Monte-Carlo Simulation Balancing

David Silver SILVER@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, AB

Gerald Tesauro GTESAURO@US.IBM.COM
IBM Watson Research Center, 19 Skyline Drive, Hawthorne, NY

Abstract

In this paper we introduce the first algorithms for efficiently learning a simulation policy for Monte-Carlo search. Our main idea is to optimise the *balance* of a simulation policy, so that an accurate spread of simulation outcomes is maintained, rather than optimising the direct *strength* of the simulation policy. We develop two algorithms for balancing a simulation policy by gradient descent. The first algorithm optimises the balance of complete simulations, using a policy gradient algorithm; whereas the second algorithm optimises the balance over every two steps of simulation. We compare our algorithms to reinforcement learning and supervised learning algorithms for maximising the strength of the simulation policy. We test each algorithm in the domain of 5×5 and 6×6 Computer Go, using a softmax policy that is parameterised by weights for a hundred simple patterns. When used in a simple Monte-Carlo search, the policies learnt by simulation balancing achieved significantly better performance, with half the mean squared error of a uniform random policy, and similar overall performance to a sophisticated Go engine.

1. Introduction

Monte-Carlo search algorithms use the average outcome of many simulations to evaluate candidate actions. They have achieved human master level in a variety of stochastic two-player games, including Backgammon (Tesauro & Galperin, 1996), Scrabble (Sheppard, 2002) and heads up Poker (Billings et al., 1999). *Monte-Carlo tree search* evaluates each state in a search-tree by Monte-Carlo simulation. It has proven surprisingly successful in deterministic two-player games, achieving master-level at 9×9 Go (Gelly & Silver, 2007; Coulom, 2007) and winning the General Game-Playing competition (Finnsson & Björnsson, 2008).

In these algorithms, many games of self-play are simulated, using a *simulation policy* to select actions for both players. The overall performance of Monte-Carlo search is largely determined by the simulation policy. A simulation policy with appropriate domain knowledge can dramatically outperform a uniform random simulation policy (Gelly et al., 2006). Automatically improving the simulation policy is a major goal of current research in this area (Gelly & Silver, 2007; Coulom, 2007; Chaslot et al., 2008). Two approaches have previously been taken to improving the simulation policy.

The first approach is to directly construct a *strong* simulation policy that performs well by itself, either by hand (Billings et al., 1999), reinforcement learning (Tesauro & Galperin, 1996; Gelly & Silver, 2007), or supervised learning (Coulom, 2007). Unfortunately, a stronger simulation policy can actually lead to a weaker Monte-Carlo search (Gelly & Silver, 2007), a paradox that we explore further in this paper.

The second approach to learning a simulation policy is by trial and error, adjusting parameters and testing for improvements in the performance of the Monte-Carlo player, either by hand (Gelly et al., 2006), or by hill-climbing (Chaslot et al., 2008). However, each parameter evaluation usually requires many complete games, thousands of positions, and millions of simulations to be executed. Furthermore, hill-climbing methods do not scale well with increasing dimensionality, and fare poorly with complex policy parameterisations.

Handcrafting an effective simulation policy is partic-

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

This work was supported in part under the DARPA GALE project, contract No. HR0011-08-C-0110.

ularly problematic in Go. Many of the top Go programs utilise a small number of simple patterns and rules, based largely on the default policy used in *MoGo* (Gelly et al., 2006). Adding further Go knowledge without breaking *MoGo*'s "magic formula" has proven to be surprisingly difficult.

In this paper we introduce a new paradigm for learning a simulation policy. We define an objective function, which we call *imbalance*, that explicitly measures the performance of a simulation policy for Monte-Carlo evaluation. We introduce two new algorithms that minimise the imbalance of a simulation policy by gradient descent. These algorithms require very little computation for each parameter update, and are able to learn expressive simulation policies with hundreds of parameters.

We evaluate our simulation balancing algorithms in the game of Go. We compare them to reinforcement learning and supervised learning algorithms for maximising strength, and to a well-known simulation policy for this domain, handcrafted by trial and error. The simulation policy learnt by our new algorithms significantly outperforms prior approaches.

2. Strength and Balance

We consider deterministic two-player games of finite length with a terminal outcome or score $z \in \mathbb{R}$. During simulation, move a is selected in state s according to a stochastic simulation policy $\pi_\theta(s, a)$ with parameter vector θ, that is used to select moves for both players. The goal is to find the parameter vector θ^* that maximises the overall playing strength of a player based on Monte-Carlo search. Our approach is to make the Monte-Carlo evaluations in the search as accurate as possible, by minimising the mean squared error between the estimated values $V(s) = \frac{1}{N}\sum_{i=1}^{N} z_i$ and the minimax values $V^*(s)$.

