2015 IEEE Symposium Series on Computational Intelligence

Temporal Difference Learning for the Game Tic-Tac-Toe 3D: Applying Structure to Neural Networks

Michiel van de Steeg
Institute of Artificial Intelligence
and Cognitive Engineering
University of Groningen, The Netherlands
Email: m.vd.steeg90@gmail.com

Madalina M. Drugan
Department of Computer Science
Technical University of Eindhoven
The Netherlands
Email: m.m.drugan@tue.nl

Marco Wiering
Institute of Artificial Intelligence
and Cognitive Engineering
University of Groningen, The Netherlands
Email: m.a.wiering@rug.nl

***Abstract*—When reinforcement learning is applied to large state spaces, such as those occurring in playing board games, the use of a good function approximator to learn to approximate the value function is very important. In previous research, multilayer perceptrons have often been quite successfully used as function approximator for learning to play particular games with temporal difference learning. With the recent developments in deep learning, it is important to study if using multiple hidden layers or particular network structures can help to improve learning the value function. In this paper, we compare five different structures of multilayer perceptrons for learning to play the game Tic-Tac-Toe 3D, both when training through self-play and when training against the same fixed opponent they are tested against. We compare three fully connected multilayer perceptrons with a different number of hidden layers and/or hidden units, as well as two structured ones. These structured multilayer perceptrons have a first hidden layer that is only sparsely connected to the input layer, and has units that correspond to the rows in Tic-Tac-Toe 3D. This allows them to more easily learn the contribution of specific patterns on the corresponding rows. One of the two structured multilayer perceptrons has a second hidden layer that is fully connected to the first one, which allows the neural network to learn to non-linearly integrate the information in these detected patterns. The results on Tic-Tac-Toe 3D show that the deep structured neural network with integrated pattern detectors has the strongest performance out of the compared multilayer perceptrons against a fixed opponent, both through self-training and through training against this fixed opponent.**

I. INTRODUCTION

Games are a popular test-bed for machine learning algorithms, and reinforcement learning [1], [2] in particular. In many board games, the utility of an action is not clear until much later, often the end of the game. For such games, temporal difference learning (TD-learning) [3] can be very useful, as shown for e.g. chess [4], [5], backgammon [6]–[8], and Othello [9]–[11].

In many of these papers, e.g. [7], the multilayer perceptron (MLP) is used as a function approximator to deal with the huge state spaces. Furthermore, often the employed MLP is fully connected, meaning the neural network has no inherent preference for how the different inputs are treated with respect to each other. However, it should be apparent that some spaces on the board of almost any game are of more importance to each other than others. Ideally, the neural network will still find the important patterns, but often the game is too complex for the neural network to learn them in a reasonable amount of time, because the neural network relies on too many trainable parameters, i.e. its weights. It would be useful to reduce the number of weights in the neural network to mitigate this issue, but losing valuable information should be avoided. The use of prior information in the design of a function approximator can be very helpful to improve learning an accurate value function. For example, in [11], the author successfully eliminated many of the trainable parameters in the function approximator (a set of lookup tables) of the so-called LOGISTELLO program by dividing the Othello board into regions (e.g., lines across the board).

Structured Neural Networks Some famous structured neural network approaches are the Neocognitron [12] and convolutional neural networks [13], [14]. These approaches work differently than fully-connected neural networks: (1) They use sparse connections between inputs and hidden layer units, (2) They are made invariant to small translations of patterns in an image. The Neocognitron makes use of unsupervised learning techniques to learn position invariant receptive fields. Convolutional neural networks learn translation invariant mappings between patterns and class labels by supervised learning using convolutional layers and sub-sampling layers. Convolutional neural networks have obtained a lot of interest recently due to their very good performance for image recognition applications [15], [16]. Although these approaches use a structure, they are different from our proposed structured topologies, since in Tic-Tac-Toe 3D the neural networks should not be translation invariant. In [10], the authors compare the performance of a structured MLP with fully connected MLPs in the game Othello. The results showed that the structured MLPs significantly outperform fully connected MLPs.

Recently, Google's DeepMind obtained very good results on learning to play many different Atari games using deep reinforcement learning [17]. In this paper we want to find an answer to the research question if using multiple hidden layers or structured neural networks can help to improve learning to play an unexplored game: Tic-Tac-Toe 3D.

