Mach Learn (2006) 63:249–286
DOI 10.1007/s10994-006-6888-8

Universal parameter optimisation in games based on SPSA

Levente Kocsis · Csaba Szepesvári

Received: February 11, 2005 / Revised: September 12, 2005 / Accepted: December 29, 2005 / Published online: 28 March 2006

Abstract Most game programs have a large number of parameters that are crucial for their performance. While tuning these parameters by hand is rather difficult, efficient and easy to use generic automatic parameter optimisation algorithms are known only for special problems such as the adjustment of the parameters of an evaluation function. The SPSA algorithm (Simultaneous Perturbation Stochastic Approximation) is a generic stochastic gradient method for optimising an objective function when an analytic expression of the gradient is not available, a frequent case in game programs. Further, SPSA in its canonical form is very easy to implement. As such, it is an attractive choice for parameter optimisation in game programs, both due to its generality and simplicity. The goal of this paper is twofold: (i) to introduce SPSA for the game programming community by putting it into a game-programming perspective, and (ii) to propose and discuss several methods that can be used to enhance the performance of SPSA. These methods include using common random numbers and antithetic variables, a combination of SPSA with RPROP, and the reuse of samples of previous performance evaluations. SPSA with the proposed enhancements was tested in some large-scale experiments on tuning the parameters of an opponent model, a policy and an evaluation function in our poker program, McRaise. Whilst SPSA with no enhancements failed to make progress using the allocated resources, SPSA with the enhancements proved to be competitive with other methods, including TD-learning; increasing the average payoff per game by as large as 0.19 times the size of the amount of the small bet. From the experimental study, we conclude that the use of an appropriately enhanced variant of SPSA for the optimisation of game program parameters is a viable approach, especially if no good alternative exist for the types of parameters considered.

Keywords SPSA · Stochastic gradient ascent · Games · Learning · Poker

Editors: Michael Bowling · Johannes Fürnkranz · Thore Graepel · Ron Musick

L. Kocsis · C. Szepesvári (✉)
MTA SZTAKI, Kende u. 13–17, Budapest, Hungary-1111
e-mail: szcsaba @sztaki.hu

L. Kocsis
e-mail: kocsis@sztaki.hu

1. Introduction

Any reasonable game program has several hundreds or thousands of parameters. These parameters belong to various components of the program, such as the evaluation function or the search algorithm. The performance of most game program depends strongly on the settings of its parameters. While humans can make educated guesses about "good" parameter values in certain cases, the hand-tuning of the parameters is a difficult, cumbersome and rather time-consuming task. An alternative approach is to find the "right" values of these parameters by means of some automatic parameter tuning algorithm.

The use of parameter optimisation methods for the performance tuning of game programs is made difficult by the fact that the objective function is rarely available analytically, hence those methods that rely on the availability of an analytic form of the gradient cannot be used. The reasons for the lack of the knowledge of such an analytic form can be multiple. First, game program designs rarely take into account the needs of parameter optimisation. As a result parameters are often buried deeply into the code and the functional form of the performance measure as a function of the parameters (if differentiable at all) will rarely, if ever, be available analytically. Further, game designs evolve constantly and thus keeping the analytic form of the gradient up-to-date might be prohibitively expensive.

As an example consider the problem of optimising parameters of some video game. For the sake of specificity, consider MMORPGs. In an MMORPG a reasonable objective is to look for parameters that achieve balance, i.e., ensuring that all character classes or skill sets have roughly the same 'power'. Formally, the optimisation problem could be set up as the task of maximising $f(\theta) = -d(p(\theta), u)$, where $p(\theta) = (p_1(\theta), \ldots, p_N(\theta))^T$ is a vector whose ith component gives the probability that character i wins given parameters θ, N is the number of character classes, $u = (1/N, \ldots, 1/N)$ is the uniform distribution and d is an appropriate distance function. The numerous parameters include strength, speed, physical attacks, physical defenses, magic attacks, magic defenses, various powerups and others. Unlike card and board games, the rules of the game for an MMORPG change constantly until the game is shipped, leaving little chance to the derivation of an analytic expression for the gradient of the objective function.

Optimisation in card and board games, on the other hand, has its own problems. Consider, for example, the problem of tuning search-control parameters in board games. Certainly, such parameters may have a strong influence on the performance of the program, hence their tuning is critical for creating a strong player. Although these parameters are often real-valued, the resulting objective function is hard to express analytically and, oftentimes, it is not even differentiable. Another difficulty may arise in games when action selection is based on some Monte-Carlo computations. In this case, even when the objective function is differentiable, its analytic form, though it might be available in a closed form, can be computationally intractable.

In order to facilitate the forthcoming discussion, let us formalise some common properties of those game-parameter optimisation problems that we are interested in here. We postulate two assumptions. The first is that the objective function is given in the form of the expected value of some random payoff, or reward, whose distribution depends on the parameters to be optimised:

$$f(\theta) = \mathbb{E}\left[R(\theta)\right].$$

Second, we assume that by means of running some computer simulations it is possible to generate independent random realisations of $R(\theta)$.

Optimisation problems with the above characteristics are the subject of *simulation optimisation*. The best known tools for simulation optimisation are infinitesimal perturbation analysis (IPA), sample path optimisation and the likelihood ratio (LR) method (e.g., Andradóttir, 1998). IPA and LR are different ways to get unbiased estimates of the gradient using a simple simulation run, which in turn can be used in a stochastic gradient ascent (SGA) procedure. When IPA methods are applicable they often yield better results than LR methods. Unfortunately, this class is rather limited. We note that policy-gradient methods (Williams, 1992) developed in the reinforcement learning (RL) community are in fact specialised LR methods (Baxter and Bartlett, 2001).

Many problems in games can be posed as the task of controlling the learner's environment in an optimal manner. These problems can be attacked by value-functions based methods (Sutton and Barto, 1998). State-value functions assign a value to each state which is then used e.g. in 1-ply search to select the best controls (actions). Given an action-value function, on the other hand, control is generated by selecting the action with the highest value. Temporal-difference methods, one cornerstone of RL, are designed for learning (optimal) value functions (Sutton, 1988).[1] Note that TD-methods have their own objective function, separate from $f(\theta)$. It follows that it is only when special conditions are met that TD-methods automatically give rise to solutions that maximise f. However, when such conditions are met, TD-methods are likely to be the method of choice (Sutton and Barto, 1998).

Interest in policy-gradient methods have increased recently in the RL community as researchers try to extend the scope of current algorithms. As mentioned before, policy-gradient methods are ultimately based on the LR method which, unfortunately, without no further structural assumptions is well-known to scale badly with the length of the simulated trajectories (the variance of the gradient estimates is huge). Hence, the central topic of research on policy-gradient (and LR) methods is to design methods that achieve variance reduction (Kakade & Langford, 2002) provides some examples when "pure" policy-gradient methods are likely to fail). Recently, Baird and Moore (1999) and Sutton et al. (2000) proposed ways to combine value-function and policy-gradient methods. In particular, Baird and Moore (1999) demonstrates by means of an example that the combined algorithm (which they call VAPS) may have a considerable edge over methods that are either entirely value-function based or use policy gradient alone. By analysing the variance of the gradient estimates, Greensmith, Bartlett and Baxter (2002) find also that using value-function estimates in the gradient-estimates can indeed be beneficial. We know of no systematic comparison of pure value-based (i.e., when the value-function is estimated using TD-methods and the policy is defined purely from the estimated value-function) and policy-gradient methods.

Returning to parameter optimisation in games, probably the best known success story in learning to play uses TD-methods: Tesauro's TD-Gammon program was trained by TD(λ) and self-play where a simple extension of the above outlined greedy-action selection was used. TD-Gammon achieved a level of play that exceeds that of the best human players (Tesauro, 1992). Another example that uses the value-function based approach is KnightCap, where a variant of TD(λ) was used to tune the parameters of the evaluation function of a chess program (Baxter, Tridgell & Weaver, 2000).

As mentioned previously, LR-methods are potentially advantageous to value-based methods as they optimise for the objective function of interest directly. Despite this, experience with using policy-gradient methods in real-world games to date is scarce.

[1] From our perspective, we call a value function optimal if it gives rise to optimal behaviour when control is generated by the above described methods.

Bowling and Veloso (2002) demonstrated that the algorithm due to Sutton et al. (2000) where value-functions were trained by Sarsa(0) "can learn" in the card-game Goofspiel.

When the objective function is non-differentiable or an analytic form of the gradient is unavailable then one possibility is to resort to some general-purpose search procedure. Examples of such methods used in game parameter optimisation include the work of Kocsis (2003) who considered several algorithms for parameter tuning, some of them being adaptations of general-purpose gradient-free algorithms (like tabu-search), whilst others were specifically designed to exploit the properties of the particular optimisation problems. Another example is the work of Chellapilla and Fogel (1999) who used genetic algorithms to evolve neural-networks to play the game of checkers. Tuning of search extension parameters in game-tree search was considered by Björnsson and Marsland (2003), who suggested an algorithm that we might view as a variant of the Finite-Difference Stochastic Approximations (FDSA), an algorithm that we will consider later. Despite all the work done so far, we think that automatic tuning of game program parameters remains a largely unexplored area of game development.

The purpose of this article is to investigate the use of SPSA (Simultaneous Perturbation Stochastic Approximation), a gradient-free stochastic hill-climbing algorithm, for tuning the parameters of game programs. Being a gradient-free method that is extremely easy to implement, SPSA is an appealing choice, especially if a tractable analytic form of the objective function's gradient is not available. Unfortunately, in difficult problems such as, e.g., when the objective function is observed in heavy noise, basic SPSA has little chance for producing acceptable solutions or even just making progress. Hence, we propose and discuss several methods to enhance SPSA.

In order to test the limits of the enhanced SPSA, we chose Omaha Hi-Lo Poker, a game where payoffs are wildly random, as our test domain. Omaha Hi-Lo Poker is one of the most complex poker variants, at least with regard to state-space complexity. A poker playing program, McRaise was developed and the optimisation of various parts of it were considered using the enhanced SPSA methods. Although McRaise is still in the development phase and its performance has not been tested extensively yet, preliminary results suggests that the program is much stronger than amateur players, and is probably competitive with professional players as well.

We have found that parameters obtained by the enhanced SPSA yield programs that are competitive or better in their strength than those whose parameters were obtained by other methods, including TD-methods whilst utilising equivalent computational resources. The enhancements include using common random numbers, antithetic variables, deck reuse and combining SPSA and RPROP ("resilient backpropagation") for adaptive step-size scheduling. RPROP, originally developed for the batch training of neural networks by Riedmiller and Braun (1993), employs local (componentwise) step-size adaptation and a sign-based scheme to eliminate possible harmful influences of the derivatives' magnitude on the weight updates.

The article is organised as follows: In Section 2 we give some background on stochastic gradient based optimisation. In particular, first we consider methods that assume that an analytic expression for the gradient is available: we describe the basic Robbins-Monro procedure, followed by the description of the LR method. Next, we consider two gradient-free methods, FDSA and SPSA. This section is closed by discussing the relative efficiencies of these methods and the description of the method of common random numbers that can make them competitive in terms of their asymptotic convergence-rates with gradient-based methods. We also discuss options for parameter optimisation when the parameters are discrete-valued or when the objective function is non-differentiable.

Section 3 is devoted to the description of some further methods that we propose to enhance the performance of SPSA. First, we show that when the number of simulations for estimating

the gradient is fixed then the best estimate of the gradient in terms of the estimate's variance is given when the number of perturbations is kept at maximum. Next, we describe RPROP and how it is combined with SPSA, along with the potential advantages and disadvantages of the proposed combination. A small numerical study is used to illustrate the point that the combined algorithm can be more efficient than the canonical version of SPSA. Experiments on our real-world game program, McRaise are described in Sections 4 and 5. In particular, the rules of Omaha Hi-Lo Poker and the algorithms underlying our game program, McRaise, are described in Section 4, whilst experimental results with RSPSA are given in Section 5. Here RSPSA is compared with (R)FDSA and some TD-methods. Finally, we draw our conclusions and discuss future work in Section 6.

2. Parameter optimisation via stochastic gradient ascent

2.1. Basic setup

The purpose of this section is to give a brief review of stochastic gradient ascent methods and, in particular, introduce the SPSA algorithm. Consider the task of finding a maximiser $\theta^* \in \mathbb{R}^d$ of some real valued function $f = f(\theta)$. In our case f may measure the performance of a player in some environment (e.g., against a fixed set of opponents), or it may represent an auxiliary performance index of interest that is used internally in the algorithm such that a higher value of it might ultimately yield a better play. In all cases θ represents some parameters of the game program.

We assume that the algorithm, whose task is to tune the parameters θ, can query the value of f at any point θ, but the value received will be corrupted by noise. This value is typically obtained by simulations that involve randomness. In particular, the randomness can originate from randomised decisions of the players or from the randomness of the environment. In a card playing game the randomness of cards represents a substantial source of randomness of the winning chance of a strategy, though, obviously the randomness of the players' actions will also influence it. We shall assume that the value observed in the tth step of the algorithm when the simulation is run with parameter θ_t is given by $f(\theta_t; Y_t)$, where Y_t is some random variable such that the expected value of $f(\theta_t; Y_t)$, conditioned on θ_t and given all past information equals $f(\theta_t)$:

$$f(\theta_t) = \mathbb{E}\left[f(\theta_t; Y_t) \,|\, \theta_t, \mathcal{F}_t \right].$$

Here $\mathcal{F}_t$ is the sigma-field generated by $Y_0, Y_1, \ldots, Y_{t-1}$ and the previous parameter values $\theta_0, \theta_1, \ldots, \theta_{t-1}$. The variables Y_t represent the randomness involved of the simulations. We shall call Y_t the *simulation noise*.

Stochastic gradient ascent (SGA) algorithms work by changing the parameter θ in a gradual manner so as to increase the value of f on average:

$$\theta_{t+1} = \theta_t + \alpha_t \hat{g}_t(\theta_t). \qquad (1)$$

Here θ_t is the estimate of θ^* in the tth iteration (time step), $\alpha_t \geq 0$ is a learning-rate or step-size parameter that governs the size of the changes to the parameters and $\hat{g}_t(\theta_t)$ is some approximation to the gradient of f. A typical assumption is that the expected value of $\hat{g}_t(\theta_t)$ given past data is equal to the gradient $g(\theta) = \partial f(\theta)/\partial\theta$ of f. Then, under some additional regularity assumptions on the noise sequence $\{Y_t\}$ and f, and if α_t converges to zero at

an appropriate rate (in particular, if $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$) then the parameter sequence $\{\theta_t\}$ converges to a local maximum of f with probability one.

2.2. The Robbins-Monro algorithm

In the classical Robbins-Monro setup the main assumption is that direct unbiased estimates of the gradient are available (Robbins & Monro, 1951). Consider the simplest case first, when the random variables Y_t entering f are independent, identically distributed and their common underlying distribution is independent of θ. Then, the gradient of $f(\theta) = \mathbb{E}[f(\theta; Y_t)]$ can be computed as

$$\hat{g}_t(\theta_t) = \partial f(\theta_t; Y_t)/\partial\theta, \qquad (2)$$

provided that expectation and differentiation can be exchanged and if the partial derivative of f w.r.t. θ exists. In simulation optimisation $\hat{g}_t$ as defined by Eq. (2) is called the IPA (infinitesimal perturbation analysis) gradient estimate at θ_t.

