Using Periodically Attentive Units to Extend the Temporal Capacity of Simple Recurrent Networks

A thesis presented to the Faculty
of the University at Albany, State University of New York
in partial fulfillment of the requirements
for the degree of

Master of Science
College of Arts & Sciences
Department of Computer Science

Thomas C. O'Connell
1995

ACKNOWLEDGMENTS

I would like to thank Professor George Berg for his advisement and encouragement throughout this project. I would also like to thank John Munson, Mike Polumbo, and Bill Rennie for many discussions related to this work. John Munson and George Berg provided much of the code necessary to perform the experiments contained herein. Ellen O'Connell contributed several suggestions that made this thesis more readable. I performed this work while on an educational leave of absence from IBM Corporation. I am grateful to Craig Stein, Debbie Dunn, Jack Higbee and Alan Shaffer for helping me to secure this leave and to Art Palmiotti for making the process related to this leave as painless as possible. Finally, I would like to thank Professor Rich McGovern of Marist College for his very timely support over the last nine years.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS ii

LIST OF ILLUSTRATIONS iv

Chapter

1. INTRODUCTION 6

2. ANALYSIS OF THE PROBLEM 11

3. PERIODICALLY ATTENTIVE UNITS 16

4. EXPERIMENTS WITH THE EMBEDDED REBER GRAMMAR . . . 19

5. EXPERIMENTS WITH LONG STRINGS 32

6. EXPERIMENTS WITH TIME-VARYING INDICATOR SYMBOLS . 36

7. BACKPROPAGATION THROUGH TIME 39

8. EXPERIMENTS WITH BPTT 48

9. RELATED WORK 54

10. CONCLUSION 59

REFERENCES 60

LIST OF ILLUSTRATIONS

Figure		Page
1.1.	Simple Recurrent Network Architecture	6
1.2.	Finite State Diagram for the Reber Grammar	7
1.3.	Finite State Diagram for the embedded Reber Grammar	9
2.1.	Comparison of hidden activations of an SRN with random weights over 100 complementary pairs of strings	15
4.1.	Pattern of attention for the 7 PA units	20
4.2.	Test results for 20 SRNs and PA networks on the embedded Reber Grammar	22
4.3.	Incremental Success Rates for the 6 best PA networks	23
4.4.	Activations for each unit in PA network 20 on strings BPPVVP and BTPVVT	24
4.5.	Activations for each unit in PA network 20 on string BPTSXXTVPXVPXTVVP	26
4.6.	Unit 1's context activations for 5 failure cases	27
4.7.	Recurrent weights for PA network 20	29
4.8.	Input and output weights for PA network 20	30
4.9.	Thetas for PA network 20	31
5.1.	Finite-state diagram for the embedded Reber grammar with altered state transition probabilities indicated	33
5.2.	Tests of PA networks on long strings	35
6.1.	Finite State Diagram for the embedded Reber grammar with time-varying indicator symbols	37
6.2.	Test results for SRNs and PA networks on the embedded Reber grammar with time varying indicator symbols	38

Figure Page

7.1. Backpropagation through time example 40

7.2. SRN unfolded in time viewed as BPTT(2,1) 44

7.3. Network from Figure 7.1 where unit 1 is now a PA unit 45

8.1. Test results for BPTT on embedded Reber grammar . . 49

8.2. BPTT(4,1) tested on long strings 50

8.3. BPTT(5,1) tested on long strings 51

8.4. Test results for PA networks trained with BPTT on the embedded Reber grammar 52

8.5. Test results for BPTT on embedded Reber with time-varying indicator symbols 53

9.1. A time-delay unit with 3 delay lines 56

9.2. Input sequence seen through each delay line of a time-delay unit with 3 delay lines 56

9.3. A TDNN duplicated 3 times with windows offset by 1 time step 57

CHAPTER 1

INTRODUCTION

A Simple Recurrent Network (SRN, Elman 1989) is an artificial neural network (Rumelhart, McClelland, et al. 1986) in which the activations of the hidden units at time t are input to the network at time $t+1$ (Figure 1.1). These recurrent connections give the network access to its previous state and consequently the ability to detect and learn temporal relationships in the data.

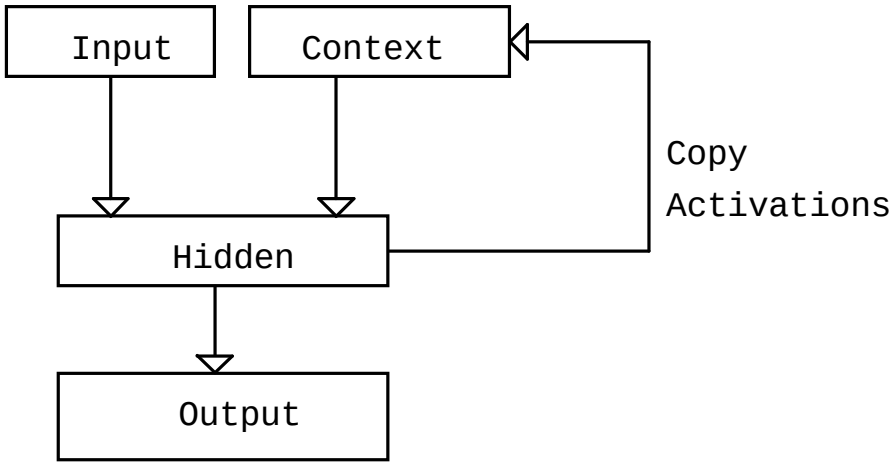


Figure 1.1. Simple Recurrent Network Architecture.

Elman (1989) used the SRN to solve sequence prediction tasks, i.e., tasks in which the network is presented a sequence of symbols one at a time and is required to predict the next symbol in the sequence at each time step. Cleeremans et al. (1989) formulated the recognition of regular languages[1] as a sequence prediction task and achieved excellent results for a language produced by a regular grammar known as the Reber grammar (Figure 1.2).

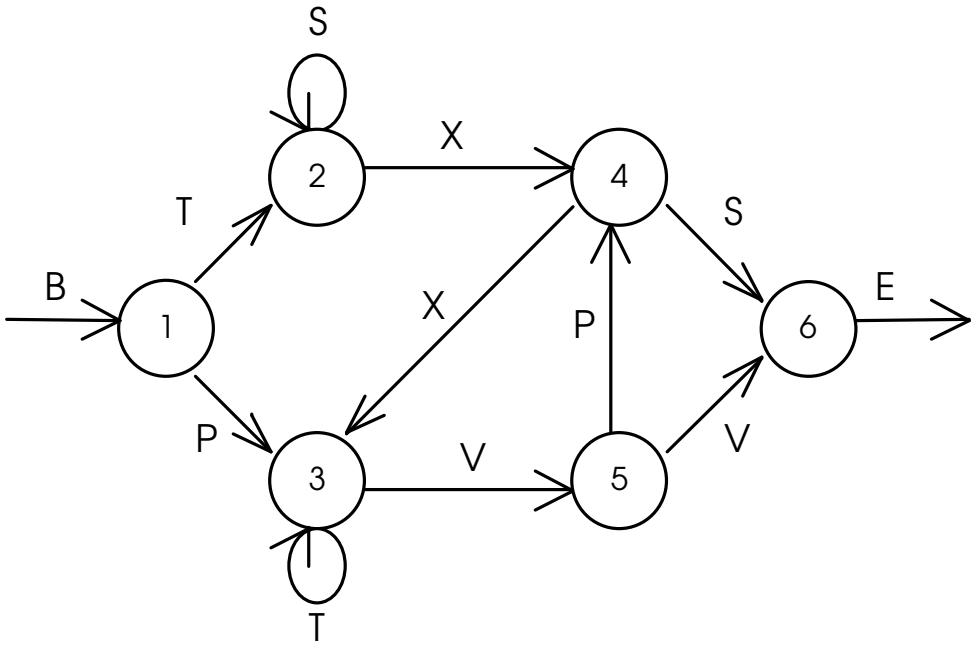


Figure 1.2. Finite State Diagram for the Reber Grammar.

[1]For formal definitions of regular languages and grammars see Hopcoft and Ullman(1979).

In Cleeremans et al. (1989), each symbol was represented by one bit in an n-bit vector where n was the number of symbols in the alphabet. A string was accepted if each input symbol was predicted as a legal successor to the previous input. A symbol was considered predicted if the activation of the output unit corresponding to that symbol was greater than 0.3 (where the activation was in the range [0.0, 1.0]). Their experiments with the Reber grammar consisted of training an SRN with 15 hidden units on 60,000 training strings. These strings were generated randomly as training proceeded with each of the two possible successor symbols at any state being equally likely. There was no attempt to make the 60,000 training strings distinct. In all likelihood, there were many repetitions in the training data. Test results showed that the SRN learned the grammar perfectly, accepting grammatical strings, even those of length greater than 100, and rejecting ungrammatical strings.

The same architecture, however, proved to be incapable of learning an embedded version of the Reber grammar (Figure 1.3). In this grammar, the first symbol after the B serves as an indicator which uniquely determines the last symbol in the string before the E. Embedded between these two symbols are strings generated by the Reber grammar. Since there was only one correct prediction for the last symbol in each string, the last symbol was considered predicted by

Cleeremans et al. if the ratio of the activation of the appropriate output unit to the sum of the activations of all output units was greater than 0.6. Even after extensive training, 2.4 million symbol presentations, the network failed to learn to differentiate between complementary strings (i.e., strings PxP and TxT where x is a string generated by the Reber grammar).

