Scaling-up RAAMs

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

January 6, 1997

Abstract

Modifications to Recursive Auto-Associative Memory are presented, which allow it to store deeper and more complex data structures than previously reported. These modifications include adding extra layers 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. The resulting system is tested on a data set of syntactic trees extracted from the Penn Treebank.

1 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 representations (Smolensky, 1990), reduced representations (Hinton et al., 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, Elman, 1990, Chrisman, 1991, Blank et al., 1992, Plate, 1994, Niklasson & van Gelder, 1994). However the data used to test these models has

generally been confined to relatively simple structures – at most 3 or 4 levels deep. Our aim was to see whether one of these architectures could be adapted to handle linguistic data of 'real world' complexity. To find such data, we took a small fragment of the Penn Treebank (Marcus et al., 1993) – a large corpus of text from the Wall Street Journal – and used a filter to strip out the text, leaving only the syntactic structures. We took the liberty of slightly modifying the parse trees to make them binary – since our purpose was not to get the syntactic details exactly right, but simply to gather data of the appropriate size and complexity.

Many of the resulting trees (see Appendix) were quite complex – 8 levels deep or more – making them problematic for traditional architectures. For example, tensor-based representations require storage space that grows exponentially with the depth of the structures to be stored, while RAAMs have difficulty learning deep structures like these because of roundoff errors that grow exponentially with the depth of the trees being encoded. We therefore developed a number of modifications to the RAAM architecture in the hope of allowing such deep structures to be processed successfully.

2 Review of RAAM

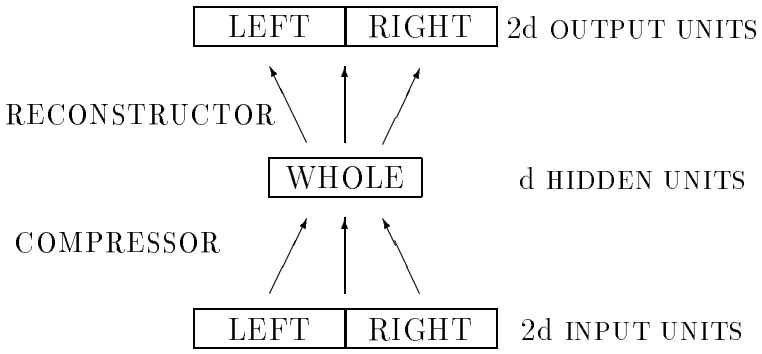


Figure 1. RAAM architecture - a single network composed of a compressor and a reconstructor.

Recursive Auto-Associative Memory or RAAM (Pollack, 1990) is a method for storing tree structures in a feed-forward neural network. It's architecture is very similar to that of encoder networks (Ackley et al., 1985, Cottrell et al., 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.

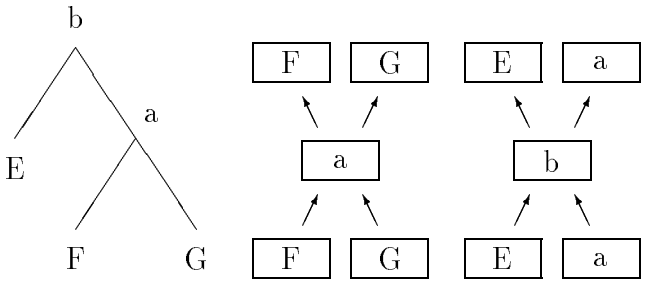


Figure 2. A simple tree and the auto-associations that encode it in a RAAM.

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 size of a data set by the number n of subtrees or 'auto-associations' required to encode it. In the above example $n = 2$.

3 Modifications to RAAM

3.1 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.

3.2 Digital outputs

One problem with RAAM has been 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 deeper structures to be stored in a noise tolerant fashion.