When simulation noise depends on θ then the computation of the gradient is less straightforward.[2] Assuming that the distribution of Y_t for any given θ admits some density, p_θ, with respect to some fixed, θ-independent measure, ν. Then $f(\theta)$ can be written as

$$f(\theta) = \int f(\theta; y) p_\theta(y)\, d\nu(y). \qquad (3)$$

In the special case when f in the integral does not depend on θ (e.g., f is a sort-of performance measure that depends only on the outcomes of the game) and provided that integration and differentiation can be exchanged, the gradient of f can be expressed as

$$f'(\theta) = \int f(y) \frac{d}{d\theta} (\ln p_\theta(y)) p_\theta(y)\, d\nu(y).$$

Here $s(\theta) = \frac{d}{d\theta}(\ln p_\theta(y))$ is called the *score function* corresponding to p_θ. An alternative form of the above equation is given by

$$f'(\theta) = \mathbb{E}\left[f(Y) \frac{d}{d\theta} (\ln p_\theta(Y)) \right].$$

Hence, $f(Y)(d/d\theta)(\ln p_\theta(Y))$ provides an unbiased estimate of the gradient. The method just described is called the *likelihood-ratio* (or score function) method in the simulation optimisation literature (Andradóttir, 1998; Spall, 2003). Policy-gradient methods that became popular recently in the RL community belong to this class. In RL p_θ often involves unknown terms; a case that arises, e.g., when Y is obtained through interacting with an *unknown* environment (Williams, 1992). A crucial observation is that despite this, the score function can still be expressed with known quantities: the derivative of the logarithm of the product of action-selection probabilities (actions serve as the input to the environment). Hence policy-gradient methods require the knowledge of (only) the analytic forms of the action-selection

[2] Note that usually there exist multiple ways to write the objective function; some forms may make the simulation noise dependent, whilst others may make it independent of the optimised parameters. Which form to use is then a non-trivial design question.

probabilities. Although, in some cases it is easy to satisfy this requirement in other cases, such as when action selection is based on random search trees, the analytic form of action-selection probabilities will be intractable. In fact, this is the case for our poker playing program that basis its action selection on evaluating the available actions by computing their values with a Monte-Carlo procedure (cf. Section 5.1).

Another difficult case for LR methods is when Y represents a long random trajectory. In this case p_θ becomes a product of a large number of terms and hence the variance of LR-based the gradient estimates, $f(Y)(d/d\theta)(\ln p_\theta(Y))$, will become huge. This problem was well-known since the early stages of simulation optimisation and it is the subject of active research in both the simulation optimisation and RL communities (see (Greensmith, Bartlett and Baxter , 2002; Spall, 2003) and the references therein).[3]

2.3. The FDSA algorithm

An alternative to estimating the gradient by exploiting the structure of the optimisation problem is to use methods that do not require the knowledge of the analytic form of f. The earliest procedure that does not require an analytic expression for the gradient is the Kiefer-Wolfowitz procedure (Kiefer & Wolfowitz, 1952). Originally, this algorithm was introduced for one-dimensional problems (i.e., when $d = 1$). The multi-dimensional version, due to Blum (1954), is called the finite difference stochastic approximation (FDSA) procedure and works by approximating the ith component $g_i(\theta)$ of the gradient of f with the following two-sided difference:

$$\hat{g}_{ti}(\theta_t) = \frac{f(\theta_t + c_t e_i; Y_{ti}^+) - f(\theta_t - c_t e_i; Y_{ti}^-)}{2c_t}. \qquad (4)$$

Here e_i represents the ith unit vector in the d-dimensional Euclidean space and c_t is step-size. Since for any positive c_t, the approximation has a non-zero "bias", in order to make the associated SGA procedure convergent c_t must converge to zero at an appropriate rate (such as, e.g., $c_t \propto t^{-1/6}$). In the above equation Y_{ti}^+ and Y_{ti}^- are random variables, independent of each other. When the distribution of the simulation noise, Y_t, depends on the parameter vector then Y_{ti}^+ is obtained by running the simulation with parameters $\theta + c_t e_i$, whilst Y_{ti}^- is obtained by running the simulation with parameters $\theta - c_t e_i$. It can be readily observed that computing an estimate of the full gradient of f via Eq. (4) requires $2\,d$ evaluations of the target function f.

2.4. SPSA

Simultaneous Perturbation Stochastic Approximation (SPSA) is a recently proposed alternative to FDSA. The main observation leading to SPSA is that FDSA may spend too much effort on getting an approximation of all the components of the gradient. When measurements are noisy, almost the same approximation accuracy to the gradient can be obtained by just considering two two-sided perturbations of the parameter vector. In order to represent all directions equally well, the perturbation vector is chosen to be a random vector. Let $\Delta_t \in \mathbb{R}^d$ be

[3] Infinitesimal perturbation analysis (IPA) generally does not suffer from this problem. However, IPA methods are generally unsuitable for parameter optimisation in games, as noted previously.

this perturbation vector. Then the SPSA based estimate of the ith component of the gradient is given by

$$\hat{g}_{ti}(\theta_t) = \frac{f\left(\theta_t + c_t\Delta_t; Y_t^+\right) - f\left(\theta_t - c_t\Delta_t; Y_t^-\right)}{2c_t\Delta_{ti}}. \qquad (5)$$

Note that the numerator of this expression does not depend on index i and hence SPSA requires running two simulations only to estimate the gradient. Despite this, SPSA provides a relatively good approximation to the gradient. In particular, the following results were proved by Spall (1992). Assume that the followings holds for any finite t:

(A1) the random perturbations Δ_t are independent of the past of the process,
(A2) $\{\Delta_{ti}\}_i$ is an i.i.d. sequence,
(A3) the distribution of Δ_{ti} is symmetric around zero,
(A4) $|\Delta_{ti}|$ is bounded with probability one, and
(A5) $\mathbb{E}[\Delta_{ti}^{-1}]$ is finite.

Then, assuming that f is sufficiently smooth, it holds that the bias of estimating the gradient, $g(\theta_t)$, by $\hat{g}_t(\theta_t)$ is of order $O(c_t^2)$. Further, if the step-sizes satisfy $\alpha_t, c_t > 0$, $\lim_{t\to\infty} c_t = 0$, $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2/c_t^2 < \infty$ then the associated gradient ascent procedure converges to a local optimum of f with probability one (Spall, 1992).[4]

A simple way to satisfy the conditions on Δ_t is to choose its components to be independent, ± 1-valued, unbiased, Bernoulli-distributed random variables. Under certain conditions on the objective function, f, this choice was claimed to be optimal in an asymptotic sense under the mean square error and probability criteria (Sadegh & Spall, 1997). By means of a heuristic argument, Spall also concludes that under 'reasonably general conditions' SPSA needs d times less evaluations of the objective function than FDSA (d is the dimensionality of the parameter space) to achieve the same asymptotic statistical accuracy (Spall, 1992). This claim was also backed up by some simulations in the same article.

The above claims are *asymptotic* by their nature. In this paper, on the other hand, we are primarily interested in the *transient* behaviour of the algorithms. This is because we do not expect them to converge within the allocated time-frame to a small neighbourhood of a stationary point, due to the inherent complexity of the optimisation task and the long running times that are typical for game-simulations.

In the transient regime, according to some authors, FDSA might actually perform better than SPSA. For example, Dippon (2003) writes that "although randomized and/or higher order methods can outperform the standard methods in their asymptotic behavior, it is to be expected that for a small or moderate number n of iteration steps the standard methods may be superior" (the standard method means FDSA here). Also, Kushner and Yin (1997, pp. 318) discusses at some length the non-asymptotic properties of some deterministic methods in comparison to their randomised counterparts.

Another interesting feature of SPSA is that it is known to converge even when the objective function is non-differentiable (He, Fu & Marcus, 2003). For parameter optimisation in games when the parameters to be optimised are discrete-valued, a variant of SPSA due

[4] The independence assumptions on the components of Δ_t make SPSA fundamentally different from the method of random directions stochastic approximation, RDSA, where the perturbation vector is sampled uniformly in the d-dimensional unit sphere and the two-sided differences are multiplied by Δ_{ti}, unlike in SPSA where they are multiplied by Δ_{ti}^{-1}.

to Gerencsér, Hill and Vágó (1999) could be used, though the analysis given by the authors there is limited to convex functions (just like for the previously cited work). Actually, in games defined over discrete structures, often the parameters are themselves continuous-valued, but the objective function is discontinuous, e.g., piecewise constant. This is the case for some of the objective functions in the poker domain that we consider later in this paper. One option then is to use the noise injection method due to Gerencsér, Kozmann and Vágó (1998).

Formally, if $f(\theta)$ is the objective function then instead of optimising f, the function $\bar{f}(\theta) = \mathbb{E}[f(\theta + U)]$ is optimised. Here U is a random variable that is assumed to admit a continuous density. The advantage of noise-injection is that the smoothness properties of the smoothed objective function will depend on the smoothness of the density underlying U only. Although this method alters the objective function, this change can be made as small as desired by making U "small". When the measurement noise of f (or that of the gradient) is big then one expects that the noise injected would have negligible effect on the finite-sample behaviour. Hence, in our experiments we did not care to inject noise.

2.5. Efficiency

It is well known that methods that rely on an analytic form of the gradient can converge substantially faster than either FDSA or SPSA. Asymptotic convergence rate results provide a firm theoretical background to this observation: the Robbins-Monro procedure converges at the rate $O(t^{-1/2})$, whilst SPSA and FDSA both converge at the rate $O(t^{-1/3})$ provided that f is three times differentiable and a decreasing learning rate of $\alpha_t = \alpha/t$ is used with a sufficiently large $\alpha > 0$. In fact, Chen (1988) and Polyak and Tsybakov (1990) showed that for randomised Kiefer-Wolfowitz procedures and the class of p-times differentiable objective functions with $p \geq 2$, the optimal rate in the minimax sense is $O(t^{-(p-1)/(2p)})$. Hence, for $p = 3$, SPSA, FDSA and, by a recent result due to Dippon (2003), also RDSA attain the optimal rate of convergence.

Besides the asymptotic rate of convergence, the asymptotic variance of the appropriately normalised mean square error is also of considerable interest. A result of Fabian (1968) can be used to show that this asymptotic variance is proportional to the variance of gradient estimates. Hence methods that reduce the variance of the gradient's estimate have the potential to improve the rate of convergence to θ^*.

When the objective function is evaluated by means of running some computer simulation then it has been observed that the method of *Common Random Numbers* (CRN) can be used to decrease the variance of the gradient's estimate (Glasserman & Yao, 1992; L'Ecuyer & Yin, 1998; Kleinman, Spall & Neiman, 1999). In fact, if this method is employed, the *convergence rate is improved to* $O(t^{-1/2})$. This was shown for FDSA by Glasserman and Yao (1992) and L'Ecuyer and Yin (1998) and later extended to SPSA by Kleinman, Spall and Neiman (1999).

The basic observation that leads to CRNs is that in both FDSA and SPSA the ith gradient is estimated as the difference $\Delta F_i = F_i^+ - F_i^-$. Since

$$\mathrm{Var}(F_i^+ - F_i^-) = \mathrm{Var}(F_i^+) + \mathrm{Var}(F_i^-) - 2\mathrm{Cov}(F_i^+, F_i^-),$$

there is an opportunity to decrease the variance of the difference Δ_i by increasing the covariance of F_i^+ and F_i^-, provided that the change does not alter the variance of F_i^+ and F_i^-. In

the case of SPSA $F_i^{\pm} \propto f(\theta \pm c\Delta; Y^{\pm})$ and so when Y^+ and Y^- are independent, we see[5]. that

$$\mathrm{Cov}\,(F_i^+, F_i^-) = 0.$$

The same equality holds for FDSA. Hence, if $F_i^{\pm}$ are redefined to depend on the same random value Y: $F_i^{\pm} \propto f(\theta \pm c\Delta; Y)$, then the variance of $F_i^+ - F_i^-$ will decrease when $\mathrm{Cov}\,(f(\theta + c\Delta; Y), f(\theta - c\Delta; Y)) > 0$. The larger this covariance is, the larger the decrease of the variance of the estimate of the gradient will be. A stronger statement giving conditions when variance is *minimised* under CRNs was given by Rubinstein, Samorodnitsky and Shaked (1985) (see also, Spall, 2003, Proposition 14.2).

In order to gain some further insight of how CRNs work, consider a simple example when f is given by $f(\theta; Y) = \theta Y$. In this case, $(f(\theta + c\Delta; Y) - f(\theta - c\Delta; Y))/(2c\Delta) = Y$. Hence, denoting the variance of Y by V, we get that the variance of the estimate that uses CRNs is V. On the other hand, the variance of $(f(\theta + c\Delta; Y^+) - f(\theta - c\Delta; Y^-))/(2c\Delta)$ (conditioned on Δ) is equal to $(\theta^2/(2c^2\Delta^2) + 1/2)V$. Hence, in this second case variance grows as c approaches zero,[6] whilst in the first case it stays bounded independent of c.

When samples of f are obtained by means of some simulations that use pseudorandom numbers and if the distribution of Y is independent of the parameter vector to be optimised, then $F_i^{\pm} = f(\theta \pm c\Delta; Y)/(2c\Delta_i)$ can be computed by saving the seed of the random number generator (RNG) before running the simulation to compute F_i^-, and then resetting the seed to the saved value before computing F_i^+. When the distribution of the simulation noise, Y, is not independent of the parameter vector, but can be separated into two parts so that the distribution of the variables in the first part is independent of θ, whilst the distribution of the variables that belong to the second part depends on θ, then it is recommended to use two independent RNGs. The first RNG is used with previous procedure to produce samples from variables of the first kind, whilst the second RNG should be used in computing the values of the random variables of the second type. This will still yield some variance reduction. The technique just described is termed *the method of Partial Common Random Numbers* (PCRN). Some experimental results comparing SPSA and FDSA with and without (P)CRN are given by Kleinman, Spall and Neiman (1999).

In order to see how PCRNs might be used in practice, consider a card game. Assume that the parameters of a player's strategy are to be optimised. The random deck (causing typically a large proportion of the variance of the payoffs) can be viewed as a variable of the first type, whilst actions generated and situations encountered by the players during a game can be viewed as random variables of the second kind. If the action-selection procedure is near-deterministic then even the action-situation trajectories generated will often remain aligned, resulting possibly in a further reduction of variance.

Note, however, that often it is possible to use CRNs (and eliminate non-aligned variables) by changing the way randomness in the simulations is modelled. Actually, as it turns out, CRNs can be used in a surprisingly large class of problems. To see how this is done, consider imperfect information, alternating Markov games with the total reward criterion.

