Abstract

In recent years, attention has been focused on the relationship between black box optimization and reinforcement learning. Black box optimization is a framework for the problem of finding the input \( x^* \in \mathcal{X} \) that optimizes the output represented by an unknown function. Because the objective function is unknown, we solve the black box optimization problem without gradient information. Reinforcement learning, by contrast, is a framework for finding a policy to optimize the expected cumulative reward from trial and error. Based on this, the solution to the black box optimization problem can be used as a solution for reinforcement learning.

In this research, we propose a reinforcement learning algorithm based on the mirror descent (MD) method \cite{2}. MD is general optimization algorithm that employs a Bregman

1. Introduction

In recent years, as stated in \cite{1}, attention has focused on the relationship between black box optimization and reinforcement learning. Black box optimization is a framework for the problem of finding the input \( x^* \in \mathcal{X} \) that optimizes the output \( f(x) : a \rightarrow \mathbb{R} \) represented by an unknown function. Because the objective function is unknown, we solve the black box optimization problem without gradient information. Reinforcement learning, by contrast, is a framework for finding a policy to optimize the expected cumulative reward from trial and error. Based on this, the solution to the black box optimization problem can be used as a solution for reinforcement learning.

In this research, we propose a reinforcement learning algorithm based on the mirror descent (MD) method \cite{2}. MD is general optimization algorithm that employs a Bregman
divergence alternative to the Euclidean distance, which is the metric of gradient descent. The derivation for this is detailed in Section 2. We call our proposed method Mirror Descent Search (MDS). In addition, MDS is expected to generalize some existing reinforcement learning algorithms. This research shows (1) that the extension method in the MD can be applied to reinforcement learning, and (2) that the relationship between existing reinforcement learning algorithms can be clarified.

1.1. Related works

In this section, we describe previous research and its relation to the research in this paper.

Relative Entropy Policy Search (REPS) \([\text{?}]\) and its derivation method focuses on information loss during policy searches. This information loss is the relative entropy of the data distribution generated from the observation data distribution. The new policy and is set as the upper limit value. This is equivalent to defining the upper limit value of the Kullback–Libnar (KL) divergence between each distribution. Episode-based REPS \([4]\) is a derivation method formalized by considering the upper-level policy. Although the equations for episode-based REPS and the proposed method are similar, our method can naturally consider distance metrics other than the KL divergence. Consequently, we can apply extension methods in MD.

In \([1, 5]\), the authors focus on the relationship between reinforcement learning and black box optimization. Specifically, \([1]\) explains the history of black box optimization and reinforcement learning, and proposes PI\textsuperscript{BB}. PI\textsuperscript{BB} refers to Policy Improvement with Path Integrals (PI\textsuperscript{2}) \([6, 7]\). It is considered a black box optimization method, where PI\textsuperscript{BB} is derived on the basis of the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) \([8]\), a black box optimization algorithm. Through a comparison with PI\textsuperscript{2}, the authors discuss the connection between reinforcement learning and black box optimization. We further discuss the connection between PI\textsuperscript{2} and the proposed method in the Appendix, below.

Previous studies proposed solutions to reinforcement learning based on MD. Indeed, \([9]\) is strongly associated with our research insofar as the authors propose a method based on MD. However, the details are different. We adopted the exponentiated gradient method (EG) using KL divergence as a regularization term. By contrast, \([9]\) argues that using the Minkowski distance with the Euclidean distance is preferable to KL divergence, because it offers flexibility when updating the gradient.

2. Methods: MDS, G-MDS, AMDS, G-AMDS

2.1. Derivation of proposed algorithm: MDS and G-MDS

2.1.1. MDS

A reinforcement learning algorithm is an algorithm aimed at obtaining an optimal policy to maximize reward (i.e., by minimizing cost). Consider the problem of minimizing the objective function \(J(\theta)\). Rather than dealing with policy parameters \(\theta \in \Theta\) directly, we
consider the probability function \( p(\theta) \). Therefore, we search the following domain:

\[
\mathcal{P} = \left\{ p(\theta) = [p(\theta_1), \ldots, p(\theta_M)] \in \mathbb{R}^M \mid \forall j p(\theta_j) \geq 0, \sum_{j=1}^{M} p(\theta_j) = 1 \right\}.
\]

(1)

The decision variable is \( p(\theta) \), and the objective function is the expectation of the cost \( J(\theta) \).

\[
\mathcal{J} = \sum_{j=1}^{M} p(\theta_j) J(\theta_j)
\]

(2)

