# Riemannian Q-Functions for Policy Iteration in Reinforcement Learning

Minh Vu, Konstantinos Slavakis
Department of Information and Communications, Institute of Science Tokyo, Japan
vu.d.a5c3@m.isct.ac.jp, slavakis@ict.eng.isct.ac.jp

***Abstract*—This paper establishes a novel role for Gaussian-mixture models (GMMs) as functional approximators of Q-function losses in reinforcement learning (RL). Unlike the existing RL literature, where GMMs play their typical role as estimates of probability density functions, GMMs approximate here Q-function losses. The new Q-function approximators, coined GMM-QFs, are incorporated in Bellman residuals to promote a Riemannian-optimization task as a novel policy-evaluation step in standard policy-iteration schemes. The paper demonstrates how the hyperparameters (means and covariance matrices) of the Gaussian kernels are learned from the data, opening thus the door of RL to the powerful toolbox of Riemannian optimization. Numerical tests show that with no use of experienced data, the proposed design outperforms state-of-the-art methods, even deep (D)RL methods which use experienced data, on benchmark RL tasks.**

***Index Terms*—Gaussian-mixture models, reinforcement learning, Q-functions, Riemannian manifold, optimization.**

## I. Introduction

Reinforcement learning (RL) [1, 2] is a machine learning paradigm in which an "agent" interacts with an unknown environment to identify an optimal policy that minimizes the total costs of its "actions". RL learns through trial and error guided by the feedback data from environment, offers a mathematically sound framework for solving arduous sequential decision problems in real-world applications, as in operations research, dynamic control, data mining, and bioinformatics [1].

A central concept in RL is the value function, also known as *Q-function*, which estimates the expected long-term costs that an agent would suffer had it taken an action at a given state under a particular policy. By Q-functions, RL strategies evaluate the corresponding policies, then thereby identify the optimal one. The classical Q-learning [3] and state-action-reward-state-action (SARSA) [4] algorithms evaluate Q-functions by look-up tables, populated by Q-function values at *every possible* state-action pair. Although such approaches appear to be successful in discrete-state-action RL, many practical problems which involve very large, or even continuous state-action spaces could render tabular RL methods computationally intractable. To overcome this difficulty, algorithms built on functional approximations (non-linear models) of Q-functions have attracted considerable interest [1].

Functional approximations of Q-functions have a long history in RL. Classical kernel-based (KB)RL methods [5–7] model Q-functions as elements of Banach spaces; usually, spaces comprising all essentially bounded functions. On the other hand, temporal difference (TD) [8], least-squares (LS)TD [9–11], Bellman-residual (BR) methods [12], as well as very recent nonparametric designs [13–15], model Q-functions as elements of user-defined reproducing kernel Hilbert spaces (RKHSs) [16, 17] in a quest to exploit the geometry and computational convenience of the associated inner product and its reproducing property. Notwithstanding, the number of design parameters of all of the aforementioned kernel-based designs scale with the number of observed data, which usually inflicts memory and computational bottlenecks when operating in dynamic environments with time-varying data distributions. Dimensionality reduction techniques have been introduced to address this issue [11, 13], but reducing the number of basis elements of the approximating subspace may hinder the quality of the Q-functions estimates. Deep neural networks (DNNs) have been also used as non-linear Q-function approximators in the form of deep Q-networks (DQNs), e.g., [18, 19]. Typically, DQN models require experienced data [20] from past policies for their parameters to be learned, and may even require re-training during online mode to learn from data with probability density functions (PDFs) which are different from those of the past data (experience-replay buffer). Such requirements may yield large computational times and complexity footprints, discouraging the application of DQNs into online learning where lightweight operations and swift adaptability to a dynamic environment are desired.

Aiming at a novel class of Q-function estimates with rich approximating properties and with *no* need for past experienced data, this paper introduces the class of *Gaussian-mixture-model Q-functions (GMM-QFs)*, a weighted sum of multivariate Gaussian kernels (functions) [21]. By leveraging ability of GMMs to approximate complex functions with limited number of parameters, GMM-QFs provide a flexible but yet compact representation of Q-functions, therefore effect dimensionality reduction, robustness to errorneous information and swift adaptability to dynamic environments. Contrast to the aforementioned literature of KBRL [5–12], where the hyperparameters of the user-defined kernels are directly parametrized by the observed data and are not considered variables of learning tasks, here not only the weights but also the hyperparameters of the Gaussian kernels are free to be learned. GMMs have been already used in RL, but via their typical role as estimates of PDFs: either of the joint PDF $p(Q, \mathbf{s}, a)$ [22–25], where the Q-function $Q$, as well as state $\mathbf{s}$ and action $a$ are considered to be random variables (RVs),

or of the conditional PDF $p(Q \mid \mathbf{s}, a)$ [26]. This classical usage of GMMs and its intimate connection with maximum-likelihood estimation [27, 28] lead naturally to expectation-maximization (EM) solutions [22–25]. In contrast, motivated by the *universal approximation properties* of GMMs [29], this paper departs from the typical GMM usage [22–26], employs GMMs to model Q-functions *directly,* and not their PDFs, follows the lines of Bellman-residual (BR) minimization [12, 15] to form and minimize a smooth objective function via Riemannian optimization [30] and to exploit the underlying Riemannian geometry [31] of the hyperparameter space. GMMs and Riemannian optimization have been used to model policy functions as PDFs [32], under the framework of policy search [33]. The use of GMMs to model Q-functions directly via Riemannian optimization seems to appear here for the *first time* in the RL literature.

