot sensors are unreliable. Thus, state descriptions computed from sensors are bound to have inaccuracies in them.

- *Nondeterministic actions.* Since the robot has an incomplete model of its environment, the same action will not always have the same effect. Planning becomes difficult because one has to allow for situations when a given action sequence fails to accomplish a goal.

- *Reactivity.* A robot must respond to unforeseen circumstances in real time. In terms of learning, the learning algorithm must be tractable, i.e., every step of the algorithm must terminate quickly.

- *Incrementality.* A robot has to collect the experience from which it is to learn the task. The data forming the experience is not available off-line. The need for efficient exploration dictates that the learning algorithm must be incremental, so that the robot can adapt its exploration strategy as soon as new data are available.

- *Limited training time.* The training time available on a real robot is very limited.
- *Groundedness.* All the information that is available to a robot must come ultimately from its sensors or be hardwired from the start. Since the state information is computed from sensors, the learning algorithm must be able to work with the limitations of perceptual devices.

Obviously, the way these issues are addressed determines the success of any learning algorithm which is used in a real robot.

2.2 Main Types of Learning

We are concerned with the interaction of a robot with its environment. In general, this interaction is formalised using the concept of *behaviour*. According to Steels [36], a behaviour is a regularity in the interaction dynamics between the robot agent and the environment. For example, we may observe that a robot maintains a certain distance from a wall. As long as this regularity holds, we – the observers – may say that there is an obstacle-avoidance behaviour.

To realise a behaviour, there must be some sort of mechanism available to the robot. The mechanism must be physically implemented using a set of components (sensors, body parts, actuators) and a control program relating changes in sensory values and internal state variables to changes in actuator parameters and internal state variables. The observed behaviour is due to the interaction between the operation of the mechanism and a particular environment in which the robot agent finds itself.

Learning has the purpose of facilitating the actions the robot takes, by making them more relevant, appropriate, or precise. The robot's actions are determined by information at different levels. Therefore, different types of learning will be applicable. For example, Brooks and Matarić [6] identified the following types of learning:

- *Learning numerical functions for calibration or parameter adjustment.* This type of learning optimises operational parameters in an existing behavioural structure.
- *Learning about the world.* This type of learning constructs and alters some internal representation of the world.

- *Learning to coordinate behaviours.* This type of learning uses existing behavioural structures and their effects in the world to change the conditions in which they are triggered and their sequence.
- *Learning new behaviours.* This type of learning builds new behavioural structures.

2.3 Robot-Environment Interaction

In this work, we are concerned with learning of rather simple behaviours. The behaviours and learning tasks can be described in terms of sensations, actions, and reinforcement. In the following, a formal framework is given which allows to define an enormous range of specific tasks. The framework is based on definitions given by Ring [34], and Thrun and Mitchell [42].

2.3.1 Robotics Environments

The Robot. Any robot can be described as the implementation of a set of mappings from current and previous sense signals and actions, to action signals. For a robot to perform a task, it must produce actions through its actuators, possibly as a function of its previous actions and its previous and current sensations. In the discrete time case, this can be formalised as follows:

$$\mathbf{a}(t) = f_t(\mathbf{s}(0), \mathbf{a}(0), \mathbf{s}(1), \mathbf{a}(1), \ldots, \mathbf{s}(t-1), \mathbf{a}(t-1), \mathbf{s}(t)) \tag{2.1}$$

where $\mathbf{a}(\tau)$ is a vector of actuator signals describing the motor activity of the robot at time τ, $\mathbf{s}(\tau)$ is the vector of sensory signals the robot receives at time τ, and f_t is a function mapping a sequence of $2t+1$ vectors onto a single vector. f_t is not necessarily deterministic but might choose randomly one of many possible actions-vector candidates. This formalisation is general enough to describe any discrete-time robot. The continuous-time case will not be considered here.

One can think of the robot's next action also as a function of its last action, its current sensory inputs, and its internal state. This idea can be expressed using the following equations[1]:

$$\mathbf{a}(t) = f(S(t)) \tag{2.2}$$

$$S(t) = g(S(t-1), \mathbf{a}(t-1), \mathbf{s}(t)) \tag{2.3}$$

[1]Equations 2.2 and 2.3 are identical to Equation 2.1 when g is the concatenation operator, and $f(S(t))$ simply translates its argument into a call of f_t.

where $S(t)$ is the state of the robot at time t.

The Environment. The robot's environment interprets the sequence of action vectors and generates the sequence of sense vectors. It can be described as nearly the mirror image of the robot:

$$\mathbf{s}(t) = f'(E(t)) \tag{2.4}$$
$$E(t) = g'(E(t-1), \mathbf{a}(t-1)) \tag{2.5}$$

where $E(t)$ is the state of the environment at time t. Just as with f_t in Equation 2.1, and f and g in Equations 2.2 and 2.3, both f' and g' may be stochastic: different possible states might result from a given action in a given state, and different possible sense vectors can be produced in the same state on different occasions.

The Task. Equations 2.2 to 2.5 define a protocol by which a robot can interact with an environment. The robot acts in response to the sensations it receives. The environment responds to the robot's actions. Besides describing the actions that are performed by a robot, the functions f and g implicitly describe the task the robot performs. Since Equations 2.2 and 2.3 impose no limits on the complexity of the robot, they also impose no limit on the complexity of the robot's task. However, the task might often require less than all the information supplied in Equation 2.1.

2.3.2 Learning Tasks

Supervised Learning Tasks. Because $\mathbf{a}(t)$ can be used to describe the behaviour of a robot agent that performs a task, it can also be used to express the desired behaviour of an agent that learns to perform the task. In this case, Equations 2.2 and 2.3 describe a set of training examples for a *supervised-learning* agent. The training input to the agent at time τ would be $\mathbf{s}(\tau)$ – and possibly $\mathbf{a}(\tau - 1)$ – and the target output would be $\mathbf{a}(\tau)$.

Reinforcement Learning Tasks. A *reinforcement-learning* agent is somewhat more sophisticated than the supervised-learning agent. A teacher must be present to provide the supervised-learning agent with correct responses for each situation. In reinforcement learning, the correct action is never given. Instead, the agent must learn for itself which actions are correct in each situation.

To make learning possible without a teacher, a reinforcement environment supplies the agent with a *reinforcement signal*. The agent monitors changes in the reinforcement signal to decide which actions are *best*, where the best actions maximise the agent's expected reinforcement over time. More formally, the reinforcement signal is a function of the previous state of the environment and the most recent action taken:

$$r(t) = R(E(t-1), \mathbf{a}(t-1)) \tag{2.6}$$

where $E(t)$ was given in Equation 2.5.

The *correct* action at time t is any of the possible actions that maximises the expected sum of the future reward signals:

$$\mathbf{a}(t) = \underset{\mathbf{a}}{\operatorname{argmax}} \ \mathrm{E}\left[\sum_{\tau=1}^{\infty} \gamma^{\tau} r(t+\tau)\right] \tag{2.7}$$

where $\underset{a}{\operatorname{argmax}} f(a)$ returns the argument a that maximises $f(a)$, and γ is a *discount factor*, whose value often chosen to be less than 1.0 to avoid infinite sums. The expectation operator $\mathrm{E}[\cdot]$ is necessary for stochastic environments, where the next state is not a deterministic function of the current state and the action taken. The term inside the expectation in Equation 2.7 is the sum of the future reward signals and is called *cumulative discounted reinforcement*.

Given the space $\langle \mathbf{S}, \mathbf{A}, E, R \rangle$ where $\mathbf{S}$ is the set of all possible sensations, $\mathbf{A}$ is the set of all possible actions, E is the reinforcement environment, and R is the reward function for this environment, the *reinforcement-learning problem* is the problem of finding a control function $F : \mathbf{S}^* \rightarrow \mathbf{A}$ such that F maximises the reward R over time. $\mathbf{S}^*$ is the set of all possible sequences of sensations over time. F is often called control *policy*. Methods to solve such a learning problem will be reviewed in Section 2.4.1.

Lifelong-Learning Tasks. It is possible now to describe life-long learning for a robot agent. Life-long learning means that the robot has to learn a collection of control policies F_i for a variety of related reinforcement-learning tasks. Each of these reinforcement learning tasks, $\langle \mathbf{S}, \mathbf{A}, E_i, R_i \rangle$, involves the same robot with the same set of sensors, actuators, and may only vary in the particular environment E_i, and in the the reward function R_i that defines the goal states for this learning problem.

Of course, the agent could approach the lifelong-learning problem by handling each reinforcement-learning problem independently. However, the idea of continual learning offers the opportunity for synergy among the different reinforcement learning problems, which can speed up learning over the lifetime of the agent, since the various single tasks can be defined in terms of the same $\mathbf{S}$, $\mathbf{A}$, and potentially the same E. The agent should be able to reduce the difficulty of solving the i-th reinforcement learning problem by using the knowledge it acquired from solving earlier learning tasks.

2.4 Main Methods of Learning

The tradeoff between the amount of built-in knowledge and learned information has been acknowledged as one of the key issues in robot learning [6]. While reducing the built-in knowledge eases the programming task and reduces the learning bias, it slows down the learning process, and restricts therefore the applicability of automatic learning.