Therefore, the optimal generative probability is

\[
p^*(\theta) = \arg \min_{p(\theta) \in \mathcal{P}} \mathcal{J}.
\]

(3)

Next, we consider obtaining the optimal policy by updating \( p(\theta) \). As a means for updating \( p(\theta) \), we use MD, given as follows:

\[
\beta_t = \arg \min_{\beta \in \mathcal{B}} \{ \langle g_t, \beta \rangle + \eta_t B_\phi (\beta || \beta_{t-1}) \}.
\]

(4)

The parameter \( \beta_t \) in (4) is the probability distribution \( p_t(\theta) \) of the policy parameter \( \theta \) at update step \( t \). Thus,

\[
p_t(\theta) := \beta_t
\]

(5)

Substituting the above equation into Eq. (4), we obtain the following:

\[
p_t(\theta) = \arg \min_{p(\theta) \in \mathcal{P}} \{ \langle g_t, p(\theta) \rangle + \eta_t B_\phi (p(\theta) || p_{t-1}(\theta)) \}
\]

(6)

where \( B_\phi \) is the Bregman divergence, which has an arbitrarily smooth convex function \( \phi \) and is defined as \( B_\phi (x, x') = \phi(x) - \phi(x') - \langle \nabla \phi(x'), x - x' \rangle \). The domain of the decision variable is found on the simplex \( \mathcal{P} \). We can select the Bregman divergence as the KL divergence \( \phi(x_t) = \sum_{j=1}^{N} x_{t,j} \log(x_{t,j}) \), but we can also use the Euclidean distances assumed on the simplex [10]. Moreover, we can select a different Bregman divergence, as discussed in [10, 11]. Note that \( g_t \) in Eq. (6) is the gradient of the objective function \( \nabla_{p(\theta)} \mathcal{J} \). We derive this as follows:

\[
\nabla_{p(\theta)} \mathcal{J} = \left[ \frac{\partial \mathcal{J}}{\partial p(\theta_1)}, \ldots, \frac{\partial \mathcal{J}}{\partial p(\theta_j)}, \ldots, \frac{\partial \mathcal{J}}{\partial p(\theta_M)} \right]
\]

\[
= [J(\theta_1), \ldots, J(\theta_j), \ldots, J(\theta_M)]
\]

\[
= J(\theta).
\]

(7)

That is, \( \nabla_{p(\theta)} \mathcal{J} \) is a value obtained without using derivatives of \( J \). From the above, \( p_t(\theta) \) can be updated using Eq. (6) and learning can proceed.
In a typical reinforcement learning problem, we employ the expected cumulative reward—derived with Eq. (8)—as the objective function $J(\theta)$:

$$J(\theta_j) = \int_T p(\tau^\theta_j) r(\tau^\theta_j) \, d\tau$$

where the trajectory generated from the policy parameter $\theta_j$ is $\tau^\theta_j \in T$, the generating probability of the trajectory $\tau^\theta_j$ is $p(\tau^\theta_j)$, and the reward in the trajectory $\tau^\theta_j$ is $r(\tau^\theta_j)$.

We can approximate this using a Monte Carlo integral, as follows:

$$J(\theta_j) \simeq \frac{1}{N} \sum_{i=1}^{N} r(\tau_i^\theta_j)$$

$$\propto \sum_{i=1}^{N} r(\tau_i^\theta_j)$$

In order to solve this problem, we must generate trajectories of $N$ kinds for $M$-type policy parameters, and make $M \times N$ attempts for one update. Here, we use the concept of online learning.

Considering this as online learning, the gradient of the objective function is derived as follows:

$$\nabla_{p(\theta)} J \simeq \left[ r\left(\tau_1^\theta\right), \cdots, r\left(\tau_j^\theta\right), \cdots, r\left(\tau_M^\theta\right) \right] = r\left(\tau_i^\theta\right)$$

where $r\left(\tau_i^\theta\right)$ is a vector of the cumulative reward before calculating the expected value. Thus, $\nabla_{p(\theta)} J \simeq r\left(\tau_i^\theta\right)$ can be used as the gradient of MD $g_t$.

Because this derived algorithm is a policy search based on MD, it is called MDS.

