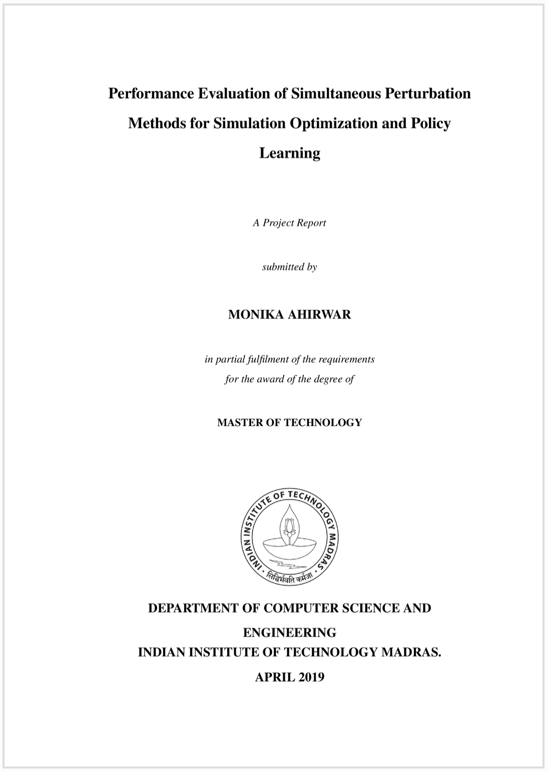
Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Monika Ahirwar
Assignment title:	Final M.Tech Project REport
Submission title:	mtech thesis
File name:	ThesisFinal.pdf
File size:	1.3M
Page count:	49
Word count:	9,717
Character count:	50,122
Submission date:	27-Apr-2019 09:31PM (UTC+0530)
Submission ID:	1115914415



Performance Evaluation of Simultaneous Perturbation Methods for Simulation Optimization and Policy Learning

A Project Report

submitted by

MONIKA AHIRWAR

in partial fulfilment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

APRIL 2019

Performance Evaluation of Simultaneous Perturbation Methods for Simulation Optimization and Policy Learning

A Project Report

submitted by

MONIKA AHIRWAR

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

APRIL 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Performance Evaluation of Simultaneous Perturbation Methods for Simulation Optimization and Policy Learning**, submitted by **Monika Ahirwar**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Prashanth L.A.
Research Guide
Assistant Professor
Dept. of Comp. Sc. & Engg.
IIT-Madras, 600 036

Place: Chennai

Date: 7th April 2019

ACKNOWLEDGEMENTS

All of the work described in this thesis was done in collaboration with my guide, Dr. Prashanth L.A., who has continually pointed me in the right direction and provided inspiration to do my best work. I would like to express my sincere gratitude and a deep sense of respect for his consistent support and guidance throughout this project. Furthermore, I would like to thank him for helping me understand the concepts of simulation optimization. Pursuing his courses on Linear Algebra and Random Processes and Reinforcement Learning were really helpful for my project. Also, his constant feedback on project report has helped me improved my writing skills. Finally, I am grateful for being provided with valuable facilities like dedicated PC in DON lab, which made my work easier.

I would also like to express my gratitude for Dr. Anurag Mittal for being an extremely supportive faculty adviser by providing valuable advise, timely support and feedback. Furthermore, I would like to thank my course instructors Dr. Deepak Khemani and Dr. Mitesh Khapra for helping me understand concepts in AI and Deep Learning.

Finally, this thesis is dedicated to my parents, for all the years of love and support, and for doing so much to further my education. Also, I would like to thank my friends Prasanna, Monika, Kanchan and Nitesh for their support, encouragement and inputs during this journey.

ABSTRACT

KEYWORDS: Stochastic optimization, Simultaneous Perturbations (SP), SPSA, RDSA, Kiefer Wolfowitz algorithm, Reinforcement learning, REINFORCE, Policy gradient

Gradient based methods are popular due to their ease of implementation in high dimensional problems. Simultaneous perturbation (SP) technique is one such method that is used in estimating gradients in a simulation optimization problem where the goal is to minimize a function given noisy observations. Due to its black-box optimization nature and two function measurements per iteration, it has applications in solving high-dimensional reinforcement learning (RL) problems.

In the first part of the thesis, we study several SP algorithms and provide a detailed comparative performance evaluation on three multimodal functions with increasing dimensions. The second part of the thesis focuses on implementing SP methods on RL problems. For implementation of these problems, we are using two SP methods: SPSA and RDSA along with a neural network based function approximator. Further, we analyze these algorithms on common discrete and continuous control environments and compare performance with the popular REINFORCE algorithm. The experimental studies show that SPSA i) is easy to implement ii) takes less time in training iii) requires two function measurements per iteration and iv) outperforms REINFORCE in walking robot task.

Contents

List of Tables

List of Figures

ABBREVIATIONS

SP	Simultaneous Perturbation
SPSA	Simultaneous Perturbation Stochastic Approximation
RDSA	Random Direction Stochastic Approximation
RL	Reinforcement Learning
DNN	Deep Neural Network
DP	Deterministic Perturbations
RP	Random Perturbations
KW	Kiefer Wolfowitz
MDP	Markov Decision Process
DFO	Derivative Free Optimization
PG	Policy Gradient

Chapter 1

INTRODUCTION

1.1 Motivation and Overview

Gradient based methods are popular due to their ease of implementation in high dimensional problems. Simultaneous perturbation (SP) technique is one such method that is used in estimating gradients in a simulation optimization problem where the goal is to minimize a function given noisy observations. Due to its black-box optimization nature and two function measurements per iteration, it has applications in solving high-dimensional RL problem.

An RL problem consists of an agent and an environment where the agent acts on the environment to get a scalar reward. The task of the agent is to learn an optimal policy - a mapping from states to action, which maximizes the expected sum of rewards. This can also be viewed as an optimization problem where the goal is to maximize the expected total reward with respect to the parameters of the policy. The methods used to solve RL problems of this form are called policy optimization methods. Policy Gradient (PG) is one popular policy optimization method which searches for a local maximum in the objective function by ascending the gradients of policy with respect to the parameters. But the back-propagation of gradients in PG takes time for training high-dimensional models. Hence, we are considering alternative approaches such as SP methods, which are black-box in nature and derivative free. In this thesis, we are using two SP methods: Simultaneous Perturbation Stochastic Approximation (SPSA) and Random Direction Stochastic Approximation (RDSA). These algorithms require only two function measurements per iteration, irrespective of dimension. Moreover, to improve performance of these algorithms in RL problem, we used a neural network based function approximator for parameterization of policy.

In the first part of the thesis, we study several SP algorithms and provide a detailed comparative performance evaluation on three multimodal functions with increasing dimensions. The second part of thesis focuses on implementing SP methods on a RL problem. We used SP methods studied in first part of thesis, mainly SPSA and RDSA to solve RL problem using a neural network function approximator. We analyze these algorithms on common discrete and continuous control environments and compare their performance with REINFORCE algorithm.

1.2 Outline of chapters

Chapter 2 provides the background on stochastic optimization and a detailed study on simultaneous perturbation techniques. We consider only first order SPSA and RDSA algorithms in our experiments. There are two main variants of first order SPSA, one with a random perturbation vector and the other with deterministic perturbations. Similarly, there are many stochastic and deterministic perturbation variants of RDSA. We aanalyze the algorithms on three problems with a multimodal objective function. From the results, we observe that all the algorithms converge differently in high dimensional setting, random perturbations work in a better manner. Also, deterministic perturbation algorithms outperform random perturbation for low-dimension setting.

Chapter 3 provides the background on policy optimization in an RL environment. We consider simultaneous perturbation techniques mainly SPSA and RDSA with random perturbations as discussed in Chapter 2 for optimization of policies. Moreover, we use a neural network function approximator for parameterization of policies. We analyze these algorithms on common discrete and continuous control environments and compare performance with REINFORCE in similar setting. Experimental studies shows that SPSA outperforms REINFORCE in walking robot task and converges with low fluctuations for continuous action space.

Chapter 4 provides the concluding remarks and also discusses some important future research directions.

Chapter 2

SIMULATION OPTIMIZATION USING SP METHODS

2.1 Introduction

A general optimization problem can be stated as:

$$min_{\theta \in \mathcal{C}} J(\theta). \tag{2.1}$$

Let J be the objective function, $\theta \in \mathcal{R}^p$, C is a compact and convex set. If J and its higher order derivatives are perfectly known, then the problem is deterministic in nature. Furthermore, one can use this information to deterministically compute the gradient and perform an incremental search to find θ^*. On the other hand, in simulation optimization problems, only noisy observations of the objective function are available and the objective gradient is not directly available. More precisely, $J(\theta)$ is obtained as $J(\theta) = E_\zeta[h(\theta, \zeta)]$, where, $E_\zeta[.]$ is the expected value over noisy observations of samples $h(\theta, \zeta)$ with random noise ζ. Gradient Descent is a popular solution approach for solving (1). Here, one performs incremental search for θ^* as follows:

$$\theta_{k+1} = \theta_k - a_k \nabla J(\theta_k), \tag{2.2}$$

where, a_k is the step-size and $\nabla J(\theta_k)$ is the gradient estimate. One of the oldest algorithms for estimating gradients under noisy estimates is Finite Difference Stochastic Approximation (FDSA), also known as Kiefer Wolfowitz algorithm Spall [2005]. The gradient estimate is of this form:

$$\nabla J(\theta_k) = E\left[\frac{h(\theta_k + \delta_k e_i, \zeta^+) - h(\theta_k - \delta_k e_i, \zeta^-)}{2\delta_k}\middle| \theta_k\right], i = 1 \dots p, \tag{2.3}$$

where perturbation parameter $\delta_k \leftarrow 0$ as $k \leftarrow \infty$, ζ^+, ζ^- are i.i.d. and e_i is the unit vector with 1 in the i^{th} place, and 0 elsewhere. The main drawback of this algorithm is that it requires $2p$ function measurements per k^{th} update. Thus, it is very hard to use for a large system such as for Neural Network which commonly involves the estimation of hundreds of parameters (weights). Using two measurements for each parameter, FDSA requires hundreds of measurements for a single iteration. Therefore, alternatives such as SPSA and RDSA have been analysed that aim to produce a fewer measurements to achieve a solution.

