



UvA-DARE (Digital Academic Repository)

Rules and associations : hidden Markov models and neural networks in the psychology of learning

Visser, I.

Publication date
2002

[Link to publication](#)

Citation for published version (APA):

Visser, I. (2002). *Rules and associations : hidden Markov models and neural networks in the psychology of learning*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

2 On the computational power and interpretation of subsymbolic processes

Abstract

In debates between connectionists and cognitivists it is often claimed that neural networks do not have the computational power to represent all cognitive functions of interest. In the last decade some mathematical results have appeared that characterize the computational powers of neural networks and the subsymbolic processes that they instantiate. There are three types of results. The first results show that neural networks are universal function approximators (Hornik et al., 1989). Although this result establishes neural networks as powerful computational devices, the type of neural network described in this result has only limited applicability in cognitive science. The second result shows that it is possible to construct a neural network that implements a universal Turing machine (Siegelmann and Sontag, 1995). For modeling purposes, this type of result is not very helpful, because the neural network so constructed shares many of its fundamental properties with the classical cognitive architecture. It therefore misses typical advantageous properties of a neural architecture such as noise tolerance and graceful degradation. The third type of result is concerned with learning formal languages by recurrent neural networks. This type of result is a promising line of research for cognitive science for several reasons. These networks perform like rule-based cognitive architectures and at the same time retain the advantages of a neural architecture. Analysis of these networks provides insight into the nature of subsymbolic processing. We discuss these results, and our own work in the analysis of neural network behavior and the internal representations that give rise to it. Our aims are to establish the computational power of neural networks, to provide insight into the nature of subsymbolic processing and, in so doing, clarify the relation between symbolic and subsymbolic processing.

2.1 Cognitive architectures

The importance of rule-like behavior, as observed for example in language use and learning, is virtually undisputed in psychology (see Shanon, 1993, for discussion of this point). With respect to language use, Chomsky's influential paper has made it clear that behaviorist and associationist models have insufficient resources to model linguistic competence (Chomsky, 1959b). Although Chomsky's paper is concerned only with linguistic behavior, or rather with competence in language use, his rule-based account is now used widely in many fields in psychology, such as

memory research, learning, perception. Fodor and Pylyshyn (1988) argued that the classical cognitive architecture is needed. The basis of this architecture is formed by “representational states that have combinatorial syntax” (Fodor and Pylyshyn, 1988, p. 1) in order to sustain rule-like behavior. Representations or representational states can be manipulated using syntactical rules which manipulate the syntax of an expression, and are independent of its content¹.

In the last 15 years, connectionist models have been used extensively in cognitive psychology as an alternative to the classical architecture (see e.g. Chauvin and Rummelhart, 1995; Quinlan, 1991; Arbib, 1995; Baddeley et al., 1999, for overviews of applications and algorithms). In the connectionist architecture, symbol manipulation is replaced by subsymbolic processing. Smolensky (1988) argues that the connectionist architecture can give rise to rule-like behavior, but also reveals limitations in processing symbolic information. He identifies a number of important challenges and goals for the connectionist enterprise. First, one of Smolensky’s goals is “determining whether the approach [of connectionist modeling] offers computational power adequate for human cognitive competence and appropriate computational mechanisms to accurately model human cognitive performance” (Smolensky, 1988, p. 2). Second, these computational mechanisms — read subsymbolic processes — need to be clarified in order to see how they can give rise to symbolic information processing or rule-like behavior.

2.1.1 Computational power

Computational power refers to the capabilities that a representational or computational system possesses in principle. A system may be capable, for example, of producing or recognizing languages of a certain strength, e.g. regular languages or context-sensitive languages (see Hopcroft et al., 2001, for an introduction to formal languages). As a consequence of this computational power, the system is capable of performing cognitive functions or producing certain behaviors. In fact, recognition or production of formal languages is the paradigmatic test of computational power. Now it is often argued against connectionism that connectionist models can *not* model certain types of behavior, because they lack the computational power to do so. As Levelt (1990) puts it: “It makes little sense to spend years implementing a domain of knowledge in a network that cannot contain it.” More recently Marcus et al. (1999) argued that the way in which infants learn artificial grammars can not be modeled by a neural network. In response to this, both Christiansen et al. (2000) and Schultz and Bale (In press) have designed connectionist networks that model artificial grammar learning in infants. We believe that such disputes can be resolved in a formal manner using the mathematical results that we discuss in this chapter.