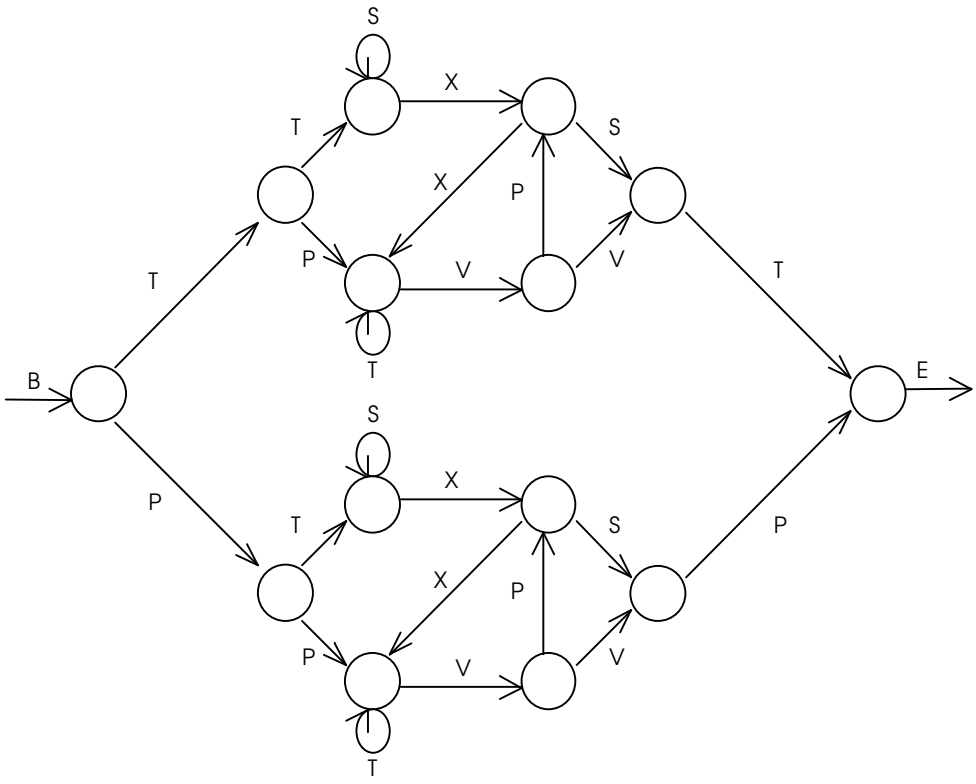


Figure 1.3. Finite State Diagram for the embedded Reber Grammar.

Although learning to recognize strings generated by a regular grammar is the problem chosen for analysis, this paper is not primarily concerned with the problem of learning grammars from examples[2], rather, the problem is used to illustrate the strengths and weaknesses of SRNs in learning temporal dependencies. This paper explores some of the reasons that the temporal characteristics of the embedded Reber grammar prove so difficult for SRNs to learn and attempts to develop an extension to the SRN architecture that addresses some of its shortcomings.

---

[2]See Omlin and Giles(1994) and the references therein for a discussion of the application of recurrent networks to the problem of grammar induction.

CHAPTER 2

ANALYSIS OF THE PROBLEM

A closer look at the two grammars in question shows that in the Reber grammar each input symbol can be predicted solely on the basis of the previous two inputs. For example, if the last two symbols were TS then we must be in state number 2 (Figure 1.2) and, therefore, the next symbol must be an X or an S. In the embedded Reber grammar, the indicator symbol must be remembered over the entire varying-length string in order to predict the last symbol. The difficulty the network faces in remembering the indicator symbol is that as each subsequent input symbol is presented the effect of the indicator symbol on the hidden activations attenuates. For the network to maintain the hidden activations caused by the indicator symbol the weights from the context layer to the hidden layer must be relatively large compared to the weights from the input layer to the hidden layer. In this way, the previous hidden activations play a stronger role than the input in determining the next hidden activations and the attenuation is consequently reduced.

Since the initial random weights are small, early in training it is unlikely that the hidden activations at the end of a string will significantly reflect the indicator symbol. The error generated early in training, therefore, is unlikely to significantly reflect the long term dependency in the data. Since the short term prediction part of the task, the Reber grammar, requires only that the network remember the last two symbols, this part of the task will account for a much larger proportion of the error in the early stages of training. This proportion is further increased because short term error is generated at each time step whereas long term error is generated only once per string.

Therefore, for the network to detect the long term dependency it must develop large enough recurrent weights for the final hidden activations to reflect the indicator symbol and its means of developing these weights is by learning the short term prediction task. Assuming the recurrent weights ever do become large enough for the final hidden activations to significantly reflect the indicator symbol, by the time this occurs, the network may have committed so much of its resources to the short term prediction task that it is impossible for the network to develop the weights necessary to perform the long term prediction task. In other words, the network may move into

a local minimum in error versus weight space before even being able to detect the long term dependency in the data.

This conflict between long and short term resource allocation may be further exacerbated by the fact that successful long term prediction cannot occur without successful short term prediction. The network must learn the Reber grammar well enough to know when it is at the last state so that it can then uniquely determine the next symbol based on its memory of the indicator symbol.

To get an idea of how much of the short term task is learned before the differences in the hidden unit activations become significant a random network was trained on 100,000 randomly generated strings.[1] After each 10,000 strings the network was tested on 100 complementary pairs of strings and the differences in hidden activations for each of these pairs were recorded. Figure 2.1 shows the percentage of strings that the network successfully predicted, the average sum of the differences between hidden unit activations for each pair and the average greatest difference in any single unit over the 100 strings. As expected, the differences for the random initial network are very small with an average greatest difference less than 0.001. Even after 10,000 string presentations the average greatest difference is still less than 0.01. After 90,000

---

[1]As in Cleeremans et al. (1989), these strings are not distinct.

string presentations the network has learned the short term dependencies well enough to successfully predict all the embedded symbols in each of the 200 strings but it has yet to learn the long term dependencies well enough to successfully predict the last symbol in any of the strings. This data lends further support to the conjecture that a significant amount of short term learning is occurring before the long term dependency is detected.

Number	embed	final	avg sum	avg max
0	0.0	0.0	0.00430	0.00065
10,000	0.0	0.0	0.03186	0.00547
20,000	40.0	0.0	0.09872	0.01860
30,000	57.5	0.0	0.09840	0.02049
40,000	23.5	0.0	0.07317	0.01848
50,000	69.5	0.0	0.06907	0.01760
60,000	90.5	0.0	0.06002	0.01536
70,000	89.5	0.0	0.05663	0.01470
80,000	93.5	0.0	0.06184	0.01632
90,000	100.0	0.0	0.06332	0.01672
100,000	100.0	0.0	0.06840	0.01823

Figure 2.1. Comparison of hidden activations of an SRN with random weights over 100 complementary pairs of strings.

Number = number of training strings presented to this point.

embed = % of test strings in which all of the embedded symbols were successfully predicted

final = % of test strings in which the last symbol was successfully predicted

$$\text{avg sum} = \frac{\sum_{j=1}^{100}\left(\sum_{i=1}^{n}\left|u_i(Px_j)-u_i(Tx_j)\right|\right)}{100}$$

$$\text{avg max} = \frac{\sum_{j=1}^{100}\left(\max\left\{\left|u_1(Px_j)-u_1(Tx_j)\right|,\ldots,\left|u_n(Px_j)-u_n(Tx_j)\right|\right\}\right)}{100}$$

where:

$u_i(s)$ is the activation of unit i after seeing input string s

Px_j is the string formed by symbol P followed by string x_j

Tx_j is the string formed by symbol T followed by string x_j

n is the number of hidden units.

CHAPTER 3

PERIODICALLY ATTENTIVE UNITS

Given that the SRN is apparently forced to learn short term dependencies first, what can be done to improve an SRNs ability to learn long term dependencies without resorting to more computationally expensive learning algorithms such as Backpropagation Through Time (BPTT, Rumelhart, Hinton, and Williams 1986) or Real Time Recurrent Learning (Williams and Zipser 1989)? The modification proposed here is to limit the time steps at which each hidden unit receives input. During the time period in which a unit is not receiving input, its activation will remain unchanged. For example, if a unit's activation is 0.3 then its activation will remain 0.3 until it receives another input. These units will be referred to as periodically attentive (PA) units. In this research we will consider PA units that receive input at some regular interval. For example, suppose unit 1 is attentive at time 1 and every fourth time step thereafter. If the input string is $S_1S_2S_3S_4S_5S_6S_7S_8S_9S_{10}S_{11}$, then this unit will see the input string $S_1S_5S_9$. If this unit's function is to remember the first input over the length of the string then the number of inputs that interfere with

this function has been cut from 10 to 2. Clearly, a smaller recurrent weight will be required for this unit's final activation to significantly reflect the first input symbol. This should enable the network to differentiate strings based on the indicator symbol earlier in training.

Another benefit of PA units in learning the embedded Reber grammar is that if the duration of inattention is greater than 3, these units will be of little use in the short term prediction part of the task. This effectively imposes a partition of the units into long and short term predictors rather than leaving this to the network to determine. Thus, even if the network must learn a significant portion of the short term prediction task prior to detecting the long term dependency these units should still be available since they cannot have been satisfactorily allocated as short term predictors. Periodically attentive units therefore address both the problem of dependence on the strength of the recurrent connections and the conflict between long and short term resource allocation.

During training, error will be propagated normally except that if a unit is not currently attentive, it will not pass error back to the previous layer and the weights into the unit will not be adjusted. Clearly, this causes the error calculations to stray further from the true gradient which may result in instability during training.

The alternative would be to record the input and context activations at each time step that a PA unit is receiving input. These activations would then have to be retained over the entire period of inattention so that the weights into the PA unit could be adjusted during this period. Such a training scheme would seriously reduce the computational benefits of avoiding BPTT and will not be investigated here.

CHAPTER 4

EXPERIMENTS WITH THE EMBEDDED REBER GRAMMAR

Several experiments were conducted to compare the standard SRN architecture with the SRN containing periodically attentive units[1]. The networks consisted of 7 input, 15 hidden and 7 output units as in Cleeremans et al. (1989). The activations of each unit in the network were updated using the standard logistic activation function $f(net) = \frac{1}{1 + e^{-net}}$ (Rumelhart et al. 1986). Twenty random weight sets were created with each weight being in the range [-1.0, 1.0]. Each network was trained on 2.4 million randomly generated strings. At each time step the next symbol was randomly chosen from the two legal successor symbols with equal probability except in the case of the last symbol where there is only one choice. There was no attempt to avoid duplicate strings during training. Given the nature of the grammar and the string generation process, it is likely that most of the strings generated were relatively

[1]In the remainder of this paper, SRN will be used to refer to the standard SRN architecture trained with the usual learning algorithm while PA network will be used to refer to the SRN with PA units.

short and that many of these strings appeared repeatedly throughout training. If a network has not learned the grammar after such exhaustive training, we can be relatively certain that it is incapable of learning rather than merely slow. The learning rate was set to 0.01 and the momentum term was set to 0.3. Larger learning rates and smaller momentum terms seemed to have little effect on the SRNs but caused the PA networks to perform poorly perhaps due to the additional instability created by blocking error propagation at inattentive units. PA networks achieved their best results using eight normal hidden units and seven PA hidden units which were attentive for a single time step every seven time steps (Figure 4.1).

time step	unit 0123456
0	A000000
1	0A00000
2	00A0000
3	000A000
4	0000A00
5	00000A0
6	000000A
7	A000000
8	0A00000
etc.	

