Computers Ops Res. Vol. 18. No. 4, pp. 343–353, 1991
Printed in Great Britain.

0305-0548/91 $3.00 + 0.00

TRAINING AN ARTIFICIAL NEURAL NETWORK THE PURE PURSUIT MANEUVER

DON HOMMERTZHEIM[1,*], JOHN HUFFMAN[2,†] and IHSAN SABUNCUOGLU[3,‡]

[1] Industrial Engineering Department, Wichita State University, Wichita, KS 67208,
[2] Operations Research Unit, Boeing Military Airplanes, Wichita, KS 67277, U.S.A. and
[3] Industrial Engineering Department, Bilkent University, Ankara, Turkey

(Received October 1989; in revised form August 1990)

Scope and Purpose—Artificial neural networks attempt to emulate the massively parallel and distributed processing of the brain. They have the ability to learn, generalize, categorize, and self organize. They are being examined for a variety of problems that have heretofore been very difficult to solve/model with current serial type computing and processing. Neural networks applications are showing up in such diverse fields as manufacturing, speech recognition, risk analysis, military systems, and robotics. This paper will discuss the authors' attempts to train a neural network to fly a pure pursuit maneuver against a target. Discussions will focus on selecting the architecture of the network, selecting the training sets, the training, the failures (along with the reasons for the failures), and the successes.

Abstract—Artificial neural networks' ability to learn, categorize, generalize, and self organize make them potentially very useful for a variety of application areas. This paper discusses some experiences in attempting to teach a neural network how to perform the pure pursuit maneuver. Several problems were encountered in defining a proper training set. Three and four layer backpropagation networks were utilized to capture the input to output mappings.

INTRODUCTION

Artificial neural networks are currently being considered for a variety of application areas in the military, business, and manufacturing. The networks utilize an architecture and processing manner similar to the brain. Thus, some of the characteristics usually associated with the brain are exhibited in these networks. These include the ability to learn from examples, to generalize from situations, to classify examples into categories, and to self organize information. (Not all of the artificial neural network models/paradigms possess all of these characteristics.) The research of McCulloch and Pitts [18], Hebb [9], Rosenblatt [20], and Widrow and Hoff [26] provided the underpinnings of the field. Due to some limitations of the theory and some personal infighting (Minsky and Papert [19]), the research underwent a dormant stage until somewhat recently when there has been a significant resurgence of research in the area. This revival is due primarily to the work of Anderson [2], Grossberg [7], Hopfield [10], Kohonen [14], and Rumelhart and McClelland [22].

Neural networks are being used for a variety of applications. These include text-to-speech conversion (Sejnowski and Rosenberg [23]), robot control (Josin [12]), handwritten character recognition (Fukushima and Miyake [5]), optimization (Hopfield and Tank [11]), vision (Grossberg [8]), speech recognition (McClelland and Elman [17]), target recognition (Ruck [21]), manufacturing process monitoring (Sutton [24]), financial risk analysis (Collins, Ghosh, and Scofield [4]), and target tracking (Kuczewski [15]).

As evidenced by the above list of applications, neural networks are being examined for use in military type systems. A very active area of Operations Research is the evaluation of military

* Don Hommertzheim is Chairman and Professor of Industrial Engineering at Wichita State University. He received a B.Sc. and M.Sc. in Mathematics from Friends University and Wichita State University, respectively, and a Ph.D. in Industrial Engineering from the University of Arkansas. His research interests are simulation and neural networks.

† John Huffman is a Ph.D. student at Wichita State University in the Industrial Engineering Department. He is a lead engineer and systems analyst in the Operations Decision Support Group at the Wichita Division of the Boeing Commercial Airplanes Group. He holds a B.Sc. in Civil Engineering from Illinois Institute of Technology and an M.Sc. in Engineering Management Science from Wichita State University. His research interests include data bases, knowledge based systems, and neural networks.

‡ Ihsan Sabuncuoglu is an Associate Professor in the Industrial Engineering Department at Bilkent University, Ankara, Turkey. He received a B.Sc. and M.Sc. in Industrial Engineering from the Middle East Technical University and a Ph.D. in Industrial Engineering from Wichita State University. His research interests are simulation, knowledge based systems, flexible manufacturing systems, and neural networks.

systems and tactics associated with their use. A variety of techniques is utilized in these evaluations with simulation being one of the principal tools. Numerous simulation models have been written and used in this area. However, there are several important functions that these models have exhibited difficulty in modeling properly. These include command and control, sensor fusion, and pilot decision logic. While knowledge-based systems have been somewhat successful in capturing some of the decision making features of these situations, they may not always be the most appropriate tool. This is due mainly to the extensive demands for knowledge that these systems require. For instance, if a model was being developed for pilot decision logic during an air-to-air encounter, then a knowledge engineer would need to: (a) define a series of questions to ask the pilot; (b) have several sessions with the pilot asking the questions; and (c) structure and code the knowledge. Due to the quickness required of pilots in making these decisions in a real encounter, they do not have time to "think" but must respond "instinctively" to the situation. Therefore, many times the pilots (and the knowledge engineers) would have difficulty in elucidating and structuring their decision processes.