Simultaneous Perturbation Stochastic Approximation (SPSA)Spall [2005] and Random Direction Stochastic Approximation (RDSA) Kushner and Clark [1978] require two function measurements per iteration, irrespective of dimension. Even the basic version of SPSA algorithm converges fast for large number of variables S. Bhatnagar and Prashanth [2013]. To reduce further function measurements, one function measurement version of SPSA has been experimented to work for certain class of problems Spall [1997] but it doesn't work well in applications because of large bias in gradient estimate. Higher order variants of SPSA, in particular, second order has proved to have good convergence and efficiency. Moreover, in practical applications, SPSA has been implemented in service systems L.A. Prashanth and Desai [2015] and road traffic control Prashanth [2013]. Hence, we will be using it as baseline for comparison with other algorithms.

Similar to SPSA, RDSA variants such as RDSA-unif and RDSA-AB use only two function measurements L.A. Prashanth *et al.* [2017]. The only difference between the two variants is that RDSA-unif employs uniform distribution parameters to choose the random perturbations whereas RDSA-AB make use of asymmetric Bernoulli distribution. These algorithms are proved to have asymptotic unbiasedness of both gradient and Hessian estimates as well as asymptotic (strong) convergence. Also, these algorithms are analyzed under various noise parameter settings and for different objective functions L.A. Prashanth *et al.* [2017]. Our objective in this report is to observe performance of these algorithms on a multimodal function, and also compare the random perturbations based SP algorithms counterparts.

Recently, a class of RDSA algorithms with deterministic perturbations referred to as RDSA-DP algorithms, have been proposed in L.A. Prashanth *et al.* [2018]. The main algorithms in this class are RDSA-Lex-DP and RDSA-Perm-DP. RDSA-Lex-DP incorporates deterministic perturbations that are based on semi-lexicographic sequence, whereas, RDSA-Perm-DP, uses permutation matrix based sequence. Both approaches can be shown to have asymptotically unbiased gradient estimates. In contrast to random perturbation RDSA variants L.A. Prashanth *et al.* [2017], the generation of deterministic perturbations require additional storage and the function measurements depend upon dimensions. For RDSA-Lex-DP, the dependency is exponential whereas, RDSA-Perm-DP has linear dependency. In this work, we will analyze their performance in various dimensions and compare them with random perturbation-based algorithms.

In order to examine convergence rates of SPSA and RDSA algorithms discussed above, all the variants are implemented with a multimodal objective function, the latter is used as a benchmark in simulation optimization Miller and Shaw [1995], Xu and Hong [2010]. Best parameter settings for convergence are obtained by numerical experiments. Finally, we compare their performance under various dimensions. The final results are obtained by averaging over many replications.

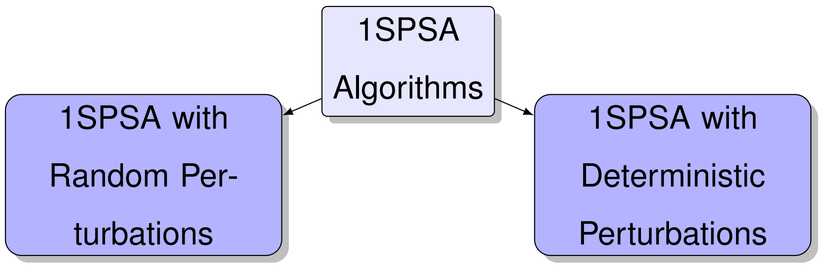
2.2 First Order Methods

In this section, we are considering first order SPSA and RDSA algorithms. In Kiefer-Wolfowitz scheme discussed above, all parameter perturbations are performed along each co-ordinate direction separately. In SPSA and RDSA, these perturbations are varied differently and are explained in detail in this section.

2.3 Simultaneous Perturbation Stochastic Approximation (SPSA)

In this algorithm, all component directions are perturbed simultaneously using perturbation vector and the gradient estimate requires only two function measurements. There are 2 main variants of 1SPSA, one with random perturbation vector and the other with deterministic perturbations.

Variants of 1SPSA Algorithms


1SPSA Algorithms
1SPSA with Random Perturbations
1SPSA with Deterministic Perturbations

2.3.1 1SPSA with Random Perturbations

In this algorithm, all component directions are perturbed simultaneously using a perturbation vector composed of symmetric, zero-mean, values +1 or -1 and Bernoulli i.i.d. random variables. The cost objective function for 1SPSA is $J(\theta) = E_\zeta[h(\theta, \zeta)]$, where $h(\theta, \zeta)$ is noisy measurement of $J(\theta)$, and ζ is a zero-mean random variable that corresponds to the noise in the measurement. θ is a p-dimensional vector. The gradient estimate $\nabla J(\theta)$ is obtained as follows:

$$\nabla J(\theta_k) = E\left[\frac{h(\theta_k + c_k\Delta_k, \zeta^+) - h(\theta_k - c_k\Delta_k, \zeta^-)}{2c_k\Delta_k}\middle|\theta_k\right], \tag{2.4}$$

where noise samples ζ^+ and ζ^- are martingale sequence and k $\geq$ 0. The expectation is taken over ζ^+, ζ^- and perturbation random vector $\Delta_k = (\Delta_{k1}, \dots, \Delta_{kp})$.
The update rule for this algorithm is as follows:

$$\theta_{k+1} = \theta_k - a_k \nabla J(\theta_k), \tag{2.5}$$

where c_k is a positive scalar and a_k is step size.
As recommended in Spall [2005], the parameters c_k and a_k are chosen as follows :

$$a_k = \tfrac{a}{(k+1+A)^\alpha}, c_k = \tfrac{c}{(k+1)^\gamma},$$

where a, A, c, α, γ are constants (see Section-III for details).

Since numerator is same in all p-components, hence the number of loss measurements to estimate the gradient is two, regardless of dimension p. Algorithm 1 represents psuedocode for 1SPSA Algorithm.

Algorithm 1 Basic 1SPSA Algorithm

```
Input  : Initialize $\theta_0$;
         Set simulation budget (M), number of replications(R);
         Choose step size $a_k$ and $c_k$;
         Bernoulli(p), a random independent Bernoulli $\pm$ sampler with probability p
         for +1 and 1-p for -1;
         $h(\theta, \zeta)$, noisy measurement of objective function J;
Output: Estimate of Optimal $\theta_M$
repeat
    $k \leftarrow 0$;
    repeat
        forall $i \leftarrow 1$ to $p$ do
            $\Delta_{ki} \leftarrow Bernoulli(1/2)$;
        end
        $Y^+ \leftarrow h(\theta + c_k\Delta, \zeta^+)$;
        $Y^- \leftarrow h(\theta - c_k\Delta, \zeta^-)$;
        forall $i \leftarrow 1$ to $p$ do
            $\theta_{(k+1)i} \leftarrow \theta_{ki} - a_k \frac{Y^+ - Y^-}{2c_k\Delta_{ki}}$;
        end
        $k \leftarrow k + 1$;
    until $k < M$;
    return $\theta$
until $r \leq R$;
```

Convergence Conditions: We make following assumptions for the convergence claim of the algorithm above. The assumptions below are common to the analysis of simultaneous perturbation methods, hence, all the algorithms that are discussed in the next section also employ similar assumptions.

- **Assumption 1** The map $J : R^p \rightarrow R$ is Lipschitz continuous, $\|\nabla J\| < K_0 < \infty$ where K_0 is some constant and is differentiable with bounded second order derivatives, $\|\nabla^2 J\| < K_1 < \infty$ (K_1 is some constant). Further, the map $L : R^p \rightarrow R^p$ defined as $L(\theta) = -\nabla J(\theta), \forall \theta \in R^p$ and the map $h : R^p X R^k \rightarrow R$ are both Lipschitz continuous.

- **Assumption 2** The step sizes $a_k, c_k > 0, \forall k$ and $a_k, c_k \rightarrow 0$ as $k \rightarrow 0$, $\sum_k (a_k/c_k)^2 < \infty$.

- **Assumption 3** $\zeta^+{}_k, \zeta^-{}_k, k \geq 0$ are R^k-valued, independent random vectors having a common distribution and with finite second moments. $\frac{\zeta^+{}_k - \zeta^-{}_k}{2c_k\Delta}$ forms a martingale sequence under an appropriate filtration.

- **Assumption 4** The random variables Δ_{ik}, $k \geq 0$, $i = 1, \ldots, p$, are mutually independent, mean-zero, have a common distribution and satisfy $E[(\Delta_{ik})^{-2}] \leq K, \forall k \geq 0$, for some $K < \infty$.

- **Assumption 5** The iterates (2.2) remains uniformly bounded almost surely, i.e., $sup_k \|\theta_k\| < \infty$, a.s..