### 2.2. G-MDS

For the experiment, we considered a case where the Bregman divergence $B_\phi$ in Eq. (6) is the KL divergence. That is, in $B_\phi$, $\phi$ is $\phi \left(x_t\right) = \sum_{j=1}^{N} x_{t,j} \log \left(x_{t,j}\right) \left(x \in \mathbb{R}^N, x_{t,j} > 0\right)$. Then, it can be rewritten as follows:

$$p_t \left(\theta_i\right) = \frac{\exp \left(-\eta_t g_{t,i}\right) p_{t-1} \left(\theta_i\right)}{\sum_{j=1}^{N} \exp \left(-\eta_t g_{t,j}\right) p_{t-1} \left(\theta_j\right)}$$

In this paper, we considered $p_t \left(\theta_i\right)$ as the Gaussian distribution of the average $\mu_{t-1}$ and the variance $\Sigma_{t-1,i}$, where $\theta_i$ is generated accordingly:

$$p_t \left(\theta_i\right) = \mathcal{N} \left(\theta \mid \mu_{t-1}, \Sigma_{t-1,i}\right)$$

Here, we consider the average $\mu_t$ of the Gaussian distribution. From Eq. (12), $\mu_t$ can be calculated as follows:

$$\mu_t = \sum_{i=1}^{N} \theta_i p_t \left(\theta_i\right) = \frac{\mathbb{E}_{p_{t-1}} \left[\theta_i \exp \left(-\eta_t g_{t,i}\right)\right]}{\mathbb{E}_{p_{t-1}} \left[\exp \left(-\eta_t g_{t,i}\right)\right]}$$
By using the Monte Carlo integral for Eq. (14), the average value $\mu_t$ can be estimated as $\hat{\mu}_t$ when $N$ is sufficiently large:

$$\hat{\mu}_t = \frac{1}{N} \sum_{i=1}^{N} \theta_i \exp (-\eta t g_{t,i})$$

(15)

Furthermore, from Eq. (13), $\theta_i$ for $\epsilon_{t,i} \sim \mathcal{N}(0, \Sigma_{\epsilon_{t,i}})$ during the update step $t$ can be expressed as follows:

$$\theta_i = \mu_{t-1} + \epsilon_{t-1,i}$$

(16)

Substituting this into Eq. (15), we have the following:

$$\hat{\mu}_t = \frac{1}{N} \sum_{i=1}^{N} (\hat{\mu}_{t-1} + \epsilon_{t-1,i}) \exp (-\eta t g_{t,i})$$

$$= \hat{\mu}_{t-1} + \frac{1}{N} \sum_{i=1}^{N} \left( \exp (-\eta t g_{t,i}) \epsilon_{t-1,i} \right)$$

(17)

Because this derived algorithm is an instance of MDS that assumes that the policy follows a Gaussian distribution, it is called G-MDS.

2.3. Derivation of the proposed algorithm: AMDS and G-AMDS

2.3.1. AMDS

Next, the accelerated mirror descent (AMD) method [11] is applied to the proposed method. AMD is an accelerated method that generalizes Nesterov’s accelerated gradient such that it can be applied to MD. Here, Eq. (13) with AMD yields the following equations:

$$p_t(\theta) = \lambda_{t-1} p_{t-1}^x(\theta) + (1 - \lambda_{t-1}) p_{t-1}^x(\theta), \text{ with } \lambda_{t-1} = \frac{r}{r + t}$$

(18)

$$p_t^\phi(\theta) = \text{arg min}_{p^\phi(\theta) \in P} \left\{ \langle g_t, p^\phi(\theta) \rangle + B_\phi \left( p^\phi(\theta) \| p_{t-1}^\phi(\theta) \right) \right\}$$

(19)

$$p_t^x(\theta) = \text{arg min}_{p^x(\theta) \in P} \left\{ \langle g_t, p^x(\theta) \rangle + R \left( p^x(\theta) \| p_{t-1}^x(\theta) \right) \right\}$$

(20)

where $R(x, x') = B_\omega(x, x')$, which represents the Bregman divergence of the arbitrarily smooth convex function $\omega(x)$.

We here explain the parameters for AMD. First, Eqs. (18)–(20) consist of parallel MDs. Therefore, it seems that $\lambda$ defines the mixture ratio of Eqs. (20) and (19). In addition, $\lambda$ is initially close to 1, such that AMD applies Eq. (20). As $\lambda$ comes close to 0, AMD applies Eq. (19).

Furthermore, consider Eq. (19), where $\frac{ls}{r}$ corresponds to a reciprocal learning rate. This increases as the number of updates increases. Therefore, it can be said that the learning rate decreases as the number of updates increases. This is equivalent to a simulated annealing operation. Moreover, the existing method [6, 7] includes simulated annealing heuristically, yet AMD can include it naturally.