Figure 4.1. Pattern of attention for the 7 PA units.

A - indicates attentive
0 - indicates inattentive

So as not to overly bias the networks toward the embedded Reber grammar the pattern of attention was chosen

under the assumption that the indicator symbol could occur at any time step rather than only at time step 1. Thus the durations of inattention are such that there is always one PA unit receiving input at each time step. Hopefully, this restriction gives us a pattern of attention that is useful for tasks other than the embedded Reber grammar[2].

Each trained network was tested on 1000 randomly generated strings. The test strings were generated in the same manner as the training strings except that duplicate test strings were removed. The training and test sets were not forced to be mutually exclusive (and almost certainly were not). The average length of a string in the test data set was 16.4. The maximum length of a test string was 34 and the minimum length was 6. The results (Figure 4.2) show that the PA network is far superior to the SRN in performing long term prediction. Although it did not satisfactorily learn the long term dependency in a majority of the trials, the PA network was clearly able to detect this dependency, achieving a success rate of over 50% in almost half of the trials and over 85% in six of the trials. The SRN, on the other hand, proved to be almost completely incapable of learning the long term dependency as had been shown by Cleeremans et al. (1989).

---

[2]As we shall see in Chapter 6, however, the rigidity of the pattern of attention does appear to limit the applicability of periodically attentive units.

	SRN		PA Network	
Network	Embed	Final	Embed	Final
1	92.5	15.3	68.6	53.2
2	100.0	0.1	100.0	0.0
3	98.3	23.5	98.8	1.7
4	100.0	0.7	97.6	8.9
5	100.0	2.3	93.2	53.4
6	99.2	11.4	100.0	**90.5**
7	100.0	10.0	100.0	**90.7**
8	98.6	0.1	100.0	**87.8**
9	100.0	2.0	67.4	21.4
10	100.0	13.0	100.0	44.7
11	100.0	7.4	100.0	59.9
12	100.0	10.1	91.0	14.5
13	100.0	18.2	99.8	**92.9**
14	100.0	1.9	99.1	**90.8**
15	88.7	2.1	90.0	44.6
16	97.9	6.8	93.6	36.1
17	100.0	0.6	100.0	0.0
18	100.0	1.3	98.0	11.3
19	96.9	1.1	99.5	24.9
20	100.0	13.4	100.0	**99.5**
average	98.6	7.1	94.8	46.3

Figure 4.2. Test results for 20 SRNs and PA networks on the embedded Reber Grammar.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

Parameters:

initial weight range = [-1.0, 1.0]
learning rate = 0.01
momentum = 0.3
number of training strings = 2.4 million
number of test strings = 1000
average test string length = 16.4
maximum test string length = 34
minimum test string length = 6

In order to determine if over training adversely affected the networks' ability to generalize to new strings, each network was tested after the presentation of every 400,000 input strings. Figure 4.3 shows the results for the six most successful PA networks. Only network 13 suffered a degradation in performance as training continued. Networks 6, 8 and 20 continued to improve while networks 7 and 14 remained remarkably consistent.

	PA net 6		PA net 7		PA net 8	
Number	Embed	Final	Embed	Final	Embed	Final
4	94.3	75.3	100.0	87.5	100.0	0.0
8	100.0	85.6	99.1	90.7	81.1	28.7
12	99.2	82.5	100.0	90.5	100.0	75.8
16	100.0	87.9	100.0	90.7	100.0	81.9
20	99.6	88.3	100.0	90.7	100.0	83.5
24	100.0	90.5	100.0	90.7	100.0	87.8

	PA net 13		PA net 14		PA net 20	
Number	Embed	Final	Embed	Final	Embed	Final
4	94.0	91.7	90.0	88.1	89.8	15.7
8	99.7	97.1	98.3	90.7	94.8	47.6
12	99.4	98.6	96.9	91.4	81.3	80.9
16	99.6	98.0	100.0	90.7	100.0	97.9
20	99.3	96.0	99.6	91.2	100.0	99.1
24	99.8	92.9	99.2	90.8	100.0	99.5

Figure 4.3. Incremental Success Rates for the 6 best PA networks.

Number = String presentation number in 100,000s

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which the final symbol was correctly predicted.

The activations of network 20 were recorded (Figure 4.4) on several test strings to determine how the network was using the PA units to achieve such a high success rate.

Input EVPXSTB		Period 0123456	Hidden 012345678901234	Actual EVPXSTB	Target EVPXSTB	
000000A	[B]	A000000	0**5**55555A0070890	0140050	00000A0	[T]
00000A0	[T]	0A00000	0**2**5555510109A01	0050050	00A0000	[P]
00A0000	[P]	00A0000	0**2**95555A09018A9	0500050	0A00000	[V]
0A00000	[V]	000A000	0**2**91555AAAA0A88	0640000	0A00000	[V]
0A00000	[V]	0000A00	0**2**91555A03A0A17	00000A0	00000A0	[T]
00000A0	[T]	00000A0	0**2**9155590419110	A000000	A000000	[E]

Input EVPXSTB		Period 0123456	Hidden 012345678901234	Actual EVPXSTB	Target EVPXSTB	
000000A	[B]	A000000	0**5**55555A0070890	0140050	00A0000	[P]
00A0000	[P]	0A00000	0**8**5555590009A98	0050050	00A0000	[P]
00A0000	[P]	00A0000	0**8**75555A09019A9	0500050	0A00000	[V]
0A00000	[V]	000A000	0**8**71555AAAA0A88	0640000	0A00000	[V]
0A00000	[V]	0000A00	0**8**71555AA1A0A16	00A0000	00A0000	[P]
00A0000	[P]	00000A0	0**8**71575A001A032	A000000	A000000	[E]

Figure 4.4. Activations for each unit in PA network 20 on strings BPPVVP and BTPVVT.

Activations appear rounded in tenths with an activation of A indicating 1.0.

Period = Pattern of attention for first 7 hidden units.
A - attentive.
0 - inattentive.

We can see that hidden unit number 1, a PA unit that is attentive when the indicator symbol is presented, becomes active when the indicator symbol is a P and inactive when the indicator symbol is a T. This unit can then act as a cue for the final symbol prediction if it maintains this activation over the length of the string. In both cases from Figure 4.4, unit 1 does maintain its activations until the last time step and the final symbol is correctly predicted. However, if we look at another example (Figure 4.5), we see that even though this unit maintains its activation only up through the time step prior to the time step in which the long term prediction must occur, the network still correctly predicts the final symbol. This activation, though, will appear in the context at the time the final prediction is to be made and therefore maintaining the activation to this point should be sufficient for successful prediction.

Input EVPXSTB		Period 0123456	Hidden 012345678901234	Actual EVPXSTB	Target EVPXSTB	
000000A	[B]	A000000	0555555A0070890	0140050	00A0000	[P]
00A0000	[P]	0A00000	085555590009A98	0050050	00000A0	[T]
00000A0	[T]	00A0000	081555590901721	0104500	0000A00	[S]
0000A00	[S]	000A000	081A555AA530A09	0015500	000A000	[X]
000A000	[X]	0000A00	081A055A0190000	0005501	000A000	[X]
000A000	[X]	00000A0	081A02520910900	0501150	00000A0	[T]
00000A0	[T]	000000A	081A02690991A10	0600040	0A00000	[V]
0A00000	[V]	A000000	081A026AAAA0A97	0640000	00A0000	[P]
00A0000	[P]	0A00000	081A026A0040034	0004500	000A000	[X]
000A000	[X]	00A0000	081A02620900900	0600150	0A00000	[V]
0A00000	[V]	000A000	081A026AAAA0A96	0640000	00A0000	[P]
00A0000	[P]	0000A00	081A026A0040034	0005600	000A000	[X]
000A000	[X]	00000A0	081A01620900900	0601150	00000A0	[T]
00000A0	[T]	000000A	081A01690A91A10	0600040	0A00000	[V]
0A00000	[V]	A000000	0**8**1A016AAAA0A97	0640000	0A00000	[V]
0A00000	[V]	0A00000	0**1**1A016AA1A0A26	00A0000	00A0000	[P]
00A0000	[P]	00A0000	012A016A000A042	A000000	A000000	[E]

Figure 4.5. Activations for each unit in PA network 20 on string BPTSXXTVPXVPXTVVP.

Activations appear rounded in tenths with an activation of A indicating 1.0.

Period = Pattern of attention for first 7 hidden units
A - attentive.
0 - inattentive.

Figure 4.6 shows unit 1's final context activations produced by the 5 strings for which PA network 20 failed to correctly predict the final symbol. Of the 490 test strings beginning with P, strings 1, 2 and 3 were the only such strings that produced final context activations for unit 1 below 0.45. Only 2 of the 510 strings beginning with a T produced final context activations for unit 1 above 0.45. String 4 produced the highest such activation of these two

strings. String 5 remains somewhat of an anomaly since there were four strings for which the final P was successfully predicted but which produced final context activations below 0.545. However, the actual activations that string 5 produced shows that this string was not far from being considered a success. In fact, had the success criteria been highest activation rather than ratio of activation above 0.6 both strings 4 and 5 would have been considered successes.

String	Symbol	Length	Unit 1 Context	Actual EVPXSTB
1	P	27	0.256	0011190
2	P	26	0.341	0010090
3	P	21	0.441	0041170
4	T	21	0.486	0031150
5	P	20	0.545	0061130

Figure 4.6. Unit 1's context activations for 5 failure cases.