Contributions In the current paper, we extend the idea of the structured MLP from [10] to the game Tic-Tac-Toe

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.89

3D. We do this by introducing a structured MLP that has a second hidden layer after its structured hidden layer, that is fully connected to both the structured hidden layer and the output layer. This allows the neural network to integrate the different patterns it detects on the board of Tic-Tac-Toe 3D. In total we compare five different MLP structures, namely a structured MLP without extra layer, a structured MLP with an extra layer, as well as three fully connected MLPs of different sizes or number of layers. We test each of these MLPs against a benchmark player by either training through self-play, or by training against the same benchmark player it is tested against. We run 10 trials for all ten experiments, lasting 1,000,000 training games each. The MLP's performance is computed by testing it against the benchmark player after every 10,000 games. The results show that the second hidden layer in the structured MLP gives it an enormous boost in performance when training through self-play, compared to both the smaller structured MLP and the unstructured MLPs. When training by playing against the benchmark player, it is also the strongest playing program.

Outline In Section II, we will briefly explain how the game of Tic-Tac-Toe 3D works. In Section III we describe the theoretical background behind reinforcement learning, and TD-learning specifically. In Section IV we discuss the structures of the different multilayer perceptrons we are using. In Section V we describe our experiments and present their results, followed by a brief discussion of the results. Finally, in Section VI we provide a conclusion and outline some directions for future work.

II. Tic-Tac-Toe 3D

In Tic-Tac-Toe 3D, the board consists of 4x4x4 spaces and it has approximately the same size of the state space as Othello: we estimate it to have more than 10^{20} states. We chose this game especially because it has a clear structure inside: the board is a cube in which winning combinations are on particular lines of length four in the cube.

Following the physical game, we chose to incorporate gravity, meaning that a piece will always be placed on the lowest empty z-coordinate. Two players, white and black, alternate turns in placing pieces of their color onto the board. The first player to place four pieces in a row wins. This row may be horizontal, vertical, or diagonal (in either two or three dimensions). If all 64 spaces are filled without any of the rows being fully occupied by pieces of the same color, the game ends in a draw. Fig. 1 illustrates a game position in which white just won.

In Tic-Tac-Toe 3D, there are 64 fields, each of which can have 3 different possible values (occupied by the player, the opponent, or unoccupied). While there are many restrictions to the number of possible states due to alternated turns, gravity, and winning states, the state space is much too large to use lookup tables for storing state values. Because of this, an alternative approach, such as a neural network, is required for determining the value of different board states. As the winning condition of Tic-Tac-Toe 3D is a row being completely occupied by pieces of the same color, the patterns in this game are very explicit, and therefore easily incorporated into a neural network.

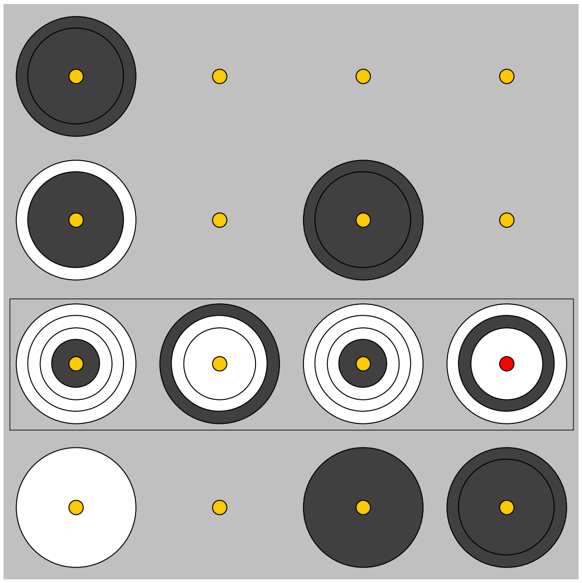

Fig. 1. An example of a final game state with a view from above. Larger circles represent lower pieces. The white player just won by placing a piece in the third row, fourth column, resulting in the third row being completed at height 3.

