Reinforcement learning for Checkers: an Evaluation

David Garcia

dg446@kent.ac.uk

School of Computing

University of Kent

COMP6200

Word Count: 5504

April 6, 2023

Abstract - **Checkers is a popular board game where each player aims to win by eliminating all the opponent's pieces. The game features a search space of 10^{20} possible positions, which means to get a perfect solution requires a large amount of storage and processing power. In this project, we aim to research how a combination of reinforcement learning and deep neural networks can be exploited to create an agent capable of playing checkers at a high level without the heavy computation requirements of a perfect solution. To do this we will create a model that estimates the winning probability based on heuristic checkers metrics such as the number of opponents left. This model will be used to create a board evaluation model which can then be reinforced using deep Q-learning to increase the number of games ending in a win. An evaluation into whether this is an effective method to create a checker's agent will then be conducted.**

1. Introduction

Checkers is a board game that involves two players taking turns and has the characteristics of being zero-sum, complete information, deterministic, and relatively complex. The game contains approximately 10^{20} unique states, making it inefficient for a practical agent to develop a state-dependent strategy. Researchers have recognized Checkers as a significant challenge for the AI community, aiming to inspire innovative solutions for handling high-dimensional state spaces.

1.1 The Perfect Checkers Player: Chinook

In 2007, Schaeffer et al. developed Chinook, the ultimate Checkers player, which never loses a game. To achieve this, Chinook employs an opening book for the perfect game starts and an endgame database for backward searches from winning positions; or drawing positions if necessary. Schaeffer et al. solved Checkers by creating a search algorithm that can handle the immense state space of Checkers. However, their approach is not generally applicable to other high-dimensional state spaces, as it relies heavily on in-game information to frame the game as a search problem. Even with data compression, Schaeffer et al.'s endgame database comprises of 237GB of Checkers game configurations. This is an unrealistic amount of storage for our project. This study aims to apply a more efficient and widely applicable approach, namely reinforcement learning, to the high-dimensional Checkers domain. This study is particularly interesting because it evaluates the success of utilizing reinforcement learning to solve a problem that has already been solved using search methods.

1.2 Overview

In the next sections, we will describe and explain the rules of checkers as they are used in our work. We will also describe our approach to the problem and the algorithms used. At the end, we will explain and discuss the results of our experiments and take into consideration possible future work.

2. Background

In order to understand how we will create our checker's agent; we can first look at some key techniques and technologies that we will implement into our project. Below are some brief descriptions of these techniques and how they can be used to create the agent.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functionality of biological neural networks. They consist of interconnected nodes or neurons organized in layers that process and transmit information to solve complex problems. ANNs can learn from data, making them highly suitable for tasks like pattern recognition, classification, and prediction.

In the context of creating a checkers agent, ANNs can be utilized to learn optimal strategies and decision-making patterns. By feeding the network game states and outcomes from numerous checkers matches, the ANN adapts its

weights and biases to minimize the error in predicting the best moves. As a result, the checker's agent becomes increasingly proficient in evaluating board positions, selecting promising moves, and ultimately, outperforming its opponents.

2.2 Learning Through Backpropagation

Backpropagation is a widely-used supervised learning algorithm for training ANNs. It involves computing the gradient of the loss function concerning each weight and bias by applying the chain rule, which enables efficient calculation of weight updates. The algorithm adjusts the network's weights and biases in the direction of the negative gradient, minimizing the error between the predicted outputs and the actual targets. As the number of wrong predictions decreases, the ability of the checker's agent will increase.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties and aims to maximize the cumulative rewards over time. This learning process is driven by the exploration-exploitation trade-off, where the agent balances between trying new actions (exploration) and leveraging its knowledge to choose the best action (exploitation).

To create a checkers agent using RL, the agent represents a player, and the environment is the game of checkers. The agent learns by playing multiple games, making moves, and receiving rewards or penalties based on the outcomes. Typically, positive rewards are given for advantageous moves or winning games, while negative rewards are assigned for unfavourable moves or losing games. Through trial and error, the agent refines its decision-making policy, gradually learning to make more strategic moves, which ultimately leads to improved gameplay and increased chances of winning.