Symbol = starting and ending symbol of the string

Length = length of the string

Unit 1 Context = context activation of unit 1 at the time of final prediction.

Actual = actual output activation produced as final prediction.

Network 20's weights and thetas[3] are given in Figures 4.7 through 4.9. From Figure 4.7 we can see that unit 1 is able to maintain its activation with a moderately strong self-recurrent weight of 3.2. It also has an extremely strong connection to hidden unit 8. Figure 4.8 shows that hidden unit 8 has a strong positive connection to the output unit corresponding to P and a strong negative connection to the output unit corresponding to T. Unit 8, therefore, is likely to play a major role in final symbol prediction. This gives us further evidence that the success of the network depends on unit 1's sustained activation. The exact mechanism which the network uses to make these predictions is not essential. For our purposes, it suffices to note that the appropriate PA unit does develop into a detector for the indicator symbol and this unit's activation is used to make the final prediction.

---

[3]See Rumelhart et al. (1986) for a description of the theta term and its effect on the activation function.

		from PA unit number						
		0	1	2	3	4	5	6
	0:	-0.4	-0.3	0.1	-1.1	-0.1	0.5	0.1
	1:	0.0	**3.2**	-0.7	-0.6	0.1	0.8	0.2
	2:	-1.2	-0.1	2.1	-1.8	0.3	0.2	-0.3
to	3:	0.2	-0.6	-2.5	5.0	-0.6	-0.1	-0.5
hidden	4:	0.3	0.1	0.6	-3.6	0.5	0.2	0.0
unit #	5:	0.3	-0.2	0.6	-1.2	0.8	0.2	-0.8
	6:	0.5	0.7	0.7	0.2	-1.0	-0.7	0.5
	7:	0.2	-0.8	0.1	-0.8	1.1	0.1	0.7
	8:	-2.2	**14.2**	-7.3	-3.1	3.9	-0.0	-1.0
	9:	-1.7	-1.3	-2.1	-1.6	-0.5	0.3	-0.0
	10:	1.0	1.9	0.9	2.7	1.8	-0.6	-0.0
	11:	0.2	-0.7	-1.1	0.2	-0.3	0.1	1.1
	12:	1.2	0.5	0.3	-2.4	2.3	-0.2	0.9
	13:	0.7	-0.1	-1.5	1.3	3.5	0.2	-0.1
	14:	-1.5	-0.9	-0.9	-0.8	0.9	-0.2	-0.1

a) weights from periodically attentive hidden to hidden

		from hidden unit #							
		7	8	9	10	11	12	13	14
	0:	-1.1	-0.7	-0.3	0.2	-1.3	-0.6	0.3	-0.5
	1:	-1.0	-0.8	-0.6	1.6	0.7	0.0	-2.4	-0.9
	2:	0.7	-0.6	-0.1	-0.2	0.9	0.5	-1.3	-0.4
to	3:	-0.7	-0.0	1.1	0.3	-1.0	1.8	0.8	-1.9
hidden	4:	-0.2	-0.3	-0.0	0.0	1.0	-0.6	1.0	-0.3
unit #	5:	-0.1	-0.0	0.4	1.1	-0.5	0.1	-0.7	-0.6
	6:	-1.0	-0.4	-0.2	0.7	0.2	-0.0	-0.8	0.1
	7:	-0.6	0.2	3.0	-0.4	3.8	1.2	-2.0	1.1
	8:	-1.4	-7.0	0.9	-3.4	1.8	-0.8	2.0	-3.4
	9:	0.9	-7.3	5.4	-1.4	4.4	-0.3	-1.2	0.8
	10:	-0.9	2.2	3.6	0.2	-2.6	-0.9	-2.4	0.8
	11:	2.0	0.2	-7.5	-0.8	-6.1	2.0	-2.4	0.3
	12:	2.5	-8.7	4.6	-3.2	-1.3	-0.9	6.7	-8.7
	13:	-0.2	-3.1	0.4	0.2	1.1	0.8	-1.0	0.7
	14:	0.4	-0.7	0.4	-0.3	2.3	0.8	1.7	-1.6

b) weights from normal hidden to hidden

Figure 4.7. Recurrent weights for PA network 20.

		input unit symbol and number						
		[E]	[V]	[P]	[X]	[S]	[T]	[B]
		0	1	2	3	4	5	6
	0:	-0.9	-0.5	1.2	0.6	0.5	0.1	-0.6
	1:	-0.2	-0.0	2.7	0.2	0.1	-0.3	0.1
	2:	0.3	0.0	1.4	0.2	-0.9	-1.6	0.7
	3:	0.8	-1.8	-0.1	2.1	2.1	-2.3	-0.8
to	4:	0.1	0.3	0.3	-0.3	-0.2	0.4	0.9
hidden	5:	-0.1	0.3	0.7	0.1	-0.8	-0.3	0.0
unit #	6:	0.1	-0.5	0.1	0.1	1.3	-0.5	-0.8
	7:	-0.0	1.4	2.3	-0.6	-0.3	-1.5	0.4
	8:	0.7	**14.9**	-5.1	-5.8	**4.6**	-7.4	-4.7
	9:	0.1	3.4	0.1	3.9	-4.2	0.3	-1.8
	10:	-0.9	4.2	-3.4	-1.9	-1.9	-0.2	0.8
	11:	-0.7	-0.3	3.0	-3.2	-2.8	3.1	-0.3
	12:	0.9	7.1	-0.9	-0.3	-0.7	-2.8	1.1
	13:	-0.0	1.4	2.3	-3.8	-4.8	-2.6	3.0
	14:	-0.4	1.8	1.1	-2.2	3.0	-3.1	-2.2

a) weights from input layer to hidden layer

			PA unit #						
			0	1	2	3	4	5	6
	[E]	0:	0.0	0.3	0.1	-2.4	0.1	-4.2	-1.0
	[V]	1:	-0.2	-0.1	0.3	-2.3	-4.1	-0.1	0.1
output	[P]	2:	-0.2	-0.1	1.3	-0.7	-2.1	-0.2	0.3
unit	[X]	3:	-0.4	0.5	-1.0	2.2	4.8	-0.0	0.1
symbol	[S]	4:	0.2	0.5	-1.0	2.2	4.7	-0.2	-0.0
and #	[T]	5:	-0.3	-1.0	-0.0	-2.7	-4.2	0.1	0.5
	[B]	6:	0.4	-0.4	0.0	-0.7	-0.8	0.2	-0.9

b) weights from periodically attentive hidden to output

			Hidden unit number							
			7	8	9	10	11	12	13	14
	[E]	0:	0.7	-2.6	2.1	-2.0	9.4	-6.1	3.3	-4.6
	[V]	1:	-2.2	-2.2	4.5	1.0	-0.8	3.6	3.4	-0.9
output	[P]	2:	-5.6	**6.1**	-8.4	3.2	1.0	6.2	4.7	0.4
unit	[X]	3:	1.9	-0.1	0.0	-1.9	-4.9	-3.5	-3.0	-0.0
symbol	[S]	4:	2.0	-0.1	0.2	-2.0	-5.0	-3.5	-2.9	0.0
and #	[T]	5:	-3.4	**-8.4**	-1.4	2.2	-2.6	6.9	0.8	3.3
	[B]	6:	-0.1	-0.2	-1.4	0.5	-0.5	-1.2	0.5	-0.5

c) weights from normal hidden to output

Figure 4.8. Input and output weights for PA network 20.

	0	1	2	3	4	5	6	7	8	9
0:	-1.1	-0.3	-0.9	-1.0	-0.3	-1.0	0.4	1.1	-5.0	0.8
10:	-3.8	-1.4	3.7	-1.7	-1.0					

a) hidden thetas

	[E]	[V]	[P]	[X]	[S]	[T]	[B]
	0	1	2	3	4	5	6
0:	-2.1	-4.3	-5.7	-2.5	-2.3	-1.1	-1.7

b) output thetas

Figure 4.9. Thetas for PA network 20.

CHAPTER 5

EXPERIMENTS WITH LONG STRINGS

Two additional experiments were conducted to see how well the PA networks generalize to long strings. The strings were generated by modifying the transition probabilities of the grammar to favor long strings. At each transition where there was a choice of moving to a state closer to the accepting state or moving to a state whose distance from the accepting state is greater than or equal to the current distance, the probability of getting no closer to the accepting state was set to 0.9 (Figure 5.1). This method of generation exaggerates the bias of the test set toward strings with many T's since the T-loop appears within the V-P-X loop. Large numbers of intermediate T's could help a unit maintain activation if it became activated by a T but would hinder a unit that was deactivated by a T from maintaining activation.

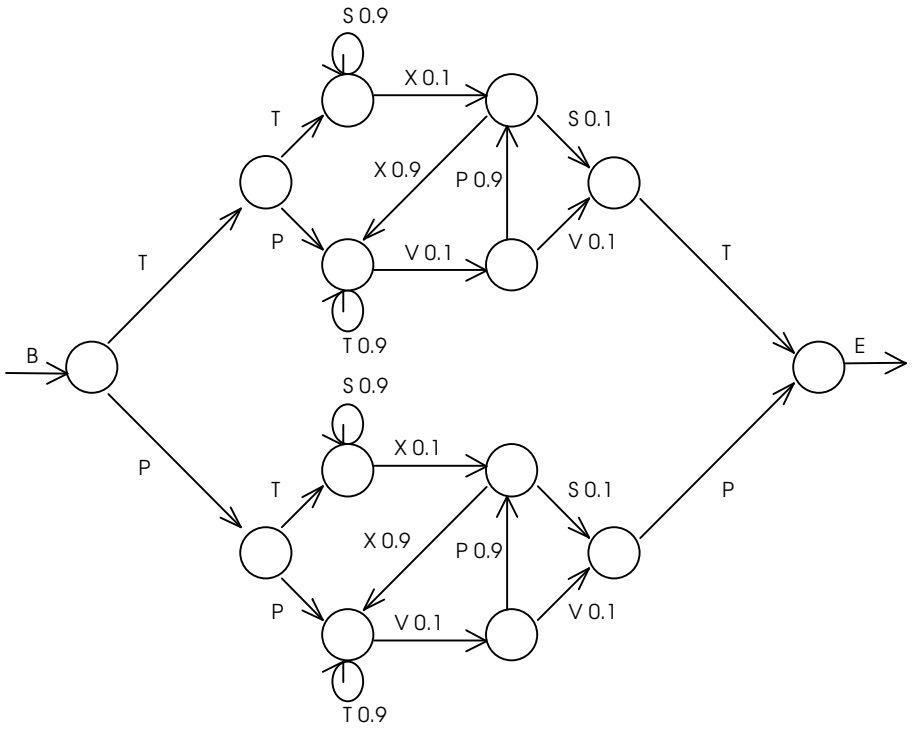


Figure 5.1. Finite-state diagram for the embedded Reber grammar with altered state transition probabilities indicated.