Physical simulators, with real pilots in-the-loop, are frequently used to evaluate systems and to test new tactics. But these systems are very expensive to build and to operate. A significant portion of the cost is associated with the pilots' time needed to operate these systems. One concept that has emerged is to capture the pilots' responses while they are operating these simulators and to use these responses as a training set for a neural network. It is thought that neural networks could be trained to emulate the decision process. Then these networks could be embedded in analytical simulation models, which in turn could be used in large studies. One advantage of this approach is that it does not require that the pilots' decisions be known explicitly.

This paper will discuss the authors' attempts to train a neural network to perform a pure pursuit maneuver. It will discuss the maneuver, how the training sets were defined, the network architectures that were selected, some problems encountered in the training, and results.

It should be noted that the pure pursuit maneuver is extremely simple mathematically and could be modeled with much simpler means than with neural networks. It was the authors' intent to first model the maneuver without any human induced error and later to have pilots using a physical simulator to provide more realistic training sets for this and other maneuvers. These training sets would have captured human errors, noise, and subtle nuances that are typically very difficult to model accurately with other tools. But, as will be discussed later in this paper, several problems arose in training the network for the simple pure pursuit case. Similar problems to these could be encountered in training any network. And since the initial training situation was relatively simple and of low dimensionality, it was possible to plot the functions being modeled and hence to understand why the network was experiencing difficulty in learning the maneuver. These same problems would have occurred by using the pilot generated data, but it would have been more difficult to identify their associated sources and consequently possible remedies. Much was learned by the authors through this exercise and it was felt that other researchers desiring to apply neural networks would benefit from these experiences.

PURE PURSUIT MANEUVER

The pure pursuit maneuver is one in which the pursuer is always pointed directly at the object being pursued. This is contrasted to a lead pursuit maneuver where the pursuer points a fixed number of degrees ahead of the object being pursued. In pure pursuit, the pursuer will always converge on the target (object being pursued) if the pursuer has a speed advantage. This is true for both maneuvering and non-maneuvering targets. In practice, there exists some limitations to executing this maneuver. For instance, due to g-limits the pursuer may not be able to perform as tight a turn as the maneuver requires (particularly at close ranges against a maneuvering target). Also, sensors aboard the pursuer may impose field-of-view limits which the pursuer must maintain the target within. These may consequently place restrictions on the turns that the pursuer can execute. Examples of the pure pursuit maneuver are shown in Fig. 1. The target is non-maneuvering and traveling at half the speed of the pursuer. The target is labeled with a T and the pursuer with a P in the figure (and some of the later figures). While it is customary to present movement from

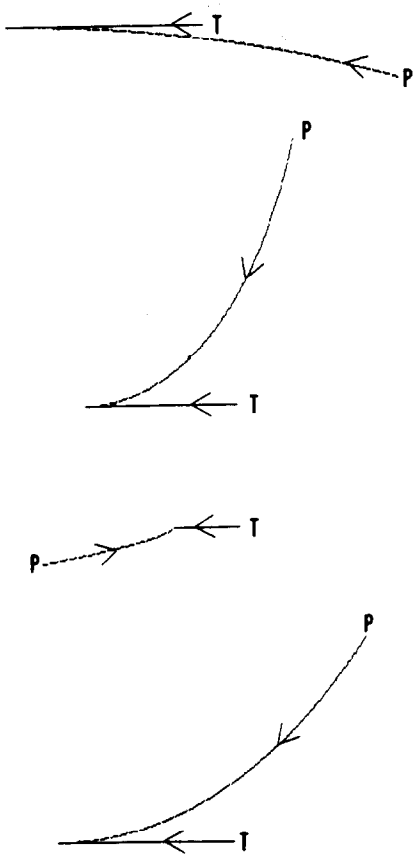


Fig. 1. Pure pursuit paths.

the left to the right, it was reversed in Fig. 1 and some of the subsequent figures in order to accentuate a problem in one of the models.

In two dimensions, the instantaneous pursuer heading can be analytically calculated as the arctangent of the difference in Y positions (between the target and the pursuer) divided through by the difference in the X positions plus 180° (so that the pursuer is pointed directly at the target as opposed to away from it). This procedure can easily be extended to three dimensions but we will restrict ourselves to two dimensions in this paper.

NEURAL NETWORK APPROACHES

Several approaches to training a neural network to learn the pure pursuit maneuver were tried. Multiple approaches were necessary because the first ones were not successful. These approaches will be discussed along with why some failed.

One of the first decisions to be made was the type (architecture) of neural network to be used. Since the network essentially has to perform an input to output mapping, a backpropagation network architecture was selected due to its ability to perform these mappings. Funahashi [6] recently proved that any continuous mapping can be approximated by a backpropagation network with at least one hidden layer when output functions for the network are sigmoid. A summary of the computations of processing elements and backpropagation is provided in the Appendix.

