Diplom-Thesis

Implementation and Evaluation of a Connectionist Learning Architecture in a Simulated "Brio Labyrinth Game"

Larbi Abdenebaoui
1662320

University of Bremen
Fachbereich 3
Informatik

Supervisor:
Prof. Dr. Frank Kirchner

June 25, 2007

Abstract

This thesis presents a connectionist architecture, which is able to learn how to play the popular Swedish game "Brio Labyrinth Game". The aim of the game is to manoeuvre a steel ball through a complex maze with holes and walls by tipping it in two planar directions using two knobs. The thesis has two main goals. The first goal is to simulate the game using ODE (Open Dynamics Engine) [23] that allows a realistic reproduction of the physical properties of the game. The second is the design and implementation of a biologically inspired agent, which is able to learn how to play the game autonomously.

The proposed solution combines the principles of neural networks and reinforcement learning. To learn the proper movements and to deal with the convergence difficulty of reinforcement learning in a continuous state-space representation, a hierarchical learning architecture based on the connectionist Q-learning framework QCON [10] has been developed. As observed in human players and following the divide and conqueror paradigm, the labyrinth was subdivided into small regions, where a QCON is assigned to each region. Efficiency analyses of the algorithms were performed experimentally. It was shown that with the right parameters values, the solution scales up easily as the number of sub-areas increases.

Statement of Originality

I hereby declare to have written this Diploma Thesis on my own, having used only the listed resources and tools.

Contents

1 Introduction **7**

1.1 Structure of the Thesis 8

2 The Simulated Game **10**

2.0.1 Open Dynamics Engine 12

2.0.2 The Editor 12

3 Theoretical background **16**

3.1 Artificial Neural Networks 17

3.1.1 Historical Background 17

3.1.2 Biological Model 18

3.1.3 Artificial Neural Networks 19

3.1.4 The Backpropagation Algorithm 21

3.1.5 Summary 22

3.2 Reinforcement Learning 24

3.2.1 Introduction 24

3.2.2 Markov Decision Processes 26

3.2.3 Value Function 32

3.2.4 Optimality 33

3.2.5 Value iteration algorithm 34

3.2.6 Q-learning 35

3.2.7 Exploration versus Exploitation 36

3.2.8 Summary 37

3.3 Continuous Q-learning 38

3.3.1 Existing Approaches 38

3.3.2 Summary 41
3.4 Convergence in Reinforcement Learning 41
3.4.1 Convergence of Value-Iteration in Discrete RL. 41
3.4.2 Convergence in Continuous RL 44

4 The Learning Architecture 45
4.1 The Continuous Q-learning Framework QCON 45
4.2 the two holes Problem 50
4.2.1 Actions Representation 52
4.2.2 State Representation 52
4.2.3 Parameters sensitivity 53
4.3 Experiments 56
4.4 Results 57

5 Conclusion 62
5.1 Summary 62
5.2 Outlook 63

Chapter 1

Introduction

Motivated by biological paradigms, this thesis presents the design and evaluation of control architecture of a virtual player for playing the game "Labyrinth of Brio". The player can learn the task only through interaction with a predefined environment. The game "Labyrinth of Brio" is very interesting and requires concentration, a good perception of events, motor coordination, and fine motor skills. The aim of the game is to manoeuvre a steel ball from a starting position to a final position on the board by tipping it so that the ball moves without falling into any of the holes. The path along which to steer the ball is marked by a line and is partially bordered by the walls.

To enable an artificial agent to play the game, a physical simulation based on ODE [23] has been realized. The simulation allows intensive and qualitative tests of the controlling algorithm saving thereby the cost of the hardware and learning times. The Labyrinth is simulated in such a way that it is easy to change its complexity in form of the number of holes and walls and their distribution on the main board.

Like many other tasks in our every day-life, playing "Labyrinth of Brio" requires a learning phase, which is necessary to permit an effective use of the responsible muscles and an adequate coordination between sensing and acting to get the desirable ball balancing. In contrast to the classical engineering techniques, which assume generally an absolute knowledge about the agent environment to define a complete and accurate model of the problem, in this thesis

a biological concept is followed in coming up with a learning approach that deals with the complexity and the uncertainty of the environment. The learning approach used to solve the task is the reinforcement learning (RL) method Q-learning [4]

RL is an approach in machine learning that enables an autonomous agent to adapt incrementally its policy to choose actions through interaction with the environment based on received rewards. Like most real world problems, the input state space of the player in the "Labyrinth of Brio" is continuous and the learning agent uses artificial neural networks for generalizing over similar states.

The combination of Q-Learning and artificial neural networks is on one hand very fascinating and promising for real world problems. On other hand it is problematic since the discrete Q-Learning's guarantee of convergence no longer applies. To deal with the convergence difficulty, a divide and conqueror paradigm is followed in which a hierarchical learning architecture based on the connectionist Q-learning framework QCON [10] has been developed. Different parameter combinations are evaluated using statistical methods on a simpler version of the game, example the two holes problem(see Section (4.2)). It was shown that with the right parameter combination, the agent was able to learn how to play the whole labyrinth game.

1.1 Structure of the Thesis

This section gives the structure of the thesis with a short summary for each chapter.

Chapter two presents briefly the game and its physical simulation. Chapter three introduces the basic methods used in the learning architecture with their theoretical background with an overview over the related works. The first section gives a brief introduction to neural networks focussing on the multilayered neural networks with backpropagation as learning algorithm. The second section gives an overview of reinforcement learning with an illustration of the importance of MDP using a simple example for both completely and partially observable environments. The third section introduces the continuous Q-learning, a kind of reinforcement learning with continuous state space representation. In

addition to this, the convergence of the reinforcement learning in discrete RL for value iteration is discussed. Chapter four discuses the existing methods for solving tasks in reinforcement learning with a continuous state space representation, especially in the areas of combining the concepts of reinforcement learning and artificial neural networks. The QCON architecture which is introduced by Lin [10] is also introduced in this chapter. Chapter five introduces the implementation of the learning architecture and gives a performance evaluation using different sets of parameters.

Chapter 2

The Simulated Game

The "Labyrinth of Brio" is physically simulated to permit a realistic reproduction of all the physical parameters in terms of friction between the ball and the different elements of the labyrinth. For this purpose, the physic engine ODE (Open Dynamics Engine) [23] is used. The graphic rendering is realized with OpenGL Graphic Library (OpenGL), which is developed by Silicon Graphics (SGI). The basic simulator interface was already developed in the context of an internal project under the name "walker-sim" in "Robotics Group, University of Bremen". One important contribution of this thesis is the development and inclusion of the simulator for the "Labyrinth of Brio" into the exiting framework.

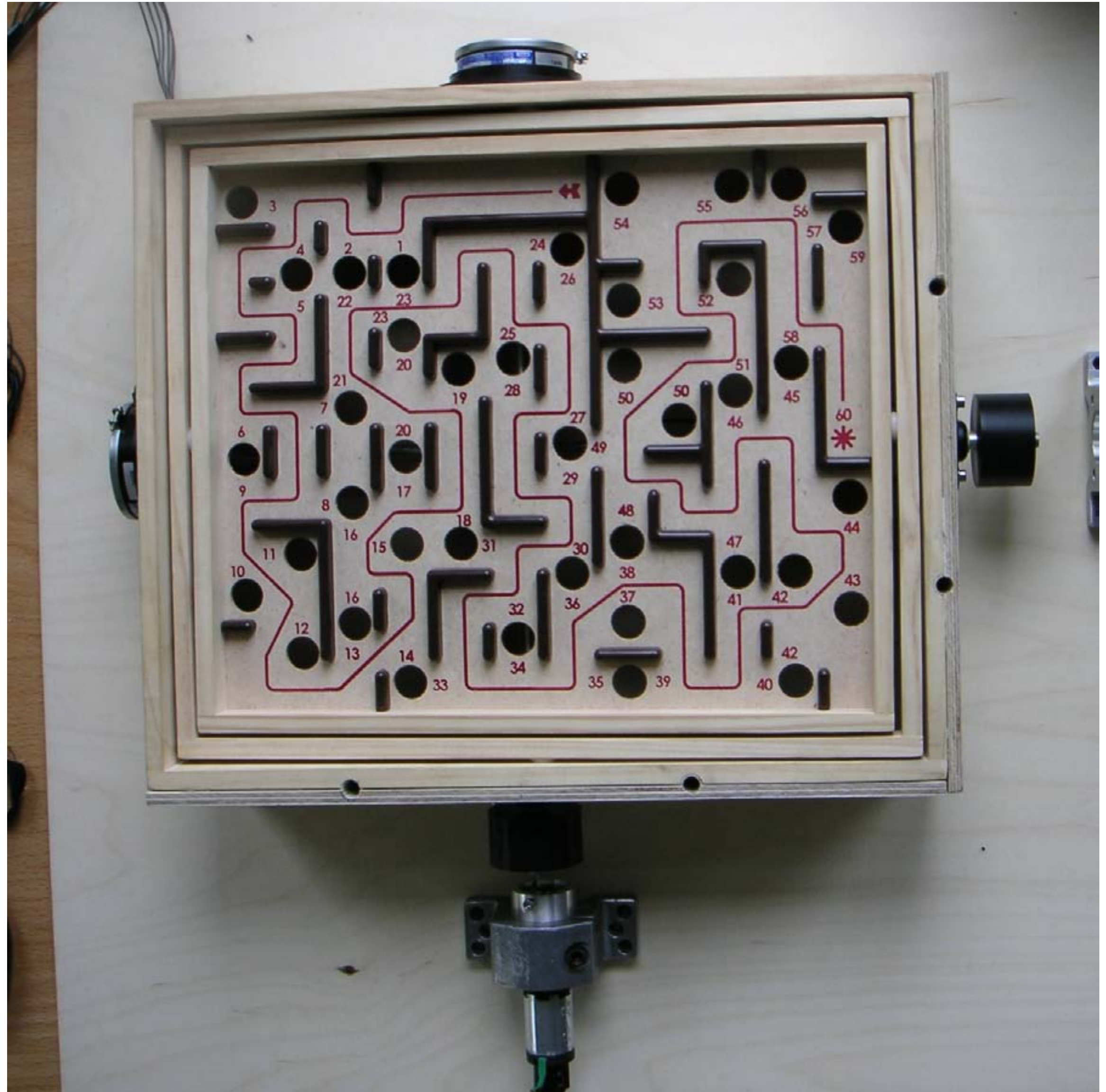

Figure 2.1: The original game of Brio in the construction phase to be fitted for controlling through a robot system. Motors should replace the knobs; two potentiometers are also implemented in the system.

The original game consists of a solid wooden box composed of a board and two knobs (Pitch, Roll) that permit to tilt a board in two dimensions by transmitting the torque through a flat belt system (see Figure 2.1). The board has a labyrinth with holes and walls, which are distributed so that they compose a path, which is clearly marked. The Goal for the player is to go as close as possible to the goal, which imply that the player follows the given path and avoids the holes. The further the player proceeds, the higher he or she scores. The Labyrinth is 32 cm long and 29 cm wide. It contains 40 holes.

2.0.1 Open Dynamics Engine

The simulator is based on the physics framework Open Dynamics Engine (ODE) [23], which is developed in 2001 from Russel L. Smith and is supported and continuously improved along the years from a large community . The ODE API is under the GNU Lesser General Public License. In this work the version 0.5 is used. The ODE engine permits high performance simulation of rigid body dynamics. It is platform independent using C/C++ API. It integrates advanced solutions to simulate several joint types and permits also to simulate collision between the geometries with respect to the friction. ODE uses a first order integrator characterized by its stability and rapidity. Higher order integrators are planned in future versions. The contact and friction model are based on the Dantzig LCP solver as described by Baraff, although a faster approximation to the Coloumb friction model is implemented.