When the number of simulations N is large, the mean squared error is dominated by the bias of the simulation policy with respect to the minimax value, $V^*(s) - \mathbb{E}_{\pi_\theta}[z|s]$, and the variance of the estimate (i.e. the error caused by only seeing a finite number of simulations) can be ignored. Our objective is to minimise the mean squared bias, averaged over the distribution of states $\rho(s)$ that are evaluated during Monte-Carlo search.

$$\theta^* = \operatorname*{argmin}_\theta \mathbb{E}_\rho \left[\left(V^*(s) - \mathbb{E}_{\pi_\theta}[z|s] \right)^2 \right] \tag{1}$$

where $\mathbb{E}_\rho$ denotes the expectation over the distribution of actual states $\rho(s)$, and $\mathbb{E}_{\pi_\theta}$ denotes the expectation over simulations with policy π_θ.

In real-world domains, knowledge of the true minimax values is not available. In practice, we use the values $\hat{V}^*(s)$ computed by deep Monte-Carlo tree searches, which converge on the minimax value in the limit (Kocsis & Szepesvari, 2006), as an approximation $\hat{V}^*(s) \approx V^*(s)$.

At every time-step t, each player's move incurs some error $\delta_t = V^*(s_{t+1}) - V^*(s_t)$ with respect to the minimax value $V^*(s_t)$. We will describe a policy with a small error as *strong*, and a policy with a small expected error as *balanced*. Intuitively, a strong policy makes few mistakes, whereas a balanced policy allows many mistakes, as long as they cancel each other out on average. Formally, we define the strength $J(\theta)$ and k-step imbalance $B_k(\theta)$ of a policy π_θ,

$$J(\theta) = \mathbb{E}_\rho \left[\mathbb{E}_{\pi_\theta} \left[\delta_t^2 | s_t = s \right] \right] \tag{2}$$

$$\begin{aligned} B_k(\theta) &= \mathbb{E}_\rho \left[\left(\mathbb{E}_{\pi_\theta} \left[\sum_{j=0}^{k-1} \delta_{t+j} | s_t = s \right] \right)^2 \right] \\ &= \mathbb{E}_\rho \left[\left(\mathbb{E}_{\pi_\theta} \left[V^*(s_{t+k}) - V^*(s_t) | s_t = s \right] \right)^2 \right] \end{aligned} \tag{3}$$

We consider two choices of k in this paper. The *two-step imbalance* $B_2(\theta)$ is specifically appropriate to two-player games. It allows errors by one player, as long as they are on average cancelled out by the other player's error on the next move. The *full imbalance* B_∞ allows errors to be committed at any time, as long as they cancel out by the time the game is finished. It is exactly equivalent to the mean squared bias that we are aiming to optimise in Equation 1,

$$\begin{aligned} B_\infty(\theta) &= \mathbb{E}_\rho \left[\left(\mathbb{E}_{\pi_\theta} \left[V^*(s_T) - V^*(s) | s_t = s \right] \right)^2 \right] \\ &= \mathbb{E}_\rho \left[\left(\mathbb{E}_{\pi_\theta}[z|s_t = s] - V^*(s) \right)^2 \right] \end{aligned} \tag{4}$$

where s_T is the terminal state with outcome z. Thus, while the direct performance of a policy is largely determined by its strength, the performance of a policy in Monte-Carlo simulation is determined by its full imbalance.

If the simulation policy is optimal, $\mathbb{E}_{\pi_\theta}[z|s] = V^*(s)$, then perfect balance is achieved, $B_\infty(\theta) = 0$. This suggests that optimising the strength of the simulation policy, so that individual moves become closer to optimal, may be sufficient to achieve balance. However, even small errors can rapidly accumulate over