[5] Here, expressions involving expectations are conditioned on Δ. In fact, with $Z = F_i^+ - F_i^-$, the identity $\mathrm{Var}\,(Z) = \mathbb{E}\,[\mathrm{Var}\,(Z|\Delta)] + \mathrm{Var}\,(\mathbb{E}\,[Z|\Delta])$ can be used to complete the argument. Here $\mathrm{Var}\,(Z|\Delta) = \mathbb{E}\,[(Z - \mathbb{E}\,[Z|\Delta])^2|\Delta]$

[6] Because $\alpha_t \to 0$ sufficiently fast, the SPSA iterate converges despite that the variance of the gradient estimate diverges as $O(c_t^{-2})$.

Assume for the sake of simplicity that rewards are deterministic and that a game lasts for at most T steps. Certainly many card and board games fit this description, including poker. Consider optimising the payoff of Player 1 as a function of some parameters, θ, of its policy. Let $Y = (U_1, \ldots, U_T, U'_1, \ldots, U'_T)$ be a collection of independent, uniformly distributed random variables. It should be obvious that the payoff of Player 1 can be written in the form $f(\theta; Y)$, though f, in general, will be a very complicated function of its arguments.[7] What is important for us, however, is that since distribution of Y is independent of θ, the CRN method applies. Note also that more often than not, $f(\theta; Y)$ will not be differentiable w.r.t. θ, whilst $f(\theta) = \mathbb{E}[f(\theta; Y)]$ will be (e.g., LR-based gradient-estimates, based on an alternative representation, can be computed). We note in passing that there are cases when the variance of the SPSA-difference based gradient estimate will be smaller than that of the LR-based gradient estimate, showing that there is no easy rule of thumb to decide if LR-based or SPSA based methods should be used (Spall, 2003, pp. 420 gives a related example).

The (P)CRN method is not the only way to improve the performance of SPSA. In what follows we will describe several other techniques that are potentially useful for decreasing the variance of gradient estimates and hence for increasing the convergence rate. We note that antithetic random variables work on principles similar to those underlying CRNs. We shall discuss the relationship of CRNs and antithetic variables in Section 3.3.

3. SPSA implementation issues

In this section we consider several ways to enhance the performance of SPSA. In particular, in the next section (Section 3.1) we consider how to select the number of evaluations per perturbation. An expression for the mean square error is derived that will be used to show that, in fact, a single evaluation per perturbation is preferable. Next, in Section 3.2, we introduce RPROP, a method that is known to enhance the performance of training neural networks. Here we propose a way to combine SPSA with RPROP. The behaviour of the combined algorithm, RSPSA, will be illustrated on a synthetic task. We close the section by discussing how the method of antithetic variables can be used in conjunction with SPSA.

3.1. Multiple evaluations vs. multiple perturbations

In this section we investigate the issue of how many simulations to run per perturbation when computing SPSA differences. Here, for the sake of simplicity, we assume that the distribution of the simulation noise term, Y, entering the objective function is independent of the parameter to be optimised.

If the variance of the evaluations is high then the estimate of the gradient as given by Eq. (5) will posses an equally high variance. A natural idea is to average the results of a few simulations to increase the precision of the estimate of the gradient before iterate θ_t is

[7] To see how f is constructed, notice that state transitions can be written in the form $X_{t+1} = g(X_t, A_t, U_t)$ with appropriate g. Further, for stationary policies, the t action can be written as $A_t = \pi(X_t, U'_t)$. The construction is finished by noting that the reward at step t is a deterministic function of X_t, A_t. The idea extends trivially to non-stationary, observation-based policies, as well as to other classes of sequential problems.

updated. In the terminology of the neural-network literature this is called the technique of using *mini-batches*. When the perturbations are kept fixed the resulting estimate is as follows:

$$\hat{g}_{q,i}(\theta) = \frac{1}{2c\Delta_i}\left(\frac{1}{q}\sum_{j=1}^{q} f(\theta + c\Delta; Y_j^+) - \frac{1}{q}\sum_{j=1}^{q} f(\theta - c\Delta; Y_j^-)\right)$$
$$= \frac{1}{q}\sum_{j=1}^{q} \frac{f(\theta + c\Delta; Y_j^+) - f(\theta - c\Delta; Y_j^-)}{2c\Delta_i}.$$

Since we assumed that the distributions of the random variables $Y_j^{\pm}$ do not depend on the respective parameter vectors, we may employ the idea of CRNs. Using $Y_j^+ = Y_j^- = Y$, we arrive at

$$\hat{g}_{q,i}(\theta) = \frac{1}{q}\sum_{j=1}^{q} \frac{f(\theta + c\Delta; Y_j) - f(\theta - c\Delta; Y_j)}{2c\Delta_i}. \quad (6)$$

In the limit, as the number of simulations q grows to infinity, the estimate will tend to $\partial_{i,c\Delta} f(\theta) = (f(\theta + c\Delta) - f(\theta - c\Delta))/(2c\Delta_i)$ with probability one. Hence, ultimately, the variance that can be attributed to the simulation noise disappears and the final approximation error becomes equal to the numerical error associated with the error of the two-sided difference, $\partial_{i,c\Delta} f(\theta)$. It should be clear that due to this error, a very large sample size q will hardly ever pay off.

By taking the average of r independent samples, $\{\hat{g}_{q,i}^{(j)}(\theta)\}_{j=1,\ldots,r}$, of $\hat{g}_{q,i}(\theta)$, we may further reduce the error of approximation. Denoting the resulting average by $\hat{g}_{r,q,i}(\theta)$, it follows by the law of large numbers that $\hat{g}_{r,q,i}(\theta)$ converges to $f_i'(\theta) + O(c^2)$ as $r \to +\infty$ with probability one (i.e., the estimate's ultimate bias is of the order $O(c^2)$). After a certain point, increasing $p \stackrel{\text{def}}{=} rq$ will not necessarily improve the rate of convergence and/or the finite-sample performance of the optimisation algorithm. This is because although increasing p increases the quality of approximation of the gradient, at the same time, it also decreases the frequency of parameter updates.[8]

In order to gain further insight into the best choice of p, q and r, let us consider the mean squared error of approximating the ith component of the gradient by $\hat{g}_{r,q,i}$: $M_{r,q,i} = \mathbb{E}\,[(\hat{g}_{r,q,i}(\theta) - f_i'(\theta))^2]$. The following expression is derived for $M_{r,q,1}$ in Appendix A:

$$M_{r,q,i} = \frac{1}{r}\mathbb{E}\left[\Delta_1^2\right]\mathbb{E}\left[1/\Delta_1^2\right]$$
$$\cdot \sum_{j=2}^{d}\left\{\left(1 - \frac{1}{q}\right)\mathbb{E}[f_j'(\theta, Y_1)^2] + \frac{1}{q}\mathbb{E}[f_j'(\theta, Y_1)]^2\right\}$$
$$+ \frac{1}{rq}\,\mathbb{E}[(f_1'(\theta, Y_1) - f_1'(\theta))^2] + O(c^2). \quad (7)$$

[8] Spall (1992) gives a heuristic calculation that shows that using decreasing gains of the form $\alpha_t = a/t^{\alpha}$ and $c_t = c/t^{\gamma}$ with $\beta = \alpha - 2\gamma > 0$, $0 < \alpha \leq 1$, $0 < \gamma$, the optimal choice of p is a minimiser of an function, $\eta(p)$, of the form $p^{\beta-1}A + p^{\beta}B$. Here $A, B > 0$ are some, generally unknown, system parameters. Note that η has a unique minimum at $p = (1-\beta)A/(\beta B)$. However, since A, B are unknown, this result is of little practical use, except that it shows the nature of the tradeoffs involved in selecting p.

Here $f_i'(\theta; Y)$ is the partial derivative of f w.r.t. θ_i:

$$f_i'(\theta; Y) = \frac{\partial f(\theta; Y)}{\partial \theta_i}.$$

It follows from Eq. (7) that for a fixed budget of $p = qr$ function evaluations, the smallest mean squared error is achieved by taking $q = 1$ and $r = p$ (disregarding the $O(c^2)$ bias term which we assume to be "small" as compared to the other terms). Under mild conditions on f and Y_1 (ensuring that the expectation and the partial derivative operators can be exchanged), $\sum_{j=2}^{d} \mathbb{E}[f_j'(\theta, Y_1)]^2 = \sum_{j=2}^{d} f_j'(\theta)^2$. Hence, in this case with the choices $q = 1$, $r = p$, the mean square error becomes equal to

$$\frac{1}{p}\left\{\mathbb{E}\left[\Delta_1^2\right]\mathbb{E}\left[1/\Delta_1^2\right]\sum_{j=2}^{d} f_j'(\theta)^2 + \mathbb{E}[(f_1'(\theta, Y_1) - f_1'(\theta))^2]\right\} + O(c^2), \qquad (8)$$

which is composed of two terms in addition to the bias term $O(c^2)$. The first term, which contains the sum $\sum_{j=2}^{d} f_j'(\theta)^2$, represents the contribution of the "cross-talk" of the derivatives of f to the estimation error of the gradient, whilst the second term, $\mathbb{E}[(f_1'(\theta, Y_1) - f_1'(\theta))^2]$ gives the mean square error of approximating $f_1'(\theta)$ with $f_1'(\theta, Y_1)$ (which is equal to the variance of $f_1'(\theta, Y_1)$ in this case). The first term can be large when θ is far from a stationary point of f, whilst the size of the second term depends on the amount of noise in the evaluations of f_1'. When the magnitude of these two terms is larger than that of the bias term $O(c^2)$ then increasing p will increase the efficiency of the procedure, at least until $1/p$ becomes comparable to $O(c^2)$. Hence, as another general rule of thumb, we conclude that it can be beneficial to increase p as c is decreased, i.e., near equilibrium points.

3.2. Resilient SPSA: Combining SPSA and RPROP

SPSA, like other stochastic approximation algorithms has a number of parameters that need to be tuned by the user. These parameters are the gain sequences α_t, c_t, the batch-size parameter, p and the common distribution underlying the perturbations Δ_t. When function evaluations are expensive, small sample behaviour becomes important. Tuning the SPSA parameters to optimise the transient performance is a non-trivial task.

Consider, e.g., selecting the perturbation size, c. It should be clear that if the magnitude of $\bar{\Delta} = c\Delta$ is large then the numerical error due to the use of two-sided differences will dominate the overall estimation error. On the other hand, when the magnitude of $\bar{\Delta}$ is small then in order to let the variance of the estimate match the size of the asymptotic bias term (whose order is of $O(c^2)$), larger batches should be used, as suggested previously. However, large batch-sizes may slow down the rate of convergence. Hence, selecting good perturbation-sizes is highly non-trivial.

It is likely that performance can be substantially improved if componentwise step-sizes are introduced. In all previous works on SPSA known to us it was assumed that the d components of the perturbations, Δ_t, have identical distributions. When different coordinates have different scales (which might be very common in practice) then it makes sense to let each component has its own perturbation size parameter.[9]

[9] Actually, having different scales for the different axes is a special case of when the problem is anisotropic, in which case one might also want to consider distributions of Δ that are not even aligned with the

The problem of selecting the 'right' scales arises in deterministic gradient methods, too. Classically, second and higher-order methods are suggested as the natural way to approach this issue. Gradient-free counterparts of such methods (e.g., 2SPSA) have been investigated both theoretically and empirically by Spall (2000) and later by Dippon (2003). The excessive computational complexity and memory requirements of these methods, however, make them less attractive for practitioners. Further, although these methods (e.g., 2SPSA) are guaranteed to achieve higher asymptotic convergence-rates, their superiority is less clear in the small sample-size case.

Hence, we looked for options that avoid the excessive computational complexity of gradient-free, higher order methods. The RPROP ("resilient backpropagation") algorithm due to Riedmiller and Braun (1993) and its variants have low computational complexity and are amongst the best performing first-order batch neural-network gradient training methods. Recently, Igel and Hüsken (2003) conducted an empirical comparison of RPROP and its variants with alternative, gradient ascent algorithms such as BFGS, CG and others. They have found RPROP methods to be generally very fast, accurate, robust to the choices of their parameters and scale well with the dimensionality of the parameter vector. Additional advantages of RPROP are that it is easy to implement, and since the updates are dependent only on the sign of the partial derivatives of the objective function[10] and not on the magnitude of these partial derivatives, RPROP is thought to be well suited to applications where the gradient is numerically estimated and/or is noisy. We shall discuss these matters after the description of the algorithm.

3.2.1. RPROP

We shall consider here a variant of RPROP, called iRPROP$^-$, due to Igel and Hüsken (2000) in a form when it is applied to maximising an objective function $f = f(\theta)$. iRPROP$^-$'s update equation is as follows:

$$\theta_{t+1,i} = \theta_{t,i} + \delta_{ti}\ \mathrm{sign}(g_{ti}), \quad i = 1, 2, \ldots, d,\ t = 1, 2, \ldots.$$

Here $\delta_{ti} \geq 0$ is the step-size for the ith component and g_{ti} is a gradient-like quantity:

$$g_{ti} = \mathbb{I}(g_{t-1,i} f_i'(\theta_t) \geq 0) f_i'(\theta_t). \qquad (9)$$

In words, g_{ti} equals to the ith partial derivative of f at θ except when a sign reversal is observed between the current and the previous partial derivative, in which case g_{ti} is set to zero. Here $\mathbb{I}(\cdot)$ is a $\{0, 1\}$-valued function working on Boolean values and $\mathbb{I}(\mathcal{L}) = 1$ if and only if $\mathcal{L}$ is true, and $\mathbb{I}(\mathcal{L}) = 0$, otherwise.

The individual step-sizes, δ_{ti}, are updated in an iterative manner based on the sign of the product $p_{t,i} = g_{t-1,i} f_i'(\theta_t)$:

$$\eta_{ti} = \mathbb{I}(p_{t,i} > 0)\eta^+ + \mathbb{I}(p_{t,i} < 0)\eta^- + \mathbb{I}(p_{t,i} = 0), \qquad (10)$$

$$\delta_{ti} = P_{[\delta^-,\delta^+]}(\eta_{ti}\delta_{t-1,i}), \qquad (11)$$

coordinate system. In this article, however, we restrict ourselves to the task of finding good componentwise perturbation-sizes.

[10] RPROP, though it was originally worked out for the training of neural networks, is applicable to any optimisation task where the gradient can be computed or approximated.

where $0 < \eta^- < 1 < \eta^+$, $0 < \delta^- < \delta^+$, and $P_{[a,b]}$ clamps its argument to the interval $[a, b]$.

Igel and Hüsken (2000) proposed another rule, iRPROP$^+$, and demonstrated experimentally that it performs slightly better than the rule just described. The difference between iRPROP$^+$ and iRPROP$^-$ is that when a sign change is observed and if the objective function decreases then iRPROP$^+$ backtracks to the previous value of the given parameter. This is called *blocking* in stochastic search (Spall, 2003). If the objective function is observed in noise then the utility of blocking will be limited – hence we omitted it from our implementation.[11]

Now, let us discuss one claim of Igel and Hüsken (2000); namely, that the sign-based approach is well suited for applications where the precise value of the gradient is unavailable, e.g., when the gradient is observed in noise. Let us approach this claim by comparing how closely the trajectories of the noisy updates simulate that of the respective trajectories with noise-free updates.

