Monte-Carlo Game Tree Search: Advanced Techniques

Tsan-sheng Hsu

徐讚昇

tshsu@iis.sinica.edu.tw

`http://www.iis.sinica.edu.tw/~tshsu`

Abstract

- **Adding new ideas to the pure Monte-Carlo approach for computer Go.**
    - On-line knowledge: domain independent techniques
        - ▷ *Progressive pruning*
        - ▷ *All moves as first and RAVE heuristic*
        - ▷ *Node expansion policy*
        - ▷ *Temperature*
        - ▷ *Depth-i tree search*
    - Off-line domain knowledge: domain dependent techniques
        - ▷ *Node expansion*
        - ▷ *Better simulation policy*
        - ▷ *Better position evaluation*

- **Conclusion:**
    - Combining the power of statistical tools and machine learning, the Monte-Carlo approach reaches a new high for computer Go.

Domain independent refinements

- **Main considerations**
  - **Avoid doing un-needed computations**
  - **Increase the speed of convergence**
  - **Avoid early mis-judgement**
  - **Avoid extreme bad cases**
- **Refinements came from on-line knowledge.**
  - **Progressive pruning.**
    - ▷ *Cut hopeless nodes early.*
  - **All moves at first and RAVE.**
    - ▷ *Increase the speed of convergence.*
  - **Node expansion policy.**
    - ▷ *Grow only nodes with a potential.*
  - **Temperature.**
    - ▷ *Introduce randomness.*
  - **Depth-i enhancement.**
    - ▷ *With regard the initial phase, the one on obtaining an initial game tree, exhaustively enumerate all possibilities instead of using only the root.*

Progressive pruning (1/5)

- **Each position has a mean value μ and a standard deviation σ after performing some simulations.**
  - Left expected outcome $\mu_l = \mu - r_d * \sigma$.
  - Right expected outcome $\mu_r = \mu + r_d * \sigma$.
  - The value r_d is a constant fixed up by practical experiments.
- **Let p_1 and p_2 be two children of a position p.**
- **A move p_1 is *statistically inferior* to another move p_2 if $p_1.\mu_r < p_2.\mu_l$, and $p_1.\sigma < \sigma_e$ and $p_2.\sigma < \sigma_e$.**
  - The value σ_e is called *standard deviation for equality*.
  - Its value is determined by experiments.
- **Two moves p_1 and p_2 are *statistically equal* if $p_1.\sigma < \sigma_e$, $p_2.\sigma < \sigma_e$ and no move is statistically inferior to the other.**
- **Remarks:**
  - Assume each trial is an independent Bernoulli trial and hence the distribution is normal.
  - We only compare nodes that are of the same parent.
  - We usually compare their raw scores not their UCB values.
  - If you use UCB scores, then the mean and standard deviation of a move are those calculated only from its un-pruned children.

Progressive pruning (2/5)

- **After a minimal number of random games, say 100 per move, a move is pruned as soon as it is statistically inferior to another.**
  - **For a pruned move:**
    - ▷ *Not considered as a legal move.*
    - ▷ *No need to maintain its UCB information.*
  - **This process is stopped when**
    - ▷ *this is the only one move left for its parent, or*
    - ▷ *the moves left are statistically equal, or*
    - ▷ *a maximal threshold, say 10,000 multiplied by the number of legal moves, of iterations is reached.*

- **Two different pruning rules.**
  - **Hard: a pruned move cannot be a candidate later on.**
  - **Soft: a move pruned at a given time can be a candidate later on if its value is no longer statistically inferior to a currently active move.**
    - ▷ *The score of an active move may be decreased when more simulations are performed.*
    - ▷ *Periodically check whether to reactive it.*

Progressive pruning (3/5)

- **Experimental setup:**
  - **9 by 9 Go.**
  - **Difference of stones plus eyes after Komi is applied.**
  - **The experiment is terminated if any one of the followings is true.**
    - ▷ *There is only move left for the root.*
    - ▷ *All moves left for the root are statistically equal.*
    - ▷ *A given number of simulations are performed.*

Progressive pruning (4/5)

- **Selection of r_d.**
  - **The greater r_d is,**
    - ▷ *the less pruned the moves are;*
    - ▷ *the better the algorithm performs;*
    - ▷ *the slower the play is.*

  - **Results [Bouzy et al'04]:**

r_d	1	2	4	8
score	0	+ 5.6	+ 7.3	+9.0
time	10'	35'	90'	150'

- **Selection of σ_e.**
  - **The smaller σ_e is,**
    - ▷ *the fewer equalities there are;*
    - ▷ *the better the algorithm performs;*
    - ▷ *the slower the play is.*

  - **Results [Bouzy et al'04]:**

σ_e	0.2	0.5	1
score	0	-0.7	-6.7
time	10'	9'	7'

- **Conclusions:**
  - r_d plays an important role in the move pruning process.
  - σ_e is less sensitive.
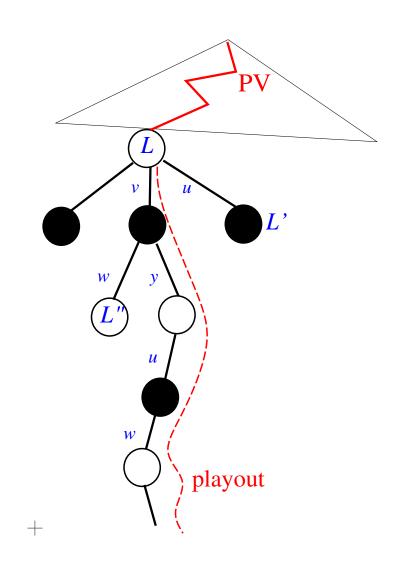
Progressive pruning (5/5)