2.4 Q-Learning - The Q function is an evaluation function that gives a value Q for a state with an action that can be taken. The Q function can be defined as Q(*s*, *a*). The Q value is obtained from a reward value from reaching the derived state, combined with the discounted value of following the optimal policy learned so far (Mitchell, 1997). The reward value can be obtained from the reward function r(*s*, *a*), returning the immediate reward from taking action *a* in state *s*. The max value V * of a state can be obtained by examining the highest Q value for all actions in a state, V *(*s*) = max Q(*s*, *a*). Thus, if the Q values for all actions are initialized to zero, the non-zero reward for the final states will allow values to be fed backwards (discounted by γ) to populate the table. This can be mathematically visualised as:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

2.5 Bellman Equation

The Bellman equation is a fundamental concept in reinforcement learning, expressing the relationship between the value of a state or an action and the expected future rewards. It serves as the basis for learning optimal policies in Markov Decision Processes (MDPs).

For the action-value function Qπ(s, a), the Bellman equation can be visualised as:

$$Q_\pi(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma \cdot \sum_{a'} \pi(a'|s') \cdot Q_\pi(s',a'))$$

The Bellman equation forms the foundation for various reinforcement learning algorithms, such as value iteration, policy iteration, and Q-learning, which aim to estimate the optimal value functions and derive the optimal policy.

2.6 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework used to model sequential decision-making problems under uncertainty. MDPs consist of four main components: states (S), actions (A), rewards (R),

and transitions (P). The process is characterized by the Markov property, which states that the future state depends only on the current state and action, not on the history of past states or actions.

In an MDP, an agent interacts with an environment by taking actions, transitioning between states, and receiving rewards. The objective is to find an optimal policy, which is a mapping from states to actions that maximizes the expected cumulative reward over time. This is typically expressed as the discounted sum of rewards, where a discount factor (γ) between 0 and 1 determines the relative importance of immediate and future rewards.

2.7 Activation Functions – In this study we mainly focused on the ReLu activation function for our models. The Rectified Linear Unit (ReLU) is a popular activation function used in Artificial Neural Networks (ANNs) for introducing non-linearity into the model. ReLU is computationally efficient and helps mitigate the vanishing gradient problem, which can occur when training deep networks. The function is defined as:

$$ReLU(x) = \max(0, x)$$

ReLU takes an input value of x and returns the maximum between 0 and x. If x is positive, the function returns x, and if x is negative or zero, it returns 0. This simple, piecewise linear function effectively filters out negative values, retaining only positive activations in the network.

3. Checkers As An Environment

To create an environment for our checkers game we must consider what our agent, states, episodes, actions, and rewards will be.

Our agent will be the player which is interacting with the environment. We will train the agent in our environment to improve its performance.

The states can be expressed as the matrix of the checker's board with each pawn and king position. The game can be modelled using an 8 by 8 matrix, with an "X" representing player 1, and an "O" representing player 2. We can use this model to help visualize the board to a human player. A more compressed version that implements a 32 by 1 array can also be used, which neglects unreachable tiles in the game. This representation will be used for our models due to its reduced size, therefore increasing efficiency. Each element in the 32-bit array represents a reachable tile on the board. The value of the element in the array represents the piece or lack of piece in that position. If the tile is empty, then the element in the array is set to 0. Men are represented with a value of 1 or -1 and kings with a value of 3 or -3 depending on which player they belong to. This representation fulfils the Markov property as it contains all the relevant information for deciding the next move. The position of the pieces on the board and which player they belong to.

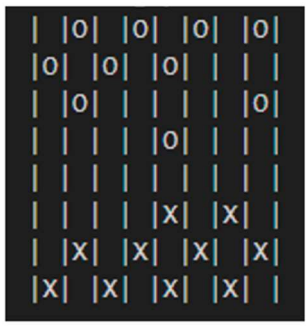


Figure 1: Representations of checker's board

An episode will be represented by a game played.

