International Journal of Innovative Computing, Information and Control
Volume **12**, Number **3**, June **2016**

 ISSN 1349-4198
pp. 941–957

ACTOR-CRITIC ALGORITHMS WITH ϵ-GREEDY GAUSSIAN POLICY IN MULTIDIMENSIONAL CONTINUOUS ACTION SPACES

Chunyuan Zhang[1,2], Qingxin Zhu[1], Yigui Ou[2] and Xinzheng Niu[1]

[1]School of Computer Science and Engineering
University of Electronic Science and Technology of China
No. 2006, Xiyuan Ave., West Hi-Tech Zone, Chengdu 611731, P. R. China
zcy7566@126.com; {qxzhu; xinzhengniu}@uestc.edu.cn

[2]College of Information Science and Technology
Hainan University
No. 58, Renmin Ave., Meilan District, Haikou 570228, P. R. China
ouyigui@126.com

Received December 2015; revised March 2016

Abstract. *In actor-critic (AC) algorithms, the Gaussian policy is widely used for solving the sequential decision problems with continuous action spaces. However, this policy has a tendency of over-exploration due to lack of greediness, which often makes AC algorithms difficult to obtain good convergence speed and quality. In this paper, we propose a novel ϵ-greedy Gaussian policy, and present two compatible AC algorithm frameworks for successfully using it. The proposed policy can be viewed as a hybrid of the traditional Gaussian policy and the ϵ-greedy policy. At each time step, it generates some candidate actions by performing symmetric Gaussian perturbations on the current action mean, and then uses the ϵ-greedy policy to select the behaviour action based on advantage functions. To the best of our knowledge, this is the first time to introduce the ϵ-greedy policy into AC algorithms for solving multidimensional continuous action problems. Theoretical analysis shows that compatible AC algorithms can obtain better convergence quality with the proposed policy than with the traditional Gaussian policy. Finally, experimental results on a mountain-car problem and a puddle-world problem demonstrate the effectiveness of the proposed policy and compatible AC algorithm frameworks.*
Keywords: Actor-critic, Gaussian policy, Continuous space, Compatible function approximation, Policy gradient, Reinforcement learning

1. **Introduction.** In the real world, many sequential decision problems have continuous state and action spaces. Currently, there are three main categories of reinforcement learning (RL) methods for solving such problems, that is, actor-only, critic-only and actor-critic (AC) algorithms [1]. Typically, actor-only algorithms work with a parameterized family of policies over which the optimal policy can be searched directly, critic-only algorithms discretize the continuous action space and work with state-action value functions over which the optimal policy can be derived indirectly, and AC algorithms can be viewed as the combination of actor-only and critic-only algorithms. In general, AC algorithms have two separate memory structures, namely, an actor for policy improvement and a critic for policy evaluation [2]. This property makes AC algorithms easy to combine the strong points of actor-only and critic-only algorithms [3]. By introducing the state value function to reduce the variance of policy gradient, AC algorithms have better convergence performance than actor-only algorithms [4]. By representing and improving the policy

explicitly, AC algorithms have more reliable convergence guarantees [3, 4] and higher control precision [5] than critic-only algorithms. Therefore, AC algorithms have become a preferred RL method for continuous action problems.

However, AC algorithms often converge slower and worse than critic-only algorithms in many cases [6, 7]. In recent years, some studies have attempted to overcome this problem. Melo and Lopes proposed a fitted natural AC algorithm by using importance sampling [8], but it is only suitable for batch learning and the implementation is complicated. Wawrzynski and Tanwani proposed two sequential AC algorithms with experience replay [9, 10], whereas they require a large amount of memory to cache history samples. Hasselt and Wiering proposed a continuous AC learning automaton algorithm by using the action error and the sign of the temporal-difference (TD) error for policy update [11, 12]. However, it lacks reliable guarantees in terms of near-optimality of the resulting policy, and it may perform badly in problems with specific types of noise [13]. Vien and Ertel proposed a compatible AC TAMER framework [14], which requires human feedback and thus deviates from the original intention of RL. Lazaric et al. proposed an AC algorithm in which the policy is estimated by sequential Monte Carlo methods [5]. Unfortunately, it is built on a discrete state space, which narrows the applicability in practice. Kimura and Kobayashi proposed an AC algorithm using a binary tree action selector [15], but it does not truly offer continuous actions since the selector is actually a variant of the Boltzmann policy.

Different from the above studies, we argue that the slow convergence of AC algorithms is mainly caused by the inherent limitations of their exploration policies. For continuous action problems, AC algorithms widely use a parameterized normal distribution to represent the exploration policy [16, 17, 18, 19, 20, 21], which is known as the Gaussian policy. In order to distinguish it from our policy proposed in Section 3.3, we call it the traditional Gaussian policy in this paper. At each time step, the traditional Gaussian policy only generates one action by performing a Gaussian perturbation on the current action mean, and has no choice but to use it as the behaviour action. Although we can adaptively adjust the perturbation size by using the state-dependent exploration [22], the Gaussian perturbation still has strong randomness. By contrast, critic-only algorithms widely use an ϵ-greedy policy as the exploration policy, which greedily selects the behaviour action with probability $1-\epsilon$. In an ϵ-greedy policy, ϵ denotes the exploration strength and is often small. Thus, compared with an ϵ-greedy policy, the traditional Gaussian policy obviously lacks greediness and has a tendency of over-exploration. In RL, it is well known that over-exploration may result in sample inefficiency [2]. This also explains why AC algorithms often converge slower than critic-only algorithms.

Intuitively, we may improve the convergence performance of AC algorithms by increasing the greediness of the traditional Gaussian policy. However, the traditional Gaussian policy lacks the ability to select the greedy action, and a full search in a continuous action space to find the optimal action is often unfeasible [5, 23]. To deal with these two problems, we proposed a novel AC algorithm framework with symmetric perturbation sampling for one-dimensional continuous action problems in our previous work [24]. At each time step, this framework generates two candidate actions by performing two symmetric Gaussian perturbations on the current action mean, and takes them to interact with the environment in parallel. Then, it selects the behaviour action and updates the value-function parameters based on their maximum TD error, and updates the traditional Gaussian policy parameters based on their average regular gradient or natural gradient. Theoretical analysis and experimental results show that this framework has reliable and fast convergence performance. Nevertheless, it requires two interactions per time step,

which undermines the practical applicability seriously. If we extend it for n-dimensional continuous action problems, it will require 2^n interactions per time step.

In this paper, we further investigate how to increase the greediness of the traditional Gaussian policy without multi-interactions. The main contributions of this paper include three aspects. (i) We propose a novel ϵ-greedy Gaussian policy, which can be viewed as a hybrid of the traditional Gaussian policy and the ϵ-greedy policy. (ii) For successfully using this policy, we present two compatible AC algorithm frameworks called RLSEGAC and SEGAC. Compared with our previous framework, they are mainly built on compatible function approximation rather than value function approximation. In particular, they only require one interaction per time step, even for n-dimensional continuous action problems. (iii) We prove that compatible AC algorithms can obtain better convergence quality with this policy than with the traditional Gaussian policy in theory. The rest of this paper is organized as follows. In Section 2, some background materials are provided. In Section 3, the traditional Gaussian policy and the ϵ-greedy policy are reviewed, and then the ϵ-greedy Gaussian policy is proposed. In Section 4, two compatible AC algorithm frameworks are presented for using the proposed policy. In Sections 5 and 6, the performance of compatible AC algorithms with the proposed policy is analyzed theoretically and demonstrated experimentally. Finally, conclusions are summarized in Section 7.

2. **Background.**

In this section, we introduce the basic definitions, assumptions and notations, which will be used throughout the paper without any further mention. We also review the policy gradient theorem and conventional AC algorithms, which are needed to establish our main results described in Sections 3-5.

2.1. **Preliminaries.**

In this paper, we assume that the sequential decision problem with a continuous state space $\mathcal{S} \in \mathbb{R}^m$ and a continuous action space $\mathcal{A} \in \mathbb{R}^n$ can be modeled as a discrete-time Markov decision process [16]. For any state $\boldsymbol{s}_t \in \mathcal{S}$ at time step $t \in \mathbb{N}$, the agent takes an action $\boldsymbol{a}_t \in \mathcal{A}$ drawn from a stochastic parameterized policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t) = p(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{\theta})$ with a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^N$, then transfers to a successor state $\boldsymbol{s}_{t+1}$ drawn from the state transition distribution $\tau(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, and earns a reward $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \in \mathbb{R}$. In addition, we assume that $\pi_{\boldsymbol{\theta}}$ is continuously differentiable w.r.t. $\boldsymbol{\theta}$, and use π to denote $\pi_{\boldsymbol{\theta}}$ for simplicity of notation.

Under the above assumptions, the goal of RL is to learn an optimal $\boldsymbol{\theta}^*$ for maximizing the long-run expected return J^π. Depending on the nature of the problem, there are two formulations for defining J^π [21, 25]. One is the discount-return formulation, i.e., $J^\pi = \mathbf{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t|\boldsymbol{s}_0]$, where $\gamma \in [0, 1]$ is the discount factor and $\boldsymbol{s}_0$ is an initial state drawn from the start-state distribution $p(\boldsymbol{s}_0)$. The other is the average-return formulation, i.e., $J^\pi = \lim_{l\to\infty} \frac{1}{l}\mathbf{E}_\pi \left[\sum_{t=0}^{l-1} r_t\right]$. Both formulations can be unifiedly described as

$$J^\pi = \int_{\mathcal{S}} d^\pi(\boldsymbol{s}) \int_{\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) R(\boldsymbol{s}, \boldsymbol{a}) \mathrm{d}\boldsymbol{a}\mathrm{d}\boldsymbol{s} \tag{1}$$

where $R(\boldsymbol{s}, \boldsymbol{a}) = \mathbf{E}_\tau[r_t|\boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}]$ is the expected reward in $(\boldsymbol{s}, \boldsymbol{a})$, and $d^\pi(\boldsymbol{s})$ is the state distribution. For the discount-return formulation, $d^\pi(\boldsymbol{s}) = \sum_{t=0}^{\infty} \gamma^t P(\boldsymbol{s}_t = \boldsymbol{s}|\boldsymbol{s}_0, \pi)$, whereas, for the average-return formulation, $d^\pi(\boldsymbol{s}) = \lim_{t\to\infty} P(\boldsymbol{s}_t = \boldsymbol{s}|\boldsymbol{s}_0, \pi)$.

