



# Policy Learning Using SPSA

R. Ramamurthy<sup>1,2</sup>(✉), C. Bauckhage<sup>1,2</sup>, R. Sifa<sup>1,2</sup>, and S. Wrobel<sup>1,2</sup>

<sup>1</sup> Department of Computer Science, University of Bonn, Bonn, Germany  
ramamurt@iai.uni-bonn.de

<sup>2</sup> Fraunhofer Center for Machine Learning, Sankt Augustin, Germany

**Abstract.** We analyze the use of simultaneous perturbation stochastic approximation (SPSA), a stochastic optimization technique, for solving reinforcement learning problems. In particular, we consider settings of partial observability and leverage the short-term memory capabilities of echo state networks (ESNs) to learn parameterized control policies. Using SPSA, we propose three different variants to adapt the weight matrices of an ESN to the task at hand. Experimental results on classic control problems with both discrete and continuous action spaces reveal that ESNs trained using SPSA approaches outperform conventional ESNs trained using temporal difference and policy gradient methods.

**Keywords:** Echo state networks · Recurrent neural networks  
Reinforcement learning · Stochastic optimization

## 1 Introduction

Creating systems that learn to solve complex tasks from interactions with their environment is one of the primary goals of artificial intelligence research. Recently, much progress has been made in this regard, mainly achieved through modern reinforcement learning (RL) techniques [1, 21]. Examples of recent successes include systems which exceed human level performance in playing console-based Atari games [12] or can navigate 3D virtual environments [11], and AlphaGo Zero [17] became the first program to beat world class GO players by learning from self-play only. Function approximators such as deep neural networks, when used with off-policy and bootstrapping methods such as Q-learning, which used to be unstable and were referred to as a “deadly-triad” [20], have now been proven to be a competent approach using techniques such as experience replay [8] which stabilize learning with the help of a large replay memory.

Spurred by these successes, another line of recent research has considered alternative approaches to RL using black-box optimization methods which do not require back propagation of gradient computations. Corresponding contributions include systems [10, 14] that are trained using so called evolution strategies which achieve competitive performance in playing Atari games. Similar performance was obtained in [19] where genetic algorithms were found to scale better than evolution strategies. This revived interest in black-box methods for solving

RL problems as these can be parallelized when using modern distributed architectures. However, most real-world systems must deal with limited and noisy state information resulting in partial observability as encountered in partially observable Markov decision processes (POMDPs). To learn policies under such circumstances, systems need to have internal memory. Therefore, recurrent RL methods to cope with partial observability have recently been investigated but were found to be difficult to train [4].

In this paper, we focus on these kind of problems and consider RL in partially observable environments. Since echo state networks [5] are known for their simple architecture and short-term memorization capabilities, we choose them in order to train parameterized control policies. In particular, we propose to use simultaneous perturbation stochastic optimization (SPSA), a gradient approximation technique, as a training algorithm, which at each iteration requires only two evaluations of objective function regardless of dimension of the parameter. Using SPSA, we devise three types of ESN training that differ in how the weight matrices are chosen in each iteration. Finally, we use such ESNs to learn policies and test them against baselines on classic control problems.

Previous work on black-box methods for training echo state networks seeks to combine genetic algorithms to train internal weights of the reservoir and stochastic gradient descent to train the output weights [3, 15]. Similar work was done in [6] where output weights and spectral radii of internal weight matrices were evolved. Alternatively, more recent work [16] concerning different learning strategy focused on using hebbian learning rules to adapt reservoir matrices. An interesting hybrid of using hebbian learning and temporal difference learning was later proposed in [7] to adapt actor-critic ESNs. In contrast to these previous approaches, we use SPSA to optimize the entire network weights which has several noteworthy properties: (i) it requires only two loss measurements at each iteration, (ii) it does not require back propagation of gradients, (iii) it does not require any maintenance of candidate solutions as in genetic algorithms, and (iv) it can handle stochastic returns and hence does not require averaging over multiple measurements to account for the noisy returns.

## 2 Simultaneous Perturbation Stochastic Approximation

In this short section, we briefly recall the main ideas behind simultaneous perturbation stochastic approximation (SPSA) for derivative free optimization; readers familiar with this technique may safely skip ahead.

Consider the general problem of maximizing a differentiable objective function  $f(\boldsymbol{\theta}) : \mathbb{R}^d \rightarrow \mathbb{R}$ , that is, consider the problem of finding  $\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ .

For many complex systems, the gradient  $\partial f / \partial \boldsymbol{\theta}$  cannot be computed directly so that  $\partial f / \partial \boldsymbol{\theta} = \mathbf{0}$  can often not be solved. It is, however, typically possible to evaluate  $f(\boldsymbol{\theta})$  at various values of  $\boldsymbol{\theta}$  which, in turn, allows, for computing stochastic approximations of the gradient. One method in this regard is SPSA due to Spall [18] which iteratively updates estimates of the optimal  $\boldsymbol{\theta}$  as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + l_k \hat{\mathbf{g}}_k(\boldsymbol{\theta}_k) \quad (1)$$

where  $\hat{\mathbf{g}}_k(\boldsymbol{\theta}_k)$  is an estimator of the gradient at  $\boldsymbol{\theta}_k$  and  $l_k$  is the learning rate in iteration  $k$ . To estimate the gradient, two perturbations are generated, namely  $(\boldsymbol{\theta}_k + c_k \boldsymbol{\delta}_k)$  and  $(\boldsymbol{\theta}_k - c_k \boldsymbol{\delta}_k)$  where  $\boldsymbol{\delta}_k$  is a perturbation vector and  $c_k$  is a scaling parameter. Then, the possibly noisy objective function  $F(\cdot) = f(\cdot) + \textit{noise}$  is measured at  $F(\boldsymbol{\theta}_k + c_k \boldsymbol{\delta}_k)$  and  $F(\boldsymbol{\theta}_k - c_k \boldsymbol{\delta}_k)$  and the gradient is estimated using a two-sided gradient approximation

$$\hat{\mathbf{g}}_k(\boldsymbol{\theta}_k) = \frac{F(\boldsymbol{\theta}_k + c_k \boldsymbol{\delta}_k) - F(\boldsymbol{\theta}_k - c_k \boldsymbol{\delta}_k)}{2 c_k \boldsymbol{\delta}_k}. \quad (2)$$

The convergence of the SPSA algorithm critically depends on the choice of its parameters  $l_k$ ,  $c_k$  and  $\boldsymbol{\delta}_k$ . In particular, the learning rate  $l_k$  must meet the Robbins-Monro conditions [13], namely  $l_k > 0$  and  $\sum_{k=1}^{\infty} l_k = \infty$ , and a common choice in practice therefore is  $l_k = \frac{l}{(L+k)^\alpha}$  where  $l, \alpha, L > 0$ . Similarly, the scaling factor  $c_k$  must satisfy  $\sum_{k=1}^{\infty} \left(\frac{l_k}{c_k}\right)^2 < \infty$  so that a good choice amounts to  $c_k = \frac{c}{k^\gamma}$  where  $c, \gamma > 0$ . And, essentially, each element of the perturbation vector  $\boldsymbol{\delta}_k$  is sampled from a uniform distribution over the set  $\{-1, +1\}$ .

### 3 Learning Policies Using Echo State Networks

In this section, we first briefly review policy learning under partial observability as well as echo state networks and then introduce our approach towards policy learning using echo state networks trained via SPSA.

#### 3.1 Partial Observability

Consider an agent interacting with an environment. At any time  $t$ , the agent observes the state  $\mathbf{s}_t$  of the environment and performs an action  $a_t$  by following a policy  $\pi(a_t|\mathbf{s}_t)$  which is a mapping of state  $\mathbf{s}_t$  to the probability of choosing action  $a$  at time  $t$ . In return, the environment responds with a reward  $r_t$  and finds itself in a new state  $\mathbf{s}_{t+1}$ .

However, in environments that are only partially observable, the agent does not receive all relevant state information because of limited sensory inputs. In this case, the state  $\mathbf{s}_t$  does not satisfy the Markov property because it does not summarize what has happened in the past so that an informed decision cannot be taken. For such non-Markovian states, it is necessary to make the policy dependent on a history of states  $h_t = \{\mathbf{s}_t, \mathbf{s}_{t-1}, \dots\}$  rather than on the current state  $\mathbf{s}_t$  only. Hence, the policy becomes  $\pi(a_t|h_t)$ .

This, however, becomes impractical to compute whenever different tasks require arbitrary lengths of histories. In situations like these, an echo state network can be used to integrate the required history in its reservoir states. In this way, we are able to parameterize the policy with weights of an echo state network  $\boldsymbol{\theta}$  as  $\pi(a_t|\mathbf{s}_t, \boldsymbol{\theta})$  which takes the current state  $\mathbf{s}_t$  as the input and returns probabilities of actions by compacting the history of input states in the reservoir memory.

