FAKULTÄT FÜR
ELEKTROTECHNIK UND
INFORMATIONSTECHNIK

INSTITUT FÜR INFORMATIONS- UND KOMMUNIKATIONSTECHNIK (IIKT)

Implicit Sequence Learning in Recurrent Neural Networks

DISSERTATION

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

von

Dipl.-Ing. Stefan Glüge

geb. am 16.07.1982 in Magdeburg, Deutschland

genehmigt durch die
Fakultät für Elektrotechnik und Informationstechnik
der Otto-von-Guericke-Universität Magdeburg

Gutachter: Prof. Dr. rer. nat. Andreas Wendemuth
Prof. Dr. Günther Palm
Jun.-Prof. PD. Dr.-Ing. habil. Ayoub Al-Hamadi

Promotionskolloquium am 11.10.2013

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die Hilfe eines kommerziellen Promotionsberaters habe ich nicht in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen fr Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Verwendete fremde und eigene Quellen sind als solche kenntlich gemacht.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann.

Ich erkläre mich damit einverstanden, dass die Dissertation ggf. mit Mitteln der elektronischen Datenverarbeitung auf Plagiate überprüft werden kann.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 26.06.2013

Dipl.-Ing. Stefan Glüge

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich in den letzten Jahren bei der Arbeit an meiner Dissertation unterstützt haben.

Den größten Anteil hat sicherlich Prof. Dr. Andreas Wendemuth, der mich wissenschaftlich betreut hat. Zusätzlich zu den Ideen und Anregungen bezüglich der wissenschaftlichen Fragen, hat er mit seiner offenen Art für ein Klima gesorgt, in dem man gern arbeitet. Prof. Dr. Günther Palm danke ich vor allem für die Bereitschaft, trotz der Bergen von Arbeit auf seinem Schreibtisch, meine Dissertation zu begutachten. Dasselbe gilt für Prof. Dr. Ayoub Al-Hamadi, der sich ebenfalls bereit erklärt hat als Gutachter zu fungieren.

Des Weiteren danke ich meinen Kollegen am Lehrstuhl für Kognitive Systeme an der Otto-von-Guericke Universität. Besonders erwähnen möchte ich die tolle Zusammenarbeit und die vielen interessanten Diskussionen und Projekte mit Ronald Böck.

Der Otto-von-Guericke Universität und dem Land Sachsen-Anhalt danke ich für die finanzielle Unterstützung während der Promotion.

Letztendlich gilt mein liebster und innigster Dank meiner Frau Jule und meiner Familie, die immer an mich glauben. Ohne ihre Unterstützung wäre diese Arbeit nicht möglich gewesen.

Abstract

This thesis investigates algorithmic models of implicit learning, and presents new methods and related experiments in this field.

Implicit learning is a common method of acquiring knowledge and therefore happens in childhood development and in everyday life. It can be shortly defined as incidental learning without awareness of the learned matter. As this is a highly desirable feature in machine learning, it has many applications in computational neuroscience, engineering applications and data analysis.

The first part of this thesis is focused on cognitive modelling of implicit sequence learning as it was observed in behavioural experiments with human subjects. The experimental setup is known as *conditional associative learning* scenario. Insights gained in this process are then used in the second part of this work. Here, the implicit learning of sequential information by recurrent neural networks is investigated in the context of machine learning.

For cognitive modelling a Markov model is used to analyse the *explicit* part of the associative learning task which was given to the subjects. Thereafter, *simple recurrent networks* are applied to model the *implicit* learning of temporal dependencies that occurred in the experiments. Therefore, the development and storage of representations of temporal context in the networks is further investigated.

Recurrent networks are a common tool in cognitive modelling, but even more an important method in the machine learning domain. Whenever it comes to sequence processing the capability of these networks is of great interest. One particular problem in that area of research is the learning of long-term dependencies, which can be traced back to the problem of *vanishing error gradients* in gradient based learning. In my thesis I investigate the capabilities of a hierarchical recurrent network architecture, the *Segmented-Memory Recurrent Neural Network*, to circumvent this problem. The architecture itself is inspired by the process of memorisation of long sequences observed in humans. An extended version of a common learning algorithm adapted to this architecture is introduced and compared to an existing one concerning computational complexity and learning capability. Further, an unsupervised pre-training procedure for the recurrent networks is introduced that is motivated by the research in the field of deep learning.

The learning algorithm proposed in this thesis dramatically reduces the computational complexity of the network training. This advantage is paid with a reduction of the time span between inputs and outputs that my be bridged by the network. However, this loss can be compensated by the application of a pre-training procedure.

In general, recurrent networks are of interest in cognitive modelling, but in fact, this leads to their application in rather technical sequence classification and prediction tasks. This work shows, how those networks learn task-irrelevant temporal dependencies implicitly, and presents progress which is made to make them applicable to machine learning and information engineering problems.

Kurzfassung

Diese Arbeit untersucht algorithmische Modelle des impliziten Lernens und präsentiert neue Methoden, sowie Experimente, auf diesem Feld.

Implizites Lernen ist ein Weg des Wissenserwerbs und passiert im alltäglichen Leben, vor allem in der Kindheit. Es wird kurz als zufälliges Lernen, ohne Bewusstsein für das Gelernte, definiert. Diese Eigenschaft ist auch für technische Systeme interessant und findet unter anderem Anwendung in der Neuroinformatik, den Ingenieurswissenschaften und im Bereich der Datenanalyse.

Der erste Teil dieser Arbeit befasst sich mit der kognitiven Modellierung von implizitem Lernen von Sequenzen, wie es in einem Verhaltensexperiment mit Versuchspersonen beobachtet wurde. Derartige Experimente werden in der Kognitionsbiologie zur Untersuchung von *konditionellem assoziativen Lernen* genutzt. Die durch die Modellierung gewonnen Einsichten, werden im zweiten Teil der Arbeit weiter verwendet. Dort wird, im Kontext des maschinellen Lernens, das implizite Lernen von Zeitabhängigkeiten durch rekurrente neuronale Netze analysiert.

Um den *expliziten* Teil der assoziativen Lernaufgabe zu analysieren, wird ein Markov Modell vorgeschlagen. Danach werden einfache rekurrente Netze (*simple recurrent networks*) genutzt, um den *impliziten* Teil des Lernens von Zeitabhängigkeiten, wie sie im Experiment auftraten, zu modellieren. Hierzu wird vor allem die Entwicklung und Speicherung von Repräsentationen des zeitlichen Kontextes in den Netzen untersucht.

Rekurrente Netze sind ein gutes Werkzeug für die Modellierung kognitiver Prozesse, aber ein ebenso wichtiges Werkzeug im Bereich des maschinellen Lernens. Vor allem beim Verarbeiten von Informationssequenzen sind die Eigenschaften dieser Netze von großem Interesse. Ein bedeutendes Problem in dieser Domäne ist das Lernen von Langzeitabhängigkeiten, welches auf das Abnehmen der Fehlergradienten beim gradientenbasierten Lernen (*vanishing gradient problem*) zurückgeführt werden kann. In meiner Arbeit untersuche ich, in wie weit ein hierarchisches rekurrentes Netz (*Segmented-Memory Recurrent Neural Network*) das vanishing gradient problem umgehen kann. Die Netzarchitektur ist inspiriert durch die Art und weise, wie sich Menschen längere Sequenzen merken. Für die Architektur wird eine erweiterte Version eines bekannten Lernalgorithmus vorgeschlagen (BPTT), und bezüglich des Rechenaufwandes und der Fähigkeit Langzeitabhängigkeiten zu lernen untersucht. Außerdem wird ein nicht überwachtes Vortraining beschrieben, welches durch die Forschung im Bereich der tiefen neuronalen Netze (deep learning) inspiriert ist.

Der in dieser Arbeit vorgeschlagene Lernalgorithmus verringert den Rechenaufwand für das Netzwerktraining erheblich. Die Zeitspanne zwischen Ein- und Ausgangssignalen des Netzes, die überbrückt werden kann, ist jedoch kleiner als bei einem etablierten Algorithmus. Dieser Nachteil kann weitestgehend durch das Vortraining kompensiert werden.

Im Allgemeinen haben rekurrente Netze interessante Eigenschaft für kognitive Modelle, aber im Speziellen werden diese oft in technischen Sequenzverarbeitungsaufgaben angewandt. Diese Arbeit zeigt, wie die Netze zeitliche Zusammenhänge implizit lernen und trägt dazu bei, sie für technische Anwendungen nutzbar zu machen.

Contents

1 Introduction

From our own experience we know that learning may happen unintentionally and unconsciously: for instance, after learning how to ride a bicycle it is hard to explain to somebody how to keep balance when riding it, nonetheless oneself simply 'knows' how it works. Often the same issue arises when a native speaker shall explain a certain phrase or statement to a non-native speaker. For the native speaker the phrase just sounds right without having a detailed description why. Another example is the ability to walk. Usually people learn it in the early childhood and just know how to do it. It seems impossible to give instructions to somebody on 'how to walk'. Psychological research on this phenomenon can be subsumed under the term *implicit learning*, and shortly defined as: incidental learning without awareness of the learnt matter (Cleeremans, 1993).

The first half of this thesis deals with questions concerning implicit learning as they arise form studies in experimental psychology and cognitive biology. I focus on implicit sequence learning like it is observed in humans in the *conditional associative learning* scenario. That is, subjects implicitly learn temporal dependencies even though they are task-irrelevant. *Simple recurrent networks* turned out to be a good tool to model such behaviour. After a discussion of modelling aspects I investigate the question, how temporal context is represented in these kind of networks.

Based on the findings in the modelling of implicit learning in humans, the second part of my work deals with recurrent networks in the context of machine learning. To utilize their implicit learning capabilities for technical tasks two main problems have to be solved. First, the problem of learning long-term dependencies, particularly the *vanishing gradient problem*, has to be circumvented. Second, computational feasible training methods have to be developed.

The first point is tackled by the usage of an enhanced network architecture, namely *Segmented-Memory Recurrent Neural Network*. The architecture is inspired by the process of memorisation of long sequences, as it is observed in humans. It is a stack of two simple recurrent networks with a segmentation of the outputs of the first stage, before it is processed further in the second stage.

Regarding the problem of the computational complexity of the network training I introduce an extended version of a common algorithm adapted to this hierarchical network architecture. It is compared to an established algorithm concerning its ability to learn long-term dependencies on a benchmark problem. Further, I analytically derive the computational complexity of both algorithms to study their usability for real world applications, where considerably large networks might be used. Thereafter I apply a layer-local unsupervised pre-training procedure prior to the actual supervised training and evaluate it on the benchmark problem.

Cognitive Modelling

The first part of my thesis deals with cognitive modelling of implicit learning. As this discipline is rather uncommon in the engineering community, I want to motivate and introduce the basic concepts. An extensive overview of the field of cognitive modelling is given in the first chapter of Cooper (2002). I summarise the main points here to establish a link to the topics covered in my thesis.

Cognitive modelling is the creation of computational models of (mostly) human cognition. The fundamental idea is that the development of computer models can further our understanding of those processes by allowing us to evaluate computational mechanisms that underlie behaviour. Therefore, a computer model in cognitive science is an abstract representation of a cognitive process. The main use of such models is to simulate and predict human behaviour.

The modern era of cognitive science dates back to the end of the 19th century. Early attempts in empirical psychology were based on introspection. For instance, Ebbinghaus et al. (1913) studied the processes of memory by learning lists of nonsense words. Such studies soon were criticised as being subjective and non-scientific. This rejection of introspection was accompanied by the rise of behaviourism which dominated the psychology for the first half of the 20th century. Behaviourists argued for an objective study of internal mental states by the claim that simple stimulus-response patterns explain all kinds of behaviour.

Then, in the middle of the 20th century, it became accepted that stimulus-response links alone could not explain the full range of human behaviours, for instance, language. Instead, a new picture of cognition evolved that understood the mind as an information processor and cognition as information processing. This view regards sensory processes, such as vision or hearing, as input devices that convert environmental information into internal representations. Mental processes manipulate and transform these representations, which in turn, may lead to a response.

During the last decades computer simulation techniques were adopted to evaluate competing theories of cognitive processing on empirical phenomena. Such computational modelling, and the simulation, is one of the distinguishing features of cognitive science.

As in any scientific domain, modelling provides a way of investigating the rules that govern a complex system that yet is not understood. Further, the simulation is the basic method of studying the model's characteristics. In cognitive psychology three different aspects of cognition are distinguished: behaviour, processes underlying behaviour, and theories of those processes. A model generates behaviour according to an implemented theory and simulates cognitive processes. By that, modelling serves an important role in cognitive science.

While few theories within cognitive psychology are stated in a specific manner, modelling forces precision, because it requires the theory to be computationally complete. Thereby, a computational model can be an elegant expression of a theory in objective terms. Compared to verbal or diagrammatic theory specification, a model, described in a computer language, is not open to interpretation. In some cases the formal analysis

of a model's properties allows the derivation of consequences from theoretical assumptions, even without running a simulation. Further, modelling facilitates evaluation of theoretical proposals and enables us to investigate the impact of changes in theoretical assumptions on the model's behaviour. Thus, it allows an evaluation and exploration of a theory.

Even though modelling is motivated by these benefits it is not without difficulties. They primarily arise from the need to make detailed assumptions about the representations and processing that are necessary to construct and run a model. Such details may be hard to justify empirically and physically.

However, while those who practice cognitive modelling generally agree on the benefits of this approach, they often disagree about the specific strategy. There are several schools of cognitive modelling, and representatives of one are usually critics of another. They differ in their assumptions about mental representation and the relation between a cognitive model and the brain.

Connectionists argue that the neural tissues implementing information processing mechanisms of the mind are the key to understand the brain's work. In contrast, symbolic cognitive models make the assumption that information processing can be described by the manipulation of symbolic representations. Here the neural substrate is regraded as an implementation of the representations, that is of secondary importance. Both approaches to cognitive modelling share little, except the idea that the functioning of the mind is computational describable and so may be simulated by a machine.

To complete the picture two more approaches to modelling shall be listed here: the architectural approach and the dynamical approach. The former aims at the formulation of a hypothesised organisation of the complete set of information processing structures that comprise the mind. Then, models are developed along the lines of this theory. The dynamical approach emphasises the mathematical nature of cognition. In its most extreme form it denies the existence of mental representations. Instead, it claims that mental processing my be described by differential equations. So the mental processing does not involve solving equations, rather it involves responding to the mental equivalents of forces.

Structure and Research Goals of the Thesis

As discussed above, cognitive modelling provides an approach to further our understanding of cognitive processes in general. The first half of my thesis deals with the modelling of one specific aspect of cognition, the implicit learning of sequential information, and how this supports explicit learning. In particular, simple recurrent networks are applied to model the *implicit* learning of temporal dependencies.

Such recurrent networks are a common tool in cognitive modelling, but further, of interest in engineering applications. Therefore, the second half of my thesis is devoted to the application of recurrent networks in the field of machine learning. There are two main problems that are considered in this context. First, the problem of learning long-term dependencies, and second, the computational complexity of the training algorithms.

So, the research goals of my thesis can be summarised as follows:

- Establish a model of implicit learning as one particular process of human cognition and thereby gain a deeper insight into this process.
- Simple recurrent networks are able to reproduce the effects of implicit sequence learning in humans. The basic mechanisms of this ability shall be investigated.
- The ability of implicit learning in recurrent networks is used to solve rather technical sequence classification problems. Therefore, the effect of vanishing error gradients has to be circumvented.
- Training algorithms that are computational feasible have to be developed.

The following chapters address the different goals and discuss in what respect they are achieved:

Chapter 2 gives a short literature review on research of implicit learning. Therefore, the concept itself, as it is defined in the field of psychology, is introduced. Further, experimental studies, the role of implicit sequence learning, and the use of recurrent networks for modelling aspects are discussed. Subsequently, the conditional associative learning scenario which is used in cognitive biology is introduced. This scenario was developed to study especially the role of implicit learning of a task-irrelevant temporal context. The chapter finishes with an explanatory description of the main types of problems in machine learning, because it is of interest later and shall be specified once. In particular the concepts of supervised, unsupervised, and reinforcement learning are introduced. This is complemented with an overview on sequence learning as it is perceived in the machine learning community.

Chapter 3 presents two different computational models of the processing in the conditional associative learning scenario. Both models use different approaches to the problem and account for different aspects of the learning situation. At first, a Markov model is introduced that aims towards the *explicit* aspects of the learning task. It requires a number of assumptions, but in turn, the analysis of the model makes distinct predictions concerning the influence of model parameters. Those parameters can be controlled by the experimental setup and therefore, may be verified in the laboratory. A connectionist approach is introduced to address the *implicit* learning of task-irrelevant temporal information in that scenario. A simple recurrent network is simulated on the task. It shows qualitatively the same behaviour as the human subjects, which means that it is able to reproduce the effect observed during the behavioural experiment.

Chapter 4 follows the finding that simple recurrent networks are able to reproduce the effects of implicit sequence learning in humans. The focus is put on the investigation of the basic mechanisms of this ability of the networks. Using the example of a simple encoding task, it shows how networks develop representations of implicitly learnt temporal information. Therefore, the influence of the sequential input during training and testing is examined and the networks are tested on various input

sequences.

Chapter 5 turns the focus towards the applicability of recurrent networks in machine learning tasks. The question is whether the ability of implicit learning in such networks can be used to solve certain sequence classification problems. As a matter of fact, simple recurrent networks cannot be used right away. To explain why, I reveal how the idea of a segmented memory helps to attenuate the effect of vanishing gradients that prevents simple recurrent networks from learning long-term dependencies. Segmented-Memory Recurrent Neural Networks were proposed to circumvent the problem of vanishing gradients. The established learning algorithm for this architecture has a very high computational complexity. Therefore, I introduce an alternative algorithm that significantly reduces the computational cost. Concluding, both algorithms are evaluated on the information latching problem concerning their capability of learning long-term dependencies.

Chapter 6 shows how to apply the idea of an unsupervised pre-training from the domain of deep multilayer feed-forward networks to the training of Segmented-Memory Recurrent Neural Networks. This improves the ability to learn long-term dependencies with these networks and extends their area of application.

Chapter 7 summarises and highlights the main results of my thesis. Additionally, topics that yet are not sufficiently examined and remain future work are suggested.

Before ending with the introductory part, I want to point out that the work presented in the following evolved with the help and collaboration of my colleagues. As I am the author, the term "I" is used throughout my thesis. Nevertheless, I do not claim that this work has been done all by myself. The list of publications at the end of the thesis names all the co-authors that contributed to my research. It simply could not be achieved without their support.

2 State of the Art in Implicit Learning

Contents

This chapter gives a survey of the research on *implicit learning* from different perspectives. After a short introduction, an overview on the implicit learning literature is given. Thereafter, the ways implicit learning was studied are presented in more details. Finally, the motivation for a focus on *implicit sequence learning* and connectionist models as proposed by Cleeremans (1993) is given.

The literature on implicit learning dates back to the 1960th, when Arthur S. Reber (1967) explicitly used the term in an article. Today, there exists a large collection of literature on the topic from analytic and applied psychology. Short reviews can be found in Cleeremans et al. (1998), Shanks (2005) and Perruchet (2008). For a comprehensive overview, amongst others, the works of Reber (1993); Underwood (1996); Berry (1997) and Stadler & Frensch (1998) should be referred.

In biology, implicit learning is not regarded to be a phenomenon that occurs on its own, but in connection with associative learning. It is often referred as *context-dependent learning*. A short overview on the biological research according to context-dependent learning and the special relevance of *temporal* context can be found in the first chapter of Hamid (2011). As he states, context condition learning is also known as context conditioning, occasion setting, model-based reversal learning, goal-directed behaviour

and outcome re-valuation/devaluation. This shows that research on the topic is much more scattered from the side of biology. One reason for this diversity may be the fact that the biological research is focused on the explanation of phenomena on the physical level, for instance, which brain regions are involved in learning of that kind. Therefore, the experimental setup has to be very basic, such that measured signals can be linked to the events that occur during the experiment. Further, ethical concerns often prohibit the assignment of human subjects.

On the other hand, behaviour experiments in psychology are much more complex and done with human subjects. The results are not used to explain processes on a physical level, but to set up and form a theory of human cognition.

2.1 Implicit Learning in Psychology

The term *implicit learning* is typically used to characterize those situations where a person learns about the structure of a fairly complex stimulus environment, without necessarily intending to do so, and in such a way that the resulting knowledge is difficult to express (Berry & Dienes, 1993). Or more generally, implicit learning is said to occur when there is an increase in task performance without an accompanying increase in verbal knowledge about how to carry out the task (Underwood, 1996). Frensch & Stadler (1998) list as much as eleven definitions of implicit learning. They basically share the same core ideas:

1. During the learning phase, learning happens incidentally without awareness.
2. The implicit learning results in implicit knowledge, as a form of abstract representations that cannot be verbalised.

Several studies prove that humans make decisions not rationally but based on implicit knowledge. For instance, the statistician who is making a decision that violates bayesian principles or a physician making inappropriate choices in a triage-type setting (Reber, 1993; Kahneman & Tversky, 1982). All this shows that implicit learning is a phenomenon which is hard to define. Psychologists proposed a number of experimental designs to study the implicit learning process. The most relevant shall be discussed shortly.

2.1.1 Empirical Studies of Implicit Learning

It is helpful to have a look on the methods that were used to study implicit learning in order to gain a deeper understanding for the phenomenon. This section shall create a feeling for the tasks that have been used to study implicit learning and outline the main results. For a more comprehensive coverage of the empirical research see Reber (1993); Berry (1994, 1997); Cleeremans (1993).

One has to name at least three paradigms that dominate the psychological literature on implicit learning over the last decades. That is artificial grammar learning, control of complex systems, and sequential pattern acquisition. According to Cleeremans et al. (1998), all of these tasks that were used to study implicit learning involve three components:

1. A complex rule-governed environment to induce incidental learning conditions.

2. A *measure to observe subjects' ability to express acquired knowledge* about the environment through performance on the same or on a different task.

3. A *measure of the amount of consciousness of the knowledge* subjects have acquired.

Grammar Learning

Basically, Arthur S. Reber is regarded as the initiator of the research on implicit learning. His Master's thesis (Reber, 1965) and the article *"Implicit learning of artificial grammar"* (Reber, 1967) was the starting point for a growing interest in learning without awareness.

In experiments, Reber showed that people become sensitive to the constraints of a synthetic grammar. During the learning phase human subjects were shown series of letters generated by a finite state grammar (cf. Figure 2.1). They were told to memorise the letter strings and were not informed about the grammatical nature of the series. A control group simply learnt random strings. In the recall phase subjects were informed about the existence of an underlying grammar and asked to classify a new set of strings being grammatical or not. The set consisted of half grammatical and half ungrammatical samples (cf. Table 2.1).

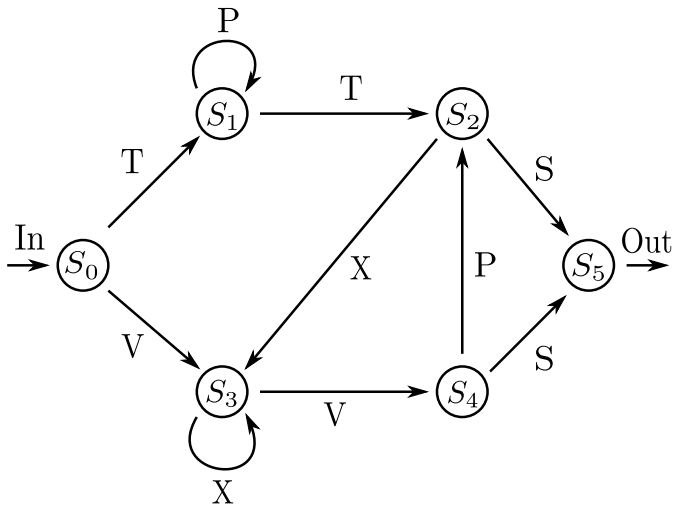


Figure 2.1: Schematic state diagram of the grammar used by Reber (1967). Strings are generated by entering the *In* node and moving from node to node until the *Out* node is reached. Each transition produces the letter linked to the connection between the nodes.

The main results of this study are: (i) subjects who had seen grammatical strings in the learning phase performed significantly above chance level on the classification task, and (ii) subjects were not able to explain how they made their decisions, or what the rules of the underlying grammar might be.

Table 2.1: Possible grammatical strings (left) and ungrammatical strings (right). Grammatical string are produced as described in Figure 2.1. Ungrammatical strings are generated by switching at least one letter to another or adding/removing letters to a grammatical string.

grammatical	ungrammatical
$TPPTS$	$TPPTSS$
$VVPXXVS$	$VXVVPXVS$
$TPTXVPS$	$VTPTXVPS$

As one can see, Reber's experiments yield the core ideas of implicit learning. The effect is very robust and was approved in several subsequent studies by several different authors (Reber, 1976; Brooks, 1978; Dulany et al., 1984; Servan-Schreiber & Anderson, 1990; Berry, 1997).

Another issue in implicit learning research is knowledge transfer. Subjects are able to transfer their (implicitly acquired) knowledge from one setting to another. In Reber (1969) subjects memorized strings from a finite-state grammar and subsequently were asked to memorize new strings which were generated either from the same grammar or from a different one. Further, either the same set of letters was used or a different one. It was found that subjects performed better on the second set of strings than on the first when both sets where generate from the same grammar, even when the set of letters was changed (Cleeremans, 1993).

Complex Process Control

A more complex scenario in empirical research of implicit learning is process control. Berry & Broadbent (1984) investigated the relationship between subjects' performance on the actual learning task and explicit (verbalizable) knowledge. Subjects were asked to control a computer program with the aim to reach and hold several output variables while manipulating one or more input variables. The setting of the task was, for instance, the control of a sugar production factory or a computer simulated person.

The results of these studies showed the same tendency as Rebers' experiments. Practice on the task improved subjects' performance significantly, but did not improve their ability to answer questions on the task afterwards. Further, an instruction on the best way to control the task had no effect on the performance. In other words, subjects were unable to use the explicit knowledge (Cleeremans, 1993).

Again these results were approved in follow-up studies (Berry & Broadbent, 1987; Berry, 1991; Stanley et al., 1989; Dienes & Fahey, 1995; Berry, 1997).

Sequential Pattern Acquisition

Sequence-learning tasks were found to be a useful paradigm to study implicit learning. Basically, it is assumed that subjects' responses reflect their sensitivity to the sequential

properties of the presented stimulus material. Therefore, the reaction time or prediction accuracy is used to measure the acquired knowledge of the subjects. In general, the sensitivity to temporal context is implicit, that is, not verbalizable. Sequence-learning was studied in three different settings: probability-learning tasks, prediction tasks, and serial reaction time tasks (Cleeremans, 1993).

Nissen & Bullemer (1987) introduced the serial reaction time task which was used in many studies in a more or less modified form. The original setup used a light that appeared on one of four positions on a monitor. Subjects should press the one key below the position on the screen where the light appeared. Their reaction time was measured and the sequence of lights was either random or repeated as 10-trail sequence. A significant decrease of reaction time was measured in training with repeating sequences. During random training no change in reaction times was observed. Most of the subjects noticed the repeating sequences and some could describe parts of it. In a later study Willingham et al. (1989) showed that many subjects performed procedural learning of the sequences, without explicit knowledge of it (Berry, 1997).

Prediction tasks use the same idea as serial reaction time tasks. Here subjects are required to predict the next stimulus. This may be a question of *what* the next stimulus will be or *where* it will appear. Usually the percentage of correct predictions is measured. As a result subjects' prediction accuracies improve over training, while they remain unable to specify the underlying rules of the sequence (Kushner et al., 1991; Cleeremans, 1993).

In probability-learning, subjects observe a series of events and then try to reproduce it. Millward & Reber (1968, 1972) showed subjects a sequence of more than hundred two-choice trails that contained increasingly remote contingencies. The sequences were structured, such that certain events were dependent on earlier trails with increasing temporal distance. Subjects learnt to encode the contingencies, as they showed a higher likelihood to produce the contingent event on trails on which it was supposed to occur (Cleeremans, 1993).

2.1.2 The Empirical Problem

The studies described above, as empirical studies on implicit learning in general, share the same elementary problem. In any learning task, subjects' performance arises most likely from both: (i) explicit code-breaking strategies and (ii) the passive, unspecific learning called *implicit learning*. This evident fact makes it hard, for instance, to find unassailable arguments for or against a position concerning the question whether the acquired knowledge is abstract or rather specific (cf. Section 2.1.3).

Following Cleeremans (1993), two factors influence the mixture of learning processes in a particular task, namely *regularity salience* and *task demands*.

Regularity salience, original a multidimensional construct itself, can be considered as stimulus associability. It represents the extent to which stimuli can be combined and related to each other in meaningful ways. The higher the regularity salience, the higher the chance that subjects use explicit problem-solving strategies. On the contrary, subjects presumably use implicit learning strategies in tasks with low regularity salience.

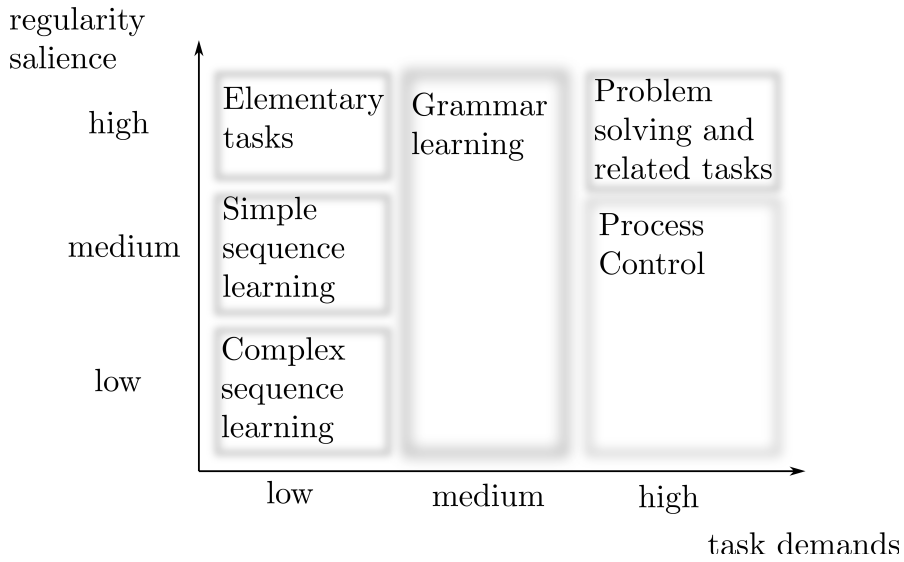


Figure 2.2: Illustration of different learning paradigms in regularity salience - task demands space. Low regularity salience is reported in studies of complex sequence learning (Cleeremans & McClelland, 1991), grammar learning (Reber, 1967), and process control (Berry & Broadbent, 1984). Medium regularity salience is assumed in simple sequence learning (Lewicki et al., 1988), grammar learning (Reber et al., 1980), and process control (Berry & Broadbent, 1988). High regularity salience arises in elementary tasks (Seibel, 1963), grammar learning (Reber et al., 1980), and problem solving (Newell, 1972).

Task demands stand for the complexity to encode stimulus and context, in which it is presented, to perform the learning task. The higher the demands, the more likely subjects use explicit learning strategies.

Figure 2.2 illustrates where different learning paradigms are located in this two dimensional space. It is inspired by Table 1.1 in Cleeremans (1993). Here a coordinate system instead of a table is used to emphasise the continuous character of the dimensions and the fuzzyness of the positioning of the different learning scenarios.

Learning paradigms with low regularity salience and low task demands should be positioned in the lower left part of Figure 2.2, for instance, sequence learning. In such scenarios implicit learning most likely yields a high performance. The upper right area, on the other hand, contains tasks with high demands and high regularity salience, for instance, problem solving. In these tasks it is evident that explicit learning strategies are better qualified to achieve success.

2.1.3 Debates in Implicit Learning

Even though a large number of empirical studies proofs the existence of implicit learning, many claims made by researches in the field are still controversial. In Underwood (1996) a review on three key issues is given, namely: (i) the degree to which the acquired knowledge is really implicit, (ii) the form of appearance of the underlying representations, and (iii) the degree to which the learning is unintentional.

The basic idea and claim of implicit learning is that it leads to knowledge which subjects are not aware of. In many cases, the fact that they are not able to explain their actions is used to proof this claim. The position against this "knowing without awareness of the knowledge" is twofold. Researchers argue that the acquired knowledge is actually explicit, for instance (Dulany et al., 1984; Perruchet & Pacteau, 1990; Shanks & St. John, 1994). Further, it is argued that subjects become aware of the knowledge after some time and/or in an incomplete form (Sanderson, 1989; Dienes et al., 1991). Dulany et al. (1984) modified the recall phase of Reber's grammar learning task. Besides the classification of strings, subjects were asked to underline and/or cross out the crucial parts of it that are ungrammatically. The underlining and crossing out should give an indication of the explicit knowledge used for the decisions. It was found that the rules, deduced from subjects' marks of the key elements in the strings, were sufficient to explain the performance on the classification task. This shows that classification in an artificial grammar task should not be taken as a pure measure of implicit influences. Further, the recognition on the other hand, cannot be assumed to reflect a complete index of conscious knowledge (Cleeremans et al., 1998).

Even those researchers that argue for the idea of implicit knowledge, accept that at least some explicit encoding occurs. For instance, Reber & Lewis (1977) report that subjects with advanced practice on the grammar learning tasks are able to give a far better verbal description of their knowledge than in previous studies. Nevertheless, they highlight the existence of a considerable gap between classification performance and verbal reports. In summary, it appears that the claim for implicit knowledge very much depends on the criterion chosen to measure awareness (Cleeremans et al., 1998).

When it comes to modelling the discussion about the nature of the underlying representations of knowledge arises. According to Cleeremans et al. (1998) early research in that direction described implicit knowledge as *abstract*. It was based on the finding that subjects are able to transfer their knowledge when asked to classify novel letter strings in a grammar learning scenario (Reber, 1967, 1989, 1993). Likewise, it has often been assumed that a decrease in reaction time in sequence-learning reflects a basic knowledge about the underlying rules, which were used in the generation of the stimulus sequences (Lewicki et al., 1987, 1988). The idea of abstract knowledge is unspecific regarding the form of the knowledge, except that it somehow represents the structure of the stimuli and their relationship. On the other hand, there is evidence that a non-abstract approach is sufficient to explain observed transfer of knowledge. The performance of subjects in artificial grammar learning can be explained with the explicit knowledge of grammatical or ungrammatical strings that are partly similar to those shown in the recall phase (Brooks, 1978; Brooks & Vokey, 1991; Vokey & Brooks, 1992). Perruchet & Pacteau (1990) argue

that knowledge acquired in artificial grammar learning and sequence-learning tasks consists of small fragments/chunks explicitly memorised during training. Hence, learning and transfer performance depend on the amount of memorised chunks in the novel material. This mechanism is incompatible to distributed memory models like connectionist models proposed by Dienes (1992) and Cleeremans & McClelland (1991).

The third subject of discussion is the claim, that implicit learning is an automatic, passive process (Reber, 1993; Berry & Dienes, 1991). Given the difficulty of assessing awareness, researchers consider implicit learning as an automatic learning process regardless of the type of resulting knowledge. Instead, the focus shifted on exploring the influence of intention to learn, attention, stimulus complexity, and task demands (Cleeremans et al., 1998). The term *automatic* implies several characteristics. Concerning learning, automatic actions are: developed with extensive practice, performed smoothly and efficiently, resistant to modifications, unaffected by other activities, initiated without intention, and not under conscious control. Yet, there is no decision criterion for categorising an activity as being automatic rather than volitional. To consider implicit learning as effortless it has to happen, at least in parts, automatic (Underwood, 1996). At this point frequency detection has been proposed as an automatic process that underlies implicit learning (Hasher & Zacks, 1984; Wattenmaker, 1993). In several studies it was shown that our knowledge of the frequencies of events is learnt without effort (Hasher & Zacks, 1979; Hintzman, 1969; Zacks et al., 1982).

The discussion above shows that implicit learning is a fundamental process in cognition. Yet, the theoretical development in the field is insufficient to give a generally accepted definition of features for implicit learning. Cleeremans et al. (1998) argue for the need of a better understanding of the nature of consciousness and more sophisticated empirical methods to explore implicit learning. Besides these rather psychological methods, computational modelling may help to shed light on the differences between direct and indirect learning tasks. Further, the field could benefit from functional brain imaging and neuropsychological data to understand the biological fundamentals of implicit learning.

2.1.4 Recent Work in Implicit Learning

Of course, the research on implicit learning is still in progress. Thereby, the original problem is divided into more and more partial aspects, which are investigated in detail by different researchers. This work is only of limited interest for the purpose of my thesis, but should shortly be listed to reveal the general trend during the last years.

One major point of interest is the question what people can or cannot learn without awareness to the learnt matter. It is accepted that complex rules can be learnt (Halford et al., 2005; Lewicki et al., 1992; Nissen & Bullemer, 1987), but these learning processes are goal dependent and require some attention of the subjects (Dijksterhuis & Aarts, 2010). Generally, the load of attention does not influence learning. It is rather the selective attention to crucial information that is needed in order to learn (Jimenez & Mendez, 1999). Further, it was shown that implicit learning is influenced by goals. In an experiment by Eitam et al. (2008) half of the participants were primed with the goal

to achieve, while the other half was not primed. They found that those participants who were primed with achievement performed better than the others, which means, they implicitly learnt more. Nevertheless, the ability to verbalize what they had learnt was equally poor in both conditions (Dijksterhuis & Aarts, 2010).

Another research topic deals with the kind of material that can be learnt implicitly. It is hypothesized that bidirectional structures, like associations, can be learnt automatically, while predictive relations between events, like causal rules, require strategic processing and awareness (Berry & Dienes, 1993; Sloman, 1996). Experiments of Alonso et al. (2006) support this hypothesis. They found that the formation of bidirectional associations can occur without awareness. For the formation of unidirectional relations, that is, structures that capture a predictive relation between events, awareness was needed (Dijksterhuis & Aarts, 2010).

Recent results, published in Custers & Aarts (2011), suggest that conscious awareness is not the critical factor that determines how predictive relations are acquired. It rather needs a process in which attention is directed outside of awareness by processing task-relevant goals (Dijksterhuis & Aarts, 2010).

2.2 Implicit Learning in Cognitive Biology

From a biological point of view, implicit learning is not a term that is loaded with such a fixed meaning as in psychology, where the term itself rises the question for consciousness and awareness. These questions fall in a rather philosophical category that can hardly be approached by biological research. Instead, *context-dependent learning*, a form of *associative learning*, is studied. Compared to implicit learning, as it is perceived in psychology, context-dependent learning shares the idea that learning happens incidentally triggered by the environmental conditions. The question whether this is an unconscious process, and whether the resulting knowledge is verbalizable, is not considered.

Further, biological research is committed to provide explanations for cognitive processes on a physical level. This, not least, includes to find the neural substrate[1] that underlies these specific behaviours. Hence, animal learning is a well established field of study. It allows the investigation of learning processes in organisms of lower complexity, according to the pure number of neural units and connections. Animal experiments with, for instance rats, do not raise as many ethical concerns as experiments with human subjects.

The Encyclopædia Britannica defines 'associative learning' as "any learning process in which a new response becomes associated with a particular stimulus". So, the term refers to learning situations where two different events occur or happen together. 'Context' is defined as a "situation within which something exists or happens, and that can help explain it"[2]. Applied to associative learning, context-dependent learning appears when a stimulus triggers more than one response. In such situations the context, in which the stimulus appears, determines the appropriate response. In animal learning, this may happen under two conditions: (i) the relation between the context and the different meanings of a stimulus should be well defined (Dickinson, 1980; Mackintosh, 1983), and (ii) there should be enough opportunities (time/repetitions) for the context to become associated with the meaning of a stimulus (Hall, 1994). Hence, the context acts very much the same as an additional cue, and is often seen as just another stimulus (Fanselow, 1986; Kiernan et al., 1995; Hamid, 2011)

2.2.1 Sequence Learning and Temporal Context

The introduction of "Sequential Pattern Acquisition" and the discussion on "The Empirical Problem" around Figure 2.2 in Section 2.1.1 already emphasised the role of sequential information in implicit learning. It can be subsumed under the assumption that sequence-learning tasks most likely lead to implicit learning strategies.

In research on context-dependent learning, temporal information was also found to be of great importance (Miyashita, 1988; Hamid et al., 2010). Here, 'temporal context' is "the amount of reward-relevant information provided by the temporal statistics of an environment in terms of the conditional probability for an event to be preceded or

[1] Set of brain structures which may be widely separated anatomically but which interact to support or drive a specific behaviour or psychological state.

[2] from Cambridge Advanced Learners Dictionary

followed by some other event" (Hamid, 2011). Further, it is assumed that incidental learning of consistent sequence information represents a suitable strategy for learning scenarios in which environmental cues may change. This assumption is in coincidence with the idea that frequency detection is an automatic process which underlies implicit learning (Hasher & Zacks, 1984; Wattenmaker, 1993) (cf. Section 2.1.3).

2.2.2 Studies of Temporal Order Effects

The role of temporal context in associative learning was investigated in animal experiments and in studies with human subjects. Hamid (2011) gives a broader overview of this research. I focus only on the main points here, as this is sufficient for the purpose of my thesis. Typically, conditional associative tasks were used in the experiments. A set of visual stimuli is mapped randomly onto a set of motor responses. Then subjects learn by trail and error which response is correct (yields reward) in the case of each stimulus. This means, subjects should learn to link each stimulus to a specific response that ensures reward.

