CS 5100 – Foundations of Artificial Intelligence

Project Report

Minesweeper AI Bot Using Deep Reinforcement Learning

Mitul Nakrani, Mihir Walvekar, Girish Raut, Yutika Chougule

Abstract

The Minesweeper AI bot project is designed to automate gameplay in the classic Minesweeper game using advanced reinforcement learning techniques, specifically Deep Q-Networks (DQN) and Dueling Deep Q-Networks (DDQN). The project is implemented in Python and utilizes the PyTorch library for building and training neural network models.

The core of the project consists of a dynamic game environment, and methods for interacting with the game. The Minesweeper game environment handles game state initialization, updates, and resets. The project contains two main model classes. Each class encapsulates a model (DQN or DDQN), The game's graphical interface provides a visual representation of the game state if enabled.

Overall, this project demonstrates the application of modern reinforcement learning techniques to a traditional puzzle game, showcasing the potential of AI to solve complex problems in a structured environment.

Project Definition

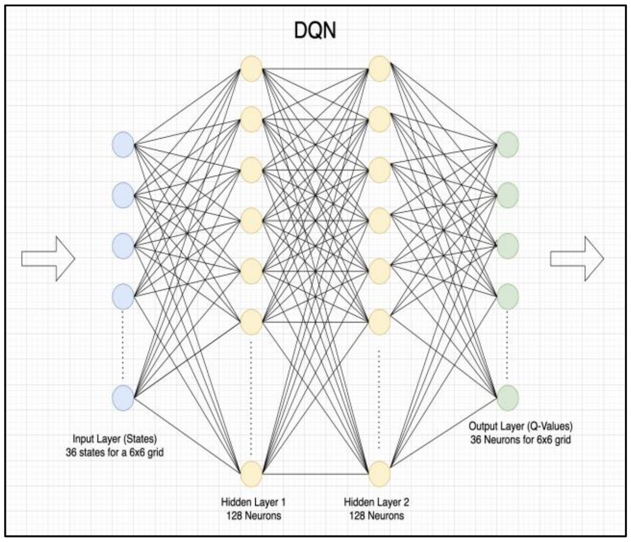
- **Objective**: To create an AI that can efficiently and effectively play Minesweeper, aiming to maximize the win rate over a series of games.
- **Models**:
  - DQN (Deep Q-Network): A reinforcement learning algorithm that learns the value of each possible action in each state.
  - Dueling DQN (Dueling Deep Q-Network): An architecture for a neural network that separately estimates (a) the state value and (b) the advantages of each action, which can lead to better policy evaluation in some environments.
- **Environment**: The game environment is Minesweeper, where the AI interacts with the game by choosing cells to reveal, aiming to avoid mines.
- **Game Representation**: The game is represented as a grid where each cell can be in a state of hidden or revealed. The AI uses a combination of the current state and a "fog of war" (cells that are still hidden) to decide its actions.
- **Visualization**: The project includes a GUI component for visualizing the game state, which can be toggled on or off.
- **Performance Metrics**: The AI's performance is measured by its win rate across multiple games, with specific attention to avoiding immediate losses (stepping on a mine on the first move).

Reason for Choosing this project:

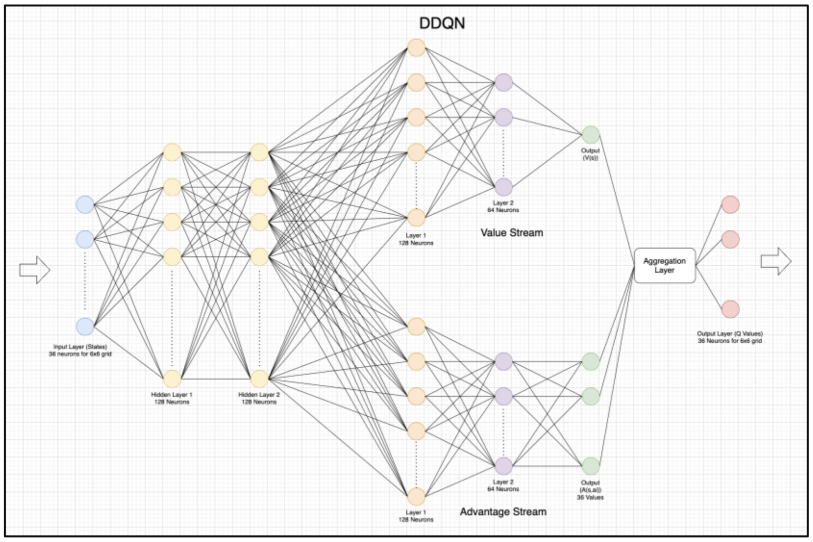
- **Educational Value**: Implementing an AI for Minesweeper can be a great learning experience in areas such as reinforcement learning, neural networks, and game theory.
- **Challenge**: Minesweeper is a partially observable and stochastic environment, making it a challenging and interesting problem for AI research.
- **Research Interest**: The project could be part of a research study aiming to compare the effectiveness of different neural network architectures (DQN vs. DDQN) in game-playing scenarios.

Methodology

Model Selection



Deep Q Network architecture



Dueling Deep Q Network architecture

Deep Q Network*:* A DQN architecture represents a single multi-layered neural network that outputs a vector of Q-Vaues from each action. A Q value represents the expected reward from taking a particular action in each state. In our DQN model we decided to use 2 hidden layers with 128 neurons each. This allows us to use a reasonable level of representational capacity while not making the model too big. This neither causes overfitting nor underfitting of the data. We chose the ReLU (Rectified Linear Unit) as our activation function to introduce nonlinearity within the dataset.

Dueling Deep Q Network*:* On the other hand, the Dueling Deep Q Network architecture (2 layers of 128 neurons) splits the neural network into 2 streams. The first stream (Value stream) is a hidden layer with 64 neurons that calculates the value of being at a particular state and the second stream (Advantage stream) which calculates the advantage of taking a particular action over all other actions in that state. An aggregation layer combines the values to output a vector of Q-Values. The reason for the splitting of the neural network into separate streams is that it allows the model independently to calculate the value of being in a state without needing to learn the effect of an action on the environment.

The selection of Deep Q-Network (DQN) and Dueling Deep Q-Network (DDQN) models for the Minesweeper bot project is driven by several factors that make these models particularly suitable for the challenges presented by the game of Minesweeper:

1. **Handling Discrete Action Spaces**: Minesweeper involves a discrete set of actions (choosing cells to reveal), which aligns well with the capabilities of DQN and DDQN models. These models are designed to output a value for each action in each state, making them ideal for environments with discrete and finite action spaces.
2. **Learning from Sparse Rewards**: In Minesweeper, rewards are typically sparse and delayed (often only realized at the end of the game when the player wins or hits a mine). DQN and DDQN are effective in environments where learning from delayed rewards is necessary because they use the Bellman equation to bootstrap the reward information from future states, helping to propagate reward information back to earlier states.
3. **Generalization Over States**: Both DQN and DDQN use deep neural networks, which can generalize across many states by learning features from the raw input state representation. This is beneficial in Minesweeper, where the number of board configurations is extremely large.

4. **Stability and Convergence**: DQN introduces techniques such as experience replay and fixed Q-targets to stabilize training in environments with high variance in state transitions and rewards, like Minesweeper. Experience replay allows the network to learn from a more diverse set of experiences, while fixed Q-targets help mitigate the feedback loop issue in Q-learning updates.
5. **Advantage of Dueling Architecture (DDQN)**: The dueling architecture used in DDQN separately estimates the state value and the advantage for each action. This is particularly useful in Minesweeper where the value of many actions (cells to reveal) may be very similar in many states. By decoupling the estimation of state values and the advantages of actions, DDQN can learn more precise Q-values, leading to better policy evaluation and potentially more strategic gameplay decisions.