There is a great variety of useful learning techniques such as neural networks, evolutionary algorithms, dynamic programming as well as various combinations of these. All these techniques perform some kind of heuristic search or optimisation. Unfortunately, virtually all pure search methods scale up very badly as the complexity of the task to be learnt increases. Complex tasks usually require complex controllers, and these in turn require a lot of parameters to specify them. Moreover, most search methods rely on heuristics such as hill-climbing or genetic recombination, and these usually work badly where the parameter space is rugged and where performance feedback is sparse. Both these factors mean that trying to learn complex skills from scratch in a reasonable time using general learning methods is very hard.

According to Thrun and Mitchell [42], there will be no optimal, general learning technique for autonomous robots, since learning techniques are characterised by a trade-off between the degree of flexibility, given by the size of the gaps in the provided world model, and the amount of search required for filling these gaps. Generally speaking, the more universal a robot learning architecture, the more experimentation the robot is expected to take to learn successfully.

Being aware of this problem, Kaelbling *et al.* [23] have recently argued to give up *tabula rasa* learning techniques. They propose to guide the learning process by problem decomposition and incorporating prior knowledge about the robot and the learning task. On the other hand, we believe that by combining several learning methodologies, one can overcome the drawbacks of the individual approaches.

There are three main types of learning which become more and more popular in the robotics community. These are reinforcement learning, evolutionary approaches, and neural-network based techniques.

2.4.1 Reinforcement Learning Approaches

Reinforcement learning (RL) has been used for both learning new behaviours and learning to coordinate existing ones. It is currently perhaps the most popular methodology for various types of learning.

The robot's job is to find a behaviour policy, mapping states to actions. Reinforcement learning systems try to learn the policy by attempting all of the actions in all of the available states in order to rank them according to their appropriateness. There are two classes of reinforcement learning algorithms:

- *Model-free algorithms*. A policy is learned without a model of the state-transition and reinforcement function, e.g., Q-learning [44], adaptive heuristic critic [5, 39], and the TD(λ) algorithm [40].

- *Model-based algorithms*. Such a model is learned and used to find an optimal policy, e.g., prioritised sweeping [31] and real-time dynamic programming [4].

According to Kaelbling *et al.* [23], it is not yet clear which approach is best in which circumstances. In this work, we are mainly interested in model-free reinforcement learning algorithms.

One possible approach to temporal credit assignment – the problem of how to apportion reward and punishment to each of the states and actions that produced the final outcome of the sequence – is to base the appointment of reward on the difference between successive predictions. Algorithms using this approach have been termed *temporal difference methods* and have been studied for many years [5, 40].

In temporal difference learning methods, e.g. as described by Sutton [41], there is an evaluation function and an action policy. The evaluation function generates a value $e(x)$ for the current state x, which measures the goodness of x. An action a is chosen according to a certain action policy, based on the evaluation function $e(x)$. The action performed leads to a new state y and a reinforcement signal r. The evaluation function is then modified so that $e(x)$ is closer to $r + \gamma e(y)$, where $0 < \gamma < 1$ is a discount factor. That is, the function $e(x)$ is updated so that $r+\gamma e(y)-e(x)$ is smaller. At the same time, the action policy is also updated, to strengthen or weaken the tendency to perform the chosen action a, according to the error in evaluating the state: $r+\gamma e(y)-e(x)$. That is, if the situation is getting better because of the action performed, the tendency to perform that action is increased; otherwise, the tendency is reduced. In that way, the learning process is dependent upon the temporal difference in evaluating each state.

Q-learning by Watkins [44] is a variation of the temporal difference learning method, in which the policy and the evaluation function are merged into one function $Q(x,a)$, where x is the current state and a is an action. An action a is chosen based on the values of $Q(x,a)$, given x and considering all the possible actions a. The updating of $Q(x,a)$ is done based on minimising $r+\gamma e(y)-Q(x,a)$, where $e(y)=\max_{a} Q(y,a)$, that is, the temporal difference in evaluating the current state and the action chosen.

The basic limitations of all traditional, theoretical reinforcement learning algorithms is that they assume finite state-action spaces and discrete time models in which the state information is assumed to be immediately available. In real-life problems however, the state-action spaces are infinite, usually non-discrete, the time is continuous, and the system's state is not measurable. To date, no complete and theoretically sound solution has been found which is able to deal with such problems. However, work continues in this area [24].

The problem of learning the optimal policy can be seen as a search for paths connecting the current state with the goal in the state space. The longer the distance between a state and the goal, the longer it takes to learn the policy. Breaking the problem into modules effectively shortens the distance between the reinforcement signal and the individual actions, but requires built-in information.

In most RL work so far, the algorithms are not able to use previously learned knowledge to speed up the learning of a new behaviour. Instead, the robot must either ignore the existing policy, or worse, the current policy may be harmful while learning the next one. A notable exception is the work done by Thrun and Mitchell [42].

2.4.2 Evolutionary Approaches

Evolutionary algorithms constitute a considerably different learning approach. While reinforcement learning techniques use statistical and dynamic programming methods to estimate the utility of taking actions in states of the world, genetic algorithms and classifier systems perform a search in the space of behaviours in order to find one that performs well in the environment.

Genetic algorithms (GAs) can be seen as a technique for solving optimisation problems in which the elements of the solution space are coded as binary strings and in which there is a scalar objective function that can be used to compute the 'fitness' of the solution represented by any string. The GA maintains a 'population' of strings, which are initially chosen randomly. The fitness of each member of the population is calculated. Those with low fitness values are eliminated and members with high fitness values are reproduced in order to keep the population at constant size. During the reproduction phase, operators are applied to introduce variation in the population. Common operators are crossover and mutation [18].

A *classifier system* is a control system based on simulated evolution. It consists of a population of production rules, which are encoded as strings. The rules can be executed to implement an action function that maps external inputs to external actions. When the rules chain forward to cause an external action, a reinforcement value is received from the world. Holland developed a method, called the bucket brigade algorithm [21], for propagating reinforcement back along the chain of production rules that caused the action. This method is an instance of the class of temporal difference methods, thoroughly investigated in the field of reinforcement learning. The standard genetic operations of reproduction, crossover, and mutation are used to generate new populations of rules from old ones.

Evolutionary approaches have recently attracted much attention; a lot of work has been conducted to evolve robot controllers, e.g., Colombetti and Dorigo [8], Cliff *et al.* [7], and Miglino *et al.* [29]. Yet, the tasks achieved so far, such as obstacle avoidance or light seeking, are relatively simple. It remains to be shown how evolution-based approaches scale up, i.e., whether they can evolve controllers for complex tasks as well.

2.4.3 Connectionist Approaches

Artificial neural networks have become a very popular tool in robot learning. Typically, they have been used to learn numerical functions for calibration or parameter adjustment, and also to learn new behaviours.

Neural networks derive their properties and capabilities from the collective behaviour of simple computational mechanisms at individual neurons. Computational advantages offered by neural networks include:

- *Knowledge acquisition under noise and uncertainty.* Neural networks can perform generalisation, abstraction, and extraction of statistical properties from noisy and inconsistent input data.
- *Adaptivity.* Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment.
- *Flexible knowledge representation.* Neural networks can create their own representation by self-organisation.
- *Efficient knowledge processing.* Neural networks can carry out computation in parallel.

- *Fault tolerance*. Through distributed knowledge representation and redundant information encoding, the system performance degrades gracefully in response to faults.

The main advantage of connectionist systems is based in their inherent structure and function: they simply provide certain features from the beginning which are much more difficult to achieve by other techniques. For example, they have the advantage of that they immediately incorporate a mechanism for learning.

Usually, a single neural network – even with multiple layers – is not enough to build a complete robot agent. More structure is needed in which different neural networks can be hierarchically combined.

Chapter 3

Ontogenic Neural Networks

A relatively new class of neural networks tries to overcome the problem of a fixed network topology by adapting the network architecture during the training process. The flexible topology enables incremental learning and helps avoid local minima.

3.1 Topology Adaptation

One of the main strengths of artificial neural networks is their ability to adapt their interconnection weights to solve a given problem. One of their drawbacks, on the other hand, is the lack of methodology for determining the topology of the network, which affects important characteristics of the network's learning, such as training time and generalisation capability. *Ontogenic neural networks* try to overcome this problem by allowing the networks to adapt their topology as well as their weights during the learning process [15].

Ontogenic Algorithms. Learning algorithms for ontogenic networks are used not only to train the weights, but also to learn the topology needed to correctly learn the training data. If a problem is given to a conventional multi-layer network, the network may be too small to solve the problem, thus requiring more units before the network can map the entire training set correctly. On the other hand, larger networks are capable of learning simple mappings but are inefficient, and their excess of parameters usually result in poor generalisation. In these cases, a small network would be more appropriate.

At present, there is no formal way by which the network structure can be computed given a certain training set or application. The usual approach is to proceed by trial-and-error. In this process, the network designer has to pursue one of two possible strategies:

- *Network growing.* One starts with a feed-forward network too small for accomplishing the task at hand, and then adds a new neuron or a new layer of hidden neurons until the network is able to meet the design specification.
- *Network pruning.* One starts with a large feed-forward network with an adequate performance for the problem at hand, and then prunes the network by weakening or eliminating certain synaptic weights in a selective and orderly fashion.

