Monte-Carlo Simulation Balancing

David Silver[1] and Gerald Tesauro[2]

[1] Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta
[2] IBM TJ Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 USA

Monte-Carlo search has dramatically outperformed traditional search methods in two-player games such as Go, Backgammon and Scrabble. The performance of Monte-Carlo search is primarily determined by the quality of the simulation policy. However, *strong* simulation policies, constructed by hand, supervised learning, or reinforcement learning, do not typically produce the diverse, well-balanced simulations that are desirable for a Monte-Carlo player. Instead, we introduce a new paradigm for learning a simulation policy that explicitly optimizes a Monte-Carlo objective.

We consider training a softmax simulation policy $\pi_\theta(s,a)$ with parameter vector θ, to perform Monte-Carlo simulations in deterministic two-player games. Rather than making π_θ itself as strong as possible, our goal is to maximize the strength of a Monte-Carlo player using π_θ. Intuitively, this should be achieved when the Monte-Carlo simulations are as accurate as possible, i.e., the mean outcome of Monte-Carlo trials from a given position s should match the game-theoretic value $V^*(s)$ or some suitable proxy (e.g., the mean outcome in expert vs. expert play).

We formalize the above intuition by introducing the concept of *balance* as follows. The policy's decision at each time step t incurs some error $\delta_t = V^*(s_{t+1}) - V^*(s_t)$, with δ alternating in sign for the two players. The *k-step balance* B_k of policy π_θ is the policy's expected squared sum of errors δ_t over k adjacent time steps. From the definition of δ_t we may write:

$$B_k(\theta) = \mathbb{E}_\rho[(\mathbb{E}_{\pi_\theta}[V^*(s_{t+k}) - V^*(s_t)|s_t = s])^2] \tag{1}$$

where $\mathbb{E}_\rho$ denotes expectation over the distribution of states $\rho(s)$.

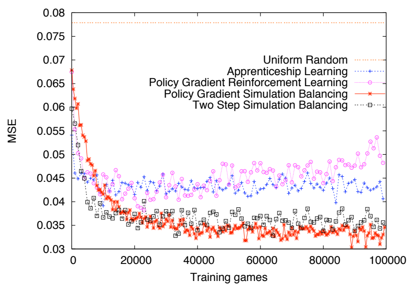
Here we consider two choices of k. The *two-step balance* $B_2(\theta)$ allows errors by one player, as long as they are on average balanced out by the other player's error on the next move. The *full balance* B_∞ allows errors to be committed at any time, as long as they balance out by the time the game is finished. It corresponds exactly to the objective of optimizing the accuracy of Monte-Carlo expected outcomes: $B_\infty(\theta) = \mathbb{E}_\rho[(\mathbb{E}_{\pi_\theta}[V^*(s_T) - V^*(s_t)|s_t = s])^2] = \mathbb{E}_\rho[(\mathbb{E}_{\pi_\theta}[z|s_t = s] - V^*(s))^2]$, where s_T is the terminal state with outcome z.

We propose algorithms to optimize B_2 and B_∞ by gradient descent which we respectively call *two-step simulation balancing* and *policy gradient simulation balancing*. In both cases the gradient expression involves a product of a bias term $b(s)$ times a policy gradient term $g(s)$. In two-step simulation balancing these terms can be calculated analytically, while in policy gradient simulation balancing, they can be estimated by (independent) stochastic sampling.

We compare our balancing algorithms with two alternatives based on optimizing policy strength. One alternative, *apprenticeship learning*, is trained on a database of expert moves. It minimizes the expected KL-divergence between the simulation policy $\pi(s,a)$ and the observed expert policy, $\hat{\mu}(s,a)$, by gradient descent. The other alternative uses *policy gradient RL*, specifically a form of Williams' REINFORCE algorithm [1].

The above algorithms were applied in 5×5 and 6×6 Go. The apprenticeship learning and simulation balancing algorithms made use of a data-set of positions from 1000 games of randomly played 5×5 and 6×6 Go games. We used the open source Monte-Carlo Go program *Fuego* to evaluate each position, using a deep search of 10k simulations from each position. The results of the search are used to approximate the optimal value $\hat{V}^*(s) \approx V^*(s)$. For the two step simulation balancing algorithm, a complete tree of depth 2 was also constructed from each position, and each leaf position evaluated by a further 2k simulations. These leaf evaluations approximate the optimal value after each possible move and response.

The softmax policies used 107 weights for each unique 1×1 and 2×2 pattern of stones, given rotational, reflectional and colour inversion symmetries. We trained the simulation policy using 100k training games, starting with initial weights of zero. All four algorithms converged to stable solutions, and significantly reduced the Monte-Carlo evaluation error compared to the uniform random policy. The simulation balancing algorithms achieved less than half the MSE of the uniform random policy, and 10-30% lower error than apprenticeship learning or policy gradient RL (Figure 1a).



	5x5		6x6	
Simulation Policy	Direct	MC	Direct	MC
Uniform random	0	1031	0	970
Apprenticeship learning	671	1107	569	1047
Policy gradient RL (20k)	816	1234	531	1104
Policy gradient RL (100k)	**947**	1159	**850**	1023
Policy gradient sim. balancing	719	**1367**	658	**1301**
Two step simulation balancing	720	1357	444	1109
GnuGo 3.7.10 (level 10)	1376	N/A	1534	N/A

Fig. 1. a) Mean-squared error for Monte-Carlo evaluation of 5×5 Go positions during training, b) Elo rating of fully trained simulation policies in 5×5 Go and 6×6 Go tournaments.

Finally, we ran a tournament between players based on each simulation policy: firstly selecting moves directly according to the simulation policy; and secondly using the simulation policy in a Monte-Carlo search algorithm. Our search algorithm was intentionally simplistic: for every legal move a, we simulated 100 games starting with a, and selected the move with the greatest number of wins. We included two simulation policies that were trained by policy gradient RL, which maximized strength and minimized MSE respectively, after 100k and 20k games of training respectively. The results of the tournament are shown in Figure 1b, and compared to GnuGo, a deterministic Go program with sophisticated, handcrafted knowledge and specialized search algorithms.

When the simulation policies are used directly, policy gradient RL (100k) is by far the strongest, 200 Elo points stronger than simulation balancing. However, when used as a Monte-Carlo policy, simulation balancing is much stronger, 200 Elo points above policy gradient RL (100k), and almost 300 Elo stronger than apprenticeship learning. Two-step simulation balancing performed almost as well as Monte-Carlo simulation balancing in 5×5 Go, but appeared to get stuck in poor local optima in 6×6 Go.

[1] Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8** (1992) 229–256

Topic: Control (reinforcement learning)
Preference: oral