The actions are all legal moves that can be made in the current state. This includes moving pieces or capturing an opponent's piece. Once the agent chooses the action it is returned to the environment and the current state is updated to reflect the action made. This new state will then be passed on to the opponent so they can make a move. Once the opponent has made a move, the environment gives the agent the new board state.

The aim of the game is to win, this can be achieved by eliminating all the opponent's pieces from the board or preventing the opponent from

being able to make a legal move. The loss conditions follow from the winning conditions for the opponent. A draw can occur when the game goes on for more than 200 turns.

The reward function is set to +10 for a win and -10 for a loss. For the rest of the states during the game, the reward value is given by the output of the neural net.

4. My Learning Approach

The game of checkers is of a large enough complexity to make it impossible to use a tabular representation for the values of the different states. Estimates have placed the number of board states to be approximately 10^{20}, while the game tree is considered to be of rank 10^{31} . These numbers make it necessary to use a function approximator. The function approximator that is chosen is an artificial neural network.

To create our agent we use two neural networks and a reinforcement model. The models will be trained by playing against a random agent. Since the Checkers implementation follows the tournament rule that players must capture a piece when a capture is available, the random agent's policy more accurately reflects a greedy policy.

4.1 Metrics Model

Generative models are a type of statistical model that generate new data instances. They are commonly used in unsupervised learning to estimate probability and likelihood, model data points, and classify entities based on probabilities.

For our specific application, we will create a generative model that takes an array of metrics as input and then predict the probability of winning. This model will only consider the 10 heuristic scoring metrics for labeling.

These 10 metrics include:

- Capped – number of captured players
- Potential – potential moves
- Men – number of men left
- Kings – number of kings left
- Caps – number of capturable pieces
- Semicaps – number of pieces they arent uncapturable or capturable
- Uncaps – uncapturables
- Mid – number of pieces in the middle of the board
- Far – number of pieces on the far side of the board
- Won – game-winner

Once the model was established, we used a tree exploration technique to generate 100,000 game states that began from a specified board. For each game state, we computed its heuristic metrics. Using this data, we trained the algorithm to predict the outcome of the game. After training the model has a prediction accuracy of 85 percent.

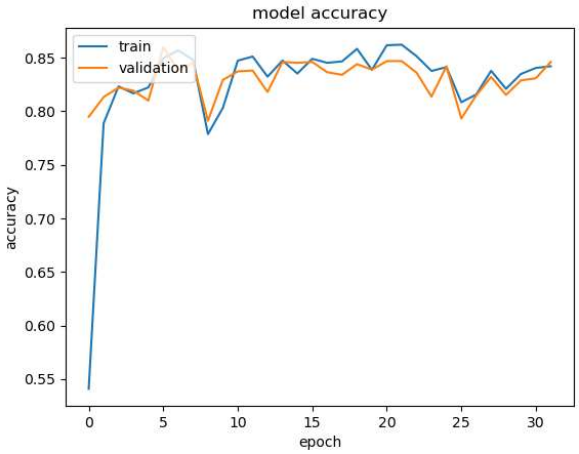


Figure 2: Accuracy of model during training

4.2 Board Model

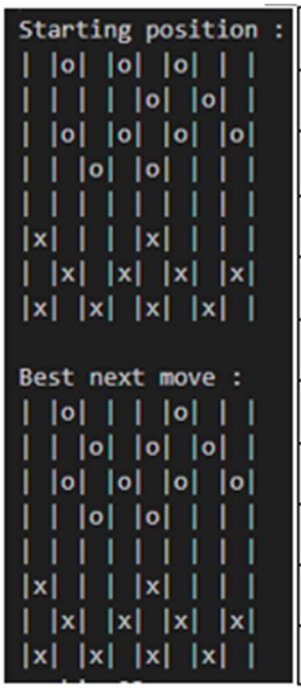
The board model endeavors to utilize the data procured from the metrics model to establish the initial Q-values in the context of the checkers board. In contrast to the metrics model, which incorporates heuristic labels as inputs, the board model accepts a 32-bit integer array representing the checkers board. Upon receiving the board and corresponding states, the model strives to ascertain the move that would maximize the probability of victory.