First, notice that in both cases the size of the mini-batches, p, can be used to control the 'simulation-error'. Consider first the unmodified gradient-ascent update rule, Eq. (1). After several steps of simplification, we get that after t steps the expected root mean squared difference between the trajectories is of the order $\sqrt{t\sigma_p^2}$. Here σ_p^2 is the variance of the gradient estimates with a mini-batch of size p. For the sign-based rule, the difference is $\sqrt{t2\delta_p(1-\delta_p)}$. Here δ_p is the probability that the sign of the estimated gradient is different from that of the "true" gradient. Now, by the independence of measurements, $\sigma_p^2 = 1/p\sigma_1^2$. If G is a positive lower bound on the (true) gradient in the neighbourhood of the current iterate, then $\delta_p \sim \exp(-pG^2/(2\sigma_1^2))$. Hence, the difference scales with $1/\sqrt{p}$ in the first case, whilst it scales like $\exp(-p\,\text{const})$ in the second case: The non-linear sign function is very effective at suppressing noise.[12] Although this argument is far from being rigourous, we think that it still provides a nice intuitive explanation of why RPROP (or a sign-based update rule) might be more successful.

Notice that the above reasoning is critically dependent on the assumption that the parameter vector is *not* close to the stationary point of the objective problem (hence the gradient can be bounded away from zero). As pointed out by Schraudolph (1999), one problem with RPROP is that it can behave badly near the stationary points. This is because the sign function does not commute with the expectation and hence the zero-mean property of the gradient estimates at equilibrium will not transfer to that of the sign of the gradient estimates. In fact, this may cause the step-sizes to converge to suboptimal values (divergence of the step-sizes is prevented as they are bound to $[\delta^-, \delta^+]$ by the update rule).[13] Although this is a valid point, we think that the problem will likely turn up only close to the equilibrium points, whilst our interest here lies in the transient behaviour. Hence, in the present context, we do not see the problem a serious limitation of the method. Further, when equilibrium behaviour is important, the problem can be mitigated by increasing the size of mini-batches (as we saw beforehand, for optimal performance, the size of the mini-batches should be increased near equilibrium points anyway). Yet another possibility might be to adopt a variant of RPROP that renders it globally convergent, such as, e.g., the GRPROP algorithm due to Anastasiadis, Magoulas and Vrahatis (2005).

[11] In some weak form, blocking still exists in our code: In practice the performance is typically monitored during the search procedure, i.e, time-to-time the parameters are evaluated. It is then natural to keep at the end those parameters that were observed to result in the best performance.

[12] In a rigourous derivation, Bernstein's inequality could be used to derive a bound on δ_p.

[13] Note that another source of non-convergence of RPROP is that typically $\delta^- > 0$ in which case, near equilibriums, at best it will behave as a constant gain stochastic gradient ascent procedure.

3.2.2. RSPSA

We call the combined SPSA-iRPROP$^-$ algorithm RSPSA ("resilient SPSA"). The simplest combined algorithm works by replacing $f_i'(\theta_t)$ in the iRPROP$^-$ equations with its noisy estimate, $\hat{g}_{p,1,i}(\theta_t)$. By virtue of the previous discussion, we speculate that the performance of RSPSA might often surpass that of SPSA.

However, there is an extra potential that arises from the combination of RPROP and SPSA, that we discuss now. Assuming that $|f_i'(\theta)| > G$, Markov's inequality[14] gives the following bound:

$$\mathbb{P}(\text{sign}(\hat{g}_{p,1,i}(\theta)) \neq \text{sign}(f_i'(\theta))) \leq \mathbb{P}(|\hat{g}_{p,1,i}(\theta)) - f_i'(\theta)| \geq G) \leq \frac{M_{p,1,i}}{G^2}.$$

Hence the performance of RSPSA will be bound by the mean square error $M_{p,1,i}$. As discussed previously, the distribution of the perturbations Δ_{ti} may influence this quantity strongly and it is not reasonable to expect that the same scales will work well for all coordinates.

Obviously, there is no way to decide *a priori* the 'best' scales. In fact, in flat areas (with a smaller average gradient magnitudes) larger scales are desirable, whilst where the objective function varies a lot, smaller scales could be more beneficial. Thus, ideally, the perturbations should fit the local characteristics of the objective function.

An attractive idea then is to couple the SPSA perturbation parameters Δ_{ti} and the step-sizes of RPROP. This is motivated by the observation that the step-sizes, δ_{ti}, of RPROP are expected to grow in "flat areas" where the sign of appropriate partial derivatives does not change, whilst they are expected to decay in areas where the sign of the partial derivatives varies a lot. A simple way to couple the two step-sizes is to set

$$\Delta_{ti} = \rho\, \delta_{ti},$$

where ρ is some positive constant, to be selected by the user.

3.2.3. Empirical results on a synthetic task

In order to gain some experience with RSPSA and compare its performance to that of SPSA, we have tested it on a toy problem, the 10-5-10 encoder, used by Riedmiller and Braun (1993) and others. The problem is to tune the weights of a neural network so as the trained network maps specific binary inputs to outputs that are exactly identical to the inputs. The 10 input patterns have the form $(0, 0, \ldots, 1, \ldots, 0)$ such that the ith pattern has exactly one non-zero value at its ith position. The task is non-trivial as the number of neurons in the hidden layer is less than that in the input layer. The objective function to be maximised is written as

$$f(\theta) = \mathbb{E}\,[f(\theta; Y)] = -\mathbb{E}\left[\sum_{i=1}^{10} \|h(x_i;\theta) - Y_i(x_i)\|^2\right], \qquad (12)$$

where x_i goes through the 10 input patterns and $Y_i(x_i)$ is the noise-corrupted output pattern corresponding to x_i. Some details of these experiments, such as the definition of the outputs, $Y_i(x_i)$, or the choice of the various step-sizes used in the algorithms are given in Appendix B.

[14] We could use here a tighter tail inequality, e.g., Hoeffding's or Bernstein's. However, the bounds would not change as far as their scaling behaviour is concerned with respect to the mean square error and G.

The research questions investigated here were as follows:

– How does SPSA perform on this simple problem? In particular, we were interested in the dependence of its performance on parameter c and on the size of the mini-batches.
– How does RSPSA perform as a function of the above two parameters? Does coupling help to improve RPSA's performance?

We also experimented with SMD, another local step-size adaptation rule due to Schraudolph (1999). However, no results are presented here for SMD, as they showed that, in this case at least, SMD is not effective in improving performance (it did not prove to be competitive with either SPSA or RSPSA).

Each experiment was repeated 15 times. The performance of a network is defined as its mean square error (MSE) over the 10 patterns. The curves in the figures show the smallest errors obtained until a given moment. This somewhat unconventional measure is motivated by current practice of parameter tuning in game programs: since parameter tuning takes a long time, one typically monitors performance during learning, if not for other reasons than to make sure that things work normally. At the end of training, since performance measurements are available for many parameter settings, it is natural to keep the best parameter settings encountered. The values shown in the figures below are the average of these values over 15 runs, whilst the error bars shown are computed as the minimum and maximum of the 'middle' 13 values of the 15 values (i.e., the error bars correspond roughly to 0.05 and 0.95 percentiles). In all figures, three curves are plotted, each curve corresponding to the performance after a specific number of function evaluations (1000, 10, 000 and 50, 000, marked respectively by 1 k, 10 k and 50 k there).

When testing SPSA, we used constant step- and perturbation-sizes in these experiments in line, optimised to the task. Further, a momentum term was added so as to further improve performance (the gain of the momentum term was also tuned to the task). Figure 1 shows the results obtained for SPSA when there is no gradient averaging (the size of mini-batches is 1). The figure shows performance as a function of the size of SPSA perturbations (c). The exact values of the step-sizes are given in Appendix B). As it can be observed from the

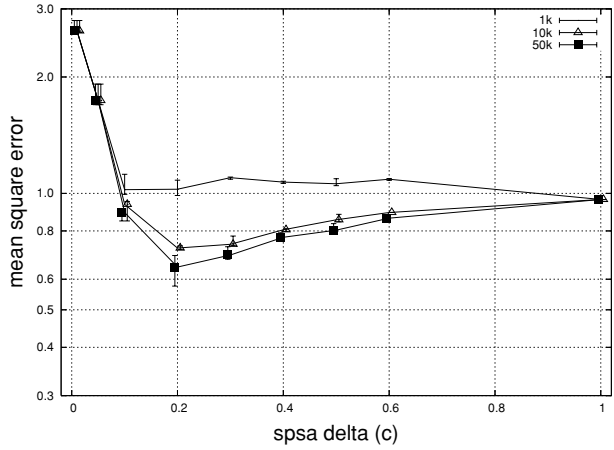

Fig. 1 Mean square error of SPSA as a function of the size of SPSA perturbations. The size of mini-batches is 1. The three curves correspond to performance after a specific number of objective function evaluations ($1k$, $10k$ and $50k$ mean 1, 000, 10, 000 and 50, 000 evaluations, respectively)

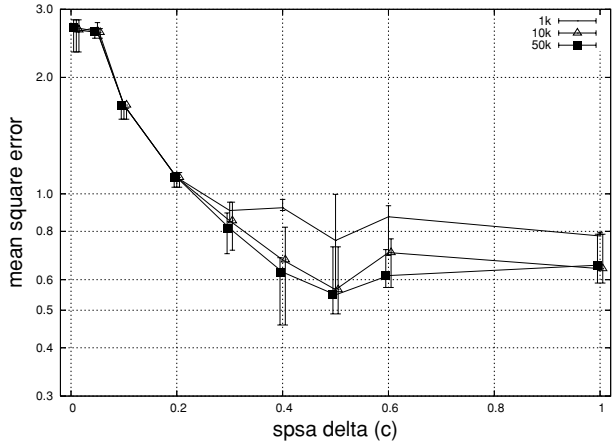

Fig. 2 Mean square error of SPSA as a function of the size of SPSA perturbations. The size of mini-batches is 50. The three curves correspond to performance after a specific number of objective function evaluations ($1k$, $10k$ and $50k$ mean $1,000$, $10,000$ and $50,000$ evaluations, respectively)

figure, performance is not very sensitive to the value of the perturbation size, at least when the perturbation size is above a certain critical value. However, convergence is slow. We believe that this is caused by the large noise of the output patterns, which, in turn, is inherited by the gradient estimates.

Next, we investigated how the number of samples in the mini-batches effect performance. Results indicated that batch-sizes around 25–50 are optimal for this task. Accordingly, in the next figure (Fig. 2) the size of the mini-batches is increased to 50 samples. Note that in the figures performance is given as a function of the number of samples drawn from $f(\theta; Y)$, as opposed to the number of updates of the parameters. Thus, compared with the previous figure, the number of updates for a given curve here is 50 times less than that for the corresponding curve of the previous experiment. Despite this, we see that performance improves at least when comparing the best results. In fact, for the best results, the conclusion holds for all the sample-sizes shown, indicating an overall speedup of convergence.

Next, we investigated the performance of the RPROP + SPSA combination. Initially, the combination was tested with no coupling between the SPSA perturbation-sizes and the SPSA step-sizes. The batch-size was kept at 50. (Experiments with other batch-sizes showed that, in line with our expectations, performance gets worse faster than for SPSA as the batch-size is decreased. For larger batch-sizes the two algorithms seem to behave qualitatively in the same manner.) Results shown in Fig. 3 indicate a significant performance improvement for the respective best perturbations.

Finally, Fig. 4 shows the performance of RSPSA when SPSA step-sizes are coupled to RPROP step-sizes. In contrast to previous figures, performance here is shown as a function of the coupling factor. The parameters of RPROP were kept the same as those found to perform the best in the previous experiment.

Notice that coupling is very effective: For a wide range of the coupling factors convergence is faster with coupling than without it. Further, (i) the performance curve seems to be more-or-less convex as a function of the coupling parameter and (ii) optimal performance is achieved for a wide range of the coupling factors. We also note that the shapes of the three curves in Fig. 3 are roughly identical: As a result, in this case at least, it looks possible to

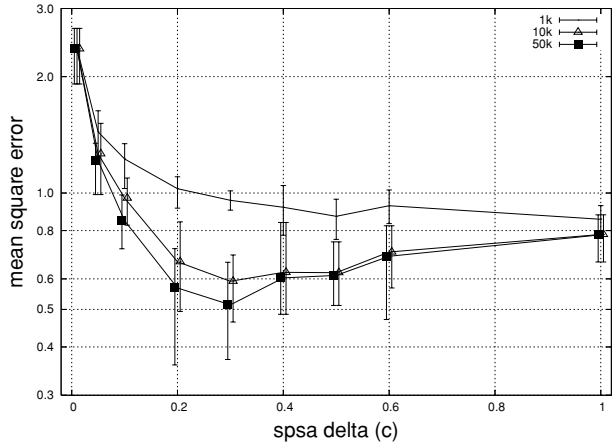

Fig. 3 Mean square error of RSPSA as a function of the size of SPSA perturbations with *no* coupling between the RPROP step-sizes and the SPSA perturbation sizes. The size of mini-batches is 50. The three curves correspond to performance after a specific number of objective function evaluations (1 k, 10 k and 50 k mean 1,000, 10,000 and 50,000 evaluations, respectively)

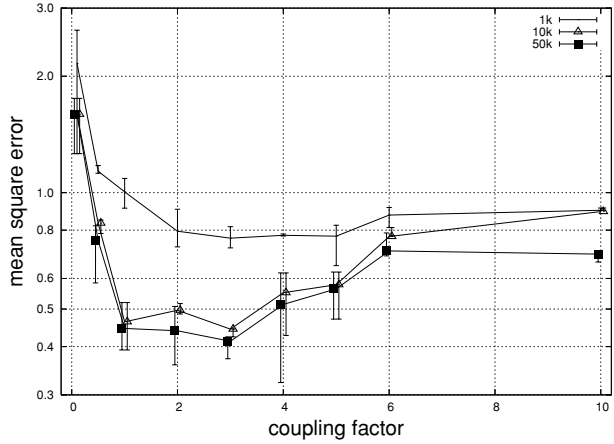

Fig. 4 Mean square error of RSPSA as a function of the parameter that couples SPSA perturbation-sizes to RPROP step-sizes. The size of mini-batches is 50. The three curves correspond to performance after a specific number of objective function evaluations (1 k, 10 k and 50 k mean 1,000, 10,000 and 50,000 evaluations, respectively)

find good coupling factors by testing a few selected values while keeping the sample size small. These observations suggest that simple methods might work well for identifying a near-best coupling factor in an efficient manner (a similar conclusion seems to hold for the perturbation-size parameter, as well; cf. Figs. 1 and 2). On another note let us remark that compared with the standard RPROP parameter values the parameters that we found to perform well for these tasks are less extreme. This is indeed what can be expected taking into account the noise in the gradient estimates.

In summary, the investigated problem illustrates that having somewhat larger mini-batches and using RSPSA with coupling can indeed be beneficial in improving the performance of

SPSA. Further, at least in this simple problem, it is possible to find good values of the coupling factor in an efficient manner.

