We turn first to a very simple problem in which the system is induced to invent a shift register to solve the problem.

**Learning to be a shift register** Perhaps the simplest class of recurrent problems we have studied is one in which the input and output units are one and the same and there are no hidden units. We simply present a pattern and let the system process it for a period of time. The state of the system is then compared to some target state. If it hasn't reached the target state at the designated time, error is injected into the system and it modifies its weights. Then it is shown a new input pattern and restarted. In these cases, there is no constraint on the connections in the system. Any unit can connect to any other unit. The simplest such problem we have studied is what we call the shift register problem. In this problem, the units are conceptualized as a circular shift register. An arbitrary bit pattern is first established on the units. They are then allowed to process for two time-steps. The target state, after those two time-steps, is the original pattern shifted two spaces to the left. The interesting question here concerns the state of the units between the presentation of the start state and the time at which the target state is presented. One solution to the problem is for the system to become a shift register and shift the pattern exactly one unit to the left during each time period. If the system did this then it would surely be shifted two places to the left after two time units. We have tried this problem with groups of three or five units and, if we constrain the biases on all of the units to be negative (so the units are off unless turned on), the system always learns to be a shift register of this sort.10 Thus, even though in principle any unit can connect to any other unit, the system actually learns to set all weights to zero except the ones connecting a unit to its left neighbor. Since the target states were determined on the assumption of a circular register, the left-most unit developed a strong connection to the right-most unit. The system learns this relatively quickly. With \( \eta = 0.25 \) it learns perfectly in fewer than 200 sweeps through the set of possible patterns with either three- or five-unit systems.

The tasks we have described so far are exceptionally simple, but they do illustrate how the algorithm works with unrestricted networks. We have attempted a few more difficult problems with recurrent networks. One of the more interesting involves learning to complete sequences of patterns. Our final examples comes from this domain.

**Learning to complete sequences** Table 10 shows a set of 25 sequences which were chosen so that the first two items of a sequence uniquely determine the remaining four. We used this set of sequences to test out the learning abilities of a recurrent network. The network consisted of five input units (A, B, C, D, E), 30 hidden units, and three output units (1, 2, 3). At time 1, the input corresponding to the first item of the sequence is turned on and the other input units are turned off. At Time 2, the input unit for the second item in the sequence is turned on and the others are all turned off. Then all the input units are turned off and kept off for the remaining four steps of the forward iteration. The network must learn to make the output units adopt states that represent the rest of the sequence. Unlike simple feedforward networks (or their iterative equivalents), the errors are not only assessed at the final layer or time. The output units must adopt the appropriate states during the forward iteration, and so during the back-propagation phase, errors are injected at each time-step by comparing the remembered actual states of the output units with their desired states.

The learning procedure for recurrent nets places no constraints on the allowable connectivity structure.11 For the sequence completion problem, we used one-way connections from the input units to the hidden units and from the hidden units to the output units. Every hidden unit had a one-way connection to every other hidden unit and to itself, and every output unit was also connected to every other output unit and to itself. All the connections started with small random weights uniformly distributed between \(-0.3\) and \(+0.3\). All the hidden and output units started with an activity level of 0.2 at the beginning of each sequence.

We used a version of the learning procedure in which the gradient of the error with respect to each weight is computed for a whole set of examples before the weights are changed. This means that each connection

---

10 If the constraint that biases be negative is not imposed, other solutions are possible. These solutions can involve the units passing through the complements of the shifted pattern or even through more complicated intermediate states. These trajectories are interesting in that they match a simple shift register on all even numbers of shifts, but do not match following an odd number of shifts.

11 The constraint in feedforward networks is that it must be possible to arrange the units into layers such that units do not influence units in the same or lower layers. In recurrent networks this amounts to the constraint that during the forward iteration, future states must not affect past ones.
we consider it to be an important research problem to clarify the
results of recent studies on this topic. The results of recent studies on this topic
contradicts the findings of previous research. The data presented in this section
indicates that the results obtained from the current study are consistent with the
findings of previous research.

The data presented in this section is based on a comprehensive review of the
literature and a large dataset that was collected through a variety of methods
including surveys, interviews, and experiments.

Conclusion

Despite the limitations of the current study, the results presented in this section
provide valuable insights into the topic under investigation. The findings of this
study can contribute to the development of a more comprehensive understanding
of the topic and can serve as a basis for further research.

Table 11: Performance of the network on the novel test sequences

<table>
<thead>
<tr>
<th>Network</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVA112</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>EVA113</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>EVA114</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>EVA115</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Note: The values shown in the table represent the performance of the network on the novel test sequences.
date (or reject) our intuitive judgement that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting "learning theorem" for the multilayered machine will be found. (P. 231–232)

Although our learning results do not guarantee that we can find a solution for all solvable problems, our analyses and results have shown that as a practical matter, the error propagation scheme leads to solutions in virtually every case. In short, we believe that we have answered Minsky and Papert’s challenge and have found a learning result sufficiently powerful to demonstrate that their pessimism about learning in multilayer machines was misplaced.

One way to view the procedure we have been describing is as a parallel computer that, having been shown the appropriate input/output exemplars specifying some function, programs itself to compute that function in general. Parallel computers are notoriously difficult to program. Here we have a mechanism whereby we do not actually have to know how to write the program in order to get the system to do it. Parker (1983) has emphasized this point.

On many occasions we have been surprised to learn of new methods of computing interesting functions by observing the behavior of our learning algorithm. This also raised the question of generalization. In most of the cases presented above, we have presented the system with the entire set of exemplars. It is interesting to ask what would happen if we presented only a subset of the exemplars at training time and then watched the system generalize to remaining exemplars. In small problems such as those presented here, the system sometimes finds solutions to the problems which do not properly generalize. However, preliminary results on larger problems are very encouraging in this regard. This search is still in progress and cannot be reported here. This is currently a very active interest of ours.

Finally, we should say that this work is not yet in a finished form. We have only begun our study of recurrent networks and sigma-pi units. We have not yet applied our learning procedure to many very complex problems. However, the results to date are encouraging and we are continuing our work.