



## 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.

# Hidden Markov model interpretations of neural networks

---

## Abstract

It is well known that simple recurrent networks can learn regular languages, that is the languages from finite state automata (Cleeremans et al., 1989). In the last decade also the reverse process has become increasingly popular, that is extracting rules from neural networks in order to get interpretable automata. Extracting rules from such neural networks is generally done by partitioning the hidden state space of the network. Hidden Markov models, alternatively called stochastic finite automata, can be estimated from a series of responses from a neural network. The difference with the approach used by Cleeremans et al. (1989) is that it is not necessary to use the hidden state activities of the network to extract the finite state automaton: only the input-output relations of the network are necessary for fitting a hidden Markov model. Nevertheless equivalent automata can be extracted. Hidden Markov models can thus be used to provide interpretations for the representations of neural networks, such as the simple recurrent network for implicit learning.

## 3.1 Introduction

Modeling cognitive tasks with (back-propagation) neural networks has become increasingly popular over the last decade and neural networks are successful in different fields such as memory, vision and learning. This success is, at least partially, due to the ease of use of neural networks as a means of modelling. It is fairly easy and straightforward to apply neural networks to a given cognitive task that one is interested in: simply feed the network with the input a subject would get in an experiment and give it feedback about its errors. Another marked advantage of neural networks over classical models is that learning is a natural part of the system. Not only can a neural network represent certain cognitive abilities, it also shows how such representations are built from scratch during the learning process.

One of the problems with neural networks is that, although they are successful at replicating certain data from human subjects, they do not really show us how this is accomplished. One could defend the view that in using a neural network for a cognitive task one just creates another black box, just as the human mind one wants to model, that, accidentally, performs the same task. Obviously, this extreme view is not held by many cognitive scientists, but the core of the argument is not to be taken lightly. More specifically, the weights of a neural network and the internal states it has during responding to a certain input are, in general, hard to interpret.

Several researchers have tried to overcome this problem by interpreting the dynamics of neural networks in terms of finite state automata (Giles et al., 1992; Cleeremans et al., 1989; Omlin and Giles, 1996; Tino and Köteles, 2000). In this chapter we also extract finite state automata from neural networks. The main difference between our approach and earlier approaches is that we only use the overt behavior of the network, that is input/output pairs, to build a finite state representation instead of dissecting the hidden state space of the neural network in one way or another. The remainder of this chapter is organized as follows: first we outline the general approach used by above authors to extract finite state machines from networks. Then we contrast this with our approach which is based on hidden Markov models. In the next two sections we show simulation results with hidden Markov model extraction from recurrent neural networks. Finally we discuss our results and see how they can shed some light on interpretational problems that exist with neural networks.

### 3.2 Extracting finite state automata from networks

The main reason for extracting finite state automata (FSA) from neural networks is for purposes of interpretation. An FSA abstracts the activities of the hidden state space of a neural network into a set of clearly expressible rules. These rules are stated in the form of (stochastic) dependencies between consecutive stimuli/responses.

The common starting point for extracting FSAs from neural networks is to record the activities of the hidden state space during a training or testing session of the network. Suppose that the network has  $N$  hidden units, then the result of this recording is a trajectory through an  $N$ -dimensional state space. From here on, there are several ways to continue.

The first approach to interpreting neural network behavior is based on some form of clustering. Again the starting point is a recorded trajectory through the state space of the hidden nodes of a neural network that has learned a finite state grammar. Cleeremans et al. (1989) use clustering to identify states of a FSA that is represented by a neural network. Cleeremans et al. (1989) however do not actually construct FSAs from their trained neural networks but use the clusters to draw inferences about the internal representations of the networks.

Giles et al. (1992) begin with partitioning the hidden state space of the network into hypercubes. These hypercubes are numbered and they are the states of the FSA<sup>1</sup>. Transitions between states in the FSA are added whenever there is a corresponding transition in the recorded state space trajectory. The transition is labeled with the input which is fed to the network at that point in the trajectory. The derived FSA is then minimized using standard algorithms (Hopcroft and Ullman, 1979).

