Two Layer Digital RAAM

Alan D. Blair
Dept. of Computer Science
Volen Center for Complex Systems
Brandeis University
Waltham, MA 02254-9110
blair@cs.brandeis.edu

Abstract

We present modifications to Recursive Auto-Associative Memory which increase its robustness and storage capacity. This is done by introducing an extra layer to the compressor and reconstructor networks, employing integer rather than real-valued representations, pre-conditioning the weights and pre-setting the representations to be compatible with them, and using a quick-prop modification. Initial studies have shown this method to be reliable for data sets with up to three hundred subtrees.

Introduction

In the late 1980's a number of new connectionist models were developed in response to criticisms (e.g. Fodor & Pylyshyn, 1988) that connectionism lacked the flexibility and representational adequacy needed for higher level cognitive tasks.

Chief among these were coarse coding (Touretzky, 1986), tensor based representation (Smolensky, 1990), reduced representations (Hinton, McClelland & Rumelhart, 1986), and RAAM (Pollack, 1990). Compared to earlier systems, they had the advantage of compositionality built more explicitly into their design, and they have shown a great deal of promise in a number of areas (Chalmers, 1990, Plate, 1994). However, all of them typically run into difficulties when the structures involved are scaled up to a level of complexity commensurate with real world problems.

In this paper, we describe a number of modifications to the RAAM architecture designed to address some of these inadequacies, and examine the feasibility of storing large, complex data structures within a connectionist system.

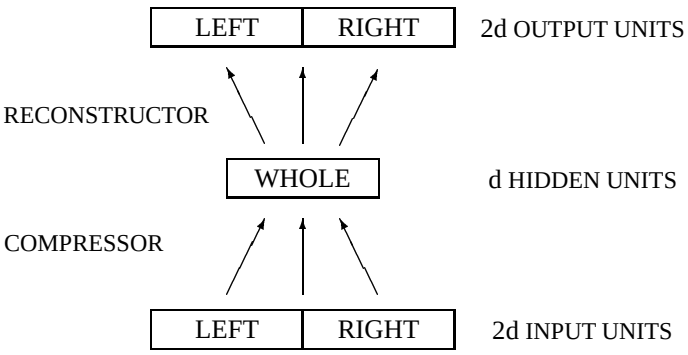


Figure 1. RAAM architecture - a single network composed of a compressor and a reconstructor.

Review of RAAM

Recursive Auto-Associative Memory or RAAM (Pollack, 1990) is a method for storing tree structures in a feed forward network. It's architecture is very similar to that of encoder networks (Ackley, Hinton & Sejnowski, 1985, Cottrell, Munro & Zipser, 1987), consisting of a compressor unit and a reconstructor unit. The principal difference is that in a RAAM the compressor and reconstructor are used *recursively* to encode and decode, respectively.

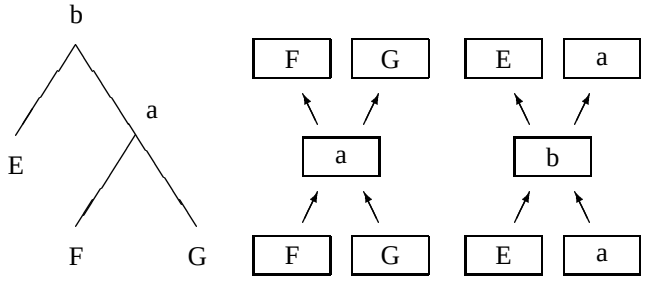


Figure 2. A simple tree and the auto-associations that encode it in a RAAM.

Figure 2 shows how a RAAM encodes the tree (E (F G)). First we feed (F G) into the compressor network, giving output a. Then we feed in (E a), giving b. To decode, we feed b into the reconstructor network, giving (E a). At that point we need some kind of 'terminal test' to tell us that E, F & G are terminals (requiring no further decoding), while a is a non-terminal that must be fed again into the reconstructor - giving (F G).