In non-human primates direct evidence for an effect of the temporal order on associative memory comes from electrophysiological recordings. Monkeys were trained to determine whether a sample stimulus matches with a delayed test stimulus. It was found that neurons in the inferior temporal cortex increase their firing rates during the delay interval *selectively* for some of the visual stimuli (Miyashita & Chang, 1988). Further, some neurons in the inferior temporal cortex develop a task-irrelevant selectivity for successive pairs of stimuli when they are shown in a consistent order (Yakovlev et al., 1998). Neuronal selectivity for pairs of different objects that are presented successively was found in the same neurons (Sakai & Miyashita, 1991; Sakai et al., 1994)

For human subjects, behavioural results are consistent with the idea that temporal order shapes associative learning (Blumenfeld et al., 2006; Preminger et al., 2009). For instance, humans lose the ability to distinguish two faces when viewing image sequences in which the face changes as the head rotates (Wallis & Bülthoff, 2001). The correlated appearance over time leads the human observers to assign two different faces to the same person. More generally, temporal order effects are known from psychological experiments as described in Section 2.1.

2.2.3 Task Irrelevant Temporal Context in Conditional Associative Learning

In this section a finer point is put to the experimental part within the conditional associative learning scenario. It is crucial for an understanding of the computational models introduced in the following Chapter of this thesis. The very details of the experiment can be found in Hamid et al. (2010) and Hamid (2011). This also includes a Reinforcement Learning (RL) model fitted to the data gathered in the experiment.

The investigation starts with the question, how temporal context affects the learning of arbitrary visuo-motor associations. Human subjects learnt to associate one of four

buttons (motor responses) to highly distinguishable fractal objects (visual stimuli). The temporal context between objects was manipulated simply by the fact that some objects were consistently preceded by specific other objects. For each object, one response was set to be 'correct' while the remaining three were 'incorrect'. The subjects' task was to learn the 'correct' response for each object by trial and error (cf. Figure 2.3).

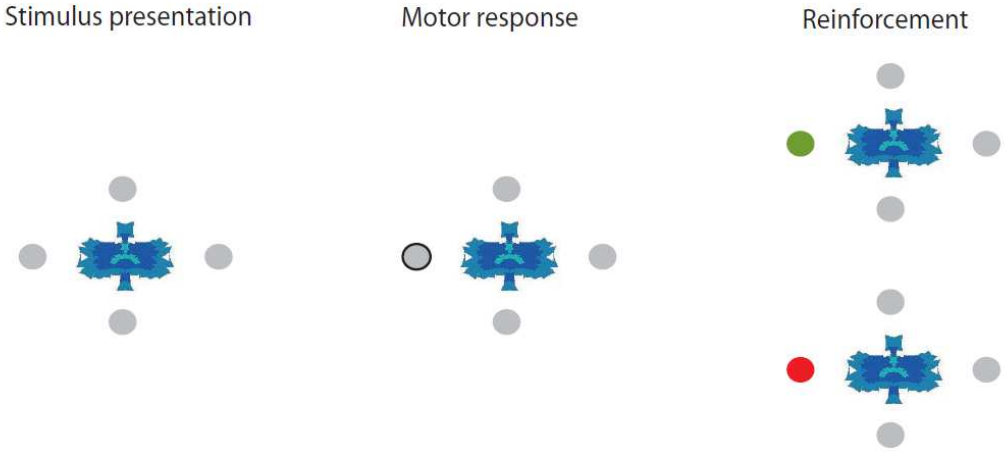


Figure 2.3: Experimental design (schematic): Each trial comprises three phases: stimulus presentation, motor response, and reinforcement. Firstly, a fractal object appears, surrounded by four response options (grey discs). Secondly, the subject reacts by pressing the key that corresponds to one response option (outlined disk). Thirdly, a colour change of the chosen option provides reinforcement (green if correct, red if incorrect) (Hamid et al., 2010).

It was made clear that no pattern or system exists that predicts the required response for a particular object based on its appearance. Further, the sequence in which objects are shown was not mentioned or referred.

Behavioural data was gathered in five experimental sessions. Each session provided the subjects with a different kind of temporal relation between the fractal objects. In the first experiment, sequences of eight objects were divided in two classes, either *deterministic* or *random*. In the deterministic case, the eight objects were repeatedly shown in the same order. By that, preceding objects were just as predictive about the 'correct' response in the current trail as the current object. The random sequence guaranteed that each object followed every other object with equal probability, with the constraint that an object is not immediately repeated. Thus, preceding objects provided almost no information about the 'correct' response in the current trail. Subjects quickly understood the existence and nature of the two types of sequences, even though the instructions had been silent on this point. The behavioural data displayed in Figure 2.4, shows that the temporal context significantly accelerates the conditional associative learning. Those objects that were presented in a deterministic order were learnt significantly faster than those which were presented in a random order. This result was approved in all five experiments.

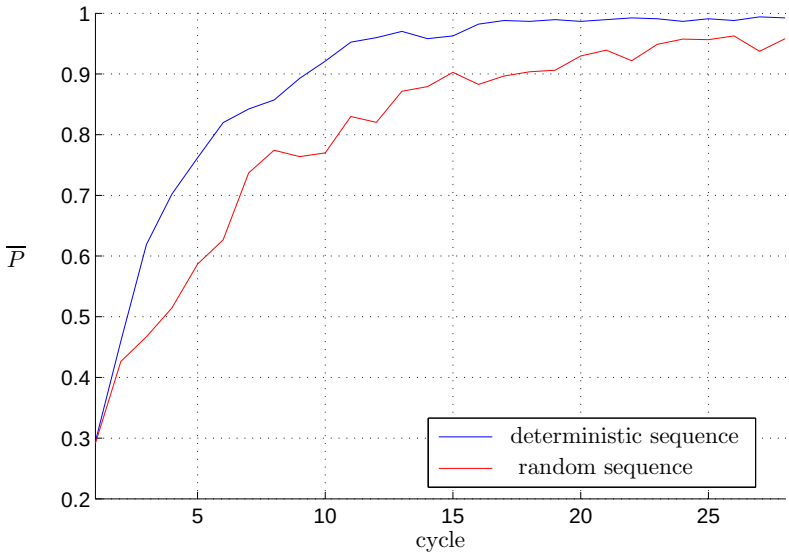


Figure 2.4: Learning curve of human subjects in conditional associative learning. In every cycle eight fractal objects were shown in either *deterministic* or *random* order. The mean probability of success $\overline{P}$ is averaged over 10 subjects for every cycle. The correct association for the objects were learnt faster when they were shown in a deterministic sequence than in a random sequence.

At this point I would like to emphasise the irrelevance of the temporal context for the solution of the association task. Each fractal object had exactly one 'correctly' associated button. The association is specific for each object and independent of all other preceding and subsequent objects. This property makes the learning scenario different from implicit learning task like grammar learning (Reber, 1967) or sequence prediction tasks (Kushner et al., 1991), where the sequential information of letters/stimuli directly contribute to the solution of the task (cf. Section 2.1.1).

One can divide the conditional associative learning task presented here in two subtasks:

1. A task irrelevant sequence learning part with low regularity salience and low task demands. This part is likely to be learnt implicitly (cf. Section 2.1.1 and Figure 2.2).

2. An associative learning part with high regularity salience and low, medium or high task demands, dependent on the number of objects to be associated. This rather explicit task covers, and is supported by, the first one.

The first task is comparable to serial reaction time tasks as used in Cleeremans (1993) and Reed & Johnson (1994) to study implicit learning. At the same time, the second task clearly makes a different to these serial reaction tests. Therefore, the conditional associative learning task seems to be suited to study how the implicit learning of temporal information may support the rather explicit learning of stimulus - response associations.

2.3 Computational Models of Implicit Learning

Following Cleeremans et al. (1998), computational models of all three learning paradigms in implicit learning have been proposed (Cleeremans, 1993; Servan-Schreiber & Anderson, 1990; Dienes & Fahey, 1995; Gibson et al., 1997; Dienes, 1992). Figure 2.5 shows different modelling approaches applied to the artificial grammar learning task.

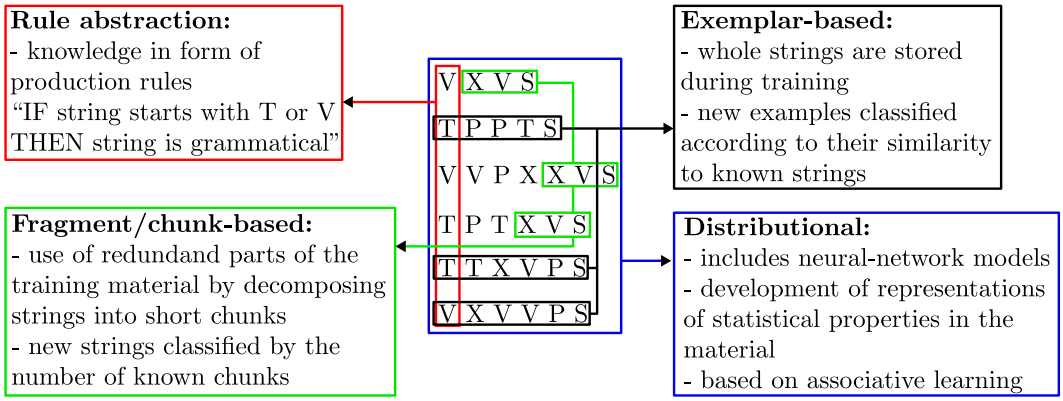


Figure 2.5: Illustration of computational models to artificial grammar learning after Cleeremans et al. (1998). Each approach assumes different mechanisms in processing and memorization of the strings.

Basically, one can distinguish *neural networks*, and *fragment-based models*. The latter one assumes a continuous process of chunk/fragment creation and application. While no model can claim generality, both share central properties:

- Learning is based on elementary association or memorization processes, which are highly sensitive to statistical features of the training data.
- Learning is incremental and continuous.
- Learning generates distributed knowledge from the processing of training examples.
- Learning is unsupervised.

These properties of the models let implicit learning appear as a form of priming. The experience during training continuously shapes memory, while stored traces, in turn, continuously influence further processing. Such priming is far away from the ideas of passive and automatic acquisition of abstract knowledge as discussed in Section 2.1.3. Further, it is dependent on the complexity of the task and the degree of similarity between learning and transfer conditions (Whittlesea & Dorken, 1993; Whittlesea & Wright, 1997).

Both, fragment-based and neural-network models, explain how statistical properties of an environment can be learnt from the processing of training examples. They differ in the question, whether features of the training material are represented explicitly or

rather computed when needed. Thus, it appears that the knowledge acquired in implicit learning is rather somewhere between explicit exemplar-based representations and general abstract representations. This characteristic is especially well represented in neural networks.

2.4 Connectionist Model of Implicit Learning

After the rather broad view on computational models I will now concentrate upon connectionist models of implicit learning. As described above, learning typically proceeds through some form of strengthening of the most task-relevant representations that have developed through exposure to the material (Cleeremans, 1993). Especially artificial neural networks incorporate these mechanism, and further have there origin at biological foundations.

Neural networks constitute a class of powerful but simple learning algorithms (Rumelhart & McClelland, 1986). Learning in neural networks is based on the development of task-relevant representations of the stimulus material. Further, associations between these representations to some desired network responses are established. The adaptation of relevant connections[3] as a function of the task constitutes the actual learning process. This may be error-driven, or units compete to respond to input patterns without external feedback.

Simple Recurrent Networks

The usage of Simple Recurrent Networks (SRNs) to model human behaviour in implicit learning tasks was firstly proposed by Servan-Schreiber et al. (1989). A comprehensive discussion on the motivation for their use can be found in Cleeremans (1993).

Early models of sequence processing used all cues of a sequence in parallel, that is, the complete sequence was processed in one piece. Therefore, one needed the assumption that the sequence of relevant elements is of a fixed length, cf. (Fanty, 1986; Sejnowski & Rosenberg, 1987; Hanson & Kegl, 1987). Typically, these models used a set of input units for every event presented at time t to $t + n$, with n denoting the fixed length of the time interval (Cleeremans, 1993). This approach is often referred as *moving window* or *sliding window*.

Elman (1988, 1990) was the first who described the connectionist architecture of SRNs, which are also known as Elman Networks. In his work the network architecture showed its potential to process sequential material on a simple temporal version of the XOR problem. Further, he showed that it is able to discover the syntactic and semantic features of words. Recurrent connections to a context layer provide the network with a dynamic memory. The usage of such recurrent links was firstly proposed by Jordan (1986).

In contrast to the sliding window approach, the processing in SRNs is *local in time* in the sense that the elements of the sequence are processed at the time of their appearance.

[3] may be interpreted as smallest "knowledge unit"

It does not need the assumption of a fixed time window of relevant information. Further and equally important, SRNs are able to *learn an internal representation of the input sequence*. The recurrent network architecture proposed by Jordan (1986) is already able to learn how to use the succession of internal states, but cannot learn an encoding of the sequential information.

Processing in Simple Recurrent Networks

Standard Feed-Forward Networks (FFNs) are able to develop internal representations of the input pattern in their hidden units. These internal representations are then used to produce the correct output assigned to some input (cf. Figure 2.6a). If such a network has more than one hidden layer, it is referred to as a Multilayer Feed-Forward Network.

The structures of a FFN and an SRN differ in one substantial point. Besides the hidden layer, a so called context layer is introduced. This layer stores the internal state of the hidden layer at the present time t. At the next time step $t+1$, this internal state is fed back to the hidden layer (cf. Figure 2.6b). This simple addition has a huge effect on the processing in the network. As the context layer remembers the previous internal state and provides this information to the hidden layer, the hidden units get a broader task. In an SRN the external input *and* the previous internal state have to be mapped to the desired output. The hidden–context layer pair must find a representation of some input pattern and, at the same time, find a reasonable encoding for the sequential structure of these representations. Therefore, Elman (1990) concludes: "..., the internal representations that develop are sensitive to temporal context; the effect of time is implicit in these internal states."

Cleeremans (1993) view of SRNs behaving as finite-state automata helps to gain a better understanding of the processing in the network. Generally, finite-state automata are able to do some form of sequence processing based on previous inputs. Minsky (1967) defined the finite-state automaton by:

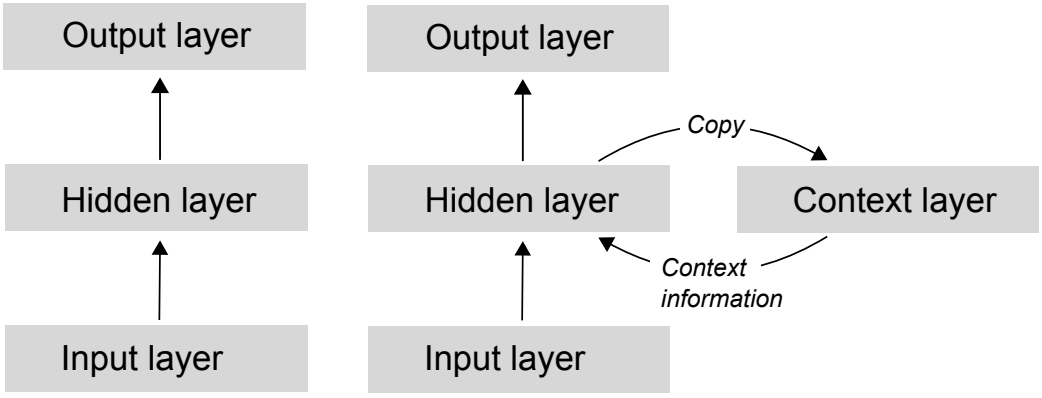
$$h(t+1) = G(h(t), i(t)), \tag{2.1}$$
$$o(t+1) = F(h(t+1)). \tag{2.2}$$

This definition says that the internal state of the automaton h at time $t+1$ is a function G of its previous state $h(t)$ and the previous input $i(t)$. The output o at time $t+1$ is a function F of its current internal state $h(t+1)$.

In an SRN, the internal state $h_{\mathrm{srn}}(t)$ (activation at the hidden layer) is a function of the previous internal state $h_{\mathrm{srn}}(t-1)$ (stored in the context layer) and the *current* input $i_{\mathrm{srn}}(t)$. Further, the activation at the output layer o_{srn} is a function of the new internal state:

$$h_{\mathrm{srn}}(t) = G(h_{\mathrm{srn}}(t-1), i_{\mathrm{srn}}(t)), \tag{2.3}$$
$$o_{\mathrm{srn}}(t) = F(h_{\mathrm{srn}}(t)). \tag{2.4}$$

The similarity between Equations (2.1),(2.2) and Equations (2.3),(2.4) is obvious. There is a difference in the point of time when the input effects the output. While the input



(a) FFN (b) SRN

Figure 2.6: Feed-Forward Network (FFN) (a) and Simple Recurrent Network (SRN) (b). Each box represents a network layer (set of units), and each forward arrow represents trainable connections form each unit in the lower layer to *all* units in the upper layer. The backward arrow in the SRN denotes a copy operation. This is achieved by connecting every single unit in the hidden layer to *one* corresponding unit in the context layer. This recurrent connections are not trainable.

of the finite-state automaton has an effect on the successive output, the input to a SRN is processed directly and influences the output at the same time.

Servan-Schreiber et al. (1991) could show that SRNs are able to learn to mimic finite-state automata in their behaviour and their state representations. Further, it could be shown that it is possible to hard-wire a linear SRN to behave like an update graph, which is another way to represent a finite-state automaton (Mozer & Bachrach, 1991). The contrast between discrete states in traditional automata and the "graded nature of representations" in a neural network motivated Servan-Schreiber et al. (1991) to constitute a new class of automata called *Graded State Machines*. SRNs and similar architectures fall into this class of automata.

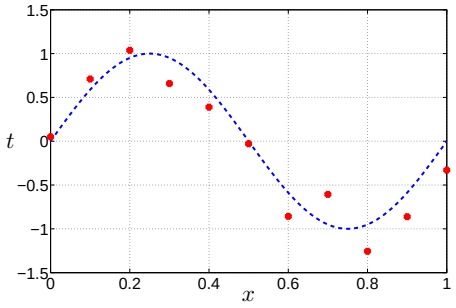
2.5 Supervised, Unsupervised and Reinforcement Learning

Yet, this Chapter dealt with the term implicit learning in the context of cognitive science. However, if it comes to the computational modelling of learning, we have to consider the knowledge that emerged in the rather technical field of artificial intelligence research. The two research branches cannot, and should not, be treated separately as they share fundamental principles and often use the same methods. For the purpose of my thesis I introduce the three main types of learning problems as they are defined in the field of Machine Learning, namely: supervised, unsupervised, and reinforcement learning. They shall be specified once as foundation for the discussion in later chapters, where the focus turns towards the applicability of recurrent networks in machine learning tasks.

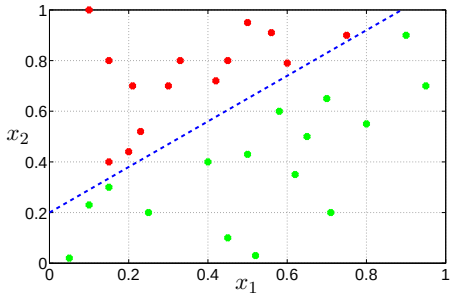
Neural networks are able to cope with all of these problems, depending on the task at hand. In each of these problem areas learning algorithms were proposed and applied to all kinds of technical tasks. A complete discussion of this matter is beyond the scope of this section. I give only an overview on each type of problem to provide a basis for the characterisation of tasks that might appear. Further, I will address these issues again, when they have some bearing on the questions being considered in later Chapters.

Supervised learning

In supervised learning the task is to learn a function between input objects and their corresponding target objects. The data set consists of pairs of training examples $\{X, T\}$. Each pair is made up from an input vector $x_i \in X$ and target vector $t_i \in T$. The target to a certain input is called supervisory signal or teacher signal. If the aim is to assign each input to one of a finite number of discrete categories, the task is a *classification* problem (cf. Figure 2.7b). If the target consists of one or more continuous variables, the task is called *regression* (cf. Figure 2.7a) (Bishop, 2006).



(a) Regression task: target data t shown as red dots plotted against the input value x. The blue curve shows the function $\sin(2\pi x)$ which underlies the data. The task is to predict the value of t given x.



(b) Classification task: the input values x_1,x_2 have to be matched to one of two classes, figured as red and green dots. In the example the classes are linearly separable by the straight line $x_2 = 0.9x_1 + 0.2$.

Figure 2.7: Examples for a simple regression (a) and classification task (b)

The function between input and target that is found by some learning algorithm, should provide a correct output for any valid input. Therefore, the algorithm has to *generalise* from the training data to unseen input values.

Supervised learning has parallels to concept learning. It is studied in human and animal psychology. Bruner et al. (1967) define concept learning as "the search for and listing of attributes that can be used to distinguish exemplars from non exemplars of various categories", which is very similar to a classification task.

Unsupervised learning

The problem in unsupervised learning is to find an underlying structure in unlabeled data. The training data consists only of a set of input values X. No target data is given, such that a possible solution cannot be evaluated. Typical unsupervised learning task are: (i) *clustering*, where the goal is to find groups of similar examples in the data (cf. Figure 2.8), (ii) *density estimation*, where the distribution of the data in the input space shall be determined, and (iii) *visualisation*, where high dimensional data is projected down on two or three dimensions for the purpose of illustration.

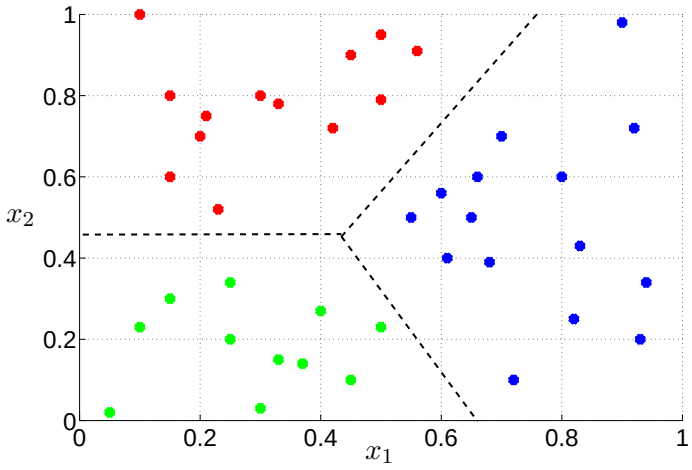


Figure 2.8: Clustering task: the input values x_1,x_2 have to be grouped in clusters, figured as red, green, and blue dots. The dashed lines show possible boarders between the three clusters. Input pairs in a cluster are more similar to each other (in some sense) than to inputs in a different cluster. The fundamental difference to the classification task in Figure 2.7b is that the labels (red, green, blue) are not given, but have to be determined.

Reinforcement Learning

Inspired by behaviourist psychology, the technique of Reinforcement Learning (RL) (Sutton & Barto, 1998) describes a class of machine learning methods that allow an agent to

learn how to act in its environment based on a *scalar* reward signal r. The actions a an agent takes may change the state s of the environment, while the observations o inform the agent about the current state. Which actions the agent takes in which situation is described by the policy π. Based on the reward, the agent shall learn an optimal policy, that is to "behave" in an optimal way in its environment (cf. Figure 2.9).

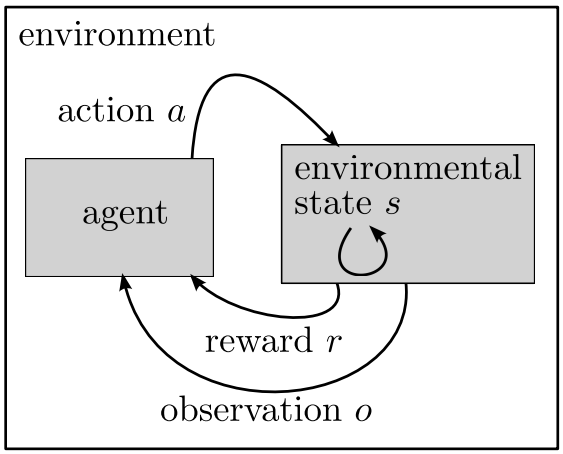


Figure 2.9: Illustration of the Reinforcement Learning situation

In contrast to supervised learning, in reinforcement learning no examples of optimal outputs (here optimal actions) are available, instead they have to be discovered by the agent by trial and error. The reward signals only whether a state is "desirable" or not. It does not provide any information which actions should have been taken to achieve a "desired" state. Usually, a RL learning algorithm interacts with the environment and produces a sequence of states and actions that is evaluated regarding a policy that maximises the reward. Often an evaluation of the current reward to an action is not sufficient, as the action taken may also influence the reward of all subsequent actions. In that cases the objective is to learn a policy that maximises the long-term reward.
Concerning the error signal, RL can be located between supervised learning, which gives a complete target value to every training input, and unsupervised learning, which does not provide any target at all.

2.6 Sequence Learning as a Machine Learning Discipline

From the discussion above (cf. Sections 2.1 and 2.2) we saw that sequences play an important role in implicit learning, but also in human skill learning in general (Sun et al., 2001), high-level problem solving and reasoning (Anderson, 1995). Therefore, it is reasonable to address sequence learning also in the field of intelligent systems as a component in machine learning. Sun & Giles (2001) wrote one of the most cited articles in this context. In their tutorial they summarize the state of the art techniques, approaches, and paradigms of that time. Even though the algorithms and techniques developed further, the problem formulation and basic sequence learning approaches are still up to date. Those concepts shall be introduced in the following to put the work presented in Chapters 5 and 6 into a broader context. The following summary is given along the lines of Sun & Giles (2001).

In general, the sequence learning problem can be subdivided in three categories. If s denote the elements and N denotes the length of a sequence they can be formulated as follows:

- Sequence classification/recognition – determine if a sequence satisfies some criteria. $s_{t-N}, s_{t-(N-1)}, \ldots, s_t \rightarrow$ yes/no, that is, given $s_{t-N}, s_{t-(N-1)}, \ldots, s_t$ we want to determine whether this sequence belongs to a predefined class and/or satisfies a certain criterion.

- Sequence prediction – predict future element(s) of a sequence on the basis of prior elements of the sequence. $s_{t-N}, s_{t-(N-1)}, \ldots, s_t \rightarrow s_{t+1}$, that is, given $s_{t-N}, s_{t-(N-1)}, \ldots, s_t$ we want to predict s_{t+1}. If N is the total length we make predictions based on all the previously seen elements. If we set $N = 1$ the prediction is only based on the actual element of the sequence.

- Sequential decision making – select a sequence of actions to accomplish a given goal. $s_{t-N}, s_{t-(N-1)}, \ldots, s_t; G \rightarrow a_t, \ldots, a_t + j$, that is, given sequence $s_{t-N}, s_{t-(N-1)}$ $, \ldots, s_t$ and goal G, we want to chose a series of actions $a_t, \ldots, a_t + j$ that leads to this goal. The goal itself may be a goal state, a goal trajectory or a reinforcement-maximizing goal.

More specific problems in sequence learning can be derived from these basic categories, for instance, the segmentation of sequences for a compact representation (Nevill-Manning & Witten, 1997) or to learn to deal with temporal dependencies (Mccallum, 1996; Sun & Sessions, 2000). Further, the formulation of hierarchies of sequential segmentation during learning to facilitate the learning process is a research topic itself (Sun & Giles, 2001).

The most prominent approaches to sequence learning are Hidden Markov Models (HMMs), recurrent neural networks and the temporal-difference method within the RL framework. As we saw in Section 2.4 recurrent networks use hidden units as memory, where a representation of the previously experienced sequence evolves. Such networks are suitable for sequence classification and prediction tasks. I come back to this point in Chapter 5 where a special network architecture is introduced for this purpose.

HMMs learn underlying state transition probability distributions from which observed data can be generated. Therefore, they are suited for sequence generation (Baum, 1972). Further, HMMs can be applied to sequence classification by the computation of the probabilities to generate a certain sequence from the underlying models.

The temporal-difference method (Sutton & Barto, 1998; Wendemuth, 2007) can be applied to the problem of sequential decision making. It is a RL technique based on an evaluation function which generates a "measure of goodness" for the current state. This measure is called *action value*. Accordingly, the method selects an action on the basis of the evaluation function leading to a new state and a reinforcement signal. Then the action policy is updated based on the reward and action value, that is, increase or decrease the probability to chose the action again in this situation. If the policy and evaluation function is merged in on single function this method is called *Q-learning* $Q(x, a)$, where x is the current state and a is an action (Sun & Giles, 2001).

Many approaches exist where different learning techniques are combined to form hybrid models. For instance using SRNs to learn a RL policy (Bakker, 2004), or train an SRN with a RL signal (cf. Section 3.2.1).

One significant issue in sequence learning remains the learning of temporal dependencies. This occurs when a sequence element or the class of a sequence depends on what has happened before or even on what has happened a long time ago. For dynamic systems in general it is hard to deal with long-term dependencies (Bengio et al., 1994). I put a finer point on this issue in Chapters 5 and 6.

2.7 Discussion

In this chapter the concept of implicit learning was introduced as 'incidental learning resulting in knowledge which cannot be verbalised'. First of all, implicit learning was investigated empirically in psychology. The most prominent results from studies with grammar learning, complex process control, and sequential pattern acquisition were outlined to gain a deeper understanding for the phenomenon. This was enhanced by a discussion of the problems that arise in empirical research and the general debates in the psychology community concerning the nature of implicit learning.

Then implicit learning was reviewed from the point of view of cognitive biology, where the term itself is not as established as in psychology. Here researchers rather talk about context-dependent learning, which shares the idea with implicit learning that it happens incidentally, triggered by the environmental conditions. Especially temporal context, that is, the role of sequential information, was found to be of great importance in that field. Several behavioural studies with non-human primates and human subjects support this assumption. Further, it is in coincidence with the idea from psychology that frequency detection is a process that underlies implicit learning.

In particular a behavioural experiment using the conditional associative learning scenario was highlighted. It is the basis for the computational models that are introduced in the next chapter. Basically, it investigates the question how task-irrelevant temporal context affects the learning of arbitrary visuo-motor associations in humans. The task can be divided in two subtasks, an implicit sequence learning part and an explicit associative learning part.

Subsequently, different models of implicit learning were introduced. While non of them can claim generality, they share central properties that are especially well incorporated in artificial neural networks. This motivated the use of SRNs in cognitive modelling of implicit learning, firstly proposed by Servan-Schreiber et al. (1989). As those networks are of special interest in this thesis the general idea, as compared to feed-forward networks, and the processing in SRNs was introduced in detail.

Thereafter a short definition of three problem types in machine learning was given, that is, supervised, unsupervised, and reinforcement learning. They are introduced as foundation for the discussion in later chapters. The major approaches and paradigms in the field of sequence learning were outlined for the same reason.

In the next chapter the conditional associative learning scenario is regarded as a reinforcement learning problem. However, the interest in this behavioural experiment arises from its roots in cognitive biology. Therefore, the intention is first of all the formulation of a computational model to gain an understanding for the cognitive processes underlying the observed behaviour.

3 Computational Models of Conditional Associative Learning

Contents

In the following I introduce two fundamental different computational models for conditional associative learning. Based on the learning scenario established in Hamid et al. (2010) (cf. Section 2.2.3) a Markov model is introduced. It gives the chance to an analysis of the *explicit* associative learning task. To capture the *implicit* learning of task-irrelevant temporal context in that scenario, a connectionist model is proposed. Finally both models are compared to each other and to the RL model introduced by Hamid et al. (2010).

The models I present in this chapter may be regarded as accepted in the corresponding scientific community. They are separately published. The Markov model as chapter in the "Lecture Notes in Computer Science" – Volume 6686 (Glüge et al., 2011b) and the connectionist model as article in the journal "Cognitive Computation" (Glüge et al., 2010)[1].

3.1 Markov Model of Conditional Associative Learning

Before I give a detailed description of the model, I would like to highlight the fact that this model only represents the *explicit* part of the task presented in Section 2.2.3. It refers to the learning of the 'correct' response by trial and error for each presented object

[1] Parts of the text in the Sections 3.1 and 3.2 are taken verbatim from these publications.

as it is illustrated in Figure 2.3. The *implicit* part of learning the temporal order in which objects are presented is not captured by the Markov model. Nevertheless, it provides the opportunity to analyse the explicit learning task and judge the influence of model parameters that can be controlled by the experimental setup.

3.1.1 Markov Property and Markov Model

The Markov property describes a special kind of random process. It says that *knowledge about the limited history of the process is sufficient to predict the future development of the process* as if one would know the entire history. For instance, in a first order Markov process the next state depends *only* on the current state and not on the past. In a second order Markov process the next state depends on the current and the previous state, etc.

A Markov model is based on a stochastic process which assumes the Markov property. Figure 3.1 illustrates such model consisting of three states. In its simplest form, the system states are modelled as random variables that evolve over time. In this case, the Markov property guarantees that the actual distribution of the state variables depends only on the previous distribution of state variables.

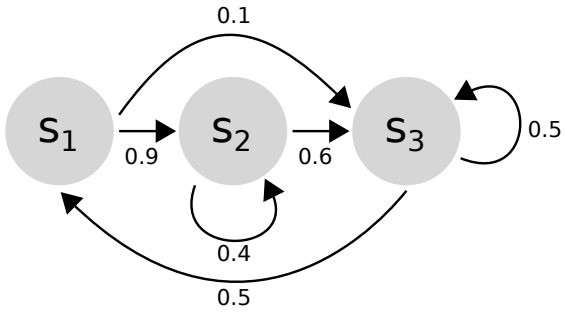


Figure 3.1: Markov graph of transition probabilities between states $s_{1,\ldots,3}$

3.1.2 Behavioural Markov Model

Along the line of the underlying learning task, as it is described in Section 2.2.3, the focus lies on three events which effect the subjects' performance. Those are:

- Subjects may not remember whether the last trial on an object was successful or not.
- If subjects remember the current object, they might not remember which actions they chose in the previous trials.
- It might be hard to clearly distinguish various fractal objects of one another which leads to confusion of objects.

The effects of confusing objects, memorising previous actions, and memorising success for objects are represented by separate parameters of the model.

At first, the state space for the learning process of a single object is defined. The history of that learning process is accumulated in the state of the model. To narrow down the possibilities of state transitions it is reasonable to assume a rational policy in choosing the action on a certain object. This is: (i) once the successful button was found, this choice is repeated in later trials, and (ii) these choices which so far have not been successful are not taken again. The remaining options are chosen with equal probability.

Based on this policy the Markov model for associative learning of the motor response for an object is set up. It consists of 16 states ($S = \{s_1, s_2, \ldots, s_{16}\}$). Each state represents one possible situation a subject might face during learning. At the beginning, the subject has no clue which button to take (s_1). For each possible decision we define four states ($s_2, \ldots, s_5$). After the first choice the subject has three possibilities left, which lead to the next states and so on. Table 3.1 shows all possible situations/states. The last state s_{16} represents success. In this state the subject found the correct button and memorised it.

Table 3.1: Possible states during associative learning of one object by trial and error.
'x' implies 'chosen and memorised to be wrong' and
'.' implies 'not chosen yet'.

state s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
left	.	x	.	.	.	x	x	x	.	.	.	x	x	x	.	success
right	.	.	x	.	.	x	.	.	x	x	.	x	x	.	x	success
up	.	.	.	x	.	.	x	.	x	.	x	x	.	x	x	success
down	.	.	.	.	x	.	.	x	.	x	x	.	x	x	x	success

Given the state space of the model, transition probabilities define the way through it. It is summarised in the state transition matrix $\mathbf{T}$. The general structure of the matrix comprises the behaviour of a subject during learning. At this point a parameter is introduce, which allows the model to make a mistake in recognition of success (Reinforcement - green/red disk, cf. Figure 2.3). If the system indicates that the chosen button was correct, the subject might memorise the event. On the other hand, it may simply forget this event in the following. This uncertainty is represented by the *success recall probability* p_{suc} indicating the chance a subject memorises a success. Hence, p_{suc} is the probability to get into the success state (s_{16}) if the correct button was pressed. The ideal case denotes $p_{\text{suc}} = 1$. A lower p_{suc} lowers the ability of the model to recall a

success. With this parameter the transition matrix takes the form

$$\mathbf{T}_{\text{left}} = \begin{pmatrix}
0 & \frac{1-p_{\text{suc}}}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{p_{\text{suc}}}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1-p_{\text{suc}}}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{p_{\text{suc}}}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1-p_{\text{suc}}}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{p_{\text{suc}}}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1-p_{\text{suc}}}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{p_{\text{suc}}}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1-p_{\text{suc}}}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1-p_{\text{suc}}}{2} & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1-p_{\text{suc}}}{2} & \frac{1}{2} \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}. \tag{3.1}$$

Equation 3.1 shows the transition matrix in the case that the button 'left' is correct for the concerning object. For the three other cases the matrix looks slightly different, but the general structure is the same. If $p_{\text{suc}} < 1$ a correct choice may be erroneously considered as unsuccessful and therefore, not be chosen again. This leads to a case where all four choices have been taken and all are recognised to be 'incorrect'. In that situation the model will reveal that something went wrong and start over, meaning going back to state s_1 (highlighted rows 12, 13, and 14 in $\mathbf{T}_{\text{left}}$).

The graphical representation of the model is shown in Figure 3.2 with the proposed state space (Table 3.1) and the transition matrix $\mathbf{T}_{\text{left}}$.

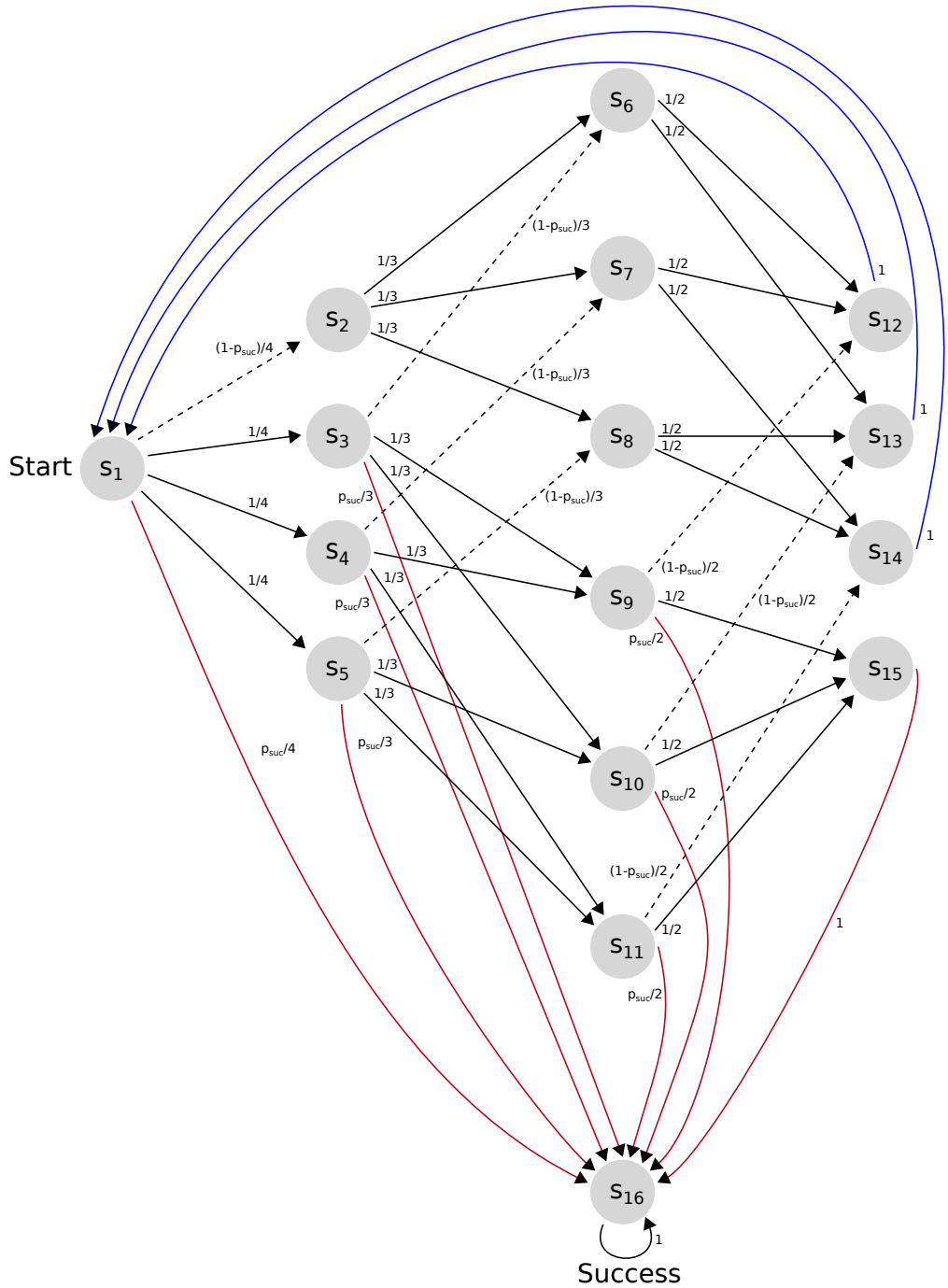


Figure 3.2: Markov model for the associative learning of an object with the probability to memorise success p_{suc}, which is the probability to get into the success state (s_{16}) if the correct button 'left' was pressed (red arrows). Otherwise, the correct button is memorised to be wrong (dashed arrows) leading to cases where all four choices have been taken and all are recognised to be 'incorrect' (s_{12},s_{13},s_{14}). Further, the nth column of states represents the models options in the nth trail.

