Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Grounding of Language in Sensorimotor World Interaction of a Humanoid Robot, using Neural Networks

Bachelor thesis
im Arbeitsbereich Knowledge Technology, WTM
Prof. Dr. Stefan Wermter

Department Informatik
MIN-Fakultät
Universität Hamburg

vorgelegt von
Thomas Christian Blank
am
03.02.2012

Gutachter: Prof. Dr. rer. nat. habil. Ralf Möller
Dipl.-Inform. Stefan Heinrich

Thomas Christian Blank
Matrikelnummer: 20835682
Eißendorfer Straße 167
21073 Hamburg

Abstract

Symbol systems provide techniques to emulate processes of the human mind, such as abstract reasoning, natural language and problem solving, in artificial intelligent agents. To purposefully employ these techniques in artificial agents, however, these symbols need to be grounded, i.e. the agent must be able to connect a symbol to its real world referent and *vice versa*. Marocco et al. (2010) [11] proposed a method that uses neural networks to ground symbols in the sensorimotoric experience of a humanoid robot. After training, the networks were capable to identify three different situations and utter a corresponding proto-word. We tried to employ the proposed method on a different humanoid robot, NAO. In the process, we identified and investigated the parameters that influence the stability of the proposed method and the quality of its results.

Zusammenfassung

Mit Hilfe von Symbolsystemen können Prozesse des menschlichen Geistes, wie abstrakte Betrachtung, natürliche Sprache oder strukturierte Problemlösung, von künstlichen intelligenten Agenten nachgeahmt werden. Jedoch erfordert eine sinn volle Implementation dieser Techniken, dass die verwendeten Symbole hinreichend fundiert werden, was bedeutet, dass der Agent in der Lage sein muss, die Verbindung zwischen den abstrakten Symbolen und realen Entitäten herzustellen und nachzuvollziehen. Marocco et al. (2010) [11] haben eine Methode vorgestellt, mit der man unter Verwendung von neuronalen Netzen eine solche Fundierung von Symbolen in der sensorimotorischen Wahrnehmung eines humanoiden Roboters durchführen kann. Diese Netze waren nach dem Training in der Lage, drei Situationen voneinander zu unterscheiden und mit der Ausgabe eines entsprechenden Proto-Wortes zu reagieren. Wir haben versucht, diese Methode auf einem anderen Roboter, NAO, anzuwenden, und gleichzeitig untersucht, welche Parameter maßgeblichen Einfluss auf die Stabilität des vorgestellten Mechanismus' und die Qualiät der Resultate haben.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The symbol grounding problem describes a problem in language understanding and language synthesis of artificial intelligence.

Much of the human thinking and communication is based on the abstract concepts the human mind has of the world [10]. This abstraction makes it possible to condense and aggregate knowledge and experience, reapply it on previously not yet encountered situations and transfer that knowledge between humans with the help of language.

Symbols serve as designators for these abstract concepts, e.g. words, letters, signs or icons. It is impossible to exchange thoughts between humans directly, but symbols can have easily transferable physical manifestations, like script or sound. One can combine a set of symbols with a set of rules that state how the symbols can be arranged and manipulated to form a *symbol system*.

In natural language, ideally, those rules allow the symbols to be arranged and manipulated in the same way that the thoughts in a persons mind may be rearranged or manipulated. This makes it possible to convey new thoughts, as the listener can simply apply the manipulations performed on the symbols on her own thoughts to derive the thought the speaker intentioned to convey. For example, a person unfamiliar with zebras, but with horses and stripes, will be able to build up a fairly accurate imagination of a zebra, if someone would tell her it is a "striped horse."

But it is also possible to drop the direct connection to the real world for a short time and perform complex tasks of reasoning, like e.g. it is done in mathematics. During the calculation, the operations are performed solely on the symbols[1]. Those results can afterwards be translated back into thoughts about the real world.

Both of these capabilities, communication and complex reasoning, are desired capabilities of artificial intelligent agents. But to actually make a computer be able to exploit these mechanisms that symbol systems provide, it is necessary to *ground* at least a small initial set of symbols and give that computer some

[1]Some concepts, like imaginary numbers, do not even have a real world equivalent, but exist solely as symbols and the abstract thoughts. Yet, they produce accurate results.

sort of understanding of these words[2]. Throughout the last few paragraphs, we implicitly assumed that the agent is able to connect symbols and their real world referents, e.g. that a program can derive a formula from a situation it encounters, then solve and translate the result back; or that the program can translate an encountered situation into the sentences that describe this situation and derive an imagination from heard sentences. This translation capability is crucial to purposefully utilizing the communication and abstract reasoning capabilities that symbol systems provide in artificial intelligent agents. We therefore need means to connect the symbols to real world objects and concepts.

Marocco et al. (2010) describe a mechanism capable of solving this problem in their article [11]. Neural networks can be trained to reproduce a sequence of values, in this case target angle values for the actuators of a robot. Marocco et al. extended this network by a few additional in-/outputs. During the training of the rest of the network, these outputs were "forced" to target combinations that formed proto-words (like [0 0 1] or [0 1 0]). Afterwards, Marocco et al. found that when the networks reproduced the steering sequences, they also reproduced the corresponding words.

This bachelor thesis aims to further investigate this mechanism, especially in respect to its stability and applicability in real life towards building language understanding intelligent systems. In such real life applications, compared to simulation, numerous factors degrade the quality of the data the network receives. Also do many parameters of the network itself influence its capability to learn to reproduce the steering sequences and identify them. We will try to vary some of these parameters to find out how they influence the outcome of these experiments.

In chapter 2, we will give an introduction to the two basic concepts in this work. The first part introduces neural networks and their training. The second part gives an introduction to symbols, symbol systems, their capabilities and restrictions.

Chapter 3 gives a detailed description of the experiments of Marocco et al. and ourselves. We will clarify which exact questions we are trying to answer and how we will try to reach a conclusion. We will also identify and explain the single components that were used for these experiments and describe the steps taken to measure the impact of parameter variations on the network's performance.

In chapters 4 and 5, we present our results and discuss them.

[2]Other words can then be grounded by defining them in terms of these initial symbols. This is called grounding transfer. For more informations on grounding transfer refer to [2]

Chapter 2

Basics

This chapter is subdivided into two parts. Section 2.1 will introduce neural networks as a programming technique. It will describe how they work and how they can be trained to solve classification and value sequence prediction tasks. In the second section 2.2, we will give an introduction to symbols. We will discuss discuss the processes involved in interpreting symbols and the capabilities of symbol systems.

2.1 Neural networks

Neural networks are a programming model used in the programming of artificial intelligence. They try to remodel the working of biological neural networks as in the human brain.

2.1.1 Basic structure

Neural networks consist of *cells*, i.e. neurons, and *connections* between these neurons, i.e. nerves. A model of a neuron and its internal processes is depicted in figure 2.1.

Very much like biological neural cells generate electric impulses at a certain rate and propagate them via the nerves to other neurons, artificial neural cells generate and propagate *excitement* or *activation values*, which are modelled as simply a numerical value o_j for neuron j. These excitement values are supposed to model the rate at which electrical impulses in a biological network would be triggered. Each neuron sums up all the excitement values it receives and processes that sum net_j through an *activation function* f, generating its own output that it in turn propagates to all the following cells. For the following sections, we will assume a *semilinear* activation function, i.e. functions that do not decrease and that are fully differentiable [21].

In biological networks, a nerve's thickness and other properties determine how well it transmits electrical impulses [6]. Thus, a neuron firing at constant rate will excite another neuron that is connected to it through a thick nerve more than

one that is connected through a thin nerve. This is modelled in artificial networks by assigning the connections a *weight*, a numerical value that the transmitted excitement value is multiplied with. We write w_{ji} for the weight that transmits from neuron i to neuron j. The seemingly "reverse" indexing is for compliance with matrix element indexing conventions.

As a last property, the neurons in artificial networks have a numerical value that indicates how likely or unlikely a neuron is to generate high activation values independent of its input, the *bias* b_j.

This leads to the following equations:

$$net_j = \sum_j (w_{ij} \cdot o_i) + b_j \tag{2.1}$$

$$o_j = f(net_j) \tag{2.2}$$

Typical examples for an activation function are the step function[1] or the logistic function [9] with some *slope parameter* a that defines its steepness:

$$\phi(v) = \frac{1}{1 + e^{-av}}$$

While these functions range from 0 to 1, some use cases require functions that range from -1 to +1. For such cases, the signum function or the hyperbolic tangent function are used [9].

Please note that we can consider the bias a weight that is connected to a neuron which always has the value 1 as output.

2.1.2 Network architecture

A common pattern is to organize the neurons in layers. Each neuron of one layer gets connections from all neurons on the previous layer, and connects to all the neurons on the following layer[2]. A useful network consists of at least two layers. An example architecture is shown in figure 2.2.

The first layer, the *input layer*, is where the network gets its input values. Its size depends on the size of the input data vectors. The individual data items get injected as the output of these neurons so that effectively no computation is done on this layer. The neurons only serve as place holders for the input values. This is why networks with two layers of neurons are referred to as *single layer networks*. We shall denote the set of neurons that constitute this layer I.

The last layer is called the *output layer*. Its size depends on the size of the desired output data vectors. The set of neurons in this layer is O.

It is possible to introduce an arbitrary number of arbitrarily sized layers between the input and the output layer, which are called *hidden layers*, as they "can't

[1] The step and the signum function are not differentiable, so the training techniques described in this chapter can not be applied. Other techniques that are able to train neurons with activation functions that are not differentiable are presented in [9].

[2] We here make a difference between "connected to" and "receiving a connection from" to clarify the flow of information.

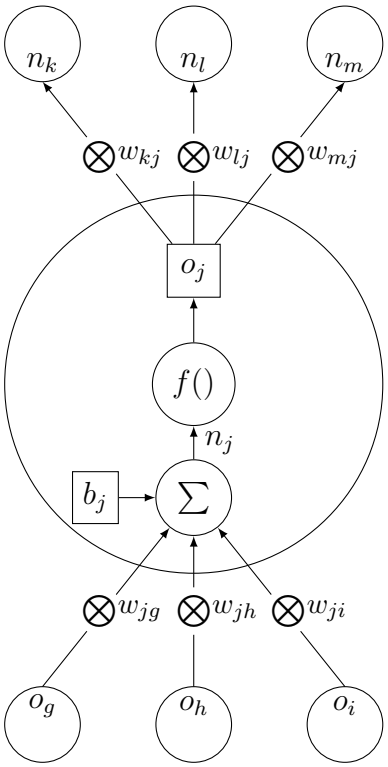


Figure 2.1: A neuron performing a forward pass. The output values of all previous neurons, weighted with the respective connection weights, are summed up with the bias to generate net_j (marked n_j.) That value is in turn used by the activation function f to generate the output o_j.

be seen from the outside." Neurons in these layers constitute the set we shall call H. Networks consisting of more than two (meaning more than one computationally effective) layers are consistently called *multi-layer*, or more specific, *n-layer networks*, where n stands for the number of computationally effective layers.

2.1.3 The back-propagation training algorithm

In this subsection we wish to describe how networks can be trained to approximate a function. Finding a set of weights that appropriately solves a given task is non-trivial, but an algorithm has been found that lets the weights converge towards an optimal set most of the time. This section largely follows [21]. For a more detailed derivation of the algorithm, the reader may be referred there.

Let us evaluate the error

$$E_p = \frac{1}{2} \cdot \sum_j e_{pj}^2 = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2 \tag{2.3}$$

i.e., for a given input-output pair p, the square mean of the individual neuron j's

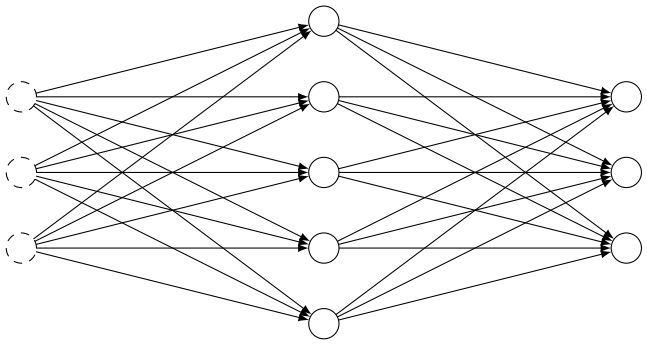


Figure 2.2: A basic neural network with each three in- and output units and five hidden units. The input units are dashed, as they do not perform computation, but are only input lines. Each connection has one individual weight connected to it.

error[3] e_{pj}, which is in turn the difference between the output value o_{pj} and the predefined desired, or *target output* value for this neuron and pair, t_{pj}. Especially let us investigate how the networks' weights influence that error.

In mathematical terms, the network computes a function from input to output vectors. But with given input-output pairs, it can also be interpreted as a function from the weights to the overall error. The back-propagation algorithm tries to find the minimum of this function by performing a steepest descent in this error space.

Single layer Network

Using the chain rule, we find the derivative of the error with respect to changes of the weights w_{ij} that connect an output unit, indexed j, with an input unit, labelled i, to be

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial w_{ji}}. \tag{2.4}$$

Please recall from section 2.1.1 that the mentioned weights also include the bias.

The first part of the right hand term is the change in overall error in respect to changes in the output of output node j. This derivative can easily be derived from equation (2.3).

$$\frac{\partial E_p}{\partial o_{pj}} = -e_{pj} = -(t_{pj} - o_{pj}) \tag{2.5}$$

The latter two terms can be derived taking into account equations (2.1). The second part is the change of the neurons output o_{pj} in respect to changes in its net input, which is simply the derivative of the activation function, $f'(net_{pj})$. The

[3]More precisely, this term is called the *absolute error* to distinguish it from other error terms used in the backpropagation algorithm.

third is the change of the net input with respect to changes of a weight w_{ji} of the connection from input neuron i, which is that neurons output, o_{pi}.

This leaves us with:

$$\frac{\partial E_p}{\partial w_{ji}} = -(t_{pj} - o_{pj}) \cdot f'(net_{pj}) \cdot o_{pi} \tag{2.6}$$

We will now introduce δ as

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = e_{pj} \cdot f'(net_{pj}) \tag{2.7}$$

This value is individual for each neuron and pattern, and gives a measure for the desired change in net input of that neuron. Those actual changes of individual weights $\Delta_p w_{ij}$ that lead to this change in net input are then computed by:

$$\Delta_p w_{ji} = \eta \cdot \delta_{pj} \cdot o_{pi} \tag{2.8}$$

with η being a general proportionality parameter that describes how big the applied changes shall be, i.e. how rigorously the current direction of steepest descent shall be followed. It is called the *learning rate*. Bigger learning rates will let the network train faster, but also might lead to strong oscillations or overshoots. Small learning rates are less prone to these problems, but lead to poorer time performance of the training process.

Multi-layer networks

For multi-layer networks, above rule is only able to compute weight changes for those weights that connect to the output units, as for these neurons it is possible to directly give an equation to compute their individual error. Hidden neurons have no designated target output values, which makes it harder to assign an error value to them, from which then appropriate and directed weight changes could be derived. Yet there is a possibility to compute and assign a value for δ to them, from which then the weight changes can be derived.

Reconsider equation (2.4). Assume that node $j \in H$ is a hidden unit instead of an output unit. The second and third terms in that equation are the same as in the case of a single layer network. We solve the three factors in the right hand term in order.

The first term is the change in overall error in response to a change of the output of unit j. For that purpose, we observe that a change in the output of a unit in the penultimate layer affects the overall error by changing the net input of all those nodes $k \in O$ that this particular node j is connected to.

The second factor, the effect of changes in net input on the output of a neuron, has already been computed in the single layer case.

The third, which denotes the change in net input caused by altering the weight of the connection to node j from some node i, is just the output of that previous neuron i.

To solve the first term, we therefore write:

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}}$$

The net input of node k is influenced by the weighted outputs of all nodes i that k gets connections from. That index set I will also include the particular node j. As the weights between unconnected nodes can be assumed to be zero, we can let the index i iterate over all neurons.

$$\begin{aligned}\frac{\partial E_p}{\partial o_{pj}} &= \sum_k \left(\frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial}{\partial o_{pj}} \sum_i w_{ki} o_{pi} \right) \\ &= \sum_k \frac{\partial E_p}{\partial net_{pk}} w_{kj} \\ &= \sum_k \delta_{pk} w_{kj} \end{aligned} \tag{2.9}$$

Thus, we find that we can determine the delta value for a hidden node from the delta values of all those nodes that it connects to. We write the delta rule for the hidden and input units in multilayer networks:

$$\Delta_p w_{ji} = \eta \cdot \delta_{pj} \cdot o_{pi} \tag{2.10}$$

$$\delta_{pj} = f'(net_{pj}) \cdot \sum_k \delta_{pk} w_{kj} \tag{2.11}$$

Equation (2.8) can be used to compute the delta values for the output units. These values can then be used to compute the delta values for all nodes in the penultimate layer, and so forth, until the error has been propagated back through the whole network. Then, in each neuron, the appropriate weight changes can be computed and applied. These processes are very similar to the forward pass of a neuron, and depicted in figure 2.3.

