

THESIS REPORT

Ph.D.

The Adaptive Time-Delay Neural Network: Characterization and Applications to Pattern Recognition, Prediction and Signal Processing

by D-T. Lin

*Advisors: P.A. Ligomenides
J.E. Dayhoff*

Ph.D. 94-12



*Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry*

**The Adaptive Time-Delay Neural Network:
Characterization and Applications to
Pattern Recognition, Prediction and
Signal Processing**

by

Daw-Tung Lin

Dissertation submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1994

Advisory Committee:

Professor Panos A. Ligomenides, Chairman/Advisor
Dr. Judith E. Dayhoff (Co-advisor)
Professor Charles B. Silio
Professor Virgil D. Gligor
Professor Bilal M. Ayyub

Abstract

Title of Dissertation: The Adaptive Time-Delay Neural Network:
Characterization and Applications to
Pattern Recognition, Prediction and
Signal Processing

Daw-Tung Lin, Doctor of Philosophy, 1994

Dissertation directed by: Professor Panos A. Ligomenides
Department of Electrical Engineering
and
Dr. Judith E. Dayhoff
Institute for Systems Research

Dynamic analysis of temporally changing signals is a key issue in real-time signal processing and understanding. Such changing signals may arise from moving objects in visual images, spoken words, target trajectories, medical recording and other kinds of sensor data in a wide variety of applications. Neural network architectures with dynamic and temporal capabilities are promising for signal analysis. In this dissertation, we describe a solution to temporal problems with a flexible architecture: the Adaptive Time-Delay Neural Network (*ATNN*). A rich repertoire of capabilities of the *ATNN* are characterized in this thesis. A series of novel applications are implemented that for the first time show the capabilities and promise of the *ATNN* network architecture on classes of engineering problems.

Inspired by the time delays that occur in neurobiological signal transmissions, the dynamic properties of this network are formulated on-line through the

adaptation of its internal parameters based on gradient descent rules according to the evolution of observed inputs and outputs. An arbitrary number of interconnections with different time delays are placed between any two processing units, and time delays are adapted, unconstrained with respect to one another. Thus the network automatically attempts to optimize its configuration, which overcomes limitations of previous works.

We have proposed a network analysis tool, the Network Unfolding Algorithm (*NUA*), and with complexity analysis of the algorithm, demonstrated that the *ATNN* reduces the cost of network complexity at least by a factor of $O(n)$ compared to a standard *Backpropagation* network. We apply the theorems of Funahashi, Hornik *et al* and Stone-Weierstrass to state the general function approximation ability of the *ATNN*. We have evaluated the behavior of internal hidden units over a series of applications.

The *ATNN* is capable of spatiotemporal trajectory learning, trajectory generation, and pattern completion with different topologies, different training arcs, various data sampling rates, and any number of dimensions. Relationships have been discovered between the complexity of the topology and the number of hidden units, between pattern generation abilities and different training arcs and data sampling rates. It is shown that the *ATNN* is resilient to perturbations in the data, and is capable of recovering a spatiotemporal signal from noisy data. With a special recurrent configuration, the *ATNN* has the ability to learn closed trajectories. These trajectories become trained attractors of the dynamic network.

The *ATNN* is applied to chaotic time series prediction, and we demonstrate that this complicated prediction problem can be accomplished by a simple

ATNN architecture. Furthermore, the *ATNN* can detect the embedded dynamics. A real world problem of distinguishing multiple type components of exo-atmospheric targets is demonstrated by applying the *ATNN*. The *ATNN* performs highly complex mapping on a limited set of training data and achieves better generalization to overall trends of situations. Its performance outstrips that of related neural networks.

© Copyright by

Daw-Tung Lin

1994

Dedication

To my parents and my wife

Acknowledgements

I would like to express my deepest gratitude to my dissertation advisor Professor Panos Ligomenides and my co-advisor, Dr. Judith Dayhoff, without whose patient and enlightening guidance I would not have succeeded. I am also indebted to Professors David Elliott, Virgil Gligor, Charles Silio and Bilal Ayyub for their invaluable suggestions and helpful discussions.

Also, I would like to gratefully acknowledge partial support offered by the National Science Foundation under Grant No. NSF D CDR 8803012 and NSF EEC 94-02384, and the Applied Physics Laboratory of Johns Hopkins University.

Furthermore, I want to thank Professor Guangming Zhang, my parents and parents-in-law, for their encouragement and support.

Finally, I thank my wife, Shihjane, for her understanding and devotion, and patience and endless love while taking care of me and our children, Kevin and Joyce.

Table of Contents

<u>Section</u>	<u>Page</u>
List of Tables	ix
List of Figures	xii
1 Introduction	1
1.1 Preliminaries and Problem Formulation	1
1.2 Methodology	4
1.3 Contributions	7
1.4 Guide to the Dissertation	11
2 Review of Related Work	14
2.1 Fundamentals: BP and BP Through Time	15
2.2 Spatial Transformation	18
2.3 Time Delay Networks	19
2.4 Recurrent Networks	22
2.5 Temporal Integration Nets	23
2.6 Oscillatory Network and Attraction	24
2.7 Further Literature	26

3	Adaptive Time-Delay Neural Network	28
3.1	Introduction	28
3.2	Network Architecture and Theory	30
3.3	Learning Paradigm	36
3.4	Learning Algorithm Derivation	38
3.5	ATNN: Universal Spatiotemporal Function Approximator	42
3.6	NUA and Complexity Analysis	45
3.7	Hidden Units and Single Layer Time-Delay Adaptation	50
3.8	Relation to Other Networks	54
3.9	Conclusion	54
4	Spatio-Temporal Pattern Processing	56
4.1	Definition and Scope	56
4.2	Trajectory Production	58
4.2.1	Trajectory Topology Learning	59
4.2.2	Neuron Dynamics	64
4.3	Pattern Completion and Recognition	65
4.4	Spatio-Temporal Feature Recognition	68
4.4.1	It is not memorizing each point	68
4.4.2	Training Position Invariance	76
4.4.3	Pattern Completion	77
4.5	Perturbation Resilience and Noise Removal	81
4.5.1	Noise Tolerance	83
4.5.2	Noise Removal of Spatiotemporal Signal	83
4.5.3	Noise Removal as Portion of the Pattern is Trained	90
4.5.4	Remarks	95

4.6	Attractor Learning	96
4.7	Sampling Rate Effects	99
4.7.1	Sampling Rate vs. Training Speed and Performance	99
4.7.2	Reproduction From Different Starting Positions	100
4.7.3	Noise Tolerance	104
4.7.4	Noise Removal as Patterns with Different Sampling Rate are Trained	107
4.7.5	Remarks	113
4.8	Discussions	113
5	Time Series Analysis: Prediction and Modeling	116
5.1	Introduction	116
5.2	Problem Description	118
5.3	Performance of TDNN on Prediction Performance	119
5.4	ATNN for Prediction	123
5.4.1	Training Set Size and Performance	123
5.4.2	Resolving Embedding Dynamics	126
5.4.3	Prediction Over Time	130
5.4.4	Noise Tolerance	131
5.5	Time Series Modeling	133
5.5.1	Chaotic Time Series Modeling	133
5.5.2	Harmonic Wave Form Modeling	136
5.6	Conclusion	139
6	A Real World Application: Target Discrimination	141
6.1	Automatic Target Recognition	141

6.2	TDNN Technique	143
6.2.1	Simulation Setups	143
6.2.2	Simulation Scheduling and Analysis	144
6.2.3	Improvement on Higher Sampled Data	147
6.2.4	Number of Time Steps vs. Sampling Rate	148
6.3	ATNN: a Superior Approach	151
6.3.1	Earlier Discrimination on More Realistic Sensor Data . . .	151
6.3.2	Lemma Support	154
6.3.3	Environment Variations and Robustness	154
7	Conclusions	163
7.1	Summary	163
7.2	Final Remarks and Future Research	167
	Appendix	172
A	Trajectory Recognition with a Time-Delay Neural Network	172
A.1	Network Architecture	173
A.2	Trajectory Recognition Results	176
A.2.1	Shift-Invariance	177
A.2.2	Shift-Invariance by Training	178
A.2.3	Shift-Invariance by Recall with Bias	179
A.2.4	Graceful Degradation	180
A.2.5	Noise Tolerance	181
A.3	Conclusion	182
B	Time-Delay Matrices: Simulation Results	184

B.1	Resulting Time-Delay Matrices of Various Training Scenarios . .	184
B.2	Resulting Time-Delay Matrices of Sampling Rate Effects	186
C	Target Discrimination Simulation Results	189
C.1	Simulation Results on 0.5 second data.	189
C.2	More Results on 0.1 second data.	196
C.3	Simulation Results on partial 0.1 second data.	198
C.4	Resulting ATNN Time-Delay Matrix	201
	Bibliography	217

List of Tables

<u>Table</u>	<u>Page</u>
3.1 Comparison of the original and unfolded network configuration. .	50
4.1 The resulting time-delay variables after training is complete, where $\max(T_1)$ and $\max(T_2)$ denote the maxima element of time-delay matrix on layer one and two respectively, and T_{atnn} is the total delay of trained network where $T_{atnn} = \max(T_1) + \max(T_2)$	72
4.2 <i>RMSE</i> of reproduced figure under different situation and starting position.	81
4.3 <i>RMSE</i> of successively generated circles from varied noisy initial data, $\sigma_{noise}^2 = 0.0001$ to 0.5, when full circle is trained.	88
4.4 <i>RMSE</i> of successively generated circles from variance noisy initial data, $\sigma_{noise}^2 = 0.0001$ to 0.5, when $\frac{3}{4}$ circle is trained.	91
4.5 <i>RMSE</i> of successively generated circles from variance noisy initial data, $\sigma_{noise}^2 = 0.0001$ to 0.5, when $\frac{1}{2}$ circle is trained.	92
4.6 <i>RMSE</i> comparison of various sampling rate during training phase	101
4.7 <i>RMSE</i> comparison of various sampling rate during training phase	101

4.8	The resulting time-delay variables after training is complete, where T_1 and T_2 denote the time-delay matrix on layer one and two respectively, and T_{atnn} is the total delay of trained network where $T_{atnn} = \max(T_1) + \max(T_2)$	102
4.9	<i>RMSE</i> of reproduced figure under different starting positions when various training sampling rates were employed.	104
4.10	<i>RMSE</i> of successively generated circles from noisy initial data. The amount of noise was varied: $\sigma_{noise}^2 = 0.0001$ to 0.5, and 32 points per circle is trained.	108
4.11	<i>RMSE</i> of successively generated circles from noisy initial data. The amount of noise was varied: $\sigma_{noise}^2 = 0.0001$ to 0.5, and 21 points per circle is trained.	108
4.12	<i>RMSE</i> of successively generated circles from noisy initial data. The amount of noise was varied: $\sigma_{noise}^2 = 0.0001$ to 0.5, and 16 points per circle is trained.	109
5.1	Experiments of different configurations as the cutoff C_{RMSE} is set to 0.02, $T = 1$	124
5.2	Experiments with different network configurations trained to predict the Mackey-Glass equation. Each table entry is the average of three runs.	136
5.3	The error measure <i>NRMSE</i> of this system output under different approximation to a square wave	137
6.1	Successful identification rate of different test sets by using <i>TDNN</i> trained on 0.5 second increment data.	145

6.2	Over all performance of <i>TDNN</i> trained on 0.5 second increment data.	145
6.3	Comparison of discrimination performance with different data sets, various training methods and different network topologies. .	146
6.4	The comparison of network performance when various situations of time-delay adaptation is applied, T_1 and T_2 denote the time-delays in first and second layer respectively.	155
6.5	Performance of <i>ATNN</i> trained on pure data and tested with varying amounts of noise added.	157
6.5	(cont.) Performance of <i>ATNN</i> trained on pure data and tested with varying amounts of noise added.	158
6.6	Performance of <i>ATNN</i> trained on pure and noisy data and tested on varying amounts of noise added.	160
6.6	(cont.) Performance of <i>ATNN</i> trained on pure and noisy data and tested on varying amounts of noise added.	161

List of Figures

<u>Number</u>		<u>Page</u>
1.1	General neurons and connection structure with time-delay τ_1, τ_2, τ_3 and τ_4 embedded.	3
1.2	Delay Block: Basic time-delay connections between two processing units (node i of layer $h - 1$ and node j of layer h) in $ATNN$, and n is the number of delays applied ($n = K_{ji,h-1}$).	5
1.3	Schematic block diagram of system adaptation.	7
3.1	Delay Block: Basic time-delay connections between two processing units (node i of layer $h - 1$ and node j of layer h) in $ATNN$, and $n_{ji,h-1}$ is the number of delays applied.	31
3.2	Layered $ATNN$: symbolic diagram	32
3.3	Three layered $ATNN$	32
3.4	Explicit demonstration of unfold operation.	49
3.5	Alternative NUA by unfolding second layer time-delay first.	51
4.1	Circular trajectory on x, y, and time domain t	57
4.2	Figure eight trajectory on x, y, and time domain t	57
4.3	The progress of learning circular trajectory.	60

4.4	<i>RMSE</i> along the training for circle pattern.	60
4.5	The progress of learning figure eight trajectory. Network configuration: $2 \xrightarrow{4} 5 \xrightarrow{6} 2$	61
4.6	The progress of learning figure eight trajectory. Network configuration: $2 \xrightarrow{4} 8 \xrightarrow{6} 2$	62
4.7	The progress of learning figure eight trajectory. Network configuration: $2 \xrightarrow{4} 12 \xrightarrow{6} 2$	62
4.8	Learning curves for eight figure pattern utilizing various number of hidden units	63
4.9	Hidden neurons activities during training time steps 1 to 1000. The dotted line under O1 and O2 is the target trajectory	65
4.10	Hidden neurons activity during training time steps 3000 to 5000. The dotted line under O1 and O2 is the target trajectory	65
4.11	Hidden neurons activities after trained with circular trajectory: each unit responds to different input orientations.	66
4.12	Hidden neurons activities after trained with figure eight trajectory: each unit responds to different input orientations.	66
4.13	The results of circle reproduced by <i>ATNN</i> start from different initial position (a) <i>RMSE</i> = 0.0113 (b) <i>RMSE</i> = 0.0162 (c) <i>RMSE</i> = 0.0114	67
4.14	The results of figure eight reproduced by <i>ATNN</i> start from different initial position (a) <i>RMSE</i> = 0.0507 (b) <i>RMSE</i> = 0.0508 (c) <i>RMSE</i> = 0.0474.	67
4.15	<i>RMSE</i> of various training trajectories during training phase. . . .	70

4.16	<i>RMSE</i> of learning curve and testing curve when trained on 3/4 circle and tested on half circle.	70
4.17	<i>RMSE</i> of learning curve and testing curve when trained on 1/2 circle and tested on the other half circle.	71
4.18	<i>RMSE</i> of learning curve and testing curve when trained on 1/4 circle and tested on the half circle.	71
4.19	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 3/4 circle.	73
4.20	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 3/4 circle.	73
4.21	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 3/4 circle.	73
4.22	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 3/4 circle.	73
4.23	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 1/2 circle.	74
4.24	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 1/2 circle.	74
4.25	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 1/2 circle.	74
4.26	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 1/2 circle.	74
4.27	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 1/4 circle.	75

4.28	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 1/4 circle.	75
4.29	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 1/4 circle.	75
4.30	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 1/4 circle.	75
4.31	Learning <i>RMSE</i> curves of various training trajectories arcs with the same length ($\frac{3}{4}$ circle) but with different location.	76
4.32	<i>RMSE</i> of learning curve and testing curve when trained on first 3/4 circle and tested on the other 3/4 circle.	77
4.33	<i>RMSE</i> of learning curve and testing curve when trained on second 3/4 circle and tested on the other 3/4 circle.	77
4.34	(a) <i>ATNN</i> learning phase and (b) testing on different 3/4 part of the circle, when trained 5 times on 3/4 circle.	78
4.35	(a) <i>ATNN</i> learning phase and (b) testing on different 3/4 part of the circle, when trained 100 times on 3/4 circle.	78
4.36	(a) <i>ATNN</i> learning phase and (b) testing on different 3/4 part of the circle, when trained 200 times on 3/4 circle.	78
4.37	(a) <i>ATNN</i> learning phase and (b) testing on different 3/4 part of the circle, when trained 600 times on 3/4 circle.	78
4.38	Learning <i>RMSE</i> curves of various training trajectories arcs with the same length (half circle) but with different location.	79
4.39	<i>RMSE</i> of learning curve and testing curve when trained on first 1/2 circle and tested on the other half circle.	79

4.40	<i>RMSE</i> of learning curve and testing curve when trained on second 1/2 circle and tested on the other half circle.	79
4.41	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 1/2 circle.	80
4.42	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 1/2 circle.	80
4.43	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 1/2 circle.	80
4.44	(a) <i>ATNN</i> learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 1/2 circle.	80
4.45	The results of circle reproduced by <i>ATNN</i> (trained from 3/4 circle) start from different initial position (a) <i>RMSE</i> = 0.0165 (b) <i>RMSE</i> = 0.0180 (c) <i>RMSE</i> = 0.0170	82
4.46	The results of circle reproduced by <i>ATNN</i> (trained from 1/2 circle) start from different initial position (a) <i>RMSE</i> = 0.0336 (b) <i>RMSE</i> = 0.0460 (c) <i>RMSE</i> = 0.0439	82
4.47	Noise tolerance of various trained network. The full circle and 3/4 circle traces are nearly identical.	84
4.48	<i>ATNN</i> removes the noise (variance $\sigma_{noise}^2 = 0.001$) from a time-series sequence and complete the subsequent path.	85
4.49	<i>ATNN</i> removes the noise (variance $\sigma_{noise}^2 = 0.01$) from a time-series sequence and complete the subsequent path.	86
4.50	<i>ATNN</i> removes the noise (variance $\sigma_{noise}^2 = 0.1$) from a time-series sequence and complete the subsequent path.	86

4.51	<i>ATNN</i> removes the noise (variance $\sigma_{noise}^2 = 0.3$) from a time-series sequence and complete the subsequent path.	87
4.52	<i>ATNN</i> removes the noise (variance $\sigma_{noise}^2 = 0.5$) from a time-series sequence and complete the subsequent path.	87
4.53	<i>ATNN</i> ($2 \xrightarrow{4} 5 \xrightarrow{6} 2$ trained after $T = 40000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).	89
4.54	<i>ATNN</i> ($2 \xrightarrow{4} 5 \xrightarrow{6} 2$ trained after $T = 80000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).	89
4.55	<i>ATNN</i> ($2 \xrightarrow{4} 8 \xrightarrow{6} 2$ trained after $T = 40000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).	89
4.56	<i>ATNN</i> ($2 \xrightarrow{4} 8 \xrightarrow{6} 2$ trained after $T = 75000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).	90
4.57	<i>ATNN</i> removes the noise (variance $\sigma^2 = 0.01$) from a time-series sequence and complete the subsequent path when $\frac{3}{4}$ circle is trained.	92
4.58	<i>ATNN</i> removes the noise (variance $\sigma^2 = 0.3$) from a time-series sequence and complete the subsequent path when $\frac{3}{4}$ circle is trained.	93
4.59	<i>ATNN</i> removes the noise (variance $\sigma^2 = 0.01$) from a time-series sequence and complete the subsequent path when $\frac{1}{2}$ circle is trained.	93
4.60	<i>ATNN</i> removes the noise (variance $\sigma^2 = 0.3$) from a time-series sequence and complete the subsequent path when $\frac{1}{2}$ circle is trained.	94
4.61	<i>RMSE</i> of various training trajectories during regeneration of successive circles.	94
4.62	The network finally follows a track of some attractor for widely varying radii of the initial circle.	98

4.63	The network can not approach basin of attraction of figure eight (with net: $2 \xrightarrow{4} 5 \xrightarrow{6} 2$).	98
4.64	The network approaches basin of attraction of figure eight (with net: $2 \xrightarrow{4} 8 \xrightarrow{6} 2$)	98
4.65	<i>RMSE</i> of various sampling rate during training phase.	100
4.66	Performance progress during training when 64 point per circle is employed	101
4.67	Performance progress during training when 32 point per circle is employed	102
4.68	Performance progress during training when 21 point per circle is employed	103
4.69	Performance progress during training when 16 point per circle is employed	103
4.70	The results of circle reproduced by <i>ATNN</i> start from different initial position (a) <i>RMSE</i> = 0.0827 (b) <i>RMSE</i> = 0.1700 (c) <i>RMSE</i> = 0.0190 when trained on 32 points per circle pattern. . .	105
4.71	The results of circle reproduced by <i>ATNN</i> start from different initial position (a) <i>RMSE</i> = 0.2215 (b) <i>RMSE</i> = 0.1036 (c) <i>RMSE</i> = 0.0729 when trained on 16 points per circle pattern. . .	105
4.72	The results of circle reproduced by <i>ATNN</i> start from different initial position (a) <i>RMSE</i> = 0.1741 (b) <i>RMSE</i> = 0.0776 (c) <i>RMSE</i> = 0.0313 when trained on 16 points per circle pattern. . .	106
4.73	Noise tolerance of network trained from various sampling rate pattern.	106

4.74	Comparison of <i>RMSE</i> of successive circle reproduced when various sampling rate signals are trained.	109
4.75	<i>ATNN</i> can barely remove small amount of noise (variance $\sigma^2 = 0.01$) at the 2nd sequence but not for the subsequent path when 32 points per circle is trained.	110
4.76	<i>ATNN</i> can no longer clean up the noise when larger amount of initial noise is added (variance $\sigma^2 = 0.3$) as 32 points per circle is trained.	110
4.77	<i>ATNN</i> can barely remove small amount of noise (variance $\sigma^2 = 0.01$) at the 2nd sequence but not for the subsequent path when 21 points per circle is trained.	111
4.78	<i>ATNN</i> can no longer clean up the noise when larger amount of initial noise is added (variance $\sigma^2 = 0.3$) as 21 points per circle is trained.	111
4.79	<i>ATNN</i> can barely remove small amount of noise (variance $\sigma^2 = 0.01$) at the 2nd sequence but not for the subsequent path when 16 points per circle is trained.	112
4.80	<i>ATNN</i> can no longer clean up the noise when larger amount of initial noise is added (variance $\sigma^2 = 0.3$) as 16 points per circle is trained.	112
5.1	System variation of Mackey-Glass equation from $t = 1$ to 1000: (a) function output (b) strange attractors on state space.	120
5.2	<i>TDNN</i> configuration for prediction	121
5.3	Simulation results, $\tau_1 = 1, \tau_2 = 1, m = 4, n = 8$	122
5.4	Simulation results, $\tau_1 = 4, \tau_2 = 1, m = 4, n = 8$	122

5.5	Simulation results of <i>ATNN</i> with the Mackey-Glass equation prediction as $T = 1$ and test on $t = 3000$ to 4000 sample data period.	125
5.6	The resulting weight matrix of first layer trained with different criteria C_{RMSE} .	129
5.7	The resulting (a) time-delay matrix and (b) weight matrix of second layer trained with criteria $T = 1, C_{RMSE} = 0.02$.	129
5.8	The resulting (a) time-delay matrix and (b) weight matrix of second layer trained with criteria $T = 1, C_{RMSE} = 0.05$.	129
5.9	The resulting (a) time-delay matrix and (b) weight matrix of second layer trained with criteria $T = 2, C_{RMSE} = 0.03$.	130
5.10	Performance comparison of neural net and conventional methods: relative quantity diagram. Adopted from Lapedes and Farber [83].	131
5.11	Prediction performance of <i>ATNN</i> on different network configuration with 400 training data.	132
5.12	System performance comparison of different variance of noise added to input patterns	132
5.13	Simulation results of <i>ATNN</i> with Mackey-Glass equation modeling $t=0$ to 1000 , $NRMSE = 0.045$	135
5.14	Simulation results of <i>ATNN</i> with Mackey-Glass equation modeling, $t=9000$ to 10000 , $NRMSE = 0.063$	135
5.15	Simulation of identifying the fundamental wave: first harmonic.	138
5.16	Simulation of identifying the wave with a sum of 1st to 5th add harmonics	138
5.17	Simulation of identifying the wave with a sum of 1st to 9th add harmonics	139

6.1	Target components discrimination.	142
6.2	Recognizing target with aspect angle 30° , sampling time interval 0.1 sec.	148
6.3	Recognizing target with aspect angle 30° , sampling time interval 0.5 sec.	148
6.4	Recognizing target with aspect angle 60° , sampling time interval 0.1 sec.	149
6.5	Recognizing target with aspect angle 60° , sampling time interval 0.5 sec.	149
6.6	Recognizing target with aspect angle 90° , components 1 to 4 tum- ble at 90° , 60° , 60° , and 30° respectively, sampling time interval 0.1 sec.	149
6.7	Recognizing target with aspect angle 90° , components 1 to 4 tum- ble at 90° , 60° , 60° , and 30° respectively, sampling time interval 0.5 sec.	149
6.8	Recognizing target with aspect angle 90° , components 1 to 4 tum- ble at 30° , 60° , 90° , and 30° respectively, sampling time interval 0.1 sec.	150
6.9	Recognizing target with aspect angle 90° , components 1 to 4 tum- ble at 30° , 60° , 90° , and 30° respectively, sampling time interval 0.5 sec.	150
6.10	<i>ATNN</i> performance on bumpy data with aspect angle 90° , sam- pling time interval 0.1 sec.	153
6.11	<i>TDNN</i> performance on bumpy data with aspect angle 90° , sam- pling time interval 0.1 sec.	153

6.12	<i>ATNN</i> performance on bumpy data with aspect angle 60° , sampling time interval 0.1 sec.	153
6.13	<i>TDNN</i> performance on bumpy data with aspect angle 60° , sampling time interval 0.1 sec.	153
6.14	<i>ATNN</i> performance on bumpy data with aspect angle 30° , sampling time interval 0.1 sec.	154
6.15	<i>TDNN</i> performance on bumpy data with aspect angle 30° , sampling time interval 0.1 sec.	154
6.16	Performance of the <i>ATNN</i> as a function of the amount of noise in the data. For noisy environments, performance is superior when training is done on noisy data.	162
7.1	Physiological data from a patient with sleep apnea. Data includes: heart rate, respiration force, blood oxygen concentration. Time series sampled every 0.5 second. Data source: Time Series Prediction Competition, Santa Fe Institute, 1992 [158].	169
7.2	Symbolic block diagram of neural net application to prediction of physiology systems.	170
A.1	<i>TDNN</i> with two layers — input and hidden layer.	174
A.2	Training set and target values. Three output units represented the three trajectory types.	176
A.3	Test for distorted trajectories.	176
A.4	Lesion	181
A.5	(a) Noise tolerance of different test set (b) Average noise tolerance.	182
C.1	Recognizing target in trajectory 1 with aspect angle 30°	189

C.2	Recognizing target in trajectory 1 with aspect angle 45°	189
C.3	Recognizing target in trajectory 1 with aspect angle 60°	190
C.4	Recognizing target in trajectory 1 with aspect angle 75°	190
C.5	Recognizing target in trajectory 1 with aspect angle 90°	190
C.6	Recognizing target in trajectory 2 with aspect angle 30°	190
C.7	Recognizing target in trajectory 2 with aspect angle 60°	191
C.8	Recognizing target in trajectory 2 with aspect angle 90°	191
C.9	Recognizing target with aspect angle 90° , front $\frac{1}{3}$ component 2 replaces whole component 2.	191
C.10	Recognizing target with aspect angle 90° with back $\frac{1}{3}$ component 2 replaces whole component 2.	191
C.11	Recognizing target with aspect angle 90° with front $\frac{1}{2}$ component 3 replaces whole component 3.	192
C.12	Recognizing target with aspect angle 90° with front $\frac{1}{3}$ component 4 replaces whole component 4.	192
C.13	Recognizing target with components tumbling angles: 90° , 60° , 90° , and 90° respectively.	192
C.14	Recognizing target with components tumbling angles: 90° , 60° , 60° , and 90° respectively.	192
C.15	Recognizing target with components tumbling angles: 90° , 60° , 60° , and 30° respectively.	193
C.16	Recognizing target with components tumbling angles: 60° , 90° , 60° , and 60° respectively.	193
C.17	Recognizing target with components tumbling angles: 60° , 90° , 30° , and 60° respectively.	193

C.18 Recognizing target with components tumbling angles: 60°, 90°, 90°, and 60° respectively.	193
C.19 Recognizing target with components tumbling angles: 60°, 30°, 30°, and 60° respectively.	194
C.20 Recognizing target with components tumbling angles: 60°, 30°, 60°, and 90° respectively.	194
C.21 Recognizing target with components tumbling angles: 60°, 30°, 90°, and 90° respectively.	194
C.22 Recognizing target with components tumbling angles: 30°, 60°, 30°, and 30° respectively.	194
C.23 Recognizing target with components tumbling angles: 30°, 60°, 90°, and 30° respectively.	195
C.24 Recognizing target with components tumbling angles: 30°, 60°, 90°, and 60° respectively.	195
C.25 Recognizing target with aspect angle 90°.	196
C.26 Recognizing target with aspect angle 90°, front 1/3 component 2 replaces the whole.	196
C.27 Recognizing target with aspect angle 90°, components tumbling angle: 60°, 90°, 90°, and 30° respectively.	196
C.28 Recognizing target with aspect angle 90°, front 1/3 component 2 middle 1/3 component 3, front 1/2 component 4.	196
C.29 Recognizing target with aspect angle 90°, front 1/3 component 2, middle 1/3 component 2, and component 4	197
C.30 Recognizing target with aspect angle 30°, front 1/3 component 2 replaces whole component 2.	197

C.31 Recognizing target with aspect angle 30°, front 1/3 component 2, middle 1/3 component 2, component 3	197
C.32 Recognizing target with aspect angle 30°, front 1/3 component 2, back 2/3 component 2, and component 4.	197
C.33 Aspect angle 30°.	198
C.34 Aspect angle 60°.	198
C.35 Aspect angle 90°.	198
C.36 Aspect angle 30°, front 1/3 component 2 replaces whole compo- nent 2.	198
C.37 Aspect angle 30°, back 1/3 component 2 replaces whole compo- nent 2, front 1/3 component 2 replaces whole component 3.	199
C.38 Aspect angle 30°, front 1/3 component 2 replaces whole compo- nent 2, middle 1/3 component 2 replaces whole component 3.	199
C.39 Aspect angle 90°, with components tumbling angle: 30°, 60°, 90°, and 30° respectively.	199
C.40 Aspect angle 90°, with components tumbling angle: 60°, 90°, 90°, and 30° respectively.	199
C.41 Aspect angle 90°, time 323-325 second.	200
C.42 Aspect angle 90°, with components tumbling angle: 90°, 60°, 60°, and 30° respectively.	200
C.43 Aspect angle 90°, front 1/3 component 2 replaces whole compo- nent 2, middle 1/3 component 2 replaces whole component 3.	200
C.44 Aspect angle 90°, front 1/3 component 2 replaces whole compo- nent 2, middle 1/3 component 2 replaces whole component 3, front 1/2 component 3 replaces component 4.	200

Chapter 1

Introduction

1.1 Preliminaries and Problem Formulation

Dynamic analysis of temporally changing signals is a key issue in real-time signal processing and understanding. Such changing signals may arise from moving objects in visual images, spoken words, target trajectories, medical examinations and other kinds of sensor data in a wide variety of applications. Neural network paradigms involving architectures with dynamic and temporal capabilities are promising for signal analysis.

In the past decade, there has been a tremendous interest in understanding the dynamic behavior of artificial neural networks (*ANN*) [22, 67]. Furthermore, researchers have developed *ANN* methodologies to generate and process temporal sequences [56, 106, 38] and to emulate the time dependence of dynamic systems [113, 124]. Considerable efforts have been devoted to finding the embedded dynamics of time sequences [105] or capturing the characteristics of chaotic behaviors by using techniques based on neural network

paradigms [123, 135, 147, 164]. Researchers are aiming not only at a better understanding of physiology, but also to develop novel systems of dynamic information processing.

The time-delay neural network (*TDNN*) proposed by Waibel et al [151] employs time-delays on connections and has been successfully applied to phoneme recognition [152, 153, 146]. The time-delay neural network also classifies spatiotemporal patterns and provides robustness to handling noise and allowing graceful degradation [86]. In the *TDNN* architecture, each neuron takes into account not only the current information from its input neurons of the previous layer, but also a certain amount of past information from those neurons due to delays on interconnections. Typically the time delays are evenly spaced over a time interval called the frame window, although arbitrary time delays may be used. Training is done with spatiotemporal patterns, and the classification of those patterns is reported at each time step by the output layer. After training, weights are strengthened along interconnections whose time delays are important to recognition.

A limitation of the *TDNN* as originally posed [151] was its inability to learn or adapt the values of the time delays. Time delays are fixed initially and remain the same throughout training. To address this limitation, we have used an adaptive time delay neural network (*ATNN*), which adapts time delays as well as weights during training. The result is a dynamic learning technique for spatiotemporal classification [84].

Biological studies have shown that time delays do occur along axons due to different conduction times and differing lengths of axonal fibers, and that, in addition, temporal properties such as temporal decay and integration occur at

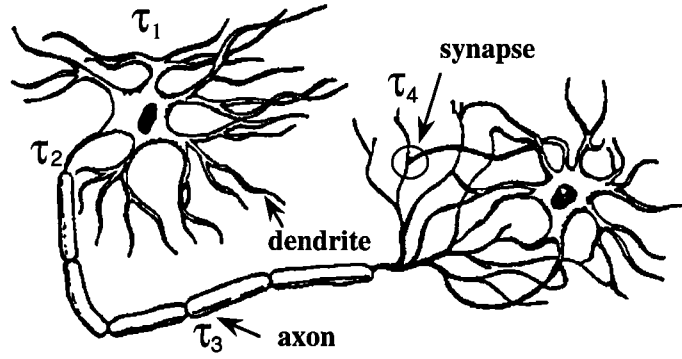


Figure 1.1: General neurons and connection structure with time-delay τ_1, τ_2, τ_3 and τ_4 embedded.

synapses (illustrated in Figure 1.1). Thus biological systems inspire the *TDNN* and *ATNN* architectures. Furthermore, for biological systems, research on these architectures can be used to inspire hypotheses that ultimately can be tested experimentally on living neural systems.

The main goals of our research are to develop an adaptive and optimal configuration network architecture utilizing time delays, to provide more flexibility for the neural net model to approach efficient performance, to model unknown linear or nonlinear functions and to capture the embedded dynamics of observed data sequences.

The proposed network model is promising for the following types of problems:

1. Nonlinear dynamic system identification: The objective of system modeling problem is to construct a suitable identification function $\hat{\mathcal{P}}$, for a given plant $\mathcal{P}$ and its input-output pair $\{ \langle x_1, x_2, \dots, x_i \rangle, \langle y_1, y_2, \dots, y_m \rangle \}$ (where i and m denote the dimension of input and output vectors respectively), such that $\|\hat{y} - y\| = \|\hat{\mathcal{P}}(x) - \mathcal{P}(x)\| \leq \epsilon$ for some desired $\epsilon \geq 0$. To

construct a dynamic model directly from observed data involves two steps: [72]

- (a) state space reconstruction: use the past behavior of the time series to reconstruct the current state of the system.
 - (b) nonlinear function approximation: find the nonlinear function f which maps the current state $s(t)$ to a future state $s(t + T)$. The function f can be an equation or a neural net representation.
2. Learning to generate temporal patterns or spatiotemporal patterns and trajectories.
 3. Performing classification on temporal patterns or spatiotemporal patterns and discrimination between different targets that have similar time-varying signals.
 4. Generating spatiotemporal signals, and closed trajectories in such a way that nearby states are attracted to the trained signal that is generated.
 5. Prediction of chaotic time series or other series that are difficult to predict by conventional prediction techniques.

1.2 Methodology

Studies on the temporal behavior of neural network models have examined signal transmission along neurons [120, 140] and identification of temporal events [46, 141]. Related techniques such as backpropagation through time [161] and the *TDNN* network [151] have been applied to temporal pattern recognition but not

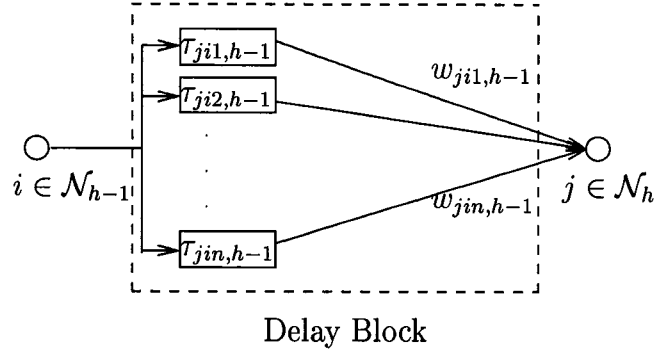


Figure 1.2: Delay Block: Basic time-delay connections between two processing units (node i of layer $h-1$ and node j of layer h) in *ATNN*, and n is the number of delays applied ($n = K_{ji,h-1}$).

with adaptive time delays. Although other artificial neural network architectures that make use of time delays have been suggested (for example in [143, 142, 26, 13, 20]) these paradigms employ different training rules or different network topologies compared to the *ATNN*. Review of related researches will be given in Chapter 2.

We use an algorithm with adaptive time-delays [85, 28] which is a powerful tool for the dynamic learning of a temporal network and we call this resulting network an Adaptive Time-Delay Neural Network (*ATNN*). The *ATNN* model employs modifiable time delays along the interconnections between two processing units, and both time delays and weights are adjusted according to system dynamics in an attempt to achieve the desired optimization. The schematic architecture of the connections from one processing unit to another processing unit of the *ATNN* is depicted in Figure 1.2. The configuration of multiple interconnections between a single pair of units, each with its own delay, is called a

delay block. The entire network is constructed by the delay blocks which connect neurons layer by layer.

The system is schematically illustrated in Figure 1.3. Assume the task is to emulate the embedding linear or nonlinear function of a plant at time T due to the temporally changing input signals $\hat{x}(t)$. During the training phase, switch $SW1$ is on (closed) and input signals are fed into a three layered network constructed by delay blocks. The inputs are pre-processed if necessary. After inputs are fed forward through the network, the outputs are obtained from the last terminal and compared with the desired outputs of the plant. An error vector E is propagated backwards through the previous layers and used to adjust the weights and delay variables in the intermediate delay blocks. Switch $SW2$ remains off (open) during training. The adaptation of the delays and weights are derived based on the gradient descent method to minimize the energy or cost function during training. Weight modification is based on error back-propagation ([98]) and the mathematical derivation of the time-delay modifications is done with a gradient descent approach [27, 28, 85, 84]. The weights and time-delays are updated step by step proportional to the opposite direction of the error gradient respectively. Processing units do not receive data through a fixed time window, but gather important information from various time delays which are adapted via the learning procedure. With these mechanisms, the network implements the dynamic delays along the interconnections of the *ATNN*.

In some experiments, a production or recall phase is used, in which switch $SW1$ is turned off and $SW2$ is on and the outputs are recurrently fed back to the input terminal. The network can then generate a pre-trained trajectory or signal, and can thereby produce a good prediction of a series as complex as chaos.

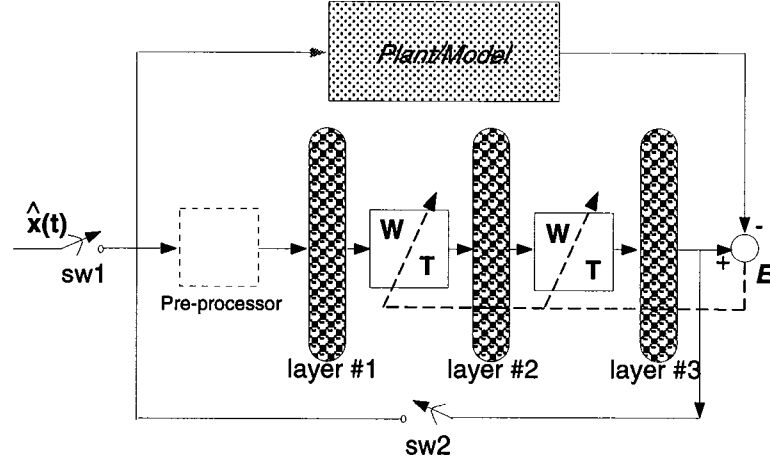


Figure 1.3: Schematic block diagram of system adaptation.

The feedback loop also allows for dynamic self-sustained activity and enables us to train attractors and trajectories into the network. The *ATNN* has been trained to distinguish the temporal properties and spatiotemporal correlations of various input patterns, and to perform predictions and signal generation [85, 90, 92, 89]. The network architecture is simple and efficient, and can achieve good generalization on applications problems.

1.3 Contributions

The principle contributions in this dissertation are to characterize performance and uses of an adaptive time-delay neural network, a recently developed architecture, on a series of novel applications that for the first time allow us to show the capabilities and promise of the network architecture on entire classes of engineering problems. The specific claims are described below:

1. We have characterized the promising aspects of the *ATNN*. The *ATNN* provides more dynamics and flexibility to approximate and classify spatiotemporal patterns with efficient performance. A recurrent configuration provides the capability to produce signals continuously based on past and current outputs. The network automatically attempts to optimize its configuration, which overcomes the limitations of previous work using arbitrary fixed delays [85] (see Chapter 3).
2. It has previously been known that a feed-forward network with time delays can be unfolded into a conventional feed-forward network with a time history as input. We have shown explicitly how this unfolding operation can occur, with a newly defined Network Unfolding Algorithm (*NUA*) that involves creation of virtual units and moving all time delays to a preprocessing stage consisting of the time histories. The *NUA* provides a tool for analyzing the complexity of the *ATNN*. From this tool, we concluded that the *ATNN* reduces the cost of network complexity by at least a factor of $O(n)$ compared to an unfolded *Backpropagation* net. We then applied the theorem of Funahashi, Hornik et al and Stone-Weierstrass to state the general function approximation ability of the *ATNN*. We furthermore show a lemma (Lemma 1) that the adaptation of time-delays is mathematically equivalent to the adjustment of interconnections on a unfolded feed-forward network provided there are a large enough number ($h2^{nd}$) of hidden units. Since this number of hidden units is often impractically large, we can conclude that the *TDNN* and *ATNN* are thus more powerful than *BP* with a time history. Thus interconnections can be changed by a gradient descent paradigm (see Chapter 3 Section 3.5, 3.6 and 3.7).

3. We have evaluated the behavior of internal hidden units over a series of applications. The internal nodes are selective and concentrate on the recent or past data due to the time-delay variables on incoming connections. This provides an extremely flexible way to explore dynamic representation and learning in situations where inputs arrive over a period of time and outputs are produced over a period of time (see Chapter 3) [84]. Each hidden layer node is a feature detector, like in a classic feedforward network trained by Back-Propagation. But in the case of the *ATNN*, hidden units detect features that are temporal patterns. When the network generated the closed trajectories studied here, the feature detectors become oscillators (see Section 4.2.2).
4. The *ATNN* is capable of spatiotemporal trajectory learning and trajectory generation with different topologies, different training arcs, various data sampling rates, and any number of dimensions [90, 87]. Relationships have been discovered between the complexity of the topology and the number of hidden units, between pattern completion and different training arcs and data sampling rates (in Chapter 4).
5. It is shown that the *ATNN* is resilient to perturbation, and is capable of recovering a spatiotemporal signal from noisy data. The *ATNN* can then perform a pattern completion on a spatiotemporal pattern (e.g. multi-channel signals over time) [91, 87] (see Section 4.5).
6. With a special recurrent configuration, the *ATNN* has the ability to learn a closed trajectory that becomes an attractor of the dynamic network. Thus we show how to train an attractor into a neural network. This is a

novel capability that can be naturally applied to a wide range of problems. Whereas most work on nonlinear dynamic systems is to identify attractors given the equations that govern the system, we are taking the opposite approach of choosing the attractor then training the system parameters (e.g. the weights) so that the neural net has the chosen attractor(s). We have identified promising applications for this capability (see Section 4.6).

7. The *ATNN* is applied to chaotic time series prediction with the example of the Mackey-Glass delay differential equation. We demonstrate that this complicated prediction problem can be accomplished by a simple *ATNN* architecture. Performance evaluation is made by training the *ATNN* with various choices of adaptive parameters, and the resulting network is superior to related networks. The noise tolerance ability of the *ATNN* is also shown better than the *TDNN*. Furthermore, the embedding of nonlinear system dynamics can be resolved adaptively, unlike exhausting trial and error efforts in conventional methods. We can conclude from this study that the *ATNN* is a good tool to tackle temporally changing signal processing and prediction problems [84, 89] (see Chapter 5).
8. The feasibility of distinguishing multiple type components of exo-atmospheric targets is demonstrated by applying the *ATNN*. Exo-atmospheric targets are especially difficult to distinguish using currently available techniques because all target parts follow the same spatial trajectory. Thus classification must be based on light sensors that record signal over time. The *ATNN* performs highly complex mapping on a limited set of training data and achieves better generalization to overall trends of situations (see Chap-

ter 6).

9. The related architecture of the *TDNN* is capable of graceful degradation, classification of spatiotemporal patterns, and location independence by bias offset (see Appendix A).

1.4 Guide to the Dissertation

This dissertation is organized as following: Chapter 2 provides background materials and a review of related work. Chapter 3 sets up the basic theoretical framework of this thesis. In the following three chapters, Chapter 4–6, we show how the *ATNN* works as a dynamic learning machine and demonstrate its ability in applications to spatio-temporal pattern generation, chaotic time-series prediction and pattern recognition. Finally, Chapter 7 concludes the work and discusses some possible future research issues. A brief overview of this dissertation is as following:

Chapter 2: The fundamental background of the feedforward neural network is provided. Related work to this research is reviewed.

Chapter 3: We describe the derivation of the mathematical theory of time-delay adaptation, based on Gradient Decent and Backpropagation theory. The network is set in a general frame work. The unfolding algorithm (*NUA*) is explicitly defined for translating *ATNN* and *TDNN* to a feed-forward network with a time lag in input history. A theorem of universal spatiotemporal approximator is formulated. A lemma and properties are provided and suggest that with either layer of time-delays adapted, the

ATNN can perform as well or better than a feed-forward network with a large amount of hidden units.

Chapter 4: A rich repertoire of capabilities and novel properties are demonstrated in the application to spatiotemporal trajectory pattern processing. This network is used to generate circular and figure-eight trajectories, to perform signal production, and to learn repetitive spatial motions. Spatiotemporal signal production has the property that initial segments of signals that contain large amounts of noise can be "cleaned up" to result in trained trajectory motions. The network also displays attractor production consistent with those trained trajectories, training position invariance, and noise removal when only part of the pattern is trained.

Chapter 5: We further apply the *ATNN* to the chaotic time series prediction of the example Mackey-Glass differential equation. We discuss the embedding dynamics, training set size and performance, and prediction over time. We demonstrate this neural network approach to be flexible, to predict the chaotic dynamics, and to achieve good performance. A comparison between the *ATNN* and related networks (*BP* and the *TDNN*) is reviewed.

Chapter 6: A real world application of target component discrimination is presented. The *ATNN* is shown to recognize targets as earlier as possible regardless of different target types, different aspect angles, tumbling situations or broken pieces. The network is shown to be robust.

Chapter 7: We summarize our research results. Further discussions of the implication of this research on various related research fields are made. We

also comment on future directions for research.

Chapter 2

Review of Related Work

The class of temporal or spatiotemporal processing and classification networks can be relatively categorized by the ways of representing and convolving input data, and by how the data is processed [95, 48]. In this chapter we shall review literature which we consider as landmarks in the development of temporal and spatiotemporal network theory in addition to all literature that is closely related to the subject of this thesis, and its applications to specific problems in practice.

Section 2.1 gives the basis of feedforward network theory and then introduces how the time factor is incorporated as time-delays to make possible dynamic system identification and dynamic behavior. In Section 2.2 to Section 2.5, we review key literature based on the categories of networks which work with preprocessing and spatial representation of temporal data, time-delay architectures, partial or fully recurrent connections, and short-term or long-term memory integration. Further notes on published research are in Section 2.7

2.1 Fundamentals: BP and BP Through Time

Since Rosenblatt invented the *Perceptron* in 1957 [126], enthusiasm of using neural nets type concepts to computation and analysis has been abundant. In spite of a quiescent period in the support of the field (1969 – 1986), interest has flourished recently. One of the key attributes of this unique approach is that neural nets can learn from their experience in a training environment. Engineers and computer scientists see neural nets as one style of *parallel distributed computing* which may usefully be recruited for solving complex problems in pattern recognition and classification, associative memory, and function optimization. A very important milestone in the development of this technology is ‘Back error propagation’ or simply called *Backpropagation* (*BP*) which gave significant inspiration to the field. *BP* has been widely implemented to practical applications, with tremendous impact.

As reviewed by Grossberg in [54], *BP* was discovered by Werbos [159], rediscovered independently by Parker [108], and made popular by Rumelhart, Hinton and Williams [128].

Essentially, the idea is to perform a gradient descent on the error of the network:

$$E = \frac{1}{2} \sum_{j=1}^N (t_j - y_j)^2, \quad (2.1)$$

where N is the total number of network output nodes, t_j is the desired target value and y_j denotes the net’s response to an input vector or pattern. The weights are adjusted to reduce the difference of the net’s response and the desired target value according to the *delta rule* by the following formula:

$$\Delta w_{j,i} = \eta \sigma'(a_j)(t_j - y_j)x_i \quad (2.2)$$

where $\sigma'(a_j)$ represents how quickly the activation changes due to excitation of input x_i , and η denotes the learning rate. In general, the delta rule may be written as

$$\Delta w_{i,j} = \eta \delta_j x_i \quad (2.3)$$

For output nodes:

$$\delta_j = \sigma'(a_j)(t_j - y_j) \quad (2.4)$$

The errors of output nodes then further propagate backwards to the units in the hidden layer(s) and contribute the changes of weights. Consider a hidden node k and let I_k be the set of nodes which node k connects to (I_k is the *fan-out* of k), therefore taking into account the contributions to the error from all of these nodes in I_k . We must sum these products, so δ^k becomes Equation 2.5.

$$\delta_k = \sigma'(a_k) \sum_{j \in I_k} \delta_j w_{j,k} \quad (2.5)$$

In most real world phenomena, information is contained in the temporal or spatio-temporal relationships between collected or observed data. Recently, physical sciences researchers have started to investigate these problems. Traditionally physics has sought explanations by supposing that there is some simple underlying global model, usually consisting of a differential equation, to which the real situation is supposed to approximate as a function of observations over time [5, 16]. With the hypothesis that current or future variations are affected by past events, the evaluation at time t will be more accurate if we take into account the changes observed at earlier times.

"Backpropagation through time", proposed by Werbos, is a extended technique of Backpropagation which is suitable for dynamic systems analysis [161]. Early in 1981, an idea similar to "Backpropagation through time" was imple-

mented as a prediction tool for a natural gas market [162]. The general formula is

$$\hat{x}(t+1) = F(x(t), u(t), b), \quad (2.6)$$

where $\hat{x}(t+1)$ represents the vector we want to predict based on the current value of vector $x(t)$, current auxiliary input vector $u(t)$, and the parameter vector b . Although this is a short term prediction technique (from t to $t+1$), long-term prediction can be achieved by a recurrent loop from output to input if the parameter set b can be obtained appropriately.

The "BP through time" technique was formulated by Werbos in his 1990 paper [161], in which he redid one type of generalization of the *BP* to a temporal domain. The time factor was introduced but not in a general way. Two extra terms were added to the net input of a non-input unit i shown in the following equation (using Werbos's notation [161]):

$$\text{net}_i(t) = \sum_{j=1}^{i-1} W_{ij} x_j(t) + \sum_{j=1}^{N+n} W'_{ij} x_j(t-1) + \sum_{j=1}^{N+n} W''_{ij} x_j(t-2) \quad (2.7)$$

where node 1 to $i-1$ are input units, and there are totally $N+n$ units in the whole net.

This general formula expresses that the net input to unit i includes not only the weighted sum of its inputs at time t , but also potentially takes the input values of any of the neurons (including itself) from the two previous time slots ($t-1$ and $t-2$). This allows all active units other than input neurons to recurrently feed in the outputs of any other units if there is a time lag in the connection. The matrices W' and W'' are the weights on those time-lagged connections. The network weight adjustment was either in a big batch mode or in pattern learning mode. This method has been applied to some practical problems, e.g. speech

recognition and dynamic system identification [156, 162, 160]. Nevertheless, the "BP through time" technique is limited because only two time steps are considered. The network will lack proper information if the true system behavior is based on influence of longer history steps.

2.2 Spatial Transformation

One technique for temporal signal processing is to obtain the transformation of temporal data (e.g. power spectrum of speech signal) so that the transformed spatial or static representation can be further processed by connectionist paradigms or conventional methodologies. As described in Lippmann's 1989 review paper [93] of speech recognition networks, most techniques used static representation of signal as input.

One example can be seen in work of Tom and Tenorio (1989) for speech recognition [144]. The extracted word data was segmented into 32 frames, each frame contained four feature representations and four coefficients from a linear predictive coding of that frame data. A multilayer BP network was used for training and recognition from the transformed static patterns.

One of the most prominent pieces of work in the neural network field was NETTalk by Sejnowski and Rosenberg (1987) [132]. This network simply trained by using a window of size 7 characters and scanned through a sentence. The output was a phoneme code of the middle character of each window. The network achieved 95% accuracy on the training set and 78% accuracy on the test set. This was a time history net, in which text data was scanned only on an input window, and the network was in standard *BP* fashion.

Goldberg and Pearlmutter (1989) used similar methods to learn inverse dynamics by training a *BP* net on the measured response of a physical arm. Rather than feeding the network position, velocity and acceleration data from a single point in time, they gave the network a window of position of two joints of the robot arm and tried to match the desired torque vector [50].

Gorman and Sejnowski (1988) applied *BP* to the classification of sonar returns from two objects. The input to the network was the spectral envelope preprocessed by superimposing a set of sampling aperture over the 2D display of a short-term Fourier Transform spectrogram of the sonar return signals. It can achieve a classification accuracy of 100% on training data and 90.4% on the test set [51, 52]. A similar approach was used by Roitblat et al. (1989) for identification of returning dolphin echoes with counterpropagation nets [125] and by Hemminger and Pao (1994) for detection and classification of underwater acoustic transients by using self-organizing network [62].

The advantage of this technique is that it is very straight forward to implement. However the performance is highly dependend on preprocessing schemes. Another disadvantage is that pattern duration is limited to window size, and the network itself is static. The only way that time is considered is through the input of a time history.

2.3 Time Delay Networks

A network that solves temporal sequence problems must temporarily store the feature information and then provide key elements for recognition at a later stage. Based on this approach, researchers have developed architectures either

by adding tapped-lines to input nodes or to hidden nodes or in combinations of both.

Tam and Perkel (1989) proposed a time-shifted hybrid model to compute the temporal correlation of firing patterns among many simultaneously recorded neuronal spike trains [141]. This model employed k independent hidden layers which connect to the same output layer. At the k th hidden layer, all inputs were shifted by $k - 1$ time steps. The correlation was observed by the value of weight matrices.

To tackle word-break problems and time-warping problems in speech recognition, variable length delay lines were designed by Tank and Hopfield (1987) [143, 142]. The network has a set of "delay" filters, and each delay filter was associated with a delay function (Gaussian shape) detecting stimulus over time. Then the outputs of delay line filters were sent to a decision maker and recognize the signal as a whole. However, the number of the delay lines was fixed and must be pre-determined before training.