History of Choices and Objects: Besides the possibility to be unsure about success or failure, subjects might not remember which actions they chose in previous trials at all. This shall be captured by a parameter in the model. The history of actions is encoded in the model's state. This means, the probability to remember the current state directly represents a subject's ability to remember its previous actions on an object. Therefore, p_{sr} is defined as *state recall probability*, which implies that $1-p_{\mathrm{sr}}$ represents the probability to confuse the current state with all other possible states. The success state s_{16} should not be confused. Thus p_{sr} is located between an ideal learner ($p_{\mathrm{sr}} = 1$) and a total confused one ($p_{\mathrm{sr}} = 1/15$). The state confusion matrix $\mathbf{C}$ holds these probabilities, where the element C_{ij} is the probability of assigning state s_i to s_j. In Equation 3.2 the model's attention is uniformly distributed over all states. For an ideal learner $\mathbf{C}$ becomes the identity matrix.

$$\mathbf{C} = \begin{pmatrix} p_{\mathrm{sr}} & \frac{1-p_{\mathrm{sr}}}{14} & \frac{1-p_{\mathrm{sr}}}{14} & \cdots & \frac{1-p_{\mathrm{sr}}}{14} & 0 \\ \frac{1-p_{\mathrm{sr}}}{14} & p_{\mathrm{sr}} & \frac{1-p_{\mathrm{sr}}}{14} & \cdots & \frac{1-p_{\mathrm{sr}}}{14} & 0 \\ \vdots & & \ddots & & \vdots & \vdots \\ \frac{1-p_{\mathrm{sr}}}{14} & \cdots & \frac{1-p_{\mathrm{sr}}}{14} & p_{\mathrm{sr}} & \frac{1-p_{\mathrm{sr}}}{14} & 0 \\ \frac{1-p_{\mathrm{sr}}}{14} & \cdots & \frac{1-p_{\mathrm{sr}}}{14} & \frac{1-p_{\mathrm{sr}}}{14} & p_{\mathrm{sr}} & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

Finally, subjects may confuse objects amongst each other. This is described by a third parameter in the model. The *object recall probability* p_{or} is the model's chance to remember the current object. In reverse, $1 - p_{\mathrm{or}}$ is the probability to confuse the current object with all other possible objects. A series containing eight different objects leads to $1/8 \leq p_{\mathrm{or}} \leq 1$. In that way, $p_{\mathrm{or}} = 1$ represents a perfect recall of objects and $p_{\mathrm{or}} = 1/8$ describes a learner without any memory for objects. It is summarised in the object confusion matrix $\mathbf{O}$ with O_{ij} being the probability of assigning object i to object j. Again, the attention is uniformly distributed over all objects and, as above, in case of an ideal learner $\mathbf{O}$ becomes the identity matrix.

$$\mathbf{O} = \begin{pmatrix} p_{\mathrm{or}} & \frac{1-p_{\mathrm{or}}}{7} & \frac{1-p_{\mathrm{or}}}{7} & \cdots & \frac{1-p_{\mathrm{or}}}{7} \\ \frac{1-p_{\mathrm{or}}}{7} & p_{\mathrm{or}} & \frac{1-p_{\mathrm{or}}}{7} & \cdots & \frac{1-p_{\mathrm{or}}}{7} \\ \vdots & & \ddots & & \vdots \\ \frac{1-p_{\mathrm{or}}}{7} & \cdots & \frac{1-p_{\mathrm{or}}}{7} & p_{\mathrm{or}} & \frac{1-p_{\mathrm{or}}}{7} \\ \frac{1-p_{\mathrm{or}}}{7} & \cdots & \frac{1-p_{\mathrm{or}}}{7} & \frac{1-p_{\mathrm{or}}}{7} & p_{\mathrm{or}} \end{pmatrix} \tag{3.3}$$

Learning Process in the Model: With the defined state space and transition probabilities the learning process for every single object can be described. For each object the model performs a trial by selection of a button following the state transition matrix. As this is not a deterministic process one has to calculate the probabilities to be in a certain state $\boldsymbol{\pi} = P(s_i)$ for all $s_i \in S$. Always starting in state s_1, the initial state distribution $\boldsymbol{\pi}(t = 0)$ is $\pi_1(t = 0) = 1$ and $\pi_{2,\ldots,16}(t = 0) = 0$. For every trial the new state probability vector $\boldsymbol{\pi}(t + 1)$ results from the product of the transposed transition matrix $\mathbf{T}$ and

the previous state probability vector $\boldsymbol{\pi}(t)$. Since uncertainty about previous actions is allowed the state confusion matrix gets part of the product

$$\boldsymbol{\pi}(t+1) = \mathbf{T}^{\mathrm{T}} \cdot \mathbf{C} \cdot \boldsymbol{\pi}(t). \tag{3.4}$$

To gain the total success probability, the probability of success of the actual and previous trial has to be summed up. So, if 'left' is considered to be the correct button, state s_{15} automatically leads to success (s_{16}) in the next trial (cf. Table 3.1 and Equation 3.1). Thus, this sum becomes $\pi_{15}(t+1) = \pi_{15}(t) + \pi_{16}(t)$ for that case.

Yet, the state probabilities are calculated for each object separately, or in other words, the learning processes was described for a single object so far. For a series of l objects, it is represented by the state probability vectors $\boldsymbol{\pi}^l$ for each object. The combination of these separate learning processes into one single learning process for a series of objects contains the possibility of object confusion. Therefore, the state probabilities for all objects have to be updated, even if just one is really shown to the subject. This implies that the probability of success on a presented object is the weighted sum of the success states of all objects with their probability of appearance. A series of eight objects $l = 1, \ldots, 8$ yields the probability of success P_k for object k

$$P_k(t) = \sum_{l=1}^{8} O_{kl} \cdot {\pi^l}_{16}(t) \qquad k \in \{1, \ldots, 8\}. \tag{3.5}$$

So, P_k is the final probability the model picks the correct button if object k is shown. Before the next learning step, the state probability vectors of all objects $\boldsymbol{\pi}^l(t)$ are weighted with the probability to confuse the current object k with object l

$${\boldsymbol{\pi}^l}_{\mathrm{update}} = O_{kl}\boldsymbol{\pi}^l + (1 - O_{kl})\boldsymbol{\pi}^l, \text{ with } l = 1, \ldots, 8 \text{ and } k \in \{1, \ldots, 8\}\ . \tag{3.6}$$

Equation 3.6 ensures that all state probability vectors are updated, even if just object k is shown. This is due to the fact that object k might be confused with all other objects. In case of a perfect recall of objects this step does not change the state probability vectors since $O_{kl} = 0$ if $k \neq l$ and $O_{kl} = 1$ if $k = l$.

Figure 3.3 shows the processing in the model. Briefly summarised, for every single object one learning process through 16 possible states is modelled. The state space is given by the experimental set up and the state transitions are given by the assumed policy a subject follows. Further, three parameters control typical mistakes that might occur during trial and error learning. These parameters are: (i) the success recall probability p_{suc} giving the chance to confuse 'success' and 'failure', (ii) the state recall probability p_{sr} representing the model's ability to remember its previous actions on an object, and (iii) the object recall probability p_{or}, indicating the chance to remember the current object in a later trial.

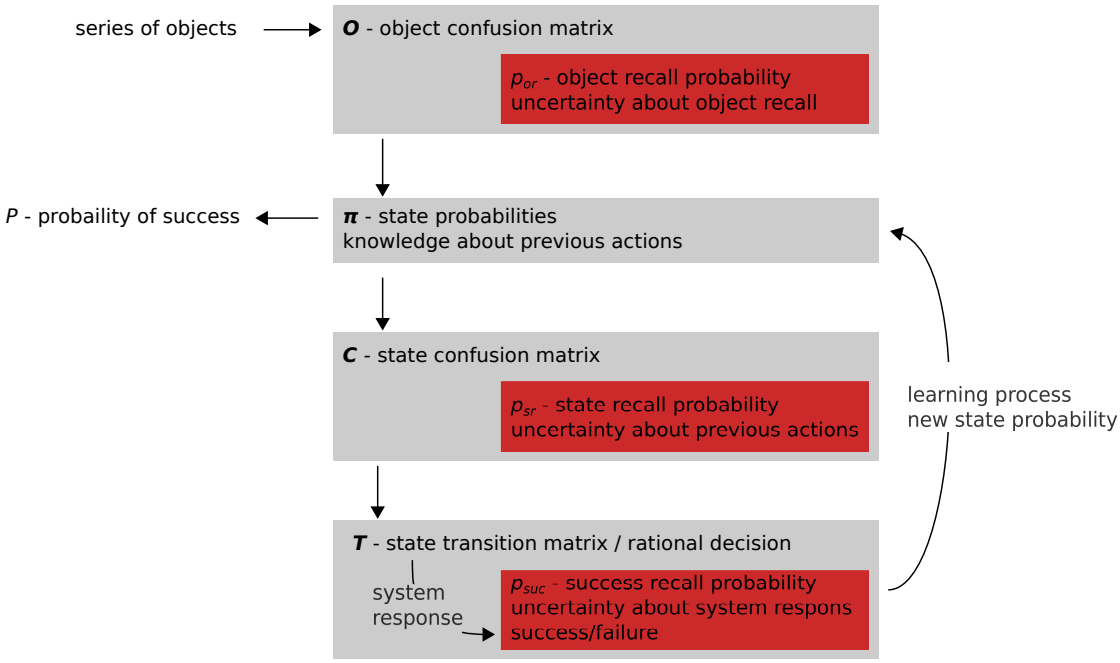


Figure 3.3: Processing of the Markov model. Learning of a single object is modelled by transitions between 16 states. Three parameters control typical mistakes that occur during associative learning: p_{suc} success recall probability, p_{sr} state recall probability, and p_{or} object recall probability.

Figure 3.4 shows the learning curves of the model for each object in a series of eight. In this simulation eight objects are shown to the model. The objects are presented for ten cycles. That is, every single object is shown once per cycle, which results in 10 trials on every object and 80 trials in total. In case of a deterministic order the objects are presented in the same succession in every cycle. For the random case the succession of the objects is randomised in every cycle.

As one can see, the model behaves similar for deterministic and random ordered sequences. It treats every single object the same, regardless of its predecessors, or in other words, regardless of its temporal context. The advantage of such basic Markov model is the possibility to an analytic view on the learning process. The following analysis is limited on the explicit associative learning task on a *single* object, which is described above until Equation 3.4.

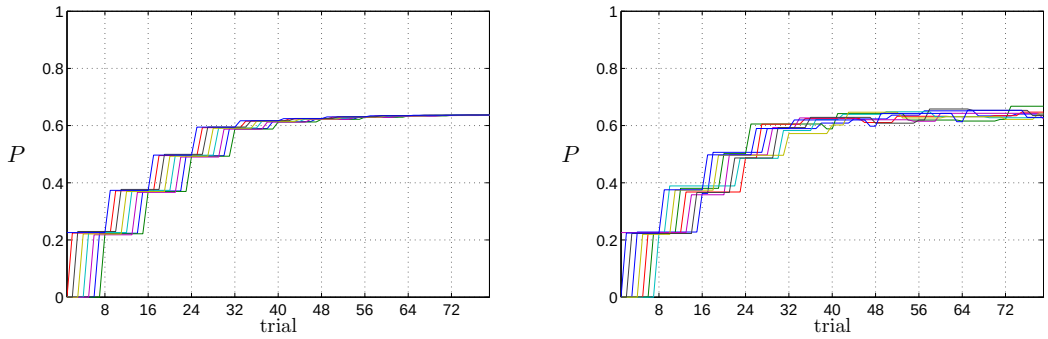


Figure 3.4: Markov model learning a series of eight objects: deterministic order (left) and random order (right). Every line shows the probability of success P (cf. Eq. 3.5) for the concerning object. Model parameters were $p_{\text{suc}} = p_{\text{sr}} = p_{\text{or}} = 0.9$.

3.1.3 Analysis of the Markov Model

The steady state and convergence of the Markov model illustrate the correlation of the parameters p_{suc} and p_{sr}. Additionally, it shows their influence during learning. No further finding on the role of the parameter p_{or} is gained by this analysis since the parameter influences the learning process for a *series* of objects, but not the learning of a single object.

The steady state has its relevance in the asymptotic character of the state probability vector $\boldsymbol{\pi}(t)$ for $t \to \infty$. Translated to the original learning situation it describes a subject that is shown the same object an infinite amount of times. For convenience, the short cut $\mathbf{T}^{\mathrm{T}} \cdot \mathbf{C} = \mathbf{A}$ is used in the following (cf. Equation 3.4). Repetitive replacement of $\boldsymbol{\pi}(t+1)$ by $\mathbf{A} \cdot \boldsymbol{\pi}(t)$ in Equation 3.4 gives the steady state:

$$\boldsymbol{\pi}_{stat} = \lim_{t \to \infty} \mathbf{A}^t \boldsymbol{\pi}(0) \quad \hat{=} \quad \boldsymbol{\pi}_{stat} = \mathbf{A}\boldsymbol{\pi}_{stat} \ , \tag{3.7}$$

leading to the linear system of equations

$$\pi_{j_{stat}} = \sum_i A_{ij} \pi_{i_{stat}} \ , \quad i, j \in S. \tag{3.8}$$

With the analytic solution for the steady state (Equation 3.8) the final state probability vector of the Markov chain can be calculated. The closed solution was found with a computer algebra system. The final probability of success $\pi_{16_{stat}}$ depends only on the parameter p_{suc} and p_{sr} in a fraction which numerator is a sixth-degree polynomial of p_{sr} and second-degree of p_{suc}. The denominator is a fourth-degree polynomial of p_{sr} and first order for p_{suc}. The structure of the whole term does not provide any extra information. Nevertheless, it represents the probability of success on the explicit task of associating one object with the rewarded choice, if the object is shown in an infinite loop. So, $\pi_{16_{stat}}$

is the upper limit for the performance of the model and can be calculated solely based on the parameters p_{suc} and p_{sr}.

Figure 3.5 shows $\pi_{16_{stat}}$ for variable p_{suc} and p_{sr}. The impact of p_{sr} is much greater than p_{suc}. If $p_{\text{sr}} < 1$ than p_{suc} only slightly affects $\pi_{16_{stat}}$. Therefore, one can conclude that the probability a subject remembers the learning history on an object (p_{sr}) is primarily responsible for the long-term development of the success probability.

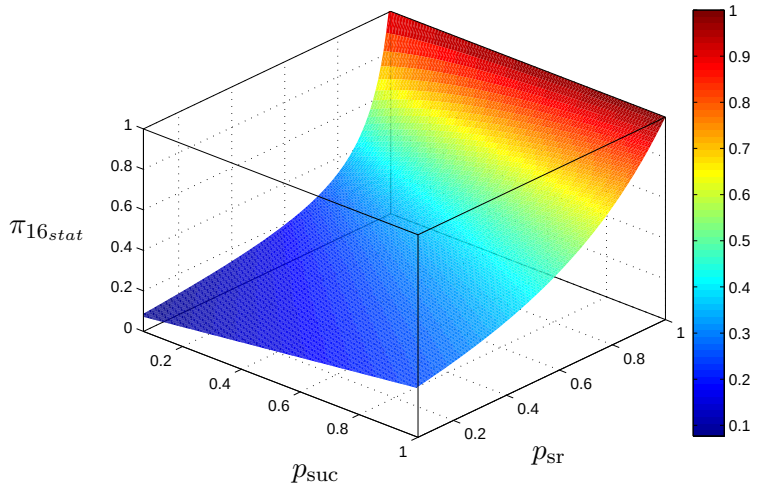


Figure 3.5: Upper limit of success probability $\pi_{16_{stat}}$ depending on p_{suc} and p_{sr}.

The convergence of the learning process is also affected by the parameters. The matrix $\mathbf{A} = \mathbf{T}^{\text{T}} \cdot \mathbf{C}$ has five non-zero eigenvalues whereas λ_1 is dominant and $\lambda_{2,\ldots,5}$ take the form

$$\begin{aligned}
\lambda_1 &= 1 \\
\lambda_2 &= \tfrac{1}{28}(12 - 12p_{\text{suc}})^{\frac{1}{4}} \cdot (15p_{\text{sr}} - 1) \\
\lambda_3 &= -\tfrac{1}{28}(12 - 12p_{\text{suc}})^{\frac{1}{4}} \cdot (15p_{\text{sr}} - 1) \\
\lambda_4 &= I\tfrac{1}{28}(12 - 12p_{\text{suc}})^{\frac{1}{4}} \cdot (15p_{\text{sr}} - 1) \\
\lambda_5 &= -I\tfrac{1}{28}(12 - 12p_{\text{suc}})^{\frac{1}{4}} \cdot (15p_{\text{sr}} - 1).
\end{aligned} \tag{3.9}$$

For reasonable combinations of p_{suc}, p_{sr} follows

$$|\lambda_{2,\ldots,5}| < 1 \qquad \text{if} \qquad \frac{1}{15} \leq p_{\text{suc}}, p_{\text{sr}} \leq 1. \tag{3.10}$$

The state probability vector $\boldsymbol{\pi}$ can be written in terms of its orthogonal basis

$$\boldsymbol{\pi} = \sum_{i=1}^{16} \frac{\langle \boldsymbol{\pi}, \boldsymbol{e}_i \rangle}{|\boldsymbol{e}_i|^2} \boldsymbol{e}_i, \tag{3.11}$$

such that Equation 3.7 at time t is

$$\boldsymbol{\pi}(t) = \mathbf{A}^t\boldsymbol{\pi}(0) = \sum_{i=1}^{16} \frac{\langle \boldsymbol{\pi}(0), \boldsymbol{e}_i \rangle}{|\boldsymbol{e}_i|^2} \mathbf{A}^t \boldsymbol{e}_i, \tag{3.12}$$

which is by the definition of the eigenvalues

$$\boldsymbol{\pi}(t) = \sum_{i=1}^{16} \frac{\langle \boldsymbol{\pi}(0), \boldsymbol{e}_i \rangle}{|\boldsymbol{e}_i|^2} \lambda_i^t \boldsymbol{e}_i. \tag{3.13}$$

Since λ_i is taken to the power of t the contribution of the eigenvalues $\lambda_{2,\ldots,5}$ to the sum in Equation 3.13 is lowered with every step ($|\lambda_{2,\ldots,5}| < 1$). The smaller they are, the faster they disappear in the sum and the faster the sum converges to the steady state. Figure 3.6 shows $|\lambda_{2,\ldots,5}|$ against the parameters p_{suc} and p_{sr}. The impact of p_{suc} is stronger than those of p_{sr}. A moderate p_{sr} but a small p_{suc} gives $|\lambda_{2,\ldots,5}| > 0.5$, which leads to a slow convergence, and slow learning respectively. This tells us that the probability a subject memorises a success correctly (p_{suc}) mainly affects the slope of the learning curve.

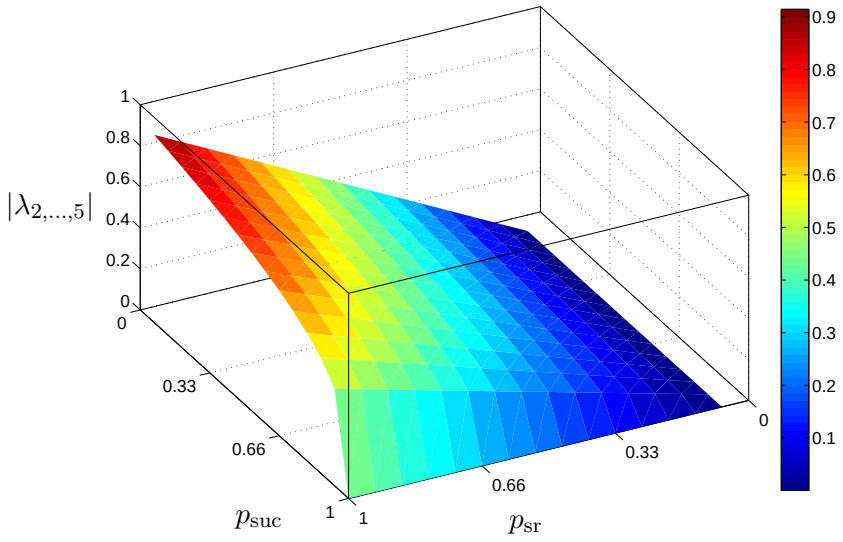


Figure 3.6: Absolute value of the eigenvalues $|\lambda_{2,\ldots,5}|$ depending on p_{suc} and p_{sr}. The smaller $|\lambda_{2,\ldots,5}|$, the faster the sum in Equation 3.13 converges to the steady state.

3.1.4 Fit of Model Parameter to Subjects' Data

The Markov model was simulated on data gathered in the experiment presented in Hamid et al. (2010) and shown in Figure 2.4. The subjects were tested on the random

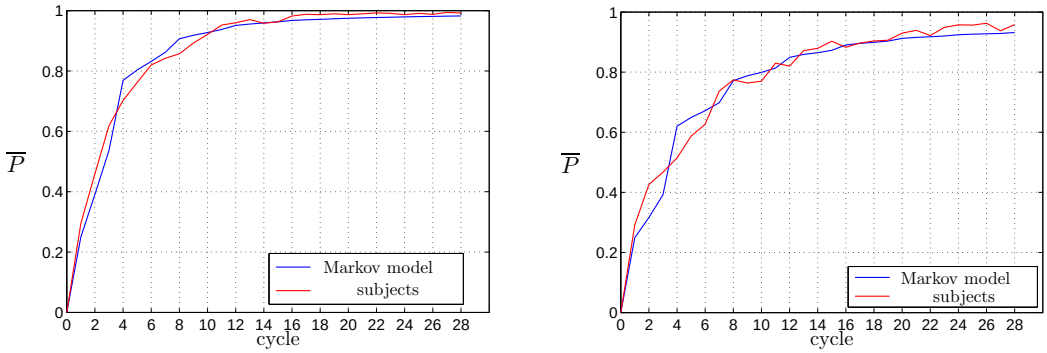


Figure 3.7: Learning process of the Markov model on the deterministic (left) and random sequence (right). The parameters were $p_{\text{suc}} = 0.6$, $p_{\text{sr}} = 1$, $p_{\text{or}} = 0.9875$ (left) and $p_{\text{suc}} = 0.3$, $p_{\text{sr}} = 1$, $p_{\text{or}} = 0.9875$ (right). The mean probability of success $\overline{P}$ is averaged over all subjects.

and deterministic ordered sequence as described in Section 2.2.3. The model parameters were manually selected to fit the subjects' learning behaviour (cf. Figure 3.7).

As the learning curve of the subjects (red) differ only in the slop, only the parameter p_{suc} has to be adjusted. This implies that a clear temporal relationship between the shown objects, basically supports a subject's ability to memorises the reinforcement signal (correct/incorrect, and green/red circle respectively) on the objects. Of cause, such conclusion is highly speculative, as it is based on limited data and just one model. Further, the model actually ignores the temporal context between objects during learning. Therefore, we cannot expect it to make predictions concerning the performance difference which occurred in the experiment.

Nevertheless, it allows to test different experimental settings that are captured by the parameters. For instance, the object recall probability p_{or} can be influenced directly by the selection of objects. Simple geometric figures instead of fractal objects might be easier to remember which would require an even higher p_{or}.

Concerning the explicit associative learning task (object association with the correct button) the analysis of the model produced two results. First, the probability a subject memorises a success correct (p_{suc}) affects mainly the slope of the learning curve (cf. Figure 3.6). Second, the probability a subject remembers the learning history on an object (p_{sr}) is primarily responsible for the long term development of the success probability. Especially in the range $0.6 < p_{\text{sr}} < 1$ small changes of p_{sr} have a great impact on the final success probability $\pi_{16_{stat}}$ (cf. Figure 3.5).

3.2 Connectionist Model of Conditional Associative Learning

As pointed out in Section 2.4, Simple Recurrent Networks (SRNs) gained great attention in the field of implicit learning. First of all, SRNs were able to capture the effects that were measured in grammar learning and serial reaction time tasks (Cleeremans, 1993) (cf. Section 2.1.1). It seems pretty natural to use the SRN approach on the conditional associative learning task introduced earlier (cf. Section 2.2.3). The explicit part of object association to some button can be learnt within the forward pass of the SRN. Here, the network needs to find an encoding of the input (object) in the hidden layer and assign the correct output (button) to it. The implicit part of sequence learning can be realised due to the recurrent connections that provide the hidden layer of the network with the knowledge about its previous state. The process is illustrated in Figure 3.8.

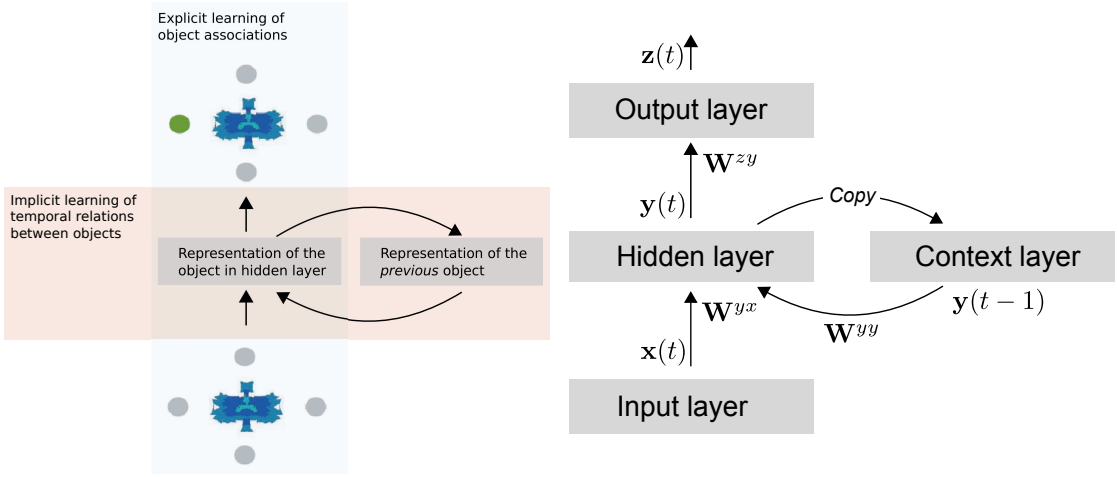


Figure 3.8: Conditional associative learning in an SRN. The general SRN architecture (right) is shown together with the processing in the conditional associative learning task (left). The explicit object-to-button association is learnt via an encoding of the objects in the hidden layer. The implicit part of the sequence of objects can be learnt through the recurrent connections that provide the hidden layer with its previous state.

3.2.1 Reinforcement Learning in Neural Networks

In terms of machine learning the conditional associative learning task is a classification problem as described in Section 2.5. For each object the correct button (class) has to be learnt. Usually, such tasks are solved using a supervised learning technique. For the problem at hand we lack a fully specific target; that is, only a feedback whether the choice was correct/incorrect is given. This directly leads to the Reinforcement Learning (RL) paradigm.

Commonly the training algorithms for neural networks are variants of gradient descent methods. The output of the network is compared to the target output and some measure of error is calculated. Then, the error is being propagated back through the network and weights are changed by the gradient descent algorithm to minimise it.

From a non-technical point of view, backpropagation algorithms have two major drawbacks: First, they are biological implausible because synapses are used bidirectionally. Neural activity is forwarded through the network, and each synapse's contribution to the error is propagated back. The second criticism aims less at a specific algorithm but at the supervised learning paradigm itself. It is cognitive implausible, as the network needs a full and correct output pattern to learn. In most natural learning situations such pattern is simply not available.

The second drawback again leads to the RL paradigm. One of the first approaches to extend the standard backpropagation algorithm for neural networks (Rumelhart et al., 1986a) is the complementary reinforcement backpropagation proposed by Ackley & Littman (1990). The Elman backpropagation algorithm was adapted to RL by Grüning (2007).

Approaches which address the biological implausibility of backpropagation are for instance, associative reward penalty (Mazzoni et al., 1991) and node perturbation (Werfel et al., 2005).

Backpropagation with Reinforcement Learning in Simple Recurrent Networks

The easiest way to train an SRN on an RL signal is a gradient descent method with the adaptation of Grüning (2007). The SRN is trained using the backpropagation algorithm for feed forward networks with an RL-like error signal. To do so, two constraints must be fulfilled:

1. The context units must be initialised with some activation for the forward propagation of the first training vector. Commonly, these initial values are zero.

2. The activation levels of the hidden layer must be stored in the context layer after each backpropagation phase. Thus, the context layer shows the state of the hidden layer delayed by one time step.

In order to meet the second constraint, the weights between hidden and context layer are fixed to 1, and the activation of the context units equal the output of the hidden units delayed by one time step (cf. Figure 3.8). By that, the SRN turns into a FFN with additional inputs from the context layer, and any algorithm for FFNs can be used to train it (Elman, 1990).

Let $\mathbf{x}(t)$, $\mathbf{y}(t)$, and $\mathbf{z}(t)$ denote the output vectors of the input, hidden, and output layer at time t and $\mathbf{a}^x(t)$, $\mathbf{a}^y(t)$, and $\mathbf{a}^z(t)$ denote the corresponding network activation vectors. Further, $\mathbf{W}^{yx}$, $\mathbf{W}^{yy}$, and $\mathbf{W}^{yz}$ are the weight matrices for the input to hidden, context to hidden and hidden to output connections (cf. Figure 3.8). Then the forward

pass of the SRN with activation function f_{net} can be written as:

$$a_i^y(t) = \sum_j W_{ij}^{yx} x_j(t) + \sum_j W_{ij}^{yy} y_j(t-1), \tag{3.14}$$

$$y_i(t) = f_{\text{net}}\left(a_i^y(t)\right), \tag{3.15}$$

$$a_i^z(t) = \sum_j W_{ij}^{zy} y_j(t), \tag{3.16}$$

$$z_i(t) = f_{\text{net}}\left(a_i^z(t)\right). \tag{3.17}$$

The layers are updated one after another for each input $\mathbf{x}(t)$. Further, each network output $\mathbf{z}(t)$ is compared to a target vector. At this point the reinforcement learning paradigm comes into play. Instead of providing a full target vector to the network only the output of the winning unit is evaluated. The way this winning unit is found will be explained later in Section 3.2.2. The selected output unit or rather the corresponding action is reinforced either with $r = 1$ for the desired action or else with $r = 0$. Therefore, the direct error signal of the winning unit k is computed by

$$e(t) = z_k(t) - r\ . \tag{3.18}$$

This error signal is used to build the complete error signal for the output layer. The derivatives $\partial E/\partial a_i^z$, which are used in standard backpropagation, are overwritten with the focus on the winning unit k

$$\frac{\partial E}{\partial a_i^z} = \begin{cases} f'_{\text{net}}\big(a_i^z(t)\big)e(t) & i = k \\ 0 & i \neq k \end{cases}\ . \tag{3.19}$$

The error signal for the hidden layer and the weight updates $\Delta\mathbf{W}$ are recursively computed with the standard backpropagation algorithm by

$$\frac{\partial E}{\partial a_j^y} = f'_{\text{net}}\left(a_j^y(t)\right) \sum_i \frac{\partial E}{\partial a_i^z}(t)\, W_{ij}^{zy}, \tag{3.20}$$

$$\Delta W_{ij}^{zy} = -\varepsilon \frac{\partial E}{\partial a_i^z}(t) y_j(t), \tag{3.21}$$

$$\Delta W_{ij}^{yy} = -\varepsilon \frac{\partial E}{\partial a_i^y}(t) y_j(t-1), \tag{3.22}$$

$$\Delta W_{ij}^{yx} = -\varepsilon \frac{\partial E}{\partial a_i^y}(t) x_j(t), \tag{3.23}$$

where ε denotes the learning rate.

So, how does this algorithm correlate with the RL concept? First, the reward r is checked against the expected reward for the actual network output, which yields the direct error signal $e(t)$ (Equation 3.18). The so called *action* of the network is simply the selected output unit k and its *value* is the output activation level $z_k(t)$. The *policy* of the agent is represented by the configuration of the network. It maps the input $\mathbf{x}(t)$ (*states*) onto the output $\mathbf{z}(t)$ (*actions*).

3.2.2 Simulation on the Conditional Associative Learning Task

In this Section, the SRN is trained on the conditional associative learning task. We will see that the network is sensitive to task-irrelevant temporal information and investigate the influence of the temporal order of the input sequence on the learning performance.

Task Formalisation

Before the network can be trained, one needs to formulate a mathematical equivalent to the experiment that was done with the human subjects (cf. Section 2.2.3). Eight different symbols (representing fractal objects) $S_1, \ldots, S_8$ will be presented to the network. The network has to associate each input with one of four predefined actions (representing the motor response) $A_1, \ldots, A_4$. In other words, the network learns the mapping $S_i \rightarrow A_j$. The symbols are presented cyclically to the network. After each symbol the network gets a scalar feedback ($r = 1$ or $r = 0$) for the selected action (Equation 3.18), and the resulting error is propagated back according to the training process described above. Since each action is evaluated right away, the process corresponds to *online* learning.

Network Configuration

Figure 3.9 shows the general structure of the used SRN. The input symbols are 1-of-8 coded. The input layer consists of eight input units which simply excite or not, given the coding of the symbol. Thus, for the k^{th} symbol we get

$$x_i(t) = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases} \qquad i = 1, \ldots, 8 \tag{3.24}$$

at the output of the input layer. The hidden layer has the *hyperbolic tangent* as activation function. Hence, the output of the hidden layer is

$$y_i(t) = \tanh\left(\beta a_i^y(t)\right) \qquad i = 1, \ldots, 3. \tag{3.25}$$

Each hidden layer unit is connected to one corresponding unit in the context layer. The connection weights are fixed to 1. The context units are fully connected to the hidden layer providing it with the hidden layer output of the previous time step. Eight symbols (binary coded) can be represented by three bits. Thus, the number of units in the hidden layer is set to three. Furthermore, the hidden units are fully connected to the four output units. As for the input, a 1-of-N coding is used to represent the network's output. Each of the four outputs symbolises one of four possible actions. Since the output units shall generate positive real values, their activation function is the *logistic sigmoid* function. The network output is

$$z_i(t) = \frac{1}{1 + e^{-\left(\beta a_i^z(t)\right)}} \qquad i = 1, \ldots, 4. \tag{3.26}$$

The form of the activation function is specified by the network designer and depends on the expected range of activation levels at the input and output of the layers. The

parameter β in Equations 3.25 and 3.26 controls the slope of the activation functions and was set to $\beta = 1$, since weight changes always allow to dynamically rescale this factor. The weights in the forward directed connections were initialised randomly from a uniform distribution in the range of $(-1, 1)$. The learning rate in all simulations was set to $\varepsilon = 0.1$.

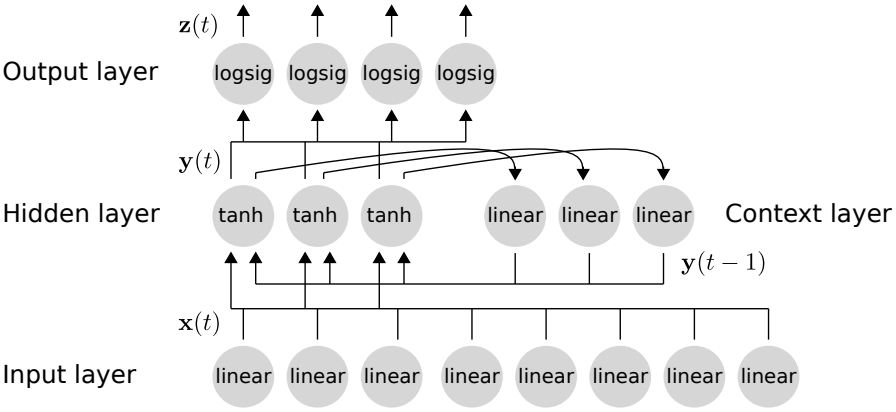


Figure 3.9: Configuration of the SRN used in the simulations on the conditional associative learning task. The input layer consists of eight *linear* input units that reproduce the coding of the input symbol. Further, three hidden units with the *hyperbolic tangent* (tanh) activation function are fully connected to the four output units with *logistic sigmoid* (logsig) activation function. The state of the hidden layer is stored in the context layer and provided as additional input to the hidden layer in the following time step.

Action Selection Method

The winning output, or rather the corresponding action, is rewarded either with $r = 1$ for the correct action or else with $r = 0$. But, how to find the winning unit?

If we engage in the interpretation of the outputs $\mathbf{z}(t)$ as *action values*, like they were defined by Sutton & Barto (1998), then $\mathbf{z}(t)$ represents the estimated reward the agent expects for the action. Thereby, the selection of the winning unit turns into the action selection problem of the RL *Action-Value* method. This directly leads to the *exploration-exploitation* problem, which is the relation between the exploration of new possibilities and the exploitation of old certainties.

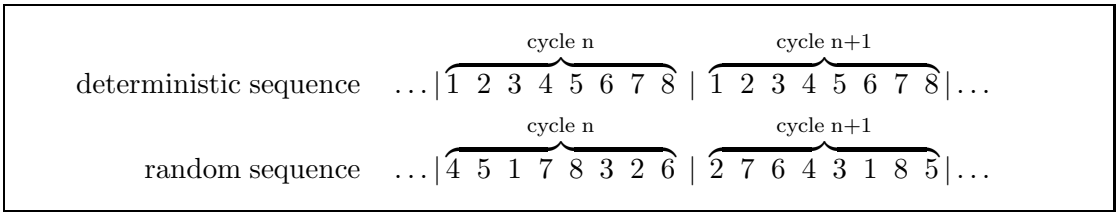
In the task at hand, each symbol is associated to one single action which guarantees the highest possible reward. There is no use to accept a lower reward on the present symbol to earn a higher reward on a later one. That motivates the use of the so called *greedy* action selection policy. This means, the winning output k is simply the one with the current highest estimated reward:

$$k = \arg\max_i z_i(t) \qquad i = 1, \ldots, 4. \tag{3.27}$$

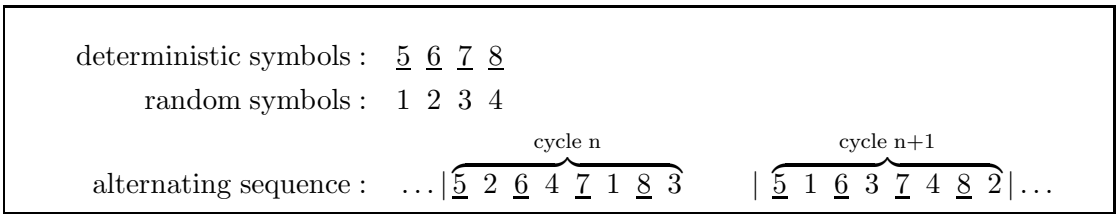
Sequences with Different Temporal Dependencies

The simulation of the associative learning task was done with three different symbol sequences. This variation of the temporal characteristics allows us to study the implicit learning of different types of task-irrelevant temporal contexts in SRNs.

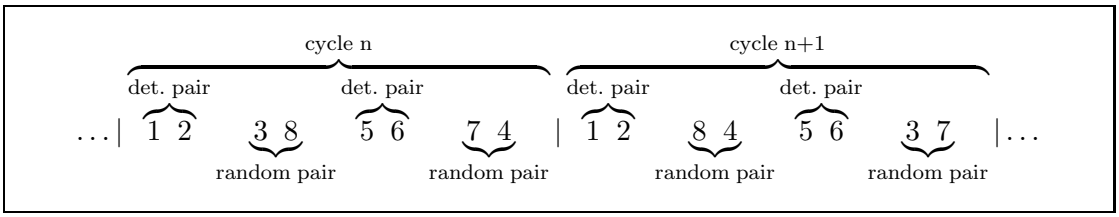
Deterministic vs. Random Sequence: This constellation is the same as used in the original experiment of Hamid et al. (2010) (cf. Section 2.2.3 and Figure 2.4). During each cycle all eight symbols are shown. The simplest case is a deterministic order of the symbols, such that the first cycle simply is repeated. This implies a very strong temporal relationship between the single symbols, each has the same successor for the whole experiment. The worst case is a random order in each cycle, which denotes a weak temporal correlation between the symbols.



Alternating Deterministic and Random Symbols: The 1st, 3rd, 5th, and 7th positions are assigned to certain fixed symbols. The remaining symbols are randomly set to the positions between them. That is, each deterministic symbol is followed by one random symbol.



Deterministic vs. Random Pairs: Again, eight symbols are presented cyclically to the network. In every cycle the 1st/2nd and the 5th/6th positions are assigned to certain pairs of symbols and repeated in every cycle. The other positions are randomly assigned to the remaining symbols. In that way, a sequence with two deterministic and two random pairs is being generated.



Results

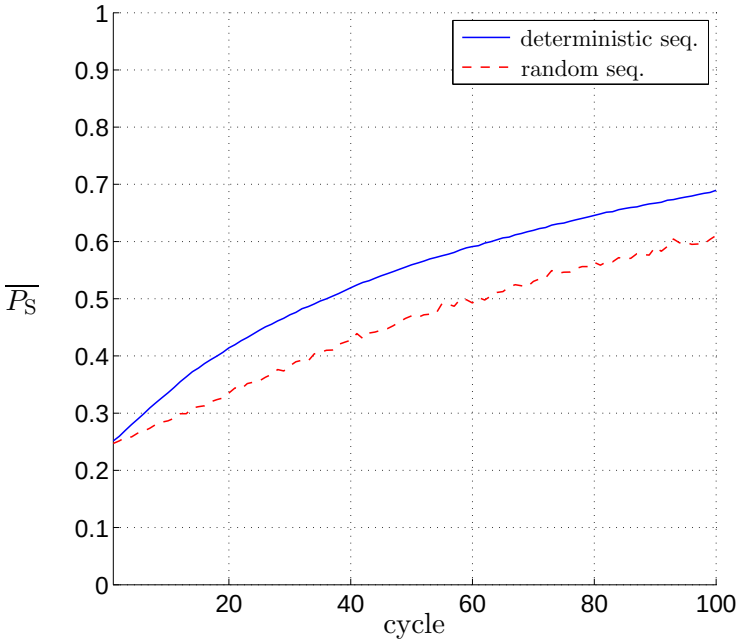
To measure the success of the network, the output is evaluated in terms of the probability of success (P_S) for each training cycle; that is, the number of correctly assigned symbols divided by the total number of symbols per cycle. When training starts the probability to excite the correct output is one out of four ($P_S = 0.25$) for each symbol in the cycle. During learning the probability of success should rise, since it gets more likely that the network takes the correct action.

