Applications of Machine Learning to the Game of Go

David Stern

Applied Games Group
Microsoft Research Cambridge

(working with Thore Graepel, Ralf Herbrich, David MacKay)
Contents

• The Game of Go
• Uncertainty in Go
• Move Prediction
• Territory Prediction
• Monte Carlo Go
The Game of Go

- Started about 4000 years ago in ancient China.
- About 60 million players worldwide.
- 2 Players: Black and White.
- Board: 19×19 grid.
- Rules:
  - Turn: One stone placed on vertex.
  - Capture.
- Aim: Gather territory by surrounding it.
- A stone is captured by an opponent surrounding it.
- A chain is captured all together.
One eye = death
Two eyes = life
Computer Go

• 5th November 1997: Gary Kasparov beaten by Deep Blue.

• Best Go programs cannot beat amateurs.
• Go recognised as grand challenge for AI.
Computer Go

• Minimax search defeated.

• High **Branching Factor**.
  – Go: ~200
  – Chess: ~35

• Complex **Position Evaluation**.
  – Stone’s value derived from configuration of surrounding stones.
Use of Knowledge in Computer Go

- Trade off between search and knowledge.
- Most current Go programs use hand-coded knowledge.
  1. Slow knowledge acquisition.
  2. Tuning hand-crafted heuristics difficult.
- Previous approaches to automated knowledge acquisition:
  - Neural Networks (Erik van der Werf et al., 2002).
  - Exact pattern matching (Frank de Groot, 2005), (Bouzy, 2002)
Uncertainty in Go

- Go is a game of perfect information.
- Complexity of game tree + limited computer speed → uncertainty.
- 味 ‘aji’ = ‘taste’.
- Represent uncertainty using probabilities.
Move Prediction
Learning from Expert Game Records
Pattern Matching for Move Prediction

- Move associated with a set of *patterns*.
  - Exact arrangement of stones.
  - Centred on proposed move.
- Sequence of nested templates.
Patterns
Patterns
Patterns
Patterns
Patterns

• 13 Pattern Sizes
  – Smallest is vertex only.
  – Biggest is full board.
Pattern Matching

- Pattern information stored in hash table.
  - Constant time access.
  - No need to store patterns explicitly.
- Need rapid incremental hash function.
  - Commutative.
  - Reversible.
- 64 bit random numbers for each template vertex:
  One for each of \{black, white, empty, off\}.
- Combine with XOR (Zobrist, 1970).

\[
\text{Black at } (-1,2) = 128379874091837 \\
\text{Empty at } (1,1) = 876542534789756
\]
Pattern Hash Key

- Pattern information stored in hash table.
  - Access in constant time.
  - No need to store patterns explicitly.
- Need rapid incremental hash function.
  - Commutative.
  - Reversible.
- 64 bit random numbers for each template vertex:
  One for each of \{black, white, empty, off\}.
- Combine with XOR (Zobrist, 1970).
- Min of transformed patterns gives invariance.

MIN

Transformation Invariant Key
Harvesting

- Automatically Harvest from Game Records.
- 180,000 games × 250 moves × 13 pattern sizes...
  
  ...gives 600 million potential patterns.
- Need to limit number stored.
  - Space.
  - Generalisation.
- Keep all patterns played more than $n$ times.
Relative Frequencies of Pattern Sizes

Smaller patterns matched later in game.

Big patterns matched at beginning of game.

Phase of the game

Pattern size

Relative frequency
Training

• First Phase: Harvest
• Second Phase: Training
• Use same games for both.
• Represent move by largest pattern only.
Bayesian Ranking Model

- Pattern value: \( u_1 \sim \mathcal{N}(u_1; \mu_1, \sigma_1^2) \)
- Latent urgency: \( x_1 \sim \mathcal{N}(x_1; u_1, \beta^2) \)
Bayesian Ranking Model

\[ \mathcal{N}(u_1; \mu_1, \sigma_1^2) \quad \mathcal{N}(x_1; u_1, \beta^2) \]
Bayesian Ranking Model

Training Example:
- Chosen move (pattern).
- Set of moves (patterns) not chosen.
Bayesian Ranking Model

\[ p(\mathbf{u}, \mathbf{x} | \text{move, position}) \]
Bayesian Ranking Model

\[ p(u|\text{move, position}) = \int p(u, x|\text{move, position}) dx \]
Online Learning from Expert Games

• Training Example:
  – Chosen move (pattern).
  – Set of moves (patterns) not chosen.

• Posterior:

\[ p(u|\text{move}, \text{position}) = \int p(u, x|\text{move}, \text{position}) \, dx \]

• Approximate (Gaussian) posterior determined by Gaussian message passing.

