Term Project: Reinforcement learning applied to Othello

GEORGE TUCKER
Othello
What is reinforcement learning?

- After a sequence of actions get a reward
  - Positive or negative

- Temporal credit assignment problem
  - Determine credit for the reward
  - Temporal Difference Methods
    - TD-lambda
    - Q-learning (TD(0))
Contrast to Conventional Strategies

- Most methods use an evaluation function
- Use minimax/alpha-beta search
- Hand-designed feature detectors
  - Evaluation function is a weighted sum

- So why TD learning?
  - Does not need hand coded features
  - Generalization
Temporal Difference Learning

\[
Output = \sum_{k=1}^{H} f\left(\sum_{j=1}^{N} I_{j,k} W^I_j\right) W^O_k
\]

\(N\) is the number of input nodes.

\(H\) is the number of hidden nodes.

\(f()\) is our non-linear function.
Temporal Difference Learning

\[ \Delta W_t = \alpha (Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k \]

- \( t \) is time (in our case move number).
- \( T \) is the final time (total number of moves).
- \( Y_t \) is the evaluation of the board at time \( t \) when \( t \neq T \).
- \( Y_T \) is the true reward (i.e. win, loss or draw).
- \( \alpha \) is the learning rate.
- \( \nabla_w Y_k \) is the partial derivative of the weights with respect to the output.
- \( d_T \) is the temporal difference.
Key Observation

- If we let

\[ e_{ijk}^t = \sum_{n=1}^{t} \lambda^{t-n} \frac{\partial P_k^n}{\partial w_{ij}^n}. \]

at time step \( t \). Then at time step \( t+1 \),

\[ e_{ijk}^{t+1} = \lambda e_{ijk}^t + \frac{\partial P_k^{t+1}}{\partial w_{ij}^{t+1}}. \]
If we let $\lambda = 0$, then, we get

$$\Delta W_t = \alpha(Y_{t+1} - Y_t) \nabla_w Y_k$$

- Widrow-Hoff rule
- Makes $Y_t$ closer to $Y_{t+1}$
Disadvantage

- Requires lots of training
- Self-play
  - Short-term pathologies
  - Randomization
• **Board:** 64 element vector
  - +1 = black
  - 0 = empty
  - -1 = white
  - Corresponds to human representation

• **Network**
  - 64 inputs
  - 30 hidden nodes – Sigmoid activation
  - Single output
    - Goal: predict final game score
Setup

- TD lambda learning
  - Lambda = 0.3
  - Learning rate = 0.005
  - Reward is endgame score

- Move selection
  - Evaluate every legal 1 ply move
  - Choose randomly with exponential weight
Player Handling

- Two Neural Networks
- Board inversion
  - On white’s move, invert board and score
  - Faster and superior learning
Training Data

- **Recall:**
  - Random play
  - Fixed opponent
  - Database play
  - Self-play

- **I focused on:**
  - Database play
  - Self-play
Opponent

- Java Othello
  - [www.luthman.nu/Othello/Othello.html](http://www.luthman.nu/Othello/Othello.html)
  - Variable levels corresponding to ply depth
- Used as benchmark
- Trained against
Database Training

- **Logistello database**
  - 120,000 games

- **Fast**
  - less than 30 minutes to train on the full set

- **Wins 10% games against a 1 ply opponent**
Self-play

- **Extremely slow improvement**
  - Even after nearly 2,000,000 iterations almost no improvement
- **Only wins 1% of games against 1 ply opponent**
Two Ply Opponent

- Opponent looks ahead one ply and chooses the best move
  - Much slower by a factor of 6 or more
Website

- For source code and reference material
  - www.cs.hmc.edu/~gtucker/othello.html
Conclusions

- Board inversion should definitely be used
- Initially, at least self-play is poor
- Database play significantly improves network
- Asymmetric self-play is far superior to standard self-play
- Playing a fixed opponent may be best

**Future Work**
- Add in additional feature detectors
- Investigate more advanced depth play