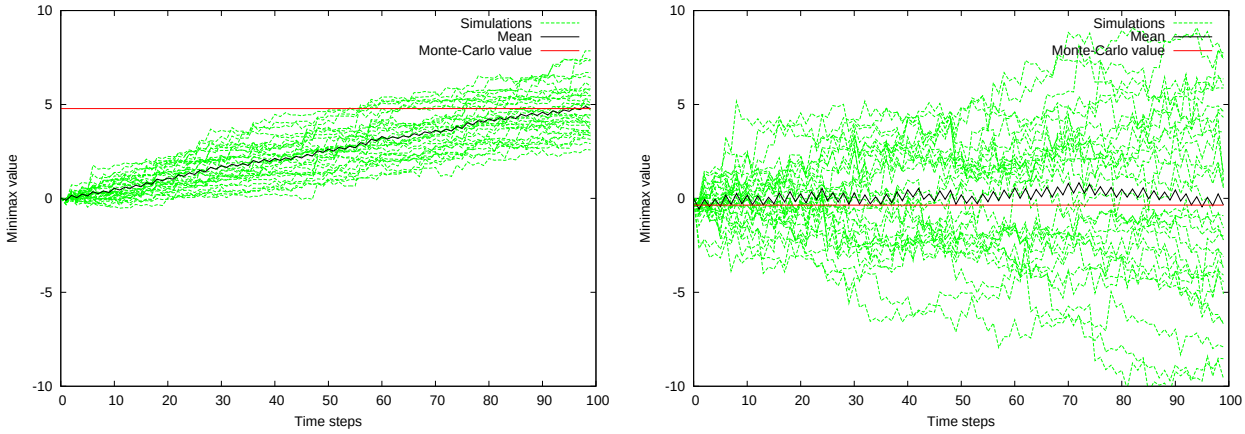

Figure 1. Monte-Carlo simulation in an artificial two-player game. 30 simulations of 100 time steps were executed from an initial state with minimax value 0. Each player selects moves imperfectly during simulation, with an error that is exponentially distributed with respect to the minimax value, with rate parameters λ_1 and λ_2 respectively. a) The simulation players are strong but imbalanced: $\lambda_1 = 10, \lambda_2 = 5$, b) the simulation players are weak but balanced: $\lambda_1 = 2, \lambda_2 = 2$. The Monte-Carlo value of the weak, balanced simulation players is significantly more accurate.

the course of long simulations if they are not well-balanced. It is more important to maintain a diverse spread of simulations, which are on average representative of strong play, than for individual moves or simulations to be low in error. Figure 1 shows a simple scenario in which the error of each player is i.i.d and exponentially distributed with rate parameters λ_1 and λ_2 respectively. A weak, balanced simulation policy ($\lambda_1 = 2, \lambda_2 = 2$) provides a much more accurate Monte-Carlo evaluation than a strong, imbalanced simulation policy ($\lambda_1 = 10, \lambda_2 = 5$).

In large domains it is not usually possible to achieve perfect strength or perfect balance, and some approximation is required. Our hypothesis is that very different approximations will result from optimising balance as opposed to optimising strength, and that optimising balance will lead to significantly better Monte-Carlo performance.

To test this hypothesis, we implement two algorithms that maximise the strength of the simulation policy, using apprenticeship learning and reinforcement learning respectively. We then develop two new algorithms that minimise the imbalance of the simulation policy by gradient descent. Finally, we compare the performance of these algorithms in 5×5 and 6×6 Go.

3. Softmax Policy

We use a *softmax policy* to parameterise the simulation policy,

$$\pi_\theta(s,a) = \frac{e^{\phi(s,a)^T\theta}}{\sum_b e^{\phi(s,b)^T\theta}} \tag{5}$$

where $\phi(s,a)$ is a vector of features for state s and action a, and θ is a corresponding parameter vector indicating the preference of the policy for each feature.

The softmax policy can represent a wide range of stochasticity in different positions, ranging from near deterministic policies with large preference disparities, to uniformly random policies with equal preferences. The level of stochasticity is very significant in Monte-Carlo simulation: if the policy is too deterministic then there is no diversity and Monte-Carlo simulation cannot improve the policy; if the policy is too random then the overall accuracy of the simulations is diminished. Existing paradigms for machine learning, such as reinforcement learning and supervised learning, do not explicitly control this stochasticity. One of the motivations for simulation balancing is to tune the level of stochasticity to a suitable level in each position.

We will need the gradient of the log of the softmax policy, with respect to the policy parameters,

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(s,a) &= \nabla_\theta \log e^{\phi(s,a)^T\theta} - \nabla_\theta \log \left(\sum_b e^{\phi(s,b)^T\theta} \right) \\
&= \nabla_\theta \left(\phi(s,a)^T\theta \right) - \frac{\nabla_\theta \sum_b e^{\phi(s,b)^T\theta}}{\sum_b e^{\phi(s,b)^T\theta}} \\
&= \phi(s,a) - \frac{\sum_b \phi(s,b) e^{\phi(s,b)^T\theta}}{\sum_b e^{\phi(s,b)^T\theta}} \\
&= \phi(s,a) - \sum_b \pi_\theta(s,b)\phi(s,b) \qquad (6)
\end{aligned}$$

which is the difference between the observed feature

vector and the expected feature vector. We denote this gradient by $\psi(s,a)$.

4. Optimising Strength

We consider two algorithms for optimising the strength of a simulation policy, by supervised learning and reinforcement learning respectively.

4.1. Apprenticeship Learning

Our first algorithm optimises the strength of the simulation policy by apprenticeship learning. The aim of the algorithm is simple: to find a simulation policy that behaves as closely as possible to a given expert policy $\mu(s,a)$.