The Time Delay Neural Network (*TDNN*) proposed by Waibel et al. (1989) has wide publicity and has been successfully applied to phoneme recognition. The delay lines were introduced to all computing units with the idea that each layer relate and compare current input to the past history of events. The recognition performance was shown superior to Hidden Markov Models [149]. Several enhancements were made to improve the *TDNN* including using an objective function for training the networks that made recognition of multiple speakers more accurate [152, 150, 153]. Waibel further constructed a modular *TDNN* by combining the hidden structure of pre-trained phonetic subnets (e.g. BDG-net, PTK-net). The resulting net formed a large scale network for all consonant

recognition. It not only achieved 96.0% recognition accuracy but also avoided the excessive learning time that would be necessary to train such large network [148]. A similar technique was applied for French digit recognition of multiple speakers by Bottou et al. (1990) [14].

Unnikrishnan, Hopfield and Tank (1991) have built an analog circuit implementing a neural network with time-delays connected between tone detectors and recognition units for speaker dependent digit recognition [146]. An analog perceptron learning rule was used. They demonstrated their recognition scores were greater than 99%.

Wan (1990) introduced a network by modeling each synapse as a linear filter restricted to discrete time and characterized as a Finite Impulse Response (FIR) [154, 155]. The resulting structure is similar to a *TDNN*.

The *TDNN* has been very widely utilized in the applications of signal recognition and pattern recognition. Nevertheless, this type of network is constrained to an *a priori* fixed and predetermined window size and fixed time-delays. Several researchers including ourselves have addressed this limitation of fixed size window. Our approach is summarized in Chapter 3 and in [85, 84, 90]. Adaptive techniques were implemented to adjust both weight and time-delays. Bodenhansen (1990) proposed a generalization of the recirculation network by adding time-delays in the connections and adjusting associated variables using separate learning procedures [12, 13]. Another good example is the “Continuous-Time Temporal Back-Propagation” proposed by Day and Davenport [27, 28], which is equivalent to the *ATNN* described in Chapter 3. In these work, they have demonstrated that with adaptive time-delays, the network is allowed to discover simpler and more accurate mappings than can be achieved with fixed delays.

Another interesting *TDNN* type of network is “*Oversampled Reverse TDNN*” proposed by Simard and Le Cun (1992) [134]. This type of network incorporates time-delays in between layers but the connections are in a reverse *TDNN* fashion with input and output interchanged. Furthermore, the states of the unit in a higher level are updated at a faster rate than the states of units in a lower layer. The resulting network was trained to generate multiple letter signatures.

2.4 Recurrent Networks

Another category of popular networks for spatiotemporal signal processing is recurrent networks, in which inputs of the network contain the current input data and previous state of the net (outputs). The recurrent connections provide ways of convolving the past network states and the current state as references to the network decision functions. Internal representation of past context can be captured by partial feedback to a set of nodes.

Pineda (1987) independently introduced recurrent backpropagation neural network as a spatiotemporal generalization of the *BP* network [111, 112]. With outputs fed back to extra input nodes, we can view the recurrent net as a dynamic system performing a mapping from a set consisting of input variables and initial system states to a set consisting of the system output states over time.

Pearlmutter (1989) has developed a continuous time recurrent network for state space trajectory learning. The network can generate the learned trajectory after 1000 epochs of training [109, 110].

Sato (1990) proposed a similar algorithm to that of Pearlmutter’s but with a somewhat different weight updating procedure. In Sato’s method, the connection

weights are fixed for a given period of time under a given external input and initial conditions. The weights are adapted with the hope that the total error from the time dependent teacher signal is maximally reduced [131, 130].

Sun et al. (1992) utilized a fully recurrent network with a special process of removing time warping, and can handle classification of time warped trajectory [139]. Another example of trajectories classification was demonstrated by Sotelino et al. (1994) [137]. Other interesting application in cursive script of handwritten character recognition can be seen in Hakim et al. [58] and Hoffman et al. [68].

The recurrent net has the advantage that no buffer or memory is needed to store the input or intermediate states. Problems of the recurrent nets include that the networks are very computational intensive, more local minima may be created (making convergence difficult during training), it is extremely sensitive to initial conditions and learning rate, and training is more of a challenge.

2.5 Temporal Integration Nets

One class of networks are with modified individual neurons so that they are able to sum time-varying data over time and store the features over time. Cohen, Grossberg and Stork (1987) introduced a special network in which individual units have a time-dependent activation function where input data decays over time. It was applied to speech analysis [23].

Mozer (1989) summarized three forms of temporal integration nets in temporal processing architectures [103, 104]. The main difference is in its kernel weight functions. The simplest type of memory is a buffer containing n most recent data

and these data are then fed into the network. It is a net with a time-delay in the input layer. We have discussed a lot of examples in Section 2.3. The other kernels are exponential trace memory and gamma memory. The exponential trace memory has been studied by Mozer (1989) [103], and Stornetta, Hogg and Huberman (1988) [138].

de Vries and Principe (1990) have proposed a memory model with kernel of the gamma density function for discrete-time processing [37, 36, 116]. The resulting network is a generalized version of *TDNN*. The enhanced focused gamma model was recently applied to transonic wind tunnel system dynamic behavior identification and word spotting [117, 118].

2.6 Oscillatory Network and Attraction

Recently, researchers have observed that neural networks with recurrent connections are capable of dynamic behaviors, including oscillations, attractors, limit cycles and chaos. These special activities imply that the network can potentially be used for pattern recognition, associative memory, optimization and control problems. In particular, attractors could represent pattern classes, memories, or control actions.

Hayashi (1994) [60] has investigated experimentally the dynamics of an oscillatory network constructed by excitatory and inhibitory neuronal groups. Each excitatory cell is connected to an inhibitory cell to form a corresponding pair and has mutual connections to other excitatory cells. On the other hand, each inhibitory cell only connects to its corresponding excitatory cell. Their simulations have shown memory patterns can be retrieved for an input near the pattern, and

then a limit cycle is generated.

Doya and Yoshizawa (1989) have proposed a neural oscillator and temporal pattern memory model by utilizing recurrent network with continuous-time *BP* learning [39]. The network receives an external oscillatory input with some desired output wave form, then, after sufficient learning, the network can regenerate this oscillatory wave form autonomously. The network performance with various numbers of hidden units and different oscillatory periods of the input waveform was discussed. The mechanisms of frequency and pattern control in the neural rhythm generators were discussed by Matsuoka (1987) [97]. These mechanisms include the regulation of the stimulus intensity of the inputs, the temporal change of input, the change of synaptic weights.

Dayhoff, Palmadesso and Richards (1994) [35] have proposed a method to develop multiple attractors in neural networks by changing connection strengths according to a perturbation schedule. The number of accumulated attractors exceeds the usable capacity of the Hopfield net associative memory.

A new type of attractor, *terminal attractors*, for content addressable memory and pattern recognition in neural net was introduced by Zak (1989) [165]. Each fixed point becomes a singular solution of the dynamic system. He also showed that by appropriate combinations of such terminal attractors one can create attracting trajectories which will be "remembered" by the dynamical model.

Any initial state close to the fixed points is likely to be attracted into that memory state. The set of all such initial states which lead to a given point is called the basin of attraction of that memory state. The Hopfield net is an example with fixed point attractors.

Keeler (1986) [77] has introduced a new algorithm by taking random slices

through the state space to analyze basins of attractions in neural nets. An important feature of this work was the ability of smoothing the shape of the attraction basin and to alleviate the problem incurred by the complicated structure. However, this algorithm required extensive computation in terms of repeating for many initial conditions (e.g. 1000), to relax to a final state.

It is of great interest to adaptively adjust the parameters itself and train the dynamic system into the network. None of the previous work discussed how networks can make such adaptations. We address this issue in Chapter 4 of this thesis.

2.7 Further Literature

Neural net paradigms for temporal and spatiotemporal processing are not limited to the four categories presented above. Combinations of various techniques are also available. Bengio et al. combined several techniques such as FFT pre-processing, ear model, input compression, BP, delays, copies of activations in hidden units, recurrent loops and knowledge from speech science for speaker independent speech recognition [11].

The use of artificial networks for the management of sensory and motor trajectories was introduced by Eckmiller in 1989. In this approach, separate networks were used, one for storing the spatial parameters and the other neural map stores time parameters [41]. Neural-based robotic systems were suggested by Eckmiller for motor control [42]. The recent models of the primate oculomotor system are reviewed and compared with an artificial neural network solution of the robotic problems. The use of time delays in neural nets for coor-

dinate transformations, inverse kinematics, and position modeling were discussed in [42].

Ghosh and Deuser have proposed the “*Adaptive Spatio-temporal Recognizer*” [48]. This network adaptively estimates the confidence of similarity between feature vectors and detects and classifies based on the intensity of estimation.

Assuming a distribution of delays, short relative to the duration of each single pattern, and assuming adaptive thresholds, a sequence of patterns can be stored and retrieved by learning at a particular speed while replayed at variable speeds (Heskes and Gielen, 1992) [65].

Evidence that biological neural systems use intricate time structures has accumulated. Dayhoff and Gerstein (1983) have observed that a large proportion of spike trains have favored patterns which occur more often than is reasonably expected at random [31, 32]. The pattern codes are available for use by the nervous systems and may be related to events of information transfer. Synchronism and oscillations observed by Gray et al. (1989) [53] are candidate mechanisms for neural coding, representation and processing [30]. Evidence of temporal coding was observed by McClurkin, Zarbock and Optican (1994) from the responses of a neuron recorded in monkey primary visual cortex to the stimulus of various colors and patterns [99]. These codes may be the fundamental elements of visual perception, and could further play an important role in understanding how we see and how memory works.

Chapter 3

Adaptive Time-Delay Neural Network

3.1 Introduction

Understanding the dynamic behavior of neural networks is a key issue in the development of network architectures that recognize and produce temporal sequences and spatiotemporal patterns. Whereas simple feed-forward networks can be trained to accomplish pattern recognition tasks with complex nonlinear boundaries, they are limited to processing static patterns - patterns that are fixed rather than temporal in nature. We propose here a network that overcomes this limitation and that is capable of processing temporally modulated signals. Networks with this capability can play an important role in applications domains that have naturally time-varying properties to their signals and dynamic situations. These domains include identification and control as well as signal processing and speech recognition.

Researchers have shown that convolving time delays in recurrent artificial networks makes it possible to store, classify and recall time-sequences of states rather than just single states [79, 80, 136, 3, 64]. An important contribution in this area has been the time-delay neural network (*TDNN*) proposed by Waibel et al [151], which employs time-delays on connections in feedforward networks and has been successfully applied to speech recognition [152, 57]. The time-delay neural network also classifies spatiotemporal patterns and provides robustness to noise and graceful degradation [86]. Techniques such as backpropagation through time [161] have been applied to temporal pattern recognition but not with adaptive time-delays. However, a limitation of the *TDNN* as originally posed [151] is its inability to learn or adapt the values of the time delays. Time delays are fixed initially and remain the same throughout training. The decision of how many time delays are needed and what are the most appropriate lengths for each time delay are often made by trial-and-error. As a result, the system may have poor performance due to the inflexibility of time delays and due to a mis-match between the choice of time delay values and the temporal location of the important information in the input patterns. In addition, the system performance may vary depending on the range of the time delay values.

To overcome this limitation, we have used an Adaptive Time Delay Neural Network model¹ (*ATNN*) [84, 27, 28]. This network adapts its time delay values as well as its weights during training, to better accommodate to changing temporal patterns, and to provide more flexibility for optimization tasks. The

¹We independently derived this architecture and were the first to publish in a widely-disseminated forum [86, 84, 90, 92, 88, 89, 87] although another publication with limited circulation proposed a similar architecture in 1991, due to Day and Davenport [27].

ATNN used here allows arbitrary placement of time delays along interconnections and adapts those time delays independently of one another. Furthermore, time-windows are not used as in previous work [13, 151] but instead classification relies on a set of individual time delay values associated with each interconnection. Although other artificial neural network architectures that make use of time delays have been suggested [143, 26, 28, 13, 20], these paradigms employ different training rules or different network topologies, and the network presented here is simple to use and has a general formulation. The *ATNN* has been applied to chaotic time series prediction with promising results, and it has been shown that the noise tolerance or generalization capability is superior to that of the *TDNN* [84].

In this chapter, we describe the *ATNN* network architecture and learning theory.

3.2 Network Architecture and Theory

The proposed *ATNN* model employs modifiable time delays along the interconnections between two processing units, and both time delays and weights are adjusted according to system dynamics in an attempt to achieve the desired optimization. The schematic architecture of the connections from one processing unit to another processing unit of the *ATNN* is depicted in Figure 3.1. The configuration of multiple interconnections between a single pair of units, each with its own delay, is called a delay block. Node i of layer $h - 1$ is connected to node j of the next layer h , with the connection line having an independent time-delay $\tau_{jik,h-1}$ and synaptic weight $w_{jik,h-1}$.

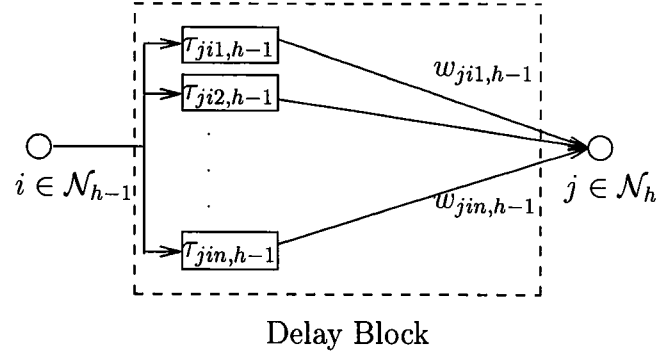


Figure 3.1: Delay Block: Basic time-delay connections between two processing units (node i of layer $h - 1$ and node j of layer h) in *ATNN*, and $n_{ji,h-1}$ is the number of delays applied.

The entire network is constructed by the delay blocks which connect neurons layer by layer as illustrated in Figure 3.2. The multilayer network is not necessarily fully connected layer by layer, but may be sparsely connected, and connections can skip layers. It is not necessary to have the same number of delays for different units in the same layer or the same delay values from different units, since the computation is local for each interconnection. Each connection can have an arbitrary delay value. For the sake of simplicity, in this discussion we assume the network is layered and fully connected layer by layer. Sparse connectivity is considered by setting some weights to zero in an otherwise fully connected architecture. To illustrate, a three layered *ATNN* is shown in Figure 3.3.

Next we propose definitions that are used to describe a general *ATNN* architecture with flexible configuration.

Definition 1 n_{ji} is the number of time-delays employed on the connections to

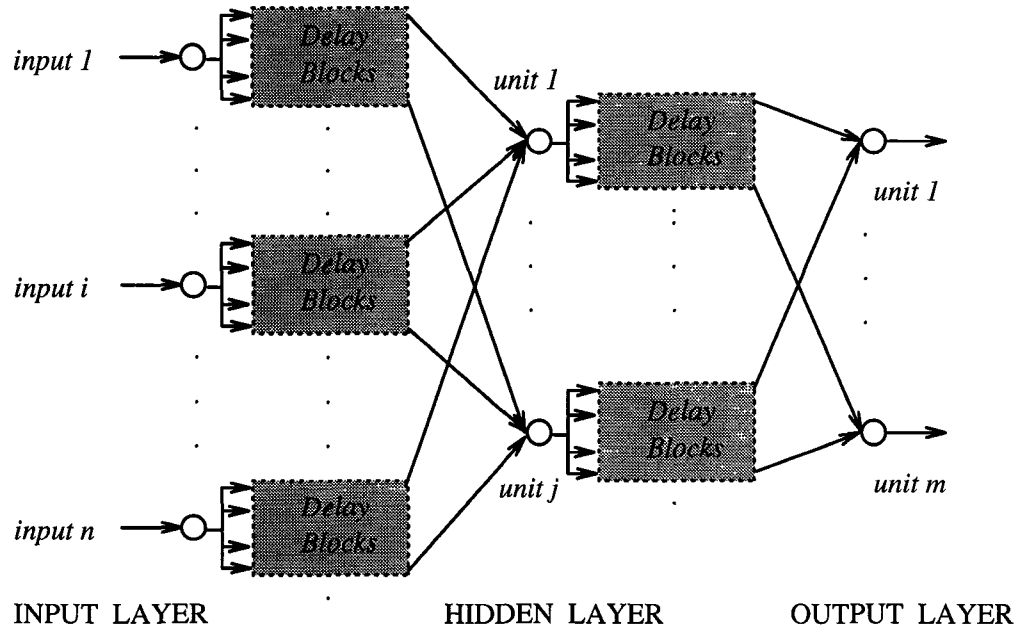


Figure 3.2: Layered *ATNN*: symbolic diagram

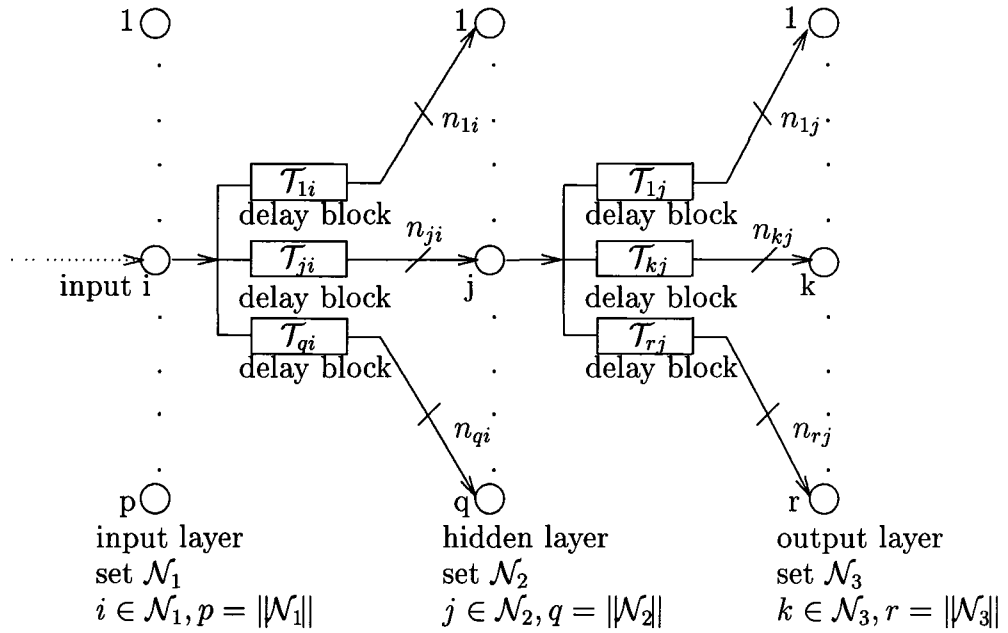


Figure 3.3: Three layered *ATNN*

node j from node i (i.e., the number of samples in the time frame $\mathcal{T}_{ji}$).

Definition 2 Time frame: Time frame $\mathcal{T}_{ji}$ is a set of time delays $(\tau_{ji1}, \dots, \tau_{jin_{ji}})$ employed on the connections to node j from node i . The time frame can have an additional index h ($\mathcal{T}_{ji,h}$) to signify layer h of time-delay.

Definition 3 The set of time frames $\mathcal{T}_i$ for connections that originate at node i is defined as:

$$\mathcal{T}_i = (\mathcal{T}_{1i}, \dots, \mathcal{T}_{ji}, \dots, \mathcal{T}_{qi}), \quad (3.1)$$

where q is the number of units that receive connections from node i .

Definition 4 $p_{ji}(t, \mathcal{T}_{ji})$ is the set of signal values transmitted to node j from node i via time frame $\mathcal{T}_{ji}$ at time t . Thus, $p_{ji}(t, \mathcal{T}_{ji}) = [s_i(t - \tau_{ji1}), \dots, s_i(t - \tau_{jin_{ji}})]$, where $s_i(t)$ is the signal from node i at time t .

Definition 5 $P_i(t, \mathcal{T}_i)$ is the set of signal values transmitted from node i to other nodes at time t . Thus,

$$P_i(t, \mathcal{T}_i) = \begin{pmatrix} p_{1i}(t, \mathcal{T}_{1i}) \\ \vdots \\ p_{ji}(t, \mathcal{T}_{ji}) \\ \vdots \\ p_{qi}(t, \mathcal{T}_{qi}) \end{pmatrix} = \begin{pmatrix} [s_i(t - \tau_{1i1}), \dots, s_i(t - \tau_{1in_{1i}})] \\ \vdots \\ [s_i(t - \tau_{ji1}), \dots, s_i(t - \tau_{jin_{ji}})] \\ \vdots \\ [s_i(t - \tau_{qi1}), \dots, s_i(t - \tau_{qin_{qi}})] \end{pmatrix}$$

Let $J_{i,h}$ be a set of nodes that receive connections from node i on layer h . Node i transmits the set of pattern values $P_i(t, \mathcal{T}_i)$ to selected subset $J_{i,h}$. In other words, $\forall j \in J_{i,h}$, node j receives pattern $P_i(t, \mathcal{T}_{ji})$ from node i through a time frame $\mathcal{T}_{ji}$. The samples in the time frame $\mathcal{T}_{ji}$ correspond to the delays $\tau_{jix,h}$, where $x = 1, 2, \dots, n_{ji}$, where $n_{ji} = \|\mathcal{T}_{ji}\|$, $j \in J_{i,h}$, and h is the index of

the layer in which i belongs. In the definitions above, we have omitted the layer index h for simplicity.

Definition 6 $\mathcal{N}_h$ is the set of nodes $\{1, 2, \dots, |\mathcal{N}_h|\}$ of layer h

Thus, each node j of layer h (eg. $j \in \mathcal{N}_h$), receives n_{ji} inputs from each $i \in \mathcal{I}_{j,h}$, where $\mathcal{I}_{j,h}$ is the subset of nodes on layer $h-1$ ($\mathcal{I}_{j,h} \subseteq \mathcal{N}_{h-1}$) that connects to unit j on layer h . The total number of inputs to j is: $m_j = \sum_{i \in \mathcal{I}_{j,h}} n_{ji}$.

In this model, we assume that $p = \|\mathcal{W}_1\|, q = \|\mathcal{W}_2\|, r = \|\mathcal{W}_3\|$ are fixed, and that the connectivity between layers is fixed ($\|J_{i,h-1}\|$ for the same layer is fixed), and for the sake of simplicity, we assume $\|J_{i,1}\| = \|\mathcal{W}_2\|$ and $\|J_{i,2}\| = \|\mathcal{W}_3\|$ (fully connected). In general, the number of samples in the time frame, n_{ji} , and the values of the delays (therefore the set $\mathcal{T}_{ji}$) are variables. In this research, we assume that n_{ji} is fixed (selected) and allow $\tau_{jix,h-1}$ to be variable. We also assume that n_{ji} is the same for all $i \in \mathcal{I}_{j,h}$ and for all $j \in J_{i,h-1}$. If n_{ji} is the same (fixed) for all $i \in \mathcal{I}_{j,h}$, then $m_j = n_{ji} \cdot |\mathcal{I}_{j,h}|$.

Definition 7 n_{h-1} is defined as the number of time-delays on connections originating at node i on layer $h-1$ where $n_{h-1} = n_{ji}$ for all j .

Then our model possesses the following property:

Property 1 For each node $i \in \mathcal{N}_{h-1}$ connecting to all $j \in J_{i,h-1}, J_{i,h-1} \subseteq \mathcal{N}_h$, there is a time frame $\mathcal{T}_{ji}$ with n_{h-1} samples, corresponding to the delays $\tau_{jix,h-1}, x = 1, \dots, n_{h-1}$. Inputs to node j (on layer h) are from node i (on layer $h-1$), and there are $n_{h-1} \cdot \|\mathcal{N}_{h-1}\|$ such inputs.

In this model, the adaptation variables are time-delay and weight ($\tau_{jik,h}$ and $w_{jik,h}$). Each node sums up the net inputs from the activation values of the

previous neurons, through the corresponding time-delays on each connection line, i.e. at time t_n unit j on layer h receives a weighted sum :

$$S_{j,h}(t_n) = \sum_{i \in \mathcal{N}_{h-1}} \sum_{k=1}^{K_{ji,h-1}} w_{jik,h-1} \cdot a_{i,h-1}(t_n - \tau_{jik,h-1}) \quad (3.2)$$

where

$a_{i,h-1}(t_n - \tau_{jik,h-1}) \equiv$ the activation level of unit i on layer $h-1$ at $t_n - \tau_{jik,h-1}$

$\tau_{jik,h-1} \equiv$ the time-delay of the k th connection to node j of layer h
from node i of layer $h-1$

$K_{ji,h-1} \equiv$ the total number of connections to node j (layer h) from
node i of layer $h-1$

$w_{jik,h-1} \equiv$ the synapse weight of the k th connection to node j from
node i of layer $h-1$, and $k = 1, 2, \dots, K_{ji,h-1}$

Then the output of node j is governed by a nondecreasing differentiable function f of the net input (sigmoid function is selected in this chapter) :

$$a_{j,h}(t_n) = \begin{cases} f_{j,h}(S_{j,h}(t_n)) & \text{if } h \geq 2 \\ a_{j,0}(t_n) & \text{if } h = 1 \end{cases} \quad (3.3)$$

where

$$f_{j,h}(x) = \frac{\beta_{j,h}}{1 + e^{-\alpha_{j,h}x}} - \gamma_{j,h} \quad (3.4)$$

and $a_{j,0}(t_n)$ denotes the sampled value of the j th channel of the input signal at time t_n and $\alpha_{j,h}$, $\beta_{j,h}$ and $\gamma_{j,h}$ are real numbers which define the symmetry point $(0, \frac{\beta_{j,h}}{2} - \gamma_{j,h})$ of $f_{j,h}(x)$ if x is symmetric to 0, the range $[-\gamma_{j,h}, \beta_{j,h} - \gamma_{j,h}]$ of $f_{j,h}(x)$ and the steepness of $f_{j,h}(x)$ (e.g. $f'_{j,h}(0) = \frac{\alpha_{j,h}\beta_{j,h}}{4}$). For simplicity, we use the same sigmoid function for each node in this model, although in practice $f_{j,h}$ may differ and be modifiable for each node.

3.3 Learning Paradigm

The schematic architecture of the connections from one neuron to another neuron of the *ATNN* has been illustrated in Figure 3.1. Node i of layer $h-1$ is connected to node j of the next layer h , with the connection having time-delay $\tau_{jik,h-1}$ and synaptic weight $w_{jik,h-1}$. The multilayer network is not necessarily fully connected layer by layer, but may be sparsely connected, and connections can skip layers. It is not necessary to have the same number of delays from different units in the same layer or the same delay values from different units, since the computation is local for each interconnection. Each connection can have an arbitrary delay value. For the sake of simplicity, we assume the network is layered and fully connected layer by layer.

We assume in this discussion that network parameters are updated in a discrete manner and the input signals are differentiable and sampled at a Nyquist rate, i.e. the sampling time step $r = \frac{1}{2f_{max}}$, where f_{max} is the maximum frequency of all input channels. We define the following notation:

$L \equiv$ the number of layers in the network.

$\mathcal{N}_h \equiv$ the set of nodes $\{1, 2, \dots, |\mathcal{N}_h|\}$ of layer h .

$\tau_{jik,h-1} \equiv$ the time-delay of the k th connection to node j of layer h from node i of layer $h-1$

$K_{ji,h-1} \equiv$ the total number of connections to node j (layer h) from node i of layer $h-1$

$\mathcal{T}_{ji,h-1} \equiv$ the set of delays on connections to node j (layer h) from node i of layer $h-1$, i.e. $\mathcal{T}_{ji,h-1} = \{\tau_{ji1,h-1}, \tau_{ji2,h-1}, \dots, \tau_{jim,h-1}\}$, where $m = K_{ji,h-1}$

$a_{i,0}^\mu(t) \equiv$ the i th channel of the input training pattern μ at time t

$t_n \equiv$ the n th sampling time, where r is a single time step (e.g., $t_n = nr$)

$w_{jik,h-1} \equiv$ the synapse weight of the k th connection to node j from node i of layer $h - 1$, and $k = 1, 2, \dots, K_{ji,h-1}$

The adaptation of the delays and weights are derived based on the gradient descent method to minimize the error measure E during training. The training set consists of a set of spatiotemporal patterns and target outputs over time. Thus, for spatiotemporal inputs, training may proceed in any of three ways: *batch mode*, *pattern mode*, and *incremental mode*. Theoretically, the true gradient descent should be based on the total error measure over all spatiotemporal training patterns over all times when patterns are presented. This is called *batch mode* learning: the parameters are updated after all patterns have been presented. It requires additional local storage for each connection. Instead of measuring all patterns, *pattern mode* takes into account a single complete spatiotemporal pattern and the network parameters are updated after each spatiotemporal pattern is presented.

The third alternative is *incremental mode*. This incremental approach updates all parameters after every time step of each spatiotemporal pattern is considered. For a static back-error propagation network [29], pattern mode learning is the same as incremental mode learning, and batch mode has been shown equivalent to pattern mode for small learning rates [119]. For any of these training modes, learning will become inefficient and parameters may overshoot if the learning rate is too large. The relative effectiveness of these differing approaches depends on the problem, but the latter one is likely to be superior for very regular or redundant training sets [63].

We will follow the incremental updating method (on-line learning) in this discussion. An instantaneous error measure is defined as MSE:

$$E(t_n) = \frac{1}{2} \sum_{j \in \mathcal{N}_L} (d_j(t_n) - a_j(t_n))^2 \quad (3.5)$$

where L denotes the output layer and $d_j(t_n)$ indicates the desired (target) value of output node j at time t_n .

The time-delay is modified step by step proportional to the opposite direction of the error gradient with respect to this delay. The updating rule is therefore:

$$\Delta \tau_{jik,h-1} \equiv -\eta_1 \frac{\partial E(t_n)}{\partial \tau_{jik,h-1}} \quad (3.6)$$

where η_1 is the learning rate.

3.4 Learning Algorithm Derivation

By the chain rule

$$\frac{\partial E(t_n)}{\partial \tau_{jik,h-1}} = \frac{\partial E(t_n)}{\partial S_{j,h}} \frac{\partial S_{j,h}(t_n)}{\partial \tau_{jik,h-1}} \quad (3.7)$$

The second factor of Equation (3.7) can be expressed as

$$\begin{aligned} \frac{\partial S_{j,h}(t_n)}{\partial \tau_{jik,h-1}} &= \frac{\partial}{\partial \tau_{jik,h-1}} \sum_{p \in \mathcal{N}_{h-1}} \sum_{q=1}^{K_{ji,h-1}} w_{jpq,h-1} a_{p,h-1}(t_n - \tau_{jpq,h-1}) \\ &= -w_{jik,h-1} a'_{i,h-1}(t_n - \tau_{jik,h-1}) \end{aligned} \quad (3.8)$$

We define

$$\rho_{j,h}(t_n) = \frac{\partial E(t_n)}{\partial S_{j,h}} \quad (3.9)$$

Substitute Equation (3.8) and (3.9) into Equation (3.7), we obtain

$$\frac{\partial E(t_n)}{\partial \tau_{jik,h-1}} = -\rho_{j,h}(t_n) w_{jik,h-1} a'_{i,h-1}(t_n - \tau_{jik,h-1}) \quad (3.10)$$

$$\stackrel{(3.6)}{\implies} \Delta\tau_{jik,h-1} = \eta_1 \rho_{j,h}(t_n) w_{jik,h-1} a'_{i,h-1}(t_n - \tau_{jik,h-1}) \quad (3.11)$$

To derive $\rho_{j,h}(t_n)$, we need to apply chain rule and consider two cases:

$$\begin{aligned} \rho_{j,h}(t_n) &= \frac{\partial E(t_n)}{\partial S_{j,h}} \\ &= \frac{\partial E(t_n)}{\partial a_{j,h}} \frac{\partial a_{j,h}(t_n)}{\partial S_{j,h}} \\ &= \frac{\partial E(t_n)}{\partial a_{j,h}} f'(S_{j,h}(t_n)) \end{aligned} \quad (3.12)$$

To find $\frac{\partial E(t_n)}{\partial a_{j,h}}$, we consider the following two cases:

1. If j is an output unit:

$$\frac{\partial E(t_n)}{\partial a_{j,h}} = -(d_j(t_n) - a_{j,h}(t_n)) \quad (3.13)$$

$$\stackrel{(3.12)}{\implies} \rho_{j,h}(t_n) = -(d_j(t_n) - a_{j,h}(t_n)) f'(S_{j,h}(t_n)) \quad (3.14)$$

2. If j is a hidden unit:

$$\begin{aligned} \frac{\partial E(t_n)}{\partial a_{j,h}} &= \sum_{p \in \mathcal{N}_{h+1}} \frac{\partial E(t_n)}{\partial S_{p,h+1}} \frac{\partial S_{p,h+1}(t_n)}{\partial a_{j,h}} \\ &= \sum_{p \in \mathcal{N}_{h+1}} \frac{\partial E(t_n)}{\partial S_{p,h+1}} \frac{\partial}{\partial a_{j,h}} \left(\sum_{i \in \mathcal{N}_h} \sum_{q=1}^{K_{pi,h}} w_{piq,h} a_{i,h}(t_n - \tau_{piq}) \right) \\ &= - \sum_{p \in \mathcal{N}_{h+1}} \rho_{p,h+1}(t_n) \left(\sum_{q=1}^{K_{pi,h}} w_{pj q,h}(t_n) \right) \end{aligned} \quad (3.15)$$

$$\stackrel{(3.12)}{\implies} \rho_{j,h}(t_n) = -[\sum_{p \in \mathcal{N}_{h+1}} \sum_{q=1}^{K_{pi,h}} \rho_{p,h+1}(t_n) w_{pj q,h}(t_n)] f'(S_{j,h}(t_n)) \quad (3.16)$$

We have now found ρ .

From Equation (3.11) it remains to find $a'_{i,h-1}(t_n - \tau_{jik,h-1})$. The value of $a'_{i,h-1}(t_n - \tau_{jik,h-1})$ can be approximated as following: From the elementary calculus we know that if the function $f(x)$ is differentiable, then

$$f(x_0 + h) - f(x_0) = h \frac{df}{dx} \Big|_{x=x_*} \quad (3.17)$$

for some point x_* such that $x_0 \leq x_* \leq x_0 + h$. The derivative is evaluated at a point x_* (between x_0 and $x_0 + h$), and by the Mean Value theorem of differential calculus, x_* always exists. Accordingly, we get

$$a'_{i,h-1}(t_n - \tau_{jik,h-1}) \approx \begin{cases} \frac{a(t_n) - a(t_{n-1})}{r} & \text{if } \tau_{jik,h-1} = 0 \\ \frac{a(t_{k+1}) - a(t_{k-1})}{2r} & \text{if } t_n - \tau_{jik,h-1} = t_k, \tau_{jik,h-1} \neq 0 \end{cases}$$

where $r = t_k - t_{k-1}, k \in \{0, 1, \dots, n\}$.

Theoretically, $\tau_{jik,h-1}$ can be zero or any positive real number, and the value $a_{i,h-1}(t_n - \tau_{jik,h-1})$ can be easily again approximated by the numerical approximation method. But, in this way, the network will be trained by a simulated pattern or noisy signal due to the approximation. Therefore, more consideration of this effect is necessary and this issue will be addressed elsewhere. In this model, we will consider all time-delays as integer times of time step, i.e. $\tau_{jik,h-1} = nr, n \in \mathcal{N}$, and $\Delta\tau_{jik,h-1}$ should be rounded before updated or save the non-rounded value and use the rounded value for updating.

The time-delay learning rule is summarized as follows:

$$\Delta\tau_{jik,h-1} = \eta_1 \rho_{j,h}(t_n) w_{jik,h-1} a'_{i,h-1}(t_n - \tau_{jik,h-1}) \quad (3.18)$$

where

$$\rho_{j,h}(t_n) = \begin{cases} -(d_j(t_n) - a_{j,h}(t_n)) f'(S_{j,h}(t_n)) & \text{,if } j \text{ is an output unit} \\ -[\sum_{p \in \mathcal{N}_{h+1}} \sum_{q \in \mathcal{T}_{pj,h}} \rho_{p,h+1}(t_n) w_{pqj,h}(t_n)] f'(S_{j,h}(t_n)) & \text{,if } j \text{ is a hidden unit} \end{cases}$$

Similarly, the learning rule for weights can also be obtained in the same manner and is summarized as following:

$$\Delta w_{jik,h-1} = \eta \delta_{j,h}(t_n) a_{i,h-1}(t_n - \tau_{jik,h-1}) \quad (3.19)$$

where $\delta_{j,h}(t_n) = -\rho_{j,h}(t_n)$, i.e.,

$$\delta_{j,h}(t_n) = \begin{cases} (d_j(t_n) - a_{j,h}(t_n))f'(S_{j,h}(t_n)) & , \text{if } j \text{ is an output unit} \\ (\sum_{p \in \mathcal{N}_{h+1}} \sum_{q \in \mathcal{T}_{p,h}} \delta_{p,h+1}(t_n) w_{pqh}(t_n))f'(S_{j,h}(t_n)) & , \text{if } j \text{ is a hidden unit} \end{cases}$$

Furthermore, the continuous real values of desired output are condensed to the center regime of the sigmoid output. Specifically, the target values are scaled to range $[-c\gamma_{j,h}, c(\beta_{j,h} - \gamma_{j,h})]$, where $-\gamma_{j,h}$ and $(\beta_{j,h} - \gamma_{j,h})$ are the upper bound and lower bound of the sigmoid function respectively, and c denotes a scale factor. We selected the scale factor $c = 0.5$ in this thesis, which significantly improved the mapping accuracy by a factor of 2 to 10 in terms of $NMSE$ ² compared to our previous simulation results that did not use scaling. These improved results occurred because with scaling, the target value is a real number that never reaches the upper or lower bound of the sigmoid function. Thus, the weight adjustment was more limited and stable when the asymptotic region was avoided.

The time delay values $\Delta\tau_{jim,h-1}$ can cross one another during learning, and can specify $\tau_{jim,h-1}$ to go below zero. We reset $\tau_{jim,h-1}$ to zero in this case. Processing units do not receive data through a fixed time window, but gather important information from various time delays which are adapted via the learning procedure. The largest time delay between two units can change during adap-

²normalized mean square error: $NMSE = \frac{E[(x(t) - \hat{x}(t))^2]}{E[(x(t) - E[x(t)])^2]}$, where $x(t)$ is the original signal value and $\hat{x}(t)$ is the network output value

tation, and the interconnection that has the largest time-delay can also change. With these mechanisms, the network implements the dynamic delays along the interconnections of the *ATNN*.

3.5 ATNN: Universal Spatiotemporal Function Approximator

The capability of multilayer feedforward networks has been theoretically studied. In previous work, Funahashi, Hornik, Stinchcombe and White have concluded that *"standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators"* [71]. Similar theorems can also be seen in [45, 114, 21]. The superpositions of a sigmoidal function in achieving universal approximation is also discussed [10].

In this section, we rephrase this mathematical statement and then extend these results to networks with time-delays.

Let $r \in \mathcal{N}$ and A^r be the set of all affine functions from $\mathcal{R}^r$ to $\mathcal{R}$: $A(x) = w \cdot x + b$ where w and x are vectors in $\mathcal{R}^r$, and $b \in \mathcal{R}$ is a scalar. Let $\phi(\cdot)$ be a measurable function from $\mathcal{R}$ to $\mathcal{R}$ and let $\Sigma^r(\phi)$ denote the class of functions $\{f(x) = \sum_{i=1}^q c_i \phi(A_i(x))\}$ from $\mathcal{R}^r$ to $\mathcal{R}$, where $x \in \mathcal{R}^r$, $c_i \in \mathcal{R}$, $A_i \in A^r$, $q \in \mathcal{N}$. We adopt Funahashi's main results as below:

Theorem 1 (adopted and restated from Funahashi [45]) *Let $\{x_1, \dots, x_n\}$*

be a set of distinct points of compact set K in Euclidean space $\mathcal{R}^n$ and let $f : \mathcal{R}^n \rightarrow \mathcal{R}$ be an arbitrary real valued function on K . If ϕ is a bounded and monotone increasing differentiable function, then for any $\epsilon > 0$, there exists an integer N and real constants c_i , θ_i , and $w_{i,j}$ ($i \in \{1, \dots, N\}, j \in \{1, \dots, n\}$) such that $\hat{f}(x_1, \dots, x_n) = \sum_{i=1}^N c_i \phi(\sum_{j=1}^n w_{i,j} x_j - \theta_i)$ of $\Sigma^N(\phi)$ satisfies $\forall x \in K, |f(x) - \hat{f}(x)| < \epsilon$. That is, with N hidden units, a three layered network of class $\Sigma(\phi)$ can approximate any function to a desired accuracy.

Proof of Theorem 1 can be referred to Funahashi (1989) [45] and Hornik, Stinchcombe and White (1989) [71].

Similar theorems can be applied to the TDNN and ATNN to approximate a spatio-temporal function with the extra degrees of freedom in the time-delay domain, considering the network is unfolded. Because the Stone-Weierstrass theorem played a central role in the proof of Theorem 1, we state it here for reference and later use. First we give a few necessary definitions before we introduce the Stone-Weierstrass theorem.

Definition 8 A family $\mathcal{A}$ of real functions defined on a compact set K is an algebra if $\mathcal{A}$ is closed under addition, multiplication and scalar multiplication.

Definition 9 A set $\mathcal{A}$ separates points on compact set K if for every $x, y, x \neq y$ in K , there exists a function f in $\mathcal{A}$ such that $f(x) \neq f(y)$.

Definition 10 A set $\mathcal{A}$ vanishes at no point of K if for each x in K there exists f in $\mathcal{A}$ such that $f(x) \neq 0$.

Theorem 2 (Stone-Weierstrass) Let $\mathcal{A}$ be an algebra of real continuous functions on a compact set K . If $\mathcal{A}$ separates points on K and if $\mathcal{A}$ vanishes at no point of K , then $\mathcal{A}$ consists of all real continuous functions on K .

Let $x_1[t], x_2[t], \dots, x_n[t]$ be n input signal channels measured/observed in time interval $[t_0, T]$ in a discrete manner. For fixed $\tau_{j,k} \geq 0, j, k \in \mathcal{N}$ and $t - \tau_{j,k}$ in interval $[t_0, T]$, we let $\{< x_1(t - \tau_{1,1}), \dots, x_1(t - \tau_{1,d}) >, \dots, < x_n(t - \tau_{n,1}), \dots, x_n(t - \tau_{n,d}) >\}$ be a set of distinct points of compact set K^d in Euclidean space $\mathcal{R}^{nd}$. We extend Theorem 1 and obtain the following corollary:

Corollary 1 (original contribution) *Let $f : \mathcal{R}^{nd} \rightarrow \mathcal{R}$ be an arbitrary real valued function on K^d . If ϕ is a bounded and monotone increasing differentiable function, then for any $\epsilon > 0$, there exists integer N , and real constants c_i, θ_i , and $w_{i,j,k}$ ($i \in \{1, \dots, N\}, j \in \{1, \dots, n\}$), such that*

$$\hat{f}(x) = \sum_{i=1}^N c_i \phi(\sum_{j=1}^n \sum_{k=1}^d w_{i,j,k} x_j(t - \tau_{i,j,k}) - \theta_i)$$

satisfies $\forall x \in K^d, |f(x) - \hat{f}(x)| < \epsilon$. In other words, with at least one hidden layer of N hidden units and d time-delays elements in each input-hidden connections pairs, a three layered TDNN network (with delays employed on the first layer) can approximate any spatiotemporal function to a desired accuracy.

Proof: We apply the Stone-Weierstrass Theorem and use the notations above. Given $\phi(\cdot) : \mathcal{R}^{nd} \rightarrow \mathcal{R}$ and let $\mathcal{A}$ is the set of all affine functions from $\mathcal{R}^{nd}$ to $\mathcal{R}$ such that $\mathcal{A}(x) = w \cdot x + b, x \in K^d$, i.e. $x = \{< x_1(t - \tau_{1,1}), \dots, x_1(t - \tau_{1,d}) >, \dots, < x_n(t - \tau_{n,1}), \dots, x_n(t - \tau_{n,d}) >\}$. We denote $\Sigma^{nd}(\phi)$ as the class of functions $\hat{h}(x) : \{\sum c_i \phi(\mathcal{A}(x))\}$. We need to show $\Sigma^{nd}(\phi)$ consists of all real functions on K^d . First we show $\phi(\mathcal{A})$ is separating on K^d that ensures $\Sigma^{nd}(\phi)$ is separating on K^d . Let $K^d \subset \mathcal{R}^{nd}$ be a compact set, so for ϕ from $\mathcal{R}^{nd}$ to $\mathcal{R}$, $\Sigma(\phi)$ is an algebra on K^d . Pick $a, b \in \mathcal{R}, a \neq b$ such that $\phi(a) \neq \phi(b)$. Pick $\mathcal{A}(\cdot)$ such that $\mathcal{A}(x) = a$ and $\mathcal{A}(y) = b, x, y \in K^d$, then we obtain $\phi(\mathcal{A}(x)) \neq \phi(\mathcal{A}(y))$. Therefore ϕ is separating on K^d .

Secondly, we need to show $\phi(\mathcal{A})$ vanishes at no point on K^d . Pick $b \in \mathcal{R}$ such that $\phi(b) \neq 0$ and set $\mathcal{A}(x) = 0 \cdot x + b$. For all $x \in K^d$, $\phi(\mathcal{A}(X)) = \phi(b)$, the result follows. $\square$

The proof is mathematically equivalent to that of Hornik, Stinchcombe and White's theorem of universal approximation with multilayer feedforward networks. We extend from n dimensional input space to nd dimensional space. In other words, each distinct input point x_i in n dimensional input space is expanded into a distinct points set $\langle x_i(t - \tau_1), \dots, x_i(t - \tau_d) \rangle$ in nd dimensional input space.

3.6 NUA and Complexity Analysis

From previous literature, researchers have indicated that a time-delay embedded network can be unfolded into feedforward network, but left the procedure of unfolding vague. In this section we elucidate this procedure in a very explicit four-step operation: Network Unfolding Algorithm (NUA).

For notational simplicity, we use a three layer ATNN to illustrate the unfolding operation NUA and to keep the number of time-delays in the same layer to be consistent. We use definitions of the network configuration in Section 3.2. Use Definition 6 and let $n_I = |\mathcal{N}_I|, n_H = |\mathcal{N}_H|, n_O = |\mathcal{N}_O|$ be the number of nodes on the input, hidden and output layers respectively. Use Definition 7 and let $n_1 = d_1, n_2 = d_2$ be the number of delays on connections originating at nodes on the input and hidden layers respectively. From Property 1, there are $d_1 \cdot n_I$ and $d_2 \cdot n_H$ inputs to each node on the hidden layer and the output layer respectively. Knowing the number of connections, we can unfold the ATNN by the following

algorithm:

Algorithm 1 (Network Unfolding Algorithm (NUA)) :

Step 1 – unfold inputs:

For each hidden node j do in parallel:

For each input node i do in parallel:

1.1 duplicate $(d_1 - 1)$ new input nodes and spread these nodes horizontally next to each original input node

1.2 move $(d_1 - 1)$ of the original connections to new input nodes correspondingly and retain the weight and time-delay on each connection

end

end

Step 2 – re-adjust input time lag:

For each hidden node j do in parallel:

2.1 remove the time-delays between input and hidden units

2.2 set the input values as $[p_{j1}(t, \mathcal{T}_{j1}), \dots, p_{jn_I}(t, \mathcal{T}_{jn_I})]$ accordingly, where

$\mathcal{T}_{ji}$ is the time frame (in Definition 2) on the connections from node i to

j , p_{ji} is the signal value vector transmitted from node i to j (Definition 4)

end

Step 3 – unfold hidden nodes:

For each output node k do in parallel:

For each hidden node j do in parallel:

3.1 duplicate $(d_2 - 1)$ new input nodes and spread these nodes horizontally next to each original hidden node

3.2 move $(d_2 - 1)$ of the original connections to new hidden nodes correspondingly and retain the weight and time-delay on each connection

to output node k

3.3 for each newly created hidden node do in parallel:

copy the whole branch which associates with the original hidden node in Step 2, and then connect to that new node as its branch and retain the weights

end

end

end

Step 4 – re-adjust input time lag:

For each each hidden node j and its newly created node from Step 3 do in parallel:

4.1 remove the associated time-delays between hidden and output units

4.2 re-adjust the input node time lag, such that the input node of each branch takes the vectors of $[p_{j1}(t - \tau_{k,j,1}, \mathcal{T}_{j1}), \dots, p_{jn_I}(t - \tau_{k,j,1}, \mathcal{T}_{jn_I})], \dots, [p_{j1}(t - \tau_{k,j,d_2}, \mathcal{T}_{j1}), \dots, p_{jn_I}(t - \tau_{k,j,d_2}, \mathcal{T}_{jn_I})]$ correspondingly

end

□

Let n_I, n_H, n_O be the number of units in input, hidden and output layer respectively. Let d_1 and d_2 be the number of time-delay connections between input-hidden node pairs and hidden-output node pairs respectively. We present a table (Table 3.1) which shows how the number of input and hidden units in a feedforward network or *BP* (with zero weight on connections to other nodes) are increased exponentially from unfolding. If the number of nodes in each layer is n , then the number will be increased by a factor of $O(n)$ and the input nodes is increasing by a factor of $O(n^2)$. Therefore the ATNN is indeed a very economic

architecture to achieve the same goal as feedforward network. For a network with more than two hidden layers, the unfold operation can be executed by repeating *Step 3* and *Step 4* from the lowest hidden layer to the highest layer, and expanding the net until reaching the output layer.

Figure 3.6 explicitly and graphically elucidates the unfold operation of an ATNN with net configuration $2 \xrightarrow{2} 2 \xrightarrow{2} 1$ (labeled as *original*). In *Step 1*, two new nodes ($2 \times 2 - 2 = 2$) are created (drawn in dotted circle) which are labeled as I'_1 and I'_2 , then spread these two newly created nodes next to its original node. In *Step 2*, each associated time-delay is removed by adjusting the time lag of each input node so that each node receiving information at correct time slot ($t - \tau_1, t - \tau_2, t - \tau_3$, and $t - \tau_4$). In *Step 3*, since there are two synapse connections from node H_1 to output node O_1 , we need to create another hidden node (labeled in H'_1) and then copy the whole branch of node H_1 and hook up to node H'_1 . In *Step 4*, the time-delays τ_5 and τ_6 are removed from connections between hidden node and output node. These time lag factors are pushed down to the input layer, so that each branch inherits its value from the upper level. In other words, the input vector of the branch under node H_1 becomes $[x_1(t - \tau_1 - \tau_5), x_1(t - \tau_2 - \tau_5), x_2(t - \tau_3 - \tau_5), x_2(t - \tau_4 - \tau_5)]$. The input vector of the other branch (under node H'_1) is $[x_1(t - \tau_1 - \tau_6), x_1(t - \tau_2 - \tau_6), x_2(t - \tau_3 - \tau_6), x_2(t - \tau_4 - \tau_6)]$.

The NUA can also be executed in an alternative order by unfolding the second layer first using the same procedure. This is illustrated graphically in Figure 3.6. In *Step 1* and *Step 2*, the hidden layer is unfolded first by creating a virtual node H'_1 and duplicating the entire branch of the original node H_1 . The input time lags of each branch are adjusted to corresponding time-delay τ_5 and

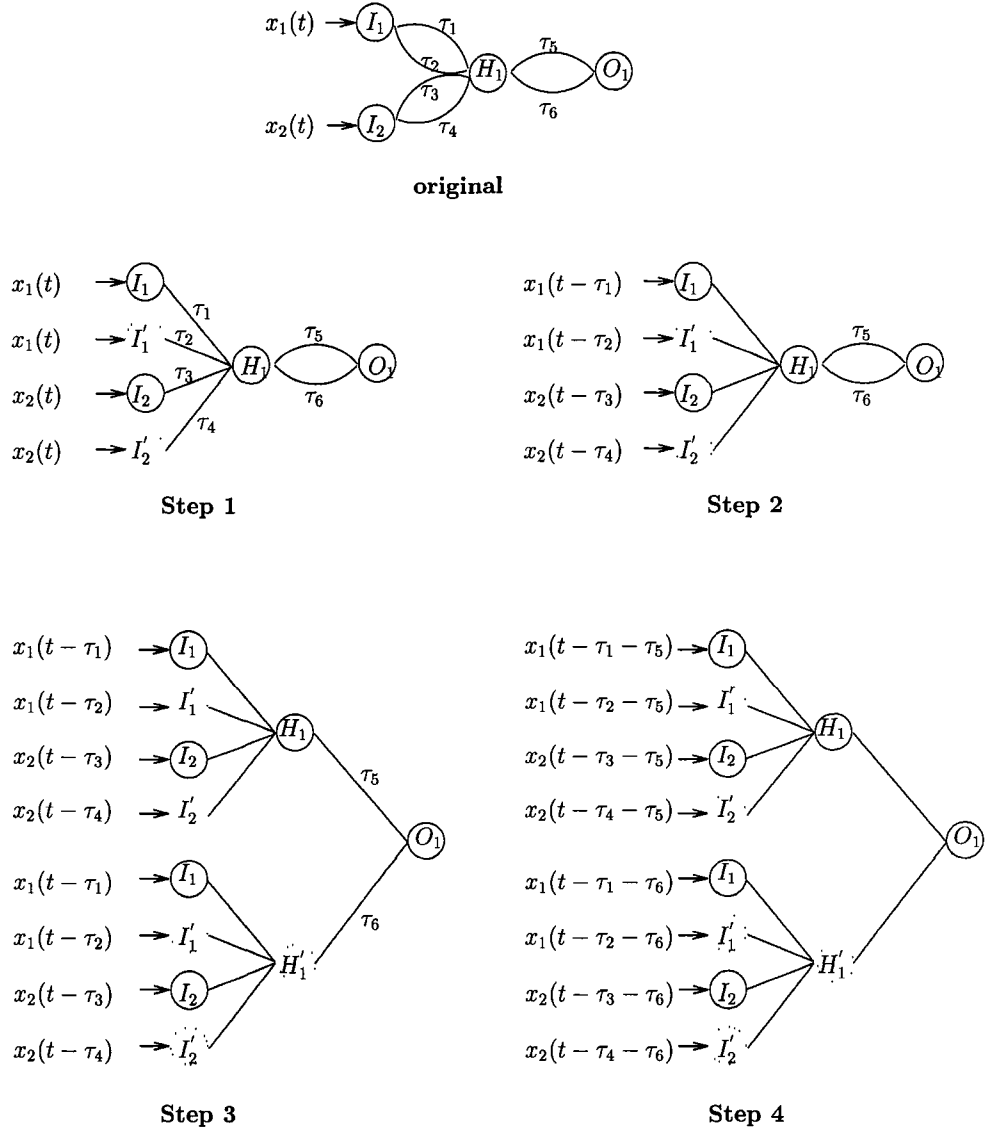


Figure 3.4: Explicit demonstration of unfold operation.

	original net		unfolded net	
# of output nodes	n_O	$O(n)$	n_O	$O(n)$
# of connections b/t each H/O node pair	d_2	$O(1)$	1	$O(1)$
# of hidden nodes	n_H	$O(n)$	$n_H d_2 n_O$	$O(n^2)$
# of connections b/t each I/H node pair	d_1	$O(1)$	1	$O(1)$
# of input nodes	n_I	$O(n)$	$n_I d_1 n_H d_2 n_O$	$O(n^3)$
# of connections to each output node	$n_H d_2$	$O(n)$	$n_H d_2$	$O(n)$
# of connections to each hidden node	$n_I d_1$	$O(n)$	$n_I d_1$	$O(n)$
total # of connections to output layer	$n_H d_2 n_O$	$O(n^2)$	$n_H d_2 n_O$	$O(n^2)$
total # of connections to hidden layer	$n_I d_1 n_H$	$O(n^2)$	$n_I d_1 n_H d_2 n_O$	$O(n^3)$

Table 3.1: Comparison of the original and unfolded network configuration.