Next, we describe the implementation of AMDS in detail. AMDS proceeds as follows. By repeating the following series of flows, AMD can be treated as reinforcement learning:
1. Sample from the continuous distribution $p_t(\theta)$;
2. Calculate the discrete distribution $\tilde{p}_{t-1}(\theta)$ and $\tilde{p}_{t-1}(\theta)$ from the continuous distribution $p_t(\theta)$, using the obtained samples;
3. Evaluate objective values for each obtained samples as inputs;
4. Calculate the discrete distribution $\tilde{p}_t(\theta)$ and $\tilde{p}_t(\theta)$ based on Eqs. (19) and (20);
5. Perform fitting for the discrete distributions $\tilde{p}_t(\theta)$ and $\tilde{p}_t(\theta)$ as the continuous distribution (e.g., with a Gaussian distribution);
6. Calculate the continuous distribution $p_t(\theta)$ for the next sampling.

![Figure 1: AMD as RL procedures](image)

### 2.3.2. G-AMDS

We derive the same procedure as G-MDS when using KL divergence. Let the Bregman divergence $B_\phi$ from Eq. (19) be the KL distance, and let $R = B_\omega$ in Eq. (20) be $\omega(x) = \varepsilon \sum_{i=1}^{N} (x_i + \varepsilon) \log (x_i + \varepsilon) (x \in \mathbb{R}^N, x_{t,j} > 0)$. Accordingly, this method is referred to as G-AMDS. Furthermore, the result cannot be calculated analytically. Indeed, it is known that an efficient and numerical calculation is available.

Finally, we approximate the distributions $\tilde{p}_t(\theta)$ and $\tilde{p}_t(\theta)$ with a Gaussian distribution.

### 3. Results

#### 3.1. 2DOF Via-point task

We performed a 2DOF Via-point task to evaluate the proposed method. The agent is represented as a point on the x–y plane. This agent learns to pass through the point (0.5,
0.2) at 250 ms. Before learning this, an initial trajectory from (0, 0) to (1, 1) is generated. The reward function is as follows:

\[ r_t = 5000f_t^2 + 0.5\theta^T\theta \]  
\[ \Delta r_{250\text{ms}} = 10000000000 \left((0.5 - x_{250\text{ms}})^2 + (0.2 - y_{250\text{ms}})^2\right) \]  

Here, DMP [12] is used for the parameterization of the policy, and the agent is seeking a policy for each x-axis and y-axis.

The parameter settings are as follows: 1000 updates, 15 rollouts, and 10 basis functions.

3.2. Experimental Results

In this section, we describe the experimental results. We summarize the results for G-MDS and G-AMDS in Fig. Figure 2. In the figure, the thin line represents a standard deviation of 1. Table Table 1 shows the average and the variance at convergence. The variance \( \Sigma_\epsilon \) for each search noise shall be 1.0.

\[ \begin{array}{|c|c|c|} \hline & \text{G-AMDS} & \text{G-MDS} \\ \hline \text{Cost} & 1.15 \times 10^9 \pm 1.69 \times 10^8 & 2.02 \times 10^9 \pm 1.20 \times 10^6 \\ \hline \end{array} \]

From the above, we confirm that G-AMDS learns at a faster rate than G-MDS. Therefore, it is effective to apply the proposed extension for MD to reinforcement learning.
4. Conclusions

In this study, we proposed MDS. We explained the theoretical derivations of MDS, G-MDS, AMDS, and G-AMDS. According to the experimental results, learning progressed faster with the proposed G-AMDS. Moreover, based on the fact that AMD is a generalization of Nesterov’s acceleration method, we expect that the acceleration will be effective for an objective function with a saddle point.

Acknowledgment

The research was supported by JSPS KAKENHI (Grant numbers JP26120005, JP16H03219, and JP17K12737).

Appendix A. Relationship between G-MDS and PI² algorithm

We here demonstrate that the algorithm is equivalent to PI². The symbols are replaced as follows:

\[ P_{t-1,i} = \frac{\exp(-\eta_t g_{t,i})}{\sum_{j=1}^{N} \exp(-\eta_t g_{t,j})} \]  
\[ M_{t-1,i} = 1 \]

That is, Eq. (17) becomes similar to Eq. (A.3):

\[ \hat{\mu}_t = \hat{\mu}_{t-1} + \sum_{i=1}^{N} (P_{t-1,i} M_{t-1,i} \epsilon_{t-1,i}) \]

The PI² algorithm is equivalent to Eq. (A.3), provided that we employ DMP \[12\] as the policy function and ignore \( M_{t-1,i} \). The algorithm here can be written such that it resembles that in Table Table A.2. There are some differences between PI² and the algorithms obtained here, however. For instance, with PI², the decision variable is updated sequentially using the provisional cumulative reward at each point during one trial, whereas the G-MDS uses only cumulative rewards. Moreover, G-MDS is an algorithm with fewer procedures. Finally, PI² assumes a Gaussian distribution, whereas G-MDS can be generalized as MDS, which calculate similar algorithms using arbitrary probability distributions.