A fixed number of Gaussian kernels are used in GMM-QFs to address the problem of an overgrowing model with the number of data [5–12], effecting dimensionality reduction, and providing low-computational load as well as the adaptability to dynamic environments and robustness to erroneous information. Numerical tests on benchmark control tasks demonstrate that the advocated GMM-QFs outperform other state-of-the-art RL schemes, even DRLs which require experienced data. Due to limited space, definitions and arguments of Riemannian geometry, proofs, results on convergence, and further numerical tests are deferred to the journal version of this manuscript.

## II. The Class of GMM Q-Functions (GMM-QFs)

Let $\mathfrak{S} \subset \mathbb{R}^D$ denote the *continuous* state space, with state vector $\mathbf{s} \in \mathfrak{S}$, for some $D \in \mathbb{N}_*$ ($\mathbb{N}_*$ is the set of all positive integers). The usually discrete action space is denoted by $\mathfrak{A}$, with action $a \in \mathfrak{A}$. An agent, currently at state $\mathbf{s} \in \mathfrak{S}$, takes an action $a \in \mathfrak{A}$ and transits to a new state $\mathbf{s}' \in \mathfrak{S}$ under transition probability $p(\mathbf{s}' \mid \mathbf{s}, a)$ with an *one-step* loss $g(\mathbf{s}, a)$. The Q-function $Q(\cdot,\cdot)\colon \mathfrak{S} \times \mathfrak{A} \to \mathbb{R}\colon (\mathbf{s}, a) \mapsto Q(\mathbf{s}, a)$ stands for the *long-term* loss/cost that the agent will suffer/pay, if the agent takes action $a$ at state $\mathbf{s}$. For convenience, the state-action tuple $\mathbf{z} \coloneqq (\mathbf{s}, a) \in \mathfrak{Z} \coloneqq \mathfrak{S} \times \mathfrak{A} \subset \mathbb{R}^{D_z}$, where $D_z \in \mathbb{N}_*$.

Following [1], consider the set of all mappings $\mathscr{M} \coloneqq \{\mu(\cdot) \mid \mu(\cdot)\colon \mathfrak{S} \to \mathfrak{A}\colon \mathbf{s} \mapsto \mu(\mathbf{s})\}$. In other words, $\mu(\mathbf{s})$ denotes the action that the agent will take at state $\mathbf{s}$ under $\mu$. The set of policies is defined as $\Pi \coloneqq \mathscr{M}^{\mathbb{N}} \coloneqq \{\mu_0, \mu_1, \dots, \mu_n, \dots \mid \mu_n \in \mathscr{M}, n \in \mathbb{N}\}$. A policy will be denoted by $\pi \in \Pi$. Given $\mu \in \mathscr{M}$, a stationary policy $\pi_\mu$ is defined as $\pi_\mu \coloneqq (\mu, \mu, \dots, \mu, \dots)$. It is customary for $\mu$ to denote also $\pi_\mu$.

Motivated by GMMs [21] and their universal approximation properties [29], for a user-defined $K \in \mathbb{N}_*$, GMM-QFs are defined as the following class of functions:

$$\mathscr{Q} \coloneqq \Big\{ Q(\mathbf{z}) \coloneqq \sum_{k=1}^{K} \xi_k \mathcal{G}(\mathbf{z} \mid \mathbf{m}_k, \mathbf{C}_k) \;\Big|\; \xi_k \in \mathbb{R}, \mathbf{m}_k \in \mathbb{R}^{D_z}, \mathbb{R}^{D_z \times D_z} \ni \mathbf{C}_k \text{ is positive definite}, \forall k = 1, \dots, K \Big\}, \quad (1)$$

where $\mathcal{G}(\mathbf{z} \mid \mathbf{m}_k, \mathbf{C}_k) \coloneqq \exp[-(\mathbf{z} - \mathbf{m}_k)^\intercal \mathbf{C}_k^{-1} (\mathbf{z} - \mathbf{m}_k)]$, with $\mathbf{m}_k$ and $\mathbf{C}_k$ being the hyperparameters of $\mathcal{G}(\cdot)$, widely known as the "mean" and "covariance matrix" of $\mathcal{G}(\cdot)$, respectively, while $\intercal$ stands for vector/matrix transposition. The parameter space of GMM-QFs takes the form

$$\begin{aligned}\mathfrak{M} &\coloneqq \Big\{ \boldsymbol{\Omega} \coloneqq (\xi_1, \dots, \xi_K, \mathbf{m}_1, \dots, \mathbf{m}_K, \mathbf{C}_1, \dots, \mathbf{C}_K) \;\Big|\; \xi_k \in \mathbb{R}, \\ &\qquad \mathbf{m}_k \in \mathbb{R}^{D_z}, \mathbf{C}_k \text{ is positive definite}, \forall k = 1, \dots, K \Big\} \\ &= \mathbb{R}^K \times \mathbb{R}^{D_z \times K} \times (\mathbb{S}_{++}^{D_z})^K, \end{aligned} \quad (2)$$

where $\mathbb{S}_{++}^{D_z}$ stands for the set of all $D_z \times D_z$ positive-definite matrices. Interestingly, $\mathfrak{M}$ is a Riemannian manifold [30, 31] because all of $\mathbb{R}^K$, $\mathbb{R}^{D_z \times K}$, and $\mathbb{S}_{++}^{D_z}$ are.