III. Reinforcement Learning

Reinforcement learning is used for the agent (in this case the Tic-Tac-Toe 3D player) to learn from its experience from interacting with the environment through positive and negative feedback following its actions [2]. After enough experience, the agent should be able to choose those actions that maximize its future reward intake. To apply reinforcement learning, we assume a Markov Decision Process (MDP) [18]. The MDP is formally defined as follows:

(1) A finite set of states $S = \{s^1, s^2, s^3, ..., s^n\}$, $s_t \in S$ denotes the state at time t.

(2) A finite set of actions $A = \{a^1, a^2, a^3, ..., a^m\}$, $a_t \in A$ denotes the action executed at time t.

(3) A transition function $P(s, a, s')$, specifying the probability of arriving at any state $s' \in S$ after performing action a in state s.

(4) A reward function $R(s, a)$, specifying the reward the agent receives upon executing action a while in state s, r_t denotes the reward obtained at time t.

(5) A discount factor $0 \leq \gamma \leq 1$, which makes the agent value immediate rewards more than later rewards.

With reinforcement learning we are interested in finding the optimal policy $\pi^*(s)$ for mapping states to actions, that is, a policy that selects actions to obtain the highest possible cumulative discounted expected reward. The value of a policy π, $V^\pi(s)$, is the expected cumulative reward the agent will get from following the policy, starting in state s. $V^\pi(s)$ is defined in Equation 1, with $E[.]$ denoting the expectancy operator:

$$V^\pi(s) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi\right] \tag{1}$$

The value function can be used by a policy to choose the best estimated action a in the following way, see Equation 2:

$$\pi(s) = \arg\max_a \sum_{s'} P(s,a,s')(R(s,a) + \gamma V^{\pi}(s')) \tag{2}$$

A. TD-learning

TD-learning [3] is a reinforcement learning algorithm that uses a temporal difference error to update the state value for state s_t after using action a to transition into state s_{t+1} and receiving a reward r_t. It does so by using the update rule in Equation 3:

$$V^{new}(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)) \tag{3}$$

where $0 < \alpha \leq 1$ is the learning rate. When TD-learning is combined with a neural network as function approximator, there is already a learning rate for the neural network, so the TD-learning rate can be set to 1, resulting in a simplified rule as in Equation 4:

$$V^{new}(s_t) \leftarrow r_t + \gamma V(s_{t+1}) \tag{4}$$

As the state values are represented by the neural network, we use the target output value $V^{new}(s_t)$ for the network input corresponding to state s_t. If s_t is a terminal state the target output is the final reward r_t. The backpropagation algorithm can then be used to update the value function in an online manner.

B. Application to Tic-Tac-Toe 3D

The algorithm updates the values of afterstates (the state resulting from the player's move, before the opponent's turn), as these are the state values that are used for move selection. Because Tic-Tac-Toe 3D is played against an opponent, we must wait for our opponent to make its move before learning an afterstate's value. The TD-error is the difference between subsequent afterstate values in between which the opponent also makes a move. As we are using online TD-learning, the update rule is applied on each of the player's turns (except the first one). In Tic-Tac-Toe 3D, there is only a reward at the end of the game. The player receives a reward of 1 upon winning, a reward of -1 upon losing, and a reward of 0 in case the game ends in a draw (if the entire board is filled without either player winning).

The TD-learning algorithm with a neural network as function approximator is applied to Tic-Tac-Toe 3D as described in algorithm 1 . On the TD player's first move, only steps 1, 2, and 6-8 are done, as there is no previous afterstate yet. The policy π for selecting an action uses the ϵ-greedy exploration algorithm, in which the agent chooses the (perceived) optimal action for a proportion of $1 - \epsilon$ of its moves, and a random action otherwise. In our implementation, $\epsilon = 0.1$.

When training against the fixed opponent, the above algorithm is used and therefore the TD-player only learns from its own moves. When the TD-player is trained by self-play it uses the above algorithm for both the white and the black player with the same neural network. In this way, the afterstates of the white player are trained on the values of afterstates of the white player after two moves have been done, one for white and one for black. The same is done for the positions of the black player. Although learning from self-play has therefore the advantage that two times more training data are generated for each game, the disadvantage is that it does not directly learn from playing games against the opponent against which it is finally tested.

