Artificial Intelligence on "Dots & Boxes" Game

Mehdi Kabir

June 2, 2004

1 Abstract

Dots-and-Boxes is a two-player game that admits many levels of strategy and remains analytically unsolved. The methods of temporal difference learning and backpropagation are applied to this game. An Artificial Intelligence (AI) player uses a feed forward neural network that is initialized with a random set of weights. The player receives a feedback signal only at the end of each training game and learns to consistently beat the simplest heuristic agent. I also observed some success is observed against more advanced heuristics.

2 Introduction

Artificial intelligence is a field with potential applications across a broad range of domains. Game playing offers a controlled environment in which to evaluate and develop different techniques and approaches in the field. This environment is ideal because the rules are clearly

defined, play can be easily automated, and performance measures are readily available. Furthermore, the decision-making required in games often has an analog in the real world.

Within the domain of game playing there are two general approaches: adaptive and nonadaptive. The latter involve the use of static evaluator functions coupled with some algorithm for searching the problem space. This project focuses on the former: techniques that involve machine learning. The goal was to investigate what the machine could learn about a particular game using as little information as possible about the rules and strategy.

I focused on the game of Dots-and-Boxes. It is described in detail in section 3. I selected the game because the board size is scalable and, unlike most other games that have been studied (e.g., Backgammon, Othello, Chess, Go), there is no piece differentiation. That is, each line "drawn" is identical from a strategic standpoint irrespective of which player "drew" that line.

In 1995, Tesauro devised a program that played the game of Backgammon near the level of grandmaster. His "TD-Gammon" learned through self-play using neural networks and a technique known as temporal-difference learning (TD learning). Neural networks and TD learning are described in section 4. The ability of "TD-Gammon" to abstract complex strategies from minimal feedback served as the inspiration for this project. I hope to develop an AI player that learns the most advanced strategies of Dots-and- Boxes through self-play and minimal reinforcement. To that end, I implemented a feed forward neural network and trained it using the TD method.

3 Dots-and-Boxes

3.1 Rules

The game board is a rectangular grid with each point on the grid marked by a dot. The size of the game is measured by the number of boxes the grid has horizontally and vertically. In Figure A there is an example of a game board of size 3.

For the purposes of this project only square grids (i.e. 2x2, 3x3, etc.) were considered. Two players who alternate turns and make legal moves play the game. A legal move consists of connecting two horizontally or vertically adjacent dots that were not previously connected. If a player completes the fourth side of a 1x1 box, then he is awarded a point, and is required to make another move. It is possible to complete two 1x1 boxes with a single line; I call such a move a "double-cross" move. A player who can complete a box on his turn is not required to do so. If players are obliged to complete a box on their turn, then the analysis of the game is simplified.

The players continue to alternate moves until all the horizontally or vertically adjacent dots are connected. The player who completes the most boxes (scores the most points) wins the games. Since the grid size may be even, ties are possible.

3.2 Strategies

There are number of different strategies that were considered for the project ranging from random play to one of the deepest strategies admitted:

Level 0: The Level 0 strategy consists of selecting a legal move uniformly at random (and independent of previous choices), and making it.

Level 1: The Level 1 strategy consists of completing the fourth side of a box if possible. Otherwise, a move is made using the Level 0 strategy. This strategy defeats a Level 0 player in $99.9963 \pm 0.0003\%$ of games played (base on 10,000 games of size 3).

Level 2: The Level 2 strategy consists of completing the fourth side of a box if possible. Otherwise a move that will not allow the opponent to complete the fourth side of a box is made. Finally, if no such move is possible, the Level 0 strategy is reverted to. This strategy defeats a Level 1 player in 87.18% of games played (based on 10,000 games of size 3).

Level 3: This strategy is sometimes referred to as the "double dealing endgame." It involves learning to concede boxes to the opponent in certain situations. When the opponent completes these boxes, the opponent has usually been "double-crossed" into opening up a longer series of boxes. For example, in the game below, Player 1 uses this strategy to force Player 2 into opening up a series of 4 boxes. In the next section, there is a thorough explanation of this strategy.

4 Background

The project draws on three concepts in AI: neural networks, backpropagation, and TD learning.

