Report

Implementing Othello Game Using Bayesian Neural Networks (BNN)

This report describes the implementation and training of a Q-Learning algorithm for the game of Othello, using Bayesian Neural Networks (BNN).

The game Othello, also known as Reversi, is a strategy board game for two players, played on an 8×8 uncheckered board. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed, and another disk of the current player's color is turned over to the current player's color.

The code implementation is divided into several parts: defining the Othello game logic, defining the Q-Learning algorithm, defining and training the Bayesian Neural Network (BNN), and then using the trained BNN to generate a policy for playing the game.

Exploring the Othello Game Logic:

Let's first take a closer look at the Othello class, which encapsulates the game logic. The 8×8 board is represented as a 2D array, with each cell containing either a 0 (empty), 1 (white piece), or -1 (black piece). The game state includes not only the board but also the current player's turn and the count of white and black pieces, which are updated after each move.

The validAction function is essential as it guides the possible moves at any given point. It calculates the available actions for the current player based on the game's rules. The function iterates through each cell on the board, and for each cell, it checks in all eight directions (up, down, left, right, and the four diagonals) for valid moves, adding them to a list which it then returns.

Q-Learning Algorithm:

The QLearning class is where the reinforcement learning takes place. The policy function, which implements the ε-greedy strategy, is a core component of this class. This function takes the current state as input and returns an action. With probability ε, it selects a random action (exploration), and with probability 1-ε, it selects the action with the highest predicted Q-value (exploitation).

The collect_experience function uses this policy to play the game multiple times, collecting experiences (state, action, reward, next state) from each game. These experiences are stored in a buffer, which serves as the dataset for training the BNN. The buffer is implemented as a circular buffer, which means that when it reaches its maximum capacity, it starts overwriting the oldest experiences with the new ones.

Understanding the Bayesian Neural Network (BNN):

The BNN is a type of neural network that can model uncertainty in its weights. This is achieved by representing the weights as probability distributions instead of fixed values. In this implementation, each weight in the BNN is represented as a Gaussian distribution, characterized by a mean and a standard deviation. These parameters are learned during training, allowing the BNN to adjust the uncertainty of each weight based on the data.

Training the BNN involves adjusting the weights to minimize the difference between the predicted Q-values and the actual Q-values (rewards). This is done using a variant of the backpropagation algorithm, which takes into account the uncertainty of the weights. The training process continues over multiple epochs, with each epoch consisting of one pass through the entire buffer.

Playing the Game with Trained BNN:

The final part of the implementation is using the trained BNN to play the game against a human player. The human player is prompted to enter their moves manually, while the AI uses the policy generated by the BNN to make its moves. After each move, the new game state is displayed, and the process continues until the game ends.

The AI's moves are determined by the policy function, which uses the trained BNN to predict the Q-values of all possible actions from the current state. It then selects the action with the highest predicted Q-value. However, to ensure that the AI continues to explore different strategies, the ε-greedy strategy is still used, meaning that with a small probability ε, the AI will select a random action instead of the one with the highest Q-value.

Future Work:

The implementation described in this report provides a solid foundation for developing a reinforcement learning AI for the game of Othello. However, there are several potential improvements and extensions that could be explored in future work. These include:

Fine-tuning the BNN: The performance of the AI could potentially be improved by fine-tuning the structure and hyperparameters of the BNN. This could involve changing the number of layers, the number of neurons per layer, the type of activation function, the learning rate, and other parameters.

Improving the Exploration-Exploitation Trade-off: The ε-greedy strategy used in this implementation is simple and effective, but there may be more sophisticated strategies that could lead to better performance. For example, one could implement an ε-decay strategy, where the probability of selecting a random action gradually decreases over time. This would allow the AI to explore more in the early stages of learning and exploit more in the later stages.

Multi-Agent Learning: In this implementation, the AI learns to play the game by playing against itself. However, it could potentially learn more effective strategies by playing against other AI agents with different policies. This would require implementing a multi-agent learning environment.

Transfer Learning: The learned BNN could potentially be used as a starting point for training an AI to play other similar board games. This would involve transferring the weights of the BNN to a new AI and continuing the training with the new game. This approach, known as transfer learning, could significantly reduce the amount of training required for the new game.

Exploring Other Reinforcement Learning Algorithms: While Q-Learning is a powerful reinforcement learning algorithm, there are many other algorithms that could potentially lead to better performance. These include SARSA, Deep Q-Learning, and various forms of policy gradient algorithms.

Conclusion:

This project demonstrates the potential of combining reinforcement learning with Bayesian neural networks to develop an AI for playing the Othello game. The AI is capable of learning effective strategies for the game through self-play, and it is able to handle the inherent uncertainty in the game by modeling uncertainty in its weights.

It is our hope that this work will inspire others to explore the exciting field of reinforcement learning and its many applications. Whether it's for developing sophisticated game-playing AI, optimizing industrial processes, or enabling autonomous vehicles, reinforcement learning has the potential to revolutionize many areas of technology and society.