Ontogenic learning algorithm perform these steps automatically, i.e., with no or minimal intervention of a user. We can classify ontogenic networks by the training procedure they are using.

Growing, or *constructive*, ontogenic neural networks typically start with a very small topology, for example, a single hidden neuron, and keep adding new neurons and connections, until the task at hand is solved.

Pruning, or *destructive*, neural networks, on the other hand, start with a large topology and reduce their size by eliminating neurons, connections, or both.

There are also *growing-pruning* methods which perform both operations, either in separate stages or in an interleaved manner.

An extensive survey and comparison of ontogenic networks goes beyond the scope of this work. Nevertheless, we will explain some typical network models in greater detail in order to give the reader good insight into how they work. For more information, we refer to Fiesler and Cios [15] and Fritzke [17].

3.2 Supervised Ontogenic Networks

The temporal transition hierarchies by Ring [34] are based on a growing method. In this network model, high-order connections are created by adding new units. A hierarchical organisation allows modulation of the weights using information from past time steps, without the need to add memory to the network using recurrent connections. The network starts with a two-layer topology with constant (single-order) connections. Learning is performed by gradient descent. High-order units are added when a weight is forced to increase in some training epochs and to decrease on other ones. Transition hierarchy networks will be discussed in more detail in Section 5.2.

GAL (Grow And Learn) is a supervised, growing-pruning technique proposed by Alpaydın [1]. It is designed to allow incremental category learning. The topology used is a three-layer network where only the size of the input layer is fixed. Each neuron in the output layer represents a category. The basic idea is that a new hidden node is created to match a specific training pattern. When a pattern is given to the system, each hidden unit is activated in proportion to its Euclidean distance from the pattern that it was created to match. The most highly active unit 'wins' and activates the output units to which it is connected with non-zero weights. When an input pattern activates a set of output units that is not the target pattern, a new unit is created to match that pattern. The output connections of a new unit are set to zero for all output units that are 'off' in the current target pattern.

GAL also allows connections weights to be modified in a manner similar to radial-basis-function networks. When a pattern activates the output units correctly, the hidden unit that 'won' is modified so that it will become even more strongly activated the next time this pattern is presented. This is done by modifying the hidden unit so that the pattern it best responds to is closer to the current pattern. During the so-called 'sleep' phase, units that were previously learned but which are no longer necessary due to recent modifications are removed to minimise the network complexity.

GAL is extremely fast and generally learns a training set in a few passes. The complete training set will be learned, but generalisation is to some degree sacrificed, due to the ease with which new units can be created solely for purposes of memorisation of a single pattern. Another drawback of GAL is that it only works for binary classification tasks. If the output is continuous, or even it is discrete but not binary, then the algorithm fails. GAL is somewhat inefficient: it creates new units with random initial weights and simply allows those units to learn appropriate values to reduce the error. Using a 'node splitting' algorithm, it could be tried to configure new units on the basis of the mapping learned so far.

3.3 Unsupervised Ontogenic Networks

There are also ontogenic networks for unsupervised learning. Typical representatives are the growing cell structures by Fritzke [16] and the topology-representing networks – a combination of neural gas and competitive Hebbian learning – by Martinez and Schulten [27]. I will review only Fritzke's model here.

The main advantage of Fritzke's growing cell structures over traditional approaches is the capability of the network model to automatically find a suitable network structure and size. The network model is able to generate dimensionality reducing mappings, which may be used, for example, for visualisation of high-dimensional data or for clustering. The basic building blocks of the topology are hypertetrahedrons of a certain dimensionality k chosen in advance ($k = 3$ results in a triangle structure).

In contrast to Kohonen's self-organising feature map, which serves similar purposes, neither the number of units nor the exact topology has to be predefined in Fritzke's model. Instead, a growth process successively inserts units and connections. This makes it possible to continue the growth process until a specific network size is reached or a certain performance criterion is fulfilled. Because the topological neighbourhood relations are known, it is possible to insert new unit by 'node splitting'. The weight vector of the new neuron is interpolated from the weight vectors belonging to the ending points of the split edge.

Fritzke's network model has been successfully applied in mobile robotics. Zimmer [45] has used a modified version of the growing cell structures to learn qualitative topological world models. Representing and learning of qualitative topological maps is a life-long learning task. The learning system has continuously to classify input signals containing the sensory information of the robot's environment. In this case, a network with a predefined regular structure cannot be used as it is not know in advance which environments the robot will encounter during its lifetime.

3.4 Constructive Networks for Temporal Processing

3.4.1 Hidden State Problem

Standard feed-forward networks are insufficient for solving problems with a temporal component. In many robotics tasks, the robot's sensory input alone is insufficient to determine the correct action; see Equation 2.1. There may be locations in the environment where the sensory information is ambiguous. In the robotics literature, this condition is termed *perceptual aliasing* or *hidden* *state* problem. If the environment is more complex than Markov-1,[1] then its state, $E(t)$ in Equation 2.5, is *hidden*, since it cannot necessarily be deduced from the current sensory input alone. A sufficiently demanding task in an environment with a non-zero, finite number of hidden states requires the current state to be disambiguated using previous sensory information.

Hidden-state issues have been addressed in a variety of neural network models, such as TDNN (time-delay neural network), RCC (recurrent cascade correlation), the Jordan and Elman networks, BPTT (backpropagation through time), and RTRL (real time recurrent learning). A detailed discussion of these architectures is beyond the scope of this work. For this reason, we want to refer to Ring [34]. He compared ten state-of-the-art connectionist architectures concerning their suitability for solving the hidden-state problem. Quoting Ring's conclusions:

> In general, recurrent networks and other approaches to learning temporal tasks have great limitations. Specifically, they are either incapable of solving any but the simplest tasks, or they are dreadfully slow. Second-order recurrent networks are capable of learning complex grammars, but by sacrificing either speed or incremental learning. Because of their poor scaling behaviour, it is unreasonable to try using them in problems with more than just a few input and output units. [34, p. 33]

There are constructive neural network models particularly designed for solving temporal problems. Specifically, Ring's temporal transition hierarchies perform very well in comparison with other, non-constructive, network models. In the following, we describe a performance test which should help the reader to understand why this particular network model has been chosen in our work as learning mechanism. The details of the neural network model are described in Section 5.2.

3.4.2 Learning Gap Tasks

Transition hierarchy networks have been tested on the gap tasks introduced by Mozer [32]. These tasks are designed to test the ability of a learning algorithm to bridge long time delays. The training set in each task consists of two strings, such as `XabXcdefghi...` and `YabYcdefghi...`. Each string

[1] In a Markov-k environment, the probabilities of the next state depend upon the past k state/action pairs.

consists of a series of 43 input elements. In a supervised learning task, the neural network has to predict the next element of the sequence. Each string is presented to the network one input at a time – with the following input as target – until the network can predict both sequences correctly. A sequence is found to be predicted correctly if the highest activated output unit corresponds to the next item in the sequence for every element after the first.

The gap task is difficult because the two sequences are nearly identical. Only one item distinguishes them: the initial input element (an `X` or `Y`), which is presented again after a gap of several other inputs. In order to predict its second appearance, the initial element must be remembered across this gap while the other items are presented. The task can be made more difficult by increasing the size of the gap. In the example given above, the gap is two.

We have re-implemented the experimental setup described by Ring [34, pp. 79-81]. We can confirm his results: a temporal transition hierarchy network can learn gap task very efficiently. For example, it can learn gaps of 2, 10, and 40 in 4, 12, and 42 epochs, respectively, by creating 8, 24, and 84 units, respectively.

These results, which are very good in comparison to other network models for temporal processing, are only possible if inputs are locally encoded (one input unit for each unique sequence item). In our experiments, we found that the network converges much slower when distributed representations are used.

From these experimental results, we can conclude that transition hierarchy networks have the following properties:

- The network learns very quickly using locally encoded input patterns. Each new unit is dedicated to deal with a specific, exceptional situation in the sequence of input patterns.
- The local representation of information and the growing process may avoid local minima in the search for a suitable network configuration.
- The flexible topology can enable the absorption of new data through retraining. That is, the network learns to some extent incrementally.
- No trial-and-error process is needed to find a suitable network topology.

These features are very relevant for continual learning.

Chapter 4

Continual Robot Learning

4.1 Related Learning Approaches

4.1.1 Skill Acquisition in Psychology

There are many theories about learning and memory in psychology addressing issues of continual development. In research on humans and animals, it is often distinguished between the acquisition of declarative knowledge (memory) and the acquisition of procedural knowledge (skills). Herein, we are mainly concerned with skill acquisition.

Anderson [2] proposed that skills go through three characteristic stages as they develop. In the first stage, called *cognitive stage*, the learner often works from instructions or an example of how the task is to be performed. In the second stage, called *associative stage*, the skill is said to make a transition from a declarative representation to a procedural representation. It becomes more fluid and error free. The third stage is the *autonomous stage* in which the skill becomes continuously more automated and rapid, and cognitive involvement is gradually eliminated.