### 3.2 Echo State Networks

We next briefly recall the notion of echo state networks. These belong to reservoir computing paradigm in which a large reservoir of recurrently interconnected neurons processes sequential input data. In our setup, given that the state of the environment  $\mathbf{s}_t \in \mathbb{R}^{n_s}$  is given as the input to the network, the hidden states and output of our policy network are given by  $\mathbf{h}_t \in \mathbb{R}^{n_h}$  and  $\boldsymbol{\pi}_t \in \mathbb{R}^{n_a}$ , respectively. The temporal evolution of such a network is governed by the following, non-linear dynamical system

$$\mathbf{h}_t = (1 - \beta) \mathbf{h}_{t-1} + \beta f_h(\mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{W}^s \mathbf{s}_t) \quad (3)$$

$$\boldsymbol{\pi}_t = f_\pi(\mathbf{W}^a \mathbf{h}_t) \quad (4)$$

where  $\beta \in [0, 1]$  is called the leaking rate and  $\mathbf{W}^s$ ,  $\mathbf{W}^h$ , and  $\mathbf{W}^a$  are the input, reservoir, and output weight matrices, respectively. The function  $f_h(\cdot)$  is understood to act component-wise on its argument and is typically a sigmoidal activation function. For the output layer, however,  $f_\pi(\cdot)$  is usually just a linear or softmax function depending on the application context.

### 3.3 Policy Learning Using Echo State Networks

At any time, the goal of the agent is to maximize the expected cumulative reward or the return received over a period of time which is defined as  $R_T = \sum_{t=1}^T r_t$ .

Hence, the objective function that is to be maximized is  $f(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_T]$  and finding an optimal policy amounts to finding  $\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$  where we now write  $\boldsymbol{\theta}$  to denote the set of weights of an echo state network used to approximate the policy  $\pi(a_t | \mathbf{s}_t, \boldsymbol{\theta})$ .

According to our discussion in Sect. 2, we can then iteratively learn an optimal  $\boldsymbol{\theta}$  according to a stochastic gradient ascent rule that follows the gradient  $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_T]$ . In particular, we can resort to SPSA in order to approximate this gradient as

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_T] \approx \frac{F(\boldsymbol{\theta} + \boldsymbol{\epsilon}) - F(\boldsymbol{\theta} - \boldsymbol{\epsilon})}{2\boldsymbol{\epsilon}} \quad (5)$$

where  $F(\cdot)$  is the stochastic return from the environment by running an episode where, in each step, the agent follows the policy  $\pi(a_t | \mathbf{s}_t, \boldsymbol{\theta})$  approximated by the ESN and where  $\boldsymbol{\epsilon}$  is the perturbation generated by SPSA. A summary of this learning method can be found in Algorithm 1.

### 3.4 Deterministic and Stochastic Policies

An agent’s policy can either be deterministic or stochastic. In a discrete action space, the agent may apply a deterministic, greedy, “winner-takes-all” strategy to select an action, i.e.  $a_t = \operatorname{argmax}_a \pi(a | \mathbf{s}_t, \boldsymbol{\theta})$ . However, in order to encourage exploration, the agent can follow a stochastic softmax policy in which actions are sampled based on action probabilities according to the policy  $\pi(a_t | \mathbf{s}_t, \boldsymbol{\theta})$ , i.e.  $a_t \sim f_\pi$  where  $f_\pi$  is the softmax function. In a continuous action space, the agent’s actions are sampled from a Gaussian policy parameterized by mean and variance neurons, that is  $f_\pi$  is considered a Gaussian probability distribution.

---

**Algorithm 1.** Learn policies using SPSA

---

**Input:** SPSA parameters  $l, c, L, \alpha, \gamma$  and initial weight  $\theta_0$ **for**  $k = 0$  **to**  $k_{max}$  **do**

$$l_k = \frac{l}{(L + k)^\alpha}$$

$$c_k = \frac{c}{k^\gamma}$$

$$\delta_k \sim \mathcal{U}(-1, 1)$$

$$\theta^+ = \theta_k + c_k \delta_k$$

$$\theta^- = \theta_k - c_k \delta_k$$

Compute returns  $F(\theta^+)$  and  $F(\theta^-)$  by running an episode with weights  $\theta^+$  and  $\theta^-$  respectively

$$\hat{g}_k(\theta_k) = \frac{F(\theta^+) - F(\theta^-)}{2 c_k \delta_k}$$

$$\theta_{k+1} = \theta_k + l_k \hat{g}_k(\theta_k)$$

**end**

---

### 3.5 Three Variants of Echo State Network Training

Typically, in echo state networks, only the output weight matrix  $\mathbf{W}^a$  is optimized. However some tasks require tuning of the input- and reservoir weights  $\mathbf{W}^s$  and  $\mathbf{W}^h$  in order to extract relevant information from observations or to construct missing state information. Therefore, we consider three variants of our SPSA algorithm using different choices of  $\theta$  at each iteration