Several trees may be stored in the same RAAM at once. In what follows, we shall measure the complexity of a data set by the number n of subtrees or 'auto-associations' required to encode it. In the above example $n = 2$.

Modifications to RAAM

Hidden layers

We enlarge the compressor and reconstructor networks to two layers each as shown in Figure 3, in order to increase the number of functions computable by the network.

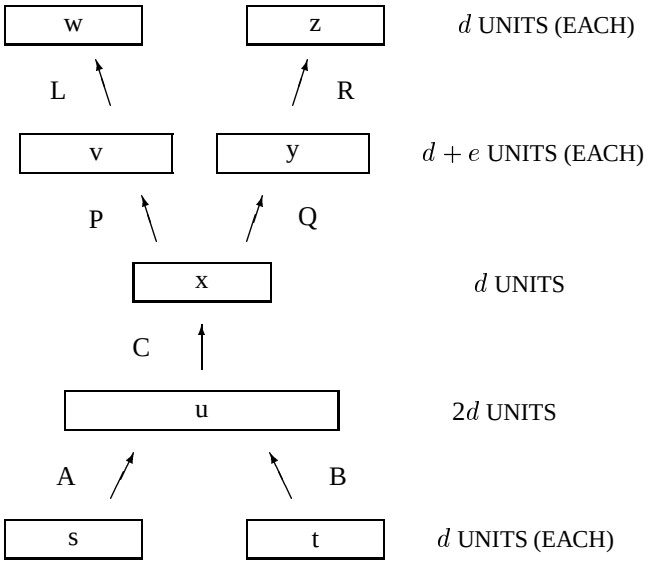


Figure 3. Architecture for Two Layer RAAM.

Digital outputs

One problem with RAAM is that, since the representations are allowed to take on non-integer values, greater accuracy is required as the depth of the trees increases, in order to prevent accumulation of round-off errors. We modify the network so that each output must take on a discrete value ($+1$ or -1), thus allowing larger structures to be stored in a noise tolerant fashion. This is done by using a threshold function Θ at the second layer of the compressor and reconstructor networks, while a hyperbolic tangent is used at the hidden layers:

$$x_i = \Theta\left(C_{i0} + \sum_{j=1}^{2d} C_{ij} \tanh\big(A_{j0} + \sum_{k=1}^{d} (A_{jk} s_k + B_{jk} t_k)\big)\right)$$

$$w_i = \Theta\left(L_{i0} + \sum_{j=1}^{d+e} L_{ij} \tanh\big(P_{j0} + \sum_{k=1}^{d} P_{jk} x_k\big)\right)$$

$$z_i = \Theta\left(R_{i0} + \sum_{j=1}^{d+e} R_{ij} \tanh\big(Q_{j0} + \sum_{k=1}^{d} Q_{jk} x_k\big)\right)$$

Pre-conditioned weights

It is well known that the success of neural network training using back-propagation is sensitive to the initial weight configuration (Kolen & Pollack, 1990). The complete randomness of the initial representations and weights becomes a significant problem as RAAM s are scaled up. To increase the likelihood of convergence, we adopt the following strategy for choosing the initial weights:

First, randomly choose two signed permutation matrices $\mathbf{P}_0$ and $\mathbf{Q}_0$. For example, if $d = 4$, we may have

$$\mathbf{P}_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{Q}_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Once $\mathbf{P}_0$ and $\mathbf{Q}_0$ are chosen, we assign the initial weights for the reconstructors as follows:

$$\mathbf{P} = \frac{1}{d}\begin{bmatrix} \mathbf{P}_0 \\ \mathbf{0} \end{bmatrix} \quad \mathbf{Q} = \frac{1}{d}\begin{bmatrix} \mathbf{Q}_0 \\ \mathbf{0} \end{bmatrix} \quad \mathbf{L} = \mathbf{R} = \left[\begin{array}{c|c} \frac{\mathbf{I}(d)}{d} & \frac{\mathbf{I}(e)}{n} \\ \mathbf{0} & \end{array}\right]$$