To learn the "optimal" parameters from (2), BR minimization [12, 15, 34–36] is employed. Motivation comes from the classical Bellman mappings [1], which quantify the total loss (= one-step loss + expected long-term loss) to be paid by the agent, had action $a$ been taken at state $\mathbf{s}$. More specifically, if $\mathscr{B}$ stands for the space of Q-functions, usually being the Banach space of all essentially bounded functions [1], then the classical Bellman mappings $T_\mu^\diamond, T^\diamond\colon \mathscr{B} \to \mathscr{B}\colon Q \mapsto T_\mu^\diamond Q, T^\diamond Q$ are defined as [1]

$$(T_\mu^\diamond Q)(\mathbf{s}, a) \coloneqq g(\mathbf{s}, a) + \alpha \mathbb{E}_{\mathbf{s}'|(\mathbf{s},a)}[Q(\mathbf{s}', \mu(\mathbf{s}'))], \quad (3a)$$

$$(T^\diamond Q)(\mathbf{s}, a) \coloneqq g(\mathbf{s}, a) + \alpha \mathbb{E}_{\mathbf{s}'|(\mathbf{s},a)}[\min_{a' \in \mathfrak{A}} Q(\mathbf{s}', a')], \quad (3b)$$

$\forall (\mathbf{s}, a)$, where $\mathbb{E}_{\mathbf{s}'|(\mathbf{s},a)}[\cdot]$ stands for the conditional expectation operator with respect to the potentially next state $\mathbf{s}'$ conditioned on $(\mathbf{s}, a)$, and $\alpha \in [0, 1)$ is the discount factor. Mapping (3a) refers to the case where the agent takes actions according to the policy $\mu$, while (3b) serves as a greedy variation of (3a).

Given mapping $T : \mathscr{B} \to \mathscr{B}$, its fixed-point set $\operatorname{Fix} T \coloneqq \{Q \in \mathscr{B} \mid TQ = Q\}$. It is well-known that the fixed-point sets $\operatorname{Fix} T_\mu^\diamond$ and $\operatorname{Fix} T^\diamond$ play central roles in identifying *optimal* policies which minimizes the total loss [1]. Usually, the discount factor $\alpha \in [0, 1)$ to render $T_\mu^\diamond$, $T^\diamond$ strict contractions [1, 37]; hence, $\operatorname{Fix} T_\mu^\diamond$ and $\operatorname{Fix} T^\diamond$ become singletons. It is clear from (3) that the computation of $\operatorname{Fix} T_\mu^\diamond$ and $\operatorname{Fix} T^\diamond$ requires the knowledge on the transition probabilities to be able to compute the conditional expectation $\mathbb{E}_{\mathbf{s}'|(\mathbf{s},a)}[\cdot]$. However, in most cases of practice, transition probabilities are unavailable to the agent. To surmount this lack of information, designers utilize models for Q-functions. This manuscript utilizes (1).

Motivated by the importance of fixed points of Bellman mappings, and for the data samples $\mathscr{D}_\mu \coloneqq \{(\mathbf{s}_t, a_t, g_t, \mathbf{s}_t')\}_{t=1}^T$, for a number $T$ of time instances under a stationary policy $\mu$, the corresponding BR minimization via a smooth objective function $\mathcal{L}_\mu(\cdot)$ over the manifold $\mathfrak{M}$ in (2) is used to identify the desired fixed-point Q-functions for the policy $\mu$:

$$\min_{\boldsymbol{\Omega} \in \mathfrak{M}} \mathcal{L}_\mu(\boldsymbol{\Omega}) \coloneqq \sum\nolimits_{t=1}^{T} \Big[ g_t + \alpha \sum\nolimits_{k=1}^{K} \xi_k \mathcal{G}(\mathbf{z}_t' \mid \mathbf{m}_k, \mathbf{C}_k) - \sum\nolimits_{k=1}^{K} \xi_k \mathcal{G}(\mathbf{z}_t \mid \mathbf{m}_k, \mathbf{C}_k) \Big]^2, \quad (4)$$

where $\mathbf{z}_t' \coloneqq (\mathbf{s}_t', \mu(\mathbf{s}_t'))$. Task (4) is solved by Algorithm 2.

Albeit the similarity of (4) with standard BR minimization [12, 15, 34–36], (4) is performed over a parameter space, parametrized not only by the weights $\boldsymbol{\xi}$, as in [12, 15, 34–36],

**Algorithm 1** Policy iteration by Riemannian optimization

1: Arbitrarily initialize $\mathbf{\Omega}_0 \in \mathfrak{M}$, $\mu_0 \in \mathscr{M}$.
2: **while** $n \in \mathbb{N}$ **do**
3: **Policy evaluation** By current policy $\mu_n$, generate the dataset $\mathscr{D}_{\mu_n} := \{(\mathbf{s}_t, a_t, g_t, \mathbf{s}'_t)\}_{t=1}^{T}$ .
4: Update $\mathbf{\Omega}_{n+1}$ via Algorithm 2.
5: Given $\mathbf{\Omega}_{n+1}$, compute $Q_{n+1} \in \mathcal{Q}$ via (1).
6: **Policy improvement** Update $\mu_{n+1}(\mathbf{s}) := \arg\min_{a\in\mathfrak{A}} Q_{n+1}(\mathbf{s}, a)$.
7: Increase $n$ by one, go to Line 2.
8: **end while**

but also by the parameters $\{\mathbf{m}_k, \mathbf{C}_k\}_{k=1}^{K}$. In other words, and for a fixed $K$, (4) provides more degrees of freedom and a richer parameter space than the state-of-the-art BR-minimization methods [12, 15, 34–36].