We consider a data-set of L training examples (s_l, a_l^*) of actions a_l^* selected by expert policy μ in positions s_l. The apprenticeship learning algorithm finds parameters maximising the likelihood, $\mathcal{L}(\theta)$, that the simulation policy $\pi(s,a)$ produces the actions a_l^*[1]. This is achieved by gradient ascent of the log likelihood,

$$\begin{aligned}
\mathcal{L}(\theta) &= \prod_{l=1}^{L} \pi(s_l, a_l^*) \\
\log \mathcal{L}(\theta) &= \sum_{l=1}^{L} \log \pi(s_l, a_l^*) \\
\nabla_\theta \log \mathcal{L}(\theta) &= \sum_{l=1}^{L} \nabla_\theta \log \pi(s_l, a_l^*) \\
&= \sum_{l=1}^{L} \psi(s_l, a_l^*) \qquad (7)
\end{aligned}$$

This leads to a stochastic gradient ascent algorithm, in which each training example (s_l, a_l^*) is used to update the policy parameters, with step-size α,

$$\Delta\theta = \alpha \psi(s_l, a_l^*) \qquad (8)$$

4.2. Policy Gradient Reinforcement Learning

Our second algorithm optimises the strength of the simulation policy by reinforcement learning. The objective is to maximise the expected cumulative reward from start state s. *Policy gradient* algorithms adjust the policy parameters θ by gradient ascent, so as to find a local maximum for this objective.

We define $\mathcal{X}(s)$ to be the set of possible games $\xi = (s_1, a_1, ..., s_T, a_T)$ of states and actions, starting from $s_1 = s$. The policy gradient can then be expressed as an expectation over game outcomes $z(\xi)$,

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}[z|s] &= \sum_{\xi \in \mathcal{X}(s)} Pr(\xi) z(\xi) \\
\nabla_\theta \mathbb{E}_{\pi_\theta}[z|s] &= \sum_{\xi \in \mathcal{X}(s)} \nabla_\theta (\pi_\theta(s_1, a_1) ... \pi_\theta(s_T, a_T)) z(\xi) \\
&= \sum_{\xi \in \mathcal{X}(s)} \pi_\theta(s_1, a_1) ... \pi_\theta(s_T, a_T) \\
&\qquad \left(\frac{\nabla_\theta \pi_\theta(s_1, a_1)}{\pi_\theta(s_1, a_1)} + ... + \frac{\nabla_\theta \pi_\theta(s_T, a_T)}{\pi_\theta(s_T, a_T)} \right) z(\xi) \\
&= \mathbb{E}_{\pi_\theta} \left[z \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t) \right] \\
&= \mathbb{E}_{\pi_\theta} \left[z \sum_{t=1}^{T} \psi(s_t, a_t) \right] \qquad (9)
\end{aligned}$$

The policy parameters are updated by stochastic gradient ascent with step-size α, after each game, leading to a REINFORCE algorithm (Williams, 1992),

$$\Delta\theta = \frac{\alpha z}{T} \sum_{t=1}^{T} \psi(s_t, a_t) \qquad (10)$$

5. Optimising Balance

We now introduce two algorithms for minimising the full imbalance and two-step imbalance of a simulation policy. Both algorithms learn from $\hat{V}^*(s)$, an approximation to the minimax value function constructed by deep Monte-Carlo search.

5.1. Policy Gradient Simulation Balancing

Our first simulation balancing algorithm minimises the full imbalance B_∞ of the simulation policy, by gradient descent. The gradient breaks down into two terms. The *bias*, $b(s)$, indicates the direction in which we need to adjust the mean outcome from state s: e.g. does black need to win more or less frequently, in order to match the minimax value? The *policy gradient*, $g(s)$, indicates how the mean outcome from state s can be adjusted, e.g. how can the policy be modified, so as to make black win more frequently?

$$\begin{aligned}
b(s) &= V^*(s) - \mathbb{E}_{\pi_\theta}[z|s] \\
g(s) &= \nabla_\theta \mathbb{E}_{\pi_\theta}[z|s] \\
B_\infty(\theta) &= \mathbb{E}_\rho \left[b(s)^2 \right] \\
\nabla_\theta B_\infty(\theta) &= \mathbb{E}_\rho \left[b^2 \right] = -2\mathbb{E}_\rho \left[b(s) g(s) \right] \qquad (11)
\end{aligned}$$

[1] These parameters are the log of the ratings that maximise likelihood in a generalised Bradley-Terry model (Coulom, 2007).