4.1 Neural Networks

A neural network, in its most abstract form, is a system of adaptable nodes that learn by example, store the learned information, and make it available for later use. A node also referred to as a neuron consists of a processing unit, a number of weighted inputs, a threshold, and a single output. To simplify the implementation, the threshold is specified as an input, 0, with a weight of -1. The node's output is a function of the weighted sum of its inputs. Letting $\vec{x}$ represent the vector of n inputs, $\vec{w}$ the vector of n weights (corresponding to the weight of each input), and o, the output of the neuron:

$$o = f(\sum_{i=0}^{n} w_i x_i) \tag{1}$$

The function, f, is referred to as the activation function and is used to restrict the range of outputs. For this project, the sigmoid activation function was selected:

$$f(s) = \frac{1}{1 + e^{-s/Q}} \tag{2}$$

The constant, Q, is referred to as the temperature of the neuron. The higher the temperature the more gently the sigmoid changes. At very low temperatures it approaches a step function. The temperature is set via experimentation. The sigmoid function activation function was selected because it has been applied successfully to a number of networks in different environments (i.e. the neural network in TD-Gammon used a sigmoid squashing function). I note that it is differentiable, a requirement for the backpropagation learning rule.

In a neural network, the neurons are interconnected (the outputs of certain neurons serve as the inputs of other neurons). The connectivity of the neurons is referred to as the topology of the network. There were a number of different models available but, because of its simplicity (from an implementation perspective) and the wealth of learning paradigms available for it, the feed forward model was selected. This model consists of arranging the neurons into layers: an input layer, any number of hidden layers, and an output layer.

Connections are only permitted between neurons in adjacent layers. For the purposes of the project, fully connected (every permitted connection is made) feed forward networks with one hidden layer were used. the neurons in the input layer serve to pass the values they receive on to the hidden layer. There, the values are processed and then passed on to the output layer. The neurons in the output layer process their inputs and produce the network's final output values.

4.2 Backpropagation

A neural network can be uniquely defined in terms of its topology, activation function, temperature, and the weights of its connections. The central question is how to adjust these parameters so that the network produces the correct output for each input it receives. This is commonly referred to as the learning methodology. The focus is usually on adjusting the weights of the networks and fixing the other parameters (which are selected via experimentation). Although I could have also dynamically adjusted the activation functions, temperatures, and topologies, this would have been a significant undertaking because little

previous work has been done to this effect.

The most common learning method is known as backpropagation and is sometimes also referred to as the generalized delta rule. An error value known as the mean square error is used to provide a measure of the difference between the desired and actual outputs. it is defined as:

$$E_{\vec{s}} = \frac{1}{2}\sum_{k=1}^{n}(t_{\vec{s}\,k} - o_{\vec{s}\,k})^2 \tag{3}$$

where $E_{\vec{s}}$ is the error for a given set of inputs, $\vec{x}$, $t_{\vec{s}\,k}$ is the target output of the k^{th} output neuron, and $o_{\vec{s}\,k}$ is the actual output of the k^{th} output neuron. For any given input, $\vec{x}$, the error, $E_{\vec{s}}$ is a function of the weights of the network. Each weight can be viewed as a dimension in an N-dimensional error space. For example, in a network with three weights, the error, E_x, might look like the the surface below.

Backpropagation tries to adapt the weights such that the network will be driven towards the minima of the error surface. It does this by considering the negative gradient of the error function with respect to the weights. This vector points in the direction that will most quickly reduce the error function. Thus, backpropagation adjusts the weights such that:

$$\Delta_{\vec{x}} w_{ji} \propto -\left(\frac{\partial E_{\vec{x}}}{\partial w_{ji}}\right) \tag{4}$$

where $\Delta_{\vec{x}} w_{ji}$ is the change in the weight connecting i^{th} neuron in layer L-1 (the source neuron) to the j^{th} neuron in layer L (the destination neuron). Each weight change corresponds to a step in the direction of the gradient. By substitution and evaluation of the partial

derivative in **(4)**, an easily implemented rule for adapting the weights is derived:

$$\Delta_{\vec{x}} w_{ji} = \eta \delta_{\vec{x}j} o_{\vec{x}i} \tag{5}$$

where η is a constant referred to as the learning rate, $\delta_{\vec{x}j}$ is a value referred to as the error signal of the j^{th} neuron in layer L and $o_{\vec{x}i}$ is the output of the i^{th} neuron in layer L-1. The error signal, $\delta_{\vec{x}j}$, is computed as follows:

$$\delta_{\vec{x}j} = (t_{\vec{x}j} - o_{\vec{x}j}) o_{\vec{x}j} (1 - o_{\vec{x}j}) \tag{6}$$

for output neuron and

$$\delta_{\vec{x}j} = o_{\vec{x}j} (1 - o_{\vec{x}j}) \sum_{m=0}^{n} \delta_{\vec{x}m} w_{mj} \tag{7}$$

for hidden neurons where $o_{\vec{x}j}$ is the output of the j^{th} neuron in layer L, $\delta_{\vec{x}m}$ is the error signal of the m^{th} neuron in layer L+1 and w_{mj} is the weight of the connection between the m^{th} neuron in layer L+1 and the j^{th} neuron in layer L.

