GAUSSIAN-MIXTURE-MODEL Q-FUNCTIONS FOR REINFORCEMENT LEARNING BY RIEMANNIAN OPTIMIZATION

Minh Vu   Konstantinos Slavakis

Tokyo Institute of Technology, Japan
Department of Information and Communications Engineering
Emails: {vu.d.aa, slavakis.k.aa}@m.titech.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 Q-networks which use experienced data, on benchmark RL tasks.

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

1. INTRODUCTION

In reinforcement learning (RL) [1, 2], an "intelligent agent" interacts with an unknown environment (typically modeled as a Markov decision process (MDP)) to identify an optimal policy that minimizes the total costs of its "actions." RL 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].

To identify an optimal policy, RL strategies typically compute/evaluate the *value/loss* (Q-function) associated with an action at a given state by observing feedback data from the environment. 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, there are many practical problems which involve very large, or even continuous state-action spaces that 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 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, with few parameters to be learned to effect dimensionality reduction, robustness to erroneous information, and swift adaptability to dynamic environments, and with *no need for* past experienced data, this paper introduces the class of *Gaussian-mixture-model Q-functions (GMM-QFs)*. GMM-QFs are weighted sum averages of multivariate Gaussian kernels, where not only the weights, but also the hyperparameters of the Gaussian kernels are free to be learned [21]. This contrasts the aforementioned literature of KBRL [5–12], where the hyperparameters of the user-defined kernels are directly parameterized by the observed data and are not considered variables of learning tasks. 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, this paper departs from the typical GMM usage and their EM solutions [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 a smooth objective function, and relies on Riemannian optimization [29] to minimize that objective function and to exploit the underlying Riemannian geometry [30] of the hyperparameter space. GMMs and Riemannian optimization have been used to model policy functions as PDFs [31], under the framework of policy search [32]. 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 nonparametric model with

the number of data [5–12], effecting dimensionality reduction, and providing low-computational load as well as stable performance under erroneous information. Indeed, numerical tests on benchmark control tasks demonstrate that the advocated GMM-QFs outperform other state-of-the-art RL schemes, even DQNs which require experienced data. Due to limited space, detailed definitions and arguments of Riemannian geometry, proofs, results on convergence, and further numerical tests are deferred to the journal version of this manuscript.

2. THE CLASS OF GMM Q-FUNCTIONS (GMM-QFs)

2.1. RL notations

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 $Q(\cdot, \cdot) : \mathfrak{S} \times \mathfrak{A} \to \mathbb{R} : (\mathbf{s}, a) \mapsto Q(\mathbf{s}, a)$ 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 $\mathbf{z} := (\mathbf{s}, a) \in \mathfrak{Z} := \mathfrak{S} \times \mathfrak{A} \subset \mathbb{R}^{D_z}$, where $D_z \in \mathbb{N}_*$.

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

2.2. GMM-QFs

Motivated by GMMs [21], and for a user-defined positive integer K, GMM-QFs are defined as the following class of functions:

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

where $\mathcal{G}(\mathbf{z} \mid \mathbf{m}_k, \mathbf{C}_k) := \exp[-(\mathbf{z} - \mathbf{m}_k)^{\mathsf{T}} \mathbf{C}_k^{-1} (\mathbf{z} - \mathbf{m}_k)]$, 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 T stands for vector/matrix transposition. The parameter space of GMM-QFs takes the form

$$m := \Big\{ \mathbf{\Omega} := (\xi_1, \ldots, \xi_K, \mathbf{m}_1, \ldots, \mathbf{m}_K, \mathbf{C}_1, \ldots, \mathbf{C}_K) \mid \xi_k \in \mathbb{R},$$
$$\mathbf{m}_k \in \mathbb{R}^{D_z}, \mathbf{C}_k \text{ is positive definite}, \forall k = 1, \ldots, K \Big\}$$
$$= \mathbb{R}^K \times \mathbb{R}^{D_z \times K} \times (\mathbb{S}_{++}^{D_z})^K, \quad (2)$$

where $\mathbb{S}_{++}^{D_z}$ stands for the set of all $D_z \times D_z$ positive-definite matrices. Interestingly, m is a Riemannian manifold [29, 30] 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, 33–35] 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 $T_\mu^\diamond, T^\diamond : \mathscr{B} \to \mathscr{B} : Q \mapsto T_\mu^\diamond Q, T^\diamond Q$ are defined as [1]

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

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

$\forall(\mathbf{s}, a)$, 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 stationary policy μ, while (3b) serves as a greedy variation of (3a).