Experimental Setup

- Minesweeper Class: This class defines the Minesweeper game environment. It initializes the game grid, places bombs, and generates hints (numbers indicating the count of adjacent bombs). The choose method simulates the action of choosing a cell, which can result in uncovering a cell, hitting a bomb, or uncovering an area of zero-valued cells.
    - ‘unfog_zeros’ Function: This function is used for the "flood fill" feature, where selecting a cell with zero adjacent mines uncovers all connected zero-valued cells and their immediate neighbors.
- DQN and DuelingDQN Classes: Both classes define neural network architectures for learning the optimal policy. The DQN directly outputs Q-values for each action from the state input, while the DuelingDQN separates the estimation into state values and advantages for actions, potentially leading to more stable learning.
    - ‘act’ Method: Both models use an epsilon-greedy strategy for action selection, balancing exploration and exploitation. As training progresses, epsilon is typically decayed to favor exploitation over exploration. The epsilon decay only occurs when during training the loss reduces to a certain threshold for each iteration.
- DisplayGame Class: Manages the graphical display of the Minesweeper game using Pygame. It updates the game's visual representation in real-time, showing changes as the AI makes decisions.
- AI_Player_DQN and AI_Player_DDQN Classes: These classes encapsulate the interaction between the AI models and the Minesweeper environment. They manage the loading of trained models, execution of actions in the environment, and rendering.
    - ‘playMultipleGames_DQN’ and ‘playMultipleGames_DDQN’ Functions: These functions automate the process of playing multiple games, allowing the AI to be tested over many episodes. They track performance metrics such as win rates.

Learning and Testing Process:

1. Initialization: The AI models are initialized with random weights.
2. Training Loop:
    a. The AI interacts with the environment by selecting actions based on its current policy (initially random due to high epsilon).
    b. The outcomes of these actions (next state, reward, and whether the game ended) are stored in a replay buffer.
    c. Periodically, the AI samples from this buffer to update its policy based on the reward received and the estimated future rewards (using the Bellman equation).
    d. The network weights are updated using backpropagation to minimize the difference between predicted Q-values and target Q-values.

3. Model Evaluation: After training, the performance of the AI is evaluated by playing multiple games with the trained model, where the exploration rate (epsilon) is set very low or to zero, ensuring that the AI exploits its learned policy.
4. Visualization: Optionally, the games can be rendered using the DisplayGame class to visually verify the AI's strategy. (Not recommended during training as it considerably increases the training time).

Parameter Usage in Testing:

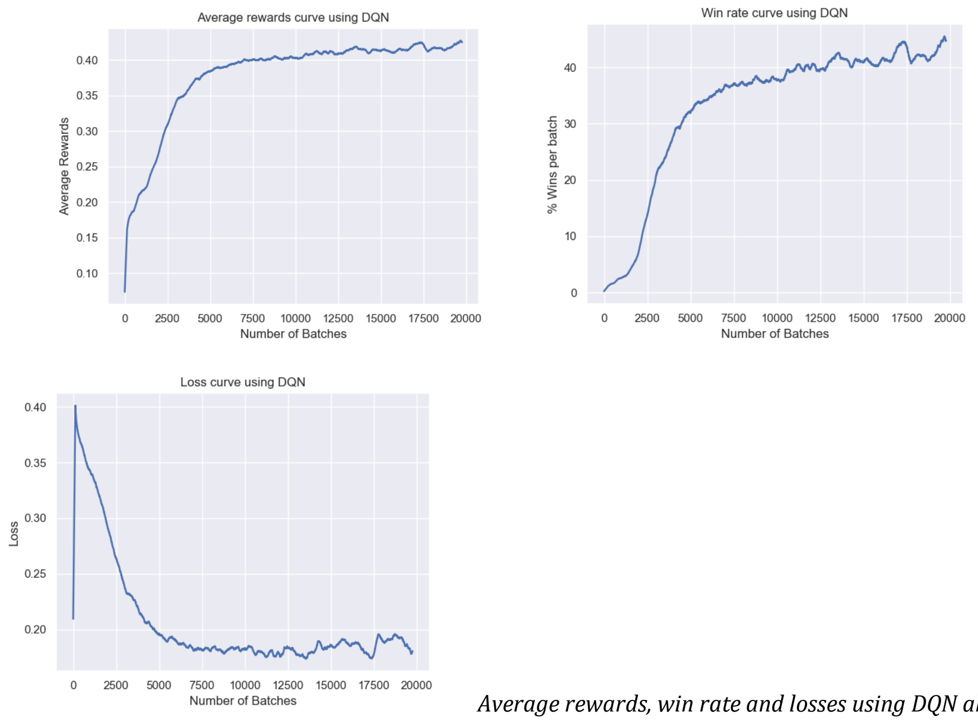
→ Trained Models: The AI uses the weights learned during training to make decisions.
→ Epsilon: Set to zero or a very low value during testing to use the learned policy without random actions.
→ Replay Buffer: Not used during testing, only during training.