2.0.2 The Editor

An editor is implemented that permits the user to change quickly the configuration the labyrinth. The user can add walls, holes or delete them by selecting the appropriate cells (see Figure 2.3)

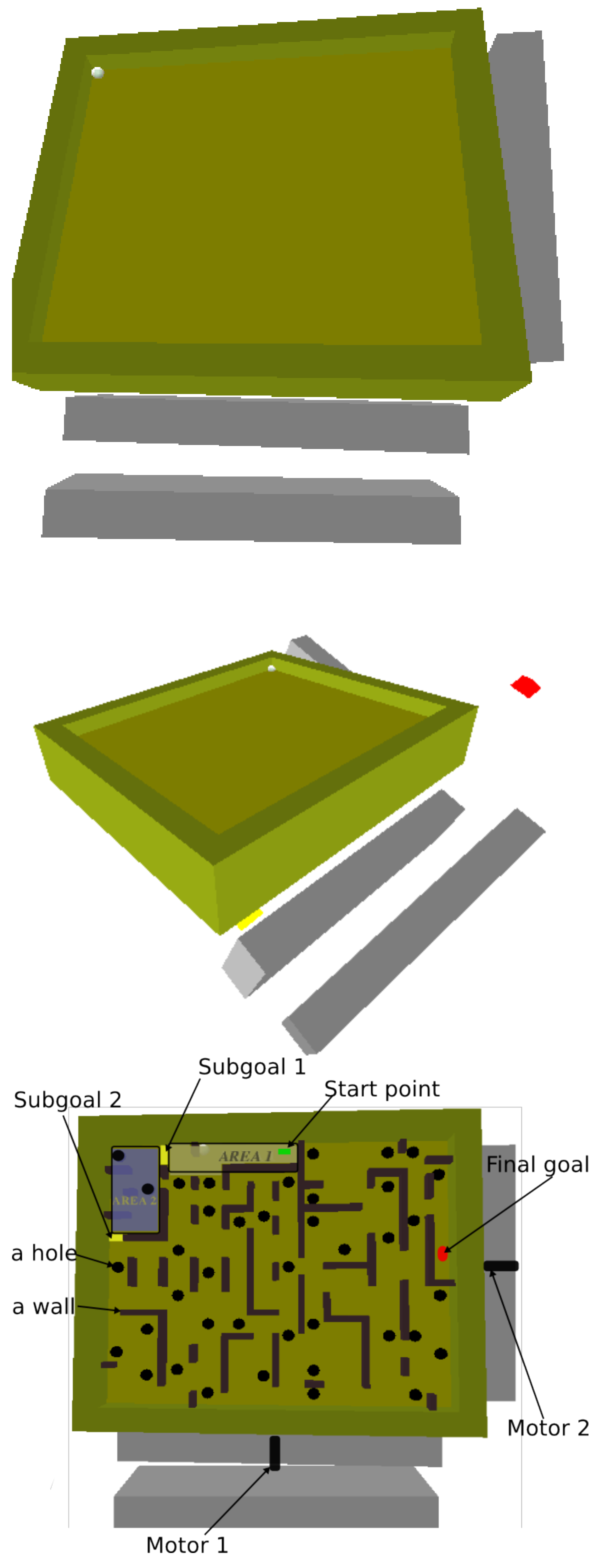

Figure 2.2: The simulation of the "Brio Labyrinth Game"

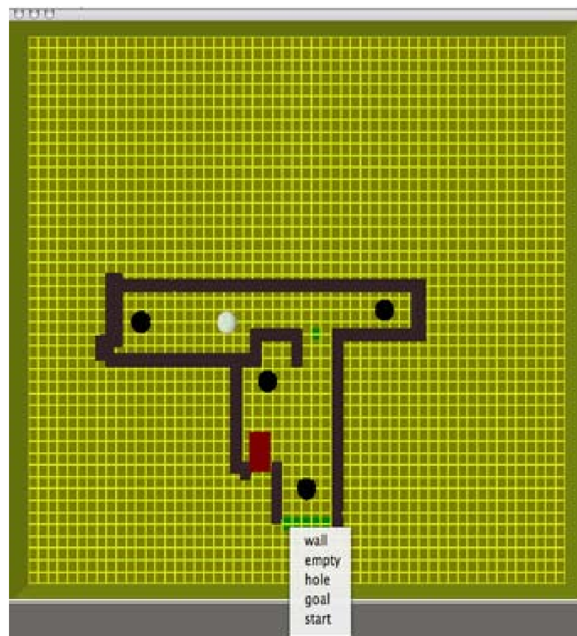

Figure 2.3: Illustration of the editor on the simulated game "Labyrinth of Brio"

Chapter 3

Theoretical background

3.1 Artificial Neural Networks

The Artificial Neural Network (ANN) presents a simplified[1] mathematical model of the biological nervous system and consists of an intensively connected group of processing units called artificial neurons. Their high degree of connectivity allows a distributed computation and with it an ability to modulate and learn complex relationships between inputs and outputs.

In the solution presented in this thesis the ANNs are used for the representation of the action-value functions allowing a generalization over continuous state-space.

3.1.1 Historical Background

One of the first neural model is from McCulloch and Pitts [11] and uses solely binary signals. Rosenblatt proposed in 1962 the perceptron [16], which uses numerical weights and presents a more general computing model. The perceptron generated much interest because of its ability to solve certain pattern classification problems. This interest started to fade in 1969 when Minsky and Papert [12] provided mathematical proofs of the limitations of the perceptron computational capability. In particular, it is incapable of solving the classic exclusive-or (XOR) problem.

The last decade, however, has seen renewed interest in neural networks, both among researchers and in areas of application. The development of more powerful networks, better training algorithms, and improved hardware has all contributed to the revival of the field. Neural-network paradigms in recent years include the Boltzmann machine, Hopfield's network, Kohonen's network, Rumelhart's competitive learning model, Fukushima's model, and Carpenter and Grossberg's Adaptive Resonance Theory. The field has generated interest from researchers in such diverse areas as engineering, computer science, psychology, neuroscience, physics, and mathematics. The power and usefulness of artificial neural networks have been demonstrated in several applications.

In this thesis we describe and use one of the most used neural model, the feed-forward network with the Backpropagation algorithm.

[1] the mathematical model is considered simple because of the complexity of the biological paradigm

3.1.2 Biological Model

The main objective of the ANN is to reproduce the characteristics of the biological computing which allow animals to cope autonomously with different real-life problems beginning with the so-called simple skills like reflexes, coordination of movement, surmounting obstacles and ending with very complex cognitive behaviors like learning, applying knowledge and changing preferences.

The basic building blocks thereby are the neurons. A neuron is a small cell that receives electro-chemical stimulus from several sources and responses with an electrical impulse which will be transfused again to other neurons or effector-cells. One neuron is typically composed of a cell body -or a soma-, a nucleus, dendritic trees presenting the input connections through synapses to other neurons and an axon responsible of carrying the nerve signals away from the soma and transmit it to the target neurons through the axon terminal (Fig.3.1).

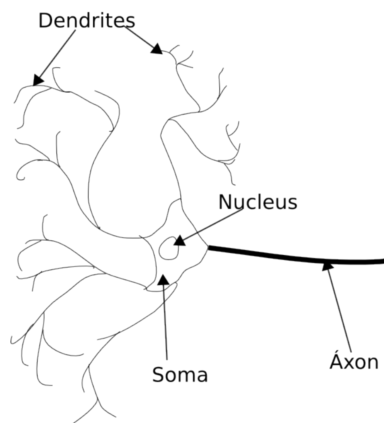

Figure 3.1: Structure of a typical neuron

One neuron can be connected to thousands of other neurons. The connections are done via inhibitory or excitatory synapses. Those can either increase or decrease activity in the target neuron. The Activity of neurons is determined from an intern electrical potential called the Membran-potential. When it attains a threshold the neuron fires by sending a set of action potentials through the axon to the (chemical) synapses where neurotransmitter are released causing again a localized change in potential in the membrane of the target neuron.

In the actual thesis we will not deal with more details but it is important here to notice how difficult it is to simulate the behaviour of the biological neural network which is influenced by a multitude of parameters. However there is a major theory, claims that the basics of learning result from the synaptic plasticity allowing connection strengths between two neurons to change.[21], [5] Donald Hebb [7] was the first to introduce the importance of metabolic changes for learning in biological systems and states:

> *Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability ... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*

The Hebbian learning refers in general to an abstraction of the original principle proposed by Hebb. In The ANN the most of learning methods try to implement this principle for adjusting weights between nodes so that each weight better represents the relationship between the nodes.

3.1.3 Artificial Neural Networks

One Neuron

In general one neuron can be simulated with a composition of a weighted summation and a primitive function f. The first part reduce the n argument to a single numerical value and the activation function f produces the output of this unit taking that single value as its argument (Fig.3.2). Note that in many literature the function f is called transfer function. We find that both terms have the same meaning in this context because the transfer can happen only, and only if the neuron is activated. The type of the activation function influences the behaviour of an ANN. Which typically falls into one of three categories:

- linear: The output activity is proportional to the total weighted output.
- threshold : the neuron is fired, only if the weighted summation of the input is greater than or less than some threshold value.

- sigmoid : the output varies in this case continuously but not linearly as the input changes. That allows neural networks based on it to represent non-linear relationships between inputs and outputs. Sigmoid types bear a greater resemblance to the behaviour of real neurons than do linear or threshold units.

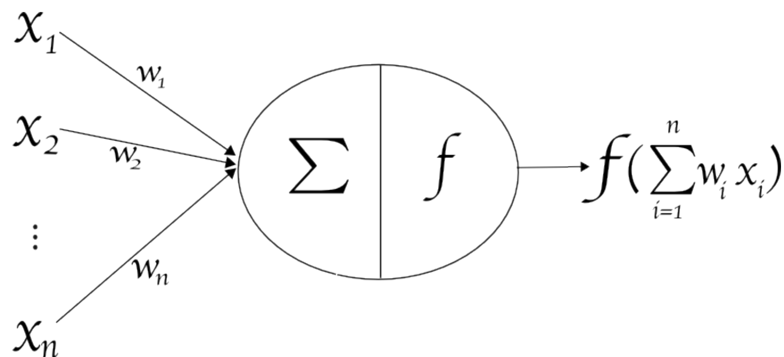

Figure 3.2: A general neuron model. The output is a result of the composition of two operations first the summation of the inputs multiplied with the associated connection weight and an activation function f

Feedforward Neural Network

A feedforward neural network consists of several layers, and each layer has a number of neurons in it. Neurons in one layer are connected to multiple or all neurons in the next layer. Input is fed to the neurons in input layer, and output is obtained from the neurons in the last layer. Feed-forward ANNs allow signals to travel one way only, from input to output. There is no feedback i.e. the output of any layer does not affect that same layer.

The Feed-forward neural network consists of three groups, or layers of units: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units. (See Figure 3.3). The activity of the input units represents the raw information given to the network, the activity of each hidden units is determined by the activities received from the input units and from the weighted connections. In the same way the behavior of the output units depends on the activity of the hidden units and on the connections that link them to the output units.

