Step Size Adaptation in Evolution Strategies using Reinforcement Learning

Sibylle D. Müller, Nicol N. Schraudolph, Petros D. Koumoutsakos
Institute of Computational Sciences
Swiss Federal Institute of Technology
CH - 8092 Zürich, Switzerland
muellers,nic,petros@inf.ethz.ch

Abstract- We discuss the implementation of a learning algorithm for determining adaptation parameters in evolution strategies. As an initial test case, we consider the application of reinforcement learning for determining the relationship between success rates and the adaptation of step sizes in the (1+1)-evolution strategy. The results from the new adaptive scheme when applied to several test functions are compared with those obtained from the (1+1)-evolution strategy with a priori selected parameters. Our results indicate that assigning good reward measures seems to be crucial to the performance of the combined strategy.

1 Introduction

The development of step size adaptation schemes for evolution strategies (ES) has received much attention in the ES community. Starting from an early attempt, the so-called "1/5 success rule" [1], applied on two-membered ES's, mutative step size control [2] and self-adaptation schemes [3] were developed, followed by derandomized mutational step size control schemes [4], [5]. The latter two methods have become state-of-the-art techniques that are usually implemented in ES's. These control schemes employing empirical rules and parameters have been proven successful for solving a wide range of real-world optimization problems.

We consider replacing a priori defined adaptation rules by a more general mechanism that can adapt the step sizes during the evolutionary optimization process automatically. The key concept involves the use of a learning algorithm for the online step size adaptation. This implies that the optimization algorithm is not supplied with a pre-determined step size adaptation rule but instead the rules are evolved by means of learning.

As an initial test for our approach, we consider the application of reinforcement learning (RL) to the 1/5 success rule in a two-membered ES.

In Section 2, we present the concept of RL and an overview of algorithms considered for our problem. The combination of RL with the ES is shown in Section 3 and results are presented in Section 4.

2 Reinforcement Learning

Reinforcement learning (RL) is a learning technique in which an *agent* learns to find optimal *actions* for the current *state* by interacting with its environment. The agent should learn a control strategy, also referred to as *policy*, that chooses the optimal actions for the current state to maximize its cumulative *reward*. For this purpose, the agent is given a reward by an external trainer for each action taken. The reward can be immediate or delayed. Sample RL applications are learning to play board games (e.g. Tesauro's backgammon [7]) or learning to control mobile robots, see e.g. [6]. In the robot example, the robot is the *agent* that can sense its environments such that it knows its location, i.e., its *state*. The robot can decide which *action* to choose, e.g., to move ahead or to turn from one state to the next. The goal may be to reach a particular location and for this purpose the agent has to learn a *policy*. The diagram in Figure 1 shows the interaction between agent and environment in RL.

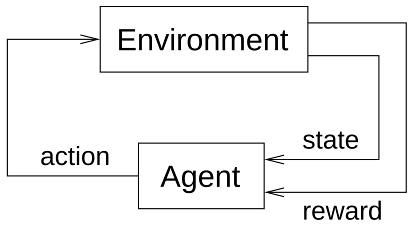

Figure 1: The interaction between environment and agent in RL.

2.1 The Learning Task

The learning task can be divided into discrete time steps, t. The agent determines at each step the environmental state s_t, and decides upon an action a_t. By performing this action, the agent is transferred to a new state $s_{t+1} = \delta(s_t, a_t)$ and given a reward r_{t+1}. This reward is used to update a *value function* that can be either a *state-value function* $V^\pi(s)$ depending on states or an *action-value function* $Q^\pi(s, a)$ depending on states and actions. To each state or state-action pair, the largest expected future reward is assigned by the *optimal* value functions $V^*(s)$ or $Q^*(s, a)$, respectively. The optimal state-value function $V^*(s)$ can be learned only if both the reward function r and the function δ that describes how the agent is transferred from state s_t to state s_{t+1} are explicitely known. Usually, however, r and δ are unknown, In this case, the optimal action-value function $Q^*(s, a)$ can be learned.

