Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2015, Article ID 157983, 8 pages
http://dx.doi.org/10.1155/2015/157983

Research Article

Generalisation over Details: The Unsuitability of Supervised Backpropagation Networks for Tetris

Ian J. Lewis and Sebastian L. Beswick

School of Engineering and ICT, University of Tasmania, Private Bag 87, Sandy Bay, TAS 7001, Australia

Correspondence should be addressed to Ian J. Lewis; ian.lewis@utas.edu.au

Received 19 January 2015; Accepted 1 April 2015

Academic Editor: Matt Aitkenhead

We demonstrate the unsuitability of Artificial Neural Networks (ANNs) to the game of Tetris and show that their great strength, namely, their ability of generalization, is the ultimate cause. This work describes a variety of attempts at applying the Supervised Learning approach to Tetris and demonstrates that these approaches (resoundedly) fail to reach the level of performance of hand-crafted Tetris solving algorithms. We examine the reasons behind this failure and also demonstrate some interesting auxiliary results. We show that training a separate network for each Tetris piece tends to outperform the training of a single network for all pieces; training with randomly generated rows tends to increase the performance of the networks; networks trained on smaller board widths and then extended to play on bigger boards failed to show any evidence of learning, and we demonstrate that ANNs trained via Supervised Learning are ultimately ill-suited to Tetris.

1. Introduction

Tetris, created by Pajitnov for the *Elektronika-60* machine in 1984 [1], is one of the most continuously popular video games of all time [2]. While many versions of the game have incorporated hand-crafted AI opponents, research has also been performed into applying biologically inspired methods such as Artificial Neural Networks (ANNs) to the problem.

ANNs, as first developed by McCulloch and Pitts [3], are a network of neurons [4], called nodes, that each fires when the sum of their inputs exceeds a certain threshold value. The simplest type of ANN is the single-layer feedforward network (perceptron network), in which every input is directly connected to every output.

It was realised that perceptron networks cannot be used to solve complex problems [3, 4]; however, this can be overcome by adding extra layers of neurons to the network, and connecting every neuron in each layer to every neuron in the next layer to create multilayer feed-forward ANNs.

Backpropagation, first developed by Bryson and Ho [5], is the most common learning method in multilayer feed-forward ANNs [4]. Backpropagation networks differ from perceptron networks in that, after the output is assessed, the net error is calculated and individual neurons are rewarded or punished depending on how much they contributed to the error. This procedure is performed after every epoch, and is a training technique known as Supervised Learning (SL). ANNs have been used to successfully solve problems that are trivial to humans but typically difficult to approach algorithmically, such as classification [6].

Every Tetris game must terminate at some point; it is statistically impossible to continue playing for an infinite amount of time, as a nontessellatable alternating sequence of S and Z pieces is inevitable (however, as the piece sequence must in be reality generated pseudorandomly, it is extremely unlikely that this sequence will be generated in practice) [7]. Farias and Roy [8] state that it would be possible to use dynamic programming to find an optimal strategy for Tetris but that it would be computationally infeasible due to the large state space of the game.

Breukelaar et al. [9] proved that even if the entire piece sequence is known in advance, Tetris is NP-complete. Importantly, they suggest that it is computationally unfeasible to compute the entire set of state spaces algorithmically and this justifies the use of machine learning approaches to approximate solutions to the optimal policy. To date, such

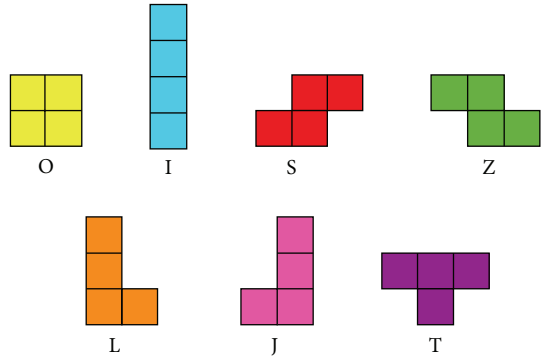

Figure 1: The standard naming convention for the seven Tetrominoes.

research has been primarily focused on the Reinforcement Learning (RL) approach or on restricted versions of the game (see Section 2.2).

In this paper, we attempt to train an ANN using Supervised Learning to play Tetris successfully. A number of strategies are considered, including training separate networks for each piece, adding random rows to the training set, and training the networks on a board of reduced width.