- **Comments:**
  - **It makes little sense to compare nodes that are of different depths or belong to different players.**
  - **Another trick that may need consideration is progressive widening or progressive un-pruning.**
    - ▷ *A node is effective if enough simulations are done on it and its values are good.*
  - **Note that we can set a threshold on whether to expand or grow the end of the PV path.**
    - ▷ *This threshold can be enough simulations are done and/or the score is good enough.*
    - ▷ *Use this threshold to control the way the underline tree is expanded.*
    - ▷ *If this threshold is high, then it will not expand any node and looks like the original version.*
    - ▷ *If this threshold is low, then we may make not enough simulations for each node in the underline tree.*

All-moves-as-first heuristic (AMAF)

- **How to perform statistics for a completed random game?**
  - **Basic idea: its score is used for the first move of the game only.**
  - **All-moves-as-first AMAF: its score is used for all moves played in the game as if they were the first to be played.**
- **AMAF Updating rules:**
  - **If a playout S, starting from the position following PV towards the best leaf and then appending a simulation run, passes through a move v with a sibling move u, then**
    - ▷ *the counters at the node v leads to is updated;*
    - ▷ *the counters at the node u leads to is also updated if S later contains the move u.*
  - **Note, we apply this update rule for all nodes in S regardless nodes made by the player that is different from the root player.**

Illustration: AMAF

- **Assume a playout is simulated from the position L with the sequence of v, y, u, w, $\cdots$.**
- **The statistics of nodes along this path are updated.**
- **The statistics of node L' that is a child of L, and node L'' are also updated.**
- **In this example, 3 playouts are recorded though only one is performed.**

AMAF: Pro's and Con's

- **Advantage:**
  - **All-moves-as-first helps speeding up the convergence of the simulations.**
- **Drawbacks:**
  - **The evaluation of a move from a random game in which it was played at a late stage is less reliable than when it is played at an early stage.**
  - **Recapturing.**
    - ▷ *Order of moves is important for certain games.*
    - ▷ *Modification: if several moves are played at the same place because of captures, modify the statistics only for the player who played first.*
  - **Some move is good only for one player.**
    - ▷ *It does not evaluate the value of an intersection for the player to move, but rather the difference between the values of the intersections when it is played by one player or the other.*

AMAF: results

- **Results [Bouzy et al'04]:**
  - Relative scores between different heuristics.

AMAF	basic idea	PP
0	+13.7	+ 4.0

    - ▷ *Basic idea is very slow: 2 hours vs 5 minutes.*

  - Number of random games N: relative scores with different values of N using AMAF.

N	1000	10000	100000
scores	-12.7	0	+3.2

    - ▷ *Using the value of 10000 is better.*

- **Comments:**
  - The statistical natural is something very similar to the history heuristic as used in alpha-beta based searching.

AMAF refinement − RAVE

- **Definitions:**
  - **Let $v_1(p)$ be the score of a position p without using AMAF.**
  - **Let $v_2(p)$ be the score of a position p with AMAF.**
- **Observations:**
  - **$v_1(p)$ is good when sufficient number of trials are performed starting with p.**
  - **$v_2(p)$ is a good guess for the true score of the position p when**
    - ▷ *it is approaching the end of a game;*
    - ▷ *when too few trials are perforped starting with m such as when the node for p is first expanded.*

- **Rapid Action Value Estimate (RAVE)**
  - **Let revised score $v_3(p) = \alpha \cdot v_1(p) + (1 - \alpha) \cdot v_2(p)$ with a properly chosen value of α.**
  - **Other formulas for mixing the two scores exist.**
  - **Can dynamically change α as the game goes.**
    - ▷ *For example: $\alpha = \min\{1, N_p/10000\}$, where N_p is the number of play-outs done on p.*
    - ▷ *This means when N_p reaches 10000, then no RAVE is used.*

RAVE

- $v_3(p) = \alpha \cdot v_1(p) + (1 - \alpha) \cdot v_2(p)$
  - **When setting $\alpha = 0$, it is pure AMAF.**
  - **When setting $\alpha = 1$, it uses no AMAF.**
- **Some other forms of formula for using the RAVE values are know.**
- **Silver in his 2009 Ph.D. thesis [Silver'09].**
  - **Let $\beta = 1 - \alpha$.**
  - **Let $\tilde{N}_p = N_p + N'_p$ where N_p is the number of simulations done at the position p and N'_p is the number of simulations from AMAF at p.**
  - **$\beta = \dfrac{\tilde{N}_p}{N_p + \tilde{N}_p + 4b^2 N_p \tilde{N}_p}$ where b is a constant to be decided empirically.**
- **Discussion:**
  - **$\beta = \dfrac{1}{\frac{N_p}{\tilde{N}_p} + 1 + 4b^2 N_p}$**
  - **We know $\tilde{N}_p \geq N_p$, hence $\frac{1}{2 + 4b^2 N_p} \leq \beta \leq \frac{1}{1 + 4b^2 N_p}$.**
  - **During updating, when N'_p increases a lot due to AMAF being applied on many of its children, then β becomes larger.**