The network weights are initialised randomly, therefore the learning curves for single networks may differ considerably. As the general behaviour of the network is in the focus of interest, 100 networks are trained and the mean of their probability of success ($\overline{P_S}$) is calculated.

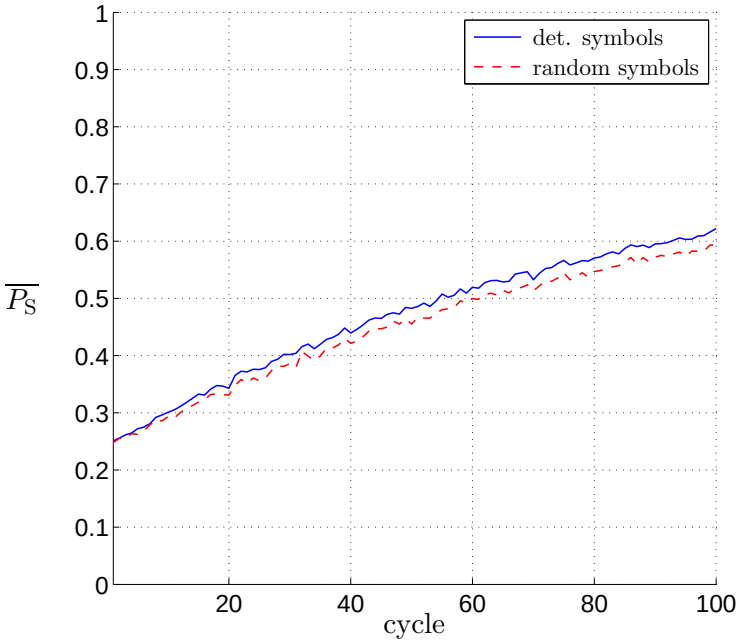
Figure 3.10a shows the learning curve of the SRNs on the deterministic and random sequence for a training of 100 cycles. One can see that the deterministic sequence is learnt faster than the random one. After 100 training cycles, the difference of the mean success probability $\overline{P_S}$ is 8%. The networks benefit from the temporal relationship between the input symbols even if it is task-irrelevant. This result is qualitatively comparable to the learning curve of the human subjects in Figure 2.4. Both, human subjects and SRNs clearly benefit from the existence of a temporal relationship between the symbols/objects.

Figure 3.10b shows the performance of the SRNs on sequences consisting of alternating deterministic and random symbols. In the plot both kind of symbols are distinguished. The difference of the mean probability of success ($\overline{P_S}$) is small, about 2%. Compared to the complete deterministic sequence (Figure 3.10a) the SRNs perform 5% to 6% worse on single deterministic symbols separated by random symbols (Figure 3.10b). Apparently, the better performance on the completely deterministic order arises from a relation between direct predecessor and successor in a sequence. This conclusion was tested in the third experiment.

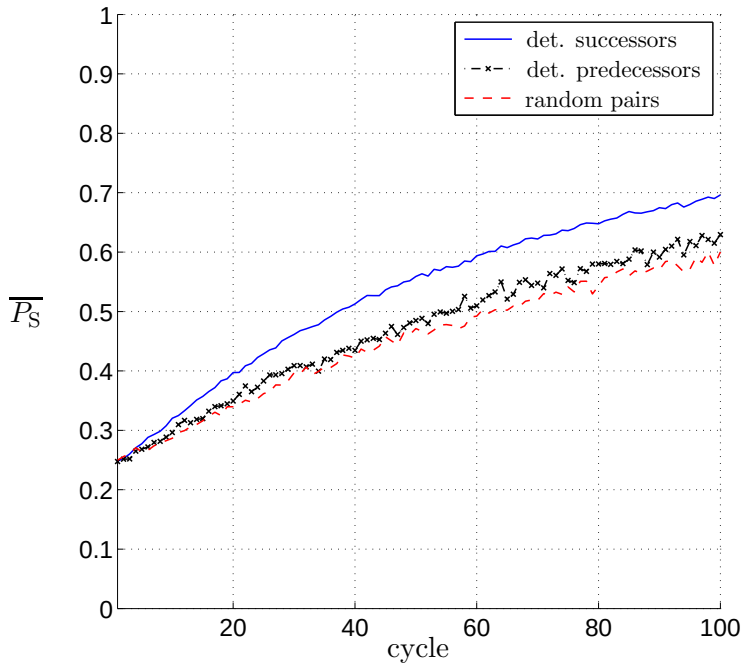
Figure 3.9c shows the performance of the SRNs on sequences of deterministic and random pairs. The different pairs are plotted separately. Once more the network performs better if a strong temporal relationship between the symbols is present. Especially, successors in a deterministic pair are very close connected to their predecessors. The performance on these symbols is comparable to the performance on a completely deterministic sequence (Figure 3.10a). Thus, one concludes that the network is first of all sensitive to direct temporal relations between the present and previous input. A deterministic predecessor in a deterministic pair on the other hand is successor of the previous deterministic successor, thus they are connected to a symbol which lies three steps back in the past. This temporal connection is not recognised by the networks in the simulations. There is no substantial difference in the mean probability of success ($\overline{P_S}$) between deterministic predecessors and random pairs (Figure 3.9c).



(a) Eight symbols were shown in either deterministic or random order. The correct association for the symbols were learnt faster when they were shown in a deterministic sequence.



(b) Deterministic and random symbols in an alternating sequence. The difference between deterministic and random symbols is about 2%.



(c) Deterministic and random pairs in a sequence. Symbols with a strong temporal relationship such as successors in a deterministic pair are learnt faster. The performance on these symbols is comparable to the performance on a completely deterministic sequence (cf. (a)).

Figure 3.8: Learning curve of SRNs on the associative learning task with three different symbol sequences: (a) deterministic vs. random sequence, (b) alternating sequence of single deterministic and random symbols and (c) sequence of deterministic and random pairs of symbols. The mean probability of success $\overline{P_S}$ is averaged over 100 SRNs.

3.2.3 Summary of the Experiment

Comparing the learning curves of the human subjects (cf. Figure 2.4) with those of the SRNs (cf. Figure 3.10a) one can state, that the SRNs perform at least qualitatively like human subjects. There is an evident difference in the learning performance between random and deterministic sequences.

The SRN is endowed with an autoregressive context layer and therefore, it can use temporal information *implicitly*. An SRN uses the temporal context which is inherent in the task, even if it is not necessary to solve the overlying classification problem. The more the temporal context is pronounced, the better the network performs on the task. The simulations suggest that a succession of deterministic symbols is the major momentum for the formation of the temporal context. As one deterministic symbol follows another without interruption by a random one, the learning performance for the successor equals the performance for a deterministic sequence (cf. Figures 3.10a and 3.9c).

For a temporal relation that spans over more than one time step, the influence of the temporal context for the learning performance decreases. If deterministic symbols are separated by one random symbol we observe a slightly better performance than for random symbols only (Figure 3.10b). For deterministic symbols separated by two random symbols we see no substantial difference in the mean probability of success compared to random symbols only. This is shown by the third experiment in Figure 3.9c. The deterministic predecessors in that sequence are separated by two random symbols (random pair) to the next deterministic symbol.

3.3 Discussion of the Models

In this chapter, two different models for conditional associative learning were introduced. On the one hand, the Markov model in Section 3.1 describes learning by means of a stochastic process. The scenario is described with 16 possible states and the assumed rational behaviour gives the state transitions. The parameters of the model (p_{suc}, p_{sr}, p_{or}) represent typical human mistakes. On the other hand, the SRN model in Section 3.2 does not need such assumptions, and can be considered to be closer to the biological example.

Nevertheless, the Markov model with its assumptions allows an analysis of the explicit task of object association with the correct button. It yielded two predictions. First, the probability a subject memorises a success correctly affects mainly the slope of the learning curve. This tells us that a strong reinforcement learning signal is the key to associative learning of a single object. Second, the probability that a subject remembers the learning history on an object is responsible for the long-term development of the success probability. That means, the more complex the task, for instance eight buttons per object instead of four, the lower the chance to finally learn all presented objects. Both statements are reasonable and can be used in the design of future experiments of that kind. In summary, it can be stated that the explicit design of the model requires a number of assumptions that can be called into question, but it also enables us to

make some explicit statements on the experimental setup that may be verified in the laboratory.

In contrast to the Markov model, the SRN model is focused on the implicit learning of the task-irrelevant temporal context that is present in the scenario. It utilizes the sequential order of the input when solving the classification problem, which results in a higher learning rate. A comparable effect was observed for human subjects that implicitly learnt the order of presented objects when they were asked to assign the correct key to them. Further, a direct succession of certain objects is the key element in learning their temporal relation. If the succession is interrupted by one or two random objects, the performance on the deterministic objects decreases. A similar behaviour could also be observed for human subjects trained on such mixed sequences in Hamid et al. (2010).

Finally, the RL model introduced in Hamid et al. (2010) must be mentioned when discussing models of conditional associative learning. It is a model-free RL approach (Sutton & Barto, 1998; Dayan & Niv, 2008) were the response choice is probabilistic. The *expected* reward for a choice is accumulated in the action values which are decreased or increased depending on the *actual* reward for the choice. The key element, which allows the model to learn temporal dependencies, is the fact, that the response choice depends on three action values attached to the actual object and its two predecessors. In that way, the two previous objects influence the response on the current object. Moreover, Hamid et al. (2010) introduced a recognition parameter $0 \leq \gamma \leq 1$ to model confusion about objects. Within the general framework of RL, plus the assumption that two predecessors contribute to the performance on the current object and the recognition parameter γ, the model accounts qualitatively and quantitatively to the behavioural data gathered in the experiments.

In terms of the implicit learning of temporal information, the SRN model is able to reproduce the effect observed during the experiment without making any explicit assumption on the processing of this information. In contrast, the RL model needs the assumption that the last two objects contribute to the learning of the present one.

In conclusion, the question "Which is the best model?" cannot be answered, as the question itself is too general. Regarding the explicit learning of object-button associations, the Markov model could give insights on the different factors that influence this process. The broadly accepted RL idea used by Hamid et al. (2010) led to a model that could reproduce the observed data. For this, one explicit assumption on the nature of the processing of temporal information had to be made and suitable values for the recognition parameter had to be chosen. Last but not least, the SRN model showed the qualitative properties that cover the observed behaviour characteristics without an explicit assumptions concerning the processing of sequential information. For instance, it does not rely on the length of the history window that has to be considered. Instead, the amount of temporal context to be considered is learnt by the model.

4 Representation of Temporal Context in Simple Recurrent Networks

Contents

As we saw in the previous chapters, temporal order plays an important role in human learning. Chapter 3 discussed different models for the special case of the conditional associative learning scenario, where the temporal component is task-irrelevant but still supports learning. In terms of the psychology of learning and biological plausibility, there is no doubt about the need for models based on recurrent networks, since biological neural networks are recurrent. This motivates the use of connectionist models like SRNs (cf. Section 2.4). Further, Section 3.2 emphasized the potential of SRNs to reproduce the effects of implicit sequence learning observed in humans.

We expect a good model to reproduce the experimental data, but further, also to explain it and make verifiable predictions. Therefore, one basic requirement is an understanding of the processes in the network during learning. Heskes & Kappen (1991) have studied the learning process and the learning dynamics of neural networks from a mathematical point of view. They have shown that a nonzero learning parameter enables a network to adapt to changing environments. This leads to fluctuations in the plasticities of the synapses which, in turn, leads to a trade-off between adaptability and accuracy. In Heskes & Kappen (1993), a theory for online learning in artificial neural networks is proposed. Online learning is necessary if not all training patterns are available at once, which is common in natural learning scenarios. The authors again emphasize the conflict between the adaptability and the confidence of the network's representation in a changing environment.

In the cognitive science community, recurrent neural networks are often employed to process symbol sequences that represent natural language structures. Here, the aim

is to study possible neural mechanisms of language processing and aid in development of artificial language processing systems (Čerňanský et al., 2007). In this research, Elman (1991) is the first who tackles the question how complex structural relations like language can be represented in an SRN. He uses a principal component analysis on the activation patterns of the hidden units to reveal the distributed representations that encode grammatical relations.

Čerňanský et al. (2007) investigate the organization of the state space of an SRN that is trained to predict the next symbol in a "language like" sequence. In the evolving representations they find two properties. First, the activations emerging before training are meaningful. They correspond to states of variable memory length Markov models. Second, after training, the prediction is based on grammatical subcategories rather than individual words.

In the following, the mechanisms of implicit sequence learning in SRN shall be investigated. It is shown how implicitly learnt temporal information is represented in SRNs. For this purpose, the influence of the sequential input during training and testing is examined. Further, the trained networks are tested on different input sequences under different conditions, for instance with and without context memory. The work presented here is published as contribution to the "International Joint Conference on Computational Intelligence" (Glüge et al., 2010) and parts of the text are taken over verbatim.

4.1 The 4-2-4 Encoder Simple Recurrent Network

To be able to study detailed effects of sequence learning in SRNs, a small network and an auto-encoding task are used. Auto-encoders were introduced by Rumelhart et al. (1986b). In general, the aim is to learn an efficient coding for a set of data. More precisely, the problem is to learn an encoding of an N bit pattern into $\log_2 N$ bit pattern and further, learn to decode this representation into the output pattern (Rumelhart et al., 1986b).

For the problem at hand, the task is designed according to the idea of the conditional associative learning scenario discussed earlier (cf. Section 2.2.3). A *task-irrelevant* implicit sequence learning process shall support the explicit learning of associations (input - output pairings).

4.1.1 Encoding Task

The encoding task is the conversion of four 1-of-4 coded input vectors into a binary representation and vice versa. For the network, the task is first to find a mapping for the 1-of-4 coded input into a binary code. The second part is the retrieval of the binary coded input into the 1-of-4 coded output. Table 4.1 shows one possible solution for the problem.

Table 4.1: 4-2-4 encoding task: a four bit input vector has to be represented by two bit and vice versa.

Input	Binary Code	Output
0001	00	0001
0010	01	0010
0100	10	0100
1000	11	1000

Attention should be paid to the fact that the encoding is independent of the sequence of input vectors. The network learns a direct mapping between input and output, which represents the explicit learning of associations. Nevertheless, the SRN implicitly learns a temporal relation between inputs and uses this to its advantage for memorising. Figure 4.1 illustrates the learning situation.

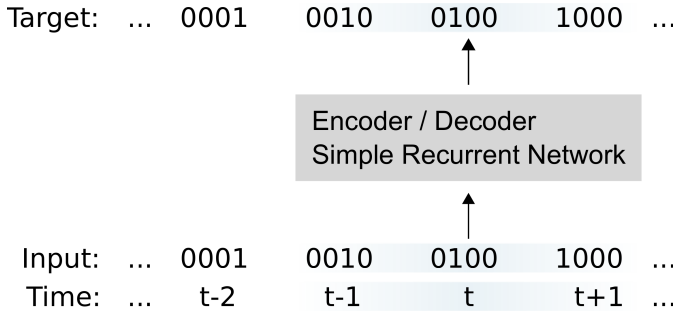


Figure 4.1: The encoding-decoding of the vectors is independent of their sequential presentation, even though, the succession of the vectors supports learning.

4.1.2 Network Configuration

Figure 4.2 shows the network used in the simulations on the encoding task. The same notation as in Section 3.2.1 is used. $\mathbf{x}(t)$, $\mathbf{y}(t)$, and $\mathbf{z}(t)$ denote the output vectors of the input, hidden, and output layer at time t. The corresponding network activation vectors of the input, hidden, and output layer are represented by $\mathbf{a}^x(t)$, $\mathbf{a}^y(t)$, and $\mathbf{a}^z(t)$. The weight matrices for the input to hidden, context to hidden, and hidden to output connections are denoted $\mathbf{W}^{yx}$, $\mathbf{W}^{yy}$, and $\mathbf{W}^{yz}$. With the activation function f_{net} the forward processing in the network equals those in Section 3.2.1 and is described by Equation 3.14, 3.15, 3.16, and 3.17.

The input layer consists of four input units which simply excite or not, given the coding of the input vector. The hidden layer has the *hyperbolic tangent* as activation function. Hence, the output of the hidden layer is

$$y_i(t) = \tanh\left(a_i^y(t)\right) \qquad i = 1, 2. \tag{4.1}$$

Each hidden layer unit is connected to one corresponding unit in the context layer. The connection weights are fixed to 1. The context units are fully connected to the hidden

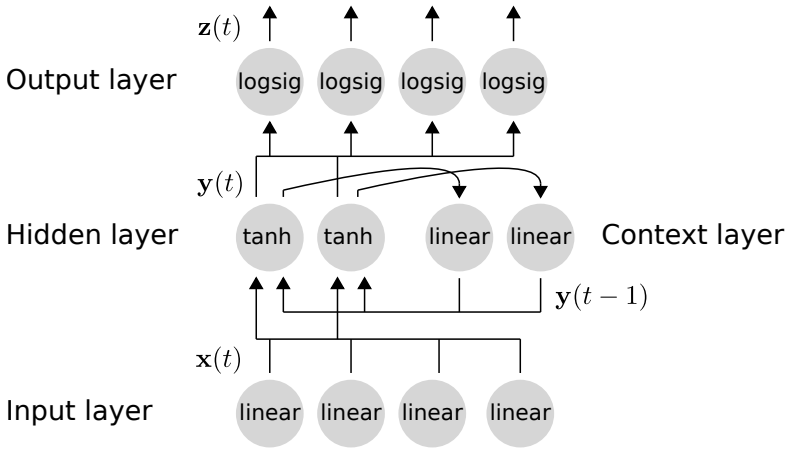


Figure 4.2: Configuration of the SRN used on the 4-2-4 encoding task. The input is directly forwarded by four *linear* input units. Then, the input is represented by two hidden units with a *hyperbolic tangent* (tanh) activation function and forwarded to four output units with *logistic sigmoid* (logsig) activation function. The state of the hidden layer is stored in the context layer and provided as additional input to the hidden layer in the following time step.

layer, providing it with the hidden layer output from the previous time step. Four different input vectors can be represented by two bits. Since the network shall code the input vectors into a binary representation, the number of units in the hidden layer is set to two. Furthermore, the two hidden units are fully connected to four output units. This corresponds to the retrieval of the binary coded input into a 1-of-4 coded output. As the output units should generate values between 0 and 1 their activation function is the *logistic sigmoid* function. The network output is

$$z_i(t) = \frac{1}{1 + e^{-a_i^z(t)}} \qquad i = 1, \ldots, 4. \tag{4.2}$$

Note that the hyperbolic tangent at the hidden layer produces values between -1 and 1. Therefore, the internal representation for the input will consist of values in the interval $(-1, 1)$ and not $(0, 1)$ as in Table 4.1. The central interval about zero of this activation function supports gradient based learning, as it allows a better flow of the error signal backwards compared to the logistic sigmoid. Nevertheless, it is still required that the internal representation be *binary* in the sense that the hidden layer has to produce coding values for a uniquely distinguishable mapping at the output layer. The input and output was presented to the network as shown in Table 4.1 and Figure 4.1.

4.1.3 Network Training

As pointed out earlier (cf. Section 3.2.1), the SRN can be seen as a feedforward network with additional inputs from the context layer and any algorithm for feedforward networks can be used to train it (Elman, 1990). To use the backpropagation algorithm the context

units must be initialised with some activation for the forward propagation of the first training vector. Here, the initial values were set to zero. Further, the activation levels of the hidden layer were stored in the context layer after each backpropagation phase. Thus, the context layer shows the state of the hidden layer delayed by one time step.

After each input, the network output is compared to the desired output and the mean square error is propagated back through the network. The weights are updated with the constant learning rate $\varepsilon = 0.1$. Since each output is evaluated right away, the process corresponds to *online* learning. As already pointed out in the introduction to this chapter, online learning appears when not all training patterns are available at once. It is a biologically plausible scenario, because it is simply most common in natural situations.

The weights are initialised with uniformly distributed random values in the interval $[-0.3, 0.3]$, apart from the fixed hidden to context layer weights. Learning rate and weight initialisation interval are chosen according to preliminary tests, where the combination that yielded best training results after 1000 training cycles was chosen.

Since we are interested in the network's ability of implicit sequence learning, the training input is presented in two different ways: a sequential and a random one. One training cycle consists of a presentation of all four input vectors that are shown to the network one after another. For the case of a deterministic order, the first cycle is repeated for the whole training. This implies a strong temporal relationship between the input vectors since each one has a fixed successor. For the case of a random order in each cycle the temporal correlation between the input vectors is very weak.

If the numbers from one to four denote the input vectors, one can describe the two types of sequences as follows:

deterministic sequence $\ldots |\overbrace{1\ 2\ 3\ 4}^{\text{cycle n}}|\overbrace{1\ 2\ 3\ 4}^{\text{cycle n+1}}|\overbrace{1\ 2\ 3\ 4}^{\text{cycle n+2}}|\ldots$

random sequence $\ldots |\overbrace{4\ 3\ 1\ 2}^{\text{cycle n}}|\overbrace{2\ 1\ 3\ 4}^{\text{cycle n+1}}|\overbrace{3\ 2\ 1\ 4}^{\text{cycle n+2}}|\ldots$

Each network is trained for 1000 cycles. Hence, every input vector is shown 1000 times to the network. This results in 4000 training steps or rather 4000 weight updates.

To measure the success of the network the output is evaluated according to the winner-take-all principle. The unit with the highest activation is counted as 1, the remaining units as 0. Thus, the network's output is always mapped onto a corresponding target vector. Then the network output can be expressed in terms of the probability of success (P_S) for each training cycle. Figure 4.3 shows this procedure.

When training starts, the probability to excite the correct output is one out of four ($P_S = 0.25$). At the end of training the network should have learnt the coding and always deliver the target vector, therefore one expects $P_S = 1$.

Since the weights are initialised randomly, the learning curves for single networks may differ considerably. To compare the general behaviour of the network, 100 networks are

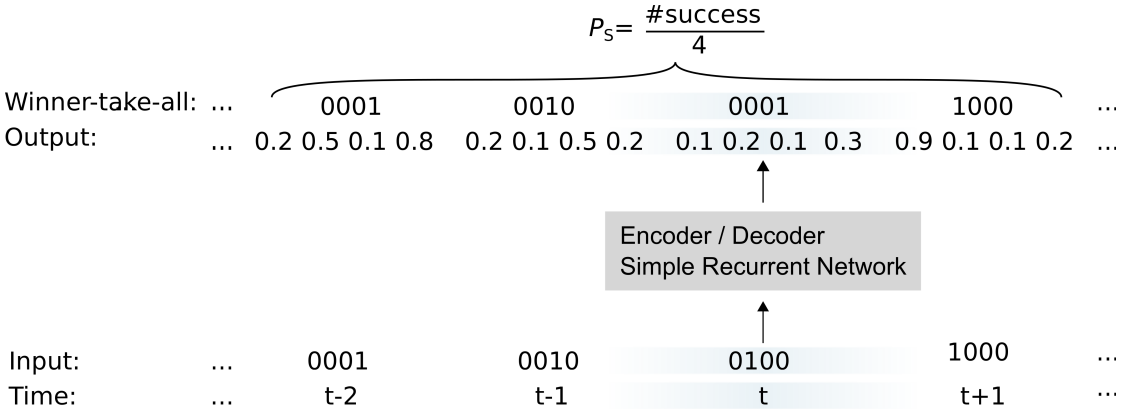


Figure 4.3: Measure for the success of the network: the output is evaluated according to the winner-take-all principle which yields a corresponding target vector. Probability of success (P_S) is calculated for each training cycle

trained on each type of input sequence. Afterwards the mean probability of success is calculated over the 100 networks ($\overline{P_S}$) for the two test cases.

4.2 Results of the Training

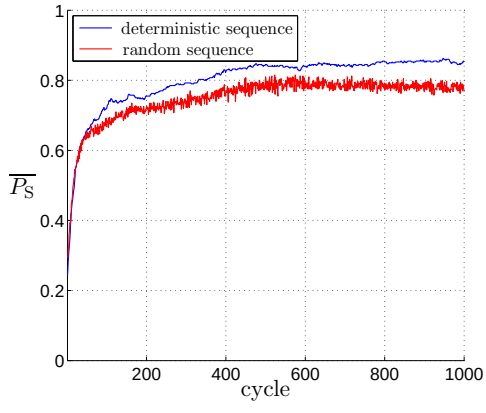
In the following we have a look on the influence of the temporal context during the training of the SRN. This includes, first of all, the ability to learn the encoding task. Further, the solutions found by the network and the role of the context layer weights are examined. The performance of the networks during testing is treated separately in the next section.

In Figure 4.4a, the mean probability of success $\overline{P_S}$ during training is plotted against the number of training cycles. In general, the networks perform better on a deterministic input sequence than on a random one. In both cases, however, the expected $\overline{P_S} = 1$ is not reached, which will be explained in the following.

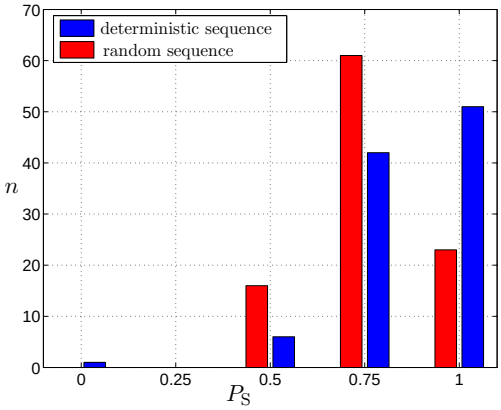
Figure 4.4b shows the distribution of the individual success probability P_S for the 100 networks after training. The number of networks n is plotted against the final probability of success P_S. Trained with a deterministic sequence one half of the networks ($n = 51$) learnt the encoding completely ($P_S = 1$ by the end of the training). On the other hand, only 23 networks could succeed if trained with a random sequence.

In summary, it is more likely that a network trained with a deterministic sequence succeeds in training. The reason is the temporal correlation between the input vectors. This extra information, which is only provided in the deterministic sequence, raises the probability of the SRN to learn the task.

Apparently, the learning task is quite demanding, since a high percentage of the networks got stuck in local minima or plateaus. Those networks that did not succeed in training learnt an unstable solution. A more sophisticated training algorithm, for instance a variable learning rate, a better measure for the network error, etc., might



(a) Mean probability of success $\overline{P_S}$ plotted against the training cycles.



(b) Distribution of final success probabilities. The number of networks n of 100 is shown against the probability of success P_S.

Figure 4.4: Success of the training of 100 SRNs trained on the 4-2-4 encoding task. Not all networks were able to learn the task completely. However, the chance to learn the task is higher given a deterministic than a random sequence.

be able to avoid unsuccessful training. Since the influence of task irrelevant temporal information and the role of the context layer are in the focus of this study, the learning algorithm was not optimised. By that, the training procedure differs *only* in the sequential presentation of training inputs.

Figure 4.5 exemplarily shows the output of the two hidden units during training for a network that did not find a distinct encoding. The output of each hidden unit is plotted for each specific input. The combination of the two hidden units' outputs represents the network's internal state and therefore, the coding of the input vector. At the beginning of the training all inputs are represented by values around zero. At the end of the training, one can see that input vector 2 and 4 are well distinguishable, while the inputs 1 and 3 result in a nearly identical activation pattern. The network did not learn to distinguish these input vectors.

The four input vectors have a representation as hidden units' output pairs (y_1, y_2). Figure 4.6 shows this constellation after successful (crosses) and unsuccessful (dots) training. For the case of successful training, each pair of outputs lies in a different quadrant of the y_1-y_2 coordinate system, which makes them easily separable. For the unsuccessful training two inputs result in hidden layer outputs that are very close to each other; here in the upper left quadrant of the y_1-y_2 plane.

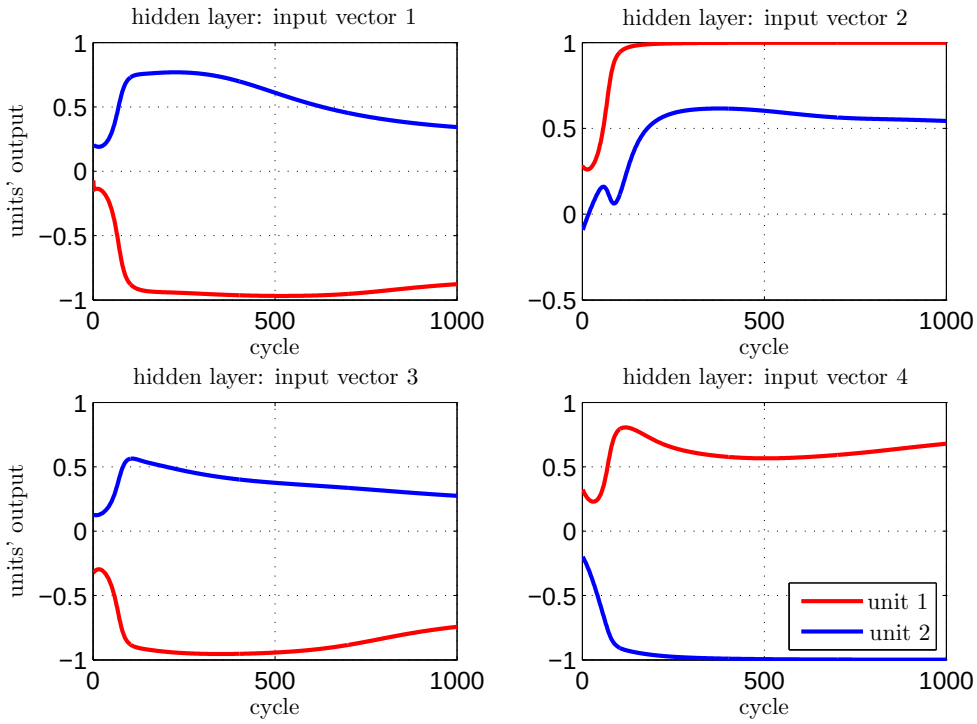


Figure 4.5: Hidden layer output during training of a network that did not succeed in training. For input vector 1 and 3 no distinct coding was learnt.

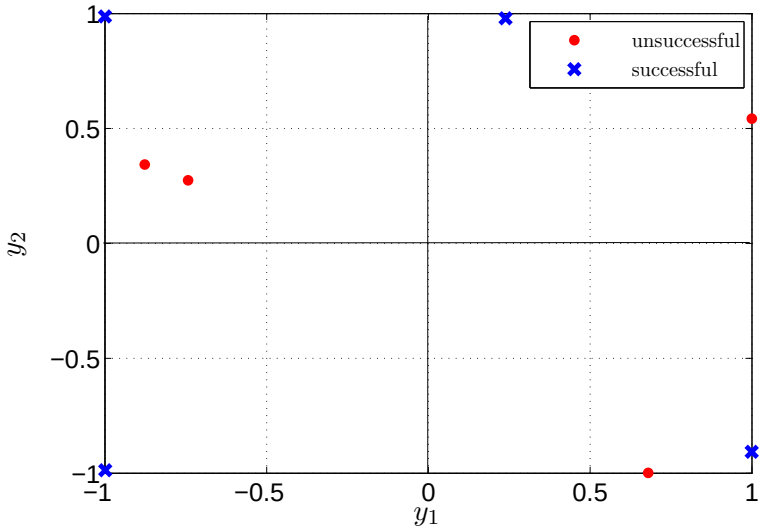


Figure 4.6: Hidden layer outputs for each of the four input vectors after training. One network succeeded in training (crosses) and one did not succeed in training (dots).

Weights of the Context Layer

The discussion above shows that the temporal relation of the input vectors was used during learning. But, how is this relation represented in the network? The answer lies in the weights of the network, since this is the only parameter that is changed during training. Further, the key component to sequence learning is the context layer, thus the development of the weights of this layer should be investigated.

Two context units are fully connected to two hidden units. This results in four weights in the context weight matrix, W_{ij}^{yy} where $i, j = \{1, 2\}$ (cf. Equation 3.14). Figure 4.7 exemplarily shows the weights of the context layer during training. The four context weights of two networks that solved the task ($P_S = 1$ at the end of the training) are plotted over the training steps. Each training cycle consists of four input vectors. A training of 1000 cycles results in 4000 weight updates. The red lines show the weights of a network trained with a random sequence, the blue lines those of a network trained with a deterministic sequence. Each line represents the development of a single weight of the weight matrix.

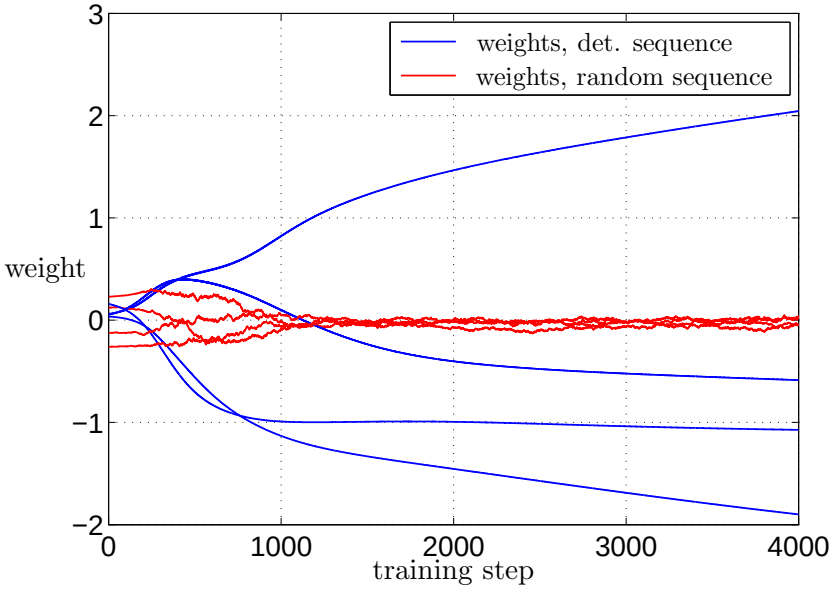


Figure 4.7: Context layer weights of two networks that learnt the task. The blue and red lines show the development of the single weights during training on a deterministic sequence and a random sequence, respectively.

For a deterministic sequence the weights take a value between -2 and 2. Longer training would result in higher weights, since the network tries to generate exactly 1 at the output. This value is never reached by the activation function of the output units.

The random sequence leads to vanishing context weights, which means the weights tend towards zero. Put another way, the network learns that there is no temporal

dependency in the input. This leads to the remarkable result that the SRN turns into a standard feed-forward network omitting contextual information.

4.3 Results of the Testing

Yet, the differences that occurred during training with a deterministic and a random sequence were discussed. In the following, the performance of the networks during testing shall be investigated. Those networks that solved the task at least in the last training cycle ($P_S = 1$) were chosen for testing. This criterion turns out to not guarantee that the solution that the network has found is stable. The training in Section 4.2 yielded 51 successfully trained networks with a deterministic sequence and 23 successfully trained networks with a random sequence (cf. Figure 4.4b).

To study how the existence of a temporal context during training influences the generalisation, the networks were tested on *both* sequences for 1000 cycles. Therefore, we have to distinguish four cases: (i) trained deterministic and tested deterministic, (ii) trained deterministic and tested random, (iii) trained random and tested random and (iv) trained random and tested deterministic.

Table 4.2 shows that only cases were the sequence differs between training and testing (ii, iv) is of further interest. Networks tested with their training sequence unsurprisingly perform very well in the test run.

On the other hand, networks trained with the deterministic sequence performed very poor in the test with a random sequence ($\overline{P_S} \approx 0.5$). The networks did not learn just the input itself, but also the temporal correlation between inputs. Tested on a random sequence, the stored temporal information is misleading. For instance, if the network learnt that input vector 1 is followed by 2 but in the next step vector 3 is presented, then the network has to process two conflicting informations. The input layer indicates vector 3 to the hidden layer but the *context* at the present time indicates vector 2 to the hidden layer. The test result shows that this contextual information is not just some add on but absolutely necessary for the networks to solve the task. Since just one type of input sequence was shown during training, the networks did not generalise to other types of sequences, which results in a poor overall performance in the test on a random sequence.

In contrast, those networks that were trained with a random sequence turned into feed-forward networks during training (cf. Section 4.2 and Figure 4.7). Therefore, they could deal with any kind of input sequence since the encoding of an input is independent of its context.

The result of the test run with the random sequence shows that networks trained with the deterministic sequence heavily rely on the temporal structure of the input. To investigate the influence of the context layer the networks are tested again. This time the context weights are set to zero. Thereby, the networks trained with a deterministic sequence lose the previously learnt temporal correlation between input vectors. Those networks trained with a random sequence have zero context weights already, thus, the replacement by zero should have no effect.

Table 4.2: Results of testing: mean probability of success for the four combinations of training and testing.

	$\overline{P_S}$	training	
		det. sequence	random sequence
testing	det. sequence	0.9657	0.9457
	random sequence	≈ 0.5	≈ 0.95

The modified networks were tested with the deterministic sequence for 1000 cycles. Table 4.3 shows the mean probability of success in this test. Compared against Table 4.2, the performance of the networks trained with a deterministic sequence decreases dramatically, 97% with, and 52% without context layer. Networks trained with a random sequence still perform very well, 95% with, and 92% without context layer.

Table 4.3: Results of testing *without* context layer: mean probability of success when omitting context layer weights after training.

	$\overline{P_S}$	training	
		det. sequence	random sequence
testing	det. sequence	0.5196	0.9239

4.4 Representation of Temporal Context

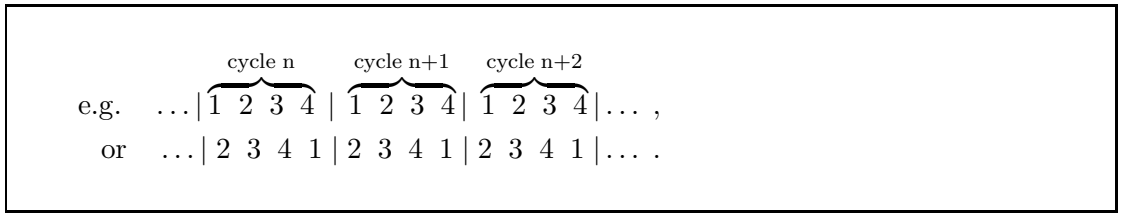
To understand how the context layer weights encode sequential information it is helpful to have a look onto the activation pattern of the network. As recurrent networks tend to oscillate, they produce an activation in the output layer, even if the input is zero. In the following we have a look on these activations. The sequences that are generated at the networks' output can be observed after one initial input.

Those networks trained with a random sequence do not oscillate, since the feedback connections are zero. However, networks trained with a deterministic sequence produce a variety of sequences. For the networks that succeeded training with the deterministic sequence ($n = 51$, cf. Figure 4.4b), one could observe four different classes of oscillations at their output. The types of sequences are:

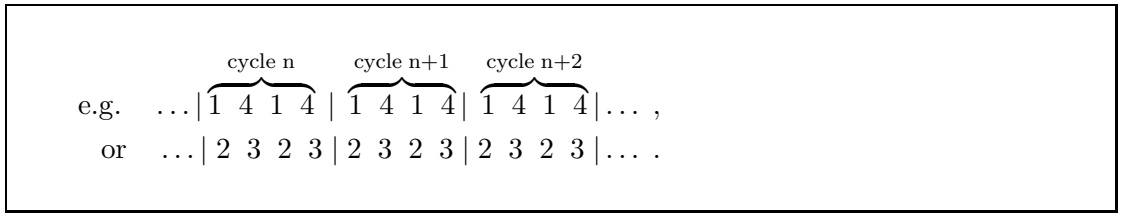
1. full cycle oscillation (FCO),
2. half cycle oscillation (HCO),
3. constant after transient oscillation (CTO),
4. other.

This distinction is not universally valid, but seems to be adequate to classify the observed behaviour.

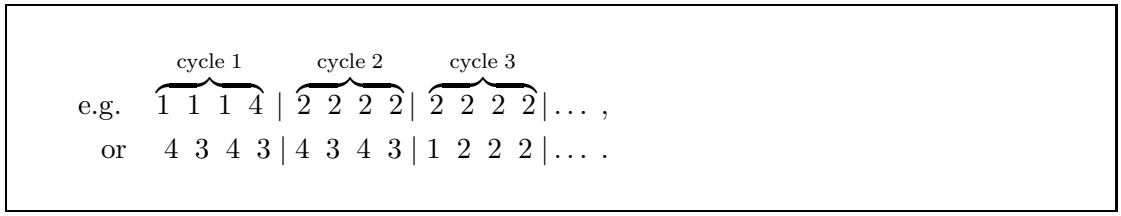
Full Cycle Oscillation is a sequence that reproduces the trained input sequence completely. In every cycle all four training inputs appear in the order of the deterministic sequence (1, 2, 3, 4), but the cycle does not necessarily start with "1".



Half Cycle Oscillation is a sequence that contains some part of the trained input sequence with the period of a half cycle. Two input patterns appear, alternating two times per cycle.



Constant after Transient Oscillation is a sequence that takes a constant value after a transient oscillation of about two cycles,



Other sequences are those that do not fit into the patterns above, like sequences that reproduce the trained input with a blemish, or sequences that produce an oscillation with a period that spans over several cycles.