where $\mathbf{I}(d)$ is the $(d \times d)$ identity matrix, and $\mathbf{0}$ denotes a zero matrix of the appropriate dimensions. In other words, the first d nodes of the hidden layer are connected in a 1-to-1 fashion with those of the input and output layers by connections with synaptic strength d^{-1}, in such a way that the connections to the output layer are component-wise and excitatory, while those to the input layer are randomly assigned and may be excitatory or inhibitory. The remaining e nodes are connected componentwise to the first e nodes of the output layer by weaker excitatory links with strength n^{-1} (where n is the number of subtrees to be stored). All other connections are initially set to zero. Each layer also has bias inputs, which are also initialized to zero. The initial compressor network is wired thus:

$$\mathbf{A} = \begin{bmatrix} \mathbf{P}' \\ \mathbf{0} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{Q}' \end{bmatrix} \quad \mathbf{C} = \frac{1}{d}\begin{bmatrix} \mathbf{I}(d) & \mathbf{I}(d) \end{bmatrix}$$

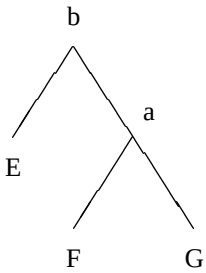
$\mathbf{P}'$ and $\mathbf{Q}'$ denote the transpose of $\mathbf{P}$ and $\mathbf{Q}$. This setup has the following advantages:

(a) the initial compressor is a left inverse for the initial reconstructor,

(b) it produces compressors and reconstructors with much longer transients than would be the case with random initial weights, thus allowing the network to store trees of greater depth.

Initial representations

In single layer RAAM, non-terminal representations are determined by the network as an artifact of the training. This approach has the disadvantage that two or more representations may become fused in the course of the training (Angeline, 1992). The fusion problem gets more pronounced as the number of nodes increases, and is even more prevalent when the representations become digital. We circumvent this difficulty by assigning the representations at the outset, in a way that is compatible with the initial weights. To see how this is done, consider our earlier example:



Now imagine a *linearized* version of the problem, in which the compressor and reconstructors are effected by (linear) matrix multiplications, rather than two-layer neural networks. In fact the initial weights as defined above do just that, using the matrices $\begin{bmatrix} \mathbf{P}_0' & \mathbf{Q}_0' \end{bmatrix}$, $\mathbf{P}_0$ & $\mathbf{Q}_0$, respectively. Now suppose we assign a random representation to the root node b. For

instance, we could assign

$$x(b) \;=\; \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

Then it would be natural to use our initial (linearized) reconstructors $\mathbf{P}_0$ & $\mathbf{Q}_0$ to determine representations for the other nodes, putting

$$x(\mathrm{E}) = \mathbf{P}_0.x(b) = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \quad x(a) = \mathbf{Q}_0.x(b) = \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

$$x(\mathrm{F}) = \mathbf{P}_0.x(a) = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \quad x(\mathrm{G}) = \mathbf{Q}_0.x(a) = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

This is the strategy we follow in general, with the following provisos:
(a) In general there will be several trees in the data set, and we assign a random representation to each root node.
(b) The above example is particularly simple because each terminal appears only once. In general a typical terminal or subtree will appear several times throughout the data set, and the above procedure will generate multiple representations for it. We extract a single representation from this multitude by first computing their *average*, then rounding off each unit to $+1$ or -1, depending on its sign.
(c) It may happen that two nodes end up having exactly the same representation. In this case, we must select $\mathbf{P_0}$ and $\mathbf{Q_0}$ anew, and repeat the above procedure, choosing different representations for the root nodes. In order to estimate the probability of this problem arising, note that the total number of available representations is 2^d. Suppose the number of terminals and subtrees to be represented is N, and choose d large enough that $2^d > N^2$. If each representation were chosen at random (which is not strictly the case, but is probably a 'reasonable' assumption), the probability of them all being distinct would be

$$\prod_{i=0}^{N-1} \left(1 - \frac{i}{N^2}\right) \;>\; 1 - e^{-1} \;>\; 0.6$$

So, by repeating this procedure a couple of times if necessary, we should soon satisfy the requirement that all representations be distinct.