- **Assumption 6** Let E denote the globally asymptotically stable equilibria (i.e. the local minima of J) of the ODE $\theta^{\cdot}(t) = -\nabla J(\theta(t))$, the E is a compact subset of $\mathcal{R}^p$.

Theorem 1: Under assumptions 1 - 6, the parameter update (3) satisfy $\theta_k \to E$ with probability one.

Thus, $\theta_k \to \theta^*$ a.s. as $k \to \infty$. For complete proof of convergence, please refer S. Bhatnagar and Prashanth [2013].

2.3.2 1SPSA with Deterministic Perturbations

In this algorithm, we perform deterministic perturbations. These deterministic perturbations are based on Hadamard matrices, by cyclically updating through a construction defined below.

Let H_{2^k}, $k \geq 1$ be Hadamard matrices of order $2^k \times 2^k$ that are recursively obtained as:

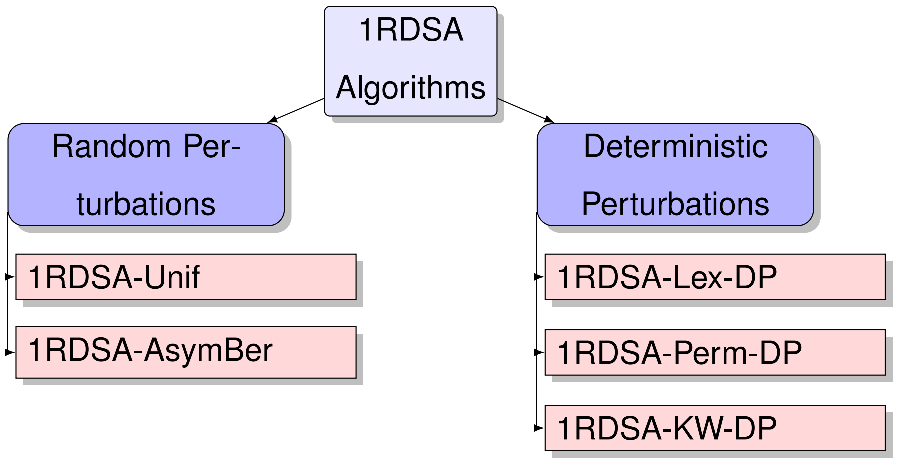
$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ and } H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}, k > 1.$$

Let $e(r)$, r = $1, \ldots, M$, be the M rows of $H_M{}'$. Set $\Delta_k{}^T$ = e (k mod M+1), $\forall n \geq 0$. The perturbation vectors Δ_k are formed again by cyclically going through the rows of $H_M{}'$ with $\Delta_0{}^T = e(1), \Delta_1{}^T = e(2), \ldots, \Delta_{M-1}{}^T = e(M), \Delta_M{}^T = e(1)$.

2.4 Random Direction Stochastic Approximation (RDSA)

Unlike 1SPSA, in 1RDSA, all component directions are perturbed using independent random vectors. 1RDSA algorithms with random perturbations such as 1RDSA-unif and 1RDSA-asymBer require only two function measurements per simulation, regardless of dimension. On the other hand, 1RDSA algorithms with deterministic perturbations require $2p$ function measurements per iteration where p is the dimension of θ.

Variants of 1RDSA Algorithm



2.4.1 1RDSA-Unif

In this variant, all component directions are perturbed using independent random vectors that are uniformly distributed over the surface of the $p-$dimensional unit sphere. Perturbation vectors are chosen as follows: d_k^i, $i = 1, \ldots, p$ to be independently identically distributed $U[-\eta, \eta]$ for some $\eta > 0$, where $U[-\eta, \eta]$ denotes the uniform distribution on the interval $[-\eta, \eta]$.
The gradient estimate $\nabla J(\theta)$ is obtained as follows:

$$\nabla J(\theta_k) = \frac{3}{\eta^2} d_k \left[\frac{h(\theta_k + c_k d_k, \zeta^+) - h(\theta_k - c_k d_k, \zeta^-)}{2c_k} \right], \tag{2.6}$$

where noise samples ζ^+ and ζ^- are martingale difference, sequence $\{d_k\}$ is independent of noise sequence and k $\geq$ 0.

The update rule for this algorithm is as follows:

$$\theta_{k+1} = \theta_k - a_k \nabla J(\theta_k). \tag{2.7}$$

2.4.2 1RDSA-AsymBer

In this variant, all component directions are perturbed using independent random that follows asymmetric Bernoulli distribution. In this case, we choose d_i^k $i = 1, \ldots, p$, i.i.d. as follows:

$$d_i^k = \begin{cases} -1 & w.p.\ \left(\frac{1+\epsilon}{2+\epsilon}\right) \\ 1+\epsilon & w.p.\ \left(\frac{1}{2+\epsilon}\right) \end{cases}, \tag{2.8}$$

where $\epsilon > 0$ is an arbitrary small constant. Note that, $\forall i = 1, \ldots, p : E[d_i^k] = 0$, $E[(d_i^k)^2] = 1 + \epsilon$.

The gradient estimate $\nabla J(\theta)$ is obtained as follows:

$$\nabla J(\theta_k) = \frac{1}{1+\epsilon} d_k \left[\frac{h(\theta_k + c_k d_k, \zeta^+) - h(\theta_k - c_k d_k, \zeta^-)}{2c_k} \right], \tag{2.9}$$

where noise samples ζ^+ and ζ^- are martingale difference, sequence $\{d_k\}$ is independent of noise sequence and k $\geq$ 0.

The update rule for this algorithm is as follows:

$$\theta_{k+1} = \theta_k - a_k \nabla J(\theta_k). \tag{2.10}$$

2.4.3 1RDSA-Lex-DP

This algorithm make use of semi-lexicographic sequence based perturbations which loops through a deterministic sequence to cancel out the bias in the gradient estimate. The regular RDSA achieves the same in expectation through zero-mean random perturbations. The deterministic sequence for a general N is obtained as follows:

Set d_1^1 = $\begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix}$ and apply the following recursion d_{k+1}^i = $\begin{bmatrix} d_k^{i-1} \\ d_k^{i-1} \\ d_k^{i-1} \end{bmatrix}$, $i = 2, \ldots, k+1$ to $N-1$ times to obtain d_N^i

The deterministic perturbation sequence loops through the rows in the matrix, say D_p, with columns d_p^i. Each column d_p^i, i = 1,. . ., p in D_p is of length 3^p. Further, in the first column of D_p, the first 2×3^k elements are -1 and the remaining 3^k elements are 2. On the other hand, the columns 2 through p in D_p are obtained from d_1^{p-1},. . .d$_{p-1}^{p-1}$, respectively by concatenating the d_{p-1}^i columns thrice.

For each k calculate g_m as

$$g_m = d_m \left[\frac{h(\theta_k + c_{k3^p+m} d_m, \zeta^+) - h(\theta_k - c_{k3^p+m} d_m, \zeta^-)}{2c_{k3^p+m}} \right], m = 1 \ldots 3^p - 1, \tag{2.11}$$

then gradient estimate is estimated as follows:

$$\nabla J(\theta_k) = \frac{1}{3^p} \sum_{m=0}^{3^p-1} g_m. \tag{2.12}$$

The update rule for this algorithm is as follows:

$$\theta_{k+1} = \theta_k - a_k \nabla J(\theta_k). \tag{2.13}$$

For any p, we require that

$$\frac{1}{2 \times 3^p} \sum_{m=0}^{3^p-1} d_m d_m^T = I_p. \tag{2.14}$$

to ensure that the gradient estimate above is asymptotically biased.

The disadvantage of this algorithm is that the inner loop runs for exponential in p, resulting in high time complexity.

2.4.4 1RDSA-Perm-DP

This approach is based on permutation matrix. A permutation matrix is a matrix whose rows are the rows of an identity matrix in some order. Let D_p be the permutation matrix and let d_m be a row of this matrix. For instance, the permutation matrix in 2 dimensions are of the following form:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

For each k, calculate g_m as

$$g_m = d_m \left[\frac{h(\theta_k + c_k d_m, \zeta^+) - h(\theta_k - c_k d_m, \zeta^-)}{2c_k} \right], m = 1 \ldots p, \tag{2.15}$$

then gradient estimate is estimated as follows:

$$\nabla J(\theta_k) = \sum_{m=0}^{p-1} g_m. \tag{2.16}$$

The update rule for this algorithm is as follows:

$$\theta_{k+1} = \theta_k - a_k \nabla J(\theta_k). \tag{2.17}$$

2.4.5 1RDSA-KW-DP

This is similar to 1RDSA-Perm-DP. The only difference is that identity matrix is used instead of permutation matrix. The gradient estimate and update rule is identical to 1RDSA-Perm-DP.

2.5 Experiments

2.5.1 Objective function

We are using smooth and continuous multimodal function F of Simopt defined by:

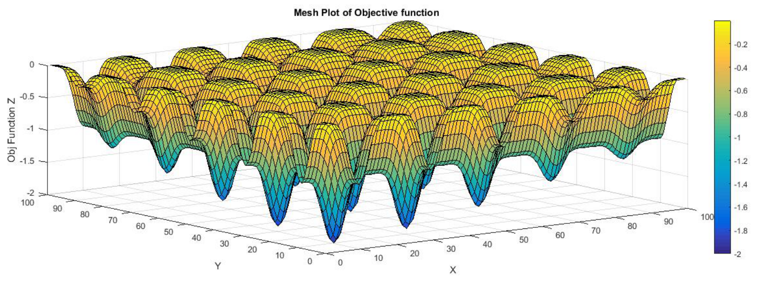
$$F = \frac{sin^6(0.05\pi x)}{2^{2(\frac{x-10}{80})^2}}, 0 \leq x \leq 100, \tag{2.18}$$

and the function $g(x_1, x_2)$ is defined as

$$g(x_1, x_2) = -\left(F(x_1) + F(x_2)\right) + \mathcal{N}(0, 0.3^2), 0 \leq x_1, x_2 \leq 100, \tag{2.19}$$

where $\mathcal{N}$ is a normal random variable with mean 0 and standard deviation 0.3 that is independent at different x's.