3.3. Antithetic variables

Suppose that one wishes to compute the expected value, I, of a random variable R: $I = \mathbb{E}[R]$. The variance of the straightforward Monte-Carlo estimate,

$$I_n = \frac{1}{n}(R_1 + \cdots + R_n),$$

is $(1/n)\,\mathrm{Var}(R)$. Here $R_1, \ldots, R_n$ are i.i.d. and share the distribution of R. Now, assume that n is even, say $n = 2k$, and consider the estimate

$$I_n^A = \frac{1}{k}\sum_{i=1}^{k} \frac{R_i^+ + R_i^-}{2},$$

where now it is assumed that $\mathbb{E}[R_i^+] = \mathbb{E}[R_i^-] = I$ (thus $\mathbb{E}[I_n^A] = I$) and that $\{R_1^+, \ldots, R_k^+\}$ are i.i.d., just like $\{R_1^-, \ldots, R_k^-\}$. Note that no assumption is made on the independence of R_i^+ and R_j^-. Since

$$\mathrm{Var}\left(\frac{R_i^+ + R_i^-}{2}\right) = \frac{\mathrm{Var}(R_i^+) + \mathrm{Var}(R_i^-) + 2\mathrm{Cov}(R_i^+, R_i^-)}{4},$$

we see that

$$\begin{aligned}\mathrm{Var}\left(I_n^A\right) &= (1/k)\,\mathrm{Var}((R_1^+ + R_1^-)/2)\\ &= (1/4k)\,(\mathrm{Var}(R_i^+) + \mathrm{Var}(R_i^-) + 2\mathrm{Cov}(R_i^+, R_i^-))\\ &\leq \mathrm{Var}(I_n),\end{aligned}$$

provided that $\mathrm{Var}(R_i^+) + \mathrm{Var}(R_i^-) \leq 2\mathrm{Var}(R_i)$ and $\mathrm{Cov}(R_i^+, R_i^-) \leq 0$. When R_i^+, R_i^- share a common distribution, but $\mathrm{Cov}(R_i^+, R_i^-) < 0$ then we say that R_i^+ and R_i^- are *antithetic*. It follows then that if R_i^+ and R_i^- are antithetic and the common distribution of $R_i^\pm$ is identical to the distribution of R then the estimated variance of I_n^A will be smaller than that of I_n. Notice the similarity to the CRN method introduced in Section 2.5: In both cases introducing correlation helps decreasing the variance of some estimates. Actually, however, the two methods work in a complementary way. The CRN method introduces *positive* correlation between random variables that are *subtracted* from each other, whilst the method of antithetic variables introduces *negative* correlation between random variables that are *added* together, whilst in both cases the random variables share a common distribution.

As an example on how to use antithetic variables (AVs) in SPSA, or more generally, in game parameter optimisation, consider the performance of a player in a non-deterministic game. Let us collect all random choices *external* to the players into a random variable Y and let $f(Y; W)$ be the performance of the player in the game. Here W collects the random choices of the players. For example, in back-gammon variable Y would collect the outcomes of dice-rolls, whilst in a card-game, such as poker, Y can be chosen to represent the cards of the shuffled deck.

Assuming that the player's play reasonably well, the influence of the random choices Y on the outcome of the game will be strong. By this we mean that the value of $f(Y; W)$ is largely determined by the value of Y. For example, it may happen that in backgammon the dices roll in favour of one of the players. Another example is in poker when one player gets a strong hand, whilst the others get weak ones. Staying with this example and assuming two-players, a natural idea to mitigate the influence of Y on the measured performance is to reverse the hands of the players: the hand of the first player becomes that of the second and vice versa (the community cards are kept the same). Denoting the cards in this new scenario by Y', it is expected that $\mathrm{Cov}(f(Y; W_1), f(Y'; W_2)) < 0$ (here W_1 represents the random choices in the first play with cards Y and W_2 represents the random choices in the play with cards Y'; W_1 and W_2 are assumed to be independent of each other). Since the distribution of Y and Y' are identical (the mapping between Y and Y' is a bijection), if the players play "perfectly", the same will hold for the distribution of $f(Y; W_1)$ and $-f(Y'; W_2)$. Hence $f(Y; W_1)$ and $f(Y; W_2)$ will be antithetic. When the random choices Y influence the outcome of the game strongly then we often find that $f(Y; W_1) \approx -f(Y'; W_2)$. More generally, this is the case then $\mathrm{Cov}(f(Y; W_1), f(Y'; W_2)) \approx -\mathrm{Var}(f(Y; W))$ and thus $\mathrm{Var}\left(I_n^A\right) \approx 0$. Of course, $f(Y; W_1) = -f(Y'; W_2)$ will rarely, if ever, hold and hence the variance of I_n^A will of course remain positive. Nevertheless, the argument shows that we can expect to achieve sizeable variance reduction by using this method.

This method can be used in the estimation of the gradient or when the performance of the players is evaluated. A simple way to introduce AVs in the case of gradient estimation using SPSA is to replace the SPSA gradient estimate defined by Eq. (5) with[15]

$$\hat{g}_{ti}(\theta_t) = \frac{1}{4c_t\Delta_{ti}}((f(\theta_t + c_t\Delta_t; Y_t) + f(\theta_t + c_t\Delta_t; Y_t')) - (f(\theta_t - c_t\Delta_t; Y_t) + f(\theta_t - c_t\Delta_t; Y_t'))).$$

In our poker experiments we measured a reduction of the variance by a factor of ca. 20 in game evaluations. We expect that similar improvements are possible in other games when 'external randomness' influences the outcome of the game strongly.

4. Test domain: Omaha Hi-Lo Poker and McRaise

Let us now turn to the description of our test domain. We start by the description of the rules of Omaha Hi-lo Poker. Next, we discuss topics that arise when measuring relative strengths of players. The game program, McRaise, used in the experiments is introduced in Section 4.2.

4.1. The rules

Omaha Hi-Lo is an N-person, zero-sum, alternating Markov game with imperfect information. It is played by 2–10 players, who place their betting actions sequentially in clockwise order.

A game begins with two forced bets, the *small blind* and the *big blind*. After the blinds, four *hole* (private) cards are dealt to each player, followed by the first round of betting. In a betting round, the player on turn (usually) has three options: *fold*, *check/call*, or *bet/raise*. On fold,

[15] Note that the same technique can be used to reduce the variance of FDSA gradient estimates.

the player becomes inactive, not having the possibility of further bets, nor winning a share of the pot. On check/call, the player levels his contribution to the pot with the outstanding bet level. On bet/raise, the player increases the bet level. A maximum of four bets or raises are allowed per betting round. A betting round terminates when all active players matched the current bet level. After the first betting round, three *community* cards are dealt face up (the *flop*). This is followed by a second betting round. On the *turn* a fourth community card is dealt, followed by the third betting round. A fifth community card is dealt on the *river*, followed by the final (fourth) betting round.

After the last betting round, at *showdown*, the pot is split among the active players depending on the strength of their cards. The players form a *high hand* and (if possible) a *low hand*, each consisting of exactly two hole cards and three community cards. The *high side of the pot* is won by the best high hand according to the usual ranking of poker hands. For the *low side*, a hand with five cards with different numerical values from Ace to eight has to be constructed. The winning low hand is the one with the lowest high card. If more players have the same high card, the second highest will decide, and so on. Ties are possible for both sides of the pot. In this case that side is split. If there is at least one low hand then both sides equal half of the pot, otherwise the whole pot goes to the high side.

There are several rules for limiting the amount that can be raised by a player. We use the *fixed-limit stake structure*, which in the first two betting rounds sets this amount to a value termed *small bet*, and in the next two rounds to the *big bet*. The big bet is twice the value of the small bet.

A natural performance measure of a player's strength is the average amount of money won per hand divided by the value of the small bet (sb/h). Typical differences between players are in the range of 0.05 to 0.2 sb/h. For showing that a 0.05 sb/h difference is statistically significant in a two player game, one has to play up to 20,000 games. This number was estimated by means of extensive simulations.

To illustrate the difficulty of obtaining reliable performance estimates, let us consider some bounds on the number of games sufficient for detecting a difference of 0.05 sb/h with an error probability limited to 5%. (In fact, these were the actual design parameters of our experiments.)

Without any further information about the distribution of the payoffs, the best bound is obtained by Hoeffding's inequality. Unfortunately, the resulting number comes out to be very large: with δ denoting the error probability and K denoting the maximum gain/loss per player in a single game, we get that $m(\delta) = 2\ \ln(2/\delta)\,(K/0.05)^2$ games are required, which, for the case considered, is about 1,700,000, even if only two-player games are considered. This huge number is the result of the conservative nature of Hoeffding's inequality that relies only on the maximum gain/loss per player, which, in two-player games can be as large as 24 sb in a single game.[16] When the variance of the reward distribution is known, then Bernstein's inequality gives tighter bounds than Hoeffding's. In fact, in our case the variance of the rewards is estimated to be about 36 $(\text{sb/h})^2$. Plugging this into Bernstein's inequality yields the bound of 107,420 games. (This number is just about the same as the number that can be obtained from the central-limit theorem.)[17]

Fortunately, we can do even better by introducing AVs. As suggested in Section 3.3, in every second game each player is dealt his/her opponents' cards of the previous game, while

[16] All these figures get substantially larger when the number of players is larger than two.

[17] That according to our extensive Monte-Carlo simulations 20,000 games proved to be sufficient is caused by the significant non-normality of the payoff-distribution.

the community cards are kept the same. As we will see, such *antithetic dealing* results in a drastic reduction of the number of required games. We note that this is crucially important, since running 20,000, let alone 100,000 simulations every time a player's strength needs to be measured would be prohibitive. (Simulating a single game takes ca. 1 second on average on an AMD Opteron, 2 GHz machine. Hence playing 20,000 games lasts ca. 5 and a half hours.)

With antithetic dealing, variance per play drops to 1.44 sb/h. Plugging this into Bernstein's inequality gives the modest bound of 5,430 games, a reduction by a factor of about 20. Compared with playing 20,000 games, the reduction factor is still considerable (4) and antithetic deals have the added benefit of stronger theoretical guarantees. Of course, antithetic dealing assumes that players do not exploit this specific dealing strategy. This holds, of course, for our computer programs, by their design.[18] Given the huge variance reduction it buys, in all of our experiments described below we used antithetic dealing. In fact, when we report some player's strength, then this is always the empirical mean of the payoffs of 20,000 antithetic games. During learning, for monitoring purposes, however, the number of antithetic deals is kept at 5,000.

4.2. McRaise

Let us now describe our poker programs', McRaise's, action-selection mechanism. Our program, McRaise, borrows several ideas from other poker playing programs developed for another variant of poker, Texas Hold'em (Billings et al., 2002, 2003, 2004). The name of the program originates from the use of Monte-Carlo simulations and the program's aggressive style.

McRaise, as used in the experiments employs a rather simple 1-ply search. Although the program is capable of looking further ahead, the resulting gain in performance is small, whilst the computation time can easily double. Hence, in the experiments we restricted ourselves to the simple, 1-ply search variant that we describe now.

4.2.1. Basic principles of McRaise's action-selection strategy

Probably the most important element in poker play is to estimate one's winning chances, or more precisely to predict how much share one will get from the pot. In order to succeed at this task, a player has to evaluate correctly the strength and potential of his/her own cards, which, however, can be done only in light of his/her opponents' (unknown) hole cards. One approach to this is to guess the opponents' cards based on their betting actions of the current, and possibly previous games. Betting actions encountered in previous games can be used to adjust a model of the opponent (human players do something like this when categorising their opponent into categories like 'tight-passive', 'loose-passive', 'tight-aggressive', and 'loose-aggressive'). Given an opponent model, the player might try to 'deduce' his/her opponents' hole cards, given the betting actions of the current game. Obviously, guessing the opponent's private cards should result in no particular hand, but a probability distribution over the possible hands.

This is exactly McRaise's approach: The opponents' betting models are used to derive a probability distribution over the possible hands, taking into account all previous information,

[18] When playing against humans, antithetic dealing can still be used by making assumptions on the number of games a human can remember and then mixing 'well' a large number of pairs of antithetic decks.

namely McRaise's private cards, the community cards and the betting history. Ideally, one would like to know the full probability distribution over all possible hands, as it was done in Poki by Billings et al. (2002). Unfortunately, this is clearly infeasible to achieve, especially when playing against multiple opponents: Whilst in Texas Hold'em (Poki's domain) every player has two hole cards, in Omaha Hi-Lo the number of private cards is four. McRaise's solution to this is to represent the distribution by a weighted random sample. Given such a weighted sample, McRaise computes the expected payoff, $Q(s, a)$, for each of the available actions a (here s denotes the information available to McRaise). Given $Q(s, a)$, the action with the highest value is selected:

$$a(s) = \operatorname*{argmax}_a Q(s, a).$$

4.2.2. Estimating action values under the "everyone's checking" assumption

Consider a game situation when it is McRaise's turn to act. Let s be the corresponding game history known to McRaise.

We define the value of action a given history s, $Q(s, a)$, as McRaise's expected payoff provided that McRaise's next action is a, and assuming that subsequently it plays optimally against its opponents (it follows that we assume here that the opponents' strategies are fixed). If R denotes the pot size at the end of the game, when McRaise's total contribution to the pot is R_1, then $Q(s, a) = \mathbb{E}[w(C, I)R - R_1|s, a]$. Here C is a random card configuration and R, R_1 and I are random variables whose distribution depends on s and a. In particular, I is the index-set of the active players at the end of the game and for any such index set and card configuration c, $w(c, I)$ is the percentage of pot won by McRaise.

Computing $Q(s, a)$ obviously requires looking ahead in the game tree. Since the variance of future payoffs can be very high, one may want to opt for an alternative, lower variance estimate. An option, that from the point of view of computational efficiency looks particularly appealing, is to estimate $Q(s, a)$ under the assumption that every player checks (including McRaise) from the current point of the game until showdown. We call this the "Everyone's Checking" (EC) Assumption. In what follows we shall denote by $\hat{Q}(s, a)$ the corresponding value.

Let us now discuss the consequences of adopting EC. First, note that since $\hat{Q}(s, \text{fold}) = Q(s, \text{fold})$, the discussion can be restricted to the case when the action considered is either 'raise' or 'check'. Clearly, the pot size obtained under EC will never be larger than R except when some player folds before he/she would match the current bet level. Note that the pot-size under EC, denoted by $\Pi(s, a)$, can be computed from s and a alone. Further, the expected proportional payoff (EPP) under EC, which we denote by $\bar{w}(s, a) = \mathbb{E}_{\text{EC}}[w(C, I)|s, a]$, is always lower than the EPP without EC. This is because if no opponent folds then these two values are the same. On the other hand, when some opponents fold then the winning chance of McRaise increases. Hence, $\bar{w}(s, a)$ is a lower bound on the proportion of the pot won. In summary, unless some of the opponents folds before the end of the current betting round, the value computed under EC will be a lower bound on the true value. Further, for the fold action the two values are equal.