Momentum

The backpropagation algorithm has shown to be prone to local minima, preventing the algorithm from converging towards a global minimum. A technique to overcome this problem is to add a *momentum* term to the delta rule. On each weight update, a fraction of the previous update is added. Therefore, if the vector of all weight updates in one presentation of samples has been "pointing" in a certain direction, this direction changes more slowly, which is sometimes enough to overcome a small area around a local minimum in which the overall error increases, but which is necessary to avoid being "trapped" in the local minimum. With n denoting the number of presentations of the sample in question and β being the *momentum term* describing to which degree this technique is applied:

$$\Delta w_{ji}(n+1) = \eta \cdot \delta_{pj} \cdot o_{pi} + \beta \cdot \Delta w_{ji}(n) \tag{2.12}$$

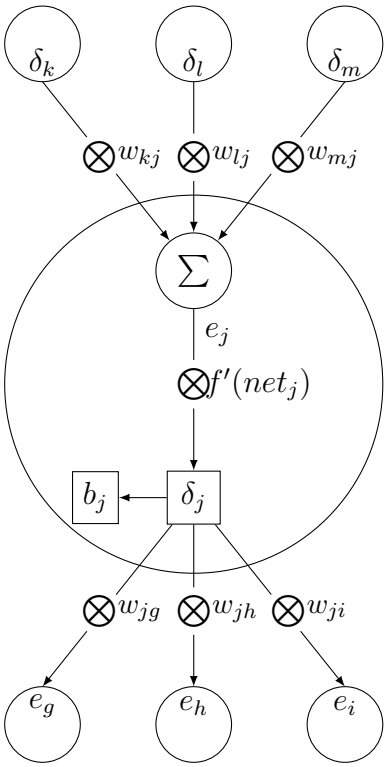


Figure 2.3: A neuron performing a backward pass of the backpropagation algorithm.

2.1.4 Recurrent neural networks

It is possible to define networks where the input and the output layer have equal strengths. This opens the possibility to feed the networks output back into the network as input of the next time step. Recall that input is fed into a network as the output that the input nodes give. The input nodes perform no computation, so that actually once the network has been fed some input, the neurons in I and in O can be merged and only the hidden layer H and the output layer O pass the values back and forth (see fig. 2.4.)

One can interpret a time sequence of target vectors $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_k$ as a *transition function*:

$$\begin{aligned} f : \mathbb{R}^m &\to \mathbb{R}^m \\ \mathbf{t}_n &\mapsto \mathbf{t}_{n+1} \end{aligned} \tag{2.13}$$

By training a network of adequate configuration[4] to approximate that function, it is possible to store these sequences in the network, which can then be recalled by giving as input the values of the first vector and then iterating over time.

[4]The size of the input/output-layer needs to be of the size of the vectors $\mathbf{t}_k$.

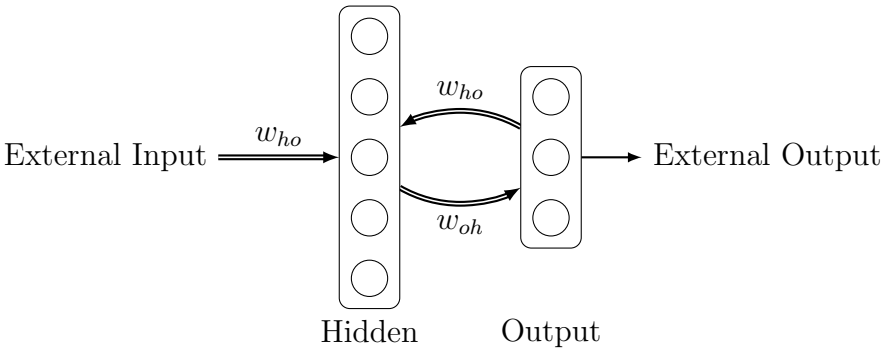


Figure 2.4: A recurrent network. Double line arrows denote a full connection between all nodes in the respective layers.

2.1.5 Backpropagation through time algorithm

A problem of storing sequences of values in recurrent networks is that during training, a possible error produced by the network in an early time step causes errors in all following time steps, so that the errors add up over time. To directly address this problem, the backpropagation algorithm has been modified independently by Werbos [26, 27] and Rumelhart et al. (1986) [21] to the *backpropagation through time algorithm*. We follow the review of the algorithm and related issues that Williams and Zipser gave in 1995 [28].

We here describe the epochwise version of the algorithm. Various other versions have been developed. Further information on these variants can be found in [27] and [9].

Assume a network $\mathcal{N}$ with n nodes of which m are in-/output units. We try to train this network to replicate a sequence s of m-sized vectors $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_k$ describing some time interval $[T_0, T_1]$.

For the algorithm, the network is *unfolded* (see fig. 2.5.), i.e. replicated as many times as there are time steps in the time interval in question. Note that there is one replication for each time *step*. A sequence consisting of k vectors requires $k-1$ replications of the network.

The time step between two points in time $\mathbf{t}_n$ and $\mathbf{t}_{n+1}$ shall be indicated by the running variable $\tau = n$. The unfolded network, denoted $\mathcal{N}^*$, is a feed-forward network and can be trained as such. It is mathematically equivalent to $\mathcal{N}$, as long as the requirement holds that the weights $w_{ij}(\tau)$ are the same for all time steps $\tau \in [T_0, T_1]$ and unless computation in $\mathcal{N}$ leaves the time interval $[T_0, T_1]$. We will call the set of output neurons of $\mathcal{N}^*$ O^* and its input unit set I^*. In the following section, let also still the set O contain the output neurons in $\mathcal{N}$ or its equivalents in $\mathcal{N}^*$.

We will now combine the two basic backpropagation training processes of training $\mathcal{N}$ with the samples $\mathbf{t}_\tau \mapsto \mathbf{t}_{\tau+1}$ and training $\mathcal{N}^*$ with the single sample $\mathbf{t}_{T_0} \mapsto \mathbf{t}_{T_1}$. The first training process teaches the network the single time steps. The second one enhances this process by assuring and correcting the long-term accurateness of the sequence reproduction during training.

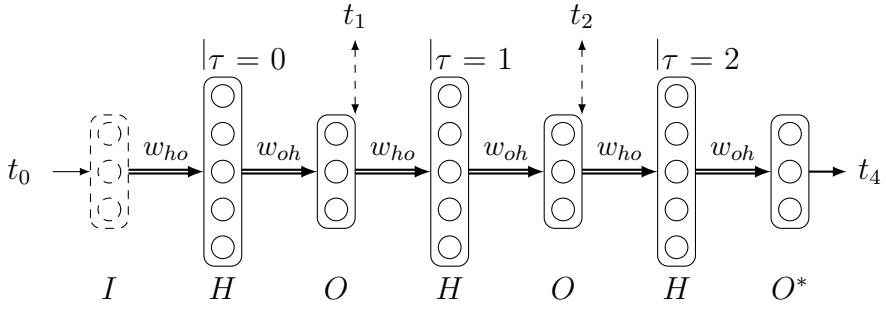


Figure 2.5: A recurrent network unfolded. This network has two layers and is replicated three times to accommodate learning a sequence consisting of four values.

The error injected in the unit[5] j at time $T_1 - 1$, ε_{j,T_1-1}, is in both cases the deviation of the output of the previous time step from the target vector $\mathbf{t}_{T_1}$. That error relates to the error that is injected in the output units in the backpropagation algorithm for feed-forward networks, e. Recall that the network is supposed to compute $\mathbf{t}(\tau + 1)$ from $\mathbf{t}(\tau)$, hence the offset in the time indexes of t and o.

$$\varepsilon_{j,T_1-1} = e_j(T_1 - 1) = t_{j,T_1} - o_j(T_1 - 1) \tag{2.14}$$

In the process of training $\mathcal{N}$ with the samples $\mathbf{t}_\tau \mapsto \mathbf{t}_{\tau+1}$, this scheme continues. The error injected is called the *external* or *absolute error*:

$$\varepsilon_{j,\tau} = e_j(\tau) = t_{j,\tau+1} - o_j(\tau) \tag{2.15}$$

In $\mathcal{N}^*$, the errors for all units except those in the last layer O^* are computed according to the delta rule in equation (2.10). This error is not computed from some predefined external target value, but rather backpropagated from all nodes k that j connects to. This gives the "illusion" of *virtual target values*, and is hence called *virtual error*:

$$\varepsilon_j(\tau) = \sum_k w_{kj}\delta_k(\tau + 1) \tag{2.16}$$

To combine these two processes, we just add the injected errors for each process, leading to the *delta rule for backpropagation through time*:

$$\Delta_p w_{ji}(\tau) = \quad \eta \cdot \delta_j(\tau) \cdot o_{pi}(\tau) \tag{2.17}$$

$$\delta_j(\tau) = \begin{cases} f'\left(net_j(\tau)\right) \cdot e_j(\tau) & \text{for } j \in O^* \\ f'\left(net_j(\tau)\right) \cdot \left(e_j(\tau) + \sum_k w_{kj}\delta_k(\tau + 1)\right) & \text{for } j \in O \setminus O^* \\ f'\left(net_j(\tau)\right) \cdot \sum_k w_{kj}\delta_k(\tau) & \text{for } j \in H \end{cases} \tag{2.18}$$

[5]We here use the name of ε to separate the injected error from the absolute error e.

With these formulas, the network can be trained just like with the "normal" backpropagation algorithm. Note that while $\mathcal{N}^*$ contains independent connection weights $w_{ji}(\tau)$ for each time step that are all replications of one single connection weight w_{ji} of the original network $\mathcal{N}$. Thus, the backpropagation through time algorithm may compute several different weight changes for one single weight. It is common practice to just compute the average of all weight changes and apply that.

The formulas above describe the process for one sequence s. It is also possible to train the networks to with several sequences out of a set of sequences S. For this, all sequences in S are used to perform one forward and one backward pass as described above. Only after the training of the last sequence, the average of all computed weight changes for all sequences and time steps is applied. The process to train all sequences $s \in S$ and once apply the average weight changes afterwards is called an *epoch*.

2.1.6 Teacher forcing

There is a further technique to address the problem of errors that accumulate over time and decrease the learning performance, called *teacher forcing*. It was developed by various researchers and described by Williams and Zipser [28]. In the backpropagation through time algorithm, those neurons in $\mathcal{N}^*$ that relate to later times will not be trained effectively before the training of the neurons relating to earlier times is almost complete. Until then, the outputs of the earlier neurons still contains high errors, so that the neurons relating to later times can not yet train to approximate their target output from their inputs, as these inputs are still subject to change.

To overcome that, we select a subset F of those neurons in the network, for which the predecessor neurons have predefined target outputs. This set of neurons now is not presented with the outputs from the previous neurons, but instead with a linear combination of those actual outputs and the target output that those neurons would have produced if the network would already represent a perfect approximation of the transition function. Thus, the error made at earlier times is partly ignored at later times, preassuming that the previous neurons will deliver an accurate approximation after training is complete.

When teacher forcing is applied to a set of neurons F, the input net_k of a node k is found using the following rule. The value e_pj stands for the error that a neuron j with a predefined target output t_{pj} for an input/target-pair p produces.

$$net_k = \begin{cases} \sum_j (o_j w_{kj} + \lambda e_{pj}) & \text{if } k \in F \\ \sum_j o_j w_{kj} & \text{if } k \in H \cup O \setminus F \end{cases} \tag{2.19}$$

If we set $\lambda = 0$, all no modifications are made, and the input of the node is taken from the outside world if k is an input unit or computed via standard forward

propagation from the unchanged outputs of the preceding nodes, otherwise. For $\lambda = 1$, the output of the neurons preceding those in F is entirely substituted by those nodes' target output t.

Consequently, in the backward propagation of the error from the nodes in F, the virtual error (Equation (2.16)) gets multiplied by $(1 - \lambda)$. The virtual error is to account for the accumulating error of previous neurons, which has been blocked out by teacher forcing. Therefore, the backpropagated virtual error also has to be blocked out.

Additionally to addressing the problem of accumulating errors, the teacher forcing on a set of neurons units makes these units, to a small amount, "expect" the preceding units to produce correct results, as what they receive from their predecessors during training gets corrected. So when they get trained to correctly predict the following time step by changing their weights accordingly, they will rely on input values that are more similar to those that they probably will receive once training of the previous neurons has succeeded.

Those weight changes do also influence the backpropagation of error values. The preceding neurons will be trained to conform to those virtual target values that they receive from their successors. These target values will, through the earlier teacher forcing, be more similar to those values towards which the teacher forcing corrected the values in the first place. Therefore, the teacher forcing is a measure to take influence on the virtual target values for those neurons preceding those in F. These values are normally not under control of the teacher.

2.1.7 Recapitulation

In our experiments, we used the backpropagation through time algorithm with a momentum term and teacher forcing applied. The exact parameters and modalities will be detailed in section 3.5.2. For easier reference, we will here collect and combine all of the above formulas to give those that were used in the training in our experiments.

The recurrent network $\mathcal{N}$ learns to reproduce a sequence of m-sized vectors $\mathbf{t}_k$, which describe the m target outputs for some time interval $T = [T_0, T_1]$. The k time points define $k-1$ time steps τ. For each time step, the network has to derive the value of the next time point from the previous value.

It was unfolded to derive $\mathcal{N}^*$. The neurons in $\mathcal{N}^*$ were divided into the sets of neurons O^*, which contains all neurons that are output units in $\mathcal{N}^*$; O contains those nodes in $\mathcal{N}^*$ that correspond to output units in $\mathcal{N}$; in the same way, I^* contains the input neurons of $\mathcal{N}^*$ and I those corresponding to input neurons in $\mathcal{N}$; H is the set of all other, hidden, units. Teacher forcing was applied on those units in $F = H$.

Forward pass

At the first time step,the nodes in I are fed the training sequences first values $\mathbf{t}(\tau = 0)$ as their outputs without any modification. From those values, the network

produces a first guess of the next time step, consisting of the output values for individual neurons $j \in O$, $\mathbf{o}(\tau = 0)$. These values would usually just be fed into the network again, but as teacher forcing was applied in units in H, those values are corrected a little before being passed on. For all other units, the input values are computed the usual way.

$$net_k(\tau) = \begin{cases} \sum_j (o_j(\tau) w_{kj} + \lambda e_j(\tau)) & \text{if } k \in F = H \\ \sum_j (o_j(\tau) w_{kj}) & \text{if } k \in O, O^* \end{cases} \tag{2.20}$$

Note that there is no rule on how to compute input values for neurons in I^*, as these neurons coincide with those in O.

Backward pass and weight changes

Based on the network's output the error to be fed back into the network is now computed and propagated back to derive a value $\delta_j(\tau)$ for every time step τ and neuron j. As those neurons in $O \setminus O^*$ receive their error values from the nodes in H, on which teacher forcing is applied, the virtual error propagated back from those neurons is weighted with $(1 - \lambda)$.

$$\delta_j(\tau) = \begin{cases} f'\left(net_j(\tau)\right) \cdot e_j(\tau) & \text{for } j \in O^* \\ f'\left(net_j(\tau)\right) \cdot \left(e_j(\tau) + (1-\lambda) \cdot \sum_k w_{kj} \delta_k(\tau + 1) \right) & \text{for } j \in O \setminus O^* \\ f'\left(net_j(\tau)\right) \cdot \sum_k w_{kj} \delta_k(\tau) & \text{for } j \in H \end{cases} \tag{2.21}$$

These values where then used to compute a weight change for every weight w_{ji}:

$$\Delta w_{ji}(n+1) = \eta \cdot \frac{\sum_{\tau \in T} \delta_j(\tau) \cdot o_i(\tau)}{|T| - 1} + \beta \cdot \Delta w_{ji}(n) \tag{2.22}$$

2.2 The symbol grounding problem

In the introduction, we have given a quick overview over the capabilities of symbol systems. Here, we want to give a more detailed description of how they work and their limitations.

