Better Move Ordering in Combinatorial Games via Learnable Heuristics in a 1D Clobber Solver

Akash Saravanan (asaravan@ualberta.ca)
Debraj Ray (debraj1@ualberta.ca)

UNIVERSITY OF ALBERTA

Outline

- **Introduction**
- Baseline - Assignment 2 Solver
- Deep Learning using CNN
  - Architecture
  - Fundamentals of training
  - Inference Accuracy
  - Results
- Deep Reinforcement Learning
  - Overview
  - Training
  - Results & Evaluation
  - Advantages & Limitations
- Future Work
- Conclusion

Why Move Ordering through Learnable Heuristics?

- Choosing a great move quickly can preclude a lot of searching effort.
- This is especially true if we can find a good move near the top levels of the game tree.
- Heuristics are good for this purpose, but finding an effective heuristic (manually) is challenging.
- Subgame database and endgame database are very helpful, but they don’t scale due to the huge state space of larger boards.
- Machine learning approaches can do a really good job
- Machine learning models are just a few kbs

Two different learning approaches explored for 1D clobber

Two Approaches:

- Deep Learning using a CNN
- Deep Reinforcement Learning

Outline

- Introduction
- **Baseline - Assignment 2 Solver**
- Deep Learning using CNN
  - Architecture
  - Fundamentals of training
  - Inference Accuracy
  - Results
- Deep Reinforcement Learning
  - Overview
  - Training
  - Results & Evaluation
  - Advantages & Limitations
- Future Work
- Conclusion

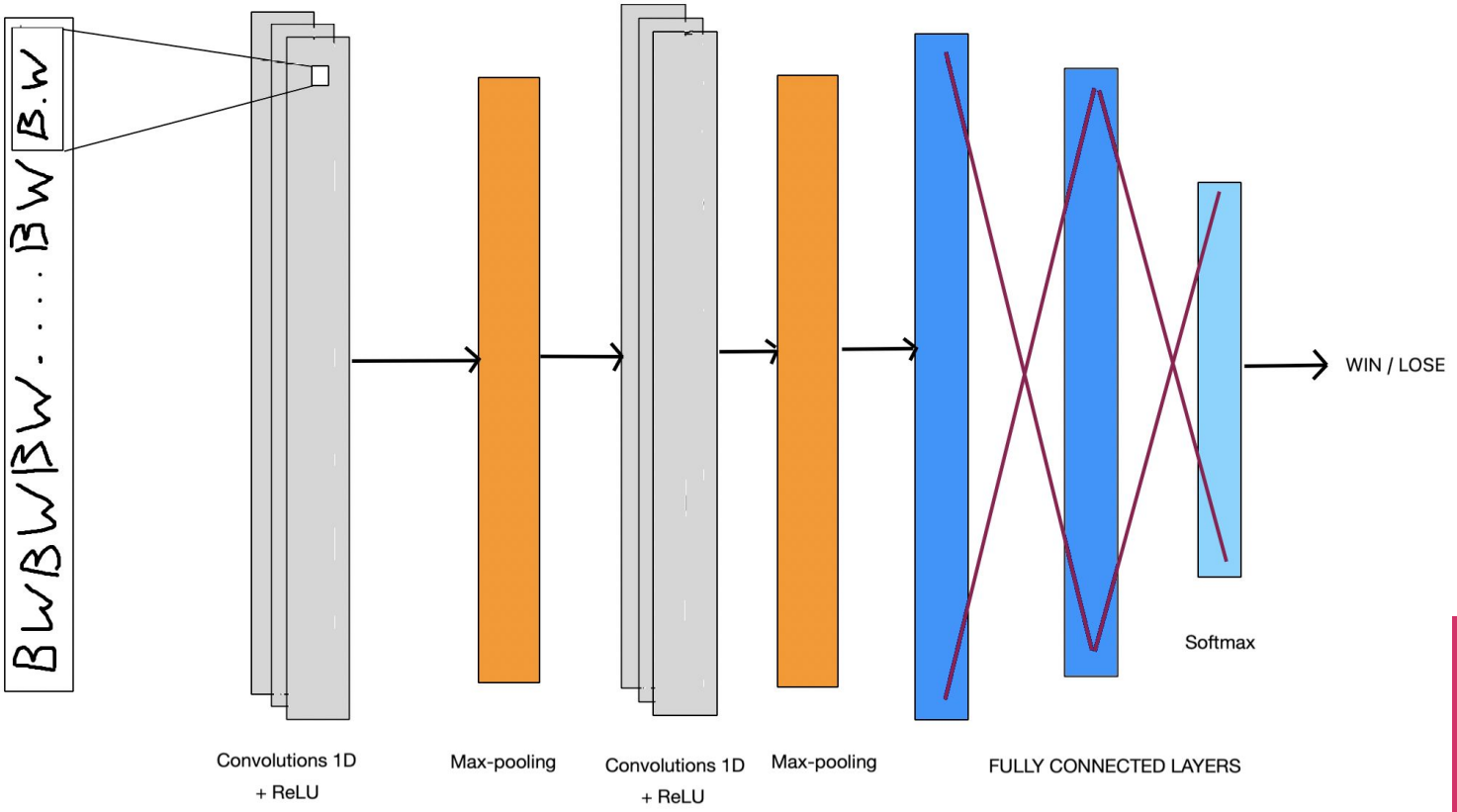
Baseline

- Clobber Solver from Assignment 2
- Uses a basic move ordering
- Plays moves in R games first, L games last and other subgames in the decreasing order of size.
- Experiments showed that such a move ordering is at least better than random move ordering