τ_6 . In *Step 3*, the new input nodes are generated and the entire input layer is unfolded. Then finally in *Step 4*, each input time lags is readjusted.

3.7 Hidden Units and Single Layer Time-Delay Adaptation

Property 2 *The TDNN is equivalent to its unfolded feedforward network with sufficient number of hidden and input units. However, it is usually impractical to have such large number of hidden nodes and input nodes in an unfolded configuration. An example of a three layered network is shown in Table 3.1, the total number of hidden units in the unfolded network may grow up to $O(n^2)$, while the total number of input nodes may grow up to $O(n^3)$.*

Lemma 1 (single layer time-delay adaptation) *Given a TDNN with ATNN*

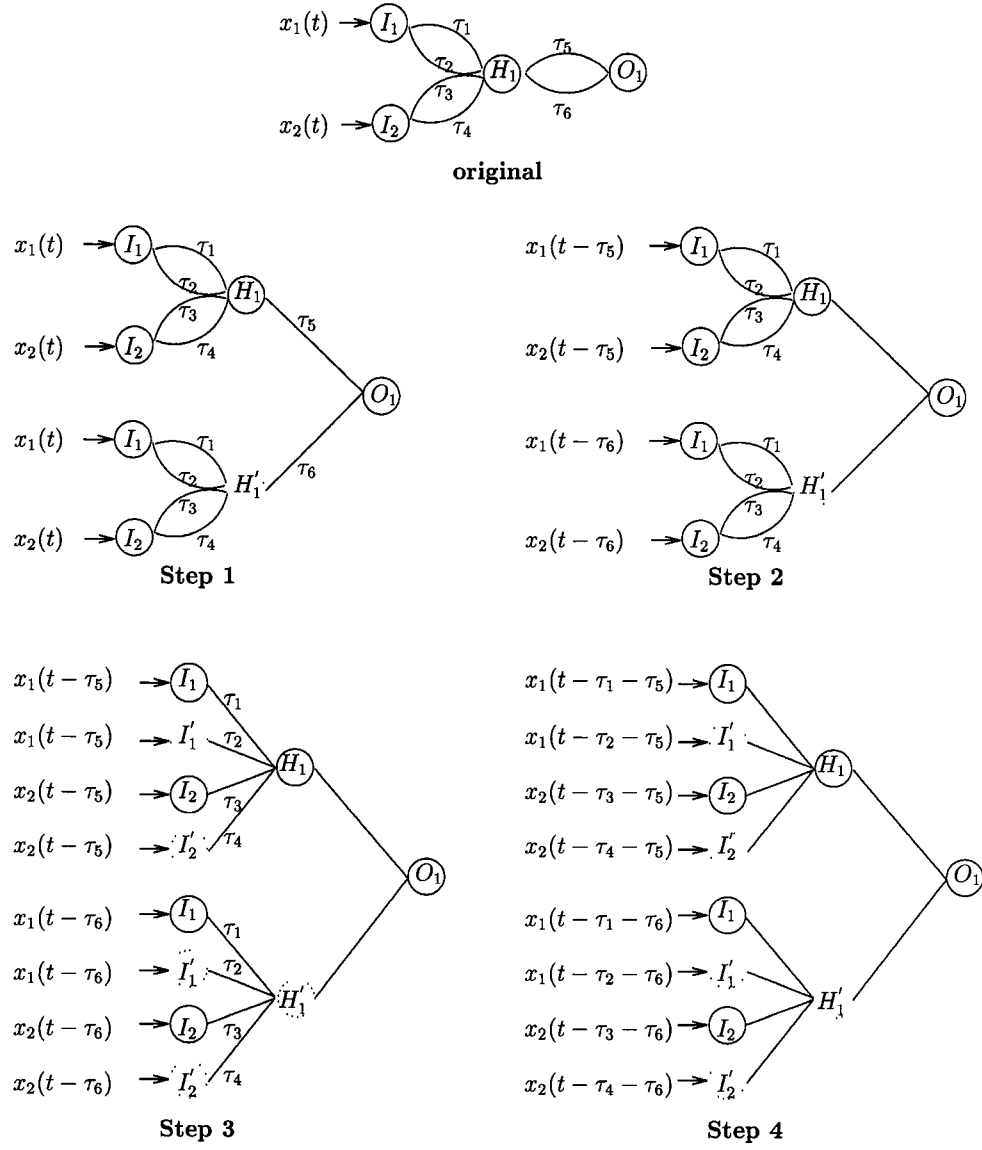


Figure 3.5: Alternative NUA by unfolding second layer time-delay first.

algorithm applied to adapt time-delays on only one layer (layer one or layer two) of connections and adapt weights on both layers. Then, after each iteration of training, the time-delay changes of the TDNN can be realized in a feedforward network with a sufficiently large number of hidden units $h2^{nd}$, where h and n are the number of hidden and input units of the ATNN respectively, d denotes the sum of the maximum time-delay variables in layer one and layer two of TDNN (i.e., $d = \max(T_1) + \max(T_2)$, where T_1 and T_2 are time-delay matrices of layer one and two respectively).

Proof: Given n input units and sufficient number of historical time step d , each hidden node is associated with $n \cdot d$ inputs (Property 1). For all $n \cdot d$ elements, there are at most 2^{nd} of combinations (denoted as set $\mathcal{A}$) in terms of whether each element is selected or unselected. Since d is sufficiently large, all the changes in any time-delay elements will fall in the range $[0, d]$.

Given an feedforward net with $h2^{nd}$ hidden units in which each hidden unit of the original ATNN is expanded to 2^{nd} units, such that each of the 2^{nd} units is associated with one of the possible combinations (in set $\mathcal{A}$) of a connection configuration to the historical time steps of all input units.

Applying NUA operation, the ATNN can be unfolded into a feedforward net with the input node taking data $a_i(t - \tau_{layer-one} - \tau_{layer-two})$ out of input node i , where $\tau_{layer-one}$ and $\tau_{layer-two}$ represent the time-delay in layer one and two respectively. For example, $\tau_{layer-one}$ is one of τ_1, τ_2, τ_3 , or τ_4 in Figure 3.6, $\tau_{layer-two}$ is τ_5 or τ_6 .

If $\tau_{layer-two}$ is fixed, changing layer one of time-delays is equivalent to making selection of a input set $a_i(t - \tau_{layer-one})$ with a fixed amount of shift $\tau_{layer-two}$. The input data set is a subset of all input data combinations set $\mathcal{A}$.

If $\tau_{layer-one}$ is fixed, changing layer two of time-delays is equivalent to shifting all input data $a_i(t - \tau_{layer-one})$ in the same time step $\tau_{layer-two}$. The shifted data set is also a subset of set $\mathcal{A}$.

Therefore, given $h2^{nd}$ hidden units will be sufficient to represent all different combinations of connection configurations to the historical record of all input nodes. With layer one or layer two of time-delay adapted, the ATNN is equivalent to an unfolded feedforward net with extra and sufficient enough number of hidden units $h2^{nd}$. $\square$

Given the fact of Theorem 1 and Corollary 1, we need to consider the optimal weights c_i and time-delays $\tau_{i,j,k}$, where the ATNN is applied. Given sufficient number of hidden units and time delays, ATNN with one hidden layer is capable of approximating any spatio-temporal function to any desired degree of accuracy. Practically, for a small number of hidden units, then we need to adapt delays in both layers in order to achieve better mapping or performance. Consistent simulation results will be discussed in Chapter 6 Section 6.3.2.

Remark 1 *Given a lower bound number of hidden units, with a proper choice of weights and time-delays, the ATNN can approximate any complex spatiotemporal function to desired accuracy. However, it can get stuck in a local minimum. The convergence is expected to improve if additional hidden units are employed [66]. This argument is also consistent with our simulation results in Chapter 4 Section 4.2.*

3.8 Relation to Other Networks

The proposed *ATNN* is indeed a generalization of the Error-Back Propagation neural network (*BP*). *TDNN* and *BP* can be viewed as special cases of *ATNN* if certain parameters of the *ATNN*'s paradigm are fixed.

Case 1: If we fixed the time-delay $\tau_{jik,h-1}$ in Equation (3.2) and applied weight learning (Equation (3.19)) without updating time-delay variables, it becomes a typical Time-Delay Neural Network.

Case 2: The Error-Backpropagation Network is also a subset case of *ATNN*. If we set $K_{ji,h-1} = 1$ and $\tau_{jik,h-1} = 0$, then Equation (3.2) will become:

$$S_{j,h}^{\mu} = \sum_{i \in \mathcal{N}_{h-1}} w_{jik,h-1} \cdot a_{i,h-1}^{\mu} \quad (3.20)$$

This equation above is exactly in the same form of Error-Backpropagation's net input rule. If we applied gradient descent learning and update weights, the *ATNN* turns out to be an Error-Backpropagation Network in this restricted case.

3.9 Conclusion

We have described the extended algorithm of time-delay adaptation which can be viewed as a generalization of the *TDNN* and the error back-propagation learning paradigm. Unlike previous studies of *TDNNs* whose time-delay values have to be predetermined and remain fixed throughout training, in our approach the time-delays as well as the weights are adjusted with a learning procedure to achieve the desired system criterion. This work provides more dynamics and

flexibility for the network itself to approach an efficient performance level and to optimize its configuration. Our results show that the time-delays and connection strengths in an *ATNN* can be optimized automatically. The learning paradigm proposed here is expected to improve performance of convergence speed and generalization significantly compared to the utilization of fixed delays in the conventional *TDNN*. A network unfolding algorithm (*NUA*) is explicitly defined to formulate the unfolding operation from *TDNN* or *ATNN* to feedforward network. This procedure provides ways to analyze the complexity of *ATNN*. Extended corollary and lemmas are proposed such that the *ATNN* inherits the properties and capabilities of feedforward network as a universal function approximator.

Chapter 4

Spatio-Temporal Pattern Processing

4.1 Definition and Scope

There has been tremendous interest in learning the trajectory and topological transformation by using neural network [96, 2, 139, 145, 76, 109]. We intend to bring up more valuable aspects from this research.

A *spatiotemporal pattern* is in general defined as a function $f(x(t))$ in a n -dimensional Euclidean space $\mathcal{R}^n$ measuring a sequence of vectors $x(t) \in \mathcal{R}^n$ from time interval $[t_0, T]$. Generally speaking, a spatiotemporal pattern is a trajectory or path in n -dimensional space, parameterized by time. Two examples we will discuss in this chapter are spatio-temporal spirals as shown in Figure 4.1 and 4.2.

Spatiotemporal signals may classify into two categories: Sequences with limit cycles, oscillation, or periodic are considered in the applications of storing, re-

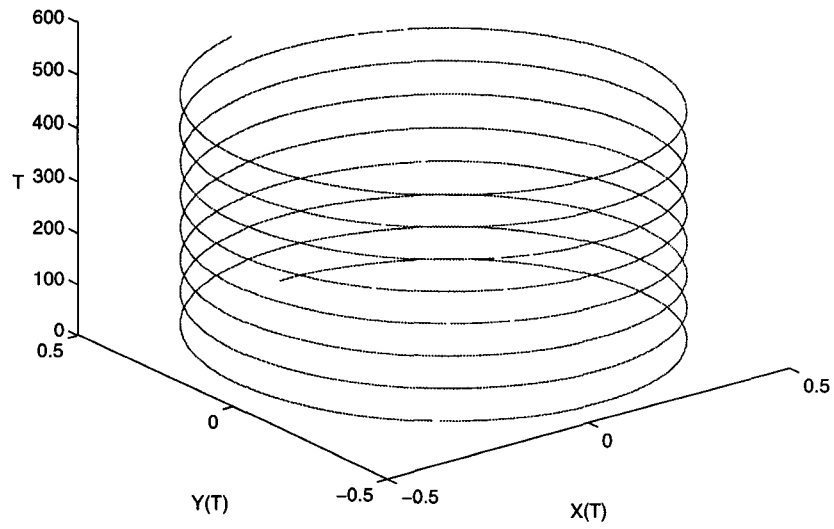


Figure 4.1: Circular trajectory on x , y , and time domain t

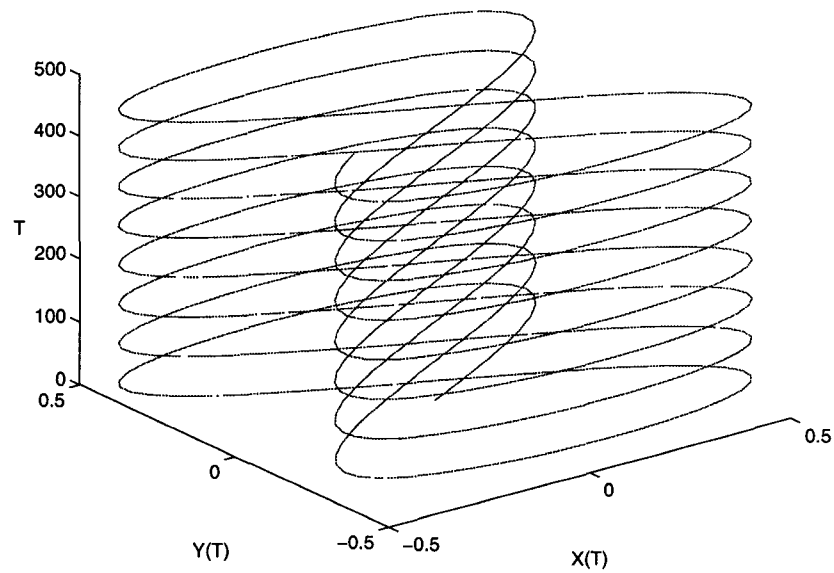


Figure 4.2: Figure eight trajectory on x , y , and time domain t

calling, generation and learning sequence of patterns [109, 110, 145, 137, 130, 60, 39, 70]. The other type of sequences is irregular, aperiodic, chaotic or with limited lengths. Applications can be found in speech recognition [149], recognition of handwritten signature based on a two-dimensional temporal recording as in [58, 68, 134] and trajectory recognition [85, 139, 137].

Generally, most sequences do not necessarily change at regular intervals [24, 64, 78]. To analyze or process these sequences, a network must possess a scheme which will resolve the range of time of pattern distribution and embed sequences into the choice of connections. In this chapter, the *ATNN* is applied to spatiotemporal pattern learning, production and completion. We examine the effects of sampling rate to *ATNN*'s learning and pattern regeneration. Perturbation tolerance and noise removal of sequence are also discussed.

4.2 Trajectory Production

In this section, we demonstrate the ability of the *ATNN* to learn the embedded transformation of given inputs to outputs in sequences. We trained an *ATNN* to follow the "circular trajectory" and the "figure eight trajectory", as in [109, 110] and [145]. The formation of these two trajectories are:

$$\text{circular trajectory: } \begin{cases} x(t) = A \sin \omega \Delta t T \\ y(t) = A \cos \omega \Delta t T \end{cases} \quad (4.1)$$

$$\text{figure eight: } \begin{cases} x(t) = A \sin \omega \Delta t T \\ y(t) = A \sin 2\omega \Delta t T \end{cases} \quad (4.2)$$

We selected $A = 0.5$, $\omega = 1$, and time step $\Delta t = 0.1$; therefore, it requires 63 time steps to complete a full circle.

4.2.1 Trajectory Topology Learning

A three layered *ATNN* was configured with two input units (taking data from $x(t)$ and $y(t)$), five hidden units and two output units. There were 4 time delays for each pair of units from input layer to hidden layer, and 6 time delays from each hidden layer unit to each output layer unit. A notation $2 \xrightarrow{4} 5 \xrightarrow{6} 2$ is used in this chapter to illustrate this configuration. The initial synapse weights were random numbers with uniform distribution between 1 and -1. Time-delays were initially set at $0, \tau, 2\tau$, and 3τ for four time-delay variables respectively on the first layer, and at $0, \tau, \dots, 5\tau$ for time-delay variables on the second layer.

At each iteration of training, the network used a segment of the figure (circle or figure eight) as the initial pattern feeding the input layer. Consecutive iterations used consecutive segments along the figure. At each new iteration, only the newest point along the segment needed to be added to the input; other points were remembered by the network. Because the segment is drawn over a period of time (as $x(t), y(t)$), and because the time-delays in the network require a certain interval of data before the network is loaded (e.g. before data is available at even the largest time-delays in the network), the required length of the segment was determined by the current values of the largest time-delayed path through the network.

The progress of learning is plotted in Figure 4.3(a)-(c). Figure 4.3(a) shows the learning of the first 500 samples, where the network starts at a position determined by its initial (random) weights and initial delays. The initial point was calculated by using a segment of arc as input and reading the point's coordinates from the output of the network. Each segment of arc that was input was taken from the original figure; the network outputs were not used as successive inputs

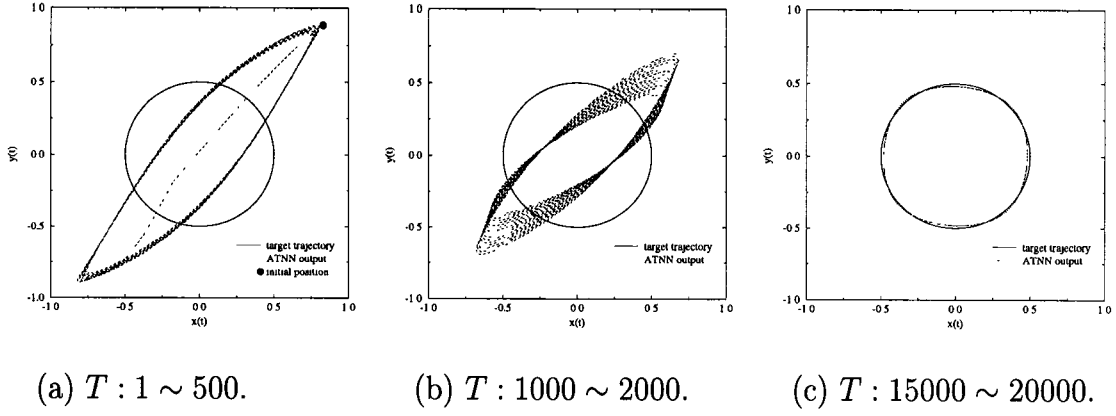


Figure 4.3: The progress of learning circular trajectory.

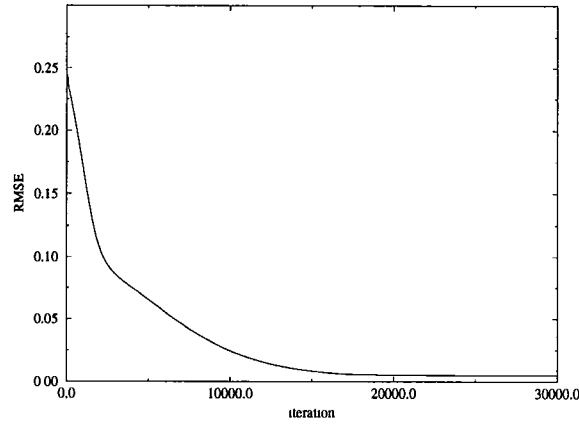
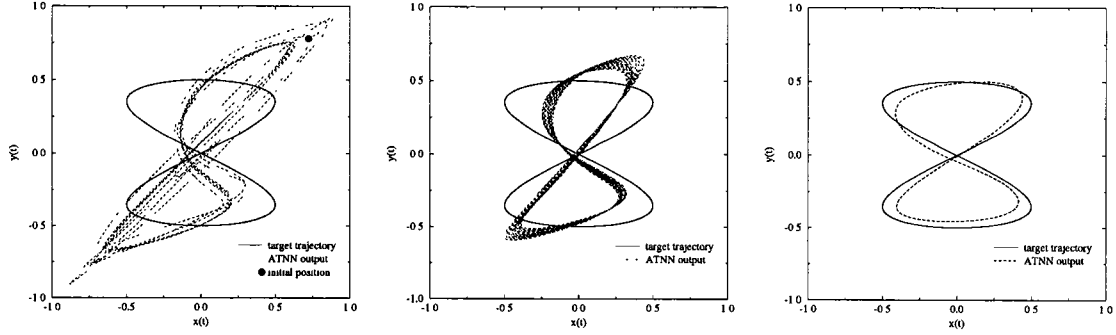


Figure 4.4: $RMSE$ along the training for circle pattern.

for these simulations in this section. The network then learned the trajectory topology (no cross-overs) very quickly. The network learned the circular trajectory with average $RMSE$ of 0.0054 after 20000 samples were presented to the network (shown in Figure 4.3(c)). The learning curve in terms of $RMSE$ is drawn in Figure 4.4. Observing from the resulting values of time delays (the maximum value of time-delays between input and hidden layer is 3, and the maximum value of those between hidden and output layer is 14), the network demands a total length of 17 points to reveal the next position. The length is about a quarter of a complete circle.



(a) $T : 1 \sim 500$.

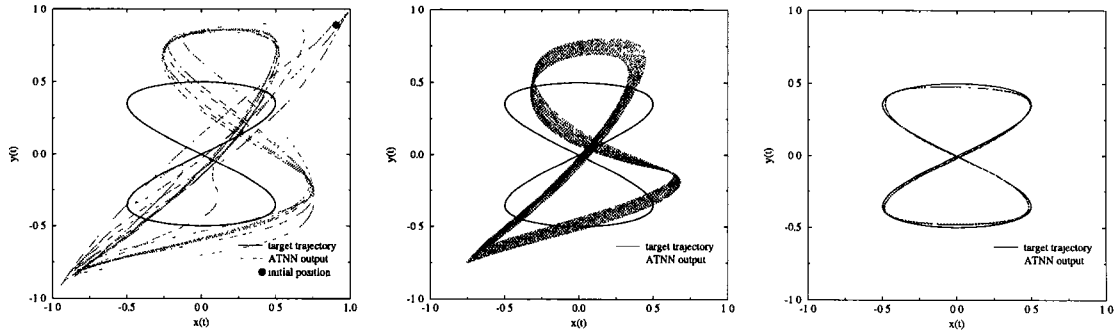
(b) $T : 1000 \sim 2000$.

(c) $T : 35000 \sim 40000$.

Figure 4.5: The progress of learning figure eight trajectory. Network configuration: $2 \xrightarrow{4} 5 \xrightarrow{6} 2$

Next, we trained the network to learn the figure eight with the same architecture ($2 \xrightarrow{4} 5 \xrightarrow{6} 2$) and training conditions as in the previous simulation. The learning process is shown in Figures 4.5(a)-(c). Apparently, learning the figure eight is a more difficult task, since the trajectory crosses itself. Nevertheless, the network learned the topology (i.e., the presence of a crossover) immediately. A trajectory similar to the figure eight appeared within the first 500 time steps of learning, and the rest of the learning time was spent to approach the exact position. The *RMSE* approaches to 0.0441 after 80000 training steps. After training, the network used information spanning 65 time steps to reveal the next position, as the maximum time-delay value in the first connection layer is 12, and the maximum time-delay value in the second connection layer is 53.

The circle and eight are two different topologies, but the network learned the general topology of each shape within the first 10% of the learning steps, then smoothed and adjusted the shapes during the remaining learning process. This network characteristic can be compared to Kohonen's self-organized feature map, which learns topology in the first 10% of iterations [81], but requires the remaining 90% to smooth its results to match the data effectively.



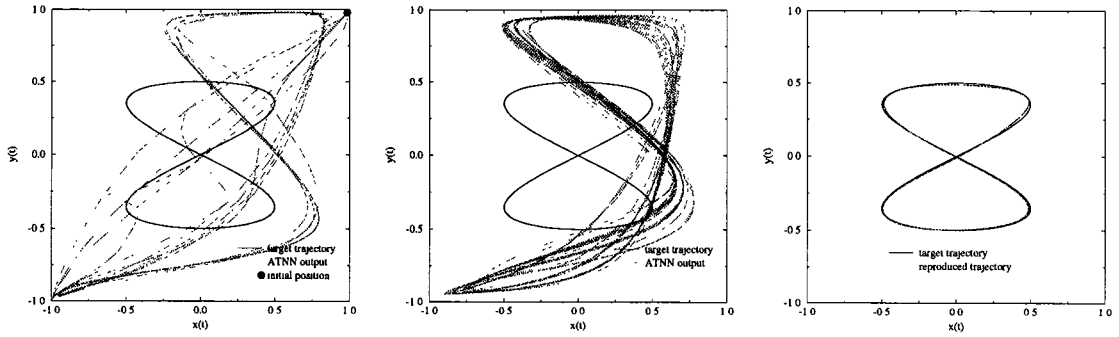
(a) $T : 1 \sim 500$.

(b) $T : 1000 \sim 2000$.

(c) $T : 35000 \sim 40000$.

Figure 4.6: The progress of learning figure eight trajectory. Network configura-

tion: $2 \xrightarrow{4} 8 \xrightarrow{6} 2$



(a) $T : 1 \sim 500$.

(b) $T : 1000 \sim 2000$.

(c) $T : 35000 \sim 40000$.

Figure 4.7: The progress of learning figure eight trajectory. Network configura-

tion: $2 \xrightarrow{4} 12 \xrightarrow{6} 2$

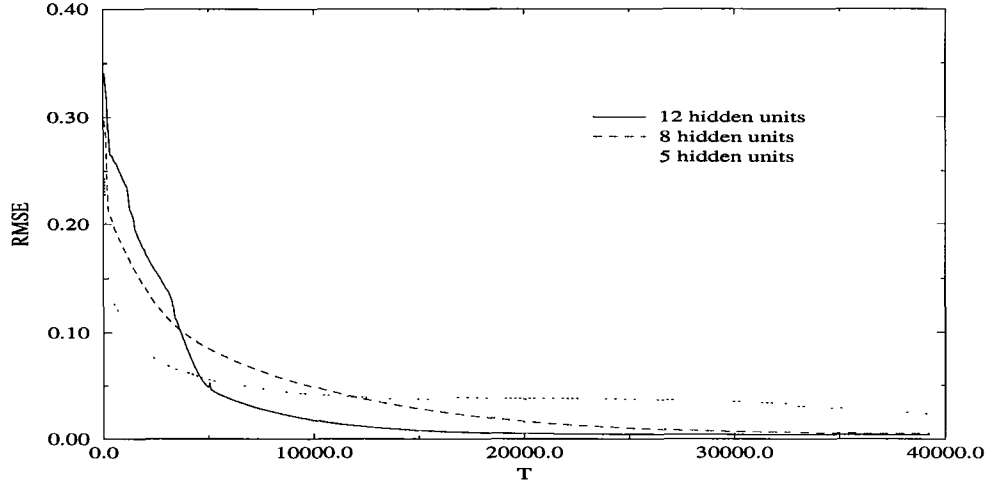


Figure 4.8: Learning curves for eight figure pattern utilizing various number of hidden units

Studies were conducted by varying the number of hidden units and observing how well the network could learn a desired shape. We selected the figure eight as a desired pattern since it is a more difficult task. The network configuration for these experiments was the same as we used previously except the number of hidden units was different (i.e., configuration $2 \xrightarrow{4} H \xrightarrow{6} 2$, where $H \in \{5, 8, 12\}$). The learning progress of each network is shown in Figure 4.5 to Figure 4.7. For comparison, the same column of each figure is arranged to show the learning process in the same place. For example, Figure 4.5(a), Figure 4.6(a) and Figure 4.7(a) are illustrating the progress of training during time step 1 to 500. As we can see from these pictures, the networks can approach to the desired shape but with different speed and different learning paths. The learning curve in terms of average *RMSE* is shown in Figure 4.8. The learning curve of using five hidden units is plotted with a dotted line. Learning curves for eight hidden units and twelve hidden units are plotted with dashed line and solid line respectively. Apparently, a network employing 12 hidden units can converge

faster, the *RMSE* reduces to 0.025 at time step 10000. Using 5 hidden units, the network learning was slower. Using more hidden units, the network has more dimensions to travel along its energy terrain.

4.2.2 Neuron Dynamics

We examined the dynamics of learning by observing the activity of hidden and output units during the learning phase. The hidden neurons responded for particular orientations of stimulus. The corresponding activities of hidden units ($h_1, h_2, \dots, h_5$) and output units (O1 and O2) are shown in Figures 4.9 to 4.10. As we can see from these pictures, the hidden units react stronger and stronger gradually. Each unit is an oscillator and stops growing to attain a stable oscillation. During the first 1000 training samples, the output unit 2 responds earlier than unit 1 and most hidden units react equally except unit 4. From the 3000th to 5000th samples, O1 becomes more active than O2, and h_1 and h_3 react significantly. The oscillation of h_1 and h_3 gradually become stable when O1 develops to a stable state. O2 becomes stable after 18000 samples, as the amplitude of h_5 is getting larger, while h_2 and h_4 remain in the same small range after O1 is stable. We can conclude that h_1 and h_3 are the main units that contribute to the revolution of O1, and h_5 leads the development of O2. Figure 4.12 shows hidden and output unit activity for the *ATNN* trained to produce the figure eight.

We can observe from Figure 4.11 and 4.12 that each hidden unit reacts with selective input orientation, similar to orientation-specific neurons found in a biological system [17]. Combinations of orientation-selective units determine the exact direction of motion in a linear summation model [47]. Back-propagation



Figure 4.9: Hidden neurons activities during training time steps 1 to 1000. The dotted line under O1 and O2 is the target trajectory

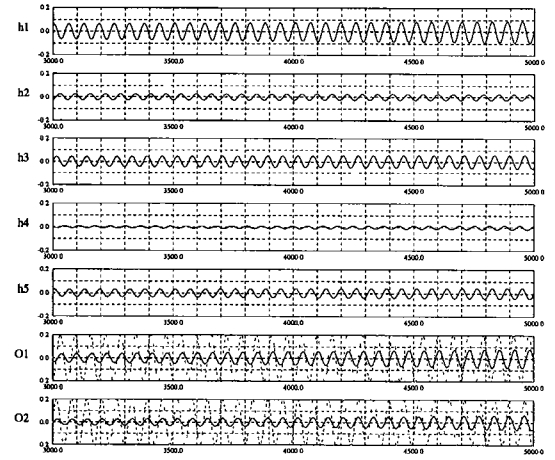


Figure 4.10: Hidden neurons activity during training time steps 3000 to 5000. The dotted line under O1 and O2 is the target trajectory

networks trained on robotic trajectories have also been observed to develop orientation-specific units in their hidden layers [73]. These networks are internally static in contrast to the *ATNN* which is internally dynamic in its temporal structure. This is similar to observations from biological systems, in which individual neurons are orientation-selective and contribute to a motor system code in which the activity of orientation-selective movements correspond to the direction of limb motion primates.

4.3 Pattern Completion and Recognition

One of the important problems of learning sequences is **sequence reproduction**. The network should be able to produce the rest of trained sequences itself when it is provided only a subset of the sequence. The sequence is usually in

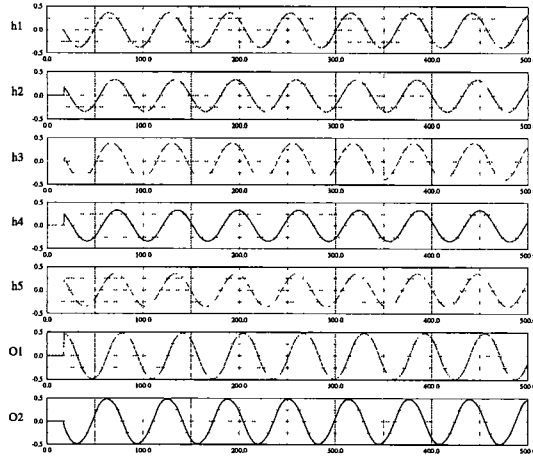


Figure 4.11: Hidden neurons activities after trained with circular trajectory: each unit responds to different input orientations.

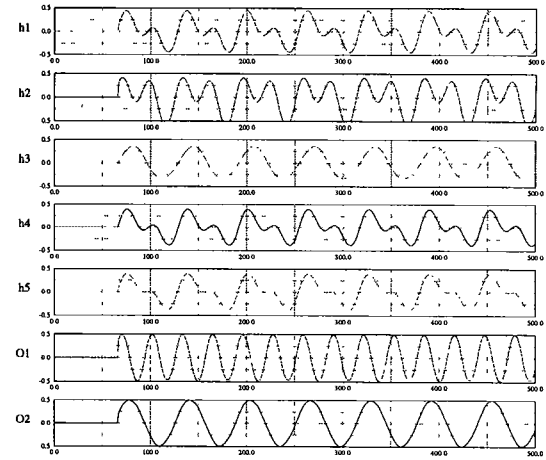


Figure 4.12: Hidden neurons activities after trained with figure eight trajectory: each unit responds to different input orientations.

a limit cycle [63]. The completion task may be generalized to finding the missing information given a partial description of a situation [129].

We tested the network's ability to generate figures given its own outputs as input signals. Given any required segment of the original circle starting at any point along the circular trajectory, the network generated the next new position, then completed the remaining trajectory recursively by using the new segment generated by the network.

From the experiments, the reproduced trajectories of the circle are plotted with dotted lines in Figure 4.13 (a)-(c). The average *RMSE* values of these three examples are 0.0113, 0.0162 and 0.0114, which are impressively low. In these runs, the network was initially given only a segment of the original circle (length of 17 in this case), but after the initial segment, the network utilized its own past trajectory as input and then completed the whole circle. The network

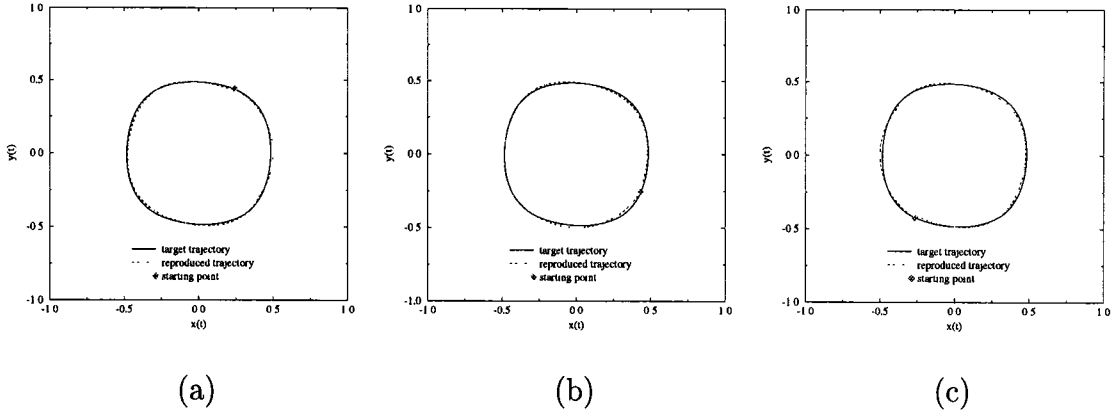


Figure 4.13: The results of circle reproduced by *ATNN* start from different initial position (a) $RMSE = 0.0113$ (b) $RMSE = 0.0162$ (c) $RMSE = 0.0114$

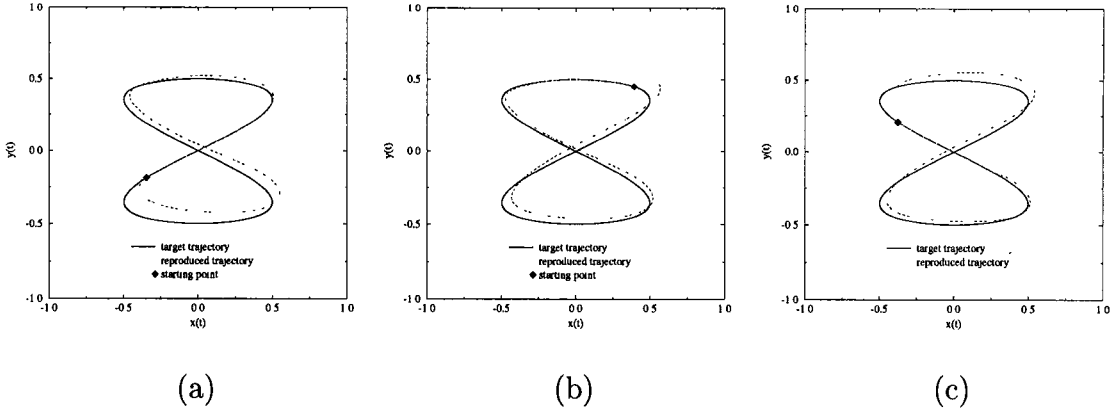


Figure 4.14: The results of figure eight reproduced by *ATNN* start from different initial position (a) $RMSE = 0.0507$ (b) $RMSE = 0.0508$ (c) $RMSE = 0.0474$.

was tested with different starting position along the circle, and completed the figure no matter where the starting position was. These results demonstrate the capability of the network to perform predictive recognition and figure completion when only partial information of the original data is available.

Similar experiments were done for the figure eight, as shown in Figure 4.14 (a)-(c). While the performance for the figure eight reproduction is less ($RMSE$ about 0.05), the accuracy improves when the network learns for an adequate time. It appears that the more difficult the target trajectory is, the more error

is encountered during signal production.

4.4 Spatio-Temporal Feature Recognition

One of the main differences between static pattern recognition and spatio-temporal recognition is that the number of static patterns is usually fixed, such as 26 letters for character recognition or image patterns, while there may be an infinite number of training patterns in spatio-temporal pattern recognition and classification. The main task in spatio-temporal signal and pattern recognition is to learn the real properties of the given training signal and using these learned features to fulfill the recognition job. Furthermore, the learning could be speeded up since it is not necessary to feed the entire pattern to train a successful recognizer if the network can learn the essential features of the pattern. In this section, we test the use of the general features of the figure as opposed to memorization point-by-point. We then verify the capability of the *ATNN* in accomplishing spatio-temporal signal recognition and production by training on part of the pattern and testing on the unseen parts of the pattern (generalization).

4.4.1 It is not memorizing each point

Several experiments were performed to verify that the network is learning the feature of a circle rather than memorizing each point. The *ATNN* was trained on different partial trajectories (e.g. $\frac{3}{4}$ circle, $\frac{1}{2}$ circle and $\frac{1}{4}$ circle) and tested on the other part of the circle where most of the points were unseen during the corresponding training scenario. The *RMSE* value during the learning phase in each case is presented in Figure 4.15. As we can observe from this figure, all cases

can achieve similar *RMSE* value as low as that trained on the full circle except for the training on $\frac{1}{4}$ circle, which can no longer improve its performance after 200 training iterations. The learning curve and testing curve during training procedure are shown in Figures 4.16 to 4.18 correspondingly. The performance of *ATNN* trained from $\frac{1}{4}$ circle is no better than that of any other cases.

Examining the time-delay variable matrices (see Appendix B), the changes always occur at about the same elements, which reflect those are important feature extractors. A summary of the time-delay variables is shown in Table 4.1 in which T_{atnn} denotes the maximum total delay through the entire trained network. As we can see from Table 4.1, T_{atnn} is equal to 17 in the cases where the network was trained on a full and $\frac{3}{4}$ circle. It was 11 for the half circle and only 8 for the quarter circle. It appears that the time-delay variables did not have a chance to adapt to capture the circular feature in the case of the $\frac{1}{4}$ circle. The network in that training scenario was the worst one and was stuck with $RMSE = 0.05$.

In summary, the network can figure out the rest of the trajectory as long as a large enough number of points (e.g. a long enough segment) is in the training set. The longer arcs in training achieved the better performance. Half circle training is sufficient to achieve reasonable accuracy. A quarter circle is not enough to provide good knowledge for the network; it can never get better performance even if training time is extended without limit.

To demonstrate the learning progress and network performance, Figure 4.19 to Figure 4.22 show the learning and testing performance when three quarters of the circle is trained. In each figure, part (a) illustrate the learning phase and part (b) is the performance on testing of the resulting network on the remainder of the

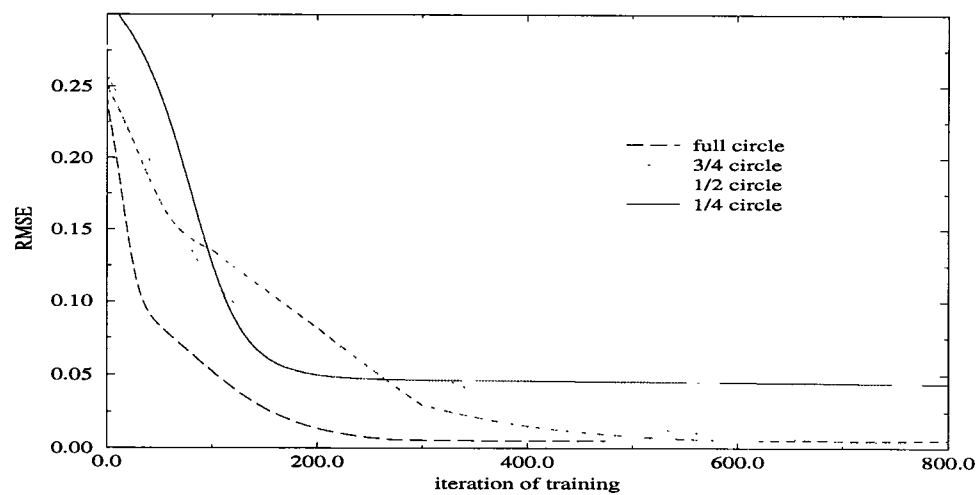


Figure 4.15: *RMSE* of various training trajectories during training phase.

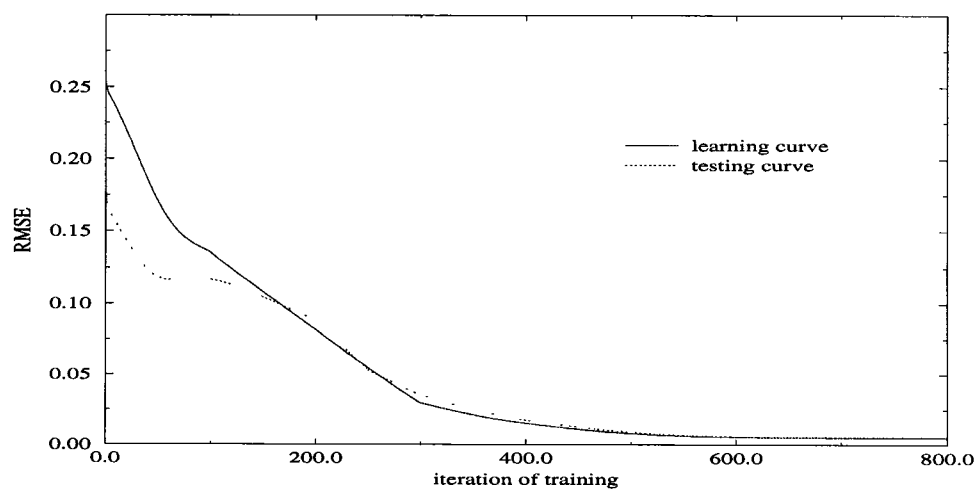


Figure 4.16: *RMSE* of learning curve and testing curve when trained on 3/4 circle and tested on half circle.

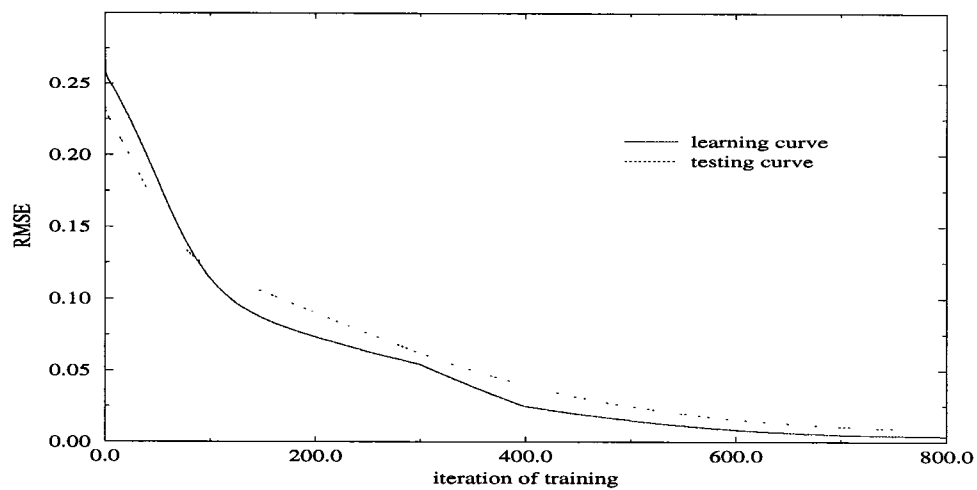


Figure 4.17: *RMSE* of learning curve and testing curve when trained on $1/2$ circle and tested on the other half circle.

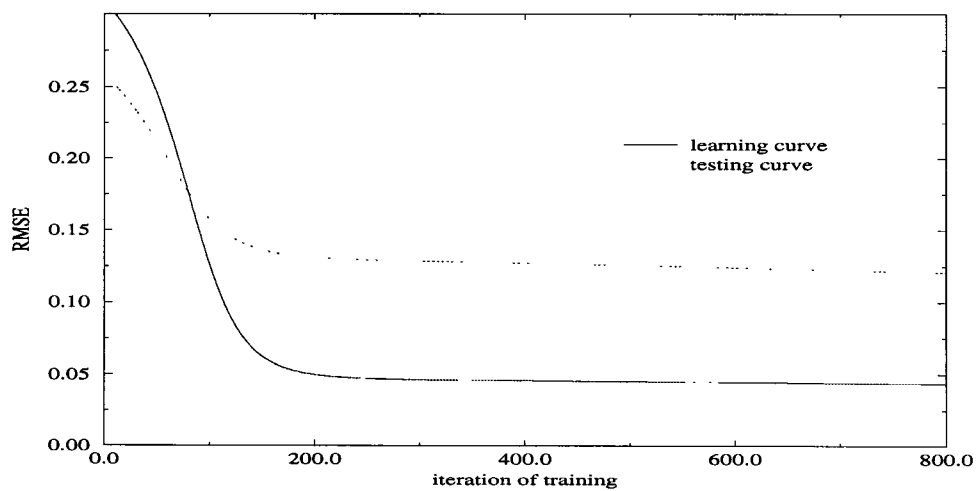


Figure 4.18: *RMSE* of learning curve and testing curve when trained on $1/4$ circle and tested on the half circle.

	portion of circle trained			
	full	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
$max(T_1)$	3	3	3	3
$max(T_2)$	14	14	8	5
T_{atnn}	17	17	11	8

Table 4.1: The resulting time-delay variables after training is complete, where $max(T_1)$ and $max(T_2)$ denote the maxima element of time-delay matrix on layer one and two respectively, and T_{atnn} is the total delay of trained network where $T_{atnn} = max(T_1) + max(T_2)$.

circle. Figure 4.19 shows results after 5 iterations of training, and Figure 4.22 shows results at 600 iterations, when training is completed. As we can see from these figures, the network trained on $\frac{3}{4}$ circle can almost learn the shape of a circle at iteration 200. At iteration 600, the network has learned the pattern, and the output of the *ATNN* (plotted in dashed line) overlaps with the solid target line in Figure 4.22. The results of a network trained on $\frac{1}{2}$ circle perform analogy with above work (see Figure 4.23 to Figure 4.26).

Similarly, the results of a network trained on the quarter circle are shown in Figures 4.27 to 4.30. Comparing Figure 4.26 and 4.30, the *ATNN* output and target values are very close in Figure 4.26, while in Figure 4.30, the output is still far away from the target line. The network trained on only a quarter circle can not accomplish the recognition. It has difficulty learning the pattern since the length of the training pattern is not sufficient.

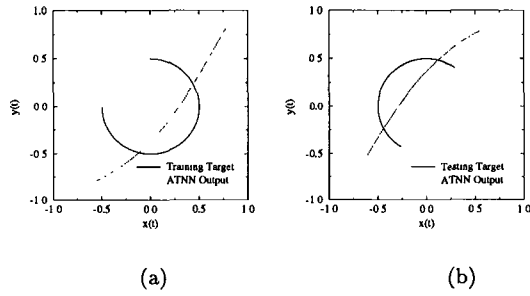


Figure 4.19: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 3/4 circle.

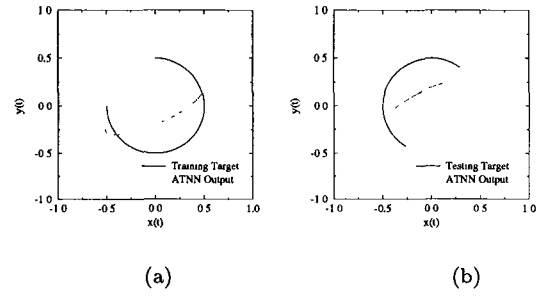


Figure 4.20: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 3/4 circle.

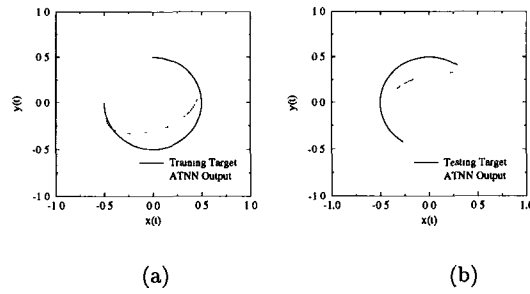


Figure 4.21: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 3/4 circle.

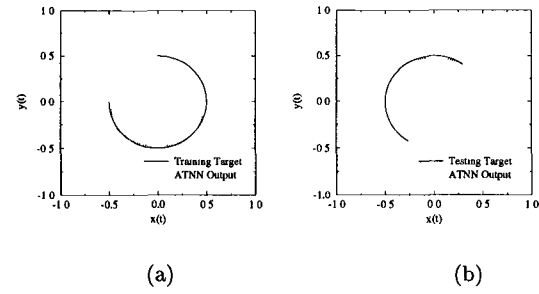


Figure 4.22: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 3/4 circle.

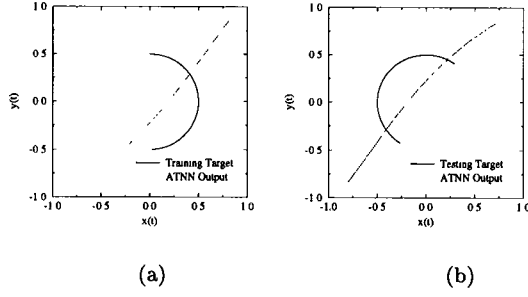


Figure 4.23: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 1/2 circle.

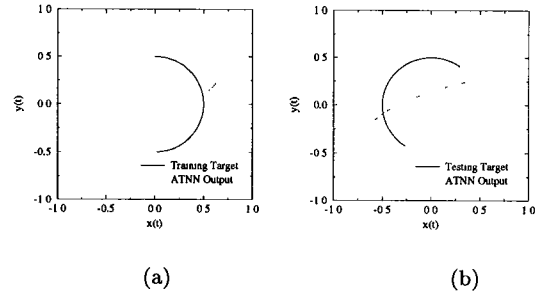


Figure 4.24: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 1/2 circle.

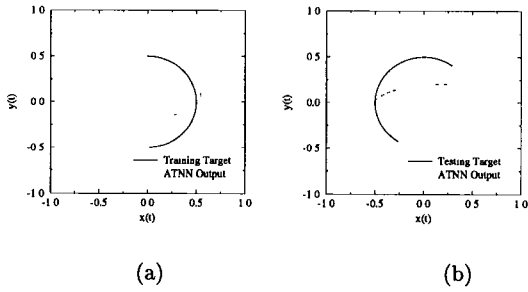


Figure 4.25: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 1/2 circle.

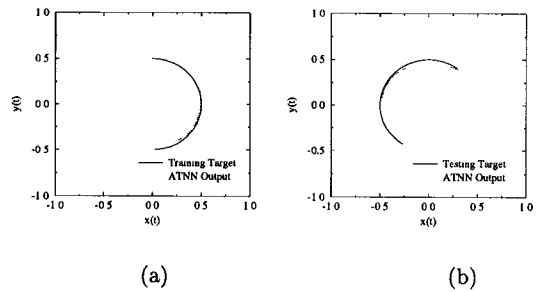
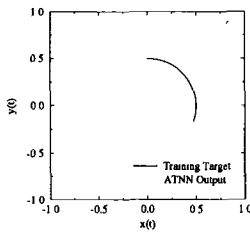
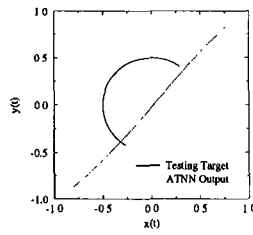


Figure 4.26: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 1/2 circle.

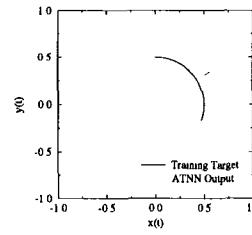


(a)

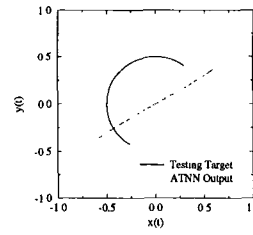


(b)

Figure 4.27: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 1/4 circle.

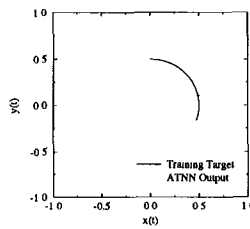


(a)

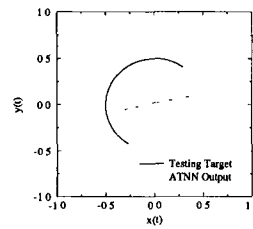


(b)

Figure 4.28: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 1/4 circle.

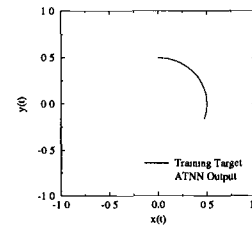


(a)

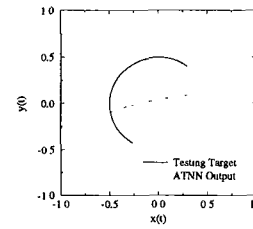


(b)

Figure 4.29: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 1/4 circle.



(a)



(b)

Figure 4.30: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 1/4 circle.