Harnad published the symbol grounding problem in its original form in 1990 [8]:

> How can the *semantic interpretation* of a *formal symbol system* made *intrinsic to the system*, rather than just parasitic on the meanings in our heads? How can the meanings of the meaningless symbol tokens, manipulated solely on the basis of their (arbitrary) shapes, be grounded in anything but other meaningless symbols?

We have emphasized those central terms that we will try to explore and explain further in the following sections.

2.2.1 Symbols

We already gave a short overview on how symbols work, which possibilities they provide and that their grounding is crucial to successful employment of symbol systems in construct a system with strong artificial intelligence.

Ogden and Richards recapitulated these mechanisms involved in the generation and interpretation of symbols in the *semiotic triangle* [15] (Fig. 2.6). One can easily separate three physically distinct entities and processes involved in symbol use: The symbol itself, or rather its physical manifestation like ink on paper or sound waves in the air; The meaning of or concept behind the symbol, which consists of processes inside the head of the interpretor; And the object or physical manifestation of the designated meaning, like one specific horse or one specific occurrence of a horse eating grass. The mind identifies physical objects, properties or affairs and connects them to the respective concepts it has of them. In the same way, the physical manifestations of symbols, like written letters or spoken words, get connected to the concepts they resemble. The concept in the human mind is the central element, over which the connection between the symbol and its referent is made.

There is evidence that "data structures", such as abstract concepts which are formed in the human mind, are tightly connected to those structures that perform basic sensor and motor tasks. There are many studies that show that the mere act of understanding a sentence can have an impact on movements performed during listening or reading [7] [16]. This observation, among others, led to the development of the embodiment hypothesis, which states that there can be no artificial intelligence that does not have some sort of body with sensor and motor skills at its disposal, as the workings of the body considerably shape the mind of the body's owner. For further information on this hypothesis and grounded cognition we will refer to Barsalou [1], Gallese and Lakoff [5], and Pecher and Zwaan [17].

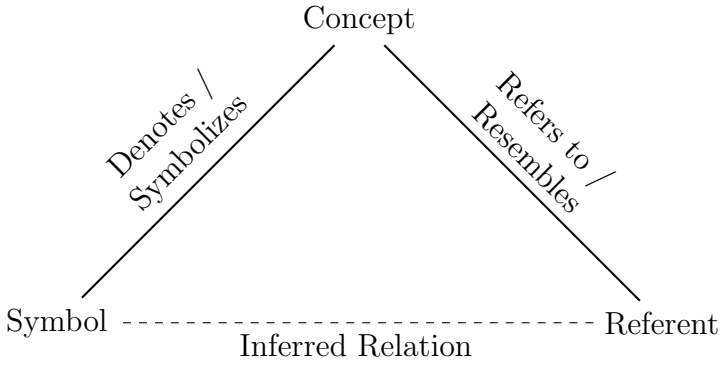


Figure 2.6: The semiotic triangle depicts the relations between those entities involved in the process of connecting a symbol to its physical referent.

2.2.2 Symbol systems

In his article, Harnad gives a definition of a symbol system. He calls it a reconstruction from several sources ([4], [13], [19]). As there exist several different definitons of symbols and symbol systems (e.g. [3], [12], [18]) and this is the one that Harnad refers to when posing his problem, we will cite it:

> A symbol system is:
>
> 1. A set of arbitrary physical tokens[:] Scratches on paper, holes on a tape, events in a digital computer, etc., that are
> 2. Manipulated on the basis of "explicit rules" that are
> 3. Likewise physical tokens and strings of tokens. The rule-governed symbol-token manipulation is based
> 4. Purely on shape of the tokens (not their "meaning"), i.e., it is purely syntactic, and consists of
> 5. "Rulefully combining" and recombining symbol tokens. There are
> 6. Primitive atomic tokens and
> 7. Composite symbol-token strings. The entire system and all its parts – the atomic tokens, the composite tokens, the syntactic manipulations both actual and possible and the rules – are all
> 8. "Semantically interpretable:" The syntax can be systematically assigned a meaning e.g., as standing for objects, as describing states of affairs.

As an illustrating example take natural language and let us identify the described parts: language consists of a set of tokens, words (6), which can be uttered in script, sound, gestures, magnetic tape or other physical form (1). There are grammatical rules (2) that describe how these words may be combined (5) to form sentences (7). These grammatical rules do not take into account whether the constructed sentences make sense, but only take into account a few properties of the word (4), like whether it's a noun or a verb, its gender, whether it's in singular or plural etc. So both "The horse eats the grass." and "The grass eats the horse." are correct sentences, "Grass the ate horses." is not. These rules can be written down in the same language (3). Words and phrases or sentences constructed from them contain a meaning (Which does not imply it makes sense, however.) (8).

Other examples for symbol systems include chess and other games, the lambda calculus or the operation of a computer, which consists of bits (1, 6), an instruction set (2, 3), and a state including any form of data in its memory (7) that it generates or manipulates (4, 5). That data can then be interpreted in an appropriate way (8).

2.2.3 Capabilities provided by symbol systems

Symbol systems are more abstract than the real world and hereby make it possible to use the knowledge about the symbol system in many real life applications. E.g. the basic rules of arithmetic can be used to add numbers of all sorts of physical objects:

$$\begin{aligned} 2\,\text{Ducks} + 2\,\text{Ducks} \quad &\Rightarrow \quad 2+2=4 \quad \Rightarrow \quad 4\,\text{Ducks}. \\ 2\,\text{Horses} + 2\,\text{Horses} \quad &\Rightarrow \quad 2+2=4 \quad \Rightarrow \quad 4\,\text{Horses}. \end{aligned}$$

After having counted the objects, it is completely irrelevant which objects we talk about. As depicted by the first arrow, we can drop all information except the numbers and the plus sign. Then, one only needs to strictly follow the rules of arithmetics to compute the result. The information dropped before can then be picked up again to find what the abstract result means in this concrete situation, but it needs not be present in the actual calculation. This property simplifies the process very much and is the reason for the re-applicability of our arithmetic knowledge. Simplification of the thinking process and re-usability of knowledge are desired properties in the design of intelligent artificial systems.

2.2.4 Issues in using symbol systems

We saw that the information loss that symbol systems introduce is quite useful in many ways. However, we have not yet found a way how the transition from the real world to the symbol system and back can be done by an artificial intelligent agent. Conditions 1 through 7 in the definition of symbol systems constitute a system that is completely self-referential. Condition 8 gives a hint to the fact that the ability to interpret and generate symbols is crucial to being able to use symbol systems.

As an illustration, John Searle [22] brings in his *"Chinese Room Argument"*. When we consider a computer program that works solely on symbols, following some semantic rules (like the rules of mathematics or word concatenation), Searle states that this program might even become able to pass the Turing Test, but that this fact will not necessary guarantee that the program will have any understanding of language or mathematics at all. To illustrate this, he imagines himself in a locked room with a chinese/chinese dictionary and a concise manual, written in english, of chinese grammar, style and other language paradigms. Now, he says, if someone tried to lead a conversation with him in chinese, perhaps by shoving paper slips through under the door, the person outside might not be able to tell that Searle does not speak one word of chinese. He would be able to produce answers to any letter he receives by manipulating the received symbols and those that are closely related through the dictionary by just carefully applying the rules in his manual, without understanding one word of what he receives or sends.

In subsection 2.2.1, we have found that in humans, this connection is made by connecting both the object and the symbol to the concept or meaning in our mind. In artificial agents, we so far have found no way to generate and store some sort of

data structure that might resemble a "meaning," and can thus not use it to make this connection.

Also note another aspect of Searle's experiment. The chinese speaking correspondent to Searle might interpret Searle's letters to contain meaning (which they might do – to the reader.) That observance might fool her into believing that that meaningfulness was under Searle's control in that he actually understood what he writes (which he does not). We conclude that the meaningfulness of the output of a system might be *parasitic* on the capability of the observer to interpret the output. This effect might let observers wrongly conclude that an agent has succeeded to connect a symbol with its meaning.

We therefore need a mechanism that enables an artificial agent to translate between real world objects and affairs and symbols. In developing such a mechanism, we must take special care that the artificial agent does not parasitically rely on our own semiotic knowledge.

2.2.5 Excourse: Zero semantical commitment condition

In the course of research on the symbol grounding problem it has been found that many proposed solutions circumvent the problem of finding and grounding symbols and rather propose a mechanism by which already existing and grounded symbols can be learned by the artificial agent [23]. They do provide a possibility to connect a symbol to a real world object, but require semantical resources and knowledge that are external or pre-built into the system. The actual finding and first grounding of those symbols is done by the researchers conducting those experiments.

Any approach that uses neural networks falls into this category, as the data sets used to train the networks can only be generated by an external agent that already has strong knowledge about the symbols, their meanings and semantics. These solutions will be further evaluated to see how well this approach can be used to implement language and language understanding in robots, but in the long run it will be more interesting to have intelligent agents not only learn to properly use those symbols that we, humans, teach them, but that they also learn to build up their own symbol systems, i.e. "languages in which they think" from scratch.

This condition shifts the problem from what one might think of as the problem to connect the symbol to anything that one can understand to the problem of how one could generate symbols to understand the world, that then other symbols can get connected to.

Chapter 3

Approach

In this thesis we will re-conduct and discuss a set of experiments that were originally made by Marocco et al. [11] in simulation, as a proof-of-concept. This work tries to make these experiments on a real robot and to further investigate their symbol grounding mechanism. Therefore, in this section, we will first give a short description of their experiments. we will then further analyse their methods to find a) possible methods to port the experiments to reality and b) those parameters, design decisions, etc., that may have a considerable impact on system performance.

3.1 Experiments in Marocco et al. 2010

For their experiments, Marocco et al. set up three virtual situations in a simulation [24] of the iCub robot [14]. In all of them a simulated robot model stood in front of a table with an object lying on it (see figure 3.1). That object was either a sphere, a cube or a cylinder. The sphere and the cube were free to roll or slide, respectively, while the cylinder was fixed to the table surface and did not move when hit or pushed.

When the robot was advised to hit the object (i.e. change one joint angle in its shoulder), the object reacted accordingly. A program moved the robot's head so that the robot would always look at the object.

The robot could sense these movements. Also did the robot have a simulated touch sensor in it's hand. From what the robot saw, an algorithm detected the object and calculated a roundness value, indicating how round the perceived object is, too.

During this process, for 15 seconds the angles in the head and shoulder joints, the touch sensor values and the roundness values were recorded with a frame rate of two samples per second.

These values contain enough information to distinguish which object the robot encountered. As the robot was hard-coded to follow the object with its head, the head angle values contain information about where the object was at a specific time. The shoulder angle gives hints to the type of object, as e.g. in case of the cylinder, the arm could not move past the point when it hit the object. The

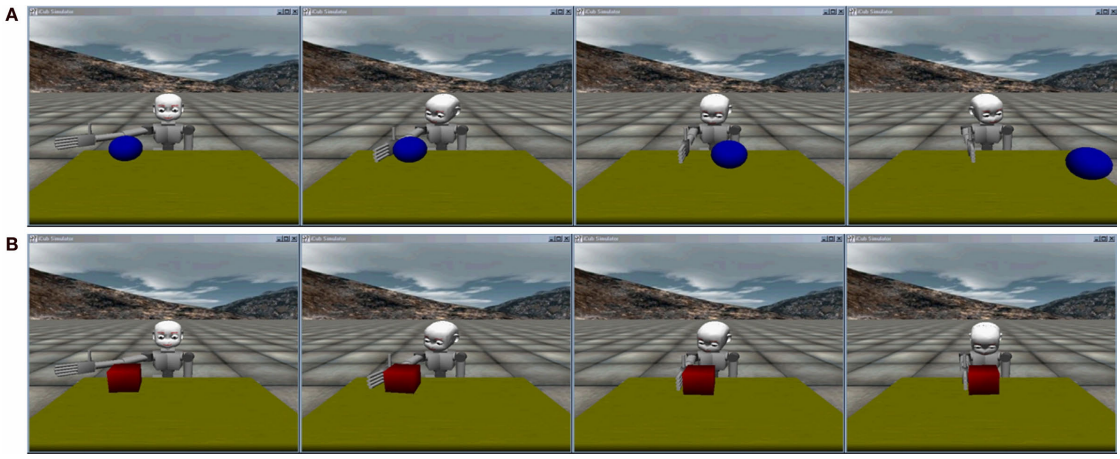


Figure 3.1: A set of screenshots showing two of the situations in the experiments of Marocco et al. Picture taken from [11]

sphere rolls away when hit, so that the touch sensor is triggered only shortly, while it remains triggered once the robot had contact with the cube, which only slides as far as it gets pushed, or with the cylinder, which does not move at all. The roundness value also gave helpful information to tell the objects apart, as all of them had a different shape, and therefore a different roundness value.

In the next stage of the experiments, Marocco et al. used these values recorded to train neural networks to reproduce these sequences, using the backpropagation through time algorithm described in chapter 2.1.5.

They added a set of three inputs which were set to [1 0 0] for the sphere, [0 1 0] for the cube, and to [0 0 1] for the cylinder situation. These inputs resemble primitive words, and simulate that the robot was told these words when the object was presented to it. These neurons were not directly trained, but rather "co-trained" during the process by the backpropagated virtual error (see equation (2.16)) and the mechanisms introduced by teacher forcing (see section 2.1.6).

These trained networks were then connected to the simulation, getting their input from the sensors and giving their outputs as commands to the shoulder actuator[1]. They reproduced the learned sequences, i.e. steered the arm movement while constantly predicting the sensor values in the next time step. The results of Marocco et al. show that, as a side-effect, the linguistic in- and outputs were set to the according word.

3.2 Identification of Components

It is possible to divide the system that Marocco et al. used into several subsystems. The component of interest is the neural network that itself poses as a link between a sensorimotor experience and a linguistic symbol. That grounding task is the main focus of the network. It can be further subdivided into the task of following

[1]The head was still steered by the object tracking program.

and predicting the sequence that the robot encounters, and the task of identifying the sequence and uttering the corresponding word. The former is trained by the backpropagation through time algorithm, the latter is a result of the effects mentioned in section 2.1.6

To make it easier to examine the network in by which manner and quality it solves this task, it is desirable to keep the in- and outputs as simple as in the approach of Marocco et.al. For that purpose, all calculations that are not part of the network's task are done by external components.

One of these components is the head tracking system. It ensures that the robot's head looks at the object at all times. The purpose of this is to transfer the information of the object's position from the raw camera data to the head angle values, which are fewer and therefore lead to a smaller and more focused network setup that is easier to examine.

A system that works in a quite similar way is that program that calculates the roundness value in the experiments of Marocco et al. Also does it only extract and pre-process information from the vision system for easier processing by the network that can then use this information in the form of one single value. Note that this component is not applied in the setup used for this thesis, as Marocco et al. have found that removing it and simply setting all its values to zero had no effect on the outcome of the experiments.

3.3 Open questions

The experiments made by Marocco et al. serve only as a proof of concept. To actually use mechanism proposed by Marocco et al. for grounding language in the experiences of a robot, more information is needed. They do not give procedures to find the parameters used in training the networks or those defining the network structure itself. They have made some experiments that explore the impact of variations of the initial position and size of the object. Also did they find that one value that they used, the roundness, could be left out without degrading the quality of the results.

However, for a real life implementation of the mechanism, the impact of more variables and parameters has to be explored. These include, but are not limited to the number of hidden neurons, the learning rate, the properties of the data that is used for training and testing and several sources for artefacts and errors.

The following parameters were found to determine the outcome of the recorded values itself:

- Time step in the simulator and the recording: With a higher time step, the arm of the robot moves further into the object before a collision is detected, resulting on a higher force exerted on the object. That changes the way the object moves afterwards. For us, there is no feasible way to find that time step value that resembles reality most closely, leaving us having to guess a value.

- Object position relative to the robot and object size: This also, especially in the sphere situation, greatly influences the objects reaction to the hit.
- Vision tracking parameters: The program might introduce a time lag in the reaction of the head rotation to object movements.

In the real life experiments, the following parameter contribute to the outcome of the data the network receives:

- Object position: This parameter influences the data outcome in the same way as in the simulated experiments. Error magnitudes increase in the real life experiments.
- Vision tracking: In real life, the pictures from a camera contain a significant amount of noise and artefacts, impoverishing the vision subsystem's performance. Additionally, the lighting changes more than in the simulated environments. Above comments on how quickly the tracking reacts to object movement also apply.
- Other artefacts, e.g. inaccuracies in the robot sensors like contact bounce.