Let us now look at the problem of computing $\hat{Q}(s, a)$. Under EC game-tree search can be avoided altogether:

$$\begin{aligned}\hat{Q}(s, a) &= \mathbb{E}_{\text{EC}}[w(C, I)R - R_1|s, a] \\ &= \bar{w}(s, a)\Pi(s, a) - B(s, a). \quad (13)\end{aligned}$$

Here $B(s, a) = \mathbb{E}_{\mathrm{EC}}[R_1|s, a]$ is the total contribution of McRaise to the pot, assuming EC. It is easy to see that $B(s, a)$ can be computed from s and a alone.

In particular, when a is the fold action then $B(s, a) = -B_0(s)$ and $\bar{w}(s, a) = 0$ and therefore $\hat{Q}(s, a) = -B_0(s)$. Here $B_0(s)$ represents McRaise's current contribution to the pot. When a is not the fold action then $B(s, a) = B_0(s) + D(s, a)$, where $D(s, a)$ is McRaise's contribution to the pot when action a is selected. Further, $\Pi(s, a) = \Pi_0(s) + (P_1(s) - P_2(s))(D(s, a) - D(s, \mathrm{check})) + P_2(s)D(s, a)$, where $\Pi_0(s)$ is the pot-size in s, $P_1(s)$ is the number of active players and $P_2(s)$ is the number of players who have not yet matched the current bet level in the current betting round (before McRaise's turn). Note that the action selected by McRaise can be expressed as a function of $D(s, \mathrm{raise})$, $D(s, \mathrm{check})$, $P_1(s)$, $P_2(s)$, $\Pi_0(s)$, and $\bar{w}(s, \mathrm{raise}) = \bar{w}(s, \mathrm{check})$, alone. The exact the functional form of action selection can be found after some straightforward algebra. What is perhaps surprising at the first glance, at least, is that $B_0(s)$, McRaise's current contribution to the pot, does not enter the action-selection function. We note in passing that this will be true for any action-selection procedure that is based on estimates of the *future* payoffs.

4.2.3. Estimating the expected proportional payoff

The only remaining undefined term in the definition of $\hat{Q}(s, a)$ is $\bar{w}(s, a)$. Obviously, if a is the fold action then this quantity equals zero. Otherwise, under EC, we have $I = I(s)$ and thus

$$\bar{w}(s, a) = \mathbb{E}[w(C, I(s))|s, a] = \sum_c w(c, I(s))p(c \mid s).$$

Here c goes through all the card configurations and $p(c|s)$ denotes the probability of a card configuration given history s:

$$p(c \mid s) = \frac{p(c \mid s)p(c)}{p(s)}.$$

McRaise estimates $\bar{w}(s, a)$ by weighted importance sampling (WIS) by sampling random card configurations. Before going into how this is done, it will be beneficial to discuss how WIS works. Consider the problem of estimating the expected value $\mathbb{E}[f(X)] = \sum_x f(x)p(x)$. Assume that we can draw independent samples $Y_1, \ldots, Y_n$ from a distribution q whose support includes that of p (i.e., $q(y) \neq 0$ whenever $p(y) \neq 0$ holds for any y). The WIS-estimate of $\mathbb{E}[f(x)]$ is given by

$$S_n = \frac{\sum_{i=1}^n f(Y_i)\lambda(Y_i)}{\sum_{i=1}^n \lambda(Y_i)},$$

where $\lambda(Y_i) = p(Y_i)/q(Y_i)$ are the so-called *importance weights*. Since $\mathbb{E}[f(Y_i)p(Y_i)/q(Y_i)] = \mathbb{E}[f(X)]$ and $\mathbb{E}[p(Y_i)/q(Y_i)] = 1$, it follows by the law of large numbers that the WIS estimate will converge to $\mathbb{E}[f(X)]$ with probability one as $n \to \infty$. A crucial property of WIS-based estimation is that for computing S_n the values of the importance weights, $\lambda(Y_i)$, are needed only up to a constant factor. Note that the 'missing' constant factor can depend on anything except the random variables themselves.

Accordingly, when computing the importance weights of a card configuration, it follows that it is sufficient to compute $p(c \mid s)$ (or a value that is proportional to it up to a constant

factor, independent of c). Clearly, since the history s contains the actions of the opponents, $p(c \mid s)$ will depend on these actions and the action-selection strategies of the opponents. Assume for the sake of specificity that the total number of actions in s is m. Let $i_1, i_2, \ldots, i_m$ be the playing order of the players in s, and let $a_t(s)$ denote the tth action in s. By Bayes' law and since all players must base their decisions on information available to them, we arrive at

$$p(c \mid s) = p(\pi_1(s) \mid c) \prod_{t=1}^{m} p_{i_t}(a_t(s) \mid s_t, c_{i_t}(c)).$$

Here $\pi_1(s)$ denotes the private cards of McRaise, p_i is the action-selection model of player i, s_t is the public history up to stage t (including actions and known community cards, but excluding $\pi_1(s)$) and $\pi_i(c)$ denotes the private cards of player i given the card configuration c. When computing the value of $p(c \mid s)$, McRaise replaces p_{i_t} with its own models of the other players' respective action-selection strategies. In order to simplify the notation, we do not introduce new symbols for these functions.

Note that $p(\pi_1(s) \mid c) \propto \mathbb{I}(\pi_1(s) = \pi_1(c))$. Of course, in practice, sampling will take into account this: McRaise will only sample card configurations that satisfy the compatibility relation $\pi_1(s) = \pi_1(c)$.

Now, let $r(s)$ denote the number of betting rounds in s. Fix r and i and collect those t time indexes that belong to actions of round r and player i. Denote the set of these time indeces by $T(r, i)$. Reordering the terms in the above product yields[19]

$$p(c \mid s) \propto \mathbb{I}(\pi_1(s) = \pi_1(c)) \prod_{i=1}^{n} \prod_{r=1}^{r(s)} \prod_{t \in T(r,i)} p_i(a_t(s) \mid s_t, c_i(c)).$$

Notice that when computing the importance weights, the product of action-selection probabilities corresponding to McRaise's own actions ($i = 1$) need not be calculated as $p_1(a_t(s) \mid s_t, c_1(c)) = p_1(a_t(s) \mid s_t, c_1(s))$ is independent of c.

Since complex opponent models are hard to tune, we chose a simple one where the action probabilities for a given history and opponent depend only on a small number of quantities, such as the *a priori* strength of the opponent's cards, the actual bet size, pot level, and simple statistics of the action histories of the opponent's opponents like the maximum of the opponent's opponents' discounted empirical raise probabilities. In particular, we decided that the opponent's own previous actions should not influence these action-probabilities directly. However, if such an opponent model is plugged into the above equation then it will fail to capture that players typically try to balance their actions throughout the game. Our solution is to take into account the dependency among the actions at the level of betting rounds, by modelling the joint probability of an action sequence, up to a constant factor by the following equation:[20]

$$\prod_{t \in T(r,i)} p_i(a_t(s) \mid s_t, c_i(c)) \propto \frac{1}{|T(r,i)|} \sum_{t \in T(r,i)} \frac{q(a_t(s) \mid \phi(s_t, c_i(c)))}{q(a_t(s))}. \qquad (14)$$

[19] The empty product is defined as 1.

[20] Implicitly, we are assuming therefore that players' actions belonging to different betting rounds are independent of each other.

Here $\phi(s_t, \pi_i(c))$ are features extracted from history s_t and the private card configuration, $\pi_i(c)$, of player i, $q(a|\phi)$ is the probability of action a as a function of ϕ and $q(a)$ is the probability of seeing action a. In words, this equation says that the probability of seeing a sequence of actions of opponent i in a given round is proportional to the average of the excess probabilities of the individual actions. Compared with the values that would be obtained by invoking the independence assumption, this equation will weight those action sequences where $q(a_t(s)|\phi(s_t, \pi_i(c))) > q(a_t(s))$ substantially heavier. This means that those card configurations will get larger weights for which these excess opponent-model probabilities are large. In order to get a better understanding of this equation let us assume that q in (14) does not depend on the features, ϕ, but just on the card configuration. Now assume that for some card configuration the ratio of the probability of raise to its *a priori* probability is smaller than the same ratio for check (the considered hand is 'weak'). Consider a sequence of two actions. Assume that the first action of the sequence is raise. Using Eq. (14) it is then possible to show that the probability that the second action is raise is smaller than it was in the first round. Conversely, the probability of check is larger than its probability in the first round. Hence, the equation indeed balances the action-selection procedure. Although this equation is admittedly ad-hoc, we have found it to work pretty well in practice.

5. Experiments

We tested RSPSA by optimising two components of McRaise, the opponent model and the action selection. For both cases, we compare the performance gain achieved using RSPSA with that of obtained by some competing algorithms.

For the opponent model we consider the tuning of the parameters of the opponent models (denoted by $q(a|\phi)$ above). In the case of optimising action selection two options were considered: (i) optimising the parameters of an after-state value function that is then used in a 1-ply search, and (ii) introducing and optimising an action-network that directly assigns scores to the actions, that is used in turn to define a probability distribution over the actions. The experiments for the opponent-model optimisation are described in Section 5.1, whilst those for action-selection optimisation are given in Section 5.2.

5.1. Tuning the opponent model

The opponent model of McRaise assigns probabilities to the opponent's actions given some features ϕ. The current opponent model uses in total six parameters. The *a priori* values of these parameters were obtained by tuning them to some game databases. Here, we consider if RSPSA is able to improve the tuning of these parameters of McRaise whilst playing against McRaise with the default parameters. Before describing the experiments, however, let us look at the differentiability of the objective function, and, in connection to this, the potential of using the LR (or policy-gradient) method described earlier in Section 2.2 for this problem.

The objective function can be written as

$$f(\theta) = \mathbb{E}\left[R_\theta(C)\right],$$

where $R_\theta(C)$ is the payoff of the McRaise given the card configuration C. Denoting the sequence of actions of the players by U and the set of players that are active at the end of the game by $I(U)$, it should be clear that (i) the distribution of U depends on C and θ, and *(ii)* $R_\theta(C) = r(U, C)$ for some (deterministic) function r. Hence, $f(\theta)$ can be written

in the form

$$f(\theta) = \frac{1}{N} \sum_c \sum_u w(c, I(u)) r(u, c) p_\theta(u \mid c).$$

Here N is the number of card configurations and $p_\theta(u \mid c)$ is the probability of an action sequence given card configuration c. Hence, the objective function takes the form of Eq. (3) in Section 2.2. Therefore, in theory, the LR method can be used to estimate the gradient of f. As discussed previously, for this we need to calculate the score function $(\partial/\partial\theta) \ln p_\theta(u \mid c)$, which in turn boils down to the calculation of the score functions associated with the action-selection probability functions, $p_\theta(a \mid s)$. By construction, the action probability $p_\theta(a \mid s)$ can be written in the form $P(F_a(W_{M,\theta}(s, a)) \geq 0)$ for some smooth function F_a (see Section 4.2.2). Here $W_{M,\theta}(s, a)$ is the estimate of $\bar{w}(s, a)$, computed by sampling M card configurations, $C_1, \ldots, C_M$, and weighted importance sampling, as described in the previous section. Denoting by $\lambda_\theta(C_i, s)$ the importance weight calculated for card configuration C_i and assuming that a is not the fold action, we get

$$p_\theta(a \mid s) = \left(\frac{1}{N_{\pi_1(s)}}\right)^M \sum_{c_1,\ldots,c_M \text{ s.t.} \pi_1(c_i)=\pi_1(s)} \Psi_\theta(c_1, \ldots, c_M, s, a),$$

where $N_{\pi_1(s)}$ is the number of card configurations c that satisfy $\pi_1(c) = \pi_1(s)$ and where

$$\Psi_\theta(c_1, \ldots, c_M, s, a) = \chi\left(F_a\left(\frac{\sum_{i=1}^M w(c_i, I(s))\lambda_\theta(c_i, s)}{\sum_{j=1}^M \lambda_\theta(c_j, s)}\right)\right).$$

Here $\chi(s) = \mathbb{I}(s \geq 0)$ is the characteristic function of $[0, \infty)$. Hence, in general $p_\theta(a \mid s)$ will not be differentiable w.r.t. θ (as noted in Section 2.4, the non-differentiability of the objective function does not need to create any major difficulties for SPSA). One possible remedy is to smooth the characteristic function in the above expression. Still, in this case, exact differentiation of the action-selection probability function would be computationally intractable. One idea to overcome this is to use sampling. In particular, it looks natural to reuse the same card configurations that define the value of $W_{M,\theta}(s, a)$. Unfortunately, this would result in an intolerably large bias. This is because for an action that is *actually* selected, by construction, $\Psi_\theta(C_1, \ldots, C_M, s, a) = 1$. This means that the score function value estimates based on $C_1, \ldots, C_M$ alone would always be severely biased towards zero. When another single card configuration set, say $C'_1, \ldots, C'_M$, were used, then the variance of the resulting estimate would be huge.[21] Hence, although it is certainly possible to use the LR method and there are other possibilities to make it work besides the one considered here, we remain dubious regarding the potential of LR-methods in this special case.

Let us now return to the description of the experiments. For problems where the number of parameters is small, FDSA is a natural competitor to SPSA. Therefore, we have tested SPSA and FDSA with and without RPROP (we shall call the combination of FDSA and RPROP

[21] Consider the case when χ is replaced by a sigmoid, $\sigma(x) = 1/(1 + \exp(-\gamma x))$, γ controlling the approximation error. Since $\sigma'(x)/\sigma(x) = \gamma(1 - \sigma(x))$, we get that the score function scales with γ. In fact, when $C'_1, \ldots, C'_M$ is such that $\Psi_\theta(C'_1, \ldots, C'_M, s, a) = 0$, then the value of the score function will scale with γ, making the estimate's variance large when γ is big. Sampling a large number of card configurations is not feasible, either, due to the increase in computational complexity.

Table 1 Learning parameters of RSPSA and RFDSA for opponent model (OM), RSPSA and TD for evaluation function (EF) and RSPSA for policy (POL) learning. $\eta+$, $\eta-$, δ_0 (the initial value of δ_{ti}), δ^- and δ^+ are the RPROP parameters; ρ is the RSPSA (or RFDSA) coupling factor, λ is the parameter of TD; $batchsize$ is the number of performance evaluations (games) within an iteration which, for RSPSA and RFDSA, is equal to the product of the number of perturbations (r), the number of directions (2) and the number of evaluations per perturbation (q).

	$\eta+$	$\eta-$	δ_0	δ^-	δ^+	$\rho(\lambda)$	*batchsize*
RSPSA (OM)	1.1	0.85	0.01	1e-3	1.0	1	$40 \times 2 \times 250$
RFDSA (OM)	1.1	0.85	0.01	1e-3	1.0	1	$6 \times 2 \times 1500$
RSPSA (EF)	1.2	0.8	0.05	1e-3	1.0	10/7	$100 \times 2 \times 100$
RSPSA (POL)	1.1	0.9	0.01	1e-3	1.0	10/3	$100 \times 2 \times 100$
TD (EF)	1.2	0.5	0.1	1e-6	1.0	0.9	10000

'RFDSA'). Despite all efforts early experiments with SPSA and FDSA (without RPROP) failed to produce acceptable results: Hence, we give only the results for RSPSA and RFDSA here.