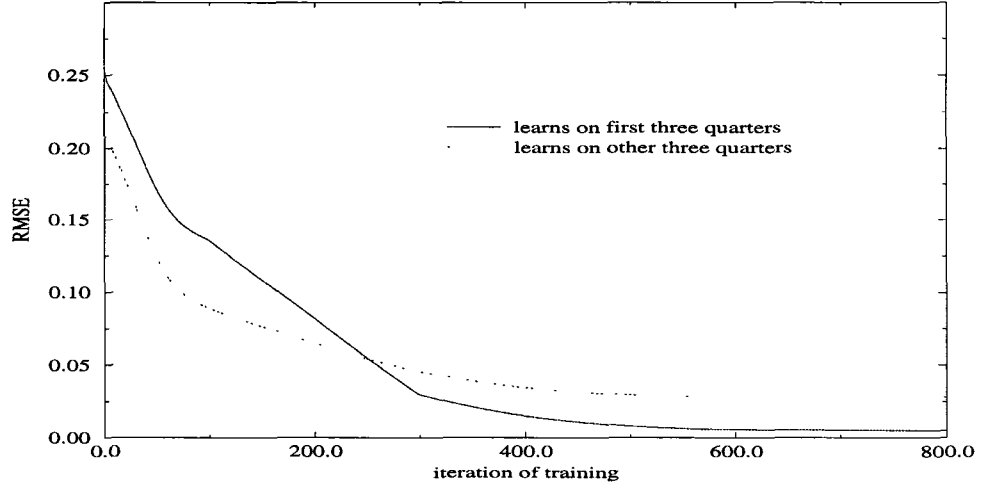


Figure 4.31: Learning *RMSE* curves of various training trajectories arcs with the same length ($\frac{3}{4}$ circle) but with different location.

4.4.2 Training Position Invariance

The same length of training arcs with different locations were used as training patterns to test if the network still possessed good recognition when the location is changed in each training scenario. In the cases of a training pattern with full, $\frac{3}{4}$ and $\frac{1}{2}$ circle, when arcs with different locations were selected, the network could resolve the complete pattern independent of the portion and position of the training arc.

In the case of training pattern with $\frac{3}{4}$ circle, arcs with different location were selected. Two learning curves are plotted in Figure 4.31 in which two curves reduce to the relative small *RMSE* value and complete the learning. The learning curve and test curve are shown in Figure 4.32 and 4.33.

To show the performance of network capability of performing remainder pattern retrieval regardless of the portion of training sequence with $\frac{3}{4}$ circle, the *ATNN* learning phase and test phase are drawn in Figure 4.34 to 4.37 in the same form as shown in previous section (e.g. Figure 4.19 to Figure 4.22). Sim-

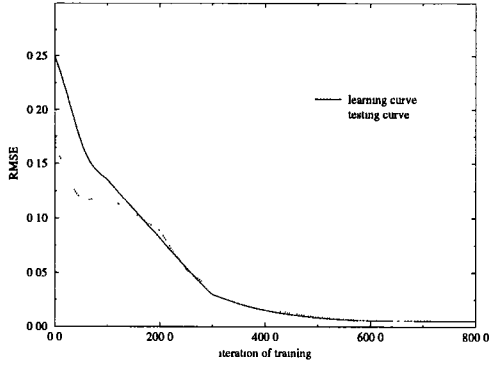


Figure 4.32: *RMSE* of learning curve and testing curve when trained on first $3/4$ circle and tested on the other $3/4$ circle.

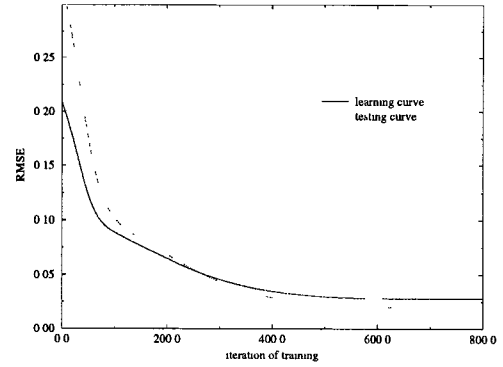
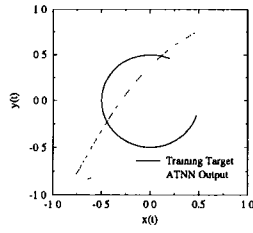


Figure 4.33: *RMSE* of learning curve and testing curve when trained on second $3/4$ circle and tested on the other $3/4$ circle.

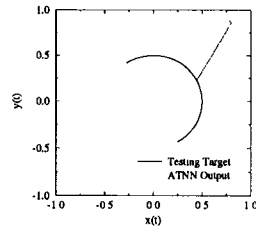
ilarly, the experiments on training employing different locations of $\frac{1}{2}$ circle are also performed, and the results are shown in Figure 4.38 to Figure 4.44. All results prove the network maintain good recognition independent of the position of training pattern. In other words, *ATNN* is indeed capable of learning the features of spatio-temporal pattern and signals instead of memorizing the static position of each point at its particular location.

4.4.3 Pattern Completion

Given various initial arcs with different positions, the network could recursively use its output and reproduce the whole pattern with low *RMSE* when only a partial circle was trained. Reproduced patterns with different starting points were obtained close to the target trajectory, similar to that in Figure 4.13. Reproduced patterns with different starting points of *ATNN* trained from $\frac{3}{4}$ circle are plotted in Figure 4.45(a) to (c) in which solid line represents the ideal circle

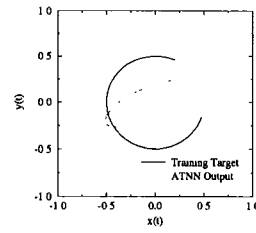


(a)

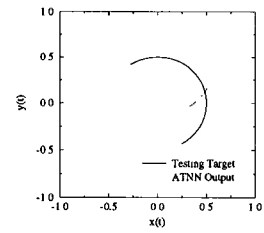


(b)

Figure 4.34: (a) *ATNN* learning phase and (b) testing on different 3/4 part of the circle, when trained 5 times on 3/4 circle.

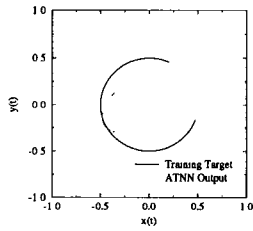


(a)

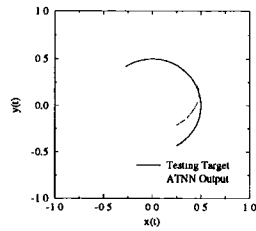


(b)

Figure 4.35: (a) *ATNN* learning phase and (b) testing on different 3/4 part of the circle, when trained 100 times on 3/4 circle.

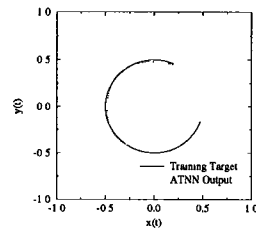


(a)

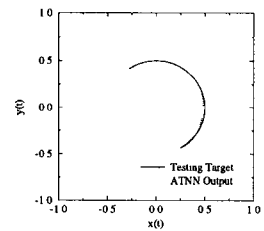


(b)

Figure 4.36: (a) *ATNN* learning phase and (b) testing on different 3/4 part of the circle, when trained 200 times on 3/4 circle.



(a)



(b)

Figure 4.37: (a) *ATNN* learning phase and (b) testing on different 3/4 part of the circle, when trained 600 times on 3/4 circle.

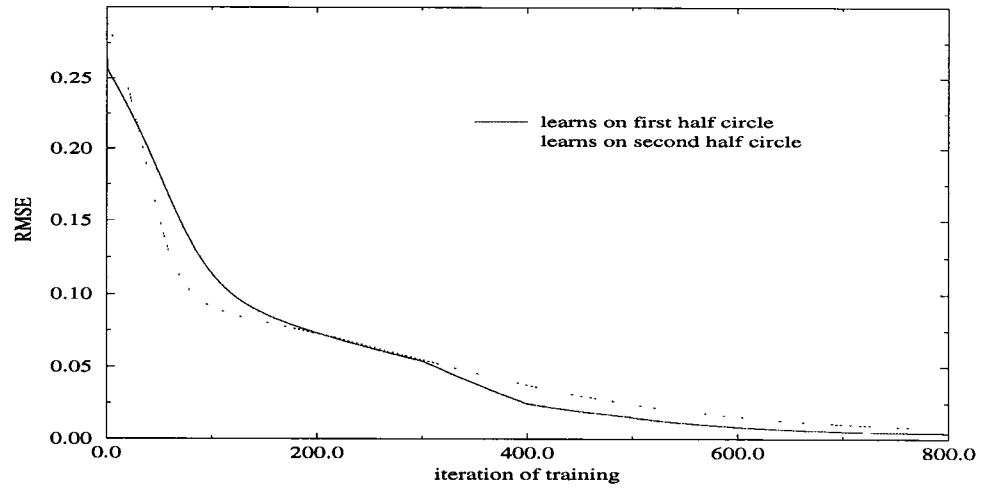


Figure 4.38: Learning $RMSE$ curves of various training trajectories arcs with the same length (half circle) but with different location.

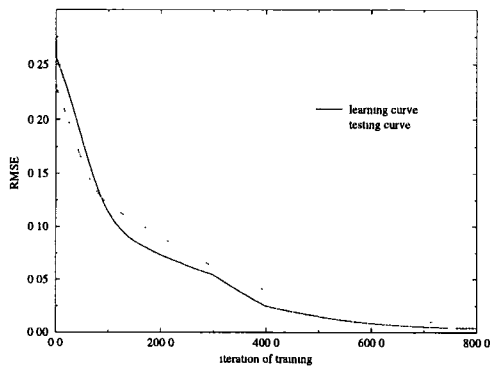


Figure 4.39: $RMSE$ of learning curve and testing curve when trained on first $1/2$ circle and tested on the other half circle.

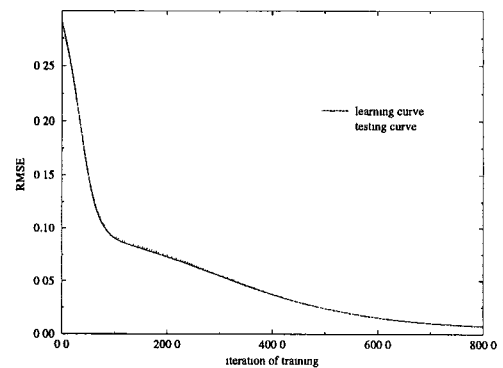


Figure 4.40: $RMSE$ of learning curve and testing curve when trained on second $1/2$ circle and tested on the other half circle.

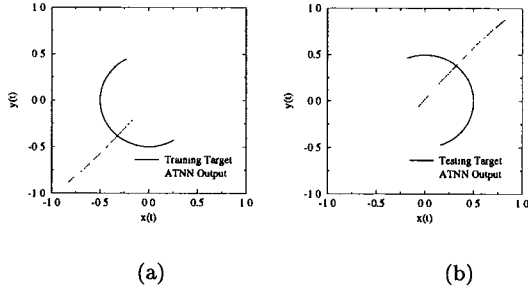


Figure 4.41: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 5 times on 1/2 circle.

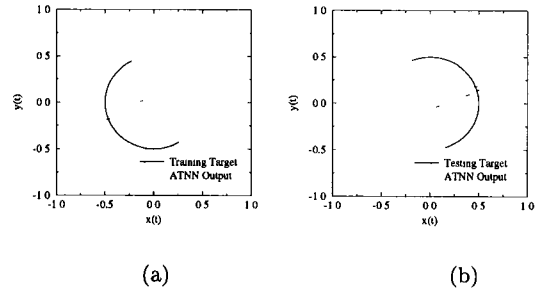


Figure 4.42: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 100 times on 1/2 circle.

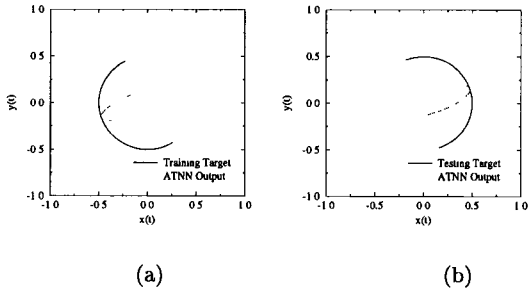


Figure 4.43: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 200 times on 1/2 circle.

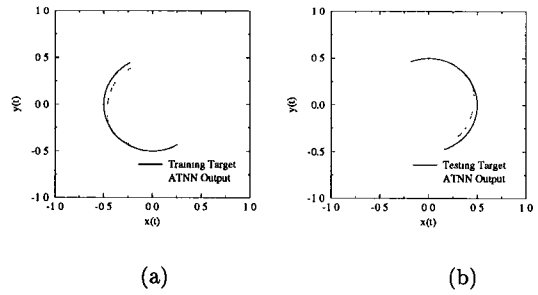


Figure 4.44: (a) *ATNN* learning phase and (b) testing on different 1/2 part of the circle, when trained 600 times on 1/2 circle.

portion trained	different starting position			average
	(a)	(b)	(c)	
full	0.0113	0.0162	0.0114	0.0130
$\frac{3}{4}$	0.0165	0.0180	0.0170	0.0172
$\frac{1}{2}$	0.0336	0.0460	0.0439	0.0412

Table 4.2: *RMSE* of reproduced figure under different situation and starting position.

and the dashed lines denotes the new produced track starting from a position depicted in grey dot. The reproduced circle of network trained from half circle are shown in Figure 4.46(a) to (c).

The performance becomes less accurate (shown in Figure 4.46) when a smaller portion (e.g. $\frac{1}{2}$ circle) is trained. The comparison of *RMSE* under different situations is shown in Table 4.2. As we can observe from this table, the average *RMSE* value of the reproduced pattern of the network trained on the full circle is about 0.0130, while the *RMSE* of the network trained from the half circle is around 0.0412.

4.5 Perturbation Resilience and Noise Removal

An important property of artificial neural networks is noise tolerance. In other words, one of the main performance measures of a network is its ability to respond properly to new data that is similar to training patterns but contains noise. This reflects how well the network generalizes to the target patterns and generates temporal sequences.

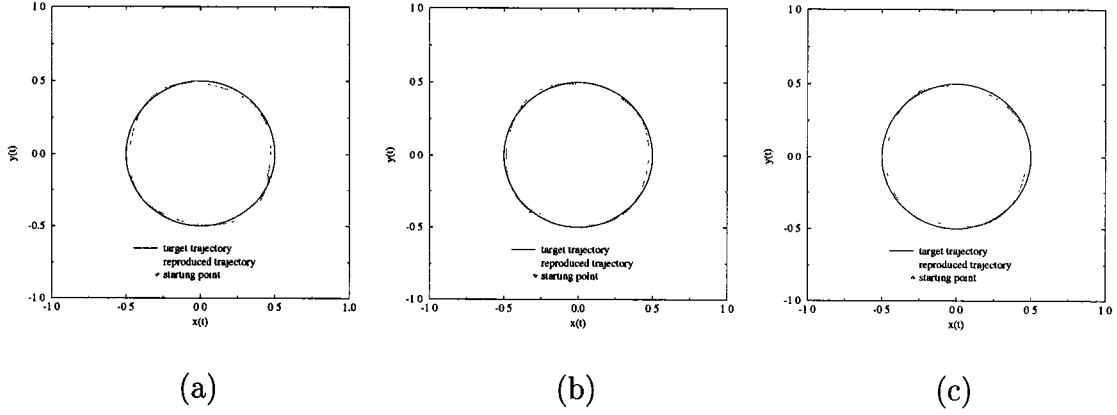


Figure 4.45: The results of circle reproduced by *ATNN* (trained from 3/4 circle) start from different initial position (a) $RMSE = 0.0165$ (b) $RMSE = 0.0180$ (c) $RMSE = 0.0170$

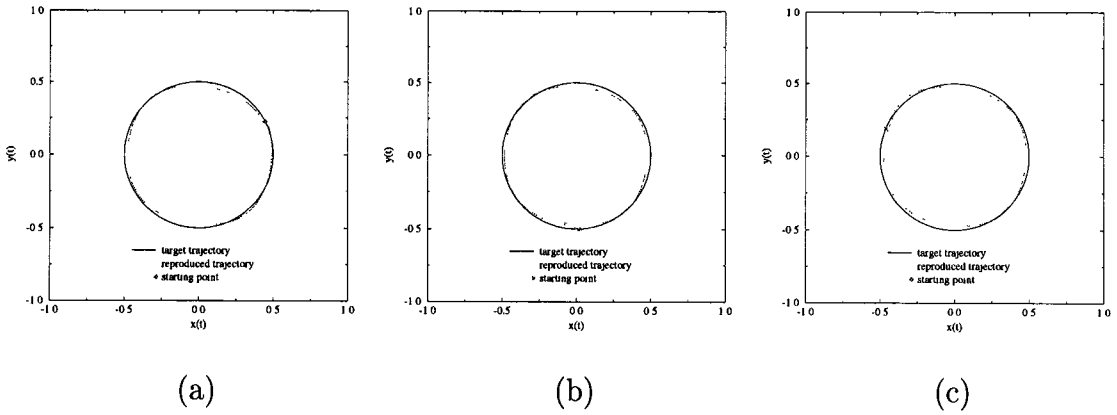


Figure 4.46: The results of circle reproduced by *ATNN* (trained from 1/2 circle) start from different initial position (a) $RMSE = 0.0336$ (b) $RMSE = 0.0460$ (c) $RMSE = 0.0439$

4.5.1 Noise Tolerance

Experiments were done by corrupting the original signal with additive noise to investigate how well the proposed system is resilient to noise. For example we perturbed the circular trajectory (Equation 4.1), such that the test signal becomes:

$$\begin{cases} x(t) = A \sin \omega \Delta t T + n(\sigma^2, \mu) \\ y(t) = A \cos \omega \Delta t T + n(\sigma^2, \mu) \end{cases} \quad (4.3)$$

where σ^2 and μ denote variance and mean of additive noise respectively.

Different variance of random noise with normal distribution and zero mean was added to the testing signal and tested on the *ATNN*. Generally speaking, the systems maintains 99.4% correction (*RMSE* less than 0.06) when the noise variance is up to 0.1. As shown in Figures 4.47, system performance decreased as noise variance was increased. The noise tolerance of the system with different parts of training pattern is also shown in Figure 4.47. The performance of the noise resilience of trained *ATNN* from full circle and $\frac{3}{4}$ circle are very similar so that those two lines are almost overlap, while the noise tolerance of trained network from $\frac{1}{2}$ circle is worse, which is plotted in the dashed line in Figure 4.47.

4.5.2 Noise Removal of Spatiotemporal Signal

The ability of the *ATNN* to remove noise from a time-series sequence was tested. The goal was to see whether a noisy initial segment could result in noise-free trajectory completion. A network trained without recurrent inputs was presented with a noisy arc as its initial input to generate the whole circular trajectory. Differing amounts of normally distributed noise (mean $\mu = 0$, variance $\sigma_{noise}^2 : 0.0001 \sim 0.5$) were used, and the network attempted to complete the rest of

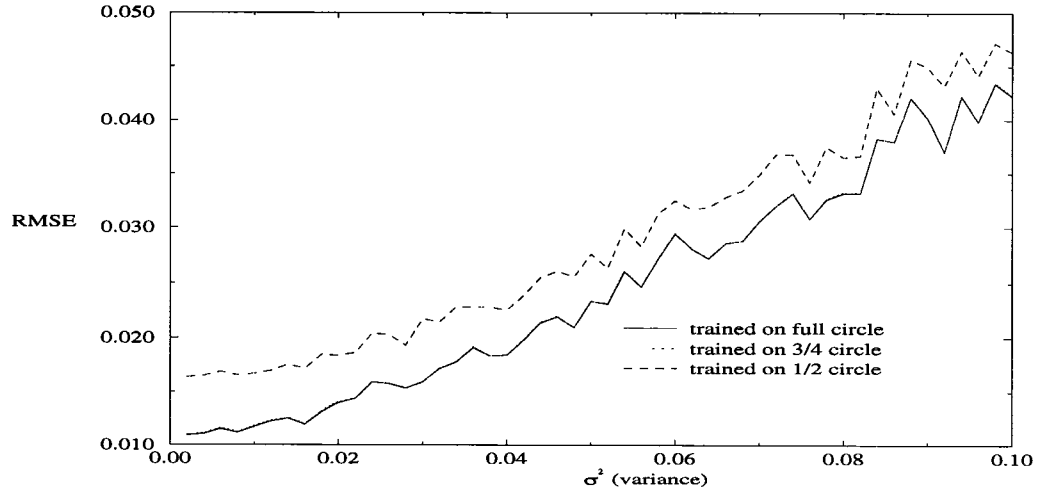
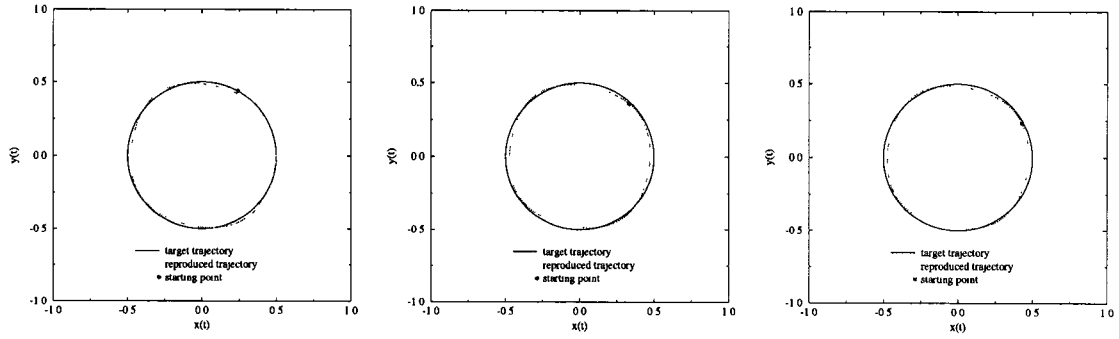


Figure 4.47: Noise tolerance of various trained network. The full circle and 3/4 circle traces are nearly identical.

the circle. Each output of the network was in turn used as its next input, to determine the subsequent point along the network's path.

The tests showed that the network could generate and complete the whole trajectory by using only a small portion of noisy initial data. Figure 4.50 shows such a case. An initial segment of length 17 was input to the network, consisting of an arc of the circle with normally distributed noise (variance of 0.1) added to each coordinate value. The initial segment contained crossing lines and departed from the arc significantly (maximum departure 0.25). The segment crossed the arc in addition to crossing itself. Figure 4.50 (a) shows the 1st circle generated, with the noisy initial arc. The 2nd circle generated, in Figure 4.50 (b), matched almost exactly the original circle, as did the 7th, shown in Figure 4.50 (c). Figure 4.51 shows a case where increased noise was added to the initial segment, although the length of the initial segment was the same as in Figure 4.50. A variance of 0.3 noise was added to each point in the initial segment in Figure 4.51, which is 3 times the magnitude of noise used in Figure 4.50. The initial segment

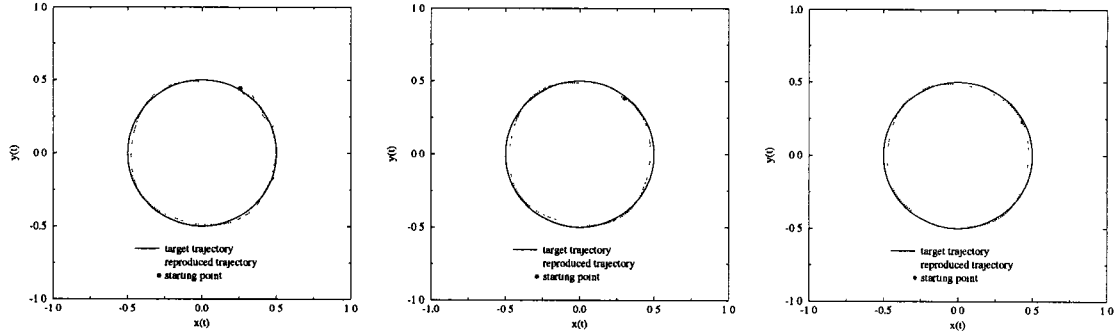


(a) 1st circle regeneration, $RMSE = 0.0120$ (b) 3rd circle regeneration, $RMSE = 0.0159$ (c) 7th circle regeneration, $RMSE = 0.0160$

Figure 4.48: *ATNN* removes the noise (variance $\sigma_{noise}^2 = 0.001$) from a time-series sequence and complete the subsequent path.

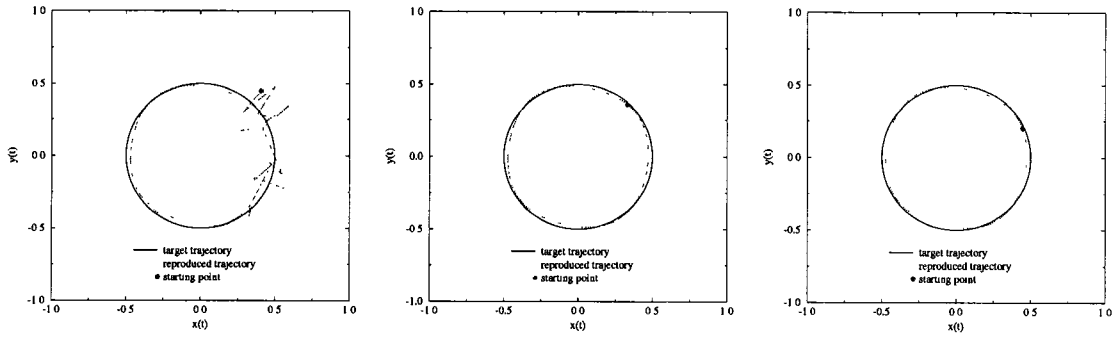
departed from the circle farther (maximum departure of 0.51), and crossed itself as well as crossing the circle. The first circle generated is shown in Figure 4.51 (a), with the noisy initial segment. Figure 4.51 (b) shows the 2nd circle generation, which matches the circle well, and Figure 4.51 (c) shows the 7th circle generation, indicating that the match has continued to improve over 7 generations.

Table 4.3 shows the $RMSE$ values of successively generated circles during the removal of noise, for cases of noise variance varying from 0.0001 to 0.5. The first seven successive circles generated were measured. For small amounts of noise (0.0001 to 0.01), the initial segment does not appear to depart very much from the circle. The successive regeneration of the circle then increase the $RMSE$ slightly. For larger amounts of noise (0.1 to 0.3), the initial segment appears to depart significantly from the circle. In the case where the noise was 0.1, there was first a period of time where the $RMSE$ decreased upon circle regeneration (for the first 5 generations). This was followed by $RMSE$ increase over the remaining generations, with only a slight increase upon each regeneration. In



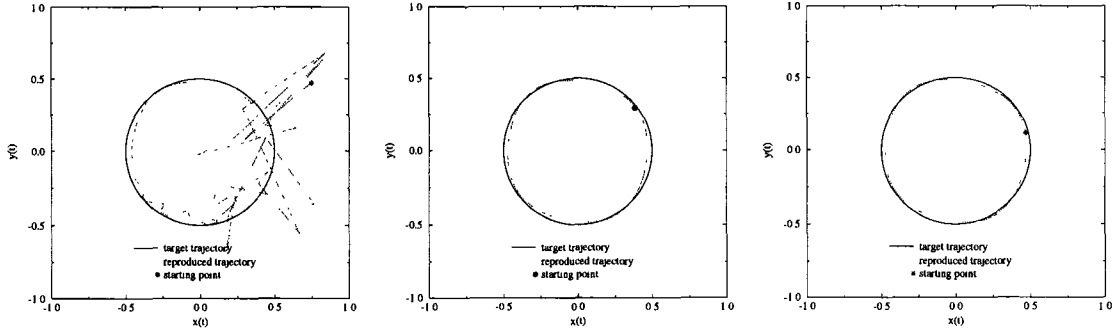
(a) 1st circle regeneration, $RMSE = 0.0131$ (b) 2nd circle regeneration, $RMSE = 0.0156$ (c) 7th circle regeneration, $RMSE = 0.0160$

Figure 4.49: *ATNN* removes the noise (variance $\sigma_{noise}^2 = 0.01$) from a time-series sequence and complete the subsequent path.



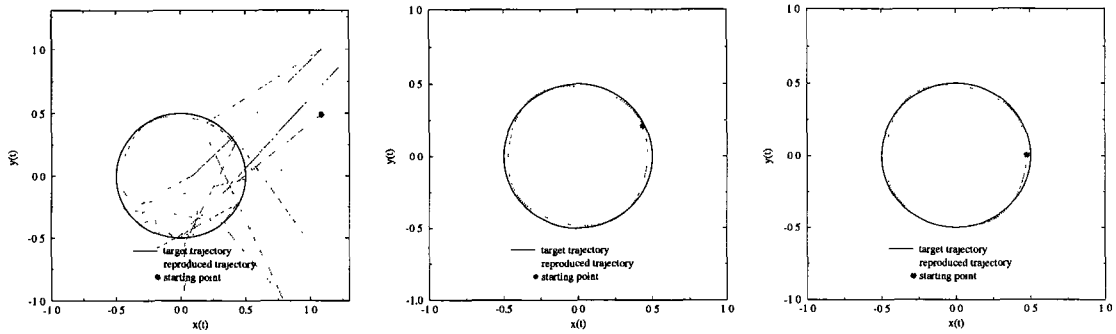
(a) 1st circle regeneration, $RMSE = 0.0549$ (b) 2nd circle regeneration, $RMSE = 0.0165$ (c) 7th circle regeneration, $RMSE = 0.0160$

Figure 4.50: *ATNN* removes the noise (variance $\sigma_{noise}^2 = 0.1$) from a time-series sequence and complete the subsequent path.



(a) 1st circle regeneration $RMSE = 0.1598$ (b) 2nd circle regeneration $RMSE = 0.0182$ (c) 7th circle regeneration $RMSE = 0.0160$

Figure 4.51: *ATNN* removes the noise (variance $\sigma_{noise}^2 = 0.3$) from a time-series sequence and complete the subsequent path.



(a) 1st circle regeneration, $RMSE = 0.2503$ (b) 2nd circle regeneration, $RMSE = 0.0186$ (c) 7th circle regeneration, $RMSE = 0.0160$

Figure 4.52: *ATNN* removes the noise (variance $\sigma_{noise}^2 = 0.5$) from a time-series sequence and complete the subsequent path.

σ_{noise}^2	order of circle reproduced						
	1st	2nd	3rd	4th	5th	6th	7th
0.0001	0.0120	0.0155	0.0159	0.0159	0.0160	0.0160	0.0160
0.001	0.0120	0.0155	0.0159	0.0159	0.0160	0.0160	0.0160
0.01	0.0131	0.0156	0.0159	0.0159	0.0160	0.0160	0.0160
0.1	0.0549	0.0165	0.0160	0.0160	0.0160	0.0160	0.0160
0.3	0.1598	0.0182	0.0162	0.0160	0.0160	0.0160	0.0160
0.5	0.2503	0.0186	0.0162	0.0160	0.0160	0.0160	0.0160

Table 4.3: *RMSE* of successively generated circles from varied noisy initial data, $\sigma_{noise}^2 = 0.0001$ to 0.5, when full circle is trained.

the cases where the noise was at least 0.3, the first seven generations were in the "cleanup" period, where *RMSE* decreased upon each regeneration.

Simulations on figure eight were conducted in the same way as for the circular discussed above. Figure 4.53 and Figure 4.54 shows the reproduction of the *ATNN* (with 5 hidden units) given noisy initial inputs after the network has done various amount of practice. The reproduction ability improved if the network is provided adequate learning. The noise clean up capability increases dramatically if we allow appropriate amount of hidden units and learning time. The recalling performance of *ATNN* employing eight hidden units are demonstrated in Figure 4.55 and Figure 4.56.

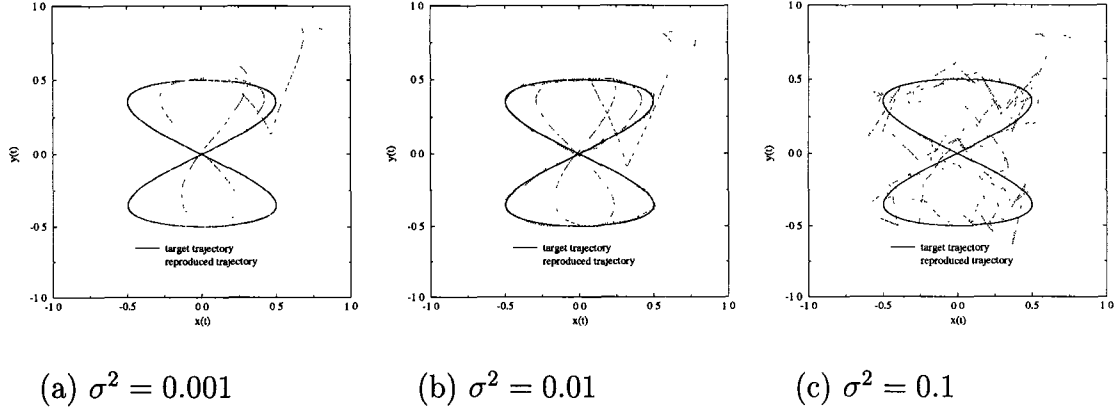


Figure 4.53: $ATNN (2 \xrightarrow{4} 5 \xrightarrow{6} 2)$ trained after $T = 40000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).

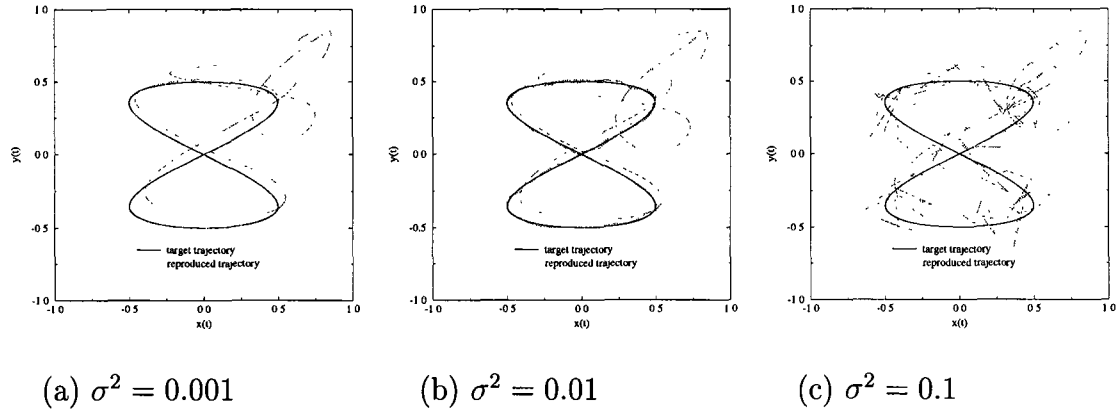


Figure 4.54: $ATNN (2 \xrightarrow{4} 5 \xrightarrow{6} 2)$ trained after $T = 80000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).

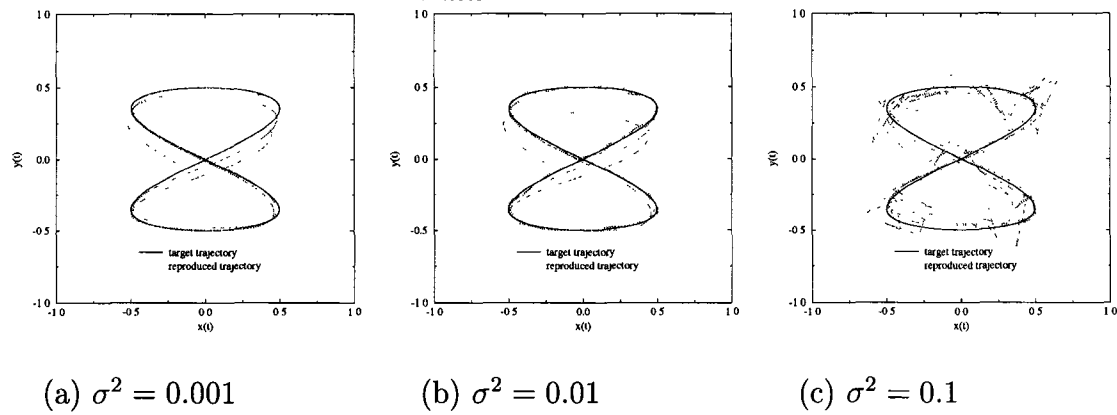


Figure 4.55: $ATNN (2 \xrightarrow{4} 8 \xrightarrow{6} 2)$ trained after $T = 40000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).

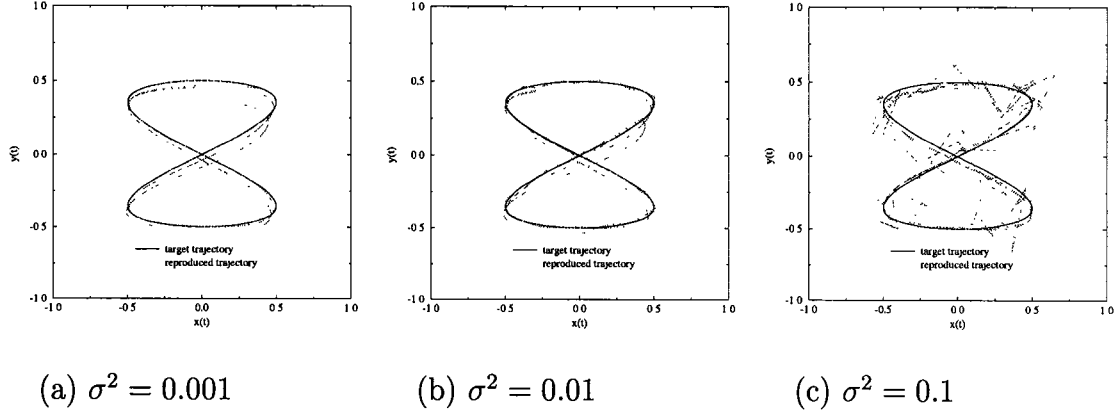


Figure 4.56: $ATNN$ ($2 \xrightarrow{4} 8 \xrightarrow{6} 2$ trained after $T = 75000$) reproduction by using noisy initial segment ($\sigma_{noise}^2 = 0.001 \sim 0.1$).

4.5.3 Noise Removal as Portion of the Pattern is Trained

The noise removal capability of the $ATNN$ has been demonstrated in previous section. This feature is an unique contribution of the developed network and is very important in robotic and control applications under noisy environments. In this section, the capability is further tested on a network trained on a small portion of the pattern to show how the network can still perform noise removal.

The trained network as described in section 4.4.1 for $\frac{3}{4}$ and $\frac{1}{2}$ circle are both tested for noise removal. Given a small piece of noisy data, the network tries to iteratively approach the next point by feeding back the newly produced data. Various amounts of normally distributed noise with variance σ_{noise}^2 of the range 0.0001 to 0.5 were added to the original pattern and the network tried to remove the noise along each newly generated circle, as described in the previous section. The $RMSE$ values on successive produced patterns under various noise circumstances are recorded in Table 4.3, 4.4, and 4.5 for networks trained from a full circle, $\frac{3}{4}$ circle and $\frac{1}{2}$ circle, respectively.

To show how the network trained on the three quarter circle removed noise

σ_{noise}^2	order of circle reproduced						
	1st	2nd	3rd	4th	5th	6th	7th
0.0001	0.0121	0.0179	0.0191	0.0193	0.0193	0.0193	0.0193
0.001	0.0121	0.0179	0.0191	0.0193	0.0193	0.0193	0.0193
0.01	0.0131	0.0178	0.0191	0.0193	0.0193	0.0193	0.0193
0.1	0.0531	0.0174	0.0191	0.0193	0.0193	0.0193	0.0193
0.3	0.1547	0.0164	0.0189	0.0192	0.0193	0.0193	0.0193
0.5	0.2427	0.0158	0.0189	0.0193	0.0193	0.0193	0.0193

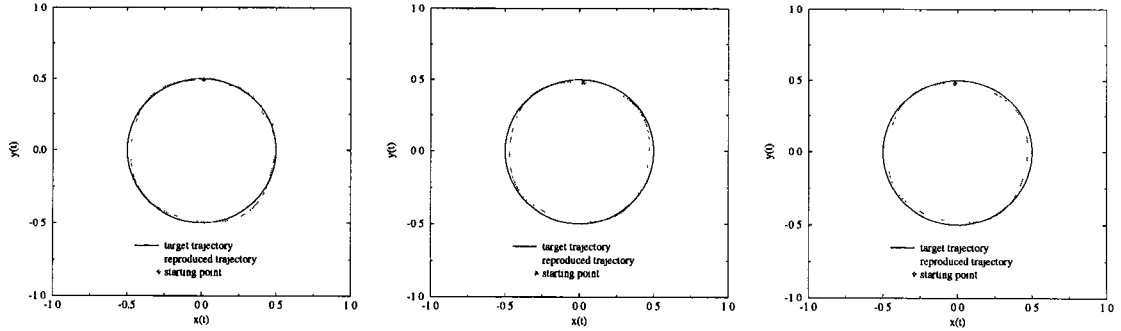
Table 4.4: *RMSE* of successively generated circles from variance noisy initial data, $\sigma_{noise}^2 = 0.0001$ to 0.5, when $\frac{3}{4}$ circle is trained.

along the pattern, the 1st, 2nd and 7th regenerated circle are plotted in Figure 4.57 (a) to (c) with noise variance 0.01 added to the initial segment. The results of regenerated circles of the same network trained on $\frac{3}{4}$ circle as noise variance increased to 0.3 are shown in Figure 4.58. After the first circle generated, the trajectory matched the circle closely. The output of noise removal of the *ATNN* trained on the half circle are drawn in Figure 4.59 and 4.60.

Generally speaking, the noise removal ability decreased as a smaller portion of the original pattern was used for training. To obtain immediate comparison, Figure 4.61 shows the average *RMSE* values along successive circles during the production procedure while the noise variance σ_{noise}^2 was 0.3. All networks could eventually clean up the noise, while the network trained on the half circle had lower performance.

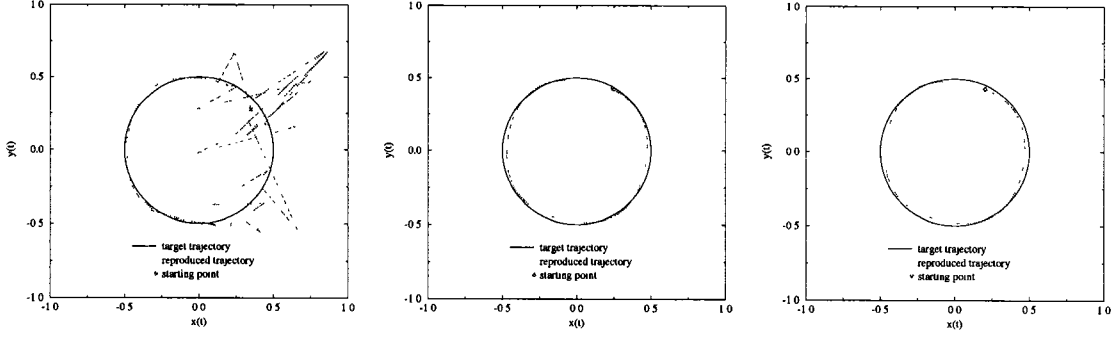
σ_{noise}^2	order of circle reproduced						
	1st	2nd	3rd	4th	5th	6th	7th
0.0001	0.0135	0.0219	0.0228	0.0227	0.0226	0.0225	0.0225
0.001	0.0136	0.0219	0.0228	0.0227	0.0226	0.0225	0.0225
0.01	0.0144	0.0220	0.0228	0.0227	0.0226	0.0225	0.0225
0.1	0.0445	0.0224	0.0227	0.0226	0.0225	0.0225	0.0225
0.3	0.1235	0.0216	0.0224	0.0225	0.0225	0.0226	0.0227
0.5	0.1823	0.0200	0.0222	0.0225	0.0226	0.0227	0.0227

Table 4.5: *RMSE* of successively generated circles from variance noisy initial data, $\sigma_{noise}^2 = 0.0001$ to 0.5, when $\frac{1}{2}$ circle is trained.



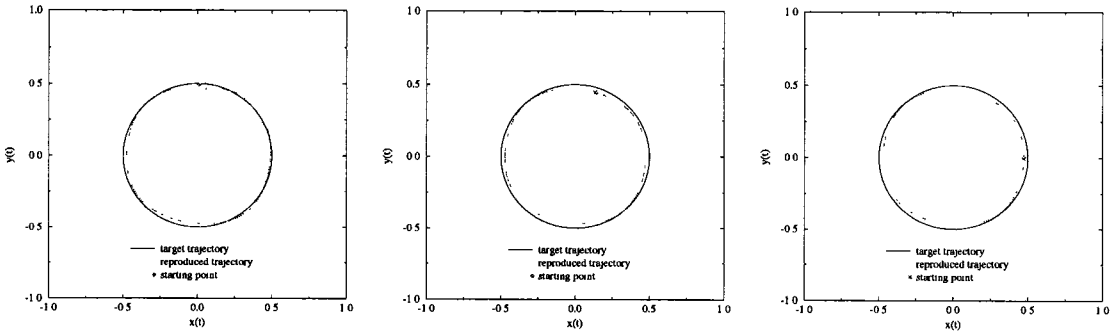
(a) 1st circle regeneration, *RMSE* = 0.0165 (b) 2nd circle regeneration, *RMSE* = 0.0174 (c) 7th circle regeneration, *RMSE* = 0.0548

Figure 4.57: *ATNN* removes the noise (variance $\sigma^2 = 0.01$) from a time-series sequence and complete the subsequent path when $\frac{3}{4}$ circle is trained.



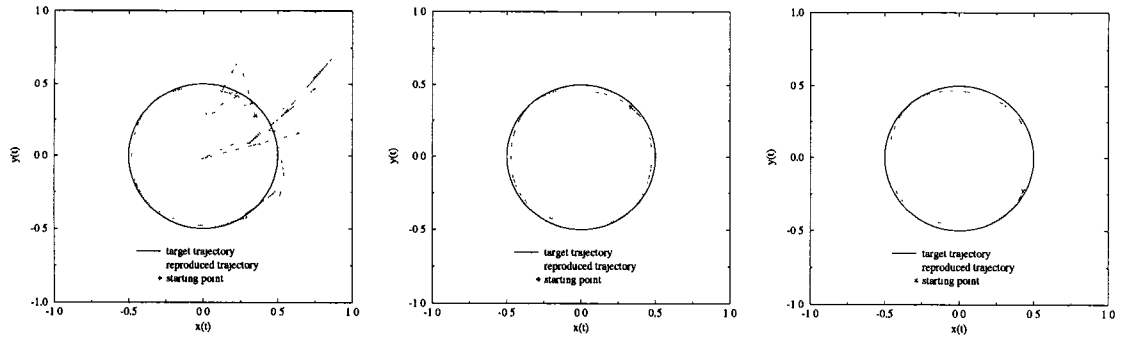
(a) 1st circle regeneration, $RMSE = 0.2047$ (b) 2nd circle regeneration, $RMSE = 0.1535$ (c) 7th circle regeneration, $RMSE = 0.0973$

Figure 4.58: *ATNN* removes the noise (variance $\sigma^2 = 0.3$) from a time-series sequence and complete the subsequent path when $\frac{3}{4}$ circle is trained.



(a) 1st circle regeneration, $RMSE = 0.0262$ (b) 2nd circle regeneration, $RMSE = 0.0770$ (c) 7th circle regeneration, $RMSE = 0.3253$

Figure 4.59: *ATNN* removes the noise (variance $\sigma^2 = 0.01$) from a time-series sequence and complete the subsequent path when $\frac{1}{2}$ circle is trained.



(a) 1st circle regeneration, $RMSE = 0.1999$ (b) 2nd circle regeneration, $RMSE = 0.2313$ (c) 7th circle regeneration, $RMSE = 0.4573$

Figure 4.60: *ATNN* removes the noise (variance $\sigma^2 = 0.3$) from a time-series sequence and complete the subsequent path when $\frac{1}{2}$ circle is trained.

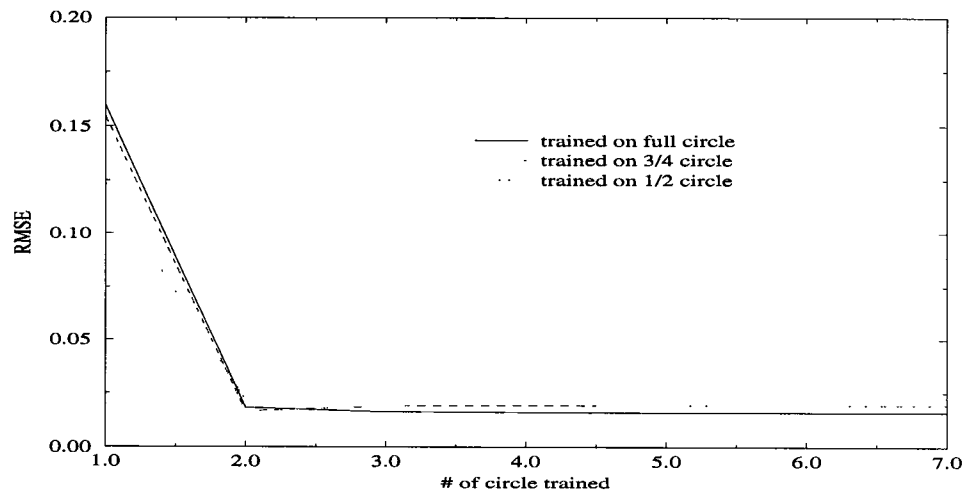


Figure 4.61: $RMSE$ of various training trajectories during regeneration of successive circles.

4.5.4 Remarks

The work shown in this section can be considered an extension of the approach of Kanerva (1988) with his sparse distributed memory (*SDM*). The *SDM* was trained in a static situation to produce a sequence of images with each successive image used as input. The *SDM* was able to generate successively cleaner images when presented with a noisy initial image. Our intention was to test the noise cleaning property of the *ATNN* in a similar situation except that the *ATNN* produces a dynamic time-varying output as a function of temporal properties and parameters within the network.

In summary, the *ATNN* is indeed capable of learning the features of spatio-temporal patterns and signals instead of memorizing the static position of each point. We showed that the *ATNN* learned a full circle from a partial circle, starting at different positions, and still could find the full circle in spite of initial noise.

In these experiments, the *ATNN* network is acting as a dynamic network because of the learning configuration. The results of the output layer are used as input to the network when the network produces its next values for the next position on the trajectory. This creates an effective connection from the output layer to the input layer, which is a recurrent loop. The presence of this loop creates a dynamic network as opposed to a feed-forward type of net. Dynamic networks are capable of dynamic activity, including attractors that are multistate oscillations, limit cycle oscillations, and chaotic oscillations, as well as fixed point attractors. This rich repertoire of dynamic activity shows much promise for a variety of applications, including process control, signal recognition, and robotic motion control. A key question is how to train the attractors into a neural

network. We will discuss this issue in the next section.

4.6 Attractor Learning

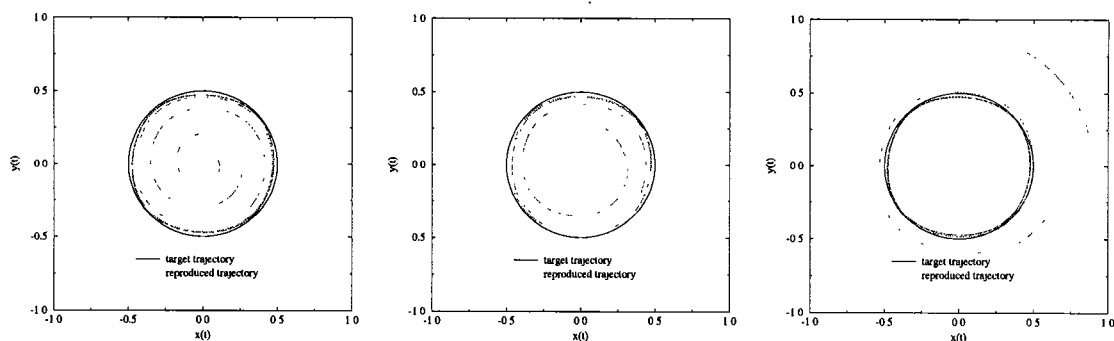
Attractors are a very interesting dynamic behavior of dynamic systems. There are three classic types of dynamical motion: (1) equilibrium, (2) periodic motion or limit cycle, and (3) quasi-periodic motion. These states are called attractors [102]. The idea of storing memory states or patterns to be recognized or recalled as static attractors of the neural network dynamics implies that initial configurations of neurons in some neighborhood of a memory state will be attracted to it [165]. In applications, these attractors could represent memories, pattern classes, or stable control actions. Attractors have been shown that can be developed by tuning the weights of connections [35]. Amari had pointed out that some networks shows interesting dynamical behaviors of recalling processes: when starting with an initial state close to a memorized pattern, the network monotonically approaches the memorized one [1]. Assume x_t denotes a state of the net at time t , and T is the non-linear state transition operator as $x_{t+1} = Tx_t$, a state x is equilibrium if $x = Tx$. The basin $B(x)$ of an equilibrium state x is the set of those states which will falls in the state x after a finite number of state transitions, i.e., $B(x) = \{x_0 | \exists N \text{ such that } T^N x_0 = x\}$.

The results on noise removal shown above suggest that the trajectories rained into the *ATNN* networks are in fact attractors. The circular and figure eight trajectories can be considered limit cycle attractors or multistate oscillations because of the repetition of points along the closed figures. In spite of the fact that different initial arcs were used, including very noisy arcs, when starting the

network trajectory, the nets always arrived at the trajectory for which they were trained. Thus the initial arcs were within the basin of attraction for the trained figures (attractors), both for the circle and figure eight.

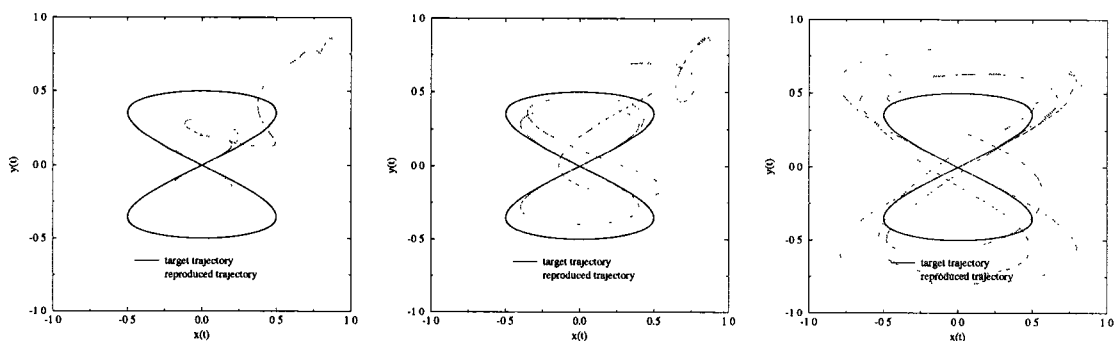
We conducted additional experiments on the basins of attraction for the circle and figure eight attractors in these networks. We used a network with the same configuration as before ($2 \xrightarrow{4} 5 \xrightarrow{6} 2$), and allowed the point specified by the output units to be used as input in the next iterations. The network that was trained on the circle trajectory was used. We used as starting segments arcs from circles that were larger or smaller than the trained circular figure, but with the same center position. Figure 4.62 shows the results. The network generated a spiral path from the initial circular arc to the trained circle, then continued to generate the trained circle indefinitely. When the initial arc was from a smaller circle, the trajectory spiraled outward until it met the trained circle, and when the initial arc was from a larger circle, the trajectory spiraled inwards. Figure 4.63 and Figure 4.64 show the result of network approaches to the figure eight attractors. It is much more difficult than previous experiment on circular trajectory, as it contains multiple loops. The performance improved when eight hidden units were incorporated (Figure 4.64).

The impact of this result is that an autonomous or controlled system, such as a robotic arm, a vehicle, or other moving object, could be trained to generate repetitive desired motions, and could attain this repetitive motion from arbitrary starting trajectories. The starting trajectories would have to be in the basin of attraction for the attractor that is the trained repetitive motion. However, the results here indicate that the basin of attraction is quite large for the situation under study.