The first attempt (model 1) used two inputs (the X difference and the Y difference of the objects current positions) and one output (the desired heading of the pursuer). A three layer network (input, one hidden, and output layers) was utilized. Sixteen processing elements were selected for the hidden layer. This provided 65 connections (32 connections from the input to the hidden layer, 16 from the hidden to the output layer, 16 bias connections for the hidden layer elements, and 1 bias connection for the output layer element). It was felt that this would provide ample connections to capture the topology of the mapping plus not be so large that if the trained network was

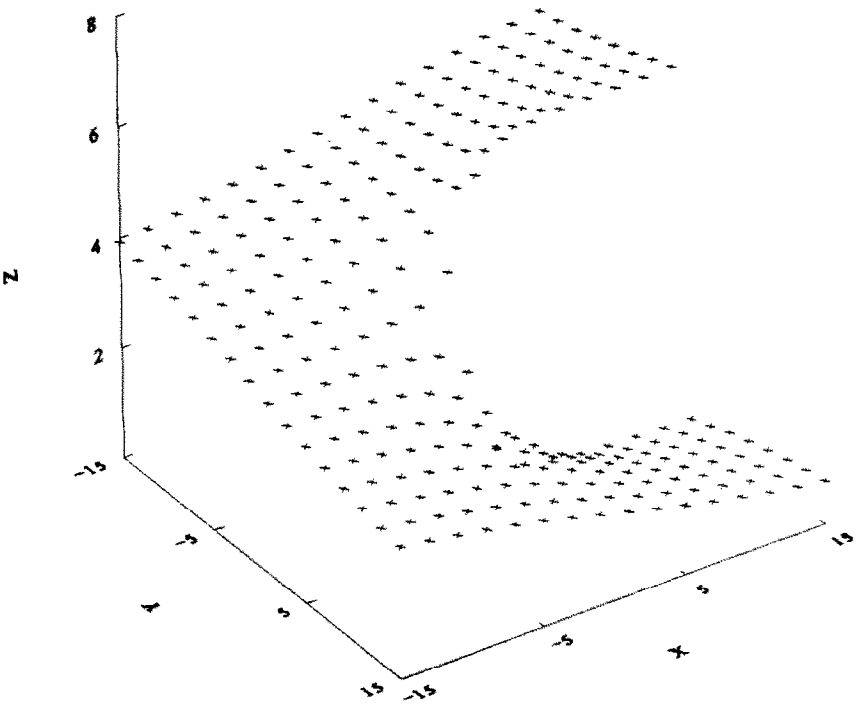


Fig. 2. Model 1 training set.

embedded in a simulation model it would not require excessive times to forward propagate through the network. Also, it is desirable that the network "generalizes" the mapping as opposed to memorizing the training set, which can occur if there are too many connections in relation to the number of examples in the training set.

For the training set, points were selected in a uniform grid pattern covering all four quadrants surrounding the target. The points went from -15 to 15 nautical miles (NM) in steps of 2 NM in both the X and Y directions. Thus there were a total of 256 examples (cases) in the training set. At each grid point, the X and Y values were used as the inputs and the desired heading for the pursuer (for the output) was calculated using the arctangent function. A three-dimensional plot of the training set is presented in Fig. 2. The X and Y axes refer to the X and Y differences in position (input variables) and the Z axis is the desired heading (output variable) in radians. The Y axis is reversed from its normal direction in this figure to provide a better view of the discontinuity (which is discussed later). It should be mentioned that this figure was produced after the network was trained when we were trying to understand its behavior.

The training of the network was performed with the ANSim (*ANSim* [3]) software package. ANSim was selected because it was available, easy to use, flexible, and provides graphical facilities to view the progress of the training. [Other packages, like NeuralWorks (Klimasauskas [13]) could have been used.] The network was trained for a total of 34,835 cycles. The average root mean squared (RMS) error was 0.025 (reasonably low) while the maximum unit error was 0.111 (somewhat high).

A program was written in Turbo Basic (*Turbo Basic* [25]) to accept an ASCII output file (which contained the trained weights and biases of the network) from ANSim and to drive a graphical animation of the paths taken by the target and the pursuer. When this animation was run for random starting points for the pursuer (random angle from the target but with an initial range of 15 NM), it was obvious that the network was not properly trained. Figure 3 presents some of the results of the encounters. (The target is moving from the right to the left and is represented with a solid line while the pursuer is represented by a dashed line.) The pursuer would initially perform a maneuver similar to pure pursuit but when the pursuer would close on the target it would turn towards the right of the screen (which was exactly the opposite direction that the target was heading).

