A Connectionist Technique for On-Line Parsing

Ronan Reilly
Educational Research Centre
St Patrick's College, Dublin 9

Abstract

A technique is described that permits the on-line construction and dynamic modification of parse trees during the processing of sentence-like input. The approach is a combination of simple recurrent network (SRN) and recursive auto-associative memory (RAAM) . The parsing technique involves teaching the SRN to build RAAM representations as it processes its input item-by-item. The approach is a potential component of a larger connectionist natural language processing system, and could also be used as a tool in the cognitive modelling of language understanding. Unfortunately, the modified SRN demonstrates a limited capacity for generalisation.

A Connectionist Technique for On-Line Parsing

Ronan Reilly
Educational Research Centre
St Patrick's College, Dublin 9

One way of characterising the problem of natural language processing is to view it as an effort to reconstruct the complex non-linear structure of the sender's message from its linear representation in the speech stream. The brain solves this problem successfully by using a combination of structural and contextual clues. The focus of this paper will be on the structural regularities, or syntax, of language. A technique will be described that permits the on-line construction and dynamic modification of parse trees during the processing of sentence-like input. The approach is a combination of two recent innovations in connectionist representation: the simple recurrent network (SRN) of Elman (1990) and the recursive auto-associative memory (RAAM) developed by Pollack (1990). The parsing technique involves teaching the SRN to build RAAM representations as it processes its input item-by-item.

Some Promising Techniques

A localist approach to representation (i.e., for a given level of description one unit is used to represent one term in the description language) has tended to dominate connectionist research on syntax and parsing. With this approach, the greater the complexity of the representation, the more units required. Furthermore, the maximum number of units required, needs to be known in advance. This is a major weakness when contrasted with the productive capacity and fundamental open-endedness of symbolic systems. The question then is, how can one achieve the desirable productive capacity of a conventional symbolic system within a fixed capacity connectionist network?

Representing Complex Structure

A recent development by Pollack (1990) has provided some insight into how this goal might be achieved. Pollack's recursive auto-associative memory (RAAM) is based on an application of the backpropagation algorithm called an encoder. Encoders were first described by Rumelhart, Hinton, & Williams (1986). The idea behind them is quite simple. A three-layer feedforward network is trained to reproduce on its output units, the pattern of activation that is on its input units. A key feature of encoder networks is that the number of intervening hidden units is less than the number of input units. This means that the network must learn an encoding of the input pattern that is sufficiently precise to allow the reconstruction of the pattern on the output units. So, for example, an 8-3-8 encoder (8 input and output units, three hidden units) must learn to develop something like an octal code in order to encode a set of one-in-eight bit patterns. This simple mechanism gives us a means of preserving information through change in the width of the representation. Thus, by taking the pattern of activation on the three hidden units in conjunction with the weights from the hidden units to the output units, it is possible to reconstruct the original 8-bit pattern.

<Figure 1 about here>

Consider now the binary parse-tree in Figure 1. The terminals of the tree can be taken to represent the syntactic categories *determiner* (D), *noun* (N), *verb* (V), and *adjective* (V). The tree can be represented as a list: ((D N) (V (D (A N)))). This might be the structural analysis given to a sentence such as *The boy kicked the red ball.* Using a 2k-k-2k encoder (see Figure 2a), it can be encoded in a fixed-width representation by recursively training the encoder on the following set of patterns:

input pattern	**hidden pattern**	**output pattern**
(D N)	$R_1(t)$	$(D'(t)\ N'(t))$
(A N)	$R_2(t)$	$(A'(t)\ N'(t))$
$(D\ R_2(t))$	$R_3(t)$	$(D(t)'\ R_2(t)')$
$(V\ R_3(t))$	$R_4(t)$	$(V(t)'\ R_3(t)')$
$(R_1(t)\ R_4(t))$	$R_5(t)$	$(R_1(t)'\ R_4(t)')$

where A, D, N, and V are represented by *k* width one-in-*k* bit vectors. The input at each training step consists of two adjacent vectors. For any given epoch *t*, all of the above training patterns are presented. Obviously, the R_n vectors will be changing as the training proceeds. Consequently, these particular patterns present something of a moving target to the learning algorithm. Nevertheless, given a suitable choice of learning parameters, the representations converge and stabilize. At the end of the training procedure, the pattern R_5 can be said to represent the entire tree structure. The elements of this structure can be recursively unpacked by taking R_5, inserting it into the hidden units of the encoder network, and using it to generate the patterns R_1' and R_4' on the output units, which will be close approximations to R_1 and R_4. These patterns, in turn, can be used to recursively generate the terminal elements of the tree.