We estimate the bias, $\hat{b}(s)$, by sampling M simulations $\mathcal{X}_M(s)$ from state s,

$$\hat{b}(s) = \hat{V}^*(s) - \frac{1}{M} \sum_{\xi \in \mathcal{X}_M(s)} z(\xi) \tag{12}$$

We estimate the policy gradient, $\hat{g}(s)$, by sampling N additional simulations $\mathcal{X}_N(s)$ from state s and using Equation 9,

$$\hat{g}(s) = \sum_{\xi \in \mathcal{X}_N(s)} \frac{z(\xi)}{NT} \sum_{t=1}^{T} \psi(s_t, a_t) \tag{13}$$

In general $\hat{b}(s)$ and $\hat{g}(s)$ are correlated, and we need two independent samples to form an unbiased estimate of their product (Algorithm 1). This provides a simple stochastic gradient descent update, $\Delta\theta = \alpha\hat{b}(s)\hat{g}(s)$.

Algorithm 1 Policy Gradient Simulation Balancing

```
θ ← 0
for all s_1 ∈ training set do
  V ← 0
  for i = 1 to M do
    simulate (s_1, a_1, ..., s_T, a_T; z) using π_θ
    V ← V + z/M
  end for
  g ← 0
  for j = 1 to N do
    simulate (s_1, a_1, ..., s_T, a_T; z) using π_θ
    g ← g + z/(NT) Σ_{t=1}^{T} ψ(s_t, a_t)
  end for
  θ ← θ + α(V̂*(s_1) − V)g
end for
```

5.2. Two-Step Simulation Balancing

Our second simulation balancing algorithm minimises the two-step imbalance B_2 of the simulation policy, by gradient descent. The gradient can again be expressed as a product of two terms. The *two-step bias*, $b_2(s)$, indicates whether black needs to win more or less games, to achieve balance between time t and time $t+2$. The *two-step policy gradient*, $g_2(s)$, indicates the direction in which the parameters should be adjusted, in order for black to improve his evaluation at time $t+2$.

$$\begin{aligned} b_2(s) &= V^*(s) - \mathbb{E}_{\pi_\theta}[V^*(s_{t+2})|s_t = s] \\ g_2(s) &= \nabla_\theta \mathbb{E}_{\pi_\theta}[V^*(s_{t+2})|s_t = s] \\ B_2(s)(\theta) &= \mathbb{E}_\rho\left[b_2(s)^2\right] \\ \nabla_\theta B_2(s)(\theta) &= -2\mathbb{E}_\rho\left[b_2(s)g_2(s)\right] \end{aligned} \tag{14}$$

The two-step policy gradient can be derived by applying the product rule,

$$\begin{aligned} g_2(s) &= \nabla_\theta \mathbb{E}_{\pi_\theta}[V^*(s_{t+2})|s_t = s] \\ &= \nabla_\theta \sum_a \sum_b \pi_\theta(s_t, a)\pi_\theta(s_{t+1}, b)V^*(s_{t+2}) \\ &= \sum_a \sum_b \pi_\theta(s_t, a)\pi_\theta(s_{t+1}, b)V^*(s_{t+2}) \\ &\qquad \left(\frac{\nabla_\theta \pi_\theta(s_t, a)}{\pi_\theta(s_t, a)} + \frac{\nabla_\theta \pi_\theta(s_{t+1}, b)}{\pi_\theta(s_{t+1}, b)}\right) \\ &= \mathbb{E}_{\pi_\theta}\left[V^*(s_{t+2})(\psi(s_t, a_t) + \psi(s_{t+1}, a_{t+1}))|s_t = s\right] \end{aligned} \tag{15}$$

Both the two-step bias $b_2(s)$ and the policy gradient $g_2(s)$ can be calculated analytically, with no requirement for simulation, leading to a simple gradient descent algorithm (Algorithm 2), $\Delta\theta = \alpha b_2(s)g_2(s)$.

Algorithm 2 Two-Step Simulation Balancing

```
θ ← 0
for all s_1 ∈ training set do
  V ← 0, g_2 ← 0
  for all a_1 ∈ legal moves from s_1 do
    s_2 = s_1 ∘ a_1
    for all a_2 ∈ legal moves from s_2 do
      s_3 = s_2 ∘ a_2
      p = π(s_1, a_1)π(s_2, a_2)
      V ← V + pV̂*(s_3)
      g_2 ← g_2 + pV̂*(s_3)(ψ(s_1, a_1) + ψ(s_2, a_2))
    end for
  end for
  θ ← θ + α(V̂*(s_1) − V)g_2
end for
```

6. Experiments in Computer Go

We applied each of our algorithms to learn a simulation policy for 5×5 and 6×6 Go. For the apprenticeship learning and simulation balancing algorithms, we constructed a data-set of positions from 1000 games of randomly played games. We used the open source Monte-Carlo Go program *Fuego* to evaluate each position, using a deep search of 10000 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 in the data-set, and each leaf position evaluated by a further 2000 simulations. These leaf evaluations are used in the two-step simulation balancing algorithm, to approximate the optimal value after each possible move and response.

