**Neural Network Implementation for Tron game**

Chris Dancy, Thom Conkling, Jef Harkay

Team ID: jph5076

**Abstract**

A multilayered neural network technique using back-propagation is presented to act as an intelligent player in a Tron-based game. Through the research of other neural network implementations, a system is derived which can confidently challenge other systems built for the Tron-based game.

**1. Introduction**

Artificial Intelligence has been integral to digital games over the past few decades. As new paradigms have appeared in the artificial intelligence community they have also been applied to game design. Some of the more popular implementations of artificial intelligence are: Rule-Based Expert Systems, Genetic Algorithms and Neural Networks.

This paper will specifically discuss an implementation of a neural network for the game Tron— a game adapted from the 1982 film Tron released by Walt Disney Studios [Blair et al., 1998]. In this game two players navigate an arena at constant speed, while trying to avoid the impassable boundaries created by themselves and their opponent. The objective is to be the last player standing. The neural network presented will be used to control one of the players in a simulated Tron game. Past papers have presented similar objectives such as optimal path analysis [Huse, 1990] and Air Combat Maneuvering [Schvaneveidt, 1992]. This paper presents a multilayered neural network to recognize the optimal path for a TRON player.

The next section will be an overview of applications of Neural Networks to games and similar systems. With this background established a description for the Tron game strategy and its Neural Network structure will be provided.

**2. Overview of Applications**

In this section, five different implementations of neural networks are discussed. A predator-prey approach will be discussed first, followed by an aerial combat implementation, neural networks in a mobile domain, learning in games, and finally an evolutionary neural network is examined.

**2.1 Predator-Prey Approach**

In the Predator-Prey model, a Neural Network is utilized to find an optimal path through an environment of obstacles to the target [Huse, 1990]. Artificial Neural Networks are chosen because of three important characteristics:

* Neural computing is computationally complete.
* A functional use of experiential knowledge
* Optimal performance

The input of the single-layered network is supplied by a ten by twenty grid where each (x, y) point is represented by an input neuron. Weights between the input neurons and the output neurons are fixed. A max function is used derive the output and train the network. The output is computed by finding the maximum product for each direction’s corresponding pair of connection weight and output value. After the network is trained, it is used to navigate the predator from its starting position to the prey using a minimally obstructed path.

One potential drawback to this model is the use of pre-computed weights between neurons. The implementation presented in our paper computes weights for neurons using back-propagation during the training of the network. Another disadvantage is that the prey is treated as a stationary target. In the Tron game both players will be moving serially. Advantages include a computational complexity of O(n) and adaptability to other applications with an initial state to a specific goal.

**2.2 Aerial Combat Implementation**

Another neural network application that is more similar to our own Tron game is aerial combat maneuvering, in which both players are actively trying to evade and attack each other. Though [Schvaneveidt, 1992] does not provide specific details of their neural network, the weighting data implies that a multi-layered network model with back-propagation is used. This was chosen after an initial comparison with a single layer Perceptron model. Though we did not find applicable network design information, the paper provides a discussion of the relative benefits of neural networks over rule-based systems.

**2.3 Neural Networks in Mobile**

**Domain**

A highly relevant application for neural networks is in mobile user tracking software for cell phone providers. In [Majumdar, 2005], hybrid neural networks are used to predict cellular towers. The idea is to calculate which tower will be in range based on a linear path approximation of the user’s current movements. A Self Organizing Feature Map – Multi-Layered Perceptron (SOFM-MLP) network performs this estimation. The SOFM network is devised for unsupervised learning. The training algorithm runs until it finds the nearest output node, then updates weights with respect to the nearest node and its neighborhood. After training, any input nodes will be grouped based on input patterns. The SOFM network separates the data into clusters, each of which is used to train a MLP network.

Unfortunately, the use of unsupervised learning networks is not useful for this project because Tron strategies can be readily understood and conveyed through supervised training.

**2.4 Learning in Games**

Despite all of the previous research stated, more concrete results were needed. In “Synthesizing Neural Networks for Learning in Games,” neural networks are trained to navigate agents through a maze more efficiently. To train these neural networks, traditional reinforcement learning techniques were enforced. These techniques search the whole state space while keeping track of the results to predict the final solution [Price et al., 2008]. Common demands that arise in video games are:

1. “A large state/action space.”
2. “Diverse behaviors.”
3. “Consistent individual behaviors.”
4. “Fast adaptation and sophisticated behaviors.”
5. “Memory of past states.”

[Price et al., 2008]

An evolutionary algorithm is used with neural networks to find the appropriate network configuration. To further simulate actual game playing, a real-time evolutionary algorithm is employed. In order to prevent a local maximum of optimality, the neural networks were first exposed to simple tasks, and then the difficulty was increased with each cycle.

The use of real-time training resulted in erratic behavior with some agents, which failed to satisfy the third common demand listed above. In order to combat this, the earlier generations of networks must be adept at the problems they’re trying to solve. Though this displays a good proof of concept for neural networks in gaming, the use of real-time evolutionary systems is out of scope for our project needs.

**2.5 An Evolutionary Neural**

**Network**

The “Co-evolution, Determinism and Robustness” paper describes exactly what the Tron based gaming system needs. The approach presented is a web-based version using Java, where human players play against their own evolving genetic programs [Blair et al, 1999]. Since this is an online experiment, these genetic programs have the opportunity to learn from thousands of users, which really benefits the co-evolution part of their project.