Back-prop modification

Since the representations are chosen in advance of training, the compressor and reconstructor networks may be trained separately. Training proceeds using back-propagation (Rumelhart, Hinton & Williams, 1986), with the following modification similar to Quickprop (Fahlman, 1989):
In the usual back-propagation algorithm, the value of 'delta' propagated back through the network from the output layer is

$$\begin{aligned} \delta_i &= \tfrac{\partial z_i}{\partial r_i} \cdot (t_i - z_i) \\ &= (1 - z_i z_i)(t_i - z_i) \end{aligned} \quad \Big| \quad \begin{aligned} z_i &= \tanh(r_i) \\ t_i &= \text{target value for } z_i \end{aligned}$$

We instead take

$$\delta_i = (1 - z_i t_i)(t_i - z_i)$$

Such a choice of δ prevents individual outputs from getting trapped into a flat region on the wrong side of zero, by putting more emphasis on learning the correct sign for the outputs, and less on their exact numerical values.

At the conclusion of training, the transfer function in the output layer is changed from a hyperbolic tangent to a threshold function. In view of this, the network may be said to have successfully learned the training set once the maximum error across all units of all outputs is less than 1.0. However it is prudent to allow some safety margin, and in the trials described below we continued to train until the maximum error was less than 0.6. The learning rate must be very small in order to ensure convergence. After some preliminary trials, we settled on a learning rate of $(nd)^{-1}$ for the reconstructors and $(2nd)^{-1}$ for the compressor.

Parallel training

Parallelization of the training set provides a significant speed-up to back-propagation (Blelloch & Rosenberg, 1987). By removing dependencies from the original RAAM training regimen and parallelizing the algorithm on a 4096 processor Maspar MP2, we were able to run large scale experiments with full parallelization over the training sets.

Data

We tested our methods on four different data sets, each consisting of parse trees for a collection of English sentences. Table 1 provides a summary, showing the number of trees in each data set, their average depth, and the total number of subtrees.

Table 1. Summary of Data.

Data Set	No. Trees	Avg. Depth	No. Subtrees
1	7	3.1	15
2	4	8.8	45
3	48	5.7	169
4	37	8.4	307

Data Set (1) is from (Pollack, 1990). Data Set (3) was taken from an introductory text on Syntactic Theory (Cowper, 1992). Data Sets (2) and (4) were extracted from a small fragment of the University of Pennsylvania Tree Bank [1]. The full data sets are available through the World Wide Web [2]. Here is an example of a 'typical' tree from each data set:

1. ((D N)(V(D(A N))))
2. (S(NP(S(VP(P((NP(P NP))NP)))))(NP(VP(NP(P(NP NP)))))
3. ((NP(I(V(C(NP(I VP))))))(CONJ(NP(I VP))))
4. (ADJP((VP(NP NP))((NP(S(VP NP)))(S(NP(S(VP(NP(P NP))))))))

We took the liberty of slightly modifying the trees to make them binary - since our purpose was not to get syntactic details right, but simply to test how well our scheme could cope with the kinds of structures that typically arise in linguistic applications.

[1] ftp://ftp.cis.upenn.edu/pub/treebank/doc/*
[2] http://www.cs.brandeis.edu/~blair/home.html

Results

The results are shown in Table 2, where n is the number of subtrees, d is the dimension of the representations, e is the number of 'extra' units in the hidden layer of the reconstructors, $m = (5d + 2e + 1)(2d + 1) - 1$ is the total number of connections in the network, and t_{enc}, t_{left} & t_{right} are the number of epochs to convergence for the compressor and the left and right reconstructors, respectively.