A confidence score is calculated by comparing the predictions made by the probabilistic labels made by the metrics model and the weak label calculated by the board model. This confidence value is used as the sample weight in the model, to emphasize the evaluation with higher probabilities.

$$\text{Confidence} = \frac{1}{1 + |Evaluation - Probability|}$$

The model is trained for 32 epochs. Once the model has been fit, the weights and model configuration are saved to JSON and h5 files, which will later be used for reinforcement learning.

In the example below, the board model has passed the board illustrated by the starting position. The board model is in control of the pieces represented by the character "o". From this starting position, there are 7 pieces that can be moved, 3 of the pieces can be moved diagonally left or right, with the other 4 pieces only able to move in one direction, in this case diagonally left for 3 pieces, and diagonally right for 1 piece, for a total of 10 possible moves. The table below has the value associated with each possible move. States 4 through 7 have a lower value compared to the other possible states. These states represent the possible moves that can be made by the 2 pieces furthest up the board. Moving either of these 2 pieces would lead to them being in a position where they can be taken by the opponent. Therefore they are evaluated with low values. Instead, the model will choose to make the move represented by state 0, which moves the piece at the back diagonally left, which leaves no pieces open for the opponent to take.



State	Q - Value
0	0.78000736
1	0.5922702
2	0.83670425
3	0.73784924
4	0.11197452
5	0.15785226
6	0.20754334
7	0.21244533
8	0.4777707
9	0.49358377

Figure 3: Game states and associated Q-Values

Our current model now embodies a basic level of expertise in playing checkers. The resulting output from this model will calculate V*(s), which represents the highest Q -Value for all possible actions in a state. Utilizing this approach to generate an initial Q-Value should significantly enhance the efficiency of the reinforcement learning, as our trained board model is considerably more adept at evaluating the board position in comparison to the alternative method of randomly selecting an evaluation.

4.3 Reinforcement Model

The reinforcement model uses deep q-learning to improve the board model's win rate. For each action state, the model will evaluate all the possible moves and chooses the one with the highest evaluation score, in the same way, the board model does. We use delayed reward to refit the model after n number of games have been played. Games that lead to a win have a positive reward; a loss has a negative reward. This is repeated for y generations. The q-values are updated using the Bellman equation. During each

generation, board positions are saved temporarily, along with the updated q values calculated using the Bellman equation, which is then used to refit the model at the end of the generation. This allows the performance of the model to increase during training.

The Reinforced model builds upon the learning acquired by both the metric and board model. To train the model, the agent will play 200 episodes of 200 games against a random agent, for a total of 40,000 games. A discount factor (γ) of 0.95 and a learning rate (α) of 0.5 are used for training. At the end of each generation, the model is refit using the data from the last 200 games. The following graph shows the agent's cumulative winning percentage for these 40,000 games (200 episodes).