Node expansion

- **May decide to expand potentially good nodes judging from the current statistics [Yajima et al'11].**
  - **All ends**: expand all possible children of a newly added node.
  - **Visit count**: delay the expansion of a node until it is visited a certain number of times.
  - **Transition probability**: delay the expansion of a node until its "score" or estimated visit count is high comparing to its siblings.
    - ▷ *Use the current mean, variance and parent's values to derive a good estimation using statistical methods.*

- **Expansion policy with some transition probability is much better than the "all ends" or pure "visit count" policy.**

Temperature (1/2)

- **Constant temperature: consider all the legal moves and play the ith move with a probability proportional to $e^{(K \cdot v_i)}$, where**
  - v_i **is the current value of the position obtained by taking move i;**
    - ▷ *It is usually the case $v_i \geq 0$.*
    - ▷ $e^{(K \cdot v_i)} \geq 1.$
  - $K \geq 0$ **is the inverse of the temperature used in a simulated annealing setting.**
    - ▷ *Add extra randomness by setting a constant K.*
    - ▷ *The probability of playing the ith move is $P_i(K) = \dfrac{e^{K \cdot v_i}}{\sum_{\forall q} e^{K \cdot v_q}}.$*
    - ▷ *When $K = 0$, this means temperature is ∞ and the selection is uniformly random.*
    - ▷ *If $v_i > v_j$ and $K_1 > K_2$, then $P_i(K_1) - P_j(K_1) > P_i(K_2) - P_j(K_2).$*
    - ▷ *When K becomes larger, the value of v_i contributes more in the calculation of $P_i(K)$.*

Temperature (2/2)

- **Results for constant temperature [Bouzy et al'04]:**

K	0	2	5	10	20
score	-8.1	0	+2.6	-4.9	-11.3

  - When temperature is very high ($K = 0$) when means pure random, then it looks bad.
  - When there is no added randomness ($K > 5$), it also looks bad.
  - Tradeoff between current score and randomness.

- **Simulated annealing:** $P_i(K_t) = \dfrac{e^{K_t \cdot v_i}}{\sum_{\forall j} e^{K_t \cdot v_j}}$ **where K_t is the value of** K **at the tth moment.**

  - Change the temperature over the time.
    - ▷ *In the beginning, allow more randomness, and decrease the amount of randomness over the time.*
  - Increases K from 0 to 5 over time does not enhance the performance.
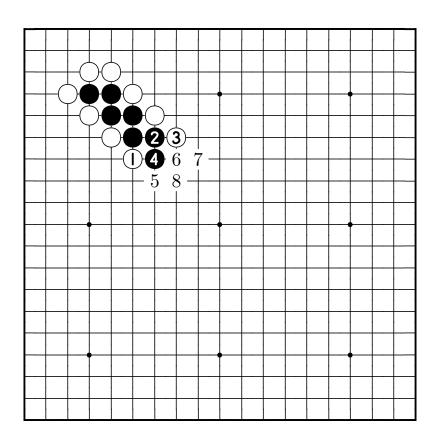
Depth-i enhancement

- **Algorithm:**
  - **Enumerate all possible positions from the root after i moves are made.**
  - **For each position, use Monte-Carlo simulation to get an average score.**
  - **Use a minimax formula to compute the best move from the average scores on the leaves.**
- **Result [Bouzy et al'04]: depth-2 is worse than depth-1 due to oscillating behaviors normally observed in iterative deepening.**
  - **Depth-1 overestimates the root's value.**
  - **Depth-2 underestimates the root's value.**
  - **It is computational difficult for computer Go to get depth-i results when $i > 2$.**

Putting everything together

- **Two versions [Bouzy et al'04]:**
  - **Depth $= 1$, $r_d = 1$, $\sigma_e = 0.2$ with PP, and basic idea.**
  - $K = 2$, **no PP, and all-moves-as-first.**
- **Still worse than GnuGo in 2004, a Go program with lots of domain knowledge, by more than 30 points.**
- **Conclusions:**
  - **Add tactical search: for example, ladders.**
  - **Add more domain knowledge besides no filling of eyes: for example, in Atari, simulate extending plys first.**
    - ▷ *An extending ply is one which can increase the amount of liberty of some strings.*
  - **As the computer goes faster, more domain knowledge can be added.**
  - **Exploring the locality of Go using statistical methods.**

Ladder

- White to move next at 1, then black at 2, then white at 3, and then black at 4, ...

Comments

- **We only describe some specific implementations of general Monte-Carlo techniques.**
  - Other implementations exist for say **AMAF** and others.
- **Depending on the amount of resources you have, you can**
  - decide the frequency to update the node information;
  - decide the frequency to re-pick **PV**;
  - decide the frequency to prune/unprune nodes.