¹It is not our goal here to provide a complete picture of the what the classical cognitive architecture amounts to. Instead, we just provide the essential features that enable a comparison with the connectionist architecture.

2.1.2 Overview of the chapter

The present chapter has three goals. First, to establish the computational power of neural networks, second to clarify the notion of subsymbolic processing and third to clarify the relationship between subsymbolic processes and symbolic information processing which is deemed necessary for some cognitive tasks. We first present mathematical results that concern the computational power of connectionist models or neural networks. There are three types of results that we discuss. The first type of result concerns the representation of neural networks as function approximation devices. Second, we discuss a proof that it is possible to implement a (Universal) Turing machine in a neural network. Third we discuss a collection of papers that have established equivalences — either by formal proofs or by simulation results — between (specific) formal languages and certain types of neural networks, mostly simple recurrent networks. In our own research with neural networks works, we relate neural network behavior, and the internal representations that give rise to that behavior, to the tradition of mathematical models of behavior which is established for example in Miller (1952). We focus on performance models rather than competence models by which we mean models of actual behavior instead of models of an idealized set of possible behaviors. The argument in Chomsky (1959b), is explicitly set against behaviorist models of *competence*, and not against performance models. Mathematical models of performance have a long tradition in cognitive psychology, which goes back at least to the 1950s (Miller, 1952; Miller and Chomsky, 1963).

2.2 The computational power of neural networks

The assessment of the computational power of neural networks originates in the classic paper by McCulloch and Pitts (1943). They analyzed neural networks by means of propositional logic, specifically by constructing logical operators such as AND, OR and NOT using binary threshold neurons. McCulloch and Pitts (1943) proved that they could represent any propositional expression using such basic neurons. Their conclusion was that any neural network can be analyzed in terms of the propositions it finds to be true. Although neural networks constructed in this way can indeed represent any proposition, as a tool for cognitive science this type of network is not very useful. There are several reasons for this. First, each proposition requires its own network and hence large numbers of networks are needed to even represent a small domain of knowledge. Second, such networks are open to the same criticisms that have been leveled against the classical cognitive architecture. They are very sensitive to ‘injury’ and can not handle degraded inputs in the way the human cognitive system can. Third, perhaps most importantly, the approach taken by McCulloch and Pitts (1943) concerns deductions made by neural networks. They have shown that networks can be constructed that deduce propositions from other propositions given as inputs. Such processing is very far removed from the cognitive functions in which psychologists are interested. Finally, these networks can not be trained to represent propositions but need to be handcrafted for each new proposition that needs to be modeled. The possibility of neural networks to learn

representations by being presented with exemplars from a knowledge domain, is an important advantage of neural networks over the classical architecture. The McCulloch and Pitts (1943) networks lack this property.

2.2.1 Computability as universal function approximation

A second approach to assessing the computational power of neural networks is more recent, with an early exponent in the 1960s. The approach is to represent neural networks as function approximators: basically, a neural network computes a function from input to output. Studying the possible functions that networks can compute therefore provides insight into the capabilities of neural networks. Of course, for any neural network to be realistic and usable as a model for cognitive behavior, it should have a finite number of inputs and outputs. Therefore, it is required that both the input and output are of finite dimension. The work by Minsky and Papert (1969) can be viewed as a formal characterization of neural networks in terms of function approximation. Minsky and Papert (1969) showed that neural networks without hidden layers — also called perceptrons — are limited to distinguishing linearly separable inputs. They also showed that perceptrons can learn these representations by training. However, the category of linearly separable inputs is very limited. It does not, for example, include the xor-problem which is often used as an initial benchmark for establishing the capabilities of neural networks. Since Minsky and Papert (1969), however, an efficient learning algorithm has been developed to train neural networks with one or multiple layers of hidden units (Rummelhart and McClelland, 1986), namely the backpropagation algorithm. This has widened the scope of applicability of neural networks immensely. Two results that concern the representation of neural networks as function approximators discuss multilayer networks: Hornik et al. (1989) and Hartman et al. (1990). These results are very similar and we limit our discussion to Hornik et al. (1989) as it appeared first, and because it hardly differs from Hartman et al. (1990).