The first test set consisted of 1000 randomly generated strings of length at least 50 with an average length of 113. The second test set consisted of 100 randomly generated strings of length at least 300. The average length of a string in this set was 367. The best PA networks dropped to a success rate of 71% on strings with length greater than 50 and to around 50% for the strings with length greater than 300. Although these results are not as good as we would like, 50% is still twice the success rate achieved by the best SRNs on strings from the original

test set where the maximum length of a string was 34. However, a 50% success rate could be achieved by indiscriminately predicting one symbol over the other (assuming the number of test strings beginning with a T and the number beginning with a P are equal). If the network did not develop weights large enough to maintain the activation of the indicator detection unit over a string of length greater than 300, it would always predict the symbol which deactivated the unit. For example, suppose unit 1 is activated by a T and deactivated by a P at the first time step. If this unit does not have a large enough recurrent weight to maintain the activation due to the T over the length of a long string, then it will always be deactivated by the time the end of the string is reached. If the network bases its last prediction on this unit, then the network will always predict a P. A 50% success rate should, therefore, be considered a failure.

	Length >= 50		Length >= 300		Original Test	
Network	Embed	Final	Embed	Final	Embed	Final
1	3.2	47.8	0.0	41.0	68.6	53.2
2	94.0	0.0	92.0	0.0	100.0	0.0
3	98.0	3.3	94.0	0.0	98.8	1.7
4	97.0	2.4	95.0	0.0	97.6	8.9
5	66.1	44.8	54.0	41.0	93.2	53.4
6	100.0	55.1	100.0	46.0	100.0	90.5
7	100.0	71.0	100.0	54.0	100.0	90.7
8	100.0	48.9	100.0	52.0	100.0	87.8
9	3.5	13.8	0.0	11.0	67.4	21.4
10	100.0	46.5	100.0	50.0	100.0	44.7
11	99.9	46.3	100.0	48.0	100.0	59.9
12	61.2	9.4	47.0	12.0	91.0	14.5
13	88.8	49.3	84.0	43.0	99.8	92.9
14	83.0	64.3	69.0	46.0	99.1	90.8
15	27.8	33.7	5.0	31.0	90.0	44.6
16	96.8	12.7	99.0	20.0	93.6	36.1
17	100.0	0.0	100.0	0.0	100.0	0.0
18	90.3	8.6	66.0	8.0	98.0	11.3
19	29.8	11.5	0.0	10.0	99.5	24.9
20	100.0	71.5	100.0	48.0	100.0	99.5
average	77.0	32.0	70.3	28.1	94.8	46.3

Figure 5.2. Tests of PA networks on long strings.

Network - network number

Length >= 50 - tests on 1000 strings of length >= 50.
average length = 113
maximum length = 535
minimum length = 50

Length >= 300 - tests on 100 strings of length >= 300.
average length = 367
maximum length = 617
minimum length = 303

Original Test - tests on 1000 strings (from Figure 4.2)
average length = 16
maximum length = 34
minimum length = 6

Embed - % of test strings in which all embedded symbols were successfully predicted.

Final - % of test strings in which final symbol was successfully predicted.

CHAPTER 6

EXPERIMENTS WITH TIME-VARYING INDICATOR SYMBOLS

Since this architecture was developed with the embedded Reber grammar in mind, it is important to determine whether it is capable of learning long term dependencies in other grammars. It is particularly important to determine whether the fixed pattern of attention makes the network dependent upon the fixed position of the indicator symbol in the embedded Reber grammar. To that end an additional experiment was performed on a slightly modified grammar (Figure 6.1) in which the indicator symbol is preceded by any number of X's followed by a V. Thus, the indicator symbol will not appear at the same numerical position in every string.

Unfortunately, test results (Figure 6.2) show that the PA networks were unable to learn the long term dependency in this new grammar. It performed no better than the SRN which again successfully acquired the short term dependency but was unable to learn the long term dependency.

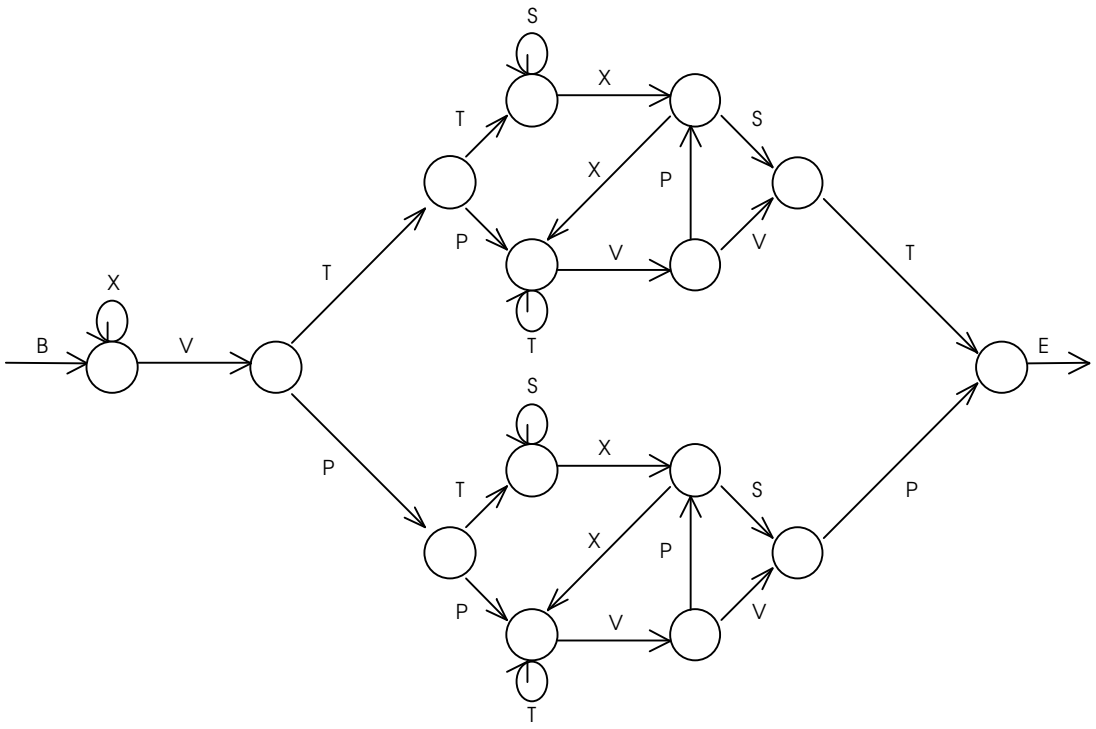


Figure 6.1. Finite State Diagram for the embedded Reber Grammar with time-varying indicator symbols.

	SRN		PA Network	
network	Embed	Final	Embed	Final
1	100.0	4.9	83.2	2.4
2	96.4	1.1	98.5	0.0
3	100.0	0.3	98.4	12.2
4	99.8	0.0	95.1	1.4
5	100.0	0.0	99.0	0.0
6	96.4	11.6	100.0	0.1
7	100.0	3.8	99.9	0.0
8	98.5	3.0	95.1	0.0
9	100.0	5.2	99.9	9.2
10	100.0	4.3	99.0	0.2
11	97.2	23.6	100.0	0.1
12	100.0	1.8	98.2	15.6
13	100.0	20.7	96.5	9.3
14	100.0	3.6	100.0	0.0
15	100.0	0.0	99.7	0.0
16	100.0	0.0	99.7	0.2
17	100.0	0.0	97.6	0.0
18	86.3	1.8	97.4	0.1
19	100.0	0.0	100.0	1.6
20	95.8	15.0	91.7	7.3
average	98.5	5.0	97.4	3.0

Figure 6.2. Test results for SRNs and PA networks on the embedded Reber grammar with time varying indicator symbols.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

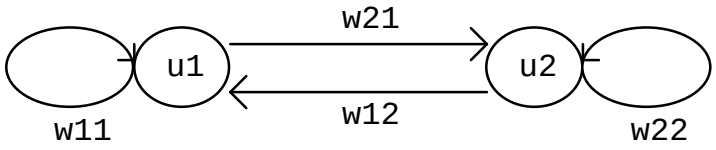
Parameters:

initial weight range	= [-1.0, 1.0]
learning rate	= 0.01
momentum	= 0.3
number of training strings	= 2.4 million
number of test strings	= 1000
average test string length	= 16.5
maximum test string length	= 37
minimum test string length	= 7

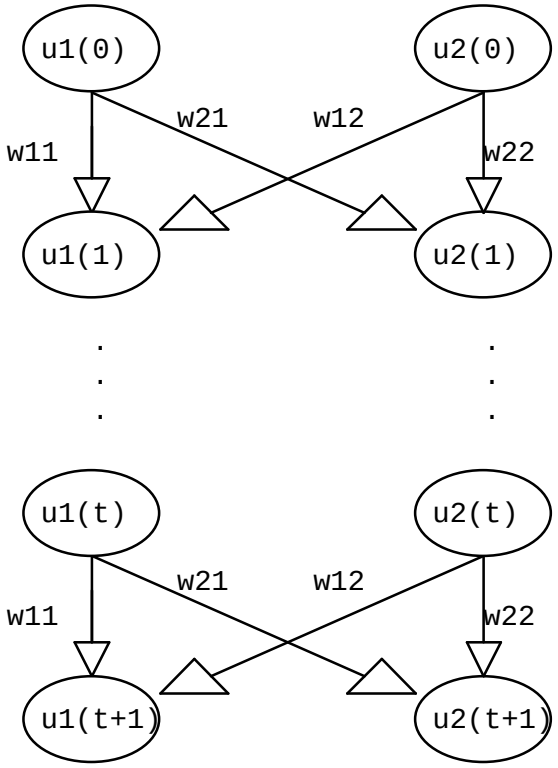
CHAPTER 7

BACKPROPAGATION THROUGH TIME

Backpropagation Through Time (BPTT, Rumelhart et al. 1986) is a learning algorithm for recurrent networks that is considered to be more powerful than the SRN learning algorithm (Williams and Peng 1990). The idea underlying BPTT is to "unfold the network in time" (Figure 6.1) until the end of the input sequence at which point the external error is calculated and error is propagated back through the network until the initial time step. The total change to a particular weight is the sum of the changes calculated for that weight at each time step. Unfortunately, this unfolding requires storage proportional to the size of the input sequence, whereas the SRN algorithm has a fixed storage requirement.



a) A fully connected 2-unit network



b) Network in (a) unfolded in time.

