Monte-Carlo Simulation Balancing

David Silver, University of Alberta
Gerry Tesauro, IBM Research
Monte-Carlo Search
Monte-Carlo in Two-Player Games

- Monte-Carlo search has achieved master level:
  - Backgammon (Tesauro & Galperin, 1997)
  - Poker (Billings et al., 1999)
  - Scrabble (Sheppard, 2002)
  - 9x9 Go (Gelly & Silver, 2007)
  - 19x19 Go (Teytaud et al. 2009)
Motivation

- Performance of Monte-Carlo depends on *simulation policy*
- In deterministic domains, need stochastic rollouts
- A strong policy does not make a good simulation policy
- Can we automatically learn a good simulation policy?
Example: Computer Go

- Simulation policy has dramatic effect on performance
  - Handcrafted trial and error (e.g. MoGo, Fuego)
  - Hill-climbing methods (e.g. Mango)
  - Supervised learning (e.g. CrazyStone, Zen)
  - Reinforcement learning (e.g. RLGO)
- Can we do better by optimising the right objective?
Introduction to Monte-Carlo Search

- Game outcome $z$
  - Black wins $z=1$
  - White wins $z=0$
- Value of position $s$
  - $V(s) = \mathbb{E}[z | s] \approx \frac{1}{N} \sum_{i=1}^{N} z_i$
Monte-Carlo Simulation

- Simulate many games from current position $s$
- Rollout each game using simulation policy (e.g., random)
- Evaluate position by mean outcome of simulations
Monte-Carlo Simulation

Current position $s$

Simulation

Outcomes

1 1 0 0
Simple Monte-Carlo Search

- Evaluate all legal moves by Monte-Carlo simulation
- Select move with highest evaluation
Monte-Carlo Tree Search

- Monte-Carlo simulation from the current position $s$
- Builds a search tree containing all visited positions
- Each position is evaluated by Monte-Carlo
- With suitable exploration, converges on minimax
Simulation Balancing
Objective

- Stochastic simulation policy $\pi_\theta(s,a)$
- Parameterised by weights $\theta$
- Simulation policy has a bias w.r.t. minimax value $V^*(s)$
  \[
  b(s) = V^*(s) - E_{\pi_\theta}[z|s]
  \]
- Find the parameters $\theta^*$ that minimise mean squared bias
- Approximate $V^*(s)$ by deep Monte-Carlo tree search
Strength and Balance

- Consider error $V^*(s_{t+1}) - V^*(s_t)$ w.r.t. to minimax $V^*(s)$

- A strong policy will have small errors

- A balanced policy will have small expected total error:
  - Allow errors by one player, if they are cancelled out by errors by other player
  - Balance over whole game (presentation), or over two steps (see paper)
Strong, Unbalanced Rollouts
Weak, Balanced Rollouts

The graph illustrates the comparison between the Minimax value and the Monte-Carlo value over time steps. The simulations are represented by the green dashed line, the mean by the black line, and the Monte-Carlo value by the red line. The data shows fluctuations over time, indicating the dynamic nature of the values as time progresses.
Policy Gradient Simulation Balancing

- Minimise MSE between minimax value and mean outcome of rollouts

\[ J(\theta) = \mathbb{E}[(V^*(s) - \mathbb{E}_\pi[z|s])^2] \]

- Use gradient descent

\[
\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta (V^*(s) - \mathbb{E}_\pi[z|s])^2]
\]
\[
= -2\mathbb{E}[(V^*(s) - \mathbb{E}_\pi[z|s])\nabla_\theta \mathbb{E}_\pi[z|s]]
\]
Stochastic Gradient Descent

• For each training position \( s \), update policy parameters

\[
\Delta \theta = \alpha (V^*(s) - E_{\pi_\theta} [z|s])
\]

- **Bias**
- **Policy gradient**

• The *bias* term indicates whether black needs to win more or less

• The *policy gradient* term indicates how black can be made to win more
Algorithm

- For each training example $s$ with minimax value $V^*(s)$
  1. Estimate the bias by Monte-Carlo simulation
     \[ b(s) = V^*(s) - \frac{1}{M} \sum z_i \]
  2. Estimate the policy gradient by REINFORCE
     \[ g(s) = \sum \frac{z_i}{N} \sum \nabla_\theta \log \pi_\theta(s_t, a_t) \]
  3. Update weights by product
     \[ \Delta \theta = \alpha b(s) g(s) \]
Softmax policy

- Parameterise policy using a vector of features $\phi(s,a)$ of position $s$ and move $a$
- Compute linear preferences $\phi(s,a)^T \theta$
- Softmax distribution favours high preferences

$$\pi(s, a) = \frac{e^{\phi(s,a)^T \theta}}{\sum_b e^{\phi(s,b)^T \theta}}$$
Results in Computer Go
Comparison of Methods

- Methods for optimising strength
  - 1. Apprenticeship learning (maximises likelihood)
  - 2. Policy gradient reinforcement learning

- Methods for optimising balance
  - 3. Policy gradient simulation balancing
  - 4. Two step simulation balancing (see paper)
Experimental Setup

- Small board Go (5x5 and 6x6)
- Softmax policy parameterised by *local shape features*
  - ~2000 features, ~100 unique weights
- Training values $V^*(s)$ generated by deep Monte-Carlo tree search (*Fuego*)
- Training positions drawn from random vs. random
- 10,000 rollouts from each position
Local Shape Features

- Binary features matching a local configuration of stones
- All possible locations and configurations from 1x1 to 2x2
- Weights shared between symmetric shapes
Weight Evolution During Training

- Apprenticeship Learning
- Policy Gradient Simulation Balancing
- Policy Gradient Reinforcement Learning
- Two Step Simulation Balancing
Mean-Squared Error (5x5 Go)

Uniform Random
Apprenticeship Learning
Policy Gradient Reinforcement Learning
Policy Gradient Simulation Balancing
Two Step Simulation Balancing
Mean-Squared Error (6x6 Go)

- Uniform Random
- Apprenticeship Learning
- Policy Gradient Reinforcement Learning
- Policy Gradient Simulation Balancing
- Two Step Simulation Balancing

MSE vs. Training games
## Tournament Results

### Simple Monte-Carlo with learnt simulation policy

<table>
<thead>
<tr>
<th></th>
<th>5x5 Direct</th>
<th>5x5 MC</th>
<th>6x6 Direct</th>
<th>6x6 MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Random</td>
<td>0</td>
<td>1031</td>
<td>0</td>
<td>970</td>
</tr>
<tr>
<td>Apprenticeship Learning</td>
<td>671</td>
<td>1107</td>
<td>569</td>
<td>1047</td>
</tr>
<tr>
<td>Policy Gradient RL</td>
<td>947</td>
<td>1159</td>
<td>850</td>
<td>1023</td>
</tr>
<tr>
<td>Full Simulation Balancing</td>
<td>719</td>
<td>1367</td>
<td>658</td>
<td>1301</td>
</tr>
<tr>
<td>Two Step Simulation Balancing</td>
<td>720</td>
<td>1357</td>
<td>444</td>
<td>1109</td>
</tr>
<tr>
<td>GnuGo (level 10)</td>
<td>1376</td>
<td>N/A</td>
<td>1534</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Conclusions

- Can learn a good simulation policy for Monte-Carlo
- Can balance stochasticity of rollouts
- Exploits expert values more effectively than supervised learning
- Optimises a more relevant objective function than reinforcement learning
- Outperforms prior methods in small board Go
Questions?