Besides using J^π as the performance metric of π, many RL algorithms also use the state value function $V^\pi(\boldsymbol{s})$ or the state-action value function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ as the performance metric of π in $\boldsymbol{s}$ or $(\boldsymbol{s}, \boldsymbol{a})$. Similar to J^π, $V^\pi(\boldsymbol{s})$ and $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ can be defined as follows

$$V^\pi(\boldsymbol{s}) = \mathbf{E}_\pi \left[\sum_{t=0}^{\infty} \left(\gamma^t r_t - \bar{r}_\pi\right) |\boldsymbol{s}_0 = \boldsymbol{s}\right] \tag{2}$$

$$Q^{\pi}(\boldsymbol{s},\boldsymbol{a})=\mathbf{E}_{\pi}\left[\sum_{t=0}^{\infty}\left(\gamma^{t}r_{t}-\bar{r}_{\pi}\right)|\boldsymbol{s}_{0}=\boldsymbol{s},\boldsymbol{a}_{0}=\boldsymbol{a}\right] \tag{3}$$

where, for the discount-return formulation, $\bar{r}_{\pi}=0$, whereas, for the average-return formulation, $\bar{r}_{\pi}=J^{\pi}$ and $\gamma=1$.

2.2. **Policy gradient theorem.** In general, $\boldsymbol{\theta}^{*}$ cannot be solved by maximizing J^{π} directly. Instead, we often resort to policy gradient algorithms, in which $\boldsymbol{\theta}$ is updated by

$$\boldsymbol{\theta}_{t+1}=\boldsymbol{\theta}_{t}+\alpha_{t}^{\theta}\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}}J^{\pi} \tag{4}$$

where α_{t}^{θ} is the step size, and $\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}}J^{\pi}$ is the policy gradient.

$\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}}J^{\pi}$ has two main types, namely, the regular gradient $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}$ and the natural gradient $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}}J^{\pi}$. By the policy gradient theorem [25], $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}$ can be defined as

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}=\int_{\mathcal{S}}d^{\pi}(\boldsymbol{s})\int_{\mathcal{A}}Q^{\pi}(\boldsymbol{s},\boldsymbol{a})\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{a}|\boldsymbol{s})\mathrm{d}\boldsymbol{a}\mathrm{d}\boldsymbol{s}=\mathbf{E}_{\pi}\left[Q^{\pi}(\boldsymbol{s},\boldsymbol{a})\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s})\right] \tag{5}$$

Since $\int_{\mathcal{A}}\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{a}|\boldsymbol{s})\mathrm{d}\boldsymbol{a}=0$, we can introduce an arbitrary baseline $b(\boldsymbol{s})$ into $Q^{\pi}(\boldsymbol{s},\boldsymbol{a})$ without affecting the unbiasedness of the gradient estimate [4, 16]. Let $A^{\pi}(\boldsymbol{s},\boldsymbol{a})=Q^{\pi}(\boldsymbol{s},\boldsymbol{a})-b(\boldsymbol{s})$. Then, Equation (5) can be rewritten as

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}=\mathbf{E}_{\pi}\left[A^{\pi}(\boldsymbol{s},\boldsymbol{a})\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s})\right] \tag{6}$$

By linearly transforming $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}$, $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}}J^{\pi}$ can be defined as [4, 16]

$$\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}}J^{\pi}=\boldsymbol{F}^{-1}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi} \tag{7}$$

where $\boldsymbol{F}(\boldsymbol{\theta})=\mathbf{E}_{\pi}\left[\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s})\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s})^{\mathsf{T}}\right]$ is the Fisher information matrix.

In the implementation of the policy gradient theorem, there are two ways to deal with $A^{\pi}(\boldsymbol{s},\boldsymbol{a})$ for estimating $\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}}J^{\pi}$. The first way is to approximate $A^{\pi}(\boldsymbol{s},\boldsymbol{a})$ with the compatible function [4, 16], and is mainly used for estimating $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}}J^{\pi}$. Let $\hat{A}(\boldsymbol{s},\boldsymbol{a})=\hat{Q}(\boldsymbol{s},\boldsymbol{a})-\hat{b}(\boldsymbol{s})=\boldsymbol{w}^{\mathsf{T}}\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s})$, where $\hat{A}(\boldsymbol{s},\boldsymbol{a})$, $\hat{Q}(\boldsymbol{s},\boldsymbol{a})$ and $\hat{b}(\boldsymbol{s})$ denote the approximations of $A^{\pi}(\boldsymbol{s},\boldsymbol{a})$, $Q^{\pi}(\boldsymbol{s},\boldsymbol{a})$ and $b(\boldsymbol{s})$, $\boldsymbol{w}\in\mathbb{R}^{N}$ denotes the compatible parameter vector, and $\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s})$ is called the compatible feature vector. Since $\int_{\mathcal{A}}\pi(\boldsymbol{a}|\boldsymbol{s})\hat{A}(\boldsymbol{s},\boldsymbol{a})\mathrm{d}\boldsymbol{a}=\boldsymbol{w}^{\mathsf{T}}\int_{\mathcal{A}}\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{a}|\boldsymbol{s})\mathrm{d}\boldsymbol{a}=0$, we have $\int_{\mathcal{A}}\pi(\boldsymbol{a}|\boldsymbol{s})\hat{Q}(\boldsymbol{s},\boldsymbol{a})\mathrm{d}\boldsymbol{a}=\int_{\mathcal{A}}\pi(\boldsymbol{a}|\boldsymbol{s})\hat{b}(\boldsymbol{s})\mathrm{d}\boldsymbol{a}=\hat{b}(\boldsymbol{s})$. Furtherly, from Equations (2) and (3), we have $\int_{\mathcal{A}}\pi(\boldsymbol{a}|\boldsymbol{s})Q^{\pi}(\boldsymbol{s},\boldsymbol{a})\mathrm{d}\boldsymbol{a}=V^{\pi}(\boldsymbol{s})$. Thus, $\hat{b}(\boldsymbol{s})$ is the approximation of $V^{\pi}(\boldsymbol{s})$ [26], and we can assume $b(\boldsymbol{s})=V^{\pi}(\boldsymbol{s})$. In this paper, $A^{\pi}(\boldsymbol{s},\boldsymbol{a})=Q^{\pi}(\boldsymbol{s},\boldsymbol{a})-V^{\pi}(\boldsymbol{s})$ is the advantage function, which can be approximated by

$$\hat{A}(\boldsymbol{s},\boldsymbol{a})=\hat{Q}(\boldsymbol{s},\boldsymbol{a})-\hat{V}(\boldsymbol{s})=\boldsymbol{w}^{\mathsf{T}}\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s}) \tag{8}$$

where $\hat{V}(\boldsymbol{s})$ denotes the approximation of $V^{\pi}(\boldsymbol{s})$. Then, $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}}J^{\pi}$ can be estimated as

$$\hat{\nabla}_{\boldsymbol{\theta}}^{\mathrm{NG}}J^{\pi}=\boldsymbol{w} \tag{9}$$

The second way is to replace $A^{\pi}(\boldsymbol{s},\boldsymbol{a})$ with some form of TD errors [4, 21], and thus the estimate of $\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}}J^{\pi}$ is associated with $V^{\pi}(\boldsymbol{s})$ and $\bar{r}_{\pi}$ rather than $A^{\pi}(\boldsymbol{s},\boldsymbol{a})$. Different from the first way, this way is mainly used for estimating $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}$.

2.3. **Conventional AC algorithms.** As stated in Section 1, an AC algorithm often has two separate memory structures, which make it easy to implement the policy gradient theorem. According to the gradient type, AC algorithms can be divided into regular-gradient AC (RAC) algorithms and natural-gradient AC (NAC) algorithms.

In conventional RAC algorithms, the actor often uses the second way to estimate $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}}J^{\pi}$ for updating $\boldsymbol{\theta}$, and the critic often uses linear TD(λ) algorithms [27, 28] for learning $V^{\pi}(\boldsymbol{s})$ and $\bar{r}_{\pi}$. Define the one-step TD error as $\delta_{t}^{V}=r_{t}-\bar{r}_{t+1}+\gamma\hat{V}(\boldsymbol{s}_{t+1})-\hat{V}(\boldsymbol{s}_{t})$, where $\bar{r}_{t+1}$

denotes the estimate of $\bar{r}_\pi$ at time step t. Let $\hat{V}(\boldsymbol{s}) = \boldsymbol{v}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{s})$, where $\boldsymbol{v} \in \mathbb{R}^M$ denotes the parameter vector of $\hat{V}(\boldsymbol{s})$, and $\boldsymbol{\phi}(\boldsymbol{s}) \in \mathbb{R}^M$ denotes the feature vector of $\hat{V}(\boldsymbol{s})$. Then, δ_t^V can be estimated as

$$\delta_t^V = r_t - \bar{r}_{t+1} + \boldsymbol{v}_t^\mathsf{T}\left(\gamma\boldsymbol{\phi}(\boldsymbol{s}_{t+1}) - \boldsymbol{\phi}(\boldsymbol{s}_t)\right) \tag{10}$$

All update rules in the critic are [21]

$$\bar{r}_{t+1} = (1 - \alpha_t^{\bar{r}})\,\bar{r}_t + \alpha_t^{\bar{r}} r_t \tag{11}$$

$$\boldsymbol{z}_{t+1}^v = \gamma\lambda\boldsymbol{z}_t^v + \boldsymbol{\phi}(\boldsymbol{s}_t) \tag{12}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \alpha_t^v\delta_t^V\boldsymbol{z}_{t+1}^v \tag{13}$$

where $\lambda \in [0,1]$, $\boldsymbol{z}_{t+1}^v \in \mathbb{R}^M$, $\alpha_t^{\bar{r}}$ and α_t^v are the trace-decay parameter, the trace of $\boldsymbol{\phi}(\boldsymbol{s}_t)$ and the step sizes, respectively. Let $\boldsymbol{z}_{t+1}^\theta$ denote the trace of $\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$, i.e.,

$$\boldsymbol{z}_{t+1}^\theta = \gamma\lambda\boldsymbol{z}_t^\theta + \nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \tag{14}$$