Algorithm 1 Learn-From-Game

```
1: For every afterstate $s'_t$ reachable from $s_t$, use the Neural Network to compute $V(s'_t)$
2: Select an action leading to afterstate $s^a_t$ using policy $\pi$ (with $\epsilon$-greedy exploration)
3: Use Equation 4 to compute the target value of the previous afterstate $V^{new}(s^a_{t-1})$
4: Forward propagate the Neural Network to compute the current value for the previous afterstate $V(s^a_{t-1})$
5: Backpropagate the error between $V^{new}(s^a_{t-1})$ (or the obtained final reward $r_t$) and $V(s^a_{t-1})$
6: Save afterstate $s^a_t$ as $s^a_{t-1}$
7: Execute the selected action leading to afterstate $s^a_t$
8: Let the opponent perform an action and set $t = t + 1$.
```

IV. Multilayer Perceptrons

We are interested to find out the effects of design choices of neural network architectures on the final playing performance. The multi-layer perceptrons can have many hidden units or more than one hidden layer, and can use sparse connections between input and hidden layers. The effects of these choices when the multi-layer perceptron is combined with reinforcement learning have not often been studied, and never for the game Tic-Tac-Toe 3D.

We compare the performance of five different multilayer perceptrons (MLPs). These five MLPs have their input and output layers in common. The input layer has 64 nodes, corresponding to the spaces on the board. For a player's own pieces on a field the input is 1, for an opponent's piece on a field the input is -1, and for an unoccupied field the input is 0. The output layer has only one node, representing the estimated game position value. Except for the input layer, which is linear, all layers follow a logistic function scaled to [-1, 1].

All weights in each of the MLPs are uniform randomly initialized in the range [-0.5, 0.5). Each of the MLPs is trained using the backpropagation algorithm together with TD-learning for computing target values of afterstates, as described in Section III.

As described before, winning configurations for Tic-Tac-Toe 3D consist of having 4 fields of some line in the cube occupied by the player's stones. This prior knowledge is useful to design particular structured neural networks. At the same time, we want to understand the effect of using more than one hidden layer in the MLP and using more or less hidden units.

Two of the five MLPs are structured and have only very sparse connections between the input layer and the first hidden

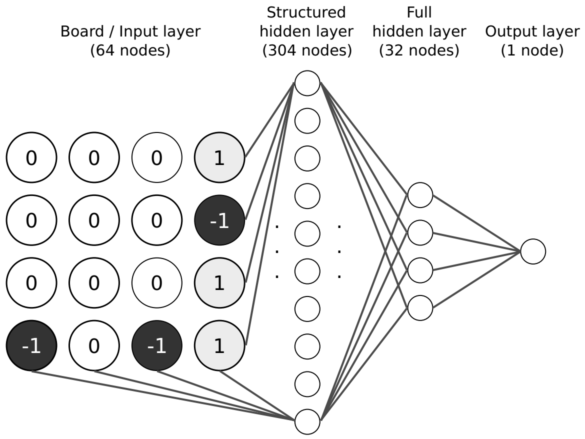

Fig. 2. This figure shows the structure of the deep structured MLP. The 64 board spaces are used as the input layer (1 for the player's own pieces, -1 for the opponent's pieces, 0 for empty spaces). The first hidden layer is structured, each of its units being connected to only the four input units corresponding to the row it represents. For every row, there are four hidden units in the structured layer. The second hidden layer is fully connected to both the first hidden layer and the output node. The output node represents the estimated game position value.

layer. The first hidden layer units represent the different rows on the board. There is a total of 76 rows: 48 horizontal and vertical rows, 24 diagonal rows that span two dimensions (XY, XZ, or YZ), and 4 diagonal rows that span 3 dimensions (XYZ). A hidden unit in the structured hidden layer is only connected to the four fields of one such row. A row can have different patterns, e.g. containing one white and 2 black stones on the four different fields. These hidden units can detect certain patterns in their corresponding row. However, as there are multiple possible patterns in each row, it is useful to have more than one detector per row. We chose to have four detectors (hidden units) per row, as adding more did not improve performance. This leads to a hidden layer of 304 units, each of which is only connected to its four corresponding input units.