Furthermore does the choice of the robot itself influence the data. iCub is a comparatively big robot. NAO, on the other hand, is rather small. Therefore, its movements are smaller and harder to measure with the same accuracy. The robots have different body proportions which further influence how equal movements may lead to different values in the sensor data.

We therefore pose the following questions:

1. **Is it possible to use the mechanism with training samples recorded on a different robot to train networks that produce results of the same quality?**
2. **If so, is it also possible to use the mechanism on a real robot instead of a simulation?**
3. **If not, does varying different parameters that define the network structure and the training process improve these results? Especially: Can increasing the number of hidden neurons, varying the learning rate, manipulating the training samples or choosing different training samples altogether improve the outcome?**

3.4 Methods

The mechanism Marocco et al. describe serves as a solution to our symbol grounding problem[2]. When confronted with one of the specified situations, the network produces the correct corresponding word.

[2]Note that by "our problem" we do not mean "the symbol grounding problem as stated by Harnad." We have already found that neural networks generally fail to solve Harnad's problem in Chapter 2.2.

We aim to further investigate this mechanism as to its stability and susceptibility to disturbances. This includes using the mechanism on a different robot with different dimensions and specifications. Furthermore, we wish to investigate the mechanism under disturbances that would commonly arise in a real life application, like artefacts in the vision system leading to slight offsets in the perceived head angles, or variations of the movement paths of object due to slight offsets in the initial position.

To solve this task, we will re-assemble the situation both in simulation and in reality. With the samples recorded, we will train networks in the way described above, and will then evaluate how well they perform.

Depending on these results, we will explore how they can be disturbed, if they are positive, or improved, if they are negative. In this way, we will be able to identify those parameters that are crucial to the applicability of the mechanism, and those that are not.

3.5 Description of the implementation of the components

3.5.1 NAO

The NAO is a humanoid robot developed by Aldebaran Robotics. It is used to make the real life experiments and depicted in figure 3.2.

Hardware

NAO is a 57cm tall humanoid robot developed specially for robotics research. It can move in 25 degrees of freedom (2 per ankle, 1 per knee, 2 per hip, 1 per hand, 1 per wrist, 1 per elbow, 3 per shoulder, 3 in the head)

It is equipped with two cameras, four sonar sensors, four pressure sensors in each foot, one three-axis accelerometer, two gyrometers, four microphones in its head, two speakers and several LEDs and push buttons.

Furthermore it has a x86 AMD Geode 500MHz CPU, 256 MB of SDRAM and 2 GB flash memory at its disposal for computations [20].

3.5.2 Implementation of the Network and BPTT algorithm

We used recurrent neural networks trained with the backpropagation through time algorithm which were described in section 2.1. The networks had three layers[3] with seven neurons in the input/output layer and ten in the hidden layer. We did use a momentum term as explained in section 2.1.3 and applied teacher forcing (see section 2.1.6) to the hidden neurons.

[3]Only two of these were computationally effective. As the networks were driven recurrently, the input and the output layer can be seen as merged into one layer. Refer to sections 2.1.2 and 2.1.4 on the terminology of multi-layer and recurrent networks.

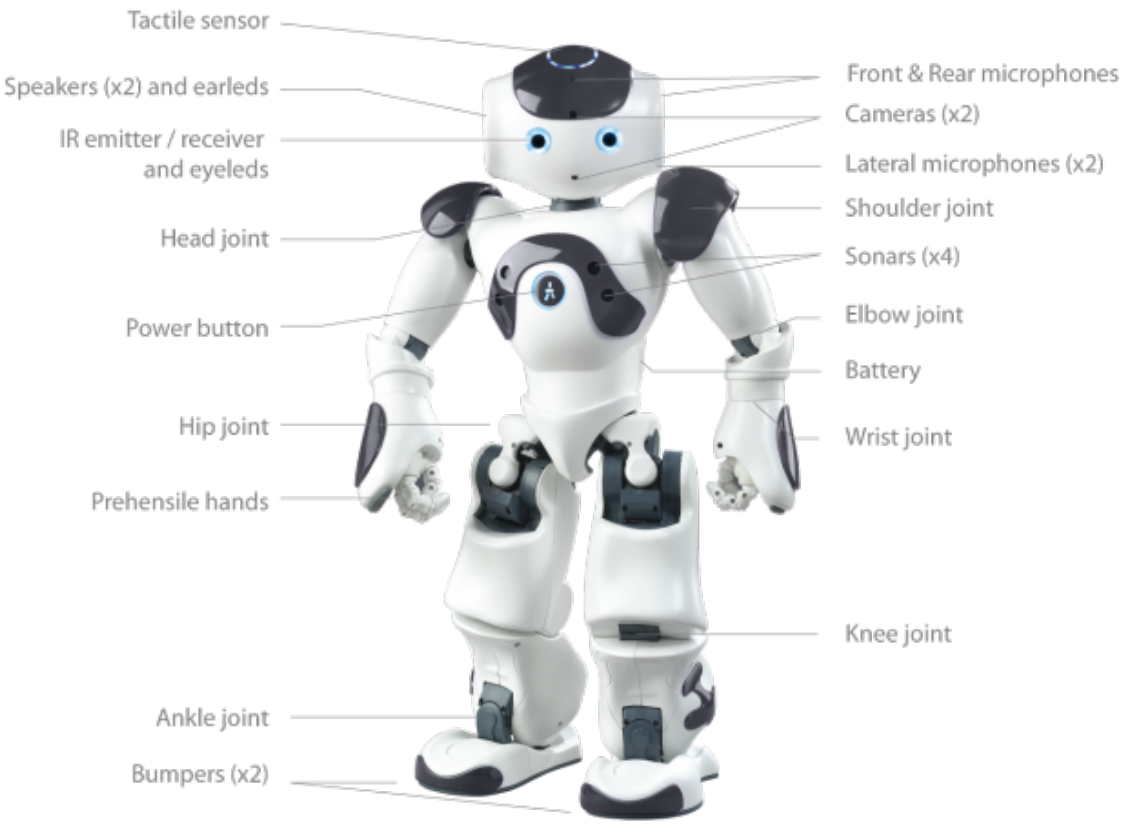


Figure 3.2: NAO humanoid robot version 3.2. In the course of these experiments, a leg-less version of this robot was used. Picture taken from the NAO documentation which is available under http://users.aldebaran-robotics.com

Of the seven input/output neurons, four, the *sensorimotoric neurons*, were to predict the sequences that we recorded on the robot and in simulations (See subsection 3.6.1), while the other three, the *linguistic neurons*, had the task to identify the sequence and react with the correct output. The sensorimotoric neurons were trained in the way described above. The linguistic neurons, on the other hand, did not receive injections of absolute error as detailed in Equations (2.14) or (2.15), but received only the backpropagated virtual error as described in Equation (2.16). Thus, the linguistic nodes' training relied exclusively on the backpropagation of error values and those effects introduced by the teacher forcing.

For further information on the excat modalities and used parameters, refer to section 3.7.

The neural networks used were implemented in C++. As C++ is an object-oriented language, we chose an object-oriented model, modelling neurons, nets and teachers as classes that are as similar to the abstract models as possible.

It would have been possible to use a more procedural programming paradigm, implementing the neurons as mere structs, the interconnections as a matrix and the teacher as a function working on these data structures. However, we chose our approach for the easier maintainability, as the code directly resembles the abstract

model already in mind, making it easier to reason about abstract structures and processes and then recognize their counterparts in the code.

To enable the usage of the backpropagation through time algorithm with teacher forcing applied, we needed a possibility to replicate the trained network while still ensuring that all weights that correspond to each other maintain the same value. For this purpose, instead of replicating the whole network, all key values in the individual neurons, i.e. the output and the error input, were implemented as vectors. These vectors were dereferenced by a global variable indicating the current time step, allowing to make and save several forward passes before making their respective backward passes, both of which were explained in detail in section 2.1.7. The weights supported the execution of several backward passes to compute the respective weight changes without applying them. Instead, the weight changes were saved as a cumulative average, which was then applied after all backward passes had been executed.

3.5.3 Implementation of the vision system

For technical reasons, we had to use a different approach to implement the vision system on the real robot and in simulation. Those API functions used in the simulation turned out to produce a very jagged movement on a real robot, so we used an approach that implements the movement one API layer below that layer used in the simulation. That layer, on the other hand, is not available in the simulator, forcing us to use two different vision system in the two different applications.

Both systems have in common that they use a simple proportional controller. They take the pictures from the camera and scan it for a certain color (The objects were of a bright neon pink.) They compute the mean of all these coordinates and designate it as the objects center. The differences between the objects center position and the center of the picture are multiplied with a parameter we called the *tracking rate* to directly compute values that were then sent to the respective motor system as correction values. (See fig. 3.3.)

3.6 Generation of training and testing data

The following table gives an overview over the types and numbers of recorded samples. The sequences differ in the setting in which they were recorded (in simulation or on the real robot). The samples from the simulation are further divided into one sequence per object in which the object was placed at a determined location. Additional 100 sequences per object were taken in which the object was placed at a random location within a 10cm x 10cm square.

Table 3.1: Recorded sample sequences

Setting	Simulation		Real robot
Object placement	determined	random	random
Samples per object	1	100	20
Samples total	3	300	60

3.6.1 Sample recording methods

Simulation

To generate samples to train the networks and test against, we set up the three situations with the three different object in the Webots robot simulator [25]. The files included the physical objects, like the ball, the box, the cylinder, the table, the NAO robot, and the supervisor, a robot which had no "physical" body and merely existed to steer the simulation.

NAO's model was extended by a touch sensor in its right hand and a radio emitter to communicate with the supervisor. The controller program steering NAO was modified to start a python script that steered its movement and the head tracker program described above. The supervisor randomly placed the object on the table, if so required, and waited until NAO signalled that the python script had exited, and closed the simulator then.

During the process, a python script that also steered the robot's arm logged for every 20ms the line index, the current time, and the values for the touch sensor, head yaw, head pitch and shoulder roll. A typical record would look like the following few lines. They are an excerpt from the sphere situation (notice the touch sensor being triggered at index 16). The lines represent the values in the order in which they were named. The time is shown as the numbers of 10ms units since the beginning of the simulation.

```
15                  16                  17                  18
2071.99993134       2073.99997711       2076.00002289       2078.00006866
0.0                 1.0                 0.0                 0.0
-0.417158335447     -0.417158335447     -0.417158335447     -0.417158335447
0.0924246013165     0.0924246013165     0.0924246013165     0.0924246013165
-0.0415359959006    -0.0173976961523    -0.0087266461923    -0.00872664619237
```

Real World

The simulated NAO model uses the same middleware as the real robot, so the same setup was used to generate the samples recorded on the real NAO. The head tracker was replaced by it's counterpart, and the touch sensor was emulated by pressing a button on its head manually while the hand had contact.

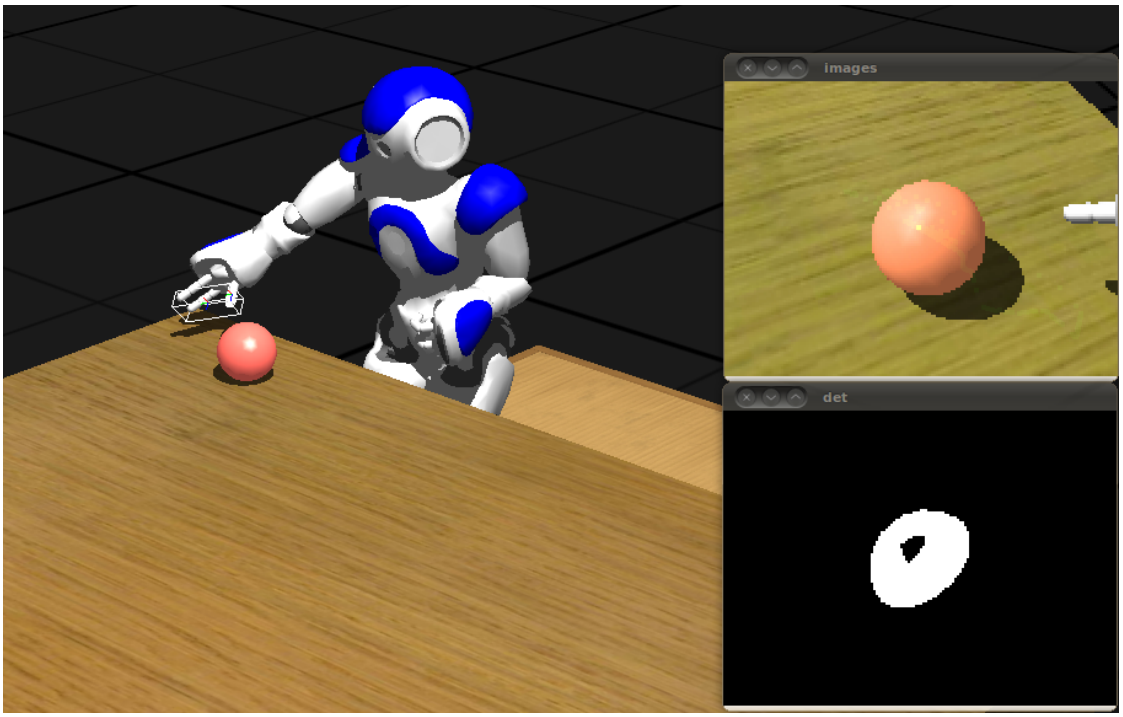


Figure 3.3: A screenshot of the simulation with the sphere. Visible on NAO's right hand is the bounding box that acts as a touch sensor. Visible to the upper right is the picture from NAO's camera, on the lower right a processed picture in which those pixels are coloured white that have been detected to be of the neon pink target colour. The tracker aims to move the head so that the robot always looks at the center of this white area (marked as a bright spot in the upper right window.)

3.6.2 Preprocessing

We generated six different training files with three different resolutions for both the deterministic samples and a set of ten random recordings. The samples from the real robot were not used for training.

Table 3.2: Generated network training files

Setting	Simulation		Real robot
Object placement	determined	random	random
Resolutions	15, 30 and 60 time steps / 4.8s		-
Samples per object	1	10	-
Total number of files	3	3	-

The first 240 lines of the samples (which describe 4.8 seconds of real time) were stripped from the index and time field. We then generated files of lengths of each 15, 30 and 60 time steps. For that purpose, only every 4^{th}, 8^{th} or 16^{th} sample was extracted. For the training file with the fully deterministic samples, we concatenated the three

respective files. For the random samples, we chose to concatenate the first ten files of each object. The files were then prepended a header containing information on how many sequences with how many channels and time steps the file contains and on the configuration of the channels (motor or linguistic) and appended the respective lines of ones or zeros for the linguistic units.

Also did we generate three train files in the same way with the three different resolutions for each sample recorded. These were later used to test each single sample on how well a network can predict and categorize it.

For the case of the deterministic samples, these sequences can be seen in figure 3.4. Each of them show the movement of the arm ("Shoulder Roll"), the excitation of the touch sensor ("Touch"), and how the object reacts in form of the values for the head yaw and pitch values. The cylinder does not move, the cube moves only a slight amount while staying in contact with the hand, the sphere rolls away.

3.7 Training

3.7.1 Training Parameters

We began the experiments with the following parameters:

Table 3.3: Initial training parameters

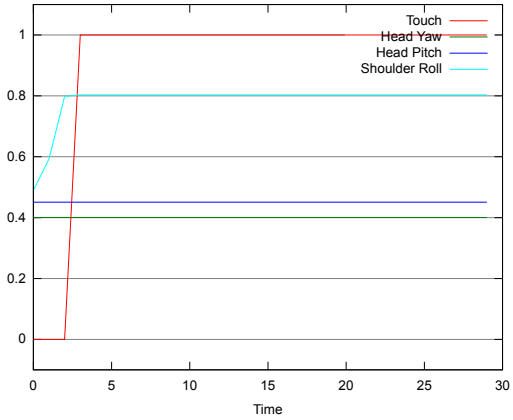
Parameter	value
Number of hidden neurons	10
Learning rate	0.005, static
Momentum	0.1
Teacher forcing quotient	0.1
Used training file	Simulated, deterministic
Maximum number of epochs	10 million
Error threshold	0.001

During training, we measured the overall error (Eq. (2.3)) for all sequences $s \in S$ and time steps $\tau \in T_s$ and normalized it to the number of individual input/target patterns in the sequences and to the number of sensorimotoric output neurons. The linguistic neurons did not have determined target values and are therefore not included into the error calculation. With M being the set of sensorimotoric input/output units, we define the *sensorimotoric error*:

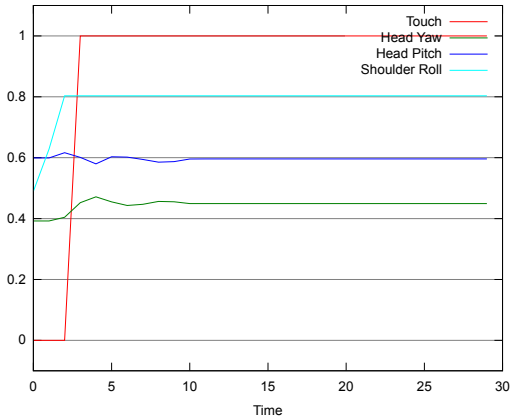
$$E = \frac{\sum_{s \in S} \sum_{\tau \in s} \sum_{j \in M} (t_j(\tau_s + 1) - o_{sj}(\tau_s))^2}{|S| \cdot |s| \cdot |M|} \tag{3.1}$$

We aborted the training when the error fell below 0.001 or after 10 million epochs of training.

