Opposing Reinforcement Learning Agents in Checkers

Bryant Saunders
CS Department
Utah State University

***Abstract*—Training reinforcement learning agents efficiently requires optimal training practices which balance both speed and accuracy. There is a need to streamline training practices in the field of reinforcement learning, as training can be both time-consuming and computationally costly . This work utilizes a dual-agent checkers environment that enables modularity between the environment and the agents that compete against one another. The bulk of the work done in this project is in extending the environment to allow for Q-Learning AIs to play on either side of the board and observing learning rates in the form of win ratios between agents trained against opponents exhibiting different behaviors. This project demonstrates the differences in learning for reinforcement learning agents that train in different environmental conditions to demonstrate the efficiency of training agents against opponents that also evolve during the training. This project examines possible ways to expedite the training of a checkers AI, focusing on the training methods used to create said AI.**

***Index Terms*—reinforcement learning, checkers, q-learning, alpha-beta pruning, training**

I. Introduction

Artificial intelligence in games warrants a fair amount of research, due to the complexity of game environments. Games like chess and checkers have enormous amounts of game states, which have a great effect on the actions an agent can take. This subsequently makes training AI for games a complex issue, as developers must balance time constraints and accuracy in order to create efficient agents for games. In this project, we will be examining the use of dual-agent training in checkers. The main objective of the research is to examine differences in reinforcement learning agents that are trained against one another.

Training reinforcement agents to play checkers accurately can be done in a multitude of ways, such as training against a random agent, training against a person, or training against another reinforcement learning agent. All of these methods in theory will eventually result in a usable checkers agent, though some methods introduce unmanageable costs. Training against a random agent can result in a trained agent that is only trained to respond to nonsensical or inefficient players, while training against a human being is extremely time-consuming.

Meanwhile, training against another reinforcement learning agent grants the ability to run many episodes while still training against a competent and evolving opponent. Ideally, this means that the agent will only be exposed to the most relevant or useful game states. This can cut out a fair portion of training time, as the agent is not forced to react to states that are not usually reached in a standard game of checkers.

II. Methodology

This project utilizes the checkers environment developed by Sam Ragusa, which simulates a checkers board that allows players to take turns moving their pieces. The environment also allows for state-value information to be stored in a .json file, which is vital to keeping the control agent uniform throughout testing.

For this project, we will be looking at three different types of players:

- A player which randomly selects a move from a list of possible moves each time it takes its turn. This player learns nothing throughout the game, and will serve as the opponent to training the control agent.
- A player which utilizes Q-Learning to train. This player will be present in all training games and serves as the independent variable in the project.
- A player which utilizes Alpha-Beta pruning techniques to select optimal moves. This player offers some diversity in the environments that the Q-Learning agents will train in.

While this project only focuses on the above player types, the environment is easily extensible to allow for a variety of different players. Early drafts of this project considered implementing a neural network, though that idea was scraped due to time constraints.

For each training opponent, a Q-Learning Reinforcement Agent is implemented using the equation below. Each agent uses the same Q-Learning function, including the same learning rate and discount factor.

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underset{\text{New Q-Value}}{\alpha}[\underbrace{R(s,a)}_{\text{Reward}} + \overbrace{\underset{\text{Discount rate}}{\gamma}\max Q'(s',a')}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)]$$

Once an agent is trained, they are tested against the control agent by setting the random move chance to zero and the learning rate to zero. This means that each agent is only using the most optimal moves available to them.

The state of this project is a series of integers that represent different components of a player's current position in the game, including number of pieces for both sides and the center of mass for each player. This simplifies state frees up memory and allows for faster training, albeit at the expense of some accuracy.

The reward for a state is the differences between the currer agents number of pieces on the board and the opposing agent number of pieces on the board. King pieces are worth tw times the standard checkers piece. A large reward is given t the agent if it wins the game and a large negative reward i given if the agent loses a game.

Every Q-Learning agent is given a random move chance a the start of each round of testing. The random move chanc and learning rate of the agent is reduced after each game b a small amount.

The Alpha-Beta AI uses a traditional alpha-beta prunin algorithm with a depth of three to determine optimal moves As this is not a reinforcement learning agent, the Alpha Beta pruning algorithm serves as a good contrast to the Q Learning opponent. This algorithm has been shown to b extremely effective in checkers and similar games, whic means it will almost always choose a competitive move an therefore provide a good set of training data for its opponen

III. Results

Results are shown as a two-dimensional graph. The x-axis represents the number of games played for each round of training or testing. The y-axis represents the amount of games a specific AI has won in the past interval of training, defined by the user in each run.

The control agent (Fig. 1) is trained first, showing an increasing ability to win against a truly random agent. This agent's behavior is saved to an external file in the form of transition information, which can then be loaded bac into memory for each test run. This guarantees an identica opponent for each of the testing runs.