Domain dependent refinements

- **Main technique:**
  - Adding domain knowledge.
- **We use computer Go as an example here.**
- **Refinements come from off-line training.**
  - During the expansion phase:
    - ▷ *Special case: open game.*
    - ▷ *General case: use domain knowledge to expand only the nodes that are meaningful with respect to the game considered, e.g., Go.*
  - During the simulation phase: try to find a better simulation policy.
    - ▷ *Simulation balancing for getting a better playout policy.*
    - ▷ *Other techniques are also known.*
  - Learning of board evaluations, not just good moves.
    - ▷ *Combined with UCB score to form a better estimation on how good or bad the current position is.*
    - ▷ *To start a simulation with a good prior knowledge.*
    - ▷ *To end the simulation earlier when something very bad or very good happened in the middle.*
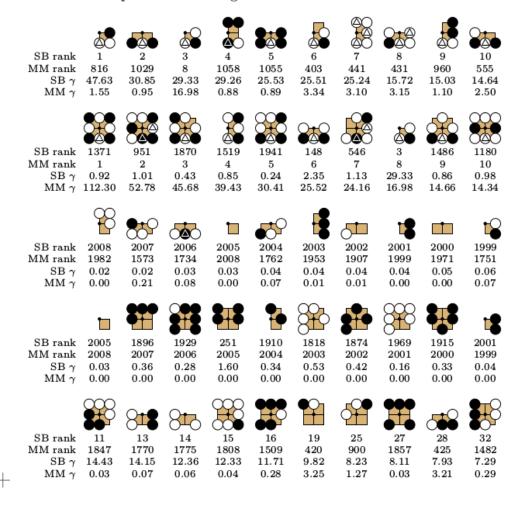
How domain knowledge can be obtained

- **Via human experts: very expensive to get and very difficult to be complete as proven by studies before year 2004 such as GNU Go.**

- **Machine learning.**
  - (Local) pattern: treat positions as pictures and find important patterns and shapes within them.
    - ▷ K *by* K *sub-boards such as* $K = 3$.
    - ▷ *Diamond shaped patterns with different widths.*
    - ▷ *. . .*
  - (Global) feature: find (high order) semantics of positions.
    - ▷ *The liberties of each stone.*
    - ▷ *The number of stones can be captured by playing this intersection.*
    - ▷ *. . .*
  - Need to take care of information that are history dependent, namely cannot be stated using only one position.
    - ▷ *Ko.*
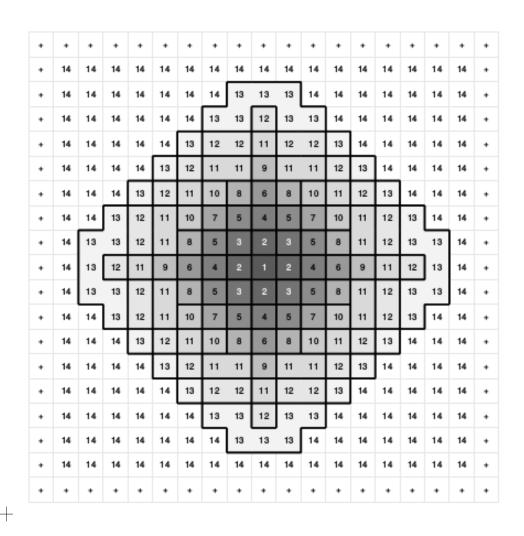   - ▷ *Features include previous several plys in a position.*

3 by 3 patterns

- **[Huang et al'10]**

Table 4. 3×3 patterns. A triangle indicates a stone in atari. Black to move.

SB rank	1	2	3	4	5	6	7	8	9	10
MM rank	816	1029	8	1058	1055	403	441	431	960	555
SB γ	47.63	30.85	29.33	29.26	25.53	25.51	25.24	15.72	15.03	14.64
MM γ	1.55	0.95	16.98	0.88	0.89	3.34	3.10	3.15	1.10	2.50

SB rank	1371	951	1870	1519	1941	148	546	3	1486	1180
MM rank	1	2	3	4	5	6	7	8	9	10
SB γ	0.92	1.01	0.43	0.85	0.24	2.35	1.13	29.33	0.86	0.98
MM γ	112.30	52.78	45.68	39.43	30.41	25.52	24.16	16.98	14.66	14.34

SB rank	2008	2007	2006	2005	2004	2003	2002	2001	2000	1999
MM rank	1982	1573	1734	2008	1762	1953	1907	1999	1971	1751
SB γ	0.02	0.02	0.03	0.03	0.04	0.04	0.04	0.04	0.05	0.06
MM γ	0.00	0.21	0.08	0.00	0.07	0.01	0.01	0.00	0.00	0.07

SB rank	2005	1896	1929	251	1910	1818	1874	1969	1915	2001
MM rank	2008	2007	2006	2005	2004	2003	2002	2001	2000	1999
SB γ	0.03	0.36	0.28	1.60	0.34	0.53	0.42	0.16	0.33	0.04
MM γ	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

SB rank	11	13	14	15	16	19	25	27	28	32
MM rank	1847	1770	1775	1808	1509	420	900	1857	425	1482
SB γ	14.43	14.15	12.36	12.33	11.71	9.82	8.23	8.11	7.93	7.29
MM γ	0.03	0.07	0.06	0.04	0.28	3.25	1.27	0.03	3.21	0.29

Diamond shaped patterns

- **[Stern et al'06]**

Supervised learning

- **Use supervised learning to get a good prediction on the move to choose when a position is given: a vast amount of expert games with possible annotations are available.**
  - **Training phase.**
    - ▷ *Feed positions and their corresponding actions (moves) in expert games into the learning program.*
    - ▷ *Feature and pattern extraction from these positions.*
  - **Prediction phase.**
    - ▷ *Predict the probability of a move will be taken when a position is encountered.*

- **Many different paradigms and algorithms.**
  - **A very active research area with many applications.**

Reinforcement learning

- **Use reinforcement learning to boost the baseline prediction, obtained from supervised learning for example, using self-play or expert annotated games.**
  - The baseline one needs to be good enough to achieve some visible improvement.
  - Feed evaluations of positions from the baseline one into the learning program.
    - ▷ *The objective of the learning is different from the supervised learning phase.*
    - ▷ *To learn which move will result in better positions, namely positions with better evaluations.*

- **Note that the predictions of moves best matched with the training data and moves best matched with better positions may be very different.**
- **Many different paradigms and algorithms.**
  - Another very active research area with many applications.

