Journal of Applied Intelligence, 12, 27–44 (2000)


# Knowledge Extraction from Transducer Neural Networks

STEFAN WERMTER
*University of Sunderland, Centre of Informatics, SCET*
*St. Peter's Way, Sunderland SR6 0DD, United Kingdom*

;

**Abstract.** Previously neural networks have shown interesting performance results for tasks such as classification, but they still suffer from an insufficient focus on the structure of the knowledge represented therein. In this paper, we analyze various knowledge extraction techniques in detail and we develop new transducer extraction techniques for the interpretation of recurrent neural network learning. First, we provide an overview of different possibilities to express structured knowledge using neural networks. Then, we analyze a type of recurrent network rigorously, applying a broad range of different techniques. We argue that analysis techniques, such as weight analysis using Hinton diagrams, hierarchical cluster analysis, and principal component analysis may be useful for providing certain views on the underlying knowledge. However, we demonstrate that these techniques are too static and too low-level for interpreting recurrent network classifications. The contribution of this paper is a particularly broad analysis of knowledge extraction techniques. Furthermore, we propose dynamic learning analysis and transducer extraction as two new dynamic interpretation techniques. Dynamic learning analysis provides a better understanding of *how* the network learns, while transducer extraction provides a better understanding of *what* the network represents.

**Keywords:** Neural network learning, symbolic interpretation, knowledge extraction, SRN networks, analysis of connectionist learning

## 1. Introduction

There has been a lot of interest lately in knowledge structures and their representation in artificial neural networks [Hölldobler, 1990, Kurfeß, 1991, Sperduti et al., 1995, Wermter, 1995, Hallam, 1995, Medsker, 1995, Sun, 1995, Wermter et al., 1996, Elman et al., 1996, Craven, 1996, Wermter, 1999]. Artificial neural networks (or connectionist networks) have already demonstrated interesting learning results for various classification tasks. However, it continues to be very difficult to understand the underlying representations within the connectionist networks which lead to this performance. A better understanding of the connectionist representations learned is not only important for improving the credibility of a computational technique, but also for improving the network performance and the integration possibilities with symbolic representations.

Several attempts have been made to interpret connectionist networks, focusing on feedforward networks in particular [Andrews and Diederich, 1996, Abe et al., 1993, Shavlik, 1994]. For instance, visualizations of internal activations or weight strengths can be used to get an impression of the internal knowledge [Hinton, 1986, Gorman and Sejnowski, 1988]. Some effort has also been made to reduce the network size in order to simplify the knowledge expressed therein by eliminating very small weights. Furthermore, groups of similar weights can be replaced with their average strength [Shavlik, 1994]. In addition, techniques such as hierarchical cluster analysis have been used to interpret connectionist networks. Nevertheless, often the interpretation of the dynamics of the learning process and the underlying knowledge has been neglected, especially in the case of dynamic recurrent neural networks.

The interpretation of recurrent networks is more difficult than that of non-recurrent feedforward networks, since the previous context in recurrent networks has an important dynamic influence within these networks. The internal states in recurrent networks do not only depend on the input but also on the internal state of the local memory based on previous inputs [Elman, 1995, Giles and Omlin, 1993, Omlin and Giles, 1996]. For this reason, to date the focus has been primarily on smaller recurrent networks and artificially generated data. For instance, an interesting current approach interprets the training of a SRN network that has two input, two output and two internal elements in learning the sequence $a^n b^n$ [Wiles and Elman, 1996]. It has been discovered that the network behaved like a spiral which moved to and from a fix point. Whereas this seems to be a plausible interpretation of the behavior of recurrent networks trained for the learning of the sequences $a^n b^n$, different interpretations are required when we move to different tasks and data sets closer to real-world scenarios.

In the past, we have developed a large "real-world" system for spoken language analysis which makes extensive use of SRN networks [Wermter and Weber, 1997, Wermter and Meurer, 1997]. The spoken input is recognized by a speech recognizer and analyzed at the syntactic, semantic and dialog levels based on an incremental analysis, parallel syntactic and semantic interpretation, and robust processing of errors. To date, however it is not yet possible to focus on the interpretation of the learning process and the interpretation of the connectionist knowledge. In this paper, we are primarily concerned with a detailed interpretation of the learning behavior as well as a symbolic interpretation of the learned knowledge after training. In order to carry out such a detailed analysis we will concentrate on a syntactic transformation task as a representative task for our large-scale speech/language system. The task for the recurrent network is to process sentences and associate their syntactic classes at the phrasal level, e.g. noun phrase, prepositional phrase etc.

Using this task, we analyze a recurrent neural network using many different techniques. We have structured the paper as follows. First, we introduce our representative syntactic transformation task. Then, we define and illustrate a) dynamic learning analysis, b) weight analysis, c) hierarchical activation analysis, d) component activation analysis, and e) transducer extraction. We rigorously compare these techniques on the same network and the same data set and argue that these different techniques provide mutually complementary interpretations. The contribution of this paper is a particularly broad and concrete analysis of the knowledge extraction process which has not been done before. Furthermore, we propose dynamic learning analysis and transducer extraction as two new interpretation techniques. Dynamic learning analysis provides a better understanding of *how* the network learns while transducer extraction provides a better understanding of *what* the network represents.

## 2. Extracting structured knowledge using syntactic analysis task

In order to examine a number of different techniques for extracting structured knowledge from connectionist networks in a rigorous manner, we will focus on a particular task. In our spoken language environment [Wermter and Löchel, 1996, Wermter and Weber, 1997, Wermter and Meurer, 1997], we have trained many variations of SRN networks [Elman, 1991] with many sentences using various corpora of several thousand words each.

Based on a corpus of sentences from the domain of scheduling appointments (2355 words), table 1 summarizes the accuracy of label assignment on the unknown test set. The related experiments and results have been reported elsewhere in detail [Wermter and Löchel, 1996, Wermter and Weber, 1997, Wermter and Meurer, 1997, Wermter, 1998]. Here we just want to illustrate the real-world network performance in table 1. The focus, however, is on an analysis of the process of extracting explicit knowledge from implicitly learned knowledge. In this paper, we concentrate on syntactic phrasal assignment (marked by *) in table 1.

*Table 1.* Performance of some networks on the test set of the appointment scheduling corpus

| Task | Accuracy on test set |
|---|---|
| Basic syntactic disambiguation | 89% |
| Basic semantic disambiguation | 86% |
| Syntactic phrasal assignment* | 84% |
| Semantic phrasal assignment | 83% |
| Dialog act assignment | 79% |
| Word repair detection | 94% |
| Phrase repair detection | 98% |

To demonstrate this process of knowledge extraction, we will here use 15 of these sentences (containing 76 words) from the domain of appointment scheduling. For illustration purposes, we concentrate on the learning of a syntactic phrasal assignment task where a sequence of basic categories of words is associated with a sequence of abstract syntactic categories. The actually occurring syntactic basic categories are noun (n), verb (v), adverb (a), adjective (j), preposition (r), determiner (d) and pronoun (u). The abstract phrasal categories are noun group (ng), verb group (vg), and prepositional group (pg). The task of the recurrent network is to learn to assign phrasal categories on the basis of basic syntactic categories in order to support a robust flat understanding of spontaneously spoken language. Below, we show some example utterances from the corpus, together with the syntactic categories at the basic and the phrasal level.

1. I (u → ng) thought (v → vg) in (r → pg) the (d → pg) next (j → pg) week (n → pg)
2. That (u → ng) is (v → vg) the (d → ng) Thursday (n → ng) after (r → pg) Easter (n → pg)

Based on these seven basic syntactic and three phrasal syntactic categories, we use an SRN network with seven input units, three internal units and three output units (the networks in the actual system contain more categories and have been trained with several thousand words, but for illustration purposes we restrict ourselves to this smaller network). The learning rate was 0.05 and momentum 0.9. The weight updates were performed incrementally after each training pattern. Each training pattern consisted of the basic syntactic category at the input layer and the abstract phrasal category at the output layer.