Tino and Köteles (2000) also use a form of clustering on the state space trajectory. In particular they use a procedure similar to K-means clustering on the

---

<sup>1</sup>Not all hypercubes actually become states in the FSA because the recorded trajectory does not necessarily pass through all the possible hypercubes.

hidden state space activations that are recorded during a test run of the network. In addition to Giles et al. (1992), the FSAs that Tino and Köteles (2000) extract from their network are stochastic FSAs: transitions between states have probabilities associated with them.

In this study we use hidden Markov models to extract finite state representations from (recurrent) neural networks. This approach differs from earlier approaches in several respects. First, we do not use recorded hidden state space trajectories but only input/output pairs from the trained networks. Because of this we do not have to make any assumptions about this state space as, for example, in K-means clustering where the number of clusters has to be decided beforehand. The optimal number of states of the hidden Markov model can be determined by several selection criteria that we discuss later. An advantage over the approach by Giles et al. (1992) is that no minimization is required after extraction of the (stochastic) FSA.

### 3.3 Recurrent neural networks

In this chapter we chose to study simple recurrent networks (SRN) for several reasons. Firstly, SRNs have been shown to be able to learn languages generated by finite state automata (Cleeremans et al., 1989). Secondly, and more importantly to us, SRNs have been used successfully in modeling cognitive tasks in the domain of language learning (Elman, 1990) and in implicit learning (Cleeremans and McClelland, 1991). As Elman (1990) points out, SRNs are successful in these domains because the recurrent connections of the network allow it to have a *memory* for past events, in casu stimuli that have been presented to the network.

The architecture of the simple recurrent network we used is from Elman (1990). The network is shown in Figure 3.1. The network consists of three layers, input, hidden and output nodes. In addition the network has a context or recurrent layer that keeps a copy of the hidden unit activity at time  $t - 1$ . The dynamics of the network are given by simple feedforward relations and learning is done with the back-propagation rule (Elman, 1990).

Because of its recurrent connections the SRN keeps a copy of its internal representations at  $t - 1$ , that is, its representation of the stimulus that was presented to it at the former trial. This kind of memory makes the SRN especially suitable to learn sequentially structured data.

### 3.4 Hidden Markov models

Hidden Markov models (HMM) are mainly used in the area of speech recognition (cf. Schmidbauer et al., 1993; Chien and Wang, 1997; Rabiner, 1989). There are a rare applications in other fields such as psychology (Yang et al., 1997) and physiology Becker et al. (1994). Another recent application is in the extraction of propositions from written text (Durbin et al., 2000)<sup>2</sup>.

---

<sup>2</sup>It should be noted that hidden Markov models are identical to latent Markov models, which have quite a few applications in psychology (for examples see Arminger et al., 1995, see also chapter 5). However, there is an important difference between these psychological applications

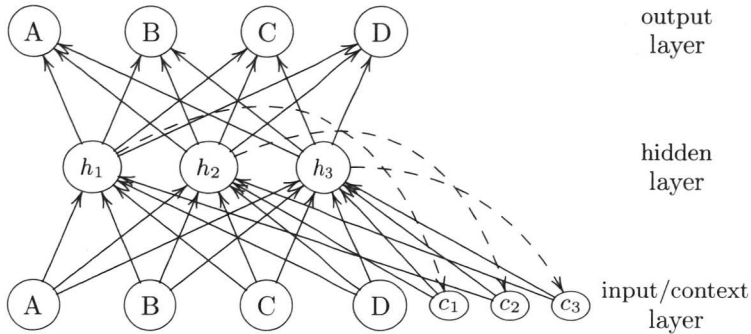


Figure 3.1: Simple recurrent network as used by Elman (1990). In this example the network has four input nodes that each correspond to a different input symbol. The output of the network is determined by feedforward relations. The network is trained with back-propagation. See the text for more details.