The advantage of the RAAM technique is that the structures can be of varying complexity, yet still be represented in a fixed-width pattern. There are, however, upper limits to the capacity of RAAM networks of given widths. This is one aspect of the approach that has some psychological appeal, since the limitations are suggestive of similar limitations in humans.

A disadvantage of RAAMs is that they do not capture the temporal character of the structures they encode. All elements of a tree structure (or set of tree structures) are simultaneously encoded. We must turn to another development in connectionist representation to discover a way of capturing the temporal dimension of language while still preserving the representational advantages of the RAAM .

<Figure 2 about here>

Representing Temporal Structure

Language is encountered in the speech modality as a temporal sequence of phonemes. The consequence of this for connectionist models is that some way must be found to integrate information over time. The main technique used until quite recently was to transform the temporal dimension into a spatial one. In the context of a speech processing model, the input to a network might consist of a window spanning a particular period of time. The obvious limitation to this approach is that the model is limited in how far back in time it can make use of the input data. The modeller must set an arbitrary limit on the temporal memory of the system. In the case of a language-processing system, this temporal-to-spatial mapping entails a fixed upper limit on the number of words allowed in a sentence (Cottrell, 1985; Fanty, 1985).

A more appealing solution to the problem of capturing temporal structure without recasting it in a spatial form is to use a recurrent network. Recurrence can be implemented by taking the state of some part of the network at time *t* and using it as input to the network at time *t+1,* in addition to the conventional input. The first researcher to use recurrence in this form was Jordan (1986). His goal was to devise a connectionist system that could generate a sequence of plan-like actions that were dependent on a particular state. The training of his system required that activation from the output units be fed back as input on the next time step. In this way, the network could use the result from one time step to help determine the next output.

A variant of this network was used by Elman (1990) to explore the temporal nature of linguistic data. Elman's model consisted of a modified feedforward network in which the activation values of the hidden units at time *t* were copied to a set of additional input units and are used as input at time *t+1*. The weights from the hidden-unit copies are modifiable in the same way as those from any other units. There are, however, no biases associated with the copy units. This type of network is usually referred to as a simple recurrent network (SRN).

Elman explored the ability of simple recurrent networks to carry information over a long sequence of inputs. Take the following linguistic example:

The boys whom the girls see in the field behind the school eat the apples

In this sentence there is a dependency between the number of the noun *boys* and the number of the verb *eat*. When a listener is presented with this sentence and is asked, for example, to judge whether it is grammatical or not, s/he must carry information about the noun and use it in assessing whether the verb number is correct. This must be done irrespective of the number of intervening words. In reality, the greater the number of intervening words, the more difficulty people have in performing this task. What Elman demonstrated was that given the task of anticipating what the next word in a sentence was going to be, a simple recurrent network was able to utilise information encountered several time-steps previously. However, just as with people, the network did not demonstrate perfect performance, rather the performance degraded as the number of intervening words increased.

Now, the task Elman set his network was by no means as demanding as having to perform a parse, but it did entail operations that were similar. The main facet of language processing that was captured by his network was the impact of expectancies at a given point in a sentence. Such expectancies do seem to be important in natural language understanding (Marslen-Wilson & Welsh, 1978).

An On-Line Parsing Architecture

The parsing technique proposed here is a straightforward combination of SRN and RAAM (see Figure 2b). The task of the SRN, rather than anticipation of the next item in the input sequence, is the construction of a RAAM representation of the input. In order to train the SRN, a necessary first step is to create the relevant RAAM representations.

The idea of deriving RAAMs from sequential input is not entirely new. Pollack (1990) sketched out a scheme in which RAAMs could be incrementally generated from sequential input using a recurrent architecture. However, the architecture he proposed was quite complex (e.g., the output from one network yields the weights for another network). The combination of RAAM and SRN proposed here is much simpler and performs the same function. Sharkey (personal communication) has also used a combination of SRN and RAAM, similar to the one described here.

Training Set

The training set consisted of sequences of word categories generated from a simple context-free grammar (the same one as used in Pollack, 1990). The terminals of the grammar were the same as described above, with an additional terminal representing the category *preposition* (P). The rules for the grammar were:

S	®	**NP VP │ NP V**
NP	®	**D AP │ D N │ NP PP**
PP	®	**P NP**
VP	®	**V NP │ V PP**
AP	®	**A AP │ A N**

This grammar generates the sample sequence given earlier, (D N V D A N), which can be represented as a bracketed binary tree, thus: ((D N) (V (D (A N)))). Some restrictions were placed on the generation process. No more than two recursive applications of the adjectival phrase rule (AP) were permitted. Thus, there could be no more than three adjectives (A) preceding a noun (N). In addition, only one **NP** ® **NP PP** expansion was permitted, which served to restrict the number of prepositions (P) in a noun phrase to one.