The cognitive stage of skill acquisition is closely related to problem solving. The theories about problem solving have been formalised by artificial intelligence. A problem-solving activity is usually seen as a search for a sequence of operators that will transform the current state into the goal state.

The associative stage of skill acquisition is characterised by a change of the representation: the declarative knowledge about states, goals and operators is somehow being converted into procedural knowledge. Production rules (condition-action pairs) are the most common form of procedural knowledge. The autonomous stage would be the execution of such rules.

4.1.2 Operator Acquisition

There are two requirements for solving a novel problem: acquiring the appropriate operators and deploying them. The mechanisms for operator selection are relatively well understood. The two principal mechanisms are difference reduction and operator subgoaling. On the other hand, only a few approaches have been proposed for operator acquisition.

Recently, Wang [43] proposed a machine learning approach to automatic acquisition of planning operators. In her approach, the operators are learned by observation and practice. During observation, the system uses the knowledge that is observable from expert solution traces. During practice, the learning system generates its own learning opportunities by solving practice problems.

The inputs to Wang's learning system are the description language for the domain, experts' problem solving traces, and practice problems to allow learning-by-doing operator refinement. The system's output is a set of operators, each described by a list of variables, preconditions, and effects. The operators are learned incrementally using an inductive algorithm. During practice, the system generates plans using incomplete and incorrect operators and repairs the plans upon execution failures.

Adding learning abilities to planning systems and the integration of planning, learning, and execution will become more and more relevant for applications in robotics.

4.1.3 Bottom-up Skill Learning

Most of the work on cognitive skill acquisition assumes a top-down approach. As described in Anderson's three-stage model, the agent first acquires declarative knowledge in a domain. Then practice changes this explicit knowledge into a more usable form, which leads to skilled performance. The knowledge used in skilled performance is procedural knowledge. It is commonly believed that skills are a result of proceduralisation of declarative knowledge. However, it has been argued that the opposite may also be true. It has been shown in computational experiments that declarative knowledge can arise out of procedural knowledge, referred to as *bottom-up* skill learning.

Sun and Peterson [38] designed an architecture for bottom-up learning in which symbolic declarative knowledge is extracted from a reinforcement learning connectionist network. Their system consists of two levels. The top level is a rule level and the bottom level is a reactive level. The reactive level contains procedural knowledge, acquired by reinforcement learning, and the rule level contains declarative knowledge, acquired through rule extraction.

The reactive level of the architecture is implemented as a neural network with four layers. The first three layers from a conventional feed-forward network that learns the utility values of actions using backpropagation. The forth layer, with only one unit, performs stochastic decision making.

The rule level of the architecture performs rule acquisition and refinement. The basic idea is as follows. If some action decided by the reactive level is successful (receives positive reinforcement), then there might be general knowledge that can be extracted. In this case, a rule is extracted that corresponds to the action selected by the reactive level. The extracted rule is added to the 'rule network'. In doing so, the most specialised rules are removed from the rule network so that only the most general ones are kept. The extracted rule is verified in subsequent interactions with the environment. If the rule is not successful, then the rule is made more specific and exclusive of the current case. If the rule is successful, it is generalised, to make it more universal.

Sun and Peterson's learning architecture belongs to the class of hybrid connectionist-symbolic models [37]. It combines the learning abilities of a connectionist model with operations for rule extraction and refinement using symbolic representations.

4.2 Ingredients of Continual Learning

One way to learn a sequential decision task, such as navigating a maze, is trial and error: repeated practice gradually gives rise to a set of procedural skills that deal specifically with the practiced situations. However, such skills may not be transferable to truly novel situations since they are also embedded in specific contexts and tangled together. In order to deal with novel situations, the agent – in our case the robot – needs to discover some general rules. Generic knowledge helps to guide the exploration of novel situations and reduces the time necessary to develop specific skills in new situations. Life-long learning, as described in Section 2.3.2, is really useful for a robot only if it can acquire generic knowledge.

As mentioned in Chapter 1, the aim of this work is to investigate mechanisms for life-long robot learning. We are interested in a particular form of life-long learning, which is called *continual learning*.

Ring defined continual learning as follows:

> Continual learning is the constant development of complex behaviours with no final end in mind. It is the process of learning ever more complicated skills by building on those skills already developed. [34, Abstract]

Constructing an algorithm capable of continual learning is a difficult business. According to Ring [34], a continual-learning algorithm has to fulfil five requirements:

- *Autonomous behaviour*. The continual-learning algorithm should be autonomous. It should be able to receive input information, produce outputs that can potentially affect the information, it receives, and respond to positive and negative reinforcement. That is, it must behave in its environment and be able to assign credit to behaviours that lead to desirable or undesirable consequences.

- *Unlimited behaviour duration*. These behaviours should be capable of spanning arbitrary periods of time, i.e., their duration should have no limit.

- *Intelligent behaviour acquisition*. The continual-learning algorithm should be able to acquire new behaviours when useful, but should avoid acquiring them otherwise.

- *Incremental learning*. The learning system updates its hypotheses as a new instance arrives without reviewing old instances. That is, learning occurs after each experience rather than from a fixed and complete set of data. For continual learning, it is not known in advance what problems will be addressed.

- *Hierarchical development*. Extant mechanisms or behaviours are subsumed by newer, more sophisticated ones. The old components of the system are used as constituents of newly created components.

The necessity of the first three ingredients is obvious. We do not discuss them here in more detail. The most significant ingredients of continual learning are incremental learning and hierarchical development.

Many learning algorithms, such as backpropagation, are so-called batch algorithms requiring all training data to be collected in advance of the algorithm's execution. For many real-world applications, it is impossible to collect the data from all the problems in advance of training, and therefore incremental learning is needed.

It is a significant advantage for a learning system to acquire knowledge incrementally. An incremental-learning system is spatially economical and temporary efficient, since it need not store and re-process explicitly old experiences. Incremental learning is especially important for a learning system which continually receives input, since the system cannot wait for all instances to be available for learning.

In a continual learning system, hierarchical development can be used to build generic knowledge. The aim is to build a hierarchy of knowledge in which general rules cover default situations, and specialised rules deal with exceptions.

In the following chapters, a continual learning approach to robot learning is presented. The learning mechanism is based on a constructive neural network which learns incrementally and hierarchically.

Chapter 5

Learning of Sequential Decision Tasks

In this chapter, we will describe the learning scenario and the basic learning mechanism, invented by Ring [34], which are going to be used in the experiments on continual learning reported in Chapter 6.

5.1 The Learning Task

Sequential decision tasks are a suitable domain for studying continual learning techniques. They generally involve selecting and performing a sequence of actions in order to accomplish an objective. At certain points, the robot agent may receive reinforcements for its actions performed at or prior to the current state.

Sequential decision tasks are difficult to solve because the long-term consequences of taking an action are seldom reflected in the immediate payoff, making it difficult to assign credit to actions. The success of the learning process depends very much on how often the robot agent receives rewards.

The robot 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

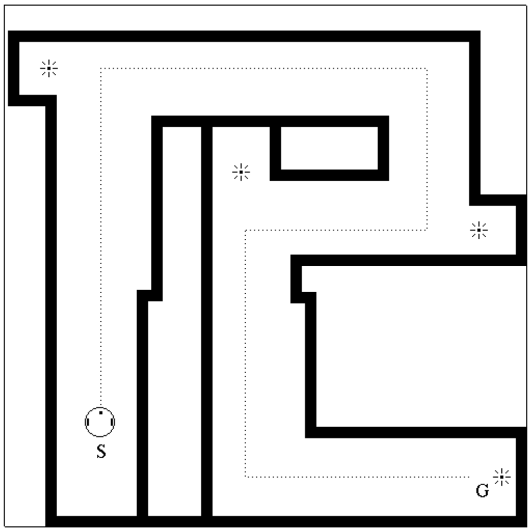


Figure 5.1: A test environment for learning a navigation task with delayed reinforcement.

from scratch, the robot is unlikely to find a sequence of actions for which it is given reward.

In this work, a learning task has been chosen where only delayed reinforcement is available. A robot agent 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 typical test environment is shown in Figure 5.1 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.

We assume that the robot agent has only a restricted set of actions and sensations. The robot can perform four different actions: *move_forward*, *turn_left*, *turn_right*, and *stop*. The action *move_forward* makes the robot move forward until it hits a wall. This allows the robot to move through an entire corridor with a single action. The actions *turn_left* and *turn_right* perform perform 90-degree turns in front of obstacles, and *stop* simply terminates the run. Using its sensor inputs, the robot is able to distinguish four different light-configurations: *light_ahead*, *light_right*, *light_left*, and *no_light*.

To solve the given navigation task, the robot agent has to learn to choose the right action each time there is a wall blocking its way. One possible approach for solving this sequential decision task is to learn an appropriate set of sensation-action rules. Preferably, the rule set should be minimal. A

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

Figure 5.2: A default hierarchy for solving the navigation task in Figure 5.1.

way of achieving this is to design a default hierarchy in which general rules cover default situations, and specialised rules deal with exceptions. A rule set with the properties of a default hierarchy can be seen in Figure 5.2. This rule set is a solution of the navigation task given in Figure 5.1. An action is selected at a particular time step t.