There were several reasons identified to explain this behavior. First, the training set did not have any points on or very close to the X-axis. The training set essentially covered odd integer coordinate

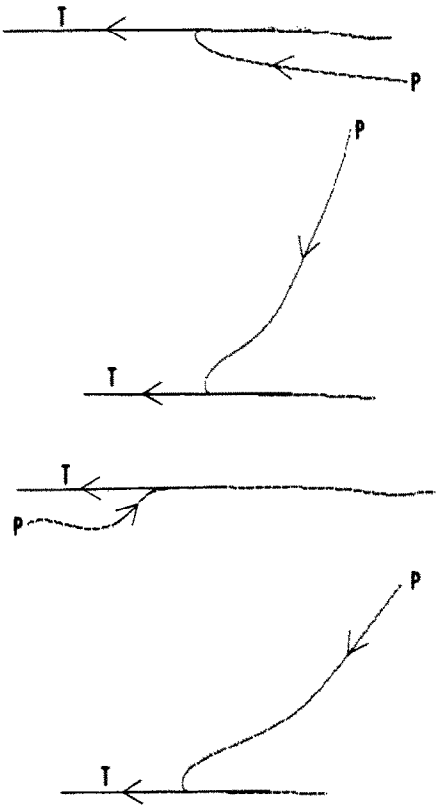


Fig. 3. Model 1 paths.

pairs like $(1, 3)$, $(-1, 1)$, etc., therefore when the pursuer was close to converging on the target there were not examples for it to have learned from. (Ahmad and Tesauro [1] showed that for the majority of functions the training was accelerated if border examples were included. These would be somewhat equivalent to training points near the axes in the pure pursuit problem.) Second, the gradient of the arctangent function is somewhat steep when Y is close to zero. Last, and probably the most important, there is a discontinuity in the heading angle function about the X-axis in the first quadrant. Y values slightly above the X-axis yield small positive angles, while Y values slightly below the same axis yield angles slightly less than 360°. Trigonometric functions have no difficulty in resolving this problem. However, the neural network, during the training, adjusts the connection weights such that a smooth continuous mapping of the response surface is effected. For a pursuer's position close to the target's X-axis, the network would yield a fitted value between 0 and 360, thus producing a value close to 180°. This resulted in the pursuer heading to the right side. At first thought, it would appear that the coordinate system could be rotated to a different orientation but this would not solve the problem but to just shift the problem to another position.

This model points up some factors that a new user of neural networks should be aware of when selecting a training set. Users cannot blindly train the network with a set of training examples without thinking about the characteristics of their problem. Very careful consideration must be given to determining the training set or the resulting network may not be trained for the problem that the user thinks that it is trained for.

A second model was attempted in which the outputs were the X and Y components of the normalized desired heading direction vector. In place of the desired heading for the pursuer, the desired incremental distances in the X and Y directions for the pursuer were to be learned by the neural network. It was thought that this approach would avoid the discontinuity problem that was encountered with the heading angle approach. Also, the training set was shifted to cover the even integer coordinates, thus covering the axes (the "end game" phase). A continuous plot of the X output component is presented in Fig. 4 (the Y output component plot is very similar to this plot). Notice the very steep gradient for Y values close to zero. A three layer backpropagation

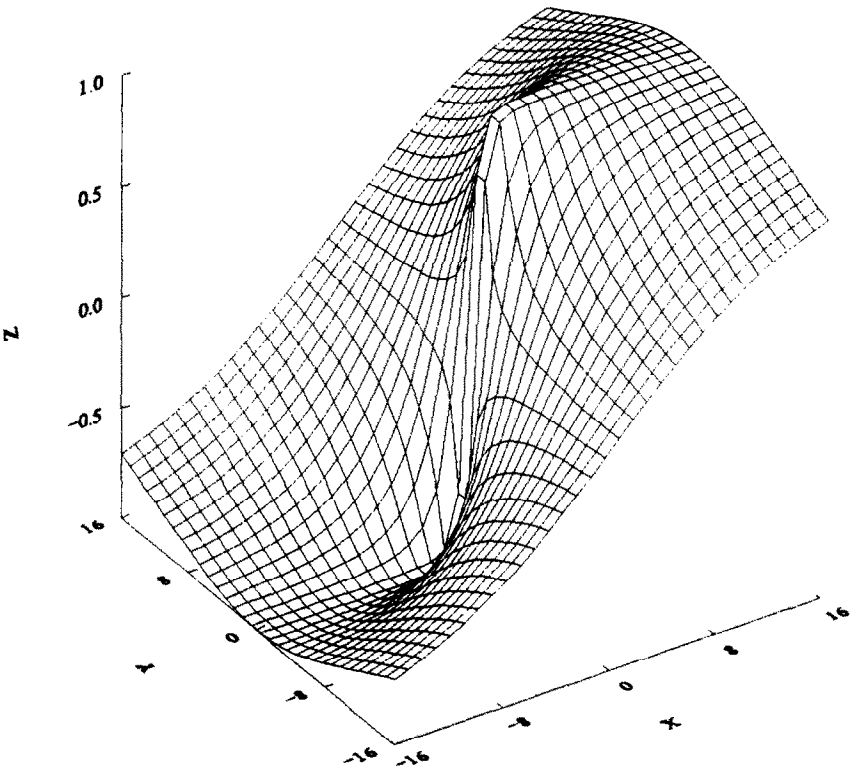


Fig. 4. Model 2 training function (X direction).