A corpus of 1000 sequences was generated using the above rules. This corpus comprised 168 unique sequences. In order to reduce their number and complexity, only sequences containing nine elements or less were considered further. This reduced the relevant set to 40. Initial attempts to train a 40-20-40 RAAM on this set were not successful, which left two options: either scale up the size of the RAAM, or reduce the size of the training corpus. The latter approach was taken, since the goal of the research was not to explore the capacity of RAAMs, but to test the viability of using SRNs to generate RAAMs. Consequently, a random sample of 20 sequences was selected from the corpus of 40, and four of these were removed for use in the tests of generalisation. The rationale for the selection of the generalisation set will be discussed later. This left 16 different sequences or sentence types for use in training (see Table 1).

<Table 1 about here>

RAAM Encoding

A 40-20-40 RAAM was constructed to encode the 16 sentence-types in the training set. The generalised, as opposed to sequential RAAM, was used for this purpose.

Consequently, training this set of 16 sequences involved decomposing them into a set of 31 unique binary sub-trees, just as in the example given at the beginning of the paper. Training these required 30,000 epochs, and involved using a modification of the standard backpropagation algorithm called Quickprop due to Fahlman (1988). This is a so-called second-order learning algorithm, which uses an interpolation method to estimate more accurately the error minimum for each weight. A potential disadvantage of this type of algorithm for the RAAM learning task is that it performs best if a number of training patterns are presented before the weights are changed. Usually, the weight changes are accumulated and put into effect at the end of each epoch. However, because the training patterns themselves change as the weights change, if weight updates do not take place often enough, learning will not converge. A way around this is to have updates occur after some portion of the training patterns have been presented. The update rate chosen for this application was every 10 patterns. The Quickprop algorithm[1] used in this way was found to be faster and more effective than conventional backpropagation (Rumelhart, Hinton, & Williams, 1986).

SRN/RAAM Parser

The RAAM representations (i.e., the vectors of hidden unit activations) associated with each sentence-type were used in the training of an SRN (henceforth, the SRN/RAAM). The SRN/RAAM consisted of six input units, 20 hidden units, and 20 output units. The output units comprised the 20 units used to represent the RAAM encoding of the input sentence. In addition to the six input units connecting to the hidden units, there were also 20 context units which were used to store a copy of the hidden unit activations from the previous time step. The standard backpropagation algorithm, as opposed to the Quickprop variant, was used for training, since algorithms like Quickprop do not perform well with recurrent architectures. Earlier attempts to use fewer hidden units (10 and 15), resulted in learning failing to converge.

The SRN/RAAM was trained on the 16 sequences for 20,000 epochs. A regime of decreasing learning rates was used. Initially set relatively high (0.5), the learning rate was decreased every 1,000 epochs to a minimum of 0.01. It should be emphasised that at any given time step, the only input to the SRN/RAAM was an element of the input sequence and the hidden unit activations from the previous time step. No information about the structure of the sequence was included in the input. Furthermore, at the end of each

[1] Some more details on learning parameters: A learning rate of 1.0 was used, but for a given unit it was divided by the fan-in to that unit. A standard sigmoid activation function was used, with output in the range 0.0 to 1.0. All other parameters, were kept at the defaults specified in Fahlman (1988).

sequence, the context units were reset. Below is a schematic of the training regime for the two sequences ((D N) V) and ((D N) (V (D N))):

Time	**Input**	**Teacher**
1	D	R_a
2	N	R_a
3	V	R_a
4	.	R_a
5	D	R_b
6	N	R_b
7	V	R_b
8	D	R_b
9	N	R_b
10	.	R_b

where R_b and R_a are the RAAM encodings of ((D N) V) and ((D N) (V (D N))), respectively. At time-step 5, the context units are reset, thus obliterating information about the preceding sequence.

Results

The main result from this experiment was the finding that it is indeed possible to train an SRN to generate accurate RAAM representations of an input sequence, though with some difficulty. As can be seen from Table 1, five out the 16 sequences were not trainable. The five that gave trouble all contained the sequence (A (A N)). During several training runs the performance of the network showed a tendency to deteriorate after around 20,000 epochs. Training was halted at this point.