The result by Hornik et al. (1989) concerns multilayer feedforward networks. Feedforward networks are neural networks that only propagate activation forward in the network from input to output layers. That is, these networks have no recurrent or feedback connections. In contrast with the neural networks analyzed by Minsky and Papert (1969), the neural networks discussed by Hornik et al. (1989) are multilayer neural networks that have an input layer, one or more hidden layers and an output layer. In fact, Hornik et al. (1989) show that one hidden layer suffices for their purposes, and that any network with more than one hidden layer can be reduced to one with a single hidden layer, with different connection strengths between the units.

Hornik et al. (1989) prove that multilayer feedforward networks can approximate any Borel measurable function to any desired degree of accuracy. What does this mean for modelers in cognitive science? The result holds for the class of Borel measurable functions and Hornik et al. (1989, p. 361) state that this class “contains virtually all functions relevant in applications”. Now it is certainly the case that the class of Borel measurable functions contains many functions of interest for function theorists. However, this is not necessarily so for the cognitive scientist.

There are two reasons for this. First, Borel measurable functions are real-valued functions with finite input and output dimensions. Symbol manipulation in general is concerned with discrete inputs and outputs, and hence real-valued functions may not be appropriate to model behavior that is best described in terms of symbol manipulation. Second, many interesting cognitive functions do not terminate on a given input. Hadley (2000) notes that partial recursive functions are defined only for certain inputs but not for others. One such function is finding a proof of a theorem of number theory, given the theorem. This function is certainly well defined and partially recursive, but, since this computation does not always halt on a given input — not all well-defined theorems have proofs — it is hard to say what we expect from a feedforward neural network in a case where no proof exists. The conclusion is that, although finding a proof for a theorem in number theory is a cognitive function, albeit highly specialized, such a function can not be appropriately modeled by a feedforward network.

A possibly more serious limitation of the theorem is that it only holds on compacta. For practical purposes, this means that the input must be bounded. Cognitive competence in language production, however, is generally thought to be unbounded (Chomsky, 1959b; Fodor and Pylyshyn, 1988), and the limitation to a finite domain may therefore be crucial. Interestingly, this point has been advanced as an argument for and against the usefulness of neural network architectures. On the one hand, Levelt (1990) argues that the finiteness is a real limitation that renders the result irrelevant to cognitive science. He argues that we can recognize and produce arbitrarily long sentences by using recursive constructions such as *and*. For example, a sentence such as “John and Peter and Karen and Frank and . . . went home” can be extended indefinitely and we would still be able to understand it. A feedforward network for recognizing such sentences would break down at a certain point. Hence, unlimited productivity as required for cognitive models by Fodor and Pylyshyn (1988), can not be captured by such feedforward networks. In contrast, van der Velde (1993) argues that this limitation to finite sentences is exactly what we need in cognitive science. Although he agrees with Levelt (1990) that recursive constructions are part of the competence of language users, he also argues that a limitation in performance is essential, given that the goal is to arrive at realistic models of actual language users. More generally, van der Velde (1993) argues that any finite and actual machine, such as computers or brains, are in fact finite state machines, and are hence bounded in the functions they can compute.

Aside from this disputed limitation of the ‘universality’ of function approximation capabilities of neural networks, there are other, and more important limitations to the use of feedforward neural networks as models for cognitive behavior. One is mentioned by Hornik et al. (1989) themselves. Their proof is limited to deterministic functions and so the result does not imply anything about stochastic functions. Some form of stochastic behavior or mechanism is often essential in modeling experimental data, and hence in modeling cognitive behavior. A second concern about these networks and the functions they can approximate relates to learnability. Although Hornik et al. (1989) prove that feedforward neural networks can represent a large class of functions, they do not specify how such representations may be learned. For cognitive science, this is an essential condition for arriving at a valid

explanation of cognitive behavior. It should be noted here that the theorem and proof of Hornik et al. (1989) does not give any hint whatsoever how this question could be answered. Finally, neural networks may represent a certain function, but this leaves unanswered the question whether it does so in a plausible way; i.e., does the network present an explanation of the internal representations that cognitive scientists are after?

Levelt (1990), for example, constructs a feedforward network that distinguishes grammatical from ungrammatical sentences. From the construction of that network, we can see that the network performs this task like a look-up table. In fact, for every possible sentence it represents whether it is grammatical or ungrammatical according to a particular grammar. The network does not in any way label different parts of the sentence as verbs and others as nouns et cetera, whereas we would certainly expect that to occur in realistic models of language processing. In conclusion, the analysis of neural networks in terms of function approximators reveals that they are very powerful computational mechanisms. However, the results also reveal that this view on neural networks is not very useful for cognitive scientists that are searching for models that can serve as explanations for cognitive behavior.