History

- **Using machine learning to aid computer Go programs is not new.**
  - NeuroGo [Enzenberger'96]: neural network based move predication.
  - IndiGo [Bouzy and Chaslot'05]: Bayesian network.
- **Pure learning approach is very difficult to compete with top Go programs with searching.**
  - Need to combine some forms of searching.
- **Hardware constraints.**

Combining with MCTS

- **Places that MCTS needs helps.**
  - **The expansion phase: what children to explore when a leaf is to be expanded.**
  - **The simulation phase.**
    - ▷ *Originally almost random games are generated: needs a huge amount of simulated games to have a high confidence in the outcome.*
    - ▷ *Can we use more domain knowledge to get a better confidence using the same number of simulations?*
  - **Position evaluation: to end a simulation earlier or to start a simulation with better prior knowledge.**
- **Fundamental issue: assume we can only afford to use a fixed amount of resources R, say computing power in a given time constraint.**
  - **Assume each simulation takes r_s amount of resources for a strategy s in generating a playout.**
    - ▷ *Hence we can only afford to have $\frac{R}{r_s}$ playouts.*
  - **How to pick s to maximize c_s, the confidence or quality?**
    - ▷ *Difficult to define confidence or quality.*
  - **Not likely that r_s is linearly proportional to c_s.**

Machine learning

- **Many different framework and theories.**
  - **Decision tree.**
  - **Support vector machine.**
  - **Bayesian network.**
  - **Artificial Neural network.**
  - **...**

- **Here we will only introduce Bayesian network and multi-layer artificial neural network which is also called convolutional neural network (CNN).**

- **For each framework, depending on how the underlying optimization problem is solved, there are many different simplified models.**
  - **We will only introduce some popular models used in game playing.**
  - **There are many open-source or public domain softwares available.**

Bayesian network based learning (1/3)

- **Bayes theorem:** $P(B \mid A) = \frac{P(A|B)P(B)}{P(A)}$.
  - A: features and patterns
  - B: an action or a move
  - $P(A)$: probability of A happens in the training data set
  - $P(B)$: probability of an action B is taken
  - $P(A \mid B)$: probability of A appears in the training set when an action B is taken.
    - ▷ *this is the training phase.*
  - $P(B \mid A)$: when A appears, the **prediction** of B is taken.
- **Assume there are two actions B_1 and B_2 that one can take in a positions with the feature set A, then use the values of $P(B_1 \mid A)$ and $P(B_2 \mid A)$ to make a decision.**
  - Take one with a larger value.
  - Take one with a chance proportional to its value.
  - $\cdots$

Bayesian network based learning (2/3)

- **When the training set is huge and the feature set A is large, it is very time and space consuming to compute.**
  - **Training data are usually huge in quantity, may contain error, and most of the time incomplete.**
  - **When there are many features in a position, it is very time and space consuming to compute $P(B \mid A)$.**

- **Use some sort of approximation.**
  - **Assume a position A consisted of features $A_1, A_2, \ldots, A_w$.**
  - **For a possible next move B in A, give each feature A_i a strength or influence parameter $q(B, A_i)$ so that it approximates the probability of $P(B \mid A_i)$.**
  - **Use a function $f(q(B, A_1), \ldots, q(B, A_w))$ to approximate the value of $P(B \mid A)$.**

Bayesian network based learning (3/3)

- **Many different models to approximate the strength or influence parameter, θ, of a party, player, feature or pattern.**
  - **Bradley-Terry (BT) model.**
    - *Given 2 players with strengths θ_i and θ_j, $P(i \text{ beats } j) = \frac{e^{\theta_i}}{e^{\theta_i} + e^{\theta_j}}$.*
    - *Generalized model: Comparisons between teams of players, say odds of players $i + j$ beats both $k + m$ and $j + n + p$ is $\frac{e^{\theta_i + \theta_j}}{e^{\theta_i + \theta_j} + e^{\theta_k + \theta_m} + e^{\theta_j + \theta_n + \theta_p}}$.*
  - **Thurstone-Mosteller (TM) model.**
    - *Given 2 players with strengths to be Gaussian distributed (or normal distributed) with $\mathcal{N}(\theta_i, \sigma_i^2)$ and $\mathcal{N}(\theta_j, \sigma_j^2)$, $P(i \text{ beats } j) = \Phi(\frac{e^{\theta_i} - e^{\theta_j}}{\sqrt{\sigma_i^2 + \sigma_j^2}})$, where $\mathcal{N}(\mu, \sigma^2)$ is a normal distribution with mean μ and variance σ^2, and Φ is the c.d.f. of the standard normal distribution, namely $\mathcal{N}(0, 1)$.*

- **May not be reasonable in real life.**
  - Does not allow cyclic relations among players.
  - Strength of a team needs not to be product of teammate's strength.
- **We will use mainly BT model to illustrate the ideas here.**