network with two elements in the input layer, nine in the hidden layer, and two in the output layer was selected for a total of 47 connections. The network was trained for 6489 cycles with a resulting average RMS error of 0.044 and a maximum output unit error of 0.331 (extremely high). The animation results for this approach were superior to the first approach but it appeared (from viewing the actual animations) that the velocity advantage of the pursuer was not maintained. This was due to the fact that the outputs were independent and therefore the magnitude of the resulting vector was not necessarily one. Also, the steep gradients mentioned above precluded a satisfactory fit, evidenced by the high maximum RMS error, by the neural network. A figure for this case is not presented because the problem mentioned above is not observable in a static drawing.

Several other attempts which involved stratified sampling in the training sets were tried with varying degrees of success but for the sake of brevity they are not reported here. The approach (model 3) that was successful and will be reported in some detail utilized a four layer backpropagation network and was trained by using random training points for only the first quadrant about the target. [By knowing the signs (plus or minus) of the X and Y differences and the resulting first quadrant angle, it is easy to determine the appropriate four-quadrant angle.] The network contained the same inputs and outputs as the first model but utilized two hidden layers each with six processing elements for a total of 67 connections. The use of two hidden layers was selected to accommodate a more complex region (Lippmann [16]) or as sometimes referred to as higher-order regularity. The training set contained 600 random points uniformly distributed over a region from 0 to 16 nautical miles in both the X and Y directions. Figure 5 shows a three-dimensional plot of the training set where Z is the desired heading in radians. It can be observed that the surface is relatively well-behaved with some steep gradients for small X values as Y approaches zero. The network was trained for 11,901 cycles which resulted in an average RMS error of 0.053 and a maximum unit error of 0.094.

The animation examples, shown in Fig. 6, adequately represent the intended behavior. Numerous cases (using different pursuer starting points) were tested and they all converged satisfactory. Longer training and a larger training set would have produced more superior results.

The animation model (not the network) was modified to allow for a maneuvering target (the target would periodically change its heading by a random amount, no g-limit restrictions were

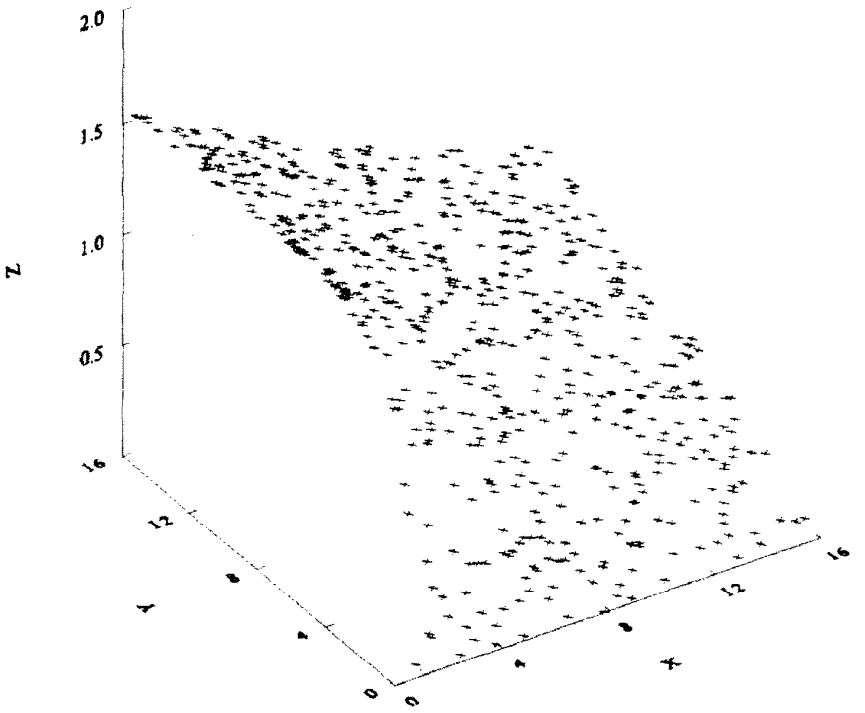


Fig. 5. Model 3 training set.