2.2.2 Universal Turing machine computability

When representing neural networks as function approximators one of the drawbacks is that only real-valued functions can be considered whereas cognitive behavior is often symbolic and discrete, albeit noisy. The generally accepted model for symbolic computation is a Turing machine. In fact, the computational mechanisms of a Turing machine define computational power (Hopcroft et al., 2001). Siegelmann and Sontag (1995) have shown that it is possible to compute any Turing computable function using a recurrent neural network (RNN), which suggests that neural networks have the same unbounded systematicity and generativity needed for modeling cognition. In contrast with the feedforward networks discussed so far, such networks have connections in both directions between groups of units. Their result would seem to settle the question concerning the computational powers of neural networks. Unfortunately, inspection of the proof given by Siegelmann and Sontag (1995) reveals that the situation is not so simple.

Siegelmann and Sontag (1995) determine the computational powers of RNNs by implementing a Universal Turing machine in the nodes and connections of such a network. As a consequence, the resulting RNN is an artificial construction, which in no way resembles the neural networks that are used in modeling cognitive functions. In particular, the Turing network contains dedicated nodes that form the memory (in Turing machine parlance called the tape), other dedicated nodes that read symbols from the memory, yet other nodes that transform these symbols, et cetera. In fact, the neural network contains all the elements that feature in the construction of the normal Turing machine. As a result it also inherits all the disadvantages of such machines, and therefore lends itself to the familiar criticism on the classical cognitive architecture. For example, the brittleness of the classical cognitive architecture carries over to the implementation of Turing machines in RNNs: removing a node from the network influences its behavior catastrophically. Moreover, neural

networks constructed in this way lose all the advantageous properties that make them desirable models of cognitive functions in the first place.

The conclusion from the result by Siegelmann and Sontag (1995) must be that — in principle — RNNs can perform any function that the classical architecture can. However, it must also be concluded that the result does not greatly help the case for connectionists. Each network has to be hand-crafted for a specific function because there are no learning routines available. The style of computation used in these hand-crafted networks is identical to cognitive functions implemented in a classical architecture. Hence, subsymbolic processing — the hallmark of connectionism (Smolensky, 1988) — does not take place in these networks. The style of computation in the networks proposed by Siegelmann and Sontag (1995) does therefore not differ from the classical model. We must conclude that systematicity and productivity *can* be captured in RNNs, but *not* in such a way that the advantageous features of a connectionist architecture are retained.

2.3 The interpretation of neural network behavior

In most applied work with RNNs, the style of computation is very different from the networks by Siegelmann and Sontag (1995). In recent work, RNNs are shown to be able to recognize (regular) languages in a way that is, at least in some aspects, different from the canonical representation of such languages by (finite state) grammars. First, these networks do not suffer from the brittleness of the networks that implement Turing machines because they have distributed representations, at least at the hidden layers. Second, these networks learn to recognize languages by being presented with examples of grammatical sentences. The goal of this section is to clarify how languages are learned by and represented in these networks.

The first result about language recognition by recurrent neural networks that we discuss is the paper by Cleeremans et al. (1989). The network architecture they use is the simple recurrent network (SRN) introduced by Elman (1990), which is depicted in Figure 2.1. In contrast to the networks analyzed by Hornik et al. (1989), this network has feedback or recurrent connections to its hidden layer. In contrast to Siegelmann and Sontag (1995), the approach taken here is characterized by learning specific languages to networks instead of proving a general recognition capacity of such networks. Before describing and evaluating the results, we first provide some background information about the SRN architecture which is essential to understanding the results.

The simple recurrent network consists of three layers: input, hidden and output. The hidden layer has recurrent connections to itself through the context units. When a stimulus or symbol is presented to the network, the hidden layer forms an internal representation of that stimulus. This internal representation is then stored in the context or recurrent units for use at the next time step. When the next stimulus or symbol is presented at the input units, a new internal representation is formed on the hidden layer. This new representation is a combination of the previous internal representation, which is stored in the context units, combined with the current input. In this way, the network keeps track of previous inputs and