In the process of estimating the derivatives we employed antithetic dealing and CRNs. Further, the following *deck reuse procedure* was implemented: Remember that the CRN principle dictates using the same decks when evaluating the two opposite perturbations. As a result, many of the decks will produce zero SPSA differences, thus producing zero contribution to the estimate of the gradient. Therefore, those relatively infrequent decks that resulted in non-zero differences were saved for reuse. In subsequent steps, half of the decks used for a new perturbation were taken from those previously stored, whilst the other half was generated randomly. Reuse was based on recency, so as to ensure that no decks persist for longer periods.[22] We note that the reuse method obviously introduces bias in the gradient estimates. Though, in theory, we could correct for this bias, it turns out that the bias introduced is sufficiently benign so that we do not care about it. This is because under reasonable conditions the bias introduced can be modelled as the multiplication of the SPSA differences by a positive constant factor (larger than 1) that depends on the probability of observing non-zero SPSA differences given a random card configuration. Hence, when this factor is constant or varies slowly as a function of the optimised parameters, we expect that its effect on the optimisation will be negligible. Actually, in the case of the RPROP based updates where the update depends only on the sign of the gradient multiplying the gradient by a positive factor larger than one is actually beneficial.

The parameters of the algorithms that were hand-tuned by running some tests for each algorithm considered, are given in the first two lines of Table 1 (rows labelled by RSPSA(OM) and RFDSA(OM)). Attention was paid to allocate the same amount of time to the tuning of the parameters of the various algorithms (including SPSA and FDSA). The experiments were conducted on a cluster of 16 computers that were used previously in the development of McRaise. Since during this development the most time consuming task was the evaluation of the new variants against the previous ones (remember that simulating 5,000 games takes

[22] The decision to use half of the saved decks for reuse is admittedly ad-hoc. Obviously, no reuse increases variance for reasons described previously and full reuse introduces serious bias. Hence reusing half of the saved decks looked like a natural idea. The recency-based strategy is preferred to random resampling from the saved decks for the same reason that in particle filters residual resampling is preferred to random resampling. For a discussion of these issues see (Douc, Cappé & Moulines, 2005).

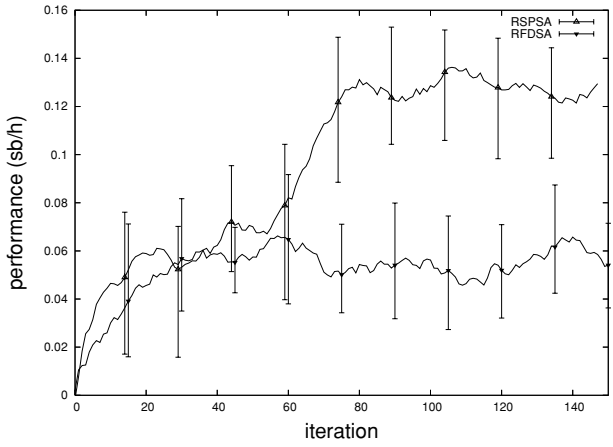

Fig. 5 Learning curves for RSPSA and RFDSA as a function of the number of iteration. The graphs are obtained by smoothing the observed performance in windows of size 15. The error bars were obtained by dropping the smallest and largest values within the same windows centred around their respective ordinates

ca. one hour and 20 minutes on a *single* machine), a software-library was developed that parallelised the evaluation process. This library was reused in the experiments presented here. One unfortunate consequence of this design is that we had to use more than one evaluation per perturbation in the SPSA experiments, too. Actually, in order to reduce communication overhead, the number of evaluations per perturbation was kept above 100 in all the experiments. As mentioned earlier, 5,000 games were used to measure the performance of the iterates. The parameters of the opponent model were initialised to the parameter settings used in McRaise.

The evolution of the performance for the two algorithms is plotted in Fig. 5 against the number of iterations. The best performance obtained for RSPSA was + 0.170 sb/h, whilst it was + 0.095 sb/h in the case of RFDSA. Since the performance of the program that uses the RSPSA-tuned parameters is almost twice as good as that of the program that uses the parameters tuned by RFDSA, we conclude that despite the small number of parameters, RSPSA is the better choice here. This is a somewhat surprising result, especially in light of the discussion of the transient behaviour of FDSA and SPSA (Section 2.5).

5.2. Learning policies and evaluation functions

As described previously, McRaise selects the action that has the best estimated value. This can be cast as a 1-ply search w.r.t. the so-called after-state evaluation function, V, defined as follows. Let $s' = T(s, a)$ be the situation right after action a is executed from situation s (and before the opponents would bet). Then $V(s')$ is defined by $-D(s, a) + V(s') = Q(s, a)$. Here $D(s, a)$ is the cost of executing action a from s. Note that V is well-defined by the definition of Q. In the first set of experiments described here, V is replaced by a neural network, V_θ. The optimisation task is to tune the weights of this 'value-network' so as to yield an increase in the average payoff per game. Action selection is implemented via $\text{argmax}_a\,[-D(s, a) + V_\theta(T(s, a))]$, both during learning and when evaluating the learnt value-function. Note that due to the highly stochastic nature of poker, introducing explicit exploration seemed to be unnecessary. This approach proved to be successful earlier in learning to play backgammon (Tesauro, 1992).

Learning evaluation functions is by far the most studied learning task in games. One of the most successful algorithms for this task is TD(λ) (Sutton, 1988) and the best known example of successfully training an evaluation function is TDGammon (Tesauro, 1992). By some, the success of TD-learning in backgammon can mostly be attributed to the highly stochastic nature of this game. As poker is similarly stochastic, TD-algorithms might enjoy the same benefit in this domain, too. We note in passing that temporal-difference learning had some success in deterministic games as well, for example, TD-based parameter tuning contributed significantly to the success of the world champion Lines of Action program, MIA (Winands et al., 2002). In our experiment we use a similar design, combining TD(λ) with RRPOP, as the one that was the highly successful for tuning the evaluation function of MIA. (In fact, the idea of combining RPROP and SPSA was partly motivated by this earlier success.)

In the second set of experiments, action selection is done in a probabilistic manner as follows: a neural network ("action network") with three outputs, each associated with one action computes the scores for the three actions. These are then normalised to yield a probability distribution over the set of available actions (when raise is not available, then it is left out in this step). The next action is then sampled from this distribution. Tuning the weights of the action-network can be thought thus implements a form of policy search.

Inputs to the neural networks include the estimate of the expected proportional payoff, the strength of the player's hand (i.e., the *a priori* chance of winning), the position of the player within the round, the pot size, the current bet level and some statistics about the recent betting actions of the opponent. After some initial experimentation, the network architectures were fixed at 12-10-1 and 12-10-3 for the value- and the action-networks, respectively.

When the parameters of the action network are optimised, the objective function will be differentiable and the LR method applies, though, again, since the EPP-estimate, $W_{M,\theta}(s, a)$, is an input of the network, the exact computation of the score functions is intractable. We leave it for future work to compare the efficiency of LR-based policy-gradient methods to the methods tested here. In the case when the after-state value function is represented by a neural-network, we face the same difficulties as those encountered in the previous section, as far as the differentiability of the objective function is concerned. Note that TD-methods avoid differentiability issues as they optimise for another criterion.

The parameters of the algorithms are given in the second half of Table 1. Again, these parameters were obtained by experimenting with the algorithms and selecting the best parameters found. Attention was paid to dedicate the same amount of time for the tuning of the various algorithms. For RSPSA the same enhancements as those given in Section 5.1 were used. We tested four algorithms:

(1) RSPSA for tuning the parameters of the after-state evaluation function (RSPSA(EF)),
(2) RSPSA for tuning the action-network (RSPSA(POL)),
(3) TD for tuning an evaluation function (TD(EF)), and
(4) TD for evaluation function tuning with a supervised start-up (spvTD(EF)).

For the latter, before TD-learning, a simple supervised algorithm was used to tune the weights of the value-network to match values estimated by McRaise at a set of states sampled using self-play.

The learning curves of the four algorithms are given in Fig. 6. The best performance obtained for RSPSA(EF) was +0.194 sb/h, for RSPSA(POL) it was +0.152 sb/h, for TD(EF) it was +0.015 sb/h, and for spvTD(EF) it was +0.220 sb/h. It is fair to say that TD performed better than RSPSA, which is a result one would expect given that TD uses more information about the gradient. However, we observe that for TD it was essential to start from a good

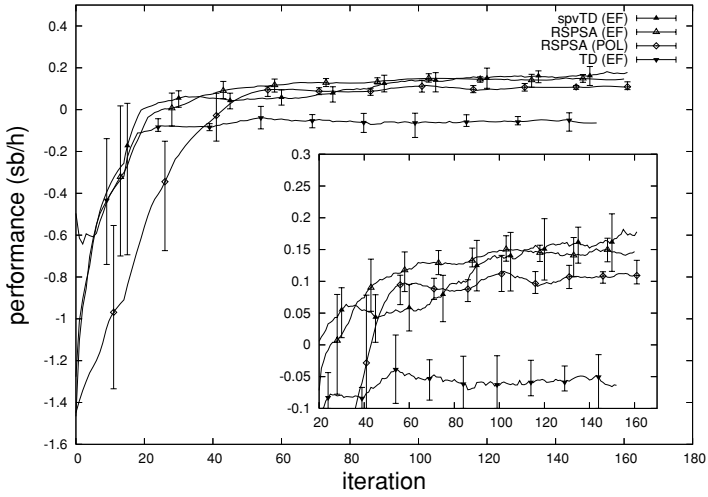

Fig. 6 Learning curves for RSPSA and TD as a function of the number of iteration. The graphs are obtained by smoothing the observed performance in windows of size 15. The error bars were obtained by dropping the smallest and largest values within the same windows centred around their respective ordinates. For an explanation of the symbols see the text

set of weights, those obtained by supervised learning. When started from random weights, the initial policy is probably too bad and learning gets stuck at a local minimum. This is in contrast to TDGammon that was able to learn starting from random weights (Tesauro, 1992).

We note that although the two RSPSA algorithms did not reach the performance obtained by the combination of supervised and TD-learning, they did give a considerable performance gain even though they were started from scratch. As noted earlier, a difference of 0.2 sb/h represents a significant difference in player-strength. Hence, given that McRaise is thought to be competitive with professional players, we think that the performance improvement achieved by RSPSA(EF) is noteworthy. Of course, since these results are obtained by training against a fixed opponent (though, a strong one), the resulting player should be tested against a wide range of players before making any definite conclusion about its playing strength. Even better, the experiments could be repeated by playing against a larger set of opponents.

We note in passing that we have also experimented with SPSA without the enhancements proposed here. Even if all the enhancements, except the combination with RPROP were used, no parameter settings were found using which SPSA could produce acceptable results in this case.

6. Conclusions and future work

This article investigated the value of a general purpose optimisation algorithm, SPSA, for the automatic tuning of game parameters. Several theoretical and practical issues were analysed, which in turn led to the design of a new variant of SPSA that we called RSPSA. RSPSA combines the strengths of RPROP and SPSA: SPSA is a gradient-free stochastic hill-climbing method that requires only function evaluations, while RPROP is a first order method that is known to improve the transient behaviour of gradient ascent. The proposed combination couples the perturbation parameter of SPSA and the step-size parameters of RPROP. It was

argued that this coupling is natural. By means of some preliminary experiments, it was shown that the combined method can indeed improve the convergence rate.

For achieving a good performance in our test domain, it proved to be essential to employ a number of other enhancements that aim at reducing the variance of the gradient estimates. The effect of performing a larger number of perturbations was analysed. An expression for the mean square error of the estimate of the gradient was derived as the function of the number of (noisy) evaluations of the objective function per perturbation (q) and the number of perturbations (r). It was found that to optimise the mean square error with a fixed budget $p = qr$, the number of perturbations should be kept at maximum.

We suggested that besides using the method of Common Random Numbers, antithetic variables should be used for a further reduction of the variance. In addition, a method for reusing decks that produced non-zero differences was proposed for the same purpose. In our test domain, these methods together are estimated to yield a speed-up by a factor larger than ten (since with the proposed methods a smaller number of function evaluations is enough to achieve the same level of accuracy in estimating the gradient). It was the overall effect of these enhancements that made it possible to apply SPSA-techniques for tuning the parameters of several variants of our poker playing program, McRaise.

In our experiments the optimisation of two components of McRaise were attempted: the opponent model and the action-selection algorithm. The latter task was attempted both directly, when the policy was represented explicitly, and indirectly via the tuning of the parameters of an after-state value function. In addition to testing RSPSA, for both components an alternative optimiser was tested (resp., RFDSA, and TD(λ)). On the task of tuning the parameters of the opponent model, RSPSA resulted in a significantly better performance as compared to that obtained by using RFDSA. This confirms some of the previous findings such as those of Spall, Kleinman, Spall and Neiman (1992, 1999), whilst it contradicts some expectations published elsewhere, such as in Kushner and Yin (1997) and Dippon (2003). In the case of policy optimisation, RSPSA was competitive with TD-learning, although the combination of supervised learning followed by TD-learning outperformed RSPSA. Nevertheless, the performance of RSPSA was encouraging on this second task, as well. In fact, using RSPSA, it was possible to achieve an impressive improvement of 0.194 sb/h in the case of tuning the action-network while playing against McRaise.

There are several lines of future research that look important. Extensive numerical studies would be needed to gain more insight into the behaviour of SPSA (and RSPSA) as a function of its parameters. In particular, we believe that the coupling of the RPROP step-size and the SPSA perturbation size can be improved by using more information such as the learning "stage" and the variance of the gradient. Also, our theoretical results indicate that p, the size of mini-batches should be selected to match the scale of the perturbation step-sizes. The optimisation of the size of the mini-batches was not attempted here, although we think that optimised batch-sizes may result in further performance improvements.

In connection to this let us note that there exist results that show that it is possible to use SPSA with deterministic perturbation sequences and with such sequences, in fact, it is possible to improve the transient behaviour (Xiong, Wang & Fu, 2002). It would be interesting to see if performance can be further improved using this technique. In this article, we have not compared the performance of the SPSA-based methods with that of LR (a.k.a., policy-gradient) methods. In cases when LR methods are applicable, they might have an advantage over SPSA-based methods as they use more information. Nevertheless, as we have discussed it extensively, LR methods are not without problems either. At present, it is largely unclear how LR methods would fare when compared with SPSA-based methods on the tasks considered here.

There exist other opportunities to enhance the convergence rate of SPSA. In fact, any adaptive step-size method could be used that is designed to enhance the performance of stochastic gradient ascent. One particularly appealing such algorithm extends conjugate gradient to a stochastic settings (Schraudolph & Graepel, 2002). Recently, Anastasiadis, Magoulas and Vrahatis (2005) introduced a globally convergent version of RPROP. It is an interesting open question if their modifications are effective in a stochastic settings, as well.