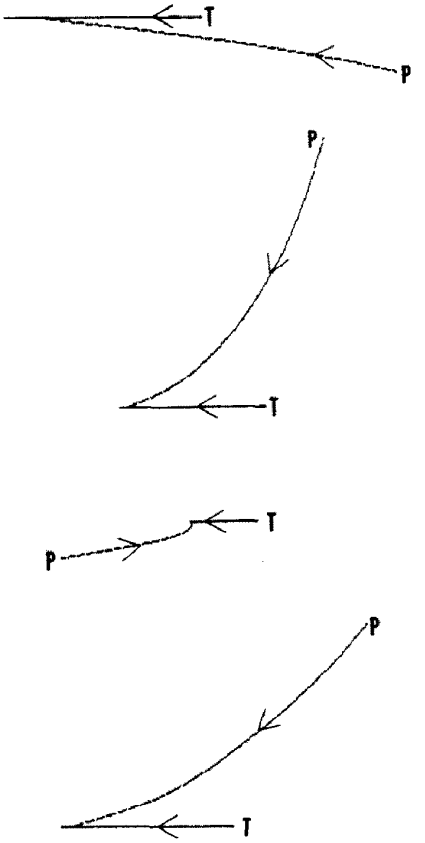


Fig. 6. Model 3 paths (nonmaneuvering target).

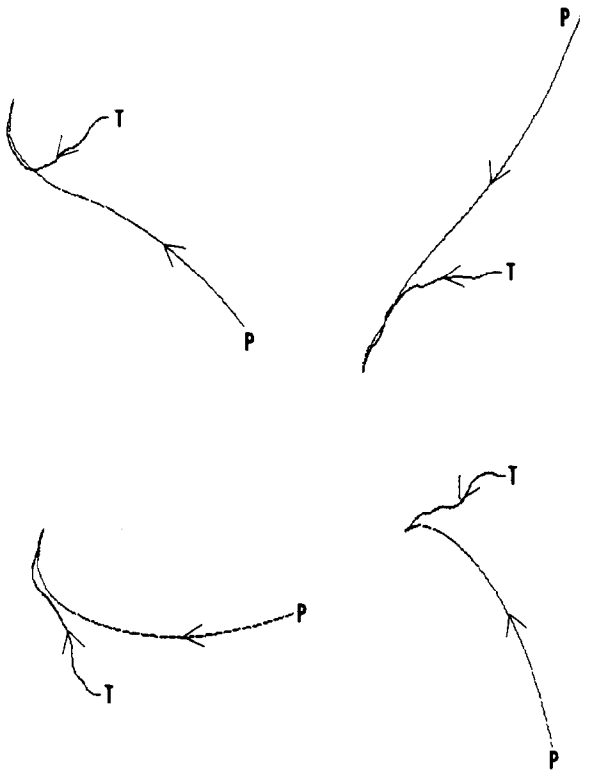


Fig. 7. Model 3 paths (maneuvering target).

placed on these maneuvers). Some of the resulting paths are shown in Fig. 7. Again, it can be noted that in every case convergence occurred.

As a final case, a model with three independent neural networks was developed and tested (model 4). One network determined the turn rate of the pursuer as a function of its velocity and its g-limit. A second network determined the desired first quadrant heading as a function of the X and Y position differences between the pursuer and the target (this is the same network that was described above and used for Figs 6 and 7), and a third network was used to translate the first quadrant angle into a full four-quadrant angle. Figure 8 presents the results of this model against a maneuvering target. (The pursuer's velocity was 1000 knots and it could pull 12 gs.) For demonstration purposes the pursuer was given a random heading at the start of the encounter. Notice that sometimes the pursuer would have to initially make a radius turn in order to head in the right direction. Also, if the pursuer either overshoots or undershoots the target (due to g-limit turn rates) then it would usually have to make a loop in order to come around again to attempt another convergence against the target.

Several items should be mentioned about training and using neural networks on a serial type computer. First, the parallel natural of the networks can be simulated by performing synchronous updating (for a backpropagation network). For example, inputs from the first layer can be weighted and summed to determine the activation level of the first processing element in the hidden layer. This same process can occur for each remaining element in the hidden layer, one at a time. After all the activation levels for the hidden layer elements are determined, then they in turn can be used to determine, in sequence, the activation levels for each of the elements in the output level. Training using the backpropagation algorithm of Rumelhart, Hinton, and Williams (Chap. 8 of Rumelhart and McClelland [22]) requires a forward propagation through the network for each example in the training set. The errors (differences between the observed and the expected output values) are summed over the training set and used then to modify the weights in the network by backpropagating through the network. This process can take a significant amount of time since: (1) there can be many examples in the training set; (2) there can be many connections in the network; and/or (3) training may be performed for many cycles (in order to reduce the errors). For example, using ANSim on a 16 MHz 386 microcomputer, the training may require 10 or more hours of training.