Figure 1 shows a simplified example of such a recurrent network for the task of syntactic phrase assignment.

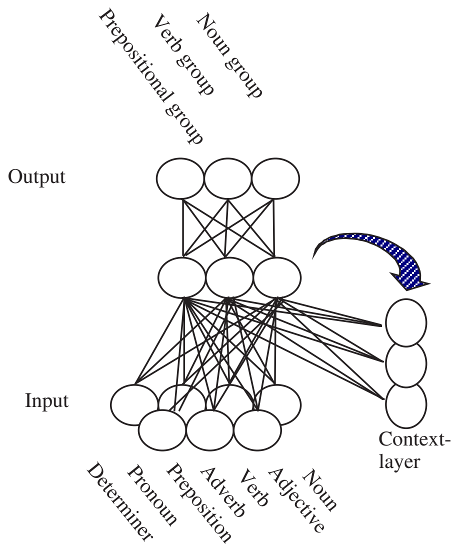


*Fig. 1.* Recurrent network for knowledge extraction for syntactic phrase assignment

The activation of an output element $O_j(t)$ at time $t$ in SRN networks is computed on the basis of the weighted activation $H_i(t)$ of all incoming connections limited by the logistic function $f$.

$$O_j(t) = f(\sum_i w_{ij} H_i(t))$$

The activation of an element on the internal layer $H_l(t)$ is computed in a similar manner. Here the activation of the input layer $I_k(t)$ at time $t$ is used as the activation of the internal layer $H_m(t-1)$ at the previous time step $t-1$.

$$H_l(t) = f(\sum_k w_{kl} I_k(t) + \sum_m w_{ml} H_m(t-1))$$

## 3. Dynamic learning analysis: knowledge structuring during lazy learning

In the past, most work on knowledge structures and connectionist networks has focused on static connectionist network representations. However, important insights can be gained by examining how certain knowledge structures emerge and develop over time before a certain task is learned completely.

Frequently, the interpretation of the learning behavior is just demonstrated by means of the learning curve of the overall error reduction over time. However, the learning curve is just the first step in a more detailed analysis and can only provide preliminary hints about the performance of a network over the training time. Figure 2 shows the learning curve with the overall sum squared error over time.

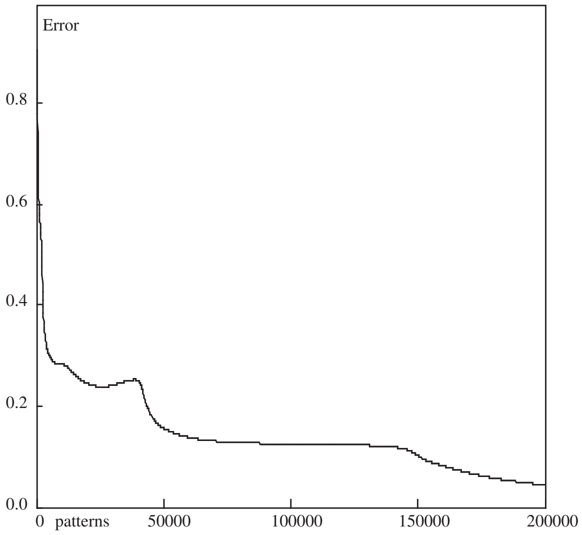


*Fig. 2.* Learning curve for syntactic phrasal assignment

The learning curve shows that the speed of learning differs substantially over time. Furthermore, we can see different stages during the learning process. In the beginning, learning proceeds fast, but later learning is slower and it takes longer to make significant improvements. For instance, between 70000 and 140000 it seems that learning is about to finish before there is a final significant improvement.

We will now examine *how* the network reaches its performance. We start the analysis directly after the random initialization of the weights. This is the state before learning starts. We want to give an overview of the overall performance for all input patterns at different time steps. To this effect, we show the error for each of the 76 patterns of the demonstration set at different time steps. Figure 3 shows the individual error for each of the 76 patterns before training.

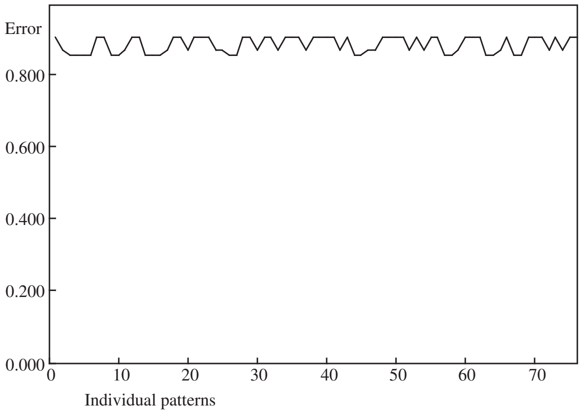


*Fig. 3.* Performance for individual patterns before learning

Based on the random initialization, all patterns show a relatively high error. At this point, it is to be expected that the values of each output element differ from the desired value 0 or 1 by 0.5. Therefore the expected error for an individual pattern for three output elements is $\sqrt{0.5^2 + 0.5^2 + 0.5^2} = 0.866$. This expected error value is confirmed in this figure.

As shown in figure 2, the error decreases quickly at the start of the training. The state after 100 patterns of the training set is shown in figure 4. First, we can observe that after 100 training patterns, the error for some of the 76 patterns shown could be reduced significantly. Other patterns still show a high error. Obviously, the network has started to learn patterns selectively.

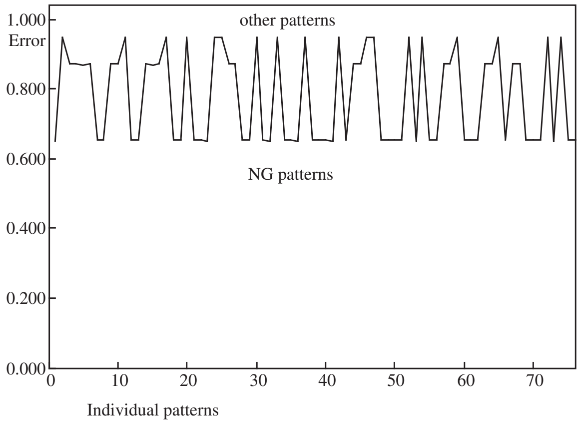


*Fig. 4.* Performance for individual patterns after 100 training patterns

A more detailed analysis revealed that the patterns with a lower error are exactly those patterns which belong to the noun group $NG$. After only 100 patterns, the network has recognized that the global error can be minimized significantly by focusing on the $NG$ patterns, since these patterns occur more frequently than, for instance, prepositional groups or verb groups. Therefore, at first the network has learned a constant mapping of all patterns to the noun group, since this reduces the overall error most at this stage. This explains why certain patterns in figure 4 still exhibit a high error and others a low error. The patterns with a low error are exactly the patterns which have been classified correctly as noun groups.

Figure 5 shows the detailed performance after 600 patterns. After the network has learned a constant mapping to $NG$, we can observe that the performance for the $NG$ patterns has improved even further. However, we also observe that $VG$ patterns have been learned. A more detailed analysis of the output preferences reveals that at this stage, in addition to all $NG$, all $VG$ patterns have also been learned correctly. This is also demonstrated in figure 5. All the remaining error patterns at this stage are those patterns which should belong to a prepositional group $PG$ but which are still categorized as noun groups $NG$. All $NG$ patterns and all $VG$ patterns are classified correctly. After the network has learned the most frequent $NG$ patterns, the second most frequent $VG$ patterns are learned. Thus, one could state that the network pursues a conservative *lazy learning* strategy and learns frequently occurring and simple regularities first.