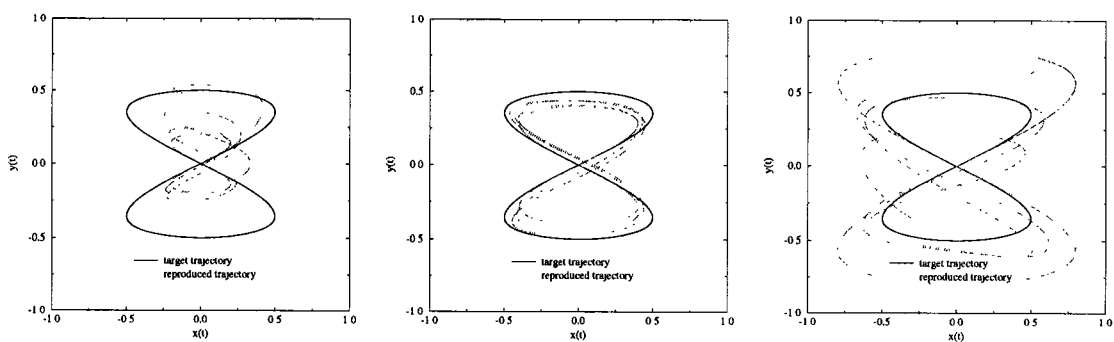
(a) initial radius is 0.1 (b) initial radius is 0.3 (c) initial radius is 0.9

Figure 4.62: The network finally follows a track of some attractor for widely varying radii of the initial circle.



(a) initial radius is 0.2 (b) initial radius is 0.4 (c) initial radius is 0.8

Figure 4.63: The network can not approach basin of attraction of figure eight (with net: $2 \xrightarrow{4} 5 \xrightarrow{6} 2$).



(a) initial radius is 0.2 (b) initial radius is 0.4 (c) initial radius is 0.8

Figure 4.64: The network approaches basin of attraction of figure eight (with net: $2 \xrightarrow{4} 8 \xrightarrow{6} 2$)

4.7 Sampling Rate Effects

Sampling rate has the effect of speeding up or slowing down the movement of the pattern along its trajectory in $\mathcal{R}^n$ and of translating forward or backward in time [61]. The pattern will still traverse the same spatial trajectory when a different sampling rate is applied. One of the basic requirements for digital computation in signal or image processing is that the signal be available in digital or binary form. Sampling theory states that a bandlimited signal sampled above its Nyquist rates can be recovered without error by low-pass filtering the sampled signal. Sampling rate effects in neural network models are not well studied as of yet. In this section, we study the effect of sampling rate on the training speed and recognition performance in the specific problem of circle pattern production.

4.7.1 Sampling Rate vs. Training Speed and Performance

The *ATNN* was trained on different sampling data: 64 points per circle, 32 points per circle, 21 and 16 points per circle, all well above the Nyquist rate. The simulation results show (Figure 4.65) that the network converges faster when the lower sampling rate data is employed. All cases could converge to similar *RMSE* values shown in Table 4.6 and 4.7. The performance of the pattern retrieval during learning progress are shown in Figure 4.66 to Figure 4.69 for various sampling rates and iterations of training. Thus the network converges with different speed for different sampling rates. There were also different time delay requirements, as shown in Table 4.8. In the case with training sequence of 16 points per circle, the maximum time delay of the whole network was 34. The

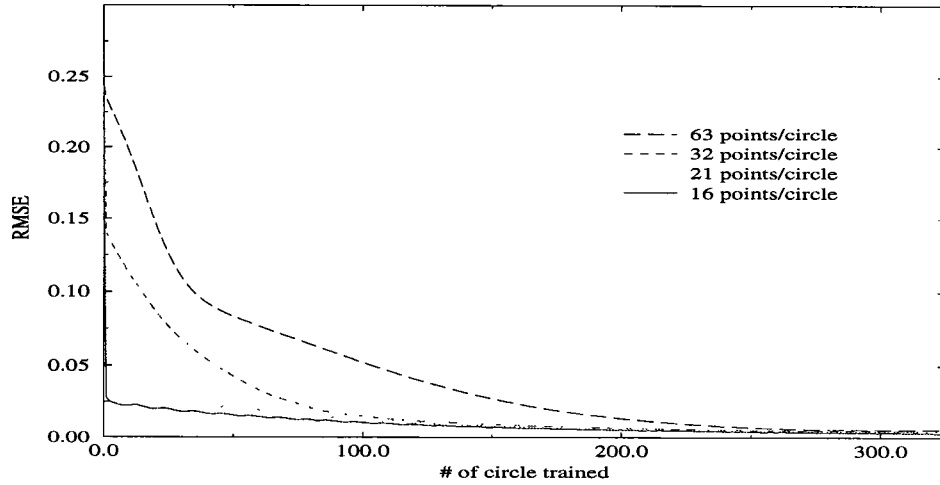


Figure 4.65: *RMSE* of various sampling rate during training phase.

resulting elements of time-delay matrices are shown in detail in Appendix B. The total time delay needed using 64 points per circle was only 17; the network trained on 16 points per circle required twice as much delay. Low sampling rate upgrades the training speed but requires a longer initial segment to recall the patterns.

Notice that the training series has a possibly infinite number of non-repeat points on the circle. Since the training pattern is constructed from $\sin 2\pi$ and $\cos 2\pi$ with 0.1 increment and π is an irrational number, each cycle has different points. This corresponds to the property of a limit cycle attractor where successive points generated by a system form a densely filled closed figure [40, 35].

4.7.2 Reproduction From Different Starting Positions

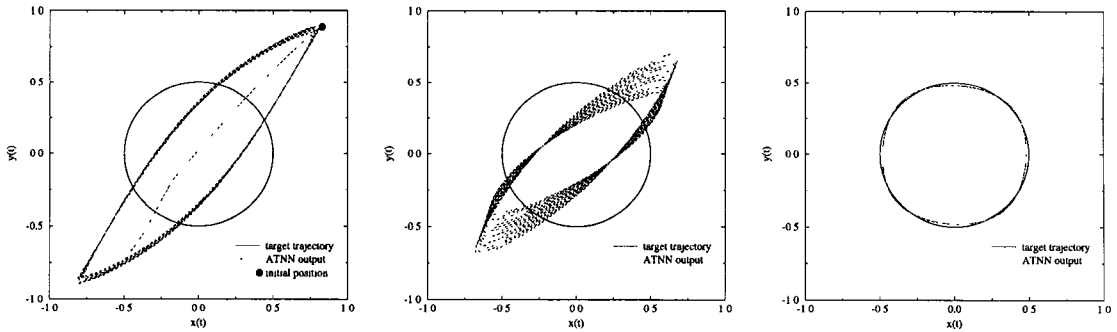
Production capability of *ATNN*'s trained on various sampling rates were tested by giving a random different starting positions for each run. The numerical data is summarized in Table 4.9. The results are plotted in Figure 4.70, 4.71 and 4.72. We can conclude from this table that the performance of networks trained from

points per circle	# of circle trained				
	1	5	10	25	50
63	2.4437e-01	2.2277e-01	2.0087e-01	1.2952e-01	8.3963e-02
32	2.2863e-01	1.2957e-01	1.1439e-01	7.8265e-02	4.3829e-02
21	2.1029e-01	3.5844e-02	3.1620e-02	2.7754e-02	2.1786e-02
16	1.6010e-01	2.4175e-02	2.2379e-02	2.0574e-02	1.6028e-02

Table 4.6: *RMSE* comparison of various sampling rate during training phase

points per circle	# of circle trained				
	100	150	200	250	300
63	5.2819e-02	2.7747e-02	1.3743e-02	7.4201e-03	5.6071e-03
32	1.5177e-02	9.6177e-03	6.8922e-03	5.4706e-03	4.9016e-03
21	1.2701e-02	8.4610e-03	5.6143e-03	4.4714e-03	4.4062e-03
16	1.0880e-02	7.3349e-03	5.7197e-03	4.2824e-03	3.3434e-03

Table 4.7: *RMSE* comparison of various sampling rate during training phase



(a) 1 to 8 circles

(b) 15 to 30 circles

(c) after 250 circles

Figure 4.66: Performance progress during training when 64 point per circle is employed

	# of points per circle			
	63	32	21	16
$\max(T_1)$	3	8	13	8
$\max(T_2)$	14	22	16	26
T_{atnn}	17	30	29	34

Table 4.8: The resulting time-delay variables after training is complete, where T_1 and T_2 denote the time-delay matrix on layer one and two respectively, and T_{atnn} is the total delay of trained network where $T_{atnn} = \max(T_1) + \max(T_2)$.

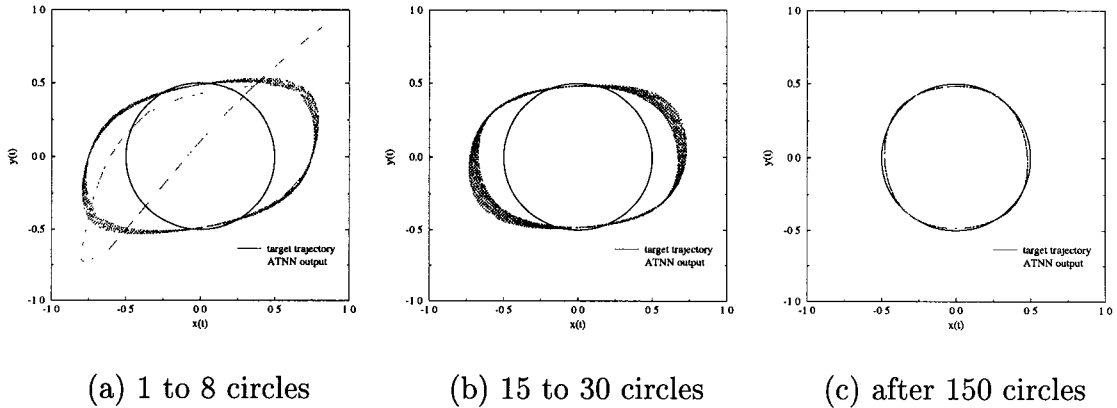


Figure 4.67: Performance progress during training when 32 point per circle is employed

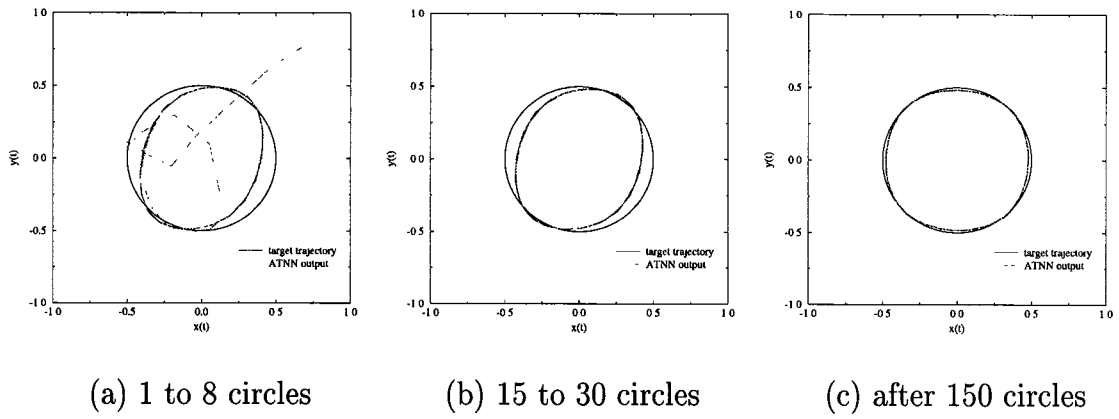


Figure 4.68: Performance progress during training when 21 point per circle is employed

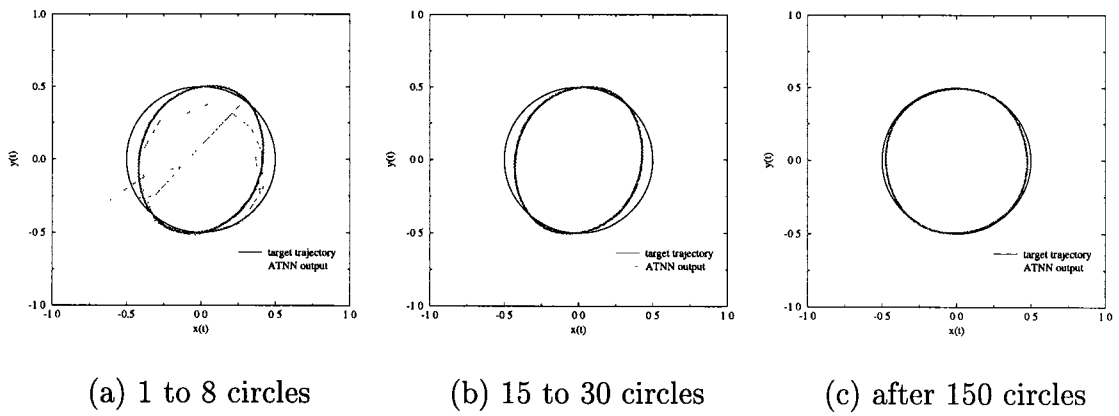


Figure 4.69: Performance progress during training when 16 point per circle is employed

points/circle trained	different starting position			average
	(a)	(b)	(c)	
64	0.0113	0.0162	0.0114	0.0130
32	0.0827	0.0170	0.0190	0.0395
21	0.2215	0.1036	0.0729	0.1326
16	0.1741	0.0776	0.0313	0.0943

Table 4.9: *RMSE* of reproduced figure under different starting positions when various training sampling rates were employed.

64 and 32 point sampling rates are similar but the performance from networks trained on 21 and 16 points are not as good.

4.7.3 Noise Tolerance

The noise tolerance is tested by adding different amounts of white noise to patterns and measuring the *RMSE* of output and desired target. The *RMSE* curves of each trained network under different sampling rate training patterns with noise variance of 0.001 to 0.1 are plotted in Figure 4.73. The noise resilience of lower sampling rate training patterns are not as good as that of network trained on 64 points per circle. The lower sampling rate is employed, the worse outcome it has.

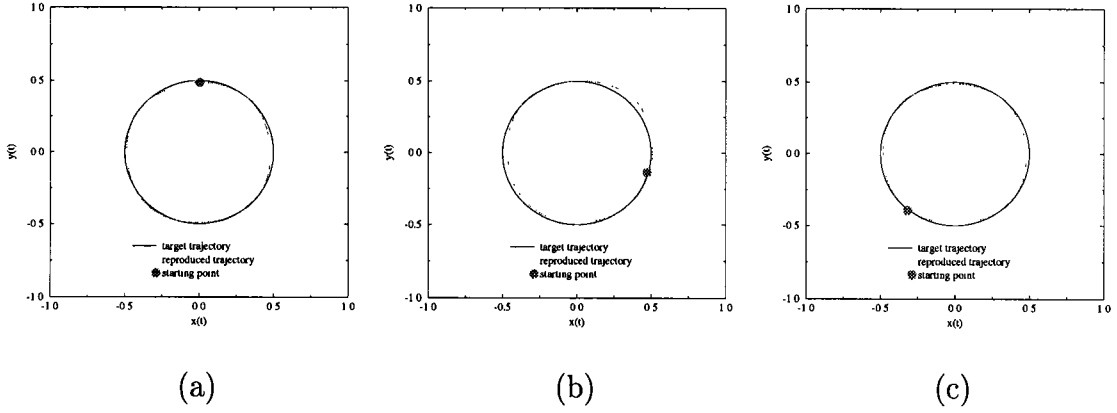


Figure 4.70: The results of circle reproduced by *ATNN* start from different initial position (a) $RMSE = 0.0827$ (b) $RMSE = 0.1700$ (c) $RMSE = 0.0190$ when trained on 32 points per circle pattern.

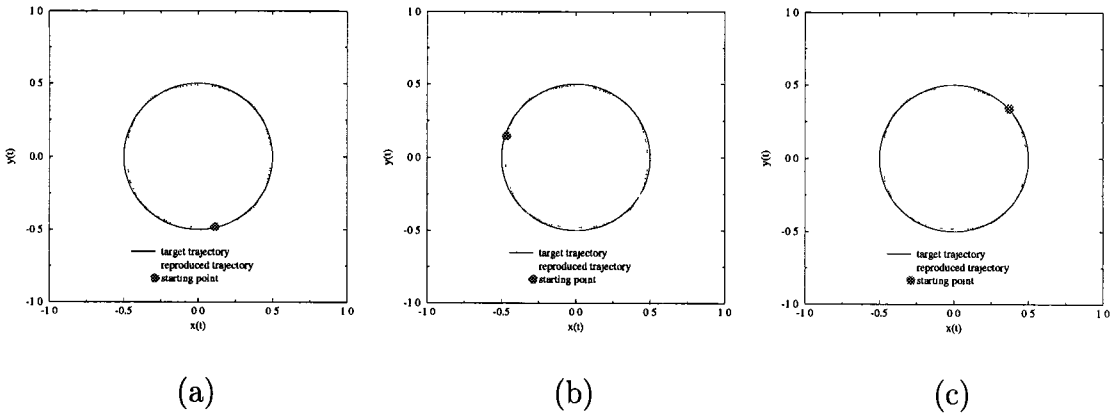


Figure 4.71: The results of circle reproduced by *ATNN* start from different initial position (a) $RMSE = 0.2215$ (b) $RMSE = 0.1036$ (c) $RMSE = 0.0729$ when trained on 16 points per circle pattern.

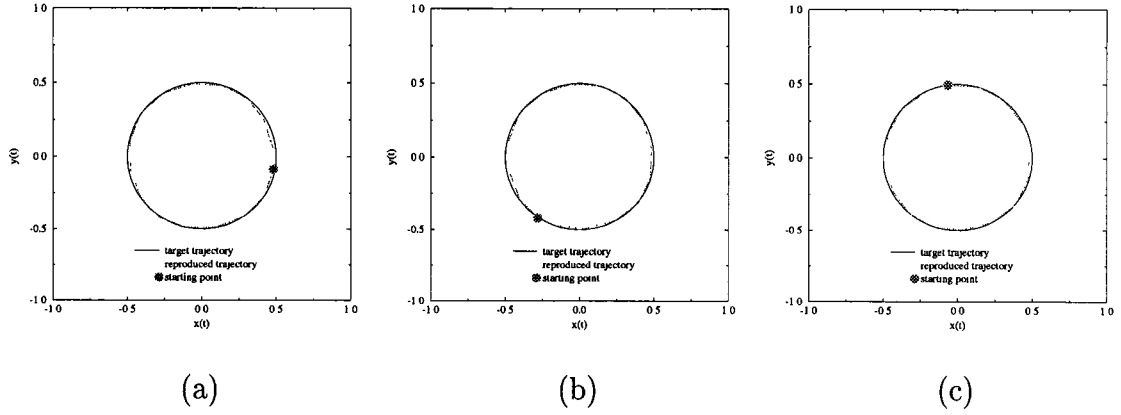


Figure 4.72: The results of circle reproduced by *ATNN* start from different initial position (a) $RMSE = 0.1741$ (b) $RMSE = 0.0776$ (c) $RMSE = 0.0313$ when trained on 16 points per circle pattern.

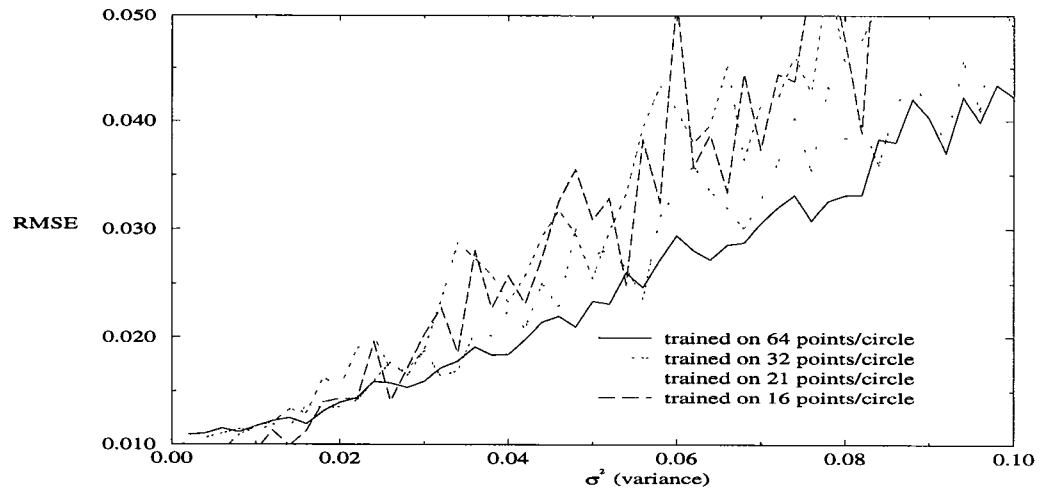


Figure 4.73: Noise tolerance of network trained from various sampling rate pattern.

4.7.4 Noise Removal as Patterns with Different Sampling Rate are Trained

In this section, the noise removal capability was further examined to see if the network can clean up noise even if a low sampling rate is applied. Various amounts of noise were added to the original signal and we provided the network sufficient length of data to evaluate the new position; the newly generated signal was used to compose further data. The average $RMSE$ of each successive generated circle from the networks trained on 32, 21 and 16 points sampling rate are shown in Table 4.10, 4.11 and 4.12. To show the performance of each network intuitively, the generated patterns along each circle pass are selectively presented in Figures 4.75 to 4.80. When a small amount of initial noise was added to test the trajectory, the network could barely maintain reproduction at the second circle (see Figure 4.75(b), 4.77(b), and 4.79(b)), but it can no longer perform well beyond this stage (e.g. in Figure 4.75(c), 4.77(c), or 4.79(c)). When a larger amount of noise was applied, the network could no longer clean up any noise. Examples are shown in Figure 4.76, 4.78 and 4.80. To give a better view of the noise removal capability of various trained networks, the $RMSE$ variation along successive circles generated from different networks is shown in Figure 4.74. We can conclude that the network trained on lower sampling rates could no longer maintain noise removal since the neighboring points are too far apart from each other and the correlation information is insufficient.

σ_{noise}^2	order of circle reproduced						
	1st	2nd	3rd	4th	5th	6th	7th
0.0001	0.0018	0.0124	0.1280	0.7236	0.9886	0.9928	0.9928
0.001	0.0020	0.0123	0.1270	0.7216	0.9885	0.9928	0.9928
0.01	0.0102	0.0220	0.2656	0.8670	0.9794	0.9808	0.9808
0.1	0.0939	0.1433	0.7683	0.9767	0.9809	0.9808	0.9808
0.3	0.3140	0.6989	0.9872	0.9926	0.9928	0.9928	0.9928
0.5	0.5697	0.4375	0.9275	0.9805	0.9809	0.9808	0.9808

Table 4.10: *RMSE* of successively generated circles from noisy initial data. The amount of noise was varied: $\sigma_{noise}^2 = 0.0001$ to 0.5, and 32 points per circle is trained.

σ_{noise}^2	order of circle reproduced						
	1st	2nd	3rd	4th	5th	6th	7th
0.0001	0.0001	0.0071	0.0221	0.1812	0.8355	1.0516	1.0547
0.001	0.0001	0.0072	0.0239	0.2009	0.8624	1.0523	1.0548
0.01	0.0110	0.0097	0.0312	0.2596	0.8608	1.0520	1.0562
0.1	0.0930	0.0682	0.3088	0.9046	1.0535	1.0562	1.0562
0.3	0.2722	0.1852	0.6787	1.0453	1.0547	1.0548	1.0548
0.5	0.6003	0.5221	0.9934	1.0542	1.0548	1.0548	1.0548

Table 4.11: *RMSE* of successively generated circles from noisy initial data. The amount of noise was varied: $\sigma_{noise}^2 = 0.0001$ to 0.5, and 21 points per circle is trained.

σ_{noise}^2	order of circle reproduced						
	1st	2nd	3rd	4th	5th	6th	7th
0.0001	0.0001	0.0001	0.0058	0.0101	0.0447	0.2267	0.7596
0.001	0.0001	0.0012	0.0060	0.0093	0.0387	0.1985	0.7111
0.01	0.0086	0.0103	0.0121	0.0533	0.2647	0.8100	1.0474
0.1	0.1067	0.0902	0.0578	0.2317	0.7526	1.0395	1.0611
0.3	0.2872	0.2972	0.2615	0.7463	1.0412	1.0611	1.0618
0.5	0.4830	0.4662	0.2583	0.3490	0.8685	1.0418	1.0493

Table 4.12: *RMSE* of successively generated circles from noisy initial data. The amount of noise was varied: $\sigma_{noise}^2 = 0.0001$ to 0.5, and 16 points per circle is trained.

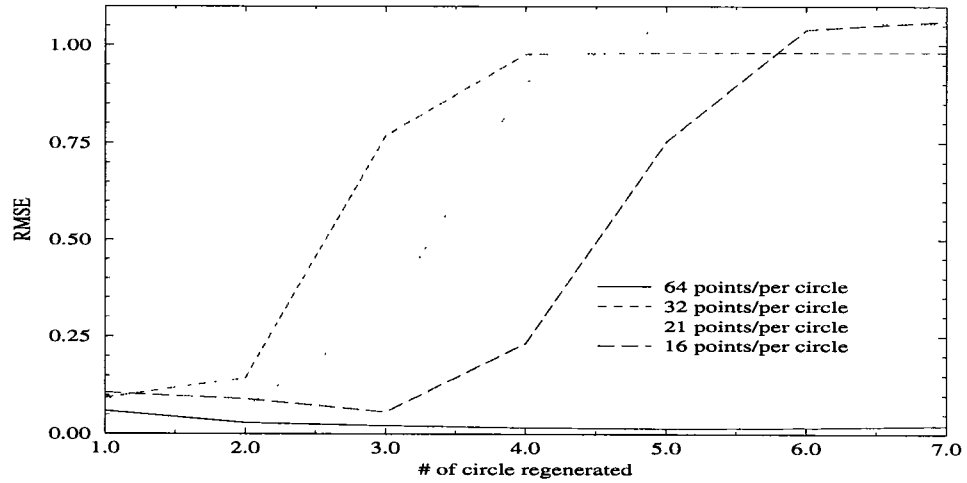
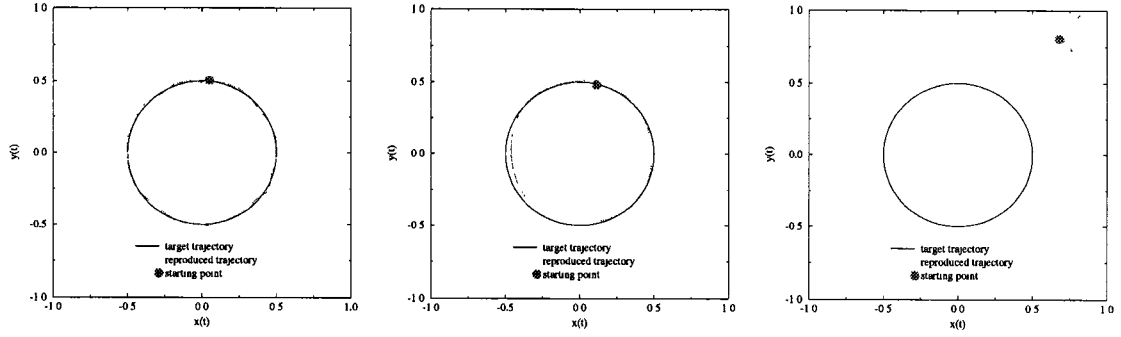
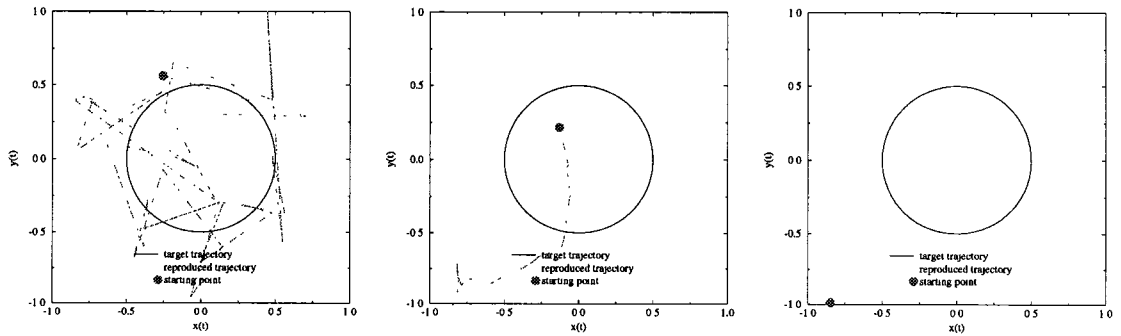


Figure 4.74: Comparison of *RMSE* of successive circle reproduced when various sampling rate signals are trained.



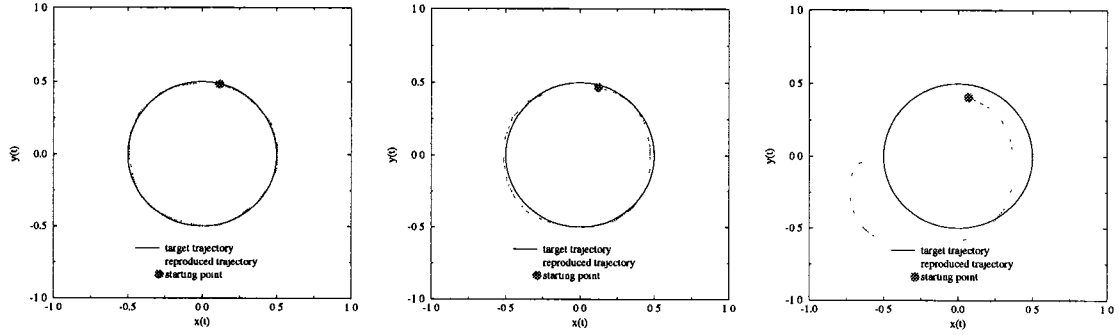
(a) 1st circle regeneration, $RMSE = 0.0102$ (b) 2nd circle regeneration, $RMSE = 0.0220$ (c) 4th circle regeneration, $RMSE = 0.8670$

Figure 4.75: *ATNN* can barely remove small amount of noise (variance $\sigma^2 = 0.01$) at the 2nd sequence but not for the subsequent path when 32 points per circle is trained.



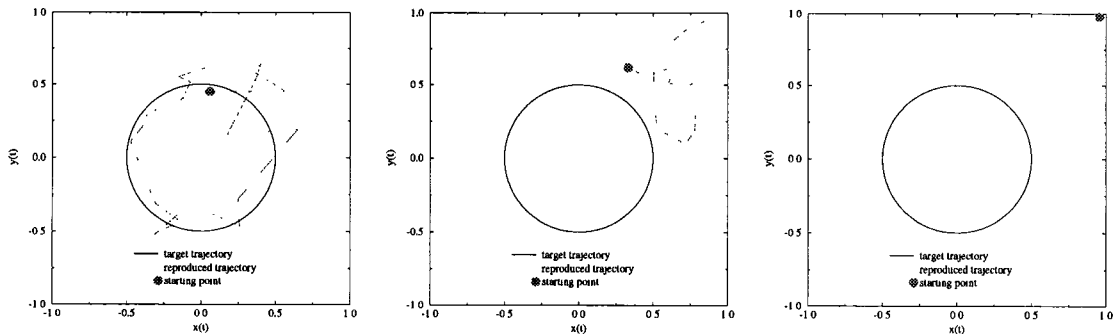
(a) 1st circle regeneration, $RMSE = 0.3140$ (b) 2nd circle regeneration, $RMSE = 0.6989$ (c) 4th circle regeneration, $RMSE = 0.9926$

Figure 4.76: *ATNN* can no longer clean up the noise when larger amount of initial noise is added (variance $\sigma^2 = 0.3$) as 32 points per circle is trained.



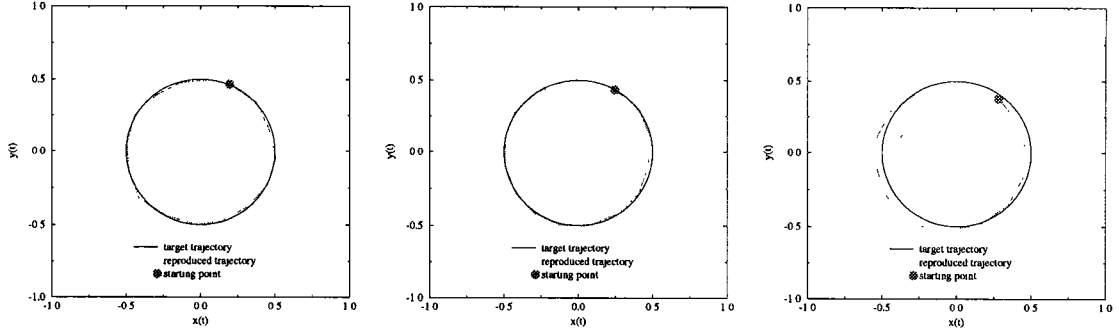
(a) 2nd circle regeneration, $RMSE = 0.0097$ (b) 3rd circle regeneration, $RMSE = 0.0312$ (c) 4th circle regeneration, $RMSE = 0.2596$

Figure 4.77: *ATNN* can barely remove small amount of noise (variance $\sigma^2 = 0.01$) at the 2nd sequence but not for the subsequent path when 21 points per circle is trained.



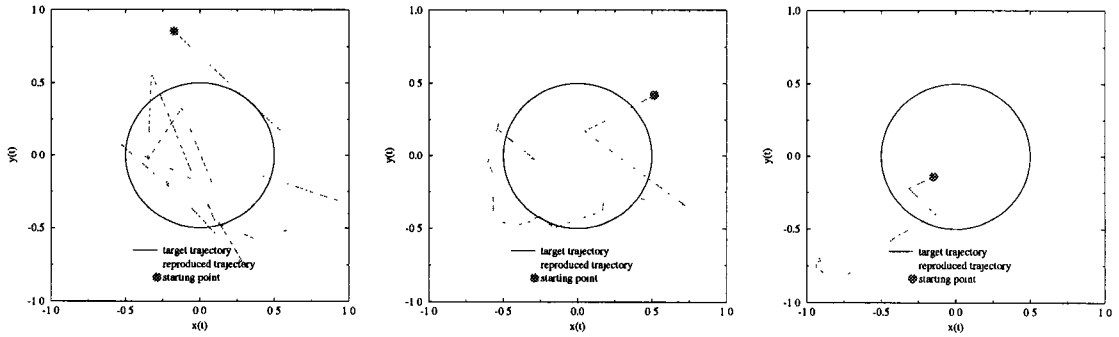
(a) 2nd circle regeneration, $RMSE = 0.1852$ (b) 3rd circle regeneration, $RMSE = 0.6787$ (c) 4th circle regeneration, $RMSE = 1.0453$

Figure 4.78: *ATNN* can no longer clean up the noise when larger amount of initial noise is added (variance $\sigma^2 = 0.3$) as 21 points per circle is trained.



(a) 2nd circle regeneration, $RMSE = 0.0103$ (b) 3rd circle regeneration, $RMSE = 0.0121$ (c) 4th circle regeneration, $RMSE = 0.0533$

Figure 4.79: *ATNN* can barely remove small amount of noise (variance $\sigma^2 = 0.01$) at the 2nd sequence but not for the subsequent path when 16 points per circle is trained.



(a) 2nd circle regeneration, $RMSE = 0.2972$ (b) 3rd circle regeneration, $RMSE = 0.2615$ (c) 4th circle regeneration, $RMSE = 0.7463$

Figure 4.80: *ATNN* can no longer clean up the noise when larger amount of initial noise is added (variance $\sigma^2 = 0.3$) as 16 points per circle is trained.

4.7.5 Remarks

We have tested sampling rate vs. training speed and performance, reproduction from different starting positions, and noise removal as patterns of different sampling rates were trained. Sampling rates are important in determining signal recovery or production, and in determining training speed. Superior performance for trajectory reproduction and noise removal were found at high sampling rates.

This work is consistent to Doya's work in the study of the adaptive neural oscillator using continuous time back-propagation learning [39]. Through their extensive experiments, the network has difficulty to learn and regenerate the input wave form if the period T of the external input is larger. In other words, the net can not catch the dynamics efficiently when the the input oscillates up and down too quickly.

4.8 Discussions

We have used the Adaptive Time-delay Neural Network (*ATNN*), which has a dynamic learning technique for adapting both weights and time-delays on-line, for training with temporally varying patterns. The network is an efficient and flexible model that addresses learning situations where inputs arrive over a period of time and outputs are produced over a period of time. The network has spatiotemporal patterns as both inputs and outputs, and adapts its time-delays and weights independently. Time-delays were trained to set individual length optimally to catch the necessary or significant information to the whole system.

The *ATNN* can be applied to a wide variety of domains, including the recognition of movements, trajectory generation, speech analysis, and signal process-

ing. We have applied the *ATNN* to examples which possess spatiotemporal complexity. The *ATNN* was trained to learn the transformation of given input sets to output sets in sequence. The simulation results show that the *ATNN* learns the topology of a circular trajectory and figure eight within 500 on-line training steps, and reproduces the circular trajectory itself dynamically with very high accuracy. These results demonstrate the capability of the network to perform predictive recognition and figure completion when only partial information of the original data is available. The trained network produces the pattern or figure no matter where the starting position is.

When noise was added to the initial segment of a circle, the network eliminates the noise and completes the trajectory of the circle. For small amounts of noise, the network produced a very close fit to the circular trajectory. When larger amounts of noise were present, the trajectory gradually attained a closer and closer fit to the circle.

Other results show that the *ATNN* successfully produced the whole circle when trained on a partial circle, as long as the length of the training segment was large enough. Experiments on sampling rates show that low sampling rate training patterns speeded up training but degraded the noise removal, and a larger length of initial segment was needed to perform recall.

In dynamic systems, attractors are one of the important properties which can describe the behavior of the system. We showed that when training on a closed trajectory, the trajectory became an attractor of the network. When trajectory arcs with different radii were presented as inputs to a trained *ATNN*, the network could retrieve the trained track for different radii of the initial arc. Initial noisy segments also resulted in the network attaining the original trained figure.

Thus, the network learned a fixed attractor and the varying initial segments were within the basin of attraction. This means that we can train a chosen attractor into a network's dynamics. Networks trained with appropriate attractors could have a significant impact on applications in robots, signal production, time series prediction and other temporally dynamic situations.

Applications to identification and control as well as signal processing and speech recognition are domains to which this type of network can be naturally applied. We will discuss and demonstrate these applications in the next two chapters. These applications show that the *ATNN* is a powerful tool and learning technique for spatiotemporal data and for any applications that involve time varying signals or patterns.

Chapter 5

Time Series Analysis: Prediction and Modeling

5.1 Introduction

Time series are sets of observations or measurements over time, where their governing parameters or embedded properties are *a priori* unknown. The lack of *a priori* knowledge is a very realistic situation in real world problems and this is where the challenges lie. Time series analysis is now very widely used in many fields of engineering, biomedical diagnosis, physical science, finance and economics.

Domains in which time series analysis applies include forecasting, identification, adaptive filtering, spectral analysis and statistical analysis. Many methodologies have been developed for analyzing and predicting observed time series. To name a few, these methods include least squares identification, and regression analysis (e.g. *AR*, *ARMA*, *ARIMA*) as in some classical literatures [5, 15, 16].

Recently, many researchers have applied neural net paradigms for the goal of time series prediction, modeling and characterization. Good examples can be seen in Santa Fe Institute Time Series Prediction Competition [158].

Prediction of evolution of temporally changing signals is an important field of research for mathematicians and physicists. Dynamic identification and modeling of temporally changing signals are important issues of real-time signal processing and understanding. Such changing signals may arise from moving objects in visual images, spoken words, target trajectories and other kinds of sensor data in a wide variety of applications. Chaotic time series prediction is an essential task of signal processing, physiological analysis and economic diagnosis. One of the significant features of chaotic behavior is that a chaotic signal continually generates new patterns, whereas simpler signals do not. Conventional methodology for probing this problem is to explore mathematical models that describe the physics behind the chaos phenomena and try to solve the parameters of the models by approximation. However, the computation may not converge to explicit solutions due to the high degree of nonlinearity, and resolution of parameters is often impractical, and not precise enough to construct a good physics model.

In neural systems, the relevant information is transformed by distributed neural processing. Therefore, the neural network is regarded as an information processing machine which distributes important information or features through a complex circuit inspired by the function of the human brain. Learning algorithms produce an estimation of the system behavior based on the observation of input-output pairs. These methodologies can be applied to the category of tasks in system identification and prediction. Neural network technology has been found to be useful for universal continuous function approximation [25, 59]. Re-

searchers have exploited connectionist technologies such as feedforward networks and radial basis functions for prediction and have demonstrated promising capability [74, 75, 83, 100, 101, 115, 157, 161].

Some neural net paradigms capture temporal dynamics, including the Adaptive Time-Delay Network (*ATNN*) and the Time-Delay Neural Network (*TDNN*), which employ time delay taps on connections. These networks have been successfully implemented for spatiotemporal pattern production and phoneme recognition [84, 90, 87, 152, 153]. In this chapter, we apply the Adaptive Time-Delay Neural Network (*ATNN*) to study the chaotic time series Mackey-Glass differential equation prediction and modeling problem. We discuss the embedding dynamics, training set size and performance, and prediction over time. We demonstrate this neural network approach to be flexible, to predict the chaotic dynamics, and to achieve good performance.

5.2 Problem Description

A dynamic system has time-varying outputs and outputs determined by a set of independent parameters. The delay differential equation of the form $\frac{dx(t)}{dt} = F(x(t), x(t - \tau))$ describes a dynamic system in which a stimulus has a delayed response. The time delay factor τ usually determines the nature of the time series. This equation fits many practical examples of physiological control systems, economics, and other fields. The model we will use in this study is the example proposed by Mackey and Glass which describes the production of white blood cells as following [94]:

$$\frac{dx(t)}{dt} = \frac{ax(t - \tau)}{1 + x^e(t - \tau)} - bx(t), \quad (5.1)$$

where $x(t)$ denotes the concentration of blood at time t , and τ is a delay time (for leukemia patient, τ is excessively long).

The dynamic properties of this differential equation have been addressed extensively by Farmer [44]. The Mackey-Glass system is infinite dimensional, which requires an infinite set of independent real numbers to specify an initial condition [44]. It possesses many dynamic properties such as nonlinearity, limit cycle oscillation, aperiodic wave forms, strange attractors, and other dynamic behaviors [44], and the Fourier spectrum is a broad band similar to that of noise. This dynamic system has been a benchmark in the prediction researches. For comparison and consistent with other works [75, 83, 100, 101], we set constrains $a = 0.2$, $b = 0.1$, $c = 10$ and the initial value $x = 0.8$. When $\tau = 17$, the resulting output is a quasi-cyclic signal with a characteristic time of $t_{char}^1 \approx 50$ and with an attractor of fractal dimension ≈ 2.1 [44]. Before collecting the training and test data set, the equation is iterated long enough (after 1000 iterations) to let the transients die out. The function evolution and state space from $t = 1$ to 1000 in training data is plotted in Figure 5.2.

5.3 Performance of TDNN on Prediction Performance

Due to the nonlinear complexity of the time-delay nets, theoretical analysis in general is intractable. Nevertheless, simplified analysis of using short term memory and delay effects were reported earlier [116, 39, 60, 8]. In summary, the effects

¹ t_{char} is the "characteristic time" for the time series estimated as the inverse of the mean frequency in the power spectrum

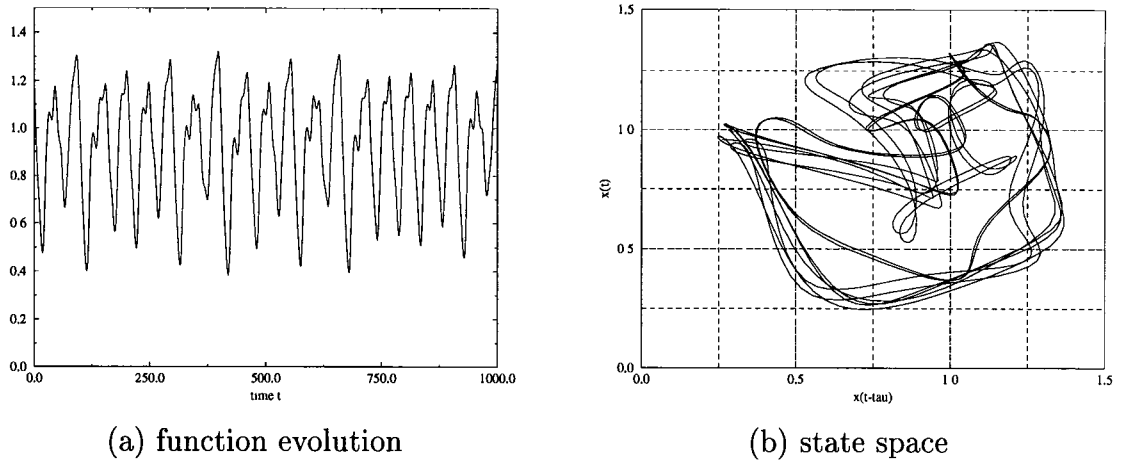


Figure 5.1: System variation of Mackey-Glass equation from $t = 1$ to 1000: (a) function output (b) strange attractors on state space.

can be divided into two areas: network dynamics and learning. Introduction of delays may push the net from fixed point state space to limit cycles or increase the duration of the transient. In learning, the network parameters such as weights and time delays are affected. Furthermore, delays may increase the range of the network's dynamical capabilities. Pinda, Peralmutter, Williams and Zipser, and Baldi have suggested that delays should be considered as an additional media or skill to achieve particular dynamical effects and optimal performance in recurrent nets [111, 110, 163, 8]. Our purpose is to examine how delays affect the *ATNN* and how delays can be tuned to achieve optimal performance.

To show the effects of changing time-delays on system performance, we have applied a time-delay neural network with fixed time-delays (*TDNN*) to the prediction of the Mackey-Glass delay differential equation.

The schematic architecture of the *TDNN* used in this study is elicited in Figure 5.2. Output from each neuron in the network is given by a single line,

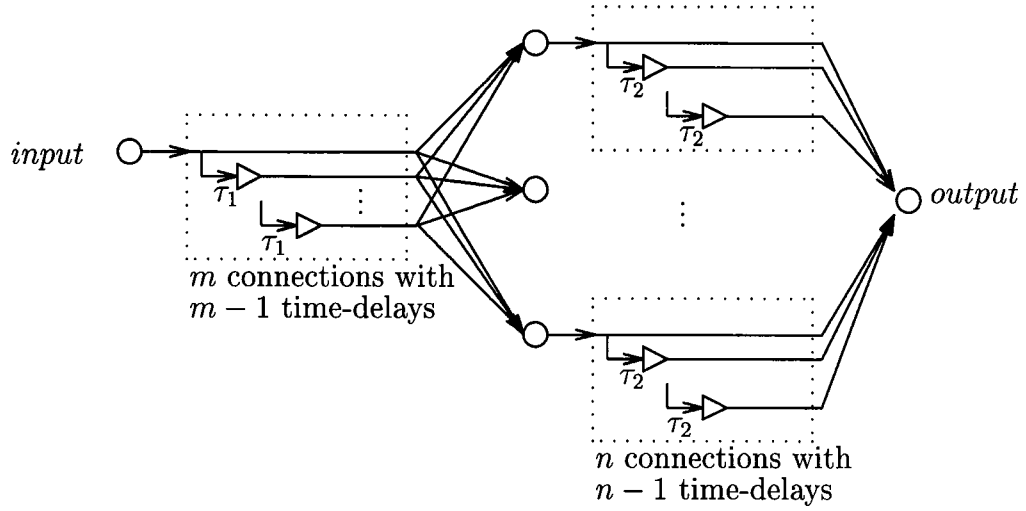


Figure 5.2: *TDNN* configuration for prediction

which is then subject to delays of various lengths. These various delayed signals are then connected as inputs to the neurons of the second layer and again, each unit in the second layer is connected in the same manner to the next layer. The synapse weights are updated by an error backpropagation algorithm until the *RMSE* goes below a threshold criteria [29]. The detailed description of the activation rule and learning rule for the *TDNN* can be found in [86].

In our simulation, we used one input unit, three hidden units, and one output unit. Experiments were conducted by changing the number of delays on each layer (m and n in Figure 5.2) and the values of the delays (τ_1 and τ_2 , indicated in Figure 5.2). For simplicity, we kept all delays in the same layer consistent (e.g., each neuron's output signals were subject to delays of $0, \tau_i, 2\tau_i, \dots, (n-1)\tau_i$).

In the experiment, the observation of prediction performance and convergence speed is made with different values of time-delay τ_1 under three different network configurations (different values of m and n); different performance levels are

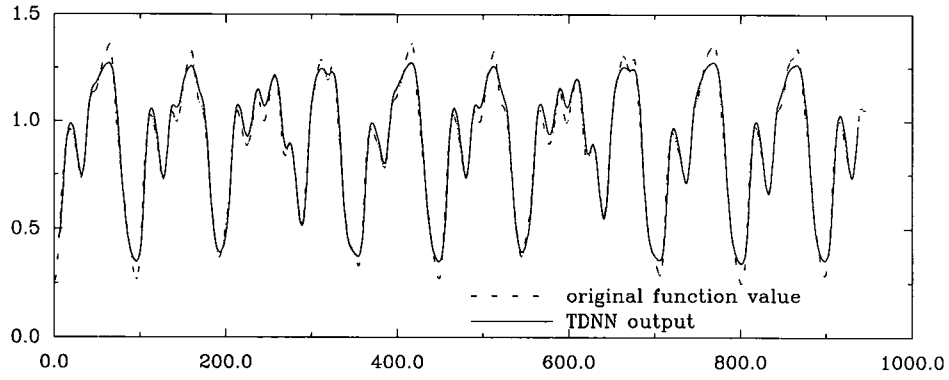


Figure 5.3: Simulation results, $\tau_1 = 1, \tau_2 = 1, m = 4, n = 8$

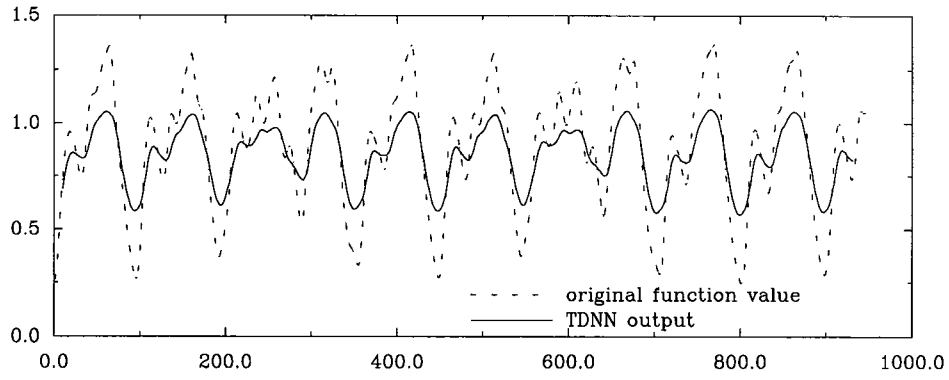


Figure 5.4: Simulation results, $\tau_1 = 4, \tau_2 = 1, m = 4, n = 8$

observed for different configurations. Two of the simulation results are shown in Figures 5.3 and 5.4, which give performance for differing values of τ_1 .

The *TDNN* performance was compared to the original function. The network output of two different trials was plotted to overlap with the system output in Figure 5.3 and 5.4, where solid lines and dashed lines stand for network output and the original function value respectively. In Figure 5.3, the network predictions match the function value relatively well, following the original function waveform closely except for some bumping peaks. In Figure 5.4, which utilizes different time delay values, the match is not very good although the overall trends are preserved.

5.4 ATNN for Prediction

5.4.1 Training Set Size and Performance

Preliminary results from the Adaptive Time-Delay Neural Network (*ATNN*) on system modeling and prediction applications have been shown to be promising [92]. The *ATNN* model employs modifiable time delays and synapse weights along the interconnections between two processing units, and both time delays and weights are adjusted according to system dynamics in an attempt to achieve the desired optimization. The mathematical formulas of the weight and time-delay modifications are described below:

$$\Delta w_{jik,h} \equiv -\eta_1 \frac{\partial E(t_n)}{\partial w_{jik,h}}$$
$$\Delta \tau_{jik,h} \equiv -\eta_2 \frac{\partial E(t_n)}{\partial \tau_{jik,h}},$$

where η_1 and η_2 are the learning rates. The training set consists of a sequence of time series and target outputs. The *ATNN* implements dynamic delays along the interconnections of an artificial neural network. We applied the *ATNN* architecture to the chaotic time series prediction of Mackey-Glass differential equation as shown in Equation 5.1. We set $a = 0.2$, $b = 0.1$, $c = 10$ and $\tau = 17$ and the initial value $x = 0.8$.

In our simulation, we use one input unit, three hidden units, and one output unit, four and six delay elements on the first and second connection layer respectively, which is a very economic configuration (in our notation $1 \xrightarrow{4} 3 \xrightarrow{6} 1$). In this configuration, the number of delay variables selected on this first connection layer is based on the chaotic series attractor dimension as discussed in Section 5.4.2, while the choice of delay element number is motivated by the intuition

training set size	epoch	$NRMSE$
200	125	0.1991
300	81	0.1991
400	64	0.1903
500	48	0.1997
600	36	0.2067
700	16	0.2078
800	11	0.2075

Table 5.1: Experiments of different configurations as the cutoff C_{RMSE} is set to 0.02, $T = 1$.

that higher level units should learn to make decision over a wider range in time based on more local abstractions at lower levels [152].

We explored the relationship between size of training set, number of epochs of training , training C_{RMSE} cutoff, and the performance during the system modeling task. In a series of experiments, the size of the training set was varied. The network was trained in batch mode (epochs), with a C_{RMSE} cutoff point based on performance on the training set. The performance was measured on the first 4000 time steps of the Mackey-Glass series. The error measure was reported as a normalized root-mean-square error² ($NRMSE$) over time of the network output and the desired prediction, e.g. predicting T steps ahead from observed information. This scheme is equivalent to resolving system modeling

² $NRMSE = \frac{(E[(\hat{x}(t+T)-x(t+T))^2])^{1/2}}{\sigma_x}$ where $x(t)$ is the original signal value and $\hat{x}(t)$ is the network prediction output, and σ_x denotes the standard deviation of the original time series.