2. Background

The gameplay of Tetris consists of a random sequence of pieces that gradually fall down a gridded board at a rate of one square per game cycle. The pieces are all seven possible/Tetrominoes/: polyominoes that contain exactly four squares [10]. In standard Tetris, the set of Tetrominoes are created from exactly four squares and follow the standard naming scheme as shown in Figure 1.

The pieces, which can be rotated by any 90 degree factor during a move, fall one at a time until any further movement would cause a collision with the floor of the well or a block that has already been placed into the well.

When one of these collisions occur, the piece is placed on to the board, and cannot be subsequently moved. When a horizontal row in the well is completely filled with squares, it is removed from the board, and each column above it is dropped vertically by one row. Using the I piece, it is possible to clear up to four rows in a single move and most implementations have a scoring mechanism that awards additional points for solving more than one row in a single move, placing a premium on such moves.

The game ends when a block from a piece that has just been placed protrudes off the top of the game board into the $(n + 1)$th row.

2.1. Deterministic Algorithms for Playing Tetris. The most successful Tetris solving systems were implemented algorithmically rather than via machine learning methods [12]. These methods can be split up into one-piece methods, which do not take into account the next piece in the queue, and two-piece methods, which consider the next piece as well as the current one when calculating the optimal solution.

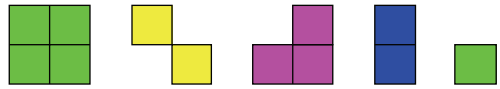

Figure 2: The restricted set of pieces used by Melax [11].

The best one-piece method was developed by Pierre Dellacherie, solving on average 650,000 rows [12]. Dellacherie's algorithm considers the current piece at each possible rotation and translation on the board and produces a heuristic evaluation derived from punishing the height a piece lands; row and column transitions; cellars (buried holes); and wells (deep valleys), and rewarding for erased lines.

El-Ashi [13], in an unreviewed self-published report, recently claimed to have refined Dellacherie's one-piece algorithm, improving it such that it solves on average 16,000,000 rows, an improvement of more than an order of magnitude.

2.2. Neural Network Approaches for Playing Tetris. As Tetris has a large number of board states ($1022^{19} \times 1023$ possible states) (as each square in the upper 19 rows can be either full of empty, but not all or none them, and the bottom row can be in all the same states, but also empty), it is practically impossible to define a nonlearning strategy [14]. Because of this, many simplifications have been made to the standard game as defined above in order to reduce the state space as discussed below.

Melax [11] drastically reduced the size and complexity of the pieces to those shown in Figure 2. This allowed him to show that artificial learning techniques, in particular RL, could be applied successfully to Tetris.

Bdolah and Livnat [14] reduced the size of the game board to six columns but allowed for an (essentially) infinite number of rows, using the justification that the limitation exists only to add an element of stress for a human player. Whether or not this is a desirable relaxation is debatable; one of the main challenges of Tetris is keeping the size of the stack low so that there is less chance of an unlucky sequence of pieces resulting in the game ending. They also restrict the set of available pieces to those of Melax. Their success metric is the height of the highest column after n moves. While they were unsuccessful in training a network that played Tetris exceptionally well, they verified the results of Melax and justified their additional optimizations. They discovered much about applying RL to Tetris, in particular evaluating different board representations, as discussed below.

Other research has been done on Tetris, including applying ANNs to the task of solving Tetris by Hashieda and Yoshida [15], and a subset of Tetris by Harter and Kozma [16]. RL has been applied to Tetris by Girgin and Preux [17] and Sarjant [18]. Driessens and Džeroski [19] looked at relational RL on a subset of Tetris. Grob et al. [20] considered Temporal Difference RL (TDRL) on a subset of Tetris. This body of research follows largely from Tesauro [21], who successfully used RL in conjunction with an ANN to learn Backgammon.

Because of the "Curse of Dimensionality" [22], it is important to carefully consider a suitable board representation. Melax encoded each square on the board as a binary