Referring to A.2 in [18], we can verify that $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}} J^\pi$ can be approximated by

$$\hat{\nabla}_{\boldsymbol{\theta}}^{\mathrm{RG}} J^\pi = \delta_t^V\boldsymbol{z}_{t+1}^\theta \tag{15}$$

Thus, the update rule in the actor is [21]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t^\theta\delta_t^V\boldsymbol{z}_{t+1}^\theta \tag{16}$$

In conventional NAC algorithms, the actor often uses the first way to estimate $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J^\pi$ for updating $\boldsymbol{\theta}$, and the critic often uses Equation (8) to approximate $A^\pi(\boldsymbol{s},\boldsymbol{a})$. However, $A^\pi(\boldsymbol{s},\boldsymbol{a})$ cannot be learned with TD-like bootstrapping exclusively. To remedy this situation, Peters and Schaal proposed a least-squares NAC (LSNAC) algorithm [16], in which the critic uses LSTD-Q(λ) for learning $A^\pi(\boldsymbol{s},\boldsymbol{a})$. However, this algorithm has $O((M+N)^3)$ computational complexity per time step and only considers the discount-return formulation. On this basis, Wawrzynski proposed a sequential NAC (SNAC) algorithm [9], and Degris et al. proposed an incremental NAC (INAC) algorithm [21]. Compared with SNAC, INAC considers both return formulations defined in Section 2.1. Here we only review INAC briefly. The other two algorithms will be left in Section 4 for further discussion. In the critic of INAC, the update rule for $\boldsymbol{w}$ is

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t^v\left(\delta_t^V\boldsymbol{z}_{t+1}^\theta - \boldsymbol{F}_t\boldsymbol{w}_t\right) \tag{17}$$

where $\boldsymbol{F}_t = \nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)^\mathsf{T}$, and the estimate of δ_t^V is the same as Equation (10). The other update rules for $\bar{r}_t$, $\boldsymbol{z}_t^v$ and $\boldsymbol{v}_t$ are the same as Equations (11)-(13). In the actor of INAC, the update rule is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t^\theta\boldsymbol{w}_{t+1} \tag{18}$$

3. **Exploration Polices.** In this section, we review the traditional Gaussian policy and the ϵ-greedy policy, and then propose the ϵ-greedy Gaussian policy by combining them.

3.1. **Traditional Gaussian policy.** The traditional Gaussian policy is a parameterized normal distribution, which is widely used in AC algorithms to explore the continuous action space. We assume that all components of the continuous action are independent of each other. Then, an n-dimensional traditional Gaussian policy can be defined as [22]

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \prod_{i=1}^{n}\mathcal{N}\left(\mu_i\left(\boldsymbol{s},\boldsymbol{\theta}_i^\mu\right),\sigma_i^2\left(\boldsymbol{s},\boldsymbol{\theta}_i^\sigma\right)\right) \tag{19}$$

where $\mu_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\mu})$ and $\sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})$ denote the mean and standard deviation of the i-th action component a_i in $\boldsymbol{s}$, and they are parameterized by $\boldsymbol{\theta}_i^{\mu}$ and $\boldsymbol{\theta}_i^{\sigma}$ respectively. From the above equation, a_i can be generated as

$$a_i = \mu_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\mu}) + \rho_i \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma}) \tag{20}$$

where $\rho_i \sim \mathcal{N}(0,1)$, and $\rho_i \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})$ is called the Gaussian perturbation. Correspondingly, the compatible feature vector can be calculated as

$$\begin{cases} \nabla_{\boldsymbol{\theta}_i^{\mu}} \log \pi(\boldsymbol{a}|\boldsymbol{s}) = \dfrac{a_i \mu_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\mu})}{\sigma_i^2(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})} \nabla_{\boldsymbol{\theta}_i^{\mu}} \mu_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\mu}) \\ \nabla_{\boldsymbol{\theta}_i^{\sigma}} \log \pi(\boldsymbol{a}|\boldsymbol{s}) = \dfrac{\rho_i^2 - 1}{\sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})} \nabla_{\boldsymbol{\theta}_i^{\sigma}} \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma}) \end{cases} \tag{21}$$

From Equations (20), (21), (6) and (7), $\rho_i \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})$, a_i and $\nabla_{\boldsymbol{\theta}}^{\mathrm{PG}} J^{\pi}$ are associated with ρ_i. Obviously, they have strong randomness since ρ_i is drawn from $\mathcal{N}(0,1)$. This may result in over-exploration and sample inefficiency. In Section 1 and our previous work [24], we have analyzed the inherent limitations of the traditional Gaussian policy in detail, so we will not repeat them here.

3.2. **ϵ-greedy policy.** In RL, the ϵ-greedy policy may be the most popular exploration policy, which is widely used in critic-only algorithms to explore the discrete action space. Let $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a} \in \mathcal{A}(\boldsymbol{s})} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$, where $\mathcal{A}(\boldsymbol{s})$ is the discrete action set in $\boldsymbol{s}$. Then, an ϵ-greedy policy can be defined as [2]

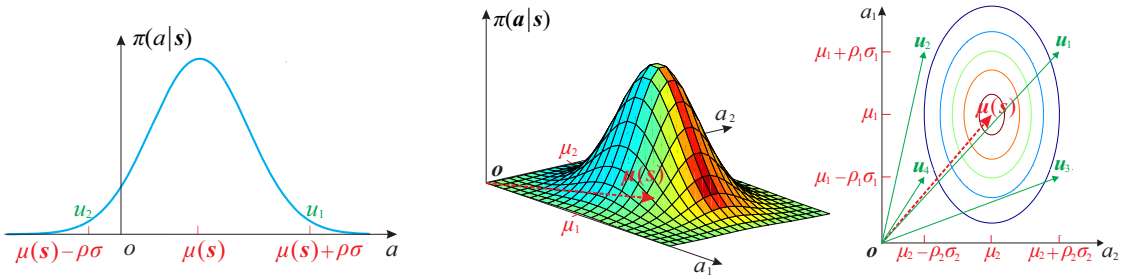
$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(\boldsymbol{s})|} & \text{if } \boldsymbol{a} = \boldsymbol{a}^* \\ \dfrac{\epsilon}{|\mathcal{A}(\boldsymbol{s})|} & \text{if } \boldsymbol{a} \neq \boldsymbol{a}^* \end{cases} \tag{22}$$

where $\epsilon \in (0, 1]$ is the exploration factor and denotes the exploration strength.

The ϵ-greedy policy has some good properties. For any ϵ-soft policy ς, any ϵ-greedy policy with respect to Q^{ς} is guaranteed to be better than or equal to ς [2]. In addition, by selecting a moderate ϵ, the ϵ-greedy policy can balance exploration and exploitation well. However, the ϵ-greedy policy is unsuitable for continuous action problems, since it requires a global maximization at each time step [5, 23].

3.3. **ϵ-greedy Gaussian policy.** Since the normal distribution is a symmetric distribution, Equation (20) can also be rewritten as $a_i = \mu_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\mu}) - \rho_i \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})$. In this paper and our previous work, $\rho_i \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})$ and $-\rho_i \sigma_i(\boldsymbol{s}, \boldsymbol{\theta}_i^{\sigma})$ are called symmetric Gaussian perturbations. Thus, an n-dimensional traditional Gaussian policy can be equivalently implemented as follows. We first generate 2^n candidate actions by using symmetric Gaussian perturbations as shown in Figure 1, and then randomly select a candidate as the behaviour action. Intuitively, if we can greedily select the behaviour action from candidates, we will reduce the randomness of the traditional Gaussian policy. In our previous work [24], we proposed an AC algorithm framework following this idea, and demonstrated its effectiveness in the one-dimensional action space. However, as mentioned in Section 1, this framework requires 2^n interactions per time step for n-dimensional continuous action problems, since it selects the behaviour action based on TD errors. From Equation (8), we have $\arg\max_{\boldsymbol{a} \in \mathcal{A}} \hat{Q}(\boldsymbol{s}, \boldsymbol{a}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} \hat{A}(\boldsymbol{s}, \boldsymbol{a})$. If we use advantage functions to replace TD errors for selecting the behaviour action, we will avoid this situation.

Based on the above analysis, we propose a novel exploration policy, called the ϵ-greedy Gaussian policy, for exploring the n-dimensional continuous action space. It can be viewed



(a) 1-dimensional symmetric perturbations (b) 2-dimensional symmetric perturbations

FIGURE 1. 1-dimensional and 2-dimensional symmetric Gaussian perturbations

as a hybrid of the traditional Gaussian policy and the ϵ-greedy policy. Firstly, by performing symmetric Gaussian perturbations on the current action mean, it generates a candidate action set $\mathcal{A}(\boldsymbol{s}) = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{2^n}\}$, where the j-th candidate action is defined as

$$\boldsymbol{u}_j = \boldsymbol{\mu}(\boldsymbol{s}) + \left[(-1)^{\lfloor \frac{j-1}{2^{n-1}} \rfloor}, \cdots, (-1)^{\lfloor \frac{j-1}{2^{n-n}} \rfloor}\right]^{\mathsf{T}} \circ \boldsymbol{\rho} \circ \boldsymbol{\sigma}(\boldsymbol{s}) \tag{23}$$

where $\boldsymbol{\mu}(\boldsymbol{s}) = [\mu_1(\boldsymbol{s}, \boldsymbol{\theta}_1^{\mu}), \cdots, \mu_n(\boldsymbol{s}, \boldsymbol{\theta}_n^{\mu})]^{\mathsf{T}}$ and $\boldsymbol{\sigma}(\boldsymbol{s}) = [\sigma_1(\boldsymbol{s}, \boldsymbol{\theta}_1^{\sigma}), \cdots, \sigma_n(\boldsymbol{s}, \boldsymbol{\theta}_n^{\sigma})]^{\mathsf{T}}$ denote the mean and standard deviation of the current action, $\boldsymbol{\rho}$ is drawn from the n-dimensional standard normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, and $\circ$ denotes the Hadamard product. Secondly, it selects the behaviour action $\boldsymbol{a}$ from $\mathcal{A}(\boldsymbol{s})$ with the ϵ-greedy policy, where $\boldsymbol{a}^* = \arg\max_{\boldsymbol{u}_j \in \mathcal{A}(\boldsymbol{s})} \hat{A}(\boldsymbol{s}, \boldsymbol{u}_j)$. The complete algorithm of this policy is shown in Algorithm 1.