## III. Policy iteration by Riemannian optimization

Following standard routes [1, 38], the classical policy-iteration (PI) strategy is used in Algorithm 1 to identify optimal policies. PI comprises two stages per iteration $n$: *policy evaluation* and *policy improvement.* At policy evaluation, the current policy is evaluated by the current Q-function estimate, which represents the long-term cost/loss estimate that the agent would suffer had the current policy been used to determine the next state. At the policy-improvement stage, the agent uses the obtained Q-function values to update the policy. Note that, this paper considers *finite* action space, and Line 6 of Algorithm 1 can be performed via exhausted search on $\mathfrak{A}$.

Nevertheless, looking more closely at Line 4 of Algorithm 1, the policy-evaluation stage is *newly* equipped here with a Riemannian-optimization task: solve (4) by the line search method of [30, §4.6.3]. To this end, the gradients of $\mathcal{L}_\mu(\cdot)$ along the directions $\boldsymbol{\xi}$, $\mathbf{m}_k$ and $\mathbf{C}_k$ are required, and provided by Proposition 1. Definitions of Riemannian concepts [30, 31], derivations and proofs are skipped because of limited space.

In Algorithm 2, the Riemannian metric [30, 31] of (5) on $\mathfrak{M}$ is adopted: $\forall \mathbf{\Omega} := (\boldsymbol{\xi}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \mathbf{C}_1, \ldots, \mathbf{C}_K) \in \mathfrak{M}$, and $\forall \mathbf{\Upsilon}_i := (\boldsymbol{\theta}_i, \boldsymbol{\mu}_{i1}, \ldots, \boldsymbol{\mu}_{iK}, \mathbf{\Gamma}_{i1}, \ldots, \mathbf{\Gamma}_{iK}) \in T_{\mathbf{\Omega}}\mathfrak{M}$, $i = 1, 2$, where $T_{\mathbf{\Omega}}\mathfrak{M}$ denotes the tangent space to $\mathfrak{M}$ at $\mathbf{\Omega}$ [30, 31],

$$\langle \mathbf{\Upsilon}_1 \mid \mathbf{\Upsilon}_2 \rangle_{\mathbf{\Omega}} := \boldsymbol{\theta}_1^{\intercal}\boldsymbol{\theta}_2 + \sum\nolimits_{k=1}^{K} \boldsymbol{\mu}_{1k}^{\intercal}\boldsymbol{\mu}_{2k} + \sum\nolimits_{k=1}^{K} \langle \mathbf{\Gamma}_{1k} \mid \mathbf{\Gamma}_{2k} \rangle_{\mathbf{C}_k} , \quad (5)$$

where $\langle \cdot \mid \cdot \rangle_{\mathbf{C}_k}$ can be any user-defined Riemannian metric of $\mathbb{S}_{++}^{D_z}$. Here, the Bures-Wasserstein (BW) metric [39] of (6) is used, because of its excellent performance comparing to the conventional affine-invariant metric [40] in numerical tests: $\forall \mathbf{C}_k \in \mathbb{S}_{++}^{D_z}$, and $\forall \mathbf{\Gamma}_{ik} \in T_{\mathbf{C}_k}\mathbb{S}_{++}^{D_z}$, $i = 1, 2$,

$$\langle \mathbf{\Gamma}_{1k} \mid \mathbf{\Gamma}_{2k} \rangle_{\mathbf{C}_k} := \langle \mathbf{\Gamma}_{1k} \mid \mathbf{\Gamma}_{2k} \rangle_{\mathbf{C}_k}^{\mathrm{BW}} := \tfrac{1}{2}\operatorname{tr}[L_{\mathbf{C}_k}(\mathbf{\Gamma}_{1k})\mathbf{\Gamma}_{2k}] , \quad (6)$$

where the Lyapunov operator $L_{\mathbf{C}_k}(\cdot)$ satisfies $\mathbf{C}_k L_{\mathbf{C}_k}(\mathbf{\Gamma}_{ik}) + L_{\mathbf{C}_k}(\mathbf{\Gamma}_{ik})\mathbf{C}_k = \mathbf{\Gamma}_{ik}$ [41]. Other Riemannian metrics on $\mathbb{S}_{++}^{D_z}$, such as the affine-invariant [40] or Log-Cholesky [42] ones can also be used in (5). Due to limited space, results obtained after employing those metrics will be reported elsewhere.

**Algorithm 2** Solving (4)

**Require:** Sampled data $\mathscr{D}_{\mu_n} := \{(\mathbf{s}_t, a_t, g_t, \mathbf{s}'_t)\}_{t=1}^{T}$; scalars $\bar{\alpha} > 0, \beta \in (0,1), \sigma \in (0,1)$, the number of steps $J$, a Riemannian metric $\langle \cdot \mid \cdot \rangle_\cdot$, and a retraction mapping $R_\cdot(\cdot)$ on $\mathfrak{M}$.
$\mathbf{\Omega}^{(0)} := \mathbf{\Omega}_n$.
**for** $j = 0, 1, 2, \ldots, J-1$ **do**
  $\mathbf{\Omega}^{(j)} := (\boldsymbol{\xi}^{(j)}, \mathbf{m}_1^{(j)}, \ldots, \mathbf{m}_K^{(j)}, \mathbf{C}_1^{(j)}, \ldots, \mathbf{C}_K^{(j)})$.
  By (7), compute:

$$\nabla\mathcal{L}_{\mu_n}(\mathbf{\Omega}^{(j)}) = (\tfrac{\partial\mathcal{L}_{\mu_n}}{\partial\boldsymbol{\xi}}(\mathbf{\Omega}^{(j)}), \ldots, \tfrac{\partial\mathcal{L}_{\mu_n}}{\partial\mathbf{m}_k}(\mathbf{\Omega}^{(j)}), \ldots, \tfrac{\partial\mathcal{L}_{\mu_n}}{\partial\mathbf{C}_k}(\mathbf{\Omega}^{(j)}), \ldots).$$