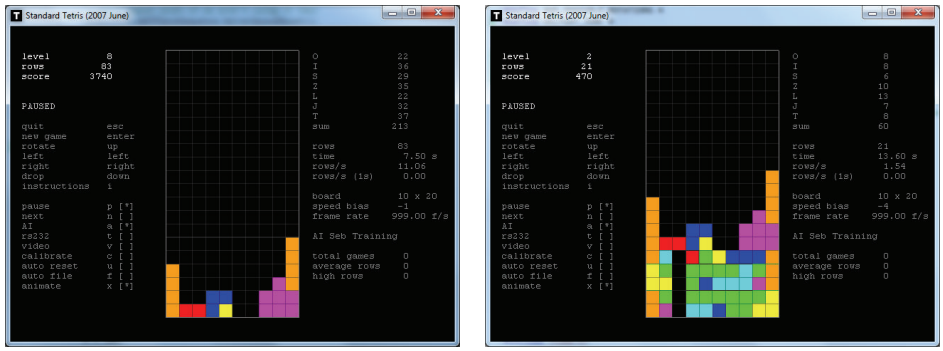

Figure 3: Two Tetris boards that should be treated in the same way by the player.

value [11]. Storing the whole state of the board after each move is clearly undesirable as the state space is enormous, and it would be wasteful to not take advantage of the optimizations that would stop the network, having to relearn the same patterns at different offsets, but it is surprisingly difficult to devise a condensed board representation without losing a significant amount of relevant information [23].

For example, it is clearly recognizable to a human player that the boards in Figure 3 should be treated in the same way (in most situations) essentially, but this cannot be expressed in many representations.

Bdolah and Livnat [14] ran experiments with the first of their two representations: the contour representation. The representation only considers the highest block in each column. It takes the height of the highest block in the first column to be zero and stores the height of the highest block in each other column as an integral offset from the height of this first column. This has the advantage of succinctly representing an approximation of the shape of the top of the board and the locations of holes and large gaps between the highest blocks in each column, on the board. They note that, for Tetris and other related problems, considering any information other than the state of the surface of the board only marginally contributes to the representation of the system, whilst greatly decreasing the chance of the ANN converging.

Counter-intuitively however, they found that an alternative approach, called Top Two Levels (TTL) representation, is superior to the contour representation. In TTL, the height of the highest block on the board is taken to be n, and the state of each square on the n and $(n-1)$th row is stored as a binary digit (and all the other board information is discarded). As this finding was discovered only during the final stages of this research, there was no time to perform comparative tests using this representation; see Section 4.1.

2.3. Summary. While some excellent deterministic algorithms have been devised to solve Tetris, previous work using machine learning techniques has seen relatively little success. Researchers have had to severely restrict the rules of the game to achieve learning, whilst results on the complete game have been poor. The ANNs that have been unsuccessfully applied to the problem have exclusively used the RL training technique. No research has been done previously on applying SL to Tetris. Lundgaard and McKee [24] suggest that this is due to the ambiguity between "good" and "bad" moves.

Regarding the research that has been carried out using RL, one of the major challenges was developing a way of representing the board in a condensed form (to decrease the change of the ANN diverging), whilst retaining ample relevant information about the state of the board [23].

This project will attempt to counter this by using Dellacherie's [12] excellent hand-coded one-piece Tetris solving algorithm, in conjunction with the contour board representation [14], to train an ANN using SL.

3. Methodology and Results

3.1. Tools. This project used the existing frameworks for both the ANN and Tetris implementations.

The ANN implementation was provided by the Fast Artificial Neural Network Library (FANN), a fast ANN implementation that supports multilayer backpropagation networks [25].

The Tetris implementation was provided by Fahey's Standard Tetris Library, as it was easy to extend and modify, and it already had an implementation of Dellacherie's one-piece Tetris solving algorithm [12].

3.2. Board Representation. Because Tetris is NP-complete [9] and has a large state space, it would be impractical to attempt to apply an ANN directly to a problem with such a large state space, so ways to condense the most important information into a compressed representation of the board must be considered.

This project uses a board representation that is functionally equivalent to the contour representation of Bdolah and Livnat [14]. The height of the highest block in the first column was normalised at 0, and the height of the highest block in each succeeding column was stored as an integral offset from 0. These offsets were clamped in $[-4, 4]$, as the height differences of more than four squares between the rows have no influence on the correct piece placement. This represents

arguably the most important information stored in the board and the shape of its skyline in a very compact way.

Dellacherie's algorithm only considers one factor that is not represented by the skyline representation: cellars and holes that lie under the skyline. However, it punishes moves that create cellars with a much higher weight than those of any other undesirable position.

3.3. Representation of Piece Rotations and Translations. Each possible rotation and translation combination was stored in a separate output (for a total of 40 outputs), in order to reduce the confusion for the network between the correct location to place a piece and the correct rotation for a piece. Each of these displacements was used as an input to the network, which was trained against the rotation and translation generated by Dellacherie's algorithm.