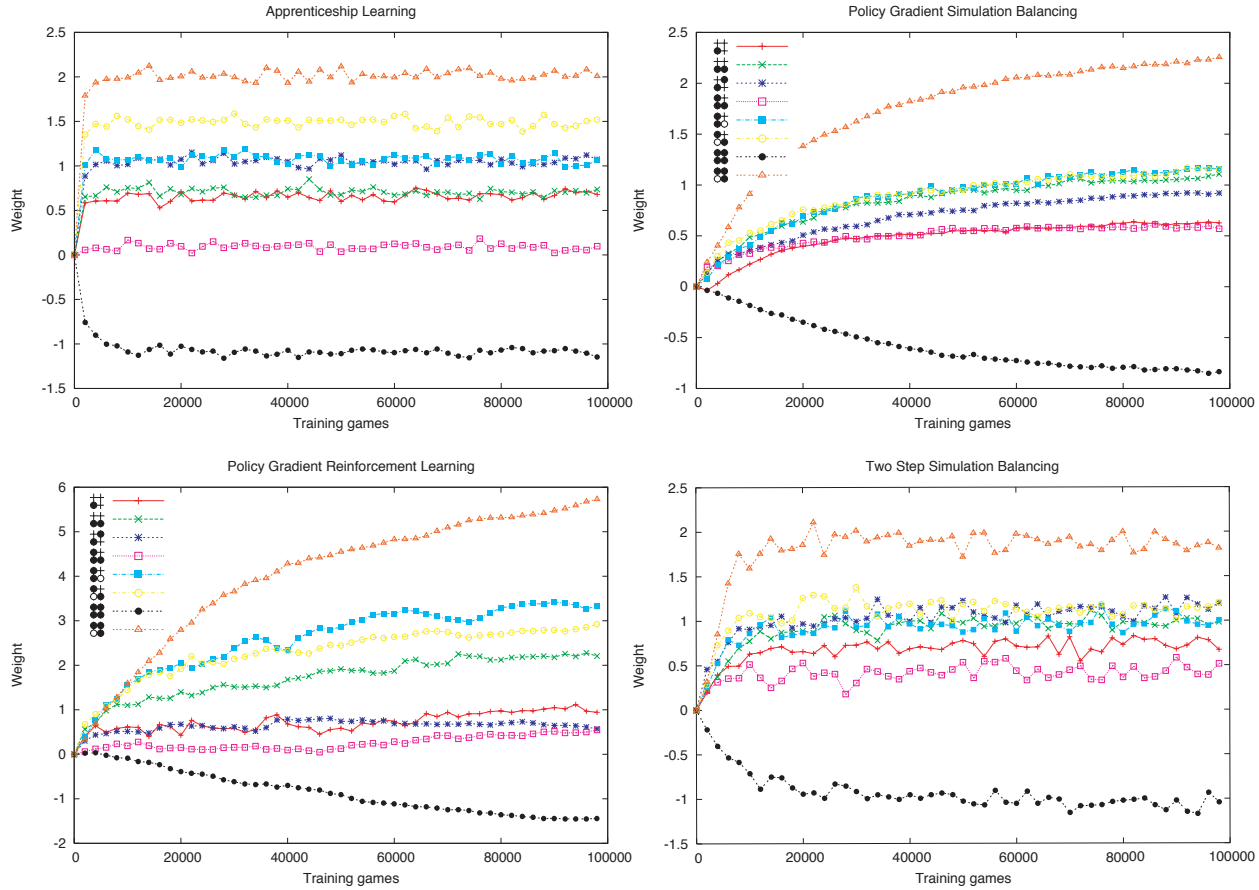

Figure 2. Weight evolution for the 2×2 local shape features: (top left) apprenticeship learning, (top right) policy gradient simulation balancing, (bottom left) policy gradient reinforcement learning, (bottom right) two-step simulation balancing.