  Let $\mathbf{\Upsilon}^{(j)} := (\boldsymbol{\theta}^{(j)}, \boldsymbol{\mu}_1^{(j)}, \ldots, \boldsymbol{\mu}_K^{(j)}, \mathbf{\Gamma}_1^{(j)}, \ldots, \mathbf{\Gamma}_K^{(j)}) := -\nabla\mathcal{L}_{\mu_n}(\mathbf{\Omega}^{(j)})$. Find the smallest $M_{\mathrm{a}} \in \mathbb{N}_*$ such that

$$\mathcal{L}_{\mu_n}(\mathbf{\Omega}^{(j)}) - \mathcal{L}_{\mu_n}\left(R_{\mathbf{\Omega}^{(j)}}(\bar{\alpha}\beta^{M_{\mathrm{a}}}\mathbf{\Upsilon}^{(j)})\right) \geq -\sigma\langle \nabla\mathcal{L}_{\mu_n}(\mathbf{\Omega}^{(j)}) \mid \bar{\alpha}\beta^{M_{\mathrm{a}}}\mathbf{\Upsilon}^{(j)} \rangle_{\mathbf{\Omega}^{(j)}} .$$

  Define the step-size $t_j^{\mathrm{A}} := \bar{\alpha}\beta^{M_{\mathrm{a}}}$.
  Update $\mathbf{\Omega}^{(j+1)} := R_{\mathbf{\Omega}^{(j)}}(t_j^{\mathrm{A}}\mathbf{\Upsilon}^{(j)})$ via (8).
**end for**
$\mathbf{\Omega}_{n+1} := \mathbf{\Omega}^{(J)}$.

**Proposition 1** (Computing gradients)**.** *Consider* $\mathbf{\Omega}^{(j)} := (\boldsymbol{\xi}^{(j)}, \mathbf{m}_1^{(j)}, \ldots, \mathbf{m}_K^{(j)}, \mathbf{C}_1^{(j)}, \ldots, \mathbf{C}_K^{(j)}) \in \mathfrak{M}$ *(see Algorithm 2), and its associated GMM-QF* $Q^{(j)}$*. Let also* $\delta_t := g_t + \alpha Q^{(j)}(\mathbf{z}'_t) - Q^{(j)}(\mathbf{z}_t)$*. Then, the followings hold true.*

***(i)*** *If the objective function in* (4) *is recast as* $\mathcal{L}_\mu(\mathbf{\Omega}^{(j)}) = \|\mathbf{g} + \boldsymbol{\Delta}\boldsymbol{\xi}^{(j)}\|^2$*, where* $\mathbf{g} := [g_1, \ldots, g_T]^{\intercal}$ *and* $\boldsymbol{\Delta}$ *is a* $T \times K$ *matrix with entries* $\boldsymbol{\Delta}_{tk} := \alpha\mathcal{G}(\mathbf{z}'_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)}) - \mathcal{G}(\mathbf{z}_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)})$*, then,*

$$\frac{\partial\mathcal{L}_\mu}{\partial\boldsymbol{\xi}}(\mathbf{\Omega}^{(j)}) = 2\boldsymbol{\Delta}^{\intercal}(\mathbf{g} + \boldsymbol{\Delta}\boldsymbol{\xi}^{(j)}) . \quad (7a)$$

***(ii)*** *Let* $\mathbf{d}_{tk} := \alpha(\mathbf{z}'_t - \mathbf{m}_k^{(j)})\mathcal{G}(\mathbf{z}'_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)}) - (\mathbf{z}_t - \mathbf{m}_k^{(j)})\mathcal{G}(\mathbf{z}_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)})$*. Then,* $\forall k = 1, \ldots, K$*,*

$$\frac{\partial\mathcal{L}_\mu}{\partial\mathbf{m}_k}(\mathbf{\Omega}^{(j)}) = \sum\nolimits_{t=1}^{T} 4\delta_t\xi_k^{(j)}(\mathbf{C}_k^{(j)})^{-1}\mathbf{d}_{tk} . \quad (7b)$$

***(iii)*** *Under the BW metric [39],* $\forall k = 1, \ldots, K$*,*

$$\frac{\partial\mathcal{L}_\mu}{\partial\mathbf{C}_k}(\mathbf{\Omega}^{(j)}) = \sum\nolimits_{t=1}^{T} 4\delta_t\xi_k^{(j)}[(\mathbf{C}_k^{(j)})^{-1}\mathbf{B}_{tk} + \mathbf{B}_{tk}(\mathbf{C}_k^{(j)})^{-1}] \in T_{\mathbf{C}_k^{(j)}}\mathbb{S}_{++}^{D_z} , \quad (7c)$$

*where* $\mathbf{B}_{tk} := \alpha(\mathbf{z}'_t - \mathbf{m}_k^{(j)})(\mathbf{z}'_t - \mathbf{m}_k^{(j)})^{\intercal}\mathcal{G}(\mathbf{z}'_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)}) - (\mathbf{z}_t - \mathbf{m}_k^{(j)})(\mathbf{z}_t - \mathbf{m}_k^{(j)})^{\intercal}\mathcal{G}(\mathbf{z}_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)})$.