The authors set up the robots to have 8 sensors, with each one representing a direction to the nearest obstacle. A number from 0.0-1.0 is assigned to the sensor depending on if there are no walls (0.0), an obstacle further away (0.1-0.9), or an immediate obstacle (1.0). Calculations of which direction has the lowest weight are carried out, and depending upon the result, the robot chooses whether to go left, right, or straight.

The implementation consists of the following:

* A two-layered feed-forward neural network with 5 hidden units.
* 8 input sensors, which are the directions to each nearest obstacle from the robot.
* 3 output units, which are the resulting directions (left, right, and straight).

The champ neural network was taken and challenged against varying networks, called mutants. If one of these mutants won, the champ's weights are then updated in favor of the mutant's weights. This was carried out for thousands of games until little variation occurred between the champ and mutant that won.

Since this article is exactly what a Tron based gaming system needs, it really helps strengthen how to understand the implementation of Tron. Another good idea proposed here is the 8 sensor input system. By using this, the player has a field of vision for all of his bases. The only downfall is that this implementation looks like it would be extremely complicated. For this project, it seems like a co-evolution approach is unnecessary, but the neural network side of it is very promising.

**3. Implementation for Tron**

The neural network used will be a multilayered neural network employing back-propagation. Similar to [Blair et al, 1999] our network will have inputs to represent seven possible directions (one input per direction). The inputs will be representative of each of the surrounding blocks (see figure 1).

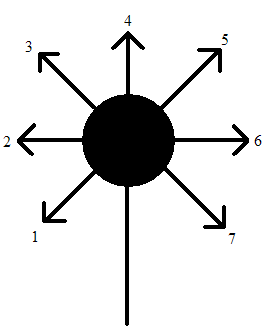


Figure 1: Directional input data.

The values for the inputs will be continuous from 0.0 to 1.0. An input will be 1.0 if there is not an obstruction within a predefined distance, 0.1 to 0.9 if there is an obstruction within the predefined distance, or 0.0 if there is an obstruction in the next block in that specific direction. If there is an obstruction more than one block from the current location, the input of a direction is determined by the distance of the obstruction divided by the predefined max distance.

The network weights will be initialized within the range suggested in “Artificial Intelligence A Guide to Intelligent Systems” [Negnevitsky, 2005]:

The output of the neurons in the hidden layer and output layer of the network will be computed using a sigmoid activation function:

The training will be represented through adjusted weights in the network. During training, the connection weights will be updated by back-propagation of output errors:

* First we calculate the error gradient for output layer:
  + Calculate weight corrections:
  + Update the weights at the output neurons:
* Calculate the error gradient for hidden layer:
  + Calculate weight corrections
  + Update the weights at the output neurons:

This process is repeated until the network is sufficiently trained.

**4. Conclusion**

Neural networks can be implemented for a wide range of applications, especially for problems dealing with pattern matching. Past research, like the work in aerial combat maneuvering [Schvaneveidt, 1992], has shown the adaptability of neural networks. Another important application of neural networks would be video games, due to their ability to react to partial knowledge.

The method presented in this paper is optimal for creating an intelligent agent in the Tron-based game. Because of this optimality, the network will be sufficient to challenge another intelligent agent in the Tron-based game. More complex algorithms, such as evolutionary neural networks, could be researched to better solve the problem but are currently out of the scope for this Tron implementation.

**5. References**

Blair, A. D., Sklar, E., and Funes, P. 1999. Co-evolution, Determinism and Robustness. In *Selected Papers From the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning* (November 24 - 27, 1998). B. McKay, X. Yao, C. S. Newton, J. Kim, and T. Furuhashi, Eds. Lecture Notes In Computer Science, vol. 1585. Springer-Verlag, London, 389-396.

Huse, S. M. 1990. Optimal path analysis using a predator-prey neural network model. In *Proceedings of the 3rd international Conference on industrial and Engineering Applications of Artificial intelligence and Expert Systems - Volume 2* (Charleston, South Carolina, United States). IEA/AIE '90. ACM, New York, NY, 1054-1062. DOI= http://doi.acm.org/10.1145/98894.99122

Majumdar, K. and Das, N. 2005. Mobile user tracking using a hybrid neural network. *Wirel. Netw.* 11, 3 (May. 2005), 275-284. DOI= http://dx.doi.org/10.1007/s11276-005-6611-x

Negnevitsky, M. 2005 *Artificial Intelligence: a Guide to Intelligent Systems*. 2nd. Addison-Wesley Longman Publishing Co., Inc.

Price, R. G. and Goodwin, S. D. 2008. Synthesizing neural networks for learning in games. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share* (Toronto, Ontario, Canada, November 03 - 05, 2008). Future Play '08. ACM, New York, NY, 189-191. DOI= http://doi.acm.org/10.1145/1496984.1497017

Schvaneveldt, Roger W., Goldsmith, Timothy E., Benson, Alan E., Waag, Wayne L., (1992), *Neural Network Models of Air Combat Maneuvering*, NEW MEXICO STATE UNIV LAS CRUCES. http://www.dtic.mil/srch/doc?collection=t3&id=ADA254653

Wise, E. 2004 *Hands-On AI with Java: Smart Gaming, Robots, and More*. McGraw-Hill/TAB Electronics.