Othello CNN - Part 2

Anthony Hordesky and Luka Starcevic

Brief Background/Goal

- Goal was to to further develop our Othello move prediction bot using:
  - Reinforcement learning via self-play
  - Performance benchmarking/testing
- Ended up developing a self-play program that generates game data that we used to further train our CNN model
- Same goal stands for it to eventually be used as a practice tool for professionals or as just a fun program to mess around with

Self-Play Data Generation

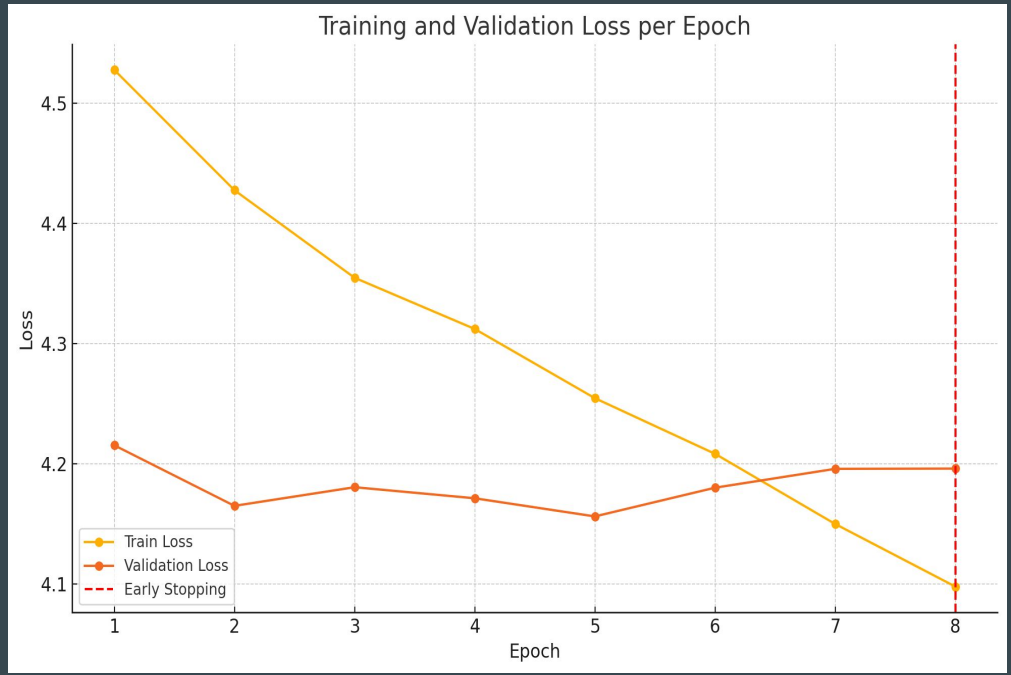
- Our self-play program generates game data for training through two main phases:
  - Exploratory self-play
  - Structured adversarial training
- The first phase of data generation is via self-play using temperature-adjusted move selection. Model plays itself at varying temperature levels that essentially control randomness of move selection.
  - Scaling is from 0.5 to 1.5 at intervals of 0.5. 0.5 or low temperature prioritizes high-confidence moves to reinforce known strategies
- The second phase of data generation is where we let the supervised CNN model play against our minimax algorithm at increasing depths from 1 to 8

Changes

- self_play.py
  - Initializes the self-play object that creates a global board object, othello game object, and opens the supervised CNN model for gameplay
  - Begins the self-play data generation method where a specified number of temperature-adjusted games are played by the CNN model and then a black and white game played against the minimax algorithm from ply 1 to ply 8
  - Generated 1016 games to continue training our model on
- sptools.py
  - Just a collection of methods of our Board and Othello object classes
  - find_available_moves() and search() for Board and minimax() for Othello.
- predict_move.py
  - Accepts a temperature hyperparameter used for scaling logits before PyTorch softmax conversion
  - Pytorch softmax is an activation function that turns the raw logits or model output values into probabilities, which is not exactly needed for move prediction but it's helpful to learn about
  - Temperature scaling modifies the logits before applying softmax to control how "peaked" or "flat" the move probability distribution becomes
  - By applying temperature-adjusted softmax only to legal moves, low temps (usually <1) return the highest-probability and high temps (>1) return a sampled or more random move

Training

- Same training method was taken and results from our run are below:
  - Training is 80% and val is 20%
  - No test set for final benchmarking



Performance Benchmarking/Testing

- We decided to change how the model played the minimax bot by letting it play a game as each color for increasing ply depths from 1 to 6.
- This method was devised to show how our newly trained model performs against varying difficulties - depth 1-3: basic, short term capture focus, and depth 4 - 6: intermediate, positional focus.
- The results of this performance benchmarking/testing are recorded in the table in the next slide

Testing Results

Game	Starting Color (CNN)	Minimax Ply Depth	Winner	Score
1	Black	1	MM	30-34
2	White	1	MM	47-17
3	Black	2	MM	29-34
4	White	2	MM	49-15
5	Black	3	MM	25-38
6	White	3	MM	37-27
7	Black	4	MM	24-40
8	White	4	CNN	30-34
9	Black	5	CNN	34-29
10	White	5	MM	37-27
11	Black	6	MM	11-53
12	White	6	MM	39-25

Conclusion

- We were still able to run training through our self-supervised, self-play approach
- By generating 1,016 training games via temperature-adjusted exploration and minimax play, we attempted to enhance the models strategy and adaptability to opponents
- Unfortunately, as the performance benchmarking/testing table shows, we were unable to improve the models prediction capabilities.
- Despite the poor results, we gained more experience with PyTorch machine learning and left knowing that nothing beats the adaptive nature of the Minimax algorithm