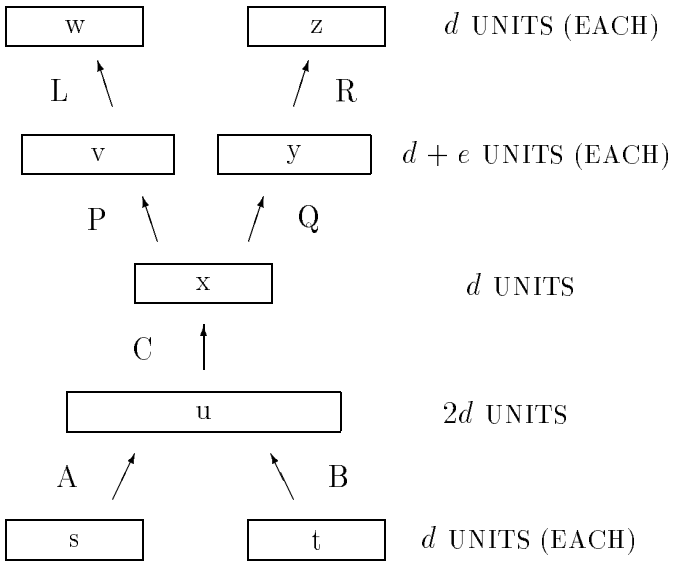


Figure 3. Architecture for Two Layer RAAM.

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:

$$
\begin{aligned}
x_i &= \Theta\left(C_{i0} + \sum_{j=1}^{2d} C_{ij} \tanh(A_{j0} + \sum_{k=1}^{d}(A_{jk}s_k + B_{jk}t_k))\right) \\
w_i &= \Theta\left(L_{i0} + \sum_{j=1}^{d+e} L_{ij} \tanh(P_{j0} + \sum_{k=1}^{d} P_{jk}x_k)\right) \\
z_i &= \Theta\left(R_{i0} + \sum_{j=1}^{d+e} R_{ij} \tanh(Q_{j0} + \sum_{k=1}^{d} Q_{jk}x_k)\right)
\end{aligned}
$$

3.3 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), and indeed can be enhanced by pre-setting some or all of the weights (Pratt et al., 1991). The complete randomness of the initial representations and weights becomes a significant problem as RAAMs 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:

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

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 of the reconstructors (i.e. layers v and y in Figure 3) and compressor (i.e. layer u) are connected in a 1-to-1 fashion with those of the input and output layer (i.e. layers w, z and x, respectively) 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 as follows, where $\mathbf{P}'$ and $\mathbf{Q}'$ denote the transpose of $\mathbf{P}$ and $\mathbf{Q}$:

$$\mathbf{A} = \frac{1}{d}\begin{bmatrix} \mathbf{P}_0' \\ \mathbf{0} \end{bmatrix} \quad \mathbf{B} = \frac{1}{d}\begin{bmatrix} \mathbf{0} \\ \mathbf{Q}_0' \end{bmatrix} \quad \mathbf{C} = \frac{1}{d}\begin{bmatrix} \mathbf{I}(d) & \mathbf{I}(d) \end{bmatrix}$$

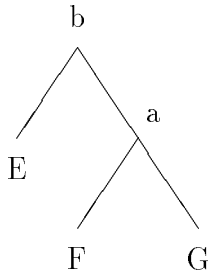
This setup has the following advantages:

(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.

3.4 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 $\left[\begin{array}{cc} \mathbf{P}_0' & \mathbf{Q}_0' \end{array}\right]$, $\mathbf{P}_0$ & $\mathbf{Q}_0$, respectively. Now suppose we assign a random representation to the root node b. For instance, we could assign

$$x(b) = \left[\begin{array}{r} 1 \\ 1 \\ -1 \\ 1 \end{array}\right]$$

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

$$x(\mathrm{E}) = \mathbf{P}_0.x(b) = \left[\begin{array}{r} -1 \\ -1 \\ 1 \\ 1 \end{array}\right] \quad x(a) = \mathbf{Q}_0.x(b) = \left[\begin{array}{r} -1 \\ -1 \\ -1 \\ 1 \end{array}\right]$$

$$x(\mathrm{F}) = \mathbf{P}_0.x(a) = \left[\begin{array}{r} -1 \\ 1 \\ 1 \\ -1 \end{array}\right] \quad x(\mathrm{G}) = \mathbf{Q}_0.x(a) = \left[\begin{array}{r} -1 \\ -1 \\ 1 \\ -1 \end{array}\right]$$

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.