BT model

- **This is also how Elo rating system is computed between players in games like Chess or Go.**
  - **Example: The Elo rating number of player i with strength θ_i is $400 \log_{10}(e^{\theta_i})$.**
    - ▷ *Assume the Elo ratings of players A, B and C are 2,800, 2,900 and 3,000 respectively.*
    - ▷ $P(C \text{ beats } B) = \frac{10^{3000/400}}{10^{3000/400}+10^{2900/400}} = \frac{10^{7.5}}{10^{7.5}+10^{7.25}} \sim 0.64.$
    - ▷ $P(B \text{ beats } A) = \frac{10^{2900/400}}{10^{2900/400}+10^{2800/400}} = \frac{10^{7.25}}{10^{7.25}+10^{7}} \sim 0.64.$
    - ▷ $P(C \text{ beats } A) = \frac{10^{3000/400}}{10^{3000/400}+10^{2800/400}} = \frac{10^{7.5}}{10^{7.5}+10^{7}} \sim 0.76.$
    - ▷ *Note that $P(i \text{ beats } j) + P(j \text{ beats } i) = 1$ assuming no draw.*

- **Fundamental problem:**
  - **When data are incomplete but huge, how to compute the strength parameters using limited amount of resources?**
  - **The problem is even bigger when data may contain some errors.**

Minorization-Maximization (MM)

- **Minorization-Maximization (MM): an approximation algorithm for the BT model [Coulom'07].**
  - Patterns: all possible, for example $3*3$ patterns, i.e., $3^9 = 19,683$ of them.
  - Training set: records of expert games.
- **When doing the simulation, use the prediction algorithm to find a random playout.**
  - It is easy to have an efficient implementation.
  - Can add some amount of randomness in selecting the moves, such as using the idea of temperature.
- **Results are very good: 37.86% correctness rate using 10,000 expert games [Wistuba et al'12]**
  - A very good playout policy may not be good for the purpose of finding out the average behavior.
    - ▷ *The samplings must consider the average "real" behavior of a player can make.*
    - ▷ *It is extremely unlikely that a player will make trivially bad moves.*
  - Need to balance the amount of resources used in carrying out the policy found and the total number of simulations can be computed.

Simulation balancing (SB)

- **Use the idea of self-play games to boost the performance [Huang et al'11].**
  - **Supervised learning.**
  - **Feature set can be smaller.**
  - **Normally does not learn what positions are played in expert games, but how good or bad a position is.**
    - ▷ *Some forms of position evaluation.*

- **Results are extremely positive for 9 by 9 Go.**
  - **Against GNU Go 3.8 level 10.**
    - ▷ 62.6% *winning rate using SB against a good baseline program of* 50%.
    - ▷ 59.3% *winning rate using MM against a good baseline of* 50%.

- **Results are not as good on 19 by 19 Go against one using MM along.**
- **Erica, a computer Go program using SB based on [Huang et al'11] won 2010 Computer Olympiad 19x19 Go Gold medal.**

How they are used

- **Assume using the BT model.**
- **Generation of the pattern database:**
  - **Manually construct.**
  - **Exhaustive enumeration: small patterns such as 3 by 3.**
  - **Find patterns happened more than a certain number of times in the training set.**
    - ▷ *Patterns, for example diamond-shapes, are too large to enumerate.*

- **Training.**
- **Setting of the parameters:**
  - **Assume after training, feature or pattern i has a strength θ_i.**
  - **Let the current position be p with b possible child positions $p_1, \ldots, p_b$.**
  - **Let F_i be the features or patterns occurred in p_i.**
  - **Let the score of p_i be $S_i = \Pi_{j \in F_i} \theta_j$.**

- **Child p_j is chosen with the probability $\dfrac{S_j}{\sum_{i=1}^{b} S_i}$.**
  - ▷ *The best child is one with the largest score.*

Comments

- **Implementation:**
  - Incrementally update the features and patterns found.
  - Use some variations of the Zobrist hash function to efficiently find the strength of a feature or pattern.
- **We only show one possible avenue of using Bayesian network based learning via using the BT model, MM and SB.**
  - There are many other choices such Bayesian full ranking.
- **The training phase needs to be done once, but takes a huge amount of space and time.**
  - Usually use some forms of iterative updating algorithms to obtain the parameters, namely the strength vector, of the model.
  - For MM with k distinct features or patterns, n training positions and an average b legal moves for each position, it takes $O(kbn)$ space, and X iterations each of which takes $O(bnkh + k^2bn)$ time, where h is the size of the pattern or feature and X is the number of iterations needed for the approximation algorithm to converge [Coulum'07].
- **The prediction phase takes only $O(kh)$ space and time.**
- **Q: Can the part of feature extraction and weights of multiple features be done automatically?**

Convolutional Neural network (CNN)

- **Using a complex network of neurons to better approximate non-linear optimizations.**
  - Usually called **deep learning** when the number of artificial neural network layers is more than 1.
- **A hot learning method inspired by the biological process of the animal visual cortex.**
  - Each neuron takes input from possibly overlaid neighboring sub-images of an image, and then assigns appropriate weights to each input plus some values within the cell to compute the output value.
  - This process can have multiple layers, namely a neuron's output can be other neurons' inputs, and forms a complex network.
  - Depending on the network structure, Bayesian network approaches tends to need less resources than the CNN approach.
  - There are also training phase and prediction phase.
- **Many different tools which can be parallelized using GPU.**
  - Need a great deal of resources to do training and some amount of time to do prediction.