The small structured MLP only has this one hidden layer and connects all hidden units directly to the output unit. The deep structured MLP has an additional hidden layer that is fully connected to both the first (structured) hidden layer and the output node, integrating the patterns detected in the structured hidden layer. The deep structured MLP is displayed for clarity in Fig. 2. The deep structured MLP has the advantage compared to fully connected MLPs by using sparse connections and thereby it can use more hidden units while still having a similar number of total learnable parameters (weights). Compared to the shallow structured MLP, it has the advantage that the information of multiple patterns on different rows of a position are combined in the second hidden layer.

Three of the MLP are fully connected (unstructured). There is a small unstructured MLP (one small hidden layer), a wide unstructured MLP (one large hidden layer), and a deep unstructured MLP (two hidden layers). The wide and the deep unstructured MLPs have a similar number of weights as the deep structured MLP. The architectures of the five different MLPs are listed in Table I.

TABLE I. THE DIFFERENT MLPS, THEIR LAYER SIZES, AND THE TOTAL NUMBER OF WEIGHTS. THE DEEP AND WIDE UNSTRUCTURED MLPS WERE CONSTRUCTED SUCH THAT THEIR NUMBER OF WEIGHTS IS SIMILAR TO THAT OF THE DEEP STRUCTURED MLP.

MLP type	Layer sizes	Number of Weights
Small structured	64-304-1	1520
Deep structured	64-304-32-1	10976
Small unstructured	64-50-1	3250
Deep unstructured	64-95-51-1	10976
Wide unstructured	64-169-1	10985

V. EXPERIMENTS AND RESULTS

For each of the MLPs, two different experiments were performed. In the first experiment, the MLPs learn the game through self-play, after which it is tested against a fixed benchmark player described below. In the second experiment, the MLPs learn the game by playing against the same benchmark player they are tested against, without learning from the moves of the opponent.

For both experiments, for every MLP, ten trials were performed. In each trial, the MLP trained by TD-learning for a total of 1,000,000 games. After every 10,000 training games, the TD player was tested against the benchmark player for 1,000 games. This results in 100 test phases per trial. The number of wins and draws are stored for each of these test phases, resulting in 100 data points (draws counting as half a win). During the training phases, the TD player had a learning rate α of 0.005, a discount factor γ of 1, and a constant 0.1 exploration chance ϵ. During the testing phases, the TD player did not learn or explore, but simply played the move with the highest afterstate value.

A. Benchmark Player

In both experiments, the TD player is tested against a benchmark player, and in one of them it also trains against it. The benchmark player is a fixed player; its policy does not change. The moves of the benchmark player are determined by the following algorithm:

```
1: Get all possible moves in random order.
2: If there is a move that would win the game, do the first
   one that does.
3: Otherwise, if the opponent has any way to win next turn,
   block the first one found.
4: Otherwise, do the first possible move which does not allow
   the opponent to win the game by placing a piece on top
   of it.
```

This benchmark player incorporates the rule-based knowledge of not making stupid mistakes. However, because it does not use lookahead search mechanisms, it can be defeated if the opponent places a stone to create two threats in which it can win during the next move.

B. Self-training

Results Fig. 3 plots the resulting win rates of the MLPs over time for the experiment where the TD player trained by playing against itself. Table II shows the peak win rates and the mean win rate over the last 40 measurements of all the MLPs in this experiment, as well as their standard deviations.