2.2 Temporal Difference Learning

In this paper, we consider only RL methods for which the agent can learn the Q function, thereby being able to select optimal actions without knowing explicitely the reward function r and the function for the state $\delta(s,a)$ resulting from applying action a on state s. From this class of Temporal Difference learning (TD), we present two algorithms, namely Q-learning and SARSA. Both need to wait only until the next time step to determine the increment to $Q(s_t, a_t)$. Such techniques, denoted as TD(0) schemes, combine the advantages of both Dynamic Programming and Monte Carlo methods [8]. The difference between Q-learning and SARSA lies in the treatment of estimation (updating the value function) and choice of a policy (selecting an action). In off-policy algorithms such as Q-learning, the choice and the estimation of a policy are separated. In on-policy algorithms such as SARSA, the choice and the estimation of a policy are combined. Experiments in the "Cliff Walking" example in [8] compare Q-learning and SARSA. The results of these experiments indicate that the Q-learning method approaches the goal faster but it fails more often than SARSA. Focussing more on safety than on speed, we decided to implement SARSA for our online learning problem presented in the next section. It should be noted that the convergence of SARSA(0) was proven only recently [10]. The SARSA pseudocode reads as shown in Figure 2 [8].

```
Initialize $Q(s_t, a_t)$ arbitrarily.
Repeat (for each episode):
  Initialize $s_t$.
  Choose $a_t$ from $s_t$ using policy derived from $Q(s_t, a_t)$
  using e.g. an $\varepsilon$-greedy selection scheme.
  Repeat (for each step of episode):
    Take action $a_t$, observe $r_t$, $s_{t+1}$.
    Choose $a_{t+1}$ from $s_{t+1}$ using policy from $Q(s_{t+1}, a_{t+1})$
    using e.g. an $\varepsilon$-greedy selection scheme.
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
    $s_t \leftarrow s_{t+1}, a_t \leftarrow a_{t+1}$
  until $s_t$ is terminal.
```

Figure 2: The SARSA algorithm

2.3 Learning Parameters

SARSA employs three learning parameters: α is a learning rate, γ a discount factor, and ε a greediness parameter. Constant learning rates α cannot be used because we have to deal with a non-deterministic environment. For the considered problem, a learning rate of

$$\alpha_t = \frac{1}{N_v(s_t, a_t)} \tag{1}$$

is recommended in [11] where N_v is the total number of times the state-action pair (s_t, a_t) has been visited until and including the t-th iteration. Discount factors $\gamma < 1$ are necessary for continuing learning tasks that are dealt with, in order to avoid that the expected future reward becomes infinite. We set $\gamma = 0.9$ arbitrarily. The greediness parameter ε means that the action with the highest value function is chosen with a probability $(1-\varepsilon)$ (greedy action-selection) and that an action is selected at random with (a small) probability ε. We choose $\varepsilon = 0.1$.

3 Combination of RL and ES

The idea is to use a RL method to learn a step size adaptation rule in ES's. We consider a two-membered ES employing the 1/5 success rule. The 1/5 success rule was originally formulated as follows [9]:

After every n mutations, check how many successes have occured over the preceding $10n$ mutations. If this number is less than $2n$, multiply the step lengths by the factor 0.85; divide them by 0.85 if more than $2n$ successes occured.

In our approach, the frequency with which step sizes are updated is kept the same (after every $10n$ mutations). Also, the step size adaptation factor of 0.85 remains constant.

RL is introduced to learn from the measured states, i.e. the success rates; the actions can be (1) to increase the step size (divide by the step size adaptation factor), (2) to decrease the step size (multiply by the step size adaptation factor), or (3) to keep the step size constant. As the success rate is determined by looking back over the last $10n$ mutations, the total number of different states is $10n + 1$, including the case of a success rate of zero. Therefore, the $Q(s,a)$ table to be learned consists of $(10n+1)\cdot 3$ state-action pairs. We describe four different approaches to assign rewards in Section 4. $Q(s,a)$ is initialized with uniformly random numbers from the range [-1,1]. The combined algorithm is called (1+1)-RL-ES, or short RL-ES.