In the following section, it is discussed how such a rule set can be represented and learned by a constructive neural network.

5.2 Transition Hierarchy Networks

Ring has developed a constructive, high-order neural network, known as temporal transition hierarchies, that can learn continuously and incrementally. The following description of the network model is based on definitions given by Ring [34, pp. 51–53].

Network Structure. 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 robot 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.

This can be summarised as follows. Each unit u^i in the network is either: a sensory unit s^i; an action unit a^i; or a high-level unit l^i_{xy} that dynamically modifies w_{xy}, the connection from the sensory unit y to the action unit x. The action and high-level units can be referred to collectively as non-input units n^i.

Generally speaking, when no high-level units are present, the transition hierarchy network behaves like a traditional feed-forward neural network with no hidden units. On the input side of the network, we can find the sensations of the robot agent. On the output side, we can find its actions.

Network Dynamics. The activation of the units is very much like that of a classical single-layer network with a linear activation function:

$$n^i(t) = \sum_j \hat{w}_{ij}(t) s^j(t)$$

That is, the activation of the i-th action or high-level unit is simply the sum of the sensory inputs multiplied by the weights $\hat{w}_{ij}$ of their connections to n^i.

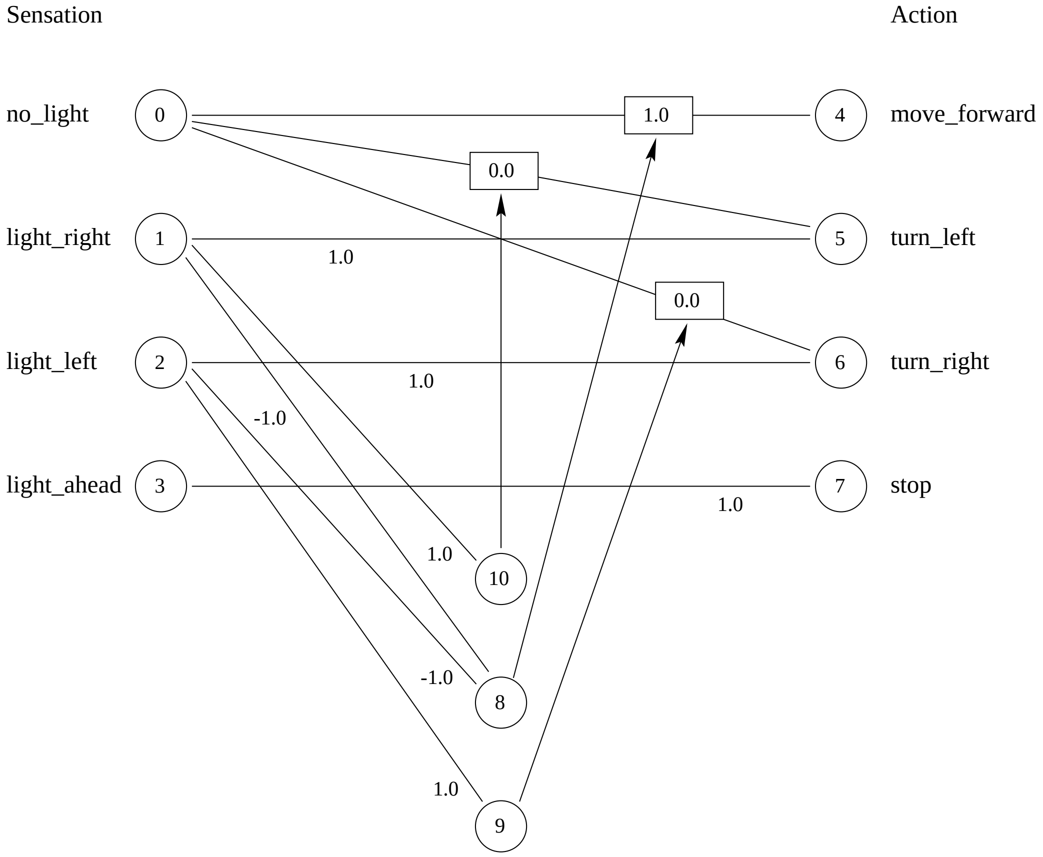
The use of a linear activation function and the lack of hidden units would normally cause limitations. However, these are *high-order connections* and are therefore capable of non-linear classifications. The *high-order weights* $\hat{w}_{ij}$ are defined as follows:

$$\hat{w}_{ij}(t) = \begin{cases} w_{ij} + l_{ij}(t-1) & \text{if a high-level unit } l_{ij} \text{ for weight } w_{ij} \text{ exists} \\ w_{ij} & \text{otherwise} \end{cases}$$

That is, if no l unit exists for the connection from j to i, then w_{ij} is used as the weight. If there is such a unit, its previous activation value is added to w_{ij} to compute the high-order weight $\hat{w}_{ij}$. It should be noted, that high-order connections in the classical sense are multiplicative, whereas the high-order connections used here are additive. However, it can be shown that these qualify as high-order connections in the usual sense [34].

An Example. The function of the high-level units can be illustrated considering the environment in Figure 5.1. Until it reaches the goal position, the robot agent has to change its direction of travel several times. The optimal sequence of decisions can be described through sensation-action rules, which can be implemented using a transition hierarchy network. Indeed, the network shown in Figure 5.3 is a computational representation for the rule set in Figure 5.2.

The net outputs are computed each time the robot detects an obstacle in its way. 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 light however, the decision on which action to take depends also on the sensation in the previous time step. This ambiguity – the hidden state problem – is solved using the high-level units 8, 9, and 10 as follows.



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

Figure 5.3: A transition hierarchy network to solve the navigation task in Figure 5.1 (above) and the outputs produced at different time steps (below).

In one case ($t = 1$), the robot agent should go forward when it senses darkness, while in another case ($t = 3$), it should turn right. To decide whether to go forward or turn right after darkness, the robot needs only to know whether it sensed light on the left-hand side in the previous step. A high-level unit can be built to deal with this specific situation. The high-level units 8 and 9 are activated after sensing light on the left-hand side. In this case, unit 8 has a negative activation and weakens the connection from unit 0 (*no_light*) to unit 4 (*move_forward*), whereas unit 9 has a positive activation and strengthens the connection from unit 0 (*no_light*) to unit 6 (*turn_right*) at the same time.

The function of the three high-level units in the example network can be summarised as follows. High-level unit 8 inhibits the default rule

$$\text{no\_light}(t) \rightarrow \text{move\_forward}$$

if *light_right* or *light_left* was active in the previous time step. High-level unit 9 activates the specialised rule

$$\text{no\_light}(t) \wedge \text{light\_left}(t-1) \rightarrow \text{turn\_right}$$

if *light_left* was active in the previous time step. High-level unit 10 activates the specialised rule

$$\text{no\_light}(t) \wedge \text{light\_right}(t-1) \rightarrow \text{turn\_left}$$

if *light_right* was active in the previous time step.

This example shows that a transition hierarchy network can represent a default hierarchy of sensation-actions rules. The interesting point is that the connection weights and the network topology can be determined automatically as described in the next section.

Supervised Learning. There is a supervised learning algorithm to determine the connection weights of transition hierarchy networks. The activation values of the units are continuous, and the activation function is differentiable. Therefore, a learning rule can be derived that performs gradient descent on the error surface. Since the activations of the high-level units at one time step are not required until the following time step, all unit-activations can be computed in a single forward-propagation.

As common with gradient-descent learning techniques, the network weights are modified so as to reduce the total sum-squared error E defined as follows:

$$E = \sum_t E(t)$$

$$E(t) = \frac{1}{2} \sum_i \Big(T^i(t) - a^i(t)\Big)^2$$

where $T^i(t)$ is the target value for $a^i(t)$.

Although the derivation of the learning rule is a bit lengthy (see Ring [34, pp. 54–59]), the result at the end is a simple learning rule, easy to understand as well as to implement. The final learning rule is:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \Delta w_{ij}(t)$$

$$\Delta w_{ij}(t) = s^j(t-\tau^i) \begin{cases} a^i(t) - T^i(t) & \text{if } n^i \text{ is an action unit } a^i \\ \Delta w_{xy}(t) & \text{if } n^i \text{ is a high-level unit } l^i_{xy} \end{cases}$$

where η is the learning rate, and τ^i is a constant value for each action or high-level unit n^i that specifies how many time steps it takes for a change in unit i's activation to affect the network's output. Since this value is directly related to how 'high' in the hierarchy unit i is, τ is very easy to compute:

$$\tau^i = \begin{cases} 0 & \text{if } n^i \text{ is an action unit } a^i \\ 1+\tau^x & \text{if } n^i \text{ is a high-level unit } l^i_{xy} \end{cases}$$

The only values needed to make a weight change at any time step t are: (1) the error computable at that time step, (2) the input recorded from a specific previous time step $t - \tau^i$, and (3) other weight changes already calculated. The third point is not necessarily obvious. However, each high-level unit is higher in the hierarchy than the units on either side of the weights it effects. This means that the weights may be modified in a simple bottom-up fashion. Error values are first computed for the actions units, then weight changes are calculated from the bottom of the hierarchy to the top so that the $\Delta w_{xy}(t)$ will already have been computed before $\Delta w_{ij}(t)$ is computed, for all high-level units l^i_{xy} and all sensory units s^j.