• Online Learning (Assumed Density Filtering):
  – After each training example we have new \( \mu_i \) and \( \sigma_i \) for each pattern.
  – Replace values and go to next position.
Message Passing

\[ m_{f \rightarrow v}(v) = \int f(v) \prod_{v_j \in \text{neigh}(f) \setminus v} m_{v_j \rightarrow f}(v_j) dv \]

\[ m_{v \rightarrow f}(v) = \prod_{f_j \in \text{neigh}(v) \setminus f} m_{f_j \rightarrow v}(v) \]
Marginal Calculation

\[ p(v) = \prod_{f_k \in \text{neigh}(v)} m_{f_k \rightarrow v}(v) \]
Gaussian Message Passing

- All messages Gaussian!
- Most factors Gaussian.
- Messages from ‘ordering’ factors are approximated:
  - Expectation propagation.
- True Marginal Distribution:
  \[ p(v_i) = m_{f_k \rightarrow v_i}(v_i) \cdot m_{v_i \rightarrow f_k}(v_i) \]

- Approximation:
  \[ q(v_i) = \tilde{m}_{f_k \rightarrow v_i}(v_i) \cdot m_{v_i \rightarrow f_k}(v_i) \]

- Moment Match:
  \[ \tilde{m}_{f_k \rightarrow v_i}(v_i) = \operatorname{MM} \left[ \frac{m_{f_k \rightarrow v_i}(v_i) \cdot m_{v_i \rightarrow f_k}(v_i)}{m_{v_i \rightarrow f_k}(v_i)} \right] \]
Gaussian Message Passing

\[ N(u_1; \mu_1, \sigma_1^2) \quad N(x_1; u_1, \beta^2) \]

\[ N(u_2; \mu_2, \sigma_2^2) \quad N(x_2; u_2, \beta^2) \]

\[ \vdots \]

\[ N(u_n; \mu_n, \sigma_n^2) \quad N(x_n; u_n, \beta^2) \]

\[ \mathbb{I}(x_1 > x_2) \]

\[ \mathbb{I}(x_1 > x_n) \]
Gaussian Message Passing
– EP Approximation

\[ \mathcal{N}(u_1; \mu_1, \sigma_1^2) \quad \mathcal{N}(x_1; u_1, \beta^2) \]
\[ \mathcal{N}(u_2; \mu_2, \sigma_2^2) \quad \mathcal{N}(x_2; u_2, \beta^2) \]
\[ \vdots \quad \vdots \]
\[ \mathcal{N}(u_n; \mu_n, \sigma_n^2) \quad \mathcal{N}(x_n; u_n, \beta^2) \]

\[ \mathbb{I}(x_1 > x_2) \]
\[ \mathbb{I}(x_1 > x_n) \]
Gaussian Message Passing
- Posterior Calculation

\[ p(u_i | \text{move, position}) = \prod_{f \in \text{neigh}(u_i)} m_{f \rightarrow u_i}(u_i) \]
Move Prediction Performance

100 Moves / second

- Ranking Model
- Van der Werf et al. (2002)
Rank Error vs Game Phase

The graph illustrates the rank error at different phases of the game. The y-axis represents the rank error on a logarithmic scale, while the x-axis shows the phase of the game. The data is presented using box plots for each phase, indicating the distribution of rank errors.
Rank Error vs Pattern Size
Hierarchical Gaussian Model of Move Values

- Use all patterns that match – not just biggest.
  - Evidence from larger pattern should dominate small pattern at same location.
  - However – big patterns seen less frequently.
- Hierarchical Model of move values.
Pattern Hierarchy
Hierarchical Gaussian Model

\[
p(x_{10} | x_{00}) = \mathcal{N}(x_{10}, x_{00}, \beta_0^2)
\]

\[
p(x_{00}) = \mathcal{N}(x_{00}; \mu_0, \sigma_0^2)
\]
Move Prediction Performance

The diagram illustrates the cumulative density function for expert move ranks, comparing two models:

- **Max Pattern** (red line)
- **Hierarchical Model** (black line)

The x-axis represents the expert move rank, ranging from 0 to 30. The y-axis represents the cumulative density function, ranging from 0 to 1. The graph shows that the hierarchical model performs slightly better than the max pattern model across all ranks.
Predictive Probability

![Graph showing the comparison between max pattern and hierarchical model](image)

- Predictive Probability
- $\max$ pattern
- Hierarchical model

Phase of the game

1 2 3 4 5 6 7 8 9 10 11

Mean ln(predictive probability)
Territory Prediction
Territory

1. Empty intersections surrounded.
2. The stones themselves (Chinese method)
Predicting Territory

Go Position

 Territory Hypothesis 1
Predicting Territory

Territory Hypothesis 2
Predicting Territory

Territory Hypothesis 3
Predicting Territory

- Board Position: $c \in \{\text{black, white, empty}\}^N$
- Territory Outcome: $s \in \{+1, -1\}^N$
- Model Distribution: $P(s|c)$
- $E(\text{Black Score}) = \sum_i \langle s_i \rangle P(s|c)$
- Boltzmann Machine (CRF):
 Territory Prediction Model

\[ P(s|c) = \frac{1}{Z(c, \theta)} \exp \left( \sum_{(i,j)} E_{(i,j)} \right) \]