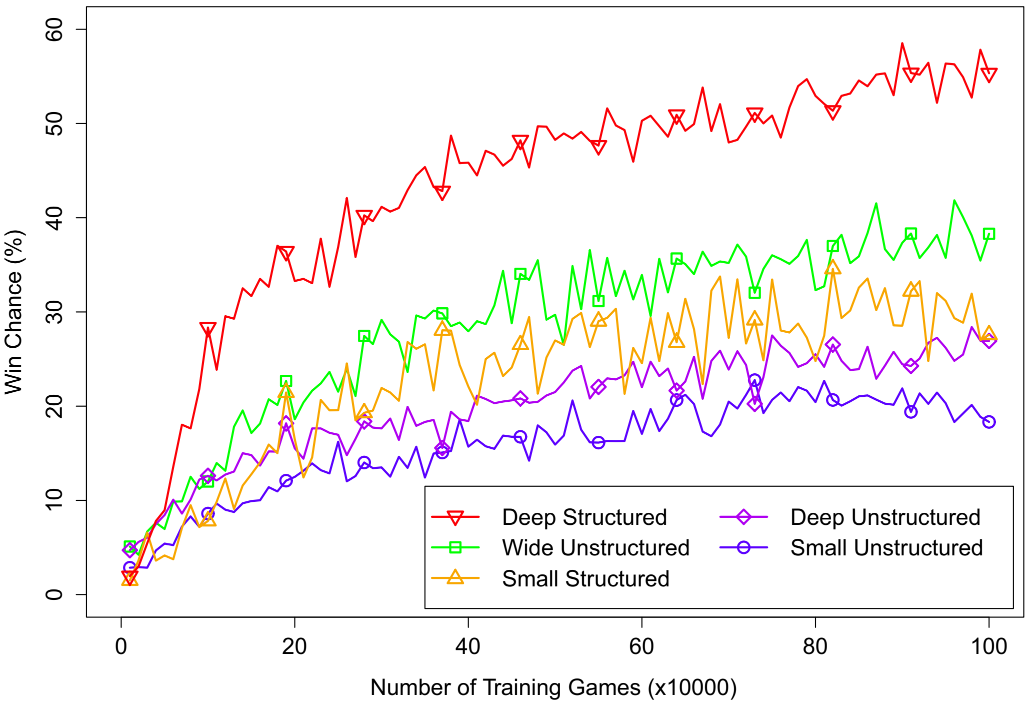

Fig. 3. This figure plots the performance of all five MLPs against the benchmark player while trained through self-play. There is a measurement point after every 10,000 training games. Draws count as half a win. The shown performance is the average over 10 trials.

TABLE II. PERFORMANCE OF THE DIFFERENT MLPS AGAINST THE BENCHMARK PLAYER WHILE TRAINING AGAINST ITSELF. PEAK WIN PERCENTAGE IS THE HIGHEST WIN RATE PER TRIAL, AVERAGED OVER TEN TRIALS. END WIN PERCENTAGE IS THE MEAN WIN RATE OVER THE LAST 40 (OUT OF 100) MEASUREMENT POINTS, AVERAGED OVER TEN TRIALS. DRAWS COUNT AS HALF A WIN. STANDARD DEVIATIONS ARE ALSO SHOWN.

MLP type	Peak win %	End win %
Small structured	$50.6 \pm 4.9\%$	$29.4 \pm 8.8\%$
Deep structured	$\mathbf{65.9 \pm 5.0\%}$	$\mathbf{52.7 \pm 6.8\%}$
Small unstructured	$31.2 \pm 6.2\%$	$20.2 \pm 4.3\%$
Deep unstructured	$38.2 \pm 5.4\%$	$24.8 \pm 4.9\%$
Wide unstructured	$54.1 \pm 5.5\%$	$36.1 \pm 7.1\%$

TABLE III. PERFORMANCE OF THE DIFFERENT MLPS AGAINST THE BENCHMARK PLAYER WHILE TRAINING AGAINST THE SAME BENCHMARK PLAYER IT IS TESTED AGAINST. PEAK WIN PERCENTAGE IS THE HIGHEST WIN RATE PER TRIAL, AVERAGED OVER TEN TRIALS. END WIN PERCENTAGE IS THE MEAN WIN RATE OVER THE LAST 40 (OUT OF 100) MEASUREMENT POINTS, AVERAGED OVER TEN TRIALS. DRAWS COUNT AS HALF A WIN. STANDARD DEVIATIONS ARE ALSO SHOWN.

MLP type	Peak win %	End win %
Small structured	$94.3 \pm 3.1\%$	$90.7 \pm 2.4\%$
Deep structured	$\mathbf{98.7 \pm 0.3\%}$	$\mathbf{97.3 \pm 0.8\%}$
Small unstructured	$93.3 \pm 1.8\%$	$90.4 \pm 1.5\%$
Deep unstructured	$98.1 \pm 0.5\%$	$96.2 \pm 1.0\%$
Wide unstructured	$96.8 \pm 0.8\%$	$94.4 \pm 1.5\%$

Discussion In Fig. 3 we can see that all of the MLPs start with a win percentage close to 0%. A learning effect can be immediately seen in each of them, and throughout the whole learning period, the different MLPs can be clearly distinguished from each other in their performance. The deep structured MLP learns much faster than the other architectures and is the only one that achieved an end win percentage above 50% (see Table II). This is significantly higher than the MLP with the second highest end win percentage, which is the wide unstructured MLP, at 36.1%. Interestingly, the small structured MLP outperforms the deep unstructured MLP, despite its lower number of layers and weights. The small unstructured MLP performs the poorest of the five. The wide unstructured architecture outperforms the deep unstructured MLP. The best average peak winning rate is also obtained with the deep structured MLP, leading to a score of 65.9%.

