Othello board game

Project 4 - Reinforcement learning MT7051

Florence Hugh, August Jonasson & Amanda Tisell

2025-03-19

Introduction

- Two-player board game with black and white pieces.
- Whoever controls the most squares at the end wins.

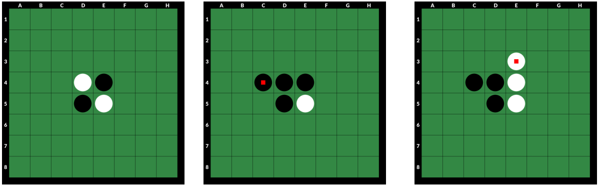


Figure: Starting state and first two moves of a game.

Challenges

- How to assess training for unsolved game?
  - Reduce to 6×6 Othello where known that white has an advantage.
  - Reduces state space from $\approx 10^{28}$ to $\approx 10^{12}$ (still huge).

Challenges

- How to assess training for unsolved game?
  - Reduce to 6×6 Othello where known that white has an advantage.
  - Reduces state space from $\approx 10^{28}$ to $\approx 10^{12}$ (still huge).
- How to learn without going through the whole tree of games?
  - Monte Carlo Tree Search (MCTS).

Challenges

- How to assess training for unsolved game?
  - Reduce to 6×6 Othello where known that white has an advantage.
  - Reduces state space from $\approx 10^{28}$ to $\approx 10^{12}$ (still huge).
- How to learn without going through the whole tree of games?
  - Monte Carlo Tree Search (MCTS).
- How to deal with agent/opponent?
  - Use self-play with minimax algorithm.

MDP framing

- $|\mathcal{S}| \approx 10^{12}$
- $|\mathcal{A}| \leq 32$
- Reward function:

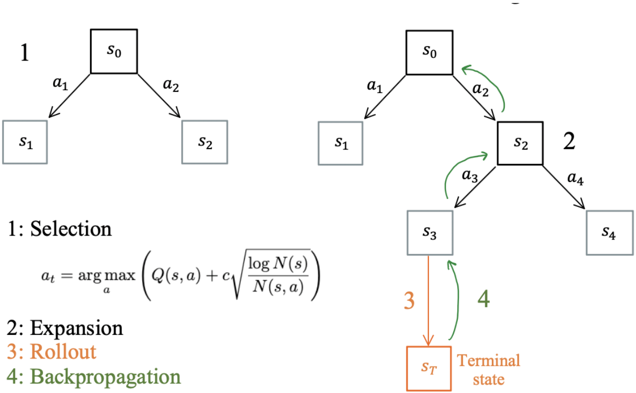
$$\begin{cases} 1 & \text{if black wins the game} \\ 0 & \text{if the game is drawn} \\ -1 & \text{if white wins the game} \end{cases}$$

- Dynamics are completely deterministic
- Current state is represented by an 6×6 matrix s with

$$s_{ij} = \begin{cases} 1 & \text{if square } (i,j) \text{ is occupied by a black piece} \\ -1 & \text{if square } (i,j) \text{ is occupied by a white piece} \\ 0 & \text{if square } (i,j) \text{ is blank} \end{cases}$$

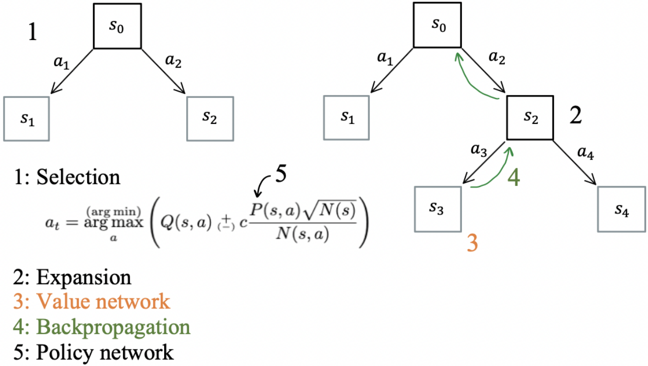
Monte Carlo Tree Search (MCTS)

- Tree-based *decision-time planning* algorithm for episodic tasks.
- Each iteration consists of four steps.



Modified Monte Carlo Tree Search (MCTS)

- Selection based on minimax algorithm and a policy network.
- Replace the rollout policy with a value network.