4 Results

The RL algorithm is tested in the optimization of the sphere and the Rosenbrock function in several dimensions and compared with the original (1+1)-ES. The two functions are defined as

1. $f_{sphere}(\boldsymbol{x}) = \Sigma_{i=1}^{n}(x_i - 1)^2, \sigma^{(0)} = 1, \boldsymbol{x}^{(0)} = \mathbf{0}$,
2. $f_{Rosenbrock}(\boldsymbol{x}) = \Sigma_{i=1}^{n-1}(100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2), \sigma^{(0)} = 0.1, \boldsymbol{x}^{(0)} = \mathbf{0}$,

and the optimization is terminated as soon as $f < 10^{-10}$. If the termination criterion is not met within N_t generations, the run is called *not converged*. We determine a *convergence rate* that is the ratio of converged runs over the total number of runs. The *convergence speed* is measured by counting the number of function evaluations until convergence.

The convergence rate and the average and standard deviation for the number of generations to reach convergence are listed in Table 1 for the (1+1)-ES and in Table 2 for two methods of the RL-ES.

In the RL-ES, the lower bound for the step size is set to 10^{-15} while the upper bound is set to 10^{15} based on our experience about useful step size limits. If the limits are exceeded, the reward is set to -1. From our experiments, defining proper limits turned out to be a crucial factor.

In the following subsections, we discuss the results for different ways to define a reward measure.

Problem	(1+1)-ES
Sph 1D	97 ± 17
Sph 5D	470 ± 35
Sph 20D	1928 ± 70
Sph 80D	8234 ± 80
Ros 2D	16205 ± 973
Ros 5D	140227 ± 1487

Table 1: Number of iterations until convergence is reached for the (1+1)-ES. Results are averaged over 30 runs, and the convergence rate is 1.0 for all problems.

Problem	RL-ES (Method 1) (conv. rate; N_t)	RL-ES (Method 2) (conv. rate; N_t)
Sph 1D	709 ± 1391 (0.72; 10^4)	349 ± 613 (1.00; 10^4)
Sph 5D	1642 ± 1513 (0.64; 10^4)	1350 ± 987 (0.98; 10^4)
Sph 20D	4534 ± 1764 (0.50; 10^4)	4510 ± 1857 (0.92; 10^4)
Sph 80D	24610 ± 16277 (0.57; 10^5)	20481 ± 10440 (1.00; 10^5)
Ros 2D	129231 ± 178327 (0.63; 10^6)	46843 ± 71354 (0.99; 10^6)
Ros 5D	2872258 ± 2440214 (0.77; 10^7)	377565 ± 285542 (1.00; 10^7)

Table 2: Number of iterations until convergence is reached for two methods of (1+1)-RL-ES. *Method 1* (described in Section 4.1) denotes the original reward definition in terms of a success rate increase or decrease while *method 2* (described in Section 4.2) uses as a reward measure the difference in function values between the current and the last reward computation. Results are averaged over 1000 runs for the sphere and over 30 runs for Rosenbrock's function.

4.1 Method 1

In a first approach called *method 1*, the reward from the environment is defined to be either +1 if the success rate has increased, 0 if the success rate did not change, or -1 if the success rate decreased.

For the sphere, this RL-ES method converges about 3 to 7 times slower than using the (1+1)-ES. For Rosenbrock's function, the RL-ES achieves an iteration number about one order of magnitude worse than the (1+1)-ES.

Note that for both functions, the RL-ES is extremely unrobust. Convergence rates as low as 50 % are unacceptable. From our investigation on the sphere function, the basic problem is that with the selected scheme no information is available if the success rate is zero. When the strategy is in a state of zero success, it often either oscillates between choosing the actions "increase" and "decrease" or it gets stuck by always choosing the action "keep". A no-success run usually happens if the Q values are initialized such that in the first phase of the optimization the step size is increased. This causes a zero success rate and often yields one of the two behaviors described above.

Another interesting feature is the $Q(s, a)$ table at the end of the optimization and the $N_v(s, a)$ table. From the 1/5 success rule, we would expect a Q value table as shown for the 1D case in Table 3. Note that "+" denotes the highest Q value per row.