In order to get a better understanding of the behaviour and utility of RSPSA for game-program parameter tuning, it should be tested in several other games. A first indication that RSPSA can be successful in other games than poker is given in Kocsis (2005) where RSPSA is used for tuning the realisation probability weights of MIA in Lines of Action.

Regarding the poker environment, several components of McRaise could be improved. The opponent model, for example, could be made adaptive (or replaced entirely). Preliminary results where bandit algorithms are used to select the best opponent model from a fixed pool given past plays are reported in Kocsis and Szepesvári (2005). As we mentioned earlier, our experience so far suggests that deep searches do not yield a sizeable performance gain. However, foreseeing the future betting of the opponents plays an important role in human play, suggesting that higher gains should be attainable by an improved search algorithm. Currently an important weakness of the program might be its predictability. A potential solution to this problem is the game-theoretic approach (i.e., attempting to find Nash-equilibrium strategies) that proved successful in Texas Hold'em (Billings et al., 2003).

From the point of view of poker play another significant deficiency of the the experiments of the present paper is that training happened whilst playing against a *single* opponent. Such an approach may result in strategies that perform very poorly against some opponents. Training against a larger set of opponents would be a simple-minded and expensive solution. Ideas from bandit problems or active learning could be borrowed to improve the performance of this approach. We believe that substantially more work would be desirable to explore this exciting area.

Appendix A: Multiple perturbations vs. Multiple evaluations

In this section we provide a derivation of Eq. (7). Let

$$\delta f(\theta, Y, c\Delta) = f(\theta + c\Delta; Y) - f(\theta - c\Delta; Y).$$

Using elementary analysis, it can be shown that if f is three times continuous differentiable in a sufficiently large neighbourhood of θ then

$$\frac{\delta f(\theta, Y, c\Delta)}{2c\Delta_i} = f_i'(\theta; Y) + \sum_{\substack{j=1 \\ j\neq i}}^{d} f_j'(\theta; Y)\frac{\Delta_j}{\Delta_i} + O(c^2). \qquad (15)$$

Without the loss of generality we will consider the approximation of f_1' only. Consider

$$A_q = \frac{1}{q}\sum_{i=1}^{q}\left\{\frac{\delta f(\theta, Y_i, c\Delta)}{2c\Delta_1} - f_1'(\theta)\right\},$$

where $\Delta \in \mathbb{R}^d$ is a random variable such that its components are i.i.d., Y_i are i.i.d. random variables, and Δ is independent of $\{Y_1, \ldots, Y_q\}$. We assume just like Spall (1992) that Δ_i is symmetrically distributed around zero. We shall further assume that $| f_j'(\theta; Y) |$, $| \Delta_i |$ and $1/| \Delta_i |$ are bounded by some common deterministic upper bound $K > 0$ with probability one.[23]

Let $A_q^{(1)}, \ldots, A_q^{(r)}$ be independent realisations of A_q. Defining

$$A_{r,q} = \hat{g}_{r,q,1}(\theta) - f_1'(\theta)$$

it is clear that $A_{r,q} = (1/r) \sum_{j=1}^{r} A_q^{(j)}$. Hence,

$$\begin{aligned}\mathbb{E}\left[A_{r,q}^2\right] &= 1/r^2 \left(\sum_{j=1}^{r} \mathbb{E}\left[(A_q^{(j)})^2\right] + \sum_{j \neq j'} \mathbb{E}\left[A_q^{(j)} A_q^{(j')}\right] \right) \\ &= (1/r)\, \mathbb{E}\left[(A_q^{(1)})^2\right] + O(c^2),\end{aligned}$$

where we have used that by Lemma 1 of (Spall, 1992) $\mathbb{E}\left[A_q^{(j)}\right] = O(c^2)$ (this follows from Eq. (15)).

Therefore it suffices to consider $\mathbb{E}\left[A_q^2\right]$. Using Eq. (15) we get

$$A_q = \frac{1}{q} \sum_{i=1}^{q} \sum_{j=2}^{d} f_j'(\theta, Y_i) \frac{\Delta_j}{\Delta_1} + \frac{1}{q} \sum_{i=1}^{q} (f_1'(\theta, Y_i) - f_1'(\theta)) + O(c^2).$$

Denoting the first and second terms on the right hand side by V_q and W_q, respectively, we get $\mathbb{E}[A_q^2] = \mathbb{E}[V_q^2] + \mathbb{E}[W_q^2] + O(c^2)$, where the cross term $\mathbb{E}[V_q W_q]$ cancels because Δ_j is independent of $\{Y_i\}$ and $\mathbb{E}[\Delta_j] = 0$. Now, standard calculations give

$$\mathbb{E}\left[W_q^2\right] = (1/q)\, \mathbb{E}\left[(f_1'(\theta, Y_1) - f_1'(\theta))^2\right]$$

and

$$\mathbb{E}\left[V_q^2\right] = D \sum_{j=2}^{d} \left\{ \left(1 - \frac{1}{q}\right) \mathbb{E}[f_j'(\theta, Y_1)^2] + \frac{1}{q} \mathbb{E}[f_j'(\theta, Y_1)]^2 \right\} + O(c^2),$$

where $D = \mathbb{E}[\Delta_1^2]\, \mathbb{E}[1/\Delta_1^2]$. Therefore

$$\begin{aligned}\mathbb{E}\left[A_{r,q}^2\right] = D \sum_{j=2}^{d} &\left\{ \left(\frac{1}{r} - \frac{1}{rq}\right) \mathbb{E}[f_j'(\theta, Y_1)^2] + \frac{1}{rq} \mathbb{E}[f_j'(\theta, Y_1)^2] \right\} \\ &+ \frac{1}{rq}\, \mathbb{E}\left[(f_1'(\theta, Y_1) - f_1'(\theta))^2\right] + O(c^2).\end{aligned}$$

This shows that if $p = r\,q$ is fixed then choosing $r = p$ yields the smallest mean square error since in this case the term multiplied by $(1/r - 1/(rq))$ cancels, whilst the other terms

[23] With some more work the condition on the boundedness on $| 1/\Delta_{ij} |$ could be replaced by $\mathbb{E}\left[|1/\Delta_{ij}|\right] < K$.

do not change. In this case the mean squared error of the approximation of the gradient becomes

$$\frac{1}{p}\mathbb{E}\left[\Delta_1^2\right]\mathbb{E}\left[1/\Delta_1^2\right]\left\{\sum_{j=2}^{d}\mathbb{E}\left[f_j'(\theta, Y_1)^2\right]+\mathbb{E}\left[(f_1'(\theta, Y_1)-f_1'(\theta))^2\right]\right\}+O(c^2),$$

which is the same as the expression given in Eq. (8).

Appendix B: Details of the experiments on the synthetic task

In this section we describe some of the details of the experiments on the 10-5-10 encoder problem (cf. Section 3.2.3). The architecture of the neural network is fixed by the problem. Noise was injected in the outputs as follows: For input $(0, \ldots, 0, 1, 0, \ldots, 0)$, we used $(Z_1, \ldots, Z_{i-1}, 1 - Z_i, Z_{i+1}, \ldots, Z_1 0)$, where the random variables Z_i are independent and uniformly distributed in the interval $[0, 0.5]$. Network weights were initialised using the same fixed seeds to small random weights.

In the experiments reported the following parameter values were used:

- Figure 1, SPSA, update rule step-size is 0.1, momentum 0.5
- Figure 2, SPSA, update rule step-size is 0.01, momentum 0.5
- Figure 3, RSPSA with no coupling, $\eta^+ = 1.05$, $\eta^- = 0.95$, $\delta_0 = 0.5$, $\delta^- = 1e-6$, $\delta^+ = 1.0$
- Figure 4, RSPSA with coupling, $\eta^+ = 1.05$, $\eta^- = 0.95$, $\delta_0 = 0.5$, $\delta^- = 1e-6$, $\delta^+ = 1.0$ (same values as used when RSPSA was used with no coupling).

Acknowledgments The authors wish to thank the reviewers for their many useful suggestions and remarks. Specifically, we would like to acknowledge one of the reviewers for suggesting the parameter optimisation example for MMORPGs that was described in the introduction.

We would like to acknowledge support for this project from the Hungarian National Science Foundation (OTKA), Grant No. T047193 (Cs. Szepesvári) and from the Hungarian Academy of Sciences (Cs. Szepesvári, Bolyai Fellowship).

References

Anastasiadis, A. D., Magoulas, G. D., & Vrahatis, M. N. (2005). New globally convergent training scheme based on the resilient propagation algorithm. *Neurocomputing*, *64*, 253–270.

Andradóttir, S. (1998). A review of simulation optimization techniques. In *Proceeding of the 1998 Winter Simulation Conference* (pp. 151–158).

Baird, L. & Moore, A. W. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11* (pp. 968–974). Cambridge MA: MIT Press.

Baxter, J., & Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.

Baxter, J., Tridgell, A., & Weaver, L. (2000). Learning to play chess using temporal differences. *Machine Learning*, *40*(3), 243–263.

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., & Szafron, D. (2003). Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence* (pp. 661–668).

Billings, D., Davidson, A., Schaeffer, J., & Szafron, D. (2002). The challenge of poker. *Artificial Intelligence*, *134*, 201–240.

Billings, D., Davidson, A., Shauenberg, T., Burch, N., Bowling, M., Holte, R., Schaeffer, J., & Szafron, D. (2004). Game tree search with adaptation in stochastic imperfect information games. In *Proceedings of Computers and Games (CG'04)*.

Björnsson, Y., & Marsland, T. A. (2003). Learning extension parameters in game-tree search. *Journal of Information Sciences*, *154*, 95–118.

Blum, J. R. (1954). Multidimensional stochastic approximation methods. *Annals of Mathematical Statistics*, *25*, 737–744.

Bowling, M., & Veloso, M. (2002). Scalable learning in stochastic games. In *AAAI Workshop on Game Theoretic and Decision Theoretic Agents*.

Chellapilla, K., & Fogel, D. B. (1999). Evolving neural networks to play checkers without expert knowledge'. *IEEE Transactions on Neural Networks*, *10*(6), 1382–1391.

Chen, H. (1988). Lower rate convergence for locating a maximum of a function. *Annals of Statistics*, *16*, 1330–1334.

Dippon, J. (2003). Accelerated randomized stochastic optimization. *Annals of Statistics*, *31*(4), 1260–1281.

Douc, R., Cappé, O., & Moulines, E. (2005). Comparison of resampling schemes for particle filtering. In *4th International Symposium on Image and Signal Processing and Analysis (ISPA)*.

Fabian, V. (1968). On asymptotic normality in stochastic approximation. *Annals of Mathematical Statistics*, *39*, 1327–1332.

Gerencsér, L., Hill, S. D., & Vágó, Z. (1999). Optimization over discrete sets via SPSA. In *Proceedings of the 1999 Winter Simulation Conference* (pp. 466–470).

Gerencsér, L., Kozmann, G., & Vágó, Z. (1998). Non-smooth optimization via SPSA. In *Proceedings of the Conference on the Mathematical Theory of Networks and Systems MTNS 98* (pp. 803–806).

Glasserman, P., & Yao, D. D. (1992). Some guidelines and guarantees for common random numbers. *Management Science*, *38*, 884–908.

Greensmith, E., Bartlett, P. L., & Baxter, J. (2002). Variance reduction techniques for gradient estimates in reinforcement learning. In *Advances in Neural Information Processing Systems 14* (pp. 1507–1514).

He, Y., Fu, M. C., & Marcus, S. I. (2003). Convergence of simultaneous perturbation stochastic approximation for nondifferentiable optimization. *IEEE Transactions on Automatic Control*, *48*, 1459–1463.

Igel, C., & Hüsken, M. (2000), Improving the Rprop learning algorithm. In H. Bothe, & R. Rojas (Eds.), *Proceedings of the second international ICSC symposium on neural computation (NC 2000)* (pp. 115–121). ICSC Academic Press.

Igel, C., & Hüsken, M. (2003). Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, *50*(C), 105–123.

Kakade, S., & Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)* (pp. 267–274).

Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, *23*, 462–466.

Kleinman, N. L., Spall, J. C., & Neiman, D. Q. (1999). Simulation-based optimization with stochastic approximation using common random numbers. *Management Science*, *45*(11), 1570–1578.

Kocsis, L. (2003). Learning search decisions. Ph.D. thesis, Universiteit Maastricht, The Netherlands.

Kocsis, L., & Szepesvári, Cs. (2005). Reduced-variance payoff estimation in adversarial bandit problems. In *Proceedings of the ECML'05 Workshop on Reinforcement Learning in Non-Stationary Environments* (in print).

Kocsis, L., Szepesvári, Cs., & Winands, M. H. M. (2005). RSPSA: Enhanced parameter optimisation in games. In *Proceedings of the 11th Advances in Computer Games Conference (ACG-11)*, in press.

Kushner, H. J., & Yin, G. G. (1997). *Stochastic approximation algorithms and applications*. New York: Springer.

L'Ecuyer, P., & Yin, G. (1998). Budget-dependent convergence rate of stochastic approximation. *SIAM J. on Optimization*, *8*(1), 217–247.

Polyak, B. T., & Tsybakov, A. B. (1990). Optimal orders of accuracy for search algorithms of stochastic optimization. *Problems of Information Transmission*, *26*, 126–133.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning The RPROP algorithm. In E. H. Ruspini (Eds.), *Proceedings of the IEEE international conference on neural networks* (pp. 586–591). IEEE Press.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, *22*, 400–407.

Rubinstein, R. Y., Samorodnitsky, G., & Shaked, M. (1985). Antithetic variables, multivariate dependence and simulation of complex stochastic systems. *Management Sciences*, *31*, 66–77.

Sadegh, P. & Spall, J. C. (1997). Optimal random perturbations for stochastic approximation using a simultaneous perturbation gradient approximation. In *Proceedings of the American Control Conference*, Albuquerque, NM (pp. 3582–3586).

Schraudolph, N. (1999). Local gain adaptation in stochastic gradient descent. In *Proc. 9th International Conference on Artificial Neural Networks*, Edinburgh (pp. 569–574). London: IEE.

Schraudolph, N. N. & Graepel, T. (2002). Towards stochastic conjugate gradient methods. In *Proceedings of the 9th International Conference on Neural Information Processing* (pp. 1351–1358).

Spall, J. C. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, *37*, 332–341.

Spall, J. C. (2000). Adaptive stochastic approximation by the simultaneous perturbation method. *IEEE Transactions on Automatic Control*, *45*, 1839–1853.

Spall, J. C. (2003). *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, Hoboken, NJ: Wiley.

Sutton, R. & Barto, A. (1998). *Reinforcement learning: An introduction.* Bradford Book.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12* (pp. 1057–1063), MIT Press, Cambridge MA.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277.

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

Winands, M. H. M., Kocsis, L., Uiterwijk, J. W. H. M., & Van den Herik, H. J. (2002). Temporal difference learning and the neural movemap heuristic in the game of lines of action. In *Proceedings of 3rd International Conference on Intelligent Games and Simulation (GAME-ON 2002)* (pp. 99–103).

Xiong, X., Wang, I.-J., & Fu, M. C. (2002). Randomized-direction stochastic approximation algorithms using deterministic sequences. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA (pp. 285–291).