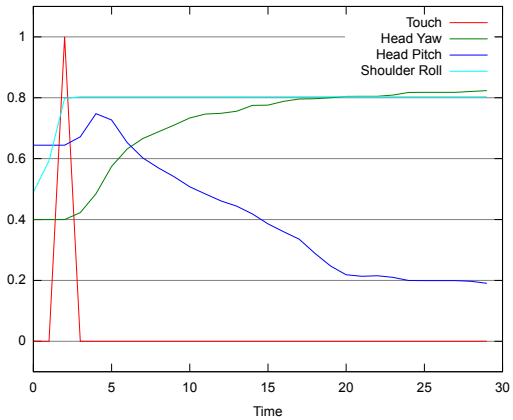
We also closely studied the training process itself. For that purpose, the networks error, current output under testing conditions and weights were saved every 10000 epochs.



(a) Cylinder



(b) Cube



(c) Sphere

Figure 3.4: The deterministic training samples that were used for the training and testing. These values have already been mapped from the range of their possible values to the range between zero and one.

3.7.2 Variations in training parameters

We trained several further networks, under variation of each of these parameters:

- Learning Rate: We additionally used the learning rates 0.002 and 0.01. As a second measurement, we tried using a dynamically adjusted learning rate: Every 10000 iterations, the current average square error for the sensorimotoric units was computed and used as the learning rate for the following 10000 epochs. This has the effect that the network converges quicker while it is farther away from the optimal solution, but decreases the learning rate once it approaches a minimum to prevent overshoot.

- Number of hidden neurons: We increased the number of hidden neurons from 10 to 20 and 30. Networks with higher numbers of hidden neurons tend to have a stronger ability to differentiate between presented inputs and react more differently to similar inputs.

- Used training file: The file containing the deterministic sequences consisted of only three different sequences, while that of the random samples contained 30. This stresses the networks "capacity," i.e. it's ability to store a certain number of different sequences, which is determined by the number of neurons, several properties of the presented data and the parameters of the training process.

3.7.3 Tested configurations

While the variation of above parameters allows for a multitude of different configurations, only the following were chosen to be examined further. The increase or decrease of the static learning rate showed no considerable improvement, which is why that approach was abandoned. With increasing numbers of neurons we also decided to not use the random sample training files with a static learning rate, as the increase in neuron count and in count of sequences lead to an unacceptable increase in computation time. The following table lists the different configurations which were used. Furthermore, it states which resolutions of the different training files (with either random or deterministic object placement) have been trained on these networks. Altogether, 25 networks have been trained.

3.8 Testing

We tested each generated network against each sample recorded in the simulator[4]. We test the capabilities a) to predict and b) to identify the situation that is replayed to the network. We took the sensorimotoric error as a measurement for how accurately the network can predict and follow the replayed sequences. In the same manner, we define the *linguistic error* as the average square error over all linguistic neurons, and take that value as a measurement for how well the identification task is solved.

We replayed sequences to the networks, i.e. the sequences are fed into the sensorimotoric neurons one time step at a time and the outputs are saved. During that process,

[4]The testing sample had to be of the same resolution as the file the network was trained with, of course.

Table 3.4: Overview over the varied configurations of all networks trained.

# Hidden neurons	Learning Rate	Object placement	Sample resolutions
10	0.01	deterministic	15, 30, 60
		random	30
	0.005	deterministic	15, 30, 60
		random	30
	0.002	deterministic	15, 30, 60
		random	30
	dynamic	deterministic	30
20	0.005	deterministic	15, 30, 60
	dynamic	deterministic	15, 30, 60
30	0.005	deterministic	15, 30, 60
	dynamic	deterministic	15, 30, 60
		Total:	25 networks

the linguistic neurons are driven recurrently so that they propagate their output into the next time step, while the sensorimotor neurons are driven in a feed-forward manner. In the first time step, the networks sensorimotoric inputs are set to those values $\mathbf{t}_j(0)$ from the recorded samples, while the linguistic neurons' inputs are set to zero. Like this, the network predicts the sequences' next time step from the current one. During that process, the linguistic neurons are free to establish more long time stable configurations. These configurations get influenced by and influence the state of the sensorimotor neurons, too.

As the networks did not show the intended behaviour in response to these sequences, we refrained from testing those sequences recorded on the real robot.

Chapter 4

Results

In the first section, we will present the outcome and development of the training with the initial parameters given in table 3.3. In the following sections, we will present the results for those networks that showed different behaviour when some of these parameters were changed. Where different parameters were used, this will be stated explicitly. Parameters not explicitly given can be assumed to have the value in table 3.3.

Of the networks trained, many yielded very similar results. To keep this section at a reasonable length, we have classified the networks into groups. Those networks in the same group yielded similar results. The entry for the network on which we have given a description that shall serve for all networks in that group is put in bold text. Networks that did not converge but suffer from over-fitting are marked with "ov."

Table 4.1: Overview of the results.

# Hidden neurons	Learning Rate	Object placement	Sample resolutions		
			15	30	60
10	0.01	deterministic	1	1	2
		random	-	ov.	-
	0.005	deterministic	1	**1**	2
		random	-	**4**	-
	0.002	deterministic	1	1	1
		random	-	4	-
	dynamic	deterministic	-	ov.	-
20	0.005	deterministic	4	1/3	**3**
	dynamic	deterministic	4	1	3
30	0.005	deterministic	4	3	**2**
	dynamic	deterministic	4	3	3

4.1 Training results with initial parameters

The training with the initial parameters was aborted after 10 million epochs of training without reaching a sensorimotoric error below the threshold. During the training, every 10,000 epochs we saved the current state of the network to be able to conduct tests on these configurations, too.

4.1.1 After 10 million epochs

Refer to figure 4.1 to see how this network reacts to the replay of the recorded sequences under the testing conditions described in section 3.8.

The value of the touch sensor is predicted well for the cube and cylinder cases. For the sphere situation, it does go up when the hand makes contact and down afterwards, yet the prediction is not accurate. Note that for all cases, the touch output goes up one frame after the touch sensor in the recorded sequence gets triggered.The prediction for the shoulder roll value is accurate in all three situations, except for a minimal overshoot when the hand makes contact with the object. For the head yaw and pitch values, the prediction is very accurate. The slight differences between the cube and the cylinder are identified and obvious from the output. The completely different outputs for the sphere situation are also predicted very accurately.

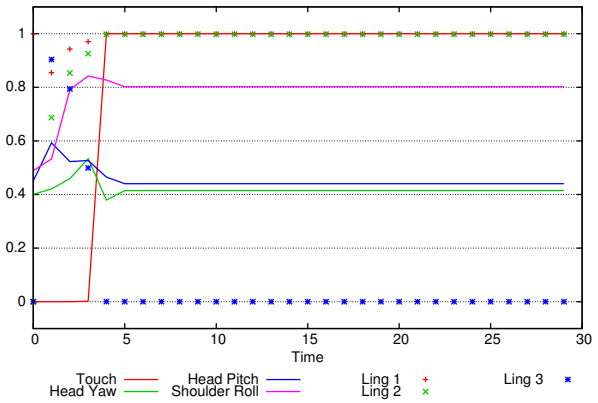
As it gets obvious from these figures, those neurons responding the the cylinder (Ling 1) and the cube situation (Ling 2) tend to produce a high output, while the neuron corresponding to the sphere situation (Ling 3) tends to produce a low output. In the sphere situation, there is a minimal deviation from this pattern.

4.1.2 At 1 and 2 million epochs

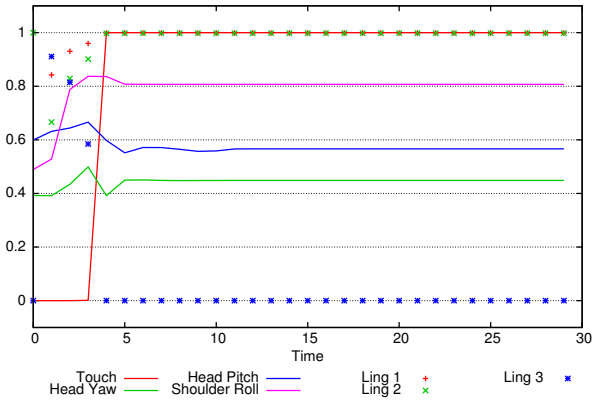
Refer to figure 4.2 to see the equivalent network outputs after 1 million and 2 million epochs, respectively. The graphs reveal that the prediction of the touch sensor value is more accurate than it is after 10 million epochs of training. The low values after the sphere has rolled away are predicted more precisely. The training sequences of the cube and the cylinder only differ in the head yaw and pitch values. After 1 million epochs of training, the network here still produces completely equal outputs for the cylinder and the cube situation. After 2 million epochs, these outputs already reflect the slight differences of the two sequences. Regarding the sphere situation, we can observe that the prediction of the head yaw and pitch values is not very accurate after 1 million epochs, but it does follow the general pattern. The accuracy greatly improves during the following steps of training.

In the outputs of the linguistic neurons the pattern described earlier is already observable. Linguistic neurons 1 and 2 produce the same, high outputs at all times, while the third neuron produces low outputs. The sphere situation deviates from this pattern to some extent, but the deviation decreases with the number of training epochs.

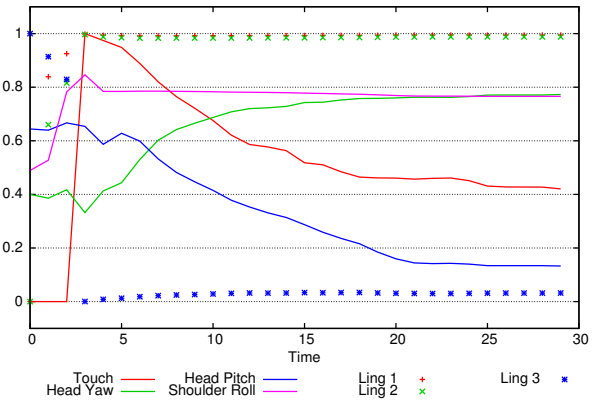
In figure 4.3 we present how the average error values for the sensorimotoric (left) and the linguistic (right) neurons develop during training for each of the situations. The sensorimotoric error falls quickly at early training for all situations. At 2 million epochs, another local minimum is observable. From then on, the error for the sphere situation continually rises, while the errors for the other two situations stay somewhat stable. The graphs confirm the observations made earlier on the development of the identification



(a) Cylinder

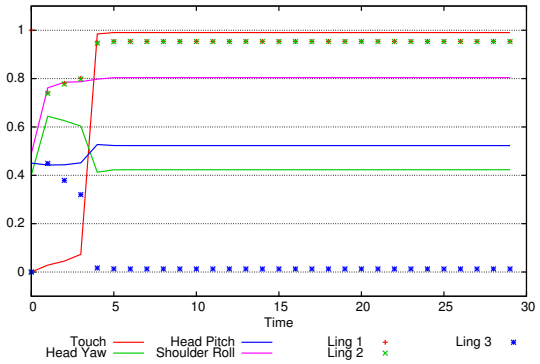


(b) Cube

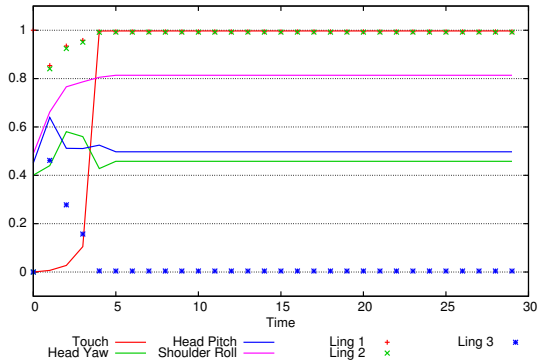


(c) Sphere

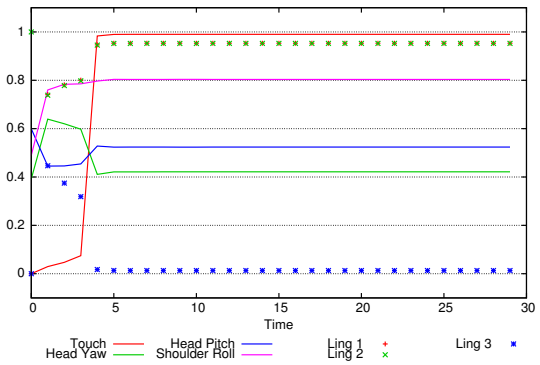
Figure 4.1: Testing outputs of the network trained with initial parameters after 10 million epochs of training.



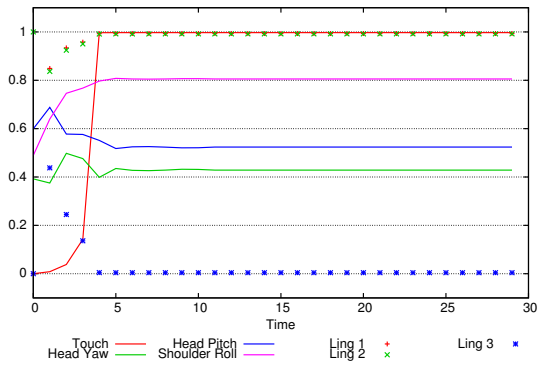
(a) Cylinder, 1 million epochs



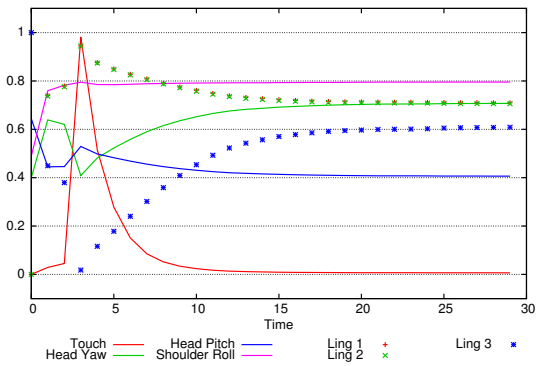
(b) Cylinder, 2 million epochs



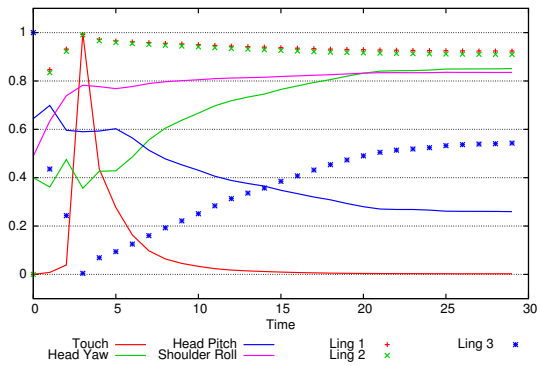
(c) Cube, 1 million epochs



(d) Cube, 2 million epochs



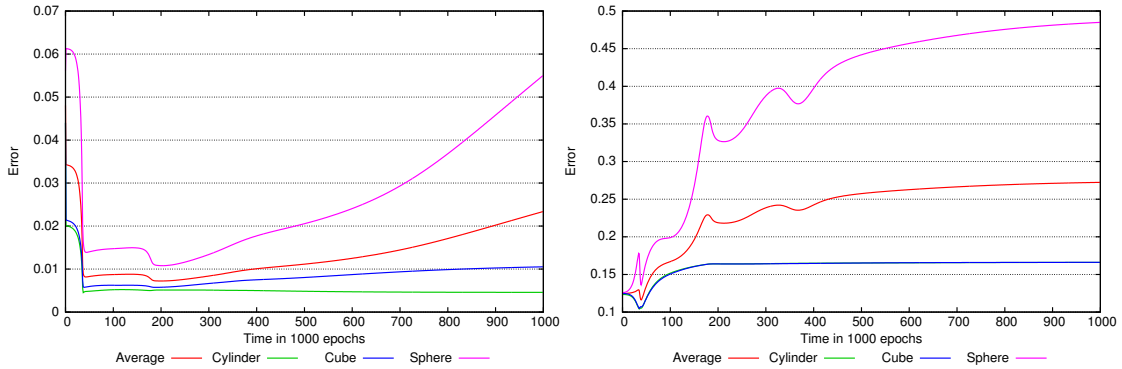
(e) Sphere, 1 million epochs



(f) Sphere, 2 million epochs

Figure 4.2: Testing outputs of the network that was trained with the initial parameters from table 3.3 at 1 million and at 2 million epochs.

performance. The linguistic error for the sphere situation continually increases. The errors for the cube and the cylinder situations remain stable, but high. With the increasing linguistic error also the sensorimotoric error begins to increase again, as we saw especially on the touch sensor value in the sphere situation.