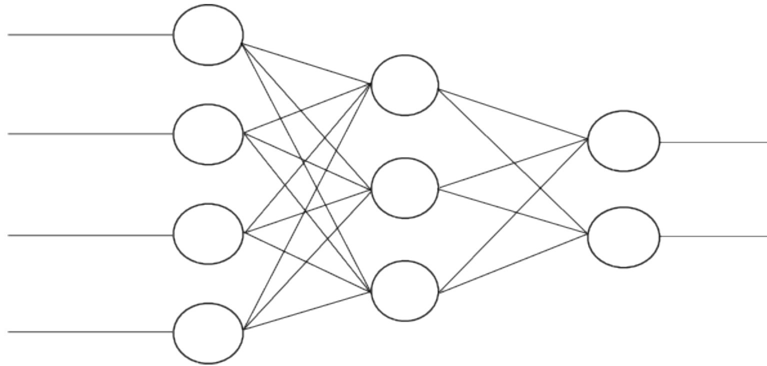

Figure 3.3: A Fully Connected neural network with 4 neurons in input layer, 3 neurons in hidden layer and 2 neurons in output layer.

The Learning algorithm , which we will discuss later, will change the strength (weights) of the connections in the network to produce a desired signal flow.

3.1.4 The Backpropagation Algorithm

The backpropagation algorithm is the most widely used method for training multilayer feed-forward artificial neural networks. The term is an abbreviation for "backwards propagation of errors" and means the errors are propagated backwards from the output nodes to the inner nodes.

The activation function

The algorithm looks for weights that minimize the error using the gradient descent method which requires that the activation function to be continuous and differentiable. One of the more popular activation functions for backpropagation networks is the *sigmoid* function, sig_c, defined by the expression:

$$sig_c(x) = \frac{1}{1 + e^{-cx}} \tag{3.1}$$

The parameter c determines the slope of the function. Higher values of c bring the shape of the sigmoid closer to that of the step function. Another alternative to the sigmoid is the *symmetrical sigmoid*, S, defined as:

$$S(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{3.2}$$

This is the hyperbolic tangent for the argument $x/2$. For both functions, the output varies continuously but not linearly as the input changes, which allows neural networks to represent non linear relationships between inputs and outputs.

The Algorithm

Given a feedforward Network and training data set $\{x^p, t^p | p = 1, 2, \ldots, P\}$,(input, target). Not that the training data can be incrementally generated using reinforcement learning algorithms (See QCON learning algorithm 4) through Interaction with the environment.

The Algorithm loop can be subdivided into tree phases:

1. Forward Pass: Compute **forward** the output of a given input x^p
2. Backward Pass: Compare the resulting output $O^p$with the desired output t^p . And propagate the error **backward**.
3. **Update the weights** for all neurons using the errors and gradient descent method.

3.1.5 Summary

The approximation of the biological computation system is on one hand very fascinating and promising for solving very complex problems. On the other hand it is very difficult to do. Based on the biological paradigms the artificial neural networks are complex nonlinear functions with many parameters: number of units, number of layers, type of connections, kind of activation function and Learning algorithm. We focused on the back-propagation algorithm which uses the gradient descent method looking for the minimum of the error function in weight space. Multilayer feed-forward neural network can represent any function given the right number of units. For our problem we use the neural networks as action-values approximator. To minimize the number of parameters, we consider only a fully connected network with one hidden layer. We make several empirical tests to find the best number of neurons in the hidden Layer.

.

Algorithm 1 The Backpropagation Algorithm for learning in multlayer networks. (taken from [15])

Require: Feed-forward network with the weight matrix w_{ji} with Q Layers and activation function f.
Training set $\{(x^p, t^p)|p = 1, 2, \ldots, P\}$,(input, target)

1: Initialize all weights w_{ji} with a small random number in $[-\lambda, \lambda]$
2: **for** each input output tuple x^p, t^p **do**
3: **repeat**
4: Compute forward the output $O_j^q$of a given input x^p for every unit j in the layer q :

$$O_j^q = f(\sum_i O_i^{q-1} w_{ji}^q))$$

5: Calculate the delta values:

$$\delta_j^Q = (O_j^Q - t_j^p) f'(H_j^Q)$$

where $H_j^q = \sum_i w_{(q-1)i} x_i^{q-1}$: the weighted summation of the inputs of the jth unit in qth layer.
6: Compute the deltas for the previous layers:

$$\delta_j^{q-1} = f'(H_j^{q-1}) \sum_i \delta_j^q w_{ij}^q$$

for every j in every layer $q = Q, Q-1, \ldots, 2$.
7: Update all weights w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}^q$$

for every layer q. with:

$$\Delta w_{ji}^q = \eta \delta_i^q O_j^{q-1}$$

η is the learning rate.
8: **until** Some stopiong criterion is satisfied.(Like the Error is small enough).
9: **end for**
10: **return** Neural network with the new weights

3.2 Reinforcement Learning

The term Reinforcement Learning (RL) denotes both a set of problems and a set of solutions. In this section the problem in terms of optimality, by using the Bellmann equation for a Markov decision process will be discussed. Two solutions will be introduced. First the dynamic programming algorithm value iteration. The second solution is the Q-Learning algorithm, which is used in our architecture to solve the learning problem in the given labyrinth.

3.2.1 Introduction

Reinforcement learning is a kind of machine learning where the agent learns "what to do" through its interaction with the environment. Rather than supervised learning where examples are provided from an external teacher, in reinforcement learning tasks the agent learns from trial and error, and receives only an indication of an actions goodness in the form of a simple scalar **reinforcement**[2] signal. This can be given in every learning step or, more commonly, only after a set of successive steps. This is known as delayed reward and leads to the credit assignment problem. An single time-step can generally been described as following (Fig.3.4) : the learning agent is in a state s and according to its experiences, follows its current **policy**, choosing an appropriate action a. It then perceives a new state s' and recieves a scalar reward r. The goal is to find or develop a policy that maximizes the agents performance which is proportional to the cumulative reward it receives.

The study of reinforcing events dates back to the end of 19.th century precisely in the psychology of Thorndike [25].His *law of effect* describes the effect of reinforcement in the learning process by animals, who tends to reselect actions proportional to the goodness of their past outcome. Animals also make an association between stimulus and response. This law is considered a principle of psychology, and is supported by more recent works like J. Herrnstein [8], which formulate the basis of the matching law describing the relationship between the relative rates of response and the relative rates of reinforcement based on experiments with pigeons.

[2] Also known as reward or payoff . in the rest of this thesis we use the term reward.

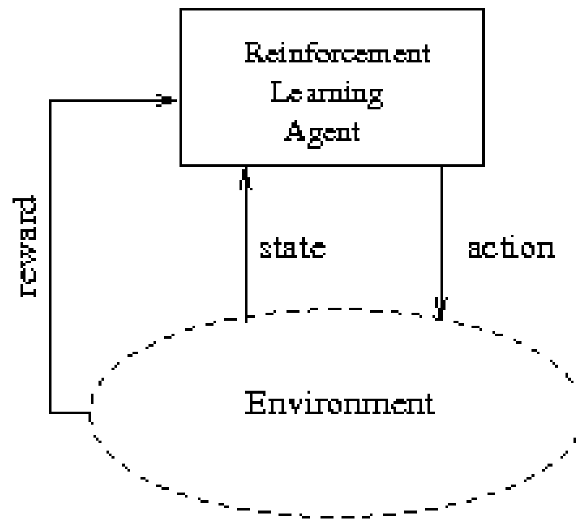

Figure 3.4: An illustration of a Reinforcement Learning framework. The Agent interacts with the environment by receiving state and reward information and performing an action.

In the field of Artificial Intelligent Arthur Samuel [18] may have been the first to implement a reinforcement learning algorithm, with his solution of the checkers-problem. But Minsky [13] was certainly the first to include psychological reinforcement learning into artificial learning.

The other important background discipline for reinforcement Learning is optimal control giving a solid mathematical and a formal description of the problem and its solutions. Optimal control theory deals with the problem of finding a controller to minimize the costs for a given dynamic system. Some very important works in this area are the results from Richard Bellman [3]. The next sections deal with the equations bearing his name, and formulate the reinforcement learning problem and its solutions particularly in the form of dynamic programming and the value iteration algorithm. First the discrete stochastic version of the optimal control problem known as the Markovian Decision Process (MDP) will be introduced.

3.2.2 Markov Decision Processes

We assume that the learning agent lives in an environment E, and observes states which are included in the finite set S, and performs actions that belong to the finite set A.

Elements of Reinforcement Learning

We assume that the learning agent interact with the environment E and can observe a set of states S and reward values R, and that its actions are included in a set A. We define in this section some entities that are found in RL problems. These definitions will simplify the formal description below.

Policy: A policy π is the mapping from the current state $s \in S$ to an action $a \in A$, defining the behavior of the agent. It is often stochastic [3], to permit the agent to deal with the exploration-exploitation dilemma. In this case the policy function maps from $S \times A$ to a value in the interval [0,1]:

deterministic policy :

$$\pi : S \to A$$

or stochastic policy:

$$\pi : S \times A \to [0, 1]$$

Reward Function: Determines the goal in a reinforcement learning task, and maps from a given state-action pair $(s, a) \in S \times A$ to a scalar value $r \in \mathbb{R}$:

$$R : S \times A \to \mathbb{R}$$

It can be stochastic too:

$$R : S \times A \times S \to \mathbb{R}$$

Value Function: Defines how good a state or a state-action pair is according to past experiences. Different from reward function where the rewards are immediate for a given state-action pair, the value function deals with the delayed reward problem.

[3] A stochastic function varies in time for instance. Its future values can not be precisely predicted, only with a certain amount of probability

The value function in the deterministic case:

$$V : S \to \mathbb{R}$$

The value function in the stochastic case:

$$Q : S \times A \to \mathbb{R}$$

Transition Function: Given a state $s \in S$ and an action $a \in A$ the transition function returns the next state $s' \in S$. In real world problems the transition function is generally stochastic. In this case it also take a possible next state $s' \in S$ as an argument and return the probability that action a in state s will lead to state s'.

deterministic transition function:

$$P : S \times A \to S$$

stochastic transition function:

$$P : S \times A \times S \to [0, 1]$$

Environment Dynamics: Assuming that the sets A and S are finite, in a RL problem, the dynamic of the environment are defined by specifying the complete Probability distribution:

$$Pr\{s_{t+1} = s', r_{t+1} = r' | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} \tag{3.3}$$

for all possible next states s' and rewards r', and all possible values of the current and past events: $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markov decision Process

In terms of a discrete-time stochastic dynamic system, if the probability distribution of future states and rewards depends only upon the present state and present reward and not on any past state or reward, then the environment processes the Markov property. In this case the dynamics of the environment are fully specified by:

$$Pr\{s_{t+1} = s', r_{t+1} = r' | s_t, a_t\} \tag{3.4}$$

for all possible states s' and rewards r', and the current event: s_t, a_t.

Formally, the Markov property is exists if and only if (3.3) is equal to (3.4) for all possible next states s' and rewards r', and all possible values of the current and past events: $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

In this case the dynamic of the environment is completely specified by the transition probability P and the reward function R.

A Markov decision process is a reinforcement learning task that satisfies the Markov property and is composed from the 4 tuple (S, A, P, R) which represents the State set, the action set, the transition probability, and the reward function, respectively.

If the state and action sets are finite then the process is called finite Markov decision process. A deterministic Markov decision process is an MDP with deterministic state transition probabilities.