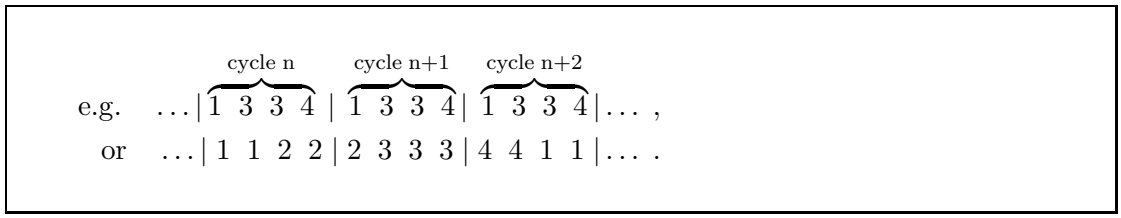


Table 4.4 shows the distribution of the generated sequences over the classes of sequences. Most of the networks ($n = 23$) generate the sequence presented during training after activation with one *single* input pattern. In terms of sequence learning these networks performed best. Another group of networks ($n = 17$) has an oscillating behaviour

with the period of a half cycle. This can be interpreted as a clock signal with two pulses/beats per cycle. A constant output is produced by five networks after a short transient oscillation. The remaining networks ($n = 6$) produced oscillations that do not fit into the aforementioned classes.

Table 4.4: Distribution of sequences generated by the 51 networks that were successfully trained with deterministic sequence.

class	FCO	HCO	CTO	other
number of networks n	23	17	5	6

The key component to the generation of an oscillation by the network is the weight matrix of the context layer. After one initial input, the input layer makes no further contribution to the processing in the network. The output layer provides the encoding from the binary representation of the network state into the 1–of–4 coded representation at the output. The sequence of network states is solely generated by the interplay of the hidden and context layer. Figure 4.8 shows this inner part of the SRN.

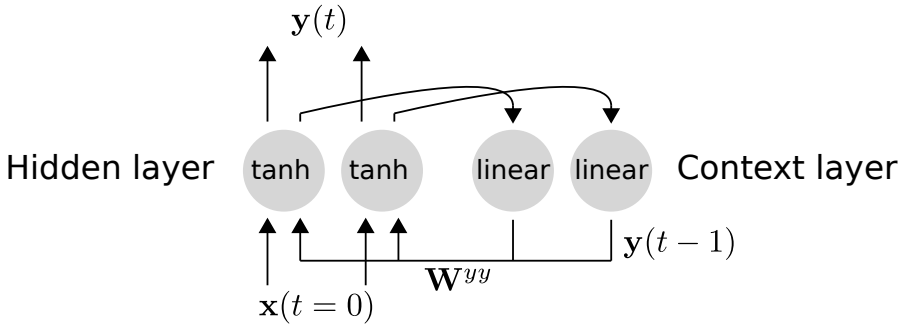


Figure 4.8: Interaction of Hidden and Context Layer

The process of sequence generation can be described by

$$\begin{aligned} y_1(t) &= \tanh\left(W_{11}^{yy} y_1(t-1) + W_{12}^{yy} y_2(t-1)\right), \\ y_2(t) &= \tanh\left(W_{21}^{yy} y_1(t-1) + W_{22}^{yy} y_2(t-1)\right), \end{aligned} \tag{4.3}$$

with initialisation

$$y_i(t=0) = \tanh\left(\sum_j W_{ij}^{yx} x_j(t=0)\right) \qquad j = 1,\ldots,4 \;\; i = 1,2. \tag{4.4}$$

After the initial input, the network state $\mathbf{y}(t)$ depends only on the last state $\mathbf{y}(t-1)$. The transition of one state to another is controlled by the context weight matrix $\mathbf{W}^{yy}$. From this follows that the properties of $\mathbf{W}^{yy}$ determine which sequence the network generates, or in other words, which sequence the network learnt during training.

Properties of the Context Matrix

By the polar decomposition of a matrix it is possible to separate the matrix into a component that *stretches* the space along a set of orthogonal axes and a *rotation* (Conway, 1990). The polar decomposition of a real valued matrix $\mathbf{A}$ has the form

$$\mathbf{A} = \mathbf{RS}, \tag{4.5}$$

where $\mathbf{R}$ is a orthogonal matrix, and $\mathbf{S}$ is a positive-semidefinite symmetric matrix. The matrix $\mathbf{S}$ represents the component that stretches the space while $\mathbf{R}$ represents the rotation. The matrix $\mathbf{S}$ is given by

$$\mathbf{S} = \sqrt{\mathbf{A}^*\mathbf{A}}, \tag{4.6}$$

where $\mathbf{A}^*$ denotes the conjugate transpose of $\mathbf{A}$. If $\mathbf{A}$ is invertible, then the matrix $\mathbf{R}$ is given by

$$\mathbf{R} = \mathbf{AS}^{-1} \ . \tag{4.7}$$

The polar decomposition can be used to extract some properties of the context weight matrix that are related to the generation of the specific types of sequences.

Full Cycle Oscillation: The component of the context matrix $\mathbf{W}^{yy}$ that represents the rotation, takes the form of a rotation matrix that acts as a rotation in Euclidean space. For example

$$\mathbf{R}^{yy} = \begin{pmatrix} \cos(\Theta) & -\sin(\Theta) \\ \sin(\Theta) & \cos(\Theta) \end{pmatrix}. \tag{4.8}$$

By that the generation of a sequence (Equation 4.3) turns into

$$\begin{aligned} y_1(t) &= \tanh\left(R^{yy}_{11} y_1(t-1) + R^{yy}_{12} y_2(t-1)\right), \\ y_2(t) &= \tanh\left(R^{yy}_{21} y_1(t-1) + R^{yy}_{22} y_2(t-1)\right), \end{aligned} \tag{4.9}$$

which is a rotation of vector $\mathbf{y}$ in the y_1-y_2–plane (counter)clockwise by an angle of Θ. This view neglects the effect of the component of $\mathbf{W}^{yy}$ that stretches the space. This is possible since the hyperbolic tangent always maps $\mathbf{y}$ on values between -1 and 1. For the observed context matrices the angle Θ lies between 78 and 100 degree. Figure 4.9 illustrates the process of the sequence generation.

At every time step the context matrix rotates $\mathbf{y}$ into a new quadrant, where the hyperbolic tangent maps the single components of the rotated vector onto the nearest 1 or -1. By that $\mathbf{y}$ passes all four quadrants and therefore, one can observe all four training inputs at the output of the network.

Half Cycle Oscillation: For networks that generate an HCO, the component of $\mathbf{W}^{yy}$ that represents the rotation takes the form of a reflection matrix. A reflection matrix is orthogonal with determinant -1. The eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = -1$. In terms of

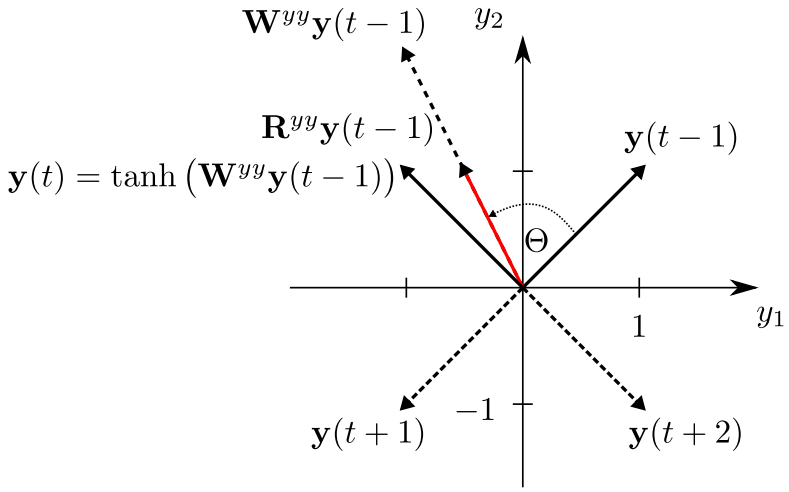


Figure 4.9: Rotation of y by the context matrix $\mathbf{W}^{yy}$

a geometric interpretation $\mathbf{W}^{yy}$ reflects $\mathbf{y}$ from one quadrant into another and reverse. Hence, $\mathbf{y}$ passes two quadrants of the y_1-y_2–plane and therefore, one can observe only two of the training inputs at the output of the network.

Constant after Transient Oscillation and Other: For these types of sequences no property in the context matrix was found that is shared by all observed matrices. Due to the diversity of the single sequences that fall into these classes, one can hardly expect to find one mathematical property that explains all of the observed behaviours.

4.5 Discussion

The aim of the study in this chapter was to investigate the mechanisms of implicit sequence learning in SRNs. Therefore, the learning task was constructed in a way, such that the sequential order of the input was present during learning, but not needed to solve the encoding task. In this sense, the learning of the input sequence was *implicit*. The network was not trained to learn it, like in sequence prediction tasks.

In Section 4.2 it was found, that a fixed sequential order results in a higher learning performance. Comparing Figure 4.4a and Figure 3.10a, one can see that this effect is confirmed in Section 3.2.2, where SRNs showed the same behaviour in the conditional associative learning scenario. Further, the pure number of networks that were able to learn the task illustrates this fact. Nearly twice as many networks learnt the coding if trained with a deterministic sequence than with a random one (cf. Figure 4.4b).

The 4-2-4 auto-encoding task was designed to allow a closer look on the inner processing of the network. That includes a simple task and a small weight matrix. The observation of the weights in the context layer showed the influence of this network layer in the process of implicit sequence learning. A random input sequence provides no sequential information. Hence, a network trained with such input simply learns that

there is no contextual information. This results in vanishing context weights, like it is depicted in Figure 4.7. By that, the SRN turns into a standard feed-forward network.

In addition, the test of the trained networks in Section 4.3 shows the importance of a previously learnt sequential correlation between single inputs. Table 4.2 shows, that the performance of the networks trained with a deterministic sequence heavily depends on the presence of the temporal context. A test without the context layer showed its relevance for the classification task. As the comparison of Tables 4.2 and 4.3 shows, the overall performance decreases dramatically when the context weights are reset to zero.

In general, there is no sequence learning without a context layer which provides the network with some memory. To deduce some specific properties of the context weights, the network input was set to zero. Then, the output sequences that could be observed after just one initial input were investigated in Section 4.4. There is no guarantee that the network learns exactly the presented sequence in the context layer. In fact, the input sequence often can only be reproduced in combination with an activation from the input layer. The variety of generated sequences points out, that the networks find different representations of the sequential information. The context weight matrix is the determining factor in this process.

For the most likely classes of sequences, namely FCO and HCO (cf. Table 4.4), a geometric representation of the sequential information was found. A full cycle through the four states that represent the input vectors is achieved by a rotation through the four quadrants of the two dimensional state-space (cf. Figure 4.9). In a similar manner, the oscillation between two states was realised by a reflection of a state vector into its opposite and reverse. With this result, it is possible to decide whether a network learnt the sequence, on the basis of the properties of its context weight matrix.

One can easily extend this result to a more complex problem like a 8-3-8 encoding, even if the variety of possible solutions grows with every dimension that is added to the state space, and further, the probability to find the optimal solution declines.

5 Learning Long-Term Dependencies in Recurrent Neural Networks

Contents

In the previous chapters the ability of SRNs to implicitly learn temporal information was deduced and highlighted. First, this ability was utilized in the context of the conditional associative learning scenario (cf. Chapter 3). The SRNs turned out to be a possible cognitive model for implicit learning, as it was observed in humans. Then, the focus shifted towards the question, how the temporal information is represented in SRNs. It could be answered partly, by taking the example of a simple encoding task (cf. Chapter 4).

Of course, the application of SRNs must not be restricted to cognitive modelling. Apart from the rather theoretical problems in cognitive biology, there exists a variety of technical applications were implicit learning of contextual information is required.

First and foremost, *sequence prediction* tasks are tackled with the help of recurrent networks in all areas. The aim is to predict the next element of a given series (cf. Section 2.6). Some examples are: load forecasting in electric power systems (Barbounis et al., 2006), automatic speech processing (Varoglu & Hacioglu, 1999), health condition

monitoring of mechanical components (Tian & Zuo, 2010), sunspot series prediction (Park, 2011), network traffic prediction (Bhattacharya et al., 2003), and of course, stock market prediction (Tino et al., 2001).

Another common task is *sequence classification*. In this case, the aim is to learn the class label corresponding to a given sequence (cf. Section 2.6). Again, the area of application is wide, for instance classification of electroencephalography signals (Forney & Anderson, 2011), visual pattern recognition like handwritten numbers (Lee & Song, 1997) and characters (Nishide et al., 2011), seismic signal classification (Park et al., 2011), and pattern recognition in images (Abou-Nasr, 2010).

Unfortunately, recurrent networks have difficulties in learning long-term dependencies, that is, learning a relationship between inputs that may be separated over some time steps. This is due to the so called vanishing gradient problem, which is the fact that error signals propagated backwards become smaller with every time step and network layer, respectively. Therefore the information about the error cannot reach its source and learning cannot take place.

There are basically two ways to circumvent this problem. One idea is to use learning algorithms that simply do not use gradient information, like simulated annealing (Bengio et al., 1994), cellular genetic algorithms (Ku et al., 1999), and the expectation-maximization algorithm (Ma & Ji, 1998). Alternatively, a variety of network architectures was suggested to tackle the vanishing gradient problem, for instance second-order recurrent neural network (Giles et al., 1992), non-linear autoregressive model with exogenous inputs recurrent neural network (NARX) (Lin et al., 1996, 1998), hierarchical recurrent neural network (El Hihi & Bengio, 1995), Long Short-Term Memory network (Hochreiter & Schmidhuber, 1997a), Anticipation Model (Wang, 2001), Echo State Network (Jaeger, 2001, 2002), Latched Recurrent Neural Network (Šter, 2003), Recurrent Multiscale Network (Squartini et al., 2003a,b), a modified distributed adaptive control architecture (Verschure & Althaus, 2003), and Segmented-Memory Recurrent Neural Network (SMRNN) (Chen & Chaudhari, 2004, 2009).

In the following the vanishing gradient problem is introduced in detail and its effect on the gradient based training of SRNs is described. Afterwards, the SMRNN architecture is introduced, and the way it reduces the problem of vanishing gradients is explained. Following this, the extended Real-Time Recurrent Learning (eRTRL) training algorithm, proposed by Chen & Chaudhari (2004), is introduced with the objective to derive its computational complexity. As this complexity is a major problem for applications where considerable large networks are used, I will introduce an alternative learning algorithm for SMRNNs. It is called extended Backpropagation Through Time (eBPTT). Thereafter, the computational complexity is analysed for this algorithm and compared to the complexity of eRTRL.

Finally, both learning algorithms are compared on a benchmark problem, which is designed to test the ability of a network to store information for a certain period of time. eBPTT is found less capable to learn the latching of information for longer time periods in general. However, it nonetheless guarantees better generalisation, that is, higher accuracy on the test set for successfully trained networks. Further, t he computational complexity of eRTRL makes eBPTT the only practicable algorithm for tasks where

rather big networks are required.

The eBPTT algorithm together with an evaluation on the information latching problem is separately published as contribution to the "International Joint Conference on Computational Intelligence" (Glüge et al., 2012)[1].

5.1 The Vanishing Gradient Problem

In order to understand, why learning long-term dependencies with recurrent neural nets is difficult, the vanishing gradient problem is sketched in this section. A more detailed discussion can be found in (Hochreiter et al., 2001).

Recurrent connections are the key element for some form of memory in a neural network. In particular, recurrent networks are able to store representations of recent inputs as an internal state (activation pattern). Even though this property is of great interest in the cognitive science community (cf. Section 2.4), it is of limited use in practical applications, yet. One of the reasons is the complex training that is required for recurrent networks. Often, simpler architectures like feedforward networks with a time window at the input provide advantages in terms of training time, parameter optimization, and therefore, final performance.

The common learning algorithms for recurrent networks are Backpropagation Through Time (BPTT) (Werbos, 1990) and Real-Time Recurrent Learning (RTRL) (Williams & Zipser, 1989). Both algorithms are based on the computation of the complete gradient information. Thereby, the error signals are carried backwards in time and tend to blow up or vanish. If they blow up the network weights oscillate, and if they vanish, learning of long-term dependencies takes a long time or simply does not take place. Obviously, both cases are undesirable. Bengio et al. (1994) and Hochreiter (1991) analytically proved this limitation of gradient based learning in recurrent networks. Their result holds regardless of the cost function, and regardless of the algorithm that is used to compute the error gradients. Bengio et al. (1994) generalized the problem to dynamical systems, which includes recurrent networks. Theorem 4 in (Bengio et al., 1994) shows that the condition leading to gradient decay is also a necessary condition for the system to robustly store discrete information for longer periods of time. In other words, if the network configuration allows the storage of information in its hidden units, the problem of vanishing gradients appears (Hochreiter et al., 2001).

Chen & Chaudhari (2009) discussed the problem particularly for SRNs. It occurs in the hidden layer, where from Equations 3.14 and 3.15 the output of the hidden layer can be written as

$$\mathbf{y}(t) = f_{\text{net}}\big(\mathbf{W}^{yx}\mathbf{x}(t) + \mathbf{W}^{yy}\mathbf{y}(t-1)\big), \tag{5.1}$$

with $\mathbf{W}$ denoting the weight matrices and $\mathbf{x}(t)$, $\mathbf{y}(t)$ the output vectors of the input and hidden layer (cf. Figure 3.8). Further, the activation function f_{net} is applied element wise.

[1] Parts of text in the Sections 5.4.1 and 5.5 are taken verbatim from this article.

If we evaluate the error function E at position T of an input sequence, the derivatives of the error function with respect to the hidden layer weights are given by

$$\frac{\partial E(T)}{\partial \mathbf{W}} = \sum_{\tau \leq T} \frac{\partial E(T)}{\partial \mathbf{y}(\tau)} \frac{\partial \mathbf{y}(\tau)}{\partial \mathbf{W}} = \sum_{\tau \leq T} \frac{\partial E(T)}{\partial \mathbf{y}(T)} \frac{\partial \mathbf{y}(T)}{\partial \mathbf{y}(\tau)} \frac{\partial \mathbf{y}(\tau)}{\partial \mathbf{W}}, \tag{5.2}$$

where $\mathbf{W}$ represents $\mathbf{W}^{yy}$ and $\mathbf{W}^{yx}$ respectively. The position in the input sequence is indexed by τ, with $\tau \leq T$. The partial derivative of the hidden layer output at the end of the sequence $\partial \mathbf{y}(T)$ with respect to some previous output $\partial \mathbf{y}(\tau)$ is the product of the derivatives between τ and T

$$\frac{\partial \mathbf{y}(T)}{\partial \mathbf{y}(\tau)} = \frac{\partial \mathbf{y}(T)}{\partial \mathbf{y}(T-1)} \frac{\partial \mathbf{y}(T-1)}{\partial \mathbf{y}(T-2)} \cdots \frac{\partial \mathbf{y}(\tau+1)}{\partial \mathbf{y}(\tau)} = \prod_{t=\tau+1}^{T} \frac{\partial \mathbf{y}(t)}{\partial \mathbf{y}(t-1)}. \tag{5.3}$$

Bengio et al. (1994) showed that the norm of each factor in Equation 5.3 must be less than 1

$$\left| \frac{\partial \mathbf{y}(t)}{\partial \mathbf{y}(t-1)} \right| < 1, \tag{5.4}$$

if the network should be able to store information over longer periods of time. Therefore, the partial derivation of the hidden layer output at the end of the sequence with respect to previous outputs converges exponentially fast to zero, as the distance between T and τ increases

$$\left| \frac{\partial \mathbf{y}(T)}{\partial \mathbf{y}(\tau)} \right| \to 0 \quad \text{where} \quad \tau \ll T \tag{5.5}$$

and thus

$$\left| \frac{\partial E(T)}{\partial \mathbf{y}(\tau)} \frac{\partial \mathbf{y}(\tau)}{\partial \mathbf{W}} \right| \to 0 \quad \text{where} \quad \tau \ll T. \tag{5.6}$$

From Equations 5.2 and 5.6 one can see that the contribution to the error gradient becomes very small for terms where τ is distal to T. In consequence, small changes in $\mathbf{W}$ effect almost exclusively the network states of the near past, where τ is close to T. So, the short-term information dominates the long-term information which makes learning of such distant dependencies difficult (Chen & Chaudhari, 2009).

5.2 Segmented-Memory Recurrent Neural Network

The problem of vanishing error gradients is the basic limitation of gradient descent learning for the weight optimisation in recurrent networks. This led to the development of alternative network architectures. One particular approach is the Segmented-Memory Recurrent Neural Network (SMRNN) proposed by Chen & Chaudhari (2004). From a cognitive science perspective, their idea has the pleasant property that it is inspired

by the memorisation process of long sequences, as it is observed in humans. Usually people fractionate sequences into segments to ease memorisation. Afterwards, the single segments are put together to form the final sequence. For instance, telephone numbers are broken into segments of two or three digits such that 7214789 becomes 72 - 14 - 789. This behaviour is not just plausible from everyday life, but evident in studies in the field of experimental psychology (Severin & Rigby, 1963; Wickelgren, 1967; Ryan, 1969; Frick, 1989; Hitch et al., 1996).

5.2.1 Forward Processing in the Segmented-Memory Recurrent Neural Network

The SMRNN architecture consists of two SRNs arranged in a hierarchical fashion as illustrated in Figure 5.1. A sequence of inputs is presented to the network symbol by symbol, that is, input vector by input vector. Separate internal states store the symbol level context (short-term information), as well as segment level context (long-term information). The symbol level context is updated for each symbol presented as input while the segment level context is updated only after each segment.

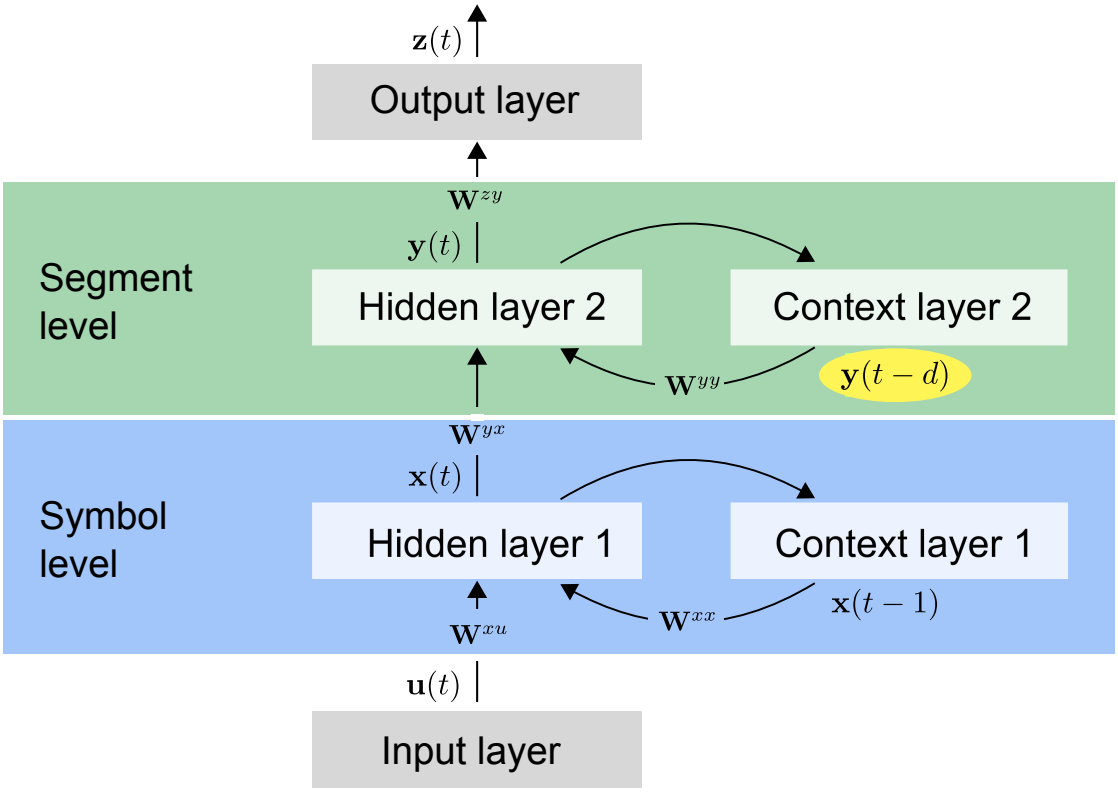


Figure 5.1: SMRNN topology – two SRNs are arranged hierarchically. The parameter d on segment level makes the difference between a cascade of SRNs and an SMRNN. Only after a segment of length d the segment level state is updated.

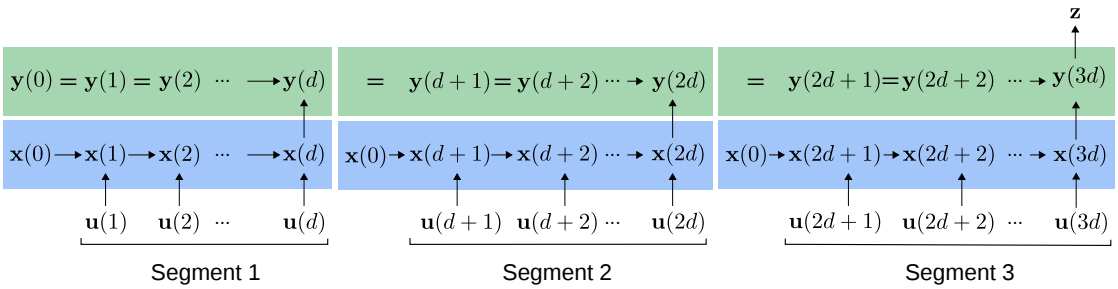


Figure 5.2: SMRNN dynamics for a sequence of three segments with fixed interval d.

In the following the receiver-sender-notation is used to describe the processing in the network. The upper indices of the weight matrices refer to the corresponding layer and the lower indices to the single units. For example, W_{ki}^{xu} denotes the connection between the k^{th} unit in hidden layer 1 (x) and the i^{th} unit in the input layer (u) (cf. Figure 5.1). Moreover, f_{net} is the transfer function of the network and n_u, n_x, n_y, n_z are the number of units in the input, hidden 1, hidden 2, and output layer.

The introduction of the parameter d on segment level distinguishes a cascade of SRNs from an SMRNN. It denotes the length of a segment which can be fixed or variable. The processing of an input sequence starts with the initial symbol level state $\mathbf{x}(0)$ and segment level state $\mathbf{y}(0)$. At the beginning of a segment (segment head SH) $\mathbf{x}(t)$ is updated with $\mathbf{x}(0)$ and input $\mathbf{u}(t)$. On other positions $\mathbf{x}(t)$ is obtained from its previous state $\mathbf{x}(t-1)$ and input $\mathbf{u}(t)$. It is calculated by

$$x_k(t) = \begin{cases} f_{\text{net}}\left(\sum_j^{n_x} W_{kj}^{xx} x_j(0) + \sum_i^{n_u} W_{ki}^{xu} u_i(t)\right) & \text{if } t = \text{SH}, \\ f_{\text{net}}\left(\sum_j^{n_x} W_{kj}^{xx} x_j(t-1) + \sum_i^{n_u} W_{ki}^{xu} u_i(t)\right) & \text{otherwise}, \end{cases} \tag{5.7}$$

where $k = 1, \ldots, n_x$. The segment level state $\mathbf{y}(0)$ is updated at the end of each segment (segment tail ST) as

$$y_k(t) = \begin{cases} f_{\text{net}}\left(\sum_j^{n_y} W_{kj}^{yy} y_j(t-1) + \sum_i^{n_x} W_{ki}^{yx} x_i(t)\right) & \text{if } t = \text{ST}, \\ y_k(t-1) & \text{otherwise}, \end{cases} \tag{5.8}$$

where $k = 1, \ldots, n_y$. The network output results from forwarding the segment level state

$$z_k(t) = f_{\text{net}}\left(\sum_j^{n_y} W_{kj}^{zy} y_j(t)\right) \quad \text{with} \quad k = 1, \ldots, n_z \ . \tag{5.9}$$

While the symbol level is updated on a symbol by symbol basis, the segment level changes only after d symbols. At the end of the input sequence the segment level state is forwarded to the output layer to generate the final output. The dynamics of an SMRNN processing a sequence is shown in Figure 5.2.

5.2.2 Effect of the Segmented Memory

Chen & Chaudhari (2009) analytically deduced the effect of a segmented memory on the vanishing gradient problem. Here, the crucial points are repeated to ease the understanding of the impact of the segment length d.

The derivatives of the error function with respect to the segment level weights $\mathbf{W}^{yx}$ and $\mathbf{W}^{xx}$ are computed similar to an SRN by Equation 5.2. Further, the partial derivative $\partial\mathbf{y}(T)/\partial\mathbf{y}(\tau)$ is given by Equation 5.3. As the segment level is updated according to Equation 5.8, the partial derivative of the segment level output does not change during symbol processing

$$\frac{\partial\mathbf{y}(t)}{\partial\mathbf{y}(t-1)} = 1 \quad \text{if } t \neq nd \text{ or } t \neq T. \tag{5.10}$$

Thus, for a sequence of length $T = Nd+m_1$ and $T > \tau = n_1d+m_2$ with $1 \leq m_1, m_2 \leq d$ it takes the special form

$$\frac{\partial\mathbf{y}(T)}{\partial\mathbf{y}(\tau)} = \frac{\partial\mathbf{y}(T)}{\partial\mathbf{y}(T-1)}\frac{\partial\mathbf{y}(Nd)}{\partial\mathbf{y}(Nd-1)}\cdots\frac{\partial\mathbf{y}((n_1+1)d)}{\partial\mathbf{y}((n_1+1)d-1)} \tag{5.11}$$

$$= \frac{\partial\mathbf{y}(T)}{\partial\mathbf{y}(T-1)}\prod_{n=n_1+1}^{N}\frac{\partial\mathbf{y}(nd)}{\partial\mathbf{y}(nd-1)}. \tag{5.12}$$

It is necessary that the norm of each factor in Equation 5.12 must be less than 1 if the network should be able to store information over longer periods of time. So, for $\tau \ll T$ the norm $|\partial\mathbf{y}(T)/\partial\mathbf{y}(\tau)|$ still converges to zero. If we compare Equations 5.3 and 5.12 on a similar sequence of length T at the same point τ, we find that the product in Equation 5.12 consists of less terms than the product in Equation 5.3. Therefore, the partial derivative in the SMRNN converges less fast to zero, that is, it vanishes slower than in an SRN.

Concerning the segment length d, the SMRNN turns into a recurrent network with multiple hidden layers for $d > T$. For $d = 1$ one gets a recurrent network with multiple hidden layers and multiple feedback connections. The advantage of a segmented memory and the slower vanishing gradient occurs only if $1 < d < T$. In other words, the length of the interval d affects the performance of an SMRNN. Obviously, the optimal value for d is task-dependent, and if it is chosen too small or too large the SMRNN fails to bridge long time lags (Chen & Chaudhari, 2009).

5.3 Extension of Real-Time Recurrent Learning for Segmented-Memory Recurrent Neural Networks

Chen & Chaudhari (2009) proved the ability of SMRNNs to learn long-term dependencies and applied it to the problem of protein secondary structure prediction. The networks were trained by the eRTRL algorithm. In the following eRTRL is introduced in order to derive the computational complexity of the algorithm.

Williams & Zipser (1995) showed that the original RTRL algorithm (Williams & Zipser, 1989) has an average time complexity in order of magnitude $\mathcal{O}(n^4)$, with n denoting the number of network units in a fully connected network. Therefore, most likely eRTRL has such a huge complexity too.

5.3.1 Extension of Real-Time Recurrent Learning

In sequence processing, an error signal occurs only at the end of a sequence. Therefore, learning is based on minimizing the sum of squared errors at the end of a sequence of N segments,

$$E(t) = \begin{cases} \sum_{k=1}^{n_z} \frac{1}{2} \left(z_k\left(t\right) - v_k\left(t\right)\right)^2 & \text{if } t = Nd, \\ 0 & \text{otherwise,} \end{cases} \tag{5.13}$$

where $v_k(t)$ is the target value and $z_k(t)$ is the actual output of the k^{th} unit in the output layer. Every network parameter $P \in \{W_{ki}^{xu}, W_{kj}^{xx}, W_{ki}^{yx}, W_{kj}^{yy}, W_{kj}^{zy}, x_k(0), y_k(0)\}$ is initialised with small random values and then updated according to the gradient information

$$\Delta P = -\alpha \frac{\partial E(t)}{\partial P} + \eta \Delta' P \tag{5.14}$$

with learning rate α and the momentum term η. The value $\Delta' P$ is the variation of P in the previous iteration. Note that the initial states of the context layer on segment and symbol level $\mathbf{y}(0)$ and $\mathbf{x}(0)$ are also adapted during learning.

Learning Output Layer Weights with Backpropagation of Error

The change in the output layer weights W_{kj}^{zy} can be calculated in one single step following the standard backpropagation algorithm

$$\Delta W_{kj}^{zy} = \frac{\partial E(t)}{\partial W_{kj}^{zy}} = \left(z_k(t) - v_k(t)\right) f'_{\text{net}}\left(a_k^z(t)\right) y_j(t), \tag{5.15}$$

$$= e_k(t)\ f'_{\text{net}}\left(a_k^z\left(t\right)\right) y_j(t), \tag{5.16}$$

$$= \delta_k^z(t)\ y_j(t). \tag{5.17}$$

The local error at output unit k is calculated as difference between the unit's output and target value

$$e_k(t) = z_k(t) - v_k(t). \tag{5.18}$$

Weighted with the derivative of the activation function at the synaptic input one gets the error signal for the connections from the second hidden layer towards the corresponding unit in the output layer

$$\delta_k^z(t) = e_k(t)\ f'_{\text{net}}\left(a_k^z(t)\right). \tag{5.19}$$

Here, $a_k^z(t)$ denotes the synaptic input of output unit k and the $\delta_k(t)$ is a short hand for $\partial E(t)/\partial a_k(t)$, representing the sensitivity of $E(t)$ to small changes of the k^{th} unit activation.

Learning Weights in the Segment Level SRN

The derivatives of $E(t)$ with respect to other parameters need much more computation. The segment level SRN and the symbol level SRN are treated as subnetworks that are trained according to the RTRL algorithm (Williams & Zipser, 1989). The error signal for the segment level subnetwork is the backpropagated error from the output layer. Hence, the derivatives of $E(t)$ with respect to the weights and the initial state of the layers is given by

$$\frac{\partial E(t)}{\partial P} = \sum_{c=1}^{n_z} \delta_c^z \sum_{b=1}^{n_y} W_{cb}^{zy} \frac{\partial y_b(t)}{\partial P}\ . \tag{5.20}$$

The term δ_c^z weighted with W_{cb}^{zy} is the error signal for the segment level subnetwork which is used to calculate the weight changes. The derivative of the hidden layer outputs with respect to parameter P is computed in a recursive way.

At time $t > 0$, the derivatives of $y_b(t)$ with respect to W_{kj}^{yy}, W_{ki}^{yx}, and $y_k(0)$ are calculated using the following equations:

$$\frac{\partial y_b(t)}{\partial W_{kj}^{yy}} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy} \frac{\partial y_a(t-1)}{\partial W_{kj}^{yy}} + \delta(b,k) y_j(t-1)\right], \tag{5.21}$$

$$\frac{\partial y_b(t)}{\partial W_{ki}^{yx}} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy} \frac{\partial y_a(t-1)}{\partial W_{ki}^{yx}} + \delta(b,k) x_i(t)\right], \tag{5.22}$$

$$\frac{\partial y_b(t)}{\partial y_k(0)} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy} \frac{\partial y_a(t-1)}{\partial y_k(0)}\right]. \tag{5.23}$$

Here, $\delta(b,k)$ denotes the Kronecker delta with $\delta(b,k) = 1$ if $b = k$ and 0 otherwise.

Learning Weights in the Symbol Level SRN

For the training of the symbol level SRN, the derivatives of $y_b(t)$ with respect to W_{kj}^{xx}, W_{ki}^{xu}, and $x_k(0)$ are calculated. This is necessary for the gradient computation in Equation 5.20. They are given as follows:

$$\frac{\partial y_b(t)}{\partial P} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy}\frac{\partial y_a(t-1)}{\partial P} + \sum_{a=1}^{n_x} W_{ba}^{yx}\frac{\partial x_a(t)}{\partial P}\right] \tag{5.24}$$

with $P \in \{W_{kj}^{xx}, W_{ki}^{xu}, x_k(0)\}$.

The derivatives of $x_a(t)$ with respect to W_{kj}^{xx}, W_{ki}^{xu}, and $x_a(0)$ are also calculated in a recursive way:

$$\frac{\partial x_a(t)}{\partial W_{kj}^{xx}} = f'_{\text{net}}\left(a_a^x(t)\right)\left[\sum_{b=1}^{n_x} W_{ab}^{xx}\frac{\partial x_b(t-1)}{\partial W_{kj}^{xx}} + \delta(a,k)x_j(t-1)\right], \tag{5.25}$$

$$\frac{\partial x_a(t)}{\partial W_{ki}^{xu}} = f'_{\text{net}}\left(a_a^x(t)\right)\left[\sum_{b=1}^{n_x} W_{ab}^{xx}\frac{\partial x_b(t-1)}{\partial W_{ki}^{xu}} + \delta(a,k)u_i(t)\right], \tag{5.26}$$

$$\frac{\partial x_a(t)}{\partial x_k(0)} = f'_{\text{net}}\left(a_a^x(t)\right)\left[\sum_{b=1}^{n_x} W_{ab}^{xx}\frac{\partial x_b(t-1)}{\partial x_k(0)}\right]. \tag{5.27}$$

They are applied to Equation 5.24 and yield:

$$\frac{\partial y_b(t)}{\partial W_{kj}^{xx}} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy}\frac{\partial y_a(t-1)}{\partial W_{kj}^{xx}} + \sum_{a=1}^{n_x} W_{ba}^{yx}\frac{\partial x_a(t)}{\partial W_{kj}^{xx}}\right], \tag{5.28}$$

$$\frac{\partial y_b(t)}{\partial W_{ki}^{xu}} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy}\frac{\partial y_a(t-1)}{\partial W_{ki}^{xu}} + \sum_{a=1}^{n_x} W_{ba}^{yx}\frac{\partial x_a(t)}{\partial W_{ki}^{xu}}\right], \tag{5.29}$$

$$\frac{\partial y_b(t)}{\partial x_k(0)} = f'_{\text{net}}\left(a_b^y(t)\right)\left[\sum_{a=1}^{n_y} W_{ba}^{yy}\frac{\partial y_a(t-1)}{\partial x_k(0)} + \sum_{a=1}^{n_x} W_{ba}^{yx}\frac{\partial x_a(t)}{\partial x_k(0)}\right]. \tag{5.30}$$

Initial derivatives $\partial y_b(t)/\partial P$ and $\partial x_a(t)/\partial P$

As the derivatives of the hidden layer outputs $\mathbf{x}(t)$ and $\mathbf{y}(t)$ with respect to the parameters P are calculated in a recursive way, initial values have to be defined. At the very beginning $t = 0$ the initial derivatives with respect to the initial states in Equations 5.27 and 5.23 are

$$\frac{\partial x_a(t)}{\partial x_k(0)} = \delta(a,k), \tag{5.31}$$

$$\frac{\partial y_b(t)}{\partial y_k(0)} = \delta(b,k), \tag{5.32}$$

where $\delta(a,k)$ denotes the Kronecker delta with $\delta(a,k) = 1$ if $a = k$, and 0 otherwise. That is, the output of a unit at $t = 0$ is only sensitive to its own initial state at this point in time. The initial derivatives with respect to the weights are set to zero

$$\frac{\partial x_a(t)}{\partial P} = 0 \qquad \text{with} \qquad P \in \{W_{ki}^{xu}, W_{kj}^{xx}\}, \tag{5.33}$$

$$\frac{\partial y_b(t)}{\partial P} = 0 \qquad \text{with} \qquad P \in \{W_{ki}^{xu}, W_{kj}^{xx}, W_{ki}^{yx}, W_{kj}^{yy}, x_k(0)\}. \tag{5.34}$$

This implies that the first output of the hidden layers is independent of the initial weights.

5.3.2 Computational Complexity of Extended Real-Time Recurrent Learning

After the description of eRTRL, as it was proposed by Chen & Chaudhari (2004), the computational complexity of the algorithm shall be derived. For this purpose, the exact computational complexity $\Theta(f(n))$, that is, the number of operations needed, and the class of complexity $\mathcal{O}(f(n))$, are distinguished. The class of complexity is the order of magnitude of the function of interest $f(n)$.

In the learning algorithm, the most prominent computation is a form of inner product where additions and multiplications occur equally often. These operations will be counted in the following. The computational cost for the forward processing of the network is ignored, as it is independent of the learning algorithm and much smaller than the cost for the gradient computation. For the same reason, the amount of computations needed to actually update the weights is ignored. The computational complexity of the learning algorithm is solely the computational requirement to calculate the gradient of the error with respect to the network parameter $\partial E/\partial P$. As an error signal is only available at the end of a sequence, the number of operations needed to compute the error gradients for a single sequence is determined.

The number of operations needed to compute the change in the output layer weights $\mathbf{W}^{zy}$ can be read from Equation 5.16. One multiplication is needed to compute $e_k(t)\ f'_{\text{net}}(a_k^z(t))$ and one for the result to be multiplied with $y_j(t)$. Further this has to be done for all $n_z \cdot n_y$ connections, which leaves us with

$$\Theta\left(\frac{\partial E(t)}{\mathbf{W}^{zy}}\right) = 2n_y n_z. \tag{5.35}$$

For all other parameters of the network, we have to perform the computation according to Equation 5.20, which costs

$$\Theta\left(\frac{\partial E(t)}{\partial P}\right) = 2n_z + n_z\left[2n_y + n_y\Theta\left(\frac{\partial y_b(t)}{\partial P}\right)\right] \tag{5.36}$$

operations.