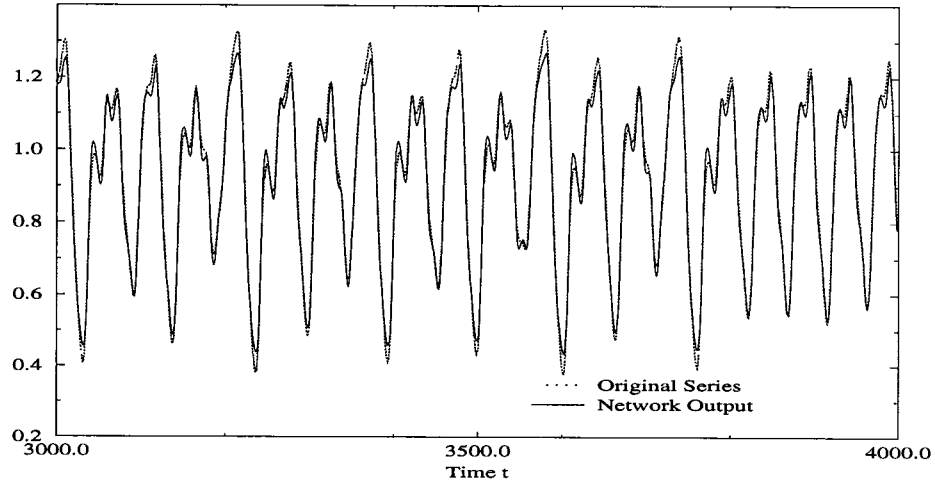


Figure 5.5: Simulation results of *ATNN* with the Mackey-Glass equation prediction as $T = 1$ and test on $t = 3000$ to 4000 sample data period.

of the Mackey-Glass equation when T is equal to one. Note that performance measure $NRMSE$, usually averaged over time, represents a perfect prediction when $NRMSE$ value is equal to zero. If $NRMSE$ approaches one, the performance is no better than a constant predictor. In Table 5.1, the training C_{RMSE} cutoff was 0.02. It shows a series of different training set sizes, and reports the number epochs of training needed in each case, and the performance on the 4000 sample test set in each case was denoted in $NRMSE$.

In principle, the larger the training set size is, the longer the network will take to compute on an epoch. Fortunately, from Tables 5.1, it is not necessary that the network spends a longer time to converge when a larger set of data is used for training. The number of epochs needed decreases as the training size increases. The performance of the network is excellent, as shown in Figure 5.5. The network output is plotted in the solid line with the dotted line plotting the original sampling data, these two plots overlap most of the time.

5.4.2 Resolving Embedding Dynamics

The state at time t of the Mackey-Glass differential equation is determined by the function $f(t) = \frac{dx}{dt}$ (Equation 5.1) at the interval $[t, t - \tau]$. However, the value τ is unknown when only the time series is observed. Then one needs to pick a reasonable delay time $\hat{\tau}$ and model the function according to the information between estimated interval $[t, t - \hat{\tau}]$. This function can be approximated by n samples taken at

$$\Delta\hat{\tau} = \frac{\hat{\tau}}{(n-1)}. \quad (5.2)$$

These n samples can be thought of as n variables of an n dimensional space vector

$$[x_1, x_2, \dots, x_n]^T = [x(t), x(t - \Delta\hat{\tau}), \dots, x(t - (n-1)\Delta\hat{\tau})]^T.$$

The prediction is accomplished by mapping this n dimensional vector space to a real value. The mapping $f : R^n \rightarrow R$ maps n measurements of the time series to the value at a future moment $x(t + T)$. Thus, the conventional prediction task was concentrating on finding a function such that $x(t + T) = f(x_1, x_2, \dots, x_n)$. The lacking of prior knowledge of choosing embedding n and $\hat{\tau}$ is the obstacle of conventional methods. These parameters are usually chosen by trial and error, although there have been suggestions for choosing n , e.g. $d_a < n < 2d_a + 1$, where d_a is the fractal dimension of the attractor [107]. If $\hat{\tau}$ were too small, the embedding would have incomplete information. If $\hat{\tau}$ were too large, then information is included that is farther in the past than needed. Farmer suggested to weight the delay coordinates before performing the function approximation, in other words, the state vector becomes: $[x(t), \exp^{-hz} x(t - \Delta\hat{\tau}), \dots, \exp^{-h(n-1)z} x(t - (n-1)\Delta\hat{\tau})]^T$, however, *a priori* knowledge is still needed to decide h and z [43].

The *ATNN* accomplishes the prediction adaptively in the manner different from the single mapping described above. It employs multiple mappings according to the following procedures in the three layered network case:

1. Distributes the information on the interval $[t, t - \hat{\tau}]$ into j vectors with m dimension, where j is the number of hidden units and m denotes the number of time delay variables employed on the connections between input processor and node j on the hidden layers. Each m dimensional space vector is transformed to a real value y_j by weight vector W_j and squashing function σ which associate with hidden node j .

$$\begin{aligned}
y_1(t) &= \sigma(W_1^T [x(t - \tau_{11,1}), x(t - \tau_{12,1}), \dots, x(t - \tau_{1m,1})]^T) \\
y_2(t) &= \sigma(W_2^T [x(t - \tau_{21,1}), x(t - \tau_{22,1}), \dots, x(t - \tau_{2m,1})]^T) \\
&\vdots \\
y_j(t) &= \sigma(W_j^T [x(t - \tau_{j1,1}), x(t - \tau_{j2,1}), \dots, x(t - \tau_{jm,1})]^T),
\end{aligned}$$

where m is the number of time-delays on the first connection layer.

2. These activation values from hidden units are again mapped to a real value which is the prediction target, in a similar manner as described above.

$$z(t + T) = \sigma(W_2^T [y_1(t - \tau_{11,2}), \dots, y_1(t - \tau_{1n,2}); \dots; y_j(t - \tau_{j1,2}), \dots, y_j(t - \tau_{jn,2})]^T),$$

where n is the number of time-delays on the second connection layer.

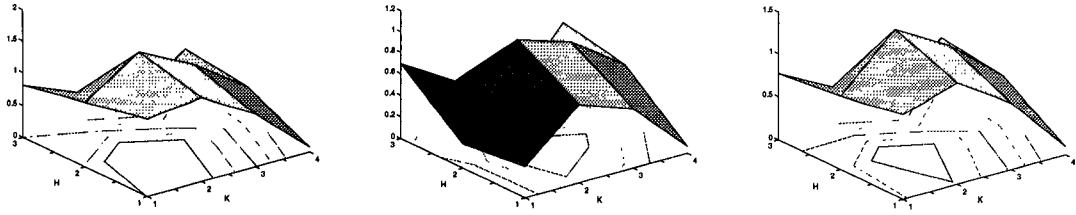
The time-delay variables and weight parameters of the above functions are adapted as the training proceeds. The appropriate value of $\hat{\tau}$ is obtained from the resulting time-delay parameters ($\hat{\tau} = \max(\tau_{jim,1}) + \dots + \max(\tau_{jim,h})$), and $\hat{\tau}$ does not have to be sampled at regular intervals as in Equation 5.2, which

gives the potential for better prediction. Therefore, the embedding dynamics is resolved automatically and efficiently from this network paradigm.

To proclaim the embedding which *ATNN* learned from Section 5.4.1, we display the time-delay matrix and its corresponding weight matrix for three training cases: (1) $T = 1, C_{RMSE} = 0.02$, (2) $T = 2, C_{RMSE} = 0.05$, and (3) $T = 2, C_{RMSE} = 0.03$. The resulting first layer time-delay matrices on all three cases are the same:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix} \quad (5.3)$$

The corresponding matrix for each case are elucidated in Figure 5.6. Perceptually, the shapes of these three matrices are similar but with different height. The weight distribution are similar. The time-delay and weight distribution on the second layer are presented in Figure 5.7 to Figure 5.9 for each case correspondingly. Figure 5.7 and Figure 5.8 are for case two but later one with lossy convergence criteria ($C_{RMSE} = 0.05$). Comparing Figure 5.7(a) and Figure 5.8(a), most of the time-delay elements are the same except ones in first connection line ($K = 1$ in the figure) from hidden unit two ($H = 2$ in the figure), and the weight matrices were obtained in different scale, though they with the similar shape. The particular time-delay element may be the key in effecting dynamics. Since that time-delay element got changed in an update, the network prediction performance improved (see results in Figure 5.11). In Figure 5.9, the amplitude of time-delay elements increased dramatically, though the weight matrix displays a similar shape as previous cases. To do prediction two step ahead may require more past observation as suggested by resulting time-delay elements.

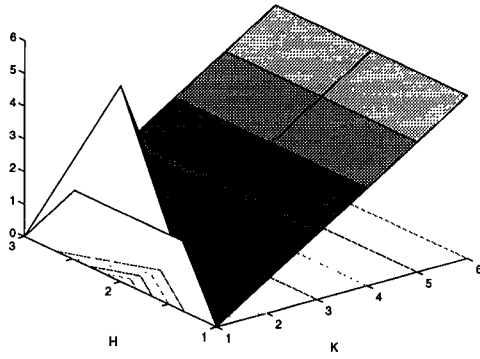


$$T = 1, C_{RMSE} = 0.02$$

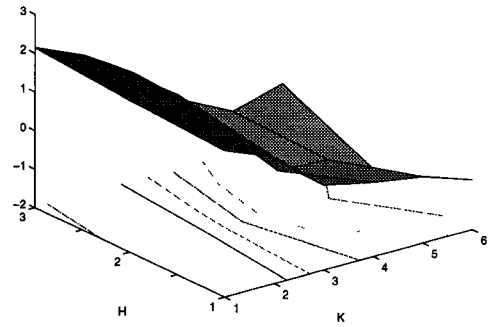
$$T = 1, C_{RMSE} = 0.05$$

$$T = 2, C_{RMSE} = 0.03$$

Figure 5.6: The resulting weight matrix of first layer trained with different criteria C_{RMSE} .

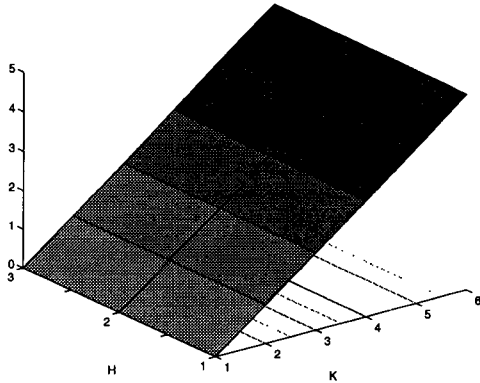


(a) Time-delay matrix

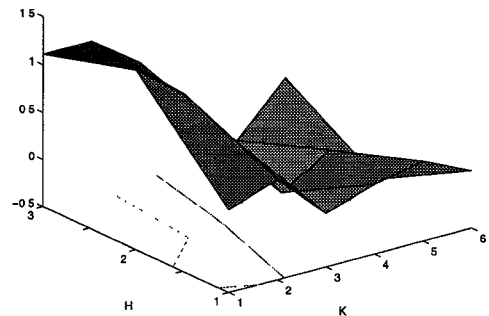


(b) Weight matrix

Figure 5.7: The resulting (a) time-delay matrix and (b) weight matrix of second layer trained with criteria $T = 1, C_{RMSE} = 0.02$.



(a) Time-delay matrix



(b) Weight matrix

Figure 5.8: The resulting (a) time-delay matrix and (b) weight matrix of second layer trained with criteria $T = 1, C_{RMSE} = 0.05$.

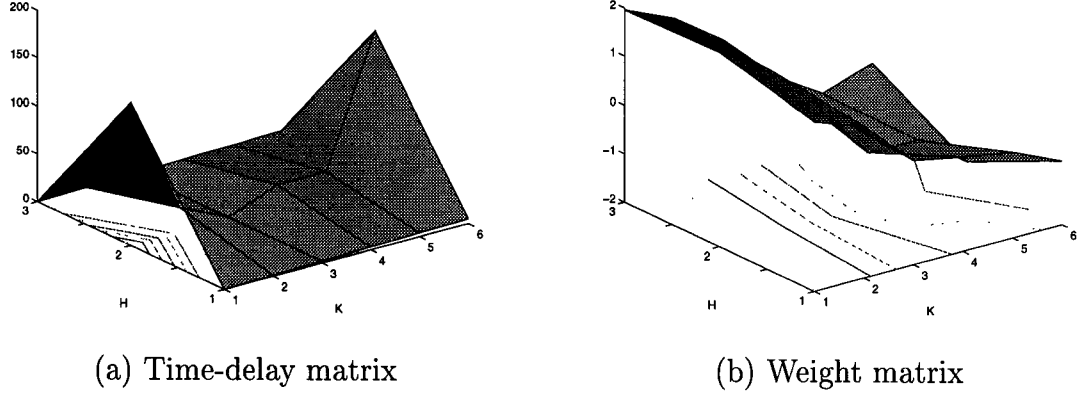


Figure 5.9: The resulting (a) time-delay matrix and (b) weight matrix of second layer trained with criteria $T = 2$, $C_{RMSE} = 0.03$.

5.4.3 Prediction Over Time

Usually, prediction error increases over time during the prediction, as in conventional methods without self adjustment. The neural network approach shows advantages. Lapedes's simulation showed that the error measure of conventional methods grow exponentially, while the performance of the neural approach (back-propagation net) continues in a more promising way and is superior to the other traditional techniques, as illustrated in Figure 5.10. Figure 5.10 shows $NRMSE$ as a function of prediction time.

In our experimentation, prediction was made iteratively step by step by recursively using the new predicted value to march along further. The results are shown in Figure 5.11 for various network configurations in which the solid black line A is the result from a network trained on $\hat{x}(t, T = 2)$ with cutoff criteria 0.03. The gray dotted line B and dashed line C are the network performance from $\hat{x}(t, T = 1)$ training with $RMSE$ cutoff at 0.02 and 0.05 respectively. Our results showed that the performance of $ATNN$ prediction on this chaotic time series confirms the high performance of neural networks as suggested by Lapedes.

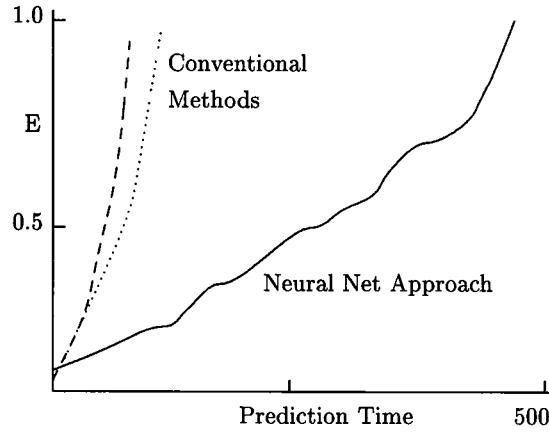


Figure 5.10: Performance comparison of neural net and conventional methods: relative quantity diagram. Adopted from Lapedes and Farber [83].

Figure 5.11 demonstrates that when $T = 2$, the network gets better prediction within 300 time steps, while in the long run, the network maintains a more consistent performance for $T = 1$. The performance is better for $T = 1$ and $NRMSE$ does not drift upwards over time.

5.4.4 Noise Tolerance

An important property of artificial neural networks is noise tolerance. In other words, one of the main performance measures of a network is its ability to respond properly to new data that is similar to training patterns but contains noise. This reflects how well the network generalizes to the target patterns, especially for temporal sequences.

A number of experiments were done to investigate how well the proposed system is resilient to noise. Different variance of random noise with normal distribution and zero mean was added to the testing signal and tested on the *ATNN* trained from the previous section. The results are shown in Figure 5.12.

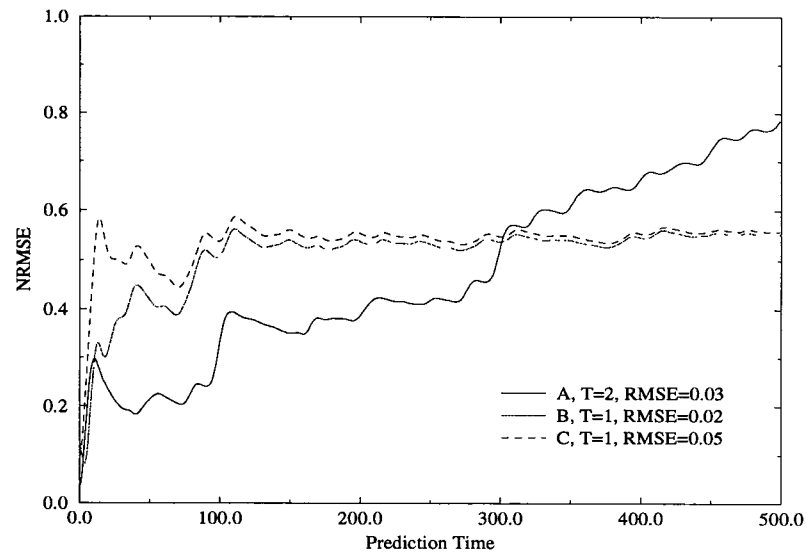


Figure 5.11: Prediction performance of ATNN on different network configuration with 400 training data.

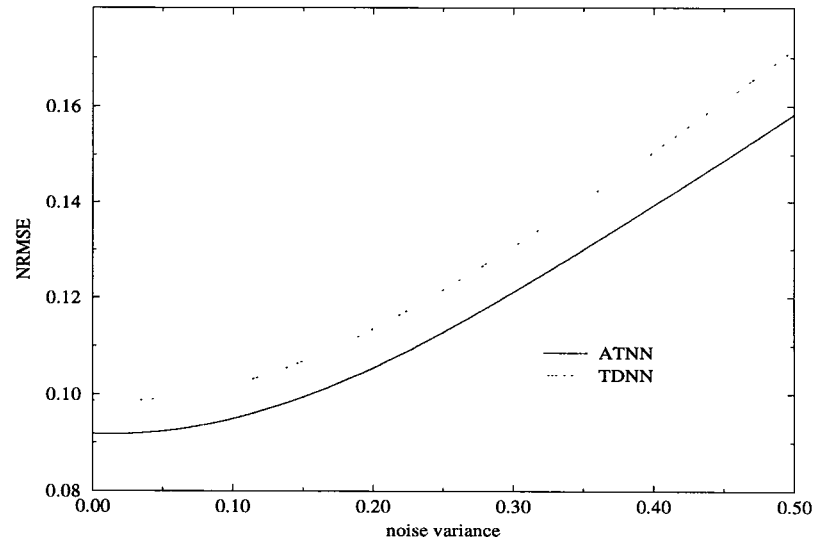


Figure 5.12: System performance comparison of different variance of noise added to input patterns

The system maintains the performance until the noise variance (σ_n^2) gets to 0.05 (where the input signal-to-noise ratio³ $SNR = 12.388dB$). The system can keep the $NRMSE$ as low as 0.11 even when σ_n^2 is increased up to 0.2. As expected, the performance of the system degrades slowly as noise variance is increased. A comparison of the $ATNN$ to the $TDNN$ was made in the same diagram (Figure 5.12) with the same network complexity. It is clear that the $ATNN$ is more resilient to noise than the $TDNN$, especially in the presence of larger interference by noise.

5.5 Time Series Modeling

The objective of time series modeling is to find a description that accurately captures features of the long-term behavior of the system. We demonstrate $ATNN$'s ability in time series modeling with the Mackey-Glass equation and harmonic wave forms. Network performance among various related network is also made.

5.5.1 Chaotic Time Series Modeling

Recently, considerable effort has been devoted to finding the embedded dynamics in input sequences, or to capturing the characteristics of chaotic environments, using neural network techniques [28, 123, 55, 105, 135, 147, 164]. The objective of the system modeling problem is to construct a suitable embedded identification function $\hat{\mathcal{P}}$, for a given plant $\mathcal{P}$ and its input-output pair $\{< x_1, \dots, x_m >, <$

³signal-to-noise ratio $SNR = 10 \log \frac{\int_0^T x^2(t) dt}{\sigma_n^2} (dB)$, where $x(t)$ denotes the function value at time t , and σ_n is the standard deviation of noise

$y_1, \dots, y_n\}$ (where m and n denotes the dimension of input and output vectors correspondingly), such that $\|\hat{y} - y\| = \|\hat{\mathcal{P}}(x) - \mathcal{P}(x)\| \leq \epsilon$ for some desired $\epsilon \geq 0$.

A chaotic time series (based on the Mackey-Glass delay differential equation) was selected for experimentation with the *ATNN*. The delay differential equation of the form has been described above.

A three layered *ATNN*, with the complexity of one unit in the input layer, three hidden units, one output unit, four and six time-delay elements on the first and second connection layers respectively, was simulated to model chaotic time series of the Mackey-Glass differential equation. The network was trained to resolve values of $\dot{x}$. The training set consisted of the first three hundred samples (from $t = 1$ to $t = 300$), and the testing set included ten thousand samples of the differential equation (from $t = 1$ to $t = 10000$).

The first 1000 time step outputs and last 1000 time step outputs (from 9000 to 10000) of the network is shown in Figure 5.13 and Figure 5.14. The solid line and dotted line represent the original equation value and the output of the *ATNN* respectively. As we can observe from these two figures, the network follows the original function, and catches the variation well.

To evaluate performance, the network was trained by various choices of adaptive parameters, and the *NRMSE* measures of the corresponding system outputs are shown in Table 5.2. When weights are adapted but no time delays are included (equivalent to *BP*), the *NRMSE* was 0.1122. When time delays were included but were fixed and weights were adapted (e.g. a *TDNN*), then the *NRMSE* was considerably improved at 0.0985. If time delays were adapted but weights were fixed, the worst performance was attained at *NRMSE* = 0.1568. When both weights and time delays were adapted with the *ATNN* paradigm,

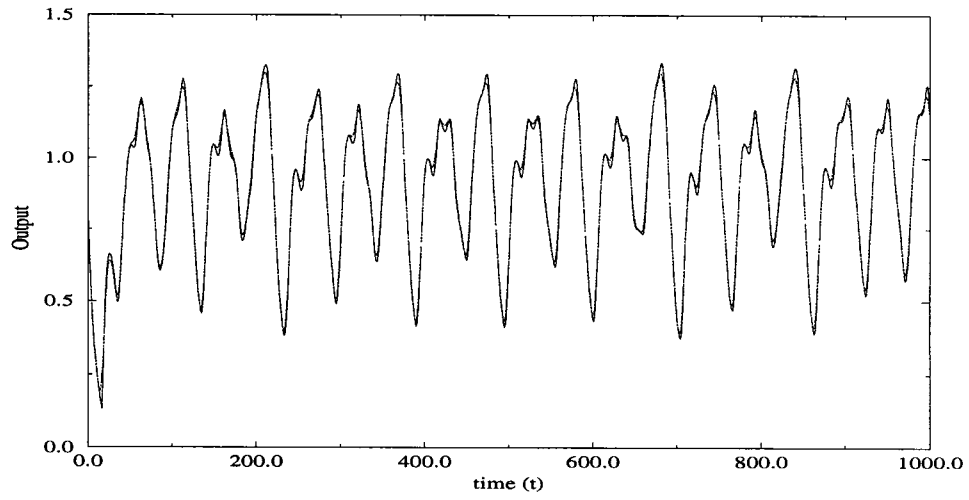


Figure 5.13: Simulation results of *ATNN* with Mackey-Glass equation modeling $t=0$ to 1000, $NRMSE = 0.045$

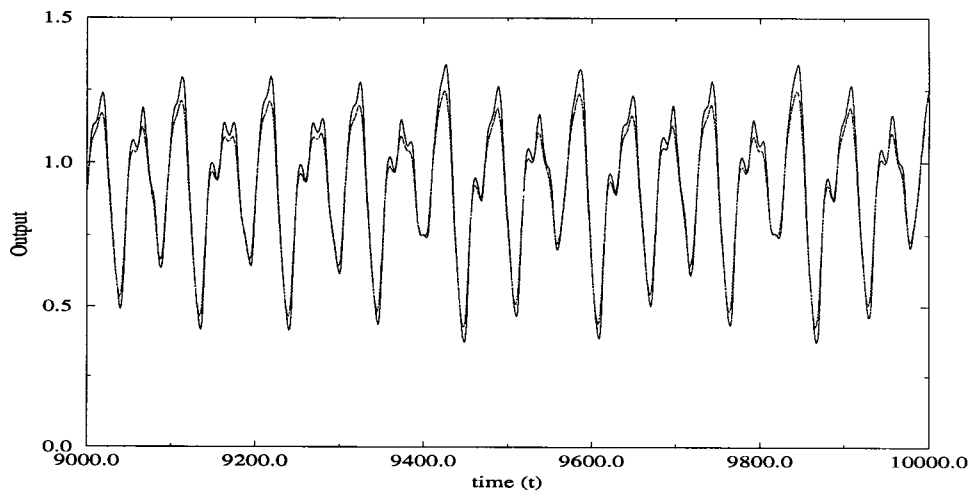


Figure 5.14: Simulation results of *ATNN* with Mackey-Glass equation modeling, $t=9000$ to 10000, $NRMSE = 0.063$

adaptive parameters	fixed parameters	equivalence	NRMSE
weights	$K_{ji,h-1} = 1, \tau_{jik,h-1} = 0$	<i>BP</i>	0.1122
weights	time-delays	<i>TDNN</i>	0.0985
time-delays	weights	-	0.1568
weights, time-delays	-	<i>ATNN</i>	0.0548

Table 5.2: Experiments with different network configurations trained to predict the Mackey-Glass equation. Each table entry is the average of three runs.

the best performance was attained at $NRMSE = 0.0548$. The results indicated that the adaptive time delays improve the network performance.

5.5.2 Harmonic Wave Form Modeling

With the help of delays, one could construct a bank of similar harmonic models or oscillators, but with different frequencies within a relatively broad spectrum. These could in turn be used for the analysis of general functions or trajectories, using a Fourier type of decomposition [7].

In this experiment, the *ATNN* is trained to model the square wave approximation from the Fourier series expansion of the sum of different odd harmonics. The class of functions which can be represented by Fourier series is surprisingly large and general. The values of various series with constant terms or periodic functions can be obtained by evaluating Fourier series at specific points [82]. If the network can capture the fundamental harmonic characteristics of various waves, then neural networks will lead to a novel approach to describe the representation of signals. Five different wave approximations are used in this study: a

	Wave with the sum of odd harmonics				
	fundamental	1st to 3rd	1st to 5th	1st to 7th	1st to 9th
NRMSE	0.0305	0.0298	0.0304	0.0316	0.0346

Table 5.3: The error measure *NRMSE* of this system output under different approximation to a square wave

fundamental sine wave or first harmonic, a wave with the sum of first and third harmonics ($y = \sin(t) + \frac{\sin(3*t)}{3}$), a wave with the sum of first to fifth harmonics ($y = \sin(t) + \frac{\sin(3*t)}{3} + \frac{\sin(5*t)}{5}$), a signal with the combination of first to seventh harmonics (i.e. $y = \sin(t) + \frac{\sin(3*t)}{3} + \frac{\sin(5*t)}{5} + \frac{\sin(7*t)}{7}$), and one with the sum of first to ninth harmonics (i.e. $y = \sin(t) + \frac{\sin(3*t)}{3} + \frac{\sin(5*t)}{5} + \frac{\sin(7*t)}{7} + \frac{\sin(9*t)}{9}$). A three layered *ATNN* with one input unit, three hidden units and one output unit was trained to identify these particular wave forms according to the adaptive learning paradigm.

Three of the simulation results are illustrated in Figures 5.15 to 5.17. The original functions are plotted in solid lines, and the dotted lines denote the output of network. The system performance is so good that the solid line and dotted line almost overlap and show high accuracy. The error measure of all wave form tests are shown in Table 5.3. The error measure *NRMSE* is lower than 3.05% in the first three cases, while the *NRMSE* is 3.46% which is still small in the complicated case of the sum of the 1st to 9th odd harmonics. This indicates that the network has learned the various complexities with similar accuracy.

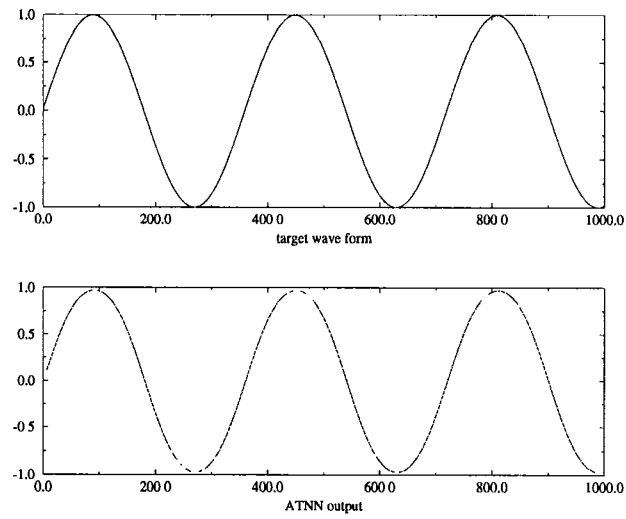


Figure 5.15: Simulation of identifying the fundamental wave: first harmonic.

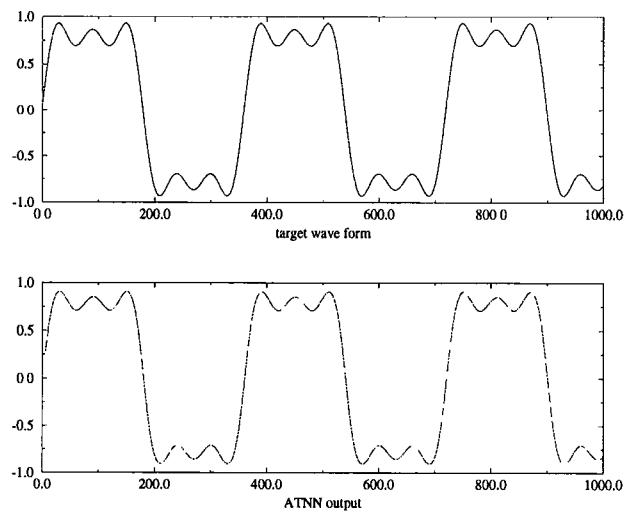


Figure 5.16: Simulation of identifying the wave with a sum of 1st to 5th add harmonics

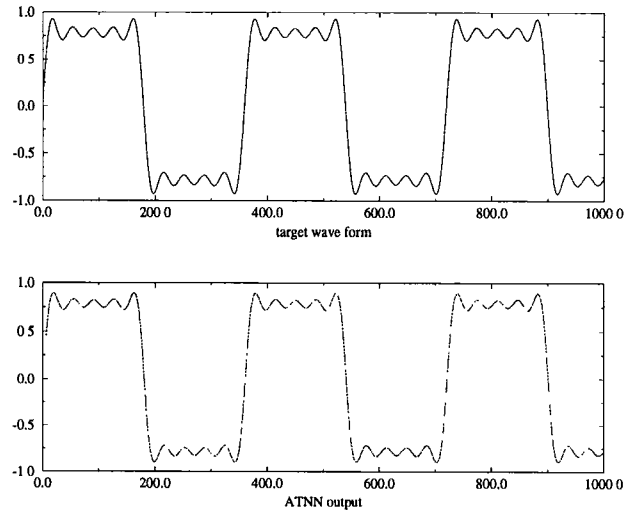


Figure 5.17: Simulation of identifying the wave with a sum of 1st to 9th add harmonics

5.6 Conclusion

In this chapter we have shown that a prediction approach based on *ATNN* technology is effective and accurate. The beauty of this work is to show this complicated problem, based on the Mackey-Glass time series prediction, can be accomplished by such a simple *ATNN* architecture, while training can be done with reasonable amounts of data and computing time. Furthermore, the embedding of the system dynamics can be resolved adaptively, unlike exhausting trial and error efforts in conventional methods. We can conclude from this study that the *ATNN* is a good tool to tackle temporally changing signal processing and prediction problems. The simulation results confirm the performance improvements of the neural network approach, as suggested by Lapedes and Farber [83].

The network accomplishes the identification of background model of Mackey-Glass delay differential equation again with high accuracy ($NRMSE = 0.0548$). Other results show that the *ATNN* successfully models harmonic waves and does

accurate time series predictions of a chaotic sequence. These applications show that the *ATNN* is a powerful tool and learning technique for spatiotemporal data and for any applications that involve time varying signals or patterns. It also explores the harmonic wave forms with a very low *NRMSE* of about 0.03.

This dynamic architecture was shown to be useful and promising for applications that involve time varying signals or patterns. It is also very promising in biomedical prediction tasks [34, 33]. Adaptive time delay neural networks can be built in modular groups (*MATNNs*), in which each module is capable to perform a particular task. Modular *ATNNs* can address classification and recognition of trajectories from moving targets, handwritten characters, motion visual images, and other sensors data patterns.

Chapter 6

A Real World Application: Target Discrimination

6.1 Automatic Target Recognition

Automatic Target Recognition (*ATR*) is a very challenging problem. It involves extraction and discrimination of critical information from complex and uncertain data. Due to target signature variability, environmental changes, lack of acquisition and representation, or limited data base, the traditional approaches of signal processing and rule-based expert systems have been only partially successful but have produced high false-alarm rates [127]. Neural net technology offers a number of tools such as learning and adaptation, generalization and robustness, feature extraction and hardware implementation, which could form basis for a potential fruitful approach to the *ATR* problem [51, 127, 6].

A real world application with *ATNN* is discussed in this chapter for exo-atmospheric target discrimination. The scenario of this problem is schemati-

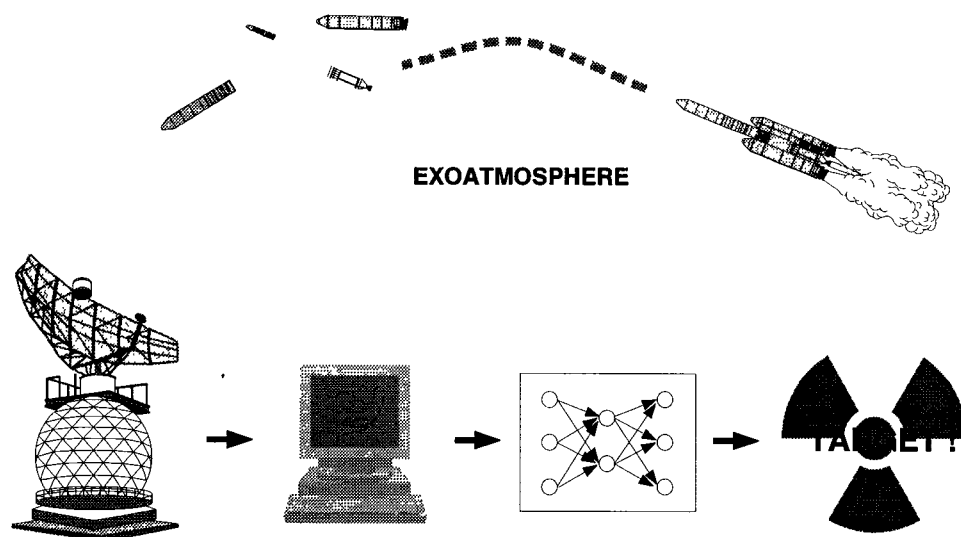


Figure 6.1: Target components discrimination.

cally illustrated in Figure 6.1. The components of a vehicle will be separated and detected by a visual sensor after it is launched. The visual sensor records intensity and size over time for each component. Among these components, we are only interested in the target component, the other pieces can be classified as non-target components. The objective is to discriminate between these components disregarding different aspect angles, tumbling or broken pieces and find the target component.

This is a very important and yet difficult task in target component discrimination, as it states in *APL* technical report by Resch [121], "The ability of a kinetic kill vehicle (KKV) to discriminate between warhead and booster parts or decoys is critical to theater ballistic target defense (TBMD). Exo-atmospheric targets are especially difficult to distinguish using currently available techniques because all target parts follow the same trajectory during the exo-atmospheric portion of the flight."

The sensors used in this analysis has dedicated specifications. The maximum

radiance and size versus time are obtained from these sensors for all four components (denoted in this context as component 1, 2, 3, and 4, in which component 1 is the target we are interested in). Furthermore, these components may tumble with different aspect angles or break into several pieces. One of the many things that makes *ATR* so hard is that the same target can vary wildly in appearance depending aspect angles, atmospheric effects, and other variables.

We address this problem by using both *TDNN* and *ATNN* and conclude that *ATNN* is more promising to this problem.

6.2 TDNN Technique

6.2.1 Simulation Setups

The *TDNN* is used to perform the discrimination task at the first place. This network contains a input layer with six inputs, one hidden layer with three hidden units and an output layer with two units. The time-delay on first layer is four and on the second layer is six ($6 \xrightarrow{4} 3 \xrightarrow{6} 2$). Time-delays on each pair of connections between neurons are fixed and are consecutively and equally spaced as $0, \tau, 2\tau, \dots$, where τ is the data sampling interval. Three of the inputs are the ratios of a piece's highest radiance to the highest radiance of the other pieces. The other three inputs are the ratios of a piece's size to that of the other pieces. Data is input to the network sequentially starting at certain seconds before intercept and ending very shortly before intercept. The target value is one or zero representing either true or false targets for each particular training set composed of the ratios described above. The following techniques were used on this data discrimination problem:

1. A symmetric sigmoid function was used on all processing units. This appears to be advantageous in convergence and recognition performance.
2. The data scaling procedure was revised so that the same scaling factor F_s was used on all data. The data was scaled and offset to be symmetric at 0 with range $[-0.5:0.5]$. The scaling factor F_s was calculated from the training set by the following equation (Equation 6.1:

$$F_s = \max(\mathcal{S}_{train}) - \min(\mathcal{S}_{train}) \quad (6.1)$$

where $\mathcal{S}_{train}$ stands for the training data space.

The training data is scaled and offset to be symmetric to 0 by Equation 6.2:

$$network\ input = \frac{(\mathcal{S}_{train} - p_{mid})}{F_s} \quad (6.2)$$

where $p_{mid} = \min(\mathcal{S}_{train}) + F_s/2$.

6.2.2 Simulation Scheduling and Analysis

Two training schedules were used and tested on five different test sets. Each test set contains more than 24 trajectories in time. These schedules are described as below:

Schedule 1: This training schedule was to begin adapting weights after the first segment of data entered the network [121]. However, we did not allow data from a previous run to remain in the network at the same time that data from a new run entered the network.

Schedule 2: A new training schedule was used in which data filled the network before weight adaptation began. Performance improved with this training

data set	Schedule 2	Schedule 1
training set	100%	89.61%
test set#1	100%	89.14%
test set#2	59.4%	53.18%
test set#3	59.4%	50.00%
test set#4	93.4%	81.82%
test set#5	97.9%	96.59%

Table 6.1: Successful identification rate of different test sets by using *TDNN* trained on 0.5 second increment data.

Overall	Schedule 2	Schedule 1
iterations	5565	14323
RMSE	0.044	0.105

Table 6.2: Over all performance of *TDNN* trained on 0.5 second increment data.

schedule as shown in Table 6.1. Fewer iterations of training were required (only 5500 as compared with 14,300 shown in Table 6.2).

For both training schedules, there was considerably less possibility of confusion of components 3 or 4 with target as compared with the previous study [121]. Usually the fuel tank and fins were clearly identified as not being the target very early. This is an improvement on previous results.

We also used various numbers of hidden units to see if performance will improve. The *TDNN* was trained to perform target discrimination with the

data set	$3h'$	$5h'$	$3h$	$3h''$
#1	89.61%	89.61%	89.61%	85.61%
#2	89.14%	88.89%	89.14%	88.13%
#3	66.36%	65.00%	53.18%	64.09%
#4	72.72%	70.45%	50.00%	72.73%
#5	82.72%	79.55%	81.82%	83.64%
#6	96.59%	94.89%	96.59%	89.77%

Table 6.3: Comparison of discrimination performance with different data sets, various training methods and different network topologies.

new scaling procedure (Schedule 2). Table 6.3 summarizes its performance for three hidden units ($3h'$) and five hidden units ($5h'$), as compared to our previous results labeled $3h$ (with three hidden units). Performance was better for the third, fourth, and fifth data sets. The results of each test data with *TDNN* trained on three hidden unit are shown in Appendix C.1 Figures C.1 to C.24.

Another experiment was performed to include additional data in the training set, with different aspect angles for different components. Previous training experiments were limited to data with the same aspect angle for different components. Performance improved on the third, fourth, and fifth data sets compared to the benchmark $3h$ run, but only the fifth data set showed an improvement over other runs with the revised scaling procedure. Performance for this experiment is labeled $3h''$ in Table 6.3. As a whole, the network figured out the target in all cases. The results show, the smaller the aspect angle the warhead, the more difficult time the network had in distinguishing the warhead as the target.

6.2.3 Improvement on Higher Sampled Data

The hypothesis of this test was whether faster time sampling improved recognition of target versus non-target parts for the more difficult data where different parts were at different aspect angles and some parts were broken into pieces. The data considered here were in two different time-samples: every half second and every tenth second.

When using tenth second data, the time delays at first layer were (0.0, 0.1, 0.2, 0.3) seconds, and with half second data they were (0.0, 0.5, 1.0, 1.5).

We obtained 100 % performance on the faster sampling data. Identification occurred much faster than on the previous data set with half second timing. In all cases, with the new tenth second sample data, the activation becomes high rapidly for the target and low rapidly for the non-target runs.

For comparison, Figures 6.2 to 6.9 show some of the simulation results on the same scenarios but with two different sampling intervals. On the left hand side are the results from the 0.1 second sampling interval (Figures 6.2, 6.4, 6.6, 6.8). On the right hand side are the results of the 0.5 second interval data (Figures 6.3, 6.5, 6.7, 6.9). More results of different aspect angles, tumbling angles and broken pieces are shown in Appendix C.2 Figures C.25 to C.32.

All figures from tenth second interval data show quick and correct identification. Several of the scenarios with half second interval data had some confusion, at least at first, in the identity of the target parts on half second signal data. But with the tenth second data, this confusion was eliminated. These cases are shown by comparing Figure 6.4 to 6.5, Figure 6.6 to 6.7, and Figure 6.8 to 6.9. Thus the new network that depends on tenth second time sampling overcomes limitations in the previous network and data. We conclude that in the

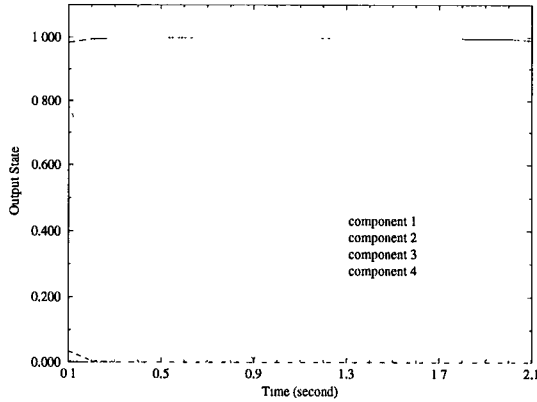


Figure 6.2: Recognizing target with aspect angle 30° , sampling time interval 0.1 sec.

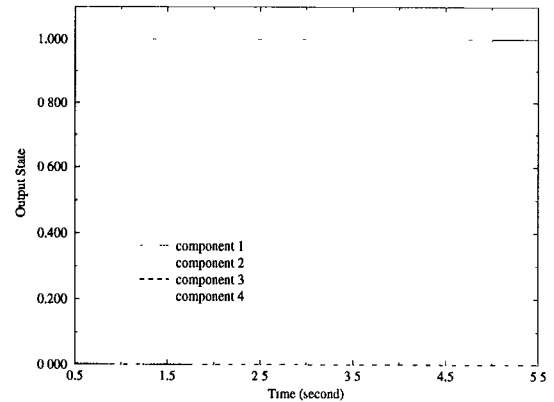


Figure 6.3: Recognizing target with aspect angle 30° , sampling time interval 0.5 sec.

application of the time-delay neural network, time sampling of every tenth second significantly improves performance in terms of speed of identification and percentage of correct recognition.

6.2.4 Number of Time Steps vs. Sampling Rate

In previous sections, we have shown that data sampled at tenth second intervals gave superior performance to data sampled at half second intervals. We have explored the reason for this difference, and tested the hypothesis that the total number of samples influences the performance. There was twice as many samples in the tenth second data set as in the half second data sets (the duration of tenth second data set is two second, while duration of the latter is five second).

We trained the network on only half of the time samples from the tenth second data. The simulation results in each various situation are shown in Appendix C.3 Figure C.33 to Figure C.44. The resulting performance was still 100%, and the

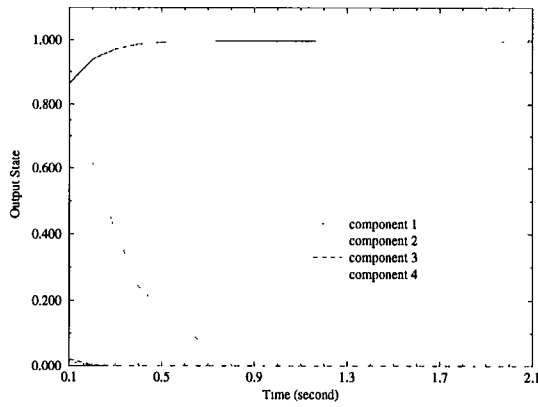


Figure 6.4: Recognizing target with aspect angle 60° , sampling time interval 0.1 sec.

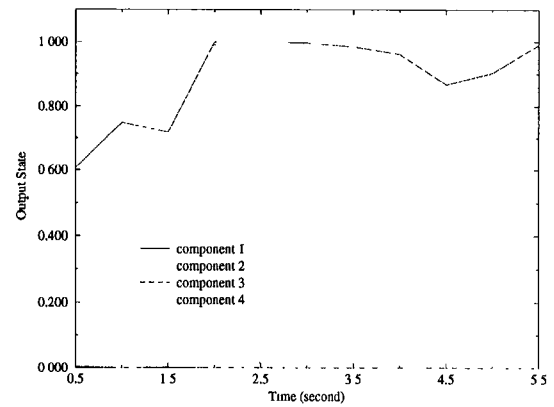


Figure 6.5: Recognizing target with aspect angle 60° , sampling time interval 0.5 sec.

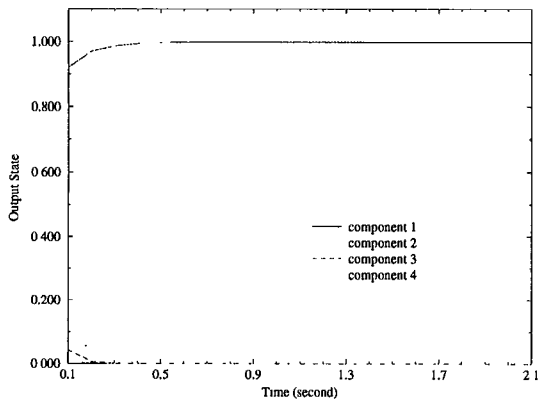


Figure 6.6: Recognizing target with aspect angle 90° , components 1 to 4 tumble at 90° , 60° , 60° , and 30° respectively, sampling time interval 0.1 sec.

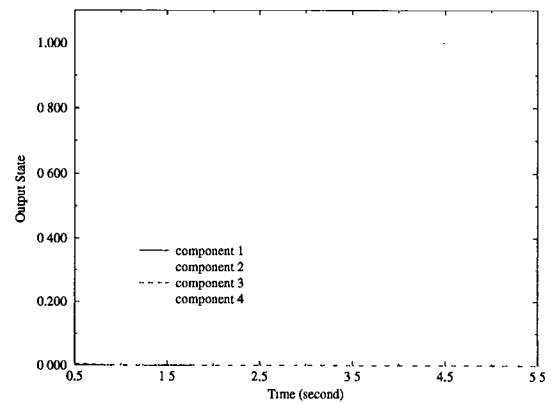


Figure 6.7: Recognizing target with aspect angle 90° , components 1 to 4 tumble at 90° , 60° , 60° , and 30° respectively, sampling time interval 0.5 sec.

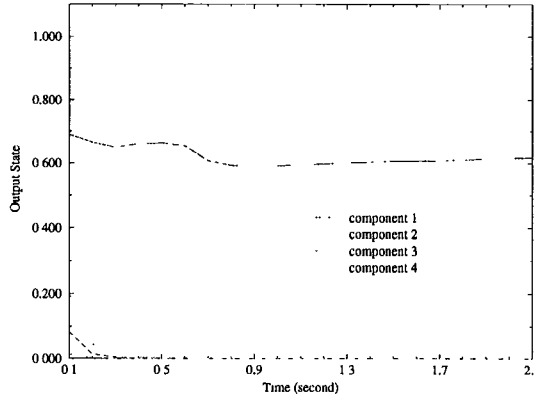


Figure 6.8: Recognizing target with aspect angle 90° , components 1 to 4 tumble at 30° , 60° , 90° , and 30° respectively, sampling time interval 0.1 sec.

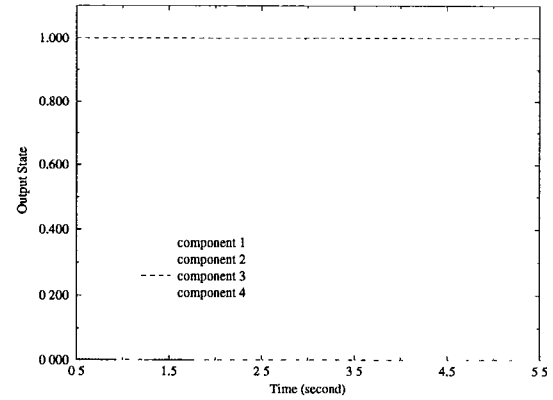


Figure 6.9: Recognizing target with aspect angle 90° , components 1 to 4 tumble at 30° , 60° , 90° , and 30° respectively, sampling time interval 0.5 sec.

tenth second data still performed in a superior fashion compared to the half second data.

The conclusion is that the number of time samples is **not** the cause of superior performance in the tenth second data. One of the reasons is that the radiance data used in analysis of 0.1 second increments is much smoother than that used in 0.5 second increments data. Furthermore, after seven seconds until intercept, the pieces begin to take up more than one pixel of the 512 by 512 grid sensors. This cause a change in slope in the data when changing from one pixel to two pixels. While in this 0.1 second increments data, it does not have these changes. The *TDNN* in this analysis performs well due to the data is smoother and the sensor stays locked on one pixel. A more realistic two pixel data will be analyzed in next section.

6.3 ATNN: a Superior Approach

6.3.1 Earlier Discrimination on More Realistic Sensor Data

In Section 6.2.3, we have shown that data sampled at tenth second intervals gave superior performance to data sampled at half second intervals with the *TDNN*. We applied the *ATNN* to similar data, sampled at tenth second intervals. Furthermore, we needed a longer data scenario to allow more possibilities for time-delays internal to the network. We applied both the *ATNN* and *TDNN* to this long data. The performance of the *ATNN* was superior to that of the *TDNN*. The reason for this improvement was that the *ATNN* has the ability to adapt time-delays in addition to weights, whereas the *TDNN* only adapts weights (and leaves time-delays fixed). Performance overall was very promising.

The data consisted of three target break-up scenarios, taken from 30°, 60°, and 90° aspect angles. The new data was for the case where the piece can move around on the sensor, so the resulting measurements are noisy. The piece can either be located on one pixel or located over two pixels (half of the piece lies in one pixel and half lies in an adjacent pixel). This is more realistic than assuming the sensor can lock the piece onto one pixel at all times [122]. Thus, the data the neural network is trained and tested on randomly jumps up and down from one time step to the next. This is more realistic than the smooth data we worked with previously. The noise made the recognition problem more challenging for the network.

The *TDNN* was initially used on this bumpy data and the result was not as good as before (with the smooth data). The *TDNN* results on the new data are

shown in Figures 6.11, 6.13, and 6.15. *TDNN* is not capable of resolving two pixel data with slopes changes. The *ATNN* was then trained on the new data and tested. The *ATNN* provided more flexibility as time-delay elements are adapted in the *ATNN* according to the characteristics of the data, and better time-delay values are chosen. Several time delays were increased to be over five time steps in the second layer time-delays. The resulting time-delay matrix on the first layer and second layer are shown in Appendix C Section C.4.

The *ATNN* achieved 100% correct recognition for all three aspect angles. When the networks were given the entire time to recognize the parts, and the recognition was performed at the end of the data stream, then the *ATNN* performed with 100% correct recognition whereas the *TDNN* had only 33% recognition at end of the data stream.

It is important to consider a scenario where the network is required to identify the parts before the end of the data stream. Thus we computed a stepwise performance percentage, which consists of the percent correct recognition averaged over all possible time steps in the data. On this basis, the *ATNN* achieved a 99.46% correct recognition rate. This means that if the *ATNN* were requested to make a decision at any time during the incoming data stream, it would have been correct 99% of the time. Early identification is made possible with *ATNN*. The *ATNN* performance was far superior to that of the *TDNN*, and was far more resilient to noise in the data. For comparison, the *ATNN* results (Figures 6.10, 6.12, and 6.14) are shown to the left of *TDNN*'s results (Figures 6.11, 6.13, and 6.15) on the same observation aspect angles correspondingly.

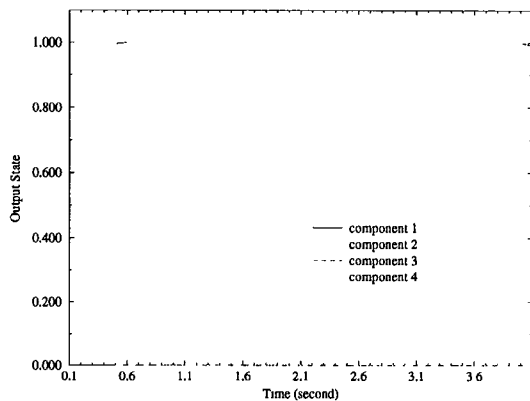


Figure 6.10: *ATNN* performance on bumpy data with aspect angle 90° , sampling time interval 0.1 sec.

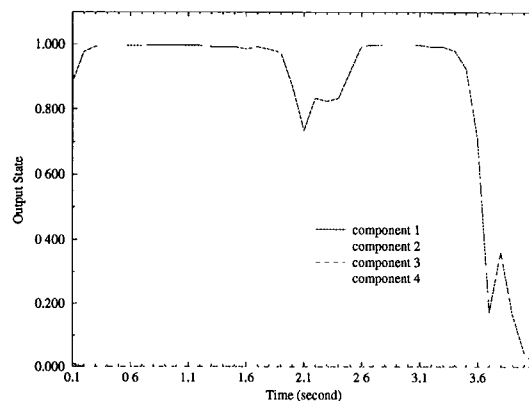


Figure 6.11: *TDNN* performance on bumpy data with aspect angle 90° , sampling time interval 0.1 sec.

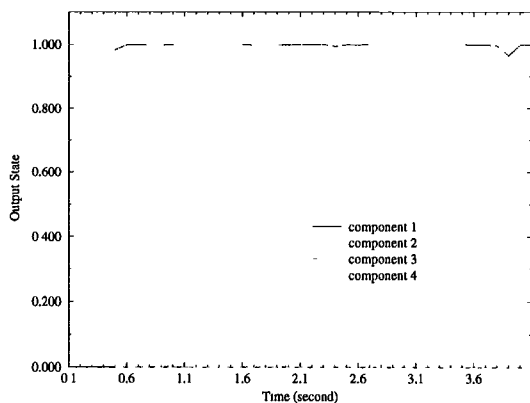


Figure 6.12: *ATNN* performance on bumpy data with aspect angle 60° , sampling time interval 0.1 sec.

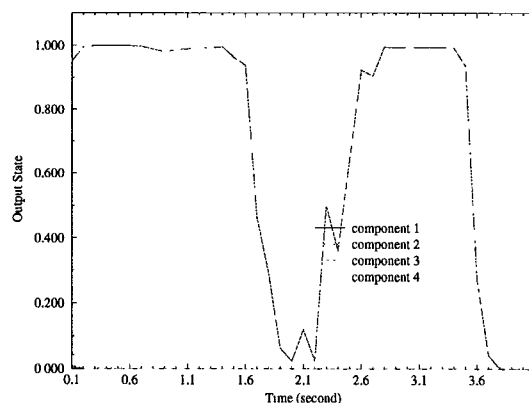


Figure 6.13: *TDNN* performance on bumpy data with aspect angle 60° , sampling time interval 0.1 sec.