To run the line search algorithm on $\mathfrak{M}$, the *retraction mapping* $R_{\mathbf{\Omega}}$ [30], which maps an element of the tangent space $T_{\mathbf{\Omega}}\mathfrak{M}$ to an element in $\mathfrak{M}$ is needed. The most celebrated retraction is the *Riemannian exponential mapping* [30, 31]. Motivated by this fact, for $\mathbf{\Omega} := (\boldsymbol{\xi}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \mathbf{C}_1, \ldots, \mathbf{C}_K) \in \mathfrak{M}$, for a tangent vector $\mathbf{\Upsilon} := (\boldsymbol{\theta}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_K) \in T_{\mathbf{\Omega}}\mathfrak{M}$, and for the step size $t^{\mathrm{A}} > 0$, met in Algorithm 2, the retraction mapping $R_{\mathbf{\Omega}}(t^{\mathrm{A}}\mathbf{\Upsilon}) = (R_{\boldsymbol{\xi}}(t^{\mathrm{A}}\boldsymbol{\theta}), \ldots, R_{\mathbf{m}_k}(t^{\mathrm{A}}\boldsymbol{\mu}_k), \ldots, R_{\mathbf{C}_k}(t^{\mathrm{A}}\mathbf{\Gamma}_k), \ldots)$ is provided by the following: $\forall k \in \{1, \ldots, K\}$,

$$R_{\boldsymbol{\xi}}(t^{\mathrm{A}}\boldsymbol{\theta}) := \boldsymbol{\xi} + t^{\mathrm{A}}\boldsymbol{\theta} , \quad (8a)$$

$$R_{\mathbf{m}_k}(t^{\mathrm{A}}\boldsymbol{\mu}_k) := \mathbf{m}_k + t^{\mathrm{A}}\boldsymbol{\mu}_k\,, \tag{8b}$$

$$R_{\mathbf{C}_k}(t^{\mathrm{A}}\boldsymbol{\Gamma}_k) := \exp^{\mathrm{BW}}_{\mathbf{C}_k}(t^{\mathrm{A}}\boldsymbol{\Gamma}_k)\,, \tag{8c}$$

where, under the BW metric,

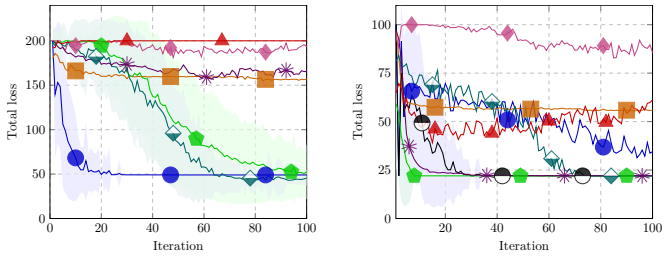
$$\exp^{\mathrm{BW}}_{\mathbf{C}_k}(t^{\mathrm{A}}\boldsymbol{\Gamma}_k) := \mathbf{C}_k + t^{\mathrm{A}}\boldsymbol{\Gamma}_k + L_{\mathbf{C}_k}(t^{\mathrm{A}}\boldsymbol{\Gamma}_k)\,\mathbf{C}_k\,L_{\mathbf{C}_k}(t^{\mathrm{A}}\boldsymbol{\Gamma}_k)\,.$$

## IV. Numerical Tests

Two classical benchmark RL tasks with finite action space, the *Inverted Pendulum* [43] and the *Mountain Car* [44], are selected to validate the proposed Algorithm 1 against: **(i)** Kernel-based least-squares policy iteration (KLSPI) [11], which utilizes LSTD in RKHS; **(ii)** online Bellman residual (OBR) [12]; **(iii)** DQN [18], **(iv)** dueling double (D)DQN [19] which use DNNs to train the Q-functions (experienced data are required), **(v)** proximal policy optimization (PPO) [45], a notable policy-search method [33] which models the policy via a DNN, and **(vi)** the GMM-based RL [25] via an online EM algorithm (EM-GMMRL). The validation criterion (vertical axes in Figures 1 and 2) measures the total loss the agent suffers until it achieves the "goal" of the task when operating under the current policy $\mu_n$, with $n$ being the iteration index of Algorithm 1 as well as the coordinate of the horizontal axes. Note that, this paper follows the framework in [1], using losses instead of rewards [2] therefore, the target here is to *minimize* the total loss. Results are averages from 100 independent tests. Software code was written in Julia/Python.

The "inverted pendulum" [43] refers to the problem of swinging up a pendulum from its lowest position to the upright one, with limited number of torques. The state $\mathbf{s} := [\theta, \dot{\theta}]^\intercal$, where $\theta \in [-\pi, \pi]$ is the angular position ($\theta = 0$ corresponds to the upright position), and $\dot{\theta} \in [-4, 4]\mathrm{s}^{-1}$ is the angular velocity. The action space is the set of torques $\mathfrak{A} := \{-5, -3, 0, 3, 5\}\mathrm{N}$. The one-step loss is defined as $g(\mathbf{s}, a) := 0$, if $\theta = 0$, and 1, if $\theta \neq 0$. To collect $\mathscr{D}_{\mu_n}$ in Algorithm 1, the pendulum starts from an angular position and explores a number of actions under the policy $\mu_n$. This exploration is called an episode, and per iteration $n$ in Algorithm 1, data $\mathscr{D}_{\mu_n}$ with $T :=$ (number of episodes) $\times$ (number of actions) $= 20 \times 70 = 1400$ are collected. KLSPI [11] and OBR [12] use the Gaussian kernel with bandwidth $\sigma_\kappa = 2$, while their ALD threshold is $\delta_{\mathrm{ALD}} = 0.01$. KLSPI and OBR need $T = 5000$ to reach their "optimal" performance for the task at hand. DQN [18] and dueling DDQN [19] use a fully-connected neural network with 2 hidden layers of size 128, with batch size of 64, and a replay buffer (experienced data) of size $1 \times 10^5$, while PPO [45] uses a memory of size 64. For EM-GMMRL [25], $T = 500$, while its threshold to add new Gaussian functions in the model is $10^{-4}$.