Example

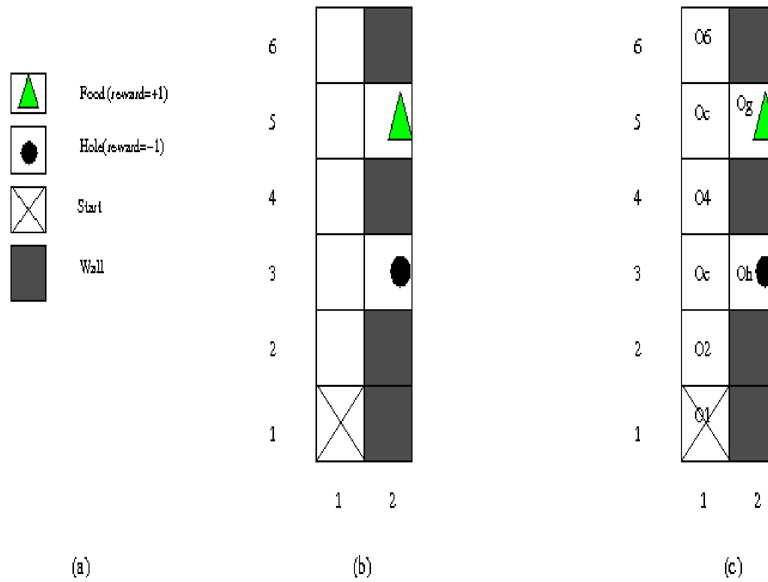

Figure 3.5: (b) A simple 2×3 maze world. (a) Illustration of the different cell types in the maze. (c) An example of an partially observable MDP where the agent don't make the difference between the two cells $c_{1,3}$ and $c_{1,5}$ and has for both the internal observation O_c

In practice, If we want to design a reinforcement learning task as an MDP for an agent in a given environment, then we have to give the agent the ability at each time step to perceive all the knowledge that it needs to decide what action is best to take next. In other words, the learning agent's state signal summarizes past sensations compactly. To make the problem more clear we consider the following example:

Suppose that a robot is situated in the small maze shown in Figure 3.5 a and b. Begining in the start state, it must choose an action at each time step. There are two possibles terminal states: The hole (which it should avoid) at cell (2,3), and the food (which is the goal) at cell (2,4). To focus on the Markov property, thereby simplifying the problem, we assume that the transition function and reward function are deterministic. The two terminal states have, respectively, the rewards -1 for the hole, and +1 for the goal. In all other states the agent receives a reward of value zero. The available actions are called Up, Down, Left, and Right. These take the agent to the neighboring cells immediately above, below, left, or right of the current cell. If the agent bumps into a wall or into the boundary, it stays in the same cell.

In summary,

Non-Markov case: We suppose that the agent has sensors that allow him to perceive if a neighbouring cell is from type wall or type start, but they do not permit him to make the difference between an empty, goal or hole neighbouring cell's type. Four light based sensors (like infrared) that are installed on the robot (one for each direction) can for example fulfil such a task. The hole is very difficult to be detected before that the robot land on the correspondent cell and the goal is very small and is also always out of measurement range of the agents sensors if it is not in the goal cell.

Unfortunately there are, amongst others, two critical cells $c_{(1,3)}$ and $c_{(1,5)}$ that represent the same observation o_c from the robot. These are critic because, of the robot's point of view, with same action "Right" the robot can lands direct to the hole or to the goal cells corresponding respectively to the internal state o_h and o_g. In the point of view from the robot the cells $c_{(1,1)}$,$c_{(1,2)}$, $c_{(1,4)}$ present respectively the pairwise different internal observations o_1, o_2 and o_4 Figure (Fig.3.5(c)).

$$Reward(c_{(i,j)}) = \begin{cases} 1 & \text{if } i = 2 \text{ and } j = 5 \\ -1 & \text{if } i = 2 \text{ and } j = 3 \\ 0 & \text{otherwise} \end{cases}$$

$$P(c_{(i,j)}, UP) = \begin{cases} c_{(i,j+1)} & \text{if } c_{(i,j+1)} \neq \text{WALL and } j < 6 \\ c_{(i,j)} & \text{otherwise} \end{cases}$$

$$P(c_{(i,j)}, Down) = \begin{cases} c_{(i,j-1)} & \text{if } c_{(i,j-1)} \neq \text{WALL and } j > 0 \\ c_{(i,j)} & \text{otherwise} \end{cases}$$

$$P(c_{(i,j)}, Left) = \begin{cases} c_{(i-1,j)} & \text{if } c_{(i-1,j)} \neq \text{WALL and } i > 0 \\ c_{(i,j)} & \text{otherwise} \end{cases}$$

$$P(c_{(i,j)}, Right) = \begin{cases} c_{(i+1,j)} & \text{if } c_{(i+1,j)} \neq \text{WALL and } i < 2 \\ c_{(i,j)} & \text{otherwise} \end{cases}$$

Figure 3.6: The dynamic of the environment for the example illustrated in the figure (Fig.3.5(a) and (b))

We have,

$$Pr\{s_{t+1} = o_h, r_{t+1} = -1 | o_c, Right, o_2, UP, s_1, UP\} = 1 \tag{3.5}$$

and

$$Pr\{s_{t+1} = o_g, r_{t+1} = +1 | o_c, Right, o_4, UP, o_c, UP, o_2, UP, o_1, UP\} = 1 \tag{3.6}$$

also

$$Pr\{s_{t+1} = o_g, r_{t+1} = +1 | o_c, Right\} = 0.5 \tag{3.7}$$

and

$$Pr\{s_{t+1} = o_h, r_{t+1} = -1 | o_c, Right\} = 0.5 \tag{3.8}$$

We have (3.5) $\neq$ (3.8), also The markov property is not verified.

Markov case: On way to make the problem verify the markov property is to give the robot the ability to distinguish between every 2 cells in the maze. This can be realized for example by investing in sensors with larger measurement range and adding sensors for the detection of the type of the lateral cells. In such case it is simple to verify that every cell in the maze, which is different from type WALL, corresponds to one, and only, one internal robot's state. If we use the same notation as in the not markov case. The cells $c_{(1,3)}$ and $c_{(1,5)}$ correspond now to different internal states s_3 and s_5 respectively further more we have now for every next state $i \in \{1, \ldots, 5\}$ and next reward $r' \in \{0, -1, 1\}$ and all possible value of the current and past events $o_t, a_t, o_{t-1}, a_{t-1}, \ldots, r_1, so_0, o_0$:

$$Pr\{s_{t+1} = s_i, r_{t+1} = r_i | s_c, Right, s_2, UP, s_1, UP\} = 1$$

$$Pr\{s_{t+1} = s_i, r_{t+1} = r_i | s_t, a_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} = 1$$

and

$$Pr\{s_{t+1} = s_i, r_{t+1} = r' | s_t, a_t\} = 1$$

Also the markov property is verified in this case.

The Markov property is important in reinforcement learning because decisions are based on values which assumed to be a function only of the current state.

3.2.3 Value Function

The value function, as described further, presents the quality of a given state, or a given state-action pair and therefore allow the learning agent to estimate the goodness of a given state, or to decide which action is eventually better to perform in a given state. Almost all reinforcement learning algorithms are based on optimizing this value. In a long run the agent goal is to maximize the received reward. An entity that summarize the received rewards is called the Return, there are many variations of it, but a general and common one is the discounted Return :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.9}$$

where the discount factor $\gamma \in [0, 1]$ determines the worthiness of the expected rewards in the future. When γ is close to 0 , the agent try to maximize only the immediate rewards and there in the future are insignificant. As γ approaches 1, the future rewards are taken into account more strongly. If rewards are bounded by r_{max} and $\gamma < 1$ then the return is also bounded:

$$\begin{aligned}
\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} &\leq \sum_{k=0}^{\infty} \gamma^k r_{max} \\
&\leq r_{max} \sum_{k=0}^{\infty} \gamma^k \quad \text{(sum of infinite geometric series)} \\
\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} &\leq \frac{r_{max}}{1-\gamma} \qquad (3.10)
\end{aligned}$$

The value of a state s depend on the policy executed form the learning agent. the value of a state is defined in respect to a specific policy π and a given state and will be denoted with $V^{\pi}(s)$.

$V^{\pi}(s)$ is the expected return when starting in s and following π :

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{3.11}$$

Where t is any time step.

Similarly, the value the pair action a and state s following a policy π is equal to the expected return starting from s, taking the action a , and following π :

$$Q^{\pi}(s, a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \tag{3.12}$$

Q^π is called the action value.

The both values verify a recursive relationship between a given state s and the possible future states s' futur and can be developed as follow:

$$\begin{aligned}
V^\pi(s) &= E_\pi\left\{R_t | s_t = s\right\} \\
&= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right\} \\
&= \sum_a \pi(s,a) \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right\}\right] \\
&= \sum_a \pi(s,a) \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V^\pi(s')\right] \qquad (3.13)
\end{aligned}$$

Likewise the recursive relationship for the action-value function is:

$$Q^\pi(s,a) = \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V^\pi(s')\right] \qquad (3.14)$$

with a deterministic transition probability the two recursive expressions look simpler:

$$V^\pi(s) = \sum_a \pi(s,a) \left[R(s) + \gamma V^\pi(s')\right] \qquad (3.15)$$

and

$$Q^\pi(s,a) = R(s,a) + \gamma V^\pi(s') \qquad (3.16)$$

where s' the next state following the deterministic transition function P : $P(s,a) = s'$. The both relationships (3.13) and (3.14) are known respectively as the Bellman Equation for state-value and action-value.

3.2.4 Optimality

As mentioned further the purpose of the most learn algorithms is to find the optimal value function, whereon the optimal policies are based. The value functions define a partial ordering over policies. If we the denote Π the set of all possible policy, then : $\pi \geq \pi'$ if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in S$ and π and $\pi' \in \Pi$ also an optimal policy verify:

$$\pi^*(s) \geq \pi(s) \quad \forall s \in S, \forall \pi \in \Pi \qquad (3.17)$$

An optimal Policy π^* is based on the Optimal state-value function:

$$\pi^*(s) = \arg\max_{a \in A(s)} V^*(s) \tag{3.18}$$

A(s) is the set of all possible actions in the state s. the Optimal state-value function $V^* = V^{\pi^*}$, is given by:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in S. \tag{3.19}$$

Similarly the Optimal action-value function is given by:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall (s,a) \in S \times A. \tag{3.20}$$

Using the Bellman Equations for state-value (3.13) and action-value (3.14) and (3.19) and (3.20) we get the Bellman optimality equations for state and action values:

$$V^*(s) = \max_{a} \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]. \tag{3.21}$$

$$Q^*(s,a) = \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]. \tag{3.22}$$

If $|S| = n$, then (3.21) presents a system of n equations with n unknowns. Due to the max operator The equations are non-linear. And therefore they can not be solved with linear algebra techniques. In the next section we discuss an iterative approach to solve this system of equations.

3.2.5 Value iteration algorithm

The Value iteration algorithm solve the Bellman optimal state value function (3.21) using an iterative approach. Starting with an arbitrary initial values for all states, the value-state will be updated using the iteration step at ith iteration:

$$V_{i+1}(s) = \max_{a} \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right] \quad \forall s \in S \tag{3.23}$$

If (3.23) infinitely often used. the final value must be the solution of the bellman optimality equation (3.21)

Algorithm 2 The Value-Iteration Algorithm

Require:

1: mdp, an MDP(S, A, P,R)

2: $\gamma \in [0, 1]$,discount factor

3: ϵ, a small positive number (the maximum error allowed in the state-value)

Ensure: An approximation of the optimal Value-function

4: **repeat**

5: $\delta \leftarrow 0$

6: Initialize V arbitrarily for all the states $s \in S$

7: **for** each $s \in S$ **do**

8: $v \leftarrow V(s)$

9: $V(s) = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V(s')]$

10: $\delta \leftarrow \max(\delta, |V(s) - v|)$

11: **end for**

12: **until** $\delta < \epsilon \frac{(1-\gamma)}{\gamma}$

13: **return** V

3.2.6 Q-learning

Q-Learning was proposed by Watkins [4] for solving Markovian decision problems, and has the properties of dynamic programming using the optimal Bellman equation to update the action-value function with the advantage of no requirement of the environment dynamic in form of the rewards R and the transition P probabilities. Therefore it is more appropriate for on-line applications. The One-step Q-learning, is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a' \in A(s)} Q(s', a') - Q(s, a) \right] \tag{3.24}$$

Where α denote the learning rate and γ the discount factor.
This update equation is calculated whenever action a is executed in state s and leading to state s' and receiving the immediate reward r.