Ring has described the intuition behind the learning rule as that each high-level unit $l^i_{xy}(t)$ learns to utilise the context at time step t to correct its connection's error $\Delta w_{xy}(t+1)$ at time step $t+1$. If the information is available, then the high-order unit uses it to reduce the error. If the needed information is not available at the previous time step, then new units may be built to look for the information at still earlier time steps.

The example network in Figure 5.3 has been created automatically by the supervised learning algorithm. Only the connection weights have been rounded to the closest integer after learning.

Adding New Units. The network learns constructively and incrementally. New high-level units are created while learning. If one unit is reliably activated after another, there is no reason to interfere with the connection between them. Only when the transition is unreliable, a new unit is required. This is the case when the connection weight should be different in different circumstances. A new unit is added whenever a weight is forced to increase and decrease at the same time. The unit is created to determine the contexts in which the weights is pulled in each direction.

In order to decide when to add a new unit, two long-term averages are maintained for every connection. The first of these, $\Delta\bar{w}_{ij}$, is the *average change* made to the weight. The second, $\Delta\tilde{w}_{ij}$, is the *average magnitude* of the change. When the average change is small but the average magnitude is large, this indicates that the learning algorithm is changing the weight by large amounts but about equally in the positive and in the negative direction. That is, the connection is being simultaneously forced to increase and to decrease by a large amount.

Specifically, a new unit is constructed for w_{ij} when:

$$\Delta\tilde{w}_{ij} > \Theta|\Delta\bar{w}_{ij}| + \epsilon$$

where Θ and ϵ are some constant values.

The long-term averages are computed as follows:

$$\Delta\bar{w}_{ij}(t) = \begin{cases} \Delta\bar{w}_{ij}(t-1) & \text{if } \Delta w_{ij}(t) = 0 \\ \sigma\Delta w_{ij}(t) + (1-\sigma)\Delta\bar{w}_{ij}(t-1) & \text{otherwise} \end{cases}$$

$$\Delta\tilde{w}_{ij}(t) = \begin{cases} \Delta\tilde{w}_{ij}(t-1) & \text{if } \Delta w_{ij}(t) = 0 \\ \sigma|\Delta w_{ij}(t)| + (1-\sigma)\Delta\bar{w}_{ij}(t-1) & \text{otherwise} \end{cases}$$

where the parameter σ specifies the duration of the long-term average. A smaller value of σ means the average is kept for a longer period of time and is therefore less sensitive to momentary fluctuations.

Conclusions. Each machine learning technique has its good and bad points. Transition hierarchy networks are no exception. One problem with the network model is that there are no hidden units in the traditional sense, and the activation function is linear. Transition hierarchy networks use high-order connections. This means that the network can in fact compute non-linear functions and can make classifications that are not linearly separable. Nevertheless, these mapping are constructed from the inputs at previous time steps. Without previous inputs, the network can only generate linear outputs from its input at the current time step. As a result, the network can compute non-linear mappings only from repeated input data, i.e., if the network's input is repeated over multiple time steps.

Another issue is that of unlimited time delays. The system described is only capable of building hierarchies that span a fixed number of time steps. This means it can only learn Markov-k-tasks, though it can learn them when k is unknown. According to Ring [34], this is not necessarily a serious drawback, since the algorithm can still learn k regardless of its size.

In the following chapter, it is described how the network's ability of representing and learning default hierarchies can be applied for continual robot learning.

Chapter 6

Experiments on Continual Learning

In this chapter, we will describe experiments that show that transition hierarchy networks can be employed for continual learning of sequential decision tasks.

6.1 The Experimental Environment

In the experiments, we used the learning task described in the previous chapter, in which a mobile robot has to find its way to a specified goal position in a maze-like environment. For the given learning task, there are no particular hardware requirements so that a rather simple mobile robot can be used. Because no real robot was available, a physically realistic simulator had to be used. In particular, we have chosen the Khepera^{TM}Simulator by Michel [28].

The Khepera^{TM} is a miniature mobile robot developed by Mondada *et al.* [30]. It has a circular shape with a diameter of 55 mm. The robot is built according to a modular concept as shown in Figure 6.1. In our experiments, the robot was used in the basic configuration, without vision and gripper.

The Khepera^{TM} can detect obstacles and light sources using its sensors. It is equipped with eight infrared proximity and light sensors, six in the front and two in the back. Each sensor has an angular resolution of about 60 degrees. The maximal distance the infrared sensors can detect is about 40 mm.

The robot has two wheels which can be individually driven by DC motors. It can move forwards or backwards equally well or wheel on the spot. In the

Figure 6.1: The Khepera^{TM}robot with additional turrets, a stereoscopic vision and a gripper turret.

experiments, the robot was supposed to travel through a structured, maze-like environment. Therefore, we have predefined a small set of medium-level motor and pattern-recognition behaviours.

We have restricted the number of actions the robot can perform by defining four motor controllers: *move_forward*, *turn_left*, *turn_right*, and *stop*. As assumed in Section 5.1, the simulated robot can move along a corridor, perform 90-degree turns in front of obstacles, and stop at any position.

The robot can detect four different light-configurations: *light_ahead*, *light_right*, *light_left*, and *no_light*. These pattern-recognition behaviours evaluate the output of the robot's light sensors.

The motor controllers and light-detection skills form a set of primitive actions and sensations, which can be used to built a robot controller for the navigation task. The primitive actions and sensations have been implemented by hand-coding. The behaviour to coordinate the primitive actions and sensations, however, should be found through learning.

6.2 Learning by Imitation

The robot does not need to learn the navigation behaviour from scratch. Several researchers have proposed to use imitation as a way for robots to learn new skills, e.g., Kuniyoshi *et al.* [25], Hayes and Demiris [20], Demiris and Hayes [11], and Bakker and Kuniyoshi [3]. 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 [20]. 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 skills in addition to the motor-controllers and light-detectors:

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.
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, an Elman network [13]. Both networks have been trained off-line using typical sensor readings. Details of the implementation are given in [19].

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 turn-detection behaviour.

Learning by imitation is meant to go beyond the simple repetition of the teacher's actions. It should help to overcome the practical problems of traditional reinforcement learning techniques by gaining information about successful action sequences on the one hand, and by reusing the acquired behaviours on the other hand. The idea of imitative learning is therefore closely related to continual learning.

The need for acquiring generic knowledge makes the situation-action association the crucial part of the imitative-learning approach. The choice of a computational mechanism for the sensation-action association depends surely on the amount of information to be stored, on the learning tasks, and on 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 in the learning process, a transition hierarchy network seemed to be a good choice. It should be noted that none of the many imitative learning approaches proposed so far include a learning mechanism that can acquire sensation-action rules in an incremental and hierarchical fashion.

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 described in Section 5.2.

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 by the teacher; 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, we have used the test environment shown in Figure 5.1. The sequence of input and target patterns obtained by observing the teacher robot is equivalent to the patterns given in the lower part of Figure 5.3. The learning algorithm did converge very quickly (in less than 30 training epochs). The transition hierarchy network shown in Figure 5.3 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.

To summarise, the robot has learned by imitation an optimal action sequence to reach the goal position in the given maze-like environment. Specifically, it has acquired a set of sensation-action rules which are stored in a transition hierarchy network. It remains to be shown that this knowledge can be used in similar situations and learning tasks.

6.3 Bootstrapping Reinforcement Learning

The robot must be able to adapt also when there is no teacher robot available. In this case, it is necessary to go back to traditional reinforcement learning techniques. However, reinforcement learning does not need to start from scratch. Instead, it can make use of the sensation-action rules that have already been learned by imitation.

To explore this idea, we have decided to use Q-learning. As described in Section 2.4.1, it is probably the most popular and well understood model-free reinforcement-learning algorithm. 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 used to control the robot in Section 6.2 represent actions and not utilities. So, theoretically one could not use the same network to estimate $Q(s,a)$. Nevertheless, we can use the outputs as initial approximations of the utilities, and the network as a starting point for the computation of the correct utility network.

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.

1. Read the current state x and present it as input pattern to the network.
2. Propagate the activations. The activations in the output layer, U_k, represent a prediction of $Q(x,k)$ for each action k.
3. Select an action a on the basis of the prediction; perform that action; read the new state y and the reinforcement r.
4. Make a prediction of $Q(y,k)$ for each action k; and update the utility $Q(x,a)$ as $u' = r + \gamma \max_k Q(y,k)$.
5. Adjust the network by back-propagating the error ΔU through it with the input x, where
$\Delta U_k = \begin{cases} u' - U_k & \text{if k = a} \\ 0 & \text{otherwise} \end{cases}$
6. Go to 1.