4 Training

Since the representations are chosen in advance of training, the compressor and reconstructor networks may be trained separately. We trained them using back-propagation (Rumelhart et al., 1986), with a modification similar to Quickprop (Fahlman, 1989)[1]. At the conclusion of training, the transfer

[1]Specifically, the cost function we used was

$$E \;=\; -\frac{1}{2}(1+s)^2 \log(\frac{1+z}{1+s}) \;-\; \frac{1}{2}(1-s)^2 \log(\frac{1-z}{1-s}) \;+\; s\,(s-z)$$

(where z is the actual output and s the desired output) which leads to a 'delta rule' of

$$\delta = (1 - sz)(s - z).$$

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 successful training. After some preliminary trials, we settled on a learning rate of $(nd)^{-1}$ for the reconstructors and $(2nd)^{-1}$ for the compressor.

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.

5 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 achieve successful encoding and decoding 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}
1.	15	9	0	873	58.2	250	100	150
2.	45	12	0	1524	33.9	800	1,100	900
3.	169	16	8	3200	18.9	1,800	11,000	7,500
4.	327	17	17	4199	13.7	16,700	42,800	31,500

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

Unfortunately, these networks had great difficulty in generalizing to retrieve new strings that were not in the training set. One possible reason for this is the strictness of the terminal test, which we implemented simply as a

lookup table. If the terminal test were to be performed in a more 'natural' way (for example by an extra, separately trained, layer in the network), it would be likely to pick up some activation vectors which were near to designated terminals, but not precisely equal to them. To test the possible effect of this in simulation, we did an experiment where an activation vector was accepted as a designated terminal if it agreed with that of the terminal in $(d-1)$ of the d activation nodes. With this modification, all the trees in the training sets were still recovered successfully and in addition the following new trees could be encoded and decoded:

Network (1):	((D (A N)) (D (A N)))
Network (2):	(NP (VP VP)) (NP (S (NP (VP VP)))) (NP (SINV VP)) (VP NP) (VP ((SINV VP) ((NP NP) (VP (NP (S VP))))))
Network (4):	(NP ADVP)

As with traditional RAAMs, generalization ability might conceivably be expanded by increasing the number of representation nodes, which in our experiments was essentially chosen as just the minimum number required to reliably store the training set.

6 Conclusion and Further Work

While existing work has adequately addressed the concerns of (Fodor & Pylyshyn, 1988) in showing how compositional structures can be manipulated within a connectionist framework, much work remains to be done in order to determine the most reliable and efficient way of doing so, especially when the structures involved become large and complex.

We have shown that RAAM networks with appropriate modifications can reliably store and retrieve compositional structures commensurate in depth and complexity with real-world linguistic data. However this increased capacity seems to have been gained at the expense of generalization ability.

Further work is called for: firstly, on how to preserve generalization ability while expanding capacity; secondly, comparing the efficiency of storing such complex data in a single large network with that of other approaches, for example an ensemble of smaller networks arranged in a modular or hierarchical fashion.

7 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.

8 References

Ackley, D.H., G.E. Hinton & T.J. Sejnowski, 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.

Blank, D.S., L.A. Meeden & J.B. Marshall, 1992. Exploring the Symbolic/Subsymbolic Continuum: A Case Study of RAAM, in: *Closing the Gap: Symbolism vs. Connectionism*, J. Dinsmore, ed. (Lawrence Erlbaum Associates).

Blelloch, G. & C.R. Rosenberg, 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.

Chrisman, L. 1991. Learning Recursive Distributed Representations for Holistic Computation, *Connection Science* **3**, 345–366.

Cottrell, G., P. Munro & D. Zipser, 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).

Elman, J.L. 1990. Finding Structure in Time, *Cognitive Science* **14**, 179–211.

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

Hinton, G.E., J.L. McClelland & D.E. Rumelhart, 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. & J.B. Pollack, 1990. Back propagation is sensitive to initial conditions, *Complex Systems* **4**, 269–280.

Marcus, M., B. Santorini, M.A. Marcinkiewicz, 1993. Building a large annotated corpus of English: the Penn Treebank, *Computational Linguistics* **19** (also at ftp.cis.upenn.edu/pub/treebank/doc/).

Niklasson, L.F. & T. van Gelder, 1994. Can Connectionist Models Exhibit Non-Classical Structure Sensitivity, *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 664–669.