(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.3: Development of the error values during training. Left side shows the errors for the sensorimotoric neurons, right side the average of the linguistic errors. The averages were taken over all samples from the simulator with random object placement.

This network is the representative network for group 1. Their training processes share the following characteristics: Very early, both errors sink very quickly. During early training, the cube and the cylinder situation stay undistinguished by both the sensorimotoric and the linguistic neurons. Ling 1 and 2 stay generally high, Ling 3 generally low. This manifests more and more in training. Towards the end of the training, the head yaw and pitch neurons learn to distinguish the cube and the cylinder, while this has no more effect on the linguistic neurons. The touch sensor neuron, that was able to clearly distinguish the situations earlier, loses most of this ability.

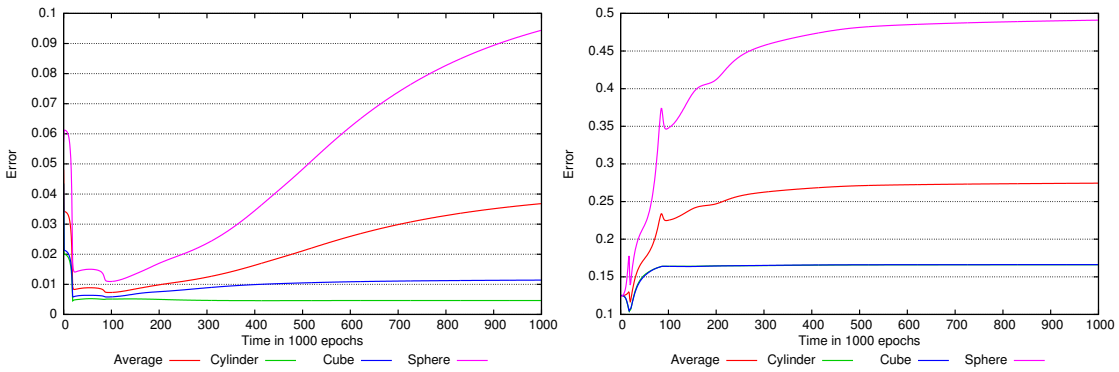
This was the most common outcome in the course of these experiments. Eight networks showed behaviour like this, while a ninth one could not be decisively be categorized into this group or group 3.

4.2 Training results: Variation of the learning rate

The networks trained with the equal parameter settings, but with a learning rate of 0.01 instead of 0.005, show the very same results like the first networks, except that the networks advance in training at twice the speed. This still qualifies it as a group 1 network. Compare figures 4.4 and 4.3. The graphs are similar to each other except that the latter graph seems squashed. The same is true for the network trained with learning rate 0.002. The results did not differ except for the training time. This outcome is to be expected, but like this we could assure that the first network was not just “trapped”

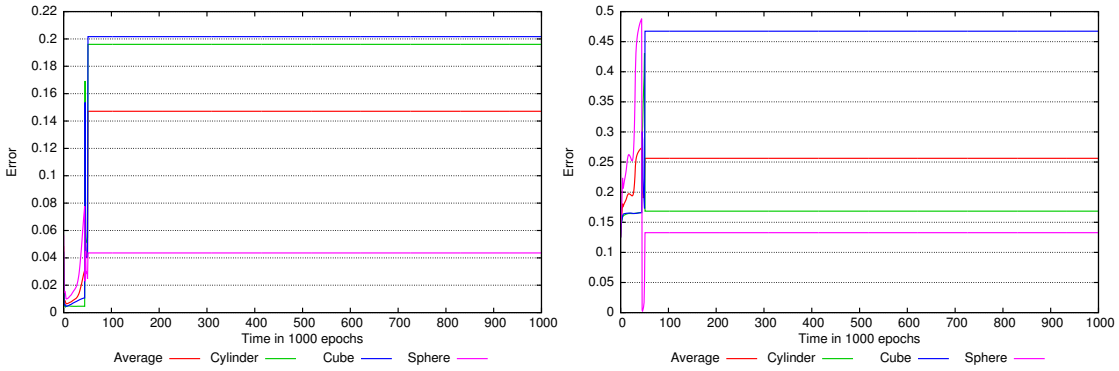
in a local minimum. Lowering the error rate to 0.002 also had no considerable effect on the outcome except that it took twice as many epochs to come to these results.

For the network with dynamically adjusted learning rate, the error development is depicted in figure 4.5. Here, after a few million epochs, the coupling of the learning rate and error value leads to an escalation of both and hence to over-fitting of the neurons. They did no more respond to input but with one specific value. Further training had no effect.



(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.4: Development of the error values during training with learning rate $\eta = 0.01$



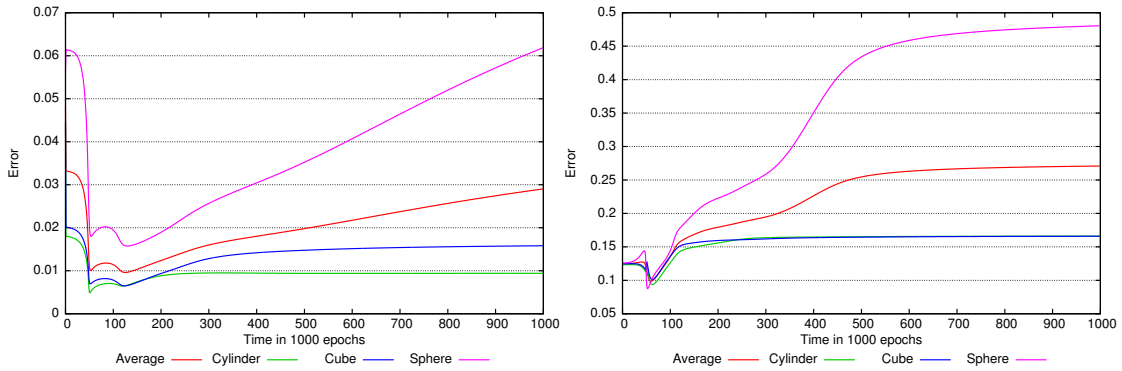
(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.5: Development of the error values during training with the dynamically calculated learning rate. After 50,000 epochs, the dynamic learning rate leads to over-fitting, and further training has no effect.

4.3 Training results: Variation of the sample resolution

We will first show the graphs that depict the error development for some networks. The mere variation of the resolution did not have any significant effect alone. Already from the error graphs, 4.6 and 4.7, can be seen that again the reactions to the cylinder and the cube situations are very similar and invariant. The linguistic errors for both cases quickly converge towards one certain value and do not change afterwards. After this has happened, as seen in figure 4.6, both the sensorimotoric and the linguistic error for the sphere situation rises. This is the same behaviour that we observed with the initial resolution. For the higher resolution, however, as we can see in figure 4.7, the linguistic error for the sphere does not rise, but stays stable. The same applies to the sensorimotoric error.

Refer to figure 4.8 to see the error development of a network with the higher 60 sample resolution and additionally a higher learning rate of 0.01. Here, the linguistic error for the sphere situation is even falling. For this particular network, we have given output graphs for the sphere situation in figure 4.9. In the cube and cylinder situations, the network shows the same behaviour as all networks so far

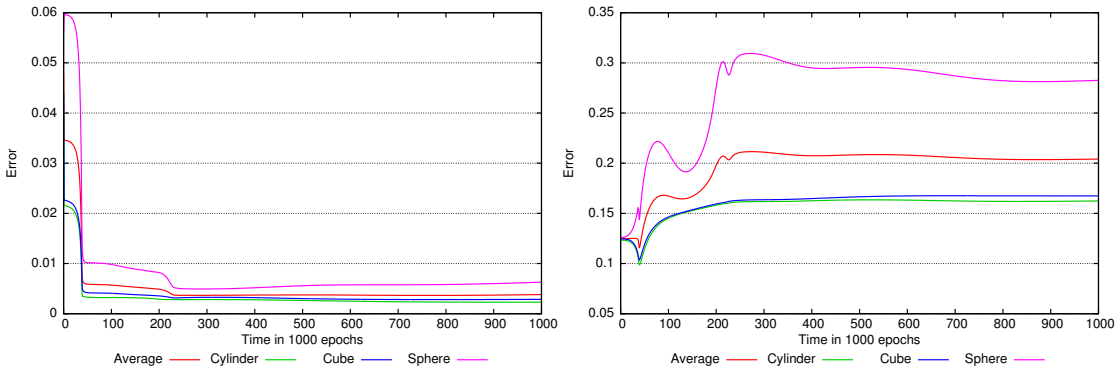


(a) Sensorimotoric Error

(b) Linguistic Error

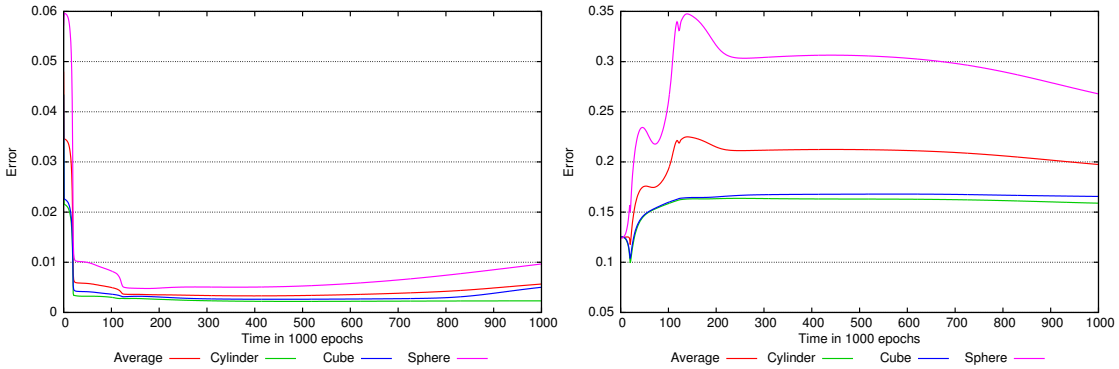
Figure 4.6: Development of the error values during training. The network was trained with 15 samples-per-file resolution training files.

We saved thee output graphs every 10,000 epochs. From replaying these graphs as a video, we could see that the reason why the linguistic error decreases is that the linguistic output 3 in the sphere situation is rising. At the same time, the sensorimotoric error rises slightly as the predicted touch values rise. These two developments can also be seen in the graphs. They are the reason these particular networks have been classified as group 2 networks. The representative network for this group is the one with 30 hidden neurons and the 60-sample resolution training file which will be presented in the following section.



(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.7: Error development for the network trained with resolution 60 samples per file.



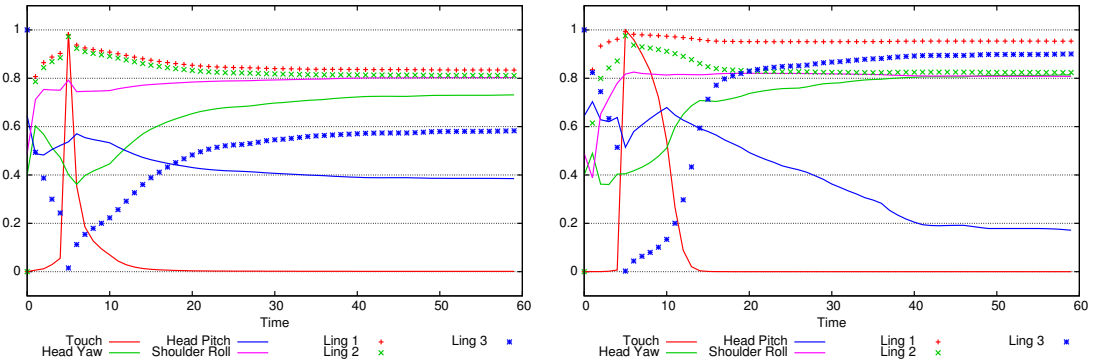
(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.8: Error development for the network trained with learning rate 0.01 and 60 samples resolution. Notice that the linguistic error for the sphere sinks later in training.

4.4 Training results: Variation of the number of hidden neurons

Raising the number of hidden neurons to 20 or 30 led to strong oscillations of the error in the sphere situations of some networks with higher resolutions 30 and 60. These five networks form the group 3. We give the error plots for some of these networks with the two higher resolutions in figures 4.10, 4.11 and 4.12. Additionally, we show the sphere output for the 30 hidden neurons network trained with 60 samples resolution training files (fig. 4.13). In the error plots, this network shows a minimal error for the sphere situation between 2 and 4 million epochs of training.

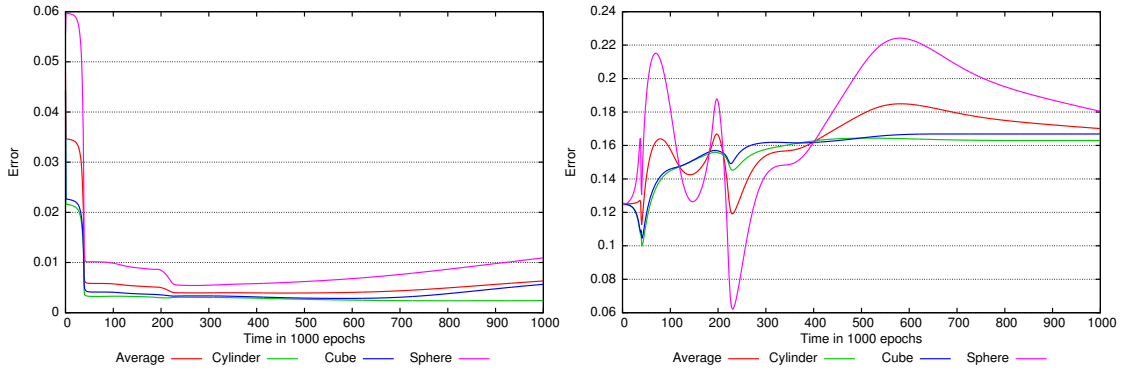
These strong oscillations stay restricted to the error of the sphere situation, however. The cube and cylinder neurons still produce very similar output, which also explains



(a) Sphere, 1 million epochs (b) Sphere, 10 million epochs

Figure 4.9: Testing outputs of the network that was trained with the 60-sample-files and learning rate 0.01 at 1 million and at 10 million epochs.

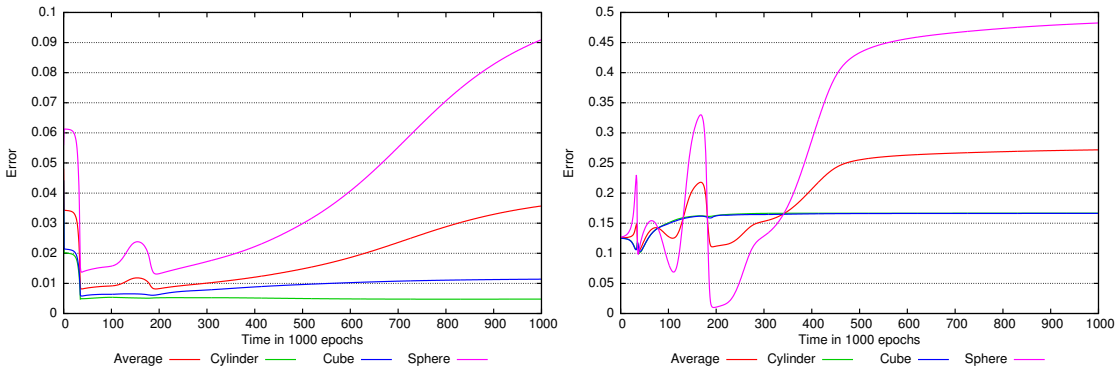
why the errors for the two situations are very similar. The sensorimotoric error takes a comparably smooth development. considering figures 4.10a and 4.10b, we can see that the outcomes of the sensorimotoric errors are quite heterogeneous in this group.