Segment Level Parameters The derivatives of $y_b(t)$ with respect to the segment level parameters W_{kj}^{yy}, W_{ki}^{yx}, and $y_k(0)$ are determined by Equation 5.21, 5.22, and 5.23. This equations are defined recursively, such that the length of the sequence influences the amount of computational steps, which is

$$\Theta\left(\frac{\partial y_b(t)}{\partial W_{kj}^{yy}}\right) = (2n_y + 3)(t-1), \tag{5.37}$$

$$\Theta\left(\frac{\partial y_b(t)}{\partial W_{ki}^{yx}}\right) = (2n_y + 3)(t-1), \tag{5.38}$$

$$\Theta\left(\frac{\partial y_b(t)}{\partial y_k(0)}\right) = (2n_y + 1)(t-1). \tag{5.39}$$

Symbol Level Parameters The symbol level derivatives of $y_b(t)$ with respect to the parameters W_{kj}^{xx}, W_{ki}^{xu}, and $x_k(0)$ are determined by Equations 5.28, 5.29 and 5.30. Again, they are defined recursively and their computational complexity depends on the number of time steps. We end up with

$$\Theta\left(\frac{\partial y_b(t)}{\partial W_{kj}^{xx}}\right) = \left[2 + 2n_y + 2n_x + n_x\Theta\left(\frac{\partial x_a(t)}{\partial W_{kj}^{xx}}\right)\right](t-1), \tag{5.40}$$

$$\Theta\left(\frac{\partial y_b(t)}{\partial W_{ki}^{xu}}\right) = \left[2 + 2n_y + 2n_x + n_x\Theta\left(\frac{\partial x_a(t)}{\partial W_{ki}^{xu}}\right)\right](t-1), \tag{5.41}$$

$$\Theta\left(\frac{\partial y_b(t)}{\partial x_k(0)}\right) = \left[2 + 2n_y + 2n_x + n_x\Theta\left(\frac{\partial x_a(t)}{\partial x_k(0)}\right)\right](t-1) \tag{5.42}$$

operations, where

$$\Theta\left(\frac{\partial x_a(t)}{\partial W_{kj}^{xx}}\right) = 2n_x + 3, \tag{5.43}$$

$$\Theta\left(\frac{\partial x_a(t)}{\partial W_{ki}^{xu}}\right) = 2n_x + 3, \tag{5.44}$$

$$\Theta\left(\frac{\partial x_a(t)}{\partial x_k(0)}\right) = 2n_x + 1. \tag{5.45}$$

Replacement of $\Theta(\partial y_b(t)/\partial P)$ in Equation 5.36 with the values of Equations 5.37-5.42 gives the number of operations needed for a *single* element in P. To get the total number

of operations we have to multiply the result with the number of elements of P which is

$$\Theta\left(\frac{\partial E(t)}{\partial \mathbf{W}^{yy}}\right) = \Theta\left(\frac{\partial E(t)}{\partial W_{kj}^{yy}}\right) \cdot n_y^2, \tag{5.46}$$

$$\Theta\left(\frac{\partial E(t)}{\partial \mathbf{W}^{yx}}\right) = \Theta\left(\frac{\partial E(t)}{\partial W_{ki}^{yx}}\right) \cdot n_y n_x, \tag{5.47}$$

$$\Theta\left(\frac{\partial E(t)}{\partial \mathbf{y}(0)}\right) = \Theta\left(\frac{\partial E(t)}{\partial y_k(0)}\right) \cdot n_y, \tag{5.48}$$

$$\Theta\left(\frac{\partial E(t)}{\partial \mathbf{W}^{xx}}\right) = \Theta\left(\frac{\partial E(t)}{\partial W_{kj}^{xx}}\right) \cdot n_x^2, \tag{5.49}$$

$$\Theta\left(\frac{\partial E(t)}{\partial \mathbf{W}^{xu}}\right) = \Theta\left(\frac{\partial E(t)}{\partial W_{ki}^{xu}}\right) \cdot n_x n_u, \tag{5.50}$$

$$\Theta\left(\frac{\partial E(t)}{\partial \mathbf{x}(0)}\right) = \Theta\left(\frac{\partial E(t)}{\partial x_k(0)}\right) \cdot n_x. \tag{5.51}$$

As a last step we sum up the amount of operations needed for every parameter in the network

$$\Theta\left(\frac{\partial E(t)}{\partial P}\right) = \sum_{P_i} \Theta\left(\frac{\partial E(t)}{\partial P_i}\right) \tag{5.52}$$

with P denoting the set of network parameters. The resulting polynomial expression is somewhat complex

$$\begin{aligned}
\Theta\left(\frac{\partial E(t)}{\partial P}\right) = {} & 6\, n_x n_y n_z + 2\, {n_y}^4 n_z t - 2\, n_x {n_y}^3 n_z - n_x {n_y}^2 n_z + {n_y}^2 n_z t \\
& + 5\, {n_y}^3 n_z t + 4\, {n_x}^2 n_y n_z + 2\, n_x n_u n_z + 2\, n_x {n_y}^3 n_z t \\
& + 4\, n_x n_u n_y n_z + 3\, n_x {n_y}^2 n_z t + 2\, n_x n_u n_z n_y n_x\,(t-1)\, n_y\,(t-1) \\
& + 5\, {n_x}^2 n_z n_y n_x\,(t-1) + 2\, n_x n_z n_y n_y\,(t-1) \\
& + 3\, n_x n_z n_y n_x\,(t-1) + 2\, {n_x}^2 n_z n_y n_y\,(t-1) \\
& + 2\, n_x n_z n_y n_x\,(t-1)\, n_y\,(t-1) + 4\, n_y n_z + 2\, n_x n_z \\
& + 2\, {n_x}^2 n_z n_y n_x\,(t-1)\, n_y\,(t-1) + 2\, n_x n_u n_z n_y n_y\,(t-1) \\
& + 5\, n_x n_u n_z n_y n_x\,(t-1) + 3\, {n_y}^2 n_z - 3\, {n_y}^3 n_z - 2\, {n_y}^4 n_z \\
& + 2\, {n_x}^2 n_z
\end{aligned} \tag{5.53}$$

and for the sake of clarity we may assume $n = n_u = n_x = n_y = n_z$

$$\begin{aligned}
\Theta\left(\frac{\partial E(t)}{\partial P}\right) = {} & -4\, n^5 + 4\, n^5 t + 8\, n^4 t + 4\, n^4\,(n\,(t-1))^2 \\
& + 14\, n^4 n\,(t-1) + 4\, n^4 + 2\, n^3\,(n\,(t-1))^2 \\
& + 5\, n^3 n\,(t-1) + n^3 t + 13\, n^3 + 6\, n^2
\end{aligned} \tag{5.54}$$

The dominating term in this sum is the polynomial $4n^4 \left(n \left(t-1\right)\right)^2$. It depends on t, which is the length of the sequence that is processed. Therefore, it is reasonable to assume $t \gg 1$ such that $4n^4 \left(n \left(t-1\right)\right)^2 \approx 4n^6 t^2$. This means, for a sequence of N segments of length d, the computational cost for the weight update according to eRTRL can be estimated with

$$\mathcal{O}\left(\frac{\partial E(t)}{\partial P}\right) = 4n^6 (Nd)^2. \tag{5.55}$$

As one can see, the amount of operations required to train an SMRNN with eRTRL rises unreasonably with the number of units in the network layers n.

5.4 Extension of Backpropagation Through Time for Segmented-Memory Recurrent Neural Networks

The high computational complexity of eRTRL makes it impractical for applications where large networks are used (cf. Equation 5.55). In the following, an extension for the BPTT algorithm (Werbos, 1990) is introduced, which has a much smaller time complexity. Real-Time BPTT is adapted to SMRNNs, that is, the error at the output at the end of a sequence is used instantaneously for weight adaptation of the network.

5.4.1 Extension of Backpropagation Through Time

Learning is based on minimizing the sum of squared errors at the end of a sequence of N segments (cf. Equation 5.13),

$$E(t) = \begin{cases} \sum_{k=1}^{n_z} \frac{1}{2} \left(z_k(t) - v_k(t)\right)^2 & \text{if } t = Nd, \\ 0 & \text{otherwise,} \end{cases} \tag{5.56}$$

where $v_k(t)$ is the target value and $z_k(t)$ is the actual output of the k^{th} unit in the output layer. The error is propagated back through the network and also back through time to adapt the weights. Further, it is not reasonable to keep the initial states $\mathbf{y}(0) = f_{\text{net}}(\mathbf{a}^{yy}(0))$ and $\mathbf{x}(0) = f_{\text{net}}\left(\mathbf{a}^{xx}(0)\right)$ fixed. Thus, the initial activations $\mathbf{a}^{yy}(0)$ and $\mathbf{a}^{xx}(0)$ are also learnt. Here, the upper indices of the activations refer to the corresponding layer and a lower index to the single units. For example, a_k^{yx} is the activation at the k^{th} unit in the second hidden layer that results from connections from the first hidden layer, which is simply $a_k^{yx}(t) = \sum_i^{n_x} W_{ki}^{yx} x_i(t)$. The gradient of $E(t)$ can be computed using the injecting error

$$e_k(t) = z_k(t) - v_k(t). \tag{5.57}$$

Applying backpropagation, we compute the delta error. Here $\delta_k(t)$ is a short hand for $\partial E(t)/\partial a_k$ representing the sensitivity of $E(t)$ to small changes of the k^{th} unit activation. The deltas for the output units δ^{zy}, second hidden layer units δ^{yy}, and first hidden layer units δ^{yx} at the end of a sequence ($t = Nd$) are

$$\delta_k^{zy}(t) = f'_{\text{net}}\left(a_k^{zy}(t)\right) e_k(t), \tag{5.58}$$

$$\delta_k^{yy}(t) = f'_{\text{net}}\left(a_k^{yy}(t)\right) \sum_{i=1}^{n_z} W_{ik}^{zy} \delta_i^{zy}(t), \tag{5.59}$$

$$\delta_k^{yx}(t) = f'_{\text{net}}\left(a_k^{yx}(t)\right) \sum_{i=1}^{n_z} W_{ik}^{zy} \delta_i^{zy}(t). \tag{5.60}$$

At that point we enroll the SMRNN on segment level to propagate the error back in time. The state of the second hidden layer changes only at the end of a segment $t = nd$ and $n = 0, \ldots, N-1$. Therefore, the delta error for the second hidden layer, and first

hidden layer units results in

$$\delta_k^{yy}(nd) = f'_{\text{net}}\left(a_k^{yy}(nd)\right) \sum_{i=1}^{n_y} W_{ik}^{yy} \delta_i^{yy}\left((n+1)d\right), \tag{5.61}$$

$$\delta_k^{yx}(nd) = f'_{\text{net}}\left(a_k^{yx}(nd)\right) \sum_{i=1}^{n_y} W_{ik}^{yy} \delta_i^{yy}\left((n+1)d\right). \tag{5.62}$$

Once the computation was performed down to the beginning of the sequence ($t = 0$), the gradient of the weights and initial activation on segment level is computed by

$$\Delta W_{ij}^{zy} = \delta_i^{zy}(Nd) y_j(Nd), \tag{5.63}$$

$$\Delta W_{ij}^{yy} = \sum_{n=1}^{N} \delta_i^{yy}(nd) y_j\left((n-1)d\right), \tag{5.64}$$

$$\Delta W_{ij}^{yx} = \sum_{n=2}^{N} \delta_i^{yx}(nd) x_j\left((n-1)d\right), \tag{5.65}$$

$$\Delta a_i^{yy} = \delta_i^{yy}(0). \tag{5.66}$$

For the adaptation of the weights on symbol level we apply the BPTT procedure repetitively for every time step $\tau = 0, \ldots, d$ for every segment of the sequence. That is, for the end of a segment ($\tau = d$)

$$\delta_k^{xx}(d) = f'_{\text{net}}\left(a_k^{xx}(d)\right) \sum_{i=1}^{n_y} W_{ik}^{yx} \delta_i^{yx}(d), \tag{5.67}$$

$$\delta_k^{xu}(d) = f'_{\text{net}}\left(a_k^{xu}(d)\right) \sum_{i=1}^{n_y} W_{ik}^{yx} \delta_i^{yx}(d). \tag{5.68}$$

Further, for $\tau < d$ we get

$$\delta_k^{xx}(\tau) = f'_{\text{net}}\left(a_k^{xx}(\tau)\right) \sum_{i=1}^{n_x} W_{ik}^{xx} \delta_i^{xx}(\tau+1), \tag{5.69}$$

$$\delta_k^{xu}(\tau) = f'_{\text{net}}\left(a_k^{xu}(\tau)\right) \sum_{i=1}^{n_x} W_{ik}^{xx} \delta_i^{xx}(\tau+1). \tag{5.70}$$

When the computation was performed to the beginning of a segment ($\tau = 0$), the gradient of the weights and initial activation on symbol level are computed by

$$\Delta W_{ij}^{xx} = \sum_{\tau=1}^{d} \delta_i^{xx}(\tau) x_j(\tau-1), \tag{5.71}$$

$$\Delta W_{ij}^{xu} = \sum_{\tau=2}^{d} \delta_i^{xu}(\tau) u_j(\tau-1), \tag{5.72}$$

$$\Delta a_i^{xx} = \delta_i^{xx}(0). \tag{5.73}$$

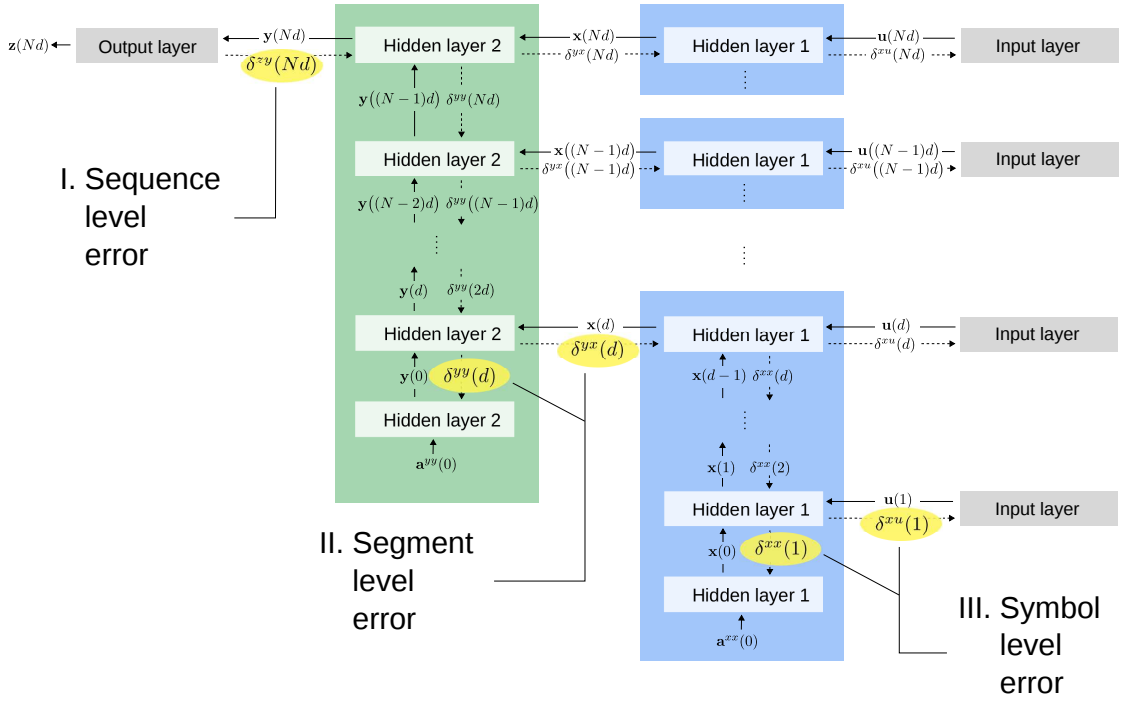


Figure 5.3: Errorflow of the eBPTT algorithm in an SMRNN for a sequence of length Nd. The solid arrows indicate the development of the states of the layers in the network. The dashed arrows show the propagation of the error back through the network and back through time.

Note that the sums in Equations 5.65 and 5.72 start at $n = 2$ and $\tau = 2$ respectively. This is due to the fact that at the beginning $t = 0$ the hidden layer 2 has no input from hidden layer 1 and hidden layer 1 has no input from the input layer (cf. Figure 5.2). Figure 5.3 illustrates the error flow in the SMRNN for one sequence of length Nd.

The computed gradients can be used right away to change the networks weights and initial activations, respectively,

$$\tilde{W}_{ij} = W_{ij} - \alpha \Delta W_{ij} + \eta \Delta' W_{ij}, \tag{5.74}$$

where α denotes the learning rate and η the momentum term. The value $\Delta' W_{ij}$ represents the change of W_{ij} in the previous iteration. The gradients may also be applied epoch-wise. That is, for an epoch of $s = 1, \ldots, M$ sequences

$$\tilde{W}_{ij} = W_{ij} - \alpha \left(\sum_{s=1}^{M} \Delta W_{ij}(s) \right) + \eta \Delta' W_{ij}. \tag{5.75}$$

5.4.2 Computational Complexity of Extended Backpropagation Through Time

In order to compare eRTRL and eBPTT, the computational complexity is derived for eBPTT in the following. The same notation as in Section 5.3.2 is used. $\Theta(f(n))$ denotes the exact computational complexity of the algorithm while $\mathcal{O}(f(n))$ denotes the class of complexity.

Again, the summation and multiplication are the operations of interest. Further, the computational costs for the forward processing in the network and the actual weight updates are ignored. They are much smaller than the cost for the gradient computation and further, they are independent of the learning algorithm. Solely the computational requirement to calculate the gradient of the error with respect to the network parameter $\partial E/\partial P$ is taken into account. Hence, I determine the number of operations needed to compute the error gradients after one sequence of Nd time steps ($t = 1, \ldots, Nd$), with d denoting the length of a segment and N the number of segments of the sequence.

The number of operations needed to compute the change in the output layer weights $\mathbf{W}^{zy}$ can be read from Equations 5.58 and 5.63. One multiplication is needed to compute δ_k^{zy} and one for the result to be multiplied with y_j. Further, this has to be done for all $n_z \cdot n_y$ connections, which gives

$$\Theta\left(\Delta \mathbf{W}^{zy}\right) = 2n_y n_z. \tag{5.76}$$

At the end of a sequence, the deltas for the first and second hidden layer are calculated according to Equations 5.59 and 5.60, which costs

$$\Theta\left(\delta_k^{yy}(t = Nd)\right) = 1 + 2n_z, \tag{5.77}$$
$$\Theta\left(\delta_k^{yx}(t = Nd)\right) = 1 + 2n_z \tag{5.78}$$

operations.

Segment Level Parameters Enrolling the network on the segment level, one calculates the delta errors for the hidden layers at the end of each segment $t = nd$ and $n = 0, \ldots, N$. The number of computational steps needed for Equations 5.61 and 5.62 is

$$\Theta\left(\delta_k^{yy}(t = nd)\right) = 1 + 2n_y, \tag{5.79}$$
$$\Theta\left(\delta_k^{yx}(t = nd)\right) = 1 + 2n_y. \tag{5.80}$$

The deltas are applied to compute the gradients of the weights and initial activations on the segment level in Equations 5.64, 5.65, and 5.66. Here,

$$\Theta\left(\Delta W_{ij}^{yy}\right) = \Theta\left(\delta_k^{yy}(t = Nd)\right) + \Theta\left(\delta_k^{yy}(t = nd)\right) \cdot (N - 1) + 2N, \tag{5.81}$$
$$\Theta\left(\Delta W_{ij}^{yx}\right) = \Theta\left(\delta_k^{yx}(t = Nd)\right) + \Theta\left(\delta_k^{yx}(t = nd)\right) \cdot (N - 2) + 2(N - 1), \tag{5.82}$$
$$\Theta\left(\Delta a_i^{yy}\right) = 1 + 2n_y \tag{5.83}$$

operations are needed.

Symbol Level Parameters Parameters on symbol level are computed for each segment. At the end of a segment $\tau = d$, Equations 5.67 and 5.68 are applied, while at times $\tau = 0 \dots d-1$, Equations 5.69 and 5.70 are used. The computational effort is

$$\Theta\left(\delta_k^{xx}(\tau = d)\right) = 1 + 2n_y, \tag{5.84}$$
$$\Theta\left(\delta_k^{xu}(\tau = d)\right) = 1 + 2n_y, \tag{5.85}$$
$$\Theta\left(\delta_k^{xx}(\tau < d)\right) = 1 + 2n_x, \tag{5.86}$$
$$\Theta\left(\delta_k^{xu}(\tau < d)\right) = 1 + 2n_x. \tag{5.87}$$

Using this deltas the changes of the parameters on the symbol level are computed in Equations 5.71, 5.72, and 5.73. It costs

$$\Theta\left(\Delta W_{ij}^{xx}\right) = \Theta\left(\delta_k^{xx}(\tau = d)\right) + \Theta\left(\delta_k^{xx}(\tau < d)\right) \cdot (d-1) + 2d, \tag{5.88}$$
$$\Theta\left(\Delta W_{ij}^{xu}\right) = \Theta\left(\delta_k^{xu}(\tau = d)\right) + \Theta\left(\delta_k^{xu}(\tau < d)\right) \cdot (d-2) + 2(d-1), \tag{5.89}$$
$$\Theta\left(\Delta a_i^{xx}\right) = 1 + 2n_x \tag{5.90}$$

computational steps.

To get the total amount of operations needed, one has to multiply the computational steps of a *single* parameter with the number of actual connections. Further, the operations on the symbol level (Equations 5.88 - 5.90) are done for each of the N segments of the sequence. Therefore, one ends up with

$$\Theta\left(\Delta \mathbf{W}^{yy}\right) = \Theta\left(\Delta W_{ij}^{yy}\right) \cdot n_y^2, \tag{5.91}$$
$$\Theta\left(\Delta \mathbf{W}^{yx}\right) = \Theta\left(\Delta W_{ij}^{yx}\right) \cdot n_y n_x, \tag{5.92}$$
$$\Theta\left(\Delta \mathbf{a}^{yy}\right) = \Theta\left(\Delta a_i^{yy}\right) \cdot n_y, \tag{5.93}$$
$$\Theta\left(\Delta \mathbf{W}^{xx}\right) = \Theta\left(\Delta W_{ij}^{xx}\right) \cdot N n_x^2, \tag{5.94}$$
$$\Theta\left(\Delta \mathbf{W}^{xu}\right) = \Theta\left(\Delta W_{ij}^{xu}\right) \cdot N n_x n_u, \tag{5.95}$$
$$\Theta\left(\Delta \mathbf{a}^{xx}\right) = \Theta\left(\Delta a_i^{xx}\right) \cdot N n_x \tag{5.96}$$

operations. The sum over the amount of operations needed for every parameter in the network,

$$\Theta\left(\frac{\partial E(t)}{\partial P}\right) = \sum_{P_i} \Theta\left(\frac{\partial E(t)}{\partial P_i}\right), \tag{5.97}$$

with P denoting the set of network parameters, gives the exact number of operations needed to compute the weight updates after one sequence according to the eBPTT algorithm

$$\begin{aligned}\Theta\left(\frac{\partial E(t)}{\partial P}\right) = {} & 2\,n_z n_y + 2\,{n_y}^2 n_z + 3\,{n_y}^2 N + 2\,{n_y}^3 N - 2\,{n_y}^3 - 3\,n_x n_y + 2\,n_x n_y n_z \\ & + 3\,n_x n_y N + 2\,n_x {n_y}^2 N - 4\,n_x {n_y}^2 + n_y + 2\,{n_y}^2 + 2\,N {n_x}^2 n_y \\ & + 3\,N {n_x}^2 d + 2\,N {n_x}^3 d - 2\,N {n_x}^3 - 3\,N n_x n_u + 2\,N n_x n_u n_y \\ & + 3\,N n_x n_u d + 2\,N {n_x}^2 n_u d - 4\,N {n_x}^2 n_u + N n_x + 2\,N {n_x}^2. \end{aligned} \tag{5.98}$$

As this expression is as complex as it is for eRTRL in Equation 5.53, we may assume $n = n_u = n_x = n_y = n_z$ and get

$$\Theta\left(\frac{\partial E(t)}{\partial P}\right) = 4\,n^3Nd + 2\,n^3N - 2\,n^3 + 6\,n^2Nd + 5\,n^2N + n^2 + Nn + n. \tag{5.99}$$

Here, the dominating term of the sum is $4n^3Nd$, such that

$$\mathcal{O}\left(\frac{\partial E(t)}{\partial P}\right) = 4n^3Nd \tag{5.100}$$

can be considered as an estimation of the computational cost for the weight update according to the eBPTT algorithm for a sequence of N segments of length d.

Comparing the computational costs of both learning algorithms, one can see that eBPTT (Equation 5.100) requires much less operations than eRTRL (Equation 5.55). The relation can be approximated with

$$\mathcal{O}\left(\frac{\partial E(t)}{\partial P}\right)_{\text{eRTRL}} \approx \left[\mathcal{O}\left(\frac{\partial E(t)}{\partial P}\right)_{\text{eBPTT}}\right]^2. \tag{5.101}$$

This fact is further illustrated in Figure 5.4, where the approximate number of operations is plotted against the number of units in the network's layers. Note that the y-axis is scaled logarithmically. Using 100 units in each layer of the network one needs approximately $4 \cdot 10^{16}$ operations with eRTRL versus $4 \cdot 10^{8}$ operations with eBPTT for the gradient computation after a sequence of length 100.

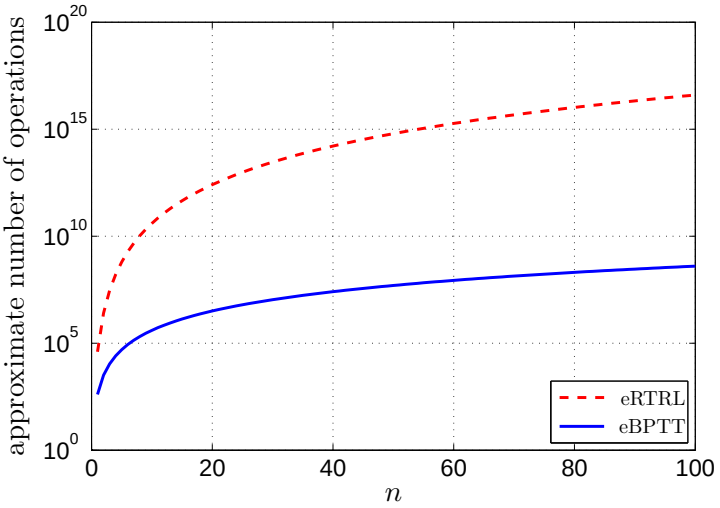


Figure 5.4: Approximate number of operations required for the gradient computation according to the eRTRL and eBPTT algorithm plotted over the number of units in each layer of the network n. The length of the sequence Nd was set to 100.

5.5 Evaluation on the Information Latching Problem

As shown in Section 5.4.2, eBPTT is highly advantageous compared to eRTRL concerning the computational costs for network training. Now, their ability to deal with the learning of long-term dependencies shall be investigated by a comparison on a benchmark problem. For this purpose, the information latching problem is used. It was designed by Bengio et al. (1994) to test a system's ability to model dependencies of the output on earlier inputs. In this context, "information latching" refers to the storage of information in the internal states of the system for some time.

Basically, it is a sequence classification problem. The idea is to distinguish two classes of sequences, where the class C of the sequence $i_1, i_2, \ldots, i_T$ depends on the first L items

$$C(i_1, i_2, \ldots, i_T) = C(i_1, i_2, \ldots, i_L) \in \{0, 1\} \text{ with } L < T. \tag{5.102}$$

For the comparison, the sequences are generated from an alphabet of 26 letters (a - z), such that the number of input neurons is 26 (1-of-N coding). A sequence is considered to be class $C = 1$ if the items $i_1, i_2, \ldots, i_L$ match a predefined string, otherwise it is class $C = 0$. All items i of a sequence that are not predefined are chosen randomly from the alphabet. Table 5.1 illustrates the problem for a sequence of length $T = 22$ with a class-defining string of length $L = 10$.

Table 5.1: Information Latching problem for a sequence of length $T = 22$ with a class-defining string of length $L = 10$

sequence	class C
p r e d e f i n e d r a n d o m s t r i n g	1
r a n d o m s t r i o m s t r i n g a b c d	0
h d g h r t z u s z j i t m o e r v y q d f	0
p r e d e f i n e d q u k w a r n g t o h d	1

As the class label is only provided at the end of each sequence, the network needs to bridge at least $T - L$ time steps to relate the label to the class-defining string. So, if L is kept fixed, the problem gets harder with increasing sequence length T. For the evaluation a fixed string $L = 50$ was used. Further, the length of the sequence T was increased gradually to test the networks ability to store the initial inputs over an arbitrary period of time. For each sequence length T two sets for training and testing were created. The sets were enlarged with increasing T to ensure generalisation. To determine the algorithms' ability to learn the task in general, 100 networks were trained with eRTRL and eBPTT, respectively. This was done for *every* sequence length T. Moreover, the sequences of the training set were shown in a random order in every epoch of the training.

The networks' configuration and the size of the training/test sets were adopted from Chen & Chaudhari (2004) where SMRNNs and SRNs are compared on the information latching problem. Accordingly, the SMRNNs comprised of $n_u = 26$ input units, $n_x = n_y = 10$ hidden layer units, and one output unit $n_z = 1$. Further, the length of a segment was set to $d = 15$ and the sigmoidal transfer function $f_{\text{net}}(x) = 1/\left(1 + \exp(-x)\right)$

was used for the hidden and output units. The input units simply forwarded the input data which were $\in \{-1, 1\}$. Initial weights were set to uniformly distributed random values in the range of $(-1, 1)$. The network output was assigned to one class by the boundary at 0.5, that is,

$$C = 1 \text{ if } z(t) \geq 0.5 \text{ and } C = 0 \text{ if } z(t) < 0.5. \tag{5.103}$$

Learning rate and momentum for each algorithm were chosen after testing 100 networks on all combinations $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ and $\eta \in \{0.1, 0.2, \ldots, 0.9\}$ on the shortest sequence $T = 60$. The shortest sequences were used for practical reasons, as these require the smallest amount of samples to ensure generalisation. By that all $\alpha\eta$-combinations could be tested within a reasonable amount of time. Those combinations that yielded the highest mean accuracy over 100 networks on the test set, were chosen for the comparison on longer sequences, that is, $\alpha = 0.1$, $\eta = 0.4$ for eRTRL and $\alpha = 0.6$, $\eta = 0.5$ for eBPTT. The learning rate and momentum found by this procedure are not necessarily optimal for longer sequences with $T > 60$. However, as both algorithms are compared, it need not be optimal, but comparable and be found in a reproducible manner.

Training was stopped when the mean squared error of an epoch fell below 0.01 and thus, the network was considered to have successfully learnt the task. For other cases, training was cancelled after 1000 epochs. Table 5.2 shows the results for eRTRL and eBPTT for sequences of length T from 60 to 130.

Table 5.2: eBPTT and eRTRL: Information latching problem with increasing sequence length T and fixed predefined string ($L = 50$). 100 SMRNNs with parameters $n_x = n_y = 10$, $d = 15$ were trained on each sequence length. The number of successfully trained networks (#suc of 100) is shown together with the mean value of the number of training epochs (#eps). Further, the mean accuracy of the successfully trained networks on the test set (ACC) and its standard deviation (STD) is shown.

T	set size	eBPTT				eRTRL			
		#suc	#eps	ACC	STD	#suc	#eps	ACC	STD
60	50	79	230.6	0.978	0.025	100	44.3	0.978	0.025
70	80	58	285.7	0.951	0.047	100	63.9	0.861	0.052
80	100	61	215.2	0.974	0.024	100	66.2	0.862	0.088
90	150	48	240.4	0.951	0.123	100	52.4	0.940	0.044
100	150	43	241.4	0.968	0.018	100	82.1	0.778	0.065
110	300	36	250.0	0.977	0.049	100	69.6	0.868	0.052
120	400	17	305.4	0.967	0.050	100	56.7	0.950	0.040
130	500	14	177.6	0.978	0.017	96	101.4	0.896	0.078
mean		44.5	243.3	0.968	0.044	99.5	67.1	0.892	0.056

The column "#suc" in Table 5.2 clearly shows a decrease of successfully trained networks for eBPTT with the length of the sequences T. On the other hand, nearly all

networks were trained successfully with eRTRL. Therefore, we can state that eRTRL is generally better able to cope with longer ranges of input-output dependencies than eBPTT.

The third column in Table 5.2 shows the performance of successfully trained networks on the test set (ACC). For eBPTT we observe higher accuracies than for eRTRL. It is also reflected by the overall accuracy of 96.8% for eBPTT compared to 89.2% for eRTRL. This implies, that successful learning with eBPTT guaranteed better generalisation.

Further, the mean number of epochs (#eps) that were needed for training is somewhat misleading. Over the whole experiment eBPTT needs an average of 243.3 epochs for successful training while eRTRL needs only 67.1 epochs. It is important to note that this does not indicate that eRTRL training takes less time than eBPTT. The high computational complexity of eRTRL ($\mathcal{O}(4n^6T^2)$ cf. Equation 5.55) results in a much longer computation time for a single epoch compared to eBPTT. This becomes more and more evident with increasing network size. Figure 5.5 shows the time that is needed to train an SMRNN for 100 epochs ($T = 60$, set size 50) depending on the number of units in the hidden layers[2]. The time-axis is scaled logarithmically to allow a comparison to Figure 5.4. For a network with $n_x = n_y = 100$ the training took about 3 minutes with eBPTT and 21.65 hours with eRTRL. For larger networks the training with eRTRL would have taken weeks, such that the experiment was stopped at $n_x = n_y = 100$ for practical reasons.

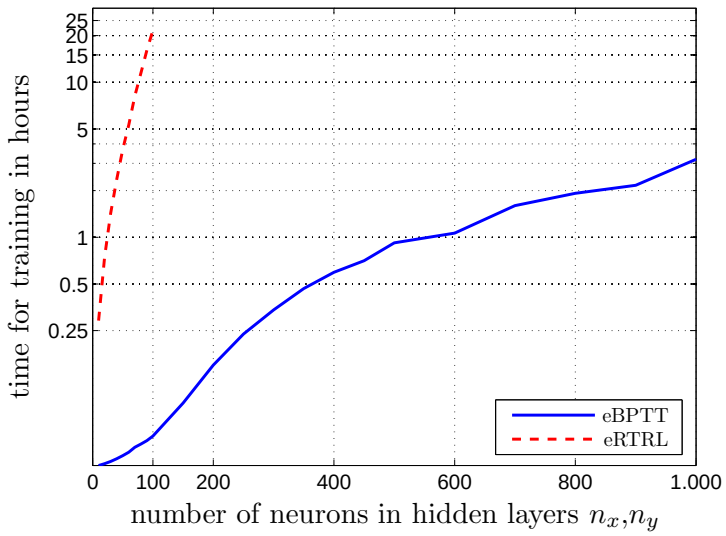


Figure 5.5: Computation time for training depending on the number of units in the hidden layers of the network. Training lasted 100 epochs with 50 sequences of length $T = 60$.

[2]Both algorithms were implemented in Matlab. Training was done on an AMD Opteron 8222 (3GHz), 8GB RAM, CentOS, Matlab R2011b (7.13.0.564) 64-bit.

5.6 Discussion

This chapter showed how the concept of a segmented memory may attenuate the problem of vanishing gradients during gradient based learning in recurrent neural networks. The SMRNN architecture implements this concept (cf. Section 5.2). Together with the architecture, the eRTRL algorithm was proposed for the network training. Unfortunately, it has a very high computational complexity, such that it is impractical for the training of large SMRNNs (cf. Section 5.3.2).

Alternatively, eBPTT was introduced (Section 5.4). It does not have the high computational costs of eRTRL, but the comparison on a benchmark problem showed that eRTRL was generally better able to cope with the latching of information over longer periods of time (Section 5.5). However, those networks that finally learnt the task with eBPTT showed higher accuracies on the test set.

Altogether, the question which learning algorithm to use for a specific task strongly depends on the character of the problem at hand. For small networks, as used for the experiment in Table 5.2, the choice depends on the time span that has to be bridged. If one expects the output to be dependent on inputs that are comparatively shortly ago ($T = 60, \ldots, 100$), eBPTT provides the better choice. There is a high chance for a successful training of the network with a good generalisation. When the outputs depend on inputs that appeared long ago ($T > 100$), the eRTRL algorithm provides the better solution. It guarantees a successful network training where eBPTT could hardly train the network.

In more complex problems, as speech recognition or handwriting recognition, the data has not such compact representation as the strings in the information latching task. To be able to learn from such data, the network size, that is, the number of processing units, has to be increased. As shown in Figure 5.5, eRTRL becomes simply impractical for such large networks (training time: 3 minutes eBPTT vs. 21.65 hours eRTRL where $n_x = n_y = 100$). In these cases eBPTT is the only viable choice of a training algorithm.

6 Unsupervised Pre-Training for Segmented-Memory Recurrent Neural Networks

Contents

The previous chapter introduced two learning algorithms for SMRNNs. An evaluation on the information latching problem (cf. Section 5.5) showed somewhat conflicting properties between both algorithms. With eRTRL the training was most likely successful for any timespan that had to be bridged between an input and its corresponding output. However, the generalisation, that is, the accuracy on the test set, was mediocre. For eBPTT on the other hand, we observed high accuracies on the test set while the probability of a successful training decreased dramatically for longer input-output dependencies (cf. Table 5.2).

The computational complexity of eRTRL makes it impractical for large networks (cf. Section 5.3.2), which most likely are required in complex applications. eBPTT has a much smaller computational complexity (cf. Section 5.4.2). Therefore, the focus is put on the improvement of eBPTT's ability to robustly latch longer ranges of input-output dependencies in the following. The approach is inspired by the attempts in the field of *deep learning*. Research on deep architectures aims at learning feature hierarchies with features from higher levels, formed by the composition of lower level features. Thereby, a system which automatically learns features at multiple levels of abstraction should be able to learn a complex function, mapping the input to the output directly from data (Bengio, 2009).

In the following a short introduction to the research on deep neural networks is given, which motivates the idea of unsupervised pre-training. Thereafter, possible ways to pre-train an SMRNN are described, for instance, as a stack of auto-encoder SRNs. The approaches are evaluated on the previously introduced information latching problem

(cf. Section 5.5). Results show that the pre-training significantly improves eBPTT's ability to cope with the learning of long-term dependencies.

The auto-encoder pre-training procedure, together with an evaluation on the information latching problem, was separately published as a contribution to the "European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning" (Glüge et al., 2013)[1].

6.1 Deep Neural Networks

The following overview shortly describes the main ideas of deep architectures and unsupervised pre-training. For an exhaustive introduction into the field of deep architectures and their training methods I suggest the article of Bengio (2009).

Following Bengio (2009), the *depth of an architecture* describes the number of levels of non-linear operations in the function that is composed during learning. Architectures with one, two or three levels are called *shallow architectures* and architectures with more than three levels are referred as *deep architectures*. In this sense, our brain is a deep architecture, which inspired researchers to train deep multi-layer neural networks (Utgoff & Stracuzzi, 2002; Bengio & LeCun, 2007).

Experimental results (Bengio et al., 2007; Erhan et al., 2009) show that the training of deep architectures is considerably more difficult as compared to shallow architectures. Using the standard random weight initialization, deep architectures generally produce poor training and generalization results (Bengio et al., 2007). This suggests, that the supervised gradient-based training of deep multi-layer networks gets stuck in local minima or plateaus. The effect gets even worse the deeper the architecture is.

Hinton et al. (2006) were the first who successfully trained a deep architecture named Deep Belief Network. They employed an unsupervised learning algorithm that trains only one layer at a time, a Restricted Boltzman Machine (Freund & Haussler, 1994). Subsequently, related algorithms using auto-encoders were proposed (Bengio et al., 2007; Ranzato et al., 2007). More recently, algorithms were introduced using neither Restricted Boltzman Machines nor auto-encoders (Mobahi et al., 2009; Weston et al., 2008). All approaches exploit the same principle, which is: unsupervised learning helps in the training of intermediate levels of representations and can be done locally at each level (Bengio, 2009).

In practice, each layer is pre-trained one after another starting at the lowest level with the actual input data (Hinton et al., 2006). The whole idea can be summarised as follows: first, the lowest layer is trained with an unsupervised learning algorithm (e.g. auto-encoder) to receive initial weights for this layer. Afterwards, the output of this layer is used as input for the next layer, which is trained in a similar way. This procedure may be repeated for several layers, until the whole network is initialised. Then, the network can be fine-tuned, according to some supervised training algorithm. Figure 6.1 illustrates the training procedure for a deep neural network with stacked auto-encoders.

[1] Parts of text in the Sections 6.2 and 6.3 are taken verbatim from this article.

The advantage of such pre-training was demonstrated in several statistical comparisons (Larochelle et al., 2007, 2009; Bengio, 2009; Erhan et al., 2009).

Note that only the input weights of each auto-encoder are used as initial weights after pre-training. Further, the representation of the inputs that were learnt in each layer, serves as input for the next layer. This layer, again, learns a representation of its input data, which results in a representation of the representation of the input data of the layer below. Following this scheme, higher-level features can evolve during pre-training of a deep architecture.