The Q-learning converge to the optimal action-value Q* with probability one and consequently yields the optimal policy, under the following assumptions [27]:

1. All state-action pairs appear in the update an infinite number of times

2. The learning rate α is decreased with a suitable schedule

3. the Q-values are presented in a lookup-table.

The algotithm of Q-Learning for a look-up table as presentations of the actions values is given in:

Algorithm 3 The Q-learning algorithm

Require:

1: Q, a table of action values

2: $\gamma \in [0, 1]$, the discount factor

3: α, the learning rate

Ensure: An approximation of the optimal Action-value Q^*

4: Initialize Q(s,a) arbitrarily for all the state-action pair (s,a)

5: **repeat**

6: Initialize s

7: Choose a using policy derived from Q

8: Take the action a, observe the reward r and the new state s'

9: $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a' \in A(s)} Q(s', a') - Q(s,a) \right.$

10: $s \leftarrow s'$

11: **until** s is terminal

3.2.7 Exploration versus Exploitation

In online reinforcement learning methods , like Q-learning, the agent has always to deal with the exploration- exploitation dilemma. The environment exploration gives more chance to yield the best solution in term of the optimal policy (see condition 1 in the convergence conditions in Q-learning). Exploration means also trying something new what is mostly accompanied with more cost. The question is how to minimize the cost of learning by exploiting the knowledge already acquired. Or better how can the agent balance between exploration and exploitation. One solution is to select the best action stochastically. The Boltzmann distribution is one of the most function used to determine the probability to execute an action based on the learned policy. Let A be the set of all actions that can executed from the learning agent. The probability of executing an action $a \in A$ in a state s is determined by the following equation.

$$P(a) = \frac{e^{Q(s,a)/T}}{\sum_{a' \in A} e^{Q(s,a')/T}}$$

Where T is a positive parameter called temperature. High temperature cause more exploration because the actions probability tends to be similar. Low Temperature cause a greater difference in selection probability for actions that differ in their value-action in this case the greedy action is very probable to be chosen which means the agent tends to exploit its learned policy.

3.2.8 Summary

In this section a brief overview over the theory of reinforcement learning is given. The importance of Markov decision property in reinforcement learning task is illustrated using a simple example for both completely and partially observable environments.

3.3 Continuous Q-learning

There are many reasons that make generalization in reinforcement learning important. The memory requirements for a look-up table representation in tasks with many possible states and actions are too high and cannot be implemented (the cruse of dimensionality).

In addition and in order to evaluate a good policy the agent has to visit all states and try all actions. This is normally quite time consuming and needs a satisfying exploration strategy. Generalization therefore can save memory and time by approximating the value function for unknown unvisited states and untried actions. In principle any function approximation method, which are extensively studied in supervised learning, can be combined with RL to generalize and estimate over state-space.

3.3.1 Existing Approaches

There are many approaches for combining the Q-learning framework with generalization methods to solve problems in continuous state spaces. The following sections briefly describe some of them.

CMAC Q-Learning

Saito and Fukuda [17] proposed a continuous state Q-learning architecture using Albus's CMAC(Cerebellar Model Articulation controller) [2]. the CMAC is a function approximation system that features spatial locality. It is a compromise between a look up table and a weight-based approximator. it computes a function $f(x_1 \ldots x_n)$, where n is the input space dimension. The input space is subdivided into hyper-rectangles, each of which corresponds to a memory cell . The contents of the memory cells are the weights, which are adjusted during training. The output of a CMAC is the algebraic sum of the weights in all the memory cells activated by the input point. In the CMAC based Q-Learning the inputs to the CMAC are the state and action and the output is the expected Q-value.

Santamaria et al. [19] investigated the CMAC-Qlearning method on optimal control tasks. The tasks included reward penalties for energy use and coarse

motions The generalization and resolution of the CMAC depends on the number of cells in the CMAC. This may detoriorate the performance of the CMAC in the situations where some cells are never used.

Memory-Based Q-Learning

Santamaria et al. [19] evaluated memory-based Q-Learning methods. Memory-based aproximators memorize statistically experiences. The current Q is computed from a weighted average over the nearest neighbors Q's according to the similarity metric. A new point can be added to the memory when the nearest neighbors are too far. In this way, the memory expands dynamically depending on the exploration of new regions of the state-action space.

Linearly Weighted Combination methods

Takahashi et al. [24] proposed the continously valued Q-learning approach using coarse discretisation of states and actions. Like CMAC Q-learning, the proposed method allocates approximation resources uniformly across the state and action space. It could be combined with the state pre-distortion approach (Santamaria et al., 1998) [19] to permit an more efficiently allocation of resources if a priori knowledge is available.

The procedure improved performance in different experiments. However, the method is rather task specific. Furthermore, no procedure for removing boundaries was provided, limiting the adaptability of the method if conditions change.

Hedger

The Hedger system proposed by Smart and Kaelbling [22] approximates the action-values using a fixed number of points. each point represent a state, action and expected value set. A hyper-elliptical hull is constructed around the available points. The action-value for a state-action pair is calculated by checking that the corresponding point is within the training data. If it is the case a Locally Weighted Regression (LWR) is used to estimate the value function output. Else if the point is outside of the training data a Local weighted Averaging

(LWA) is applied to return the appropriate value. Finding the highest values action required a time consuming iterative process.

The Hedger system was applied to a robot to solve a corridor following experiment. The state is composed from the robot's steering angle and position relative to the corridor, and the distance to the end of the corridor where the a Reward area is defined. The action is reduced to control the steering but not the translation velocity. The reward is provided if the robot reach the reward area. Smart and Kaelbling [22] also successfully demonstrated the algorithm's off-policy learning capabilities. The stability of the algorithm was enhanced by maintaining knowledge about which regions of the state-action space were believed to be approximated properly.

Q-self Organising Map

Sehad and Touzet [20] applied the Q-KOHON system based on Kohonen's [9] self organising map. The state, action, and expected value were the elements of the self organising map feature vector. The self organising map generalises between similar states and simular actions. Touzet suggests that it is possible to interpret the feature vectors in a self organising map-based implementation; whereas feedforward neural network-based systems are less transparent are therefor harder to analyse. The Q-KOHON system was evaluated on a Khepera mobile robot. The robot learnt to avoid obstacles in a constructed environment using infra-red sensors [26]. There is no mention of penalities for coarse motion or the inclusion of current velocity in the state vector. Actions were chosen by searching for the self organising map feature vector that most closely matched the state and the maximum representable value (one). The matching process has a potential flaw. If a state has a low expected value, rather than a feature vector that has a similar state vector. This could result in an action that is not appropriate for the current state.

Fuzzy Q-Learning

Fuzzy controllers map continuous state into memberschip of discrete classes (fuzzification), then pass the membership through logical rules producing actions, and finally combine the actions based on the strength of their member-

ship (defuzzificaion) [29]. Fuzzy logic appears to be an ideal tol for creating a continuous Q-learning system: fuzzy logic generalises discrete rules for contimuous data and Q-lewarning can selec t discrete rules. Amajor difficulty is that fuzzy controllers execute many discrete rules simultaneously, complicating credit assignment.

Glorennec's [6] system regarded every possible complete fuzzy controller as an action. Consequently, there was an enormous space of actions to be searched and generalisation between actions was limited.

3.3.2 Summary

Reinforcement learning problems involve finding a policy that will maximise rewards over time. Q-Learning is a model-free approach to solving reinforcement learning problems that is capable of learning off-policy. Off-policfy learning techniques have the potential to contiribute toward the parsimonious use of measured data, the most expensive resource in robotics.

3.4 Convergence in Reinforcement Learning

3.4.1 Convergence of Value-Iteration in Discrete RL.

The demonstration of the convergence of the value iteration is based on the **Banach fixed point theorem** (also known as the contraction mapping theorem or contraction mapping principle), which guarantees the existence and uniqueness of fixed points of certain self maps of metric spaces, and the convergence of sequences defined recursively and based on this contraction upon to this fixed point. We introduce this thereom and we show that the update using in the Value-iteration algorithm (eq:3.23) correspond to a self-maps required in this theorem. By the way we introduce some definitions and lemmas that are necessarry to make the proof easier.

Definition Let (X, d) be a complete metric space. A function $T : X \rightarrow X$ is said to be a contraction mapping on X if there is a constant q with $0 \leq q < 1$ such that $d(Tx, Ty) \leq q \cdot d(x, y)$ for all $x, y \in X$.

Definition A fixed point x of a function $f\colon X \to X$, is a point that remains constant upon application of that function, i.e.: $f(x) = x$.

Theorem 3.4.1 (Banach fixed point theorem) *Every contraction has a unique fixed point.*
Let T be a contraction mapping on (X, d) with constant q and unique fixed point $x^* \in X$. For any $x_0 \in X$, define recursively the following sequence $x_n = Tx_{n-1}$for $n = 1, 2, 3, \ldots$
This sequence converges, and its limit is x^*

Let $\mathbb{R}^S$ denote the space of functions mapping from the states set to $\mathbb{R}$, wherein the value functions are included:

$$\mathbb{R}^S = \{V | V : S \to \mathbb{R}\} \tag{3.25}$$

Note that $V \in \mathbb{R}^S$ can be considered as a vector of $|S|$ component:

$$V = (V(s_0), V(s_1), \ldots, V(s_{|S|}))$$

Let T denote the operation applied to update the state-value for every state in (3.23) and defined as follow $T : \mathbb{R}^S \to \mathbb{R}^S$, $V \mapsto TV$ with:

$$(TV)(s) \to \max_a \sum_{s'} P(s, a, s') \left[R(s, a, s') + \gamma V(s')\right] \quad \forall s \in S \tag{3.26}$$

Let $||V||$denote the max-norm of $V \in \mathbb{R}^S$, which return biggest component of V.

$$||V|| = \max_{s \in S} |V(s)| \quad \forall V \in \mathbb{R}^S \tag{3.27}$$

The distance $d_{||}$ between two elements $V_1, V_2 \in \mathbb{R}^S$ is the maximum difference between any two corresponding elements: $||V_1 - V_2||$

$$d_{||}(V_1, V_2) = ||V_1 - V2||, \quad \forall V_1, V_2 \in \mathbb{R}^S. \tag{3.28}$$

Lemma 3.4.2 *Let $\mathbb{R}^A$ denote the space of functions mapping from the Action set to $\mathbb{R}$*
Consider the max-map :$\max_a : \mathbb{R}^A \to \mathbb{R}$. Then:

$$|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - f(b)|. \quad \forall f, g \in \mathbb{R}^A$$

Proof we denote $\max_a f(a) = f(a_1)$ and $\max_a g(a) = g(a_2)$.
We suppose that $f(a_1) \geq g(a_2)$ then:

$$|\max_a f(a) - \max_a g(a)| = (f(a_1) - g(a_2)) \leq (f(a_1) - g(a_1)) \leq max_a|f(a) - f(b)|$$

In the other case $f(a_1) \leq g(a_2)$ we have the same resulte by symmetry.