Given mapping $T : \mathscr{B} \to \mathscr{B}$, its fixed-point set $\operatorname{Fix} T := \{Q \in \mathscr{B} \mid TQ = Q\}$. 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, 36]; 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 GMM-QFs in (1).

Motivated by the importance of fixed points of Bellman mappings in RL, and for the data samples $\mathscr{D}_\mu := \{(\mathbf{s}_t, a_t, g_t, \mathbf{s}'_t)\}_{t=1}^T$, for a number T of time instances under a stationary policy μ, the following minimization task of the smooth objective $\mathcal{L}(\cdot)$ over the Riemannian manifold m will be used to identify the desired fixed-point Q-functions for the policy μ:

$$\min_{\mathbf{\Omega} \in m} \mathcal{L}(\mathbf{\Omega}) := \sum_{t=1}^{T} \Big[g_t + \alpha \sum_{k=1}^{K} \xi_k \mathcal{G}(\mathbf{z}'_t \mid \mathbf{m}_k, \mathbf{C}_k)$$
$$- \sum_{k=1}^{K} \xi_k \mathcal{G}(\mathbf{z}_t \mid \mathbf{m}_k, \mathbf{C}_k) \Big]^2, \quad (4)$$

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

Albeit the similarity of (4) with standard BR minimization [12, 15, 33–35], (4) is performed over a parameter space, parameterized not only by the weights $\boldsymbol{\xi}$, as in [12, 15, 33–35], 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, 33–35].

3. POLICY ITERATION BY RIEMANNIAN OPTIMIZATION

Following standard routes [1, 37], 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.

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 steepest-gradient-descent method with line search of [29, §4.6.3]. To this end, the gradients of $\mathcal{L}(\cdot)$ along the directions $\boldsymbol{\xi}$, $\mathbf{m}_k$ and $\mathbf{C}_k$ are required, and provided by Proposition 1. Definitions of basic Riemannian concepts [29, 30], detailed derivations and proofs are skipped because of limited space.

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

To run computations in Algorithm 2, the Riemannian metric [29, 30] of (5) on $\mathcal{M}$ is adopted: $\forall \boldsymbol{\Omega} := (\boldsymbol{\xi}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \mathbf{C}_1, \ldots, \mathbf{C}_K) \in \mathcal{M}$, and $\forall \boldsymbol{\Upsilon}_i := (\boldsymbol{\theta}_i, \boldsymbol{\mu}_{i1}, \ldots, \boldsymbol{\mu}_{iK}, \boldsymbol{\Gamma}_{i1}, \ldots, \boldsymbol{\Gamma}_{iK}) \in T_{\boldsymbol{\Omega}}\mathcal{M}$, $i = 1, 2$, where $T_{\boldsymbol{\Omega}}\mathcal{M}$ denotes the tangent space to $\mathcal{M}$ at $\boldsymbol{\Omega}$ [29, 30],

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

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 [38] of (6) is used, because of its excellent performance in numerical tests: $\forall \mathbf{C}_k \in \mathbb{S}_{++}^{D_z}$, and $\forall \boldsymbol{\Gamma}_{ik} \in T_{\mathbf{C}_k}\mathbb{S}_{++}^{D_z}$, $i = 1, 2$,

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

where the Lyapunov operator $L_{\mathbf{C}_k}(\cdot)$ satisfies $\mathbf{C}_k L_{\mathbf{C}_k}(\boldsymbol{\Gamma}_{ik}) + L_{\mathbf{C}_k}(\boldsymbol{\Gamma}_{ik})\mathbf{C}_k = \boldsymbol{\Gamma}_{ik}$ [39]. Other Riemannian metrics on $\mathbb{S}_{++}^{D_z}$, such as the affine-invariant [40] or Log-Cholesky [41] ones can be used in (5). Due to limited space, results obtained after employing those metrics will be reported elsewhere.