The game was run through n moves using Dellacherie's algorithm, storing the list of displacements (the contour representation of the board) at the start of each move, along with the move that Dellacherie chooses as the best one. These were used as input/output pairs when training the networks. In the following results, 30 random trials were run on each network, using the same piece sequence between approaches. In each training set, 2500 moves were made. The tests were halted after 2000 rows (approximately 5000 pieces) had been completed.

3.4. Experiments

3.4.1. With 10 Nets with O and I Pieces Only. Initially, the game was restricted to a subset of Tetris containing only the O and I pieces, as it was desirable to ascertain that the ANN could learn a small subset of Tetris before attempting to tackle the whole game. These pieces were chosen because they are the simplest to tessellate.

A single network on a board of width 10 was trained under a training set using training pairs generated by applying Dellacherie's algorithm on a random set of pieces. The network tended to perform extremely well, displaying some generalisation; often when it chose a different move to the "optimal" move found by Dellacherie's algorithm, it was an equally good choice.

The network performed much better than what would be expected from random chance. It completed more than 2000 rows in roughly half of the trials, but, in the trials where the game ended, an average of 599 rows was completed. Dellacheries' algorithm, on the other hand, completed more than 2000 rows on every trial (and, if left to run to completion, it would have been expected to solve approximately 650,000 rows [12]), after which it was stopped due to time constraints.

The ANN, however, often makes suboptimal moves. At some stage in the game, the network would inevitably make a terrible move, from which it was not able to recover. This is due to the fact that after the network makes one bad move, it is often presented with a large gap between blocks in adjacent columns on the next move, a case that was not presented in the training set because these simple mistakes were never made by Dellacherie's algorithm.

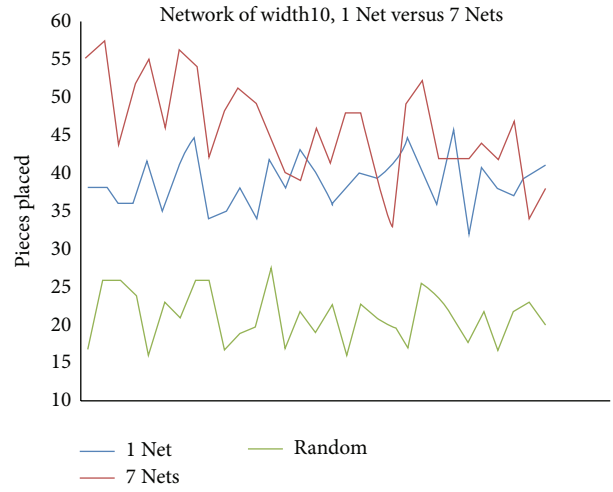

Figure 4: For a board of width 10, 7 Nets slightly outperform the 1 Net in terms of pieces placed with both outperforming randomly.

3.4.2. Junk Rows. To try to counter this problem, and hence improve the performance of the network, one or two individual rows with randomly placed blocks (*junk rows*) were added to the training set after approximately every twenty moves, and the network was retrained.

With junk rows in the training set, the ANN solved more than 2000 rows on every trial. It appears that adding the junk rows substantially improved performance as that forces the training set to include examples of poor board positions. Adding junk rows to the training set increased the amount of time the network took to train substantially, but the increase in performance seems to justify this.

3.4.3. 1 Net versus 7 Nets. When considering the optimal move from the set of possible moves at each turn, the piece that is to be placed clearly has a large impact on where it should be placed. After training a network of width 10 using all seven pieces and having little success, a new strategy was devised.

From this point, two different types of ANN were trained. The first was trained on the complete training set, learning the moves of all seven pieces: *1 Net*. In the second split, the training was set up into the moves made by each of the seven piece types and trained a separate ANN for each, resulting in a total of seven ANNs: *7 Nets*.

3.4.4. Networks of Width 10 on the Full Set of Seven Pieces. These two approaches were subsequently trained on the full set of seven pieces. Both approaches performed substantially better than what could be explained by chance (Figure 4), but they never manage to place more than 60 pieces in a single game.

On average, the 7 Nets outperformed the single net on both number of pieces placed (45 to 38) and number of rows completed (3 to 1), respectively and substantially outperformed random piece placement, which placed 21 pieces and solved 0 rows on average.