Figure 7.1. Backpropagation through time example. (from Rumelhart et al. 1986)

wij = weight from unit j to unit i
ui(t) = unit i at time t

Williams and Peng (1990) developed a computationally improved version of the algorithm denoted BPTT(h,h'). In BPTT(h,h') activation is fed forward for h' time steps and then error is propagated back for h time steps. The error calculations occur according to the following formula:

$$\delta_k(\tau) = \begin{cases} f'\left(net_k(\tau)\right)e_k(\tau) & \text{if} \quad \tau = t \\ f'\left(net_k(\tau)\right)\left[e_k(\tau) + \sum_l w_{lk}\delta_l(\tau+1)\right] & \text{if} \quad t-h' < \tau < t \\ f'\left(net_k(\tau)\right)\sum_l w_{lk}\delta_l(\tau+1) & \text{if} \quad t-h < \tau \le t-h' \end{cases} \qquad (1)$$

where:

t is the current time step.

w_{lk} is the weight from unit k to unit l.

$e_k(\tau)$ is the external error generated by unit k at time τ

f' is the derivative activation function

$\text{net}_k(\tau)$ is the net input to unit k at time τ.

At the current time step there is no error being propagated back from subsequent time steps so we need only include the external error in the overall error calculated at this time step. This gives us the first condition of equation (1). At each time step that was part of the previous forward pass, those in the range $(t-h', t)$, we have

both external error and error being propagated back from subsequent time steps which gives us the second condition. Finally, at each time step prior to the beginning of the most recent forward pass, we need to ignore the external errors since these errors were included in a previous backward pass.

The weight updates are computed by:

$$\Delta w_{ij} = \eta \sum_{\tau=t-h+1}^{t} \delta_i(\tau) x_j(\tau-1) \qquad (2)$$

where:

t is the current time step.

Δw_{ij} is the weight from unit j to unit i.

$x_j(\tau-1)$ is the activation of unit j at time $\tau-1$

η is the learning rate.

Consider using BPTT(3,2) on the network in Figure 7.1. Activation is fed forward normally for 2 time steps at which point the external errors (i.e., $e_k(1)$ and $e_k(2)$ for $k=1,2$) are calculated based on the actual and target activations for each unit at each of the first two time steps. Since at this point there is no error being propagated back from subsequent time steps, $\delta_k(2) = f'(net_k(2))e_k(2)$ for $k=1,2$. Error is propagated back from time step 2 as in standard backpropagation for 3 time steps (or in this case only 2 time steps since we are at time step 2). The weights are

then adjusted according to equation (2). Activation is then fed forward for another 2 time steps bringing us to time step 4. The external errors, $e_k(3)$ and $e_k(4)$ for $k = 1,2$, are then calculated based on the actual and target activations at time steps 3 and 4. Backpropagation occurs again for 3 time steps with error being injected at each time step for which target values exist. However, when time step 2 is reached during the backward pass, we do not want $e_k(2)$ to affect the weights a second time so these values are ignored.

The SRN learning algorithm can be viewed as a special case of BPTT(h,h'), namely BPTT(2,1) (Figure 7.2). Since two time steps are required for error to be propagated back from the output layer to the input and context layer, activation is fed forward for one time step and then error is propagated back for two time steps. (Time step 1 is irrelevant with regard to external error since there are no target activations for the hidden layer.)

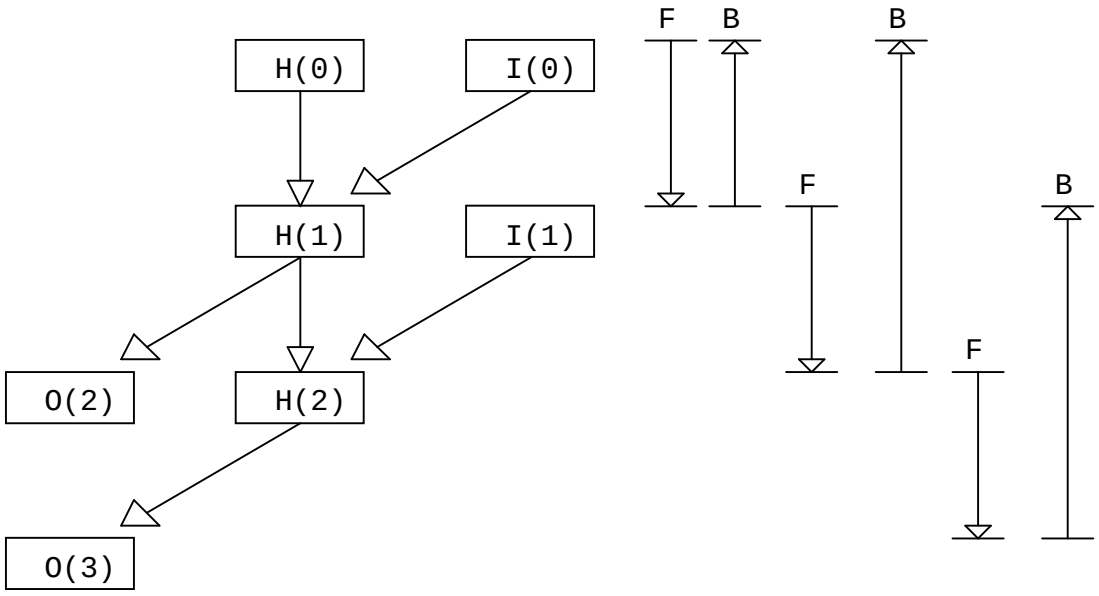


Figure 7.2. SRN unfolded in time viewed as BPTT(2,1).

F - forward pass
B - backward pass
I(t) - input layer at time t
H(t) - hidden layer at time t
O(t) - output layer at time t

Returning to the example in Figure 7.1, suppose unit 1 is a PA unit that is attentive at time step 1 and every third time step thereafter (Figure 7.3). For simplicity, assume that unit 1 never has an external target.

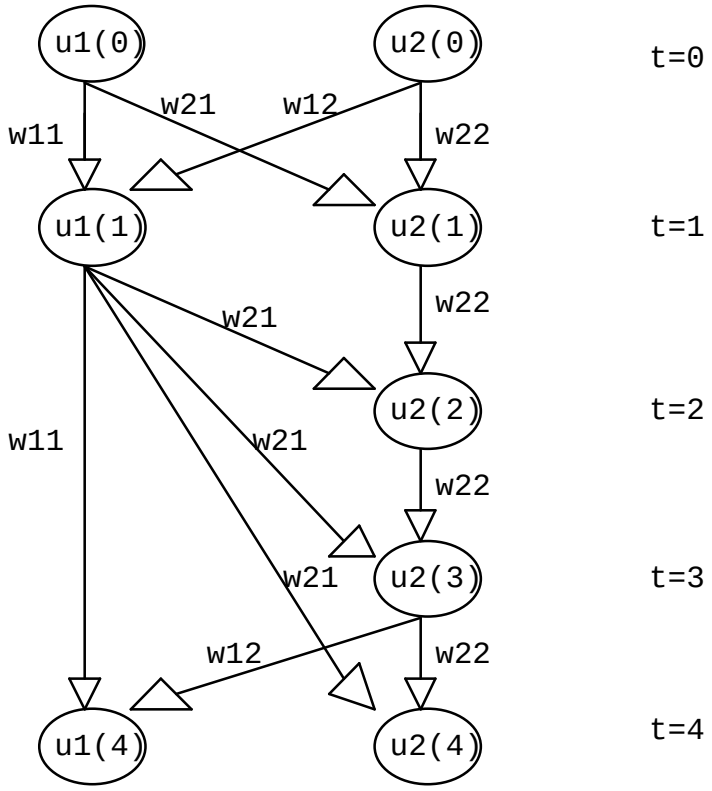


Figure 7.3. Network from Figure 7.1 where unit 1 is now a PA unit.

From Figure 7.3 we can make the following observations:

1. Error should be propagated from unit 1 back to unit 2 only when unit 1 is attentive. For example, since the

activation of unit 2 at time 3 affects the activation of unit 1 at time 4, we have:

$$\delta_2(3) = f'(net_2(3))(w_{12}\delta_1(4) + w_{22}\delta_2(4))$$

but, since the activation of unit 2 at time 2 does not affect the activation of unit 1 at time 3, we have:

$$\delta_2(2) = f'(net_2(2))\, w_{22}\delta_2(3)$$

2. Since the activation of unit 1 affects unit 2 at every time step, error should be propagated from unit 2 back to unit 1 at every time step. Since unit 1's activation does not change between times 1 and 4, however, this error should be propagated back to unit 1 at time 1. Thus we have:

$$\delta_1(1) = f'(net_1(1))(w_{11}\delta_1(4) + w_{21}\delta_2(4) + w_{21}\delta_2(3) + w_{21}\delta_2(2))$$

To simplify the delta update equation this error will be accumulated incrementally[1] i.e.,

$$\delta_1(3) = f'(net_1(1))(w_{11}\delta_1(4) + w_{21}\delta_2(4))$$
$$\delta_1(2) = \delta_1(3) + f'(net_1(1))w_{21}\delta_2(3)$$
$$\delta_1(1) = \delta_1(2) + f'(net_1(1))w_{21}\delta_2(2)$$

3. Connections into unit 1 should only be updated at time steps 1 and 4, that is, only when unit 1 is attentive.

---

[1]Note that delta's are calculated in reverse order, i.e., from time step t to time step $t-h$.