Proposition 1 (Computing gradients). *Consider a point* $\boldsymbol{\Omega}^{(j)} := (\boldsymbol{\xi}^{(j)}, \mathbf{m}_1^{(j)}, \ldots, \mathbf{m}_K^{(j)}, \mathbf{C}_1^{(j)}, \ldots, \mathbf{C}_K^{(j)}) \in \mathcal{M}$ *(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 following hold true.*

(i) *If the objective function in (4) is recast as* $\mathcal{L}(\boldsymbol{\Omega}^{(j)}) = \|\mathbf{g} + \boldsymbol{\Delta}\boldsymbol{\xi}^{(j)}\|^2$, *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,*

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

(ii) $\forall k = 1, \ldots, K$,

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \mathbf{m}_k}(\boldsymbol{\Omega}^{(j)}) \\
&= \sum_{t=1}^T 4\delta_t \xi_k^{(j)}(\mathbf{C}_k^{(j)})^{-1}\big[\alpha(\mathbf{z}'_t - \mathbf{m}_k^{(j)})\mathcal{G}(\mathbf{z}'_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)}) \\
&\qquad\qquad - (\mathbf{z}_t - \mathbf{m}_k^{(j)})\mathcal{G}(\mathbf{z}_t \mid \mathbf{m}_k^{(j)}, \mathbf{C}_k^{(j)})\big].
\end{aligned} \quad (7b)$$

(iii) *Under the BW metric [38],* $\forall k = 1, \ldots, K$,

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

Algorithm 2 Solving (4)
1: **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 $\langle \cdot \mid \cdot \rangle$, and a retraction mapping $R_{\cdot}(\cdot)$ on $\mathcal{M}$.
2: $\boldsymbol{\Omega}^{(0)} := \boldsymbol{\Omega}_n$.
3: **for** $j = 0, 1, 2, \ldots, J - 1$ **do**
4:     $\boldsymbol{\Omega}^{(j)} := (\boldsymbol{\xi}^{(j)}, \mathbf{m}_1^{(j)}, \ldots, \mathbf{m}_K^{(j)}, \mathbf{C}_1^{(j)}, \ldots, \mathbf{C}_K^{(j)})$.
5:     By (7), compute:

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

6:     Let

$$\boldsymbol{\Upsilon}^{(j)} := (\boldsymbol{\theta}^{(j)}, \boldsymbol{\mu}_1^{(j)}, \ldots, \boldsymbol{\mu}_K^{(j)}, \boldsymbol{\Gamma}_1^{(j)}, \ldots, \boldsymbol{\Gamma}_K^{(j)}) := -\nabla\mathcal{L}(\boldsymbol{\Omega}^{(j)}).$$

7:     Find the smallest $M_{\mathrm{a}} \in \mathbb{N}_*$ such that

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\Omega}^{(j)}) - \mathcal{L}\left(R_{\boldsymbol{\Omega}^{(j)}}(\bar{\alpha}\beta^{M_{\mathrm{a}}}\boldsymbol{\Upsilon}^{(j)})\right) \\
&\quad \geq -\sigma\langle\nabla\mathcal{L}(\boldsymbol{\Omega}^{(j)}) \mid \bar{\alpha}\beta^{M_{\mathrm{a}}}\boldsymbol{\Upsilon}^{(j)}\rangle_{\boldsymbol{\Omega}^{(j)}}.
\end{aligned}$$

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

---

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 steepest gradient descent in Riemannian optimization, the *retraction mapping* $R_{\boldsymbol{\Omega}}$ [29] is needed, where, loosely speaking, $R_{\boldsymbol{\Omega}}$ is a mapping which maps an element of the tangent space $T_{\boldsymbol{\Omega}}\mathcal{M}$ to an element in $\mathcal{M}$. The most celebrated retraction is the *Riemannian exponential mapping* [29, 30]. Motivated by this fact, for $\boldsymbol{\Omega} := (\boldsymbol{\xi}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \mathbf{C}_1, \ldots, \mathbf{C}_K) \in \mathcal{M}$, for a tangent vector $\boldsymbol{\Upsilon} := (\boldsymbol{\theta}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_K) \in T_{\boldsymbol{\Omega}}\mathcal{M}$, and for the step size $t^{\mathrm{A}} > 0$, met in Algorithm 2, the retraction mapping $R_{\boldsymbol{\Omega}}(t^{\mathrm{A}}\boldsymbol{\Upsilon}) = (R_{\boldsymbol{\xi}}(t^{\mathrm{A}}\boldsymbol{\theta}), \ldots, R_{\mathbf{m}_k}(t^{\mathrm{A}}\boldsymbol{\mu}_k), \ldots, R_{\mathbf{C}_k}(t^{\mathrm{A}}\boldsymbol{\Gamma}_k), \ldots)$ is provided by the following: $\forall k \in \{1, \ldots, K\}$,