This setup allows the AI to learn from many diverse scenarios within the Minesweeper game, improving its policy over time through reinforcement learning techniques. The testing phase then evaluates how well the AI has learned to play the game, aiming for a high win rate and strategic gameplay.

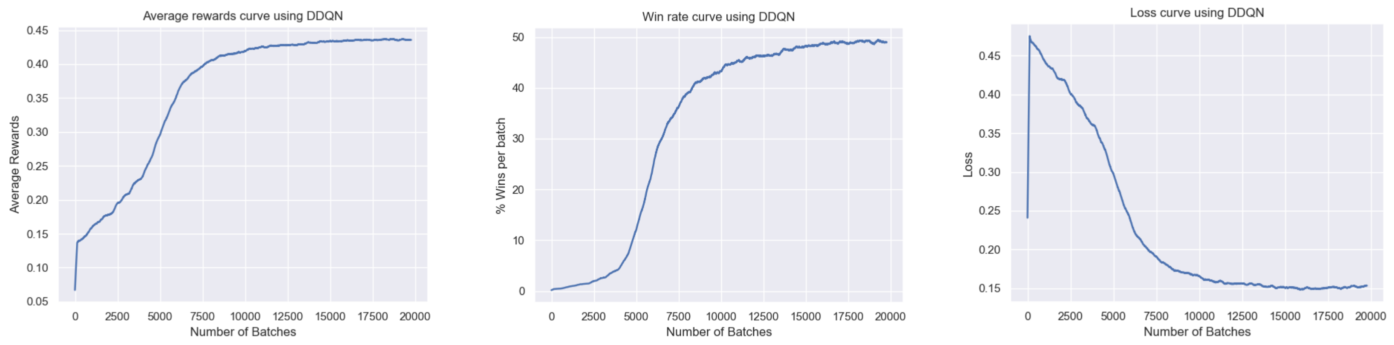
Results

<u>Training results</u>:

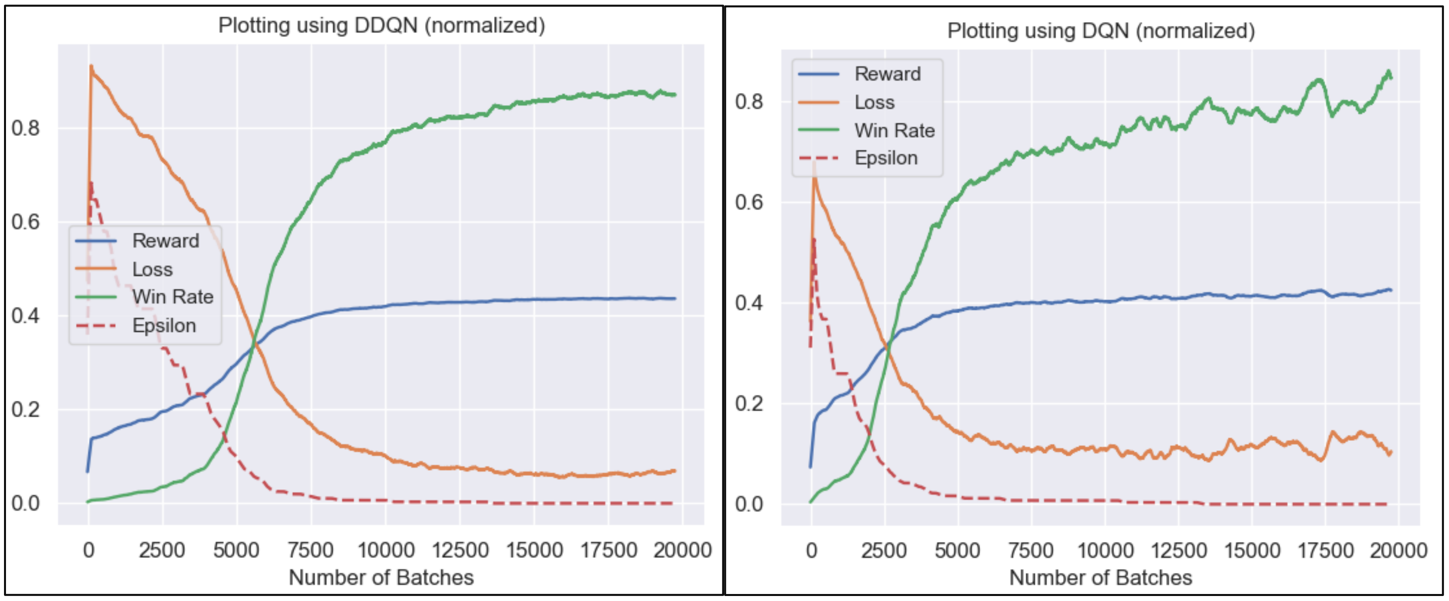
The agent was trained using the two models- DQN and Dueling DQN as discussed in the above experimental setup. Both training courses were performed for 20000 epochs, with a batch size of 4000 games per epoch. Parameters like rewards, win rates and losses, and epsilon decay per epoch were logged and plotted to study trends and differences between the two models.