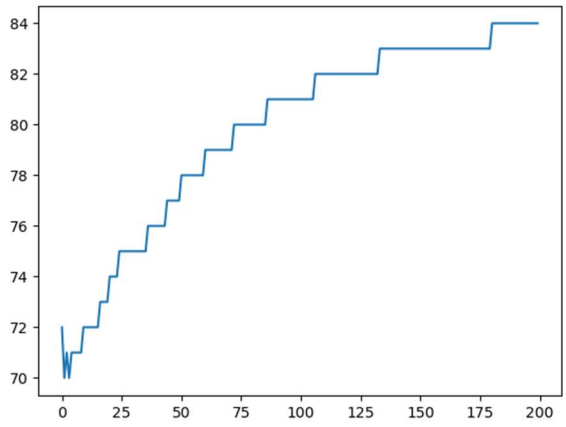
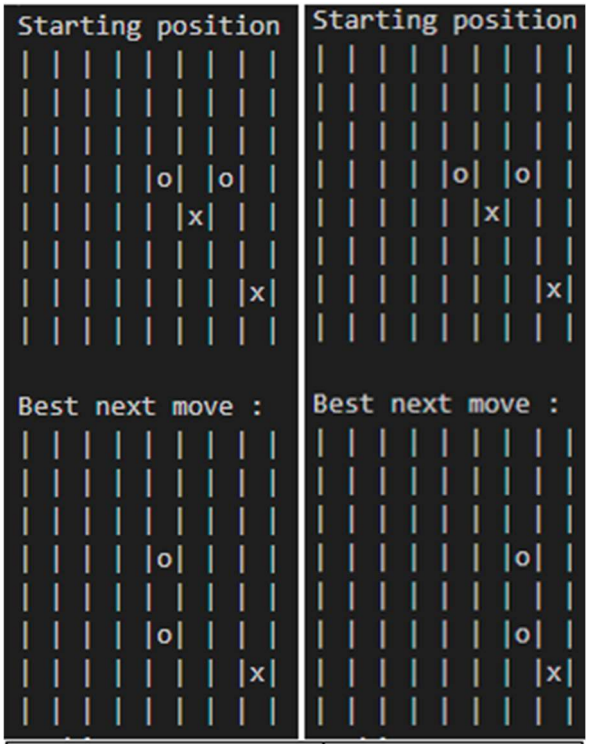


Figure 4: Training Graph

The agent's initial performance is indicative of the Board model's efficacy prior to the commencement of the reinforcement learning process. Consequently, as the reinforcement learning commences, the agent possesses an initial win rate of 70 percent against a random agent. Over the course of 200 generations, this win rate experiences an enhancement, culminating in a final win rate of 84 percent. This progression demonstrates a notable improvement in the agent's performance.

To demonstrate this improvement in the model's decision-making ability, a sample scenario is given to both the board model and the reinforced model. In the given situation, the agent and its adversary each possess a pair of remaining pieces. The agent is presented with the opportunity to capture one of the opponent's pieces; however, if the agent employs its leftmost piece for the capture, it simultaneously exposes itself to a counter-capture by the opponent. Consequently, it would be strategically advantageous for the agent to utilize its rightmost piece to execute the capture, thereby minimizing its vulnerability to subsequent retaliatory actions from the adversary. The figures below shows that the reinforced model makes the more advantageous move, whereas the board model goes for the inferior move.



Reinforced Model		Board Model	
Action	Q-Value	Action	Q-Value
0	-0.32578182	0	0.10349333
1	-0.14809692	1	0.10349333

Figure 5: Game states and associated Q-Values

4.4 Reinforcement Model Pure learning

A pure learning version of the reinforcement model was also created to test the RL agent's ability to become a better Checkers player over time without any prior knowledge of the game. To test this scenario, the board model was used in conjunction with the reinforcement learning model to train the agent. As we want to test the agent's ability to learn with no prior knowledge of the game, the board model is not trained beforehand. Instead, the agent will learn solely from the reinforcement algorithm. The agent will be trained in the same way as before, with 200 games in each of the 200 generations. The following graph shows the agent's cumulative winning percentage for these 40,000 games (200 generations).

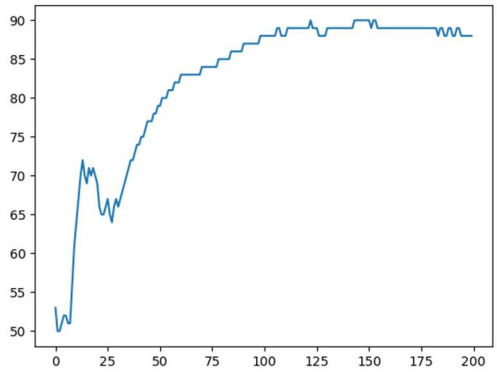


Figure 6: Training Graph

The initial performance of the agent, as indicated by a win rate of approximately 50 percent, demonstrates its equivalence to a random agent, which is anticipated given the absence of prior experience in checkers. Remarkably, the agent exhibits rapid learning capabilities. In merely the first 50 episodes, the agent attains a win rate exceeding 75 percent. Subsequently, the rate of learning decelerates, and the agent's win rate gradually approaches 90 percent. Ultimately, the agent concludes its training phase, attaining an 87 percent win rate against the random agent.