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

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

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

where, under the BW metric,

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

4. NUMERICAL TESTS

Two classical benchmark RL tasks, the *Inverted Pendulum* [42] and the *Mountain Car* [43], 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)** the popular deep Q-network (DQN) [18], which uses deep neural networks to train the Q-functions (experienced data are required); and **(iv)** the GMM-based RL [25] via an online EM algorithm (EM-GMMRL). Two scenarios for the one-step loss function g are also considered, one where g is continuous and another where it is discrete. The validation criterion (vertical
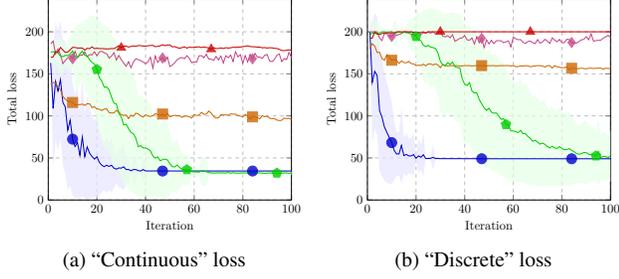
(a) "Continuous" loss   (b) "Discrete" loss

Fig. 1: Inverted-pendulum dataset. Curve markers: Algorithm 1 with $K = 5$: ●, KLSPI [11]: ■, OBR [12]: ◆, DQN [18]: ●, EM-GMMRL [25]: ▲

(a) "Continuous" loss   (b) "Discrete" loss

Fig. 2: Mountain-car dataset. Curve markers: Algorithm 1 with $K = 500$: ⬡, others follow Figure 1.

Fig. 3: Effect of different K in Algorithm 1 for the setting of Figure 2b. Curve markers: $K = 20$: ◆, $K = 50$: ◆, $K = 200$: ⊗ The curve markers for $K = 5$ and $K = 500$ follow those of Figures 1 and 2. The larger the K, the richer the hyperparameter space $\mathcal{M}$ and the faster the agent learns through the feedback from the environment, at the expense of increased computational complexity.

axes in Figures 1 to 3) measures the total loss the agent suffers until it achieves the "goal" of the task when operating under the current policy μ_n, with n being the iteration index of Algorithm 1 as well as the coordinate of the horizontal axes in Figures 1 to 3. Results are averages from 100 independent tests. Software code was written in Julia [44]/Python.

The "inverted pendulum" [42] refers to the problem of swinging up a pendulum from its lowest position to the upright one, given a limited number of torques. The state $\mathbf{s} := [\theta, \dot{\theta}]^{\mathsf{T}}$, 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 $\mathfrak{A} := \{-5, -3, 0, 3, 5\}$N. The continuous one-step loss is defined as $g(\mathbf{s}, a) := |\theta|/\pi$, while the discrete one is defined as $g(\mathbf{s}, a) := 0$, if $\theta = 0$, and $g(\mathbf{s}, a) := 1$, if $\theta \neq 0$.

To collect data samples $\mathscr{D}_{\mu_n}$ in Algorithm 1, the pendulum starts from an angular position and explores a number of actions under the current policy μ_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] uses 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 1×10^5. For EM-GMMRL [25], $T = 500$, while its threshold to add new Gaussian functions in its dictionary is 10^{-4}.