Average rewards, win rate and losses using DQN algorithm



Average rewards, win rate and losses using Dueling DQN algorithm



Average rewards, losses, win rates and epsilon decay per epoch (normalized to a scale of 1 for plotting)

<u>Testing results:</u>

The weights from the trained models using DQN and Dueling DQN were saved, and 1000 games were played using those weights to determine the percentage of wins. The instances where first click causes a loss (clicked on a mine on the first click of a new game) were removed, and the win rate was then calculated as the percentage of wins in the number of games played.

The agent trained using DQN algorithm won approximately **60%** of the games, while the win rate using the agent trained on the Dueling DQN algorithm was close to **74%**.

Discussion/Conclusion

<u>Rewards and Learning</u>: The DDQN seems to do a better job at improving its score as it plays more games, which means it is learning well. The DQN also gets better, but not as smoothly as the DDQN.

<u>Losses</u>: Both the DQN and DDQN made fewer mistakes as they learned, but the DDQN made fewer overall, showing it probably learned a better way to play the game.

Epsilon decay: Both agents were curious at the start, trying different moves, but over time they both started using what they had learned. The DDQN was a bit more cautious and spent more time exploring, which might be why it did better in the end.

Win Rate: The DDQN won more games than the DQN, with about 74% wins compared to DQN's 60%. This shows the DDQN is likely making smarter moves.

Overall, the DDQN beats the DQN. It not only scores higher more consistently but also makes fewer mistakes and wins more games. This tells us that the DDQN might be a better choice for games or problems where you need a smart and reliable solution.

Link to GitHub Repository

Minesweeper AI Bot Using Deep Reinforcement Learning

Contributions

All the members of the group equally contributed to the project. Tasks were divided about defining the environment and the GUI, choosing the best model, and the training, testing and fine-tuning of the model to fit a game like Minesweeper.

References

1. https://minesweepergame.com/strategy/how-to-play-minesweeper.php
2. *Gardea, Luis, Griffin Koontz, CS RyanSilva and Autumn. "Training a Minesweeper Solver." (2015).*
3. Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.
4. Gardea, Luis, Griffin Koontz, CS RyanSilva and Autumn. "Training a Minesweeper Solver." (2015).
5. Tang, Yimin & Jiang, Tian & Hu, Yanpeng. (2018). A Minesweeper Solver Using Logic Inference, CSP and Sampling.
6. Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2016. Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16). JMLR.org, 1995–2003.
7. https://digitalcommons.wku.edu/cgi/viewcontent.cgi?article=4558&context=theses