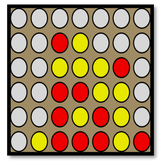
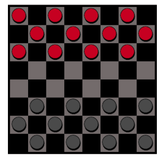
Deep Reinforcement Learning for General Game Playing

Category: Theory and Reinforcement

Mission

Create a reinforcement learning algorithm that generalizes across adversarial games.

Background

- **TD-Gammon**[1] used reinforcement learning to play backgammon, but the approach did not generalize to other adversarial games.
- **DeepMind**'s Atari player[2] generalized across single player games.

We use elements of both approaches in order to generalize across adversarial games.

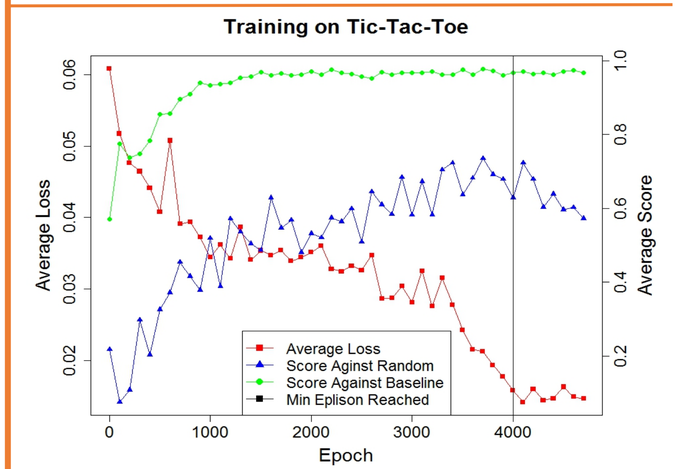
Overview

1. As input we create a **vector representation** of game states and transition functions that encode the game rules.
2. Given the rules of a game, our learning algorithm uses self-play to learn a function **Q(s, a)** approximating utility of action a in state s. Our Q function is a **feed-forward neural network**.
3. We then create an agent for the game using the learned Q-function.

Measuring Effectiveness

We measure effectiveness by taking our agent's average score across 1000 games against two different players:

1. **Random**: an agent that moves randomly.
2. **Baseline**: wins in one move if it can, avoids allowing its opponent to win in one move, random otherwise.

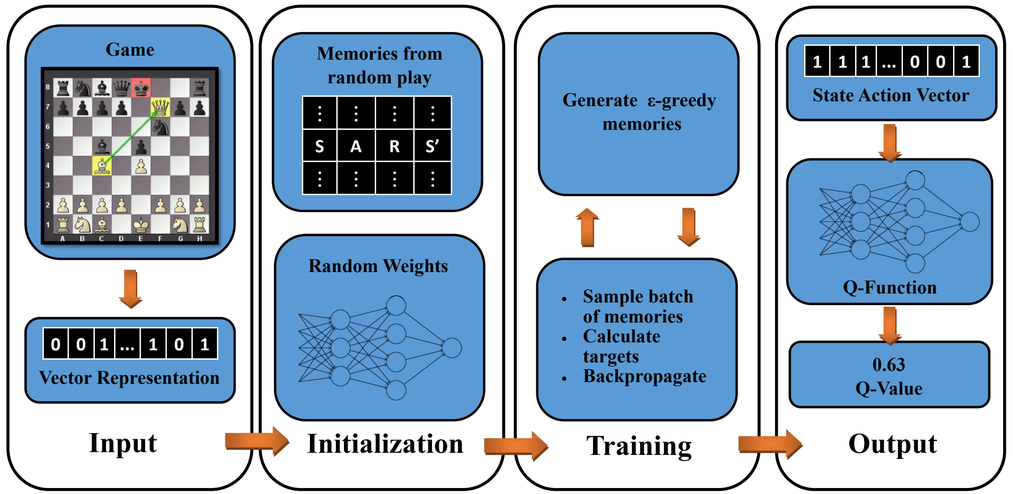


Initial Results

We first trained on Tic-Tac-Toe with a single hidden layer of 100 units. From these results we know:

- **Lower loss correlates with better play.**
- Testing against Baseline provides more information.
- Beating Baseline means that **our model can plan multiple moves in advance**, which confirms that reward signals are being propagated backwards through the game.