Basics of CNN

- **Assume the ith neuron whose output is z_i takes m_i inputs $x_{i,1}, \ldots, x_{i,m_i}$, and has internal states $y_{i,1}, \ldots, y_{i,n_i}$.**
- **We want to assign weights $w_{i,1}, \ldots, w_{i,m_i+n_i}$ so that**

$$z_i = f(\sum_{j=1}^{m_i}(w_{i,j} * x_{i,j}) + \sum_{j=1}^{n_i}(w_{i,j+m_i} * y_{i,j})),$$

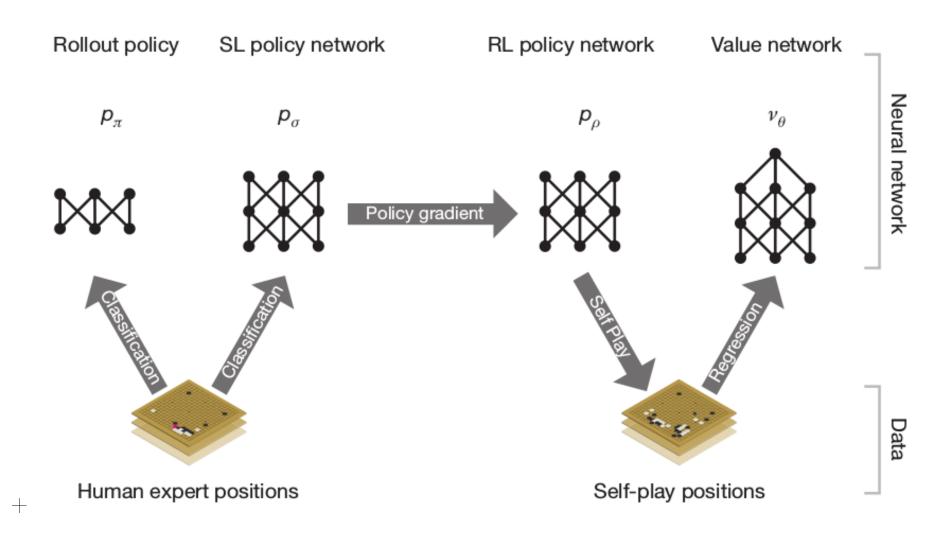 **where f is a transformation function that is not hard to compute.**
- **When there are multiple input data set, we want to find an assignment of the weights so that some error function is minimized.**
  - **The error function can be the average distance, in terms of L_1 or L_2 metric, among the training data set.**
- **Many different algorithms to compute weights.**
  - **Computation intensive.**
  - **Space usage intensive.**

Deep learning

- **Use CNN of different sizes (policies) to achieve different missions in playing 19 by 19 Go [Silver et al'15].**
  - **Supervised learning (SL) in building policy networks which spell out a probability distribution of possible next moves of a given position.**
    - ▷ *A fast rollout policy: for the simulation phase of MCTS, prediction rate is 24.2% using only 2 μs.*
    - ▷ *A better SL rollout policy: 13-layer CNN with a prediction rate of 57.0% using 3 ms.*
  - **Reinforcement learning (RL): obtain both a better, namely more accurate, policy network and a value network for position evaluation.**
    - ▷ *RL policy: further training on the top of the previously obtained SL policy using more features and self-play games that achieves an 80% winning rate against the SL rollout policy.*
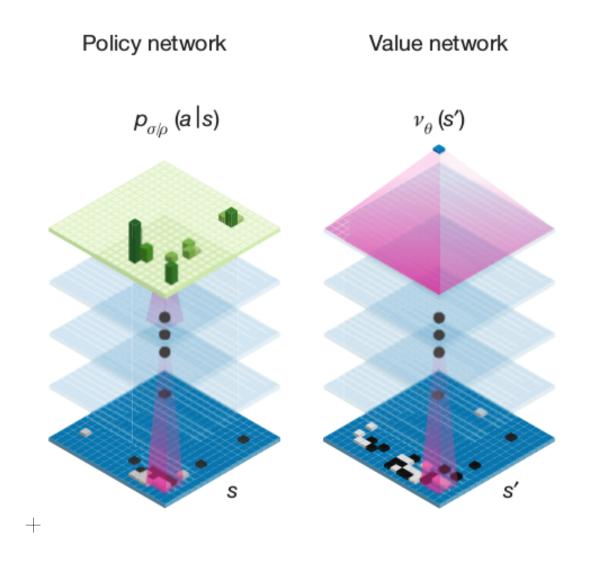   - ▷ *Value network: using the RL policy to train for knowing how good or bad a position is.*

Various networks in AlphaGo

- **[Silver et al'15]**

How networks are obtained by AlphaGo

- [Silver et al'15]

Policy network

$$p_{\sigma|\rho}(a|s)$$

Value network
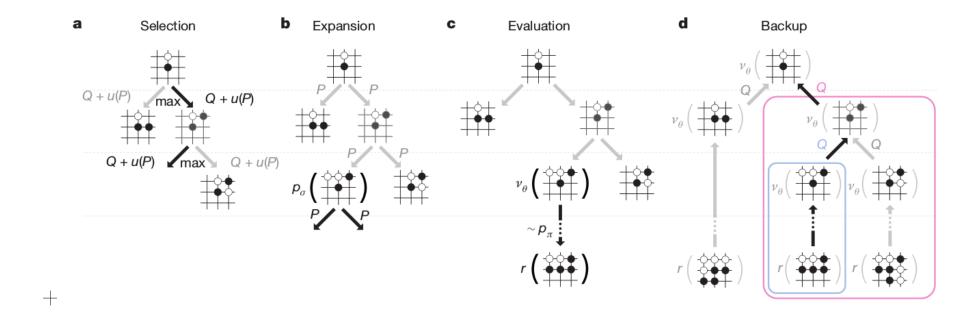
$$v_\theta(s')$$

s

s'