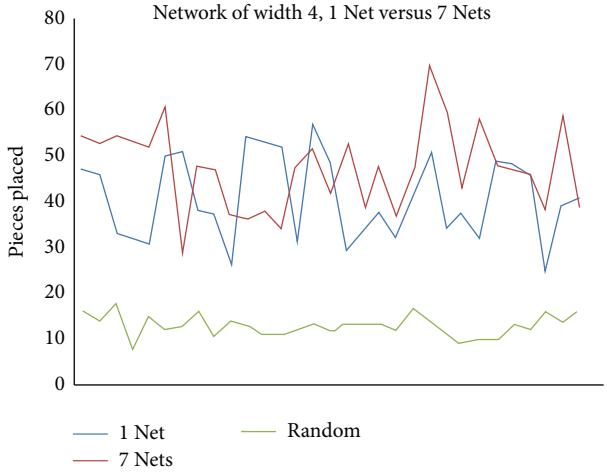

Figure 5: For a board of width 4, 7 networks slightly outperform 1 network (in terms of pieces placed) with both outperforming randomly.

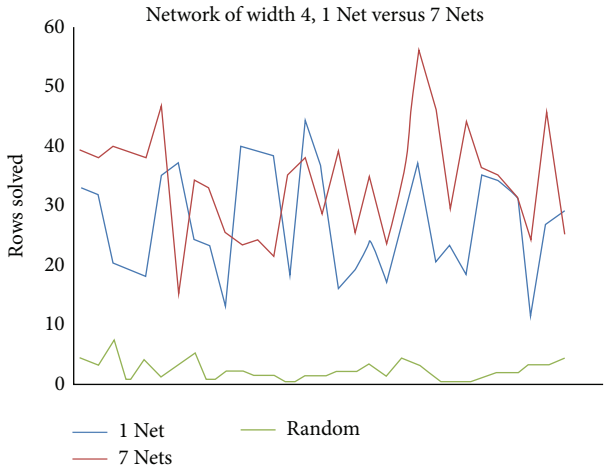

Figure 6: For a board of width 4, 7 Nets slightly outperforms 1 Net (in terms of rows solved) with both outperforming randomly.

3.4.5. Board Width Restriction. As the networks were failing to converge when trained on a board of width 10, the width of the board used in the training set was restricted to four squares, and the two approaches were retrained and retested.

Again, the 7 Nets were slightly more successful than 1 Net; the 7 Nets placed on average 47 pieces and completed 33 rows, whilst the single network placed 40 pieces and solved 27 rows. Both approaches performed substantially better than random piece placement (see Figures 5 and 6) but they are still massively worse than hand-coded techniques.

When junk rows were added to these training sets, 1 Net performed better than 7 Nets, solving an average of 41 pieces and 27 rows compared to 36 rows and 23 pieces. Both outperformed random piece placement. However, no improvement was seen over the equivalent networks that were trained without junk (Figure 7).

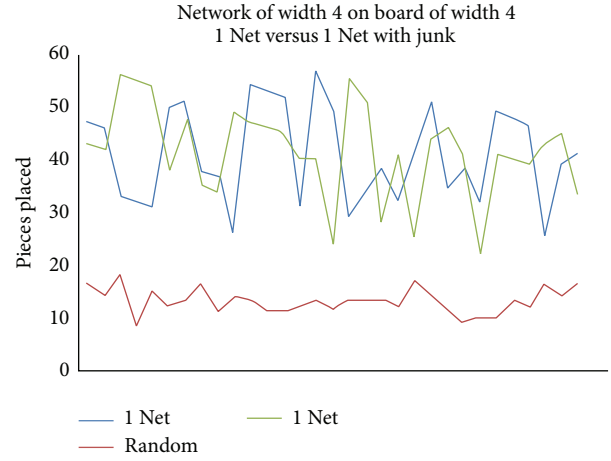

Figure 7: For a board of width 4, networks trained with or without junk performed comparably.

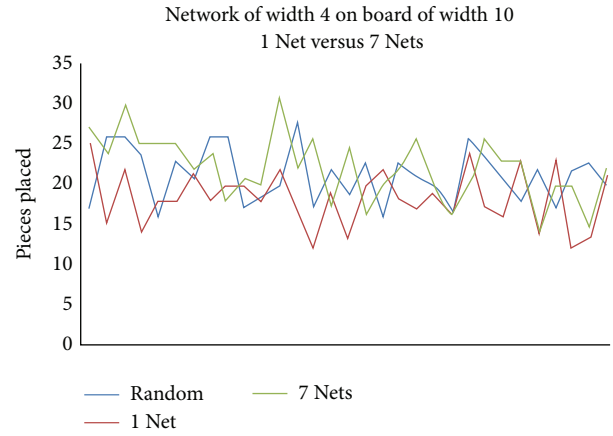

Figure 8: Networks trained for a board of width 4 applied multiple times across a board of width 10 performed no better than random.

