A Study of Sequence Processing on Neural Networks

Luyuan Fang and William H. Wilson

Discipline of Computer Science, Flinders University of South Australia
Bedford Park, S.A. 5042, AUSTRALIA
Tao Li
Department of Computer Science, Monash University,
Clayton, VIC 3168, AUSTRALIA

Abstract. Neural networks for storing and for classifying sequences are studied in this paper. A model of sequential neural networks is proposed here. This model can be used as a classifier for a window and as a memory for storage and retrieval of sequences. Some experiments have been conducted.

Introduction

The recent resurgence of neural network research has brought together many disciplines in cognitive science such as cognitive psychology, artificial intelligence, computer science, neurophysiology, and cybernetics. This is not without any special reason. Neural network research is the ground for all these disciplines and ties together many issues about cognition. Many models [3,4,6,7,10,12,13] of neural networks have been proposed. Neural networks have also been applied to solving combinatorial problems [8,9,14]. Neural networks are expected to bring a significant impact to cognitive science and will modify our view on computation.

While most of the research concentrates on the important issues of cognitive science and neural network applications, one aspect has been neglected. This is the sequential presentation problem. Most of the neural networks take a finite size input (viewed as a window). These networks have difficulties handling variable size or conceptually infinite input sequence. Some researchers [1,11] have considered the sequential presentation problem. But there remains much research work to be accomplished in this area. There seem to have three main aspects in the study of sequential neural networks: 1) grammar learning and sequence identification, 2) sequence neural memory, and 3) partial match retrieval.

Grammar Learning. It is often needed to classify sequences of patterns. This has always been an interesting problem. Sequential patterns can be of variable, and sometimes infinite, lengths. An example of sequential pattern classification is the parity problem in which a network is required to output a 1 if the number of 1's in the input sequence is even and to output a 0 otherwise. One would like to teach a neural network to learn to classify odd-even input patterns. Here the emphasis is placed on learning rather than the traditional way of designing finite state automata. The neural network is expected to generalize from the given examples during learning.

Sequence Memory. Kohonen [11] proposed another sequential memory model. The main purpose of his sequential memory is to store sequences of patterns and to recall these sequences at a later time. Therefore the neural network serves the purpose of a memory system. The difference between this sequence memory and other associative memories is that the order of input/output patterns for the sequence memory must be preserved. Again the emphasis is placed on the learning aspect of the neural network. One would like to train a network to memorize a sequence such as the music of a song and to be able to recall the musical sequence when the name of the song is given. In this case, the music notes represents the sequence to be stored and the name of the song can be viewed as the background (as in Kohonen's model).

Partial Match Retrieval. This involves the storage of some sequences and the retrieval of the sequences at a later time when only a small part of a sequence is given. This happens very often in our life. For example, humans learn to sing songs. The music and the words of a song often contain repetitions of some segments. When one hears a segment of music, he/she can immediately recognize the song. This is partial match retrieval of sequences. This is an area in which little research has been done. Although this seems only a part of the problem of sequence storage and retrieval as discussed above, it deserves special attention due to the commonality of such phenomenon.

A Sequential Neural Machine Model

Teaching a neural network to learn to recognize the important features of sequences is very important to neural network research. But this area of research has not been well-explored yet. A model of sequential neural network was recently proposed by Gallant [1]. In this section, Gallant's sequential neural network is first examined. Based on that model we propose a sequential neural machine model.

Gallant's Sequential Associative Memory

Stephen Gallant proposed an interesting model of sequential neural network which consists of highly interconnected subnetworks called Sequential Associative Memories (SAM's) and some trainable cells having modifiable connection weights. His model consists of a number of SAMs and some three states perceptron units. A SAM consists of a number of perceptrons, the weights of which are generated at random and remain fixed. SAM cells are used to determine a state for recalling previous inputs and outputs and to enable the modifiable cells in the network to learn difficult tasks. The structure of this model is shown in Figure 1.

![Figure 1. The sequential network model of Gallant](Image)

An easy learning algorithm, the pocket learning algorithm [2], was used to modify the connection weights of the trainable units. The network was able to learn to solve the parity problem, to do arithmetics of variable length operands, etc.
A Sequential Neural Machine for Classification

The reason for using random networks of SAMs was not explained in [1]. However, the SAM model might have difficulty learning sequences with repeated segments such as in music. This is because the connections and weights of SAMs are fixed. Hence the internal states generated by the SAMs are also fixed. Kohonen's sequence memory also has the same problem.