1. **output\_spsa**: at each iteration, we optimize only the output weight matrix, that is we let  $\theta = \mathbf{W}^a$
2. **all\_spsa**: at each iteration, all of the weight matrices are updated at once, that is we let  $\theta = \{\mathbf{W}^s, \mathbf{W}^h, \mathbf{W}^a\}$
3. **alternating\_spsa**: at each iteration, we update one of these matrices and alternate in the subsequent iteration.

## 4 Experiments and Results


We evaluated the above SPSA variants on a benchmark of classic control problems available from OpenAI Gym [2] and compared them against temporal difference and policy gradient learning methods.

### 4.1 Acrobot and Mountain Car


We considered two classic problems, namely Mountain Car and Acrobot, and considered discrete and continuous action selection. For both problems, we restrict state observations to include only positional information excluding velocities so that the agent has to infer velocity information in order to retrieve the full state information. An illustration of these OpenAI Gym problems and their state-action spaces is given in Fig. 1.

State and Action space	
Acrobot	<p><b>State:</b> cosine and sine of two joint angles</p> <p><b>Action:</b> the action is either applying +1, 0 or -1 torque on the joint between two links</p>
Mountain Car	<p><b>State:</b> 1-dimensional position of a car</p> <p><b>Action:</b> for the discrete version, the action is either push left, no push and push right; for the continuous version, the action is a scalar force</p>

(a)



(b)



(c)

**Fig. 1.** Test environments: (a) description of observation and action space for the acrobot and mountain car tasks; (b), (c) task illustration from OpenAI Gym.

## 4.2 Implementation Details

We used the same architecture of echo state networks consisting of 40 reservoir neurons with *tanh* activation functions for our SPSA variants and their RL baselines. The number of input- and output neurons, and the output activation function are chosen depending on the task and the type of policy being learned. The weight matrices are initialized according to parameters such as sparsity, scaling and spectral radius which are carefully set as per the guidelines in [9]. The input and reservoir matrices are chosen from a uniform distribution over values  $[-0.5, 0.5]$ . However, the output scaling is chosen differently for each task. The initial spectral radius of the reservoir matrix and the leaking rate are chosen to be 1.0 and 0.3, respectively, for all tasks. The SPSA parameters such as learning rate, scaling factor, decay rates and similarly, parameters concerning reinforcement learning methods such as discount factor and learning rates are tuned for each experiment. Table 1 lists all hyper parameters and their values.

## 4.3 Results

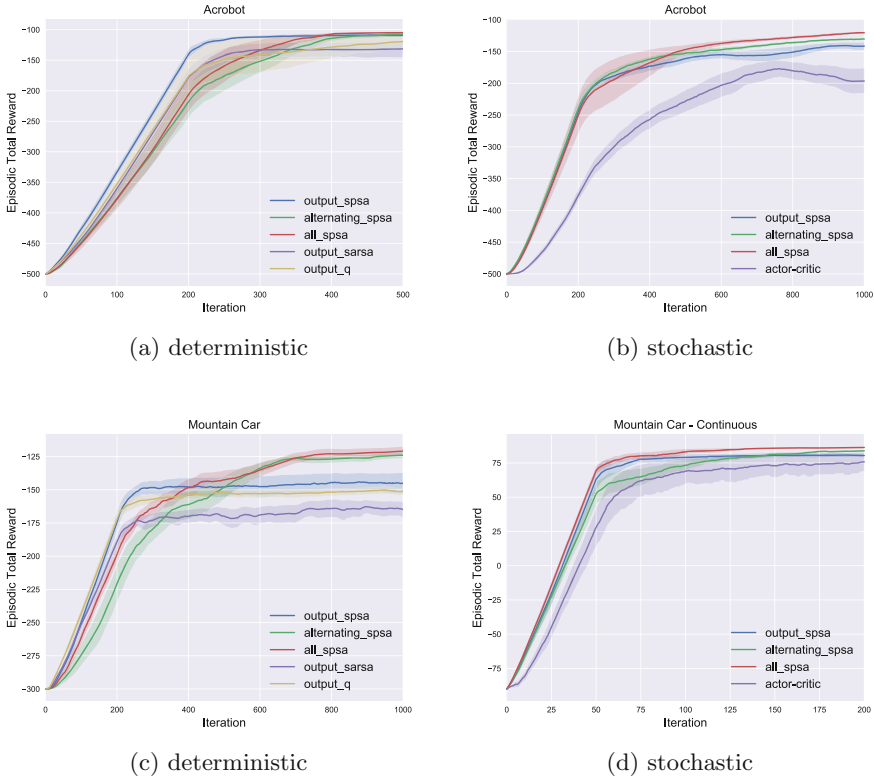
First, we tested our algorithms to train deterministic greedy policies for the discrete versions of the acrobot and mountain car tasks and found that SPSA variants are able to solve both these tasks. In a quantitative evaluation, we computed mean learning curves with 10 different random seeds and compared them to similar curves obtained using echo state networks trained with temporal difference methods such as Q-learning and SARSA learning using stochastic gradient descent. Figures 2(a) and (b) show the learning curves in terms of the evolution of episodic total reward in the learning process (the higher the better). As we can observe, all SPSA variants find better policies than Q-learning or SARSA learning.