HMMs are basically stochastic finite state automata<sup>3</sup>. Formally, an HMM consists of the following (notations are adapted from Rabiner, 1989):

1. a set of states  $S_i, i = 1, \dots, N$
2. a set  $V$  of observation symbols  $V_k, k = 1, \dots, M$
3. a matrix  $A$  of transition probabilities  $a_{ij}$  for moving from state  $S_i$  to state  $S_j$
4. a matrix  $B$  of observation probabilities  $b_j(k)$  of observing symbol  $V_k$  while being in state  $S_j$
5. a vector  $\pi$  of initial state probabilities  $\pi_i$  corresponding to the probability of starting in state  $S_i$  at  $t = 1$

The equations describing the dynamics of the model are as follows:

$$\begin{aligned} S_{t+1} &= A S_t + \zeta_{t+1} \\ O_{t+1} &= B S_t + \xi_{t+1}, \end{aligned}$$

where  $S_t$  is the hidden process and  $O_t$  is the observed process;  $\zeta_{t+1}$  and  $\xi_{t+1}$  are zero mean martingale increment processes, cf. Elliott et al. (1995, p. 20) for further

---

and the others, which is in data sets involved. In speech recognition, as in the application of HMMs discussed here, there are typically long series of observations, say with  $T > 100$ . In contrast, in psychological applications shorter sequences of observations are usual. Although this is not a principled difference, it does have profound consequences for estimation of parameters and their standard errors (see chapter 4 and 5). Possibly this difference also accounts partially for the fact that the literature on latent Markov models and hidden Markov models is largely separate.

<sup>3</sup>HMMs come in many different guises and under many different names: latent Markov models, (tractable) Boltzmann machines, (tractable) belief networks and (tractable) state space models to name but a few.

details<sup>4</sup>.

The set of parameters  $\lambda = (A, B, \pi)$  can be used to express certain notions about HMMs. For example, the HMM can be used to generate a series of observations and given the above parameters it is easy to compute the likelihood  $L(O|\lambda)$  of such a series by taking the product of the individual probabilities of moving from one state to the next and producing the observations  $O_t, t = 1, \dots, T, O_t \in V$ , in those states. More interestingly, given a series of observations, it is possible to compute the parameter values of an HMM that optimize this likelihood. Rabiner (1989) describes a version of the EM algorithm to maximize the likelihood  $L(O|\lambda)$ .

## 3.5 Simulations

### 3.5.1 Method

With the FSA depicted in Figure 3.2 we produced a string of 420.000 symbols long used for training of the network. This length of the training sequence was taken from Cleeremans et al. (1989). Apparently the network needs such a long sequence to learn the grammar.

During training the network is presented with consecutive symbols of this string. Its task is to predict the next symbol in the string, hence the error between this target and the actual prediction of the network is backpropagated. The SRN had 3 hidden nodes. The weights of the network had random starting values between -0.5 and +0.5; the learnrate was set to 0.01 and the momentum term to 0.5. The criterion for learning was that the (normalized) activation of the node corresponding with the next symbol was over 0.3.

At regular intervals during training, the SRN was made to predict a string of symbols by itself. This was done with the following steps, 1) the network was presented with the begin symbol B and activity was fed forward through the connections; 2) the output activations of the network were normalized; 3) the normalized activations were treated as a probability distribution from which the next symbol was drawn; 4) this next symbol was then given to the network as input; 5) these steps were repeated until a sequence of 1000 symbols was generated.