A model of sequential neural networks is proposed here. This model consists of two subnetworks, a state transition network and an output network. Each of the subnetworks contains three layers of perceptrons with modifiable connections and weights. The outputs of the state subnetwork are connected to the delay elements D1, D2, ..., Dx. The state subnetwork also takes some of its inputs from the delay elements. Figure 2 illustrates the new model.

Figure 2. A Sequential Neural Network Model.

This model certainly looks like a typical sequential circuit. It should also behave like a sequential circuit. However, there is an important difference between this model and the traditional sequential circuits. That is, this neural network learns to recognize (or to memorize) sequences in contrast to a traditional finite state machine which is designed to perform certain tasks. Nothing was specified for a sequential neural machine at the design time!

To use the sequential neural network for classification, the network must first be trained with a set of sample sequences. During the training process, the connection weights are adjusted according to the given input sequences and corresponding output sequences. During the classification process, a sequence is presented to the network as input. The network produces desired outputs by extracting the important features of the input sequence. Different outputs are generated for sequences with different features. Hence a trained network can classify input sequences by important features. During the classification process, the algorithms of subsections 2.3 and 2.4 (discussed next) will be employed to compute cell activations and network states and outputs.

Computing Activations for Cells

Let \( C_i \) denote the activation of the \( i \)th perceptron cell in the network, where \( C_i \) can assume the values \(+1, -1, 0\) corresponding to \{true, false, unknown\}. If the cell \( C_i \) has \( m \) inputs, then \( C_i \) is computed as

\[
C_i(t+\delta_t) = \begin{cases} +1 & \text{if } \sum_{j=1}^{m} C_j(t) W_{ij} > 0, \\ -1 & \text{if } \sum_{j=1}^{m} C_j(t) W_{ij} < 0, \\ 0 & \text{if } \sum_{j=1}^{m} C_j(t) W_{ij} = 0, 
\end{cases}
\]

where \( W_{ij} \) is the connection weight from the \( j \)th cell to the \( i \)th cell. The threshold is assumed to be 0 for every perceptron unit in the network. For computational reasons, we assume, without loss of generality, an indexing scheme for cells in which each cell can be connected to only those cells with lower index numbers.

Computing the Next States and Outputs

To produce deterministic sequential behaviour, the network must use synchronous update or iterative update with fixed order. Using the cell indexing scheme given in the previous subsection, synchronous and iterative updates are equivalent. Assuming that a network consists of \( n \) cells \( C_1, ..., C_n \) and \( p \) inputs \( I_1, ..., I_p \), the iterative update algorithm is described below.

FOR \( i = 1 \) TO \( n \) DO
BEGIN
\( C_i \leftarrow 0; \)
FOR \( j = 0 \) TO \( p - 1 \) DO
\( C_i \leftarrow C_i + I_{j+1} W_{ij}; \)
FOR \( j = p \) TO \( n - 1 \) DO
\( C_i \leftarrow C_i + C_{j+1} W_{ij}; \)
END.

In the above, a cell is assumed to be connected to all inputs and all cells with lower indices.

This iterative update algorithm is used to compute the next state and the output for the network. When a new network input vector \( N \) is given, the activations of the cells are updated and the network state and output are changed. This is accomplished in four steps as follows.

1) Form the new input vector \( I \) for the state subnetwork by combining the current network input vector \( N \) with the contents of the delay units;

2) Compute a new state from the new state input vector \( I \) by applying the iterative update algorithm to the state subnetwork and store the new state in the delay units;

3) Form the new input vector \( N \) for the output subnetwork by combining the current network input vector \( N \) with the current contents of the delay units;

4) Compute the new output from the vector \( N \) by applying the iterative update algorithm to the output subnetwork.

To learn and to recognize sequences, the network must be presented with sequential input patterns. During the recognition process, the procedure described above will be used to generate next states to compute outputs for the network.

Easy Learning Algorithm

In this section we study the learning rules for the sequential neural network model. The learning rule for individual connection weights and the learning algorithm for the network will be studied here. The learning algorithm presented here is an "easy" one. For an easy learning algorithm, the desired activation of every cell, whether an input/output cell or an internal cell, must be specified.
Learning of Connection Weights

A slightly modified perceptron learning procedure is adopted here. Integer weights are assumed for connection weights. Let \( \vec{E}, \vec{C} \) be a training pattern for a cell \( \mathbf{C} \) with \( m \) inputs, where \( \mathbf{E} = (E_1, E_2, \ldots, E_m) \) is the given input vector and \( \mathbf{C} \) is the desired activation. Also let \( \mathbf{W}(t) = (W_1, W_2, \ldots, W_m) \) be the current weight vector for the cell \( \mathbf{C} \). Then the weight vector \( \mathbf{W} \) is updated during learning according to the following rules.