The delta and weight update formulas then become:

$$\delta_k(\tau) = \begin{cases} f'\left(net_k(\tau)\right) e_k(\tau) & \text{if } \tau = t \\ \gamma\left(a_k(\tau+1)\right)\delta_k(\tau+1) + f'\left(net_k(\tau)\right)\left[e_k(\tau) + \sum_l w_{lk}\, \delta_l(\tau+1)\, a_l(\tau+1) \right] & \text{if } t - h' < \tau < t \\ \gamma\left(a_k(\tau+1)\right)\delta_k(\tau+1) + f'\left(net_k(\tau)\right)\sum_l w_{lk}\, \delta_l(\tau+1)\, a_l(\tau+1) & \text{if } t - h < \tau \le t - h' \end{cases}$$

$$\Delta w_{ij} = \eta \sum_{\tau=t-h+1}^{t} a_i(\tau)\, \delta_i(\tau)\, x_j(\tau - 1)$$

where:

$$a_l(\tau) = \begin{cases} 1 & \text{if unit } l \text{ is attentive at time } \tau \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

CHAPTER 8

EXPERIMENTS WITH BPTT

A series of experiments were conducted using BPTT with increasing history buffer size to determine how large a history buffer is required to learn the embedded Reber grammar (Figure 8.1). A history buffer of size 4 proved more than adequate. BPTT(4,1) markedly outperformed the PA networks on test strings of moderate length. On tests with long strings BPTT(4,1) achieved a 100% success rate in several trials (Figure 8.2). The addition of PA units to networks trained with BPTT enhanced the networks performance only in the case when the history buffer was 3 (Figure 8.4). PA-BPTT(3,1) performed comparably to a PA network trained with the SRN learning algorithm (i.e., PA-BPTT(2,1)). BPTT also proved quite capable of learning the embedded Reber grammar with time varying indicators (Figure 8.5) although a history buffer of length 5 is required for the problem to be learned consistently.

	BPTT(3,1)		BPTT(4,1)		BPTT(5,1)	
Network	Embed	Final	Embed	Final	Embed	Final
1	100.0	12.3	100.0	100.0	100.0	100.0
2	100.0	7.2	100.0	43.6	100.0	99.9
3	100.0	1.6	100.0	100.0	100.0	100.0
4	100.0	15.8	100.0	100.0	100.0	100.0
5	100.0	4.8	100.0	100.0	100.0	99.9
6	100.0	7.0	100.0	97.8	100.0	100.0
7	100.0	2.9	100.0	100.0	100.0	100.0
8	100.0	12.3	100.0	99.2	100.0	100.0
9	100.0	38.7	100.0	100.0	100.0	100.0
10	100.0	7.4	100.0	50.9	100.0	99.6
11	100.0	15.7	100.0	99.6	100.0	100.0
12	100.0	5.0	99.7	99.6	100.0	100.0
13	100.0	1.0	100.0	0.0	100.0	99.8
14	100.0	6.2	100.0	100.0	100.0	100.0
15	100.0	0.8	100.0	100.0	100.0	100.0
16	100.0	15.3	99.9	2.1	100.0	100.0
17	99.8	19.2	99.0	62.9	100.0	100.0
18	99.4	12.7	99.7	99.8	100.0	100.0
19	100.0	4.9	99.2	100.0	100.0	100.0
20	100.0	25.5	100.0	100.0	100.0	99.4
average	100.0	10.8	99.9	82.8	100.0	99.9

Figure 8.1. Test results for BPTT on embedded Reber grammar.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

Parameters:

initial weight range	= [-1.0, 1.0]
learning rate	= 0.01
momemtum	= 0.3
number of training strings	= 2.4 million
number of test strings	= 1000
average test string length	= 16.4
maximum test string length	= 34
minimum test string length	= 6

	Length >= 50		Length >= 300	
Network	Embed	Final	Embed	Final
1	100.0	96.5	100.0	94.0
2	100.0	58.7	100.0	63.0
3	100.0	100.0	100.0	100.0
4	100.0	100.0	100.0	100.0
5	100.0	92.7	100.0	79.0
6	100.0	50.9	100.0	50.0
7	100.0	100.0	100.0	100.0
8	100.0	52.3	100.0	50.0
9	100.0	97.8	100.0	99.0
10	100.0	48.3	100.0	50.0
11	84.4	85.5	65.0	78.0
12	89.9	92.7	84.0	91.0
13	100.0	0.0	100.0	0.0
14	93.5	91.1	88.0	85.0
15	100.0	100.0	100.0	100.0
16	98.4	0.0	100.0	0.0
17	83.0	53.5	81.0	53.0
18	14.7	52.0	0.0	50.0
19	19.2	100.0	0.0	100.0
20	81.4	97.7	70.0	98.0
average	83.3	73.5	84.4	72.0

Figure 8.2. BPTT(4,1) tested on long strings.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

Parameters:

initial weight range = [-1.0, 1.0]
learning rate = 0.01
momemtum = 0.3
number of training strings = 2.4 million

Length >= 50
number of test strings = 1000
avg. test string length = 113
maximum test string length = 535
minimum test string length = 50

Length >= 300
number of test strings = 100
avg. test string length = 367
maximum test string length = 617
minimum test string length = 303

	Length >= 50		Length >= 300	
Network	Embed	Final	Embed	Final
1	100.0	79.9	100.0	51.0
2	100.0	52.1	100.0	50.0
3	100.0	68.4	100.0	51.0
4	100.0	64.0	100.0	50.0
5	100.0	59.8	100.0	50.0
6	99.9	100.0	98.0	100.0
7	100.0	53.6	100.0	50.0
8	99.3	100.0	98.0	100.0
9	100.0	100.0	100.0	100.0
10	100.0	87.1	100.0	84.0
11	100.0	54.0	100.0	50.0
12	100.0	52.6	100.0	50.0
13	98.3	98.2	99.0	98.0
14	100.0	100.0	100.0	100.0
15	99.9	93.7	100.0	88.0
16	100.0	100.0	100.0	100.0
17	100.0	100.0	100.0	100.0
18	100.0	100.0	100.0	100.0
19	100.0	100.0	100.0	100.0
20	100.0	93.0	100.0	91.0
average	99.9	82.8	99.8	78.2

Figure 8.3. BPTT(5,1) tested on long strings.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

Parameters:

initial weight range = [-1.0, 1.0]
learning rate = 0.01
momemtum = 0.3
number of training strings = 2.4 million

Length >= 50
number of test strings = 1000
avg. test string length = 113
maximum test string length = 535
minimum test string length = 50

Length >= 300
number of test strings = 100
avg. test string length = 367
maximum test string length = 617
minimum test string length = 303

	PA-BPTT(3,1)		PA-BPTT(4,1)		PA-BPTT(5,1)	
Network	Embed	Final	Embed	Final	Embed	Final
1	99.5	1.8	100.0	89.4	99.1	99.6
2	100.0	0.0	100.0	56.1	97.0	80.5
3	100.0	41.6	99.7	97.2	100.0	100.0
4	96.4	22.2	100.0	0.0	100.0	97.1
5	100.0	16.2	99.0	77.4	100.0	99.9
6	100.0	0.0	100.0	98.9	100.0	99.8
7	100.0	24.9	99.2	4.5	100.0	100.0
8	100.0	0.1	100.0	100.0	98.3	97.4
9	100.0	0.0	100.0	98.2	100.0	100.0
10	99.8	83.0	100.0	0.0	100.0	99.3
11	98.6	74.2	100.0	100.0	100.0	100.0
12	100.0	98.7	100.0	95.8	100.0	99.9
13	100.0	0.0	99.8	86.2	100.0	99.6
14	92.6	9.4	100.0	31.6	100.0	100.0
15	98.7	59.4	99.7	57.2	99.9	99.9
16	99.4	10.1	100.0	97.6	100.0	100.0
17	100.0	0.2	97.1	69.8	100.0	99.4
18	100.0	42.1	100.0	99.9	100.0	100.0
19	99.4	25.4	100.0	16.7	100.0	99.7
20	98.5	95.5	100.0	1.7	99.5	98.9
average	99.1	30.2	99.7	63.9	99.7	98.6

Figure 8.4. Test results for PA networks trained with BPTT on the embedded Reber grammar.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

Parameters:

initial weight range	= [-1.0, 1.0]
learning rate	= 0.01
momemtum	= 0.3
number of training strings	= 2.4 million
number of test strings	= 1000
average test string length	= 16.4
maximum test string length	= 34
minimum test string length	= 6
number of PA units	= 7
number of total hidden units	= 15

	BPTT(3,1)		BPTT(4,1)		BPTT(5,1)	
Network	Embed	Final	Embed	Final	Embed	Final
1	100.0	17.3	100.0	100.0	100.0	100.0
2	100.0	14.2	100.0	2.9	100.0	100.0
3	100.0	4.3	100.0	99.5	100.0	100.0
4	100.0	0.8	100.0	18.3	100.0	100.0
5	99.5	2.5	100.0	0.0	100.0	100.0
6	100.0	8.3	100.0	100.0	100.0	100.0
7	100.0	11.5	100.0	100.0	100.0	100.0
8	100.0	8.1	100.0	35.8	100.0	100.0
9	100.0	29.3	100.0	15.5	100.0	100.0
10	100.0	9.6	100.0	60.7	100.0	100.0
11	100.0	28.0	100.0	44.0	100.0	100.0
12	100.0	4.4	100.0	7.8	100.0	100.0
13	100.0	5.6	100.0	90.9	100.0	100.0
14	100.0	5.4	99.6	26.4	100.0	100.0
15	100.0	11.8	100.0	0.2	100.0	100.0
16	99.7	8.1	100.0	99.9	100.0	100.0
17	100.0	0.2	100.0	99.4	100.0	99.8
18	99.9	25.8	99.9	99.9	100.0	100.0
19	100.0	5.5	100.0	99.8	100.0	100.0
20	98.7	6.0	100.0	0.0	100.0	99.4
average	100.0	9.9	100.0	55.1	100.0	100.0

Figure 8.5. Test results for BPTT on embedded Reber with time-varying indicator symbols.