(a) Sensorimotoric Error (b) Linguistic Error

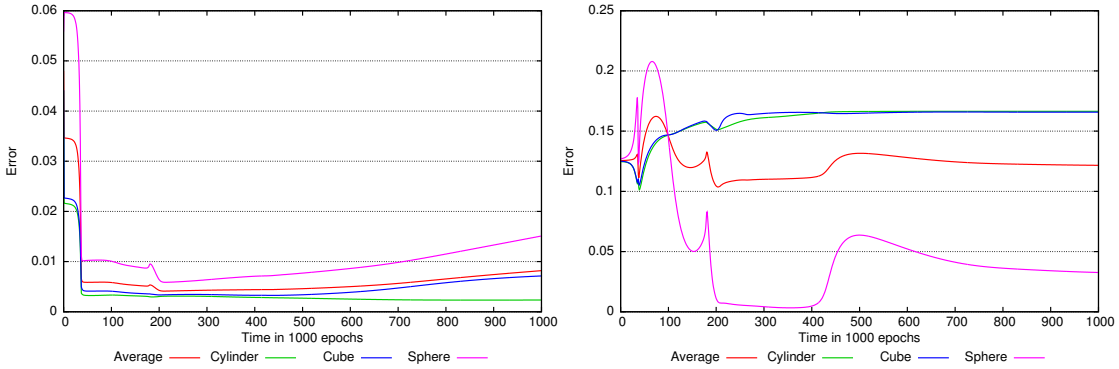
Figure 4.10: Error development for the network trained with the 60-sample-files and 20 hidden neurons.

The networks with higher counts of hidden neurons and training with the 15-sample training files showed different behaviour, and are therefore put into group 4. This group deviates from the others much more. We have given the error graphs for the 20 hidden neuron and 30 hidden neuron networks in figures 4.14 and 4.15, respectively. In these networks, the neurons ling 1 and 2 produced different outputs, instead of almost equal outputs as in all other networks tested. Six networks fall in this category. In all networks in this group, the output for the cylinder was higher than the output for the cube, resulting in low linguistic error values in the cylinder situation and high errors for the cube situation (figure 4.16. The identification errors in the sphere situation, which were the highest in all other networks, were comparably low. The prediction errors are low



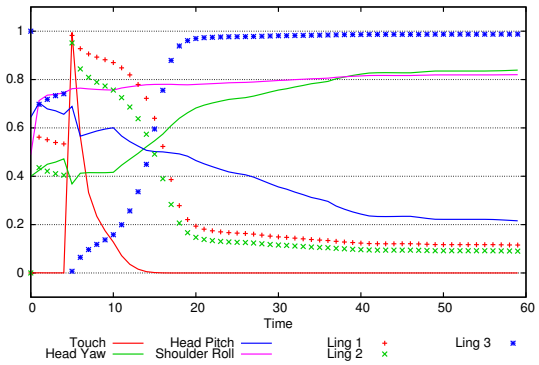
(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.11: Error development for the network with 30 hidden neurons.



(a) Sensorimotoric Error (b) Linguistic Error

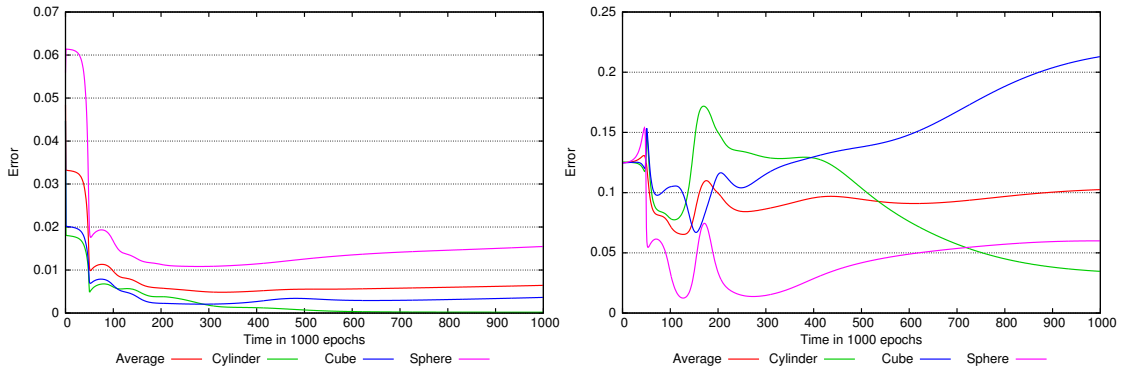
Figure 4.12: Error development for the network trained with the 60-sample-files, 30 hidden neurons



(a) Sphere, 3 million epochs

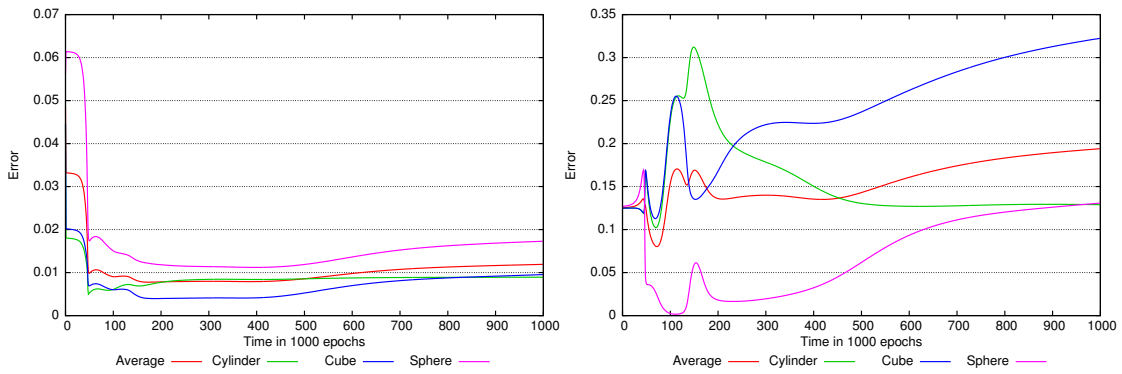
Figure 4.13: Testing outputs of the network with 30 hidden neurons that was trained with the 60-sample-files at 3 million epochs.

for both networks in all situations.



(a) Sensorimotoric Error (b) Linguistic Error

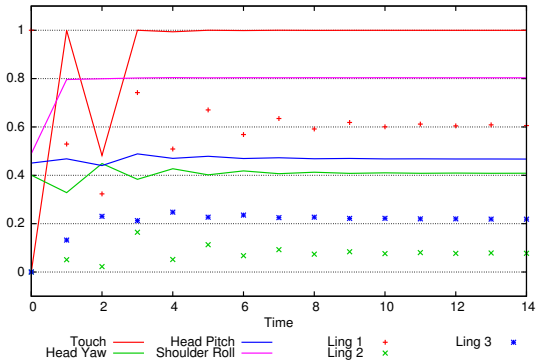
Figure 4.14: Error development for the network with 20 hidden neurons, 15 sample resolution.



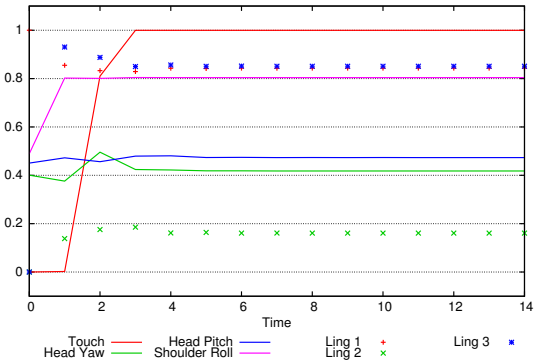
(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.15: Error development for the network trained with the 15-sample-files, 30 hidden neurons

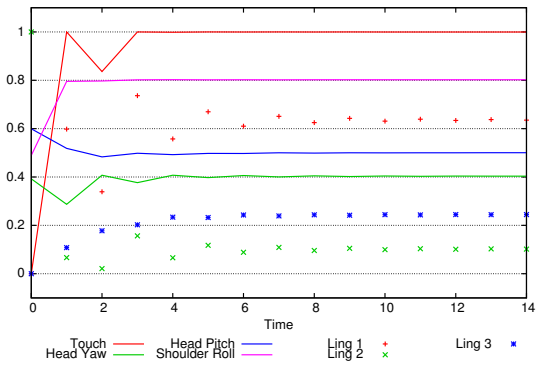
We additionally present three sphere situation output graphs for a network trained with 60 sample resolution, 30 hidden units and a dynamically computed learning rate in figure 4.17. This network is the representative for group 2 that was already mentioned in section 4.3. In these figures, one point in sample time can be seen at which the network changes its outputs very quickly. Prior to this point, the linguistic outputs show the aforementioned trend towards the output [1 1 0], and the prediction of the touch sensor is of mediocre accuracy. After that point, the outputs are more precise for both the linguistic and the touch sensor neurons. During training, that "decision point" moves forward in time, so that the network gives the "wrong" outputs for a longer time. When this point moves past $\tau = 60$, the error in the linguistic neuron 3 rises quickly (see figure 4.18). Also does the sensorimotoric error for this situation continually increase as the touch sensor neuron decreases in performance. The outcome is very similar to those



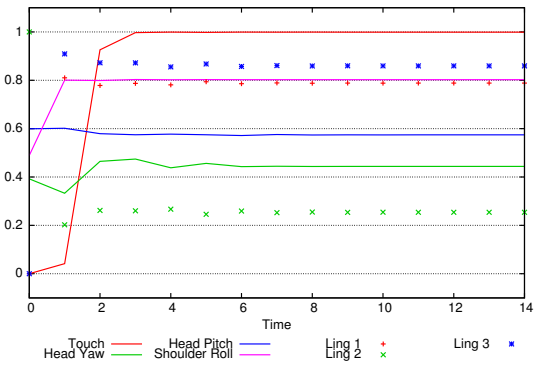
(a) Cylinder, 1 million epochs



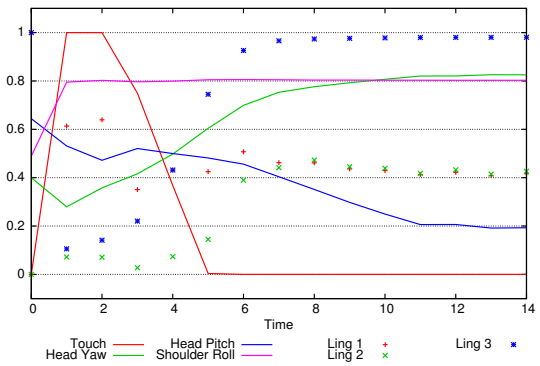
(b) Cylinder, 10 million epochs



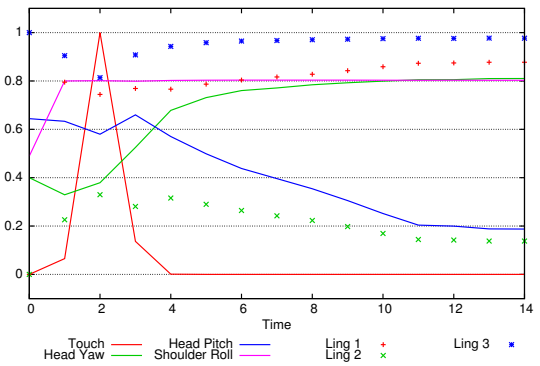
(c) Cube, 1 million epochs



(d) Cube, 10 million epochs



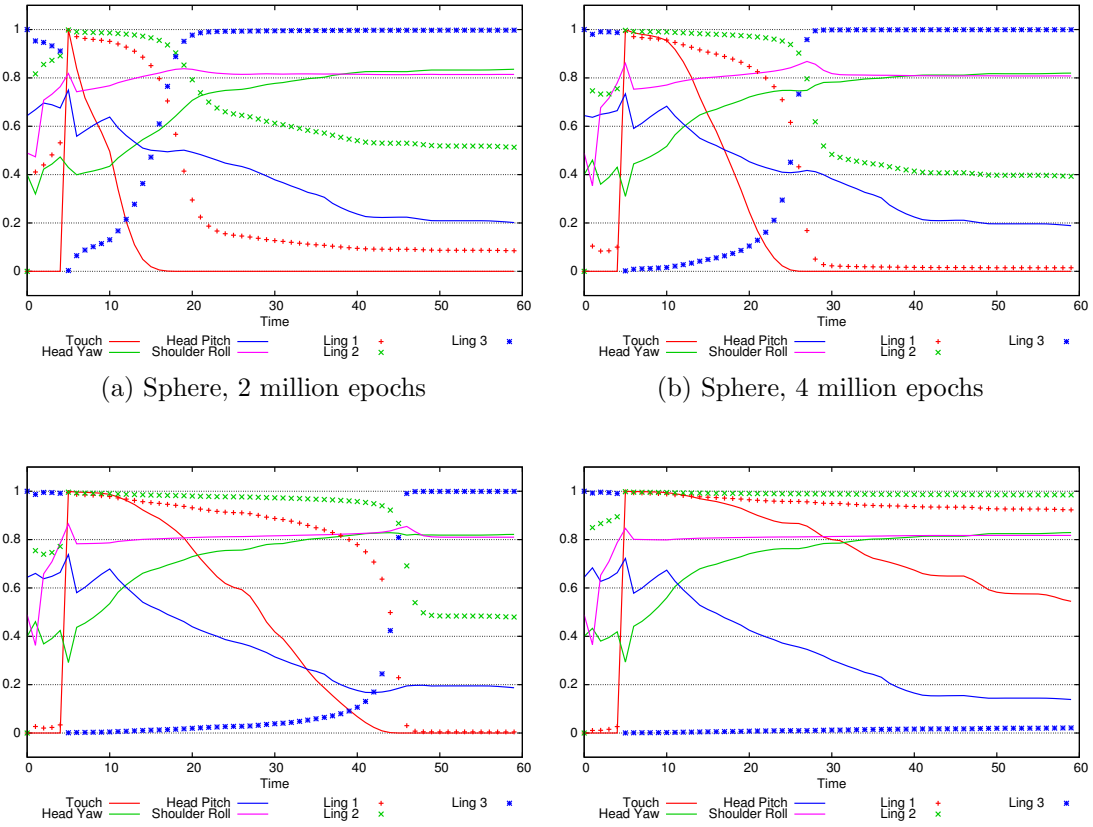
(e) Sphere, 1 million epochs



(f) Sphere, 10 million epochs

Figure 4.16: Testing outputs of the networks with 20 and 30 hidden neurons, trained with the 15-samples training files.

networks in group 1, the process, however, seems to be somewhat characteristic, so that we decided to put these three networks into a separate group.



(a) Sphere, 2 million epochs

(b) Sphere, 4 million epochs

(c) Sphere, 6 million epochs

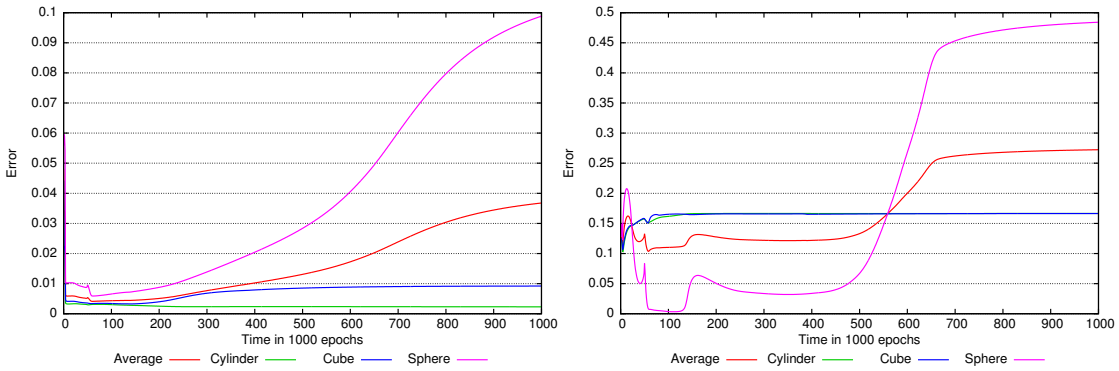
(d) Sphere, 8 million epochs

Figure 4.17: Testing outputs of the network that was trained with the 60-sample-files, 30 hidden neurons and a dynamic learning rate at 2, 4, 6, and 8 million epochs.