3.4.6. Reduced Width Network Applied to Full Board. These networks were then tested on boards of width 10. This was achieved by subdividing the board horizontally into its seven component $(4 \times n)$ grids, applying the network to each grid in turn, and choosing the strongest output of all the ANNs.

This significantly decreased the time it took the networks to train, and it was hoped that the results similar to the previous tests could be achieved.

However, this approach was unsuccessful. Neither networks performed better than what would be expected by chance (Figure 8) and both performed substantially worse than the networks that were specifically trained to play on a board of width ten (see Figures 9 and 10). On average, 1 Net placed 18 pieces and solved 0.00 rows, while 7 Nets placed 22 pieces and solved 0.16 rows. Random piece placement achieved an average of 21 pieces and 0.00 rows.

These networks were then retrained, with random junk added to the training set. As can be seen from Figure 11, little improvement in the number of pieces placed was made. On average, 1 Net and 7 Nets both placed 24 pieces, compared

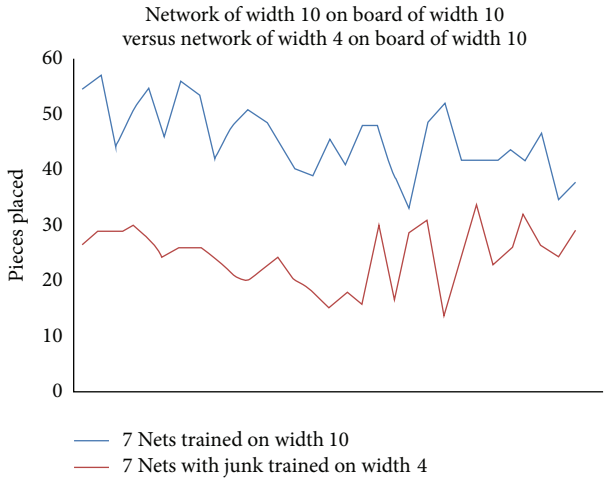

Figure 9: Networks trained for a board of width 4 applied multiple times across a board of width 10 performed substantially worse (in terms of pieces placed) than networks trained for a board of width 10.

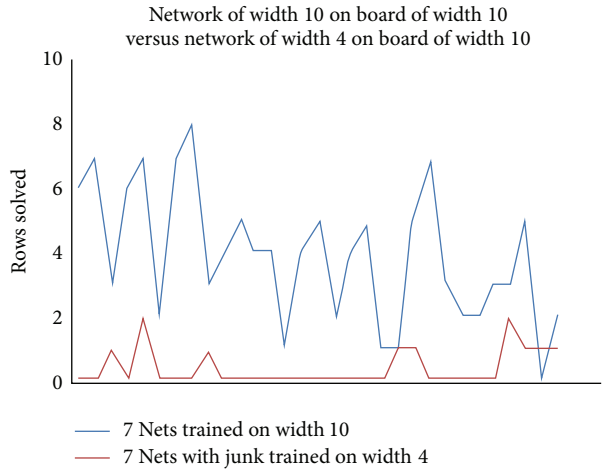

Figure 10: Networks trained for a width of 4 boards applied multiple times across a width of 10 boards performed substantially worse (in terms of rows solved) than a network trained for a width of 10 boards.

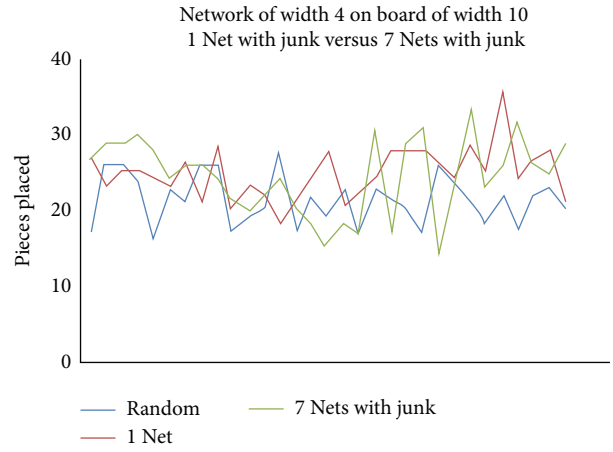

Figure 11: The addition of junk did not increase the number of pieces places for retrained networks.