5. Experiments

In this section we will describe some of the experiments we performed and their outcome. Each test is repeated 5 times, with each test consisting of 100 games. The win rate for each test will be calculated as the number of games won or drawn against the opponent. An average of all the games will be presented at the bottom of the table.

5.1 Learning With Prior Knowledge

The first set of experiments tested the RL agent's ability to become a better Checkers player after being trained with prior knowledge of the game. This previous knowledge is acquired from the metric and board model (generative and discriminative models) implemented into the learning process.

5.1.1 Board Model vs Random

Following the training phase, the model competes in 5 sets of 100 games against a random agent. The results from these experiments are presented in the table below.

Games	Win Rate
0-100	72
101-200	72
201-300	69
301-400	71
401-500	70
Average	71

The board model achieves a win rate of 71 percent, thereby indicating its effectiveness in learning from the data acquired from the training of the metrics model.

5.1.2 Reinforced Model vs Random

Games	Win Rate
0-100	82
101-200	83
201-300	82
301-400	85
401-500	84
Average	83

The reinforced model achieves an 83 percent win rate against the random agent, showing an improvement relative to the board model.

5.2 Pure learning

The second set of experiments tested the RL agent's ability to become a better Checkers player over time after being trained without prior knowledge of the game.

5.2.1 Pure Learning Vs Random Agent

Games	Win Rate
0-100	87
101-200	87
201-300	86
301-400	87
401-500	88
Average	87

The pure learning model achieves an 87 percent win rate compared to the random agent. This performance slightly exceeds the performance seen in the reinforced model.

5.2.2 Pure Learning vs Reinforced Model

Games	Win Rate
0-100	50
101-200	52
201-300	51
301-400	56
401-500	50
Average	52

This test shows how the pure learning and reinforced model possess very similar checker's proficiency, with the pure learning model beating the reinforced model 52 percent of the time.

5.3 Further Testing

Upon the analysis of data procured from various learning approaches, we observed an unanticipated performance exhibited by the pure learning methodology. Consequently, we resolved to expand our investigation by extending the training of this model for an additional 50 epochs, to explore the possibility of further model performance improvement. However, rather than employing a random agent as the opposition, the model will now compete against a trained reinforcement learning model, which exemplifies a more advanced proficiency in the game of checkers.

Subsequent to the completion of the extended training against the reinforced model, the pure learning model will be assessed once more against both the reinforcement model and the random agent. This examination aims to determine whether the extended training has culminated in a discernible enhancement in the model's overall performance.

5.3.1 Pure Learning V2

To create the pure learning V2 model, the previous pure learning agent was trained against the reinforced model for 50 generations of 200 games. The results from this training are presented in the figure below.

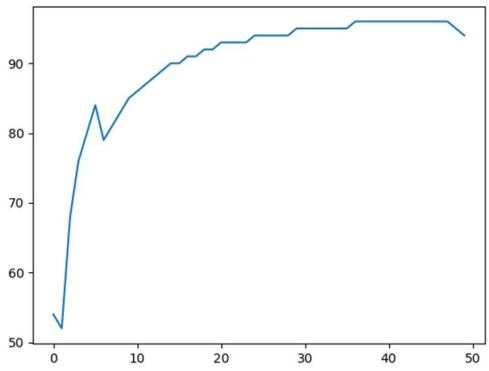


Figure 7: Training Graph

In the initial stages of the training process, the pure learning agent demonstrates a marginal advantage over the reinforcement model, securing victories in marginally over 50% of the matches, corroborating the findings from our previous tests. As the training progresses, the agent's win rate experiences a substantial improvement, culminating in a 94% win rate by the conclusion of the training period.

Now we have our trained pure learning V2 model, we can test it against the random agent and the reinforced model.