Proposition 3.4.3 *T is a contraction on* $(\mathbb{R}^S, d_{||})$

Proof

$$\begin{aligned}
d_{||}(TV_1, TV_2) &= ||TV_1 - TV2|| \quad (3.28) \\
&= \max_{s\in S} |TV_1(s) - TV2(s)| \quad (3.27) \\
&\overset{lemma(3.4.2)}{\leq} \max_{s\in S} \max_{a\in A(s)} \left| \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V_1(s')\right] - \right. \\
&\quad \left. - \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V_2(s')\right] \right| \\
&\leq \max_{s\in S} \max_a \sum_{s'} P(s,a,s')\gamma|V_1(s') - V_2(s')| \\
&\leq \gamma \max_{s\in S} \max_a \sum_{s'} P(s,a,s')||V_1 - V_2|| \\
&\leq \gamma \max_{s\in S} \max_a ||V_1(s) - V_2(s)|| \\
&\leq \gamma ||V_1 - V_2|| \\
d_{||}(TV_1, TV_2) &\leq \gamma d_{||}(V_1, V_2) \quad \square.
\end{aligned}$$

That is, the update in (3.23) is a contraction by a factor of λ on $(\mathbb{R}^S, d_{||})$. Hence, and due to banach theorem (3.4.1) the Value iteration always converges to a unique solution of the Bellman equations.

The Convergence in Discrete Q-learning

The Q-learning converge to the optimal action-value Q* with probability one and consequently yields the optimal policy, under the following assumptions [27]:

1. All state-action pairs appear in the update an infinite number of times
2. The learning rate α is decreased with a suitable schedule
3. the Q-values are presented in a lookup-table.

3.4.2 Convergence in Continuous RL

Rather than discrete states case, in the continuous state case it is not possible to exactly store the value function. The value function is only approximated, which forfeils the guarentee of convergence [27]. The partial solutions proposed are only suitable for specialised study cases.

Santamaria [19] suggests practical ways of improving the convergence of Q-learning with function approximation. We present here the spacial locality approach. Spacial locality avoid the unlearning problem which occurs with neural networks is used as approximator. The unlearning problem presents the case when the input to neural network covers only a small region of the input space for long periods time and the network forgets the correct output for other regions of state space. A local spacial locality approach consists of reducing the input space to small regions and assigning an local approximator to each one. This alleviates the unlearning problem, because the state varies only over a small range of values and the approximated values, in other regions, should be preserved.

Chapter 4

The Learning Architecture

To deal with the convergence difficulty in continuous reinforcement learning, a spatial locality based solution is implemented (See Chapter 2, section convergence in continuous reinforcement learning).

Following the divide and conqueror paradigm the labyrinth was subdivided into small regions, where a QCON is assigned to each region (Figure 4.2/4.1).

First the QCONs are trained separately on their respective subareas. In order to connect two subsequent areas, the subgoal of the first area is used as a starting region for the next one. In the play phase, based on the current position of the ball, a spatial selector module selects the appropriate learned QCON to be active and sends the output of the QCON to the actuators.

This solution is inspired by "place cells" [14] found in the hippocampal brain region of rats. Place cells are found to be selectively active when the rat is situated in different locations while performing navigational tasks in a labyrinth environment [28].

In following the proposed solution is evaluated on a simple problem, with the goal to find the best parameter combination that will be used after that for solving the whole problem.

4.1 The Continuous Q-learning Framework QCON

The continuous Q-learning framework QCON was proposed by Lin [10]. The idea is to use the advantage of Q-learning which is one of the best online re-

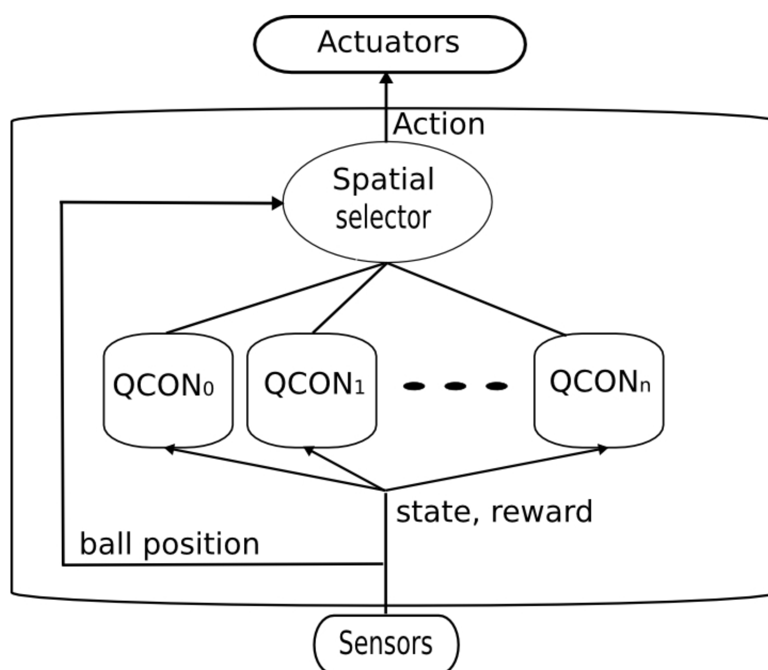

Figure 4.1: An illustration of the whole architecture. The current state and reward are inputs into the architecture. The output of the whole architecture is the output of an active QCON.

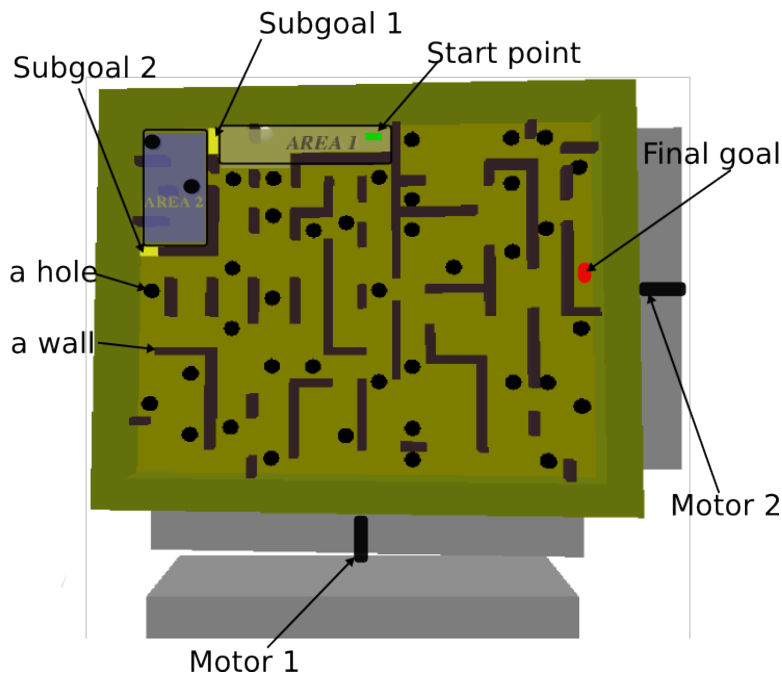

Figure 4.2: The simulation with the first two subareas labeled

inforcement learning solution. And the feed-forward artificial neural networks to generalize the Q-values between similar states. The QCON is addressed to solve problems with continuous state and discrete actions representation. The Framework has been successfully used to solve problems in nondeterministic dynamic environment and was generally better at learning than other continuous reinforcement learning frameworks like AHCON [10]. The Framework can be implemented into two possibles variations. The first one is the version with a single neural network with the state as input and with multiple outputs. The disadvantage of this version is that whenever the single utility network is modified with respect to an action, no matter whether it is desired or not (reference), the network is also modified with respect to the other actions as a result of shared hidden units between actions. The second version, which is used in this thesis, has a set of neural networks, one network for each discrete action(See Figure 4.3).

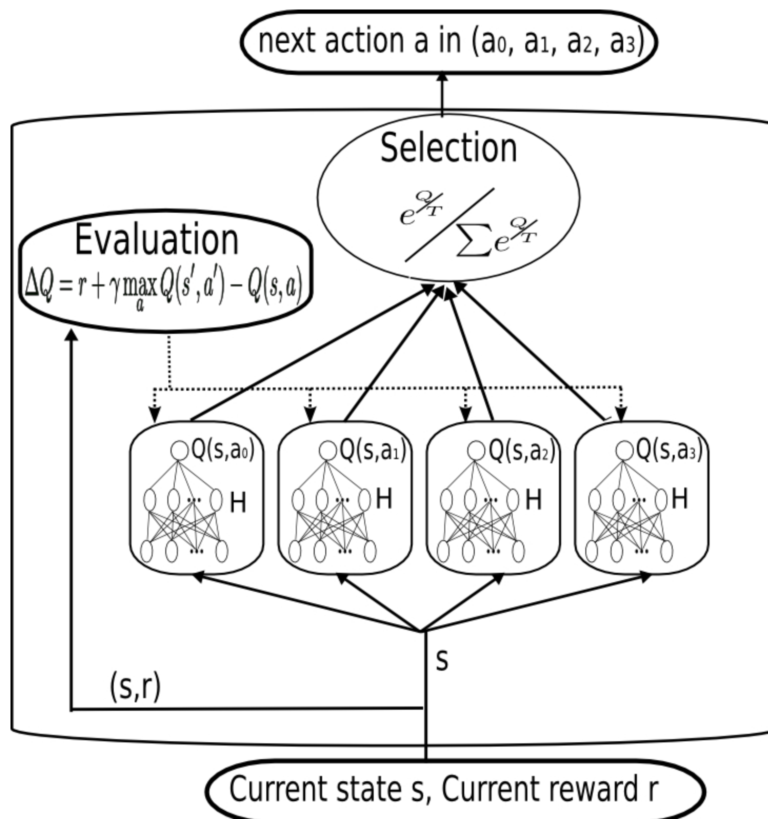

Figure 4.3: A QCON as proposed by Lin., each action-value is represented by a feedforward network with one hidden layer that is trained using back-propagation algorithm and Q-learning. We have four possible actions a_0, a_1, a_2, a_3.

Algorithm 4 A QCON learning algorithm version to solve the learn problem in the Labyrinth of Brio (Based on [10])

Require: Four feedforward networks with the dimension of s as number of inputs and a single output.

1: **repeat**
2: s ← current state ;
3: each action a_i gets the output of the neural network $ANN_i : Q(s, a_i)$
4: Select, using Boltzman distribution, an action a_k
5: Execute a ;
6: s' the new state; r *gets* reward;
7: Q' ← $r + \gamma \max_{a_j} Q(s', a_j)$
8: Adjust the network ANN_k, corresponding to the executed action by backpropagating the error ΔQ through it with input s, where

$$\Delta Q = Q' - Q(s, a_k)$$

9: **until** Some stopping criterion is satisfied

In Following a special QCON version with the parameters to control the labyrinth board is presented. The Architecture has the following components :

Input: The Input corresponds to current state s and the current reward r.

s is continuous and should presents all necessary informations to fulfill the markov property the information. It is defined as a vector of 6 elements : s=$(x, y, V_x, V_y, P_1, P_2)$ where :
(x,y) : The ball position on the board
(V_x,V_y) : The ball velocity on the board
(P_1,P_2): Motor position values

Selection: The selection between actions is stochastic allowing an active exploration for the learning agent. It follows the boltzman distribution:

$$P(a_i) = \frac{e^{Q(s,a_i)/T}}{\sum_{j=0}^{3} e^{Q(s,a_j)/T}}$$

For i=0,1,2,3. s is the current state and a_i are the possible actions. The

temperature T adjusts the randomness of action selection

Evaluation: After executing an action a_i the learning agent evaluates its goodness using the update function in Q-learning for calculating the ΔQ:

$$\Delta Q(s, a_i) = r + \gamma \max_{a' \in A(s)} Q(s', a') - Q(s, a_i)$$

The $Q(s, a_j)$ is the output of the jth neural network given the input s. The $\Delta Q(s, a_i)$ is then sended to the ith neural network to be updated. Note that only the network that is responsible of the executed action is updated, in this case the ith network.

neural networks Using the multi-networks QCON version, each neural network corresponds to one possible action. In our problem there are 4 possible actions. The Q-values are presented with feedforward networks with one hidden layer. The number of units in this hidden layer is denoted H. The Input of the neural network is the state vector and the output is the approximated Q-value for a given state and the assigned action. After an evaluation of an action-state tuple (a_i, s) the corresponding network ANN_i receives the $\Delta Q(s, a_i)$ as output error and their weights are updated using Backpropagation algorithm 1

Output The output is one of the 4 possible actions. In the defined problem the number is limited to 4 actions. For every motor there are two possible rotations: turn clockwise or turn anti-clockwise relative to the current position with an offset value.