One interesting aspect of the behavior of a the trained SRN is the way in which its structural expectancies change as a sequence is input. This is illustrated in the following sequence (D N P D N V D A N .):

Input	**Decoded RAAM Output**
D	((D ?))
N	(?)
P	(((D N) (P (D ?))) V)
D	(((D N) (P (D ?))) V)
N	(((D N) (P (D N))) V)
V	(((D N) (P (D N))) V)
D	(((D N) (P (D ?))) ?)

A	(((D N) (P (D (A N)))) V)
N	(((D N) (P (D N))) (V (D (A N))))
.	(((D N) (P (D N))) (V (D (A N))))

A "?" in the above parse trees indicates that the RAAM vector was not decodable at this point. For a given vector, the decoding process continues either until there are no more non-terminal branches[2], or until an embedding depth of 10 is reached. The latter case is marked by a "?." In the above example, an attempt is made to decode the output at each input step. When the V is encountered, there is enough information to produce a well-formed parse. This preliminary "hypothesis" must be revised as more input is received. An interesting feature of this sequence is the output produced when the A is reached. A valid parse tree is produced, but one which does not reflect the order of the input sequence. What seems to have happened is that a learned sequence has acted as an attractor. This view is confirmed when the network's generalisation performance is examined.

As well as looking at overall performance, the ability of the SRN/RAAM to generate correct parse trees for sentences that it had not been exposed to in training was tested. As has already been mentioned, four sequences from the initial corpus of 20 were selected for use in tests of generalisation. The goal of the generalisation tests was to discover if the SRN/RAAM could demonstrate a sensitivity to certain sub-structures in the training set irrespective of their position in the input sequence. The sub-structure (P (D N)) was chosen, and it occurred in different locations in the four generalisation sequences.

<Insert Table 2 about here>

The results of the generalisation test are given in Table 2. In two out of the four cases the sub-structure was correctly dealt with, although in no case was the entire parse correct. It is obvious from these results that the representations of learned sequences are acting as attractors for new ones. In the context of a richer training environment, this processing strategy would certainly yield a higher rate of correct sub-structure parsing than in the present case.

It is informative to look also at the way in which the RAAM encoder/decoder dealt with the generalisation set. The results are given in Table 3. Generalisation in this context was tested by first encoding the structures and then attempting to decode them. The decoding criteria used with the SRN/RAAM output was also applied here. In only one case, was the (P (D N)) sub-structure entirely recovered. It was partially recovered in two

[2] A vector is considered to be a non-terminal if it is not a one-in-*k* binary vector.

cases, and not at all in the third. It seems that generalisation in regular RAAM networks is more conservative than in the SRN/RAAM variant.

<Insert Table 3 about here>

Discussion

The main goal of this paper has been to demonstrate the feasibility of combining an SRN and RAAM in order to perform on-line parsing. Although the training set was trivial and the scale of the model quite small, the technique does permit the exploitation of a relatively powerful connectionist representation, the RAAM, in a form that is useful to on-line language processing. Furthermore, the temporal nature of the SRN/RAAM makes it attractive as a cognitive modelling tool in the study of natural language understanding. For example, the way in which the RAAM representation unfolds over time might be useful in modelling the processing of structurally ambiguous sentences (e.g., *The boy saw the man with the telescope*, where *with* can indicate facilitation or possession). Of course, to model this phenomenon the training set would need to comprise words rather than word categories.

There are, however, a number of problems with the SRN/RAAM parsing approach as it is currently formulated. One of the most important is the poor generalisation ability of the both the RAAM and SRN/RAAM networks. Pollack (1990) acknowledges that RAAMs do not demonstrate a desirable degree of generalisation. In the case of the SRN/RAAM, one way around poor generalisation is to enlarge the size of the training set, but this puts additional demands on the RAAM encoding process. RAAMs have limited capacity and are troublesome to train. In fact, the learning demands of the two architectures are in conflict with one another.

The answer to this problem may lie in developing a more psychologically plausible approach to the development of RAAM-like representations. The technique used for generating RAAMs cannot bear much relationship to the way in which representations are created in the brain of the language user. But, RAAMs have a number of appealing properties from a psychological point of view. It may be possible, therefore, to devise a method of learning representations with the same functional properties as RAAMs, but developed in manner that is truer to the time-varying sequential nature of language. One promising approach is described by St John and McClelland (1990), where they propose a complex representation called a "sentence gestalt" which is trained in a sequential manner to encode the case-role assignments of sentence. Such a representation might be extended to encode RAAM-like structural information as well.