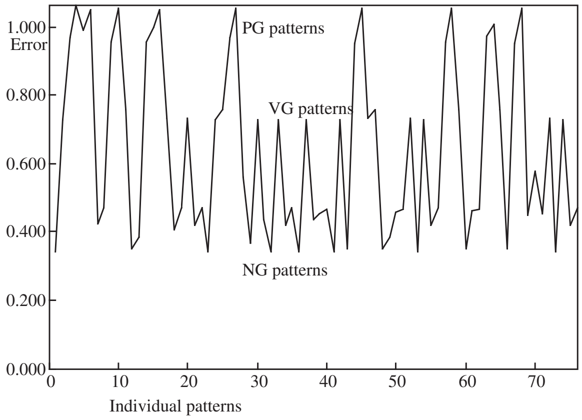


*Fig. 5.* Performance for individual patterns after 600 training patterns

Afterwards, the network attempts to improve all patterns, especially the remaining patterns for prepositional groups $PG$. The occurring nouns, pronouns, determiners, and adjectives can either be part of $PG$ patterns or $NG$ patterns. In order to resolve this potential for ambiguity, previous context must be used to learn the correct class assignment. Again, we have an example of the conservative lazy learning strategy of the network,

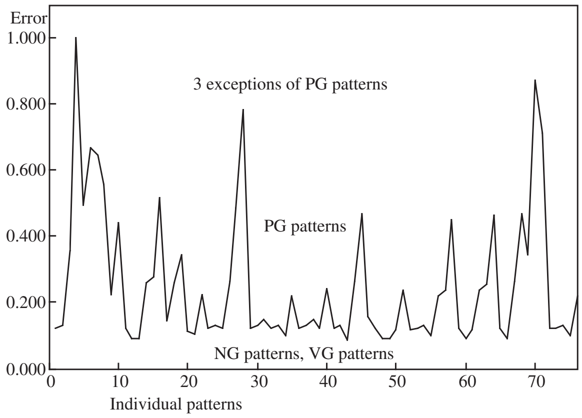


*Fig. 6.* Performance for individual patterns after 3000 training patterns

since at first the network has learned patterns which do not need previous context knowledge for the category assignment. Only after the simple non-context-dependent category assignments have been learned, are those patterns learned which require the context of previous pattern assignments.

The state of the network after 3000 patterns is shown in figure 6. All patterns are classified correctly with the exception of three. Comparing figures 5 and 6, the remaining error for the individual patterns could be reduced significantly. For the learning of the $PG$ patterns, it was necessary for the network to integrate the local preceding context. After 150000 patterns all regularities have been learned as shown in figure 7. In comparison with figure 6 we point out the smaller scaling of the vertical axis. At this stage all patterns have been learned, even though there are differences between the error rates of individual patterns. In order to reach this 100% correctness on the training set, it may be necessary to give up a reasonably good state at a certain stage in order to reach an even better stage later. This is also reflected in the global learning curve in figure 2.

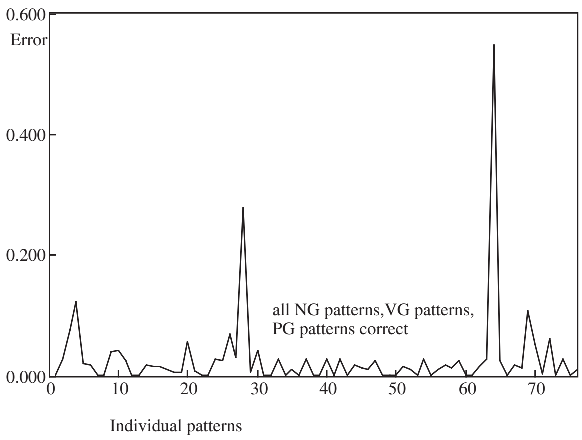


*Fig. 7.* Performance for individual patterns after 150000 training patterns

In general, the network pursues a conservative lazy learning strategy. First, simple and frequently occurring generalizations of one category are learned. Only when the network cannot minimize its error significantly any more, are other frequently occurring categories integrated. Furthermore, only when all those patterns have been learned that do not require previous local context, are those patterns learned that require context for the correct category assignment of otherwise ambiguous input. Finally, any remaining exceptions are learned. During this conservative learning process it may be possible that the overall error increases briefly in order to reach a better overall state later.

## 4. Weight analysis for knowledge extraction

Visualizations of internal weight strengths can be used to get an impression of the internal knowledge. In our experiments, the training set was learned correctly after 150000 patterns and this is where we start our analysis. We start with such a weight analysis since weights provide the lowest level of interpretation of a connectionist representation. Figure 8 shows the weights of the network for three different time steps. It is illustrated how the weights change over time during learning.

In this figure the identifiers of the source connectionist elements are shown horizontally and the identifiers of the goal connectionist elements are shown vertically. We start with the horizontal axis. From left to right, we can see the weights from the threshold element (S), from the input connectionist elements for the syntactic basic categories (n, j, v, a, r, u, d), from the three internal elements (h1, h2, h3) and from the three context elements (c1, c2, c3). In the vertical axis from top to bottom, we see the weights to the three internal elements (h1, h2, h3) and to the output elements representing the abstract syntactic categories (VG, NG, PG).

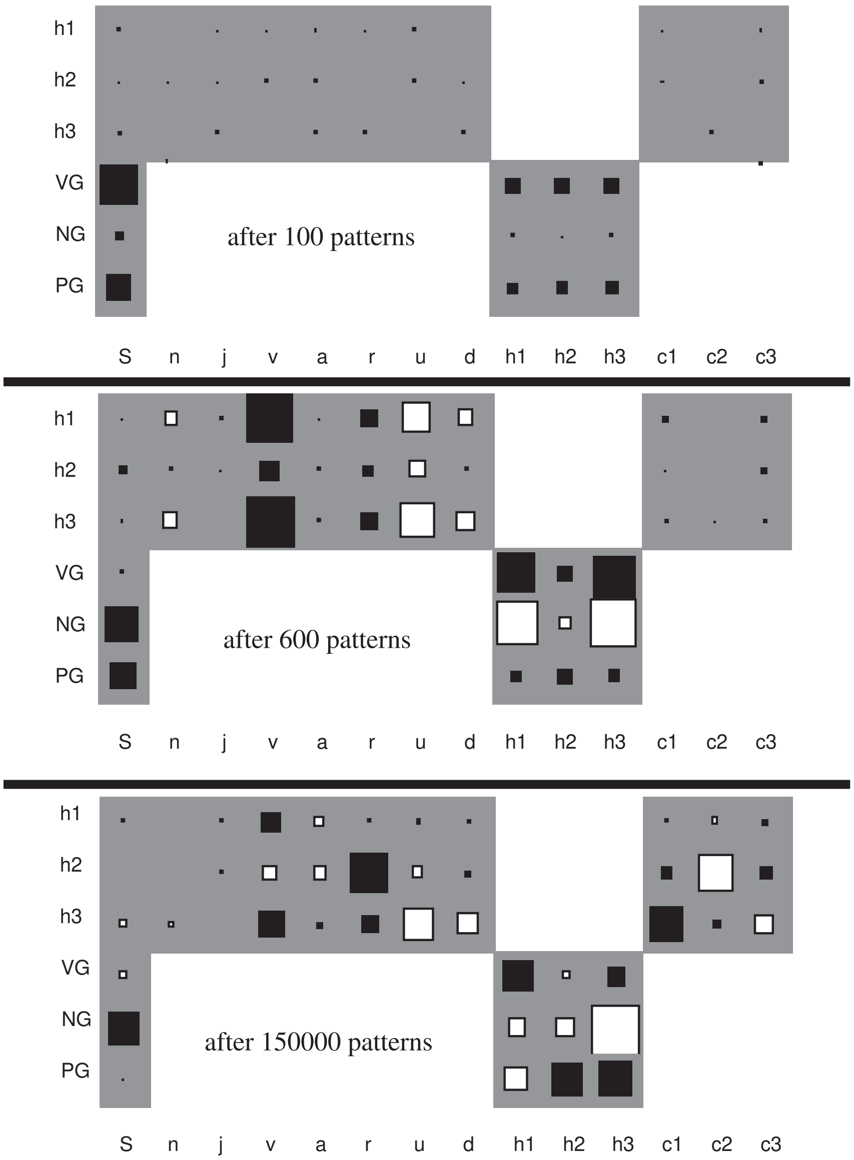


*Fig. 8.* Weight analysis at the beginning of training (100 patterns), during training (600 patterns) and after training (150000 patterns)

White boxes represent positive weights, black boxes negative weights. The size of the boxes corresponds to the size of the weights. The copy connections from the internal layer to the context layer are not changed. Therefore, they are not shown since they are always equal to 1.