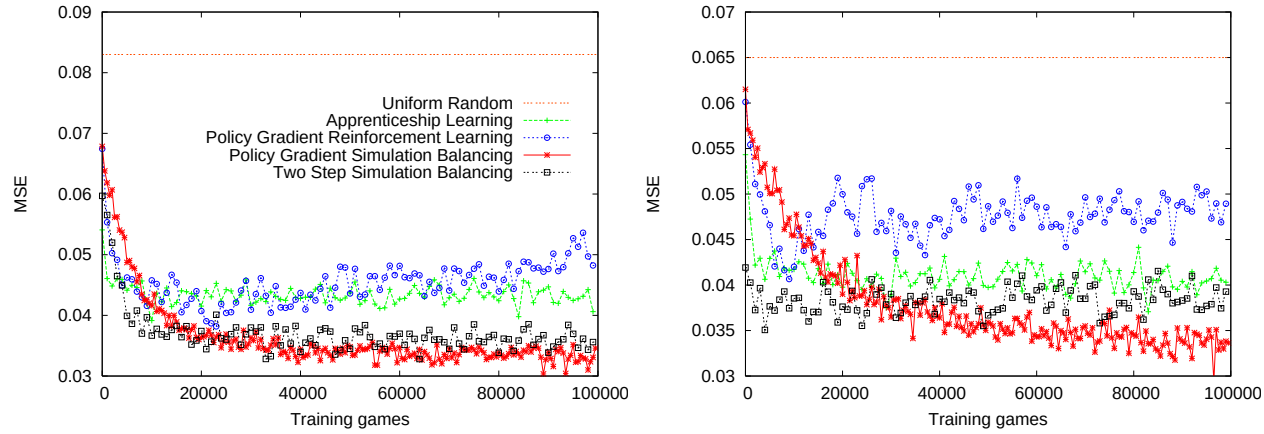

Figure 3. Monte-Carlo evaluation accuracy of different simulation policies in 5×5 Go (left) and 6×6 Go (right). Each point is the mean squared error over 1000 positions, between Monte-Carlo values from 100 simulations, and deep rollouts using the Go program *Fuego*.

We parameterise the softmax policy (Equation 5) with *local shape features* (Silver et al., 2007). Each of these features has a value of 1 if it matches a specific configuration of stones within a square region of the board, and 0 otherwise. The feature vector $\phi(s, a)$ contains local shape features for all possible configurations of stones, in all 1×1 and 2×2 squares of the board, for the position following action a in state s. Two different sets of symmetries are exploited by weight-sharing. *Location dependent* weights are shared between equivalent features, based on rotational, reflectional and colour inversion symmetries. *Location independent* weights are shared between these same symmetries, but also between translations of the same configuration. Combining both sets of weights results in 107 unique parameters for the simulation policy, each indicating a preference for a particular pattern.

6.1. Balance of shapes

We trained the simulation policy using 100000 training games of 5×5 Go, starting with initial weights of zero. The weights learnt by each algorithm are shown in Figure 3. All four algorithms converged on a stable solution. They quickly learnt to prefer capturing moves, represented by a positive preference for the location independent 1×1 feature, and to prefer central board intersections over edge and corner intersections, represented by the location dependent 2×2 features. Furthermore, all four algorithms learnt patterns that correspond to basic Go knowledge: e.g. the *turn* shape attained the highest preference, and the *dumpling* and *empty triangle* shapes attained the lowest preference.

In our experiments, the policy gradient reinforcement learning algorithm found the most deterministic policy, with a wide spectrum of weights. The apprenticeship learning algorithm converged particularly quickly, to a moderate level of determinism. The two simulation balancing algorithms found remarkably similar solutions, with the turn shape highly favoured, the dumpling shape highly disfavoured, and a stochastic balance of preferences over other shapes.

6.2. Mean squared error

We measured the accuracy of the simulation policies every 1000 training games by selecting 1000 random positions from an independent test-set, and performing a Monte-Carlo evaluation from 100 simulations. The mean squared error (MSE) of the Monte-Carlo values, compared to the deep search values, is shown in Figure 4, for 5×5 and 6×6 Go.

All four algorithms significantly reduced the evaluation error compared to the uniform random policy. The simulation balancing algorithms achieved the lowest error, with less than half the MSE of the uniform random policy. The reinforcement learning algorithm initially reduced the MSE, but then bounced after 20,000 steps and started to increase the evaluation error. This suggests that the simulation policy became too deterministic, specialising to weights that achieve maximum strength, rather than maintaining a good balance. The apprenticeship learning algorithm quickly learnt to reduce the error, but then converged on a solution with significantly higher MSE than the simulation balancing algorithms. Given a source of expert evaluations, this suggests that simulation balancing can make more effective use of this knowledge, in the context of Monte-Carlo simulation, than a supervised learning approach.

Table 1. Elo rating of simulation policies in 5×5 Go and 6×6 Go tournaments. The first column shows the performance of the simulation policy when used directly. The second column shows the performance of a simple Monte-Carlo search using the simulation policy.