Embed = % of strings in which each embedded symbol was correctly predicted.

Final = % of strings in which last symbol was correctly predicted.

Parameters:

initial weight range	= [-1.0, 1.0]
learning rate	= 0.01
momemtum	= 0.3
number of training strings	= 2.4 million
number of test strings	= 1000
average test string length	= 16.5
maximum test string length	= 37
minimum test string length	= 7

CHAPTER 9

RELATED WORK

Schmidhuber (1992), using what he calls the principle of history compression, proposed constructing a cascaded series of networks that focus only on relevant inputs for training. The first network is trained on all the inputs until it fails to improve its predictions at which point training of network 2 begins. However, network 2 is trained only on inputs that network 1 failed to predict. Network 3 would then be trained on inputs that network 2 failed to predict and so on until all the inputs are correctly predicted. In this way each network (other than the first) sees an input string of greatly reduced size. Schmidhuber actually compresses this series of networks into two networks called the "chunker" and the "automatizer". See Schmidhuber (1992) for details. See also Ring (1993) which presents a non-recurrent, higher order architecture for sequence prediction that focuses on relevant inputs only.

PA networks can be viewed as implementing a kind of the history compression occurring at the unit level rather than the network level since individual units do not see every input. For tasks such as learning the embedded Reber

grammar, where the time dependencies in the data are of significant length, this amounts to ignoring irrelevant information. However, unlike Schmidhuber's networks, PA units receive input at a fixed time interval and so have no way to learn which inputs are relevant.

PA Networks also share some similarities with Time-Delay Neural Networks (TDNN, Waibel et al. 1989). A TDNN is a feed-forward network consisting of units, called time-delay units, which receive input through some fixed number of delay lines. Each of these lines has a unique weight associated with it. The delays are consecutive so a unit with n delay lines will always have access to the last n-1 inputs. Each input, therefore, will eventually be passed through each delay line. (Except, of course, that the last n-1 inputs will not be passed through delay line n.) For example, suppose we have a time-delay unit with three delay lines (Figure 9.1) and an input sequence of ABCDE. Figure 9.2 shows the input seen by the unit through each delay line at each time step.

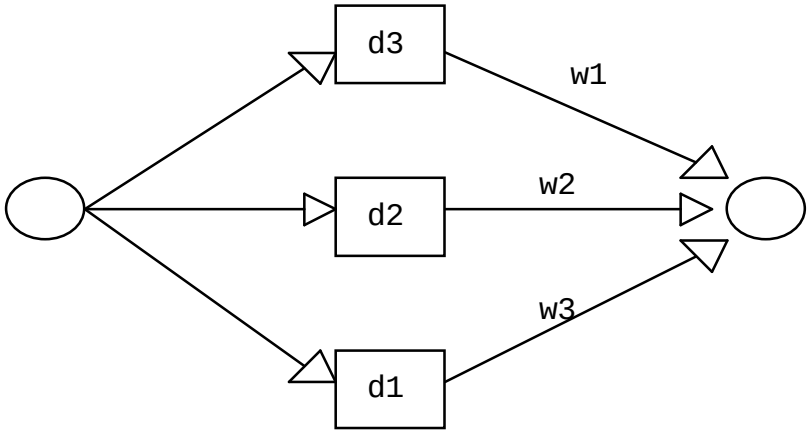


Figure 9.1. A time-delay unit with 3 delay lines.

current input	input seen through delay: d1	d2	d3
A	A	null	null
B	B	A	null
C	C	B	A
D	D	C	B
E	E	D	C

Figure 9.2. Input sequence seen through each delay line of a time-delay unit with 3 delay lines.

A time-delay unit can be viewed as having a fixed-width time window through which it views an input sequence. Waibel et al. (1989) achieve what they refer to as translational invariance by simultaneously training several copies of a TDNN. The time window of each copy begins one time step further along in the input sequence than that of the previous copy (Figure 9.3).

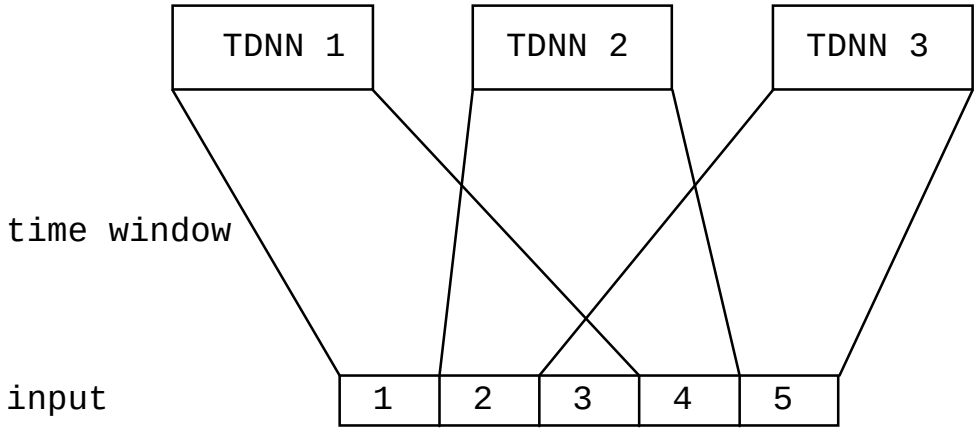


Figure 9.3. A TDNN duplicated 3 times with windows offset by 1 time step.

The adjustment made to each weight of each TDNN is the average adjustment calculated for that weight over all the TDNNs. Such a network is capable of responding to a learned pattern regardless of where it appears in the input sequence. This translational invariance is exactly what is lacking in PA networks as was shown by the experiments on strings with time varying indicator symbols. However, the fixed-width time window and the consecutive delays prevent TDNNs from being applicable to problems where there are arbitrarily long time gaps between related input symbols or where the lengths of these gaps are not known before training. Bodenhausen and Waibel (1991) have proposed an algorithm which addresses the latter problem by automatically adjusting the time delays and the window widths of a TDNN during training. See also Day and

Davenport (1993) for an algorithm to train time delays in continuous-time feed-forward networks.

CHAPTER 10

CONCLUSION

This preliminary investigation of networks with periodically attentive units has shown that such networks are far superior to Simple Recurrent Networks when learning long term dependencies if these dependencies can be detected at a fixed time. Future research with PA networks should concentrate on eliminating this fixed time limitation. PA networks are apparently much less powerful than networks trained using BPTT even when the history buffer is quite small. With only minor adjustments, however, BPTT is capable of training PA networks. More work needs to be done to determine if using PA units in conjunction with BPTT can reduce the size of the history buffer required and therefore reduce the complexity of the learning algorithm.

Additional work also needs to be done to compare PA networks with time-delay neural networks both from a theoretical and experimental point of view and to assess the applicability of the time delay adaptation algorithms of Bodenhausen and Waibel (1991) and Day and Davenport (1993) to networks with periodically attentive units.

REFERENCES

Bodenhausen, U. and Waibel, A. 1991. The Tempo 2 Algorithm: Adjusting Time-Delays By Supervised Learning. In Advances in Neural and Information Processing Systems 3, 155-181, Morgan Kaufmann.

Cleeremans, A., Servan-Schreiber, D., and McClelland, J.L. 1989. Finite-state automata and simple recurrent networks. *Neural Computation*, 1, 372-381.

Day, S.P. and Davenport, M.R. 1993. Continuous-Time Temporal Backpropagation with Adaptable Time Delays, *IEEE Transactions on Neural Networks*, Vol. 4, No. 2, 348-355.

Elman, J.L. 1990. Finding structure in time. *Cognitive Science*, 14, 179-211.

Hopcroft, J.E. and Ullman, J.D. 1979. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Reading, MA.

Horne, B.G., and Giles, C.L. 1995. An experimental comparison of recurrent neural networks. In Advances in Neural and Information Processing Systems 7 (to appear)

Jordan, M.I. 1986. Attractor dynamics and parallelism in a connectionist sequential machine. In Proceedings of the Seventh Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.

Maskara, A. and Noetzel, A. 1993. Sequence Recognition with Recurrent Neural Networks. *Connection Science*, Vol. 5, No. 2, 139-152.

Omlin, C.W., and Giles, C.L. 1994. Constructing Deterministic Finite-State Automata in Recurrent Neural Networks. Technical Report No. 94-3, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY.

Ring, M. 1993. Learning Sequential Tasks by Incrementally Adding Higher Orders. In Advances in Neural and Information Processing Systems 5, 115-122, Morgan Kaufmann.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. 1986. Learning internal representations by error propagation. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations, D.E. Rumelhart, J.L.McClelland, and the PDP Research Group, eds. MIT Press/Bradford Books, Cambridge, MA.

Rumelhart, D.E., McClelland, J.L., and the PDP Research Group, eds. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations, MIT Press/Bradford Books, Cambridge, MA.

Servan-Schreiber, D., Cleeremans, A., McClelland, J.L. 1991. Graded State Machines: The Representation of Temporal Contingencies in Simple Recurrent Networks. *Machine Learning*, 7, 161-193.

Schmidhuber, J. 1992. Learning Complex, Extended Sequences Using the Principle of History Compression. *Neural Computation*, 4, 234-242.

Waibel, A. 1989. Modular Construction of Time-Delay Neural Networks for Speech Recognition. *Neural Computation*, 1, 39-46.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K. 1989. Phoneme Recognition Using Time Delay Neural Networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 3, 328-339.

Williams, R.J., and Peng, J. 1990. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 4, 491-501.

Williams, R.J., and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1, 270-280.

Zipser, D. 1989. A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks. *Neural Computation*, 1, 552-558.