We start with the analysis of the first third of figure 8. After random initialization, this first third shows all weights of the network after 100 patterns. At this point, all $NG$ patterns can be classified correctly, but no other patterns have been learned yet. The network has learned a constant output in order to reduce the overall error as much as possible. We can see in figure 8 *why* the network produced this constant $NG$ class.

We can see that the weights from the input elements of the syntactic basic categories ($n$, $j$, $v$, $a$, $r$, $u$, $d$) to the internal elements ($h1$, $h2$, $h3$) are relatively small and similar. The same holds for the weights from the context elements ($c1$, $c2$, $c3$) to the internal elements. This is due to the random initialization at the beginning of the training. The weights from the internal elements to the output elements of the abstract syntactic categories ($VG$, $NG$, $PG$) are negative for $VG$ and $PG$; those from the internal elements to $NG$ are close to 0. This is the reason why the network produces constantly the $NG$ category at this stage.

Now we focus on the state of the network after presenting 600 patterns, also shown in figure 8. At this point, all $NG$ and all $VG$ patterns are assigned correctly. This is also reflected in the weights. We observe positive weights from $n$, $u$ and $d$ to the internal elements and positive weights from the internal elements to $NG$. However, we see negative weights from $v$ to the internal elements and from the internal elements to $VG$. The $PG$ patterns are not categorized correctly at this point. One reason for this is that the $PG$ patterns depend significantly on the previous context. However, at this point, the network has just learned the obvious preferences and is only just starting to change the weights of the context layer.

The network state after 150000 patterns is shown at the bottom of the figure. In the internal layer, a distributed representation has developed. Therefore, a direct interpretation is not easily possible. However, it is observed that the first internal element is primarily important for $PG$ detection, the second internal element plays an important role in $VG$ assignment and the third internal element is important for $NG$. Nevertheless, this is a distributed rather than a local representation and there is additional influence from other elements. Furthermore, the weights of the context layer (from $c$ to $h$) have changed. This is necessary in order to learn the $PG$ group assignment.

Generally speaking, we can explain certain phenomena using this type of weight analysis at the lowest interpretation level of a network. However, it is difficult to extract explicit knowledge and a deeper understanding of the behavior of the network directly from the weights. Reasons for this difficulty include (1) the static representation of the weights which does not show the dynamics of a recurrent network, (2) the distribution of weights and activation, and (3) the number of weights, especially in the case of larger networks. Therefore, some effort could be made to reduce the size of the network by eliminating very small weights. Furthermore, groups of similar weights could be replaced with their average strength. Nonetheless, weight analysis is still too detailed for larger networks.

## 5. Component activation analysis for knowledge extraction

Weight analysis focuses on the weights and provides a very low-level analysis. One way to address this problem is to move towards activation analysis where the activations of internal elements are analyzed. Since internal elements receive activation from a number of weighted connections, the activation of an internal element integrates several weighted connections and provides a higher abstraction level of analysis.

In order to demonstrate how such an analysis is performed, we will use the same SRN network we have introduced in the previous section and store all vector representations of the internal layer for each pattern. These vector representations constitute the input to a cluster algorithm which provides a hierarchical representation in the form of a dendrogram. Vectors with similar vector representations will end up in the same cluster.

Figure 9 shows the initial part of patterns as they were clustered according to their internal activations. It can be clearly observed that the internal representations reflect the classification according to the three classes $VG$, $NG$, and $PG$. That is, based on the weights, the internal layer has learned representations which particularly support this classification. A single word can appear in different contexts and can lead to different internal representations. For instance, the word “the” is shown with two different representations. One representation is its use as part of the $NG$ class, and the other as part of the $PG$ class. Therefore, we find both representations at different positions within the dendrogram.

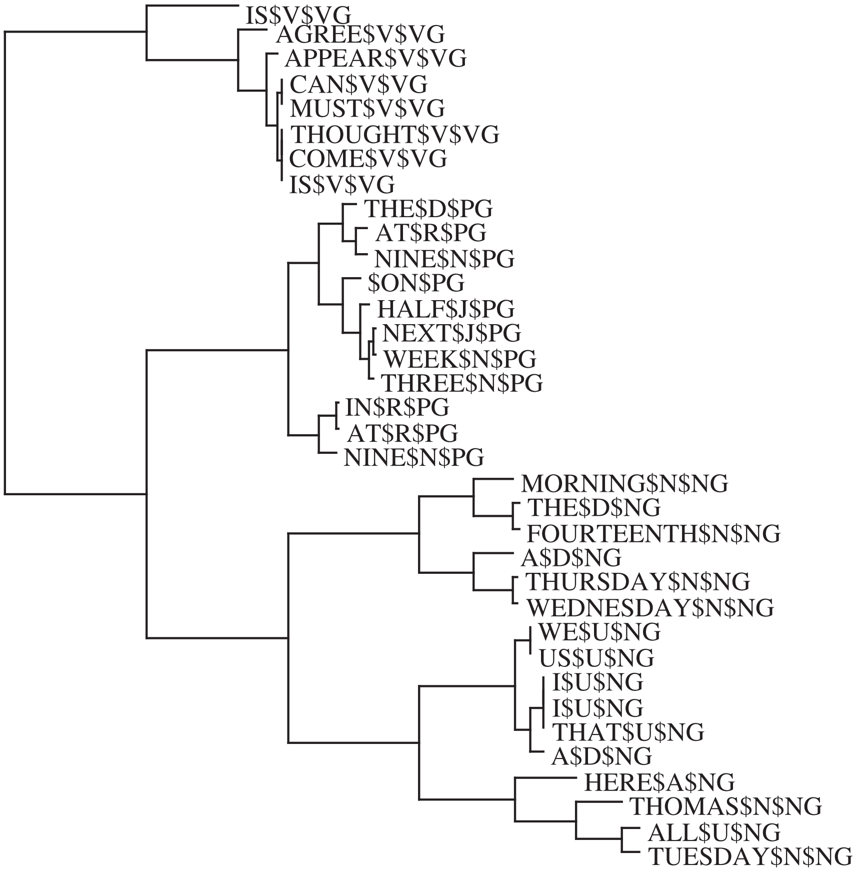


*Fig. 9.* Hierarchical cluster analysis of internal classification representations (for visibility purposes, only a portion is shown here)

## 6. Principal component analysis for knowledge extraction

Another kind of analysis which can be used for interpreting the internal representations of classifications is principal component analysis. Figure 10 shows the result of this analysis for our current task. All vectors from the internal layer and the corresponding identifiers provide the input for the principal component analysis. Vectors which differ substantially from each other are depicted in the figure with a large distance. It can also be observed that the internal representations reflect the preference mappings learned for the three category classes. $NG$, $VG$, and $PG$ patterns are distributed across different areas. Thus, the classification of the internal representations can be clearly seen. This shows that the network has actually learned the classification task well.

After learning has been completed, the internal representation characterizes the preference mapping. According to cluster analysis or principal component analysis, similar internal vector representations are responsible for the representation of similar preference assignments to equal categories. However, the interpretation of the weights by means of Hinton diagrams and of the activations via cluster analysis and principal component analysis only provides a limited form of structuring to the extracted knowledge.