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.

Pratt, L.Y., J.A. Mostow & C.A. Kamm, 1991. Direct Transfer of Learned Information Among Neural Networks, *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, 584–589.

Rumelhart, D.E., G.E. Hinton, G.E. & R.J. Williams, 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).

9 Appendix - Data Sets

Each of our data sets consisted of parse trees for a collection of English sentences. 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 Penn Treebank (Marcus et al., 1993).

Data Set (1)

```
(D(A(A(A N))))
((D N)(P(D N)))
(V(D N))
(P(D(A N)))
((D N)V)
((D N)(V(D(A N))))
((D(A N))(V(P(D N))))
```

Data Set (2)

```
((NP(PP NP))
    (S(VP(PP((NP(NP(NP(PP NP))))
    (NP(PP(NP(ORD NP)))))))))
(NP(VP(NP(PP(NP(PP((NP NP)
    (ADJP NP)))))))
(((SINV VP)((NP NP)(VP(NP(S VP)))))
    (S(NP(S(VP((PP(NP NP))
    (PP(ADJP NP)))))))
(S(NP(S(VP(PP((NP(PP NP))NP))))))
    (NP(VP(NP(PP(NP NP)))))
```

Data Set (3)

```
((D(AP N))(AUX(V(D N))))
((D N)(AUX(V(P(D(AP N))))))
(NP(AUX(V(NP(AUX(V NP))))))
(NP(AUX(V(AUX(V NP)))))
((D(AP N))(V((D N)(P(D N)))))
(NP(V AP))
(N(V((D N)P)))
(NP(V((D N)(P(D N)))))
(D(AP((N PP)PP)))
(D((N PP)PP))
(D(AP N))
(D(AP(AP N)))
(D(N PP))
(D(AP(AP(AP N))))
(D(((N PP)PP)PP))
(D(AP(N PP)))
(D((AP N)PP))
(N((V(D N))(P(D N))))
(D(AP((NP PP)PP)))
(NP(((V NP)PP)(CONJ((V NP)PP))))
(NP((V(CONJ V))NP))
(N(((V NP)(CONJ(V NP)))PP))
(D((AP N)(CONJ(AP N))))
(D(AP((N(P NP))(CONJ(N(P NP))))))
((D(AP NP))(CONJ(D(AP NP))))
(D((N(CONJ N))(P NP)))
(NP(V NP))
(N(P NP))
(NP(V(AP NP)))
(AP(P NP))
(D((N PP)PP))
(NP(CONJ(NP(CONJ NP))))
(V(NP PP))
(NP(I(V(NP(I VP)))))
(NP(I(V((C S)(CONJ(C S))))))
(NP(I(V(C(S(CONJ S))))))
(D(NP(NP(C(NP(I(V NP)))))))
(NP(I(V(C(NP(I(V NP)))))))
(VP(C(NP(I VP))))
(NP(I(V(C(NP(I VP))))))
(NP(C(NP(I(V(C(NP(I(V(C(NP(I(V NP))))))))))))))
(NP(I(V(D(NP(NP(C(NP(I(V(C(NP(I(V(C(NP(I(V NP)))))))))))))))))))
(NP(C(NP(I(V(NP(C(NP(C(NP(I(V NP))))))))))))
(NP(C(NP(V(NP(C(NP(V(NP(C(NP VP))))))))))))
(NP(C(NP(I(V(NP(C(NP(I(V NP)))))))))))
((NP(I(V(C(NP(I VP))))))(CONJ(NP(I VP))))
(NP(I(V(C(NP(I(V(AP(C(NP(I VP))))))))))))
(NP(I(V(C((C(NP(I VP)))(I(V NP)))))))
```