Algorithm 1 ϵ-greedy Gaussian policy

1: **Input:** ϵ, $\boldsymbol{s}$, $\boldsymbol{w}$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}(\boldsymbol{s})$, $\pi(\boldsymbol{u}|\boldsymbol{s})$, $\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{u}|\boldsymbol{s})$
2: **Initialize:** $\mathcal{A}(\boldsymbol{s}) = \{\}$
3: Approximate $\boldsymbol{\mu}(\boldsymbol{s})$ and $\boldsymbol{\sigma}(\boldsymbol{s})$ with $\boldsymbol{\theta}$ and $\boldsymbol{\phi}(\boldsymbol{s})$
4: Draw $\boldsymbol{\rho} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
5: **for** $j = 1, 2, \cdots, 2^n$ **do**
6: Generate candidate action $\boldsymbol{u}_j$ by Equation (23)
7: $\hat{A}(\boldsymbol{s}, \boldsymbol{u}_j) = \boldsymbol{w}^{\mathsf{T}} \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{u}_j|\boldsymbol{s})$
8: $\mathcal{A}(\boldsymbol{s}) = \mathcal{A}(\boldsymbol{s}) \cup \{\boldsymbol{u}_j\}$
9: **end for**
10: Select behaviour action $\boldsymbol{a}$ from $\mathcal{A}(\boldsymbol{s})$ with the ϵ-greedy policy

For the sake of discussion, let ϑ, κ and π denote the traditional Gaussian policy, the ϵ-greedy policy and the ϵ-greedy Gaussian policy in the rest of paper. Based on the above description, there are 2^n candidate actions in $\boldsymbol{s}$, and their probability densities are $\vartheta(\boldsymbol{a}|\boldsymbol{s})$. Thus, $\pi(\boldsymbol{a}|\boldsymbol{s})$ can be represented as

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = 2^n \vartheta(\boldsymbol{a}|\boldsymbol{s}) \kappa(\boldsymbol{a}|\boldsymbol{s}) \tag{24}$$

From Equations (23) and (24), the ϵ-greedy Gaussian policy has the following properties. (i) It has the same compatible feature vector as the traditional Gaussian policy, i.e., $\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}|\boldsymbol{s}) = \nabla_{\boldsymbol{\theta}} \log \vartheta(\boldsymbol{a}|\boldsymbol{s})$, since $\kappa(\boldsymbol{a}|\boldsymbol{s})$ is independent of $\boldsymbol{\theta}$. (ii) If $\epsilon = 1$, it will be degenerated into the traditional Gaussian policy. (iii) All candidate actions have the same generating probability $\vartheta(\boldsymbol{a}|\boldsymbol{s})$ and perturbation size $|\boldsymbol{\rho} \circ \boldsymbol{\sigma}(\boldsymbol{s})|$. (iv) The perturbation directions of $\boldsymbol{u}_j$ and $\boldsymbol{u}_{2^n-j+1}$ are completely opposite. (v) $\pi(\boldsymbol{a}^*|\boldsymbol{s}) = 2^n \left(1 - \epsilon + \frac{\epsilon}{2^n}\right) \vartheta(\boldsymbol{a}|\boldsymbol{s})$.

4. **Compatible AC Algorithms.** To successfully use the ϵ-greedy Gaussian policy for multidimensional continuous action problems, it is necessary to address the learning of the advantage function. In Section 2.3, we have reviewed three NAC algorithms for tackling this problem. In theory, the ϵ-greedy Gaussian policy can be used in them seamlessly. However, they only consider the natural gradient, and most of them only consider one return formulation. Although INAC considers both formulations, the quality of the resulting policy may be significantly affected by the choice of $\boldsymbol{\phi}(\boldsymbol{s})$ [29]. In our previous work [24] and in [21], we and Degris et al. also find that INAC does not perform very well. In this section, we present two general AC frameworks with the ϵ-greedy Gaussian policy by considering compatible function approximation. The first framework is called RLSEGAC, which is built by extending the LSNAC algorithm [16]. The second framework is called SEGAC, which is built by extending the SNAC algorithm [9].

4.1. **RLSEGAC framework.** Now we introduce our first framework. Different from the derivation of LSNAC, our derivation is based on Lemma 3 in [4] rather than the Bellman equation. In addition, our framework uses the recursive least-squares (RLS) method [30], and thus has $O((M+N)^2)$ computational complexity per time step.

From Lemma 3 in [4], we have $A^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbf{E}_{\tau}\left[\delta_t^V | \boldsymbol{s}_t, \boldsymbol{a}_t, \pi\right]$. Then, Equation (8) can be rewritten as

$$\boldsymbol{w}_t^{\mathsf{T}} \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \delta_t^V + \eta(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \tag{25}$$

where $\mathbf{E}_{\tau}\left[\eta(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\right] = 0$. Let $\hat{V}(\boldsymbol{s}) = \boldsymbol{v}^{\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{s})$, $\boldsymbol{\psi}_t = \left[\boldsymbol{\phi}(\boldsymbol{s}_t)^{\mathsf{T}}, \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)^{\mathsf{T}}\right]^{\mathsf{T}}$, $\boldsymbol{\varphi}_t = \left[\boldsymbol{\phi}(\boldsymbol{s}_{t+1})^{\mathsf{T}}, \mathbf{0}^{\mathsf{T}}\right]^{\mathsf{T}}$, $\boldsymbol{\xi}_t = \left[\boldsymbol{v}_t^{\mathsf{T}}, \boldsymbol{w}_t^{\mathsf{T}}\right]^{\mathsf{T}}$ and $\boldsymbol{z}_{t+1}^{\xi} = \left[{\boldsymbol{z}_{t+1}^{v}}^{\mathsf{T}}, {\boldsymbol{z}_{t+1}^{\theta}}^{\mathsf{T}}\right]^{\mathsf{T}} = \gamma\lambda \boldsymbol{z}_t^{\xi} + \boldsymbol{\psi}_t$. Plugging Equation (10) into Equation (25), we can get

$$\left(\boldsymbol{\psi}_t - \gamma\boldsymbol{\varphi}_t\right)^{\mathsf{T}} \boldsymbol{\xi}_t = r_t - \bar{r}_{t+1} + \eta\left(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}\right) \tag{26}$$

Multiplying both sides by $\boldsymbol{z}_{t+1}^{\xi}$, we have

$$\boldsymbol{z}_{t+1}^{\xi}\left(\boldsymbol{\psi}_t - \gamma\boldsymbol{\varphi}_t\right)^{\mathsf{T}} \boldsymbol{\xi}_t = \boldsymbol{z}_{t+1}^{\xi}(r_t - \bar{r}_{t+1}) + \eta(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\boldsymbol{z}_{t+1}^{\xi} \tag{27}$$

Since $\boldsymbol{z}_{t+1}^{\xi}$ is independent of the state transition distribution $\tau(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, we have $\mathbf{E}_{\tau}\left[\eta(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\boldsymbol{z}_{t+1}^{\xi}\right] = \mathbf{0}$. Then, we can obtain

$$\mathbf{E}_{\tau}\left[\boldsymbol{z}_{t+1}^{\xi}(\boldsymbol{\psi}_t - \gamma\boldsymbol{\varphi}_t)^{\mathsf{T}}\boldsymbol{\xi}_t\right] = \mathbf{E}_{\tau}\left[\boldsymbol{z}_{t+1}^{\xi}(r_t - \bar{r}_{t+1})\right] \tag{28}$$

Thus, we can define the optimal function for $\boldsymbol{\xi}_t$ as $\varepsilon_t^{\pi}(\boldsymbol{\xi}) = \|\boldsymbol{A}_t\boldsymbol{\xi} - \boldsymbol{b}_t\|_2^2$, where $\boldsymbol{A}_t = \sum_{i=0}^{t} \boldsymbol{z}_{i+1}^{\xi}(\boldsymbol{\psi}_i - \gamma\boldsymbol{\varphi}_i)^{\mathsf{T}}$ and $\boldsymbol{b}_t = \sum_{i=0}^{t} \boldsymbol{z}_{i+1}^{\xi}(r_i - \bar{r}_{i+1})$. By minimizing $\varepsilon_t^{\pi}(\boldsymbol{\xi})$, we can get

$$\boldsymbol{\xi}_t = \boldsymbol{A}_t^{-1}\boldsymbol{b}_t \tag{29}$$

To reduce the computational burden of matrix inversion, we use the RLS method to solve Equation (29). Let $\boldsymbol{P}_t = \boldsymbol{A}_t^{-1}$, $\boldsymbol{P}_0 = \zeta\boldsymbol{E}$, $\boldsymbol{h}_t = (\boldsymbol{\psi}_t - \gamma\boldsymbol{\varphi}_t)^{\mathsf{T}}\boldsymbol{P}_t$, $\boldsymbol{g}_t = \boldsymbol{P}_t\boldsymbol{z}_{t+1}^{\xi}$ and $k_t = 1 + \boldsymbol{h}_t\boldsymbol{z}_{t+1}^{\xi}$, where ζ is a positive number and $\boldsymbol{E}$ is the identity matrix. Then, the update rules for $\boldsymbol{P}_t$ and $\boldsymbol{\xi}_t$ in the critic of RLSEGAC are

$$\boldsymbol{P}_{t+1} = \boldsymbol{P}_t - \tfrac{1}{k_t}\boldsymbol{g}_t\boldsymbol{h}_t \tag{30}$$

$$\boldsymbol{\xi}_{t+1} = \boldsymbol{\xi}_t + \tfrac{1}{k_t}\delta_t^Q\boldsymbol{g}_t \tag{31}$$

where δ_t^Q is defined as

$$\delta_t^Q = r_t - \bar{r}_{t+1} + \boldsymbol{\xi}_t^{\mathsf{T}}(\gamma\boldsymbol{\varphi}_t - \boldsymbol{\psi}_t) \tag{32}$$

From Equation (8), we have $\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \boldsymbol{\xi}_t^{\mathsf{T}}\boldsymbol{\psi}_t$. Thus, Equation (32) can be rewritten as

$$\delta_t^Q = \delta_t^V + \hat{V}(\boldsymbol{s}_t) - \hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \delta_t^V - \hat{A}(\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{33}$$

Since we use compatible function approximation in the critic, $\nabla_{\boldsymbol{\theta}}^{\mathrm{NG}} J^{\pi}$ can be estimated by Equation (9). Then, from Equation (7), $\nabla_{\boldsymbol{\theta}}^{\mathrm{RG}} J^{\pi}$ can be estimated by $\hat{\nabla}_{\boldsymbol{\theta}}^{\mathrm{RG}} J^{\pi} = \boldsymbol{F}_t \boldsymbol{w}_{t+1}$ at time step t. For further reducing the computational cost, we rewrite $\boldsymbol{F}_t \boldsymbol{w}_{t+1} = \nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)^{\mathsf{T}} \boldsymbol{w}_{t+1} = \boldsymbol{w}_{t+1}^{\mathsf{T}} \nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. Thus, the regular-gradient update rule for $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t^{\theta} \boldsymbol{w}_{t+1}^{\mathsf{T}} \nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \tag{34}$$

and the natural-gradient update rule for $\boldsymbol{\theta}$ is the same as Equation (18).