Table 2. Summary of Results.

n	d	e	m	m/n	t_{enc}	t_{left}	t_{right}
15	9	0	873	58.2	250	100	150
45	12	0	1524	33.9	800	1,100	900
169	16	8	3200	18.9	1,800	11,000	7,500
307	17	17	4199	13.7	16,700	42,800	31,500

For large data sets, the compressor converged faster than the reconstructors - presumably due to the larger number of connections in the compressor network - and the right reconstructor converged faster than the left one. This is probably due to the fact that parse trees tend to be left-branching (in English), and the resulting 'many-to-one' nature of the left map makes it harder to learn.

Unfortunately, the system shows little or no capacity for generalization. Careful analysis of the compressor and reconstructors trained on the above data sets reveals that they were unable to store and retrieve any trees that were not explicitly in the data set. This trade-off between storage capacity and ability to generalize presumably comes about because of the way we train the compressor and reconstructors separately, and assign the subtree representations in advance, instead of letting them be determined by the network in the course of its training.

Conclusion

These results show that two-layer digital RAAM can be used to reliably find representations for sets of binary trees of a size and complexity that would confound ordinary RAAM and most other connectionist representation systems. These advantages come from stronger constraints on the initial weights and the actual representations of the non-terminals, at the expense of generalization ability. Further work on how to preserve generalization while expanding capacity is certainly called for.

Acknowledgments

The author wishes to thank Jordan Pollack for many helpful comments and suggestions. This research was funded by a Krasnow Foundation Postdoctoral Fellowship, and by ONR grant N00014-95-0173.

References

Ackley, D.H., Hinton, G.E. & Sejnowski, T.J. 1985. A learning algorithm for Boltzman Machines, *Cognitive Science* **9**, 147–169.

Angeline, P.J. 1992. Avoiding fusion in floating symbol systems, Tech. Report 92-PA-FUSION, Computer Science Dept., Ohio State University.

Blelloch, G., Rosenberg, C.R. 1987. Network learning on the Connection Machine, *Proceedings Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy.

Chalmers, D.J. 1990. Syntactic transformations on distributed representations, *Connection Science* **2**(1-2), 53–62.

Cottrell, G., Munro, P., & Zipser, D. 1987. Learning internal representations from gray-scale images: An example of extensional programming, *Proceedings Ninth Annual Conference of the Cognitive Science Society,* Seattle, WA, 461–473.

Cowper, E.A. 1992. *A Concise Introduction to Syntactic Theory* (University of Chicago Press, Chicago, IL).

Fahlman, S.E. 1989. Fast-learning variations on back-propagation: an empirical study. In D. Touretzky, G. Hinton & T. Sejnowski, eds. *Proceedings of the 1988 Connectionist Models Summer School*, Pittsburgh, PA, 38–51 (Morgan Kaufman, San Mateo).

Fodor, J.A., Pylyshyn, Z.W. 1988. Connectionism and cognitive architecture: a critical analysis, *Cognition* **28**, 3–71.

Hinton, G.E., McClelland, J.L., Rumelhart, D.E. 1986. Distributed Representations. In D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds. *Parallel Distributed Processing: Experiments in the Microstructure of Cognition* **1**: Foundations (MIT Press, Cambridge, MA).

Kolen, J., Pollack, J.B. 1990. Back propagation is sensitive to initial conditions, *Complex Systems* **4**, 269–280.

Plate, T.A. 1994. Distributed Representations and Nested Compositional Structure, Ph.D. Thesis, University of Toronto.

Pollack, J.B. 1990. Recursive Distributed Representations, *Artificial Intelligence* **46**(1), 77–105.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. 1986. Learning representation by back-propagating errors, *Nature* **323**, 533–536.

Smolensky, P. 1990. Tensor product variable binding and the representation of symbolic structures in a connectionist system, *Artificial Intelligence* **46**(1-2), 159–216.

Touretzky, D.S. 1986. BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees, *Proceedings Eighth Annual conference of the Cognitive Science Society* (Erlbaum, Hillsdale, NJ).