5.3.2 Pure learning V2 vs Random Agent

Games	Win Rate
0-100	94
101-200	94
201-300	95
301-400	95
401-500	94
Average	94

When tested against the random agent, the pure learning V2 model scores a 94 percent win rate. This demonstrates a quantitative improvement compared to the previous pure learning model which achieved an 87 percent win rate against the same random agent.

5.3.3 Pure learning V2 vs Reinforced Model

Games	Win Rate
0-100	50
101-200	52
201-300	51
301-400	56
401-500	50
Average	95

When tested against the reinforced model, the pure learning V2 model scores a 95 percent win rate. This is a major improvement compared to the previous pure learning model, which had similar performance to the reinforced model.

6. Evaluation - The experiments presented here aim to explore the agent's capacity to improve its checker's performance over time, with and without prior knowledge of the game. The methodology involves three distinct models: the metrics model, the board model, and the reinforced model, each designed to elucidate the impact of different learning mechanisms on the agent's performance.

In the first set of experiments, the agent is tested at multiple stages of its multifaceted learning process. Both the board and reinforced models were subjected to rigorous evaluation against a random agent to ascertain their respective proficiencies.

The initial model examined, designated as the board model, derives its training from the predictions generated by the aforementioned metrics model. In contrast to the metrics model, the board model necessitates a checkers board as input. This modification is an essential prerequisite, as the reinforcement learning algorithm demands a neural network model capable of processing board input. The primary objective of the board model is to assimilate the knowledge acquired from the preceding model to forecast the most advantageous move within a given board state. Consequently, this model creates an initial Q-value table, which can subsequently undergo refinement through reinforcement learning techniques. the board model exhibited a commendable 70 percent win rate against the random agent. This model possesses the notable advantage of being computationally frugal, in comparison to its reinforcement learning counterpart. The expedited training process of the board model, requiring merely a few minutes, renders it an attractive foundation upon which the reinforced model can be constructed. Despite the potential limitation of not attaining the same degree of expertise as the reinforcement learning model, the metrics model's substantially accelerated training time offers a valuable asset in certain circumstances.

Ultimately, the reinforcement model endeavors to augment the performance of the board model by employing the Bellman equation to recalibrate the model's weights. Contrasting the pure learning experiment, the reinforcement model commences with a 70 percent win rate, attributable to the knowledge previously

acquired. Upon completion of 200 training episodes, the model attains a win rate of 84 percent, a performance comparable to that achieved in the pure learning experiment. This outcome was somewhat disappointing, as the expectation was for this implementation to surpass the win rate observed in the pure learning experiment. However, unlike the pure learning experiment, this model does not exhibit signs of performance plateauing in the later stages of training, suggesting that an extended training duration with additional episodes may further enhance performance.

In the second set of experiments, the agent is tested in a scenario where it has no prior knowledge of checkers. This puts the model and agent at a disadvantage as it has no basis on how it should go about winning the game. Due to this, we expected the learning process to take longer compared to a model which has previous information to build upon. As the agent must learn how to play with no prior experience it begins with the same competency as the random agent, as expected. Less expected though is the rapid rate of improvement seen in the agent's ability. The agent is able to quickly improve its ability to play checkers, especially over the first 50 episodes of training, where it is quickly able to improve its win rate to over 75 percent. This indicates that the model is able to quickly improve its predictions using the Bellman equation to refit its model after each episode. After 100 episodes the model begins to plateau in performance, at a win rate of 85 and 90 percent. This might indicate that there are still improvements to be made in the reinforcement algorithm, to allow it to escape from this local maxima, and improve its performance further, getting closer to a 100 percent win rate.

In light of the findings gained from the preceding experiments, the decision was made to prolong the training of the pure learning model by pitting it against a more skilled checkers opponent; namely, the trained reinforcement model. The objective of this investigation was to ascertain whether the pure learning model's performance could be further enhanced, to escape the local maxima it seems to find itself in, not only in opposition to a random agent but also in games played against the reinforced model.