The Q-Learning agent trained against another Q-Learnin agent (Fig. 2) shows that neither AI is able to definitively lear a strategy that will win consistently against their opponen This is shown by the relatively stagnant win rates, in tha neither AI has a marked increase of wins as training goes or

The Q-Learning agent trained against the Alpha-Beta prun ing algorithm (Fig. 3) falls short entirely. While the agent i still updating its Q-Table, it is in no way competitive to th pruning algorithm and loses an overwhelming majority of th time.

After our three Q-Learning agents are trained (Randoml trained, Q-Learning trained, and Alpha-Beta trained) the tw agents trained with non-random methods are pitted against th randomly trained control agent.

Starting with the Q-Learning trained agent against th control agent, there is a trend for the Q-Learning trained agen to win more games than the control agent. The depicted graph (Fig. 4) shows the worst agent trained after repeating this experiment eight times. On average, the Q-Learning trained agent wins about 1.75 times the amount of games the control agent does, despite having one tenth the training time.

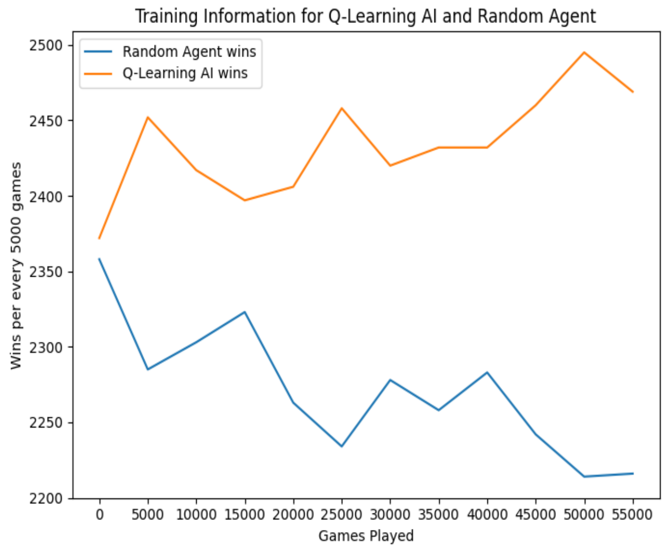

Fig. 1. Training graph for a Q-Learning agent trained against a random opponent. This Q-Learning agent will act as the control to test the other trained Q-Learning AIs

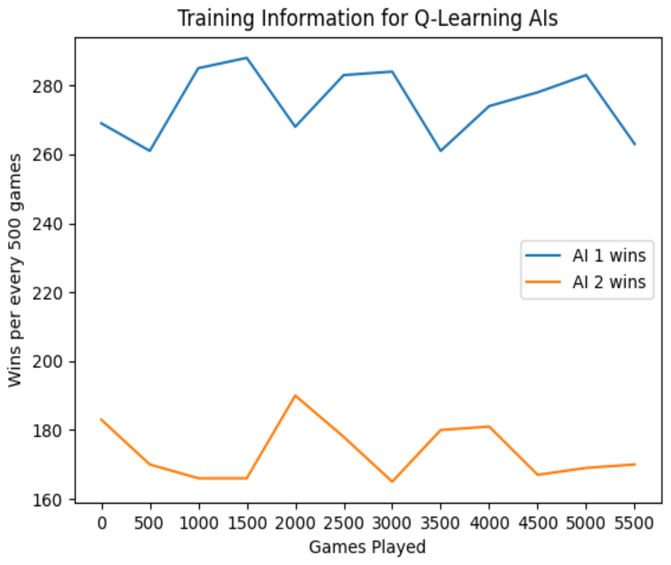

Fig. 2. Training graph for a Q-Learning agent trained against another Q-Learning agent.

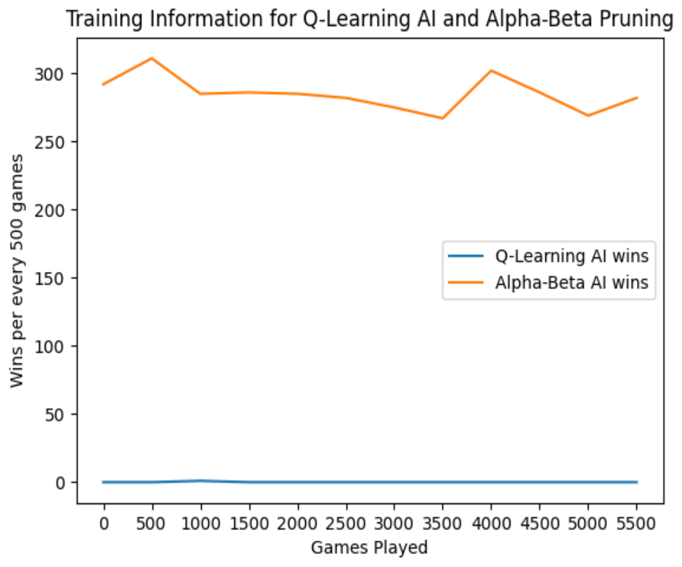

Fig. 3. Training graph for a Q-Learning agent trained against an Alpha-Beta Pruning Algorithm