**Table 1.** Hyperparameters and their values for different experiments.

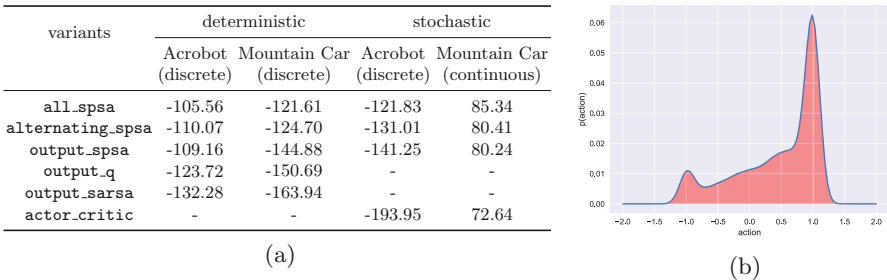
Category	Parameter	Deterministic		Stochastic	
		Acrobot (discrete)	Mountain car (discrete)	Acrobot (discrete)	Mountain car (continuous)
SPSA	Learning rate ( $l$ )	1e-6	1e-3	5e-5	5e-3
	Scaling factor ( $c$ )	1e-1	1e-1	1e-1	1e-1
	$L$	10	100	10	100
	$\alpha$	0.102	0.602	0.102	0.602
	$\gamma$	0.101	0.101	0.101	0.101
ESN	Reservoir size	40	40	40	40
	Input connectivity	0.7	0.3	0.3	0.7
	Reservoir connectivity	0.7	0.3	0.7	0.7
	Output scaling	0.1	0.1	1e-5	1e-2
	Spectral radius	1.0	1.0	1.0	1.0
	Leaking rate	0.3	0.3	0.3	0.3
RL	Discount factor	0.99	1.0	0.99	0.99
	Learning rate	1e-2	1e-2	1e-3	1e-3

Next, we tested our algorithms to learn stochastic policies for a discrete version of the acrobot- and a continuous version of the mountain car task. We found that the SPSA variants are able to solve these by finding a softmax policy and a Gaussian policy for acrobot and mountain car, respectively. In a quantitative evaluation, we again computed mean learning curves with 10 different random seeds and compared them to data obtained using actor-critic methods. In the actor-critic method, two echo state networks are used, one to learn the policy (policy network) and one to learn the state value function (value network), both act with limited state information as in our SPSA variants. Figures 2(c) and (d) show the learning curves and it is seen that SPSA variants perform better than actor-critic methods. Next, in order to visualize the learned Gaussian policy for the mountain car task, we plotted action probabilities for selected input states. As we can see in Fig. 3(b), for the same input states, the resulting action probability distribution is a mixture of Gaussians, meaning that the actions are sampled from appropriate mixture components based on the hidden states of the network which constructs the missing velocity information.