The so generated sequences are used to extract FSAs. That is, HMMs with increasing numbers of states are fitted on these sequences until a best model is found. Which model is the best model is determined by a number of factors. First, the loglikelihood: if the loglikelihood does not increase anymore when the number of states increases, this means that adding more states to the HMM does not make for a better model. The loglikelihood however does not differentiate very well between models that are very similar or have the same number of states. As a second fitmeasure for the models, a prediction error is determined (see also chapter 5, for

<sup>4</sup>The  $\zeta_{t+1}$  and  $\xi_{t+1}$  in these equations are residual terms, similar to those that occur in regression equations for example. In computing the likelihood of discrete hidden Markov models these terms drop out. Intuitively the reason for this is simple: the residual terms give rise to noise. However, the noise can not be distinguished from the normal process outputs, except in a confirmatory analysis. When fitting an HMM to an observed sequence, with noise added to that sequence, will result in close to zero parameter estimates that were not part of the original model; i.e. the noise is simply absorbed in the parameter estimation process. In confirmatory analyses it may be possible to explicitly estimate residual terms.

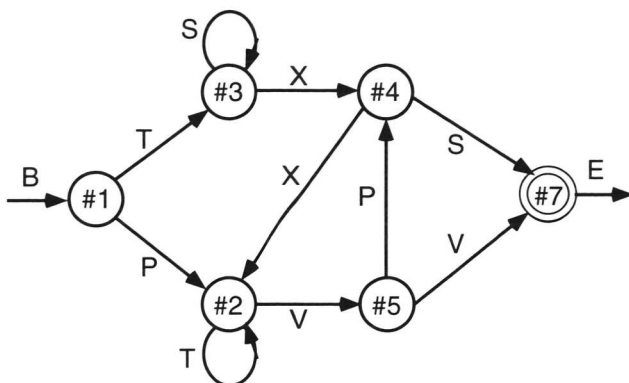


Figure 3.2: Finite state grammar which Reber(1967) used in his implicit learning experiments. Cleeremans et al. (1989) also used this grammar to train their networks. Sentences are made by this grammar by starting with the begin symbol B. With equal probability one of the arcs leaving state # 1 is chosen and the corresponding symbol is, is the next symbol in the string. This process of selecting arcs and noting the corresponding symbol continues until the end state # 7 is reached whence the process starts in state # 1.

a more complete description of this measure). This prediction error consists of comparing the number of symbols, pairs of symbols, triples of symbols et cetera in the sequence that the model is fitted on, and the expected numbers for these same symbols, pairs of symbols et cetera. For each of these categories (pairs, triples) a  $\chi^2$  distance can be computed between the actual sequence and the model predictions. A third fitmeasure that is used, is the entropy measure of complexity for symbolic sequences.

Entropy is a measure of information density that is designed for languages or sets of sentences<sup>5</sup>. The definition is as follows (Shannon, 1948; Lind and Marcus, 1995; Tino and Köteles, 2000):

$$h = \lim_{n \rightarrow \infty} h_n,$$

where  $h_n$  is the entropy rate which is defined as:

$$h_n = \sum_{\omega \in S_n} -p(\omega) \log(p(\omega)),$$

where  $\omega$  is a word (string of symbols) and  $S_n$  is the collection of allowed sentences of length  $n$ . Entropy is an interesting invariant of languages/symbolic sequences

<sup>5</sup>Entropy can be seen as a generalization of the above-mentioned prediction error. Whereas the  $\chi^2$  prediction error is computed for fixed n-tuples of observations, entropy concerns the distribution of all n-tuples simultaneously. The entropy is an invariant of such a distribution. As such, entropy provides a distance measure between the intended distribution, i.e. the observed distribution, and the modeled distribution.

because it measures a global property of such sequences. The reason for using entropy is that we can hardly expect a neural network to predict a sequence of 420.000 symbols exactly. Moreover, we are interested whether the SRN captures the general rules of the grammar and not the exact order of a given training sequence. Since entropy is a global invariant of sequences produced by, for example, finite state languages, we expect the sequences that the network generates to have the same entropy as the grammar has.