Combining networks

- **Use a fast, but less accurate, SL Rollout policy to do the simulations.**
  - Need to do lots of simulations.
- **Use a slow, but more accurate, SL policy in the expansion phase.**
  - Do not need to do node expansions too often.
- **Use a slow, resource consuming and complex, but more informatic RL policy to construct the value network.**
  - Do not need to do node evaluations too often.
- **Using a combination of the output from the value network and the current score from the simulation phase, one can decide whether to end a simulation earlier or not.**

How networks are used in AlphaGo

- **[Silver et al'15]**

Comments (1/3)

- **A very good tradeoff in performance and amount of resources used.**
  - A less accurate but fast rollout policy is used with MCTS so that the tree search part can augment correctness rate.
    - ▷ *Need to do lots of simulations so each cannot take too much time.*
  - Use a slow but more accurate policy for tasks such as expansion that do not need to carry out many times.
  - Use reinforcement learning in obtaining a value network to replace the role of designing complicated evaluating functions.
- **Now is the way to go for computer Go!**
  - Performance is extremely well and is generally considered to be over human champion.
  - Lots of legacy teams such as Zen and Crazystone are embracing CNN.
  - New teams such as Darkforest, developed by Facebook, are catching up.
- **This approach can be used in many applications such as medical informatic which includes medical image and signal reading.**
  - Anything that is pattern related and has lots of data collected with expert annotations.

Comments (2/3)

- **Take a lot of computing resources for computer Go.**
  - More than 100,000 features and patterns.
  - More than 40 machines each with 32 cores and a total of more than 176 GPU cards whose power consumption is estimated to be in the order of 10^3 KW.
- **We only know it works by building the CNN, but it is almost impossible to explain how it works.**
  - Very difficult to debug if a silly bug occurs.
  - Very difficult to "control" it to act the way you wanted to.
  - It is an art to find the right coefficients and tradeoff.
- **More studies are needed to lower the amount of resources used and to do transfer learning, namely duplicate the successful experience on one domain to another domain.**

Comments (3/3)

- **We also describe some fundamental techniques and ideas in the part of combining machine learning.**
  - Other machine learning tools are also available and used.
- **Using machine learning or MTCS along won't solve the performance problem in computer Go. However, the combination of both does the magic.**

References and further readings (1/5)

* **Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In** *Proceedings of the 24th international conference on Machine learning*, **ICML '07, pages 273–280, New York, NY, USA, 2007. ACM.**

* **David Silver. Reinforcement Learning and Simulation-Based Search in Computer Go. PhD thesis, University of Alberta, 2009.**

References and further readings (2/5)

- B. Bouzy and B. Helmstetter. Monte-Carlo Go developments. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, *Advances in Computer Games, Many Games, Many Challenges, 10th International Conference, ACG 2003, Graz, Austria, November 24-27, 2003, Revised Papers*, volume **263** of *IFIP*, pages 159–174. Kluwer, 2004.
- Hugues Juille. *Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm*. PhD thesis, Department of Computer Science, Brandeis University, May 1999.

* Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... and Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

References and further readings (3/5)

- **Shih-Chieh Huang, Rmi Coulom, and Shun-Shii Lin. Monte-Carlo Simulation Balancing in Practice. In H. Jaap van den Herik, H. Iida, and A. Plaat, editors,** *Lecture Notes in Computer Science 6515: Proceedings of the 7th International Conference on Computers and Games*, **pages 81–92. Springer-Verlag, New York, NY, 2011.**

- **Stern, D., Herbrich, R., and Graepel, T. (2006, June). Bayesian pattern ranking for move prediction in the game of Go. In Proceedings of the 23rd international conference on Machine learning (pp. 873-880). ACM.**

- **Wistuba, M., Schaefers, L., and Platzner, M. (2012, September). Comparison of Bayesian move prediction systems for Computer Go. In Computational Intelligence and Games (CIG), 2012 IEEE Conference on (pp. 91-99). IEEE.**

References and further readings (4/5)

- Coulom, R. (2007). Computing Elo ratings of move patterns in the game of go. In Computer games workshop.
- B. Bouzy and G. Chaslot, "Bayesian generation and integration of K-nearest-neighbor patterns for 19x19 Go", IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK, G. Kendall & Simon Lucas (eds), pages 176-181.
- Enzenberger, M. (1996). The integration of a priori knowledge into a Go playing neural network. URL: http://www. markus-enzenberger.de/neurogo.html.
- Clark, C., & Storkey, A. (2014). Teaching deep convolutional neural networks to play go. arXiv preprint arXiv:1412.3409.

References and further readings (5/5)

- Maddison, C. J., Huang, A., Sutskever, I., & Silver, D. (2014). Move evaluation in go using deep convolutional neural networks. arXiv preprint arXiv:1412.6564.

- Tian, Y., & Zhu, Y. (2015). Better Computer Go Player with Neural Network and Long-term Prediction. arXiv preprint arXiv:1511.06410.

- Takayuki Yajima, Tsuyoshi Hashimoto, Toshiki Matsui, Junichi Hashimoto, and Kristian Spoerer. Node-expansion operators for the UCT algorithm. In H. Jaap van den Herik, H. Iida, and A. Plaat, editors, *Lecture Notes in Computer Science 6515: Proceedings of the 7th International Conference on Computers and Games*, pages 116–123. Springer-Verlag, New York, NY, 2011.