Starting Solution: Uniformly distributed in [0,100] X [0,100].
Optimal Solution: Global optimum at (10 ,10).
Known Structure: 25 widely spaced local optima.



The surface plot of minimization function for p = 2 dimension as shown above, has 25 local minima. The plot obtained is similar to the one in this paper Xu and Hong [2010]. It can be seen from the plot that the minimum value is -2 at optimal value of θ [10,10]. The optimal solutions for this problem have been evaluated for p = 2, 10 and 100 dimensions.

2.5.2 Implementation

Let us consider the following problem:

$$min_X E_\zeta[g(X,\zeta)] \tag{2.20}$$

where X is a p-dimensional and $g(X,\zeta)$ is a sample observation of the function $F(X)$ corrupted with Gaussian zero-mean noise $\zeta(\mathcal{N}(0,0.3^2))$ with variance as 0.09. The variance of noise has been kept constant through all settings.

The first order algorithms we implemented includes, 1SPSA, 1RDSA-Unif, 1RDSA-AsymBer, 1RDSA-Lex-DP, 1RDSA-Perm-DP and 1RDSA-KW-DP. Experiments were performed for p = 2, 10 and 100. For p = 2, all these algorithms were successfully implemented but for p = 10, we could implement all other algorithms except 1RDSA-Lex-DP. The reason is due to high running time of inner loop owing to 3^{10} iterations per simulation. As p was increased to 100, 1SPSA, 1RDSA-unif and 1RDSA-AsymBer converged in reasonable amount of time. On the other hand, the rest of the algorithms did not converge due to strong dependence on p.

In the experiments performed, the distribution parameters for 1RDSA variants are set as follows:

Method	p	Distribution Parameter	Value
1RDSA-U	2, 10, 100	η	1
1RDSA-AB	2, 10, 100	ϵ	0.25

The step size and gain coefficient which are of the forms $\frac{a_0}{k+1+A}$ and $\frac{0.3}{(1+k)^{0.101}}$ do not always converge in reasonable time, especially, in case of high simulation budget. This is due to terms vanishing at high k, and hence, gradient receives very few updates. So, upon experimenting with various parameter settings, we finally obtained convergence with following values:

Dimension (p)	Step Size (a)	Gain Coefficient (c)
2	$\frac{117.04}{k+125000}$	$\frac{0.3}{(1+k)^{0.101}}$
10	$\frac{117.04}{k+125000}$	$\frac{0.3}{(1+k)^{0.101}}$
100	0.1	0.1

For p = 100, constant values for a and c worked unexpectedly well. All the results are averaged over 50 replications for p = 2, 10 and 10 replications for p = 100.

To compare algorithms, we use Normalized Mean Square Error (NMSE) as performance metric. For a given simulation budget, the NMSE measures the distance between the final iterate obtained after the final update iteration and the optimum parameter θ^*.

$$\text{Normalized Mean square Error} = \frac{||\theta_k - \theta^*||}{||\theta_0 - \theta^*||},$$

where, k is the number of times θ is updated until the end of the simulation. k varies with the algorithm and the number of function measurements. For instance, algorithms 1SPSA, 1RDSA-unif, 1RDSA-Asymber with a simulation budget of 5000 have number of updates (k) as 2500, irrespective of p because the function measurement is two per iteration. However, in case of 1RDSA-Kw-DP and 1RDSA-Perm-DP, k is 250 as there are $2p$ function measurements per iteration. 1RDSA-Lex-DP is highly dependent on p giving exponential time complexity. We are using simulation budget with p as follows:

Dimension (p)	Simulation Budget
2	5000
10	25,000
100	10,000

Since step-size is an important factor for the convergence of these SA algorithms, we find the lower bound for step size constant a_0 so that the algorithm converges to an optimum.

Lower Bound on step-size: The lower bound of step-size for 1SPSA is calculated as follows

$$a_k = \frac{a_0}{k^{\alpha}},$$

$$\lambda_0 = \lambda_{min}(\nabla^2 f(x)),$$

$$\beta = \alpha - 2\gamma,$$

$$a_0 > \frac{\beta}{2\lambda_0},$$

Hessian matrix of this objective g: $g = -(f(x_1) + f(x_2))$, where

$$f(x) = \frac{sin(0.05\pi x)^6}{2^{2\frac{x-10}{80}^{2}}},$$

$$\begin{bmatrix} \frac{\partial^2 g}{\partial^2 x_1} & \frac{\partial^2 g}{\partial x_1 \partial x_2} \\ \frac{\partial^2 g}{\partial x_2 \partial x_1} & \frac{\partial^2 g}{\partial^2 x_2} \end{bmatrix} = \begin{bmatrix} h & 0 \\ 0 & h \end{bmatrix}.$$

After solving, at point [10,10] the Hessian matrix evaluated as :

$$\begin{bmatrix} 0.1485 & 0 \\ 0 & 0.1485 \end{bmatrix}.$$

The eigenvalue λ_0 is obtained as 0.1485.

We are taking α and γ as $\alpha = 0.602$, $\gamma = 0.101$ as used by Spall [2005],

$$\beta = \alpha - 2\gamma = 0.602 - 2 * 0.101 = 0.400,$$

$$a_0 > \frac{\beta}{2 * \lambda} = \frac{0.4}{2 * 0.1485} = 1.346.$$

The lower bound of step-size for 1RDSA is calculated as follows:

$$a_0 > \frac{1}{3\lambda_0} + 1,$$

$$a_0 > 3.24.$$

Thus, a_0 has been chosen such that it satisfies the lower bound.

2.5.3 Results

Figure 1 - 3 represent NMSE vs number of simulations plots for 1SPSA and 1RDSA variants for p = 2, 10 and 100. Tables I, II, III summarizes average NMSE error and standard error.

1. **p = 2**

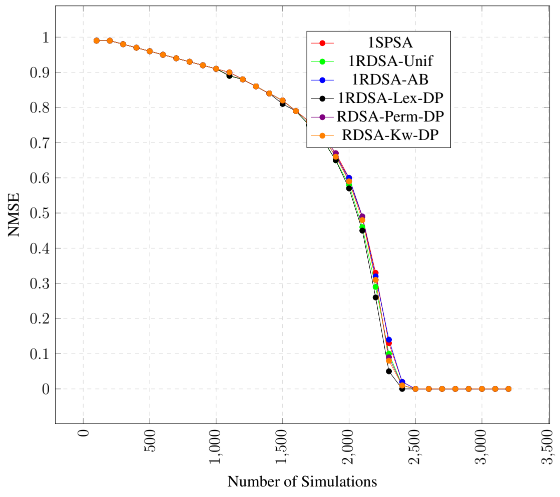


Figure 2.1: Comparison of convergence of 1SPSA & 1RDSA variants at p = 2

Table 2.1: NMSE for p = 2 with a simulation budget of 5000

Method	NMSE $\pm STD$
1SPSA	$6.41E^{-30}$ $\pm$1.88E^{-30}
1RDSA-U	$1.72E^{-32}$ $\pm$1.66E^{-32}
1RDSA-AB	$3.10E^{-30}$ $\pm$1.60E^{-30}
1RDSA-Lex-DP	$4.2919E^{-29}$ ± 0
1RDSA-Perm-DP	$5.87E^{-29}$ ± 0
1RDSA-Kw-DP	$2.07E^{-29}$ ± 0

Observation 1: As seen fron the Fig 1, all the algorithms, 1SPSA, 1RDSA-Unif, 1RDSA-AysmBer, 1RDSA-Lex-DP, 1RDSA-Perm-DP and 1RDSA-Kw-DP converge at the same rate. As expected, the NMSE error decreases first and then become constant after certain number of simulations. This experiment did not perform well with constant step-size and gain coefficient. As can be seen from Table-I, standard deviation across all replications is zero for RDSA-DP algorithms and the lowest NMSE error is observed for 1RDSA-Unif.

2. **p = 10**

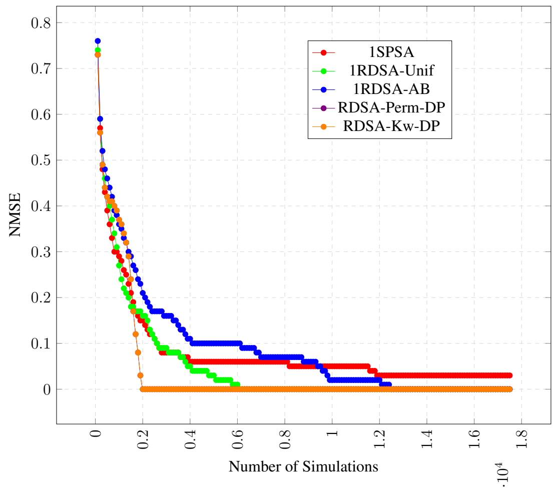


Figure 2.2: Comparison of convergence of 1SPSA and 1RDSA variants at p = 10

Table 2.2: NMSE for p = 10 with a simulation budget of 25000

Method	NMSE $\pm STD$
1SPSA	2.98E^{-02} $\pm$9.42E^{-02}
1RDSA-U	7.67E^{-31} $\pm$5.32E^{-31}
1RDSA-AB	2.86E^{-31} $\pm$1.75E^{-31}
1RDSA-Perm-DP	9.65E^{-30} $\pm$1.48E^{-45}
1RDSA-Kw-DP	9.65E^{-30} $\pm$1.48E^{-45}

Observation 2: We have implemented 1SPSA, 1RDSA-Unif, 1RDSA-AB, 1RDSA-Perm-DP and 1RDSA-Kw-DP for p = 10. Due to high time required for inner loop to run 3^{10} times, 1RDSA-Lex-DP is not implemented. 1SPSA has poor performance and highest error as compared to other algorithms. In this case, deterministic perturbation algorithms such as 1RDSA-Perm-DP and 1RDSA-Kw-DP outperform other algorithms. From Fig 2, 1RDSA-Unif converges faster than

1RDSA-AB. Finally, as per Table-II, highest error as well as highest standard deviation is observed for 1SPSA and very low standard deviation is observed for DP algorithms.