Success rate [$\cdot 10^1$]	Action: increase	Action: decrease	Action: keep
0,1		+	
2			+
3,4,5,6,7,8,9,10	+		

Table 3: Schema of the Q value table as it should look if the 1/5 success rule is learnt. The "+" denotes the highest Q value per row.

As it turns out, such a structure in the actual Q table at the end of the optimization using RL-ES may be found but it may as well be structured differently without much change in terms of convergence speed. One example for an actually obtained Q table (Table 4) and N_v table (Table 5) is shown below for the optimization of a 1D sphere function that took 108 generations. The * symbol indicates that the state-action pair was not visited during the optimization and the highest values are in bold face. Note that for success rates between 0.4 and 1.0 no learning has taken place. Then, the Q values are the initialized values. The highest Q values are found for the action "decrease" when the success rates are 0, 0.1, and 0.2. The action "increase" is assigned the highest Q value for a success rate of 0.3. Except for a success rate of 0.2, the obtained Q values match our expectations from the 1/5 success rule. The number of visits for each state-action pair also reflect that the correct actions (according to the 1/5 rule) have been selected in this particular case.

Changing the initialization of the Q table from uniformly random numbers in the range [-1,1] to zero values did not have any effect on the results.

When we compare the reward assignment in method 1 with the 1/5 success rule, we observe that our definition is

Success rate [$\cdot 10^1$]	Action: increase	Action: decrease	Action: keep
0	* -0.715020	**0.833568**	0.390386
1	0.329887	**1.073497**	0.017461
2	-0.340735	**0.810838**	* 0.451931
3	**0.008306**	* -0.868088	* -0.563195
4,5,6,7,8,9,10	*	*	*

Table 4: Q values for a converged optimization using RL-ES after 108 generations. The * symbol indicates that the state-action pair was not visited during the optimization. The highest values are in bold face.

Success rate [$\cdot 10^1$]	Action: increase	Action: decrease	Action: keep
0	* 0	**48**	2
1	3	**23**	2
2	* 0	**7**	* 0
3	**13**	* 0	* 0
4,5,6,7,8,9,10	* 0	* 0	* 0

Table 5: Number of visits N_v for a converged optimization using RL-ES after 108 generations. The * symbol indicates that the state-action pair was not visited during the optimization. The highest values are in bold face.

ill-posed. Recall that the 1/5 rule aims at an optimum success rate of 0.2. In contrast, the definition in method 1 assigns a positive reward whenever the success rate is increased even if the success rate is ≥ 0.2. Despite this ill-posed reward assignment, the results for the sphere are not affected so much because success rates larger than 0.2 are achieved less often than success rates ≤ 0.2. However, for Rosenbrock's function, the difference matters.

4.2 Method 2

In a second approach, we define as reward the difference between the current function value and the function value evaluated at the last reward computation,

$$reward = f^{(g)} - f^{(g-\Delta g)} \tag{2}$$

where Δg is the difference in generations for which the reward is computed. This reward assignment, referred to as *method 2* in the following, is better than the initial reward computation in both the convergence speed and rate. Values are given in Table 2. Although better than the original reward computation in terms of speed, this RL-ES remains slower by a factor of about 3 than the (1+1)-ES. A factor of 3 seems reasonable given the fact that the RL-ES has to learn which of the three actions to choose. Especially the convergence rate is noteworthy: It lies in the range [0.92,1.0], which is a great improvement compared with method 1.

Why is method 2 better than method 1 in terms of the convergence rate? One reason might be that the reward assignment in method 1 is ill-posed as stated earlier. Another reason is that the second reward assignment is related to the definition of the progress rate, at least for the sphere function:

The progress rate is defined [9] as

$$\varphi = \mathrm{E}(||\boldsymbol{x}^{(g)} - \boldsymbol{x}_{opt}|| - ||\boldsymbol{x}^{(g-\Delta g)} - \boldsymbol{x}_{opt}||) \tag{3}$$