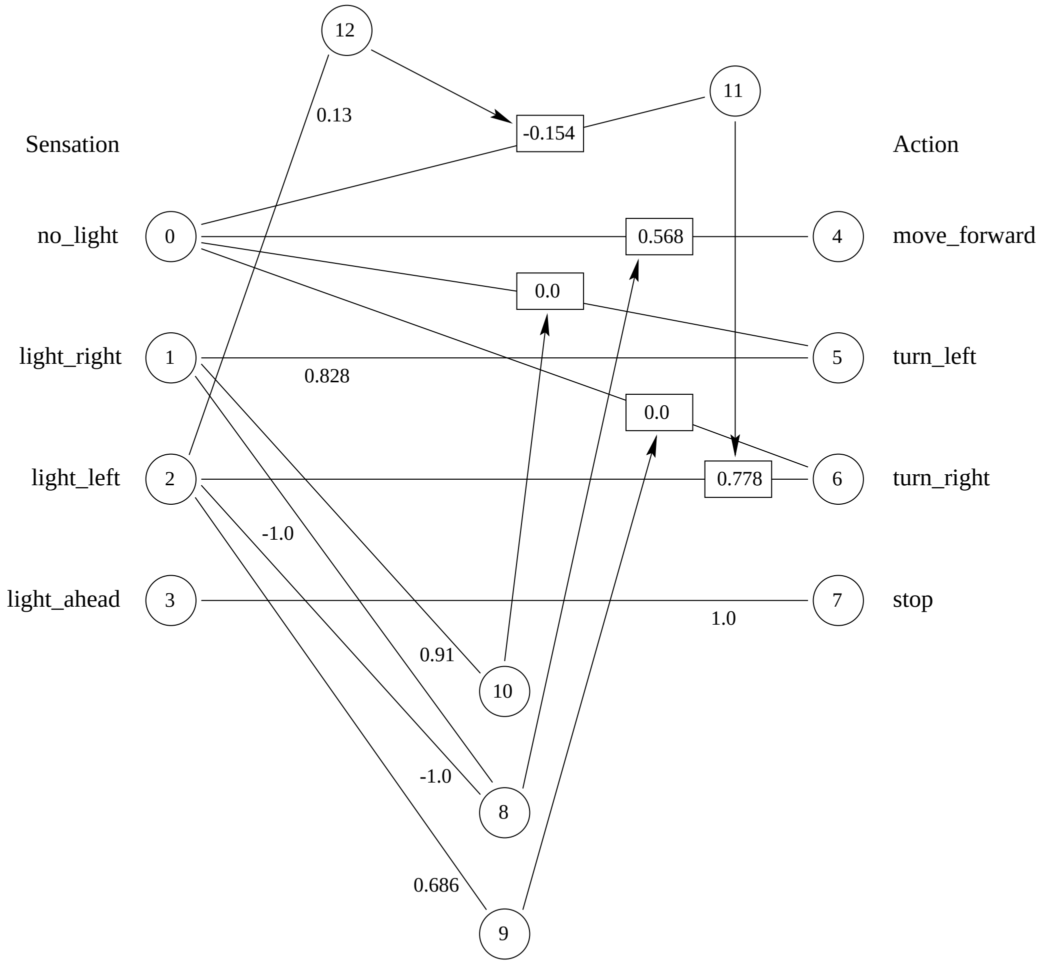
Figure 6.2: Algorithm for connectionist Q-learning by Lin [26, p. 298].

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 [26], which is given in Figure 6.2. 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 utility values at different time steps if utilities are discounted, i.e., if $\gamma < 1$. The effect of Lin's algorithm on the network in Figure 5.3 for $\gamma = 0.91$ can be seen in Figure 6.3. Two high-level units have been added during the learning process to distinguish the utility values of the rule $\langle \text{light\_left}(t) \rightarrow \text{turn\_right} \rangle$ 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



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*

Figure 6.3: A transition hierarchy network representing Q-values (above) and the outputs produced at different time steps (below).

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.

To summarise, we used a transition hierarchy network to learn Q-values for sensation-action rules, which were acquired before through learning by imitation. In that way, it is possible for a reinforcement learning agent to make use of previously acquired knowledge.

6.4 Learning Additional Behaviours

After the 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 a 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 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.
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 [34]. 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

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.

Figure 6.4: Algorithm for continual reinforcement learning with temporal transition hierarchies.

algorithm is made to learn correct utility values only for action sequences which have been found to be useful.

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 Figure 6.4. 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. 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

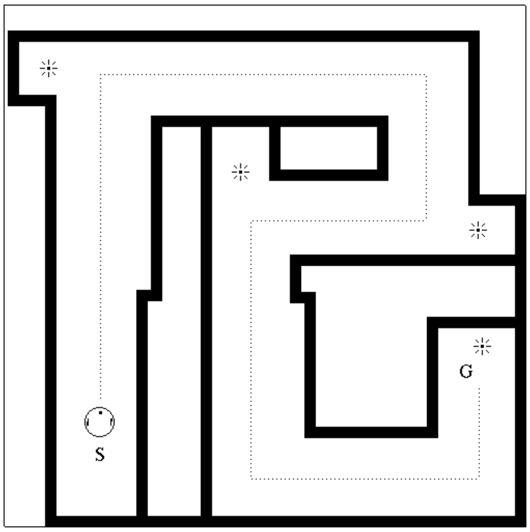


Figure 6.5: Changed test-environment requiring a revision of the rule set.

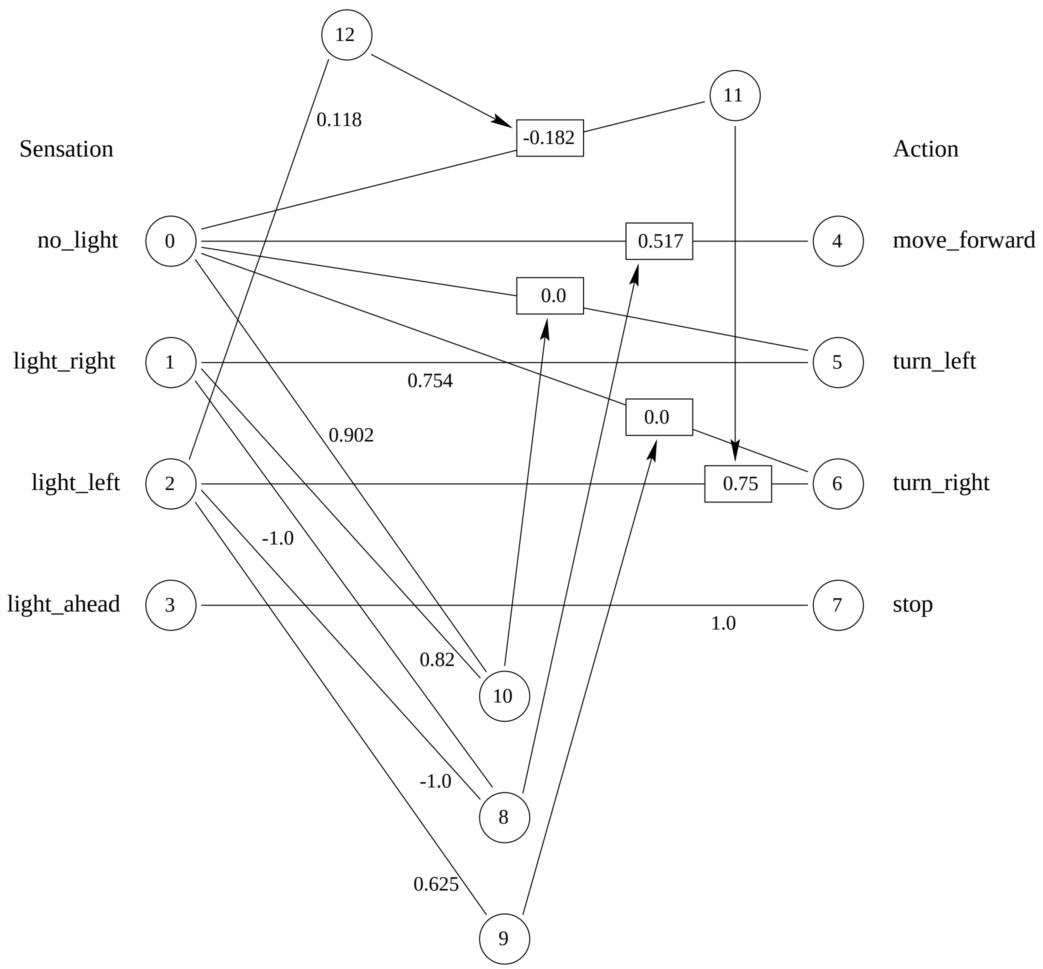
action must not be smaller than the utilities of the other actions. Therefore, the utility values of non-selected actions are reduced if necessary. Each time a solution has been found, TTH_{LTM} is trained (step 6) using the successful action sequence which has been recorded (step 5), and TTH_{STM} is 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 Figure 6.6 where the robot had to learn an additional rule to respond to the changed goal position:

$$\langle \text{no\_light}(t) \wedge \text{no\_light}(t-1) \rightarrow \text{turn\_left} \rangle$$

By applying Ring's supervised learning algorithm to the TTH_{LTM} network (step 6 in Figure 6.4), a new link between the units 0 and 10 was added and the weights were adjusted to represent the new Q-values. Indeed, the new rule was added in an optimal way.