3. **p = 100**

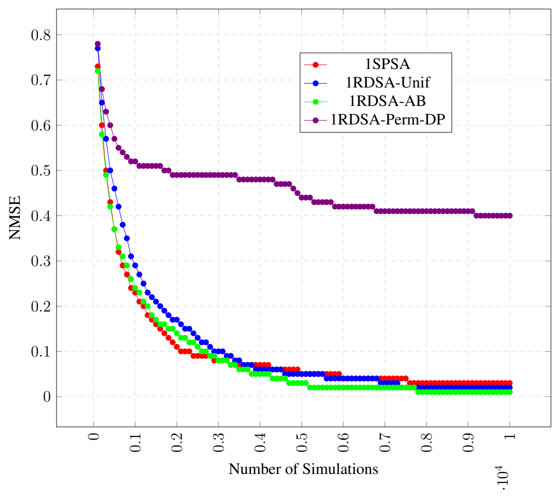


Figure 2.3: Comparison of convergence of 1SPSA & 1RDSA variants at p = 100

Table 2.3: NMSE for p = 100 with a simulation budget of 10000

Method	NMSE $\pm STD$
1SPSA	2.74E^{-02} $\pm$1.09E^{-02}
1RDSA-U	1.79E^{-02} $\pm$9.10E^{-03}
1RDSA-AB	1.35 E^{-02} $\pm$8E^{-03}
1RDSA-Perm-DP	39.74 E^{-02} $\pm$11.98E^{-02}

Observation 3: For high dimension p=100, 1RDSA-Lex-DP is almost impossible to implement owing to 3^{100} iterations of inner loop. In contrast with other experiments, in this case, we are using constant value of step-size and gain coefficient. The aforementioned step-size of form $\frac{a_0}{k}$ and gain coefficient of form $\frac{c_0}{k}$ take large number of updates to converge and vanish as number of updates increase. Further, tweaking the parameters did not work. The solution converged for 1SPSA, 1RDSA-U and 1RDSA-AB in less than 5000 iterations whereas it does not seem to converge fully for 1RDSA-Perm-DP. The constant step size is not guaranteed to work for all algorithms. Hence, the reason of deviation from convergence of 1RDSA-Perm-DP could be the use of constant step-size.

As discussed, highest NMSE is observed for 1RDSA-Perm-DP while 1RDSA-AB is observed to be having lowest error and low standard deviation as well.

Chapter 3

POLICY OPTIMIZATION USING SP METHODS

3.1 Introduction

In this chapter, we present the RL problem using the formalism of Markov decision processes (MDPs). Then, we describe the popular policy gradient (PG) approach to solve RL problems. In particular, we outline the two variants of PGs using the likelihood ratios and simultaneous perturbation for gradient estimation. We refer to Bertsekas *et al.* [2003] and Sutton *et al.* [2017] for theory and formulation.

3.2 Markov Decision Process

MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent states, and through those future rewards Sutton *et al.* [2017]. In MDP, we have a stochastic process having states in a set S and control values in set A with transition probabilities from a state s to state s' at stage k is characterized by

$$p(s'|s,a) = P[s_{k+1} = s'|s_k = s, a_k = a], \tag{3.1}$$

and satisfy the controlled Markov property:

$$P[s_{k+1} = s'|s_k = s, a_k = a, ..., s_0 = s_0, a_0 = a_0] = p(s'|s,a), \tag{3.2}$$

for any $s_0, \ldots, s$, s' in S and a_0 , $\ldots$, a in A.

Continuing on the MDP formulation, the next component is per-stage reward, given as $r_k(s,a,s')$ when the system is in state s and action a is taken leading to state s', in stage k.

Policy

A policy $\pi(a|s)$ is a mapping from states to probabilities of choosing each possible action. In experiments, we will work only with infinite state space but with discrete or continuous action space.

Value function

Let $r_{a_k}(s_k, s_{k+1})$ be the reward value at stage k for taking action a_k in state s_k by transitioning to state s_{k+1}. Then, the value function for a given admissible policy $\pi(a|s)$ can be defined by:

$$J^{\pi}(s) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{a_k}(s_k, s_{k+1}) | s_0 = s\right], \tag{3.3}$$

for all $s \in S$ where, γ is the discount factor.
For a fixed policy π, the sequence of states visited, say $\{s_0, . . \}$ forms a Markov chain. So, the objective function is to find an optimal value function J^* that maximizes the expected reward.

$$J^*(s) = \max{}_{\pi} J^{\pi}(s). \tag{3.4}$$

Further, the optimal value function satisfies the Bellman equation of optimality which expresses a relationship between the value of a state and the values of its successor states:

$$J^*(s) = \max_{a \in A(s)} \left(r_a(s, s') + \gamma \sum_{s' \in S} p(s'|s, a) J^*(s') \right), \tag{3.5}$$

for all $s \in S$.

3.3 Reinforcement Learning

A RL problem consists of an agent and an environment. The agent acts on the environment and gets a scalar reward. The reward is a way of letting the agent know as to how good the action was at that particular instant. The task of the agent is to learn an optimal policy - a mapping from states to action, which maximizes the expected sum of rewards. A typical RL environment is shown in Figure 3.1.

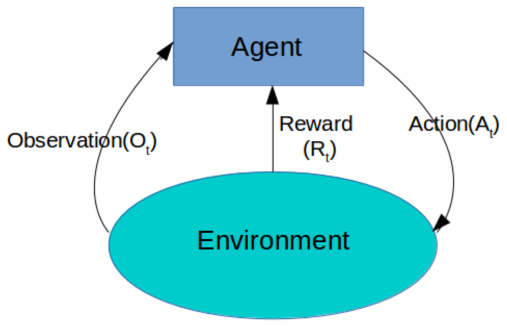


Figure 3.1: Typical RL environment

Reinforcement learning problem can be modeled as a MDP, which can be used to describe an environment. The system transitions from a state to next state probabilistically. A cost is incurred by system at each such transition. The goal of the problem is to find a policy for choosing actions which minimizes long-run cost objective.

In RL, there are different choices to approximate: value function, policies, dynamic models or some combinations of these. At top level, one can optimize using dynamic programming algorithms such as policy iteration, value iteration, or using policy optimization methods such as policy gradient (PG) methods.

3.4 Deep Neural Network

In deep neural network, the neuron takes a vector of inputs x and calculates the weighted sum of the inputs, with the weights denoted by w. The weighted sum is added to a bias term, b, and is passed through an activation function, f, to produce the output of the neuron. The complete equation for a neuron can be written as :

$$y_i = f(\sum_j x_j w_{ij} + b_i), \tag{3.6}$$

where j is the number of inputs to the neuron.

The activation function introduces non-linearity into the output of a neuron. This helps the network to learn non-linear representations from the training data. A number of activation functions have been introduced in literature, like sigmoid, hyperbolic tangent and a rectified linear unit. The commonly used activation functions are explained below:

- Sigmoid :

$$f(x) = \frac{1}{1 + e^{(-x)}} \tag{3.7}$$

- Hyperbolic Tangent (Tanh) :

$$f(x) = \frac{e^{(x)} - e^{(-x)}}{e^{(x)} + e^{(-x)}} \tag{3.8}$$

- Rectified Linear Unit (ReLU):

$$f(x) = \max(0, x) \tag{3.9}$$

The most common architecture is feed forward neural network which consists of three types of layers: an input layer, a few hidden layers and an output layer. The flow of information takes place from the input layer to the hidden layers and finally to the output layer which computes the final output as shown in Figure 3.2.

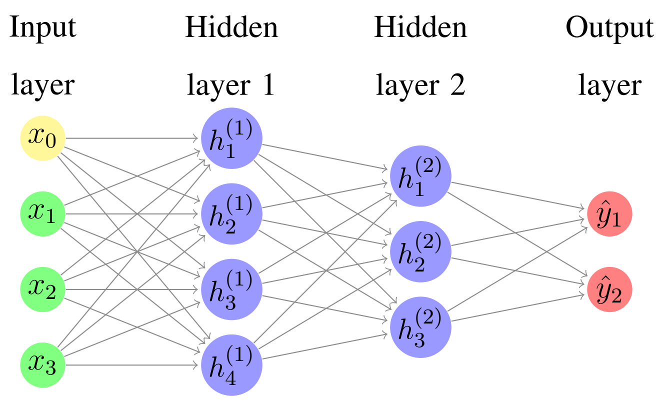


Figure 3.2: Neural Network Architecture

Modern machine learning techniques, especially, deep learning, learn complex function from data. The common approach is - choose an objective function, function approximators for hidden layers and then optimize parameters with gradient based method.