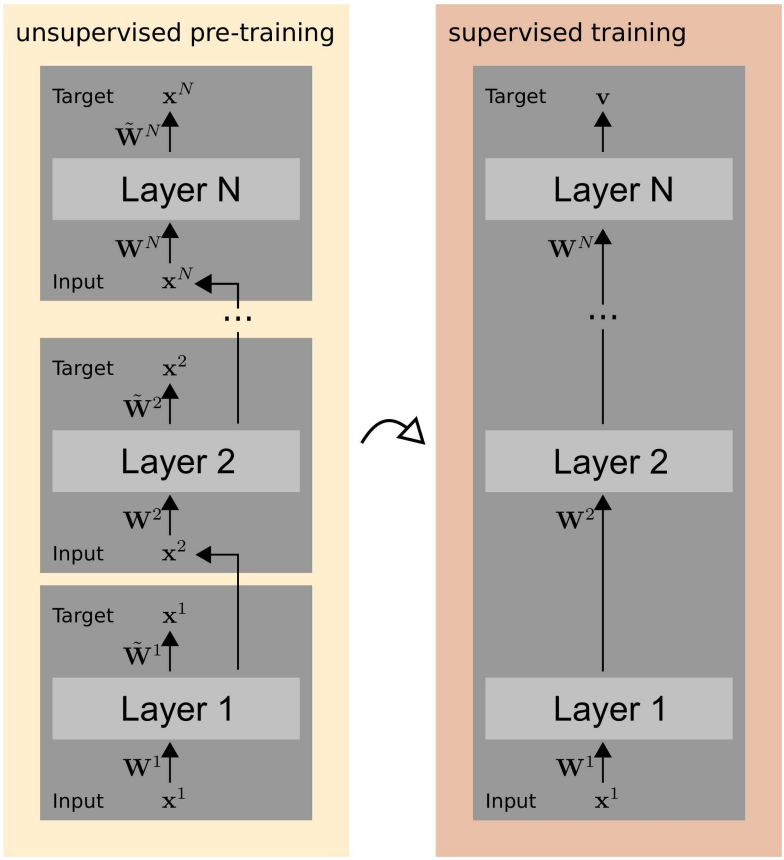


Figure 6.1: Stacked Auto-Encoders: training of a multi-layered deep neural network. During pre-training, each layer's weights are initialised separately by training as an auto-encoder $(\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^N)$. The first layer is trained with the actual input data $\mathbf{x}^1$. The output of each layer serves as input for the layer above $(\mathbf{x}^2, \mathbf{x}^3, \ldots, \mathbf{x}^N)$. Finally, the obtained weights are used as starting points for the supervised training on the actual target values $\mathbf{v}$.

The success of layer-local unsupervised pre-training may be due to several effects. Generally, it may help to guide the parameters of the layers towards better regions in parameter space. Of course, the question "What are 'better regions'?" arises in this context. To be more specific, unsupervised pre-training has a regularising effect that

leads to regions in the parameter space where solutions are allowed. That means, the solution is "near" to those of the unsupervised training, that is, near a solution that captures statistical structure of the input (Erhan et al., 2009). Experiments in Erhan et al. (2009) show that the effect of unsupervised pre-training is an advantage mostly for the lower layers of a deep architecture, as these layers are poorly trained with random parameter initialization.

With a better initialization of the lower layers by a pre-training, the training and generalization error during the supervised training can be reduced significantly. Bengio (2009) hypothesizes that in a "well trained" deep neural network the hidden layers form a good representation of the input data, which helps in obtaining good predictions. Further, the pre-training can be seen as a way to decompose the problem into sub-problems with different levels of abstraction. One layer of unsupervised learning could extract features which are regarded as *low-level* features, due to the limited capacity of one layer. Hence, learning a second layer using the previous layer's outputs should result in slightly *higher-level* features. Therefore, one could imagine that higher-level abstractions of the input emerge in a deep architecture (Bengio, 2009).

Generally, training an auto-encoder can be regarded to be easier than training a Restricted Boltzman Machine. Therefore, they have widely been used as building blocks of deep neural networks (Bengio et al., 2007; Larochelle et al., 2007; Ranzato et al., 2007; Vincent et al., 2008). An auto-encoder is trained to encode the input $\mathbf{x}$ into some representation $f(\mathbf{x})$, such that the input can be reconstructed from that representation. Accordingly, the target output of an auto-encoder is the input itself. The hope is that the evolving representation $f(\mathbf{x})$ captures the main factors of variation in the data (Bengio, 2009).

Note that the term 'unsupervised' in the context of pre-training does not indicate that the training of the auto-encoder itself is unsupervised as defined in Section 2.5. Actually, the training of an auto-encoder is supervised, as target values are provided during training. The motivation to call it nevertheless 'unsupervised' stems from the fact, that the input data itself is used as target during pre-training. The original target data is only used in the second step, when the completely initialised network is trained (cf. Figure 6.1).

6.2 Auto-Encoder Pre-Training of Segmented-Memory Recurrent Neural Networks

If one regards an SMRNN as a stack of two SRNs (cf. Figure 5.1), the idea of a layer-local pre-training seems quite plausible. Even though the architecture itself may not be regarded as *deep* in the conventional way, the recurrent character of a hidden–context layer pair allows the composition of a complex non-linear operation. Therefore, such layer-pair can be viewed as being *deep* in itself. Consequently, unfolded in time, a recurrent network can be seen as a very deep multi-layer neural network.

The positive results reported with the pre-training of deep neural networks (cf. Section 6.1) give rise to the hope that SMRNNs could also benefit from a pre-training procedure. Generally, it should lead to initial weights that lie in a region of the parameter space were it is more likely to find a solution. Therefore, a pre-training should help in rising the probability of a successful training with eBPTT.

Following the idea of layer-local pre-training, the single SRNs on the symbol and segment level are separately trained as auto-encoders. In that way, the procedure does not differ much from the pre-training of multi-layer neural networks, as described above. In an SMRNN the segment level processes the input of the symbol level only at the end of a segment. Hence, only these symbol level outputs are used for the segment level pre-training. So, for segment length d every d^{th} output of the symbol level auto-encoder SRN is used for the segment level pre-training. Hereafter, the initialised weights are used as starting points for the supervised training. Figure 6.2 illustrates this pre-training procedure. Such pre-training is independent of the learning algorithm that is used in the supervised training and vice versa.

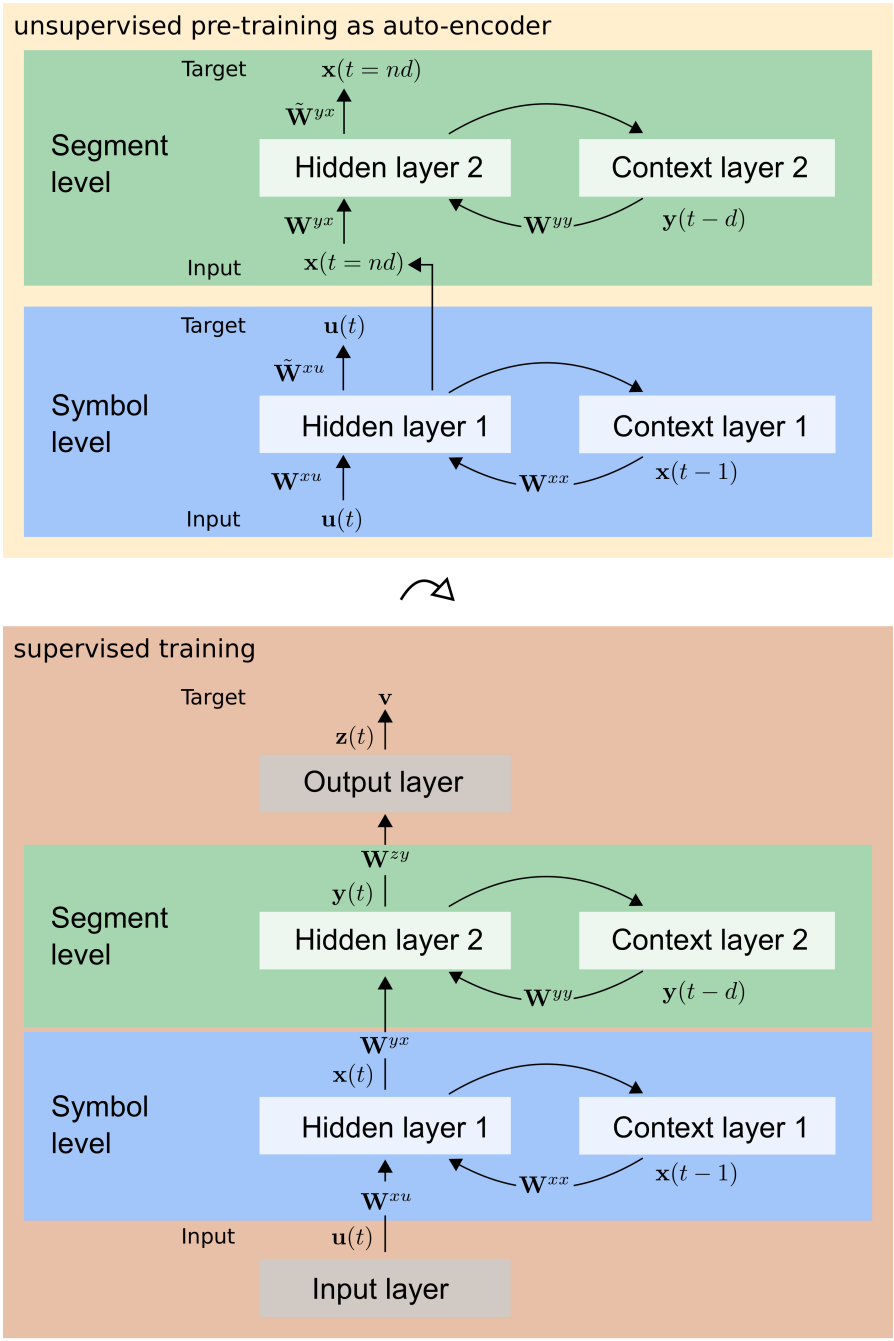


Figure 6.2: Auto-Encoder pre-training of an SMRNN. During pre-training, each SRN's weights are initialised separately by training as an auto-encoder ($\mathbf{W}^{xu}, \mathbf{W}^{xx}, \mathbf{W}^{yx}, \mathbf{W}^{yy}$). Thereby, the output of the symbol level SRN serves as input for the segment level SRN ($\mathbf{x}(t = nd)$ with $n = 1, \ldots, N$), which is trained in the same way. The weights obtained during pre-training are used as starting points for the supervised training on the actual desired targets $\mathbf{v}$.

6.3 Pre-Trained SMRNNs on the Information Latching Problem

To reveal the effect of an auto-encoder pre-training on SMRNNs, the information latching problem as described in Section 5.5 is used. Again, a fixed string with length $L = 50$ was used, and the length of the sequence T was increased gradually. For each sequence length T, two sets for training and testing were created and enlarged with increasing T. Further, 100 networks were pre-trained and randomly initialised, respectively. This was done for *every* sequence length T. Moreover, the sequences of the training set were shown in a random order in every epoch of the training. The random initial weights were set to uniformly distributed random values in the range of $(-1, 1)$. The eBPTT algorithm was applied in the supervised learning phase.

Pre-training of the symbol and segment level SRN was performed as described in Section 6.2 and depicted in Figure 6.2. Each network was trained to reproduce its input at the output with the scaled conjugate gradient backpropagation algorithm (Møller, 1993). For the pre-training, as for the supervised training afterwards, the same data was used. From this training material, 80% were shown for pre-training and 10% for testing and validation of the pre-training, respectively. During pre-training, the sequences were presented one after another, such that the sequential structure of the material was preserved. This is desirable, as SRNs are able to learn the temporal structure of the data implicitly, even though auto-encoders in general learn a direct input-output mapping (cf. Chapter 4). Pre-training was stopped after 1000 epochs or when the validation performance has increased more than six times since the last time it decreased. Such early stopping is widely used, because it is comparably easy to implement and was reported to be superior to regularization methods like Weight Decay (Werbos, 1988) and Weight Elimination (Chauvin, 1990), for instance in Finnoff et al. (1993). A further discussion on the selection of a good stopping criterion and experimental results for different criteria is being provided in Prechelt (1998).

After pre-training the weights $(\mathbf{W}^{xu}, \mathbf{W}^{xx}, \mathbf{W}^{yx}, \mathbf{W}^{yy})$ were used for the supervised eBPTT training on the information latching problem.

The same network configuration as in Section 5.5 was used for the experiment. That is, $n_u = 26$ input units, $n_x = n_y = 10$ hidden layer units, and one output unit $n_z = 1$. The input units, again, simply forwarded the input data $\mathbf{u}(t) \in \{-1, 1\}$. Further, the length of a segment was set to $d = 15$, and the output layer used the sigmoidal transfer function $f(x) = 1/\left(1 + \exp(-x)\right)$. Unlike in Section 5.5, the hyperbolic tangent $f(x) = \tanh(x)$ was applied in the hidden layers. This is advantageous as a symmetric activation function about 0 allows error gradients to flow backwards more easily (Glorot & Bengio, 2010). However, the hyperbolic tangent was not used in the comparison of eBPTT and eRTRL in Section 5.5, as it was intended to use *exactly* the same network structure that was used in the original article (Chen & Chaudhari, 2009), which introduced eRTRL. Again, the network output was assigned to one class by the boundary at 0.5, that is,

$$C = 1 \text{ if } z(t) \geq 0.5 \text{ and } C = 0 \text{ if } z(t) < 0.5. \tag{6.1}$$

With the change of the transfer functions in the hidden layers, the optimal combination of learning rate and momentum for the eBPTT training had to be determined again. Therefore, 100 randomly initialised networks were trained with each combination $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ and $\eta \in \{0.1, 0.2, \ldots, 0.9\}$ on the training set of the shortest sequence $T = 60$. The combination $\alpha = 0.2$ and $\eta = 0.1$ yielded the highest mean accuracy on the test set and thus, was used in the further experiment.

The supervised eBPTT training was stopped when the mean squared error of an epoch fell below 0.01 and thus, the network was considered to have successfully learnt the task. For other cases, training was cancelled after 1000 epochs.

Table 6.1 shows the results for pre-trained and randomly initialised SMRNNs on the information latching problem with fixed predefined string of length $L = 50$, and increasing sequence length T. 100 SMRNNs were trained and tested on each sequence length T from 60 to 130.

Table 6.1: Randomly initialised and pre-trained SMRNNs on the information latching problem with increasing sequence length T and fixed predefined string ($L = 50$). 100 SMRNNs with parameters $n_x = n_y = 10$, $d = 15$ were trained on each sequence length. The number of successfully trained networks (#suc of 100) is shown together with the mean value of the number of training epochs (#eps). Further, the mean accuracy of the successfully trained networks on the test set (ACC) and its standard deviation (STD) is shown.

T	set size	randomly initialised				pre-trained as auto-encoder			
		#suc	#eps	ACC	STD	#suc	#eps	ACC	STD
60	50	80	122.6	0.966	0.061	89	53.7	0.964	0.040
70	80	83	80.3	0.962	0.040	98	43.8	0.975	0.046
80	100	65	123.3	0.968	0.038	90	37.3	0.981	0.019
90	150	41	180.3	0.978	0.022	85	26.8	0.984	0.017
100	150	37	147.1	0.971	0.023	88	28.6	0.984	0.017
110	300	26	204.2	0.980	0.010	73	24.5	0.982	0.055
120	400	16	239.6	0.954	0.123	56	34.1	0.991	0.016
130	500	6	194.8	0.987	0.011	59	32.5	0.990	0.011
mean		44.5	161.5	0.972	0.041	79.8	35.2	0.981	0.028

As in the previous chapter in Table 5.2, in the column "#suc" one can observe the decrease of successfully trained networks with an increase of the length of the sequences T. This general trend holds for randomly initialised as well as for pre-trained SMRNNs. However, the pre-trained networks do not suffer from that behaviour as much as the randomly initialised ones. For the longest sequence $T = 130$, 59 out of 100 networks were trained successfully when pre-trained, while only 6 out of 100 were obtained from random initialisation. The accuracy on the test set (ACC) and its standard deviation (STD) does not differ significantly for both cases, and confirms the values for eBPTT in Chapter 5 (cf. Table 5.2).

Concerning the training time of pre-trained networks, significantly less epochs (#eps) for the supervised training were needed. At an average 161.5 epochs were neede from random initial weights compared to 32.5 epochs from pre-trained weights. Nevertheless, one has to notice, that the whole training procedure consists of a pre-training plus supervised training. Therefore, the time that is saved during the supervised training is partly spend on the pre-training procedure. So, the total saving of time for the training strongly depends on the time complexity of the learning algorithm that is used for the auto-encoder pre-training.

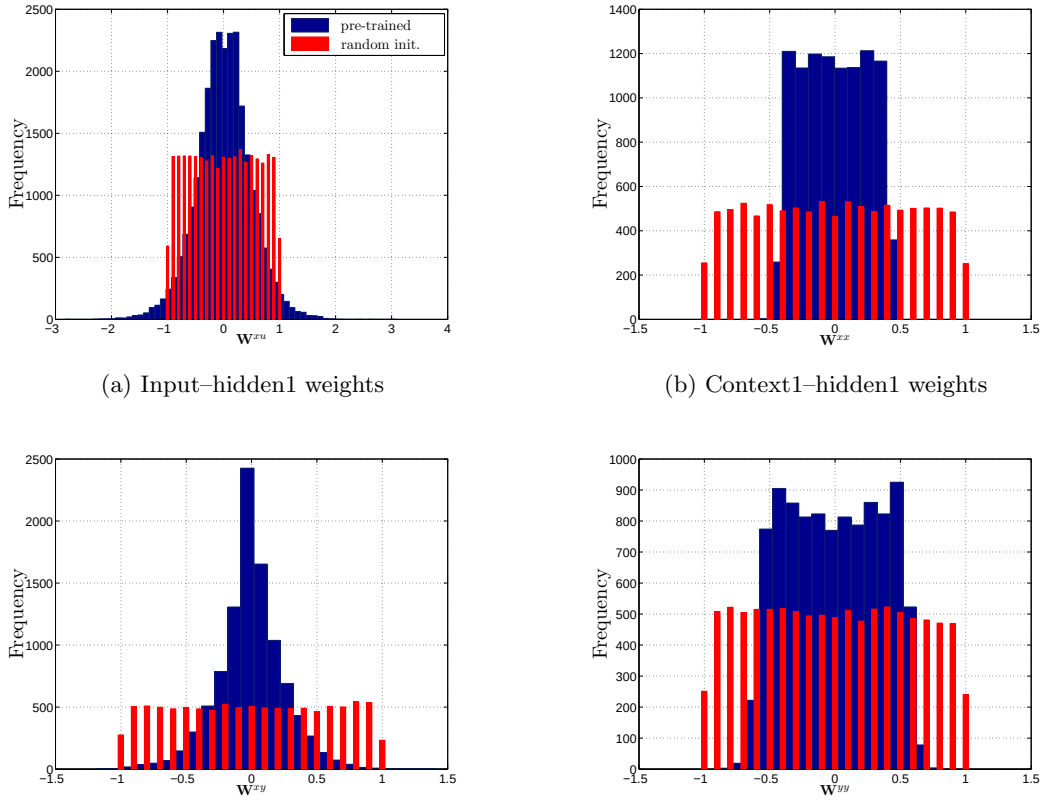
Nonetheless, the pre-training led to a distinct increase of the number of successfully trained networks, especially for long input-output dependencies, that is, $T \geq 90$.

6.3.1 Effect of the auto-encoder pre-training

To understand the effect of the pre-training, it is useful to have a look at the network weights. The weights are first of all initialised uniformly distributed in the interval $(-1, 1)$, such that the observation of a single network is somewhat unreliable for a general statement. Nevertheless, statistics over 100 networks yields a rather universal trend caused by the pre-training procedure. Figure 6.3 shows the weight distribution of the single weight matrices for random initialised (red) and pre-trained (blue) SMRNNs. The distribution of the random weights (red bars) taken by itself is of limited interest, as it is known a priori. It is plotted first and foremost for the comparison with the weight distribution after pre-training (blue bars). Even though Figure 6.3 shows the weight distributions for the SMRNNs trained on the information latching problem with sequence length $T = 120$, the same behaviour was observed for all other lengths $T = 60, \ldots, 130$.

For the direct forward connections of the SMRNNs $\mathbf{W}^{xu}$ (Figure 6.3a) and $\mathbf{W}^{xy}$ (Figure 6.3c), pre-training shifted the weights towards a distribution which appears Gaussian-shaped with a mean near zero. Compared to the initial random distribution, most of the weights attained smaller absolute values around zero, while a small number of weights attained bigger absolute values, grater than 1 or smaller than -1. This leads to the hypothesis, that the representation of the input in the hidden layers after pre-training is based on a rather small number of weights with large absolute values.

Interestingly, weights in the recurrent connections on the symbol level and segment level, that is, $\mathbf{W}^{xx}$ (Figure 6.3b) and $\mathbf{W}^{yy}$ (Figure 6.3d), show a different trend. They are still uniformly distributed after pre-training but in a smaller interval. This means, at average, pre-training uniformly lowered the absolute value of all weights in the context layer. In other words, the weights tended to zero. According to the study in Chapter 4 vanishing context weights indicate that no temporal dependency in the input has been learnt (cf. Section 4.2 and Figure 4.7).



(a) Input–hidden1 weights

(b) Context1–hidden1 weights

(c) Hidden1–hidden2 weights

(d) Context2–hidden2 weights

Figure 6.3: Each plot shows the weight distribution of a certain weight matrix, averaged over 100 SMRNNs before and after pre-training, that is, random initialised (red) and pre-trained (blue). Red bars are displayed thinner than blue bars to facilitate a direct comparison. The SMRNNs were pre-trained on the information latching problem with sequences of length $T = 120$.

The vanishing context weights may indicate that learning during pre-training mainly takes place in the direct forward connections of the network ($\mathbf{W}^{xu}$, $\mathbf{W}^{xy}$). This hypothesis was tested by setting the context weights $\mathbf{W}^{xx}$ and $\mathbf{W}^{yy}$ to zero after pre-training. Then, the supervised training was done as before. Table 6.2 shows the results of this experiment.

Table 6.2: Pre-trained SMRNNs with and without reset of the context weights: Information latching problem with fixed predefined string $L = 50$ and increasing sequence length T. 100 pre-trained SMRNNs with and without a reset of their context weights ($\mathbf{W}^{xx}$, $\mathbf{W}^{yy}$) to zero were trained with eBPTT for each sequence length T. The number of successfully trained networks (#suc of 100) is shown together with the mean value of the number of training epochs (#eps). Further, the mean accuracy of the successfully trained networks on the test set (ACC) and its standard deviation (STD) is shown.

T	set size	pre-trained as auto-encoder				pre-trained as auto-encoder and context reset to zero			
		#suc	#eps	ACC	STD	#suc	#eps	ACC	STD
60	50	89	53.7	0.964	0.040	97	47.8	0.964	0.049
70	80	98	43.8	0.975	0.046	96	33.0	0.972	0.045
80	100	90	37.3	0.981	0.019	95	60.8	0.973	0.028
90	150	85	26.8	0.984	0.017	83	77.1	0.977	0.021
100	150	88	28.6	0.984	0.017	77	46.0	0.975	0.059
110	300	73	24.5	0.982	0.055	80	52.4	0.984	0.036
120	400	56	34.1	0.991	0.016	61	78.5	0.972	0.087
130	500	59	32.5	0.990	0.011	57	65, 7	0.989	0.016
mean		79.8	35.2	0.981	0.028	80.8	57.7	0.976	0.043

The reset to zero of the context weights after pre-training apparently had no effect on the result of the supervised training. Other than some individual variations, there is no significant difference in the number of successfully trained networks (#suc), nor a significant difference in accuracy (ACC) (cf. Table 6.2). This affirms the impression that we gained from Figurs 6.3b and 6.3d, that no temporal dependency in the input was learnt during pre-training.

Of course, there are temporal dependencies between single characters of the input sequences, especially for those that occur in the keyword at the beginning (cf. Table 5.1). Yet, it seems that this dependencies are not well pronounced due to the complexity of the sequences, such that they are not learnt implicitly during pre-training.

6.3.2 Alternative Context Weight Initialisation

In Section 6.3.1 it was shown, that the auto-encoder pre-training procedure mainly affects the direct forward connections of the SMRNN, while the context weights tend to zero (cf. Figure 6.3). Even a complete reset of the context weights had no significant effect on the subsequent supervised training (cf. Table 6.2). This supports the conclusion that during auto-encoder pre-training, only representations of the actual input vectors are learnt in the hidden layer weights ($\mathbf{W}^{xu}$ and $\mathbf{W}^{xy}$). At the same time, no temporal dependencies between those representations are learnt.

This is not a general behaviour of SRNs, but rather a consequence of the pre-training as auto-encoder. Symbol level, as well as, segment level SRN do not need to encode temporal relations between their inputs to reproduce the actual input at the output. Therefore, the networks learn a direct representation of the input in the hidden layer. Further, due to the complexity of the temporal dependencies between the single inputs, this context is not learnt implicitly, which results in context weights around zero (cf. Section 4.2).

Forcing Error Propagation through Context Weights

For the task of information latching, the learning of representations for the single characters in the hidden layers during pre-training as auto-encoder turned out to be beneficial (cf. Table 6.1). However, the zeroed context weights, resulting from such pre-training, are rather counter-intuitive. The class of a string is provided at the end of the sequence, that is, at the last letter (cf. Table 5.1). The important information, determining the class, is the keyword at the beginning of the string. Therefore, it seems deceptive to use zeroed context weights, preventing a flow of information from one state to its previous state, or in other words, from the end of the string to its beginning.

A straightforward solution for this dilemma is the replacement of the context weights after the pre-training as auto-encoder. By setting the recurrent weights $\mathbf{W}^{xx}$ and $\mathbf{W}^{yy}$ to the identity matrix, one can force the network to provide the previous network state, when processing the actual input. This should facilitate the information flow from the end of the sequence to its beginning during supervised learning. Metaphorically spoken, the supervised training is started with a network which has the basic assumption, that its previous internal state is as important as the actual input.

To evaluate this approach, the experiment from Section 6.3.1 was repeated. This time, after pre-training as auto-encoder, the context weights of the networks $\mathbf{W}^{xx}$ and $\mathbf{W}^{yy}$ were replaced by the corresponding identity matrices. The remaining parameters, as network configuration, training algorithms, learning rate, training/test material, etc. were kept unchanged. Table 6.3 shows the results obtained from plain pre-training as auto-encoder along with the results obtained if the context weights were replaced.

The replacement of the context weights by the identity matrix increased the number of successfully trained networks (#suc) at average by 13.5% (79.8 versus 90.6). Further, there is no significant difference in the number of training epochs (#eps), nor a significant difference in accuracy (ACC) and its standard deviation (STD). In conclusion, the replacement of the recurrent weights of the SMRNNs by a simple identity matrix enforced

Table 6.3: Pre-trained SMRNNs with and without replacement of the context weights: Information latching problem with fixed predefined string $L = 50$ and increasing sequence length T. 100 pre-trained SMRNNs with and without a replacement of their context weights ($\mathbf{W}^{xx}$, $\mathbf{W}^{yy}$) by the identity matrix were trained with eBPTT for each sequence length T. The number of successfully trained networks (#suc of 100) is shown together with the mean value of the number of training epochs (#eps). Further, the mean accuracy of the successfully trained networks on the test set (ACC) and its standard deviation (STD) is shown.

T	set size	pre-trained as auto-encoder				pre-trained as auto-encoder and context replaced by identity matrix			
		#suc	#eps	ACC	STD	#suc	#eps	ACC	STD
60	50	89	53.7	0.964	0.040	98	60.5	0.977	0.024
70	80	98	43.8	0.975	0.046	99	30.4	0.969	0.103
80	100	90	37.3	0.981	0.019	98	26.9	0.985	0.050
90	150	85	26.8	0.984	0.017	98	29.2	0.975	0.085
100	150	88	28.6	0.984	0.017	96	27.2	0.994	0.008
110	300	73	24.5	0.982	0.055	88	26.0	0.995	0.007
120	400	56	34.1	0.991	0.016	74	45.3	0.988	0.056
130	500	59	32.5	0.990	0.011	74	47.6	0.994	0.019
mean		79.8	35.2	0.981	0.028	90.6	36.6	0.985	0.044

the backpropagation of the error signal, and thereby supported learning of long-term dependencies.

It should be noted that the modification of the weights followed the discussion about the information latching problem. This implies prior knowledge about the nature of the task to be solved, which in most cases is not available. However, generally it might be a good idea to support the error propagation through time at the beginning of the supervised training by the manual setup of the recurrent connections.

Pre-Training as Predictor

One way to enforce sequence learning during pre-training could be an SRN pre-training as predictor. That is, the SRNs are pre-trained to predict the *next* input of the sequence. Concerning the learning task, this is essentially different from the pre-training as auto-encoder, where SRNs are pre-trained to predict the *actual* input of the sequence. In theory, the prediction task forces a network to learn the temporal relations between inputs.

Technically, the pre-training procedure itself does not differ very much from the auto-encoder pre-training introduced in Section 6.2 and illustrated in Figure 6.2. The single SRNs on the symbol and segment level are separately trained as predictors. Again, the segment level processes the input of the symbol level only at the end of a segment. Hence,

only these symbol level outputs are used for the segment level pre-training. Afterwards, the initialised weights are used as starting points for the supervised training. Figure 6.4 illustrates the pre-training as predictor.

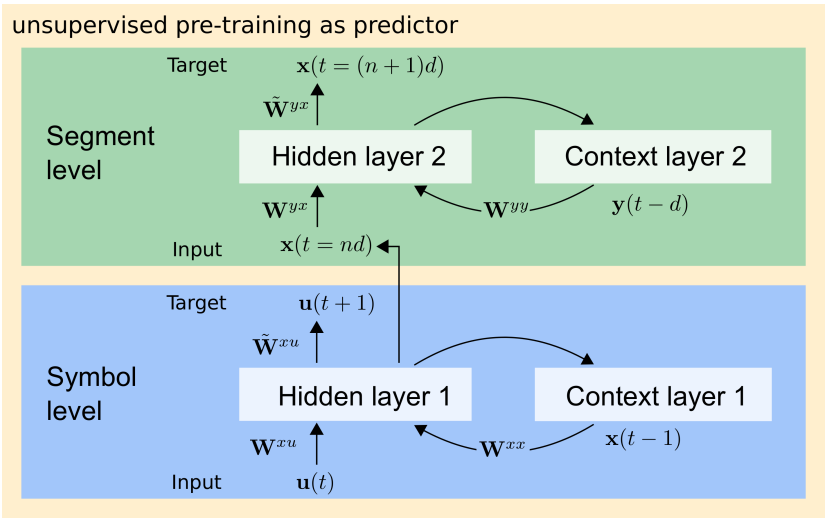


Figure 6.4: SMRNN pre-training as predictor. During pre-training, each SRN's weights are initialised separately by training as a predictor ($\mathbf{W}^{xu}, \mathbf{W}^{xx}, \mathbf{W}^{yx}, \mathbf{W}^{yy}$). That is, the SRNs are trained to predict the next element of the input sequence. The output of the symbol level SRN serves as input for the segment level SRN ($\mathbf{x}(t=nd)$ with $n=1,\ldots,N$), which is trained in the same way.

The procedure of a pre-training as predictor was evaluated on the information latching problem according to the evaluation of the pre-training as auto-encoder described in Section 6.3. The only difference was the weight initialisation by a pre-training as predictor (cf. Figure 6.4) instead of pre-training as auto-encoder (cf. Figure 6.2). All parameters such as network configuration, training algorithms, learning rate, training/test material, etc. remained unchanged. Table 6.4 shows the results obtained from random initialisation and pre-training as predictor side by side.

Apparently the pre-training as predictor provides only a slight advantage over randomly initialised weights. There is a small improvement of successfully trained networks (#suc), especially for longer sequences. However, compared to the pre-training as auto-encoder (cf. Table 6.1) it performs rather poorly.

There are several aspects that might explain this behaviour. First of all, no direct representation of the input data can be learnt during pre-training as predictor. As the task is to predict the following input, the networks fail to learn a distinct representation for the single inputs. Further, after pre-training, only the input weights of the layers ($\mathbf{W}^{xu}, \mathbf{W}^{xx}, \mathbf{W}^{yx}, \mathbf{W}^{yy}$) are used as starting points for the supervised training. The output weights of the layers ($\tilde{\mathbf{W}}^{xu}, \tilde{\mathbf{W}}^{yx}$) are discarded (cf. Figure 6.4).

This is not a problem for an auto-encoder, as input and output are the same. The mapping to be learnt is for instance $\mathbf{x}(t) \rightarrow \mathbf{x}(t)$. Hence, input weights and output

Table 6.4: Randomly initialised and pre-trained SMRNNs as predictor on the information latching problem with increasing sequence length T and fixed predefined string ($L = 50$). 100 SMRNNs were trained with eBPTT for each sequence length T. The number of successfully trained networks (#suc of 100) is shown together with the mean value of the number of training epochs (#eps). Further, the mean accuracy of the successfully trained networks on the test set (ACC) and its standard deviation (STD) is shown.

T	set size	randomly initialised				pre-trained as predictor			
		#suc	#eps	ACC	STD	#suc	#eps	ACC	STD
60	50	80	122.6	0.966	0.061	89	131.7	0.964	0.056
70	80	83	80.3	0.962	0.040	82	107.7	0.977	0.024
80	100	65	123.3	0.968	0.038	74	110.0	0.972	0.039
90	150	41	180.3	0.978	0.022	60	86.2	0.980	0.019
100	150	37	147.1	0.971	0.023	64	126.8	0.975	0.020
110	300	26	204.2	0.980	0.010	36	77.9	0.989	0.011
120	400	16	239.6	0.954	0.123	20	96.9	0.986	0.015
130	500	6	194.8	0.987	0.011	25	126.2	0.987	0.011
mean		44.5	161.5	0.972	0.041	56.3	107.9	0.979	0.024

weights, $\mathbf{W}^{xu}$ and $\tilde{\mathbf{W}}^{xx}$, are symmetric. In other words, the output weights carry no further information concerning the representation of the input.

This is not the case for the predictor. Here, the input and output of a layer differ. The mapping to be learnt is for instance $\mathbf{x}(t) \rightarrow \mathbf{x}(t+1)$ (cf. Figure 6.4). Therefore, input and output weights of a layer do not hold the same information regarding the representation of the input. That means, after pre-training as predictor, the information that was learnt in the output weights $\tilde{\mathbf{W}}^{xu}$ and $\tilde{\mathbf{W}}^{xx}$ is not available for the supervised training.

Additionally, in the experiment, the context weights $\mathbf{W}^{xx}$ and $\mathbf{W}^{yy}$ still inclined to zero. Thus, the hope, that the prediction task forces the network to learn the temporal relations between inputs, was not fulfilled. Still, no temporal relation between inputs could be learnt during pre-training.

6.4 Discussion of the Pre-Training Procedure

From the research in the field of deep neural networks we know about the positive effects of an unsupervised pre-training (cf. Section 6.1). It is reasonable to apply this idea to recurrent neural networks (Vinyals et al., 2012) as well as to SMRNN, since those can be regarded to be deep architectures. The SMRNN architecture is a stack of two SRNs, such that it suggests itself to attempt a pre-training as a stack of auto-encoders. In that way, the symbol level and segment level SRN can be pre-trained independently.

The evaluation of randomly initialised and pre-trained SMRNNs on the information latching problem showed that pre-training improves eBPTT's ability to learn long-term dependencies significantly. It reduces the chance to get stuck in local minima or plateaus and therefore, increases the number of successfully trained networks (cf. Table 6.1). The pre-training showed no effect on the generalisation error of the supervised eBPTT training, which may be a consequence of the very low error that is already achieved with random initialised weights. On the other hand, this means that there is the possibility that pre-training improves the generalisation error of a supervised eRTRL training. However, I did not apply pre-training to eRTRL as this would further increase the already unreasonable long training time. From my point of view this is possible but impractical, especially in respect of real world applications where rather large networks are used.

The auto-encoder pre-training procedure mainly affects the direct forward connections $\mathbf{W}^{xu}$ and $\mathbf{W}^{xy}$ of the SMRNN, while the context weights $\mathbf{W}^{xx}$ and $\mathbf{W}^{yy}$ tend to zero (cf. Figure 6.3). Even a complete reset of the context weights had no significant effect on the subsequent supervised training (cf. Table 6.2). This supports the conclusion, that during pre-training only representations of the actual input vectors are learnt, and no temporal dependencies are found between them. This is not a general behaviour of SRNs, but rather a consequence of the pre-training as auto-encoder. The SRN does not need to encode temporal relations between the inputs to reproduce the input at the output. Therefore, the network learns a direct representation of the input in the hidden layer. Further, due to the complexity of the temporal dependencies between the inputs, the SRNs 'learn' that there is no temporal relation between them, that is, the context weights tend to zero (cf. Section 4.2).

A replacement of the context weights by the identity matrix could further increased the number of successfully trained networks (cf. Table 6.3). It should be noted that the modification of the weights followed the discussion about the information latching problem. This implies prior knowledge about the nature of the task to be solved, which in most cases is not available. However, generally it might be a good idea to support the error propagation through time at the beginning of the supervised training by the manual setup of the recurrent connections.

The pre-training as predictor could hardly improve learning, compared to a random weight initialisation. The networks could not be forced to learn the temporal relations between inputs during such pre-training. Furthermore, the prediction task made it more difficult to learn a direct representation of the data in the forward connections, which resulted in a poor performance in supervised training.

Concerning the question which learning algorithm to choose we can conclude, that for

learning tasks which require large networks, that is, a large number of processing units, eRTRL is impractical due to its computational complexity. Thus, eBPTT is the only viable choice of a training algorithm in these cases. Further, pre-training extends the area of application of eBPTT to long(er)-term dependencies in sequence classification. Compared to eRTRL, it guarantees a better generalisation with a less time consuming training.

Finally, even though the information latching problem is a widely used benchmark task, it is exclusively constructed to test a system's ability to store information over some period of time. Hochreiter & Schmidhuber (1997b) consider it to be a trivial task, as it may be solved by random weight guessing. Regarding this objection, eBPTT yet did not prove to be applicable to nontrivial tasks. However, on the basis of the current results and the successful application of SMRNN with eRTRL in protein secondary structure prediction (Chen & Chaudhari, 2009) and speech processing (Glüge et al., 2011a) one can expect that eBPTT is also applicable to such nontrivial tasks.

7 Summary and Outlook

In this thesis, the topic of implicit sequence learning in recurrent neural networks was explored in a number of steps. Starting with the motivation for cognitive modelling in general, a short literature review on the research concerning *implicit learning* was given. Additionally, the concepts of supervised, unsupervised, and reinforcement learning, as defined in the machine learning community, were introduced. This was complemented with a short overview on the field of sequence learning in that domain.

Afterwards, two different computational models were introduced for the conditional associative learning scenario, which is used to study implicit learning of a task-irrelevant temporal context. First, the Markov model (Section 3.1) which describes learning by means of a stochastic process and second, the connectionist Simple Recurrent Network (SRN) model (Section 3.2).

With the Markov model mainly, the explicit part of the learning task was studied. The biologically motivated associative learning task was mapped onto 16 possible states plus state transitions. The transitions were deduced from the assumption of a rational behaviour. Further, three typical human mistakes were captured by separate parameters. The analysis of the model (cf. Section 3.1.3) yielded two predictions: (i) the probability to memorise a success correctly affects the slope of the learning curve and, (ii) the probability to remember previous actions on an object is responsible for the long-term development of the success probability. Thus, the explicit design of the model led to explicit statements concerning the experimental setup. At the same time, it requires a number of assumptions that are specific for the conditional associative learning task.

In contrast, the SRN model does not rely on explicit assumptions concerning the behaviour of the subjects, and can be considered to be closer to the biological reality. It aims towards the implicit learning of the task-irrelevant temporal context. The effect of a higher learning rate on objects that were presented in a deterministic instead of a random order, could be reproduced qualitatively by the SRNs. Further, it was found that the *direct succession of objects* is the key element for the networks to learn the temporal relation.

Following this, the mechanisms of implicit sequence learning in SRNs were investigated (cf. Section 4). A 4-2-4 auto-encoder SRN was used for this purpose. The task was constructed such that the sequential order of the input was not relevant for the solution. In that way, learning of the input sequence was implicit.

The weights of the context layer were found to be the major factor in terms of sequence learning in SRNs. If no sequential information is present, this weights tend to zero and turn an SRN into a Feed-Forward Network (cf. Figure 4.7). However, if sequential information is provided during learning, the network may not learn the exact sequence

in the context layer. Instead, the input sequence may be reproduced only in combination with an activation from the input layer.

The question how implicitly learnt temporal information is represented in SRNs could be answered partly for the considered task. For the most prominent sequence representations that evolved during learning, a geometric representation was found. A rotation through the four quadrants of the two dimensional state-space realised a cycle through the four states that represented the networks' input. Similarly, a reflection of a state vector into its opposite and reverse yielded an oscillation between two states.

The ability of recurrent networks to learn temporal dependencies implicitly is not just of interest as an aspect in cognitive modelling. In fact, in the context of machine learning, this ability is advantageous for technical sequence classification and prediction tasks. Regarding this, SRNs are often inapplicable, as they suffer from the problem of vanishing gradients (cf. Section 5.1), and therefore fail to learn long-term dependencies. The Segmented-Memory Recurrent Neural Network (SMRNN) architecture is one way to reduce this problem. The extended Backpropagation Through Time (eBPTT) training algorithm for this kind of networks was introduced, and it was shown that it requires dramatically less operations than the established extended Real-Time Recurrent Learning (eRTRL) algorithm (cf. Section 5.4). However, the comparison on the information latching problem showed that eBPTT is generally less capable to learn the latching of information for longer periods of time than eRTRL (cf. Section 5.5). Nevertheless, if a network was trained successfully with eBPTT, it yielded a better generalisation, that is, higher accuracy on the test set. Additionally, due to the computational complexity of eRTRL, eBPTT is the only viable choice of a training algorithm for large networks.