\[
\text{if } \mathbf{E} \cdot \mathbf{W}(t) \text{ and } \mathbf{C} \text{ have the same sign, } \mathbf{W}(t+\delta t) = \mathbf{W}(t); \\
\text{otherwise, } \mathbf{W}(t+\delta t) = \mathbf{W}(t) + \epsilon \mathbf{E}.
\]

(1)

\( \mathbf{E} \cdot \mathbf{W} \) is an inner product. When \( \mathbf{E} \cdot \mathbf{W} \) and \( \mathbf{C} \) have the same sign, the network correctly classifies the pattern. Thus there is no need to modify the weights. Otherwise, the weights are adjusted accordingly.

If the set of sample patterns is known to be linearly separable, rule (1) above can be repeatedly applied until a sample pattern is correctly classified. In other words, the rule \( \mathbf{W} = \mathbf{W} + \epsilon \mathbf{E} \) is repeatedly applied until \( \mathbf{E} \cdot \mathbf{W} \) and \( \mathbf{C} \) have the same sign. Then \( \mathbf{W}(t+\delta t) \) is set to \( \mathbf{W} \). This modified rule is used more often in practice.

A Simple Learning Algorithm

A simple learning algorithm for the sequential neural network is presented here. This algorithm is a modified version of the pocket learning algorithm [2]. In order to train a network, sample sequences and corresponding outputs must be specified. Let the set \( S \) of sample sequences be \( S_1, S_2, \ldots, S_l \) and the corresponding lengths be \( k_1, k_2, \ldots, k_l \). The set of sample sequences can be listed in the following form.

\[
\begin{align*}
R_1 & \quad R_2 & \quad \text{Rmax} \\
S_1 & \quad S_{11} & \quad S_{12} & \ldots & \quad S_{1k_1} \\
S_2 & \quad S_{21} & \quad S_{22} & \ldots & \quad S_{2k_2} \\
\vdots & & & & \ddots \\
S_l & \quad S_{l1} & \quad S_{l2} & \ldots & \quad S_{lk_l}
\end{align*}
\]

where \( \text{Rmax} = \text{max}(k_1, k_2, \ldots, k_l) \) and each \( S_{ij} \) is an input vector. This has the form of a matrix. The network can learn the patterns row-wise or column-wise.

In row-wise learning, a complete sequence \( S_i \) is presented to the network each time. For classification purpose, the initial state \( D_0 \) is the same for every input sequence. Let \( D_ij \) be the desired network state when input pattern \( S_{ij} \) is presented and \( Y_{ij} \) be the corresponding output pattern. Row-wise learning algorithm is described as follows

1) Randomly select a sequence \( S_i \) from the set \( S \) of sample sequences and delete \( S_i \) from \( S \);
2) FOR \( j=1 \) TO \( k_i \) DO
   2a) Present \( S_{ij} \) to the network;
   2b) Form an input vector \( I \) for the state subnetwork, where \( I = S_{ij} \cdot D_{ij} \);
   2c) Specify the desired activation for each cell in the state subnetwork (by combining \( D_{ij} \) and the internal cell activations);
   2d) Apply cell learning procedure to adjust connection weights in the state subnetwork; test run (using iterative update) with input vector \( I \) to see if the output is \( D_{ij} \);
   2e) Form an input vector \( 0 \) for the output subnetwork, where \( 0 = S_{ij} \cdot D_{ij} \);
   2f) Specify the desired activation for each cell in the output subnetwork (by combining \( Y_{ij} \) and the internal cell activations);
2g) Apply cell learning procedure to adjust connection weights in the output subnetwork; test run (using iterative update) with \( 0 \) to see if the output is \( Y_{ij} \);
3) If either of the tests in 2e and 2g fails, then record the number of sequences which have been learned so far as the pocket length; Otherwise, go to step 1.

This algorithm is repeatedly applied until all the sample sequences are correctly classified or time runs out. A modification to the above algorithm would be such that steps 2d and 2g test-run on all the previously tried sample sequences instead of just the current sample sequence.

The column-wise learning algorithm is similar to the row-wise learning algorithm. It is thus not discussed here.

Storage and Retrieval of Sequences

In this section, the application of the sequential neural network described in section 2 to the storage and retrieval of complete sequences is studied. Kohonen's sequence memory is first examined. The sequential neural network is then considered for the same application. The sequential neural network performs the same functions as Kohonen's sequence memory and also solves a problem in Kohonen's model.