3.5 Policy Optimization Methods

Policy optimization methods are basically policy based which maps agent's state to its action. These methods, when applied to reinforcement learning problem, can be viewed as a numerical optimization problem where the goal is to optimize value function with respect to the parameters of policy.

There are two types of policy optimization methods: derivative free optimization (DFO) and policy gradient (PG). First, there are DFO algorithms, which include evolutionary algorithms Schulman [2016]. These algorithms work by moving in the direction of good performance which may be obtained by perturbing the policy parameters in many different ways. Some of the DFO algorithms include cross-entropy method Szita and Lörincz [2006], covariance matrix adaptation Wampler and Popovi´c [2009], and natural evolution strategies D. Wierstra and Schmidhuber [2008]. Second, there are policy gradient methods Williams [1988]; Williams [1992]; T. Jaakkola and Singh [1994]; Sutton *et al.* [2017]. These methods are a class of reinforcement learning algorithms that work by repeatedly estimating the gradient of the policy's performance with respect to its parameters. Recently, policy optimization using SPSA R. Ramamurthy [2018] has experimentally proved that Echo State Networks (ESNs) trained using SPSA approaches outperform conventional ESNs trained using temporal difference and policy gradient methods. Further, these algorithms are evaluated on cart-pole in Riedmiller [2007] providing a useful baseline on parameterized policy search algorithms.

In the next section, we discuss policy gradient using methods such as likelihood ratio and SP methods in detail.

3.5.1 Gradient estimation using likelihood ratios

In this method, we use the score function (likelihood function) gradient estimator, a general method for estimating gradients of expectations. One basic algorithm called

REINFORCE, uses this approach for policy optimization.

We use a parameterized stochastic policy that gives us a probability distribution over actions denoted $\pi_\theta(a|s)$. In the following discussion, a trajectory τ refers to a sequence of states and actions $\tau \equiv (s_0, a_0, s_1, a_1, ..., s_T)$. Let $p(\tau|\theta)$ denote the probability of the entire trajectory τ under policy parameters θ, and let $R(\tau)$ denote the total reward of the trajectory. The derivation of the score function gradient estimate tells us that

$$\nabla_\theta J^{\pi_\theta}(s_0)] = E_\tau[\nabla_\theta \log p(\tau|\theta) R(\tau)]. \tag{3.10}$$

Next, we need to expand the quantity $\log(p(\tau|\theta))$ to derive a practical formula. Using the chain rule of probabilities, we obtain,

$$p(\tau|\theta) = \mu(s_0)\pi(a_0|s_0,\theta)P(s_1,r_0|s_0,a_0)\pi(a_1|s_1,\theta)...\pi(a_{T-1}|s_{T-1},\theta)P(s_T,r_{T-1}|s_{T-1},a_{T-1}), \tag{3.11}$$

where μ is the initial state distribution.

When we take the logarithm, the product turns into a sum, and when we differentiate with respect to θ, the terms $P(s_t|s_{t-1}, a_{t-1})$ terms drop out as does $\mu(s_0)$. We obtain,

$$\nabla_\theta J^{\pi_\theta}(s_0)] = E_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta) R(\tau) \right], \tag{3.12}$$

which can be estimated as follows:

$$\widehat{g}_k = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta) \sum_{t^1=t}^{T-1} r_{t^1}. \tag{3.13}$$

We update the policy using update rule :

$$\theta_{k+1} = \theta_k + \alpha.\widehat{g}_k, \tag{3.14}$$

where α is the learning rate.

The pseudocode of REINFORCE algorithm using likelihood ratios as gradient estimation is given in Algorithm 2.

```
Algorithm 2 REINFORCE algorithm
Input  : State of environment;
         Initialise weights parameters $(\theta)$ of neural network, choose hidden layer size;
Output: weights $\theta^*$
for iteration k = 0, 1, 2, . . . do
    Collect a set of trajectories by executing the current policy
    At each time-step $t$ in trajectory, compute rewards-to-go
    Return $R_t = \sum_{t^1=t}^{T-1} \gamma^{t^1-t} r_{t^1}$ summed over all trajectories and time-steps
    Set $\widehat{g}_k = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta)\mathrm{R}_t$
    Update the policy, $\theta_{k+1} = \theta_k + \alpha.\widehat{g}_k$
end
Return $\theta^*$
```

3.5.2 Gradient estimation using SPSA

In this method, we apply SP techniques for gradient estimation. In particular, we propose to use simultaneous perturbation stochastic optimization (SPSA) and random direction stochastic approximation (RDSA), a gradient approximation technique, as a training algorithm, which requires only two evaluations of objective function at each iteration, regardless of dimension of the parameter. We have already discussed SPSA and RDSA in Chapter 2.

Problem Formulation

In this part, we formulate SPSA to learn policies. We are using neural network for the parameterization of policies. The input of this neural network is current state of the environment. With this input, our network returns an action which simulates next state and reward. This continues until we reach a terminal state. In SPSA, the weights (θ) are perturbed by using a random perturbation vector (Δ) in both positive and negative directions for a random initial state. The total episodic reward obtained during a trajectory in both directions is used to update weights. In this way, training continues for many iterations until we reach convergence. This formulation is also represented as an algorithm in Algorithm 3 and Algorithm 4. For RDSA-U and RDSA-AB, the gradient is estimated as described in Section 2.4.1 using (2.6) and Section 2.4.2 using (2.9), respectively.

Value function estimation:

The trajectory τ refers to a sequence of states and actions $\tau \equiv (s_0, a_0, s_1, a_1, ..., s_T)$. Let $R(\tau)$ denote the total reward of the trajectory. Then, value function $J^{\pi_\theta}(s_0)$ is estimated as total rewards earned in a trajectory (or in an episode) for a initial state s_0.

$$J^{\pi_\theta}(s_0) = E[R(\tau)]. \tag{3.15}$$

The goal of this optimization problem is:

$$\max_\theta J^{\pi_\theta}(s_0), \tag{3.16}$$

where π_θ is parameterized policy. So, this becomes an optimization problem with respect to $\theta \in R^d$.

Parameterized policy used:

The parameterization of the policy depend on the action space of the MDP, and whether it is a discrete set or a continuous space.
For discrete action space, we are using softmax function for policy π_θ as:

$$\pi_\theta(a|s) = \frac{e^{\theta^T \phi(s,a)}}{\sum_{a^1 \in A(s)} e^{\theta^T \phi(s,a^1)}}. \tag{3.17}$$

For continuous action space, we are using hyperbolic tangent function or sampling from a Gaussian policy with parameterized mean and a fixed or a decreasing variance.

$$\pi_\theta(a|s) = N(\mu(\theta^T \phi(s,a)), \sigma^2), \tag{3.18}$$

where A(s) represents actions $a \in A$ for $s \in S$ and $\phi(s,a)$ represents featurized state and θ represents parameters. In our case, θ is weight matrix present in the output layer of neural network.

Update rule:

The parameters of neural network θ at any stage k is updated as follows:

$$\theta_{k+1} = \theta_k + a_k . \left(\frac{J^{\pi_{\theta_k}+}(s) - J^{\pi_{\theta_k}-}(s)}{2c_k\Delta_k} \right), \tag{3.19}$$

where Δ_k = $\{\Delta_k{}^0$, $\Delta_k{}^1$,.. , $\Delta_k{}^d\}$ and $\Delta_k{}^i$ is ± 1 w.p. 0.5 , c_k is a perturbation parameter, $J^{\pi_{\theta_k}+}(s)$, $J^{\pi_{\theta_k}-}(s)$ denote estimated value functions at some state s obtained by perturbing weights vectors $\theta_k{}^+$ and $\theta_k{}^-$ such that $\theta_k{}^+ = \theta_k + c_k.\Delta_k$ and $\theta_k{}^- = \theta_k - c_k.\Delta_k$ and a_k is the step size.

Algorithm 3 Policy SPSA Algorithm

```
Input  : State of environment;
         Initialise weights parameters $(\theta)$ of neural network, choose hidden layer size;
         Initialise Variables: $k$ =0 , $a$ = 1.0 , $c$ = 1.9, $\alpha$ = 1, $\gamma$ = 1/6 ;
Output: Weights $\theta^*$
for iteration k = 0, 1, 2, . . . do
    Set $a_k = \frac{a}{(k+A+1)^\alpha}$, $c_k = \frac{c}{(k+1)^\gamma}$
    Generate $\Delta_k$ = $\{\Delta_k{}^0, \Delta_k{}^1 ,.. , \Delta_k{}^d\}$ where, $\Delta_k{}^i$ is $\pm 1$ w.p. 0.5
    Perturb weights: $\theta_k{}^+$ = $\theta_k$ + $c_k$ . $\Delta_k$, $\theta_k{}^-$ = $\theta_k$ - $c_k$ . $\Delta_k$
    Obtain value function estimates by running episode for a trajectory with above per-
      turbed parameters:
    $J^{\pi_{\theta_k}+}(s)$ = Run Episode($\theta_k{}^+$, copy of environment)
    $J^{\pi_{\theta_k}-}(s)$ = Run Episode($\theta_k{}^-$, copy of environment)
    Update weights $\theta_{k+1} : \theta_k + a_k . \left( \frac{J^{\pi_{\theta_k}+}(s) - J^{\pi_{\theta_k}-}(s)}{2c_k\Delta_k} \right)$
end
Return $\theta^*$
```

Algorithm 4 Run Episode

```
Input  : Environment, weights($\theta$), done = False ; State = reset initial state
Output: Episodic reward in a trajectory
while not done do
    Update neural network with weight $\theta$ and activation function
    action = choose action according to output of neural network
    next state, reward, done = simulate environment(action)
    episodic reward + = reward
    state = next state
end
Return episodic reward
```

3.6 Experiments

We designed our experiments to investigate the following questions:

1. What is the effective and best parameter setting for convergence when optimizing episodic total reward using SPSA and RDSA on discrete and continuous action spaces?
2. How does it perform across various environments such as high-dimensional and low-dimensional states, discrete and continuous action spaces etc.?
3. Policy optimization using SPSA is implemented with neural network having hidden layer features and parameters for learning. How is the performance of this algorithm and its variants if compared with REINFORCE method?