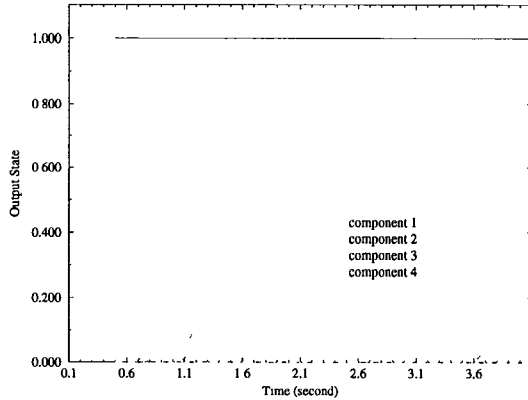


Figure 6.14: *ATNN* performance on bumpy data with aspect angle 30° , sampling time interval 0.1 sec.

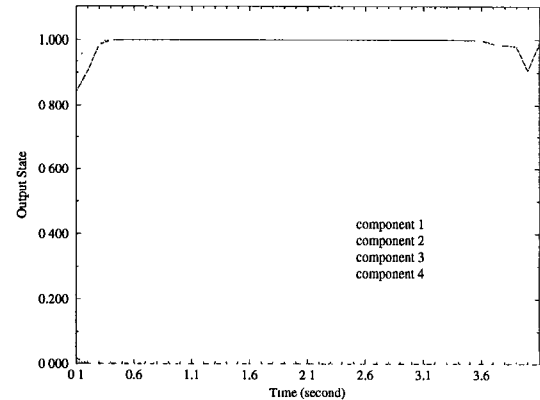


Figure 6.15: *TDNN* performance on bumpy data with aspect angle 30° , sampling time interval 0.1 sec.

6.3.2 Lemma Support

In Section 3.7, we have stated in Lemma 1 that if given enough hidden units, the *ATNN* only needs to adapt delays on one layer. The comparison of adapting different layer of time delay elements are tested on target discrimination data. The results are shown in Table 6.4. When T_1 and T_2 were fixed, the network is a *TDNN*. The network converged to a $RMSE = 0.0679$ and the overall identification performance is 95.12%. By adapting either T_1 , T_2 , or both (becomes *ATNN*), the networks all achieved around 100% correct recognition. This results give plausible support to Lemma 1.

6.3.3 Environment Variations and Robustness

One of the obscurities which degrades the performance of automatic target recognition is environmental changes. Sources of changes and noise include blurred

	fix T_1 & T_2	adapt T_2	adapt T_1	adapt T_1 & T_2
Training iterations	35677	32993	40165	40172
RMSE	0.0679	0.0492	0.05299	0.0522
Performance	95.12%	100%	99.15%	99.7%

Table 6.4: The comparison of network performance when various situations of time-delay adaptation is applied, T_1 and T_2 denote the time-delays in first and second layer respectively.

images, camera vibrations, heat, atmospheric effects, and more. The effects of noise and how to perform recognition in spite of noise will be very important for eventual deployment of this technology. Current target recognition systems are unable to modify their behavior based on the dynamic environmental changes occurring around them. In order to perform robustness, the system must be able to adapt its representation of this dynamic environment while maintaining acceptable performance [127]. We evaluated our neural network’s performance in the presence of noise, and improved the neural network robustness by training on noisy data.

To study the effects of noise on performance, we used varying scenarios in which noise was added during training or recall or both. Simulation results show that the *ATNN* is robust to moderate amounts of noise, and that its robustness improves when training is done with noisy data. This approach is important for eventual implementation in a real world environment, where substantial noise is expected. The data set is based on a simulation of a more realistic scenario than previous data sets, in which the scenario includes tenth second samples with

some noise added during simulation by the target simulator. These initial results showed that the *ATNN* recognized the target parts, in spite of noise during the simulation. We are concerned, however, that the real-world environment would have additional noise; the initial runs did not consider the effects of such additional noise. We address these effects here.

Our evaluation of noisy scenarios first required simulated data consisting of a simulated target scenario. We chose to start with noisy simulated data and to add varying amounts of additional noise. Thus, sources of noise include (1) the simulator or (2) addition of additional noise to simulated data.

Two different types of training scenario were tested:

1. the *ATNN* was trained with data direct from the target simulation, and
2. the *ATNN* was trained with noisy data consisting of simulation data with additional noise added.

The amount of additive noise was varied in each scenario.

The relation of the original raw data (simulated) and the noisy data is shown below:

$$s_n : size + K \times 0.01 \times N_r, radiance + K \times 0.001 \times N_r$$

$$r_n : radiance + K \times 0.001 \times N_r,$$

where K varies from 1 to 10, and N_r denotes a normal distribution random number with zero mean ($\mu = 0$), and variance (σ) is equal to 1, s_n is an instance of size data with additional noise added, and r_n is an instance of radiance data with additional noise added.

First, the network was trained on simulated data and tested on data with additional noise added. Table 6.5 shows the results. For each value of K (e.g.

runs	K					
	0	1	2	3	4	5
#1	99.77	100.00	98.64	87.16	75.00	25.00
#2	99.77	99.09	95.27	25.00	25.00	75.00
#3	99.77	99.77	99.09	91.21	75.00	25.00
#4	99.77	99.32	97.52	96.39	76.12	75.00
#5	99.77	100.00	99.54	89.86	75.00	75.00
#6	99.77	99.32	96.62	85.36	25.00	75.00
#7	99.77	99.54	96.62	80.85	83.78	75.00
#8	99.77	99.77	98.42	97.74	75.00	68.69
#9	99.77	100.00	98.87	92.34	25.00	75.00
#10	99.77	99.54	100.00	75.22	75.00	25.00
average	99.77	99.63	98.06	82.11	60.99	59.36

Table 6.5: Performance of *ATNN* trained on pure data and tested with varying amounts of noise added.

each level of additive noise), ten runs were performed. For each value of K used, the ten runs are shown in a column in Table 6.5. In general, the performance degraded as the amount (K) of noise increased. The data is plotted in Figure 6.16 with the solid line trace.

Figure 6.16 shows the property of graceful degradation, as performance is quite high (98-100%) until $K > 2$.

Next the network was trained with data that included additive noise. This data set contained the original simulated data in addition to the data with

runs	K				
	6	7	8	9	10
#1	31.30	25.00	75.00	25.00	25.22
#2	75.00	25.22	75.00	75.22	74.77
#3	75.00	25.00	74.77	25.00	74.32
#4	75.00	25.00	74.77	75.00	36.93
#5	75.00	25.45	80.63	75.00	25.00
#6	84.00	75.00	25.22	75.00	25.00
#7	75.00	75.00	25.22	25.00	75.00
#8	25.00	25.00	75.00	25.00	25.00
#9	25.00	75.00	75.00	26.57	75.00
#10	75.00	25.00	75.00	75.00	76.57
average	61.53	40.06	65.56	50.18	51.28

Table 6.5: (cont.) Performance of *ATNN* trained on pure data and tested with varying amounts of noise added.

additive noise. The variance values of the additive noise were 0.02 and 0.002 for size and radiance respectively.

The trained *ATNN* was again tested on varying amounts of noisy data as described above. The results of ten runs were shown in Table 6.6. The average performance is provided in the last row. Compared with the average performance in Table 6.5 (the last row), network trained with additive noise improved the identification capability. The comparison is shown in Figure 6.16.

For low noise ($K \leq 2$), there was slightly lower performance when the network was trained on noisy data. For higher noise ($2 < K < 8$), better performance was attained when the network was trained with additive noise. For even larger amounts of noise ($K \geq 8$), there was less jitter in performance when the network was trained with additive noise – more stable performance. We can conclude that the *ATNN* is more robust in noise situations when it is trained with additive noise. Similar studies and conclusions can be found in [69]. The lower range of noise, where the non-additive networks performed better, are not likely to occur in the real world.

runs	K					
	0	1	2	3	4	5
#1	94.94	93.18	96.71	96.21	87.62	72.47
#2	94.94	94.69	92.17	79.79	72.22	54.79
#3	94.94	95.95	95.95	96.21	64.39	72.72
#4	94.94	95.45	95.45	94.19	87.12	66.66
#5	94.94	96.96	96.96	91.41	80.30	83.83
#6	94.94	94.19	94.69	90.65	86.86	32.82
#7	94.94	94.44	93.18	92.17	90.40	41.91
#8	94.94	94.44	95.45	95.20	28.53	84.09
#9	94.94	95.45	95.20	94.69	83.58	63.38
#10	94.94	94.44	96.46	85.60	83.33	71.96
average	94.94	94.92	95.22	91.61	76.43	64.46

Table 6.6: Performance of *ATNN* trained on pure and noisy data and tested on varying amounts of noise added.

runs	K				
	6	7	8	9	10
#1	77.02	73.48	30.30	69.44	66.66
#2	30.05	31.31	50.25	70.95	26.76
#3	29.79	73.48	72.22	77.02	59.34
#4	78.28	28.28	71.46	30.80	66.16
#5	77.77	75.25	73.48	39.14	61.61
#6	80.80	71.46	65.90	31.56	68.93
#7	57.82	76.01	72.72	69.19	28.53
#8	73.98	71.21	27.27	67.42	70.20
#9	71.21	26.26	30.30	65.65	35.85
#10	30.30	66.41	31.56	53.78	76.01
average	60.70	59.31	52.55	57.50	56.01

Table 6.6: (cont.) Performance of *ATNN* trained on pure and noisy data and tested on varying amounts of noise added.

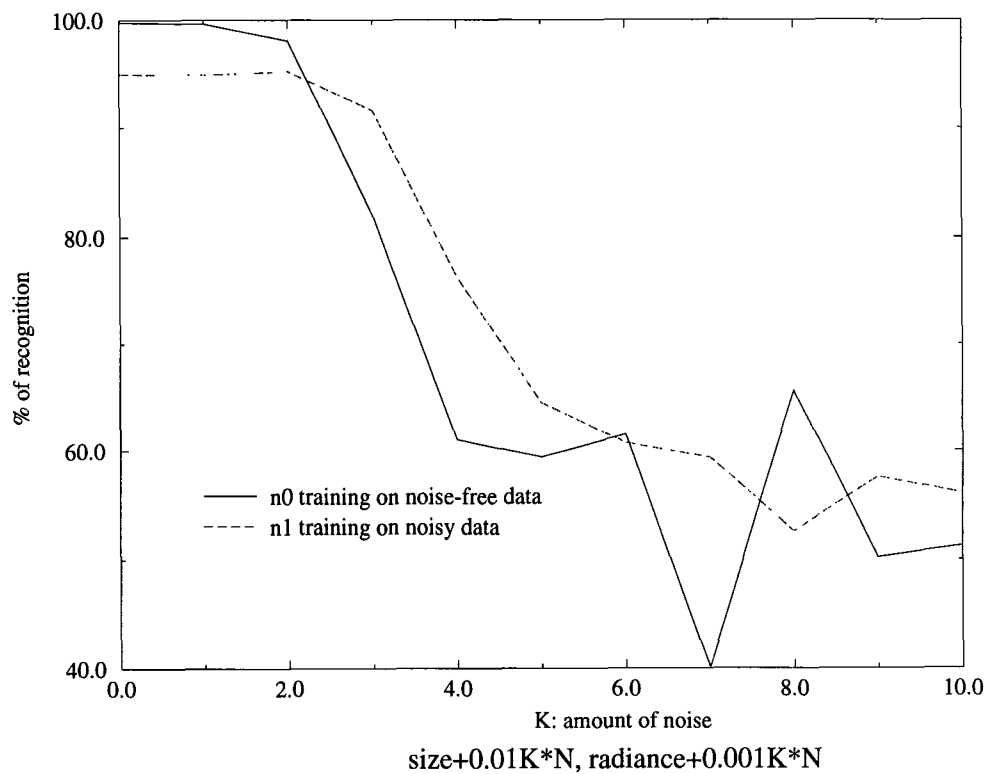


Figure 6.16: Performance of the *ATNN* as a function of the amount of noise in the data. For noisy environments, performance is superior when training is done on noisy data.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we have characterized the promising aspects of the *ATNN*, which has a dynamic learning technique for adapting both weights and time-delays on-line, for training with temporally varying patterns. The network is an efficient and flexible model that addresses learning situations where inputs arrive over a period of time and outputs are produced over a period of time, and adapts its time-delays and weights independently. Time-delays were trained to set individual length optimally to catch the necessary or significant information to the whole system. The special characteristics of this architecture include:

- Ability to learn temporal relationships between current data and the past history of events,
- Time delays are unconstrained with respect to one another,
- *ATNN* adapts time-delays as well as weights during training,

- *ATNN* reduces the network complexity by at least a factor of $O(n)$ compared to *BP* with the same mapping function.
- A trained *ATNN* can recognize temporal patterns without prior knowledge of when the pattern starts,
- *ATNN* can identify relationships in multi-channel data,
- *ATNN* can produce time-series predictions on more than one channel simultaneously.

We have elucidated that a time-delay network can be unfolded into a conventional feed-forward network (e.g. *BP*) with a time history as input. We have formulated explicitly how this unfolding operation can occur, with four sequential steps that involve creation of virtual units and moving all time delays to a preprocessing stage for the time histories. We showed with the *ATNN* how the adaptation of time-delays is mathematically equivalent to the adjustment of interconnections on a related feed-forward network. Thus interconnections can be changed by a gradient descent paradigm. We then applied the theorem of Funahashi, Hornik et al and Stone-Weierstrass to state the general function approximation ability of the *ATNN* and concluded that the *ATNN* is capable of approximating a spatiotemporal function to a desired accuracy. The *ATNN* can further play an important role in discovering the most appropriate time-delay values and connection strengths.

The *ATNN* can be applied to a wide variety of domains, including the recognition of movements, trajectory generation, speech analysis, and signal processing. We have applied the *ATNN* to examples which possess spatiotemporal complexity, with temporal sequences of patterns and with embedded attractors

and chaos. The *ATNN* was trained to learn the transformation of given input sets to output sets in sequence. The simulation results show that the *ATNN* learns the topology of a circular trajectory and figure eight within 500 on-line training iterations, and reproduces the circular trajectory itself dynamically with very high accuracy ($RMSE \leq 0.0162$). These results demonstrate the capability of the network to perform predictive recognition and figure completion when only partial information of the original data is available. The trained network produces the pattern or figure no matter where the starting position is.

When noise was added to the initial segment of a circle, the network eliminated the noise and completed the trajectory of the circle. For small amounts of noise, the network produced a very close fit to the circular trajectory. When larger amounts of noise were present, the trajectory gradually attained a closer and closer fit to the circle.

Other results show that the *ATNN* successfully produced the whole circle when trained on a partial circle, as long as the length of the training segment was large enough. Experiments on sampling rates show that low sampling rate training patterns speeded up training but degraded the noise removal, and a larger length of initial segment was needed to perform recall.

We have evaluated the behavior of internal hidden units over a series of applications. The internal nodes are selective and concentrate on the recent or past data due to the time-delay variables on incoming connections. This provides an extremely flexible way to explore the dynamic representation and learning in situations where inputs arrive over a period of time and outputs are produced over a period of time. Each hidden layer node is a feature detector, like in a classic feedforward network trained by Back-Propagation. But in the case of

the *ATNN*, hidden units detect features that are temporal patterns. When the network generated the closed trajectories studied here, the feature detectors become oscillators.

In dynamic systems, attractors are one of the important properties which can describe the behavior of the system. We showed that when training on a closed trajectory, the trajectory became an attractor of the network. When circle arcs with different radii were presented as inputs to a trained *ATNN*, the network could retrieve the trained track for different radii of the initial arc. Initial noisy segments also resulted in the network attaining the original trained figure. Thus, the network learned a fixed attractor and the varying initial segments were within the basin of attraction. This means that we can train a chosen attractor into a network's dynamics. Networks trained with appropriate attractors could have a significant impact on applications in robotics, signal production, time series prediction and other temporally dynamic situations.

We have shown that a prediction approach based on *ATNN* technology is effective and accurate. The beauty of this work is to show this complicated problem, based on the Mackey-Glass time series prediction, can be accomplished by such a simple *ATNN* architecture, while training can be done with reasonable amounts of data and computing time. Furthermore, the embedding of the system dynamics can be resolved adaptively, unlike exhausting trial and error efforts in conventional methods. A performance comparison demonstrated that *ATNN* performance outstripped that of the time-delay neural network (*TDNN*), which has adaptable weights but fixed time delays.

The feasibility of distinguishing multiple type components of exo-atmospheric targets is demonstrated by applying the *ATNN*. Exo-atmospheric targets are

especially difficult to distinguish using currently available techniques because all target parts follow the same spatial trajectory. Thus classification must be based on light sensors that record signal over time. The *ATNN* performs highly complex mapping on high dimensional input with a limited set of training data and achieves better generalization to overall trends of situations.

The related architecture of the *TDNN* is capable of generating solutions to trajectory classification problems. The network also possesses the special properties of graceful degradation, and location independence by bias offset.

Applications to identification and control as well as signal processing and speech recognition are domains to which this type of network can be naturally applied. These applications show that the *ATNN* is a powerful tool and learning technique for spatiotemporal data and for any applications that involve time varying signals or patterns.

7.2 Final Remarks and Future Research

Neural networks with time delays are inspired from biological models of nerve signal transmissions and processing. Biological studies have shown that varying time delays occur along axons due to different conduction times and differing lengths of axonal fibers, leading to variable delays on inputs to target neurons [18]. Time delays may also occur during dendritic transmission, and at synapses and cell bodies. Postsynaptic potentials exhibit temporal dependencies that may be varied depending on biochemical substrates and dendritic microstructure [17, 133]. Typically, these potentials have a peak that is delayed from the onset of a synaptic signal. Although adaptation of time delays in bio-

logical systems has not been experimentally observed, putative mechanisms for adaptation of time delays include biochemical modulation of postsynaptic responses, growth or shrinkage of axonal fibers, formation or deletion of synapses, and intracellular signaling. Thus the *ATNN* has its roots in biological systems, which are incredibly dynamic in their temporal properties. For biological systems, research on these architectures can be used to inspire hypotheses that ultimately can be tested experimentally on living neural systems.

The *ATNN* is a paradigm for training a nonlinear neural network, in which time delays and connection weights are adapted on-line according to a gradient descent approach, with time delays unconstrained with respect to one another, and an arbitrary number of interconnections with different time delays are placed between any two processing units. Networks with this capability can play an important role in applications domains that have naturally time-varying properties to their signals and dynamic situations. These domains include identification and control as well as medical data and image processing and speech recognition.

It has been an important task for physicians to investigate how organic systems regulate themselves under normal circumstances or fail to control themselves during disease. A common objective in physiology is to discover how living systems sense their current state and use the information to respond automatically and maintain proper operation [49]. Typically, medical recorded data is complex and difficult to analyze. The data often consist of ten or more channels that may be recorded continuously, signals are often nonstationary and a variety of nonlinearities abound in the interaction between the components of physiological systems, and the signals may have interference from noise or environmental changes as examples presented in Figure 7.1.

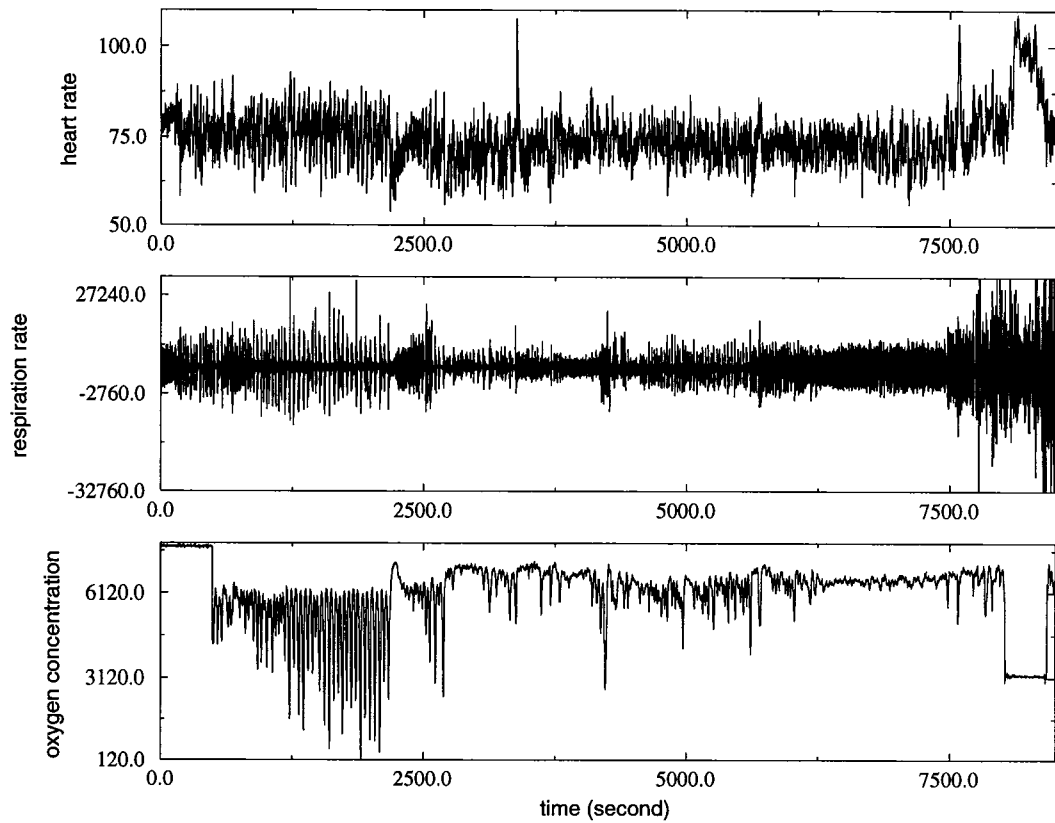


Figure 7.1: Physiological data from a patient with sleep apnea. Data includes: heart rate, respiration force, blood oxygen concentration. Time series sampled every 0.5 second. Data source: Time Series Prediction Competition, Santa Fe Institute, 1992 [158].

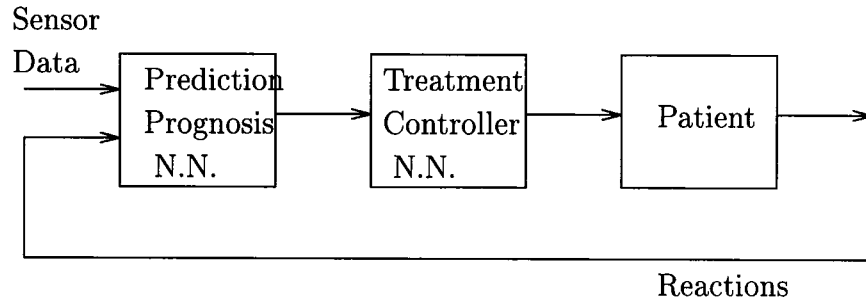


Figure 7.2: Symbolic block diagram of neural net application to prediction of physiology systems.

The state of modeling and simulation in physiology and medicine is currently unsatisfactory from the point of view of information analysis and prediction. Computer analysis of time series could provide routine diagnosis. It would be a remarkable contribution if one can develop precise analysis tool from the observed medical data to forecast life-threatening events such as sudden cardiac death, diabetic shock etc. Time-series prediction is a task that is highly applicable to a variety of problem in prognosis of physiology and medicine. We have suggested that the *ATNN* is well-suited for this type of problems due to its capability of handling multiple channel data and correlating relationships among these data, to make diagnosis possible [34]. One of the ultimate goals of prediction of physiological systems is to build a prediction and controller system in which a neural net predicts upcoming physiological properties and a controller decides on a control action to issue the treatment commands. Because of its inherent robustness, we have suggested that the *ATNN* can be very attractive in this field, and can act as both prediction and controller as shown in Figure 7.2 [35].

In addition to rich varieties of applications to which the *ATNN* can be ap-

plied, there are still open questions of how the *ATNN* can be improved in terms of adjusting the most appropriate time-delay connection numbers, configuration layout and learning procedure. The goal is to explore more flexible architectures and practical applications through analysis and simulations. Future work is to expand the capabilities of current analysis methods, and specifically, to enhance capabilities and overcome limitations in the following areas:

1. develop new and improved methods for deciding the most appropriate number of time-delays, perhaps an adding and pruning technique,
2. characterize the sensitivity of these new methods and their performance capabilities,
3. analyze the properties of basins of attraction that can be trained into network dynamics and consider applications in associative memory,
4. combine *ATNN* with a fusion model which combines pre-trained subnets with fusing strategies, built into a powerful large scale system, and finally
5. apply these new methods to specific data from time series data of economy, remote sensors and medical information, and evaluate the use of these systems, to identify the dynamic operational characteristics of these circuits.

The impact of this research is to have new mathematical and computer methods for the analysis of dynamic systems. The long-term result will impact our understanding of artificial neural systems, neural information processing, and the use of timing relationships in signal processing. New models for neural network architectures are expected to arise from this future research.

Appendix A

Trajectory Recognition with a Time-Delay Neural Network

Time-delay neural networks are powerful tools for the recognition of spatiotemporal patterns. Proposed by Waibel et al [149, 151], these networks incorporate time-delay elements along their interconnections, and can be trained by back-error propagation. Typically each unit receives a set of signals at the current time, and a time history of each signal channel. Back-error propagation then strengthens the weights along interconnections whose time delays are most relevant to the spatiotemporal pattern being recognized. The *TDNN* has been successfully used in speech recognition and compared favorably to traditional techniques such as the hidden Markov models on speaker independent phoneme recognition [149, 150, 152, 153]. Here we have applied the *TDNN* to a different but related problem, that of trajectory recognition.

Moving vehicles, insects, animals and other objects often have characteristic trajectories that distinguish different classes of those objects. For example, flying

insects have typical trajectory patterns that are different and distinguishable for different species. A moth, for example, flutters in the air whereas a fly embarks on a straighter path. Distinguishing trajectories may also be found for different aircraft (e.g. turning circles and maneuver ability may vary), planets, or other targets. In this appendix, we apply a *TDNN* to trajectory recognition and successfully classify different types of trajectories.

We have addressed the shift-invariance problem with two different approaches. First, training with identical trajectories at different positions enables the network to perform position independent recognition. Second, bias weights from the input layer are adjusted according to the estimated degree of shift after training, to allow recognition of the same trajectory at a different position. The network's tolerance to noise and degradation have also been measured. The degree of noise resilience is good, and performance degrades gracefully as a function of damage to the network.

A.1 Network Architecture

To recognize temporal trajectories, a network must have the ability to learn temporal relationships between current data and the past history of events, and it must learn temporal relationships between those events that are important. One of the main differences between biological neurons and artificial neural elements is the coding and transmission scheme for signals which take time to travel from neuron to neuron. The *TDNN* implements corresponding delays along the interconnections of an artificial neural network. Unlike conventional neural networks, which input static patterns, the *TDNN* can deal with patterns in the

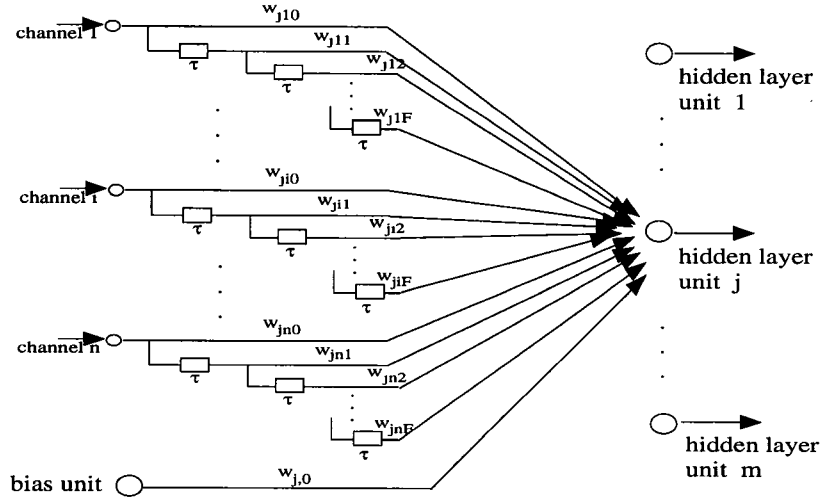


Figure A.1: *TDNN* with two layers — input and hidden layer.

time domain.

The schematic architecture of a typical *TDNN* is illustrated in Figure A.1 and a bias unit is added to the input layer during recall. For simplicity, each input channel i to the network is given by a single line, which is subject to delays by various time buffers. These various delayed inputs are then connected to the neurons of the second layer and again, each unit in the second layer is connected in the same manner to the next layer.

The second layer is considered to be a hidden layer, and hidden units are shown at the right of Figure A.1. Each unit in the hidden layer receives input in the time-delay frame window (the frame window is the number of time delay steps employed), and the activation value is stored over time. The frame window may vary for each signal channel and for different layers. Finally, the output is obtained by summing all activation values of previous layers stored over a period of time multiplied by their connection weights. This period is equal to the number of time delays which are applied between the previous layer and the

output layer.

Consider a multi-layered time-delay network with a delay of $k\tau$ on the k th connection. The activation level for unit j , pattern μ is

$$a_j^\mu(t) = f(S_j^\mu(t)) \quad (\text{A.1})$$

where

$$S_j^\mu(t) = \sum_i \sum_{k=0}^F w_{j,i,k} a_i^\mu(t - k\tau) \quad (\text{A.2})$$

and $f(x)$ is a symmetric squashing function.

The learning rule and weight adjustment are according to the back-error propagation paradigm with momentum. If unit j is in the output layer, then its error value is

$$\delta_j^\mu(t) = (T_j^\mu - a_j^\mu(t)) f'(S_j^\mu(t)) \quad (\text{A.3})$$

where T_j^μ denotes the target value for unit j (pattern μ) and $f'(x)$ is the derivative of the symmetric sigmoid.

If unit j is in the hidden layer, the weighted sum is taken of the δ values of all units that receive output from unit j . The error value of unit j is computed as:

$$\delta_j^\mu(t) = [\sum_k \sum_m \delta_m^\mu(t) w_{m,j,k}] f'(S_j^\mu(t)) \quad (\text{A.4})$$

where m and k denote the units of the next layer and delay element respectively.

The adjustment of the connection weights is done using the generalized delta rule and the momentum term is employed to speed convergence as:

$$\Delta w_{j,i,k}(t) = \eta \delta_{j,k}^\mu(t) a_i^\mu(t - k\tau) + \alpha \Delta w_{j,i,k}(t - 1) \quad (\text{A.5})$$

Learning is continued until performance goes below a criterion value, measured by root-mean-square (*RMS*) .

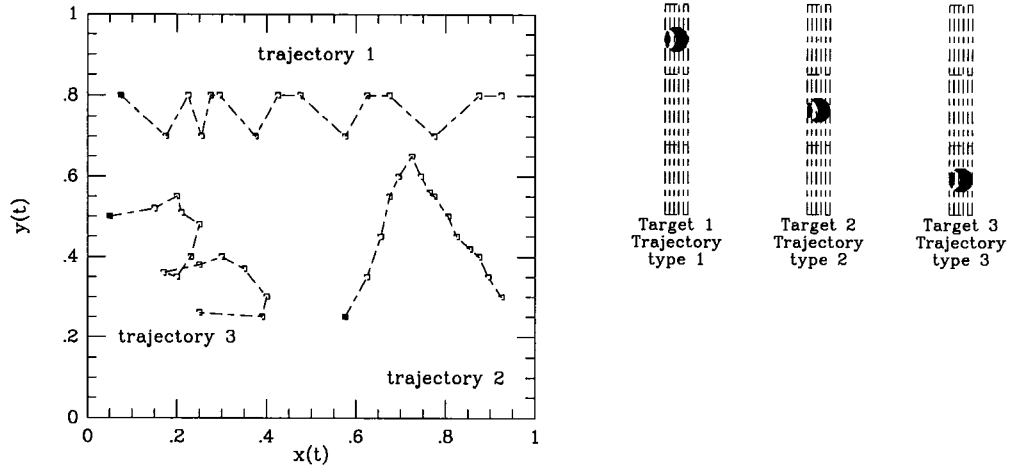


Figure A.2: Training set and target values. Three output units represented the three trajectory types.

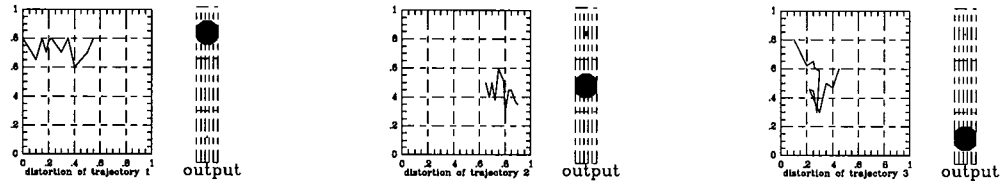


Figure A.3: Test for distorted trajectories.

A.2 Trajectory Recognition Results

A three-layered *TDNN* was simulated to recognize different spatiotemporal trajectory segments according to their special properties. Three trajectories with different patterns of motion were defined on the x-y plane. These trajectories started at the point $(x^\mu(t=0), y^\mu(t=0))$, where $x^\mu(t)$ and $y^\mu(t)$ stands for the position of object μ at time t ($\mu = 1, 2, 3$ and $t = 0, 1, 2, \dots, n$). Figure A.2 shows the three trajectories.

A three-layered time-delay network with two input units, two hidden units and three output units was used in this simulation. The input layer had two units

and was arranged to receive the input channel signal $x(t)$ and $y(t)$. Two delay elements were put along the paths from input units and hidden units, and nine time step delays were employed between the hidden layer and the output layer for collecting a wider range of activation information from the previous layer. Three output units were designed to tell the result of recognition. The desired output pattern for each trajectory is represented to the right of Figure A.2, where the size of the circle stands for the activation value of a single output unit.

The training set as defined in Figure A.2 was presented to the network 552 epochs until the *RMS* reached below 0.02. The test set, which was composed of segments from the training set, as shown in Figure A.2 , was then presented to the resulting network. The resulting network was able to recognize all these test patterns. In addition, the network showed good performance on a test set formed by adding small amounts of random noise to the training set. A number of the distorted trajectory segments were tested by the network and most results were correct, as shown in Figure A.3 . An additional experiment was done with all three trajectories placed at the same approximate position on the x-y plan. The network trained successfully in this case also.

A.2.1 Shift-Invariance

Transformation invariance is a major problem in image processing, as the universal transfer functions and mapping schemes for all types of transformations are difficult to obtain. Methodologies to obtain scale, shift, and rotation invariance have been investigated [9]. Generally, two conventional methods are used. One approach uses a special preprocessing transform function to extract the common features invariant to some transformation. Another approach tries to match the

observed object with stored models by comparing parameters. Unfortunately, these techniques are only successful for special problems in a limited range of applications.

We have used several alternative techniques on the shift-invariance problem. By training, the network was trained to recognize trajectories at different areas. By adding a bias unit, the bias weights were adjusted to recognize the shifted patterns during recall. Other techniques, such as simulated annealing, gradient hill climbing and genetic algorithms are under consideration.

A.2.2 Shift-Invariance by Training

Is it possible to design a network which possesses shift-invariance without applying any pre-transformation or post-processing ? The neocognitron developed by Fukushima was originally claimed to be shift-invariant. However, Casasent found that the neocognitron is not intrinsically shift invariant, and that some network parameters must be tuned to obtain approximate shift-invariance [19].

Without adding special pre or post-processing, one way to resolve this problem is to construct a new training set formed by duplicating the original training set evenly spaced on the x-y plane and then training the network. This is motivated intuitively by training the network to adapt to all possible locations in which a particular trajectory may appear on the plane. One may ask how many copies of translated pattern sets we need to train the system to get good performance. The answer may depend on the network architecture, how many trajectories one is trying to classify, how different the classes are, the amount of noise and other factors.

For this particular application, we found that three copies of the training set

is enough for the network to obtain shift-invariance. We trained the *TDNN* by first using a single copy of the training set. After the system converged, we then tested it with a test set that contains 170 testing patterns, formed by taking portions of each type of trajectory and spreading them randomly over the space. We then retrained the *TDNN* by adding a copy of the original training set evenly spaced on the x-y plane. We repeated this procedure until we had six copies of the training set evenly spaced on the x-y plane. The system gave a 29.41% recognition rate of the 170 test patterns when one training set was used. The performance is increased to 81.76% after two copies of the training set are applied, and it reached 100% correct recognition after employing three copies of the training set. This result shows that this system is shift-invariant with few copies of the training set.

A.2.3 Shift-Invariance by Recall with Bias

We then applied a second method to address the shift-invariance problem by making the bias weights a function of the amount of shift and the other weights

$$w_{j,0} = -\sum_i \sum_k s_i w_{j,i,k} \quad (\text{A.6})$$

where s_i is the amount of trajectory shift in channel i ($i = 1, 2$ and $k = 0, 1, \dots, F$ in this application).

Let the incoming sum of unit j in the hidden layer during recalling pattern μ at time t be $\hat{S}_j^\mu(t)$ and

$$\hat{S}^\mu(t) = a_0 w_{j,0} + \sum_i \sum_{k=0}^{d_i} w_{j,i,k} \hat{a}_i^\mu(t - k\tau), \quad (\text{A.7})$$

where a_0 is equal to 1 and $\hat{a}_i^\mu(t)$ denotes the i th input channel signal over time

with the amount of shift s_i compared to training patterns.

$$\hat{a}^\mu(t) = a_i^\mu(t) + s_i \quad (\text{A.8})$$

It can be easily shown that $\hat{S}_j^\mu(t)$ will retain the value of $S_j^\mu(t)$ by substituting Equation A.8 into A.7 .

Experiments were performed that successfully compensated for spatial shifts in the trajectories by adjusting the bias weights on recall. The network was trained on three different trajectory types, with a single instance of each trajectory. During training, no bias units were used. Shifted trajectories were then presented to the network. After adding bias weights calculated according to A.7, the shifted trajectories were recognized.

A.2.4 Graceful Degradation

The "degradation" characteristics of the network were investigated in the presence of lesions. As expected, the deficits in the network capabilities were roughly proportional to the sizes of the lesions. In this study, we examined performance of a trained network after lesions were made in which we randomly set different amounts of elements in the 72 weights of the network to zero. Figure A.4 illustrates the simulation results. In this experiment, the same test patterns were tested for different amounts of lesion. The network still performed normally when 5% of the connections were destroyed. It functioned well for trajectory 1 and 3 up to a 10% lesion. Since we appeared to have low redundancy in the network, it is reasonable that the system degraded at a 10% lesion. It is interesting that the network performed well for trajectory 1 even when 30% of the connections were damaged. The question for this special case can be answered by the

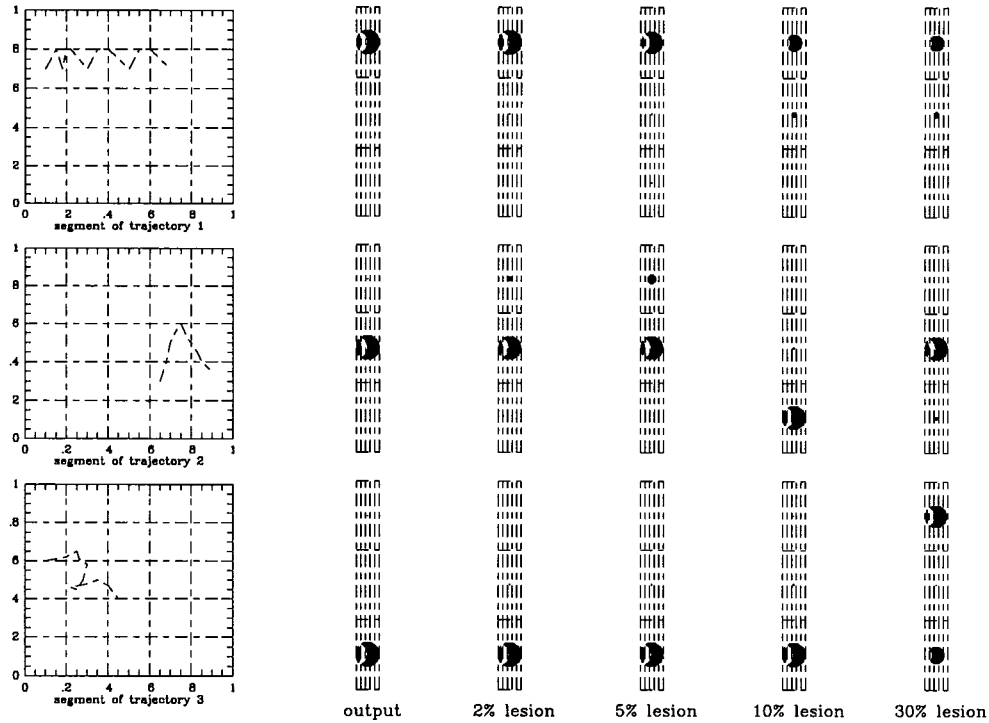


Figure A.4: Lesion

feature detectors of the network. Since not all of the feature-distinguishing elements of trajectory 1 are destroyed in this experiment, the network can maintain the recognition. This result shows that *TDNN*'s have a graceful degradation property, it was previously shown that traditional back-propagation networks have such a graceful degradation property [132].

A.2.5 Noise Tolerance

To investigate how resilient the system is to noise, a number of experiments were done. The shift-invariant network described in the last section was again tested using the same 170 pattern testing set to which different random variance noise was added. We assume the noise is independent of the input signal and is normally distributed with zero mean. Figure A.5(b) diagrams the performance of

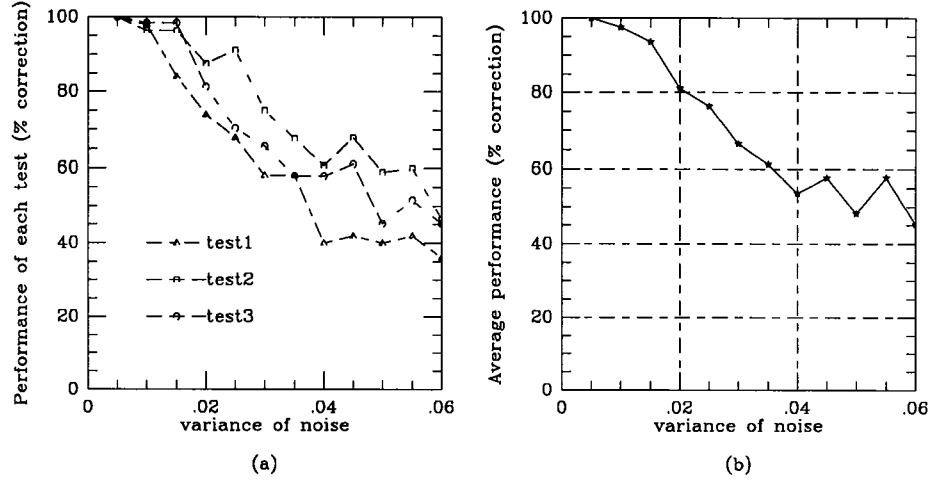


Figure A.5: (a) Noise tolerance of different test set (b) Average noise tolerance.

the network under various noisy conditions. The performance decays as the noise increases. Generally, this system can sustain a noise with 12% of the signal amplitude. Figure A.5(a) shows the noise resilience of different types of trajectories. Three dotted lines labeled by test1, test2 and test3 in Figure A.5(a) represent the results of the noise resilience of the system for recognizing trajectories 1, 2, and 3 respectively. As one can see, the network noise tolerance to trajectory 1 is worse than to the others, since the trajectory turns more often.

A.3 Conclusion

In this paper, the properties of a *TDNN* for trajectory recognition are analyzed. The results of analysis and simulations have shown that the proposed *TDNN* system is capable of generating solutions to trajectory classification problems. Different trajectories can overlap and still be recognized. Shift-invariance was addressed in two ways: by training and by adjustment of bias weights. Performance results were found for graceful degradation and for tolerance to noise in

the data.

Future study is expected to result in guidelines for obtaining the solutions to trajectory recognition with noise tolerance and shift-invariance, and will have important implications to military target recognition, moving object recognition, and classification of temporally changing data.

Appendix B

Time-Delay Matrices:

Simulation Results

B.1 Resulting Time-Delay Matrices of Various Training Scenarios

1. Time-delay matrices trained on full circle:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (B.1)$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 14 & 3 & 4 & 11 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 & 5 & 11 & 5 & 0 & 7 & 8 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (\text{B.2})$$

2. Time-delay matrices trained on $\frac{3}{4}$ circle:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (\text{B.3})$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 14 & 3 & 4 & 11 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 5 & 11 & 5 & 0 & 7 & 8 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (\text{B.4})$$

3. Time-delay matrices trained on $\frac{1}{2}$ circle:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (\text{B.5})$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 4 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 8 & 3 & 4 & 8 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 5 & 8 & 5 & 0 & 5 & 6 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (\text{B.6})$$

4. Time-delay matrices trained on $\frac{1}{4}$ circle:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (B.7)$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (B.8)$$

B.2 Resulting Time-Delay Matrices of Sampling Rate Effects

1. Time-delay matrices trained on 63 point per circle:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (B.9)$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 14 & 3 & 4 & 11 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 5 & 11 & 5 & 0 & 7 & 8 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (B.10)$$

2. Time-delay matrices trained on 32 point per circle:

$$T1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 4 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 8 \\ 3 & 4 & 2 & 3 & 0 & 1 & 7 & 8 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (B.11)$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 6 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 20 & 2 & 21 & 4 & 22 & 0 & 1 & 16 & 20 & 19 & 19 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 22 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (B.12)$$

3. Time-delay trained on 21 point per circle:

$$T1 = \begin{bmatrix} 0 & 1 & 3 & 3 & 0 & 6 & 2 & 3 \\ 0 & 6 & 8 & 8 & 0 & 11 & 10 & 13 \\ 6 & 7 & 2 & 6 & 1 & 1 & 10 & 10 \\ 3 & 1 & 2 & 7 & 0 & 1 & 10 & 7 \\ 4 & 1 & 2 & 5 & 0 & 1 & 2 & 3 \end{bmatrix} \quad (B.13)$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 10 & 1 & 10 & 8 & 12 & 5 & 0 & 1 & 13 & 8 & 4 & 13 \\ 0 & 1 & 7 & 3 & 4 & 9 & 10 & 10 & 8 & 10 & 6 & 11 & 4 & 1 & 6 & 10 & 10 & 10 \\ 0 & 1 & 2 & 11 & 16 & 8 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 5 & 6 & 12 & 5 & 9 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad (B.14)$$

4. Time-delay matrices trained on 16 point per circle:

$$T1 = \begin{bmatrix} 0 & 2 & 3 & 3 & 1 & 4 & 2 & 3 \\ 0 & 4 & 5 & 6 & 0 & 6 & 7 & 8 \\ 2 & 4 & 2 & 5 & 3 & 1 & 6 & 6 \\ 2 & 1 & 2 & 5 & 0 & 1 & 6 & 5 \\ 2 & 1 & 2 & 5 & 1 & 3 & 2 & 4 \end{bmatrix} \quad (\text{B.15})$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 9 & 1 & 9 & 7 & 9 & 5 & 0 & 1 & 11 & 10 & 4 & 11 \\ 5 & 1 & 9 & 3 & 8 & 10 & 9 & 10 & 8 & 26 & 5 & 25 & 5 & 3 & 9 & 22 & 12 & 21 \\ 0 & 1 & 7 & 11 & 11 & 10 & 0 & 1 & 2 & 3 & 6 & 5 & & & & & & \\ 0 & 10 & 11 & 22 & 7 & 9 & 4 & 3 & 2 & 8 & 7 & 7 & & & & & & \end{bmatrix} \quad (\text{B.16})$$

Appendix C

Target Discrimination

Simulation Results

C.1 Simulation Results on 0.5 second data.

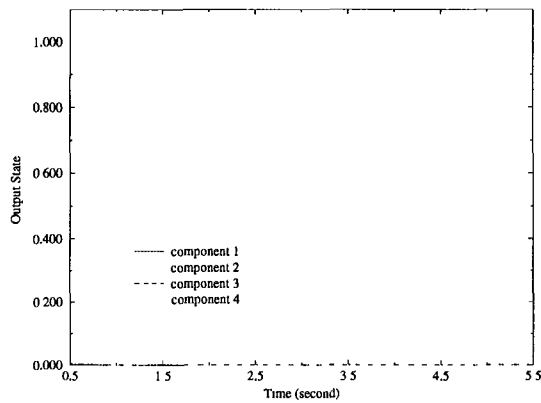


Figure C.1: Recognizing target in trajectory 1 with aspect angle 30° .

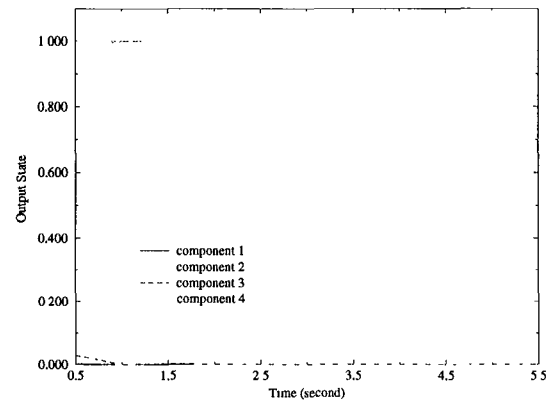


Figure C.2: Recognizing target in trajectory 1 with aspect angle 45° .

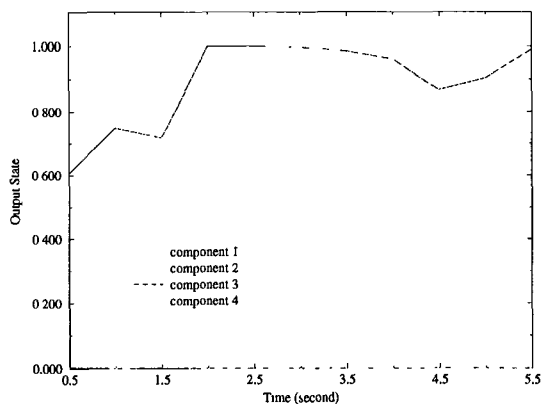


Figure C.3: Recognizing target in trajectory 1 with aspect angle 60° .

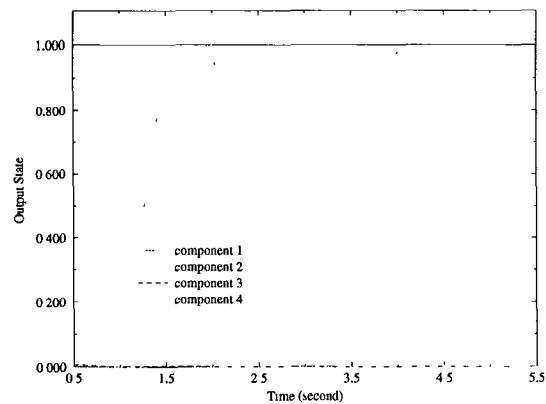


Figure C.4: Recognizing target in trajectory 1 with aspect angle 75° .

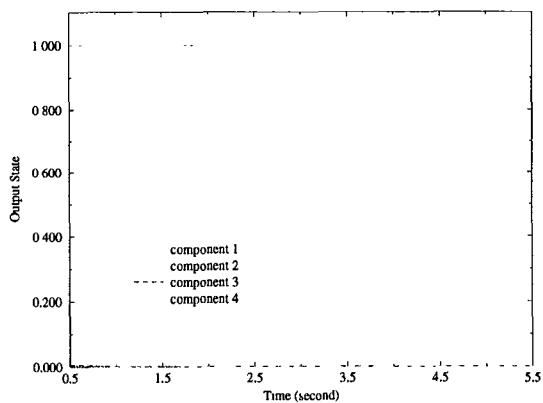


Figure C.5: Recognizing target in trajectory 1 with aspect angle 90° .

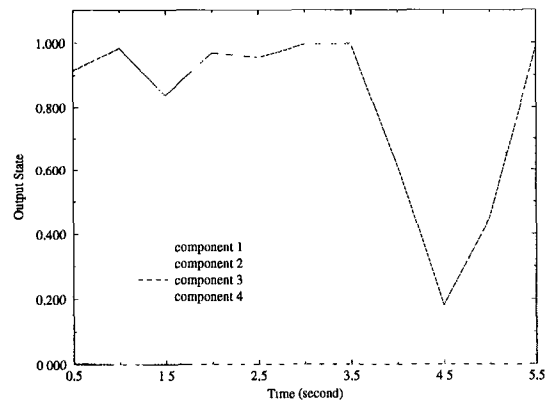


Figure C.6: Recognizing target in trajectory 2 with aspect angle 30° .

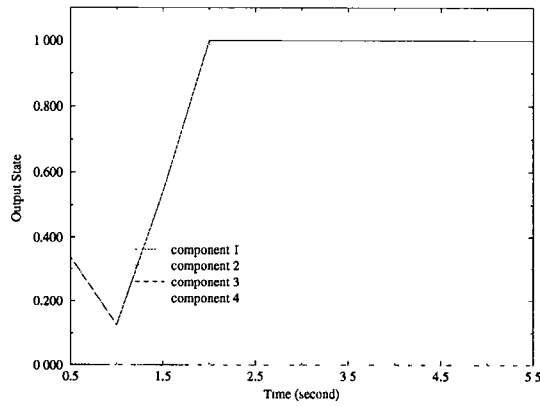


Figure C.7: Recognizing target in trajectory 2 with aspect angle 60° .

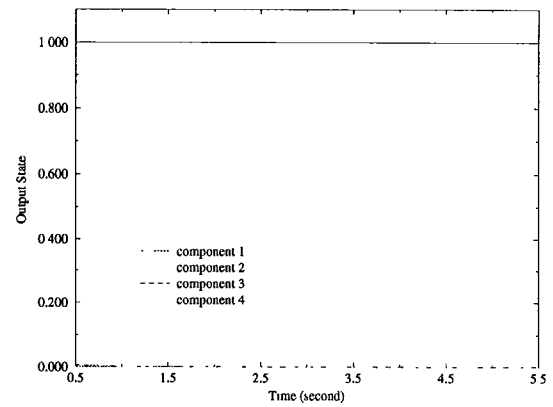


Figure C.8: Recognizing target in trajectory 2 with aspect angle 90° .

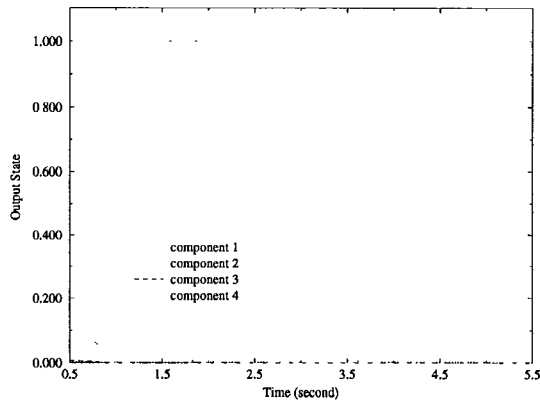


Figure C.9: Recognizing target with aspect angle 90° , front $\frac{1}{3}$ component 2 replaces whole component 2.