It can be seen from Figure 1a, that the proposed Algorithm 1 scores the best performance with no use of replay buffer (experienced data), unlike DQN [18], dueling DDQN [19] which require a large replay buffer, and exhibits slower learning speed and higher variance than GMM-QFs. PPO [45] seems to need more data to reach the optimal performance. KLSPI [11] underperforms, while OBR [12] and EM-GMMRL [25] fail to score a satisfactory performance. Notice that KLSPI and




(a) Inverted pendulum (b) Mountain car

Fig. 1: Results for benchmark datasets. Curve markers: Algorithm 1 with $K = 5$: ●, $K = 500$: ◓, KLSPI [11]: ■, OBR [12]: ◆, DQN [18]: ⬟, EM-GMMRL [25]: ▲, DDQN [19]: ◈, PPO [45]: ✳

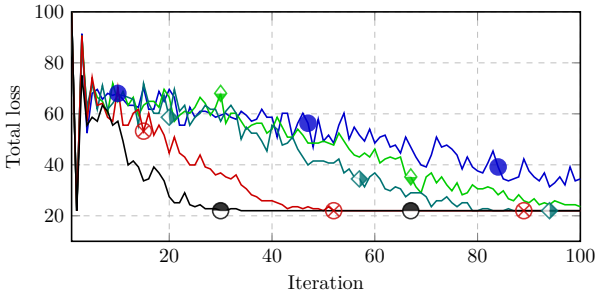



Fig. 2: Effect of different $K$ in Algorithm 1 for the setting of Figure 1b. Curve markers: $K = 20$: ◈, $K = 50$: ◈, $K = 200$: ⊗ Curve markers for $K = 5$ and $K = 500$ follow those of Figure 1. The larger the $K$, the richer the hyperparameter space $\mathfrak{M}$ and the faster the agent learns through the feedback from the environment, at the expense of increased computational complexity.

OBR are given more exploration data than Algorithm 1. It is also worth noting here that EM algorithms are sensitive to initialization [28], and that several initialization strategies were tried in all of the numerical tests. It is also worth stressing here that, the computational time of Algorithm 1 is usually less than DRL schemes, since Algorithm 2 is operated on a manifold of less parameters, while DRLs optimize over large number of parameters in deep models. Due to limited of space, results on computational time will be reported in the journal version.

"Mountain car" [44] refers to the task of accelerating a car to reach the top of the hill from the bottom of a sinusoidal valley, where the slope equation is given by $y = \sin(3x)$ in the $xy$-plane, with $x \in [-1.2, 0.6]$. The state $\mathbf{s} := [x, v]^\intercal$, where the velocity of the car $v \in [-0.07, 0.07]$. The goal is achieved whenever the car reaches a state in $\mathfrak{S}_g := \{[x, v]^\intercal \mid x \geq 0.5, v \geq 0\}$. The one-step loss is defined as $g(\mathbf{s}, a) := 1$, if $\mathbf{s} \notin \mathfrak{S}_g$, while $g(\mathbf{s}, a) := 0$, otherwise. With regards to the data $\mathscr{D}_{\mu_n}$ in Algorithm 1, a strategy similar to that of the inverted pendulum is used. More specifically, $T = 1000$ for Algorithm 1, while $T = 20000$ for KLSPI [11] and $T = 1000$ for OBR [12]. A Gaussian kernel with width of $\sigma_\kappa = 0.1$ is used for KLSPI [11] and OBR [12]. The implementation of DRL schemes [18, 19, 45] is identical to one for the inverted-pendulum case, while $T = 100$ for EM-GMMRL [25].

In Figure 1b, DQN [18] scores the best performance, followed by PPO [45] and Algorithm 1. Note again here that DQN, and dueling DDQN [19] use a large number of

experienced data (size of replay buffer is $1 \times 10^5$), while Algorithm 1 needs no experienced data to achieve the performance of Figure 1b. Observe also that by increasing the number $K$ of Gaussians in (1), GMM-QFs reach the total-loss performance of DQN in Figure 1b, at the expense of increased computational complexity per iteration; see also Figure 2. OBR [12] and EM-GMMRL [24] perform better here than in Figure 1a, with the EM-GMMRL agent showing better "learning abilities" than the OBR one in Figure 1b.

## REFERENCES


[1] D. Bertsekas, *Reinforcement Learning and Optimal Control*. Belmont, MA: Athena Scientific, 2019.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambrigde, MA: The MIT Press, 2018.

[3] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[4] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 287–308, 2000.

[5] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, pp. 161–178, 2002.

[6] D. Ormoneit and P. Glynn, "Kernel-based reinforcement learning in average-cost problems," *IEEE Transactions on Automatic Control*, vol. 47, no. 10, pp. 1624–1636, Oct. 2002.