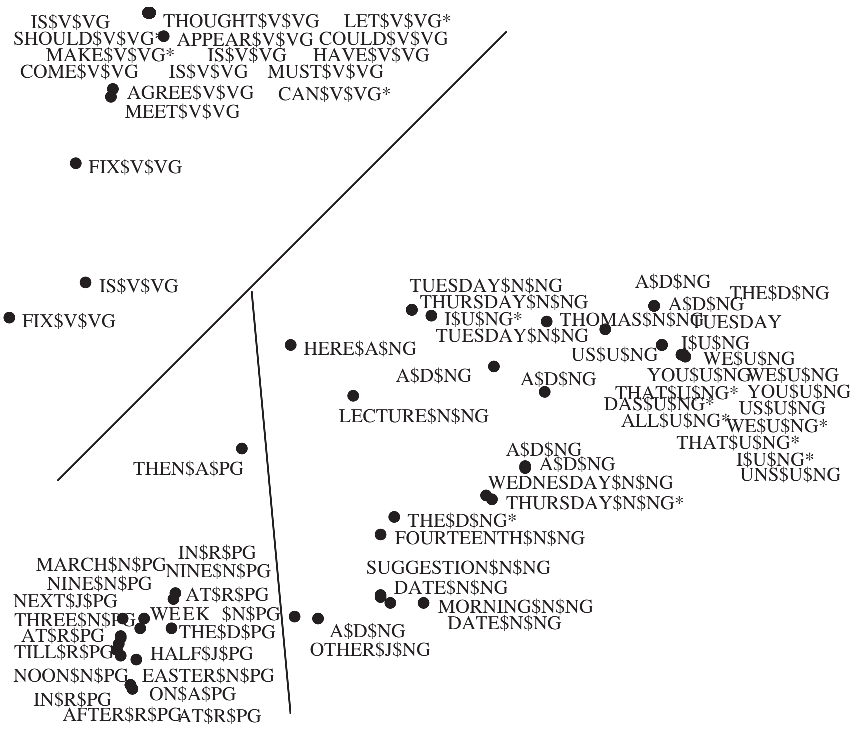


*Fig. 10.* Principal component analysis of internal classification representations

## 7. Transducer extraction

Words and sequences of words can be represented as syntactic, semantic, and pragmatic category preferences. Then they can be input, for instance, to SRN networks. Each input representing a sequence of category preferences is associated with a sequence of corresponding output preferences. This simple description of sequence analysis is similar to the function of synchronous sequential machines [Booth, 1967, Kohavi, 1970, Shields, 1987], although preferences and learning are not yet considered in such machines. Therefore, we shall focus on extensions of synchronous sequential machines for representing sequential knowledge, especially synchronous Moore machines. We start with the basic definition of a synchronous sequential machine which is also called a transducer:

**Definition of a Synchronous Sequential Machine, Transducer**

A *synchronous sequential machine* $M$ is a tuple $M = (I, O, S, f_s, f_o)$, with

1. $I$, $O$ finite, nonempty sets of input and output
2. $S$ nonempty set of states
3. The function $f_s : I \times S \to S$ is state transition function
4. The function $f_o$ is an output function. If the output depends on the state and the input, the machine is a so-called Mealy machine with the output function $f_o : I \times S \to O$. If the output only depends on the state the machine, the later is a so-called Moore machine with the output function $f_o : S \to O$. These synchronous sequential machines are sometimes called transducers.

A sequential machine assigns an output and a new state to an input and an old state. This can be done for a whole sequence of inputs and states in discrete time. The set $S$ is not necessarily finite [Booth, 1967], although this is assumed in the case of finite machines. Whereas automata or acceptors of languages decide whether a certain input belongs to the corresponding grammar, these sequential machines are transducers which change their internal states dynamically, depending on the inputs and the previous states, while also providing an output for each input.

Mealy and Moore machines are slightly different from each other. Moore machines determine the state first and afterwards this state is used to provide the output. In contrast, the output in a Mealy machine depends also directly on the current input. However, it can be shown that for each Moore machine there is an equivalent Mealy machine and vice versa [Booth, 1967, Hopcroft and Ullman, 1979].

In our case, we concentrate on Moore machines since the output in certain neural networks is based on the internal state. This holds, for instance, for feedforward networks or SRN networks. Whereas sometimes [Sun, 1995] a sequential machine has been used to model a single element of a neural network, we want to use a sequential machine as a description for a whole network. This is also motivated by the fact that real neuron systems can be seen as physical entities which perform state transitions [Churchland and Sejnowski, 1992].

Now we can specify language knowledge by describing Moore machines and their state transition function $f_s$ and output function $f_o$. We can also integrate $f_s$ and $f_o$ to a function $f : I \times S \rightarrow O \times S$. Then $f$ corresponds for instance to the transformation within a SRN network. The specification of a Moore machine could be performed by using state tables. A potential entry for the task of assigning syntactic phrasal categories to syntactic basic categories could be:

If input = verb and current state = prep. group then new state = verbal group and output = verbal group

It may not be possible to assign a direct interpretation to a state. For this reason, simple identifiers may be used:

If input = verb and current state = 4
then new state = 5 and output = verbal group

It is possible to define state transition tables which assign each combination of input and current state an output and a new state. In this way, a symbolic synchronous sequential machine is specified. If clear regularities are known beforehand and the number is limited, such tables can be composed manually. However, the number of input and state combinations quickly gets so large that automatic procedures become necessary.

The above-mentioned state transition tables are discrete symbolic. Therefore, they do not support gradual representations. For instance, the input or the state could be ambiguous and different gradual preferences could exist for different interpretations. For instance "meeting" could have a stronger preference for its syntactic interpretation as a noun and a smaller preference for a verb form. Consequently, we want to use preferences for the input, output, and states of such machines. Preferences of this type should be able to take values from $[0,1]^m$ so that multiple preferences can be represented and integrated.

If we extend a single category (as in: if $input = verb$) to an $n$-dimensional preference for the input and an $m$-dimensional preference for the output then we obtain a new synchronous machine which we will call a preference Moore machine. Now we want to describe such a synchronous sequential preference Moore machine which transforms sequential input preferences to sequential output preferences. We will see that simple recurrent networks or feedforward networks can be interpreted as neural preference Moore machines. Furthermore, we will show how symbolic and neural knowledge can be integrated quite naturally using preference Moore machines.

**Definition of a Preference Moore Machine**
A preference Moore machine $PM$ is a synchronous sequential machine which is characterized by a 4-tuple $PM = (I, O, S, f_p)$, with $I$, $O$, and $S$ being non-empty sets of inputs, outputs and states. $f_p : I \times S \rightarrow O \times S$ is the *sequential preference mapping* and contains the state transition function $f_s$ and the output function $f_o$. Here $I$, $O$ and $S$ are $n$-, $m$- and $l$-dimensional preferences with values from $[0,1]^n$, $[0,1]^m$ and $[0,1]^l$, respectively.

A generalized version of a preference Moore machine is shown in figure 11 on the left. The preference Moore machine realizes a sequential preference mapping, which uses the current state preference $S$ and the input preference $I$ to assign an output preference $O$ and a new state preference.