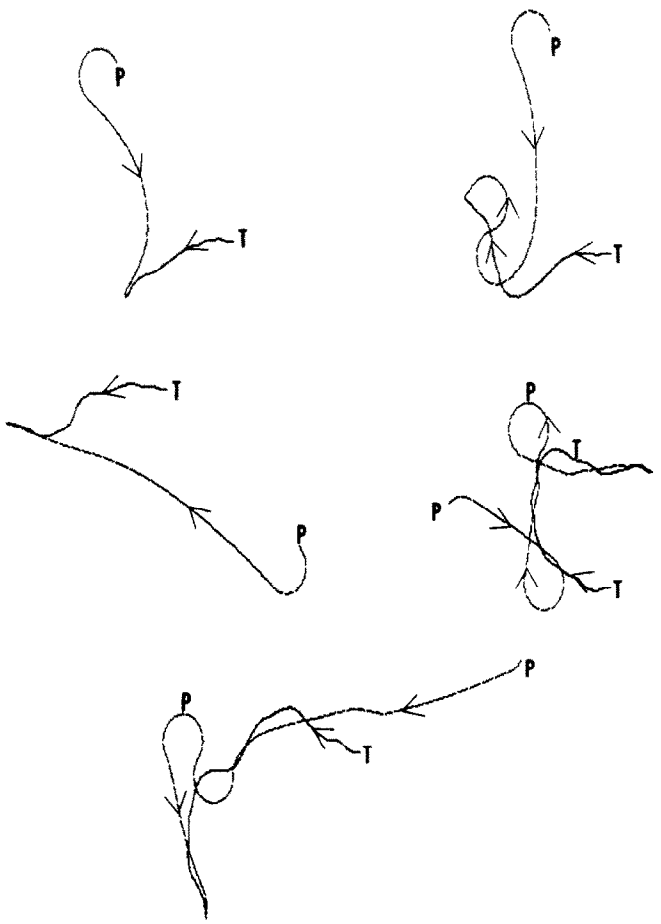


Fig. 8. Model 4 paths (g-limits and maneuvering target).

Approximating 6500 connections can be handled per second. But for the third model (67 connections and 600 cases in the training set) that was discussed, it required approximately, 6 sec per training cycle. Therefore, about 20 hr were required for the approximately 12,000 training cycles. Fortunately this can be performed during slack periods (nights, etc.) and the training can be interrupted with the current weights stored and restarted later.

OBSERVATIONS

Several observations can be made as a result of the experiences from training the networks described above. First, it is important that the training set be properly selected. Sufficient cases should be included in the training set to capture the input–output transformation. Randomly generated training examples tended to produce superior results. (Carefully selected nonrandom sets may work better, but it is very easy to bias the network if the training set is not properly selected.) Second, functions with discontinuities and steep gradients are difficult to model with neural networks. It is not always obvious that these may exist, especially for higher-order functions. Third, the number of connections should be large enough to properly model the mapping but not so large that it cannot generalize the mapping. Since the networks may be embedded in other models, forward propagating through an overly large network can significantly impede the processing speeds of the models (this is very critical in animated or real-time models/applications). Last, while training was performed using a conventional serial microcomputer, it can be very slow. There are accelerator boards available but they are very expensive. Also, specialized chips are being developed to truly accommodate the massive parallelism, but they are not available yet. This is a time of experimenting with artificial neural networks to explore their possible applications and subsequent

benefits. Their true power will hopefully be realized soon when new hardware is developed and is commonly available.

REFERENCES

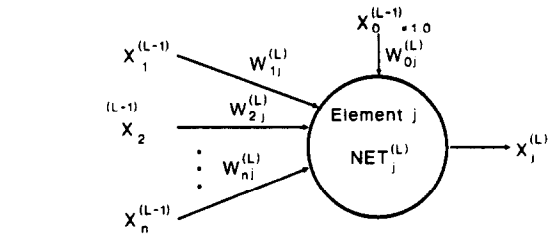
1. S. Ahmad and G. Tesauro, Scaling and generalization in neural networks: a case study. In *Advances in Neural Information Processing Systems I* (Edited by D. S. Touretzky), pp. 160–168. Kaufman, San Maeteo, Calif.
2. J. A. Anderson, A simple neural network generating an interactive memory. *Mathl Biosci.* **14**, 197–220 (1972).
3. *ANSim User's Manual.* Science Applications International Corporation, San Diego, Calif. (1988).
4. E. Collins, S. Ghosh and C. Scofield, Risk analysis. In *DARPA Neural Network Study*, Appendix G. AFCEA International Press, Fairfax, Va (1988).
5. K. Fukushima and S. Miyake, Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recogn* **15**, 455–469 (1982).
6. K. Funahashi, On the approximate realization of continuous mappings by neural networks. *Neural Netwks* **2**, 183–192 (1989).
7. S. Grossberg, How does the brain build a cognitive code? *Psychol. Rev.* **87**, 1–51 (1980).
8. S. Grossberg, Cortical dynamics of three-dimensional form, color, and brightness perception: II. Binocular theory. *Percepn Psychophys.* **41**, 117–158 (1987).
9. D. O. Hebb, *The Organization of Behavior.* Wiley, New York (1949).
10. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. *Proc. natn. Acad. Sci. U.S.A.* **79**, 2554–2558 (1982).
11. J. J. Hopfield and D. W. Tank, "Neural" computation of decisions in optimization problems. *Biol. Cybernet.* **52**, 141–152 (1985).
12. G. Josin, Integrating neural networks with robots. *AI Expert* **August**, 50–58 (1988).
13. C. C. Klimasauskas, *NeuralWorks: an Introduction to Neural Computing.* NeuralWare, Sewickley, Pa (1988).
14. T. Kohonen, *Self-Organization and Associative Memory.* Springer, Berlin (1984).
15. R. Kuczewski, Target tracking. In *DAPRA Neural Network Study*, Appendix L. AFCEA International Press, Fairfax, Va (1988).
16. R. P. Lippmann, An introduction to computing with neural networks. *IEEE ASSP Mag.* **4**, 4–22 (1987).
17. J. L. McClelland and J. L. Elman, Interactive processes in speech perception: the TRACE model. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Edited by D. E. Runelhart and J. L. McClelland), Vol. 2, pp. 58–121. MIT Press, Cambridge, Mass. (1986).
18. W. S. McCulloch and W. Pitts, A logical calculus of ideas immanent in nervous activity. *Bull. mathl Biophys.* **5**, 115–133 (1943).
19. M. Minsky and S. Papert, *Perceptrons.* MIT Press, Cambridge, Mass. (1969).
20. F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**, 386–408 (1958).
21. D. W. Ruck, Multitensor target detection and classification. Unpublished doctoral dissertation, Air Force Institute of Technology (1987).
22. D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* Vols 1–2. MIT Press, Cambridge, Mass. (1986).
23. T. J. Sejnowski and C. M. Rosenberg, Parallel networks that learn to pronounce English text. *Complex Syst.* **1**, 145–168 (1987).
24. R. S. Sutton, GTE process monitoring. In *DARPA Neural Network Study*, Appendix D. AFCEA International Press, Fairfax, Va (1988).
25. *Turbo Basic Owner's Handbook.* Borland International, Scotts Valley, Pa (1987).
26. B. Widrow and M. E. Hoff, Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pp. 96–104. IRE, New York (1960).