Kohonen's Sequence Memory

Kohonen's sequence memory consists of an auto associative network and a number of delay elements as shown in Figure 3. Some inputs of the auto associative network come from the delay elements and some outputs are connected to the delay elements. This forms a feedback loop and generates internal states.

![Figure 3. Kohonen's sequence memory.](image)

Among the network inputs, some serve the purpose of sequence inputs and some serve the purpose of backgrounds. If the input sequence represents a music, the background pattern represents the name of the music. The sequences are stored with their names. When a name is given, the sequence can be recalled. In Figure 3, \( P_1, \ldots, P_n \) are the inputs for sequence patterns and \( B_1, \ldots, B_k \) are the inputs for background patterns.

Neural Sequential Memory

The sequential neural network can be used to store and to retrieve sequences. To store several sequences, the network can be initialized to several distinct starting states. Therefore, the background inputs are not needed.

The network configuration is the same as discussed in section 2. Background inputs are eliminated. To enable the network to learn a sequence, a distinct starting state \( D_0 \) is given. The learning algorithm discussed in section 3 is then applied to the network. To retrieve the sequence, the network is initialized to the state \( D_0 \). Then a null input is given to the network. This forms a partial pattern. The network can continue with this partial
pattern from state to state and eventually retrieve the stored sequence.

Partial Match Retrieval

It is often desired to store some sequences in a network and to match a specific sequence at a later time when a segment of the sequence is given. This appears to be similar to sequence classification. But they are not exactly the same. This type of partial match retrieval occurs so frequently in practice that it deserves special attention.

For an application of this type, a special output port is set for each sequence to be stored. If a total of n sequences are to be stored, the network should have n output ports. For all the sequences, a common starting state D0 is used. During the learning process, the output is set to +1 for every pattern in the sequence. The network learns the n sequences as described in section 3.

For the purpose of partial match retrieval, a competitive network is added to the output end of the sequential neural network. This competitive network, as shown in Figure 4, consists of n cells, each of which is connected to all n output ports of the sequential neural network.

![Competition Network](image)

Figure 4. Partial match classification with competition network.

The connection from the ith output port to the ith cell is assigned a weight +1 for 1 ≤ i ≤ n. The other connections of a cell are assigned the same weight -Δ, where 0 < Δ ≤ 1/n. These cells remember their previous states (activations). The short-term memory trace for these cells is defined as

\[
\frac{d C_i(t)}{dt} = O_i(t) - \sum_{j \neq i} O_j(t) \Delta,
\]

where \( C_i \) is the activation of the ith competitive cell and \( O_j \) is the value of the jth output port. The discrete form of the short-term trace is

\[
C_i(t+\Delta) = C_i(t) + O_i(t) - \sum_{j \neq i} O_j(t) \Delta.
\]

During the partial match retrieval process, the network is initialized to starting state D0. The partial input sequence is then presented to the network. At the end of the partial sequence, select the cell which has the highest activation value. If cell \( C_i \) is selected, the sequence \( S_j \) matches the partial sequence.

Some Experimental Results

Some simple simulation experiments have been performed to study the behavior of the sequential neural network. Although simple, these experiments prove the viability of the model proposed in this paper.

The first and the simplest experiment was conducted to solve the parity problem for sequences of arbitrary length. The network was implemented as a Moore machine rather than the Mealy machine in Figure 1. The state network has a single delay element and has two inputs, one from the delay element and one from the network input. There is no cell in the output network. The output of the state network serves as the network output. The state network performs the function of an XOR gate. This is easily learned by the network from several simple example sequences.

Another experiment was conducted to solve the problem of deciding whether the total number of 1's in a sequences can be divided by 3. The network should output the remainder. This is similar in nature to the parity problem but the complexity is higher. The network also learned to solve this problem.

We are currently trying to duplicate Gallant's integer addition experiment with variable length operands. In addition, we are also conducting an experiment on the problem of repeated segments. The approach we plan to take is to assign different internal states to repeated segments if they are followed by different segments.

Summary

A new model of sequential neural network is proposed here. Three important application areas for sequence processing are also studied. These areas include the classification of sequences, the storage and retrieval of full sequences, and the partial match classification of sequences. The sequential neural network model presented in this paper can be applied to all the three areas of applications.

In the near future, we will perform more experiments to show the potential of the sequential neural network. Theoretically, the sequential neural network model can solve some problems which are solvable by previous research. A new learning algorithm, which converts the hard learning problem into an easy learning problem by automatically assigning internal states, is also being investigated. Unlike previous unsupervised learning algorithms, this algorithm has the style of a combinatorial optimization algorithm. It aims at the assignment of internal cell states to satisfy linear separability.

References