[7] J. Bae, L. S. Giraldo, P. Chhatbar, J. Francis, J. Sanchez, and J. Príncipe, "Stochastic kernel temporal difference for reinforcement learning," in *Proc. IEEE MLSP*, 2011, pp. 1–6. DOI: 10.1109/MLSP.2011.6064634.

[8] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988. DOI: 10.1023/A:1022633531479.

[9] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[10] A.-M. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor, "Regularized policy iteration with nonparametric function spaces," *J. Machine Learning Research*, vol. 17, no. 1, pp. 4809–4874, 2016.

[11] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE Transactions on Neural Networks*, vol. 18, no. 4, pp. 973–992, 2007. DOI: 10.1109/TNN.2007.899161.

[12] W. Sun and J. A. Bagnell, "Online Bellman residual and temporal difference algorithms with predictive error guarantees," in *International Joint Conference on Artificial Intelligence*, 2016, pp. 4213–4217.

[13] M. Vu, Y. Akiyama, and K. Slavakis, "Dynamic selection of p-norm in linear adaptive filtering via online kernel-based reinforcement learning," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10096825.

[14] Y. Akiyama and K. Slavakis, "Proximal Bellman mappings for reinforcement learning and their application to robust adaptive filtering," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 5855–5859. DOI: 10.1109/ICASSP48485.2024.10446701.

[15] Y. Akiyama, M. Vu, and K. Slavakis, "Nonparametric Bellman mappings for reinforcement learning: Application to robust adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 72, pp. 5644–5658, 2024. DOI: 10.1109/TSP.2024.3505266.

[16] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, vol. 68, no. 3, pp. 337–404, 1950.

[17] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. arXiv: 1312.5602.

[19] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, New York, NY, USA: J. Machine Learning Research, 2016, pp. 1995–2003.

[20] L.-J. Lin, "Reinforcement learning for robots using neural networks," UMI Order No. GAX93-22750, Ph.D. dissertation, USA, 1992.

[21] G. McLachlan and D. Peel, *Finite Mixture Models*. Wiley, 2000.

[22] M. Sato and S. Ishii, "Reinforcement learning based on on-line EM algorithm," in *Advances in Neural Information Processing Systems*, vol. 11, MIT Press, 1998.

[23] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *International Conference on Machine Learning*, ser. ICML '05, Bonn, Germany: Association for Computing Machinery, 2005, pp. 201–208. DOI: 10.1145/1102351.1102377.

[24] A. Agostini and E. Celaya, "Reinforcement learning with a Gaussian mixture model," in *International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8. DOI: 10.1109/IJCNN.2010.5596306.

[25] A. Agostini and E. Celaya, "Online reinforcement learning using a probability density estimation," *Neural Comput.*, vol. 29, no. 1, pp. 220–246, Jan. 2017. DOI: 10.1162/NECO_a_00906.

[26] Y. Choi, K. Lee, and S. Oh, "Distributional deep reinforcement learning with a mixture of Gaussians," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9791–9797. DOI: 10.1109/ICRA.2019.8793505.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B*, vol. 39, pp. 1–38, 1977.

[28] M. A. T. Figueiredo and A. K. Jain, "Unsupervised learning of finite mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 381–396, 2002. DOI: 10.1109/34.990138.

[29] L. Györfi, M. Kohler, A. Krzyżak, and H. Walk, *A Distribution-Free Theory of Nonparametric Regression*. 2002.

[30] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press, 2008.

[31] J. W. Robbin and D. A. Salamon, *Introduction to Differential Geometry*. Berlin: Springer, 2022.

[32] S. Wang, B. Zhu, C. Li, M. Wu, J. Zhang, W. Chu, and Y. Qi, "Riemannian proximal policy optimization," *Computer and Information Science*, vol. 13, no. 3, pp. 1–93, Aug. 2020.

[33] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, vol. 12, MIT Press, 1999.

[34] Z. Qin, W. Li, and F. Janoos, "Sparse reinforcement learning via convex optimization," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: PMLR, Jun. 2014, pp. 424–432.

[35] S. Mahadevan, B. Liu, P. Thomas, W. Dabney, S. Giguere, N. Jacek, I. Gemp, and J. Liu, *Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces*, 2014. arXiv: 1405.6757 [cs.LG].

[36] B. Liu, I. Gemp, M. Ghavamzadeh, J. Liu, S. Mahadevan, and M. Petrik, "Proximal gradient temporal difference learning: Stable reinforcement learning with polynomial sample complexity," *J. Artif. Int. Res.*, vol. 63, no. 1, pp. 461–494, Sep. 2018. DOI: 10.1613/jair.1.11251.

[37] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York: Springer, 2011.

[38] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, vol. 12, MIT Press, 1999.

[39] R. Bhatia, T. Jain, and Y. Lim, "On the Bures–Wasserstein distance between positive definite matrices," *Expositiones Mathematicae*, vol. 37, no. 2, pp. 165–191, 2019.

[40] X. Pennec, S. Sommer, and T. Fletcher, *Riemannian Geometric Statistics in Medical Image Analysis*. San Diego, CA: Academic Press, 2019.

[41] R. Bobiti and M. Lazar, "A sampling approach to finding Lyapunov functions for nonlinear discrete-time systems," in *European Control Conference (ECC)*, 2016, pp. 561–566. DOI: 10.1109/ECC.2016.7810344.

[42] R. Bhatia, *Positive Definite Matrices*. Princeton, NJ: Princeton University Press, 2007.

[43] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000. DOI: 10.1162/089976600300015961.

[44] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Tech. Rep., 1990.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347.