4.5 Training results: Using different training files

Instead of training with only the three deterministic samples recorded, we also trained three networks with the initial configuration, but with the learning rates 0.01, 0.005 and 0.002, with training files that consisted of each ten of the 100 generated samples in which the object was randomly placed with in a 10cm x 10cm area. Training networks with these files, however, takes ten times more computation time, as they contain ten times as many samples[1], which is why we chose to test only the very basic configuration.

[1]The networks were computed on a 4x3.60GHz Macintosh. A network of basic configuration took approximately 7 hours to be trained for 10 million epochs. Apart from the training files does also raising the number of hidden neurons increase the computation time. The time complexity class of the epochwise backpropagation through time algortihm is $|P| \cdot o(n^2)$ with n being the number of neurons in the network and P the set of trained patterns [28].

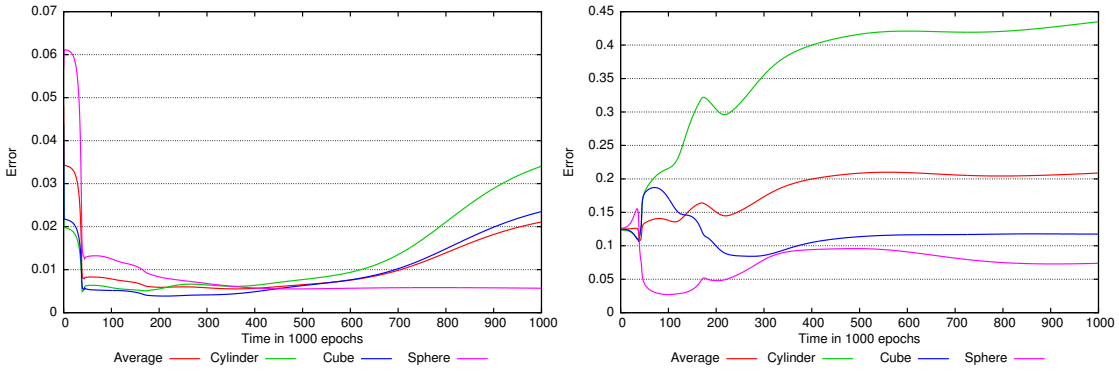


(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.18: Error development for the network trained with the 60-sample-files, 30 hidden neurons and dynamic learning rate.

Refer to figure 4.19 to see the error development of the network using the initial configuration, but the other training file. They show behaviour very similar to those networks in group four. However cube and cylinder outputs have switched their roles: In the networks with higher hidden neuron counts and 15-sample resolution files, the errors for the cube situation were very high, and those for the cylinder low.

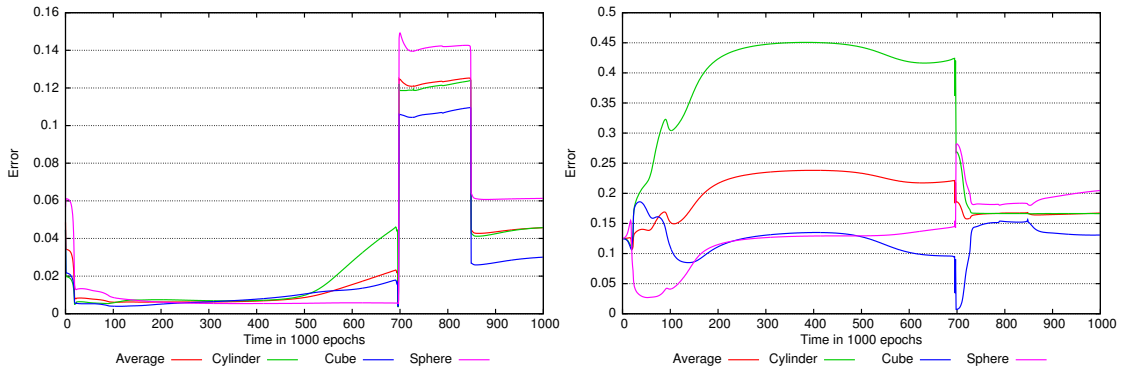
Lowering the learning rate had no effect on these results except making them take longer.



(a) Sensorimotoric Error (b) Linguistic Error

Figure 4.19: Error development for the network trained random placement training files.

Raising the learning rate to 0.01 lead to the network suffering from over-fitting of the shoulder and head pitch neurons and the neurons ling 1 and 3 (figure 4.20). The shoulder roll neuron recovered later in training.



(a) Sensorimotoric Error

(b) Linguistic Error

Figure 4.20: Error development for the network trained with random placement training files and a learning rate of 0.01.

4.6 Summary

It is possible to categorize the networks into four groups. Of the 25 different networks we tested, 8 were classified into group 1, 3 into group 2, 5 into group three, 6 into group four and one showed output that had characteristics of both groups 1 and 3. Two suffered from over-fitting. Groups 1 to 3 only differ in very specific characteristics and some networks show characteristics of several of these groups, while group 4 networks show entirely different behaviour.

Throughout all groups and situations, the head yaw, head pitch and shoulder roll values develop in the way that one would expect from training with backpropagation through time. The prediction of these values continually increases in quality. Each situation is followed very well including the unique characteristics that differ from other situation only very little (as with the cube and cylinder situations).

In groups 1 to 3, the linguistic neurons for the cube and the cylinder send nearly equal values in both the cube and cylinder situations. Consequently, the error graphs for both situations are almost equivalent. Linguistic neuron 3 tends to produce low outputs after longer training, which results in a particularly high error value in the sphere situation. The touch sensor neuron produces accurate results early in training, but sinks in performance later when it begins to give high outputs in the sphere situation, too. These two developments in the sphere situation, the degradation of performances in touch sensor prediction and classification by the linguistic neurons, are the characteristic properties of networks in group 1 to 3. In group 1, this development happens comparably smoothly. In group 2, it appears later in time and in the form of the characteristic "moving decision point" scheme. in group 3 networks, these developments come with very strong oscillations.

The networks in group 4, unlike the other networks, produce a different output for the cube and cylinder neurons. However do they not classify these two situations correctly, but instead one neuron gives high outputs in both situations, so that one is correctly classified, the other wrongly. The late degradation of touch sensor and ling 3 performance in the sphere situation does not occur. Instead, the touch sensor seems to decrease in performance in the other two situations in which it gives continually lower outputs.

Chapter 5

Conclusion

In chapter 3, we have posed the question whether the approach to ground language in sensorimotoric experience of a robot, as it was proposed by Marocco et al. (2010) [11], can be used on a different simulated robot or even in reality. Also did we ask which parameters influence this stability.

We re-conducted the experiments of Marocco et al. with a different robot and under variation of those parameters that define the network architecture and training process, and found that none of our experiments gave the same results. By observation of the training processes we could document that the networks did not converge towards an optimal solution for either the task of predicting the sequences or the task of classifying and naming them. Variations in the sample resolution and choosing a training file that contains more heterogeneous data did not considerably improve, but alter the training outcome in a way that lets us conclude that these parameters are more crucial than the learning rate or the number of hidden neurons.

5.1 Observations in the results

In the previous chapter, we have found that the networks can be divided into four groups. It is possible to identify which parameters determine what sort of output the resulting network will produce. With some exceptions, the group 1 networks all have low numbers of hidden neurons and were trained with the deterministic training files. Group 2 networks all were trained with the 60 sample resolution files. The networks in group 3 all have high hidden neuron counts. The group 4 networks were all trained either with 15 sample resolution deterministic or 30 sample resolution random placement files.

Groups 1 to 3 all give the same output for the cube and cylinder situation. The output in the sphere situation may differ, however can we observe that almost every time the sphere situation has high linguistic and sensorimotoric error values at the end of the training. This is due to the decreasing performances of the touch sensor neuron and the linguistic neuron 3.

Group 4 networks give different linguistic outputs in the cube and cylinder neurons. They do not categorize the situation correctly, but instead one of the neurons gives high outputs for both situations. However, the fact that the distinction occurs at all is a great improvement over the networks in the other groups. Which neuron gives the high output differs across the networks in the group.

5.2 Considerations of the training data

By observing the training samples for the cube and the cylinder situation, we find that these samples are very similar to each other. Also is each of them very static, so that the trained input and target output values are the same for most of the time. This explains why the cube and the cylinder neurons give the same output so often. In each of these patterns, the teacher forcing drives one neuron high. However, the differences between these sequences and between the input/output pairs they form are so small that no distinctive features can be identified. The identification of distinctive features could then lead to the network developing structures that give strong and decisive reactions to these features. Instead it seems like the very same pattern was presented to the network over and over, sometimes while driving one neuron high, sometimes the other. This also explains why the performance in the sphere situation decreases. The majority of patterns supports the development of connection configurations that give the input as an output. So once the output for the sphere situation is low, the network trained with these patterns will continue to give low outputs. The same applies to the high touch sensor value. Whether that output will be constantly low or constantly high also is determined from the set of patterns: In approximately two thirds of them, the cube and the cylinder sequences, the touch sensor is high and the target output for the sphere neuron is low.

When we compare our data with that given in [11], we find that their data makes much more use of the range between 0 and 1 that neurons can compute. For our experiments, we considered the whole possible range for a value, like the physical limits of the head yaw joint of the NAO robot, and linearly mapped that range to the interval $[0, 1]$. As the movement did not use the whole possible range of the joints, our samples did not use the whole range of this interval and appear somewhat "squashed" compared to the data in [11]. This, in turn, makes the individual time steps in and across sequences harder to distinguish, as they lack distinctive and obvious features.

Of our samples, the 15 sample resolution files and the samples with random object placement were the ones with the least uniform data. In the former case, the low resolution led to higher differences between time steps, in the latter the random placement introduced some entropy. Combining these considerations and the observations on group 4 in the previous section, we can assume that there exists some connection between the usage of high entropy training files and the difference in training outcome. We can also observe from the distribution of the group affiliations that the other parameters, the learning rate and the number of hidden neurons, did not determine the group membership to the same extent as the training file resolution and recording method did. The fact that the cube and cylinder neurons switch roles in the different networks shows that *which* neuron assumes this role is not completely deterministic within our range of parameters and training files.

5.3 Technical conclusions

While these observations explain the results, they do not give any hint for an obvious technical solution that would help using the proposed mechanism in a reliable way. The results indicate that the data that is used for training has to comply to certain yet to exactly determine requirements. However, our records resemble data as it would be

generated from real world robot experiences. They were purposefully recorded with very simple methods to reflect the quality of data as it might be recorded on cheap hardware and with little effort. For universal applicability, an approach to grounding language like this one would need to be able to be used even with poor quality data and on different hardware. Choosing only that data that shows the required variance, like sequences that describe fast moving objects, is not an optimal solution. It is desirable to make it possible to use the approach with data that is possibly sparse on distinctive features. Our results show that the approach is not yet understood and explored well enough to employ it under these conditions.

5.4 Open questions

From the results, it was possible to conclude that the lack of entropy in our training data is the central reason for why we were not able to reproduce the results of Marocco et al. (2010) [11]. Variations of the number of hidden neurons and the learning rate did not have as much of an impact. Therefore, further research could explore the effects of varying training data entropy on the stability of the proposed method. It is open to research to determine whether the differences between time steps or the differences between individual sequences are more crucial. The variety in the training data may include:

- Varying the recording time. We used 4.8 seconds throughout all of our experiments.
- Varying the resolution. We used only three different resolutions.
- Differences between sample classes. In our experiments, the cube and cylinder samples were particularly similar.
- Differences between samples in the same class. This can include the random placement of the object, but also varying size, color (which would influence the head tracker performance), etc.

Each of these factors could be explored as to its impact on the applicability of the grounding mechanism. Varying the number of hidden neurons may result in valuable results once the bounds for those more crucial parameters have been found.

Apart from that, research might also try to find methods and techniques that can be used to further stabilize this approach. We already mentioned in section 5.3 that it would be desirable to use it to ground symbols in data that does not have optimal properties. If such methods could be developed, it would become possible to employ the proposed method on a broader range of hardware platforms and applications.

Bibliography

[1] Lawrence W Barsalou. Grounded cognition. *Annual Review of Psychology*, 59(1):617–645, 2008.

[2] Angelo Cangelosi and Thomas Riga. An embodied model for sensorimotor grounding and grounding transfer: Experiments with epigenetic robots. *Cognitive Science*, 30(4):673–689, 2006.

[3] Ferdinand de Saussure. *Cours de Linguistique Générale*. Payot, 1995.

[4] Jerry A. Fodor. *The Language of Thought*. Thomas Y. Crowell Co., New York, 1975.

[5] Vittorio Gallese and George Lakoff. The brain's concepts: The role of the sensory-motor system in conceptual knowledge. *Cognitive Neuropsychology*, 22(3):455–479, 2005.

[6] Herbert S. Gasser and Joseph Erlanger. The role played by the sizes of the constituent fibers of a nerve trunk in determining the form of its action potential wave. *American Journal of Physiology*, 80(3):522–547, 1927.

[7] Arthur M. Glenberg and Michael P. Kaschak. Grounding language in action. *Psychonomic Bulletin & Review*, 9:558–565, 2002.

[8] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42:335–346, 1990.

[9] Simon Haykin. *Neural Networks and Learning Machines*. Prentice Hall International, 2008.

[10] Stephen Laurence and Eric Margolis. *Concepts and Cognitive Science*, chapter 1, pages 3–81. Bradford Books/MIT Press, 1999.

[11] Davide Marocco, Angelo Cangelosi, Kerstin Fischer, and Tony Belpaeme. Grounding action words in the sensorimotor interaction with the world: experiments with a simulated icub humanoid robot. *Frontiers in Neurorobotics*, 4:1–15, 2010.

[12] Charles W. Morris. *Foundations of the Theory of Signs*. University of Chicago Press, Chicago, IL, 1st edition, 1938.

[13] Allen Newell. Physical symbol systems. *Cognitive Science*, 4(2):135–183, 1980.

[14] Nicola Nosengo. The bot that plays ball. *Nature*, 460(August):1076–1078, 2009.

[15] Charles K. Ogden and Ivor A. Richards. *The Meaning of Meaning: A Study of the Influence of Language Upon Thought and of the Science of Symbolism.* Trübner & Co, 1923.

[16] Anne J. Olmstead, Navin Viswanathan, Karen A. Aicher, and Carol A. Fowler. Sentence comprehension affects the dynamics of bimanual coordination: Implications for embodied cognition. *The Quarterly Journal of Experimental Psychology*, 62(12):2409–2417, 2009.

[17] Diane Pecher and Rolf A. Zwaan. *Grounding Cognition: The Role of Perception and Action in Memory, Language, and Thinking.* Cambridge University Press, 2005.

[18] Charles S. Peirce. *Collected Papers of Charles Sanders Peirce.* Harvard University Press, Cambridge, MA., vols. 1-6 1931–1935, vols. 7-8 1958.

[19] Zenon W. Pylyshyn. *Computation and cognition: toward a foundation for cognitive science.* Massachusetts Institute of Technology, Cambridge, MA, USA, 1984.

[20] Aldebaran Robotics. Nao hardware platform. Website, Nov. 2011.

[21] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[22] John Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3:417–424, 1980.

[23] Mariarosaria Taddeo and Luciano Floridi. Solving the symbol grounding problem: a critical review of fifteen years of research. *Journal of Experimental and Theoretical Artificial Intelligence*, 17:419–445, 2005.

[24] Vadim Tikhanoff, Paul Fitzpatrick, Francesco Nori, Lorenz Natale, Giorgio Metta, and Angelo Cangelosi. The icub humanoid robot simulator. *Advanced Robotics*, 1(1):50, 2008.

[25] Ricardo A. Téllez and Cecilio Angulo. Webots simulator 5.1.7. developed and supported by cyberbotics ltd. (2006). *Artificial Life*, 13(3):313–318, 2007.

[26] Paul J. Werbos. *Beyond Regression: New Tools for Prediciton and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[27] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550 –1560, oct 1990.

[28] Ronald J. Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. Technical report, Hillsdale, NJ, USA, 1995.

Erklärung der Urheberschaft

Ich versichere an Eides statt, dass ich die vorliegende Bachelor thesis selbstständig und ohne unerlaubte Hilfe Dritter angefertigt habe. Alle Stellen, die inhaltlich oder wörtlich aus anderen Veröffentlichungen stammen, sind kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher noch nicht veröffentlicht.

Ort, Datum Unterschrift