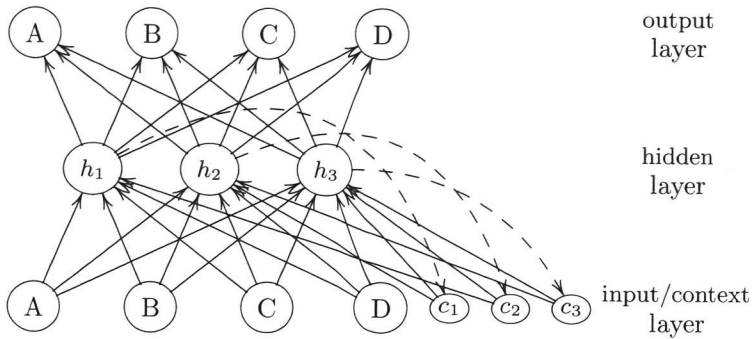


Figure 2.1: The simple recurrent network. See the text for details.

its output is conditioned on this. The network is trained with the backpropagation algorithm (Rumelhart and McClelland, 1986).

In their simulation study, Cleeremans et al. (1989) showed that the SRN builds a reliable representation of a (small) regular language, that is, a language generated by a finite state automaton (FSA)². In particular, the SRN is trained to predict the next grammatical symbol in a sequence of symbols produced by an FSA. Cleeremans et al. (1989) have shown that the network not only reproduces the behavior of an FSA, but also forms internal representations of the structure of the grammar. This work has been followed up by formal proofs that SRNs can represent FSAs (Sontag, 1995; Omlin and Giles, 1996). Although FSAs are not nearly powerful enough for many cognitive tasks, such as natural language processing, it is worthwhile to explore the representations that neural networks build in learning FSAs. Some work has been done in which SRNs are trained to represent context-free languages (Rodríguez et al., 1999) which we will discuss of at the end of this section. First, however, an exposition of the techniques used to train and analyze networks that learn formal languages is in order.

2.3.1 Exploring internal representations of neural networks

Many neural networks have been shown to be adequate models for all kinds of cognitive tasks, such as language learning (Elman, 1993), implicit learning (Cleeremans and McClelland, 1991; Dienes et al., 1999) and memory (Bower, 1993; O'Reilly and Rudy, 2000). However, as many connectionist modelers know, finding interpretations of the internal representations that give rise to the neural network behavior can be cumbersome. It is therefore interesting to find principled ways of analyzing neural network representations and, in doing so, to clarify the notion of subsymbolic processing. This also allows for comparisons between neural networks and rule-based representations of a body of knowledge. Some work has been done

²Finite state languages are the simplest languages from the Chomsky-hierarchy (Hopcroft et al., 2001).

in analyzing feedforward neural network behavior (see e.g. Berkeley, 2000; Intrator and Intrator, 2001, for recent developments in this area). In this section however, we focus on the SRN because it is naturally suited for processing sequential structures such as required for language production and recognition. Learning and recognition of symbolic languages is considered a benchmark for modeling of cognitive tasks and hence much work with neural networks has focused on accomplishing this.

After training an SRN to recognize an FSA, Cleeremans et al. (1989) studied the state space of the hidden layer units of the trained SRN. They recorded the hidden unit activations at successive stimulus presentations and, using cluster analysis, showed that hidden unit activation values represented the history of presented stimuli. Clusters of these activation values correspond closely to the states of the FSA that was used to train the SRN.

This work has seen many follow-ups. First, Giles et al. (1992) trained second-order RNNs to recognize finite state languages. In addition they refined the method to obtain useful representations from the internal state-space of the trained network using a technique called automata extraction. Giles et al. (1992) recorded trajectories of the hidden unit state space as stimuli were presented to the network. Next, they partitioned the state space into hypercubes, which were then labeled, and featured as the states of an FSA that mimics the behavior of the network. When a transition from a certain hypercube to the next occurred, a connection between the corresponding states in the FSA is drawn. Using this technique, Giles et al. (1992) showed that they can recover the original FSA used to train the network. In so doing, they show that the network accurately captures the structure of the FSA in the training phase. This technique provides information about the internal structure of the network and hence about the manner in which it represents the data that were presented to it. The method of automata extraction was subsequently refined by Tino and Köteles (2000).

Visser et al. (2001a) have done similar work in analyzing neural network behavior by representing the behavior of these networks as state space models, in particular discrete state space models which are also known as hidden Markov models (HMM). They showed that based on the input/output relations of a neural network, it is possible to construct a state space model of its behavior and its internal representations which are the states of the HMM. Visser et al. (2000) showed that estimating such models can be done reliably and accurately in the sense that it is possible to retrieve the model that was used to generate data.