References

Cottrell, G. W. (1985). Connectionist parsing. In *Proceedings Seventh Annual Conference of the Cognitive Science Society*. Irvine, CA.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179-212

Fahlman, S. E. (1988). Faster learning variations on back-propagation. In D. Touretsky, G. Hinton, & T. Sejnowski (Eds.), *Connectionist models Summer school.* San Mateo, CA: Morgan Kaufman.

Fanty, M. (1985). *Context-free parsing in connectionist networks* (Technical Report TR-174). Rochester, NY: Dept. of Computer Science, University of Rochester.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. *Proceedings of the Cognitive Science Society*. Amherst, MA, August, pp. 531- 546.

Marslen-Wilson, W. D., & Welsh, A. (1978). Processing interactions and lexical access during word recognition in continuous speech. *Cognitive Psychology, 10,* 29-63.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland and the PDP Research Group (Eds.), *Parallel distributed processing. Explorations in the microstructure of cognition. Volume 1: Foundations* (pp. 318-362). Cambridge, MA: MIT Press.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence, 46*, 77-105.

St John, M. & McClelland, J. L. (1990). Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence, 46,* 217-257.

Acknowledgements

Thanks to Pat O'Seaghdha for commenting on an earlier draft of this paper, to Gerry Orchard for his forebearance, and to two anonymous reviewers for their helpful comments.

Figure and Table Captions

Figure 1

A parse tree which could be the structural analysis for a phrase such as *The boy kicked the red ball*. Reading from left-to-right, the terminals of the tree stand for *determiner*, *noun*, *verb*, and *adjective*. The non-terminal nodes stand for *noun phrase*, *verb phrase*, and *adjectival phrase*.

Figure 2a

The architecture of a RAAM network. The lower half of the network is used for encoding nodes from individual branches of a binary tree. The top half is used to decode a node into its component branches.

Figure 2b

A modified simple recurrent network (SRN). The input is an element in a sequence along with the pattern of activations from the hidden units at the previous time step. Note that the link marked copy is not modifiable. The output from this network is a structural analysis of the input sequence encoded in a recursive auto-associative memory (RAAM) vector.

Table 1

The training set used in the study. When used in training the RAAM network the structures were broken down into 31 sub-trees, and the nodes of each of these sub-trees were auto-associated in a 40-20-40 RAAM encoder. When used to train the SRN/RAAM network, the structures were transformed into a linear sequences, all structural information was dispensed with, and the task of the network was to reproduce the correct RAAM encoding for each input sequence. The sequence numbers in boldface indicate those that the SRN/RAAM failed to learn.

Table 2

The generalisation set used in the study. Note that the input to the SRN/RAAM was strictly linear and contained no structural information. The emboldened sub-sequence was the particular focus of the generalisation test.

Table 3

The result of using the RAAM to decode the generalisation set. The decoding process entailed encoding the sequence, and then attempting to decode the resulting representation. A "?" indicates that the sequence was undecodable at that point.

Table 1

ID	**Sequence**
1	(((d (a (a n)))(p (d n)))v)
2	(((d (a n))(p (d (a n))))v)
3	(((d (a n))(p (d n)))v)
4	(((d n)(p (d (a n))))v)
5	(((d n)(p (d n)))(v (d (a n))))
6	(((d n)(p (d n)))v)
7	((d (a (a n)))(v (d (a n))))
8	((d (a (a n)))(v (d n)))
9	((d (a (a n)))(v (p (d (a n)))))
10	((d (a (a n)))v)
11	((d (a n))(v (d (a n))))
12	((d (a n))(v (p (d (a n)))))
13	((d (a n))v)
14	((d n)(v (d (a n))))
15	((d n)(v (p (d (a n)))))
16	((d n)v)

Table 2

Input Structure	**SRN Input Sequence**	**SRN Output Structure**
(((d n)(p (d n)))(v (d n)))	d n **p d n** v d n.	(((d n)(p (d n)))(v (d (a n))))
(((d (a n))(p (d n)))(v (d n)))	d a n **p d n** v d n .	(((d n)(p (d n)))(v (d (a n))))
((d (a n))(v (p (d n))))	d a n v **p d n** .	((d (a n))(v (p (d (a n))))
((d n)(v ((d n)(p (d n)))))	d n v d n **p d n** .	((d n) (v (d (a n))))

Table 3

Input Structure	**RAAM Decoded Structure**
(((d n)(p (d n)))(v (d n)))	(((d n) (p (d n))) (v (? n)))
(((d (a n))(p (d n)))(v (d n)))	(((d (a n)) (p (? n))) ?)
((d (a n))(v (p (d n))))	((d (a n) (v (p (? n))))
((d n)(v ((d n)(p (d n)))))	((d n) (v ((d ?) ?)))