Tree Traversal

Action selection when black (white) to move

$$a_t = \overset{(\arg\min)}{\arg\max_a} \left(Q(s,a) \underset{(-)}{+} c \frac{P(s,a)\sqrt{N(s)}}{N(s,a)} \right)$$

$Q(s,a)$: output from value network or backpropagated average value

c : constant controlling the greediness of the action selection

$P(s,a)$: policy network output for taking action a in state s

$N(s)$: number of state visits

$N(s,a)$: number of state-action visits

Value network architecture

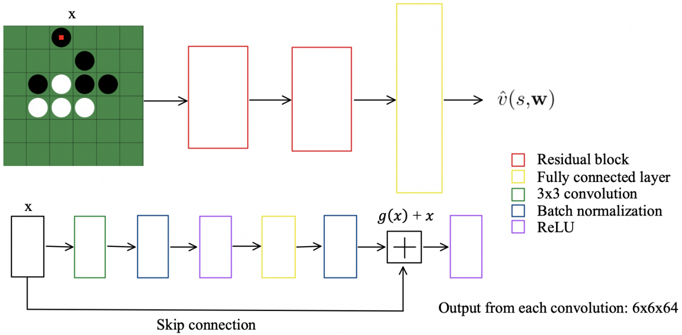


Figure: ResNet inspired convolutional neural network.

Policy network architecture

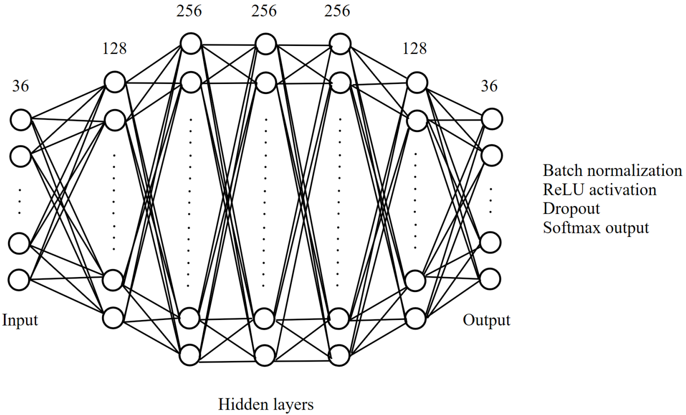
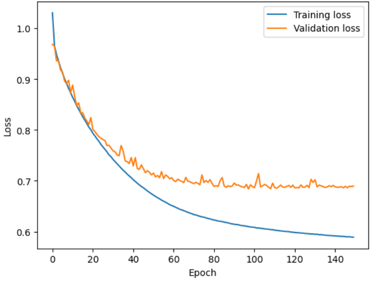


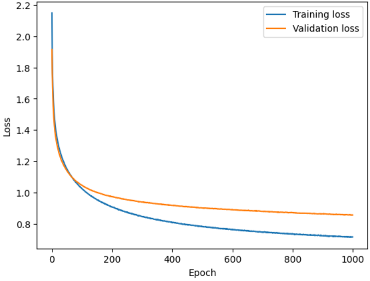
Figure: Fully connected feedforward neural network.

Training the networks for our MCTS

- Trained the value network on 100 000 randomly simulated games.
- Simulated games from MCTS + value network and used these to train policy network.



(a) Value network

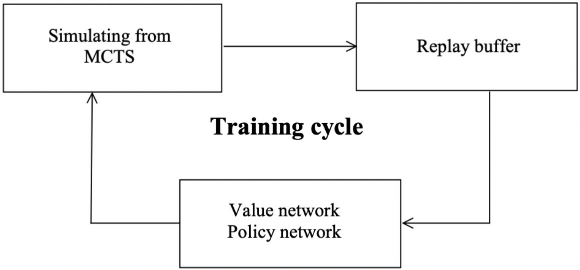


(b) Policy network

Figure: Initial training

Training the agent

- Online simulation (Loop over 30 training cycles):
  - MCTS with 25 episodes per cycle.
  - Update the weights for value and policy network.