Subsequently, eBPTT's ability to learn long-term dependencies should be improved. The resulting approach is based on the idea of a layer-local pre-training for deep multilayer neural networks known from the deep learning literature. Transferring this idea to SMRNNs, the single SRNs on symbol and segment level are separately pre-trained as auto-encoders. Afterwards, the resulting initial weights are used as a starting point for the supervised training (cf. Section 6.2). The evaluation on the information latching problem yielded a significant improvement in the learning of long-term dependencies with eBPTT. For the longest sequence, corresponding to the longest input-output dependencies, 59 out of 100 networks were trained successfully when pre-trained against 6 out of 100 when randomly initialised. Concerning the accuracy on the test set, no difference was found between both cases.

Further investigations showed that the auto-encoder pre-training mainly affects the direct forward connections of the SMRNNs. The context weights tended to zero (cf. Figure 6.3), that is, during pre-training no temporal dependency was found between the network inputs.

All in all, pre-training allows learning of long(er)-term dependencies with eBPTT. Thus, the less time consuming eBPTT training together with the pre-training, opens the possibility for the application of SMRNNs to more complex problems where rather large networks are required.

Suggestions for Future Work

It is often the case with scientific investigations that more questions than answers are produced. Concerning the proposed implicit learning models, introduced in Chapter 3, the impact has to be discussed in the corresponding scientific community of psychology and behavioural biology. From a technical point of view, the Markov model makes predictions on the role of the reinforcement signal and the influence of the probability of a subject remembering the learning history on an object. Those insights might be used in the design of future experiments.

Regarding sequence learning in the machine learning domain, it was shown that SRNs provide a good basis for the implicit learning of sequential information, even though it is task-irrelevant. I think the potential of recurrent network architectures is yet unexhausted, leaving a wide field for future research work. Therefore, there is a long list of research problems that occur in the context of recurrent networks and sequence learning in general. However, as my thesis mainly discusses the SMRNN architecture, I would like to focus on specific problems that arise from a segmented-memory architecture:

Selection of the optimal sequence length on the segment level of the SMRNN: If the optimal segment length is task dependent, there should be a way to determine it from the data. So the question arises: is it possible to learn this parameter from the data? For symbolic sequences like texts, grammar learning techniques (Nevill-Manning & Witten, 1997; Chaudhari & Wang, 2009) may be applicable to infer grammar rules, which in turn give evidence for a reasonable selection of the segment length.

For multi-variant sequences, that is, sequences of real-valued vectors without a symbolic representation, other techniques are required. In the data mining community several high level representations of time series have been proposed, for instance wavelets (Chan & Fu, 1999), symbolic mappings (Agrawal et al., 1995; Das et al., 1998; Perng et al., 2000) and piecewise linear representations (Hunter & McIntosh, 1999; Wang & Wang, 2000). Such representations may also be helpful in the selection of the appropriate segment length.

Multiple hierarchical levels: Especially grammar learning techniques yield a hierarchical structure of the sequences. For instance, a text (sequence of characters) has several levels of representation like characters, words, paragraphs, etc. Therefore, it is desirable to represent this knowledge about the structure in the classifier. Transferred to the idea of SMRNNs one could extend the network architecture to multiple levels of segments, which correspond to the assumed levels of the hierarchy. Every additional level could be realised by an additional SRN on top of the SRNs of the underlying levels.

Sequence prediction: In this thesis SMRNN were applied to a common benchmark problem in sequence classification. It remains for future work to apply SMRNN to the problem of sequence prediction. The available training algorithms allow a direct application. A prediction should be made on the basis of a sequence of preceding elements. At that point one has to decide how long the "sequence of preceding elements" should be. It

could be of fixed length or dynamically grow with the length of the training material. Of course, the selection of an appropriate length is task dependent. However, instead of guessing this parameter there might be a more general rule of thumb for a useful initial approximation.

Network pre-training: The possibilities of unsupervised pre-training (cf. Section 6.2) for SMRNN should be further explored. We saw that during pre-training as auto-encoder no temporal dependency between the inputs was learnt. To improve the learning of the temporal nature of the sequence the pre-training as predictors was proposed, but turned out to be unfavourable. Another idea to improve sequence learning could be a mixture of a pre-training as auto-encoder and predictor. For this, the forward connections of the SRNs could be trained as auto-encoders to learn a direct representation of the input data in the hidden layer. Thereafter the SRNs are trained as predictor to learn a temporal relation between the network inputs *only* in the recurrent connections, while the forward weights are kept fix. Such procedure might improve learning of temporal dependencies between inputs during pre-training, which in turn would support the solution of the actual supervised learning task. This is especially important if we consider sequence prediction.

Application to real world problems: As stated above, eBPTT training, together with the pre-training procedure, enables us to train relatively large SMRNNs ($n_x, n_y > 100$) that are still able to learn long-term dependencies. Yet, it remains to be proven that this approach is applicable to real world problems. The UCI Machine Learning Repository (Frank & Asuncion, 2010) provides a variety of sequential and time series data for classification and regression tasks. It is a good starting point to study the assets and drawbacks of SMRNNs, because the classification/regression results for some of the datasets are already published. Therefore, those datasets may constitute a useful benchmark.

List of Acronyms

BPTT Backpropagation Through Time.

eBPTT extended Backpropagation Through Time.

eRTRL extended Real-Time Recurrent Learning.

FFN Feed-Forward Network.

HMM Hidden Markov Model.

RL Reinforcement Learning.

RTRL Real-Time Recurrent Learning.

SMRNN Segmented-Memory Recurrent Neural Network.

SRN Simple Recurrent Network.

Bibliography

Abou-Nasr, M. A. (2010). Terrain identification in grayscale images with recurrent neural networks. In *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–5). 72

Ackley, D. H., & Littman, M. L. (1990). Generalization and scaling in reinforcement learning. In D. S. Touretzky (Ed.) *Advances in neural information processing systems 2*, (pp. 550–557). San Francisco, USA: Morgan Kaufmann Publishers Inc. 44

Agrawal, R., Lin, K.-I., Sawhney, H. S., & Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21th International Conference on Very Large Data Bases*, (pp. 490–501). San Francisco, USA: Morgan Kaufmann Publishers Inc. 115

Alonso, D., Fuentes, L. J., & Hommel, B. (2006). Unconscious symmetrical inferences: A role of consciousness in event integration. *Consciousness and Cognition*, *15*(2), 386–396. 15

Anderson, J. (1995). *Learning and Memory: An Integrated Approach*. New York: Wiley. 27

Bakker, P. B. (2004). *The State of Mind: Reinforcement Learning with Recurrent Neural Networks*. Leiden, the Netherlands: Universiteit Leiden. 28

Barbounis, T., Theocharis, J., Alexiadis, M., & Dokopoulos, P. (2006). Long-term wind speed and power forecasting using local recurrent neural network models. *IEEE Transactions on Energy Conversion*, *21*(1), 273–284. 71

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. In O. Shisha (Ed.) *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*, (pp. 1–8). Los Angeles, USA: Academic Press. 28

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127. 95, 96, 97, 98

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.) *Advances in Neural Information Processing Systems 19*, (pp. 153–160). Cambridge, USA: MIT Press. 96, 98

Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, & J. Weston (Eds.) *Large Scale Kernel Machines*, (pp. 1–41). MIT Press. 96

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. 28, 72, 73, 74, 91

Berry, D. C. (1991). The role of action in implicit learning. *Quarterly Journal of Experimental Psychology: Section A*, *43*(4), 881–906. 10

Berry, D. C. (1994). Implicit learning: Twenty-five years on. A tutorial. In C. Umilta, & M. Moscovitch (Eds.) *Consciousness and Unconscious Information Processing: Attention and Performance 15*. Cambridge, US: MIT Press. 8

Berry, D. C. (Ed.) (1997). *How implicit is implicit learning?*. Oxford, UK]: Oxford University Press. 7, 8, 10, 11

Berry, D. C., & Broadbent, D. E. (1984). On the relationship between task performance and associated verbalizable knowledge. *Quarterly Journal of Experimental Psychology: Section A*, *36*(2), 209–231. 10, 12

Berry, D. C., & Broadbent, D. E. (1987). The combination of explicit and implicit learning processes in task control. *Psychological Research*, *49*(1), 7–15. 10

Berry, D. C., & Broadbent, D. E. (1988). Interactive tasks and the implicit-explicit distinction. *British Journal of Psychology*, *79*(2), 251–272. 12

Berry, D. C., & Dienes, Z. (1991). The relationship between implicit memory and implicit learning. *British Journal of Psychology*, *82*(3), 359–373. 14

Berry, D. C., & Dienes, Z. (Eds.) (1993). *Implicit Learning: Theoretical and Empirical Issues*. New York, USA: Lawrence Erlbaum Associates. 8, 15

Bhattacharya, A., Parlos, A., & Atiya, A. (2003). Prediction of mpeg-coded video source traffic using recurrent neural networks. *IEEE Transactions on Signal Processing*, *51*(8), 2177–2190. 72

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York, USA: Springer. 24

Blumenfeld, B., Preminger, S., Sagi, D., & Tsodyks, M. (2006). Dynamics of memory representations in networks with novelty-facilitated synaptic plasticity. *Neuron*, *52*(2), 383–394. 17

Brooks, L. R. (1978). *Non-analytic concept formation and memory for instances*, vol. 3, (pp. 169–211). Hillsdale, USA: Erlbaum. 10, 13

Brooks, L. R., & Vokey, J. R. (1991). Abstract analogies and abstracted grammars: comments on Reber (1989) and Mathews et al. (1989). *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *120*(3), 316–323. 13

Bruner, J. S., Goodnow, J. J., Austin, G. A., & Brown, R. W. (1967). *A study of thinking*. New York, USA: Wiley. 25

Chan, K.-P. C., & Fu, A. W.-C. (1999). Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, (pp. 126–133). Washington, DC, USA: IEEE Computer Society. 115

Chaudhari, N. S., & Wang, X. (2009). Language structure using fuzzy similarity. *IEEE Transactions on Fuzzy Systems*, *17*(5), 1011–1024. 115

Chauvin, Y. (1990). Dynamic behavior of constrained back propagation networks. In D. S. Touretzky (Ed.) *Advances in neural information processing systems 2*, (pp. 642–649). San Francisco, USA: Morgan Kaufmann Publishers Inc. 101

Chen, J., & Chaudhari, N. S. (2004). Capturing long-term dependencies for protein secondary structure prediction. In F.-L. Yin, J. Wang, & C. Guo (Eds.) *Advances in Neural Networks - ISNN 2004*, vol. 3174 of *Lecture Notes in Computer Science*, (pp.

494–500). Springer Berlin Heidelberg. 72, 74, 81, 91

Chen, J., & Chaudhari, N. S. (2009). Segmented-memory recurrent neural networks. *IEEE Transactions Neural Networks*, *20*(8), 1267–1280. 72, 73, 74, 77, 78, 101, 111

Cleeremans, A. (1993). *Mechanisms of implicit learning: Connectionist models of sequence processing*. Cambridge, USA: MIT Press. 1, 7, 8, 10, 11, 12, 19, 20, 21, 22, 43

Cleeremans, A., Destrebecqz, A., & Boyer, M. (1998). Implicit learning: news from the front. *Trends in Cognitive Sciences*, *2*(10), 406–416. 7, 8, 13, 14, 20

Cleeremans, A., & McClelland, J. L. (1991). Learning the structure of event sequences. *Journal of Experimental Psychology: General*, *120*(3), 235–253. 12, 14

Conway, J. (1990). *A Course in Functional Analysis (Graduate texts in mathematics)*. New York, USA: Springer. 68

Cooper, R. P. (2002). *Modelling high-level cognitive processes*. Mahwah, USA: Lawrence Erlbaum. 2

Custers, R., & Aarts, H. (2011). Learning of predictive relations between events depends on attention, not on awareness. *Consciousness and Cognition*, *20*(2), 368–378. 15

Das, G., Lin, K.-I., Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, (pp. 16–22). Palo Alto, USA: AAAI Press. 115

Dayan, P., & Niv, Y. (2008). Reinforcement learning: The good, the bad and the ugly. *Current Opinion in Neurobiology*, *18*(2), 185–196. 53

Dickinson, A. (1980). *Contemporary animal learning theory*. Cambridge, Uk: Cambridge University Press. 16

Dienes, Z. (1992). Connectionist and memory-array models of artificial grammar learning. *Cognitive Science*, *16*(1), 41–79. 14, 20

Dienes, Z., Broadbent, D., & Berry, D. (1991). Implicit and explicit knowledge bases in artificial grammar learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *17*(5), 875–887. 13

Dienes, Z., & Fahey, R. (1995). Role of specific instances in controlling a dynamic system. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *21*(4), 848–862. 10, 20

Dijksterhuis, A., & Aarts, H. (2010). Goals, Attention, and (Un)Consciousness. *Annual Review of Psychology*, *61*(1), 467–490. 14, 15

Dulany, D. E., Carlson, R. A., & Dewey, G. I. (1984). A case of syntactical learning and judgment: How conscious and how abstract? *Journal of Experimental Psychology: General*, *113*(4), 541–555. 10, 13

Ebbinghaus, H., Ruger, H., & Bussenius, C. (1913). *Memory: a contribution to experimental psychology*. Educational reprints. New York, USA: Teachers College, Columbia University. 2

Eitam, B., Hassin, R. R., & Schul, Y. (2008). Nonconscious goal pursuit in novel environments: the case of implicit learning. *Psychological Science*, *19*(3), 261–267. 14

El Hihi, S., & Bengio, Y. (1995). Hierarchical recurrent neural networks for long-term dependencies. In D. S. Touretzky, M. Mozer, & M. Hasselmo (Eds.) *Advances in Neural Information Processing Systems 8*, (pp. 493–499). Cambridge, USA: MIT Press. 72

Elman, J. L. (1988). Finding structure in time. Tech. Rep. 8801, San Diego, CA: University of California, Center for Research in Language. 21

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. 21, 22, 44, 58

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, *7*, 195–225. 56

Erhan, D., Pierre-Antoine, M., Bengio, Y., Bengio, S., & Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. *Journal of Machine Learning Research - Proceedings Track*, *5*, 153–160. 96, 97, 98

Fanselow, M. S. (1986). Associative vs. topographical accounts of the immediate-shock freezing deficit in rats: Implications for the response selection rules governing species specific defense reactions. *Learning and Motivation*, *17*(1), 16–39. 16

Fanty, M. (1986). Context-free parsing with connectionist networks. Tech. Rep. 174, Rochester, NY: University of Rochester, Computer Science Department. 21

Finnoff, W., Hergert, F., & Zimmermann, H. G. (1993). Improving model selection by nonconvergent methods. *Neural Networks*, *6*(6), 771–783. 101

Forney, E., & Anderson, C. (2011). Classification of eeg during imagined mental tasks by forecasting with elman recurrent neural networks. In *Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN)*, (pp. 2749–2755). New York, USA: IEEE Press. 72

Frank, A., & Asuncion, A. (2010). UCI machine learning repository. URL `http://archive.ics.uci.edu/ml` 116

Frensch, P. A., & Stadler, M. A. (1998). *One concept, multiple meanings: On how to define the concept of implicit learning*, (pp. 47–104). London, UK: Sage Publications. 8

Freund, Y., & Haussler, D. (1994). Unsupervised learning of distributions on binary vectors using two layer networks. Tech. rep., University of California at Santa Cruz, Santa Cruz, USA. 96

Frick, R. W. (1989). Explanations of grouping in immediate ordered recall. *Memory and Cognition*, *17*(5), 551–562. 75

Gibson, F. P., Fichman, M., & Plaut, D. C. (1997). Learning in dynamic decision tasks: Computational model and empirical evidence. *Organizational Behavior and Human Decision Processes*, *71*(1), 1–35. 20

Giles, C. L., Chen, D., Miller, C. B., Chen, H. H., Sun, G. Z., & Lee, Y. C. (1992). Second-order recurrent neural networks for grammatical inference. In *Proceedings of the International Joint Conference on Neural Networks*, (pp. 273–281). New York, USA: IEEE Press. 72

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, *9*, 249–256. 101

Glüge, S., Böck, R., & Wendemuth, A. (2010). Implicit sequence learning - a case study with a 4-2-4 encoder simple recurrent network. In J. Filipe, & J. Kacprzyk (Eds.) *Proceedings of the International Conference on Fuzzy Computation and International Conference on Neural Computation*, (pp. 279–288). SciTePress. 56

Glüge, S., Böck, R., & Wendemuth, A. (2011a). Segmented-memory recurrent neu-

ral networks versus hidden markov models in emotion recognition from speech. In K. Madani, J. Kacprzyk, & J. Filipe (Eds.) *Proceedings of International Conference on Neural Computation, Theory and Application*, (pp. 308–315). SciTePress. 111

Glüge, S., Böck, R., & Wendemuth, A. (2012). Extension of backpropagation through time for segmented-memory recurrent neural networks. In A. C. Rosa, A. D. Correia, K. Madani, J. Filipe, & J. Kacprzyk (Eds.) *Proceedings of the 4th International Joint Conference on Computational Intelligence*, (pp. 451–456). SciTePress. 73

Glüge, S., Böck, R., & Wendemuth, A. (2013). Auto-encoder pre-training of segmented-memory recurrent neural networks. In M. Verleysen (Ed.) *Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. Louvain-la-Neuve, Belgium: Ciaco. 96

Glüge, S., Hamid, O. H., Braun, J., & Wendemuth, A. (2011b). A markov model of conditional associative learning in a cognitive behavioural scenario. In J. M. Ferrández, J. R. Álvarez Sánchez, F. Paz, & F. Toledo (Eds.) *Foundations on Natural and Artificial Computation*, vol. 6686 of *Lecture Notes in Computer Science*, (pp. 10–19). Berlin / Heidelberg, Germany: Springer. 31

Glüge, S., Hamid, O. H., & Wendemuth, A. (2010). A simple recurrent network for implicit learning of temporal sequences. *Cognitive Computation*, *2*(4), 265–271. 31

Grüning, A. (2007). Elman backpropagation as reinforcement for simple recurrent networks. *Neural Computation*, *19*(11), 3108–3131. 44

Halford, G. S., Baker, R., McCredden, J. E., & Bain, J. D. (2005). How many variables can humans process? *Psychological Science*, *16*(1), 70–76. 14

Hall, G. (1994). *Pavlovian Conditioning: Laws of Association*, (pp. 15–44). San Diego, USA: Academic Press. 16

Hamid, O. H. (2011). *On the Role of Temporal Context in Human Reinforcement Learning*. Ph.D. thesis, Otto von Guericke University Magdeburg, Faculty for Natural Science, Magdeburg, Germany. 7, 16, 17

Hamid, O. H., Wendemuth, A., & Braun, J. (2010). Temporal context and conditional associative learning. *BMC Neuroscience*, *11*(45), 1–15. 16, 17, 18, 31, 41, 48, 53

Hanson, S. J., & Kegl, J. (1987). PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, (pp. 106–119). Hillsdale, USA: Erlbaum. 21

Hasher, L., & Zacks, R. T. (1979). Automatic and effortful processes in memory. *Journal of Experimental Psychology: General*, *108*(3), 356–388. 14

Hasher, L., & Zacks, R. T. (1984). Automatic processing of fundamental information. *American Psychologist*, *39*(12), 1372–1388. 14, 17

Heskes, T. M., & Kappen, B. (1991). Learning processes in neural networks. *Physical Review A*, *44*(4), 2718–2726. 55

Heskes, T. M., & Kappen, B. (1993). On-line learning processes in artificial neural networks. In J. Taylor (Ed.) *Mathematical approaches to neural networks*, vol. 51 of *North-Holland Mathematical Library*, (pp. 199 – 233). Elsevier. 55

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554. 96

Hintzman, D. L. (1969). Recognition time: Effects of recency, frequency and the spacing of repetitions. *Journal of Experimental Psychology*, *79*(1), 192–194. 14

Hitch, G. J., Burgess, N., Towse, J. N., & Culpin, V. (1996). Temporal grouping effects in immediate recall: A working memory analysis. *Quarterly Journal of Experimental Psychology Section A: Human Experimental Psychology*, *49*(1), 116–139. 75

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen*. Master's thesis, Institut für Informatik, Technische Universität München. 73

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*, (pp. 237–243). New York, USA: IEEE Press. 73

Hochreiter, S., & Schmidhuber, J. (1997a). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. 72

Hochreiter, S., & Schmidhuber, J. (1997b). LSTM can solve hard long time lag problems. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.) *Proceedings of the Conference on Advanced Neural Information Processing Systems 9*, (pp. 473–479). Cambridge, USA: MIT Press. 111

Hunter, J., & McIntosh, N. (1999). Knowledge-based event detection in complex time series data. In W. Horn, Y. Shahar, G. Lindberg, S. Andreassen, & J. Wyatt (Eds.) *Artificial Intelligence in Medicine*, vol. 1620 of *Lecture Notes in Computer Science*, (pp. 271–280). Berlin / Heidelberg, Germany: Springer. 115

Jaeger, H. (2001). The ¨echo state¨ approach to analysing and training recurrent neural networks. Tech. Rep. GMD report 148, German national research institute for computer science. 72

Jaeger, H. (2002). Short term memory in echo state networks. Tech. Rep. GMD report 152, German national research institute for computer science. 72

Jimenez, L., & Mendez, C. (1999). Which attention is needed for implicit sequence learning? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *25*(1), 236–259. 14

Jordan, M. I. (1986). Serial order: A parallel, distributed processing approach. Tech. Rep. 8604, San Diego, CA: University of California, Center for Research in Language. 21, 22

Kahneman, D., & Tversky, A. (1982). *Judgment under uncertainty: Heuristics and biases*. New York, USA: Cambridge University Press. 8

Kiernan, M. J., Westbrook, R. F., & Cranney, J. (1995). Immediate shock, passive avoidance, and potentiated startle: Implications for the unconditioned response to shock. *Animal Learning and Behavior*, *23*(1), 22–30. 16

Ku, K., Mak, M. W., & Siu, W. C. (1999). Adding learning to cellular genetic algorithms for training recurrent neural networks. *IEEE Transactions on Neural Networks*, *10*(2), 239–252. 72

Kushner, M., Cleeremans, A., & Reber, A. S. (1991). Implicit detection of event interdependencies and a PDP model of the process. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*. Mahwah, USA: Lawrence Erlbaum Associates. 11, 19

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies

for training deep neural networks. *Journal of Machine Learning Research*, *10*, 1–40. 97

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani (Ed.) *Proceedings of the 24th International Conference on Machine Learning*, (pp. 473–480). New York, USA: ACM. 97, 98

Lee, S.-W., & Song, H.-H. (1997). A new recurrent neural-network architecture for visual pattern recognition. *IEEE Transactions on Neural Networks*, *8*(2), 331–340. 72

Lewicki, P., Czyzewska, M., & Hoffman, H. (1987). Unconscious acquisition of complex procedural knowledge. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *13*(4), 523–530. 13

Lewicki, P., Hill, T., & Bizot, E. (1988). Acquisition of procedural knowledge about a pattern of stimuli that cannot be articulated. *Cognitive Psychology*, *20*(1), 24–37. 12, 13

Lewicki, P., Hill, T., & Czyzewska, M. (1992). Nonconscious acquisition of information. *American Psychologist*, *47*(6), 796–801. 14

Lin, T., Horne, B., Tino, P., & Giles, C. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, *7*(6), 1329–1338. 72

Lin, T., Horne, B. G., & Giles, C. L. (1998). How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies. *Neural Networks*, *11*(5), 861–868. 72

Ma, S., & Ji, C. (1998). Fast training of recurrent networks based on the EM algorithm. *IEEE Transactions on Neural Networks*, *9*(1), 11–26. 72

Mackintosh, N. J. (1983). *Conditioning and Associative Learning*. Oxford, UK: Oxford University Press. 16

Mazzoni, P., Andersen, R. A., & Jordan, M. I. (1991). A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, *88*(10), 4433–4437. 44

Mccallum, A. K. (1996). Learning to use selective attention and short-term memory in sequential tasks. In P. Maes, S. W. Wilson, & M. J. Mataric (Eds.) *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, (pp. 315–324). Cambridge, USA: MIT Press. 27

Millward, R. B., & Reber, A. S. (1968). Event-recall in probability learning. *Journal of Verbal Learning and Verbal Behavior*, *7*(6), 980–989. 11

Millward, R. B., & Reber, A. S. (1972). Probability learning: Contingent-event schedules with lags. *American Journal of Psychology*, *85*(1), 81–98. 11

Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Upper Saddle River, USA: Prentice-Hall. 22

Miyashita, Y. (1988). Neuronal correlate of visual associative long-term memory in the primate temporal cortex. *Nature*, *335*(6193), 817–820. 16

Miyashita, Y., & Chang, H. S. (1988). Neuronal correlate of pictorial short-term memory in the primate temporal cortex. *Nature*, *331*(6151), 68–70. 17

Mobahi, H., Collobert, R., & Weston, J. (2009). Deep learning from temporal coherence

in video. In *Proceedings of the 26th International Conference on Machine Learning*, (pp. 737–744). New York, USA: ACM. 96

Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, *6*(4), 525–533. 101

Mozer, M. C., & Bachrach, J. (1991). Slug: A connectionist architecture for inferring the structure of finite-state environments. *Machine Learning*, *7*(2–3), 139–160. 23

Nevill-Manning, C. G., & Witten, I. H. (1997). Identifying hierarchical structure in sequences: a linear-time algorithm. *Journal of Artificial Intelligence Research*, *7*(1), 67–82. 27, 115

Newell, A. (1972). *Human problem solving*. Upper Saddle River, USA: Prentice-Hall. 12

Nishide, S., Okuno, H., Ogata, T., & Tani, J. (2011). Handwriting prediction based character recognition using recurrent neural network. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, (pp. 2549–2554). New York, USA: IEEE Press. 72

Nissen, M. J., & Bullemer, P. (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology*, *19*(1), 1–32. 11, 14

Park, D.-C. (2011). Sunspot series prediction using adaptively trained multiscale-bilinear recurrent neural network. In *Proceedings of the 9th IEEE/ACS International Conference on Computer Systems and Applications*, (pp. 135–139). Washington, DC, USA: IEEE Computer Society. 72

Park, H., Dibazar, A., & Berger, T. (2011). Discrete synapse recurrent neural network with time-varying delays for nonlinear system modeling and its application on seismic signal classification. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, (pp. 2374–2381). New York, USA: IEEE Press. 72

Perng, C.-S., Wang, H., Zhang, S., & Parker, D. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. In *Proceedings of the 16th International Conference on Data Engineering*, (pp. 33–42). Washington, DC, USA: IEEE Computer Society. 115

Perruchet, P. (2008). Implicit learning. In J. H. Byrne (Ed.) *Learning and Memory: A Comprehensive Reference*, (pp. 597–621). Oxford, UK: Academic Press. 7

Perruchet, P., & Pacteau, C. (1990). Synthetic grammar learning: Implicit rule abstraction or explicit fragmentary knowledge? *Journal of Experimental Psychology: General*, *119*(3), 264–275. 13

Prechelt, L. (1998). Early stopping - but when? In G. Orr, & K.-R. Müller (Eds.) *Neural Networks: Tricks of the Trade*, vol. 1524 of *Lecture Notes in Computer Science*, (pp. 55–69). Berlin / Heidelberg, Germany: Springer. 101

Preminger, S., Blumenfeld, B., Sagi, D., & Tsodyks, M. (2009). Mapping dynamic memories of gradually changing objects. *Proceedings of the National Academy of Sciences of the United States of America*, *106*(13), 5371–5376. 17

Ranzato, M. A., Poultney, C. S., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.) *Advances in Neural Information Processing Systems 19*, (pp. 1137–1144). Cambridge, USA: MIT Press. 96, 98

Reber, A. S. (1965). *Implicit learning of artificial grammars*. Master's thesis, Brown

University, Providence, Rhode Island, USA. 9

Reber, A. S. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, *6*(6), 855–863. 7, 9, 12, 13, 19

Reber, A. S. (1969). Transfer of syntactic structure in synthetic languages. *Journal of Experimental Psychology*, *81*(1), 115–119. 10

Reber, A. S. (1976). Implicit learning of synthetic languages: The role of instructional set. *Journal of Experimental Psychology Human Learning and Memory*, *2*(1), 88–94. 10

Reber, A. S. (1989). Implicit learning and tacit knowledge. *Journal of Experimental Psychology: General*, *118*, 219–235. 13

Reber, A. S. (1993). *Implicit learning and tacit knowledge: An essay on the cognitive unconscious*. Oxford, UK: Oxford University Press. 7, 8, 13, 14

Reber, A. S., Kassin, S. M., Lewis, S., & Cantor, G. (1980). On the relationship between implicit and explicit modes in the learning of a complex rule structure. *Journal of Experimental Psychology: Human Learning and Memory*, *6*(5), 492–502. 12

Reber, A. S., & Lewis, S. (1977). Toward a theory of implicit learning: The analysis of the form and structure of a body of tacit knowledge. *Cognition*, *5*, 333–361. 13

Reed, J., & Johnson, P. (1994). Assessing implicit learning with indirect tests: Determining what is learned about sequence structure. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *20*(3), 585–594. 19

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, (6088), 533–536. 44

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986b). Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chap. Learning internal representations by error propagation, (pp. 318–362). Cambridge, USA: MIT Press. 56

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, USA: MIT Press. 21

Ryan, J. (1969). Temporal grouping, rehearsal and short-term memory. *Quarterly Journal of Experimental Psychology*, *21*(2), 148–155. 75

Sakai, K., & Miyashita, Y. (1991). Neural organization for the long-term memory of paired associates. *Nature*, *354*(6349), 152–155. 17

Sakai, K., Naya, Y., & Miyashita, Y. (1994). Neuronal tuning and associative mechanisms in form representation. *Learning and Memory*, *1*(2), 83–105. 17

Sanderson, P. M. (1989). Verbalizable knowledge and skilled task performance: Association, disssociation and mental models. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *15*(4), 729–747. 13

Seibel, R. (1963). Discrimination reaction time for 1,023 alternative task. *Journal of Experimental Psychology*, *66*(3), 215–226. 12

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Complex Systems*, *1*(1), 145–168. 21

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1989). Advances in neural information processing systems 1. chap. Learning sequential structure in simple recurrent networks, (pp. 643–652). San Francisco, USA: Morgan Kaufmann Publishers Inc. 21, 29

Servan-Schreiber, D., Cleeremans, A., & Mcclelland, J. L. (1991). Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, *7*, 161–193. 23

Servan-Schreiber, E., & Anderson, J. R. (1990). Learning artificial grammars with competitive chunking. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *16*(4), 592–608. 10, 20

Severin, F. T., & Rigby, M. K. (1963). Influence of digit grouping on memory for telephone numbers. *Journal of Applied Psychology*, *47*(2), 117–119. 75

Shanks, D. R. (2005). *Handbook of Cognition*, chap. Implicit Learning, (pp. 202–220). London, UK: Sage Publications. 7

Shanks, D. R., & St. John, M. F. (1994). Characteristics of dissociable human learning systems. *Behavioral and Brain Sciences*, *17*(3), 367–447. 13

Sloman, S. A. (1996). The empirical case for two systems of reasoning. *Psychological Bulletin*, *119*, 3–22. 15

Squartini, S., Hussain, A., & Piazza, F. (2003a). Attempting to reduce the vanishing gradient effect through a novel recurrent multiscale architecture. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, (pp. 2819–2824). New York, USA: IEEE Press. 72

Squartini, S., Hussain, A., & Piazza, F. (2003b). A recurrent multiscale architecture for long-term memory prediction task. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 789–792). New York, USA: IEEE Press. 72

Stadler, M., & Frensch, P. (1998). *Handbook of implicit learning*. London, UK: Sage Publications. 7

Stanley, W. B., Mathews, R. C., Buss, R. R., & Kotler-Cope, S. (1989). Insight without awareness: On the interaction of verbalization, instruction and practice in a simulated process control task. *Quarterly Journal of Experimental Psychology: Section A*, *41*(3), 553–577. 10

Sun, R., & Giles, C. L. (2001). Sequence learning: From recognition and prediction to sequential decision making. *IEEE Intelligent Systems*, *16*, 67–70. 27, 28

Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science*, *25*(2), 203–244. 27

Sun, R., & Sessions, C. (2000). Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors. *IEEE Transactions on Systems Man and Cybernetics*, *30*(3), 403–418. 27

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, USA: MIT Press. 25, 28, 47, 53

Tian, Z., & Zuo, M. (2010). Health condition prediction of gears using a recurrent neural network approach. *IEEE Transactions on Reliability*, *59*(4), 700–705. 72

Tino, P., Schittenkopf, C., & Dorffner, G. (2001). Financial volatility trading using recurrent neural networks. *IEEE Transactions on Neural Networks*, *12*(4), 865–874. 72

Underwood, G. (1996). *Implicit Cognition*. Oxford, UK: Oxford University Press. 7, 8, 13, 14

Utgoff, P. E., & Stracuzzi, D. J. (2002). Many-layered learning. *Neural Computation*, *14*(10), 2497–2529. 96

Varoglu, E., & Hacioglu, K. (1999). Speech prediction using recurrent neural networks. *Electronics Letters*, *35*(16), 1353–1355. 71

Čerňanský, M., Makula, M., & Beňušková, Ľ. (2007). Organization of the state space of a simple recurrent network before and after training on recursive linguistic structures. *Neural Networks*, *20*(2), 236–244. 56

Verschure, P., & Althaus, P. (2003). A real-world rational agent: Unifying old and new ai. *Cognitive Science*, *27*(4), 561–590. 72

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, (pp. 1096–1103). New York, USA: ACM. 98

Vinyals, O., Ravuri, S., & Povey, D. (2012). Revisiting recurrent neural networks for robust ASR. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processesing (ICASSP)*, (pp. 4085–4088). New York, USA: IEEE Press. 110

Vokey, J. R., & Brooks, L. R. (1992). Salience of item knowledge in learning artificial grammars. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *18*(2), 328–344. 13

Šter, B. (2003). Latched recurrent neural network. *Elektrotehniski Vestnik/Electrotechnical Review*, *70*(1–2), 46–51. 72

Wallis, G., & Bülthoff, H. H. (2001). Effects of temporal association on recognition memory. *Proceedings of the National Academy of Sciences of the United States of America*, *98*(8), 4800–4804. 17

Wang, C., & Wang, X. S. (2000). Supporting content-based searches on time series via approximation. In *Proceedings of the 12th International Conference on Scientific and Statistical Database Management*, (pp. 69–81). Washington, DC, USA: IEEE Computer Society. 115

Wang, D. (2001). Anticipation model for sequential learning of complex sequences. In R. Sun, & C. Giles (Eds.) *Sequence Learning*, vol. 1828 of *Lecture Notes in Computer Science*, (pp. 53–79). Berlin / Heidelberg, Germany: Springer. 72

Wattenmaker, W. D. (1993). Incidental concept learning, feature frequency, and correlated properties. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *19*(1), 203–222. 14, 17

Wendemuth, A. (2007). Dynamics of temporal difference learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, (pp. 1107–1112). San Francisco, USA: Morgan Kaufmann Publishers Inc. 28

Werbos, P. (1988). Backpropagation: past and future. In *Proceedings of the IEEE International Conference on Neural Networks*, (pp. 343–353). Long Beach, USA: IEEE Press. 101

Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560. 73, 85

Werfel, J., Xie, X., & Seung, H. S. (2005). Learning curves for stochastic gradient descent

in linear feedforward networks. *Neural Computation*, *17*(12), 2699–2718. 44

Weston, J., Ratle, F., & Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine learning*, (pp. 1168–1175). 96

Whittlesea, B. W., & Dorken, M. D. (1993). Incidentally, things in general are particularly determined: An episodic-processing account of implicit learning. *Journal of Experimental Psychology: General*, *122*(2), 227–248. 20

Whittlesea, B. W., & Wright, R. L. (1997). Implicit (and explicit) learning: acting adaptively without knowing the consequences. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *23*(1), 181–200. 20

Wickelgren, W. A. (1967). Rehearsal grouping and hierarchical organization of serial position cues in short-term memory. *The Quarterly Journal of Experimental Psychology*, *19*(2), 97–102. 75

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280. 73, 78, 79

Williams, R. J., & Zipser, D. (1995). *Gradient-based learning algorithms for recurrent networks and their computational complexity*, (pp. 433–486). Hillsdale, NJ, USA: L. Erlbaum Associates Inc. 78

Willingham, D. B., Nissen, M. J., & Bullemer, P. (1989). On the development of procedural knowledge. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *15*(6), 1047–1060. 11

Yakovlev, V., Fusi, S., Berman, E., & Zohary, E. (1998). Inter-trial neuronal activity in inferior temporal cortex: a putative vehicle to generate long-term visual associations. *Nature Neuroscience*, *1*(4), 310–317. 17

Zacks, R. T., Hasher, L., & Sanft, H. (1982). Automatic encoding of event frequency: Further findings. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *8*(2), 106–116. 14

List of Authored Publications

Articles in International Journals

1 **A simple recurrent network for implicit learning of temporal sequences** (S. Glüge, O.H. Hamid, A. Wendemuth), *Cognitive Computation*, volume 2, pp. 265–271, 2010.

2 **On network representations of antennas inside resonating environments** (F. Gronwald, S. Glüge, J. Nitsch), *Advances in Radio Science*, volume 5, pp. 157–162, 2007.

Contributions in Book Series and Proceedings of International Conferences

3 **Auto-Encoder Pre-Training of Segmented-Memory Recurrent Neural Networks** (S. Glüge, R. Böck, A. Wendemuth), *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2013)*, Louvain-la-Neuve: Ciaco, pp. 29–34, 2013.

4 **Extension of backpropagation through time for segmented-memory recurrent neural networks** (S. Glüge, R. Böck, A. Wendemuth), *Proceedings of the 4th International Joint Conference on Computational Intelligence*, SciTePress, pp. 451–456, 2012.

5 **Intraindividual and interindividual multimodal emotion analyses in human-machine-interaction** (R. Böck, S. Glüge, A. Wendemuth, K. Limbrecht, S. Walter, D. Hrabal, H.C. Traue), *IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, Piscataway, NY: IEEE, pp. 59–64, 2012.

6 **Combining mimic and prosodic analyses for user disposition classification** (R. Böck, K. Limbrecht, I. Siegert, S. Glüge, S. Walter, A. Wendemuth), *Proceedings der 23. Konferenz Elektronische Sprachsignalverarbeitung*, Dresden: TUDpress, pp. 29–31, 2012

7 **Describing human emotions through mathematical modelling** (K. Hartmann, I. Siegert, S. Glüge, A. Wendemuth, M. Kotzyba, B. Deml), *MATHMOD*, ARGE Simulation News, Vienna University of Technology, 6 p., 2012.

8 **Segmented-memory recurrent neural networks versus hidden markov models in emotion recognition from speech** (S. Glüge, R. Böck, A. Wendemuth), *Proceedings of the International Conference on Neural Computation Theory and Applications*, SciTePress, pp. 308–315, 2011.

9 **A markov model of conditional associative learning in a cognitive behavioural scenario** (S. Glüge, O.H. Hamid, J. Braun, A. Wendemuth), *Foundations on Natural and Artificial Computation*, volume 6686 of *Lecture Notes in Computer Science*, Berlin / Heidelberg: Springer, pp. 10–19, 2011.

10 **IMPLICIT SEQUENCE LEARNING - A case study with a 4-2-4 encoder simple recurrent network** (S. Glüge, R. Böck, A. Wendemuth), *Proceedings of the International Conference on Neural Computation*, SciTePress, pp. 279–288, 2010.

11 **Neurobiologically inspired companion systems** (A. Wendemuth, S. Glüge, O.H. Hamid), *Proceedings of Research Workshop on "Emotion-, Speech-, and Face Recognition with advanced classifiers"*, Magdeburg: University, pp. 57–66, 2008

Contributions in Workshops

12 **Emotion detection by event evaluation using fuzzy sets as appraisal variables.** (M. Kotzyba, B. Deml, H. Neumann, S. Glüge, K. Hartmann, I. Siegert, A. Wendemuth, H. Traue, S.Walter), *Proceedings of the 11th International Conference on Cognitive Modeling*, Universitätsverlag der TU Berlin, pp. 123–124, 2012.

13 **Cognitive science and information** (S. Glüge, A. Wendemuth), *International Conference on What makes Humans Human (WMHH2010)*, March, 26-27, 2010, Ulm, Germany

14 **Simulated group meetings-insights from sociology and engineering** (T. Grosser, V. Heine, S. Glüge, I. Siegert, J. Frommer, A. Wendemuth), *International Conference on What makes Humans Human*, March, 26-27, 2010, Ulm, Germany

15 **Implicit sequence learning by recurrent neural networks** (S. Glüge, A. Wendemuth), *Proceedings of Interdisciplinary College 2010*, March, 12-19, 2010, Günne, Germany

Publications to appear

16 **Solving Number Series with Simple Recurrent Networks** (S. Glüge, A. Wendemuth), *Proceedings of the International Work-Conference on the Interplay between Natural and Artificial Computation (IWINAC2013)*, Palma de Mallorca, 2013.

17 **Dempster-Shafer theory with smoothness** (R. Böck, S. Glüge, A. Wendemuth), *Proceedings of the 3rd International Symposium on Integrated Uncertainty in Knowledge Modelling and Decision Making (IUKM2013)* Beijing, China, 2013.

18 **Modelling of emotional development within human-computer-interaction** (I. Siegert, K. Hartmann, S. Glüge, A. Wendemuth), *Proceedings des 2. interdisziplinären Workshops*, Duisburg, Germany, 2012.