It is important to note that here the critic is for a fixed policy since it approximates $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ rather than $V^{\pi}(\boldsymbol{s})$. That means samples generated by previous policies are not very good for improving subsequent polices [16, 31, 32, 33]. Comparing this critic with the critics of conventional RAC algorithms and the INAC algorithm, we can find that this adverse effect is mainly caused by the introduction of $\hat{A}(\boldsymbol{s}, \boldsymbol{a})$. To overcome this problem, Morimura et al. [33] and Wawrzynski [9] suggested to decay $\boldsymbol{w}$ after each update, i.e.,

$$\boldsymbol{w}_{t+1} = \beta \boldsymbol{w}_{t+1} \tag{35}$$

where $\beta \in [0, 1]$ is the forgetting factor. Based on the above derivation, the complete RLSEGAC framework is shown in Algorithm 2.

Algorithm 2 RLSEGAC with the ϵ-greedy Gaussian policy

Input: $\boldsymbol{\phi}(\boldsymbol{s})$, $\pi(\boldsymbol{a}|\boldsymbol{s})$, $\nabla_{\boldsymbol{\theta}} \log\pi(\boldsymbol{a}|\boldsymbol{s})$
Initialize: ϵ, γ, λ, β, $\alpha_0^{\bar{r}}$, α_0^{θ}, $\boldsymbol{z}_0^{v}$, $\boldsymbol{z}_0^{\theta}$, $\bar{r}_0$, $\boldsymbol{v}_0$, $\boldsymbol{w}_0$, $\boldsymbol{\theta}_0$, $\boldsymbol{P}_0$, $\boldsymbol{s}_0$
for $t = 0, 1, 2, \cdots$ **do**
    Draw action $\boldsymbol{a}_t$ by Algorithm 1
    Take $\boldsymbol{a}_t$, observe next state $\boldsymbol{s}_{t+1}$ and reward r_t
    Update $\bar{r}_t$, $\boldsymbol{z}_t^{v}$ and $\boldsymbol{z}_t^{\theta}$ by Equations (11), (12) and (14)
    Compute δ_t^{Q} by Equation (32)
    Compute $\boldsymbol{h}_t$, $\boldsymbol{g}_t$ and k_t
    Update $\boldsymbol{P}_t$, $\boldsymbol{v}_t$ and $\boldsymbol{w}_t$ by Equations (30) and (31)
    Update $\boldsymbol{\theta}_t$ by Equation (34) or (18)
    Decay $\boldsymbol{w}_{t+1}$ by (35)
    Update $\alpha_t^{\bar{r}}$ and α_t^{θ}
end for

4.2. **SEGAC framework.** In this subsection, we present our second framework. It can be viewed as an incremental version of RLSEGAC, and has $O(M + N)$ computational complexity per time step.

Let $\boldsymbol{\xi}_t$, $\boldsymbol{\psi}_t$, $\boldsymbol{z}_t^{\xi}$ and δ_t^{Q} have the same meaning as those in Section 4.1. Defining the optimal function $\varepsilon^{\pi}(\boldsymbol{\xi}_t) = \mathbf{E}_{\pi}\left[\left(\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) - Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right)^2 \middle| \boldsymbol{s}_t, \boldsymbol{a}_t\right]$, we have

$$\nabla_{\boldsymbol{\xi}_t} \varepsilon^{\pi}(\boldsymbol{\xi}_t) = 2\mathbf{E}_{\pi}\left[\left(\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) - Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right) \boldsymbol{\psi}_t \middle| \boldsymbol{s}_t, \boldsymbol{a}_t\right] \tag{36}$$

Furthermore, define the λ-return as

$$R_t^{\lambda} = \sum_{n=1}^{\infty} (1 - \lambda) \lambda^{n-1} R_t^{(n)} \tag{37}$$

where $R_t^{(n)}$ denotes the n-step return [2], which is defined as

$$R_t^{(n)} = \begin{cases} r_t - \bar{r}_{t+1} + \gamma \hat{V}(\boldsymbol{s}_{t+1}) & \text{if } n = 1 \\ r_t - \bar{r}_{t+1} + \gamma \hat{V}(\boldsymbol{s}_{t+1}) + \sum_{i=1}^{n-1} \gamma^i \delta_{t+i+1}^Q & \text{if } n > 1 \end{cases}$$

Let $\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \boldsymbol{\xi}_t^\mathsf{T} \boldsymbol{\psi}_t$, $\hat{V}(\boldsymbol{s}) = \boldsymbol{v}^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{s})$ and $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \approx R_t^\lambda$. Then, by the forward-backward view analysis [2, 18], $\nabla_{\boldsymbol{\xi}_t} \varepsilon^\pi(\boldsymbol{\xi}_t)$ can be estimated as

$$\hat{\nabla}_{\boldsymbol{\xi}_t} \varepsilon^\pi(\boldsymbol{\xi}_t) = -2\delta_t^Q \boldsymbol{z}_{t+1}^\xi \tag{38}$$

Thus, the update rule for $\boldsymbol{\xi}$ in the critic of SEGAC is

$$\boldsymbol{\xi}_{t+1} = \boldsymbol{\xi}_t + \alpha_t^\xi \delta_t^Q \boldsymbol{z}_{t+1}^\xi \tag{39}$$

where α_t^ξ is the step size.

The actor of SEGAC is the same as that of RLSEGAC. For the sake of brevity, we omit the derivation. Finally, we summarize the complete SEGAC framework in Algorithm 3.

Algorithm 3 SEGAC with the ϵ-greedy Gaussian policy

Input: $\boldsymbol{\phi}(\boldsymbol{s})$, $\pi(\boldsymbol{a}|\boldsymbol{s})$, $\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}|\boldsymbol{s})$
Initialize: ϵ, γ, λ, β, $\alpha_0^{\bar{r}}$, α_0^ξ, α_0^θ, $\boldsymbol{z}_0^v$, $\boldsymbol{z}_0^\theta$, $\bar{r}_0$, $\boldsymbol{v}_0$, $\boldsymbol{w}_0$, $\boldsymbol{\theta}_0$, $\boldsymbol{s}_0$
for $t = 0, 1, 2, \cdots$ **do**
    Draw action $\boldsymbol{a}_t$ by Algorithm 1
    Take $\boldsymbol{a}_t$, observe next state $\boldsymbol{s}_{t+1}$ and reward r_t
    Update $\bar{r}_t$, $\boldsymbol{z}_t^v$ and $\boldsymbol{z}_t^\theta$ by Equations (11), (12) and (14)
    Compute δ_t^Q by Equation (32)
    Update $\boldsymbol{v}_t$ and $\boldsymbol{w}_t$ by Equation (39)
    Update $\boldsymbol{\theta}_t$ by Equation (34) or (18)
    Decay $\boldsymbol{w}_{t+1}$ by (35)
    Update $\alpha_t^{\bar{r}}$, α_t^ξ and α_t^θ
end for

Remark 4.1. *For episodic problems, when $\boldsymbol{s}_{t+1}$ is an absorbing state, RLSEGAC and SEGAC should let $\boldsymbol{\varphi}_t = \mathbf{0}$, and reset $\boldsymbol{s}_{t+1}$ as the start state of next episode after $\boldsymbol{\theta}_t$ is updated. For the average-return formulation, there is no $\bar{r}_t$ and no $\alpha_t^{\bar{r}}$ in RLSEGAC and SEGAC since $\bar{r}_\pi = 0$.*

Remark 4.2. *If $\epsilon = 1$ or $\boldsymbol{a}_t$ is drawn from the traditional Gaussian policy, RLSEGAC and SEGAC will be degenerated into AC algorithm frameworks with the traditional Gaussian policy, which are called RLSTAC and STAC in this paper.*

Remark 4.3. *Different from in INAC, the parameter vector $\boldsymbol{v}_t$ in RLSEGAC and SEGAC is updated by δ_t^Q rather than δ_t^V.*

5. Performance Analysis.

In this section, we theoretically analyze the influence of the ϵ-greedy Gaussian policy on the performance of compatible AC algorithms.

5.1. Performance improvement analysis.

In compatible AC algorithms, the traditional Gaussian policy is improved only by policy gradient descent. By contrast, the ϵ-greedy Gaussian policy can be indirectly improved by maximizing the advantage functions of candidate actions, even if it does not use policy gradient descent.