Our most important evaluation results are summarized in Fig. 3(a) which shows average episodic total rewards in the last 100 iterations with 10 different random seeds. Here we observe: (i) training only the output weight matrix using SPSA yields better performance than its RL counterparts in all the experiments which indicate that SPSA is a powerful alternative to common RL methods; (ii) updating all the weight matrices at once gives the best performance in all tasks; however, training in an alternating fashion also seems to be a promising approach which warrants for further investigation; (iii) for the acrobot tasks, it is evident that SPSA works better in learning a deterministic policy than a stochastic policy. The reason could be that it is not necessary to also introduce stochasticity into action space since the exploration happens already in



**Fig. 2.** Learning curves: (a), (c) evolution of episodic total reward in learning deterministic policies for discrete versions of acrobot and mountain car. (b), (d) evolution of episodic total reward in learning of a softmax and Gaussian policy for discrete acrobot and continuous mountain car problems tasks, respectively. It is evident that SPSA variants perform better than RL methods



**Fig. 3.** Performance summary: (a) evaluation results containing average episodic total reward in the last 100 iterations of policy learning on classic problems for different variants and their baselines (the higher the value, the better the performance) (b) visualization of a Gaussian policy learned for the mountain car task.



parameter space in terms of perturbations. This concurs with the work done in [14] whose authors also seek to learn a deterministic policy when using black-box methods. Nevertheless, our approach demonstrates the general feasibility of learning both deterministic- and stochastic policies.

## 5 Conclusion

In this paper, we considered the use of SPSA in training echo state networks to solve action selection tasks under partial observability. We proposed three variants that seek to perform gradient updates without using back-propagation. Experiments on classic problems indicate that SPSA is a powerful alternative to reinforcement learning methods commonly used for policy learning. In future work, we intend to extend the ideas reported here using LSTM units to solve more complex RL problems that require long-term dependencies. We also plan to examine the alternating SPSA variant further to verify their applicability in training deep recurrent neural networks.

## References

1. Bertsekas, D.P.: *Neuro-Dynamic Programming*. Athena Scientific, Belmont (1996)
2. Brockman, G., et al.: OpenAI Gym. [arXiv:1606.01540](https://arxiv.org/abs/1606.01540) (2016)
3. Chatzidimitriou, K.C., Mitkas, P.A.: A NEAT way for evolving echo state networks. In: *Proceedings of European Conference on Artificial Intelligence* (2010)
4. Duan, Y., Chen, X., Houthoofd, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: *Proceedings of International Conference on Machine Learning* (2016)
5. Jäger, H.: The “echo state” approach to analysing and training recurrent neural networks. Technical report 148, GMD (2001)
6. Jiang, F., Berry, H., Schoenauer, M.: Supervised and evolutionary learning of echo state networks. In: *Proceedings of International Conference on Parallel Problem Solving from Nature* (2008)
7. Koprinkova-Hristova, P.: Three approaches to train echo state network actors of adaptive critic design. In: *Proceeding of International Conference on Artificial Neural Networks* (2016)
8. Lin, L.J.: *Reinforcement learning for robots using neural networks*. Technical reports CMU-CS-93-103, Carnegie-Mellon University (1993)
9. Lukoševičius, M.: A practical guide to applying echo state networks. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) *Neural Networks: Tricks of the Trade*. LNCS, vol. 7700, pp. 659–686. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-35289-8\\_36](https://doi.org/10.1007/978-3-642-35289-8_36)
10. Mania, H., Guy, A., Recht, B.: Simple random search provides a competitive approach to reinforcement learning. [arXiv:1803.07055](https://arxiv.org/abs/1803.07055) (2018)
11. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: *Proceedings of International Conference on Machine Learning* (2016)
12. Mnih, V., et al.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529 (2015)

13. Robbins, H., Monro, S.: A stochastic approximation method. *Ann. Math. Stat.* **22**(3), 400–407 (1951)
14. Salimans, T., Ho, J., Chen, X., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. [arXiv:1703.03864](https://arxiv.org/abs/1703.03864) (2017)
15. Schmidhuber, J., Wierstra, D., Gagliolo, M., Gomez, F.: Training recurrent networks by Evolino. *Neural Comput.* **19**(3), 757–779 (2007)
16. Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J.J., Stroobandt, D.: Improving reservoirs using intrinsic plasticity. *Neurocomputing* **71**(7–9), 1159–1171 (2008)
17. Silver, D., et al.: Mastering the game of Go without human knowledge. *Nature* **550**(7676), 354 (2017)
18. Spall, J.C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Autom. Control.* **37**(3), 332–341 (1992)
19. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O., Clune, J.: Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. [arXiv:1712.06567](https://arxiv.org/abs/1712.06567) (2017)
20. Sutton, R.: Introduction to reinforcement learning with function approximation. In: Tutorial at the Conference on Neural Information Processing Systems (2015)
21. Sutton, R.S., Barto, A.G., et al.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)