Outline

- Introduction
- Baseline - Assignment 2 Solver
- **Deep Learning using CNN**
  - Architecture
  - Fundamentals of training
  - Inference Accuracy
  - Results
- Deep Reinforcement Learning
  - Overview
  - Training
  - Results & Evaluation
  - Advantages & Limitations
- Future Work
- Conclusion

CNN - Architecture



Fundamentals of Training

- Two CNNs - black and white - to predict win/lose probability of black and white respectively
- Training on small boards of size less than 17 (with paddings and shiftings)
- Trained on 800,000 synthetic samples
- Target labels generated using A2 clobber solver
- Re-training of trained model on larger boards (with controlled sparsity)
- Re-training with 600,000 synthetic samples.
- Max size of board is fixed at 40.

Inference Accuracy

Board Size	CNN-Black (% accuracy)	CNN-White (% accuracy)
20	85.6	88.6
25	81	83.4
30	80.78	79.11
35	79.66	72.78

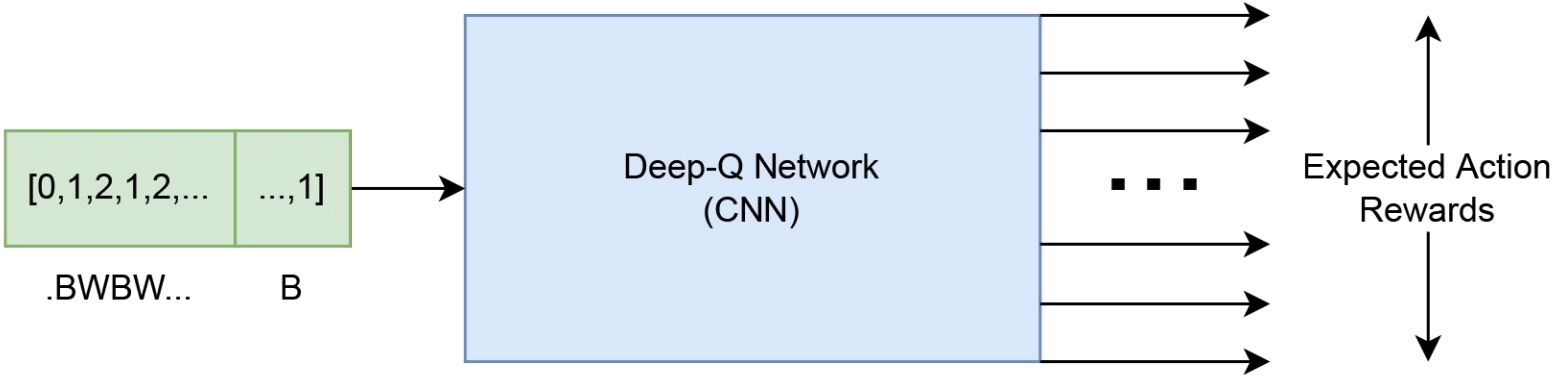
Results of deploying CNN in the 1D clobber solver

Board	Size	First Player	CNN-ordering	Baseline-ordering
$(WB)^{15}$	30	B	B 19-20 14.68 304944	B 9-8 33.15 698817
$(WB)^{15}W$	31	W	B None 82.12 1604209	B None 161.75 3347088
$(WB)^{15}W$	31	B	B 23-22 10.50 218937	B 5-4 73.95 1532934
$(BW)^{16}$	32	W	W 29-30 46.45 916989	W 21-22 84.38 1721429
$(WB)^{16}$	32	W	W 12-13 48.34 954372	W 12-13 117.33 2388654
$(BW)^{17}$	34	W	W 31-32 197.94 3715242	W 21-22 507.67 9820176
$(BW)^{17}$	34	B	B 22-23 229.40 4390859	B 18-17 329.96 6435019
$(WB)^{17}$	34	B	B 23-24 180.76 3336638	B 21-22 514.61 9820176
$(BW)^{18}$	36	B	B 22-23 808.145 10291762	B 18-17 1199.84 22390724

Outline

- Introduction
- Baseline - Assignment 2 Solver
- Deep Learning using CNN
  - Architecture
  - Fundamentals of training
  - Inference Accuracy
  - Results
- **Deep Reinforcement Learning**
  - Overview
  - Training
  - Results & Evaluation
  - Advantages & Limitations
- Future Work
- Conclusion