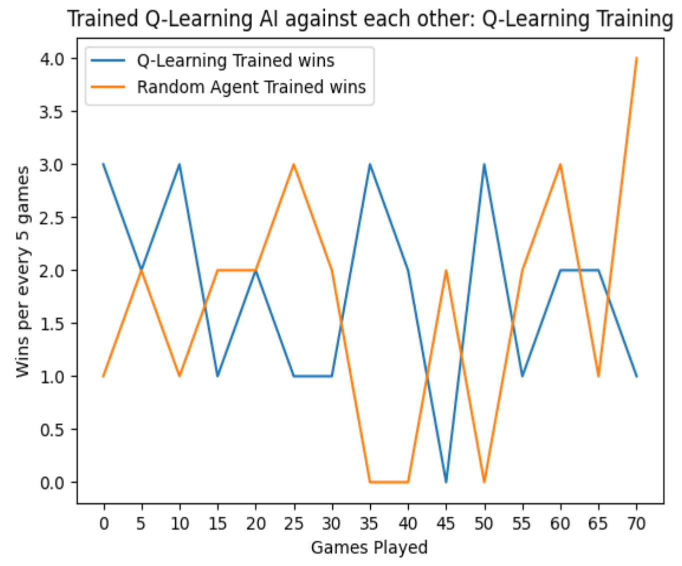

Fig. 4. Testing results for the Q-Learning AI agent trained against another Q-Learning agent (Refer to Fig. 2) and the control agent (Refer to Fig. 1).

The Alpha-Beta trained agent performs somewhat better than the Q-Learning trained agent. The depicted graph (Fig. 5) shows the worst agent trained after repeating this experiment eight times. On average, the Alpha-Beta trained agent wins about 1.85 times the amount of games the control agent does, despite having one tenth the training time. Additionally, Alpha-Beta trained agents tend to consistently do better, while Q-Learning trained AIs have more variance in their performance on different runs of this experiment.

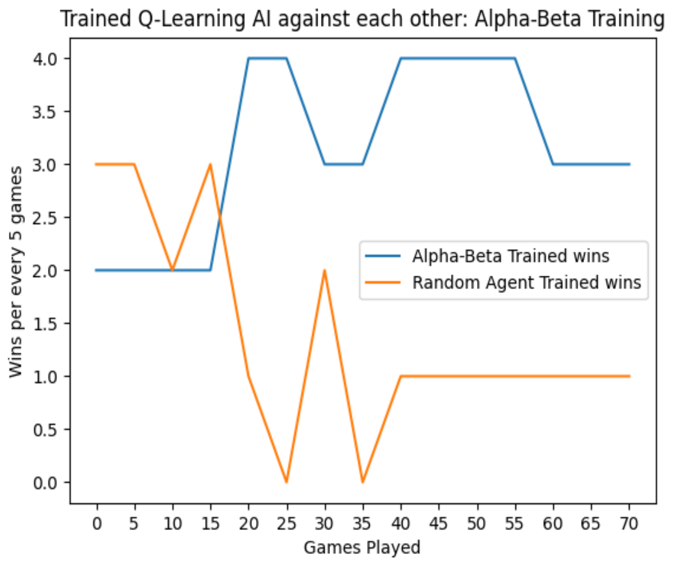

Fig. 5. Testing results for the Q-Learning AI agent against an Alpha-Beta Pruning Algorithm (Refer to Fig. 3) and the control agent (Refer to Fig. 1).

IV. Future Work

Future work on this project includes implementing even more opponent types for the Q-Learning AI to train against. A lot of effort was spent on researching a desirable neural interface, which ultimate fell flat due to time constraints. Testing different neural network architectures would add additional data sets that would be extremely valuable for this project.

V. Conclusion

Training reinforcement learning agents against one another seems to result in faster training times when compared to reinforcement learning agents that learn against random movesets. Identical agents eventually learn the same behaviors regardless of their training environments, but training against another reinforcement agent can reduce the time necessary to train an agent.

This project serves as support to the necessity of improving reinforcement learning techniques. As shown by the extreme accuracy of the Alpha-Beta pruning player, the underlying value functions that govern a reinforcement learning agent's behavior is the main determinant on how well that agent is trained. While theoretically the agents trained in this project could become as accurate as the Alpha-Beta pruning player, it is much more reasonable to invest in the later technique if the end goal is to create an AI that can win games. It is unnecessary to spend time training a simpler model when other models exist that perform just as well or better.

Acknowledgments

seoulai - Early prototyping for Q-Learning Agents: https://github.com/seoulai/

Sam Ragusa - Environment and Base Agents: https://github.com/SamRagusa/Checkers-Reinforcement-Learning

Btsan - Potential example of working neural network in checkers: https://github.com/Btsan/CheckersBot

Adil Lheureux - Further reading on neural networks in checkers: https://blog.paperspace.com/building-a-checkers-gaming-agent-using-neural-networks-and-reinforcement-learning/