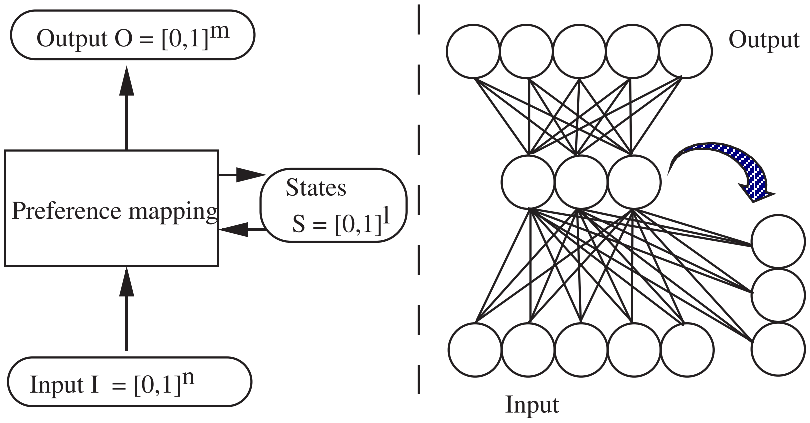


*Fig. 11.* Neural preference Moore machine and its relationship to a SRN network

Now we describe a new technique of extracting the knowledge within a recurrent network in the form of a transducer. A symbolic transducer can be extracted from our recurrent network which assigns to each input vector of basic syntactic categories a new output vector of phrasal categories depending on the previous context. In our network, the internal state and the context were represented by a three-dimensional vector. For simplicity, each strict symbolic interpretation of a three-dimensional vector can take $2^3$, that is 8 states. In order to acquire a symbolic interpretation of the network, we presented all patterns from the training set and stored the internal state vectors at the hidden layer of the network. For each output vector and for each state vector the next corner preference was determined using the Euclidean distance metric. Thus the Euclidean distance metric assigned one of three symbolic abstract syntactic phrase categories to each output vector and one of eight state number identifiers to each state vector.

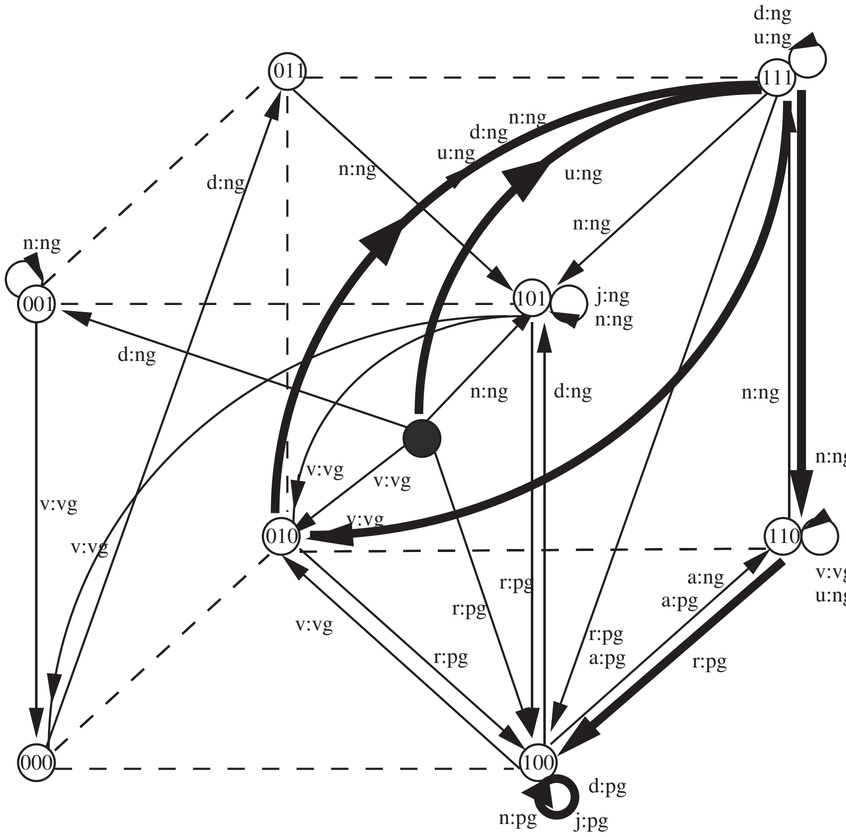


*Fig. 12.* Transducer extraction from a recurrent network for the example sentence "That (u:ng) is (v:vg) the (d:ng) Thursday (n:ng) after (r:pg) Easter (n:ng)".

Figure 12 shows the knowledge learned by the network as an extracted symbolic transducer. The corner nodes represent the eight strict states, the center node represents the start state of the transducer. At the edges we find the symbols for the single transductions. Input and output categories are separated by a colon, e.g. $d : ng$ means that - starting from the source state of this edge - a determiner preference $d$ is assigned to a noun group preference $ng$ and the transduction is made to the end state of this edge. In the extracted transducer we can see some clear regularities at certain states. For instance, the transductions to state 100 are primarily responsible for the assignments to the prepositional group $pg$. Other examples are the transductions to state 010 and to state 000, which are primarily responsible for the verbal group ($vg$) assignment. Furthermore, figure 12 shows the example transductions for the sentence "That is the Thursday after Easter". Beginning with the start state at the center, we see the transduction $u : ng$ for the word "That" which assigns the noun group $ng$ to the pronoun $u$. Then, $v : vg$ assigns a verb group $vg$ to the verb "is". Then the transductions $d : ng$ $n : ng$ assign the noun group $ng$ to "the Thursday". Finally the transductions $r : pg$ $n : pg$ assign the prepositional group $pg$ to the sequence "after Easter". Different abstract syntactic categories ($ng$, $pg$) can be assigned to the same category ($n$) depending on the learned previous context.

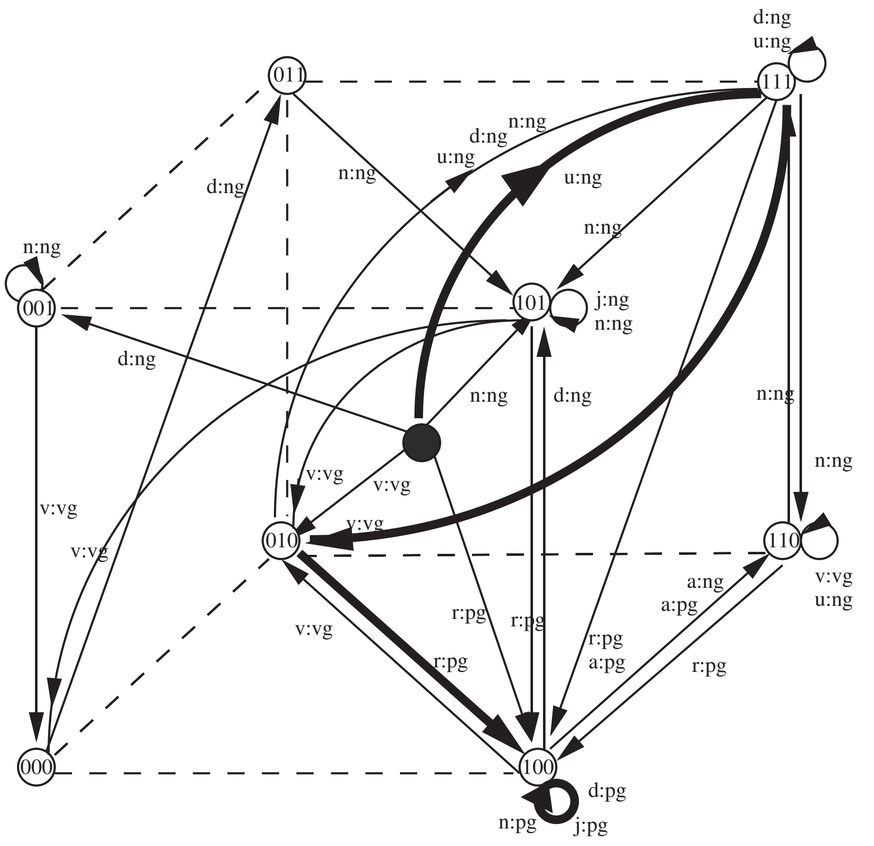


*Fig. 13.* Transducer extraction from a recurrent network for the example sentence "I (u:ng) thought (v:vg) in (r:pg) the (d:pg) next (j:pg) week (n:pg)".

More detailed (less detailed) transducers can be obtained if the state and output vectors are mapped to more (fewer) nodes. Thus, the general abstraction level of such a symbolic transducer can be quite variable. The symbolic transducer represents an abstraction of the detailed network knowledge but this abstraction also hides some of the numerical complexity and allows a direct symbolic interpretation which provides a summary of the network behavior.