Pipeline



Game Representation

A game is defined by:

1. An internal board state representation.
2. Functions that simulate games on board states.
3. A function that turns a board state and an action into a **state-action vector (SAV)**.

The SAV only contains indicator features: the state part contains an indicator for each kind of piece in each position, and the action part is a one-hot encoding.

Bellman Loss

For transition (s, a, r, s') the Bellman Equation says:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

meaning that there is continuity between consecutive Q-values. If we use the right side of the equation as our target, **we can enforce this continuity**:

$$Loss = \left(Q(s,a) - \left(r + \gamma \max_{a'} Q(s',a')\right)\right)^2$$

Modifying DeepMind's Approach

We use two techniques that helped DeepMind generalize across tasks:

1. **Experience Replay**: (s, a, r, s') transitions are stored and slowly overwritten as training games are played. These memories are randomly sampled for backpropagation.
2. **ε-greediness**: In order to force exploration of new states, there is a probability of **ε** that any given move during training is made randomly. Like DeepMind, we slowly reduce the value of **ε** throughout training.

Unlike DeepMind, we are trying to learn a policy for an adversarial game, rather than a single-player environment. We handle this by having s' represent the state reached after both the player and the opponent have moved. **The opponent then ends up representing the environment**. As **ε** decreases and our Q function improves, this "environment" goes from being completely random to being a strong player. Because we are using self-play, this strong player is (secretly) our agent.

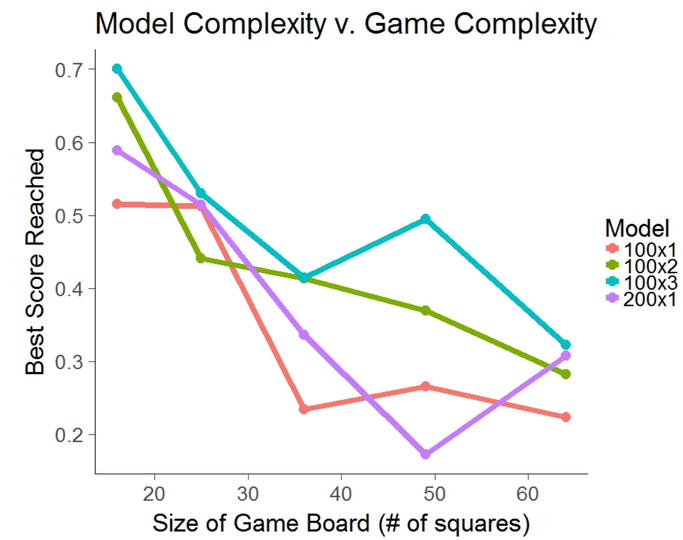
Project by Noah Arthurs & Sawyer Birnbaum

narthurs@stanford.edu sawyerb@stanford.edu

Investigating Model Complexity

We trained models with 1, 2, and 3 hidden layers on square Connect-4 grids ranging from 4x4 to 8x8. We kept track of the best win rate against Baseline for each model during training (graphed below). Our results indicate:

- There is a **roughly linear relationship between the performance of a given model and the size of the game**. With data across a larger number of games, we might be able to learn this relationship.
- More complex models are almost uniformly more effective on a given game than less complex models.
- Since two 100 unit layers generally outperformed one 200 unit layer, it seems that **the depth of the network is more important than the breadth**.
- Presumably, if we continue to ramp up the complexity, we could continue to drive up results on the larger grids.



Conclusion

- **It is effective to learn a policy for an adversarial game by modeling the opponent as a constantly evolving MDP.**
- It seems that we can increase effectiveness simply by increasing model complexity.
- With a large enough memory size, a fast enough rate of memory generation, and a large enough **ε**, **overfitting should be impossible**.
- There is a roughly linear relationship between the effectiveness of a given model and the complexity of the game it is being trained on. **This indicates that our approach scales well.**

Future Work

- We would like to gather more data on games and models of different complexities in order to more precisely **learn the relationship between game complexity, model complexity, and performance.**
- We want to test the limits of our system to find out how complex a game it can learn to play optimally.
- We would like to find out if we can improve performance by varying the values of parameters that to this point we have kept fixed.

References

1. Tesauro, Gerald. "Temporal difference learning and TD-Gammon." Communications of the ACM 38.3 (1995): 58-68.
2. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).