Data Set (4)

```
((NP(PP NP))(S(VP(PP((NP(NP(NP(PP NP))))(NP(PP(NP(ORD NP)))))))))
(NP(VP(NP(PP(NP(PP((NP NP)(ADJP NP)))))))
(((SINV VP)((NP NP)(VP(NP(S VP)))))(S(NP(S(VP((PP(NP NP))
    (PP(ADJP NP)))))))
(S(NP(S(VP(PP((NP(PP NP))NP))))))
(NP(VP(NP(PP(NP NP)))))
((PP(PP(NP NP)))(((NP(ADJP NP))ADVP)(S(VP NP))))
((VP((PP NP)(PP(VP(NP(ADJP NP))))))((NP(NP NP))(S(VP(S(VP((NP NP)
    (PP((NP NP)(NP(VP(S(VP((PP NP)(PP(NP((PP NP)(PP NP)))))))))))))))))
((PP((NP NP)(VP(PP NP))))(S((VP NP)(S(VP ADJP)))))
(((SINV(NP(VP(NP(PP(VP(NP(PP NP))))))))(NP(S VP)))(S((PP((NP ADJP)
    (NP(PP NP))))(NP(S(VP(ADJP(S(VP(NP VP))))))))))
(((NP NP)NP)(VP(PP NP)))
(NP((VP NP)((PP NP)(NP(SBAR(NP((X(VP(NP(PP(NP(NP(ADJP(PP NP)))))))))
    (X(X(VP(NP PP))))))))))
((NP(ADJP(PP(NP(NP(S((VP PP)(NP(S(VP NP))))))))))
    (VP(NP(PP(NP(PP(NP(NP(VP NP))))))))))
(NP(S(VP(PP(NP(S(VP(NP(WHNP(NP(S((VP VP)
    (VP(NP(PP(NP ADJP)))))))))))))))))
((NP(S(VP NP)))(S(NP(S(VP((PP NP)(PP NP)))))))
(NP(S(VP(S(VP(PP(NP(PP NP))))))))
(S((NP(ADJP(PP(NP(ADJP(PP NP))))))(S(VP(NP(PP((NP(PP NP))
    (WHNP(NP(S((VP NP)((PP(NP(PP NP)))(PP NP))))))))))))))
((NP(PP(NP NP)))(VP((NP(NP(ADJP NP)))(S(VP((NP(PP(VP(NP(PP NP)))))
    (S(VP(PP(NP NP))))))))))
(((NP(S(VP(VP(PP NP)))))(S(NP(VP((PP NP)(SBAR(NP(S(VP((ADJP(S VP))
    (SBAR(NP(S(VP ADJP))))))))))))))(NP VP))
((ADVP PP)((PP NP)((NP(ADJP NP))((VP(NP(ADJP NP)))(NP((PP NP)
    (PP NP)))))))
(NP(((VP NP)(VP NP))(X(X(VP NP)))))
(NP((VP NP)(VP((PP NP)NP))))
((NP(VP(NP(PP(VP(PP NP))))))(VP(NP(X(NP(PP(NP NP)))))))
(NP(VP(NP(SBAR(NP(S(VP(NP(PP NP))))))))))
(NP(NP((VP(PP NP))((PP NP)(PP(NP(PP NP)))))))
((SBAR(VP(NP(PP NP))))(NP(VP((VP((NP NP)(S(VP(S(VP NP))))))
    (VP((VP NP)((PP(NP NP))PP)))))))
((NP(PP(NP NP)))(VP(NP(PP(NP(VP(NP NP)))))))
((NP(S(VP(VP(NP((WHNP(NP((VP NP)(VP(PP NP)))))
    (VP(VP(VP(ADJP(PP NP))))))))))(VP(NP(NP NP))))
(NP(VP(NP(S(VP(S(VP(NP(SBAR(NP VP))))))))))
(ADJP((VP(NP NP))((NP(S(VP NP)))(S(NP(S(VP(NP(PP NP))))))))))
(S(NP VP))
(ADJP(VP ADJP))
((NP(S(VP(PP NP))))(NP(S(VP NP))))
((S(NP(S(VP(NP NP))))))(S(ADJP(VP(S(VP(PP NP)))))))
((PP((NP(PP(NP(PP(NP NP)))))((NP((VP NP)(VP(NP(PP NP)))))(NP NP))))
    (((NP(PP NP))(VP(PP NP)))(S(VP(PP NP)))))
(NP(VP(NP(VP(NP(S((VP ADJP)((PP(PP NP))PP))))))))
(NP((NP VP)(NP(PP NP))))
((NP(NP(NP NP)))(VP(NP(VP(NP((PP NP)((PP(NP(NP NP)))
    (PP(PP(ADJP NP)))))))))
```