References

[1] F. Stulp, O. Sigaud, Policy improvement methods: Between black-box optimization and episodic reinforcement learning, 34 pages (Oct. 2012).
[2] S. Bubeck, Convex Optimization: Algorithms and Complexity, ArXiv e-printsarXiv:1405.4980.
[3] J. Peters, K. Mulling, Y. Altun, Relative entropy policy search., in: AAAI, Atlanta, 2010, pp. 1607–1612.
[4] C. Daniel, G. Neumann, J. R. Peters, Hierarchical relative entropy policy search, in: International Conference on Artificial Intelligence and Statistics, 2012, pp. 273–281.
[5] J. Hwangbo, C. Gehring, H. Sommer, R. Siegwart, J. Buchli, ROCK*: Efficient black-box optimization for policy learning, in: 2014 IEEE-RAS International Conference on Humanoid Robots, IEEE, 2014, pp. 535–540.

[6] E. Theodorou, J. Buchli, S. Schaal, A generalized path integral control approach to reinforcement learning, Journal of Machine Learning Research 11 (Nov) (2010) 3137–3181.

[7] E. Theodorou, J. Buchli, S. Schaal, Reinforcement learning of motor skills in high dimensions: A path integral approach, in: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE, 2010, pp. 2397–2403.

[8] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary computation 9 (2) (2001) 159–195.

[9] S. Mahadevan, B. Liu, Sparse q-learning with mirror descent, arXiv preprint arXiv:1210.4893.

[10] W. Krichene, S. Krichene, A. Bayen, Efficient bregman projections onto the simplex, in: Decision and Control (CDC), 2015 IEEE 54th Annual Conference on, IEEE, 2015, pp. 3291–3298.

[11] W. Krichene, A. Bayen, P. L. Bartlett, Accelerated mirror descent in continuous and discrete time, in: Advances in Neural Information Processing Systems, 2015, pp. 2845–2853.

[12] A. J. Ijspeert, J. Nakanishi, S. Schaal, Learning attractor landscapes for learning motor primitives, Tech. rep. (2002).
Prerequisites:

- Time \( t_0 \ldots t_N \)
- Semi-definite matrix \( R \)
- Immediate cost function \( r_t = q_t + \theta^T R \theta_t \)
- Termination cost term \( \phi_t \)
- Probabilistic policy parameters \( a_t = g_t^T (\theta + \epsilon) \)
- Basic function of system dynamics \( g_t \)
- Distributed \( \Sigma_t \), average 0, noise \( \epsilon \)
- Initial value of parameter vector \( \theta \)

Repeat until cost \( R \) converges

- **Step 1**: Initial state \( x_0 \): generate \( K \) rollouts from \( (\theta + \epsilon_t) \)
- **Step 2**: Rollout \( k = 1 \ldots K \), calculated as follows:
  * **Step 2.1**: Calculate the cost \( S(\tau_k) \) for each trajectory:
    \[
    S(\tau_k) = \phi_{t_N,k} + \sum_{j=0}^{N-1} q_{j,k} + \frac{1}{2} \sum_{j=0}^{N-1} (\theta + \epsilon_k)^T R (\theta + \epsilon_k)
    \]
  * **Step 2.2**: Calculate the probability \( P(\tau_k) \) for each trajectory:
    \[
    P(\tau_k) = \frac{e^{-\frac{1}{2} S(\tau_k)}}{\sum_{k=1}^{K} e^{-\frac{1}{2} S(\tau_k)}}
    \]
- **Step 3**: Calculate the change to the policy parameters:
  \[
  \delta \theta = \sum_{k=1}^{K} [P(\tau_k) \epsilon_k]
  \]
- **Step 4**: Update policy parameters: \( \theta \leftarrow \theta + \delta \theta \)
- **Step 5**: Generate one rollout, excluding noise, and check the cost:
  \[
  R = \phi_{t_N} + \sum_{i=0}^{N-1} r_i
  \]