To give an example, figure 13 shows the transductions for the example sentence "I thought in the next week". Beginning with the start state at the center, we see the transduction $u : ng$ for the word "I", which assigns the noun group $ng$ to the pronoun $u$. Then, $v : vg$ assigns a verb group $vg$ to the verb "thought". Finally the transductions $r : pg$ $d : pg$ $j : pg$ $n : pg$ assign the prepositional group "pg" to the word sequence "in the next week". One advantage of this transducer extraction is the higher abstraction level used for the representations of the recurrent network which leads to a better understanding of its function. The original network contains more detailed knowledge in the numerical weights and activations, but it is not possible to see the declarative sequential symbolic knowledge which this network represents. The extraction of a symbolic transducer allows a better understanding of the learned sequential knowledge which is represented in a more explicit manner.

## 8. Discussion and Analysis

### 8.1. *Comparison of knowledge extraction techniques*

There has been some previous work on using individual techniques in isolation for interpreting neural networks and extracting structural knowledge from them. In this paper, we have analyzed five such different techniques using the same trained network in order to interpret the network knowledge. Such extensive comparisons of detailed network knowledge are needed in order to gain a better understanding of the knowledge extraction represented in neural networks.

We have also introduced two new techniques here: dynamic learning analysis and transducer extraction. Dynamic learning analysis examines the formation and development of categories over time during learning. Thus, it provides a much deeper understanding of *how* the neural network arrives at its learned representation. Transducer extraction was developed to represent the sequential processing in a recurrent network at a higher level of abstraction.

In general, we found that different interpretation techniques provide different views of the knowledge contained in a neural network. Thus, there is not *a* single best technique for all different aspects of knowledge extraction. The use of a particular technique depends rather on the requirements of the interpretation. In table 2, we illustrate and summarize the general properties of the five different techniques.

Dynamic Learning Analysis (DLA) is based on the output representations and provides a high level of understanding based on these known output representations. This technique is easy to interpret and can be used with other network types. On the other hand, it does not particularly support recurrent networks, symbolic integration, and flexible knowledge structuring. Furthermore, structural relationships cannot be extracted.

Transducer extraction (TE) is a new technique which uses output representations as well as internal activations. The main advantages of this technique are the high level of understanding in the form of an extracted symbolic transducer, the specific support for the sequentiality of recurrent networks and the possibility for extracting structural relationships. Such an extracted transducer can be integrated with other symbolic knowledge, e.g. other coded symbolic transducers. Furthermore, different transducers can be generated with flexibility, based on the number of states used in the internal activation layer. This leads to a relatively straightforward interpretation of the network involved compared to the other techniques, but it also requires the additional effort of extracting this symbolic transducer from the internal activations and the output representations.

If we compare DLA and TE with WA, HAA, and CAA, we can see that DLA and TE are techniques that specifically provide *high level interpretations for dynamic learning and processing.* We argue that WA, HAA, and CAA are techniques with a tendency towards a general, detailed, but low-level interpretation. DLA and TE, however, are techniques for specialized, high-level, dynamic interpretation. Focusing on output interpretations and the dynamics of recurrent networks provides a new level of understanding. Whereas a lot of previous work has focused on low-levels of interpretation, we believe that in the future, higher levels of interpretation and knowledge extraction will be required.

### 8.2. *Related work on transducer extraction and related work*

Finite state automata and transducers have been widely used in various forms within traditional symbolic processing; e.g. [Hopcroft and Ullman, 1979]. Basically, automata and transducers are always in a certain context state and they analyze a certain word (symbol). Then they move to a new state and potentially generate a new word (symbol). By using changing states, it is possible to encode the sequential context.

Although finite automata or regular languages are not sufficient to describe all possible constructions of natural language completely (see e.g. [Winograd, 1983]), automata still constitute a central minimal requirement for the representation of natural language. Thus, they occupy the lowest level in the Chomsky hierarchy of languages [Hopcroft and Ullman, 1979]. Furthermore, it is possible to design efficient realizations of finite automata for different domains [Kaplan, 1995], e.g.

*Table 2.* Comparison of different knowledge extraction techniques: Dynamic Learning Analysis (DLA), Weight Analysis (WA), Hierarchical Activation Analysis (HAA), Component Activation Analysis (CAA), Transducer Extraction (TE). Further abbreviations: **A**ctivations/**W**eights/**O**utputs and **L**ow/**M**edium/**H**igh.

| | DLA | WA | HAA | CAA | TE |
|---|---|---|---|---|---|
| Network representations used | O | W | A | A | AO |
| General level of understanding | H | L | M | M | H |
| Specific support for recurrent networks | L | L | L | L | H |
| Degree of structural relationships | L | L | M | M | H |
| Integration with symbolic knowledge | L | L | M | M | H |
| Flexibility in level of knowledge structuring | L | L | M | M | H |
| Computational effort | L | L | M | M | M |
| Easiness of interpretation | H | L | M | M | H |
| Generality and portability to other networks | H | H | H | H | M |

for morphology, lexicon access, information extraction from sentences, syntactic tagging, etc.

Recurrent networks have the potential to learn a sequential preference mapping $f_p : I \times S \rightarrow O \times S$ automatically, based on input and output examples (see figure 11), whereas traditional Moore machines or Fuzzy-Sequential-Functions [Santos, 1973] involve manual encoding. It has been recently illustrated how SRN networks can emulate each symbolic Moore machine and each finite automaton [Kremer, 1995, Kremer, 1996]. It has also been shown however [Goudreau and Giles, 1995, Goudreau et al., 1994] that a recurrent network with only a single input layer, one context layer, and one output layer, the so-called Single-layer-first-order-network, is not sufficient for the realization of arbitrary finite automata.

In natural language processing, representations have to be at least as powerful as finite automata. Consequently, Single-layer-first-order-networks are not appropriate, which is why we have used SRN networks here. These recurrent networks contain finite transducers as a special case, but also support much more powerful properties based on their gradual $m$-dimensional preference representations. For instance, it could be shown that SRN networks can emulate certain restricted properties of a pushdown automaton, in particular the recursive representation of structures with a limited depth [Elman, 1991, Wiles and Elman, 1996].

Apart from traditional symbolic regular representations, gradual and learned representations can also be represented. Furthermore, the number of input, state, and output preferences is not necessarily finite. Therefore, neural preference Moore machines are more powerful than finite transducers. Our recurrent neural networks can be seen as learning $n \times m$ Fuzzy-transducers, thereby augmenting a simple finite symbolic transducer with respect to learning within a gradual preference space. From this perspective, symbolic knowledge is a special abstract region in a neural preference space.

An important line of research on automata and recurrent networks has been reported in [Giles et al., 1992, Goudreau and Giles, 1995, Tino et al., 1995]. Giles and colleagues studied both finite state automata and neural networks, but there are substantial differences with our research. They started often with a known finite state automaton, which was used to generate sequences for it. Then these sequences were used for training a second-order neural network. Using a partition algorithm, a finite state automaton was extracted from the network activations, minimized and compared to the original known finite state automaton. In this way, Giles and colleagues could study the computational properties of the extraction particularly well, but the finite state automata also frequently relied on relatively simple 1/0 sequences.

Our motivation and methodology is different from theirs in several respects. We assume that the initial finite state automaton or transducer is not known. Especially for real-world problems, the interesting case is the one where such an automaton is not known in advance. Whereas it is interesting for comparison and sequence generation, generating sequences with a finite state automaton already introduces certain regularities into the training set. Thus, sequence generation has an important influence on the learning behavior, something which we want to rule out. In fact, we are more interested in situations where we do not know the machine which has to be extracted. Especially with noisy real-world learning data, the underlying regularities may be quite disparate from regularly generated sequences.