There are a number of refinements to gradient descent that strive to prevent convergence on local versus global minima and to speed up convergence. Most of these methods were deemed beyond the scope of this project. The only augmentation to the basic backpropagation method implemented was the addition of a momentum term. This term helps produce a larger change in the weights if the changes are currently large. This is beneficial because it speeds up convergence and helps the network avoid getting stuck in local minima early on in the training (when the weight changes tend to be fairly large). Momentum is added to

the backpropagation method by modifying **(5)**:

$$\Delta_{\vec{x}} w_{ji} = \eta \delta_{\vec{x}j} o_{\vec{x}i} + \alpha(\Delta_{\vec{x}} w_{ji})_{t-1} \tag{8}$$

where α is a constant referred to as the momentum constant. The momentum constant is determined empirically.

4.3 TD Learning

In a game situation a prediction vector, $\vec{p}_i$, is required for each state, s_i, encountered in the game. An AI player would then use that prediction to make a move. In this project, a neural network was used to generate $\vec{p}_i$. When training the network via backpropagation the obvious question was how the target for $\vec{p}_i$ that should be determined. One approach would be to wait until each game is finished, determine a final utility, $\vec{u}_f$, and then correct $\vec{p}_1$, $\vec{p}_2$, $\vec{p}_f$ using $\vec{u}_f$ as the target. This would result in the training set $\{(s_1, \vec{u}_f), (s_2, \vec{u}_f), (s_f, \vec{u}_f)\}$. This approach, commonly referred to as supervised learning, was not used because the pairings it makes do not take into account temporal locality. Moves made early on in a game contribute far less to the final state than moves made later on.

The approach selected was based on a procedure formally presented by Sutton known as temporal difference learning. In TD learning the objective is to minimized the error between temporally successive predictions. Instead of using the pairings: $\{(s_1, \vec{u}_f), (s_2, \vec{u}_f), (s_f, \vec{u}_f)\}$ a TD approach would use the pairings: $\{(s_1, \vec{p}_2), (s_2, \vec{p}_3), (s_{f-1}, \vec{p}_f), (s_f, \vec{u}_f)\}$. Sutton argues, "TD methods make more efficient use of their experience than do supervised

learning methods, converge more rapidly, and make more accurate predictions along the way." That argument coupled with Tesauro's success using the method was the motivation behind selecting TD learning.

5 Results

In evaluating the AI player a series of four tests were performed:

- The performance of networks trained by play against heuristic0, play against heuristic1, play against heuristic2, and self-play was evaluated
- The performance of networks trained by play against heuristic0, play against heuristic1, and play against heuristic2 was evaluated over number of training sessions.
- The performance of networks trained by play against heuristic0 was evaluated over a varying number of nodes in the hidden layer
- The performance of networks trained by play against heuristic0 was evaluated over games of sizes 2, 3, and 4 respectively.

The limiting factor in the evaluation phase was the computational cost of training. During training, the network's output must be calculated and the weights adapted by backpropagation for each move in the game. There are many other tests that could be imagined (i.e. testing the effect on performance of varying the learning rate) but this was not possible given the resources available. For all the networks, the following were held constant:

- learning rate = 0.1
- momentum term = 0.4
- temperature = 1.0

These values were obtained empirically by varying them and observing the effect on the ability of a neural network to learn the odd parity function. All the performance evaluations are expressed as the percentage of wins in 10,000 games.

5.1 Test I

I refer to the networks trained by play against heuristic0, play against heuristic1, play against heuristic2 and self-play as I-0, I-1, I-2, and I-3 respectively. I held the following constant:

- nodes in hidden layer = 20
- number training games = 125,000
- game size = 3

Table 1 shows the performance of each network against the three heuristics.