Theorem 5.1. *For any traditional Gaussian policy ϑ, the corresponding ϵ-greedy Gaussian policy π with respect to $Q^{\vartheta}(\boldsymbol{s},\boldsymbol{a})$ is better than or equal to ϑ.*

Proof: Let $Q^{\vartheta}(\boldsymbol{s},\boldsymbol{a}^*) = \max_{\boldsymbol{u}_j\in\mathcal{A}(\boldsymbol{s})} Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j)$. Based on the definition and properties of π, for any $\boldsymbol{u}_i,\boldsymbol{u}_j \in \mathcal{A}(\boldsymbol{s})$, we have: (i) $\vartheta(\boldsymbol{u}_i|\boldsymbol{s}) = \vartheta(\boldsymbol{u}_j|\boldsymbol{s})$; (ii) If $\boldsymbol{u}_j = \boldsymbol{a}^*$, then $\pi(\boldsymbol{u}_j|\boldsymbol{s}) = 2^n\left(1-\epsilon+\frac{\epsilon}{2^n}\right)\vartheta(\boldsymbol{u}_j|\boldsymbol{s})$, else $\pi(\boldsymbol{u}_j|\boldsymbol{s}) = \epsilon\vartheta(\boldsymbol{u}_j|\boldsymbol{s})$. Thus

$$\begin{aligned}\textstyle\sum_j \pi(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j) &= (1-\epsilon)2^n\vartheta(\boldsymbol{a}^*|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{a}^*) + \epsilon\textstyle\sum_j \vartheta(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j)\\ &\geq (1-\epsilon)\textstyle\sum_j \vartheta(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j) + \epsilon\textstyle\sum_j \vartheta(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j)\\ &= \textstyle\sum_j \vartheta(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j)\end{aligned}$$

Let $\pi(\boldsymbol{s}) \in \mathcal{A}$ denote the action determined by π in $\boldsymbol{s}$. For any $\boldsymbol{s}\in\mathcal{S}$,

$$\begin{aligned}Q^{\vartheta}(\boldsymbol{s},\pi(\boldsymbol{s})) &= \textstyle\int_{\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{a})\mathrm{d}\boldsymbol{a}\\ &= \tfrac{1}{2^n}\textstyle\sum_j \int_{\mathcal{A}} \pi(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j)\mathrm{d}\boldsymbol{u}_j\\ &\geq \tfrac{1}{2^n}\textstyle\sum_j \int_{\mathcal{A}} \vartheta(\boldsymbol{u}_j|\boldsymbol{s})Q^{\vartheta}(\boldsymbol{s},\boldsymbol{u}_j)\mathrm{d}\boldsymbol{u}_j\\ &= V^{\vartheta}(\boldsymbol{s})\end{aligned}$$

Therefore, by the policy improvement theorem [2], the claim follows.

5.2. **Convergence analysis.** From the definition of the ϵ-greedy Gaussian policy, the candidate action set $\mathcal{A}(\boldsymbol{s})$ is generated by the traditional Gaussian policy. That means the ϵ-greedy Gaussian policy is built on the traditional Gaussian policy, i.e., $\pi\sim\vartheta$. Although its greediness is increased by introducing the ϵ-greedy policy, it is still a stochastic policy rather than a deterministic greedy policy. Thus, we believe that the ϵ-greedy Gaussian policy does not cause exploration insufficiency and cannot destroy the original convergence of compatible AC algorithms with the traditional Gaussian policy.

Theorem 5.2. *For any traditional Gaussian policy ϑ, if a compatible AC algorithm with it is convergent, then the algorithm with the corresponding ϵ-greedy Gaussian policy π is also convergent.*

Proof: Since $\pi\sim\vartheta$ and $\nabla_{\boldsymbol{\theta}}\log\pi(\boldsymbol{a}|\boldsymbol{s}) = \nabla_{\boldsymbol{\theta}}\log\vartheta(\boldsymbol{a}|\boldsymbol{s})$, the differences between $\nabla^{\mathrm{PG}}_{\boldsymbol{\theta}}J^{\pi}$ and $\nabla^{\mathrm{PG}}_{\boldsymbol{\theta}}J^{\vartheta}$ are mainly manifest in $d(\boldsymbol{s})$ and $A(\boldsymbol{s},\boldsymbol{a})$. In other words, the influence of introducing the ϵ-greedy policy on the convergence of a compatible AC algorithm is mainly manifest in the critic parameters. By the two-timescale stochastic approximation theory [4], the convergence of the critic parameters is analyzed under a given $\boldsymbol{\theta}$. By Theorem 5.1, if J^{ϑ}, $\boldsymbol{v}^{\vartheta}$ and $\boldsymbol{w}^{\vartheta}$ are convergent under a given $\boldsymbol{\theta}$, then J^{π} is convergent surely, and thus $\boldsymbol{v}^{\pi}$ and $\boldsymbol{w}^{\pi}$ are also convergent surely. The claim follows.

Corollary 5.1. *RLSEGAC and SEGAC have a good convergence guarantee.*

Proof: It is well known that AC algorithms generally have a reliable convergence guarantee [3, 4], and experimental results in [9, 16] show that LSNAC and SNAC with the traditional Gaussian policy are convergent. Furtherly, RLSEGAC can be viewed as an extension of LSNAC, and SEGAC can also be viewed as an extension of SNAC. By Theorem 5.2, the claim follows.

6. **Experiments.** In this section, in order to demonstrate the effectiveness of the ϵ-greedy Gaussian policy and the proposed compatible AC algorithms, we use a continuous mountain-car problem [2, 6] and a continuous puddle-world problem [18, 23] to compare the following algorithm frameworks: (i) RLSEGAC and RLSTAC; (ii) SEGAC and STAC.

6.1. **Experimental settings.** In the mountain-car problem, the task is to learn a throttle control policy, which forces an underpowered car from a valley to the top of the rightmost hill as quickly as possible. At time step t, the state $\boldsymbol{s}_t = (x_t, \dot{x}_t)$ is updated by $x_{t+1} = bound[x_t + \dot{x}_{t+1}]$ and $\dot{x}_{t+1} = bound[\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)]$, where x_t and $\dot{x}_t$ are the position and velocity of the car, the action $a_t \in [-1, 1]$ is the throttle value, and the bound operation enforces $x_{t+1} \in [-1.5, 0.5]$ and $\dot{x}_{t+1} \in [-0.07, 0.07]$. If $x_{t+1} = 0.5$, then the reward r_t is 100 and the goal is reached; otherwise, r_t is -1. In this experiment, a run includes 100 episodes. At each episode, the car starts from an initial state $\boldsymbol{s}_0 = (-0.5, 0)$, and ends when t reaches 1000 or the car reaches the goal.

In the puddle-world problem, the task is to learn a move policy, which forces an agent from a given position to the top-right corner of the puddle world as quickly as possible. At time step t, the state $\boldsymbol{s}_t = (x_t, y_t) \in [0, 1]^2$ denotes the position in the puddle world, the action $\boldsymbol{a}_t = (a_t^x, a_t^y) \in [-0.025, 0.025]^2$ denotes the moves in both dimensions, and the reward for arriving in the successor state $\boldsymbol{s}_{t+1} = (p_x, p_y)$ is defined as $r_t = -1 - 2(\mathcal{N}(p_x, 0.3, 0.1) \cdot \mathcal{N}(p_y, 0.6, 0.03) + \mathcal{N}(p_x, 0.4, 0.03) \cdot \mathcal{N}(p_y, 0.5, 0.1) + \mathcal{N}(p_x, 0.8, 0.03) \cdot \mathcal{N}(p_y, 0.9, 0.1))$, where $\mathcal{N}(p_\bullet, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(p_\bullet - \mu)^2}{2\sigma^2}\right)$. In this experiment, a run includes 100 episodes. At each episode, the agent starts from an initial state $\boldsymbol{s}_0 = (0.2, 0.4)$, and ends when t reaches 1000 or the goal is reached (i.e., $|p_x - 1| + |p_y - 1| < 0.1$).

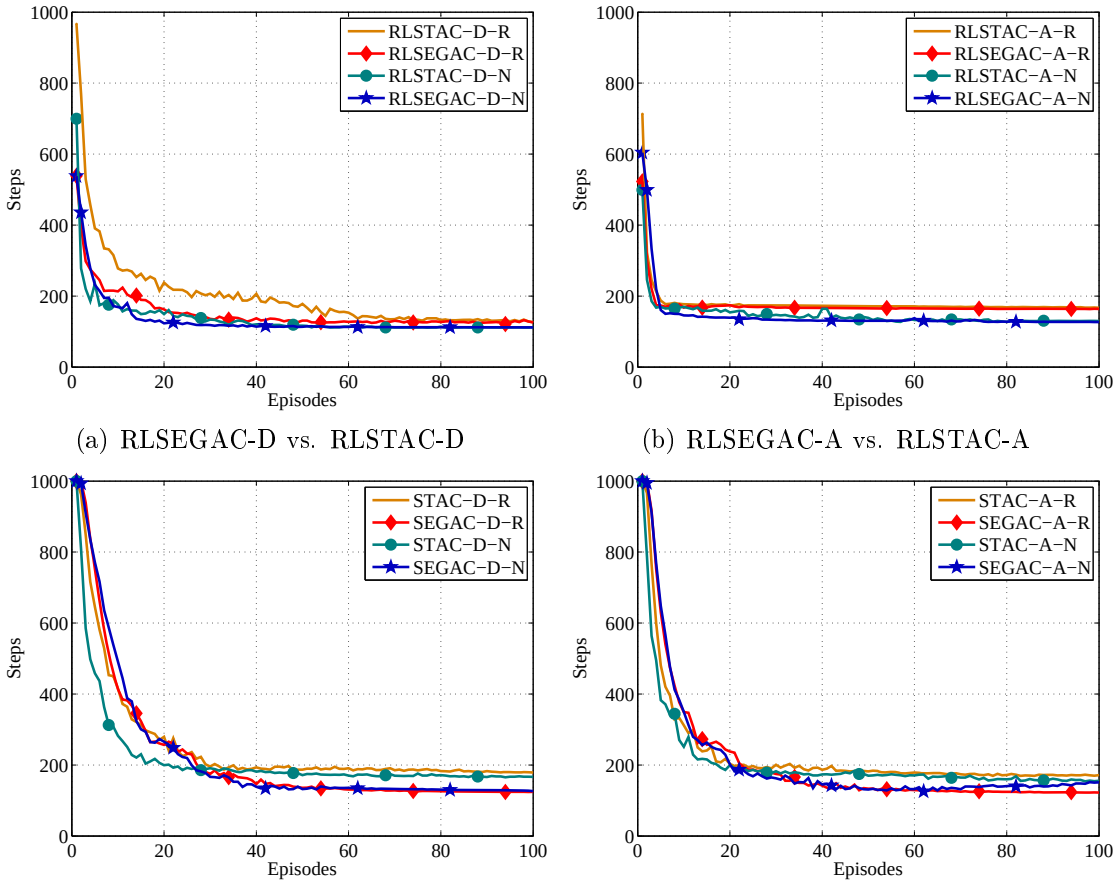
In the implementations of all compatible AC algorithms for both problems, the settings are summarized as follows. (i) The traditional Gaussian policy is defined as $\vartheta(\boldsymbol{a}|\boldsymbol{s}) = \prod_{i=1}^{n} \mathcal{N}\left(\boldsymbol{\theta}_i^{\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{s}), 1\right)$, and the ϵ-greedy Gaussian policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ is built on $\vartheta(\boldsymbol{a}|\boldsymbol{s})$ with $\epsilon = 0.01$. (ii) The feature vector $\boldsymbol{\phi}(\boldsymbol{s})$ is evenly constructed over the state space by 10×10 Gaussian RBFs, and the window width of RBFs for the i-th state component is defined as $\sigma_i = \frac{ub_i - lb_i}{9\sqrt{2}}$ $(i = 1, 2)$, where ub_i and lb_i are the upper and lower bounds of the i-th state component. (iii) All parameter vectors, eligibility traces and $\bar{r}_0$ are initialized to zero vectors, and the trace-decay parameter λ is set to 0.5. (iv) For all RLSEGAC and RLSTAC algorithms, $\boldsymbol{P}_0$ is initialized to $0.15\boldsymbol{E}$. (v) All step sizes are constants, which are selected from $\{0.00001, 0.00005, 0.0001, \cdots, 0.1, 0.5\}$ by using the parameter-sweep method [21]. Similarly, the forgetting factor β is selected from $\{0.99, 1.0\}$, and the discount factor γ is selected from $\{0.97, 0.98, \cdots, 1.0\}$ when the compatible AC algorithm uses the discount-return formulation. Considering that β will decay the compatible parameter vector $\boldsymbol{w}$ if $\beta < 1$, we allow $\alpha_\xi = \alpha_\theta$ during sweeping parameters.