APPENDIX

Neural networks are comprised of very simple processing elements (units) that receive inputs from other units, perform a relatively simple computation based on these inputs, and output a single signal that is, in turn, used as input to other units. The processing elements are typically organized in layers. In a backpropagation type network, essentially the connections allow for feed forward type processing. Thus, each element in a layer immediately preceding the current layer is fully connected to every element in the following layer. Also, there are no connections between elements in a particular layer nor are there connections in the reverse direction of flow. In what is sometimes referred to as a three layer network, inputs (layer 1) are passed along through connections to the second (hidden) layer in which each element computs an output signal, these signals (outputs) are then passed on to the output (third) layer for similar processing. The input layer only acts like a buffer for the inputs and does not perform any computations.

Figure A1 shows the inputs, computations, and output for a typical processing element. Outputs from layer $L-1$ are the inputs to element j in layer L. Each input is weighted (multiplied) by the associated connection strength. These are summed (to form NET) and then passed through a discriminator function (usually nonlinear) to arrive at the output from the processing element. The weighted summation can be considered as an inner (dot) product between the inputs and the connection weights. Also, notice that an additional input (directly above the processing element in the figure) with a value always set to 1.0 is used as a bias term.

A typical three layer network is presented in Fig. A2. The purpose of training a network is to determine the most appropriate set of connection weights such that for any set of inputs (over the domain that the network was trained for) a correct (or nearly correct) set of outputs will be produced by the network at the output layer. Before training, the



$X_i^{(L-1)}$ = output of i-th element in layer L-1

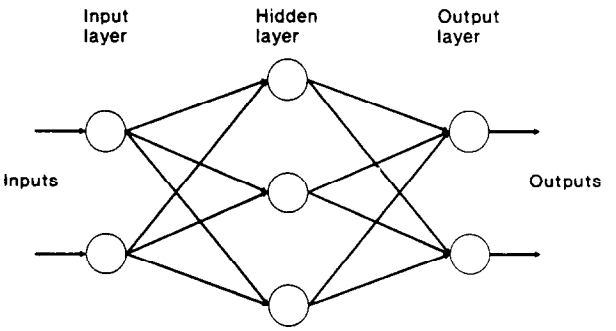
$W_{ij}^{(L)}$ = connection weight from element i in layer L-1 to element j in layer L

$NET_j^{(L)} = \sum_{i=0}^{n} W_{ij}^{(L)} X_i^{(L-1)}$ weighted summation for element j in layer L

$X_j^{(L)} = f(NET_j^{(L)})$ output (activation level) of j element in layer L

f = discriminator function (sigmoid, binary threshold, etc.)

Fig. A1. Processing element.



Characteristics of backpropagation network:
- feed forward
- fully connected between layers
- no connections within layer

Fig. A2. Three layer network.

connection weights are set to random values. During training, a training set which contains examples of corresponding inputs and outputs is presented in sequence to the network. Obviously, an untrained network will probably not produce outputs that are very close to the desired outputs. The difference between the desired and the actual output is used as an error signal to adjust the connection weights. A detailed description of the backpropagation algorithm used to perform these adjustments is described in Rumelhart and McClelland [22]. The algorithm utilizes a gradient descent procedure that minimizes the total root mean square error.