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



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*

Figure 6.6: Expansion of the network following a change in the environment. To solve the navigation task in Figure 6.5, the robot had to learn the new rule $\langle$no_light$(t) \wedge$ no_light$(t-1) \rightarrow$ turn_left$\rangle$.

task. Despite the simplicity of the learning task and the absence of contradictory rules, the network is able to solve more complex learning task because of 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 Figure 6.4). 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.

Chapter 7

Discussion and Conclusions

Discussion of Results. Transition hierarchy networks seem to meet the requirements imposed on a mechanism for continual learning. They are very fast compared to other networks and learn in an incremental and hierarchical fashion. For example, they can represent and learn default hierarchies. This is very important because default hierarchies constitute a type of generic knowledge, and transition hierarchy networks can be used to integrate learning techniques. In turn, this is important because continual learning can only be practical if the robot employs several means of learning, such as learning by imitation and reinforcement learning.

In the experiments, it has been shown that a mobile robot can learn skills incrementally and hierarchically by imitative learning. The skills have been represented as sensation-action rules in a constructive high-order neural network. Moreover, it has been 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.

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 (as mentioned in Section 6.1), which enormously reduce the size of the search space the learning algorithm has to explore. It should be noted that these basic behaviours could be learned as well, for example, using unsupervised learning techniques. In this way, the predefined behaviours would correspond to the behaviours built up by Piaget's exploration mechanism [33], namely, a set of 'base behaviours' from which to draw on to produce imitatory behaviours, and a set of 'registrations' with which to compare and recognise behaviours.

In the experiments, we have used a neural network with a fixed number of input and output neurons, which correspond respectively to the predefined pattern-recognition and motor behaviours. It would also be possible to add new input and output neurons during the learning process, as adding sensory and action units does not affect previously learned behaviours.

Contributions. First, it has been shown that temporal transition hierarchy networks are able to construct default hierarchies of sensation-action rules. Second, it has been demonstrated that generic knowledge for robot control can be acquired solely by connectionist learning. In contrast to the hybrid connectionist-symbolic models mentioned in Section 4.1.3, an entirely connectionist approach has been applied. Third, a technique has been proposed for the integration of reinforcement learning, learning by imitation, and incremental, hierarchical development.

Future Work. 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 the experiments were simple but typical for autonomous robots. In the future, the approach should be applied to more complex learning tasks, preferably on a real robot. In that way, it can be investigated in which situations incremental learning is beneficial. There are other potential applications than learning of sequential decision tasks. For example, continual learning could also be applied for behaviour coordination, active object detection, and active localisation. In an active classification or localisation approach, the robot agent would learn about the objects in the environment by actively exploring them.

Bibliography

[1] E. Alpaydın. GAL: Networks that grow when they learn and shrink when they forget. Technical Report TR-91-032, International Computer Science Institute, Berkeley, CA, USA, 1991.

[2] J. R. Anderson. Acquisition of cognitive skill. *Psychological Review*, 89: 369–406, 1982.

[3] P. Bakker and Y. 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.

[4] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[5] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. In M. Gabriel and J. W. Moore, editors, *Learning and Computational Neuroscience*, pages 539–602. MIT Press, Cambridge, MA, USA, 1990.

[6] R. A. Brooks and M. J. Matarić. Real robots, real learning problems. In Connell and Mahadevan [10], chapter 8, pages 193–213.

[7] D. Cliff, I. Harvey, and P. Husbands. Explorations in evolutionary robotics. *Adaptive Behaviour*, 2(1):73–110, 1993.

[8] M. Colombetti and M. Dorigo. Robot shaping: Developing situated agents through learning. *Artificial Intelligence*, 71(2):321–370, 1994.

[9] J. H. Connell and S. Mahadevan. Introduction to robot learning. In *Robot Learning* Connell and Mahadevan [10], chapter 1, pages 1–17.

[10] J. H. Connell and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic, Norwell, MA, USA, 1993.

[11] J. Demiris and G. Hayes. Imitative learning mechanisms in robots and humans. In V. Klingspor, editor, *Proceedings of the Fifth European Workshop on Learning Robots*, Bari, Italy, 1996.

[12] M. Dorigo and M. 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.

[13] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[14] E. Fiesler and R. Beale, editors. *Handbook of Neural Computation.* Institute of Physics and Oxford University Press, 1997.

[15] E. Fiesler and K. Cios. Supervised ontogenic networks. In Fiesler and Beale [14], chapter C 1.7.

[16] B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. Technical Report TR-93-026, International Computer Science Institute, Berkeley, CA, USA, 1993.

[17] B. Fritzke. Unsupervised ontogenic networks. In Fiesler and Beale [14], chapter C 2.4.

[18] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, USA, 1989.

[19] A. Großmann. Integrating multiple learning techniques for robot learning. Master's thesis, School of Computer Science, The University of Birmingham, Birmingham, UK, 1996.

[20] G. Hayes and J. Demiris. A robot controller using learning by imitation. In *Proceedings of the Second International Symposium on Intelligent Robotic Systems*, Grenoble, France, 1994.

[21] J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, USA, 1975.

[22] L. P. Kaelbling. *Learning in Embedded Systems.* MIT Press, Cambridge, MA, USA, 1993.

[23] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Artificial Intelligence Research*, 4:237–285, 1996.

[24] Z. Kalmár, C. Szepesvári, and A. Lőrincz. Module based reinforcement learning for a real robot. Presented at the Sixth European Workshop on Learning Robots, Brighton (EWLR-6), UK, 1997.

[25] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6), 1994.

[26] L.-J. 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.

[27] T. M. Martinez and K. J. Schulten. Topology representing networks. *Neural Networks*, 7:507–522, 1994.

[28] O. Michel. *Khepera Simulator Package 2.0.* University of Nice Sophia-Antipolis, Valbonne, France, 1996. Available via the URL http://wwwi3s.unice.fr/˜om/khep-sim.html.

[29] O. Miglino, H. H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4), 1996.

[30] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturisation: A tool for investigation in control algorithms. In *Proceedings of the Third International Symposium on Experimental Robotics*, pages 501–513, Kyoto, Japan, 1993. Springer.

[31] A. W. Moore and C. G. Atkeson. Prioritised sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

[32] M. C. Mozer. Induction of multiscale temporal structure. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 275–282. Morgan Kaufmann, San Mateo, CA, USA, 1992.

[33] J. Piaget. *Play, Dreams and Imitation in Childhood.* W. W. Norton, New York, NY, USA, 1992. Original work published 1945.

[34] M. B. 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.

[35] 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.

[36] L. Steels. The artificial roots of artificial intelligence. *Artificial Life*, 1 (1), 1994.

[37] R. Sun. Hybrid connectionist-symbolic models: A report from the IJCAI'95 workshop on connectionist-symbolic integration. Available via the URL http://www.cs.ua.edu/faculty/sun/sun.html, 1996.

[38] R. Sun and T. Peterson. A hybrid agent architecture for reactive sequential decision making. In R. Sun and F. Alexandre, editors, *Connectionist-Symbolic Integration*, chapter 7. Lawrence Erlbaum Associates, 1997.

[39] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, USA, 1984.

[40] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[41] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceeding of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[42] S. Thrun and T. M. Mitchell. Lifelong robot learning. Technical Report IAI-TR-93-7, Department of Computer Science III, University of Bonn, Bonn, Germany, 1993.

[43] X. Wang. *Learning planning operators by observation and practice*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1996.

[44] C. J. C. H. Watkins. *Learning with delayed rewards*. PhD thesis, University of Cambridge, Cambridge, UK, 1989.

[45] U. R. Zimmer. *Adaptive approaches to basic mobile robot tasks*. PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany, 1995.

Declaration

All the work enclosed was done by me except where otherwise stated. Every section that reports or paraphrases work by other authors indicates who the other authors are and gives a reference to the article, book, or other source. All quotations from other work are explicitly indicated as such and a reference to the original is given.

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfmittel verwendet, sowie Zitate kenntlich gemacht habe.

Birmingham, August 1997 Axel Großmann

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT

Name, Vorname des Diplomanden:	Großmann, Axel
Immatrikulationsnummer:	1038085

Thema:

Towards Continual Learning for Robots

Zielstellung:

Autonomous robotics must move towards real-world applications as opposed to the toy problems studied up to now. One way to deal with the arising design and control problems is to use automatic learning techniques. In particular, we will investigate continual and hierarchical learning in connectionist systems. In the experiments, we will use a robot controller which can learn by imitation as well as by reinforcement. The main idea is to use constructive, higher-order neural networks for learning the behaviours of an autonomous, mobile robot.

Betreuender Hochschullehrer:	Prof. Steffen Hölldobler
Institut:	Künstliche Intelligenz
Lehrstuhl:	Wissensverarbeitung
Beginn am:	1. März 1997
Einzureichen am:	31. August 1997

Verteiler:	Prüfungsamt	
	HSL	Unterschrift des betreuenden
	Student	Hochschullehrers