6.2. **Experimental results and analysis.** The best parameters for each compatible AC algorithm and all average performance comparisons over 30 runs are summarized in Tables 1 and 2 and Figures 2 and 3. At the end of each algorithm's name, 'D' or 'A' denotes the discount or average return, and 'R' or 'N' denotes the regular or natural gradient. For example, RLSEGAC-D-R denotes the RLSEGAC algorithm with the discount return and the regular gradient. In Tables 1 and 2, 'Steps-I' and 'Steps-II' denote the average time steps with standard errors to the goal at the 30th episode and the 100th episode. In Figures 2 and 3, steps are the average time steps to the goal as the number of episodes increases.

From Tables 1 and 2 and Figures 2 and 3, we can see the followings. (i) Compared with RLSTAC and STAC algorithms, all RLSEGAC and SEGAC algorithms can obtain better convergence quality. Since all RLSTAC and STAC algorithms also use the best parameters, the performance improvement of RLSEGAC and SEGAC algorithms mainly benefits from the introduction of the ϵ-greedy policy rather than the selection of parameters. (ii) The performance improvement of RLSEGAC and SEGAC algorithms is more significant in the puddle-world problem than in the mountain-car problem, since there

TABLE 1. Best parameters and average time steps for the mountain-car problem

	$\alpha^{\bar{r}}$	α^{ξ}	α^{θ}	β	γ	Steps-I	Steps-II
RLSTAC-D-R	–	–	0.005	1.0	0.97	207.0±49.7	126.5±15.3
RLSEGAC-D-R	–	–	0.01	1.0	0.98	135.9±22.5	125.6±26.7
RLSTAC-D-N	–	–	0.005	1.0	0.99	133.6±32.6	111.7±4.5
RLSEGAC-D-N	–	–	0.005	1.0	0.99	118.4±13.9	111.7±1.7
RLSTAC-A-R	0.0001	–	0.01	1.0	–	173.9±23.8	167.2±22.8
RLSEGAC-A-R	0.0001	–	0.005	1.0	–	167.4±3.2	163.7±2.9
RLSTAC-A-N	0.0001	–	0.01	1.0	–	147.1±33.5	129.8±28.9
RLSEGAC-A-N	0.0001	–	0.01	1.0	–	133.0±26.8	127.0±19.9
STAC-D-R	–	0.01	0.01	0.99	1.0	195.7±38.0	178.8±19.4
SEGAC-D-R	–	0.01	0.01	0.99	1.0	174.9±21.1	124.4±19.6
STAC-D-N	–	0.01	0.01	0.99	0.99	191.0±20.3	166.4±19.4
SEGAC-D-N	–	0.01	0.01	0.99	0.99	166.4±42.8	126.5±29.2
STAC-A-R	0.0001	0.01	0.005	1.0	–	200.7±35.0	171.6±21.9
SEGAC-A-R	0.0001	0.01	0.01	0.99	–	174.6±28.8	122.4±17.0
STAC-A-N	0.00001	0.01	0.01	0.99	–	182.3±22.3	155.5±25.6
SEGAC-A-N	0.0001	0.01	0.005	0.99	–	162.4±32.2	151.1±37.3



(a) RLSEGAC-D vs. RLSTAC-D

(b) RLSEGAC-A vs. RLSTAC-A

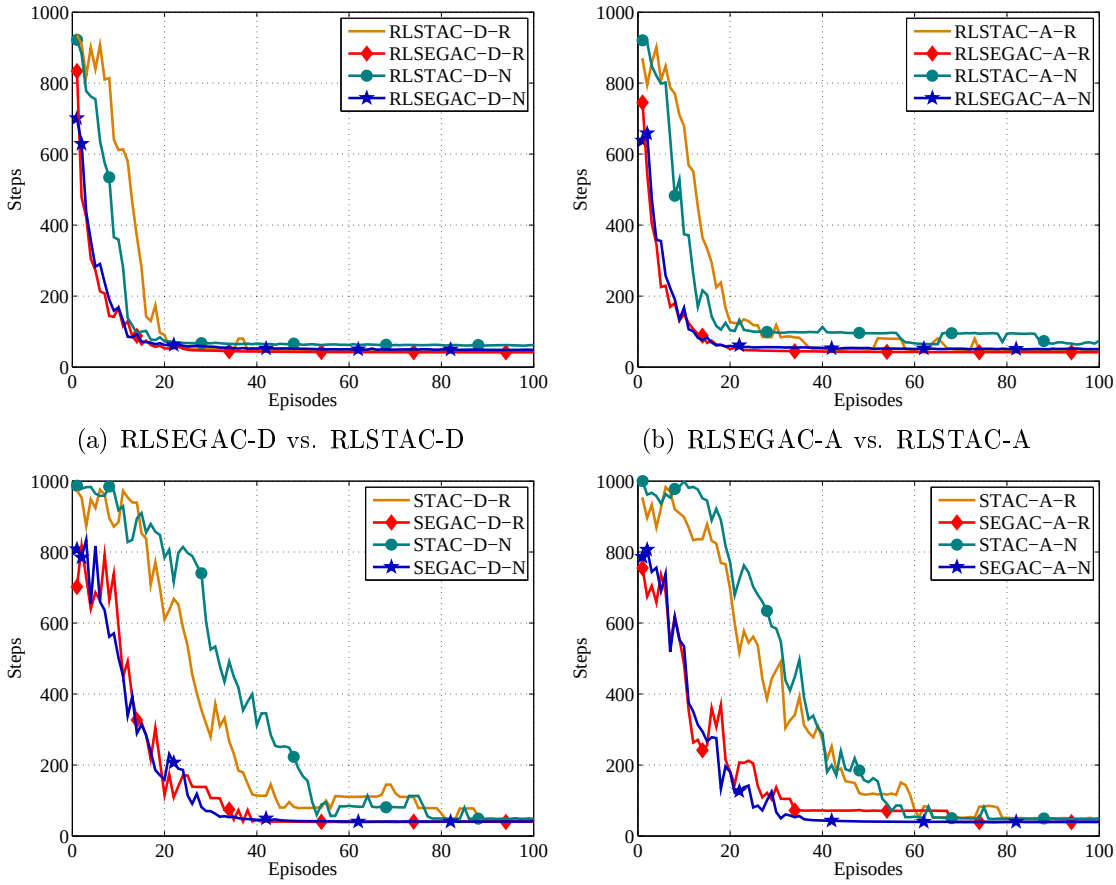
(c) SEGAC-D vs. STAC-D

(d) SEGAC-A vs. STAC-A

FIGURE 2. Average performance comparisons for the mountain-car problem

TABLE 2. Best parameters and average time steps for the puddle-world problem

	$\alpha^{\bar{r}}$	α^{ξ}	α^{θ}	β	γ	Steps-I	Steps-II
RLSTAC-D-R	–	–	0.005	1.0	1.0	49.6±13.7	46.4±8.9
RLSEGAC-D-R	–	–	0.005	1.0	1.0	46.1±9.7	41.5±5.5
RLSTAC-D-N	–	–	0.005	0.99	1.0	66.8±10.5	61.7±9.5
RLSEGAC-D-N	–	–	0.005	0.99	1.0	57.3±22.8	48.4±6.7
RLSTAC-A-R	0.00001	–	0.005	1.0	–	118.3±240.7	47.0±6.8
RLSEGAC-A-R	0.00001	–	0.005	1.0	–	45.9±9.0	41.6±7.3
RLSTAC-A-N	0.00001	–	0.005	0.99	–	97.2±171.2	74.2±61.2
RLSEGAC-A-N	0.00001	–	0.005	0.99	–	56.0±15.6	50.6±6.5
STAC-D-R	–	0.1	0.005	0.99	1.0	279.3±404.5	45.1±3.3
SEGAC-D-R	–	0.1	0.005	0.99	1.0	106.9±242.9	39.8±3.7
STAC-D-N	–	0.05	0.001	0.99	1.0	525.2±453.2	47.9±6.6
SEGAC-D-N	–	0.05	0.001	0.99	1.0	69.9±58.6	41.4±3.2
STAC-A-R	0.00001	0.1	0.005	0.99	–	467.2±460.5	51.1±9.5
SEGAC-A-R	0.00001	0.1	0.005	0.99	–	137.2±292.5	39.1±2.5
STAC-A-N	0.00001	0.05	0.001	0.99	–	584.6±453.1	48.8±5.2
SEGAC-A-N	0.00001	0.05	0.001	0.99	–	67.5±59.9	40.4±2.4



(a) RLSEGAC-D vs. RLSTAC-D

(b) RLSEGAC-A vs. RLSTAC-A

(c) SEGAC-D vs. STAC-D

(d) SEGAC-A vs. STAC-A

FIGURE 3. Average performance comparisons for the puddle-world problem

are more candidate actions per time step and thus the ϵ-greedy Gaussian policy has more greediness. That indicates the ϵ-greedy Gaussian policy is more suitable for multidimensional continuous action problems than the traditional Gaussian policy. (iii) In the mountain-car problem, SEGAC algorithms converge slower than STAC algorithms at the first 30 episodes, but they are obviously superior in the final convergence quality. That indicates the ϵ-greedy Gaussian policy is helpful to avoid trapping into a local optimal solution. (iv) Compared with SEGAC algorithms, RLSEGAC algorithms converge much faster, since they can reuse samples efficiently [17]. However, SEGAC algorithms are more suitable for real-time RL problems, since they have linear computational complexity per time step.