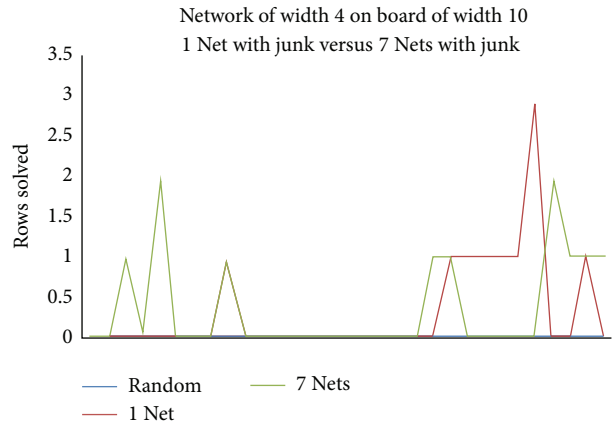

Figure 12: The addition of junk did increase the number of rows solved for retrained networks.

to randomly placing pieces on the board, which placed 21. However, a slight improvement was observed in the number of rows solved. On average, 1 Net solved 0.32 rows, and 7 Nets solved 0.35 rows, a doubling of effectiveness (Figure 12).

3.5. Further Discussion. As also reported by Bdolah and Livnat [14], this project found that the size of the state space had by far the biggest influence on the time it took the network to achieve learning. Reducing the representation of the board from the full $(1022^{19} \times 1023)$ states to the contour representation of 99 possible states made it possible to train the ANNs—convergence would not have been achieved without this drastic reduction to the state space.

In examining the situations in which the ANNs deviated from the correct move (i.e., one that would be made in that circumstance by Dellacherie's algorithm), some interesting effects were noted and some conclusions can be drawn from these.

Errors are often catastrophic in Tetris, as they require a long sequence of correct play to repair. The lack of representation of cellars only exacerbates this problem as the ANNs are not actively seeking to intelligently regain access to such regions.

The ANNs often contained many outputs of similar value that sometimes represented a group of similar moves but more often represented entirely dissimilar moves. This could result in good moves never being made as they are beaten out by other moves that are only slightly better.

Dellacherie's algorithm considers cellars when deciding on the best move, and these are not represented in the contour form. It is likely that, in many circumstances, the ANNs were presented with two "correct" placements for the same input (i.e., two different boards with cellars with identical contours).

The selected move was often close to correct in that a piece would be placed only one column away from the optimal

column, or would be in the correct column, but with the wrong rotation.

When presented with two similar board layouts with vastly different best placements, ANNs tend to produce a combination of those two placements. This property of generalisation is one of the great strengths of ANNs, but in Tetris, moves that lie "between" good moves are typically terrible ones.

4. Conclusion

This paper shows that although supervised ANNs exhibit some learning for Tetris, the properties of the game are ill-suited to such a learning approach. Supervised ANNs are far too susceptible to inappropriate generalisations and, as such, prone to catastrophic errors in piece placement.

The large state space requires a suitable board representation and the contour representation of Bdolah and Livnat [14] was found to be lacking, as it is unable to represent important information in the board, such as cellars.

Despite the general failure of the ANNs to learn, this work attempted some interesting alternative design approaches. It was found that adding junk rows to the training set favorably improves the performance of the networks, especially the number of rows completed.

Training a separate network for each piece, as opposed to a single network that deals with all the pieces, also improved the performance. Reducing the training sets to boards of width 4, and sequentially applying them to boards of width 10, though drastically reducing training time, resulted in substantially worse performance than training the networks specifically on boards of width 10; no evidence of learning was shown with this approach.

4.1. Further Work. Despite acknowledging that SL is unlikely to be capable of learning Tetris to the level of playing compared with that of hand-crafted approaches, there is a still scope for further work.

The approaches outlined in this paper could be attempted in conjunction with Melax's [11] restricted piece set in order to compare the success of RL versus SL.

As Bdolah and Livnat [14] showed that their TTL board representation outperformed the contour representation, ad one could attempt to train an ANN using SL with this technique. It is expected that this would substantially improve the performance of the networks considered in this paper as it would directly address some of the issues outlined above.