C. *Benchmark-training*

Results Fig. 4 plots the resulting win rates of the MLPs over time for the experiment where the TD player trained by playing against the same benchmark player it was tested against. Table III shows the peak win rates and the mean win rate over the last 40 measurements of all the MLPs in this experiment, as well as their standard deviations.

Discussion As can be seen in Fig. 4, in the benchmark-training experiment the performances of the different MLPs lie far closer to each other. All of their performances increase rapidly at the start of the training period, approaching near maximal performance after about 400,000 training games. The structured MLPs seem to be able to longer improve their performance levels compared to the unstructured MLPs.

While all of the MLPs easily outperform the benchmark

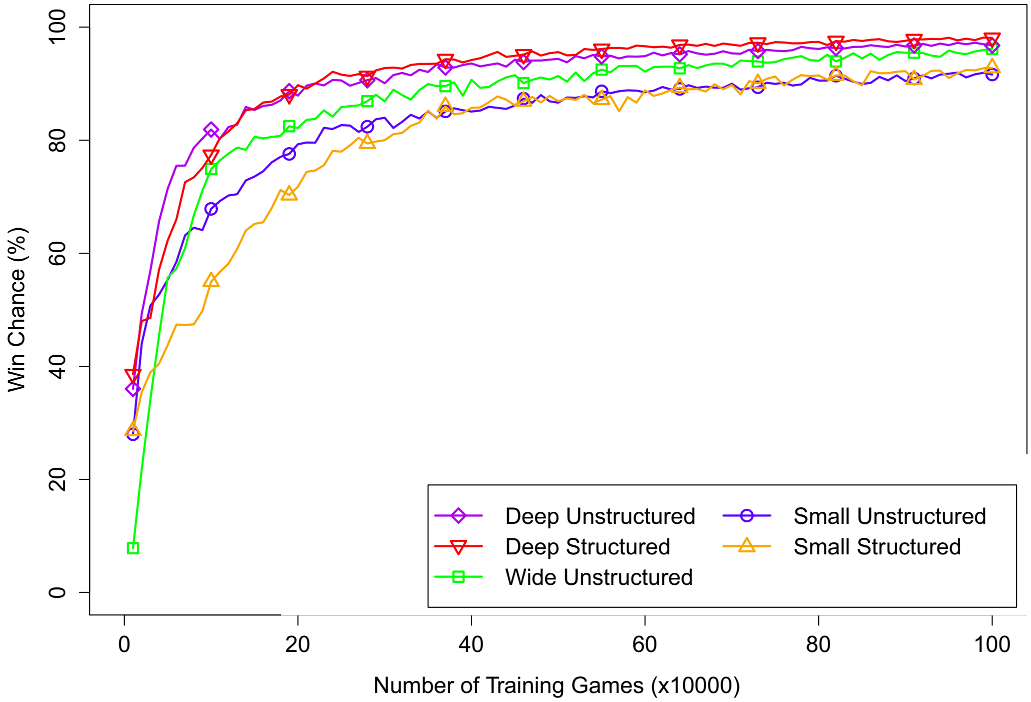

Fig. 4. This figure plots the performance of all five MLPs against the benchmark player while trained against it as well. There is a measurement point after every 10,000 training games. Draws count as half a win. The shown performance is the average over 10 trials.

player when training against it, the deep MLPs clearly outperform the wide and the small ones. An unpaired t-test reveals that the difference between the deep structured MLP and the deep unstructured MLP is statistically very significant for their peak win percentages ($P < 0.01$), and statistically significant for their end win percentages ($P < 0.05$). Now the deep unstructured MLP significantly outperforms the wide unstructured MLP. There is no significant difference between the performances of the small unstructured and the small structured MLPs.