4.2 the two holes Problem

The two holes Problem is defined to be analogous to the famous Pole Balancing Problem, which is a standard benchmark for the design of reinforcement learning solutions.

The Problem consists of an area on the board surrounded with a wall. Two Holes are situated in the extremities. And a start position for the ball is defined in the middle of the region (see figure 4.4). The goal is to avoid the holes through

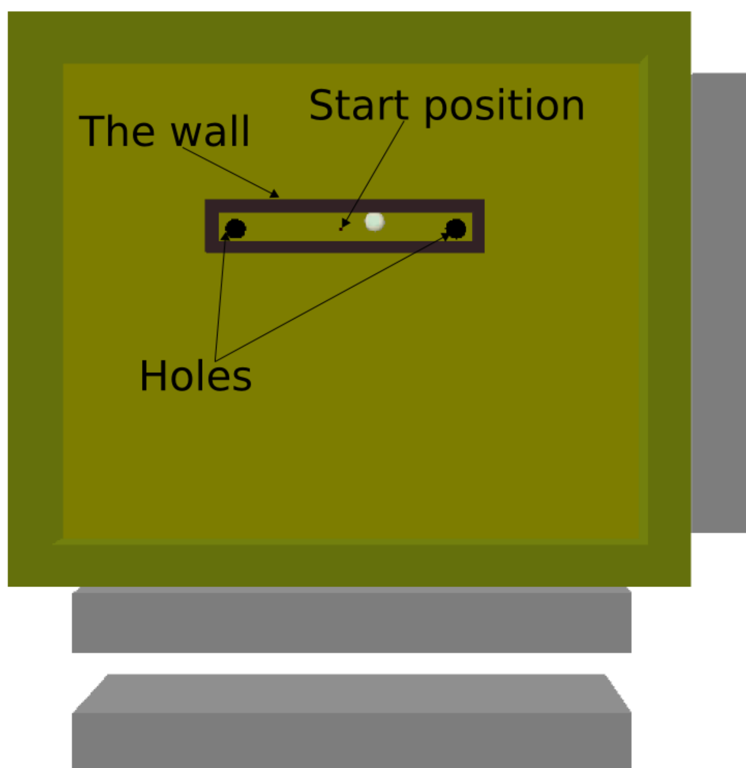

Figure 4.4: An Illustration of the two holes problem

balancing the board. Learning this task means learning implicitly the dynamics that control the ball. This problem is used to evaluate, through empirical experiments, different parameter combinations in the QCON framework.

4.2.1 Actions Representation

Starting with zero position for both motors, if the board does not move at all then the ball keeps in the centre. Of course the simplest way to solve the task may be developing a policy that chooses always the action (0,0) which means does not add any offset to the current motors positions i.e. do not turn the motors at all. To make the problem more challenging such an action is not defined. The system has a dynamical behaviour i.e. it keeps always in movement.

The four Actions defined in this study are:

- $a_0 = (-0.1, 0.1)$
- $a_1 = (-0.1, -0.1)$
- $a_2 = (0.1, 0.1)$
- $a_3 = (-0.1, -0.1)$

The values are in degree, the maximal rotation value is 3 degree and the minimal is -3 degree. As explained earlier the system has a relative action presentation which means the value sent to the motors are added to the current motor position.

4.2.2 State Representation

The state is continuous and composed from:

- The position of the ball on the board (x, y). The values are returned from the simulator as real value with the precision $10^{-4}m$, The values are multiplied with 10 and are in the intervals:

  $x \in [-0.800, 0.800]$, $y \in [-0.160, 0.160]$

- The linear velocity of the ball on the board (V_x, V_y). The values are returned from the simulator as real value with the precision $10^{-4}m/s$, The values are multiplied with 10 and are in the intervals:

  $V_x \in [-0.900, 0.900]$, $V_y \in [-0.500, 0.500]$

- The current position of the motors (P_1, P_2). The Motor positions are given in degree and are both in the interval $[-3.0, 3.0]$

4.2.3 Parameters sensitivity

The QCON framework has a number of design parameters. As mentioned before we made some restriction about the architecture of the neural networks, which is defined as fully connected and with one hidden layer. The design parameters are as follows:

1. Reward function
2. The activation function in neural networks.
3. The number of units in the hidden layer in the neural networks H.
4. The learning rate α.
5. The discount factor γ.
6. The temperature T for the stochastic action selector.
7. Motors Frequency

In order to compare objectively the performances, 20 runs were achieved for each combination the plots below show the number of trials needed to learn the task. The standard deviation is also rendered giving an overview of the stability of the results.

The Discount Facor γ

The discount factor $\gamma \in [0, 1]$ determines the worthiness of the expected rewards in the future. When γ is close to 0, the agent tries to maximize only the immediate rewards and there in the future are insignificant. As γ approaches 1, the future rewards are taken into account more strongly. In the two holes

Problem the rewards are given only at the end of trials, the value γ=0.8 is used as default value.

The Reward Function

The reward function determines the goal of the task the intuitive solution in the two holes problem is to punish the agent, i.e. give a negative reward, if it fails into a hole. The reward function in this study is given with the function R where:

- R = -1 if the ball falls in the hole.
- R= 0 otherwise.

The Activation Function

The activation function is a variation of the sigmoid function (as proposed in [10]):

$$f(x) = 1/(1 + e^{-x}) - 0.5$$

The Learning Rate α.

The learning rate α strongly influences performance on standard supervised learning problems. A learning rate that is too small leads to slow convergence with high possibility to land in local minima. High learning rates can prevent convergence because the system repeatedly steps over useful minima.

Figure 4.5 shows the performance with various learning rates, note that the learning rate is the same for all links in the neural networks.

The relationship between learning rate and performance appears to be the same as for standard supervised learning. With a small learning rate, smaller then 0.2, the agent needs more number of trials to learn the task, a high learning rate, over 0.4, can prevent the convergence and make the learning agent unstable (see the standard deviation for $\alpha = 0.5$, and $\alpha = 0.7$). A good compromise is the value $\alpha = 0.2$.

Table 4.1: Average number of trials needed to learn the 2 holes Problem over 20 runs using different values of learning rate α

Learning rate	Average number	Standard deviation
0.01	221,4	43,4
0.1	120,8	33,8
0.2	80,4	30,2
0.25	66,4	44,6
0.3	71,7	40,6
0.4	70,9	55,2
0.5	200,4	150,3
0.7	220,0	260,1

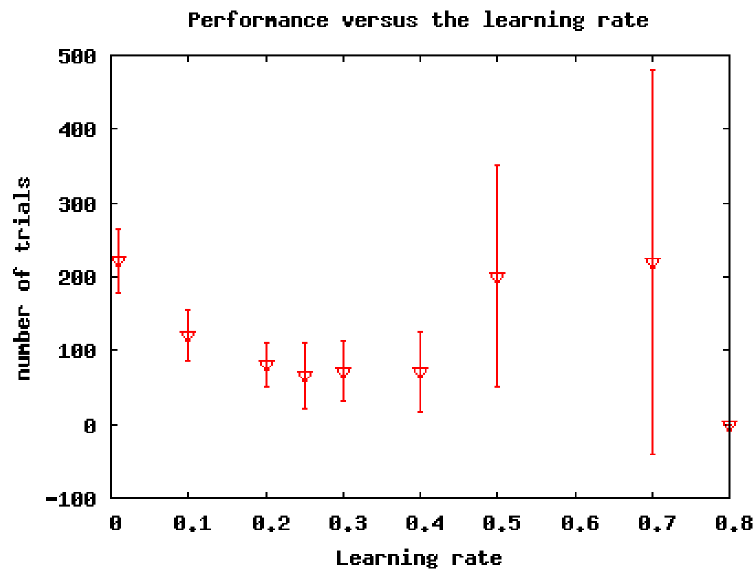

Figure 4.5: Average number of trials needed to learn the 2 holes Problem over 20 runs using different values of learning rate α

The Number of Hidden Units H

The number of hidden neurons H determines the complexity of functions that can be represented by the neural network. If H is very high then the processing needed is without reason increased and the system can be a subject of overfitting. A too small number of hidden units can prevent learning at all.

Table 4.2: Average number of trials needed to learn the 2 holes Problem over 20 runs using different numbers of hidden units H

Number of hidden neurones H	Average number of trials needed	Standard deviation
2	820	600
3	75,55	43,75
4	77,1	50,22
10	80,40	30,12
20	110,56	40,6
40	105	37.8

As can be seen in Figure 4.6 H=10 appears to be a good compromise with the smallest standard deviation and a small average number of trials needed

The Temperature T

A simulated annealing [1] approach is used. The Temperature is linearly decreased with a factor 0.09 after every trial.

4.3 Experiments

To demonstrate the stability of the learning in a given subarea, a statistical measurement of the performance of the QCON frameworks in the two first subareas is presented.

For each study we performed 10 experiments. An experiment consisted of 300 trials, and after each tenth trial the agent played with the learned greedy

[1] analogous to the annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

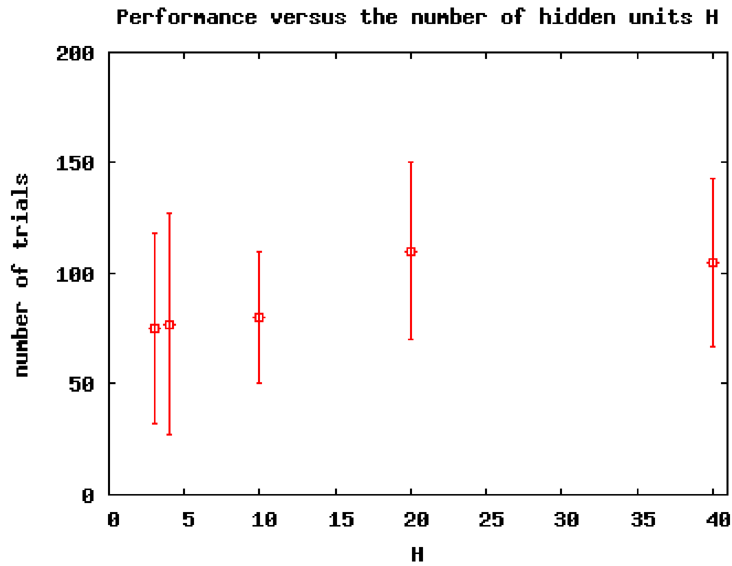

Figure 4.6: Average number of trials needed to learn the 2 holes Problem over 20 runs using different values of number of hidden units H

policy. A trial begins with a random position and terminates when the ball falls in a hole, or when it attains the subgoal, or when the number of steps is greater than 600.

We subdivided the labyrinth manually based on a predefined number of holes on a single subarea. This number is limited to two holes (see Figure 4.4).