Boltzmann Machine (Ising model)

\[ E_{(i,j)}(s_i, s_j, c_i, c_j) = h(c_i)s_i + h(c_j)s_j + w(c_i, c_j)s_is_j \]

- **Biases**
  - e.g. \( h(\text{black}) \)

- **Couplings**
  - e.g. \( w(\text{black, black}) \)
Game Records

• 1,000,000 expert games.
  – Some labelled with final territory outcome.
• Training Data is pair of
  game position, $c_i$, + territory outcome, $s_i$.
• Maximise Log-Likelihood:

\[
\sum_i \ln P(s_i | c_i, w)
\]

• Inference by Swendsen-Wang sampling
1000,000 Expert Games...

So Yokoku [6d] (black) –VS– (white) [9d] Kato Masao
Final Position
Final Position + Territory
Final Territory Outcome
Position + Final Territory
Train CRF by Maximum Likelihood

\[ s | c \]
Hypothesis: These stones are not captured

This side is owned by black

Hypothesis: This area is owned by white

Hypothesis: These stones are captured
Position + Boltzmann Sample
(Generated by Swendsen-Wang)

Hypothesis:
These stones are captured

black controls this region
Sample with Swendsen-Wang Clusters Shown

Bonds link regions of common fate
Boltzmann Machine – Expectation over Swendsen-Wang Samples

No squares indicates uncertainty

Size of squares indicates degree of certainty

These stones are probably going to be captured
Position + Boltzmann Sample
(Generated by Swendsen-Wang)

Illegal!
Monte Carlo Go
(work in progress)
Monte Carlo Go

Go Position

Boltzmann

Territory Hypothesis
Monte Carlo Go

Go Position

Boltzmann

Territory Hypothesis

MC

Territory Hypothesis
Monte Carlo Go

Go Position

Territory Hypothesis

‘Rollouts’
Monte Carlo Go

• ‘Rollout’ or ‘Playout’
  – Complete game from current position to end.
  – Not fill in own eyes.
  – Score at final position easily calculated.
• 1 sample = 1 rollout
• Brugmann’s Monte Carlo Go (MC)
  – For each available move m sample s rollouts.
  – At each position, moves selected uniformly at random.

  – Rollout Value: \( X_{m,i} \in \{1, 0\} \) (win or loss)
  – Move Value: \( \overline{X}_m = \frac{1}{s} \sum_i X_{m,i} \)
Adaptive Monte Carlo Planning

- Update values of all positions in rollout.
  - Store value (distribution) for each node.
  - Store tree in memory.
- Bootstrap policy.
  - UCT.
  - Strong Play on small boards
    E.g. ‘MoGo’ (Silvain Gelly)
Adaptive Monte Carlo Go
Adaptive Monte Carlo Go
Adaptive Monte Carlo Go
Adaptive Monte Carlo Go

This node
Seen: 3 times
Win: 2/3 times
Bayesian Model For Policy

Prior: \[ p(x_0) = \mathcal{N}(x_0; \mu, \sigma^2) \]

Result of Rollout = \text{WIN (TRUE | FALSE)}

\[ p(y_t | x_t) = \mathcal{N}(y_t; x_t, \gamma^2) \]

\[ p(w_i | y_i) = \mathbb{I}(\{(y_i > 0) \land w_i\}) + \mathbb{I}(\{(y_i < 0) \land \neg w_i\}) \]
‘Bayesian’ Adaptive MC

\[ p(x_t|x_{t-1}) = \mathcal{N}(x_t; x_{t-1}, \tau^2) \]
‘Bayesian’ Adaptive MC (BMC)

• Policy:
  – Sample from the distribution for each available move.
  – Pick best.

• Exploitation vs Exploration
  – Automatically adapted.
Exploitation vs Exploration

![Graph showing the trade-off between exploitation and exploration.](image-url)
Exploitation vs. Exploration
P-Game Trees

- Moves have numerical values
  - MAX moves drawn uniformly from [0,1]
  - MIN moves drawn uniformly from [-1,0]
- Value of leaf is sum of moves from root.
- If leaf value > 0 then win for MAX.
- If leaf value < 0 then loss for MAX.
- Assign win/loss to all nodes via Minimax.
- Qualitatively more like real Go game tree.
- Can simulate the addition of domain knowledge.
Monte Carlo Planning On P-Game Trees (B=2, D=20)
MC Planning on P-Game Trees (B=7, D=7)
Conclusions

• Areas addressed:
  – Move Prediction
  – Territory Prediction
  – Monte Carlo Go

• Probabilities good for modelling uncertainty in Go.
• Go is a good test bed for machine learning.
  – Wide range of sub-tasks.
  – Complex game yet simple rules.
  – Loads of training data.
  – Humans play the game well.