The clearest difference between the two experiments is that the MLPs that were trained against the benchmark player itself perform far better than those who were trained through self-play. Interestingly, this conflicts with the conclusion from [9], where the authors achieved the best results for TD-learning in Othello by learning through self-play, rather than by training against the same opponent it was tested against. This suggests that these results are game dependent, or possibly dependent on the benchmark player that was used.

The benchmark player was designed such that it was impossible to win by simply placing pieces in a row. Because of this, in order to win, the TD player had to learn to create two threats simultaneously. However, besides dealing with immediate threats, the benchmark player executes actions at random. When training against the benchmark player, the TD player managed to learn how to exploit the weaknesses of the benchmark player regardless of which network structure was used. However, in self-training, the TD player played against an opponent (itself) that does not share this weakness, so instead it had to learn by becoming a solid player overall, which is much more difficult than learning how to outperform a benchmark player with clear weaknesses. Only the deep structured MLP accomplished this to the extent that it can outperform the benchmark player without training against it.

VI. Conclusion

In the program LOGISTELLO [11], the influence of many different patterns from the game Othello were stored by using lookup tables. In [10], the authors expanded on this by using neural networks to represent these patterns. In the current paper, we extended this approach for the game Tic-Tac-Toe 3D, where we also looked into the effect of adding a second hidden layer behind the structured hidden layer, that is fully connected to the structured hidden layer as well as to the output layer. This extra layer enables the neural network to integrate the patterns that are detected in the structured layer. This addition showed very large improvements in playing strengths for the game Tic-Tac-Toe 3D when training against itself, as well as a small, but significant, improvement when training against the benchmark player it is tested against. When examining the number of hidden units, in the experiments better performances are obtained with more of them. Although the results show a clear benefit of using two hidden layers in the structured MLP, for the unstructured MLP this benefit is not clearly shown in the results.

For future work, more complex structures for the neural network could be explored. It would be possible that the structured MLP would perform even better with multiple structured layers. Because there are a lot of symmetries in Tic-Tac-Toe 3D, it would be a good idea to exploit these with weight sharing. Finally, we want to study the importance of design choices of multi-layer perceptrons when applied to learning policies for other problem with large state spaces, which can also be broken up in smaller pieces such as forest fire control [19].

VII. Acknowledgements

Madalina M. Drugan was supported by ITEA2 project M2Mgrid.

References

[1] M. A. Wiering and e. M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Berlin, Heidelberg: Springer-Verlag, 2012.

[2] R. Sutton and A. Barto, *Introduction to reinforcement learning*. Cambridge, MA: MIT Press, 1998.

[3] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[4] S. Thrun, "Learning to play the game of chess," *Advances in neural information processing systems*, vol. 7, pp. 1069–1076, 1995.

[5] J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal differences," *Machine Learning*, vol. 40(3), pp. 243–263, 2000.

[6] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

[7] ——, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[8] M. A. Wiering, "Self-play and using an expert to learn to play backgammon with temporal difference learning," *Journal of Intelligent Learning Systems and Applications*, vol. 2, no. 02, pp. 57–68, 1995.

[9] M. van der Ree and M. A. Wiering, "Reinforcement learning in the game of othello: Learning against a fixed opponent and learning from self-play," in *2013 IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, Apr. 2013, pp. 108–115.

[10] S. van den Dries and M. A. Wiering, "Neural-fitted td-leaf learning for playing othello with structured neural networks," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 11, pp. 1701–1713, 2012.

[11] M. Buro, "The evolution of strong othello programs," in *Entertainment Computing*. Springer, 2003, pp. 81–88.

[12] K. Fukushina, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36(4), pp. 193–202, 1980.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Back-propagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86(11), 1998, pp. 2278–2324.

[15] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., 2012, pp. 1097–1105.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015.

[18] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[19] M. A. Wiering and M. Dorigo, "Learning to control forest fires," in *Proceedings of the 12th international Symposium on "Computer Science for Environmental Protection"*, ser. Umweltinformatik Aktuell, H.-D. Haasis and K. C. Ranze, Eds., vol. 18. Marburg: Metropolis Verlag, 1998, pp. 378–388.