To answer these questions, we implemented SPSA and RDSA for classic discrete and continuous control problems and compared results with REINFORCE. The architecture, task details and results are discussed in detail in next subsections.

3.6.1 Implementation

We evaluated our approach on various classic-control environment with discrete and continuous action space which are as follows:

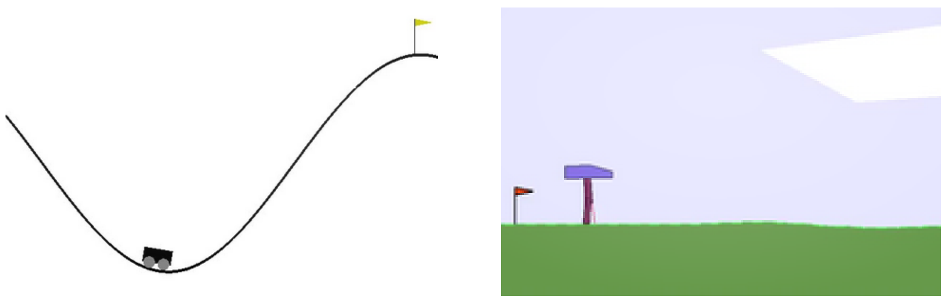
1. Balancing a pole on a cart (cartpole)
2. Swing up a two-link robot (acrobot)

3. Drive up a hill (mountain-car)
4. Train a bipedal robot to walk (bipedal-walker)

An illustration of these OpenAI Gym environments is given in Figure 3.3.

(a) Cartpole (b) Acrobot

Figure 3.3: Tasks illustration from OpenAI Gym (discrete action space)



(a) Mountain-car (b) Bipedal walker

Figure 3.4: Tasks illustration from OpenAI Gym (continuous action space)

Architecture

We employ a neural network architecture for all of the three classic tasks namely, cart-pole, acrobot, mountain-car, bipedal walker, which is a feed-forward network with a hidden layer of size 10, 20, 100 and 50 and activation functions sigmoid, tanh, leaky-relu and tanh units, respectively. The final output layer had softmax activation for discrete action space. For continuous actions, we use tanh function in output layer. Also, we execute regularization techniques such as Xavier/uniform initialization and dropout to improve results.

Task details

Cartpole: In Cartpole, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track as shown in Figure 3.3 (a). The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity. The state-space is a continuous 4-dimensional vector and action space is discrete $\{0,1\}$. The results are averaged for 20 runs. We collected 5000 trajectories in each run.

Acrobot: The acrobot system includes two joints and two links, where the joint between the two links is actuated as shown in Figure 3.3 (d). Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height. Both the state space and action space are discrete having 6 dimensions and 3 dimensions, respectively. We collected 2000 trajectories in each run and performed 20 runs keeping the same initial state.

Mountain-car: An underpowered car must climb a one-dimensional hill to reach a target as shown in Figure 3.3 (b). The target is on top of a hill on the right-hand side of the car. If the car reaches it or goes beyond, the episode terminates. On the left-hand side, there is another hill, climbing on which can be used to gain potential energy and accelerate towards the target. The state space and actions are both continuous in this case. We used tanh activation function for control to be in range [-1,1]. We performed 5 runs and collected 5000 trajectories in each run.

Bipedal-walker: In this problem, we have to get a 2D biped walker to walk through rough terrain as shown in Figure 3.3 (c). Both the state space and action space are continuous having 24 dimensions and 4 dimensions, respectively. Reward is given for moving forward, total 300+ points up to the far end. If the robot falls, it gets -100. The episode ends when the robot body touches ground or the robot reaches far right side of the environment. We used scaled tanh activation function for control to be in range [-2,2]. We performed 20 runs and collected 1000 trajectories in each run.

The SPSA parameters such as step size, scaling factor, perturbation constant, decay rate and similarly, parameters concerning neural network such as hidden layer size and activation function are tuned for each experiment. Table 3.1 lists all hyperparameters and their values.

Table 3.1: Hyperparameters and their values for different experiments

Parameters	Cartpole	Acrobot	Mountain-car	Bipedal-walker
State space dim.	4	6	2	24
Action space type	discrete	discrete	continuous	continuous
Action space dim.	2	4	1	4
Total num. policy params	70	220	400	1450
Hidden layer size	10	20	100	50
Activation units for hidden layer	sigmoid	tanh	leaky relu	tanh
Weight initialization	Xavier	uniform	Xavier	uniform
Step size a_0	1.0	1.0	2.0	1.0
Perturbation constant c_0	1.9	1.9	2.5	1.9
α, γ	1, 1/6	1, 1/6	1, 1/6	1, 1/6
Number of episodes	5000	2000	5000	1000

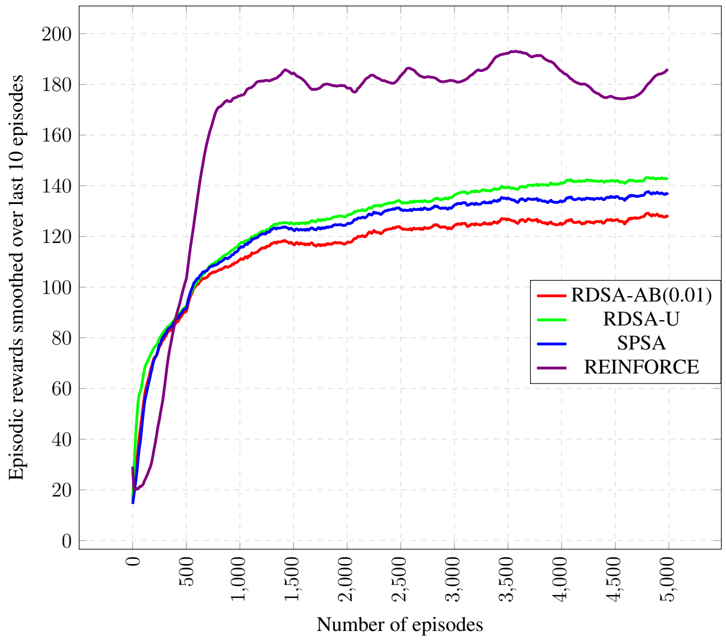
3.6.2 Results

All results are presented in terms of the cost, which is defined as negative reward and is minimized. The following algorithms are included in the comparative evaluation: SPSA, RDSA-AB, RDSA-U and REINFORCE. We present the results obtained on the four tasks outlined above. The results are obtained by averaging across 20 experiments with different random seeds.

3.6.2.1 CartPole

The learning curves are shown in Figure 3.5. The results indicate that the best results are obtained using REINFORCE algorithm. Among SPSA and RDSA variants, RDSA-U is performing better than SPSA and RDSA-AB. However, the difference in the performance is not significantly high.

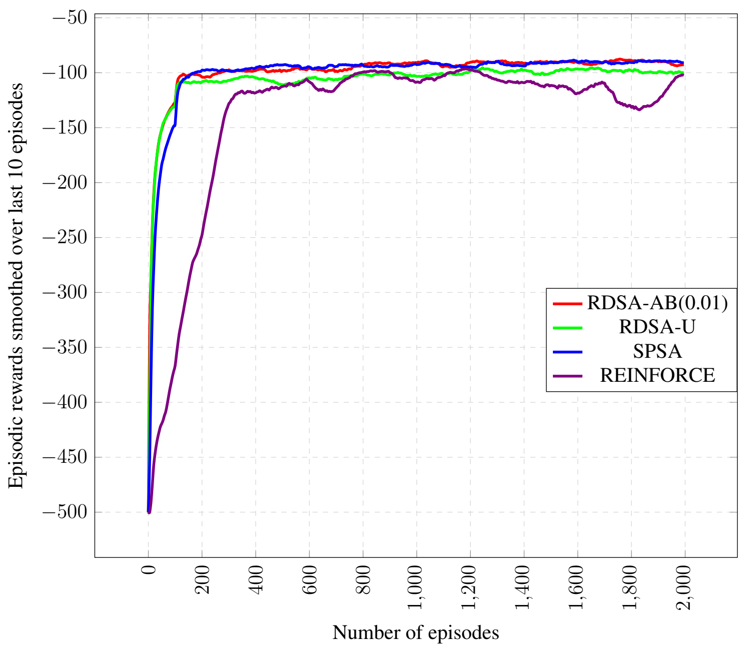
Figure 3.5: Learning curves for cart-pole task averaged over 20 iterations



3.6.2.2 Acrobot

The learning curves are shown in Figure 3.6. For this task, SPSA and RDSA variants converge faster than REINFORCE algorithm. RDSA-U performs better than REINFORCE but worse than SPSA and RDSA-AB.