Visser et al. (2001a) trained an SRN to recognize symbol sequences from a finite state grammar. When training was completed, the network was used to generate sequences of symbols. These sequences were then used to fit an HMM. The fitted HMM provides a good interpretation of the internal structure of the neural network (Visser et al., 2001a). Hidden Markov models may be viewed as stochastic counterparts of FSAs; as such HMMs are interpretable as probabilistic rule-like models. This is important because it shows that SRNs *can* represent stochastic functions. This is in contrast with the work from Hornik et al. (1989) in which neural networks are analyzed as approximators of deterministic functions³.

³Note however that, although SRNs are here *interpreted* as stochastic automata, a trained

There are a number of differences between automata extraction based on analyzing trajectories in the hidden state space of a neural network and using hidden Markov models to arrive at interpretations of neural network behavior. First, HMMs are fitted on sequences generated by a neural network. Hence, recording of the hidden state space trajectory is not necessary. HMMs are used as a model of neural network *behavior* and to make inferences about the internal representations that give rise to that behavior. Second, because fitting HMMs is done using standard maximum likelihood procedures, there are statistics available for selecting the best model. In contrast, in automata extraction, the partitioning of the state space is arbitrary and as a consequence the resulting automata may contain many redundant states. These redundant states are removed by a technique called minimization (Hopcroft et al., 2001). A disadvantage of minimization is that, when using stochastic data, there is no natural end point for the procedure. In HMMs the natural end point is clearly defined by statistics that indicate overfitting or underfitting. Third, because HMMs are models of overt behavior, similar models can be fitted on experimental data. This allows a direct comparison between experimental data and neural network models for these data. Given this possibility of fitting HMMs to experimental data directly, why should one retain neural networks as models for cognitive tasks? An important reason is that in neural networks, at least at the hidden layers, there are distributed representations which is not the case in HMMs. As a consequence of having distributed representations, many well-known advantages of neural networks arise naturally such as noise tolerance, speed-accuracy trade-offs, pattern completion et cetera (Bullinaria, 1999).

Our discussion in this section so far is limited to finite state behavior. For natural language understanding, more processing power is required. Some results have been obtained with training the SRN with sentences from more complex languages such as context-free languages (Rodríguez et al., 1999). The way in which the SRN represents these languages is similar to the above discussed networks.

Rodríguez et al. (1999) trained an SRN with the context-free language consisting of the strings $a^n b^n$. One of their networks successfully learned to recognize this language and also generalizes to longer strings, that is, it recognizes longer strings than were presented to it in the training phase. Rodríguez et al. (1999) analyzed the hidden state space of the trained network to gain insight into its internal representations. Their analysis revealed that the network counts the number of a 's and then counts the number of b 's backwards. This result is interesting for a number of reasons. First, the network is trained to recognize the language and is hence not hand-crafted such as the Siegelmann and Sontag (1995) network. Second, the analysis of the network shows that the dynamics governing its internal representations instantiated a general principle of counting which is an essential feature needed for recognizing context-free languages. Hence, the results can be generalized to arbitrary context-free languages. The internal representations that are formed by the network during training bear the hallmark of the infinite state machine needed to recognize a context-free language. The infinite number of states

network, with fixed weights, computes a fully deterministic function. For a description of more dynamic neural networks, with weights that develop, without being provided with feedback, during recognition or recall see Ramacher (1994).

have to be represented in a finite and continuous state space. As a result, due to limitations in precision, performance fails for longer strings. The representations formed in this network are very different from the network representation proposed by Levelt (1990). The internal representations of the SRN reflect characteristics of the language that is represented instead of labeling all possible sentences as belonging to the language or not belonging to the language. Hence, as a model for language users, the SRN is much more realistic.

Other network architectures have been proposed for the representation of context-free languages. Tabor (2000) provides a proof that a fractal encoding of context-free grammars lends itself for natural implementation in neural networks. The style of representation of these fractal networks is similar to the networks of Rodriguez et al. (1999), but learning of these languages is not investigated. Another network for context-free languages is the long short-term memory network (Gers and Schmidhuber, 2001) which is a recurrent network. Although the network can learn context-free languages, the units that are formed are highly specialized and there are no distributed representations. As a result, it lends itself to the same criticism as the Siegelmann and Sontag (1995) network.