The parameters of the experimental setup are these found in the two holes problem and are summarized in Table 4.3.

4.4 Results

In following the learning's results on the first two subareas is given. Two subtasks are solved by the agent. The first one is to avoid the holes,and the second one is to attain a subgoal in a given subarea as fast as possible.

Training's results in area 1

The plot in Figure 4.7 shows the results in the learn phase on the first area. The slope in the curve labelled "Terminal=hole" begins after 50 steps to rise expo-

Table 4.3: Parameters of the experimental setup.

Factor	Description
State	state s=$(x, y, V_x, V_y, P_1, P_2)$ (x,y) The ball position on the board (V_x,V_y) The ball velocity on the board (P_1,P_2) Motors position values
Action	For every motor there are two possible rotations: turn clockwise or turn anti-clockwise relative to the current position in steps of 0.1 deg; there are 4 possibles actions $a_0 = (-0.1, 0.1), a_1 = (-0.1, -0.1), a_2 = (0.1, 0.1), a_3(-0.1, -0.1)$
Reward	-0,5 if in hole; 1 if in subgoal; 0 otherwise
Learning	Discount factor λ=0.8; Learning rate α=0.2 Number of hidden units in a QCON net H=10
Actions selection	Stochastic: $\frac{e^{Q/T}}{\sum e^{Q/T}}$ Simulated annealing T:1 $\rightarrow$ 0.0002
Study	Average over 10 experiments in a single area; Play after after each 10 trials with greedy policy Maximum number of steps per play 600

nentially towards the maximum number of steps which is 600. The agent needs on average about 100 trials to learn the policy that permit it to avoids the holes. The second curve, which is labelled "Terminal=subgoal", shows that the agent's policy converges towards an optimal and stable solution after approximately 200 trials.

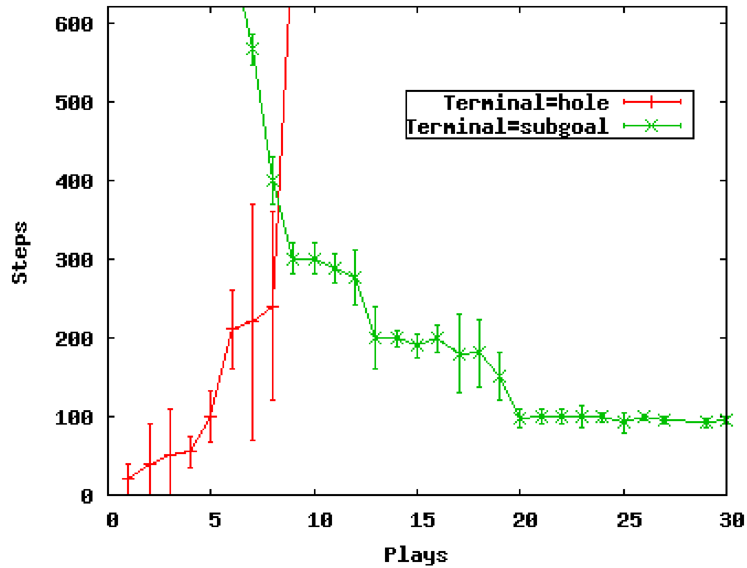

Figure 4.7: Plots of the number of steps versus plays in area 1. One step corresponds to one action, and every play was performed after 10 trials using the learned greedy policy. The red curve shows the number of steps needed before the ball falls in a hole. The green curve shows the the number of steps needed to reach the defined subgoal

Training's results in area 2

The results in the area 2 are in correlation with the results in area 1. The agent needs on average about 120 trials to learn how to avoid holes and about 250 trials to learn an optimal and stable policy.

Playing on the whole labyrinth

Based on the current position of the ball, the selector module selects the appropriate learned QCON to be active and sends the output of the QCON to the

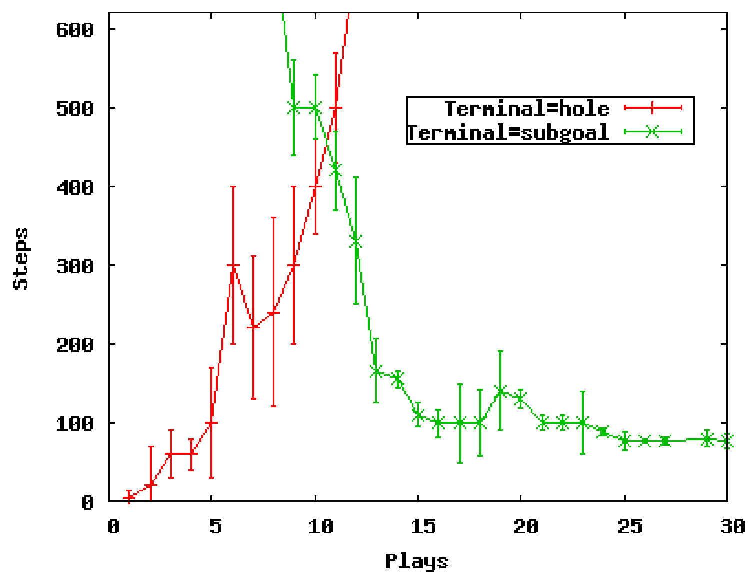

Figure 4.8: Plots of the number of steps versus plays in area 2. One step corresponds to one action, and every play was performed after 10 trials using the learned greedy policy. The red curve (Terminal =hole) shows the number of steps needed before the ball falls in a hole. The green curve (Terminal =subgoal) shows the the number of steps needed to reach the defined subgoal

actuators. Once trained, a QCON network does not need further adaptation when playing the game continually from the start point to the final goal in the whole labyrinth. This result was expected, because of the design of the subdivision and the training. Two subsequent areas, are connected using the subgoal of the first area as a starting region for the next one. After learning to reach the subgoal as fast as possible in first area, the agent lands automatically in the next area in the play phase. In the training phase of the second area the agent learns to reach the goal starting from different points in the area, furthermore due to the high exploration in the begin of each run, the agent learns to deal with a big range of velocity values, hence after that the ball lands in the second area the assigned trained QCON takes the control without any problem.

The chosen approach has the following advantages:

1. It is easier to achieve a solution with an architecture composed of a committee of QCONs than a monolithic one.

2. The solution scales up easily as the complexity of the game increases. The complexity of the architecture (the number of QCONs and the number of the hidden neurons in each QCON) is directly proportional to the complexity of the game (number of holes and walls).

Chapter 5

Conclusion

5.1 Summary

This thesis presented a connectionist architecture for learning how to play a simulated "Brio Labyrinth" game that uses the divide and conquer paradigm inspired by the way a human player plays the game. The thesis started with the theory of artificial neural networks (ANN) discussing their advantages as well as their complexity. It then proceeded to reinforcement learning, which enables autonomous and situated agents to learn and adapt to the environment. Moreover, the convergence of RL for value iteration was discussed, and it was proven that the update in value iteration algorithm can be defined as a contraction and according to Banach theorem it converges with probability 1.

For running the experiments, a physical simulation based on ODE has been realized. A QCON framework [10] based solution that combines both approaches, RL and ANN is developed and implemented for learning to play the game. In the experiments, different variable combinations were explored in a very simplified problem called the two holes problem. Since the whole labyrinth is too complex to solve with a monolithic QCON, the divide and conquer paradigm was followed and the labyrinth is subdivided into small regions, where to every region a QCON based architecture is assigned. The best parameter combination founded in the two holes problem was used to train the QCONs in the assigned subareas. The experiments on the two first subareas,

which were presented in the last chapter, illustrated the stability of the convergence in each area. It was shown that the architecture scales up easily as the number of subareas increases. The proposed solution was able to play the game successfully from the start until the the goal point.

A paper [1] on the results of this work is already accepted as poster publication at the 30th Annual German Conference on Artificial Intelligence (KI07), 2007.

5.2 Outlook

The proposed framework can be transferred to the real labyrinth. A combination between the simulation and the real labyrinth can be very practical. The simulation can be used for quantitative training, thereafter just an adaptation phase will be needed for compensating the difference between the real and simulated labyrinth. However, the real labyrinth should have the same state and action representation. The ANNs make the task easier because of their ability to deal with uncertain data coming from the real sensors. The states can be generated from potentiometers and an image-processing algorithm.

Bibliography

[1] L. Abdenebaoui, E. A. Kirchner, Y. Kassahun, and F. Kirchner. A connectionist architecture for learning to play a simulated brio labyrinth game. In *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI07)*, 2007. Accepted.

[2] J. Albus. A new approach to manipulator control: The cerebellar model ariculation controller (CMAC) (trans. of the ASME, sept. 1975). *J. Dyn. Syst. Meas. & Contr.*, pages 220–227, 1975.

[3] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

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

[5] D. Debanne, G. Daoudal, V. Sourdet, and M. Russier. Brain plasticity and ion channels. *J Physiol Paris*, 97(4-6):403–414, 2003.

[6] P. Y. Glorennec. Fuzzy Q-learning and evolutionary strategy for adaptive fuzzy control. In H. J. Zimmermann, editor, *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, volume 1, pages 35–40, Promenade 9, D-52076 Aachen, Sept. 20-23 1994. Verlag der Augustinus Buchhandlung.

[7] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Wiley, New York, 1949.

[8] R. J. Herrnstein. On the law of effect. *Journal of the experimental analysis of behavior*, 13:243–266, 1970.

[9] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, third edition, 1989. Kohonen, T.

[10] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

[11] W. S. McCulloch and W. Pitts. A logical calculus of the idea immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133, 1943.

[12] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.

[13] M. L. Minsky. Theory of neural-analog reinforcement systems and its application to the brain-model problem. Ph.d. dissertation, Princeton University, 1954.

[14] J. O'Keefe and J. Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1):171–175, November 1971.

[15] D. Patterson. *Kuenstliche neuronale Netze.* Prentice Hall, Mnchen,Germany, second edition, 1996. KNN.

[16] F. Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* Spartan Books, Washington, D.C., 1962.

[17] F. Saito and T. Fukuda. Learning architecture for real robotic systems - extension of connectionist Q-learning for continuous robot control domain. In *ICRA*, pages 27–32, 1994.

[18] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[19] J. C. Santamar'ia, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. Technical report, Feb. 01 1998.

[20] S. Sehad and C. Touzet. Self-organizing map for reinforcement learning: obstacle-avoidance with Khepera. In P. Gaussier and J. D. Nicoud, editors, *Proceedings. From Perception to Action Conference*, pages 420–3, Los

Alamitos, CA, USA, 1994. LERI-EERIE, Nimes, France, IEEE Computer Society Press.

[21] S.-H. Shi, Y. Hayashi, R. S. Petralia, S. H. Zaman, R. J. Wenthold, K. Svoboda, and R. Malinow. Rapid Spine Delivery and Redistribution of AMPA Receptors After Synaptic NMDA Receptor Activation. *Science*, 284(5421):1811–1816, 1999.

[22] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proc. 17th International Conf. on Machine Learning*, pages 903–910. Morgan Kaufmann, San Francisco, CA, 2000.

[23] R. Smith. Open dynamics engine, www.ode.org, 2005.

[24] Y. Takahashi, M. Takeda, and M. Asada. Continuous valued q-learning for vision-guided behavior acquisition. In *Proc. of 1999 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 255–260, 1999.

[25] E. L. Thorndike. The law of effect. *American Journal of Psychology*, 39:212–222, 1927.

[26] C. F. Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3–4):251–81, 1997.

[27] C. J. C. H. Watkins and P. Dayan. Technical note Q-learning. *Machine Learning*, 8:279, 1992.

[28] M. A. Wilson and B. L. McNaughton. Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058, August 1993.

[29] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.