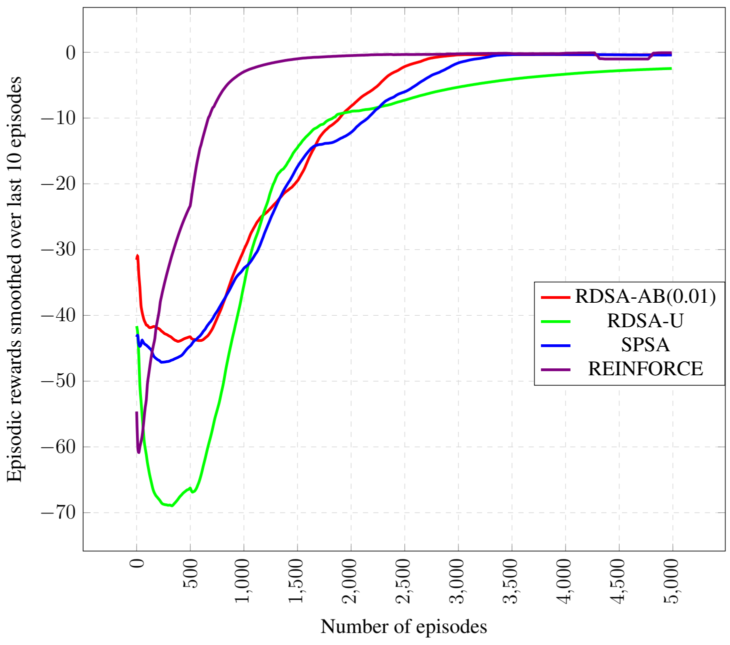
Figure 3.6: Learning curves for acrobot task averaged over 20 iterations



3.6.2.3 Mountain-car

The results are averaged across 5 runs. The learning curve of mountain-car in Figure 3.7 shows that all the algorithms except RDSA-U converge to an optimum solution. REINFORCE takes less number of episodes to converge but requires high time during training through back-propagation.

Figure 3.7: Learning curves for mountain-car task averaged over 5 iterations



3.6.2.4 Bipedal-Walker

The learning curve of biped-walker in Figure 3.8 shows that SPSA and RDSA-AB outperform REINFORCE algorithm and RDSA-U. Additionally, it is evident from figure that changing noise parameter in RDSA-AB does not have much impact on the performance.

Figure 3.8: Learning curves for Biped-walker task averaged over 20 iterations

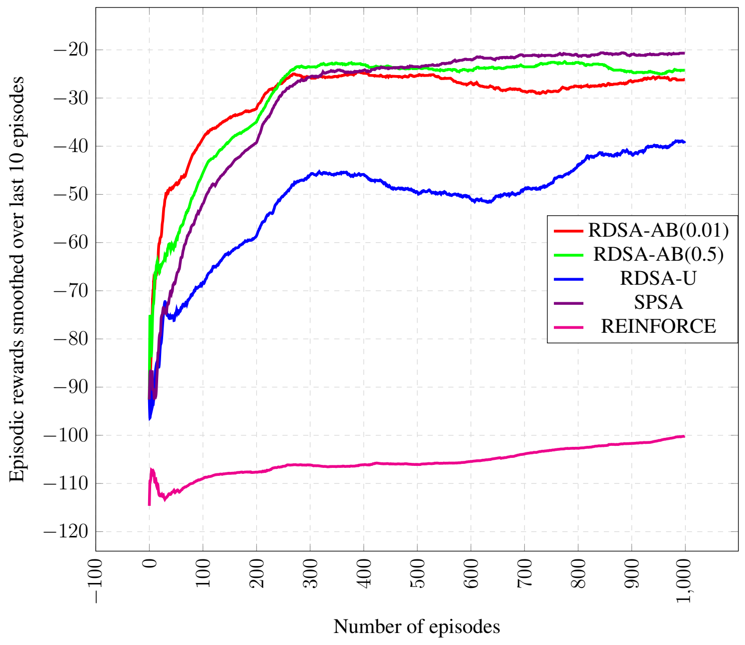


Table 3.2: Performance summary : evaluation results containing mean of episodic total reward and standard error in the last 100 iterations of policy optimization for the four tasks

Algorithm	Cartpole	Acrobot	Mountain-car	Bipedal-walker
SPSA	135.3 ±6.3	-92.0 ±3.7	-0.40 ±0.001	-19.6 ±1.4
RDSA-unifom	141.7 ±5.0	-99.7 ±3.9	-2.35 ±0.009	-37.0 ±3.2
RDSA-Asymber(0.01)	124.7 ±7.1	-90.8 ±1.9	-0.33 ±0.001	-21.3 ±1.7
REINFORCE	184.7 ±1.9	-102.0 ±4.1	-0.06 ±0.001	-99.5 ±0.5

The performance of all the algorithms for various tasks is summarized in Table 3.2. We have used mean of episodic rewards and standard error i.e. (mean ± 1.96*standard error) in last 100 iterations as performance metric. The table indicates that for bipedal-walker, SPSA and RDSA algorithms outperform REINFORCE algorithm by high margin. It also indicates that SPSA and RDSA-AB show good convergence for all the four tasks.

3.6.3 Summary

From the simulation experiments, we observe that policy gradient based SP methods perform better than REINFORCE in many cases, especially, show an outstanding performance for bipedal-walker.

Chapter 4

CONCLUSIONS AND FUTURE DIRECTIONS

In the first part of the thesis, we performed a detailed comparative evaluation of first order simultaneous perturbation based simulation optimization algorithms. The algorithms studied includes random as well as deterministic perturbations and the experiments were conducted with a objective function that had many local minima. As future work, it would be interesting to evaluate second order simultaneous perturbation methods. An orthogonal direction is to study the simultaneous perturbation algorithm using a different objective function.

In the second part of the thesis, we implemented policy learning using SP based algorithms such as SPSA and RDSA for solving RL problems. Experiments on classic problems indicate that SPSA and RDSA are powerful alternatives to reinforcement learning methods commonly used for policy learning. In future work, it would be interesting to extend the ideas reported here using LSTM units to solve more complex RL problems that require long-term dependencies. An orthogonal future work direction is to evaluate performance of second order simultaneous perturbation methods in the same framework.

Bibliography

1. **Bertsekas, D.** *et al.*, *Neuro-Dynamic Programming*. Athena Scientific, 2003.

2. **D. Wierstra, J. P., T. Schaul** and **J. Schmidhuber** (2008). Natural evolution strategies. *IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 3381–3387.

3. **Kushner, H. J.** and **D. S. Clark** (1978). Stochastic approximation methods for constrained and unconstrained. *Springer Verlag*.

4. **L.A. Prashanth, P.** and **Desai** (2015). Simultaneous perturbation methods for adaptive labor staffing in service systems. *Lecture Notes in Control and Information Sciences Series*, **91**(5), 432 – 455.

5. **L.A. Prashanth, S. B.** *et al.* (2017). Adaptive system optimization using random directions stochastic approximation. *IEEE Transactions on Automatic Control*, **62**(5), 2223–2238.

6. **L.A. Prashanth, S. B.** *et al.* (2018). Random directions stochastic approximation with deterministic perturbations. *arXiv:1808.02871 [math.OC].*

7. **Miller, B. L.** and **M. J. Shaw** (1995). Genetic algorithms with dynamic niche sharing for multimodal function optimization. *IlliGAL rep. No. 95010, Illinois Genetic Algorithms Laboratory, University of Illinois at Champaign-Urbana.*

8. **Prashanth, L.** (2013). *Resource Allocation for Sequential Decision Making under Uncertainty: Studies in Vehicular Traffic Control, Service Systems, Sensor Networks and Mechanism Design*. Ph.D. thesis, Indian Institute of Science Bangalore, Bangalore - 560012.

9. **R. Ramamurthy, R. S. S. W., C. Bauckhage** (2018). Policy learning using spsa. *International Conference on Artificial Neural Networks*, 3–12.

10. **Riedmiller, M.** (2007). Evaluation of policy gradient methods and variants on the cart-pole benchmark. *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*.

11. **S. Bhatnagar, P.** and **L. Prashanth** (2013). Stochastic recursive algorithms for optimization: Simultaneous perturbation methods. *Lecture Notes in Control and Information Sciences Series*, **434**, 41–75.

12. **Schulman, J.** (2016). *Optimizing Expectations : From Deep reinforcement learning to stochastic computation graphs*. Ph.D. thesis, University of California, Berkeley, Berkeley.

13. **Simopt** (). Simulation optimization library. URL `http://simopt.org/wiki/images/0/03/Multimodal.pdf`.

14. **Spall, J. C.** (1997). A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica*, **33**(1), 109–112.

15. **Spall, J. C.**, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. New York: Wiley, 2005.

16. **Sutton, R.** *et al.*, *Reinforcement Learning: An Introduction*. MIT Press, 2017.

17. **Szita, I.** and **A. Lörincz** (2006). Learning tetris using the noisy cross-entropy method. *Neural Computation*.

18. **T. Jaakkola, M. I. J.** and **S. P. Singh** (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation 6.6*, 1185–1201.

19. **Wampler, K.** and **Z. Popovi´c** (2009). Optimal gait and form for animal locomotion. *ACM Transactions on Graphics (TOG)*, **28**(3), 60.

20. **Williams, R. J.** (1988). Toward a theory of reinforcement-learning connectionist systems. *Technical Report NU-CCS-88-3,Northeastern University, College of Computer Science*.

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

22. **Xu, N. B. L., J. I.** and **L. J. Hong** (2010). Industrial strength compass : A comprehensive algorithm and software for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation*, **20**(1).