For the sphere function, $f_{sphere}(\boldsymbol{x}) = \Sigma_{i=1}^{n}(x_i - 1)^2$, with the optimum at $\boldsymbol{x}_{opt} = \mathbf{1}$, we have that

$$\begin{aligned}\varphi &= \mathrm{E}\left(||\boldsymbol{x}^{(g)} - \mathbf{1}|| - ||\boldsymbol{x}^{(g-\Delta g)} - \mathbf{1}||\right)\\ &= \mathrm{E}\left(\sqrt{\Sigma_{i=1}^{n}(x_i^{(g)} - 1)^2} - \sqrt{\Sigma_{i=1}^{n}(x_i^{(g-\Delta g)} - 1)^2}\right)\\ &= \mathrm{E}\left(\sqrt{f_{sphere}^{(g)}} - \sqrt{f_{sphere}^{(g-\Delta g)}}\right).\end{aligned}$$

For the sphere, the progress rate is the difference between the square roots of function values, a result similar to the reward assignment in Eqn. 2.

The results in Table 6 document the behavior of the strategy with the reward identified as the theoretical progress rate φ. The theoretical results agree well with method 2 which strengthens our assumption that method 2 works well because it indirectly incorporates information about the optimal parameter vector.

In summary, assigning a good reward measure seems to be crucial to the performance of the RL-ES.

Problem	RL-ES (Theoretical progress rate) (conv. rate; N_t)
Sph 1D	382 ± 721 (1.00; 10^4)
Sph 5D	1468 ± 1169 (0.98; 10^4)
Sph 20D	4690 ± 1862 (0.90; 10^4)
Sph 80D	20147 ± 11491 (0.98; 10^5)
Ros 2D	19562 ± 28363 (1.00; 10^6)
Ros 5D	314785 ± 255380 (1.00; 10^7)

Table 6: Number of iterations until convergence is reached for the (1+1)-RL-ES method which employs the theoretical progress rate as reward, as described in Section 4.2. Results are averaged over 1000 runs for the sphere and over 30 runs for Rosenbrock's function.

4.3 Methods with Optimum-Independent Reward Assignments

As seen above, defining the reward as the theoretical progress rate is a good measure. However, the theoretical progress rate assumes knowledge about the optimum that is usually unknown. How can we formulate a suitable reward that approximates the theoretical progress rate using only values that can be measured while optimizing? We can consider two possible forms, namely

- Form 1:
  The sign of the difference between function values, $\text{sgn}\left(f^{(g)} - f^{(g-\Delta g)}\right)$: It describes if the realized step, $(\boldsymbol{x}^{(g)} - \boldsymbol{x}^{(g-\Delta g)})$, points in the half space in which the optimum lies.

- Form 2:
  The realized step length, $||\boldsymbol{x}^{(g)} - \boldsymbol{x}^{(g-\Delta g)}||$. It is an approximation of the theoretical progress rate.

Method 3 employs only the first form,

$$reward = \text{sgn}\left(f^{(g)} - f^{(g-\Delta g)}\right) \tag{4}$$

while *method 4* combines both forms,

$$reward = ||\boldsymbol{x}^{(g)} - \boldsymbol{x}^{(g-\Delta g)}|| \cdot \text{sgn}\left(f^{(g)} - f^{(g-\Delta g)}\right) . \tag{5}$$

Results of the two methods are summarized in Table 7.

Problem	RL-ES (Method 3) (conv. rate; N_t)	RL-ES (Method 4) (conv. rate; N_t)
Sph 1D	522 ± 968	337 ± 639
	(0.96; 10^4)	(1.00; 10^4)
Sph 5D	938 ± 1440	1346 ± 1092
	(0.94; 10^4)	(0.99; 10^4)
Sph 20D	4123 ± 1687	4313 ± 1795
	(0.90; 10^4)	(0.94; 10^4)
Sph 80D	18221 ± 10265	19382 ± 9729
	(0.99; 10^5)	(1.00; 10^5)
Ros 2D	172568 ± 215376	31365 ± 65135
	(0.80; 10^6)	(1.00; 10^6)
Ros 5D	930055 ± 1649505	311704 ± 319148
	(0.77; 10^7)	(1.00; 10^7)

Table 7: Number of iterations until convergence is reached for methods 3 and 4 of (1+1)-RL-ES, as described in Section 4.3. Results are averaged over 1000 runs for the sphere and over 30 runs for Rosenbrock's function.