To be able to compute entropy we would need the total language that either a HMM or SRN can produce. In practice this is never the case so we need an approximation. The approximation we use is the so-called Lempel-Ziv entropy (LZE). Suppose we have a string  $S=ABDACDABDADACABDC$ . We parse this string into substrings by noting the smallest substring that has not yet occurred. Since we start with an empty set of substrings  $A$  is the first substring that is parsed. Continuing this procedure until the end of the string is reached results in the following set of substrings:  $A, B, D, AC, DA, BD, AD, ACA, BDC$ . The Lempel-Ziv estimate of the entropy of the string is:

$$h_{LZ} = \frac{c(S) \log c(S)}{N},$$

where  $c(S)$  is the number of substrings resulting from the parsing procedure above and  $N$  is the total length of the string  $S$  (Tino and Köteles, 2000).

How do we use this as a fitmeasure? By comparing the LZE of the sequence on which a HMM is fitted with the LZE of the HMM itself we know how closely the behavior of the HMM matches that of the sequence and hence of the SRN that produced that sequence.

### 3.5.2 Results

As expected, the SRN learned the grammar perfectly. We trained 20 SRNs with random starting values, 2 of which came to represent the grammar perfectly within 2 epochs of training. At twelve points before, during and after training we recorded a sequence of predictions by the SRN. On each of these sequences we fitted HMMs. In Figure 3.3 at the bottom the numbers of states of these twelve models are provided.

The final HMM, that is the HMM that is fitted on the final sequence generated by the SRN, has an LZE of 1.709 which is very close to the LZE of the true model which is 1.704. The consecutive models have nicely decreasing values for their LZEs which in the end converge to the entropy of the true model.

## 3.6 Discussion

HMMs can be successfully used to extract automata from networks. This can be done without having to record the hidden state space activities of the neural network<sup>6</sup>. This is an advantage since it means that the same procedure of fitting

---

<sup>6</sup>Note of course that it is still possible to do so and then link these recorded activities to the hidden states of the HMM.

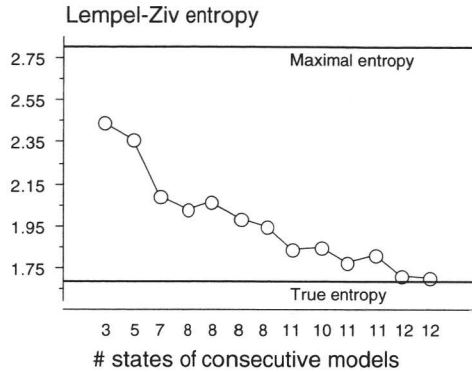


Figure 3.3: At the bottom of this figure is the number of states of the consecutive models. The first model, that is the model before the start of training, has three states, the last model has 12 states, exactly the same as the true model, that is the model that was used to generate the data. For details on the entropy see the text.

HMMs can be used for extracting models from data generated by human subjects. This can be done for example on generation data that subjects produce in implicit learning experiments (see chapters 5 and 7 for applications of hidden Markov models for analyzing implicit learning data).

Once HMMs are extracted from recurrent neural networks, they provide interpretable models of the network behavior. The description of the behavior of a network with a hidden Markov model gives clearly expressible rules that describe its behavior.

Since fitting of HMMs is done with the maximum likelihood method, all advantages of that approach are available: exact inference about parameter values, standard errors for the parameter estimates (see Visser et al., 2000, and chapter 4), the possibility of fitting HMMs with (equality) constraints on the parameters et cetera (see also chapter 5). Another advantage resulting from the maximum likelihood framework is the availability of fitmeasures. In comparison with the partitioning method from Giles et al. (1992) this has the advantage that minimization is done automatically. When fitting HMMs one will never end up with extremely large models that then have to be minimized.

The use of HMMs for extracting FSAs is closely related to the approach by (Tino and Köteles, 2000). Using HMMs expands on their method by having maximum likelihood estimation of parameters. This results in having fitmeasures that can be used for model selection and determining goodness-of-fit.