Training curves: Value network

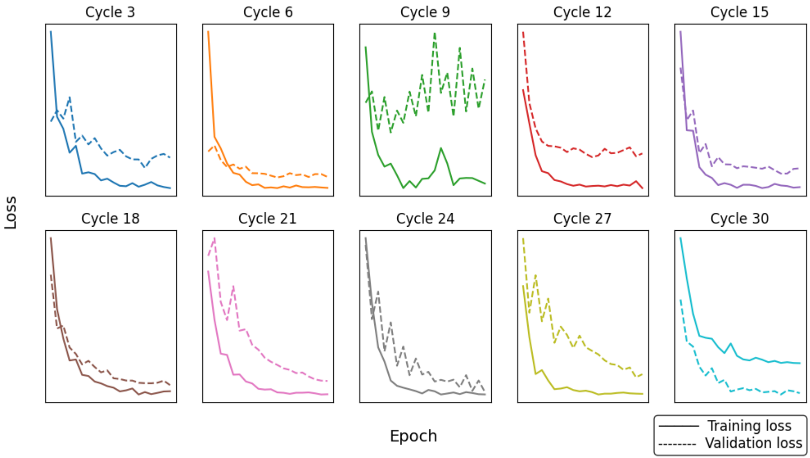


Figure: 20 epochs of training from replay buffer within each cycle.

Training curves: Policy network

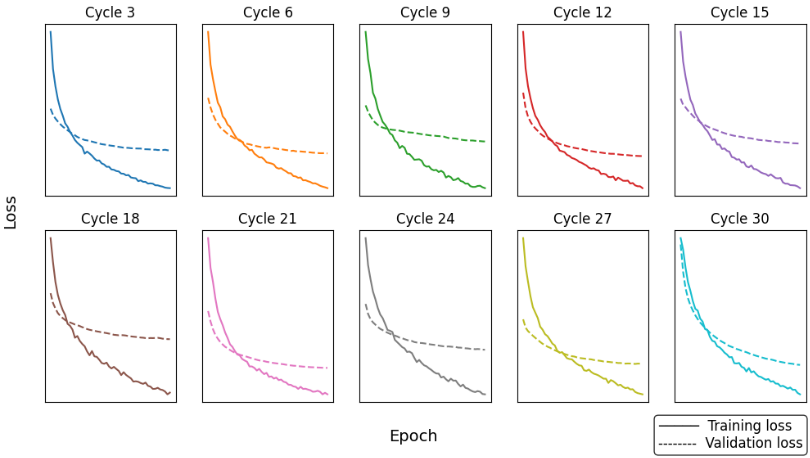


Figure: 50 epochs of training from replay buffer within each cycle.

Results

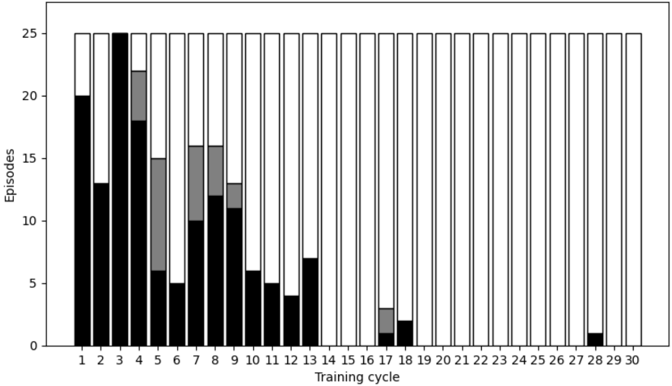
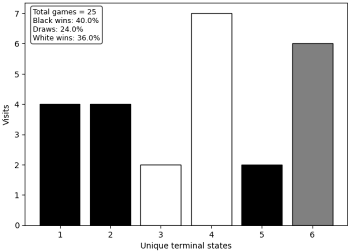
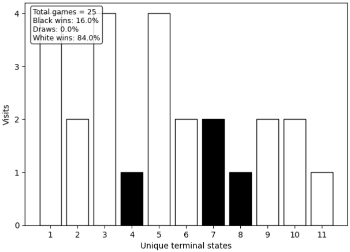


Figure: Win and draw distribution after each cycle.

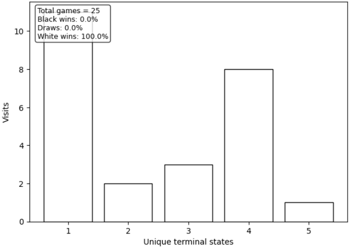
- White becomes increasingly more dominant.



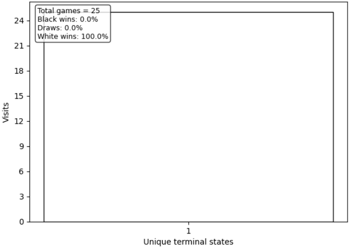
(a) Cycle 7



(b) Cycle 12



(c) Cycle 23



(d) Cycle 30

Further Improvements

- (Use an accumulating replay buffer) :(
- Improve the target for policy network.
- Train one neural network incorporating both value and policy.
- Customize environment for parallel processing.
- Play games (as a human) against the agent.