Upon evaluating the updated Pure Learning model following an additional 50 generations of training, a substantial performance enhancement was observed, particularly in competition with the reinforced model. The win rate escalated from 52% to an impressive 95%. When contending with the random agent, the win rate experienced an increase from 87% to 94%. The observed outcomes potentially indicate that the model could be overfitting to the reinforced model, given the substantial increase in the win rate against the reinforced model, while the improvement against the random agent remains relatively modest. However, it is important to consider that the pure learning model initially exhibited a higher win rate against the random agent, which inherently renders subsequent enhancements more challenging to achieve. It should also be noted that the model was able to surpass the plateau seen in the previous model, suggesting that it may not be a problem with the reinforcement algorithm, but instead the opponent it faces in training that determines its final performance.

In summary, the experiments demonstrate that each implementation possesses distinct advantages and disadvantages. The pure learning experiment employs a more straightforward checkers agent implementation, necessitating only a single model. However, the model appeared to be trapped in a local maximum, potentially resulting from the random initialization of weights due to the absence of prior knowledge. This appears to have been alleviated by training the second version of the pure learning model against a more proficient checkers agent. The mixed learning implementation, comprising three discrete models, seeks to address this issue by initializing the model's weights using a generative and

discriminative model to establish an initial Q-table. This approach enables the reinforcement algorithm to initiate with a 70 percent win rate, which subsequently progresses to a performance analogous to that of the first pure learning experiment. Notably, after 200 episodes, the model continues to exhibit improvement, indicating that an extended training period could potentially yield further performance enhancements.

7. Conclusion

Although Checkers has been solved, its solution is highly dependent on a database of built-in human knowledge. As an attempt to develop a Checkers player with a more applicable approach, this project created a checkers agent by utilizing both neural networks and a reinforcement algorithm. First, we created the metrics model that estimates the winning probability based on heuristic checkers metrics. Then, used it to create a board evaluation model. Next, using Deep Q-Learning, we proceeded to reinforce the board model to have more games ending with wins and draws instead of losses. Multiple experiments were conducted to test the agent's ability to learn the game of checkers over time. Experiments were also conducted where the agent is tasked to learn the game with no prior knowledge. From these experiments, we can conclude that reinforcement learning can be an effective approach to creating a checkers agent capable of playing at a satisfactory level, without the heavy computational cost of a perfect solution. Furthermore, it appears that the data the checkers model is trained on is a major contributor to its final performance. This phenomenon is evident in the pure learning model, wherein the agent's performance plateaued until it was subjected to training against a more proficient adversary. Should another agent be developed, it may prove advantageous to amass data derived from professional checkers matches and subsequently utilize that data for model training purposes.

7.1 Future Work – There are many possible options for future work that could be implemented to further improve the performance of the model we have created. Some examples that could work well with our implementation include increasing the number of generations, or games played per generation. Hyperparameter tuning such as tuning the discount factor or learning rate. To discover the optimal move, Alpha Beta pruning depth search techniques could be implemented into the decision process. This would allow the agent to look multiple moves ahead, improving its move decision-making process. We can also consider using expert games as an initial training dataset for the board model, instead of the heuristic metrics.

8. Acknowledgments

Thanks to Brian T. for the checkers game code, which I built the neural network and checkers agents on top of. The public library is available at:

https://github.com/Btsan/CheckersBot/blob/master/checkers.py

I would also like to thank my supervisor Elena Botoeva for her continuous help, support, and fast replies throughout.

9. References

Mitchell, T. M. (1997). *Machine Learning* (International Edition 1997 ed.)., volume 1 of 1. McGraw-Hill, Inc. New York, NY, USA: McGraw-Hill.

Ghory, I. (2004*). Reinforcement learning in board games. Technical Report* CSTR-04-004, Department of Computer Science, University of Bristol

Paperspace. (2022). *Building a Checkers Gaming Agent Using Neural Networks and Reinforcement Learning* <https://blog.paperspace.com/building-a-checkers-gaming-agent-using-neural-networks-and-reinforcement-learning/>

Schaeffer, J., N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu, and S. Sutphen. (2007) "*Checkers Is Solved*." Science 317.5844: 1518-522. Web.