For the sphere problem, the convergence speeds of methods 3 and 4 are similar and they are in the same range as with method 2. For Rosenbrock's function, method 3 is worse than methods 2 and 4, while 2 and 4 yield similar convergence speeds. The convergence rates of methods 2 and 4 are almost the same while method 3 optimizes less reliably especially for Rosenbrock's problem. In summary, method 4 seems to be better than 3 and compares well with method 2 in which information about the optimum is contained. From these preliminary results that are problem dependent, we propose the fourth method as a reward assignment for RL-ES.

4.4 Action-Selection Scheme

How is the choice of the action-selection parameter ε influencing the convergence speed? The average number of iterations to reach the goal in the 1D sphere problem as a function of ε is shown in Table 8. The number of iterations is averaged over 1000 runs, and $N_t = 10^4$. For all ε values, the success rate is in the range [0.68,0.72] for method 1 and [0.66,1.0] for method 2. Optimum strategy parameter for the considered cases lie close to $\varepsilon = 0.05$ for method 1 and $\varepsilon = 0.5$ for method 2. However, these results are not conclusive.

ε	Method 1: Average number of iterations	Method 2: Average number of iterations
1.0	2847	3066
0.5	1242	327
0.1	709	349
0.05	652	457
0.01	688	677
0.001	1017	901

Table 8: Influence of ε on the average number of iterations for the 1D sphere function, measured in 1000 runs and $N_t = 10^4$.

5 Summary and Conclusions

We propose an algorithm that combines elements from step size adaptation schemes in evolution strategies (ES) with reinforcement learning (RL). In particular, we tested a SARSA(0) learning algorithm on the 1/5 success rate in a (1+1)-ES. Heuristics in the (1+1)-ES were reduced and replaced with a more general learning method. The results in terms of convergence speed and rate, measured on the sphere and Rosenbrock problem in several dimensions, suggest that the performance of the combined scheme (called RL-ES) depends strongly on the choice of the reward function. In particular, the RL-ES with a reward assignment based on a combination of the realized step length and the sign of the function values yields the same convergence rate (100 %) as the (1+1)-ES and its convergence speed is smaller than that of the (1+1) strategy by a factor of about 3 for both sphere and Rosenbrock's function, a result that meets our expectations.

Future work may answer the question whether the proposed reward computation can be generalized for non-tested optimization problems.

Bibliography

[1] Rechenberg, I., ”Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution,” Fromann-Holzboog, Stuttgart, 1973.

[2] Rechenberg, I., ”Evolutionsstrategie ’94,” Fromann-Holzboog, Stuttgart, 1994.

[3] Bäck, Th.: ”Evolutionary Algorithms in Theory and Practice,” Oxford University Press, 1996.

[4] Hansen, N., Ostermeier, A., ”Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation,” *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC’96)*, pp. 312-317, 1996.

[5] Hansen, N., Ostermeier, A., ”Convergence Properties of Evolution Strategies with the Derandomized Covariance Matrix Adaptation: The $(\mu/\mu_I, \lambda)$-CMA-ES,” *Proceedings of the 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT’97)*, pp. 650-654, 1997.

[6] Mahadevan, S., Connell, J., ”Automatic programming of behavior-based robots using reinforcement learning,” *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991.

[7] Tesauro, G., ”Temporal difference learning and TD-Gammon,” *Communications of the ACM*, 38(3), pp.58-68, 1995.

[8] Sutton, R.S., Barto, A.G., ”Reinforcement Learning – An Introduction,” MIT Press, Cambridge, 1998.

[9] Schwefel, H.-P., ”Evolution and Optimum Seeking,” John Wiley and Sons, New York, 1995.

[10] Singh, S., Jaakkola, T., Littman, M.L., Szpesvari, C., ”Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms,” *Machine Learning*, 1999.

[11] Mitchell, T.M., ”Machine Learning,” McGraw-Hill, 1997.