The possibility of training a neural network to recognize an FSA and the reverse process of extracting FSAs or HMMs from such networks establishes an interesting link between these classes of models. In particular, it shows that neural networks can perform rule-like behavior, while retaining the advantageous properties of connectionist models such as noise tolerance. In addition, neural networks can be trained to perform these tasks by presenting them with sentences produced by FSAs which is an important advantage over the networks presented by Hornik et al. (1989) and Siegelmann and Sontag (1995).

2.3.2 *Subsymbolic processing*

The automata extraction techniques described above provide valuable insights into the internal representations of the simple recurrent network. In particular, it is found that the states of extracted automata correspond closely, sometimes even one-to-one, to the states of the FSA that was used to train the network. The hidden state space of the SRN is divided into regions that play the same role as the states of an FSA, that is, they determine the next output to be produced.

Berkeley (2000) argued that the activities of hidden units should be interpreted as subsymbols. He shows that different regions of the hidden unit state space implement the application of different rules to the input that result in the correct output. Similarly, in the SRN that we trained on an FSA, the activities of the hidden units can be interpreted as subsymbols, in the sense required by Smolensky (1988). Clusters of such units with certain activities, i.e. subsymbols, together define a region, which is identified with a (symbolic) state of an FSA.

There are important differences between subsymbolic representations of regular languages and their canonical representation in finite state automata. First, the hidden state space of an SRN is continuous. During learning, hidden unit activities cluster into regions of the state space. However, when presented with identical symbols at different times, the hidden unit activities are hardly ever exactly identical

even though they fall into the same region that represents a state of the FSA. This means that, although different clusters of subsymbols are functionally equivalent, they also show context dependence which is an important feature of cognitive representations (Smolensky, 1988). Second, a small perturbation of the hidden unit activities, by noise for example, does not affect the behavior of the network. In contrast, in an FSA, a random change of state would catastrophically influence its behavior. Third, in the trained SRN, representations at the hidden states are distributed such that a number of hidden units together define a region that corresponds to a state in an FSA.

2.4 Discussion

The viability of connectionist modelling depends to a large extent on our ability to establish the computational power of neural networks. From the results that we discussed, especially those of Siegelmann and Sontag (1995), it is clear that RNNs have the same computational power as the classical architecture. This computational power is deemed essential for models of cognitive functioning (Fodor and Pylyshyn, 1988). Hence, arguments of the form put forward by Levelt (1990) — “It makes little sense to spend years implementing a domain of knowledge in a network that cannot contain it” — are not warranted by these results because any domain of knowledge can be implemented in a neural network. However, it is equally clear that doing so by means of the neural networks constructed by Siegelmann and Sontag (1995), does not provide a viable alternative to the classical architecture, because in essence they have the same structure as that architecture.

We argued that the SRN is more promising as a model for cognitive tasks. The SRN can represent finite state languages. The representations that an SRN builds in learning a finite state language are, intensionally, very close to those of a finite state automaton with the important difference that the states are represented in a distributed way. In addition, there are many advantages that arise naturally in neural networks such as frequency effects in learning, graceful degradation after lesioning, speed-accuracy trade-offs, noise tolerance and pattern completion (see Bullinaria, 1999, for an exposition of these points). These characteristics are desirable features of models for cognitive behavior since they are observed in many experimental situations. Most importantly, the SRN *learns* to recognize a language, and hence is an important extension to classical models that have more static representations.

In our own research, we compared hidden Markov models with simple recurrent networks (Visser et al., 2001a). We showed that by fitting HMMs to sequences generated by the SRN we can get useful interpretations of the behavior of the SRN. In so doing, it is shown that the states of an FSA correspond to clusters of activities at the hidden units of the network. The hidden unit activities are thereby interpreted as subsymbols that together define a symbolic state of an FSA.

Neural networks that learn languages are characterized by the same generative capacity and systematicity as the classical architecture. For languages from finite state automata, this result is well established (Sontag, 1995). For higher order languages such as context-free languages, there are important simulation results

(Rodriquez et al., 1999; Rodriquez, In press) and a more formal result by Tabor (2000). Hence neural networks certainly have the competence required of a model of cognition. Neural networks also reflect limitations in performance. Although, in principle, people should be able to recognize arbitrary linguistic constructions, in practice their performance is very limited. Such limitations in performance naturally show up in training neural networks. These limitations are due to the fact that the state space of a neural network is finite and continuous and because arbitrary precision is impossible. From the point of view of building performance models of cognition, such limitations should therefore be viewed as a bonus.