I observe that all the networks trained by play against a heuristic learned to beat the Level 0 player consistently. I-3, however, performs even worst than a Level 0 player (i.e. it has learned some elements of a negative strategy).

After observing each of the networks in self-play I realized the problem with this paradigm as currently implemented. Neural networks are powerful learning tools because they generalize from a finite training set to the larger superset. However, the ability of the network to

make useful generalizations require that the training set be sufficiently large and representative of the superset (i.e the training set should not be compromised of the rarest inputs). Thus, the success of the TD learning method hinges on exploring a large number of states in the game space. Whenever a neural network played against itself the result was a series of identical games. This is not surprising because, as long as two states with equal predictions are not encountered, the transition from one state to the next is deterministic. Thus, in self-play, a very small subsection of the game space is explored and the network makes erroneous generalizations. This is in contrast to games such as Backgammon where a network in self-play is forced by the element of chance (i.e. the roll of the dice) to explore the game space. In my implementation of self-play, there is nothing to catapult the neural network into new regions of the game space that would reveal the error of its current strategy.

5.2 Test II

Graphs 1, 2, and 3 show the performance of I-0, I-1, and I-2 at various stages in their training. The objective was to focus on the learning curves in order to determine why these networks failed to acquire the more advanced strategies.

I observe the following:

- The networks learn to beat a Level 0 player early on in the training session and, in general, retain this knowledge.
- I-1 and I-2 oscillate in their ability to beat the Level 1 player.
- I-0 has more consistent performance throughout the training session but fails to attain

the success against a Level 1 player observed at certain points in the training of I-1 and I-2.

In section 5.1 I attributed the poor performance on I-3 against the Level 0 player to the lack of non-determinism in *Dots-and-Boxes*. The immediate question is why this feature of the game did not affect I-0, I-1, and I-2. The reason is that, by virtue of their implementations (see section 2.2), heuristic0, heuristic1, and heuristic2 introduce an element of randomness into the game. For example, in a game involving an AI player and heuristic0 the transition from one ply to the next is entirely non-deterministic (a transition from one ply to the next is a transition from one state when it is a certain player's turn to the next state when it is the same player's turn). In games involving an AI player against heuristic1 or heuristic2, the current game context (e.g. the presence of any boxes with 3 sides complete) determines whether or not the transition from one ply to the next is non-deterministic. Nevertheless, in any game involving an AI player and one of the heuristics implemented, the transition from one ply to another will be non-deterministic for a non-empty subset of the plies (i.e. the transition from the ply containing the initial move to the next ply is always non-deterministic). Based on the first observation, I conclude that each of the heuristics force the network to explore enough of the game space to learn to beat the Level 0 player consistently.

I am uncertain as to why I-1 and I-2 oscillate in their ability to beat the Level 1 player. A number of further tests should be performed to investigate this observation. First I should increase the number of training games and determine whether the oscillation continues. I

should also observe whether, on average, the peak and trough of the oscillation is increasing (indicating that, on aggregate, learning is occurring). I should also try to vary the learning curve. It may be that I need to diminish the learning rate over time or that I should add a check to ensure the momentum is in the direction of the gradient (and if not discard the momentum for that weight adaptation).

In observing I-0 and play with heuristic0 the reason it failed to learn as much of the Level 0 strategy as I-1 and I-2 became clear. Early on in the training (after 25,000 games), I-0 learns to consistently beat heuristic0 decisively. This prevents the heuristic from leading the network into those regions of the game space that would arise in play against higher-level players (e.g. states where the score is close). Thus, the network's training set is so limited that in can never go beyond the Level 0 strategy.

5.3 Test III

I named the networks trained with 9, 30, and 37 nodes in hidden layer as III-0, III-1, and III-2 respectively. I held the following constant:

- training partner = heuristic0
- number training games = 500,000
- game size = 3

The objective was to compare the performance of these networks to I-1 and determine the trade-off between computational complexity ($\propto$ size of hidden layer) and learning (performance). Note that I use I-1 from section 5.1 as the network with 20 nodes in the hidden layer.

Table 1 shows the number of nodes in the hidden layer of each network and corresponding performance against the three heuristics.

The results indicate that generalizations necessary for the networks' current level of play are not complex enough to warrant a large hidden layer. In order to fully master the Level 1 strategy on a game of size 3, a network would require at least 9 hidden nodes (one corresponding to each square in the grid). Since none of the networks reach this level, it is not surprising that they do not require more than 9 hidden nodes. Although I observed no degradation in performance when reducing the number of nodes in the hidden layer, I noted that it took III-0 longer (50,000 games) to learn to beat the Level 0 player than it did for I-1, III-1, and III-2 to do so (25,000 games).