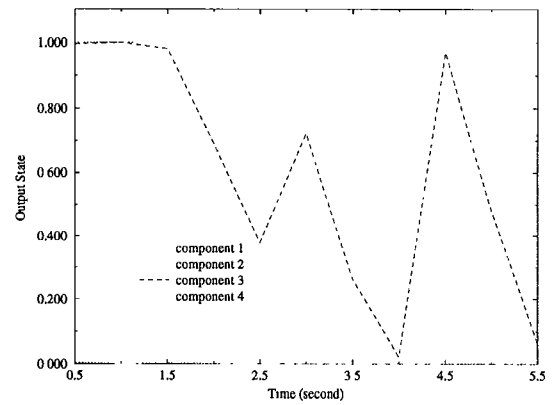


Figure C.10: Recognizing target with aspect angle 90° with back $\frac{1}{3}$ component 2 replaces whole component 2.

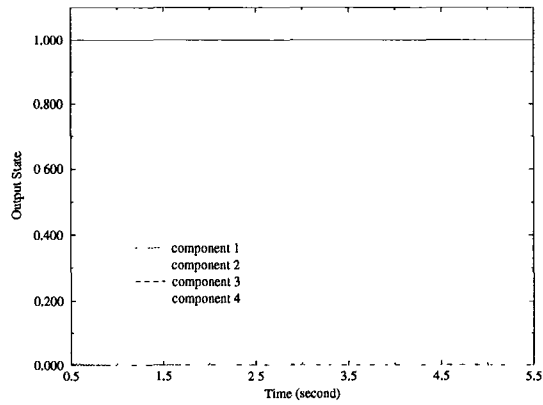


Figure C.11: Recognizing target with aspect angle 90° with front $\frac{1}{2}$ component 3 replaces whole component 3.

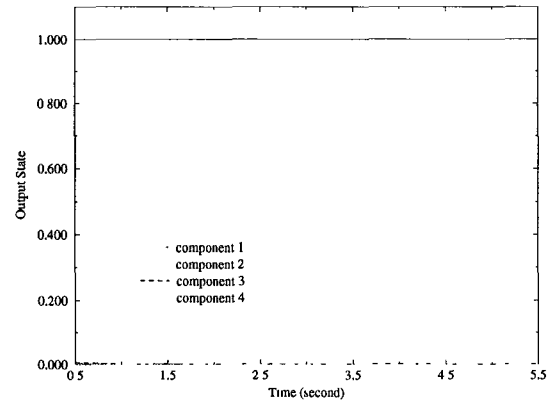


Figure C.12: Recognizing target with aspect angle 90° with front $\frac{1}{3}$ component 4 replaces whole component 4.

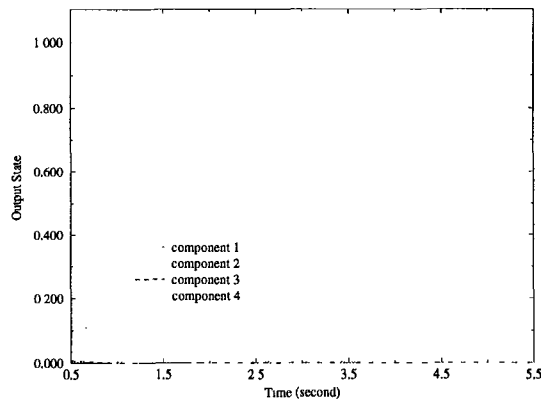


Figure C.13: Recognizing target with components tumbling angles: 90° , 60° , 90° , and 90° respectively.

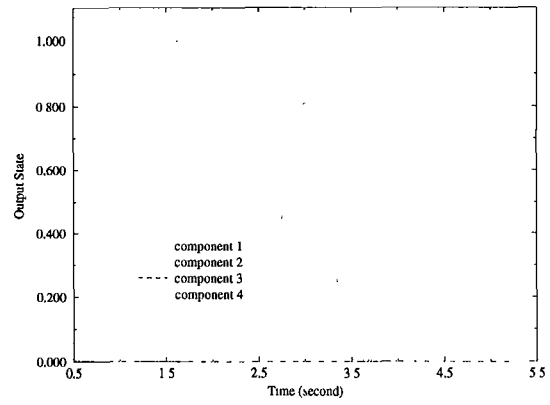


Figure C.14: Recognizing target with components tumbling angles: 90° , 60° , 60° , and 90° respectively.

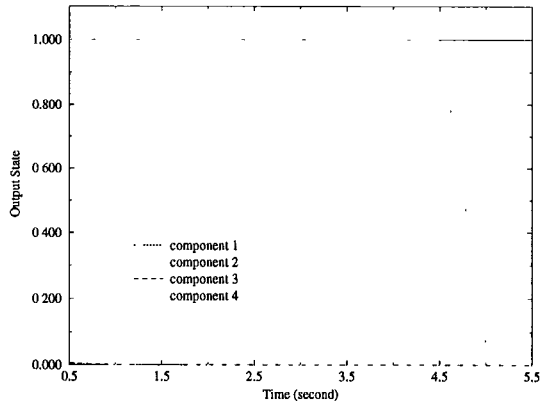


Figure C.15: Recognizing target with components tumbling angles: 90° , 60° , 60° , and 30° respectively.

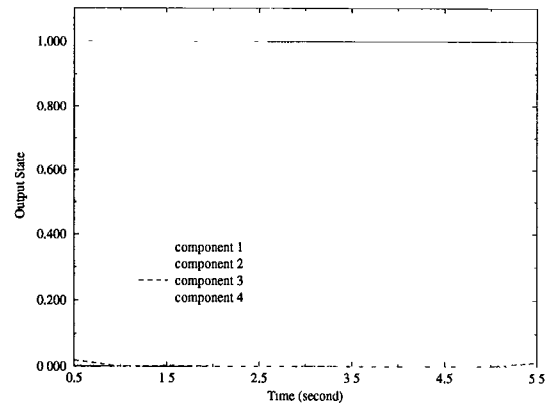


Figure C.16: Recognizing target with components tumbling angles: 60° , 90° , 60° , and 60° respectively.

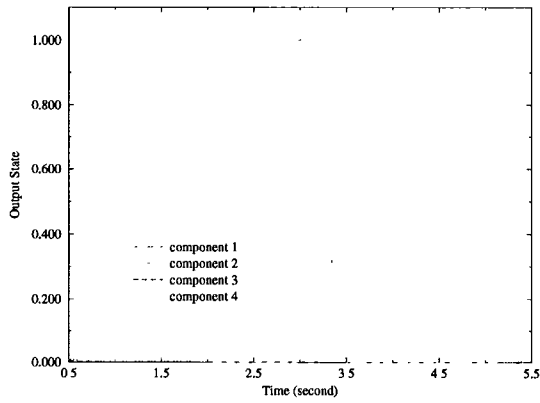


Figure C.17: Recognizing target with components tumbling angles: 60° , 90° , 30° , and 60° respectively.

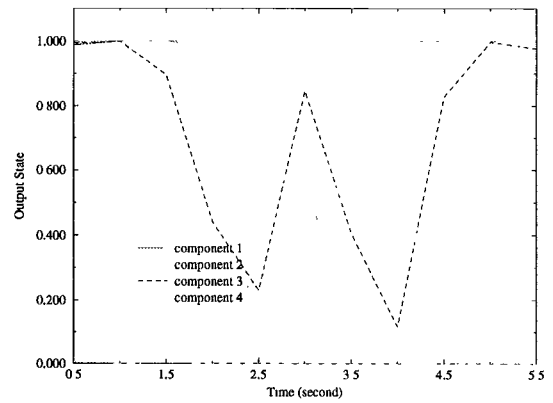


Figure C.18: Recognizing target with components tumbling angles: 60° , 90° , 90° , and 60° respectively.

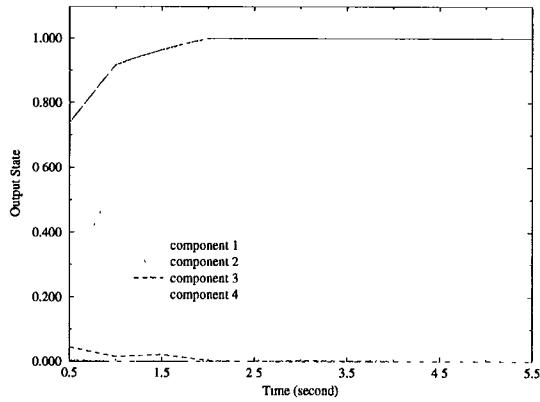


Figure C.19: Recognizing target with components tumbling angles: 60° , 30° , 30° , and 60° respectively.

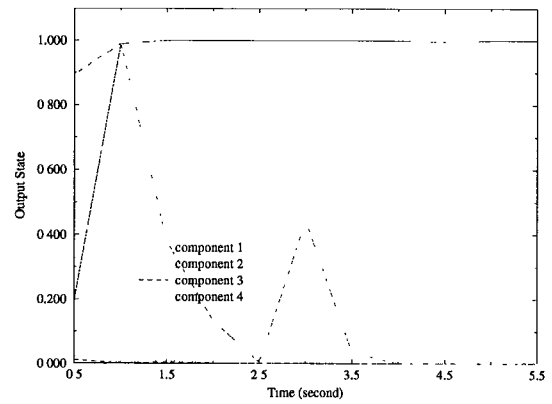


Figure C.20: Recognizing target with components tumbling angles: 60° , 30° , 60° , and 90° respectively.

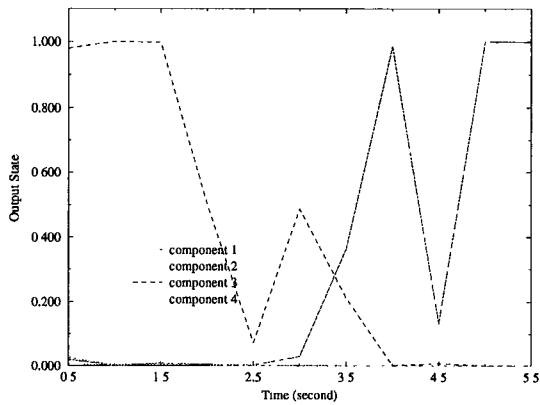


Figure C.21: Recognizing target with components tumbling angles: 60° , 30° , 90° , and 90° respectively.

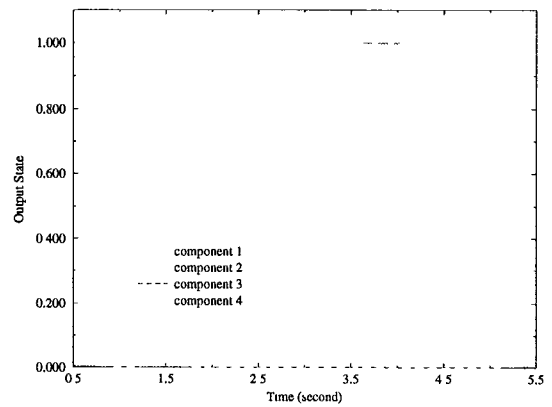


Figure C.22: Recognizing target with components tumbling angles: 30° , 60° , 30° , and 30° respectively.

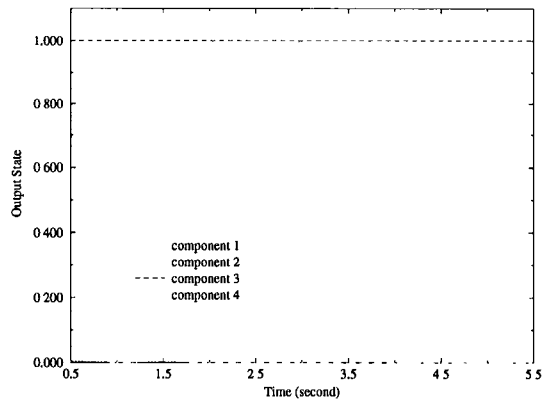


Figure C.23: Recognizing target with components tumbling angles: 30° , 60° , 90° , and 30° respectively.

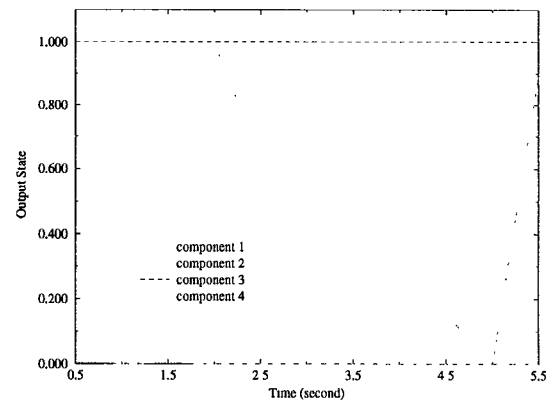


Figure C.24: Recognizing target with components tumbling angles: 30° , 60° , 90° , and 60° respectively.

C.2 More Results on 0.1 second data.

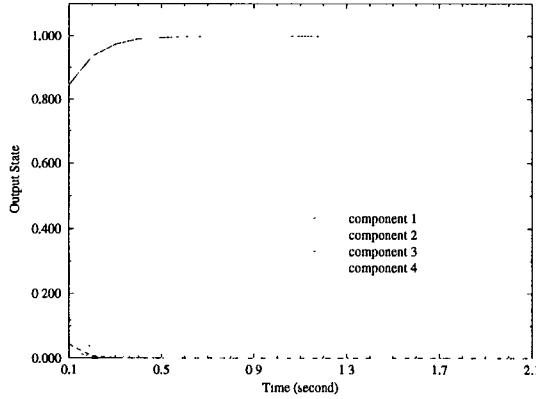


Figure C.25: Recognizing target with aspect angle 90° .

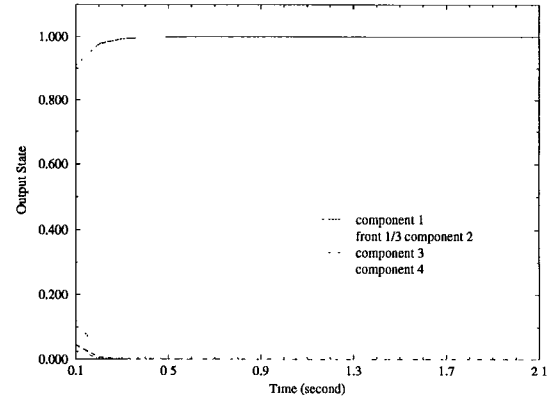


Figure C.26: Recognizing target with aspect angle 90° , front 1/3 component 2 replaces the whole.

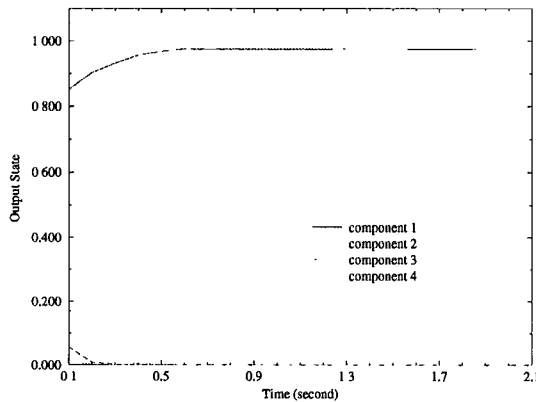


Figure C.27: Recognizing target with aspect angle 90° , components tumbling angle: 60° , 90° , 90° , and 30° respectively.

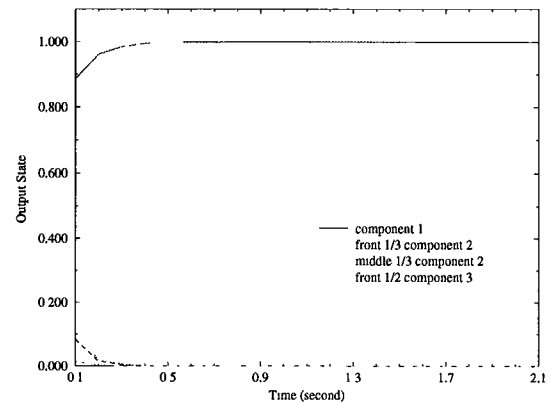


Figure C.28: Recognizing target with aspect angle 90° , front 1/3 component 2 middle 1/3 component 3, front 1/2 component 4.

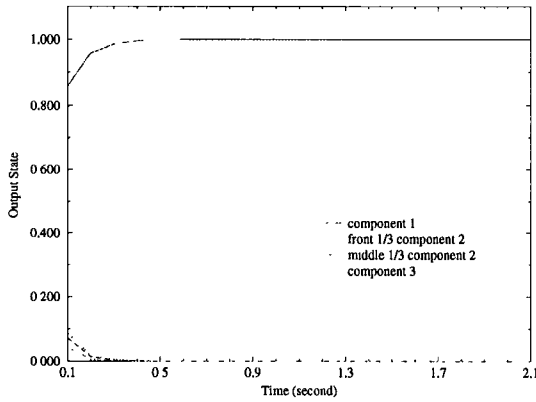


Figure C.29: Recognizing target with aspect angle 90° , front 1/3 component 2, middle 1/3 component 2, and component 4

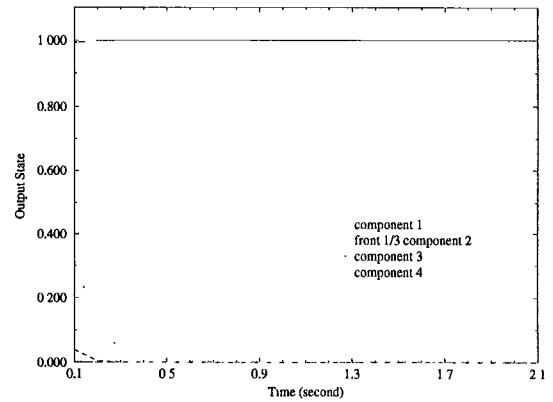


Figure C.30: Recognizing target with aspect angle 30° , front 1/3 component 2 replaces whole component 2.

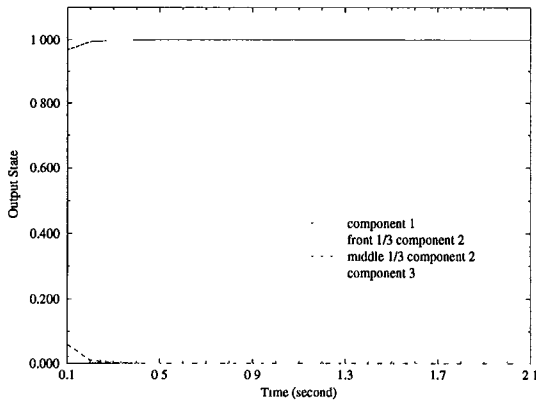


Figure C.31: Recognizing target with aspect angle 30° , front 1/3 component 2, middle 1/3 component 2, component 3

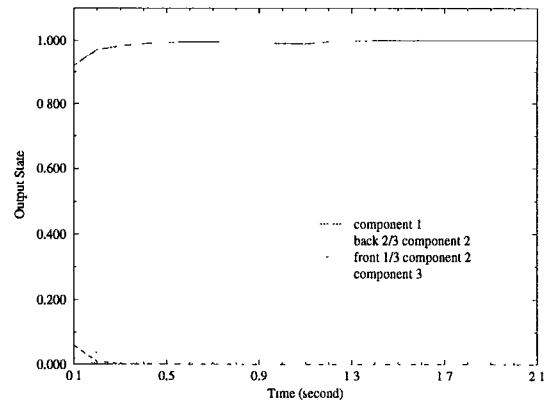


Figure C.32: Recognizing target with aspect angle 30° , front 1/3 component 2, back 2/3 component 2, and component 4.

C.3 Simulation Results on partial 0.1 second data.

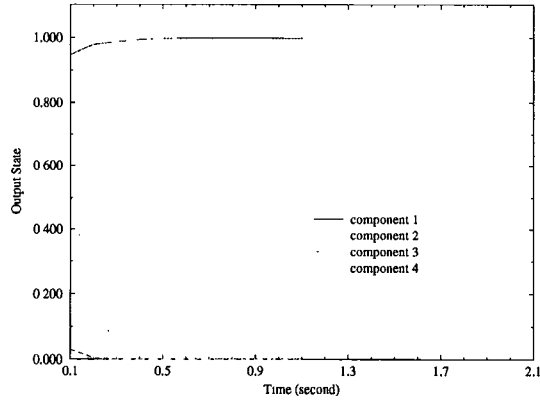


Figure C.33: Aspect angle 30°.

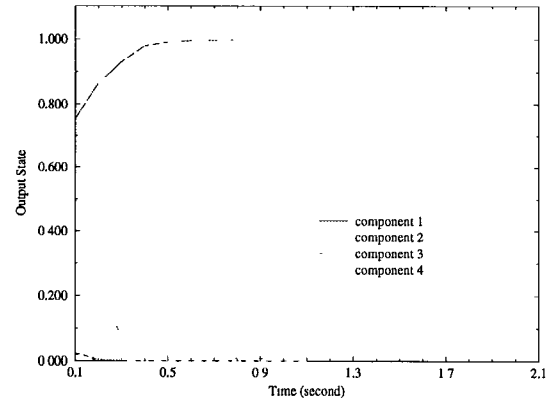


Figure C.34: Aspect angle 60°.

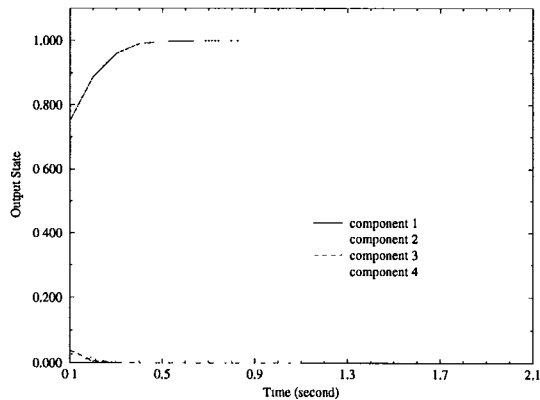


Figure C.35: Aspect angle 90°.

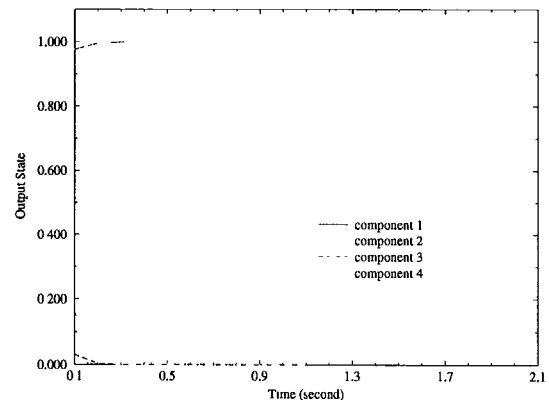


Figure C.36: Aspect angle 30°, front 1/3 component 2 replaces whole component 2.

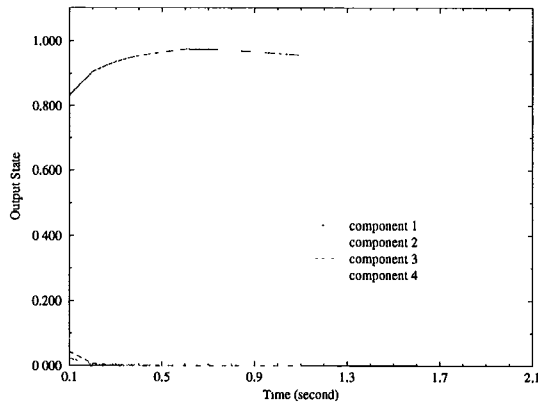


Figure C.37: Aspect angle 30° , back 1/3 component 2 replaces whole component 2, front 1/3 component 2 replaces whole component 3.

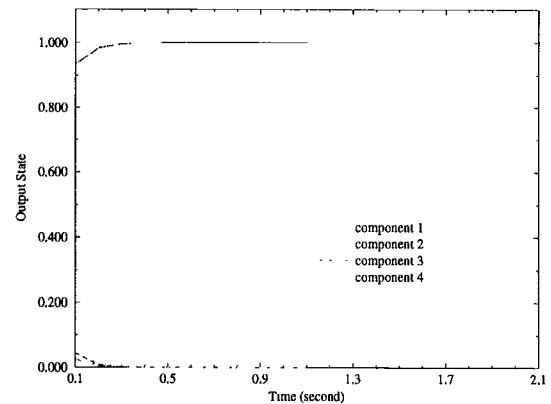


Figure C.38: Aspect angle 30° , front 1/3 component 2 replaces whole component 2, middle 1/3 component 2 replaces whole component 3.

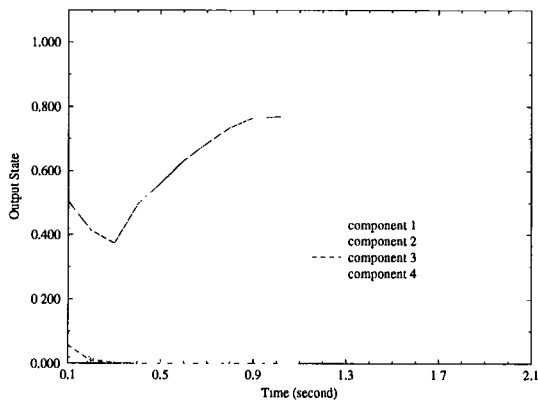


Figure C.39: Aspect angle 90° , with components tumbling angle: 30° , 60° , 90° , and 30° respectively.

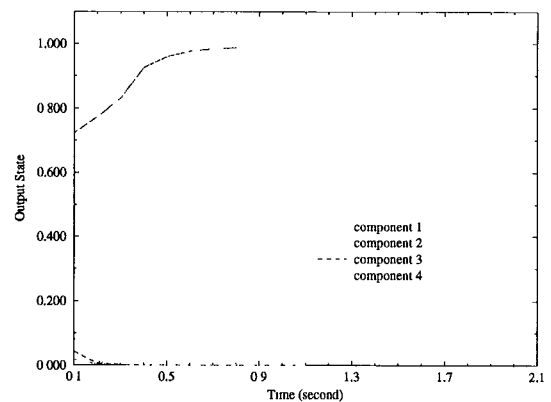


Figure C.40: Aspect angle 90° , with components tumbling angle: 60° , 90° , 90° , and 30° respectively.

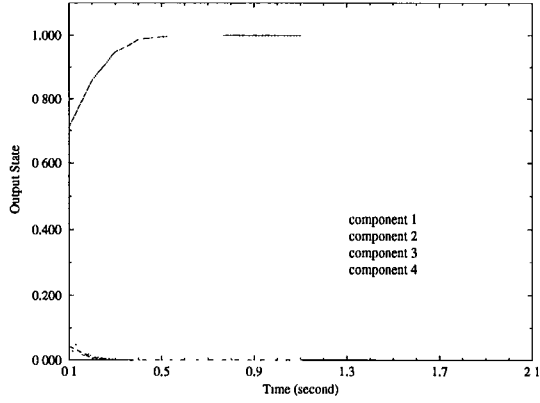


Figure C.41: Aspect angle 90° , time 323-325 second.

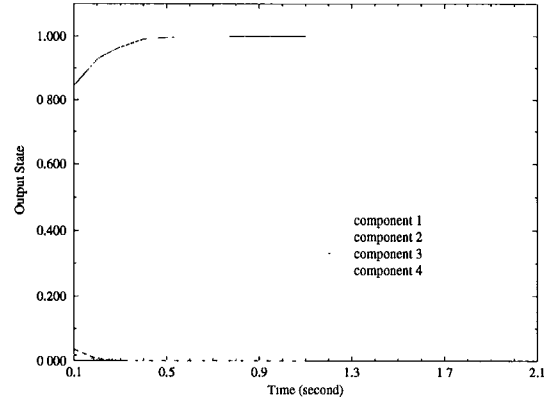


Figure C.42: Aspect angle 90° , with components tumbling angle: 90° , 60° , 60° , and 30° respectively.

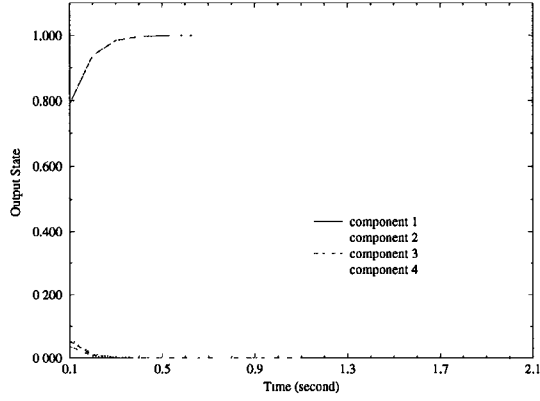


Figure C.43: Aspect angle 90° , front 1/3 component 2 replaces whole component 2, middle 1/3 component 2 replaces whole component 3.

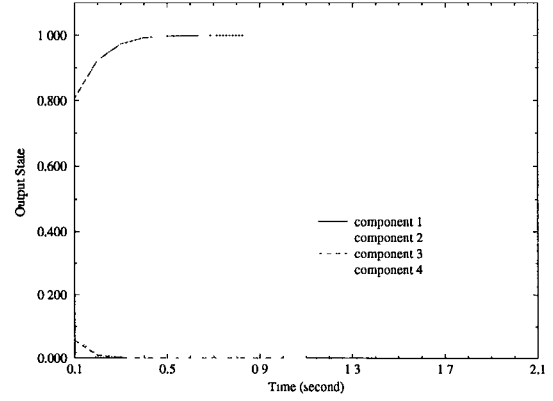


Figure C.44: Aspect angle 90° , front 1/3 component 2 replaces whole component 2, middle 1/3 component 2 replaces whole component 3, front 1/2 component 3 replaces component 4.

C.4 Resulting ATNN Time-Delay Matrix

T1 is the time-delay matrix on the first layer and T2 is the time-delay matrix for the second layer.

$$T1 = \begin{bmatrix} 1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 1 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{bmatrix}$$

$$T2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 6 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

Bibliography

- [1] S.A. Amari and K. Maginu. Statistical neurodynamics of associative memory. *Neural Networks*, 1:63–73, 1988.
- [2] B. R. Amirkian and A. V. Lukashin. A neural network learns trajectory of motion from the least action principle. *Biological Cybernetics*, 66:261–264, 1992.
- [3] D. Amit. Neural networks for counting chimes. *Proceedings of the National Academy of Sciences, USA*, 85:2141–2145, 1988.
- [4] J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, 1988.
- [5] T.W. Anderson. *The Statistical Analysis of Time Series*. John Wiley & Sons, Inc., New York, NY, 1991.
- [6] B. Bai and N.H. Farhat. Learning networks for extrapolation and radar target identification. *Neural Networks*, 5:507–529, 1992.
- [7] P. Baldi. Gradient descent learning algorithms and limitations in dynamical systems. In Y. Chauvin and D. E. Rumelhard Eds., editors, *Back-propagation: Theory, Architectures, and Applications*, Hillside, NJ, 1994. Lawrence Erlbaum Associates.
- [8] P. Baldi and A.F. Atiya. How delays affect neural dynamics and learning. *IEEE Trans. on Neural Networks*, 5(4):612–621, July 1994.
- [9] E. Barnard and D. Casasent. Invariance and neural nets. *IEEE Trans. on Neural Networks*, 2:498–508, September 1991.

- [10] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. on Information Theory*, 39(3):930–945, 1993.
- [11] Y. Bengio, R. Cardin, and R. De Mori. Speaker independent speech recognition with neural networks and speech knowledge. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 218–225, Denver 1989, 1990. Morgan Kaufmann, San Mateo.
- [12] U. Bodenhausen. Learning internal representations of pattern sequences in a neural network with adaptive time-delays. In *International Joint Conference on Neural Networks*, volume 3, pages 113–118, San Diego, 1990. IEEE, New York.
- [13] U. Bodenhausen and A. Waibel. The tempo2 algorithm: Adjusting time-delays by supervised learning. In J. E. Moody R. P. Lippmann and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 155–161, Denver 1991, 1991. Morgan Kaufmann, San Mateo.
- [14] L. Bottou, F. Fogelman Soulié, P. Blanchet, and J.S. Liénard. Speaker-independent isolation digit recognition: Multilayer perceptrons vs. dynamic time warping. *Neural Networks*, 3:453–465, 1990.
- [15] O. Bowerman. *Forecasting and Time Series: An Applied Approach*. Duxbury Press, Belmont, CA, third edition, 1993.
- [16] G. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and control*. Holden-Day, San Francisco, CA, 1976.
- [17] T. H. Bullock, R. Orkand, and A. Grinnell. *Introduction to Nervous Systems*. W. H. Freeman and Company, San Francisco, 1977.
- [18] C.E. Carr and M. Konishi. Axonal delay lines for time measurement in the owl’s brain stem. *Proceedings of the National Academy of Sciences, USA*, 85:8311–8315, 1988.
- [19] D. Casasent and E. Barnard. Shift invariance and the neocognitron. *Neural Networks*, 3:403–410, 1990.
- [20] F. Chapeau-Blondeau and G. Chauvet. Stable, oscillatory, and chaotic regimes in the dynamics of small neural networks with delays. *Neural Networks*, 5:735–743, 1992.

- [21] D.S. Chen and R.C. Jain. A robust back propagation learning algorithm for function approximation. *IEEE Trans. on Neural Networks*, 5(3):467–479, May 1994.
- [22] M. A. Cohen. The construction of arbitrary stable dynamics in nonlinear neural networks. *Neural Networks*, 5(1):83–104, 1992.
- [23] M.A. Cohen, S. Grossberg, and D. Stork. Recent developments in a neural model of real-time speech analysis and synthesis. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 4, pages 443–453, San Diego, 1987. IEEE, New York.
- [24] A.C.C. Coolen and C.C.A.M. Gielen. Delays in neural networks. *Europhysics Letters*, 7:281–285, 1988.
- [25] N. E. Cotter and O. N. Mian. A pulsed neural network capable of universal approximation. *IEEE Trans. on Neural Networks*, 3:308–314, March 1992.
- [26] S. P. Day and D. S. Camporese. Continuous-time temporal back-propagation. In *International Joint Conference on Neural Networks*, volume 2, pages 95–100, Seattle, 1991. IEEE, New York.
- [27] S. P. Day and M. Davenport. Continuous-time temporal back-propagation with adaptive time delays. Neuroprose archive, Ohio State University. Accessible on Internet via anonymous ftp on archive.cis.ohio-state.edu, in pub/neuroprose/day.tempora.ps August, 1991.
- [28] S. P. Day and M. R. Davenport. Continuous-time temporal back-propagation with adaptive time delays. *IEEE Trans. on Neural Networks*, 4(2):348–354, March 1993.
- [29] J. E. Dayhoff. *Neural Network Architectures : An Introduction*. Van Nostrand Reinhold, New York, 1990.
- [30] J.E. Dayhoff. Synchrony detection in neural assemblies. in Biological Cybernetics, 1994. To appear.
- [31] J.E. Dayhoff and G.L. Gerstein. Favored patterns in spike trains. I. Detection. *Journal of Neurophysiology*, 49(6):1334–1348, June 1983.

- [32] J.E. Dayhoff and G.L. Gerstein. Favored patterns in spike trains. II. Application. *Journal of Neurophysiology*, 49(6):1349–1363, June 1983.
- [33] J.E. Dayhoff, D.-T. Lin, and P. A. Ligomenides. Neural network architectures and prediction applications. In *Workshop of CADx in Medical Imaging*, pages 3–4, Washington D.C., 1994. Georgetown University Medical Center.
- [34] J.E. Dayhoff, D.-T. Lin, and P. A. Ligomenides. Neural networks for biomedical prediction tasks. In *World Congress on Neural Networks*, volume 1, pages 119–120, San Diego, CA, 1994. INNS Press, New York.
- [35] J.E. Dayhoff, P. Palmadesso, and F. Richards. Developing multiple attractors in a recurrent neural network. In *World Congress on Neural Networks*, volume 4, pages 710–715, San Diego, CA, 1994. INNS Press, New York.
- [36] B de Vries and J.C. Principe. A theory for neural networks with time delays. In J. E. Moody R. P. Lippmann and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 162–168, Denver 1991, 1991. Morgan Kaufmann, San Mateo.
- [37] B. de Vries and J.C. Principe. The gamma model – a new neural net model for temporal processing. *Neural Networks*, 5:565–576, 1992.
- [38] S. Dehaene, J.-P. Changeux, and J.-P. Nadal. Neural networks that learn temporal sequences by selection. *Proceedings of the National Academy of Sciences, USA*, 84:2727–2731, 1987.
- [39] K. Doya and S. Yoshizawa. Adaptive neural oscillator using continuous-time back-propagation learning. *Neural Networks*, 2:375–385, 1989.
- [40] B. Doyon, B. Cessac, M. Quoy, and M. Samuelides. Control of the transition of chaos in neural networks with random connectivity. *International Journal Bifurcation and Chaos*, 3(2):279–291, 1993.
- [41] R. Eckmiller. Neural nets for sensory and motor trajectories. *IEEE Control Systems Magazine*, 9:53–59, 1989.

- [42] R. Eckmiller, J. Goerke, and J. Hakala. Neural networks for internal representation of movements. In R. J. Mammone and Y. Zeevi, editors, *Neural Networks, Theory and Applications*, pages 97–112, San Diego, CA, 1991. Academic Press.
- [43] D. Farmer and J. Sidorowich. Exploiting chaos to predict the future and reduce noise. In W.C. Lee, editor, *Evolution, Learning, and Cognition*, pages 277–330. World Scientific, Singapore, 1988.
- [44] J.D. Farmer. Chaotic attractors of an infinite-dimensional dynamical system. *Physica D*, 4:366–393, 1982.
- [45] K.-I. Funahashi. On the approximate realization of continuous mappings by neural network. *Neural Networks*, 2:183–192, 1989.
- [46] E. Gelenbe. Temporal behavior of neural networks. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 3, pages 267–276, San Diego, 1987. IEEE, New York.
- [47] A.P. Georgopoulos, A.B. Schwartz, and R.E. Kettner. Neuronal population coding of movement direction. *Science*, 233:1416–1419, 1986.
- [48] J. Ghosh and L. Deuser. Classification of spatio-temporal patterns with applications to recognition of sonar sequence. To appear in *Neural Representation of Temporal Patterns*, edited by E. Corey, Plenum, 1995.
- [49] L. Glass and D. Kaplan. Complex dynamics in physiology and medicine. In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction*, pages 513–527. Addison Wesley, Reading, MA, 1993.
- [50] Y.K. Goldberg and B.A. Pearlmutter. Using backpropagation with temporal windows to learn the dynamics of the CMU direct-drive arm II. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 356–363, Denver 1988, 1989. Morgan Kaufmann, San Mateo.
- [51] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

- [52] R.P. Gorman and T.J. Sejnowski. Learned classification of sonar targets using a massively-parallel network. *IEEE Trans. on Acoust., Speech, Signal Processing*, 36:1135–1140, 1988.
- [53] C.M. Gray, P. König, A.K. Engel, and W. Singer. Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338:334–337, 1989.
- [54] S. Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1:17 – 61, 1988.
- [55] M.R. Guevara, L. Glass, M.C. Mackey, and A. Shrier. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13, 1983. Reprinted in [4].
- [56] H. Gutfreund and M. Mézard. Processing of temporal sequences in neural networks. *Physical Review Letters*, 61:235–238, 1988.
- [57] P. Haffner and A. Waibel. Multi-state time delay neural networks for continuous speech recognition. In S. J. Hanson J. E. Moody and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 135–142, Denver 1992, 1992. Morgan Kaufmann, San Mateo.
- [58] N.Z. Hakim, J.J. Kauman, G. Cerf, and H.E. Meadows. Cursive script on line character recognition with recurrent neural network model. In *International Joint Conference on Neural Networks*, volume 3, pages 711–716, Baltimore 1992, 1992. IEEE, New York.
- [59] E.J. Hartman, J.D. Keeler, and J.M. Kowalski. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2:210–215, 1990.
- [60] Y. Hayashi. Oscillatory neural network and learning of continuously transformed patterns. *Neural Networks*, 7(2):219–231, 1994.
- [61] R. Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, Reading, MA, 1990.
- [62] T.L. Hemminger and Y.-H. Pao. Detection and classification of underwater acoustic transients using neural networks. *IEEE Trans. on Neural Networks*, 5(5):712–718, September 1994.

- [63] J. Hertz, A. Krough, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, Reading, 1991.
- [64] A. Herz, B. Sulzer, R. Kühn, and J.L. van Hemmen. Hebbian learning reconsidered: Representation of static and dynamic objects in associative neural nets. *Biological Cybernetics*, 60:457–467, 1989.
- [65] T.M. Heskes and S. Gielen. Retrieval of pattern sequences at variable speeds in a neural network with delays. *Neural Networks*, 5:145–152, 1992.
- [66] Y. Hirose, K. Yamashita, and S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, 1991.
- [67] M. W. Hirsch. Convergent activation dynamics in continuous time network. *Neural Networks*, 2:331–349, 1989.
- [68] J. Hoffman, J. Skrzypek, and J. Vidal. Cluster network for recognition of handwritten cursive script characters. *Neural Networks*, 6:69–78, 1993.
- [69] L. Holmström and P. Koistinen. Using additive noise in back-propagation training. *IEEE Trans. on Neural Networks*, 3(1):24–38, January 1992.
- [70] J.J. Hopfield and D.W. Tank. Neural architecture and biophysics for sequence recognition. In J.H. Byrne and W.O. Berry, editors, *Neural Models of Plasticity*, pages 363–377. Academic Press, San Diego, 1989.
- [71] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [72] W. Hsu and M. F. Tenorio. Plastic network for predicting the mackey-glass time series. In *International Joint Conference on Neural Networks*, volume 2, pages 941–946, Baltimore, 1992. IEEE, New York.
- [73] L. Hutton, V. Sigillito, and H. Egeth. Experiments on constructing a cognitive map: A neural network model of a robot that daydreams. In *International Joint Conference on Neural Networks*, volume 1, pages 223–227, Washington 1990, 1990. Lawrence Erlbaum, Hillsdale.

- [74] R.D. Jones, Lee Y.C., Barnes C.W., Flake G.W., Lee K., and Qian S. Function approximation and time series prediction with neural networks. Technical Report LA-UR-90-21, Los Alamos National Laboratory, New Mexico, NM, 1990.
- [75] R.D. et al. Jones. Nonlinear adaptive networks: A little theory, a few applications. Technical Report LA-UR-91-273, Los Alamos National Laboratory, New Mexico, NM, 1991.
- [76] G. Josin. Neural-space generalization of a topological transformation. *Biological Cybernetics*, 59:283–290, 1988.
- [77] J.D. Keeler. Basins of attraction of neural network models. In *AIP Conference Proceedings*, volume 151, pages 259–265. American Institute of Physics, 1986.
- [78] M. Kerszberg and A. Zippelius. Synchronization in neural assemblies. *Physica Scripta*, T33:54–64, 1990.
- [79] D. Kleinfeld. Sequential state generation by model neural networks. *Proceedings of the National Academy of Sciences, USA*, 83:9469–9473, 1986.
- [80] D. Kleinfeld and H. Sompolinsky. Associative neural network model for the generation of temporal patterns. *Biophys Journal*, 54:1039–1051, 1988.
- [81] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 3 edition, 1989.
- [82] Erwin Kreyszig. *Advanced Engineering Mathematics*. Wiley, New York, 1979.
- [83] A. Lapedes and R. Farber. How neural nets work. In W.C. Lee, editor, *Evolution, Learning, and Cognition*, pages 331–345. World Scientific, Singapore, 1988.
- [84] D.-T. Lin, J. E. Dayhoff, and P. A. Ligomenides. Adaptive time-delay neural network for temporal correlation and prediction. In *Intelligent Robots and Computer Vision XI: Biological, Neural Net, and 3-D Methods, Proc. SPIE*, volume 1826, pages 170–181, Boston, November, 1992.
- [85] D.-T. Lin, J. E. Dayhoff, and P. A. Ligomenides. A learning algorithm for adaptive time-delays in a temporal neural network. Technical Report SRC-TR-92-59, Systems Research Center, University of Maryland, College Park, Md 20742, May 15 1992.

- [86] D.-T. Lin, J. E. Dayhoff, and P. A. Ligomenides. Trajectory recognition with a time-delay neural network. In *International Joint Conference on Neural Networks*, volume 3, pages 197–202, Baltimore, 1992. IEEE, New York.
- [87] D.-T. Lin, J. E. Dayhoff, and Panos A. Ligomenides. Trajectory production with the adaptive time-delay neural network. *Neural Networks*. To appear, 1995.
- [88] D.-T. Lin, J. E. Dayhoff, and Panos A. Ligomenides. Adaptive time-delay neural network: Architecture and applications. In *International Symposium on Artificial Neural Networks*, pages A10–19, Hsin-Chu, Taiwan, Republic of China, December, 1993.
- [89] D.-T. Lin, J. E. Dayhoff, and Panos A. Ligomenides. Prediction of chaotic time series and resolution of embedding dynamics with the ATNN. In *World Congress on Neural Networks*, volume 2, pages 231–236, San Diego, CA, 1994. INNS Press, New York.
- [90] D.-T. Lin, P. A. Ligomenides, and J. E. Dayhoff. Learning spatiotemporal topology using an adaptive time-delay neural network. In *World Congress on Neural Networks*, volume 1, pages 291–294, Portland, OR, 1993. INNS, New York.
- [91] D.-T. Lin, P. A. Ligomenides, and J. E. Dayhoff. Learning with the adaptive time-delay neural network. Technical Report IRS-TR-93-49, Institute for Systems Research, University of Maryland, College Park, Md 20742, May 1993.
- [92] D.-T. Lin, P. A. Ligomenides, and J. E. Dayhoff. Spatiotemporal topology and temporal sequence identification with an adaptive time-delay neural network. In *Intelligent Robots and Computer Vision XII: Algorithms and Techniques, Proc. SPIE*, volume 2055, pages 536–545, Boston, September, 1993.
- [93] R.P. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1:1–38, 1989.
- [94] M.C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.
- [95] A. Maren. Neural networks for spatio-temporal recognition. In C. Harston A. Maren and R. Pap, editors, *Handbook of Neural Computing Applications*, pages 295–308. Academic Press, San Diego, CA, 1990.

- [96] L. Massone and E. Bizzi. A neural network model for limb trajectory formation. *Biological Cybernetics*, 61:417–425, 1989.
- [97] K. Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological Cybernetics*, 56:345–353, 1987.
- [98] J.L. McClelland, D.E. Rumelhart, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, Cambridge, 1986.
- [99] J.W. McClurkin, J.A. Zarbock, and L.M. Optican. Temporal codes for colors, patterns, and memories. *Cerebral Cortex*, 10:443–467, 1993.
- [100] W.C. Mead, R.D. Jones, Y.C. Lee, C.W. Barnes, G.W. Flake, L.A. Lee, and M.K. O’Rourke. Prediction of chaotic time series using CNLS-net – example: The Mackey-glass equation. Technical Report LA-UR-91-720, Los Alamos National Laboratory, New Mexico, NM, 1991.
- [101] J. Moody. Fast learning in multi-resolution hierarchies. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 29–39, Denver 1989, 1989. Morgan Kaufmann, San Mateo.
- [102] F.C. Moon. *Chaotic and Fractal Dynamics: An Introduction for Applied Scientists and Engineers*. John Wiley & Sons, Inc., New York, NY, 1992.
- [103] M.C. Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems*, 3:349–381, 1989.
- [104] M.C. Mozer. Neural network architectures for temporal sequence processing. In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction*, pages 243–264. Addison Wesley, Reading, MA, 1993.
- [105] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4–26, March 1991.
- [106] H. Nishimori, T. Nakamura, and M. Shiino. Retrieval of spatio-temporal sequence in asynchronous neural network. *Physical Review A*, 41:3346–3354, 1990.

- [107] N.H. Packard, J.P. Crutchfield, J.D. Farmer, and R.S. Shaw. Geometry from a time series. *Physical Review Letters*, 45:712–716, 1980.
- [108] D.B. Parker. Learning-logic. Technical Report 581-64, Office of Technology Licensing, Stanford University, 1982.
- [109] B.A. Pearlmutter. Learning state space trajectories in recurrent neural networks. In *International Joint Conference on Neural Networks*, volume 2, pages 365–372, Washington 1989, 1989. IEEE, New York.
- [110] B.A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.
- [111] F.J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59:2229–2232, 1987.
- [112] F.J. Pineda. Recurrent back-propagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1:161–172, 1989.
- [113] R. Pineda. Time dependent adaptive neural networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 710–718, Denver 1989, 1990. Morgan Kaufmann, San Mateo.
- [114] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–1497, 1990.
- [115] I. Poli and Jones R.D. A neural net model for prediction of noisy data. Technical Report LA-UR-91-671, Los Alamos National Laboratory, New Mexico, NM, 1991.
- [116] J.C. Principe, J.-M. Kuo, and S. Celebi. An analysis of the gamma memory in dynamic neural networks. *IEEE Trans. on Neural Networks*, 5(2):331–337, March 1994.
- [117] J.C. Principe and M.A. Motter. System identification with dynamic neural networks. In *World Congress on Neural Networks*, volume 2, pages 284–289, San Diego, CA, 1994. INNS Press, New York.
- [118] J.C. Principe and L.A. Turner. Word spotting with the gamma neural model. In *World Congress on Neural Networks*, volume 4, pages 502–505, San Diego, CA, 1994. INNS Press, New York.

- [119] S.-Z. Qin, Su H.-T., and J. McAvoy. Comparison of four neural net learning methods for dynamic system identification. *IEEE Trans. on Neural Networks*, 3(1):122–130, January 1992.
- [120] M. Reiss and J. G. Taylor. Storing temporal sequences. *Neural Networks*, 4:773–787, 1991.
- [121] C.L. Resch. New time delay neural network to distinguish exo-atmospheric warheads. Technical Report AM-93-E141, Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD, August 1993.
- [122] C.L. Resch. Effects of jitter on the ability of a time delay neural network to distinguish exo-atmospheric warheads. Technical Report AM-94-E010, Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD, January 1994.
- [123] U. Riedel, R. Kühn, and J.L. van Hemmen. Temporal sequences and chaos in neural nets. *Physical Review A*, 38(2):1105–1108, 1988.
- [124] R. Rohwer and B. Forrest. Training time-dependence in neural networks. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 2, pages 701–708, San Diego 1987, 1987. IEEE, New York.
- [125] H.L. Roitblat, P.W. Moor, P.E. Nachtigall, R.H. Penner, and W.L. Au. Dolphin echolocation: Identification of returning echos using a counterpropagation network. In *International Joint Conference on Neural Networks*, volume 1, pages 295–299, Washington 1989, 1989. IEEE, New York.
- [126] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.
- [127] M.W. Roth. Survey of neural network technology for automatic target recognition. *IEEE Trans. on Neural Networks*, 1(1):28–43, March 1990.
- [128] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533 – 536, 1986.
- [129] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, 1986.

- [130] M. Sato. A learning algorithm to teach spatiotemporal patterns to recurrent neural networks. *Biological Cybernetics*, 62:259–263, 1990.
- [131] M. Sato. A real time learning algorithm for recurrent analog neural networks. *Biological Cybernetics*, 62:237–241, 1990.
- [132] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.
- [133] G.M Shepherd. *Neurobiology*. Oxford, New York, 2 edition, 1988.
- [134] P. Simard and Y. Le Cun. Reverse TDNN: An architecture for trajectory generation. In S. J. Hanson J. E. Moody and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 579–588, Denver 1992, 1992. Morgan Kaufmann, San Mateo.
- [135] H. Sompolinsky, A. Crisanti, and H.J. Sommers. Chaos in random neural networks. *Physical Review Letters*, 61(3):259–262, 1988.
- [136] H. Sompolinsky and I. Kanter. Temporal association in asymmetric neural networks. *Physical Review Letters*, 57:2861–2864, 1986.
- [137] L. G. Sotolino, M. Saerens, and H. Bersini. Classification of temporal trajectories by continuous-time recurrent nets. *Neural Networks*, 7(5):767–776, 1994.
- [138] W.S. Stornetta, T. Hogg, and B.A. Huberman. A dynamical approach to temporal pattern processing. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 750–759, Denver 1987, 1988. American Institute of Physics, New York.
- [139] G. Z. Sun, H. H. Chen, Y. C. Lee, and Y. D. Liu. Time warping recurrent neural networks and trajectory classification. In *International Joint Conference on Neural Networks*, volume 1, pages 431–436, Baltimore, 1992. IEEE, New York.
- [140] D. C. Tam. Temporal-spatial coding transformation: Conversion of frequency-code to place-code via a time-delayed neural network. In *International Joint Conference on Neural Networks*, volume 1, pages 130–133, Washington, 1990. Lawrence Erlbaum, Hillsdale.

- [141] D. C. Tam and D. H. Perkel. A model for temporal correlation of biological neuronal spike trains. In *International Joint Conference on Neural Networks*, volume 1, pages 781–785, Washington, 1989. IEEE, New York.
- [142] D.W. Tank and J.J. Hopfield. Concentrating information in time: Analog neural networks with applications to speech recognition problems. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 2, pages 455–468, San Diego 1987, 1987. IEEE, New York.
- [143] D.W. Tank and J.J. Hopfield. Neural computation by concentrating information in time. *Proceedings of the National Academy of Sciences, USA*, 84:1896–1900, 1987.
- [144] M.D. Tom and M.F. Tenorio. A spatio-temporal pattern recognition approach to word recognition. In *International Joint Conference on Neural Networks*, volume 1, pages 351–355, Washington 1989, 1989. IEEE, New York.
- [145] N. B. Toomarian and J. Barhen. Learning a trajectory using adjoint functions and teacher forcing. *Neural Networks*, 5:473–484, 1992.
- [146] K.P. Unnikrishnan, J.J. Hopfield, and D.W. Tank. Connected-digit speaker-dependent speech recognition using a neural network with time-delayed connections. *IEEE Trans. on Signal Processing*, 39(3):698–713, 1991.
- [147] L. J. van der Van, P. Verschure, and P. Molenaar. A note on chaotic behavior in simple neural networks. *Neural Networks*, 3:119–122, 1990.
- [148] A. Waibel. Consonant recognition by modular construction of large phonemic time-delay neural networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 215–223, Denver 1988, 1989. Morgan Kaufmann, San Mateo.
- [149] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1:39–46, 1989.
- [150] A. Waibel and J. Hampshire. A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Trans. on Neural Networks*, 1:216–367, June 1990.

- [151] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition: Neural networks versus hidden markov models. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages 107–110, April 1988.
- [152] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. on Acoust., Speech, Signal Processing*, 37(3):328–339, 1989.
- [153] A. Waibel, K. J. Lang, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.
- [154] E. A. Wan. Temporal backpropagation for FIR neural networks. In *International Joint Conference on Neural Networks*, volume 1, pages 575–580, San Diego, 1990. IEEE, New York.
- [155] E.A. Wan. Time series prediction by using a connectionist network with internal delay lines. In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction*, pages 243–264. Addison Wesley, Reading, MA, 1993.
- [156] R.L. Watrous and L. Shastri. Learning phonetic feature using connectionist networks: an experiment in speech recognition. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, San Diego 1987, 1987. IEEE, New York.
- [157] A. Weigend, B. Huberman, and D. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3):193–209, 1990.
- [158] A.S. Weigend and N.A. Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley Publishing Company, Reading, MA, 1993.
- [159] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, Cambridge, MA., 1974.
- [160] P. Werbos. Maximizing long-term gas industry profits in two minutes in lotus using neural network methods. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(2):315–333, 1989.

- [161] P. J. Werbos. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, volume 78, pages 1550–1560, October 1990.
- [162] P.J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1:339–356, 1988.
- [163] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.
- [164] Y. Yao and W. J. Freeman. Model of biological pattern recognition with spatially chaotic dynamics. *Neural Networks*, 3(2):153–170, 1990.
- [165] M. Zak. Terminal attractors in neural networks. *Neural Networks*, 2:259–274, 1989.