It would be interesting to use RL or a genetic algorithm to evolve a better set of weights for the factors considered by Dellacherie's algorithm. This is essentially an optimisation problem, with the fitness function being the amount of rows completed after a whole game has been played, so it is well suited to either of these approaches. First, one could attempt to derive the metrics of different board positions: the amount of holes, board height, and gaps between columns. This output could be sent to another algorithm that would attempt to learn the weights associated with each metric and output a heuristic evaluation of the piece placement. This is essentially the problem that has been discussed in this paper, broken down into its core components. If a good Tetris solver was unable to be created using this method, it would reveal interesting information about the applicability of ANNs.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] A. Pajitnov, *Tetris*, Spectrum HoloByte, Alameda, Calif, USA, 1985.

[2] P. Franklin, *At 25, Tetris Still Eyeing Growth*, 2009, http://www.reuters.com/article/2009/06/02/us-videogames-tetris-idUSTRE5510V020090602.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[4] S. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*, Pearson Education, Cranbury, NJ, USA, 3rd edition, 2010.

[5] A. E. Bryson and Y. C. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*, Blaisdell, Waltham, Mass, USA, 1969.

[6] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Addison Wesley, Essex, UK, 2004.

[7] J. Brzustowski, *Can you win at Tetris? [M.S. thesis]*, Department of Mathematics, University of British Columbia, 1992.

[8] V. F. Farias and B. V. Roy, "Tetris: a study of randomized constraint sampling," in *Probabilistic and Randomized Methods for Design under Uncertainty*, pp. 189–201, Springer, London, UK, 2006.

[9] R. Breukelaar, E. D. Demaine, S. Hohenberger, H. J. Hoogeboom, W. A. Kosters, and D. Liben-Nowell, "Tetris is hard, even to approximate," *International Journal of Computational Geometry & Applications*, vol. 14, no. 1-2, pp. 41–68, 2004.

[10] S. W. Golomb, *Polyominoes*, Allen & Unwin, 1965.

[11] S. Melax, *Reinforcement Learning Tetris Example*, 1998, http://www.melax.com/tetris.html.

[12] C. P. Fahey, *Tetris*, 2003, http://colinfahey.com/tetris/tetris.html.

[13] I. El-Ashi, El-Tetris—An Improvement on Pierre Dellacherie's Algorithm, 2011, http://ielashi.com/el-tetris-an-improvement-on-pierre-dellacheries-algorithm/.

[14] Y. Bdolah and D. Livnat, *Reinforcement Learning Playing Tetris*, 2000, http://www.math.tau.ac.il/~mansour/rl-course/student_proj/livnat/tetris.html.

[15] T. Hashieda and K. Yoshida, "Online learning system with logical and intuitive processings using fuzzy Q-learning and neural network," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 1, pp. 13–18, IEEE, July 2003.

[16] D. Harter and R. Kozma, "Task environments for the dynamic development of behavior," in *International Conference Computational Science (ICCS '01)*, vol. 2074 of *Lecture Notes in Computer Science*, pp. 300–309, Springer, Berlin, Germany, 2001.

[17] S. Girgin and P. Preux, "Feature discovery in reinforcement learning using genetic programming," in *Genetic Programming*, vol. 4971 of *Lecture Notes in Computer Science*, pp. 218–229, Springer, Berlin, Germany, 2008.

[18] S. J. Sarjant, *Creating reinforcement learning tetris AI [Bachelor of computer graphic design with honours]*, University of Waikato, Hamilton, New Zealand, 2008.

[19] K. Driessens and S. Džeroski, "Integrating guidance into relational reinforcement learning," *Machine Learning*, vol. 57, no. 3, pp. 271–304, 2004.

[20] A. Grob, J. Friedland, and F. Schwenker, "Learning to play Tetris applying reinforcement learning methods," in *Proceedings of the European Symposium on Artificial Neural Networks—Advances in Computational Intelligence and Learning*, Bruges, Belgium, April 2008.

[21] G. Tesauro, "Temporal difference learning and TD-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[22] R. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, NJ, USA, 1961.

[23] D. Carr, *Applying reinforcement learning to Tetris [Bachelor of Science (Honours) Thesis]*, Department of Computer Science, Rhodes University, Grahamstown, South Africa, 2005.

[24] N. Lundgaard and B. McKee, *Reinforcement Learning and Neural Networks for Tetris*, 2006, http://www.cs.ou.edu/~amy/courses/cs5033_fall2007/Lundgaard_McKee.pdf.

[25] S. Nissen, Fast Artificial Neural Network Library (FANN), 2009, http://leenissen.dk/fann/wp/.