Compared with the previous work reviewed in Section 1, the ϵ-greedy Gaussian policy and the proposed compatible AC frameworks have several other advantages. (i) They are simple and easy to implement. (ii) They do not require extra memory and sampling cost. (iii) The ϵ-greedy Gaussian policy bridges the gap between the traditional Gaussian policy and the ϵ-greedy policy. It allows us to adjust the greediness of the behaviour policy by selecting ϵ. (iv) By using the symmetry of the normal distribution, the ϵ-greedy Gaussian policy has the same compatible feature vector as the traditional Gaussian policy, and does not change the update formulas of policy parameters. Thus, it can be used in any compatible AC algorithm supporting continuous states and actions, and can also be used in online, offline, on-policy or off-policy settings. (v) Since the ϵ-greedy Gaussian policy is still a stochastic policy and its influence is mainly manifest in policy evaluation, it does not destroy the original convergence of compatible AC algorithms.

7. **Conclusions.** In this paper, we argue that the traditional Gaussian policy has a tendency of over-exploration due to lack of greediness. To overcome this problem, we propose a novel ϵ-greedy Gaussian policy, which can be viewed as a hybrid of the traditional Gaussian policy and the ϵ-greedy policy. At each time step, it generates a candidate action set by using symmetric Gaussian perturbations, and then uses the ϵ-greedy policy to select the behaviour action based on the advantage functions of all candidate actions. For successfully using this policy, we also present RLSEGAC and SEGAC frameworks based on compatible function approximation. Within each framework, we consider the discount return and the average return in the critic, and consider the regular gradient and the natural gradient in the actor. The theoretical analysis shows that the ϵ-greedy Gaussian policy is more powerful for policy improvement than the traditional Gaussian policy, and compatible AC algorithms with it do not suffer from divergence although the greediness is increased. Experimental results demonstrate the effectiveness of the proposed policy and frameworks for two benchmark problems with continuous state and action spaces.

This research shows that it is very promising to introduce the ϵ-greedy policy into AC algorithms for improving the convergence performance. There are also some interesting topics to be studied in future work. (i) A more thorough experimental study, especially for high-dimensional continuous action problems, should be done. (ii) The comparison of sample efficiency between the ϵ-greedy Gaussian policy and the traditional Gaussian policy should be made theoretically. (iii) The ϵ-greedy Gaussian policy requires symmetric Gaussian perturbations, which limit the number of candidate actions and also limit the increase of greediness. Thus, other perturbation methods and the corresponding AC algorithms should be further investigated.

Acknowledgment. This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61300192 and 11261015, the Fundamental Research Funds for the Central Universities under Grant No. ZYGX2014J052, and the Natural

Science Foundation of Hainan Province, China, under Grant No. 613153. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] I. Grondman, L. Busoniu, G. A. D. Lopes and R. Babuska, A survey of actor-critic reinforcement learning: Standard and natural policy gradients, *IEEE Trans. Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol.42, no.6, pp.1291-1307, 2012.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, 1998.

[3] V. R. Konda and J. N. Tsitsiklis, On actor-critic algorithms, *SIAM J. on Control and Optimization*, vol.42, no.4, pp.1143-1166, 2003.

[4] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh and M. Lee, Natural actor-critic algorithms, *Technical Report*, Department of Computing Science, University of Alberta, Canada, 2009.

[5] A. Lazaric, M. Restelli and A. Bonarini, Reinforcement learning in continuous action spaces through sequential Monte Carlo methods, *Proc. of Advances in Neural Information Processing Systems 20*, Vancouver, Canada, pp.833-840, 2007.

[6] J. A. Martin, J. de Lope and D. Maravall, Robust high performance reinforcement learning through weighted k-nearest neighbors, *Neurocomputing*, vol.74, no.8, pp.1251-1259, 2011.

[7] J. C. Santamaría, R. S. Sutton and A. Ram, Experiments with reinforcement learning in problems with continuous state and action spaces, *Adaptive Behaviour*, vol.6, no.2, pp.163-217, 1997.

[8] F. S. Melo and M. Lopes, Fitted natural actor-critic: A new algorithm for continuous state-action MDPs, *Proc. of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Database*, Antwerp, Belgium, pp.66-81, 2008.

[9] P. Wawrzynski, Real-time reinforcement learning by sequential actor-critics and experience replay, *Neural Networks*, vol.22, no.10, pp.1484-1497, 2009.

[10] P. Wawrzynski and A. K. Tanwani, Autonomous reinforcement learning with experience replay, *Neural Networks*, vol.41, pp.156-167, 2013.

[11] H. van Hasselt and M. A. Wiering, Reinforcement learning in continuous action spaces, *Proc. of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Honolulu, USA, pp.272-279, 2007.

[12] H. van Hasselt and M. A. Wiering, Using continuous action spaces to solve discrete problems, *Proc. of the International Joint Conference on Neural Networks*, Atlanta, USA, pp.1149-1156, 2009.

[13] H. van Hasselt, Reinforcement learning in continuous state and action spaces, *Chap. of Reinforcement Learning: State of the Art*, pp.207-251, 2012.

[14] N. A. Vien and W. Ertel, Learning via human feedback in continuous state and action spaces, *Proc. of AAAI Fall Symposium on Robots Learning Interactively from Human Teachers*, Arlington, USA, pp.65-72, 2012.

[15] H. Kimura and S. Kobayashi, An actor-critic algorithm using binary tree action selector, *Trans. of the Society of Instrument and Control Engineers*, vol.4, no.1, pp.1-9, 2007.

[16] J. Peters and S. Schaal, Natural actor-critic, *Neurocomputing*, vol.71, nos.7-9, pp.1180-1190, 2008.

[17] Y. Cheng, H. Feng and X. Wang, Efficient data use in incremental actor-critic algorithms, *Neurocomputing*, vol.116, pp.346-354, 2013.

[18] T. Degris, M. White and R. S. Sutton, Off-policy actor-critic, *Proc. of the 29th International Conference on Machine Learning*, Edinburgh, Scotland, pp.457-464, 2012.

[19] D. H. Lee and J. J. Lee, Incremental receptive field weighted actor-critic, *IEEE Trans. Industrial Informatics*, vol.9, no.1, pp.62-71, 2013.

[20] W. Zhu, Y. Jin, Y. Fu and X. Song, Recursive least-squares actor-critic algorithm in continuous space, *Application Research of Computers*, vol.31, no.7, pp.1994-1997, 2014.

[21] T. Degris, P. M. Pilarski and R. S. Sutton, Model-free reinforcement learning with continuous action in practice, *Proc. of American Control Conference*, Montreal, Canada, pp.2177-2182, 2012.

[22] T. Rückstieß, M. Felder and J. Schmidhuber, State-dependent exploration for policy gradient methods, *Proc. of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Database*, Antwerp, Belgium, pp.234-249, 2008.

[23] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, Deterministic policy gradient algorithms, *Proc. of the 31st International Conference on Machine Learning*, Beijing, China, pp.387-395, 2014.

[24] C. Zhang and Q. Zhu, Actor-critic algorithms based on symmetric perturbation sampling, *Control and Decision*, vol.30, no.12, pp.2161-2167, 2015.

[25] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, *Proc. of Advances in Neural Information Processing Systems 12*, Denver, USA, pp.1057-1063, 1999.

[26] M. Geist and O. Pietquin, Revisiting natural actor-critics with value function approximation, *Proc. of the 7th International Conference on Modeling Decisions for Artificial Intelligence*, Perpignan, France, pp.207-218, 2010.

[27] J. N. Tsitsiklis and B. V. Roy, An analysis of temporal-difference learning with function approximation, *IEEE Trans. Automatic Control*, vol.42, no.5, pp.674-690, 1997.

[28] J. N. Tsitsiklis and B. V. Roy, Average cost temporal-difference learning, *Automatica*, vol.35, no.11, pp.1799-1808, 1999.

[29] N. Heess, D. Silver and Y. W. Teh, Actor-critic reinforcement learning with energy-based policies, *Proc. of the 10th European Workshop on Reinforcement Learning*, Edinburgh, Scotland, pp.43-58, 2012.

[30] X. Xu, H. G. He and D. Hu, Efficient reinforcement learning using recursive least-squares methods, *Journal of Artificial Intelligence Research*, vol.16, pp.259-292, 2002.

[31] T. Mori, Y. Nakamura and S. Ishii, Off-policy natural actor-critic, *Technical Report 2005007*, Nara Institute of Science and Technology, Japan, 2005.

[32] J. Park, J. Kim and D. Kang, An RLS-based natural actor-critic algorithm for locomotion of a two-linked robot arm, *Proc. of International Conf. Computational Intelligence and Security*, Xi'an, China, pp.65-72, 2005.

[33] T. Morimura, E. Uchibe and K. Doya, Utilizing natural gradient in temporal difference reinforcement learning with eligibility traces, *Proc. of the 2nd International Symposium on Information Geometry and its Applications*, Tokyo, Japan, pp.256-263, 2005.