Furthermore, the task of our networks is quite different. The second-order networks employed by Giles and colleagues are trained for recognition. The output layer represents state representations which can be fed back to the input layer at the next step. Our recurrent networks perform an assignment task, where a sequence of inputs is associated with a sequence of outputs. We are not determining whether a certain sequence belongs to a certain automaton, but what the simple flat structure of this sequence is. That is, we are interested in transducer extraction rather than recognizer extraction. In general, there are no designated final states in our networks, since the network - and the extracted symbolic transducer - produce output as long as input is provided. This transducer behavior is therefore quite different from the recognition performance reported in [Giles and Omlin, 1993], which is based on acceptors for artificial languages.

## 9. Conclusion

The main contribution of this paper is a particularly broad analysis of knowledge extraction for recurrent networks. In addition, we propose dynamic learning analysis and transducer extraction as two new dynamic interpretation techniques. Dynamic learning analysis provides a better understanding of *how* the network learns, while transducer extraction provides a better understanding of *what* the network represents. After learning, a conservative "lazy learning" strategy leads to connectionist representations which can be described as symbolic transducers. These transducers allow for a much better interpretation of the sequential network knowledge compared to the standard analysis using hierarchical clustering or Hinton diagrams. Weight analysis, cluster analysis, and principal component analysis are detailed but static. In contrast, our new method for extracting symbolic transducers can describe the learned classification performance much better, since transducer extraction considers the sequential character of the learned representations in a recurrent network and allows a better symbolic inspection. Possibilities for direct integration with symbolic classifiers can be explored in future work. We conclude that dynamic learning analysis and transducer extraction have a lot of potential for improved knowledge structuring based on recurrent networks.

## References


[Abe et al., 1993] Abe, S., Kayama, M., Takenaga, H., and Kitamura, T. (1993). Extracting algorithms from pattern classification neural networks. *Neural Networks*, 6(5):729–735.

[Andrews and Diederich, 1996] Andrews, R. and Diederich, J. (1996). *Rules and Networks*. Queensland University of Technology, Brisbane, Australia.

[Booth, 1967] Booth, T. L. (1967). *Sequential Machines and Automata Theory*. John Wiley, New York.

[Churchland and Sejnowski, 1992] Churchland, P. S. and Sejnowski, T. J. (1992). *The Computational Brain*. MIT Press, Cambridge, MA.

[Craven, 1996] Craven, M. W. (1996). *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, University of Wisconsin-Madison. PhD Thesis.

[Elman, 1991] Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–226.

[Elman, 1995] Elman, J. L. (1995). Language as a dynamical system. In Port, R. F. and van Gelder, T., editors, *Mind as motion: explorations in the dynamics of cognition*, pages 195–225. MIT, Cambridge, MA.

[Elman et al., 1996] Elman, J. L., Bates, E. A., Johnson, M. H., Karmiloff-Smith, A., Parisi, D., and Plunkett, K. (1996). *Rethinking Innateness*. MIT Press, Cambridge, MA.

[Giles et al., 1992] Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., and Lee, Y. C. (1992). Learning and extracted finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

[Giles and Omlin, 1993] Giles, C. L. and Omlin, C. W. (1993). Extraction, insertion and refinement of symbolic rules in dynamically driven recurrent neural networks. *Connection Science*, 5:307–337.

[Gorman and Sejnowski, 1988] Gorman, R. P. and Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89.

[Goudreau and Giles, 1995] Goudreau, M. W. and Giles, C. L. (1995). On recurrent neural networks and representing finite-state recognizers. In *Proceedings of the Third International Conference on Neural Networks*, pages 51–55.

[Goudreau et al., 1994] Goudreau, M. W., Giles, C. L., Chakradhar, S. T., and Chen, D. (1994). First-order vs. second-order single layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513.

[Hallam, 1995] Hallam, J., editor (1995). *Hybrid Problems, Hybrid Solutions — Proceedings of the $10^{th}$ Biennial Conference on AI and Cognitive Science (AISB-95)*, Amsterdam. (Sheffield, UK), IOS Press.

[Hinton, 1986] Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the $8^{th}$ Meeting of the Cognitive Science Society*.

[Hölldobler, 1990] Hölldobler, S. (1990). A structured connectionist unification algorithm. In *Proceedings of the National Conference of the American Association on Artificial Intelligence 90*, pages 587–593, Boston, MA.

[Hopcroft and Ullman, 1979] Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA.

[Kaplan, 1995] Kaplan, R. (1995). Finite state technology. In Cole, R. A., Mariani, J., Uszkoreit, H., Zaenen, A., Zue, V., Varile, G., and Zampolli, A., editors, *Survey of the State of the Art in Human Language Technology*, pages 419–422. NSF, EU.

[Kohavi, 1970] Kohavi, Z. (1970). *Switching and Finite Automata Theory*. McGrawHill, New York.

[Kremer, 1995] Kremer, S. C. (1995). On the computational power of Elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004.

[Kremer, 1996] Kremer, S. C. (1996). A theory of grammatical induction in the connectionist paradigm. Technical Report PhD dissertation, Dept. of Computing Science, University of Alberta, Edmonton.

[Kurfeß, 1991] Kurfeß, F. (1991). Unification on a connectionist simulator. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pages 471–476. North-Holland.

[Medsker, 1995] Medsker, L. R. (1995). *Hybrid Intelligent Systems*. Kluwer Academic Publishers, Boston.

[Omlin and Giles, 1996] Omlin, C. W. and Giles, C. L. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52.

[Santos, 1973] Santos, E. S. (1973). Fuzzy sequential functions. *Journal of Cybernetics*, 3(3):15–31.

[Shavlik, 1994] Shavlik, J. (1994). A framework for combining symbolic and neural learning. In Honavar, V. and Uhr, L., editors, *Artificial Intelligence and Neural Networks: Steps towards principled Integration*, pages 561–580. Academic Press, San Diego.

[Shields, 1987] Shields, M. W. (1987). *An Introduction to Automata Theory*. Blackwell Scientific Publications, London.

[Sperduti et al., 1995] Sperduti, A., Starita, A., and Goller, C. (1995). Learning distributed representations for the classifications of terms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 494–515, Montreal.

[Sun, 1995] Sun, R. (1995). Schemas, logics and neural assemblies. *Applied Intelligence*, 5:83–102.

[Tino et al., 1995] Tino, P., Horne, B. G., Giles, C. L., and Collingwood, P. C. (1995). Finite state machines and recurrent neural networks. Technical Report CS-TR-3396, University of Maryland, College Park.

[Wermter, 1995] Wermter, S. (1995). *Hybrid Connectionist Natural Language Processing*. Chapman and Hall, Thomson International, London, UK.

[Wermter, 1998] Wermter, S. (1998). The hybrid approach to artificial neural network-based language processing. In Dale, R., Moisl, H., and Somers, H., editors, *A Handbook of Natural Language Processing*. Marcel Dekker.

[Wermter, 1999] Wermter, S. (1999). Preference moore machines for neural fuzzy integration. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm.

[Wermter and Löchel, 1996] Wermter, S. and Löchel, M. (1996). Learning dialog act processing. In *Proceedings of the International Conference on Computational Linguistics*, pages 740–745, Copenhagen, Denmark.

[Wermter and Meurer, 1997] Wermter, S. and Meurer, M. (1997). Building lexical representations dynamically using artificial neural networks. In *Proceedings of the International Conference of the Cognitive Science Society*, pages 802–807, Stanford.

[Wermter et al., 1996] Wermter, S., Riloff, E., and Scheler, G. (1996). *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*. Springer, Berlin.

[Wermter and Weber, 1997] Wermter, S. and Weber, V. (1997). SCREEN: Learning a flat syntactic and semantic spoken language analysis using artificial neural networks. *Journal of Artificial Intelligence Research*, 6(1):35–85.

[Wiles and Elman, 1996] Wiles, J. and Elman, J. (1996). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the AAAI Workshop on Computational Cognitive Modeling: Source of the Power*, Portland, Oregon.

[Winograd, 1983] Winograd, T. (1983). *Language as a Cognitive Process*. Addison-Wesley, Reading, MA.