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

"Mountain car" [43] 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 $\mathbf{s} := [x, v]^{\mathsf{T}}$, where the velocity of the car $v \in [-0.07, 0.07]$. The goal is achieved when the car gets beyond $x_g := 0.5$ with velocity larger than or equal to $v_g := 0$, that is, whenever the car reaches a state in $\mathfrak{S}_g := \{[x, v]^{\mathsf{T}} \mid x \geq x_g, v \geq v_g\}$. The discrete one-step loss is defined as $g(\mathbf{s}, a) := 1$, if $\mathbf{s} \notin \mathfrak{S}_g$, while $g(\mathbf{s}, a) := 0$, if $\mathbf{s} \in \mathfrak{S}_g$. The continuous one-step loss is defined as $g(\mathbf{s}, a) := [\max(x_g - x, 0) + \max(v_g - v, 0)]/2$.
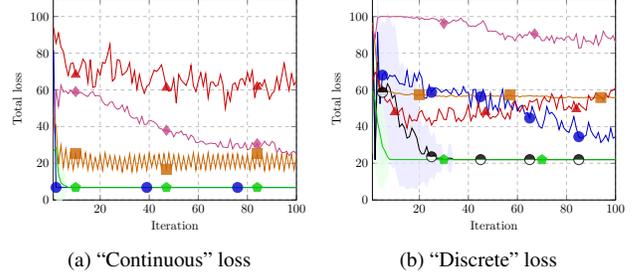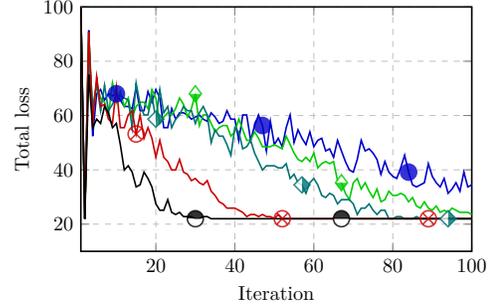
With regards to the data $\mathscr{D}_{\mu_n}$ in Algorithm 1, a strategy similar to that of the inverted pendulum is used. More specifically, $T = 1000$ for the proposed GMM-QFs, while $T = 20000$ for KLSPI [11] and $T = 1000$ for OBR [12]. A Gaussian kernel with width of $\sigma_\kappa = 0.1$ is used for KLSPI [11] and OBR [12]. The implementation of DQN [18] is identical to one for the inverted-pendulum case, while $T = 100$ for EM-GMMRL [25].

The proposed GMM-QFs outperform all competing methods in Figure 2a, while DQN [18] scores the best performance in Figure 2b. Note again here that DQN uses a large number of experienced data (size of replay buffer is 1×10^5), while the proposed GMM-QFs needs no experienced data to achieve the performance of Figure 2. Observe also that by increasing the number K of Gaussians in (1), GMM-QFs reach the total-loss performance of DQN in Figure 2b, at the expense of increased computational complexity; see also Figure 3. OBR [12] and EM-GMMRL [24] perform better here than in Figure 1, with the EM-GMMRL agent showing better "learning abilities" than the OBR one in Figure 2b, but vice versa in Figure 2a.

5. CONCLUSIONS

This paper established the novel class of GMM Q-functions (GMM-QFs), and offered a Riemannian-optimization algorithm, to learn the hyperparameters of GMM-QFs, as a novel policy-evaluation step in a policy-iteration scheme for computing optimal policies. The proposed design shows ample degrees of freedom not only because it

introduces a rich hyperparameter space, but also because it establishes the exciting connection between Q-function identification in RL and the powerful toolbox of Riemannian optimization. Numerical tests on benchmark tasks demonstrated the superior performance of the proposed design over state-of-the-art schemes.

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*, 2024. arXiv: 2403.20020 [eess.SP].

[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] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *AAAI Conference on Artificial Intelligence*, ser. AAAI'16, Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.

[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*, M. Kearns, S. Solla, and D. Cohn, Eds., 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] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press, 2008.

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

[31] 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.

[32] 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*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 1999.

[33] Z. Qin, W. Li, and F. Janoos, "Sparse reinforcement learning via convex optimization," in *International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: PMLR, 22–24 Jun 2014, pp. 424–432.

[34] 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].

[35] 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.

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

[37] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 1999.

[38] 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.

[39] 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.

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

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

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

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

[44] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017. DOI: 10.1137/141000671.