	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
Fuego simulation policy	356	689	374	785

6.3. Performance in Monte-Carlo search

In our final experiment, we measured the performance of each learnt simulation policy in a Monte-Carlo search algorithm. We ran a tournament between players based on each simulation policy, consisting of at least 5000 matches for every player. Two players were included for each simulation policy: the first played moves directly according to the simulation policy; the second used 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 for the policy gradient reinforcement learning algorithm, firstly using the parameters that maximised performance (100k games of training), and secondly using the parameters that minimised MSE (20k and 10k games of training in 5×5 and 6×6 Go respectively). The results are shown in Table 1.

When the simulation policies were used directly, policy gradient RL (100k) was by far the strongest, around 200 Elo points stronger than simulation balancing[2].

[2]The Elo scale is a statistical rating system, such that a difference of 200 Elo corresponds to a 75% winning rate.

However, when used as a Monte-Carlo policy, simulation balancing was much stronger, 200 Elo points above policy gradient RL (100k), and almost 300 Elo stronger than apprenticeship learning.

The two simulation balancing algorithms achieved similar performance in 5×5 Go, suggesting that it suffices to balance the errors from consecutive moves, and that there is little to be gained by balancing complete simulations. However, in the more complex game of 6×6 Go, Monte-Carlo simulation balancing performed significantly better than two-step simulation balancing.

Finally, we compared the performance of our Monte-Carlo search to GnuGo, a deterministic Go program with sophisticated, handcrafted knowledge and specialised search algorithms. Using the policy learnt by simulation balancing, our simple one-ply search algorithm achieved comparable strength to GnuGo. In addition, we compared the performance of the *Fuego* simulation policy, which is based on the well-known *MoGo* patterns, handcrafted for Monte-Carlo search on larger boards. Surprisingly, the Fuego simulation policy performed poorly, suggesting that handcrafted patterns do not generalise well to different board sizes.

7. Conclusions

We have presented a new paradigm for simulation balancing in Monte-Carlo search. Unlike supervised learning and reinforcement learning approaches, our algorithms can balance the level of stochasticity to an appropriate level for Monte-Carlo search. They are able to exploit deep search values more effectively than supervised learning methods, and they maximise a more relevant objective function than reinforcement learning methods. Unlike hill-climbing or handcrafted trial and error, our algorithms are based on an analytical gradient based only on the current position, allowing parameters to be updated with minimal computation. Finally, we have demonstrated that our algorithms outperform prior methods in small board Go.

We are currently investigating methods for scaling up the simulation balancing paradigm both to larger domains, using *actor-critic* methods to reduce the variance of the policy gradient estimate; and to more sophisticated Monte-Carlo search algorithms, such as UCT (Kocsis & Szepesvari, 2006). In complex domains, the quality of the minimax approximation $\hat{V}^*(s)$ can affect the overall solution. One natural idea is to use the learned simulation policy in Monte-Carlo search, and generate new deep search values, in an iterative cycle.

One advantage of apprenticeship learning over simulation balancing is that it optimises a convex objective function. This suggests that the two methods could be combined: first using apprenticeship learning to find a global optimum, and then applying simulation balancing to find a local, balanced optimum.

For clarity of presentation we have focused on deterministic two-player games with terminal outcomes. However, all of our algorithms generalise directly to stochastic environments and intermediate rewards.

References

Billings, D., Castillo, L., Schaeffer, J., & Szafron, D. (1999). Using probabilistic knowledge and simulation to play poker. *Proceedings of the 16th National Conference on Artificial Intelligence* (pp. 697–703).

Chaslot, G., Winands, M., Szita, I., & van den Herik, H. (2008). Parameter tuning by the cross-entropy method. *8th European Workshop on Reinforcement Learning*.

Coulom, R. (2007). Computing Elo ratings of move patterns in the game of Go. *Computer Games Workshop*.

Finnsson, H., & Björnsson, Y. (2008). Simulation-based approach to general game playing. *23rd Conference on Artificial Intelligence* (pp. 259–264).

Gelly, S., & Silver, D. (2007). Combining online and offline learning in UCT. *17th International Conference on Machine Learning* (pp. 273–280).

Gelly, S., Wang, Y., Munos, R., & Teytaud, O. (2006). *Modification of UCT with patterns in Monte-Carlo Go* (Technical Report 6062). INRIA.

Kocsis, L., & Szepesvari, C. (2006). Bandit based Monte-Carlo planning. *15th European Conference on Machine Learning* (pp. 282–293).

Sheppard, B. (2002). World-championship-caliber Scrabble. *Artificial Intelligence*, *134*, 241–275.

Silver, D., Sutton, R., & Müller, M. (2007). Reinforcement learning of local shape in the game of Go. *20th International Joint Conference on Artificial Intelligence* (pp. 1053–1058).

Tesauro, G., & Galperin, G. (1996). On-line policy improvement using Monte-Carlo search. *Advances in Neural Information Processing 9* (pp. 1068–1074).

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