5.4 Test IV

Networks named IV-0 and IV-1 with 4 and 16 nodes in hidden layer were trained for games of size 2 and 4 respectively. I held the following constant:

- training partner = heuristic0
- number training games = 200,000

The objective was to observe the scalability of the neural networks to games of different sizes. Note that I use III-0 from section 5.3 with 9 nodes in the hidden layer to measure scalability to games of size 3. I varied the size of the hidden layer because the number of patterns for the network to learn increases with the game size. Table 3 shows the performance of the networks on games of different sizes.

I observed that the neural networks trained for different board sizes had comparable performance to the network trained for a game size of 3.

6 Discussion

Overall, the results observed demonstrate that approach I selected is promising. By receiving a feedback signal only at the end of each game the untrained networks quickly learned how to beat Level 0 players and often came close to learning the Level 1 strategy. I observed that these results were replicable on games of different sizes indicating the approach is well suited for the scalability of *Dots-and-Boxes*. The key shortcoming identified is the lack of game space exploration.

6.1 Possible Improvements

I could compensate for the deterministic nature of *Dots-and-Boxes* by adding a randomness (noise) factor, ϕ, into the AI player. This would involve modifying the move selection scheme slightly. Instead of making the move that results in the state with the highest evaluation, ν, the AI player would choose all the moves with evaluations with ϕ predictions. During training I could start ϕ relatively high (corresponding to low confidence in the network's decisions) and diminish it gradually (as a function of the training time perhaps).

I could improve self-play by using a form of selective cloning. In this variation, I would hold a tournament (a fixed number of games) at each cloning interval. The loser of the tournament would copy the winner's weights. Although this is computationally expensive,

it would help ensure that the trainee was consistently learning to beat an opponent at or above its own level.

Lastly, I could attempt to make the training set more representative of the game space by using selective backpropagation. If the mean-square error of the prediction for a given state was very low then I would not backpropagate the error. This would help avoid over training (at the expense of rarely encountered game states) for game states that appear frequently. Fro example, if a series of game states is seen repeatedly (as was the case in section 5.1) this improvement would help prevent the network from making the erroneous conclusion that those states represent a large portion of the game space. This modification would also decrease the computational cost of training by avoiding backpropagation for minute weight changes.

6.2 Future Work

Boyan explores the use of modular neural networks to learn context dependent game strategies. Modularization involves the use of a number of specialized neural networks. Each of these networks is trained for a particular game context (e.g. a network trained for the end game). A "gating" network is responsible for forward the move decision to the network best specialized for the current game state. I think this method is well suited for *Dots-and-Boxes* because some of the higher-level strategies of the game involve unlearning lower-level strategies. For example, learning the Level 3 strategy involves learning to avoid the Level 1 strategy in certain game contexts. I would like to implement a modular approach to the

game and observe whether this facilitates learning of the higher-level strategies

7 Conclusion

Dots-and-Boxes is a game well suited to investigating machine learning because it admits many levels of strategy and remains analytically unsolved. It offered a challenging application of temporal difference learning because of its scalability, lack of piece differentiation and deterministic nature. I developed a series of simple heuristics for the game. I then developed an AI player using a feed forward neural network to make move decisions. The network is trained using the method of TD learning coupled with backpropagation.

Section 5.2 indicates that our networks are capable of Level 0 play and, at certain points in the training period, of Level 1 play. Section 5.4 indicates good scalability to games of different sizes. In section 5, I conclude that the approach used is promising and suggest three possible improvements: introducing noise into the AI player, implementing selective cloning, and implementing selective backpropagation. In the future, I would like to use modular neural networks, which I believe will converge more rapidly on the context dependent higher-level strategies.

8 References

- http://www.cs.stir.ac.uk/ lss/NNIntro/InvSlides.html
- http://www.elsevier.nl/inca/publications/store/8/4/1/

- Beale, R. and Jackson, T. Neural Computing: An Introduction. IOP Publishing Ltd., 1990.

- Bell, A.G. Game Playing With Computers. George Allen & Unwin Ltd., 1972.

- Berlecamp, E.R., Conway, J.H., and Guy, R.R. Winning Ways. Volume 1, 507-550, Academic Press, 1973.