RL Approach: Overview



- Markov Decision Process:
  - States: Current board position + current player. Example: (BWWBW, B)
  - Actions: Move played. Example: (0->1) [.BWBW]
  - Next States: Board Position + current player *after* opponent move. Example: (.W.BW, B)
  - Reward: +1 if first player win; -1 if second player win;
    -1/(100 * board_size) otherwise

RL Approach: Training

- Model:
  - Double Deep Q-Network (DDQN), soft target updates, experience replay, action mask.[5,6]
  - Policy: Exponentially Decaying Epsilon Greedy Policy.
  - Model: Deep CNN (4 CNN + 2 Linear layers; ReLU + Dropout after each layer).
- Environment:
  - Initial Board: $(BW)^N$, $(WB)^N$, $(BBW)^N$, $(WWB)^N$, $[B/W]^N$, $[B/W/.]^N$
  - Starting Player: B or W.
- Training:
  - Board Size: 40; Train vs random player* for 1,000,000 Episodes (games).
  - Validation: Winrate vs Random Player.
  - Adam Optimizer; Batch Size: 64; Learning Rate: 0.0001;

RL Approach: Evaluation & Results

Board	Player	Size	No Move Ordering		Baseline Move Ordering		CNN Move Ordering		RL Move Ordering	
			Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
$(BW)^{13}$	W	26	25.2852	204774	9.46971	78215	4.41373	32887	**4.38826**	**25444**
$(BBW)^{9}$	B	27	3.67418	18083	1.33008	7844	1.29797	7844	**1.28726**	**5709**
$(WB)^{14}W$	B	29	980.000	5889300	205.293	1270778	122.388	526587	**115.612**	**439652**
$(BW)^{15}B$	W	31	980.000	5963465	281.934	1532934	100.456	537206	**86.1686**	**300876**

RL Approach: Advantages & Limitations

- Advantages:
  - O(1) for inference. In practice, 0.0000965s per call.
  - Excellent performance *when* it works.
  - Size: 178 KB
- Limitations:
  - Doesn't always work well.
    - Sometimes worse than our baseline.
    - This could be resolved by further training.
      - We train on just 1 million boards; there are 3^{40} possible boards for board size 40.
    - The DQN algorithm we used while effective, is not the state of the art. Options exist.

Model	Winrate vs Random
Board Size 15	62.998
Board Size 20	60.513
Board Size 25	51.144
Board Size 30	51.119
Board Size 40	52.200

Outline

- Introduction
- Baseline - Assignment 2 Solver
- Deep Learning using CNN
  - Architecture
  - Fundamentals of training
  - Inference Accuracy
  - Results
- Deep Reinforcement Learning
  - Overview
  - Training
  - Results & Evaluation
  - Advantages & Limitations
- **Future Work**
- Conclusion

Extensions/Future Work

- Expanding to 2D Clobber or Impartial 1D Clobber.
- Solving 1D Clobber for $(BW)^n$ where n>19.
  - $(BW)^{19}$ was confirmed to be a 1st player win. [~1 hour with the CNN][1]
- Deep Learning using a CNN:
  - Combining CNN with MCTS.
- Deep Reinforcement Learning:
  - Stronger opponents: Self-play [In Progress!], Existing players (with bounds on time).
  - Experiment with different model architectures & RL algorithms (AlphaGo uses MCTS+RL).

Outline

- Introduction
- Baseline - Assignment 2 Solver
- Deep Learning using CNN
  - Architecture
  - Fundamentals of training
  - Inference Accuracy
  - Results
- Deep Reinforcement Learning
  - Overview
  - Training
  - Results & Evaluation
  - Advantages & Limitations
- Future Work
- **Conclusion**

Conclusions

- Do our learnable heuristics result in better move ordering?
  - Yes!
- Is it always better?
  - No.
- Can it be improved?
  - Yes!
- Are learnable heuristics a good choice for further work on move ordering?
  - Yes!

References

[1] Albert, Michael & Grossman, J.P. & Nowakowski, Richard & Wolfe, David. (2005) An introduction to Clobber. Integers (Vol 5-2).

[2] Teddy Etoeharnowo. (2017) Neural Networks for Clobber. Bachelor's thesis, Leiden Institute of Advanced Computer Science.

[3] Jeroen Claessen. (2011) Combinatorial Game Theory in Clobber. Master's thesis, Maastricht University.

[4] Griebel, Janis and Uiterwijk, Jos. (2016) Combining Combinatorial Game Theory with an α-β Solver for Clobber: Theory and Experiments. BNCAI.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.A. (2013). Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop.*

[6] Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

[7] Chen, Zhenhua & Wang, Chuhua & Laturia, Parth & Crandall, David & Blanco, Saúl. (2021 How to play Notakto: Can reinforcement learning achieve optimal play on combinatorial games? AAAI - Reinforcement Learning and Games 2021.

[8] Ruiyang Xu and Karl Lieberherr. (2019) Learning Self-Game-Play Agents for Combinatorial Optimization Problems. AAMAS '19 Proceedings. Page 2276–2278.